Abstract

The Dakota 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 describes the Dakota class hierarchies. It is derived from annotation of the source code and provides detailed class documentation, including all member functions and attributes.
Contents

1 Dakota Developers Manual ........................................ 13
   1.1 Introduction ................................................. 13
   1.2 Overview of Dakota ........................................ 13
   1.3 Services .................................................. 18
   1.4 Development Practices and Guidance ....................... 18
   1.5 Additional Resources ...................................... 19

2 Coding Style Guidelines and Conventions ...................... 21
   2.1 Introduction ................................................ 21
   2.2 C++/c Style Guidelines .................................... 21
   2.3 File Naming Conventions ................................... 23
   2.4 Class Documentation Conventions ......................... 24
   2.5 CMake Style Guidelines ................................... 24

3 Instructions for Modifying Dakota’s Input Specification .... 27
   3.1 XML Input Specification ................................... 27
   3.2 Rebuild Generated Files ................................... 30
   3.3 Update Parser Source NIDRProblemDescDB.cpp .......... 30
   3.4 Update Corresponding Data Classes ....................... 31
   3.5 Update Database Source ProblemDescDB.cpp ............ 32
   3.6 Use get_<data_type>() Functions ....................... 33
   3.7 Update the Documentation ................................ 33

4 Understanding Iterator Flow ..................................... 35

5 Interfacing with Dakota as a Library .......................... 37
   5.1 Introduction ................................................ 37
   5.2 Basic Dakota library instantiation ....................... 38
   5.3 Configuring Dakota operation ............................. 39
   5.4 Creating a simulator plugin interface ................... 42
   5.5 Retrieving data after a run .............................. 45
   5.6 Linking against the Dakota library ..................... 45

6 Performing Function Evaluations ................................ 47
   6.1 Synchronous function evaluations ....................... 47
   6.2 Asynchronous function evaluations ..................... 47
   6.3 Analyses within each function evaluation ................ 48
## Working with Variable Containers and Views

1. **Storage in Variables**
2. **Storage in SharedVariablesData**
3. **Active and inactive views**

## Demo TPL

## Namespace Index

1. **Namespace List**

## Hierarchical Index

1. **Class Hierarchy**

## Class Index

1. **Class List**

## File Index

1. **File List**

## Namespace Documentation

1. **dakota Namespace Reference**
2. **Dakota Namespace Reference**
3. **dakota::surrogates Namespace Reference**
4. **dakota::util Namespace Reference**
5. **SIM Namespace Reference**
6. **StanfordPSAAP Namespace Reference**

## Class Documentation

1. **ActiveSet Class Reference**
2. **ActiveSubspaceModel Class Reference**
3. **AdaptedBasisModel Class Reference**
4. **AddAttributeVisitor Class Reference**
5. **Analyzer Class Reference**
6. **ApplicationInterface Class Reference**
7. **Approximation Class Reference**
8. **ApproximationInterface Class Reference**
9. **APPSEvalMgr Class Reference**
10. **APPSOptimizer Class Reference**
11. **AppsTraits Class Reference**
12. **ApreproWriter Class Reference**
13. **AttachScaleVisitor Class Reference**
14. **BaseConstructor Struct Reference**
15. **BootstrapSampler\< Data \> Class Template Reference**
16. **BootstrapSampler\< Teuchos::SerialDenseMatrix\< OrdinalType, ScalarType \> \> Class Template Reference**
17. **BootstrapSamplerBase\< Data \> Class Template Reference**
18. **BootstrapSamplerWithGS\< Data, Getter, Setter \> Class Template Reference**
19. **C3Approximation Class Reference**
20. **C3FnTrainPtrs Class Reference**
21. **callback data Struct Reference**
22. **CholeskySolver Class Reference**
<table>
<thead>
<tr>
<th>Section</th>
<th>Class Reference</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>14.72</td>
<td>JEGAOptimizer::EvaluatorCreator</td>
<td>419</td>
</tr>
<tr>
<td>14.73</td>
<td>ExecutableEnvironment</td>
<td>420</td>
</tr>
<tr>
<td>14.74</td>
<td>ExperimentData</td>
<td>421</td>
</tr>
<tr>
<td>14.75</td>
<td>ExperimentResponse</td>
<td>429</td>
</tr>
<tr>
<td>14.76</td>
<td>FileReadException</td>
<td>431</td>
</tr>
<tr>
<td>14.77</td>
<td>ForkApplicInterface</td>
<td>431</td>
</tr>
<tr>
<td>14.78</td>
<td>FSUDesignCompExp</td>
<td>433</td>
</tr>
<tr>
<td>14.79</td>
<td>FunctionEvalFailure</td>
<td>437</td>
</tr>
<tr>
<td>14.80</td>
<td>GaussianProcess</td>
<td>437</td>
</tr>
<tr>
<td>14.81</td>
<td>GaussProcApproximation</td>
<td>446</td>
</tr>
<tr>
<td>14.82</td>
<td>GeneralReader</td>
<td>451</td>
</tr>
<tr>
<td>14.83</td>
<td>GeneralWriter</td>
<td>452</td>
</tr>
<tr>
<td>14.84</td>
<td>GetLongOpt</td>
<td>452</td>
</tr>
<tr>
<td>14.85</td>
<td>GP_Objective</td>
<td>455</td>
</tr>
<tr>
<td>14.86</td>
<td>Graphics</td>
<td>457</td>
</tr>
<tr>
<td>14.87</td>
<td>GridApplicInterface</td>
<td>459</td>
</tr>
<tr>
<td>14.88</td>
<td>HDF5IOHelper</td>
<td>461</td>
</tr>
<tr>
<td>14.89</td>
<td>HierarchSurrBasedLocalTraits</td>
<td>466</td>
</tr>
<tr>
<td>14.90</td>
<td>HierarchSurrModel</td>
<td>467</td>
</tr>
<tr>
<td>14.91</td>
<td>IntegerScale</td>
<td>473</td>
</tr>
<tr>
<td>14.92</td>
<td>Interface</td>
<td>474</td>
</tr>
<tr>
<td>14.93</td>
<td>Iterator</td>
<td>483</td>
</tr>
<tr>
<td>14.94</td>
<td>IteratorScheduler</td>
<td>499</td>
</tr>
<tr>
<td>14.95</td>
<td>JEGAOptimizer</td>
<td>505</td>
</tr>
<tr>
<td>14.96</td>
<td>JEGATraits</td>
<td>511</td>
</tr>
<tr>
<td>14.97</td>
<td>LabelsWriter</td>
<td>511</td>
</tr>
<tr>
<td>14.98</td>
<td>LeastSq</td>
<td>512</td>
</tr>
<tr>
<td>14.99</td>
<td>LibraryEnvironment</td>
<td>515</td>
</tr>
<tr>
<td>14.100</td>
<td>LightWtBaseConstructor</td>
<td>518</td>
</tr>
<tr>
<td>14.101</td>
<td>LinearSolverBase</td>
<td>518</td>
</tr>
<tr>
<td>14.102</td>
<td>MatchesWC</td>
<td>520</td>
</tr>
<tr>
<td>14.103</td>
<td>MatlabInterface</td>
<td>521</td>
</tr>
<tr>
<td>14.104</td>
<td>MetaIterator</td>
<td>522</td>
</tr>
<tr>
<td>14.105</td>
<td>Minimizer</td>
<td>524</td>
</tr>
<tr>
<td>14.106</td>
<td>MixedVarConstraints</td>
<td>532</td>
</tr>
<tr>
<td>14.107</td>
<td>MixedVariables</td>
<td>533</td>
</tr>
<tr>
<td>14.108</td>
<td>Model</td>
<td>535</td>
</tr>
<tr>
<td>14.109</td>
<td>MPIManager</td>
<td>572</td>
</tr>
<tr>
<td>14.110</td>
<td>MPIPackBuffer</td>
<td>573</td>
</tr>
<tr>
<td>14.111</td>
<td>MPlnpackBuffer</td>
<td>575</td>
</tr>
<tr>
<td>14.112</td>
<td>NCSUOptimizer</td>
<td>577</td>
</tr>
<tr>
<td>14.113</td>
<td>NCSUTraits</td>
<td>580</td>
</tr>
<tr>
<td>14.114</td>
<td>NestedModel</td>
<td>581</td>
</tr>
<tr>
<td>14.115</td>
<td>NIDRProblemDescDB</td>
<td>590</td>
</tr>
<tr>
<td>14.116</td>
<td>NL2Res Struct</td>
<td>596</td>
</tr>
<tr>
<td>14.117</td>
<td>NL2SOLLsLeastSq</td>
<td>596</td>
</tr>
<tr>
<td>14.118</td>
<td>NL2SOLLsLeastSqTraits</td>
<td>599</td>
</tr>
<tr>
<td>14.119</td>
<td>NL2QLPTrats</td>
<td>599</td>
</tr>
<tr>
<td>14.120</td>
<td>NLSSOLLsLeastSq</td>
<td>601</td>
</tr>
</tbody>
</table>
CONTENTS

14.12 NLSSOLLeastSqTraits Class Reference .................................................. 603
14.12 NoDBBaseConstructor Struct Reference .............................................. 603
14.12 NomadTraits Class Reference ............................................................... 604
14.12 NonD Class Reference ........................................................................... 605
14.12 NonDAdaptImpSampling Class Reference ........................................... 612
14.12 NonDAdaptiveSampling Class Reference .............................................. 617
14.12 NonDBayesCalibration Class Reference .............................................. 622
14.12 NonDC3FunctionTrain Class Reference ............................................... 634
14.12 NonDCalibration Class Reference ......................................................... 636
14.12 NonDCubature Class Reference ............................................................. 638
14.12 NonDCalibration Class Reference ......................................................... 636
14.13 NonDExpansion Class Reference .......................................................... 644
14.13 NonDGlobalEvidence Class Reference ................................................ 656
14.13 NonDGlobalInterval Class Reference .................................................. 657
14.13 NonDGlobalReliability Class Reference ............................................. 661
14.13 NonDGlobalSingleInterval Class Reference ......................................... 664
14.13 NonDGPImpSampling Class Reference .............................................. 665
14.13 NonDGPMSABayesCalibration Class Reference ..................................... 668
14.13 NonDIIntegration Class Reference ....................................................... 672
14.13 NonDInterval Class Reference ............................................................... 676
14.13 NonDLHSEvidence Class Reference ..................................................... 678
14.13 NonDLHSGlobalEvidence Class Reference ......................................... 679
14.13 NonDLHSInterval Class Reference ...................................................... 681
14.13 NonDLHSSampling Class Reference ................................................... 681
14.13 NonDLHSSingleInterval Class Reference ........................................... 685
14.13 NonDLocalEvidence Class Reference ................................................ 686
14.13 NonDLocalInterval Class Reference ................................................... 687
14.13 NonDLocalReliability Class Reference ................................................. 690
14.13 NonDLocalSingleInterval Class Reference ......................................... 700
14.13 NonDMultilevelFunctionTrain Class Reference .................................... 702
14.13 NonDMultilevelPolynomialChaos Class Reference .............................. 705
14.13 NonDMultilevelSampling Class Reference .......................................... 709
14.13 NonDMultilevelStochCollocation Class Reference ................................ 721
14.13 NonDMUQBayesCalibration Class Reference ...................................... 723
14.13 NonDPOLSFDarts Class Reference ....................................................... 726
14.13 NonDPolynomialChaos Class Reference .............................................. 729
14.13 NonDQuadrature Class Reference ....................................................... 734
14.13 NonDQUESOBayesCalibration Class Reference ................................... 739
14.13 NonDReliability Class Reference .......................................................... 745
14.13 NonDRKDDarts Class Reference ........................................................... 747
14.13 NonDSampling Class Reference ............................................................ 750
14.13 NonDSparseGrid Class Reference ........................................................ 760
14.13 NonDStochCollocation Class Reference .............................................. 763
14.13 NonDSurrogateExpansion Class Reference ........................................ 767
14.13 NonDWasABIBayesCalibration Class Reference .................................. 768
14.13 NonlinearCGOptimizer Class Reference ............................................ 771
14.13 NonlinearCGTraits Class Reference ..................................................... 774
14.13 NormalizationScaler Class Reference .................................................. 774
14.13 NoScaler Class Reference ..................................................................... 775
14.13 NOWPACBlackBoxEvaluator Class Reference ..................................... 776
| 14.170NOWPACOptimizer Class Reference | 777 |
| 14.170NOWPACTraits Class Reference | 779 |
| 14.170PSOLTraits Class Reference | 780 |
| 14.170OptDartsOptimizer Class Reference | 781 |
| 14.170OptDartsTraits Class Reference | 783 |
| 14.170Optimizer Class Reference | 784 |
| 14.170outputManager Class Reference | 790 |
| 14.170outputWriter Class Reference | 793 |
| 14.170ParallelConfiguration Class Reference | 794 |
| 14.170ParallelDirectApplicInterface Class Reference | 796 |
| 14.180ParallelLevel Class Reference | 798 |
| 14.180ParallelLibrary Class Reference | 800 |
| 14.180ParamResponsePair Class Reference | 810 |
| 14.180ParamStudy Class Reference | 813 |
| 14.180Partial_prp_equality Struct Reference | 819 |
| 14.180Partial_prp_hash Struct Reference | 820 |
| 14.180PebblDBranching Class Reference | 820 |
| 14.180PebblDBranchSub Class Reference | 821 |
| 14.180PebblDTraits Class Reference | 822 |
| 14.180PecosApproximation Class Reference | 823 |
| 14.190PolynomialRegression Class Reference | 829 |
| 14.190PrefixingLineFilter Class Reference | 833 |
| 14.190ProbabilityTransformModel Class Reference | 834 |
| 14.190ProblemDescDB Class Reference | 838 |
| 14.190ProcessApplicInterface Class Reference | 849 |
| 14.190ProcessHandleApplicInterface Class Reference | 854 |
| 14.190ProgramOptions Class Reference | 858 |
| 14.190ParameterStudy Class Reference | 863 |
| 14.190PSUADEDesignCompExp Class Reference | 865 |
| 14.190PythonInterface Class Reference | 868 |
| 14.200QMEApproximation Class Reference | 870 |
| 14.200QuesoJointPdf< V, M > Class Template Reference | 872 |
| 14.200QuesoVectorRV< V, M > Class Template Reference | 873 |
| 14.200RandomFieldModel Class Reference | 874 |
| 14.200RealScale Struct Reference | 879 |
| 14.200RecastModel Class Reference | 880 |
| 14.200ReducedBasis Class Reference | 890 |
| 14.200RelaxedVarConstraints Class Reference | 891 |
| 14.200RelaxedVariables Class Reference | 892 |
| 14.200Response Class Reference | 894 |
| 14.201RestartWriter Class Reference | 902 |
| 14.202ResultAttribute< T > Struct Template Reference | 903 |
| 14.202ResultDBAny Class Reference | 904 |
| 14.202ResultDBBase Class Reference | 906 |
| 14.202ResultDBHDF5 Class Reference | 907 |
| 14.202ResultEntry< StoredType > Class Template Reference | 909 |
| 14.202ResultFileError Class Reference | 910 |
| 14.202ResultsManager Class Reference | 911 |
| 14.202ResultsNames Class Reference | 913 |
Chapter 1

Dakota Developers Manual

Author


1.1 Introduction

The Dakota software (http://dakota.sandia.gov/) delivers advanced parametric analysis techniques enabling quantification of margins and uncertainty, risk analysis, model calibration, and design exploration with computational models. Dakota contains algorithms for optimization with gradient and nongradient-based methods, uncertainty quantification with sampling, reliability, stochastic expansion, and interval estimation methods, parameter estimation with nonlinear least squares methods, and sensitivity/variance analysis with design of experiments and parameter study capabilities. (Solution verification and Bayesian approaches are also in development.) These capabilities may be used on their own or as components within advanced algorithms such as surrogate-based optimization, mixed integer nonlinear programming, mixed aleatory-epistemic uncertainty quantification, 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 problem-solving environment for design and performance analysis of computational models on high performance computers.

The Developers Manual focuses on documentation of Dakota design principles and class structures; it derives principally from annotated source code. For information on input command syntax, refer to the Reference Manual[1], and for more details on Dakota features and capabilities, refer to the Users Manual.

1.2 Overview of Dakota

In Dakota, the environment manages execution modes and input/output streams and defines the top-level iterator. This top-level iterator may be either a standard iterator or a meta-iterator. In the former case, the iterator identifies a model and the environment executes the iterator on the model to perform a single study. In the latter case, iterator recursions are present and sub-iterators may identify their own models. In both cases, models may contain additional recursions in the case of nested iteration or surrogate modeling. In a simple example, a hybrid meta-iterator might manage a global optimizer operating on a low-fidelity model that feeds promising design points into a local optimizer operating on a high-fidelity model. And in a more advanced example, a surrogate-based
optimization under uncertainty approach would employ an uncertainty quantification iterator nested within an
optimization iterator and would employ truth models contained within surrogate models. Thus, iterators and
models provide both stand-alone capabilities as well as building blocks for more sophisticated studies.

A model contains a set of variables, an interface, and a set of responses, and the iterator operates on the
model to map the variables into responses using the interface. Each of these components is a flexible abstraction
with a variety of specializations for supporting different types of iterative studies. In a Dakota input file, the user
specifies these components through environment, method, model, variables, interface, and responses keyword
specifications.

The use of class hierarchies provides a mechanism for extensibility in Dakota components. In each of the
various class hierarchies, adding a new capability typically involves deriving a new class and providing a set of
virtual function redefinitions. These redefinitions define the coding portions specific to the new derived class,
with the common portions already defined at the base class. Thus, with a small amount of new code, the existing
facilities can be extended, reused, and leveraged for new purposes. The following sections tour Dakota’s class
organization.

1.2.1 Environment

Class hierarchy: Environment.

Environments provide the top level abstraction for managing different execution modes and managing input
and output streams. Specific environments include:

- **ExecutableEnvironment**: the environment for execution of Dakota as a stand-alone application.
- **LibraryEnvironment**: the environment for execution of Dakota as an embedded library service.

1.2.2 Iterators

Class hierarchy: Iterator. Iterator implementations may choose to split operations up into run-time phases as
described in Understanding Iterator Flow.

The iterator hierarchy contains a variety of iterative algorithms for optimization, uncertainty quantification,
nonlinear least squares, design of experiments, and parameter studies. The hierarchy is divided into MetaIterator,
Minimizer, and Analyzer branches.

The MetaIterator classes manage sequencing and collaboration among multiple methods with support for
concurrent iterator parallelism. Methods include:

- **SeqHybridMetaIterator**: hybrid minimization using a set of iterators employing a corresponding set of
  models of varying fidelity. The sequential hybrid passes the best solutions from one method in as the
  starting points of the next method in the sequence.
- **CollabHybridMetaIterator**: hybrid minimization employing collaboration and sharing of response data
  among methods during the course if iteration. This class is currently a placeholder.
- **EmbedHybridMetaIterator**: hybrid minimization involving periodic use of a local search method for refine-
  ment during the iteration of an outer global method. This class is currently a placeholder.
- **ConcurrentMetaIterator**: two similar algorithms are available: (1) multi-start iteration from several different
  starting points, and (2) pareto set optimization for several different multi-objective weightings. Employs
  a single iterator with a single model, but runs multiple instances of the iterator concurrently for different
  settings within the model.

The Minimizer classes address optimization and deterministic calibration and are grouped into:
1.2. OVERVIEW OF DAKOTA

- Optimization: Optimizer provides a base class for gradient-based (e.g., CONMINOptimizer and SNLOptimizer) and derivative-free (e.g., NCSUOptimizer, JEGAOptimizer) optimization solvers. Most of these are wrappers for third-party libraries that implement the optimization algorithms. Classes APPS-EvalMgr and COLINApplication provide the function evaluation interface for APPSOptimizer and COLINOptimizer, respectively.

- Parameter estimation: LeastSq provides a base class for NL2SOLLeastSq, a least-squares solver based on NL2SOL, SNLLLeastSq, a Gauss-Newton least-squares solver, and NLSSOLLeastSq, an SQP-based least-squares solver.

- Surrogate-based minimization (both optimization and nonlinear least squares): SurrBasedMinimizer provides a base class for SurrBasedLocalMinimizer, SurrBasedGlobalMinimizer, and EffGlobalMinimizer. The surrogate-based local and global methods employ a single iterator with any of the available Surrogate-Model capabilities (local, multipoint, or global data fits or hierarchical approximations) and perform a sequence of approximate optimizations, each involving build, optimize, and verify steps. The efficient global method, on the other hand, hard-wires a recursion involving Gaussian process surrogate models coupled with the DIRECT global optimizer to maximize an expected improvement function.

The Analyzer classes are grouped into:

- Uncertainty quantification: NonD provides a base class for non-deterministic methods in several categories:
  - Sampling: NonDSampling is further specialized with the NonDLHSSampling class for Latin hypercube and Monte Carlo sampling, and a number of other classes supporting incremental and adaptive sampling such as NonDAadaptImpSampling for multi-modal adaptive importance sampling.
  - Reliability Analysis: NonDReliability is further specialized with local and global methods (NonDLocalReliability and NonDGlobalReliability). NonDPOFDarts implements a computational geometry-based reliability method.
  - Stochastic Expansions: NonDExpansion includes specializations for generalized polynomial chaos (NonDPolynomialChaos) and stochastic collocation (NonDStochCollocation) and is supported by the NonDIntegration helper class, which supplies cubature, tensor-product quadrature and Smolyak sparse grid methods (NonDCubature, NonDQuadrature, and NonDSParseGrid).
  - Bayesian Calibration: NonDCalibration provides a base class for nondeterministic calibration methods with specialization to Bayesian calibration in NonDBayesCalibration, and specific implementations such as NonDQUESTBayesCalibration.
  - NonDInterval provides a base class for epistemic interval-based UQ methods. Three interval analysis approaches are provided: LHS (NonDLHSInterval), efficient global optimization (NonDGlobalInterval), and local optimization (NonDLocalInterval). Each of these three has specializations for single interval and Dempster-Shafer Theory of Evidence approaches.

- Parameter studies and design of experiments: PStudyDACE provides a base class for ParamStudy, which provides capabilities for directed parameter space interrogation, PSUADEDesignCompExp, which provides access to the Morris One-At-a-Time (MOAT) method for parameter screening, and DDACEDesignCompExp and FSUDesignCompExp, which provide for parameter space exploration through design and analysis of computer experiments. NonDLHSSampling from the uncertainty quantification branch also supports design of experiments when in active all variables mode.

- Solution verification studies: Verification provides a base class for RichExtrapVerification (verification via Richardson extrapolation) and other solution verification methods in development.
1.2.3 Models

Class hierarchy: Model.

The model classes are responsible for mapping variables into responses when an iterator makes a function evaluation request. There are several types of models, some supporting sub-iterators and sub-models for enabling layered and nested relationships. When sub-models are used, they may be of arbitrary type so that a variety of recursions are supported.

- SimulationModel: variables are mapped into responses using a simulation-based Interface object. No sub-iterators or sub-models are used.

- SurrogateModel: variables are mapped into responses using an approximation. The approximation is built and/or corrected using data from a sub-model (the truth model) and the data may be obtained using a sub-iterator (a design of experiments iterator). SurrogateModel has two derived classes: DataFitSurrModel for data fit surrogates and HierarchSurrModel for hierarchical models of varying fidelity. The relationship of the sub-iterators and sub-models is considered to be "layered" since they are not used as part of every response evaluation on the top level model, but rather used periodically in surrogate update and verification steps.

- NestedModel: variables are mapped into responses using a combination of an optional Interface and a sub-iterator/sub-model pair. The relationship of the sub-iterators and sub-models is considered to be "nested" since they are used to perform a complete iterative study as part of every response evaluation on the top level model.

- RecastModel: recasts the inputs and outputs of a sub-model for the purposes of variable transformations (e.g., variable scaling, transformations to standardized random variables) and problem reformulation (e.g., multi-objective optimization, response scaling, augmented Lagrangian merit functions, expected improvement).

1.2.4 Variables

Class hierarchy: Variables.

The Variables class hierarchy manages design, aleatory uncertain, epistemic uncertain, and state variable types for continuous, discrete integer, and discrete real domain types. This hierarchy is specialized according to how the domain types are managed:

- MixedVariables: domain type distinctions are retained, such that separate continuous, discrete integer, and discrete real domain types are managed. This is the default Variable perspective, and draws its name from “mixed continuous-discrete” optimization.

- RelaxedVariables: domain types are combined through relaxation of discrete constraints; i.e., continuous and discrete variables are merged into continuous arrays through relaxation of integrality (for discrete integer ranges) or set membership (for discrete integer or discrete real sets) requirements. The branch and bound minimizer is the only method using this approach at present.

Whereas domain types are defined based on the derived Variables class selection, the selection of active variable types is handled within each of these derived classes using variable views. These permit different algorithms to work on different subsets of variables. Data shared among Variables instances is stored in SharedVariablesData. For details on managing variables, see Working with Variable Containers and Views.

The Constraints hierarchy manages bound, linear, and nonlinear constraints and utilizes the same specializations for managing bounds on the variables (see MixedVarConstraints and RelaxedVarConstraints).
1.2. OVERVIEW OF DAKOTA

1.2.5 Interfaces

Class hierarchy: Interface.

Interfaces provide access to simulation codes or, conversely, approximations based on simulation code data. In the simulation case, an ApplicationInterface is used. ApplicationInterface is specialized according to the simulation invocation mechanism, for which the following nonintrusive approaches are supported:

- **SysCallApplicInterface**: the simulation is invoked using a system call (the C function `system()`). Asynchronous invocation utilizes a background system call. Utilizes the `CommandShell` utility.

- **ForkApplicInterface**: the simulation is invoked using a fork (the `fork/exec/wait` family of functions). Asynchronous invocation utilizes a nonblocking fork.

- **SpawnApplicInterface**: for Windows, fork is replaced by spawn. Asynchronous invocation utilizes a nonblocking spawn.

Fork and Spawn are inherited from ProcessHandleApplicInterface and System and ProcessHandle are inherited from ProcessApplicInterface. A semi-intrusive approach is also supported by:

- **DirectApplicInterface**: the simulation is linked into the Dakota executable and is invoked using a procedure call. Asynchronous invocations will utilize nonblocking threads (capability not yet available). Specializations of the direct interface are implemented in MatlabInterface, PythonInterface, ScilabInterface, and (for built-in testers) TestDriverInterface, while examples of plugin interfaces for library mode in serial and parallel, respectively, are included in SerialDirectApplicInterface and ParallelDirectApplicInterface.

Scheduling of jobs for asynchronous local, message passing, and hybrid parallelism approaches is performed in the ApplicationInterface class, with job initiation and job capture specifics implemented in the derived classes.

In the approximation case, global, multipoint, or local data fit approximations to simulation code response data can be built and used as surrogates for the actual, expensive simulation. The interface class providing this capability is

- **ApproximationInterface**: builds an approximation using data from a truth model and then employs the approximation for mapping variables to responses. This class contains an array of Approximation objects, one per response function, which support a variety of approximation types using the different Approximation derived classes. These include SurfpackApproximation (provides kriging, MARS, moving least squares, neural network, polynomial regression, and radial basis functions), GaussProcApproximation (Gaussian process models), PecosApproximation (multivariate orthogonal and Lagrange interpolation polynomials from Pecos), TANA3Approximation (two-point adaptive nonlinearity approximation), and TaylorApproximation (local Taylor series).

which is an essential component within the DataFitSurrModel capability described above in Models.

1.2.6 Responses

Class: Response.

The Response class provides an abstract data representation of response functions and their first and second derivatives (gradient vectors and Hessian matrices). These response functions can be interpreted as objective functions and constraints (optimization data set), residual functions and constraints (least squares data set), or generic response functions (uncertainty quantification data set). This class is not currently part of a class hierarchy, since the abstraction has been sufficiently general and has not required specialization.
1.3 Services

A variety of services and utilities are used in Dakota for parallel computing, failure capturing, restart, graphics, etc. An overview of the classes and member functions involved in performing these services is included here.

- **Multilevel parallel computing:** Dakota supports multiple levels of nested parallelism. A meta-iterator can manage concurrent iterators, each of which manages concurrent function evaluations, each of which manages concurrent analyses executing on multiple processors. Partitioning of these levels with MPI communicators is managed in `ParallelLibrary` and scheduling routines for the levels are part of `IteratorScheduler`, `ApplicationInterface`, and `ForkApplicInterface`.

- **Option management:** Global options controlling behavior are managed in `ProgramOptions`, with the help of command-line option parsing in `CommandLineHandler`.

- **Parsing:** Dakota employs NIDR (New Input Deck Reader) via `Dakota::ProblemDescDB::parse_inputs` to parse user input files. NIDR uses the keyword handlers in the `NIDRProblemDescDB` derived class to populate data within the `ProblemDescDB` base class, which maintains a `DataEnvironment` specification and lists of `DataMethod`, `DataModel`, `DataVariables`, `DataInterface`, and `DataResponses` specifications. Procedures for modifying the parsing subsystem are described in `Instructions for Modifying Dakota’s Input Specification`.

- **Failure capturing:** Simulation failures can be trapped and managed using exception handling in `ApplicationInterface` and its derived classes.

- **Restart:** Dakota maintains a record of all function evaluations both in memory (for capturing any duplication) and on the file system (for restarting runs). Restart options are managed through `ProgramOptions` (with the help of `CommandLineHandler`); file management in `OutputManager`; and restart file insertions occur in `ApplicationInterface`. The `dakota_restart_util` executable, built from `restart_util.cpp`, provides a variety of services for interrogating, converting, repairing, concatenating, and post-processing restart files.

- **Memory management:** Dakota employs the techniques of reference counting and representation sharing through the use of letter-envelope and handle-body idioms (Coplien, “Advanced C++”). The former idiom provides for memory efficiency and enhanced polymorphism in the following class hierarchies: `Environment`, `Iterator`, `Model`, `Variables`, `Constraints`, `Interface`, `ProblemDescDB`, and `Approximation`. The latter idiom provides for memory efficiency in data-intensive classes which do not involve a class hierarchy. The `Response` and parser data (`DataEnvironment`, `DataMethod`, `DataModel`, `DataVariables`, `DataInterface`, and `DataResponses`) classes use this idiom. When managing reference-counted data containers (e.g., `Variables` or `Response` objects), it is important to properly manage shallow and deep copies, to allow for both efficiency and data independence as needed in a particular context.

- **Graphics and Output:** Dakota provides 2D iteration history graphics using Motif widgets. Graphics data can also be cataloged in a tabular data file for post-processing with 3rd party tools such as Matlab, Tecplot, etc. These capabilities are encapsulated within the `Graphics` class. An experimental results database is implemented in `ResultsManager` and `ResultsDBAny`. Options for controlling output and facilities for managing it are in `OutputManager`.

1.4 Development Practices and Guidance

The following links provide guidance for core software components or specific development activities:

- **Coding Style Guidelines and Conventions** - coding practices used by the Dakota development team.
• Instructions for Modifying Dakota’s Input Specification - how to interact with NIDR and the associated Dakota classes.

• Interfacing with Dakota as a Library - embed Dakota as a service within your application.

• Understanding Iterator Flow - explanation of the full granularity of steps in Iterator execution.

• Performing Function Evaluations - an overview of the classes and member functions involved in performing function evaluations synchronously or asynchronously.

• Working with Variable Containers and Views - discussion of data storage for variables and explanation of active and inactive views of this data.

• Demo TPL - a README for bringing a new Third-Party Library (TPL) into Dakota

1.5 Additional Resources

Additional development resources include:

• The Dakota Developer Portal linked from http://dakota.sandia.gov/content/developer-portal/ includes information on getting started as a developer and links to project management resources.

• Project web pages are maintained at http://dakota.sandia.gov/ including links to frequently asked questions, documentation, publications, mailing lists, and other resources.

• A Quickstart for bringing a new Third-Party Library (TPL) into Dakota can be found in the Dakota source tree under $DAKOTA_SRC/packages/external/demo_tpl/README.md.
Chapter 2

Coding Style Guidelines and Conventions

2.1 Introduction

Common code development practices can be extremely useful in multiple developer environments. Particular styles for code components lead to improved readability of the code and can provide important visual cues to other developers. Much of this recommended practices document is borrowed from the CUBIT mesh generation project, which in turn borrows its recommended practices from other projects, yielding some consistency across Sandia projects. While not strict requirements, these guidelines suggest a best-practices starting point for coding in Dakota.

2.2 C++/c Style Guidelines

Style guidelines involve the ability to discern at a glance the type and scope of a variable or function.

2.2.1 Class and variable styles

Class names should be composed of two or more descriptive words, with the first character of each word capitalized, e.g.:

```cpp
class ClassName;
```

Class member variables should be composed of two or more descriptive words, with the first character of the second and succeeding words capitalized, e.g.:

```cpp
double classMemberVariable;
```

Temporary (i.e. local) variables are lower case, with underscores separating words in a multiple word temporary variable, e.g.:

```cpp
int temporary_variable;
```

Constants (i.e. parameters) and enumeration values are upper case, with underscores separating words, e.g.:

```cpp
const double CONSTANT_VALUE;
```
2.2.2 Function styles

Function names are lower case, with underscores separating words, e.g.:

```cpp
int function_name();
```

There is no need to distinguish between member and non-member functions by style, as this distinction is usually clear by context. This style convention allows member function names which set and return the value of a similarly-named private member variable, e.g.:

```cpp
int memberVariable;
void member_variable(int a) { // set
    memberVariable = a;
}
int member_variable() const { // get
    return memberVariable;
}
```

In cases where the data to be set or returned is more than a few bytes, it is highly desirable to employ const references to avoid unnecessary copying, e.g.:

```cpp
void continuous_variables(const RealVector& c_vars) { // set
    continuousVariables = c_vars;
}
const RealVector& continuous_variables() const { // get
    return continuousVariables;
}
```

Note that it is not necessary to always accept the returned data as a const reference. If it is desired to be able change this data, then accepting the result as a new variable will generate a copy, e.g.:

```cpp
const RealVector& c_vars = model.continuous_variables();
RealVector c_vars = model.continuous_variables();
```

2.2.3 Miscellaneous

Appearance of typedefs to redefine or alias basic types is isolated to a few header files (`data_types.h`, `template_defs.h`), so that issues like program precision can be changed by changing a few lines of typedefs rather than many lines of code, e.g.:

```cpp
typedef double Real;
```

`xemacs` is the preferred source code editor, as it has C++ modes for enhancing readability through color (turn on "Syntax highlighting"). Other helpful features include "Paren highlighting" for matching parentheses and the "New Frame" utility to have more than one window operating on the same set of files (note that this is still the same edit session, so all windows are synchronized with each other). Window width should be set to 80 internal columns, which can be accomplished by manual resizing, or preferably, using the following alias in your shell resource file (e.g., `.cshrc`):

```bash
alias xemacs "xemacs -g 81x63"
```

where an external width of 81 gives 80 columns internal to the window and the desired height of the window will vary depending on monitor size. This window width imposes a coding standard since you should avoid line wrapping by continuing anything over 80 columns onto the next line.

Indenting increments are 2 spaces per indent and comments are aligned with the code they describe, e.g.:
2.3. FILE NAMING CONVENTIONS

```c
void abort_handler(int code)
{
    int initialized = 0;
    MPI_Init(&initialized);
    if (initialized) {
        // comment aligned to block it describes
        int size;
        MPI_Comm_size(MPI_COMM_WORLD, &size);
        if (size>1)
            MPI_Abort(MPI_COMM_WORLD, code);
        else
            exit(code);
    } else
        exit(code);
}
```

Also, the continuation of a long command is indented 2 spaces, e.g.:

```c
const String& iterator_scheduling = problem_db.get_string("strategy.iterator.scheduling");
```

and similar lines are aligned for readability, e.g.:

```c
cout << "Numerical gradients using " << finiteDiffStepSize*100. << "%" << finiteDiffType << " differences to be calculated by the " << methodSource << " finite difference routine." << endl;
```

Lastly, #ifdef’s are not indented (to make use of syntax highlighting in xemacs).

2.3 File Naming Conventions

In addition to the style outlined above, the following file naming conventions have been established for the Dakota project.

File names for C++ classes should, in general, use the same name as the class defined by the file. Exceptions include:

- with the introduction of the Dakota namespace, base classes which previously utilized prepended Dakota identifiers can now safely omit the identifiers. However, since file names do not have namespace protection from name collisions, they retain the prepended Dakota identifier. For example, a class previously named DakotaModel which resided in DakotaModel.cpp/hpp, is now Dakota::Model (class Model in namespace Dakota) residing in the same filenames. The retention of the previous filenames reduces the possibility of multiple instances of a Model.hpp causing problems. Derived classes (e.g., NestedModel) do not require a prepended Dakota identifier for either the class or file names.

- in a few cases, it is convenient to maintain several closely related classes in a single file, in which case the file name may reflect the top level class or some generalization of the set of classes (e.g., DakotaResponse.[CH] files contain Dakota::Response and Dakota::ResponseRep classes, and DakotaBinStream.[CH] files contain the Dakota::BiStream and Dakota::BoStream classes).

The type of file is determined by one of the four file name extensions listed below:

- **.hpp** A class header file ends in the suffix .hpp. The header file provides the class declaration. This file does not contain code for implementing the methods, except for the case of inline functions. Inline functions are to be placed at the bottom of the file with the keyword inline preceding the function name.

- **.cpp** A class implementation file ends in the suffix .cpp. An implementation file contains the definitions of the members of the class.
• .h A header file ends in the suffix .h. The header file contains information usually associated with procedures. Defined constants, data structures and function prototypes are typical elements of this file.

• .c A procedure file ends in the suffix .c. The procedure file contains the actual procedures.

2.4 Class Documentation Conventions

Class documentation uses the doxygen tool available from http://www.doxygen.org and employs the J-AVA-doc comment style. Brief comments appear in header files next to the attribute or function declaration. Detailed descriptions for functions should appear alongside their implementations (i.e., in the .cpp files for non-inlined, or in the headers next to the function definition for inlined). Detailed comments for a class or a class attribute must go in the header file as this is the only option.

NOTE: Previous class documentation utilities (class2frame and class2html) used the ”//-” comment style and comment blocks such as this:

```c
//- Class: Model
//- Description: The model to be iterated by the Iterator.
//- Contains Variables, Interface, and Response objects.
//- Owner: Mike Eldred
```

These tools are no longer used, so remaining comment blocks of this type are informational only and will not appear in the documentation generated by doxygen.

2.5 CMake Style Guidelines

Dakota conventions for CMake files, such as CMakeLists.txt, FooConfig.cmake, etc., follow. Our goal is ease of reading, maintenance, and support, similar to the C++ code itself. Current CMake versions and build hints are maintained at the Developer Portal http://dakota.sandia.gov/developer/.

2.5.1 CMake Code Formatting

• Indentation is 2 spaces, consistent with Dakota C++ style.

• Lines should be kept to less than 80 chars per line where possible.

• Wrapped lines may be indented two spaces or aligned with prior lines.

• For ease of viewing and correctness checking in Emacs, a customization file is available: http://www.-cctrace.org/CMakeDocs/cmake-mode.el

2.5.2 CMake Variable Naming Conventions

These variable naming conventions are especially important for those that ultimately become preprocessor defines and affect compilation of source files.

• Classic/core elements of the CMake language are set in lower_case, e.g., option, set, if, find_library.

• Static arguments to CMake functions and macros are set in UPPER_CASE, e.g. REQUIRED, NO_MODULE, QUIET.

• Minimize ”global” variables, i.e., don’t use 2 variables with the same meaning when one will do the job.
• Feature toggling: when possible, use the "HAVE_<pkg/feature>" convention already in use by many CMake-enabled TPLs, e.g.,

```csharp
$ grep HAVE_SYSTEM Dakota/src/CMakeLists.txt
```

```csharp
check_function_exists(system HAVE_SYSTEM)
if (HAVE_SYSTEM)
  add_definitions("-DHAVE_SYSTEM")
endif (HAVE_SYSTEM)
```

```csharp
$ grep HAVE_CONMIN Dakota/src/CMakeLists.txt Dakota/packages/CMakeLists.txt
```

```csharp
Dakota/src/CMakeLists.txt:if (HAVE_CONMIN)
Dakota/src/CMakeLists.txt:endif (HAVE_CONMIN)
Dakota/packages/CMakeLists.txt:option(HAVE_CONMIN "Build the CONMIN package." ON)
Dakota/packages/CMakeLists.txt:if (HAVE_CONMIN)
Dakota/packages/CMakeLists.txt:endif (HAVE_CONMIN)
```

• When a variable/preprocessor macro could result in name clashes beyond Dakota scope, e.g., for library-mode users, consider prefixing the "HAVE_<pkg>" name with DAKOTA_, e.g. DAKOTA_HAVE_MPI. Currently, MPI is the only use case for such a variable in Dakota, but many examples can be found in the CMake Modules source, e.g.

```bash
grep _HAVE_* <cmake_prefix_dir>/share/cmake-2.8/Modules/*
```
Chapter 3

Instructions for Modifying Dakota’s Input Specification

To modify Dakota’s input specification (for maintenance or addition of new input syntax), specification maintenance mode must be enabled at Dakota configure time with the `-DENABLE_SPEC_MAINT` option, e.g.,

```
./cmake -DENABLE_SPEC_MAINT:BOOL=ON ..
```

This will enable regeneration of NIDR and Dakota components which must be updated following a spec change.

3.1 XML Input Specification

The authoritative source for valid Dakota input grammar is `dakota/src/dakota.xml`. The schema defining valid content for this XML file is in `dakota/src/dakota.xsd`. NIDR remains Dakota’s user input file parser, so `dakota.xml` is translated to `dakota/src/dakota.input.nspec` during the Dakota build process. To update the XML input definition:

- Make sure `ENABLE_SPEC_MAINT` is enabled in your build and necessary Java development tools are installed (see below).
- Edit the XML spec in `dakota.xml`.
- Perform a make in `dakota.build/src` which will regenerate `dakota.source/src/dakota.input.nspec` and related file.
- Review that any changes induced in the `dakota.input.nspec` file are as expected.
- Proceed with verifying code changes and making downstream parse handler changes as normal (described below).
- Commit the modified `dakota.xml`, `dakota.input.nspec`, and other files generated to `dakota.source/src` along with your other code changes.

3.1.1 XML Build Requirements

Editing the XML and then compiling Dakota requires
• Java Development Kit (JDK) providing the Java compiler javac. Java 6 (version 1.6) or newer should work, with Java 8 recommended. Can satisfy on RHEL6 with RPM packages java-1.8.0-openjdk-devel and java-1.8.0-openjdk. This is needed to build the Java-based XML to NIDR translator. If this becomes too burdensome, we can check in the generated xml2nidr.jar file.

3.1.2 XML Editing Tools
The following tools will make editing dakota.input.xml easier.

• Recommended: Eclipse Web Tools Platform. Includes both graphical and text editors.

1. Download Eclipse Standard (Classic)
2. Configure proxy if needed, setting to manual: Window > Preferences > General > Network Connection > Proxy
3. Install Web Tools Platform
   - Help > Install New Software
   - Search "Eclipse X" and install two packages under Web, XML, Java
     * Eclipse XML Editors and Tools
     * Eclipse XSL Developer Tools
   - Optionally install C/C++ Development Tools
4. Optional: add Subclipse for subversion (Subversive is the other major competing tool and I don’t think requires JavaHL) Help > Install New Software Work With: http://subclipse.tigris.org/update_1.6.x Install Subclipse On Linux: yum install subversion-javahl.x86_64
5. Alternately install Eclipse for Java or Eclipse Java EE development which includes webtools, then optionally add subclipse and C/C++ dev

• Alternate: Emacs or your usual editor. For example, Emacs supports an Nxml mode. You can tell it where to find the schema, edit XML, and have it perform validation against the schema. See help at http://www.gnu.org/software/emacs/manual/html_mono/nxml-mode.html

• Other Suggested Alternates: XMLSpy, DreamWeaver, XML Copy Editor

3.1.3 XML Features (with map to NIDR)
Out of necessity, Dakota XML dakota.xml closely mirrors dakota.input.nspec. Valid Dakota input grammar is constrained by dakota.xml, an XML document which must validate against dakota.xsd. The top-level element of interest is <input>, which is comprised of a sequence of content elements (keywords, alternates, etc.), which may themselves contain additional child content elements. The key content types are:

• Keyword (<keyword>), specified with the <keyword> element whose definition is given by keyword-Type in dakota.xsd. The required attributes are:
  – name: The keyword name (lower case with underscores) as it will be given in user input; must follow same uniqueness rules are historical NIDR. User input is allowed in mixed case, but the XML must use lower case names.
  Since the NIDR parser allows keyword abbreviation, you must not add a keyword that could be mis-interpreted as an abbreviation for a different keyword within the same top-level keyword, such as
3.1. XML INPUT SPECIFICATION

"environment" and "method". For example, adding the keyword "expansion" within the method specification would be a mistake if the keyword "expansion_factor" already was being used in this block. The NIDR input is somewhat order-dependent, allowing the same keyword to be reused multiple times in the specification. This often happens with aliases, such as lower_bounds, upper_bounds and initial_point. Ambiguities are resolved by attaching a keyword to the most recently seen context in which it could appear, if such exists, or to the first relevant context that subsequently comes along in the input file.

- **code**: The verbatim NIDR handler to be invoked when this keyword parsed. In NIDR this was specified with \{N_macro(...)}.

Optional/useful parser-related elements/attributes in order of importance are:

- **param sub-element**: Parameters and data types: A keyword may have an associated parameter element with a specified data type: `<param type="PARAMTYPE" />`. NIDR data types remain the same (INTEGER, REAL, STRING and LISTS thereof, but new data types INPUT_FILE and OUTPUT_FILE add convenience for the GUI, mapping to STRING for NIDR purposes. Parameters can also include attributes constraint, in_taglist, or taglist, which are used to help validate the user-specified parameter value. For example `constraint >= 0 LEN normal uncertain`

- **alias sub-element**: Historical aliases for this keyword (can appear multiple times). Alias has a single attribute name which must be lower case with underscores.

- **id**: Unique ID for the keyword, usually name with an integer appended, but not currently used/enforced.

- **minOccurs**: Minimum occurrences of the keyword in current context (set to 1 for required, 0 for optional)

- **maxOccurs**: Maximum occurrences of the keyword in current context (for example environment may appear at most once)

And optional/useful GUI-related attributes are:

- **help**: (Don’t add this attribute the new keywords!) A pointer to the corresponding reference manual section (deprecated as not needed with new reference manual format which mirrors keyword hierarchy).

- **label**: a short, friendly label string for the keyword in the GUI. Format these like titles, e.g., "Initial Point for Search".

- **group**: Category or group for this keyword, e.g., optimization vs. parameter study if they are to be groups for GUI purposes

- **Alternation** (`<oneOf>`): Alternation of groups of content is done with the element `<oneOf>` which indicates that its immediate children are alternates. In NIDR this was done with the pipe symbol: OptionA | OptionB. oneOf allows the label attribute and its use is recommended.

- **Required Group** (`<required>`): A required group can be specified by enclosing the contents in the `<required>` element. In NIDR this was done by enclosing the content in parentheses: ( required group... )

- **Optional Group** (`<optional>`): An optional group can be specified by enclosing the contents in the `<optional>` element. In NIDR this was done by enclosing the content in brackets: [ optional group... ]
3.2 Rebuild Generated Files

When configured with `-DENABLE_SPEC_MAINT`, performing a `make` in `dakota.build/src` will regenerate all files which derive from `dakota.xml`, include `dakota.input.nspec`, `NIDR_keywds.hpp`, and `dakota.input.-summary`. If you commit changes to a source repository, be sure to commit any automatically generated files in addition to any modified in the following steps. It is not strictly necessary to run `make` at this point in the sequence, and in fact may generate errors if necessary handlers aren’t yet available.

Warning

Please do not manually modify generated files!

3.3 Update Parser Source NIDRProblemDescDB.cpp

Many keywords have data associated with them: an integer, a floating-point number, a string, or arrays of such entities. Data requirements are specified in `dakota.input.nspec` by the tokens `INTEGER`, `REAL`, `STRING`, `INTEGERLIST`, `REALLIST`, `STRINGLIST`. (Some keywords have no associated data and hence no such token.) After each keyword and data token, the `dakota.input.nspec` file specifies functions that the NIDR parser should call to record the appearance of the keyword and deal with any associated data. The general form of this specification is

\[
\{
\text{startfcn}, \text{startdata}, \text{stopfcn}, \text{stopdata}
\}
\]

i.e., a brace-enclosed list of one to four functions and data pointers, with trailing entities taken to be zero if not present; zero for a function means no function will be called. The startfcn must deal with any associated data. Otherwise, the distinction between startfcn and stopfcn is relevant only to keywords that begin a group of keywords (enclosed in parentheses or square brackets). The startfcn is called before other entities in the group are processed, and the stop function is called after they are processed. Top-level keywords often have both startfcn and stopfcn; stopfcn is uncommon but possible for lower-level keywords. The startdata and (if needed) stopdata values are usually pointers to little structures that provide keyword-specific details to generic functions for startfcn and stopfcn. Some keywords that begin groups (such as "approx_problem" within the top-level "environment" keyword) have no need of either a startfcn or a stopfcn; this is indicated by "\{0\}".

Most of the things within braces in `dakota.input.nspec` are invocations of macros defined in `dakota.source/src/NIDRProblemDescDB.cpp`. The macros simplify writing `dakota.input.nspec` and make it more readable. Most macro invocations refer to little structures defined in NIDRProblemDescDB.cpp, usually with the help of other macros, some of which have different definitions in different parts of NIDRProblemDescDB.cpp. When adding a keyword to `dakota.input.nspec`, you may need to add a structure definition or even introduce a new data type. NIDRProblemDescDB.cpp has sections corresponding to each top-level keyword. The top-level keywords are in alphabetical order, and most entities in the section for a top-level keyword are also in alphabetical order. While not required, it is probably good practice to maintain this structure, as it makes things easier to find.

Any integer, real, or string data associated with a keyword are provided to the keyword’s startfcn, whose second argument is a pointer to a `Values` structure, defined in header file `nidr.h`.

Example 1: if you added the specification:

\[
\text{[method_setting REAL \{method_setting_start, &method_setting_details\} ]}
\]

you would provide a function

```cpp
void NIDRProblemDescDB::
method_setting_start(const char *keyname, Values *val, void **g, void *v)
{ ... }
```

in NIDRProblemDescDB.cpp. In this example, argument `&method_setting_details` would be passed as `v`, `val->n` (the number of values) would be 1 and `*val->r` would be the REAL value given for the `method_setting` keyword. The `method_setting_start` function would suitably store this value with the help of `method_setting_details`. 
For some top-level keywords, \( g \) (the third argument to the startfcn and stopfcn) provides access to a relevant context. For example, `method_start` (the startfcn for the top-level `method` keyword) executes

```c
DataMethod *dm = new DataMethod;
g = (void*)dm;
```

(and supplies a couple of default values to \( dm \)). The start functions for lower-level keywords within the `method` keyword get access to \( dm \) through their \( g \) arguments. Here is an example:

```c
void NIDRProblemDescDB::method_start(const char *keyname, Values *val, void **g, void *v)
{
    (*(DataMethod**)g)->**(String DataMethod::**)v = *val->s;
}
```

In this example, \( v \) points to a pointer-to-member, and an assignment is made to one of the components of the `DataMethod` object pointed to by \( *g \). The corresponding stopfcn for the top-level `method` keyword is

```c
void NIDRProblemDescDB::method_stop(const char *keyname, Values *val, void **g, void *v)
{
    DataMethod *p = *(DataMethod**)g;
pDDBInstance->dataMethodList.insert(*p);
delete p;
}
```

which copies the now populated `DataMethod` object to the right place and cleans up.

**Example 2:** if you added the specification

```c
[method_setting REALLIST {{N_mdm(RealL,methodCoeffs)
```

then `method_RealL` (defined in NIDRProblemDescDB.cpp) would be called as the startfcn, and `methodCoeffs` would be the name of a (currently nonexistent) component of `DataMethod`. The `N_mdm` macro is defined in NIDRProblemDescDB.cpp; among other things, it turns `RealL` into `NIDRProblemDescDB::method_RealL`. This function is used to process lists of REAL values for several keywords. By looking at the source, you can see that the list values are `val->r[i]` for `0 <= i < val->n`.

### 3.4 Update Corresponding Data Classes

The Data classes (DataEnvironment, DataMethod, DataModel, DataVariables, DataInterface, and DataResponses) store the parsed user input data. In this step, we extend the Data class definitions to include any new attributes referred to in dakota.xml or NIDRProblemDescDB.

#### 3.4.1 Update the Data Class Header File

Add a new attribute to the public data for each of the new specifications. Follow the style guide for class attribute naming conventions (or mimic the existing code).

#### 3.4.2 Update the .cpp File

Define defaults for the new attributes in the constructor initialization list (if not a container with a sensible default constructor) in same order as they appear in the header. Add the new attributes to the `write(MPIPackBuffer&)`, `read(MPIUnpackBuffer&)`, and `write(ostream&)` functions, paying careful attention to the use of a consistent ordering.
3.5 Update Database Source ProblemDescDB.cpp

3.5.1 Augment/update get_<data_type>() Functions

The next update step involves extending the database retrieval functions in dakota.source/src/ProblemDescDB.cpp. These retrieval functions accept an identifier string and return a database attribute of a particular type, e.g., a RealVector:

```cpp
const RealVector& get_rv(const String& entry_name);
```

The implementation of each of these functions contains tables of possible entry_name values and associated pointer-to-member values. There is one table for each relevant top-level keyword, with the top-level keyword omitted from the names in the table. Since binary search is used to look for names in these tables, each table must be kept in alphabetical order of its entry names. For example,

```cpp
else if ({L = Begins(entry_name, "model.")}) {
    if (dbRep->methodDBLocked)
        Locked_db();
#define P &DataModelRep::
    static KW<RealVector, DataModelRep> RVdmo[] = {
        // must be sorted
        {"nested.primary.response.mapping", P primaryRespCoeffs},
        {"nested.secondary.response.mapping", P secondaryRespCoeffs},
        {"surrogate.kriging.conmin_seed", P krigingConminSeed},
        {"surrogate.kriging.correlations", P krigingCorrelations},
        {"surrogate.kriging.max_correlations", P krigingMaxCorrelations},
        {"surrogate.kriging.min_correlations", P krigingMinCorrelations}
    };
#undef P
    KW<RealVector, DataModelRep>* kw;
    if ((kw = (KW<RealVector, DataModelRep>*)Binsearch(RVdmo, L)))
        return dbRep->dataModelIter->dataModelRep->*kw->p;
}
```

is the "model" portion of ProblemDescDB::get_rv(). Based on entry_name, it returns the relevant attribute from a DataModel object. Since there may be multiple model specifications, the dataModelIter list iterator identifies which node in the list of DataModel objects is used. In particular, dataModelList contains a list of all of the data_model objects, one for each time a top-level model keyword was seen by the parser. The particular model object used for the data retrieval is managed by dataModelIter, which is set in a set_db_list_nodes() operation that will not be described here.

There may be multiple DataMethod, DataModel, DataVariables, DataInterface, and/or DataResponses objects. However, only one specification is currently allowed so a list of DataEnvironment objects is not needed. Rather, ProblemDescDB::environmentSpec is the lone DataEnvironment object.

To augment the get_<data_type>() functions, add table entries with new identifier strings and pointer-to-member values that address the appropriate data attributes from the Data class object. The style for the identifier strings is a top-down hierarchical description, with specification levels separated by periods and words separated with underscores, e.g., "keyword.group_specification.individual_specification". Use the dbRep->listIter->attribute syntax for variables, interface, responses, and method specifications. For example, the method_setting example attribute would be added to get_drv() as:

```cpp
{"method.name.method.setting", P methodSetting},
```

inserted at the beginning of the RVdmo array shown above (since the name in the existing first entry, i.e., "nested.primary_response_mapping", comes alphabetically after "method.name.method_setting").
3.6 Use \texttt{get_<data\_type>}() Functions

At this point, the new specifications have been mapped through all of the database classes. The only remaining step is to retrieve the new data within the constructors of the classes that need it. This is done by invoking the \texttt{get_<data\_type>}() function on the \texttt{ProblemDescDB} object using the identifier string you selected in \texttt{Augment/update get_<data\_type>}() Functions. For example:

\begin{verbatim}
const String\& interface\_type = problem\_db.get\_string("interface\.type");
\end{verbatim}

passes the "interface\.type" identifier string to the \texttt{ProblemDescDB::get\_string()} retrieval function, which returns the desired attribute from the active \texttt{Data\_Interface} object.

\textbf{Warning}

Use of the \texttt{get_<data\_type>}() functions is restricted to class constructors, since only in class constructors are the data list iterators (i.e., \texttt{dataMethodIter}, \texttt{dataModelIter}, \texttt{dataVariablesIter}, \texttt{data\_InterfaceIter}, and \texttt{dataResponsesIter}) guaranteed to be set correctly. Outside of the constructors, the database list nodes will correspond to the last set operation, and may not return data from the desired list node.

3.7 Update the Documentation

Doxygen comments should be added to the Data class headers for the new attributes, and the reference manual sections describing the portions of \texttt{dakota\.xml} that have been modified should be updated by updating files in \texttt{dakota\.source/docs/Keyword\_MetaData/}. \texttt{dakota\.xml}, together with these metadata files generates the reference manual and GUI context-aware help documentation.
Chapter 4

Understanding Iterator Flow

This page explains the various phases comprising `Iterator::run_iterator()`. Prior to `Iterator` construction, when command-line options are parsed, Boolean run mode flags corresponding to PRERUN, RUN, and POSTRUN are set in `ParallelLibrary`. If the user didn’t specify any specific run modes, the default is for all three to be true (all phases will execute).

`Iterator` is constructed.

When called, `run_iterator()` sequences:

- **initialize_run()**: unconditionally called, virtual. Performs common initialization such as allocating workspaces, setting communicators and evaluation counts. When re-implementing this virtual, a derived class must call its nearest parent’s `initialize_run()`, typically before performing its own implementation steps.

- **Not implemented: pre-run input**

- IF PRERUN, invoke `pre_run()`: virtual function; default no-op. Purpose: derived classes should implement `pre_run()` if they are able to generate all parameter sets (variables) at once, separate from `run()`. Derived implementations should call their nearest parent’s `pre_run()`, typically before performing their own steps.

- IF PRERUN, invoke `pre_output()`: non-virtual function; if user requested, output variables to file.

- **Not implemented: run input**

- IF RUN, invoke virtual function `run()`. Purpose: at a minimum, evaluate parameter sets through computing responses; for iterators without pre/post capability, their entire implementation is in `run()` and this is a reasonable default for new Iterators.

- **Not implemented: run output**

- IF POSTRUN, invoke `post_input()`: virtual function, default only print helpful message on mode. Purpose: derived iterators supporting post-run input from file must implement to read file and populate variables/responses (and possibly best points) appropriately. Implementations must check if the user requested file input.

- IF POSTRUN, invoke `post_run()`: virtual function. Purpose: generate statistics / final results. Any analysis that can be done solely on tabular data read by `post_input()` can be done here. Derived re-implementations should call their nearest parent’s `post_run()`, typically after performing their specific post-run activities.

- **Not implemented: post-run output**
• `finalize_run()`: unconditionally called, virtual. Purpose: free workspaces. Default base class behavior is no-op, however, derived implementations should call their nearest parent’s `finalize_run` after performing their specialized portions.

Iterator is destructed.
Chapter 5

Interfacing with Dakota as a Library

5.1 Introduction

Tightly integrating or linking Dakota into another application can improve user experience by delivering a more unified, inter-operable software tool for optimization and UQ analyses, improving performance by eliminating file system-based interfaces, and reducing challenges with parallel computing inter-operation. This benefit has been realized within several Sandia and external simulation applications. This section describes how to link Dakota into another C++ application.

Dakota has two primary application programming interfaces (APIs). The LibraryEnvironment class facilitates use of Dakota as an algorithm service library within another application. In this case, the simulation application is providing a "front end" for Dakota. The second API, provided by the DirectApplicInterface class, provides an interface for Dakota to call the simulation code directly to perform function evaluations in core. This permits the simulation to be the "back end" for Dakota. The most complete library integration of Dakota would use both in combination, with the overall simulation framework providing both the front end and back end for Dakota, creating a sandwich, as loosely depicted here:

```
[------------
| Application
| |
| { ----- |
| { Dakota (LibraryEnvironment)
| |
| { { Function evaluation callback to Application (via DirectApplicInterface)
| | { |
| | <------/
| |
| | |
| | |
| | { ----
| |
| ------------
```

Attention

Dakota may be integrated as a library in other software applications subject to the terms of the GNU Lesser General Public License (LGPL). Refer to http://www.gnu.org/licenses/lgpl.html or the LICENSE file included with Dakota.

When Dakota is compiled and installed, the relevant library API headers are installed to CMAKE_INSTALL_PREFIX/include and the runtime libraries primarily to CMAKE_INSTALL_PREFIX/lib/ (on some platforms,
CHAPTER 5. INTERFACING WITH DAKOTA AS A LIBRARY

to CMAKE_INSTALL_PREFIX/bin/. The core C/C++ code is in the library dakota_src, while Fortran code lives in the dakota_src_fortran library. Information on using the API in Dakota headers is included throughout this section, while considerations for configuring and linking against Dakota and its various required and optional third-party libraries are emphasized in the section Linking against the Dakota library.

Steps involved in integrating Dakota into another application typically include:

1. Writing C++ code for your application to instantiate, configure, and execute Dakota’s LibraryEnvironment ("front end"); see Basic Dakota library instantiation and Configuring Dakota operation.

2. Writing C++ code for Dakota to call a function in your application to perform function evaluations ("back end"); see Creating a simulator plugin interface.

3. Compiling Dakota and linking into your application (Linking against the Dakota library).

Several source code examples demonstrate Dakota library interfaces. The classes SIM::SerialDirectApplicInterface and SIM::ParallelDirectApplicInterface demonstrate serial and parallel simulation function evaluation plug-ins. The file library_mode.cpp includes a main program that exercises Dakota libraries in serial and parallel modes with these mock simulator programs, with various ways of configuring Dakota problem definition and operation. Finally, library_split.cpp demonstrates running Dakota as a library modular on an MPI sub-communicator.

5.2 Basic Dakota library instantiation

The function run_dakota_parse() in library_mode.cpp demonstrates the basic use of Dakota library objects as one would in another main application that embeds Dakota. In this example, Dakota is configured based on a typical user-provided text-based Dakota input file (the same that would be provided at the command line with dakota -i dakota_optimization.in) and a function evaluator derived from a DirectApplicInterface is plugged into the Dakota library environment.

First, an object of type ProgramOptions which manages top-level Dakota settings is instantiated and configured to specify the name of the Dakota user input file. Additional options for output and error redirection, restart operation, and more may be set via ProgramOptions. See its class documentation for details.

```cpp
string dakota_input_file = "dakota_optimization.in";
Dakota::ProgramOptions opts;
opts.input_file(dakota_input_file);
```

Next, a LibraryEnvironment is created, passing the desired settings from opts:

```cpp
Dakota::LibraryEnvironment env(opts);
```

This standard constructor will parse the specified input and create Dakota objects. It assumes many default settings, including that the parent application initialized MPI if running in parallel mode. (In this case, Dakota will detect whether MPI was initialized and not call MPI_Init or MPI_Finalize.) For more advanced use cases described below, alternate constructors allow constructing based on MPI communicators, with delayed finalization, and with Dakota database update function callbacks. Then the application’s function evaluator implementing Dakota’s DirectApplicInterface is plugged in with a convenience function serial_interface_plugin() or parallel_interface_-plugin(). Finally, the Dakota analysis is run by calling

```cpp
env.execute();
```

The next two sections offer additional details on (1) alternative and supplementary ways to configure Dakota’s operation (Configuring Dakota operation) and (2) how to specialize Dakota’s DirectApplicInterface to provide a function evaluator plugin to Dakota (Creating a simulator plugin interface).
5.3 Configuring Dakota operation

This section describes several alternate ways to initially set and later manipulate Dakota’s configuration, including alternatives to using a text-based input file. The algorithm configuration for a particular Dakota analysis run is managed in its ProblemDescDB, which can be populated via an input file, string literal, or C++ API, and later modified through Dakota’s C++ API. All Dakota objects then draw information from this database upon instantiation.

5.3.1 Input data parsing

The simplest way for an application to configure a Dakota analysis problem is to use Dakota’s normal input parsing system to populate its problem database (ProblemDescDB). This is done by providing standard Dakota input file syntax through the library interface, via either a file name or string literal. An advantage is that native Dakota syntax can be used, but disadvantages include the requirement for an additional input file beyond those already required by the parent application and that application users also need to know Dakota syntax.

The two ways to configure Dakota via input parsing are shown near the beginning of run_dakota_mixed() in library_mode.cpp. Here the ProgramOptions are set to either parse from a named file:

```cpp
dakota::ProgramOptions opts;
opts.input_file(dakota_input_file);
```

or from a string literal provided by the wrapping application:

```cpp
string serial_input = "& Dakota input file ...";
opts.input_string(serial_input);
```

This library approach is coarse-grained in that input is parsed, objects constructed, and the environment is immediately ready to run. The next approaches are more modular.

5.3.2 Problem database insertion

A second approach to configuring Dakota’s operation is to bypass parsing phases and directly populate the ProblemDescDB with information on the methods, variables, interface, responses, etc., that define the Dakota analysis problem. This approach requires more interaction with Dakota classes and data structures. However, it can offer usability benefit when the integrating application does not want their users to interact with the full Dakota syntax, or efficiency benefit when for example there are a large number of variables to configure.

In the direct database population approach, Dakota DataMethod, DataModel, DataVariables, DataInterface, and DataResponses objects are instantiated and populated with the desired problem data. These objects are then
published to the problem database using insert_nodes(). An example of this approach is available in run_dakota_data() in library_mode.cpp, where the OPT++ Quasi-Newton method is configured to work on a plugin version of text_book or rosenbrock. The data objects are populated with their default values upon instantiation and are often sufficient for basic Dakota studies. Only the non-default values need to be specified. Moreover the default Dakota Model is a SingleModel, so this object need not be configured unless tailoring its configuration or using a more advanced model type. Refer to the DataMethod, DataModel, DataVariables, DataInterface, and DataResponses class documentation and source code for lists of attributes and their defaults. Here is an excerpt of run_dakota_data() that specifies the OPT++ solver after default construction of DataMethod:

Dakota::DataMethod dme;
Dakota::DataMethodRep* dmr = dme.data_rep();
dmr->methodName = Dakota::OPTPP_QNEWTON;

When using direct database population, it is critical to leave the database in an open, accessible state after initial construction. In this run_dakota_data() example, a flag check_bcast_bcast_construct is passed into the LibraryEnvironment constructor, indicating that it should not finalize the database and construct Dakota objects. Moreover, it is only necessary to populate the database on rank 0 of the MPI Comm on which Dakota is running. After database objects are inserted or adjusted, the LibraryEnvironment::done_modifying_db() function must be called before proceeding to execute. This synchronizes problem data across all ranks and constructs Dakota objects needed to run the specified analysis.

```cpp
bool check_bcast_construct = false;
Dakota::LibraryEnvironment env(MPI_COMM_WORLD, opts, check_bcast_construct);
if (rank == 0)
  // insert/modify DB, then lock and proceed:
  env.done_modifying_db();
env.execute();
```

### 5.3.3 Mixed mode, callbacks, and late updates

The LibraryEnvironment API also supports mixed approaches that combine the parsing of a Dakota input file (or input string literal) with direct database updates. This approach is motivated by large-scale applications where large vectors are cumbersome to specify in a Dakota input file or where later updates to an input template are needed. The example run_dakota_mixed() in library_mode.cpp demonstrates the combination of these more advanced approaches: (1) input text parsing, (2) database updates via a callback, (3) database updates via direct manipulation, and (4) further runtime updates to the Model before running.

First, a ProgramOptions class is instantiated and configured to parse either an input file or input string literal (as in earlier examples). The passed input data must contain all required inputs so the parser can validate them. Since vector data like variable values/bounds/tags, linear/nonlinear constraint coefficients/bounds, etc., are optional, these potentially large vector specifications can be omitted from the input file and updated later through the database API. Only the variable/response counts necessary for sizing, e.g.:

```cpp
method
  linear_inequality_constraints = 500
variables
  continuous_design = 1000
responses
  objective_functions = 1
  nonlinear_inequality_constraints = 100000

and not the lists of values are required in this case. To update or add data after this initial parse, we use the ProblemDescDB::set() family of overloaded functions, e.g.

```cpp
Dakota::RealVector drv(1000, 1.); // vector of length 1000, values initialized to 1.
problem_db.set("variables.continuous_design.initial_point", drv);
```
where the string identifiers are the same identifiers used when pulling information from the database using one of the get_<datatype>() functions (refer to ProblemDescDB for a full list). However, the supported ProblemDescDB::set() options are a restricted subset of the database attributes, focused on vector inputs that can be large scale.

Second, the example demonstrates a user-provided callback function which Dakota will invoke after input parsing to update ProblemDescDB. In library_mode.cpp, callback_function() is a user-provided post-parse callback that implements the type Dakota::DbCallbackFunction.

```cpp
static void callback_function(Dakota::ProblemDescDB* db, void *ptr);```

When Dakota calls this function it will pass back pointers to the ProblemDescDB instance and to user-provided data, so the application may convey its settings by calling methods on the ProblemDescDB, optionally using the provided data. An example of a user data structure is demonstrated in callback.data. In this case, when the LibraryEnvironment is constructed, it is constructed with the input data to initially parse, the callback function, and to leave it unlocked for further updates:

```cpp
bool done_with_db = false;
Dakota::LibraryEnvironment env(opts, done_with_db, callback_function, &data);
```

Third, the example demonstrates changes to the database after parsing and callback-based updates. Again, these only need happen on Dakota’s rank 0 before finalizing the DB with LibraryEnvironment::done_modifying_db(). The example demonstrates:

1. Getting access to the database through env.problem_description_db()
2. Setting the database nodes to the appropriate method through problem_db.resolve_top_method()
3. Getting data from the DB with a get string array function: problem_db.get_sa("interface.application.-analysis_drivers")
4. Setting update data with problem_db.set("variables.continuous_design.initial_point", ip);

After any of these three types updates, calling LibraryEnvironment::done_modifying_db() will broadcast any updates (including potentially large vector data and post-process specification data to fill in any vector defaults that have not yet been provided through either file parsing or direct updates. (Note: scalar defaults are handled in the Data class constructors.)

Fourth and finally, run_dakotaMixed() demonstrates modifying a Model’s data after database operations and interface plugin are complete. This involves finding the right Model (or other class) instance to modify, and directly adjusting its data through the public API. Since the database is finalized, any updates must be performed through direct set operations on the constructed objects. For example, to update other data such as variable values/bounds/tags or response bounds/targets/tags, refer to the set functions documented in Iterator and Model. As an example, the following code updates the active continuous variable values, which will be employed as the initial guess for certain classes of Iterators:

```cpp
ModelList& all_models = problem_db.model_list();
Model& first_model = *all_models.begin();
Dakota::RealVector drv(1000, 1.); // vector of length 1000, values initialized to 1.
first_model.continuous_variables(drv);
```
Remarks

If performing such data updates within the constructor of a DirectApplicInterface extension/derivation (see Creating a simulator plugin interface), then this code is sufficient since the database is unlocked, the active list nodes of the ProblemDescDB have been set for you, and the correct method/model/variables/interface/responses specification instance will get updated. The difficulty in this case stems from the order of instantiation. Since the Variables and Response instances are constructed in the base Model class, prior to construction of Interface instances in derived Model classes, database information related to Variables and Response objects will have already been extracted by the time the Interface constructor is invoked and the database update will not propagate.

Therefore, it is preferred to perform these database set operations at a higher level (e.g., within your main program), prior to allowing Environment to broadcast, construct, and execute, such that instantiation order is not an issue. However, in this case, it is necessary to explicitly manage the list nodes of the ProblemDescDB using a specification instance identifier that corresponds to an identifier from the input file, e.g.:

```cpp
problem_db.set_db_variables_node("MY_VARIABLES_ID");
Dakota::RealVector drv(1000, 1.); // vector of length 1000, values initialized to 1.
problem_db.set("variables.continuous_design.initial_point", drv);
```

Alternatively, rather than setting just a single data node, all data nodes may be set using a method specification identifier:

```cpp
problem_db.set_db_list_nodes("MY_METHOD_ID");
```

since the method specification is responsible for identifying a model specification, which in turn identifies variables, interface, and responses specifications. If hard-wiring specification identifiers is undesirable, then

```cpp
problem_db.resolve_top_method();
```

can also be used to deduce the active method specification and set all list nodes based on it. This is most appropriate in the case where only single specifications exist for method/model/variables/interface/responses. This is the approach demonstrated in run_dakota_mixed(). In each of these cases, setting list nodes unlocks the corresponding portions of the database, allowing set/get operations.

5.4 Creating a simulator plugin interface

The DirectApplicInterface class provides an interface for Dakota to call the simulation code directly to perform function evaluations mapping variables to responses. This provides the "back end" for Dakota to call back to the simulation framework. Two approaches to defining this direct interface are described here. The first is less common, while the second is recommended when possible.

5.4.1 Extension

The first approach involves extending one of the existing DirectApplicInterface subclasses (TestDriverInterface, MatlabInterface, etc.) to support additional direct simulation interfaces. For example, Dakota algebraic test problems are implemented in TestDriverInterface. One could add additional direct functions to Dakota in TestDriverInterface::derived_map_ac(). In addition, TestDriverInterface::derived_map_if() and TestDriverInterface::derived_map_of() can be extended to perform pre- and post-processing tasks if desired, but this is not required.

While this approach is the simplest, it has the disadvantage that the Dakota library will need to be recompiled when the simulation or its direct interface is modified. If it is desirable to maintain the independence of the Dakota library from the host application, then the derivation approach described in the next section should be employed.
Remarks

If the new direct evaluation function implementation will not be a member function of one of the Dakota classes, then the following prototype should be used in order to pass the required data:

```cpp
int sim(const Dakota::Variables& vars, const Dakota::ActiveSet& set, Dakota::Response& response);
```

If the new function will be a member function, e.g., in `TestDriverInterface`, then this can be simplified to

```cpp
int sim();
```

since the data access can be performed through the `DirectApplicInterface` class attributes.

5.4.2 Derivation

The second approach is to derive a new interface from `DirectApplicInterface` and redefine several virtual functions. As demonstrated in `SIM::SerialDirectApplicInterface` and `SIM::ParallelDirectApplicInterface`, a typical derived class declaration might be

```cpp
namespace SIM {

class SerialDirectApplicInterface: public Dakota::DirectApplicInterface {

public:

    // Constructor and destructor

    SerialDirectApplicInterface(const Dakota::ProblemDescDB& problem_db);
    ~SerialDirectApplicInterface();

protected:

    // Virtual function redefinitions

    int derived_map_if(const Dakota::String& if_name);
    int derived_map_ac(const Dakota::String& ac_name);
    int derived_map_of(const Dakota::String& of_name);

private:

    // Data

} // namespace SIM
```

where the new derived class resides in the simulation’s namespace. Similar to the case of `Extension`, the `DirectApplicInterface::derived_map_ac()` function is the required redefinition, and `DirectApplicInterface::derived_map_if()` and `DirectApplicInterface::derived_map_of()` are optional.

Typically the new derived_map_ac() implementation delegates to the main simulation application for a function evaluation. Here Dakota variables would get mapped into the simulation’s data structures, the simulation executed, and derived response data computed for return to Dakota.

Once a derived application class is created, it must be plugged in, or registered, with the appropriate `Interface` in the `LibraryEnvironment`. In MPI cases where Dakota is potentially managing concurrent evaluations of the simulation, the plugin must be configured to run on the right MPI sub-communicator, or Dakota analysis_comm. The simpler case is demonstrated in `serial_interface_plugin()` in `library_mode.cpp`, while a more advanced case using the analysis communicator is shown in `parallel_interface_plugin()`.

The Dakota `LibraryEnvironment` provides a convenience function to plugin an `Interface`. This example will replace any interface found matching the given model, interface, and analysis driver with the passed plugin interface:
std::string model_type(""); // demo: empty string will match any model type
std::string interface_type("direct");
std::string an_driver("plugin_rosenbrook");
Dakota::ProblemDescDB problem_db = env.problem_description_db();
Dakota::Interface* serial_iface =
    new SIM::SerialDirectApplicInterface(problem_db);
bool plugged_in =
    env.plugin_interface(model_type, interface_type, an_driver, serial_iface);

The **LibraryEnvironment** also provides convenience functions that allow the client to iterate the lists of available interfaces or models for more advanced cases. For instance if the client knows there is only a single interface active, it could get the list of available interfaces of length 1 and plugin to the first one. In the more advanced case where the simulation interface instance should manage parallel simulations within the context of an MPI communicator, one should pass in the relevant analysis communicator(s) to the derived constructor. For the latter case of looping over a set of models, the simplest approach of passing a single analysis communicator would use code similar to

```
Dakota::ModelList filt_models =
    env.filtered_model_list("single", "direct", "plugin_text_book");
Dakota::ProblemDescDB problem_db = env.problem_description_db();
Dakota::ModelIter ml_iter;
for (ml_iter = filt_models.begin(); ml_iter != filt_models.end(); ++ml_iter) {
  // set DB nodes to input specification for this Model
  problem_db.set_db_model_nodes(ml_iter->model.id());
  Dakota::Interface& model_interface = ml_iter->derived_interface();
  // Parallel case: plug in derived Interface object with an analysisComm.
  // Note: retrieval and passing of analysisComm is necessary only if
  // parallel operations will be performed in the derived constructor.

  // retrieve the currently active analysisComm from the Model. In the most
  // general case, need an array of Comms to cover all Model configurations.
  const MPI_Comm* analysis_comm = ml_iter->analysis_comm();

  // don’t increment ref count since no other envelope shares this letter
  model_interface.assign_rep(new
    SIM::ParallelDirectApplicInterface(problem_db, analysis_comm), false)
  ;
}
```

The file **library_mode.cpp** demonstrates each of these approaches. Since a **Model** may be used in multiple parallel contexts and may therefore have a set of parallel configurations, a more general approach would extract and pass an array of analysis communicators to allow initialization for each of the parallel configurations.

New derived direct interface instances inherit various attributes of use in configuring the simulation. In particular, the **ApplicationInterface::parallelLib** reference provides access to MPI communicator data (e.g., the analysis communicators discussed above), **DirectApplicInterface::analysisDrivers** provides the analysis driver names specified by the user in the input file, and **DirectApplicInterface::analysisComponents** provides additional analysis component identifiers (such as mesh file names) provided by the user which can be used to distinguish different instances of the same simulation interface. It is worth noting that inherited attributes that are set as part of the parallel configuration (instead of being extracted from the **ProblemDescDB**) will be set to their defaults following construction of the base class instance for the derived class plug-in. It is not until run-time (i.e., within derived_map/derived_map_ac/derived_map_of) that the parallel configuration settings are re-propagated to the plug-in instance. This is the reason that the analysis communicator should be passed in to the constructor of a parallel plug-in, if the constructor will be responsible for parallel application initialization.
5.5 Retrieving data after a run

After executing the Dakota Environment, final results can be obtained through the use of Environment::variables_results() and Environment::response_results(), e.g.:

```cpp
// retrieve the final parameter values
const Variables& vars = env.variables_results();
// retrieve the final response values
const Response& resp = env.response_results();
```

In the case of optimization, the final design is returned, and in the case of uncertainty quantification, the final statistics are returned. Dakota has a prototype results database, which will eventually provide better access to the results from a study.

5.6 Linking against the Dakota library

This section presumes Dakota has been configured with CMake, compiled, and installed to a CMAKE_INSTALL_ALLPREFIX using make install or equivalent. The Dakota libraries against which you must link will typically install to CMAKE_INSTALL_PREFIX/bin/ and CMAKE_INSTALL_PREFIX/lib/, while headers are provided in CMAKE_INSTALL_PREFIX/lib/.

The core Dakota C and C++ code is in the library dakota-src, while Fortran code lives in the dakota_src.fortran library. Runtime libraries for any configure-enabled Dakota third-party software components (such as DOT, NPSOL, OPT++, LHS, etc.) are also installed to the lib/ directory. Applications link against these Dakota libraries by specifying appropriate include and link directives.

There are two primary ways to determine the necessary Dakota-related libraries and link order for linking your application. First, when running CMake, a list of required Dakota and Dakota-included third-party libraries will be output to the console, e.g.,

```plaintext
-- Dakota_LIBRARIES: dakota_src;dakota_src_fortran;nidr;teuchos;pecos;pecos_src;lhs;mods;mod;fftpack;sparselip
```

While external dependencies will be output as:

```plaintext
-- Dakota_TPL_LIBRARIES: /usr/lib64/libcurl.so;/usr/lib64/openmpi/lib/libmpi_cxx.so;debug;/usr/lib64/libz.so;
```

Note that depending on how you configured Dakota, some libraries may be omitted from these lists (for example commercial add-ons NPSOL, DOT, and NLPLQ), or additional libraries may appear.

A second option is to check which libraries appear in CMAKE_INSTALL_PREFIX/bin/ CMAKE_INSTALL_LPREFIX/lib/, or more accurately, see the file Makefile.export.Dakota in the Dakota build/src/ or installation include/ directory. Here are some additional notes on specific libraries:

- Some Boost libraries (boost_regex, boost_filesystem, boost_system, boost_serialization) are required, and other Boost library components may be required depending on configuration, e.g., boost_signals when configuring with HAVE_ACRO:BOOL=TRUE

- System compiler and math libraries may need to be included, as may additional system libraries such as Expat and Curl, depending on how Dakota is configured.

- If configuring with graphics, you will need to add the dakota_sciplot library and system X libraries (partial list here):

  ```bash
  -lXpm -lXm -lXt -lXmu -lXp -lXext -lX11 -lSM -lICE
  ```
• When configuring with AMPL (HAVE_AMPL.BOOL=ON), the AMPL solver library may require d1, funcadd0.o and fl libraries. We have experienced problems with the creation of libamplsolver.a on some platforms; use the dakota-users mailing list to get help with any problems related to this.

• Optional library GSL (discouraged due to GPL license) and if linking with system-provided GSL, gslcblas may be needed if Dakota was configured with them.

• Newmat: as of Dakota 5.2, -lnewmat is no longer required

Finally, it is important to use the same C++ compiler (possibly an MPI wrapper) for compiling Dakota and your application and potentially include Dakota-related preprocessor defines as emitted by CMake during compilation of Dakota and included in Makefile.export.Dakota. This ensures that the platform configuration settings are properly synchronized across Dakota and your application.
Chapter 6

Performing Function Evaluations

Performing function evaluations is one of the most critical functions of the Dakota software. It can also be one of the most complicated, as a variety of scheduling approaches and parallelism levels are supported. This complexity manifests itself in the code through a series of cascaded member functions, from the top level model evaluation functions, through various scheduling routines, to the low level details of performing a system call, fork, or direct function invocation. This section provides an overview of the primary classes and member functions involved.

6.1 Synchronous function evaluations

For a synchronous (i.e., blocking) mapping of parameters to responses, an iterator invokes Model::evaluate() to perform a function evaluation. This function is all that is seen from the iterator level, as underlying complexities are isolated. The binding of this top level function with lower level functions is as follows:

- Model::evaluate() utilizes Model::derived_evaluate() for portions of the response computation specific to derived model classes.
- Model::derived_evaluate() directly or indirectly invokes Interface::map().
- Interface::map() utilizes ApplicationInterface::derived_map() for portions of the mapping specific to derived application interface classes.

6.2 Asynchronous function evaluations

For an asynchronous (i.e., nonblocking) mapping of parameters to responses, an iterator invokes Model::evaluate_nowait() multiple times to queue asynchronous jobs and then invokes either Model::synchronize() or Model::synchronize_nowait() to schedule the queued jobs in blocking or nonblocking fashion. Again, these functions are all that is seen from the iterator level, as underlying complexities are isolated. The binding of these top level functions with lower level functions is as follows:

- Model::evaluate_nowait() utilizes Model::derived_evaluate_nowait() for portions of the response computation specific to derived model classes.
- This derived model class function directly or indirectly invokes Interface::map() in asynchronous mode, which adds the job to a scheduling queue.
- Model::synchronize() or Model::synchronize_nowait() utilize Model::derived_synchronize() or Model::derived_synchronize_nowait() for portions of the scheduling process specific to derived model classes.
• These derived model class functions directly or indirectly invoke `Interface::synchronize()` or `Interface::synchronize_nowait()`.

• For application interfaces, these interface synchronization functions are responsible for performing evaluation scheduling in one of the following modes: master dynamic, peer dynamic or peer static.

• **NOTE:** The `Interface` evaluation scheduling in Dakota was refactored for releases 5.4 and 6.0. Discussion of the new Interface-related functions is currently missing here.

### 6.3 Analyses within each function evaluation

**NOTE:** The `Interface` evaluation scheduling in Dakota was refactored for releases 5.4 and 6.0. Discussion of the new Interface-related functions for analyses is currently missing here.

The discussion above covers the parallelism level of concurrent function evaluations serving an iterator. For the parallelism level of concurrent analyses serving a function evaluation, similar schedulers are involved to support synchronous local, asynchronous local, message passing, and hybrid modes. Not all of the schedulers are elevated to the `ApplicationInterface` level since the system call and direct function interfaces do not yet support nonblocking local analyses (and therefore support synchronous local and message passing modes, but not asynchronous local or hybrid modes). Fork interfaces, however, support all modes of analysis parallelism.
Chapter 7

Working with Variable Containers and Views

Variable views control the subset of variable types that are active and inactive within a particular iterative study. For design optimization and uncertainty quantification (UQ), for example, the active variables view consists of design or uncertain types, respectively, and any other variable types are carried along invisible to the iterative algorithm being employed. For parameter studies and design of experiments, however, a variable subset view is not imposed and all variables are active. Selected UQ methods can also be toggled into an "All" view using the `active all` variables input specification. When not in an All view, finer gradations within the uncertain variable sets are also relevant: probabilistic methods (reliability, stochastic expansion) view aleatory uncertain variables as active, nonprobabilistic methods (interval, evidence) view epistemic uncertain variables as active, and a few UQ methods (sampling) view both as active. In a more advanced NestedModel use case such as optimization under uncertainty, design variables are active in the outer optimization context and the uncertain variables are active in the inner UQ context, with an additional requirement on the inner UQ level to return derivatives with respect to its "inactive" variables (i.e., the design variables) for use in the outer optimization loop.

For efficiency, contiguous arrays of data store variable information for each of the domain types (continuous, discrete integer, and discrete real), but active and inactive views into them permit selecting subsets in a given context. This management is encapsulated into the Variables and SharedVariablesData classes. This page clarifies concepts of relaxed (formerly merged) vs. mixed, fine-grained vs. aggregated types, domain types, and views into contiguous arrays.

We begin with an overview of the storage and management concept, for which the following two sections describe the storage of variable values and meta-data about their organization, used in part to manage views. They are intended to communicate rationale to maintainers of Variables and SharedVariablesData classes. The final section provides a discussion of active and inactive views.

7.1 Storage in Variables

As described in the Main Page Variables, a Variables object manages variable types (design, aleatory uncertain, epistemic uncertain, and state) and domain types (continuous, discrete integer, and discrete real) and supports different approaches to either distinguishing among these types or aggregating them. Two techniques are used in cooperation to accomplish this management: (1) class specialization (RelaxedVariables or MixedVariables) and (2) views into contiguous variable arrays. The latter technique is used whenever it can satisfy the requirement, with fallback to class specialization when it cannot. In particular, aggregation or separation of variable types can be accomplished with views, but for aggregation or separation of variable domains, we must resort to class
specialization in order to relax discrete domain types. In this class specialization, a `RelaxedVariables` object combines continuous and discrete types (relaxing integers to reals) whereas a `MixedVariables` object maintains the integer/real distinction throughout.

The core data for a `Variables` instance is stored in a set of three contiguous arrays, corresponding to the domain types: `allContinuousVars`, `allDiscreteIntVars`, and `allDiscreteRealVars`, unique to each `Variables` instance.

Within the core variable data arrays, data corresponding to different aggregated variable types are stored in sequence for each domain type:

- continuous: [design, aleatory uncertain, epistemic uncertain, state]
- discrete integer: [design, aleatory uncertain, (epistemic uncertain), state]
- discrete real: [design, aleatory uncertain, (epistemic uncertain), state]

Note there are currently no epistemic discrete variables. This domain type ordering (continuous, discrete integer, discrete real) and aggregated variable type ordering (design, aleatory uncertain, epistemic uncertain, state) is preserved whenever distinct types are flattened into single contiguous arrays. Note that the aleatory and epistemic uncertain variables contain sub-types for different distributions (e.g., normal, uniform, histogram, poisson), and discrete integer types include both integer ranges and integer set sub-types. All sub-types are ordered according to their order of appearance in `dakota.input.nspec`.

When relaxing in `MixedVariables`, the `allContinuousVars` will also aggregate the discrete types, such that they contain ALL design, then ALL uncertain, then ALL state variables, each in aggregated type order; the `allDiscreteIntVars` and `allDiscreteRealVars` arrays are empty.

### 7.2 Storage in SharedVariablesData

Each `Variables` instance contains a reference-counted `SharedVariablesData` object that stores information on the variables configuration. This configuration data includes counts, types, IDs, and labels, which are often the same across many `Variables` instances. Thus, `SharedVariablesData` is intended to reduce the memory footprint by allowing the sharing of a single copy of redundant information among different `Variables` instances.

One of the purposes of this shared information is to support mappings between variable types, IDs, and indices into the storage arrays. Variable "types" refer to the fine-grained variable types a user would specify in an input file, as enumerated in `DataVariables.hpp`, e.g., CONTINUOUS, DESIGN, WEIBULL, UNCERTAIN, DISCRETE, STATE, RANGE, etc. `variablesComponents` is a map from these variable types to counts of how many are present.

In contrast, the `variablesCompsTotals` array stores total counts of each "aggregated type" (design, aleatory uncertain, epistemic uncertain, state) which might be selected to be active in a given view. Thus this array has length 12 to track the combinations of three domain type storage arrays with four possible aggregated variable types: \{continuous, discrete integer, discrete real\} x \{design, aleatory uncertain, epistemic uncertain, state\}. For example, the first entry of this array stores the number of continuous design variables, the second the number of discrete integer design (including both discrete design range and discrete design set integer types), and the last the number of discrete real state variables.

The arrays `allContinuousTypes`, `allDiscreteIntTypes`, and `allDiscreteRealTypes` are sized to match the corresponding core domain type storage arrays. They track the fine-grained variable type stored in that entry of the data array (since when relaxed, the continuous array may be storing data corresponding to discrete data).

Finally `allContinuousIds` stores the 1-based IDs of the variables stored in the `allContinuousVars` array, i.e., the variable number of all the problem variables considered as a single contiguous set, in aggregate type order. For relaxed (formerly merged) views, `relaxedDiscreteIds` stores the 1-based IDs of the variables which have been relaxed into the continuous array.
These counts, types, and IDs are most commonly used within the Model classes for mappings between variables objects at different levels of a model recursion. See, for example, the variable mappings in the NestedModel constructor.

### 7.3 Active and inactive views

The pair SharedVariablesDataRep::variablesView tracks the active and inactive views of the data, with values taken from the enum in DataVariables.hpp. The valid values include EMPTY and the combinations \{relaxed, mixed\} x \{all, design, aleatory uncertain, epistemic uncertain, uncertain, state\}. The ALL cases indicate aggregation of the design, aleatory uncertain, epistemic uncertain, and state types, whereas the DISTINCT cases indicate either no aggregation (design, aleatory uncertain, epistemic uncertain, state) or reduced aggregation (aleatory+epistemic uncertain). The active view is determined by the algorithm in use, managed in Variables::getView(). Any inactive view is set based on higher level iteration within a model recursion (e.g., a NestedModel), which enables lower level iteration to return derivatives with respect to variables that are active at the higher level. In the case where there is no higher level iteration, then the inactive view will remain EMPTY. It is important to stress that "inactive" at one level corresponds to active at another, and therefore the inactive set of variables should not be interpreted as the strict complement of the active set of variables; rather, active and inactive are both subsets whose union may still be a subset of the total container (more precise terminology might involve "primary" active and "secondary" active or similar). An active complement view could potentially be supported in the future, should the need arise, although this view would require management of non-contiguous portions of the aggregated arrays.

Given these groupings (views), the active and inactive subsets of the allContinuousVars, allDiscreteIntVars, and allDiscreteRealVars arrays are always contiguous, permitting vector views of the underlying data using either Teuchos::View (for numerical vectors) or Boost.MultiArray (for book-keeping arrays) views.

When a Variables envelope is constructed, its letter is initialized to either a RelaxedVariables or MixedVariables object depending on the active view. The derived classes size the contiguous storage arrays to accommodate all the problem variables, and then initialize active views into them, which could involve either subsets (DISTINCT active views) or views of the full arrays (ALL active views). Inactive views, on the other hand, are initialized during construction of a model recursion (e.g., a call to Model::inactiveView() in the NestedModel constructor). Thus, active variable subsets are always available but inactive variable subsets will be EMPTY prior to them being initialized within a Model recursion.

Accessors for continuous variables include:

- `continuous_variables()`: returns the active view which might return all (ALL views) or a subset (DISTINCT views) such as design, uncertain, only aleatory uncertain, etc.
- `inactive_continuous_variables()`: returns the inactive view which which is either a subset or empty
- `all_continuous_variables()`: returns the full vector allContinuousVars

and this pattern is followed for active/inactive/all access to discrete_int_variables() and discrete_real_variables() as well as for labels, IDs, and types in SharedVariablesData and variable bounds in Constraints.
Chapter 8

Demo TPL

This is a simple Demo which serves as a working example for bringing a new Third-Party Library (TPL) into Dakota. The Demo will serve to show minimal requirements for:

- building and running the Demo
- building a TPL under Dakota using CMake
- exposing TPL functionality to Dakota
- exposing TPL options through Dakota
- transferring data between a TPL and Dakota

Following this Demo, a developer should be able to integrate an optimization TPL/method that:

- is derivative-free
- operates over continuous variables
- supports any of the following types of constraints
  - bound constraints
  - nonlinear inequality constraints
  - nonlinear equality constraints

Quickstart: Building and Running the Demo

In order to build and run this Demo, it is necessary to build Dakota from source. Complete instructions for doing so can be found at https://dakota.sandia.gov/content/build-compile-source-code. At the point in the instructions where cmake is invoked, append -DHAVE_DEMO_TPL:BOOL=ON to the cmake invocation.

Building Dakota with the Demo TPL enabled will also activate a working example found in $DAKOTA_BUILD/test/dakota_demo_app, where $DAKOTA_BUILD is the root of the Dakota build tree. The test can be run from $DAKOTA_BUILD/test using

'ctest -R demo_app'

Summary info will be output to the screen, and test artifacts can be found in $DAKOTA_BUILD/test/dakota_demo_app.

Alternatively the example can be run in the same way a user runs Dakota. In particular, from the $DAKOTA_BUILD/test/dakota_demo_app directory, issue the following command:
The remainder of this file describes how to integrate a TPL into Dakota using the Demo (found in $DAKOTA_SRC/packages/external/demo_tpl) as an example.

Building a TPL under Dakota using Cmake

This section shows how to include the relevant parts of the Demo TPL as a library that Dakota builds and includes as part of its own native Cmake build.

Assuming the Demo tpl source code has been placed alongside other Dakota TPLs in $DAKOTA_SRC/packages/external/demo_tpl, a simple CMakeLists.txt file can be created at this location to allow Dakota to include it within its own Cmake setup. An minimal example might include:

```cmake
# File $DAKOTA_SRC/packages/external/demoTpl/CMakeLists.txt
cmake_minimum_required(VERSION 2.8)
project("DEMOTpl" CXX)
SUBDIRS(src)

In the src subdirectory of demo_tpl would be another CMakeLists.txt file which essentially identifies the relevant source code to be compiled into a library along with defining the library which Dakota will later include, e.g.

```cmake
# File $DAKOTA_SRC/packages/external/demoTpl/src/CMakeLists.txt
set(demoTpl HEADERS demoopt.hpp )
set(demoTpl SOURCES demoopt.cpp )

# Set the DEMOTpl library name.
add_library(demoTpl ${demoTpl SOURCES})

# Define install targets for "make install"
install(TARGETS demoTpl EXPORT ${ExportTarget} DESTINATION lib)
```

Note that it is possible to use Cmake’s glob feature to bring in all source and header files, but care must be taken to avoid introducing main(...) symbols which will collide with Dakota’s main at link time.

At this point, Dakota’s CMakeLists.txt files will need to be modified to include the Demo TPL. The following modifications can be used to bring in the Demo TPL, conditioned on having -DHAVE_DEMO_tpl:BOOL=ON defined when invoking cmake to configure Dakota:

```cmake
# File $DAKOTA_SRC/packages/CMakeLists.txt
<... snip ...>
option(HAVE_DEMOTpl "Build the DemoTpl package." OFF)
<... end snip ...>
<... snip ...>
if(HAVE_DEMOTpl)
  add_subdirectory(external/demo_tpl)
endif(HAVE_DEMOTpl)
<... end snip ...>
```

This next modification to Dakota will allow the Demo TPL to be used by other Dakota source code by including the necessary include paths, link-time libraries and needed #defines:
Test-Driven Code Development

Before making concrete changes, it is often helpful to create a simple Dakota test which will serve to guide the process. This is akin to test-driven development which essentially creates a test which fails until everything has been implemented to allow it to run and pass. A candidate test for the current activity could be the following:

For this test to run, we will need to be able to pass parsed options to the Demo TPL and exchange parameters and response values between Dakota and Demo TPL. These details are presented in the following sections.

Exposing TPL Functionality to Dakota

Dakota performs some internal checks in order to confirm applicability of a specified method to the problem defined. In order for Dakota to perform those checks for the Demo TPL, the functionality of the method must be communicated to Dakota. That is done via implementation of a traits class. Traits define the types of problems and data formats the Demo TPL supports by overriding the default traits accessors in TraitsBase. By default, nothing is supported, and the TPL integrator must explicitly turn on the traits for any supported features.
DemoOptTraits(); { }
virtual ~DemoOptTraits() { }
<... end snip ...>
<... snip ...>
// Heading: Virtual member function redefinitions
//
bool supportsContinuousVariables() override
{ return true; }
<... end snip ...>
}; // class DemoOptTraits

A complete list of traits can be found in $DAKOTA_SRC/src/DakotaTraitsBase.hpp. The subset applicable to the Demo TPL can be found in $DAKOTA_SRC/packages/external/demo_tpl/dakota_src/DemoOptimizer.hpp.

Exposing TPL Options Through Dakota

The simplest way to pass options to a TPL is via a file. The Demo TPL has the ability to read in a file of method options when given a file name. This file name can be specified in the Dakota input file and retrieved as illustrated below.

// File $DAKOTA_SRC/packages/external/demo_tpl/dakota_src/DemoOptimizer.cpp
<... snip ...>
// Check for native Demo Opt input file. The file name needs to be
// included in the Dakota input file.
String advOptsFile = probDescDB.getString("method.advanced_options_file");
if (!advOptsFile.empty())
{
    if (!boost::filesystem::exists(advOptsFile))
    {
        Cerr << 
            " Error: Demo Opt options file \" << advOptsFile 
            " specified, but file not found.\n"
        abort_handler(METHOD_ERROR);
    }
}
<... end snip ...>

If desired, common stopping criteria can be retrieved from the Dakota input file, rather than passed through a TPL-specific input file, as follows.

// File $DAKOTA_SRC/packages/external/demo_tpl/dakota_src/DemoOptimizer.cpp
<... snip ...>
getCommonStoppingCriteria(maxFnEvals, maxIters, convTol,
minVarChg, objTarget);
<... end snip ...>

Exchanging Parameters and Responses

Like any TPL, the Demo TPL will need to exchange parameter and objective function values with Dakota. For purposes of demonstration, an example interface between Dakota and the Demo TPL can be seen in $DAKOTA_SRC/packages/external/demo_tpl/dakota_src/DemoOptimizer.hpp (with corresponding .cpp in the same directory). Within these files is a key callback interface used by the Demo TPL to obtain objective function values for given parameter values (3 in the test above), eg:

// File $DAKOTA_SRC/packages/external/demo_tpl/dakota_src/DemoOptimizer.cpp
Real
DemoTPLOptimizer::compute_obj(const std::vector<double> & x, bool verbose)
{
    // Tell Dakota what variable values to use for the function
    // valuation. x must be (converted to) a std::vector<double> to use
    // this demo with minimal changes.
    set_variables<>(x, iteratedModel, iteratedModel.current_variables());

    // Evaluate the function at the specified x.
    iteratedModel.evaluate();

    // Retrieve the the function value and sign it appropriately based
    // on whether minimize or maximize has been specified in the Dakota
    // input file.
    double f = dataTransferHandler->get_response_value_from_dakota(iteratedModel.current_response());
    return f;
}

In this instance, the Demo TPL uses std::vector<double> as its native parameter vector data type and
is calling back to the example problem (Dakota model) via an interface to Dakota to obtain a single double
(alias to Real in Dakota) objective function value for a given set of parameter values. These data exchanges
are facilitated by used of "data adapters" supplied by Dakota with the set_variables<>(...) utility and
dataTransferHandler helper class utilized in this case.

For problems involving nonlinear equality and inequality constraints Dakota treats these as additional re-
sponses to the objective function(s). The Demo TPL supports both types for purposes of showing how these
additional responses can be computed by Dakota (via interface to an underlying model) and transferred to the
TPL. Similar to the call (by Demo) to compute_obj(...) are two additional methods to compute and transfer
nonlinear constraint responses, eg:

void DemoTPLOptimizer::compute_nln_eq(std::vector<Real> &c, const std::vector<Real> &x, bool verbose)
{
    // Tell Dakota what variable values to use for the nonlinear constraint
    // evaluations. x must be (converted to) a std::vector<double> to use
    // this demo with minimal changes.
    set_variables<>(x, iteratedModel, iteratedModel.current_variables());

    // Evaluate the function at the specified x.
    iteratedModel.evaluate();

    // Use an adapter to copy data
    dataTransferHandler->get_nonlinear_eq_constraints_from_dakota(iteratedModel.current_response(), c);
}

void DemoTPLOptimizer::compute_nln_ineq(std::vector<Real> &c, const std::vector<Real> &x, bool verbose)
{
    set_variables<>(x, iteratedModel, iteratedModel.current_variables());
    iteratedModel.evaluate();
    dataTransferHandler->get_nonlinear_ineq_constraints_from_dakota(iteratedModel.current_response(), c);
}

Both of these callback methods (to Dakota), compute_nln_eq(...) and compute_nln_ineq(...) follow the same pattern as seen for the objective function callback: 1) set the Dakota model with the current
variables (parameters), 2) evaluate the model and 3) transfer the desired response (objective or constraint) back
to the TPL. The third step is facilitated by the appropriate call to the dataTransferHandler helper class.
It should be noted that even though as many as three separate calls to evaluate the model are made for the same
parameter values, Dakota maintains an internal cache of response values for each unique set. The model will be
evaluated the first time a new set of parameter values is provided, but the cached values will simply be returned
thereafter, thereby avoiding superfluous model evaluations.

Dakota must also provide initial parameter values to the Demo TPL and retrieve final objective function and
variable values from the Demo TPL. The initial values for parameters and bound constraints can be obtained from
Dakota with the `get_variables<>(...) helpers. This example returns the values to a standard vector of doubles (Reals). These values can then be passed to the Demo TPL using whatever API is provided. The API for this last step varies with the particular TPL, and Demo provides a function `set_problem_data` in this case.

```cpp
// File $DAKOTA_SRC/packages/external/demo_tpl/dakota_src/DemoOptimizer.cpp
void DemoTPLOptimizer::initialize_variables_and_constraints()
{
    // Get the number of variables, the initial values, and the values // of bound constraints. They are returned to standard C++ data // types. This example considers only continuous variables. Other // types of variables and constraints will be added at a later time. // Note that double is aliased to Real in Dakota.
    int num_total_vars = numContinuousVars;
    std::vector<Real> init_point(num_total_vars);
    std::vector<Real> lower(num_total_vars),
                      upper(num_total_vars);

    // More on DemoOptTraits can be found in DemoOptimizer.hpp.
    get_variables(iteratedModel, init_point);
    get_variable_bounds_from_dakota<DemoOptTraits>( lower, upper );

    // Replace this line by whatever the TPL being integrated uses to // ingest variable values and bounds, including any data type // conversion needed.
    // ------------------ TPL SPECIFIC ------------------
    demoOpt->set_problem_data(init_point, // "Initial Guess"
                                lower , // "Lower Bounds"
                                upper ); // "Upper Bounds"
}
```

The TPL should be able to return an optimal objective function value and the corresponding variable (parameter) values via its API. As has been the case throughout, the data should be doubles (aliased to Real in Dakota). The following code takes the values returned by Demo via a call to `get_best_f()` and sets the Dakota data structures that contain final objective and variable values. It adjusts the sign of the objective based on whether minimize or maximize has been specified in the Dakota input file (minimize is the default). If the problem being optimized involves nonlinear equality and/or inequality constraints, these will also need to be obtained from the TPL and passed to Dakota as part of the array of best function values (responses).

```cpp
// File $DAKOTA_SRC/packages/external/demo_tpl/dakota_src/DemoOptimizer.cpp
// in method void DemoTPLOptimizer::core_run()
    double best_f = demoOpt->get_best_f(); // TPL SPECIFIC

    const BoolDeque& max_sense = iteratedModel.primary.response_fn_sense();
    RealVector best_fns(iteratedModel.response_size()); // includes nonlinear contraints
    RealVector best_nln_eqs(iteratedModel.nonlinear_eq_size()); // TPL SPECIFIC

    // Get best (single) objective value respecting max/min expectations
    best_fns[0] = (!max_sense.empty() && max_sense[0]) ? -best_f : best_f;

    // Get best Nonlinear Equality Constraints from TPL
    if( numNonlinEqConstraints > 0 )
    {
        auto best_nln_eqs = demoOpt->get_best_nln_eqs(); // TPL SPECIFIC
        dataTransferHandler->get_best_nonlinear_eq_constraints_from_tpl( best_nln_eqs, best_fns );
    }
```
// Get best Nonlinear Inequality Constraints from TPL
if (numNonlinearIneqConstraints > 0) {
    auto best_nln_ineqs = demoOpt->get_best_nln_ineqs(); // TPL_SPECIFIC
    dataTransferHandler->get_best_nonlinear_ineq_constraints_from_tpl(
        best_nln_ineqs,
        best_fns);
}

bestResponseArray.front().function_values(best_fns);

std::vector<double> best_x = demoOpt->get_best_x(); // TPL_SPECIFIC
// Set Dakota optimal value data.
set_variables<>{best_x, iteratedModel, bestVariablesArray.front()};

Member SurfpackApproximation::build () override

    Right now, we’re completely deleting the old data and then recopying the current data into a SurfData object. This was just the easiest way to arrive at a solution that would build and run. This function is frequently called from addPoint rebuild, however, and it’s not good to go through this whole process every time one more data point is added.

Member SurfpackApproximation::hessian (const RealVector &c_vars) override

    Make this acceptably efficient

Member SurfpackApproximation::hessian (const Variables &vars) override

    Make this acceptably efficient
Chapter 9

Namespace Index

9.1 Namespace List

Here is a list of all documented namespaces with brief descriptions:

- dakota
  Dakota (lowercase) namespace for new Dakota modules

- Dakota
  The primary namespace for DAKOTA

- dakota::surrogates
  Namespace for new Dakota surrogates module

- dakota::util
  Namespace for new Dakota utilities module

- SIM
  A sample namespace for derived classes that use assign_rep() to plug facilities into DAKOTA

- StanfordPSAAP
  A sample namespace for derived classes that use assign_rep() to plug facilities into DAKOTA
Chapter 10

Hierarchical Index

10.1 Class Hierarchy

This inheritance list is sorted roughly, but not completely, alphabetically:

ActiveSet ................................................................. 193
AddAttributeVisitor ..................................................... 206
Approximation .............................................................. 231
  C3Approximation ....................................................... 258
  GaussProcApproximation .............................................. 446
  PecosApproximation .................................................. 823
  QMEApproximation ................................................... 870
  SurfpackApproximation .............................................. 995
  SurrogatesGPApprox ................................................. 1020
  SurrogatesPolyApprox .............................................. 1021
  TANA3Approximation .................................................. 1028
  TaylorApproximation ............................................... 1030
  VPSApproximation ................................................... 1060
APPSEvalMgr ............................................................. 246
ApreproWriter ............................................................ 252
AttachScaleVisitor ....................................................... 253
BaseConstructor .......................................................... 253
BootstrapSamplerBase< Data > ........................................ 256
  BootstrapSampler< Data > ............................................. 254
    BootstrapSamplerWithGS< Data, Getter, Setter > ............... 257
BootstrapSamplerBase< Teuchos::SerialDenseMatrix< OrdinalType, ScalarType > > .................. 256
  BootstrapSampler< Teuchos::SerialDenseMatrix< OrdinalType, ScalarType > > ............... 255
C3FnTrainPtrs ............................................................. 263
callback_data ........................................................... 264
COLINApplication ....................................................... 266
CommandShell ............................................................. 277
ConsoleRedirector ....................................................... 289
Constraints ................................................................. 290
  MixedVarConstraints ................................................. 532
  RelaxedVarConstraints ............................................. 891
CHAPTER 10. HIERARCHICAL INDEX

DakotaROLEqConstraints ................................................. 299
DakotaROLEqConstraintsGrad ......................................... 300
DakotaROLEqConstraintsHess ...................................... 301
DakotaROLIneqConstraints ........................................... 302
DakotaROLIneqConstraintsGrad .................................... 303
DakotaROLIneqConstraintsHess .................................... 304
DakotaROLObjective .................................................. 305
DakotaROLObjectiveGrad .............................................. 306
DakotaROLObjectiveHess .............................................. 306
DataEnvironment ...................................................... 307
DataEnvironmentRep .................................................. 308
DataInterface .......................................................... 327
DataMethod ............................................................ 328
DataMethodRep ......................................................... 329
DataModel .............................................................. 346
DataModelRep .......................................................... 347
DataResponses .......................................................... 354
DataResponsesRep ...................................................... 356
DataScaler ............................................................... 360
   NormalizationScaler ............................................ 774
   NoScaler .......................................................... 775
   StandardizationScaler .......................................... 993
DataVariables .......................................................... 369
DataVariablesRep ...................................................... 371
DerivInformedPropCovLogitTK < V, M > .............................. 389
DerivInformedPropCovTK < V, M > ................................ 389
DiscrepancyCorrection ............................................... 395
JEGAOptimizer::Driver .............................................. 401
Environment ........................................................... 408
   ExecutableEnvironment .......................................... 420
   LibraryEnvironment ............................................ 515
JEGAOptimizer::Evaluator ........................................... 412
NomadOptimizer::Evaluator ........................................ 417
JEGAOptimizer::EvaluatorCreator .................................. 419
ExperimentData ....................................................... 421
FileReadException ................................................... 431
   ResultsFileError ............................................. 910
   TabularDataTruncated ....................................... 1026
FunctionEvalFailure .................................................. 437
GeneralReader .......................................................... 451
GeneralWriter .......................................................... 452
GetLongOpt ............................................................. 452
   CommandLineHandler ........................................... 276
GP_Objective .......................................................... 455
Graphics ............................................................... 457
HDF5IOHelper .......................................................... 461
IntegerScale ............................................................ 473
10.1. CLASS HIERARCHY

- Interface
- ApplicationInterface
- DirectApplicInterface
- MatlabInterface
- PythonInterface
- ScilabInterface
- TestDriverInterface
- ParallelDirectApplicInterface
- SerialDirectApplicInterface
- SoleilDirectApplicInterface
- ProcessApplicInterface
- ProcessHandleApplicInterface
- ForkApplicInterface
- SpawnApplicInterface
- SysCallApplicInterface
- GridApplicInterface
- ApproximationInterface
- Iterator
- Analyzer
- NonD
- NonDCalibration
- NonDBayesCalibration
- NonDDREAMBayesCalibration
- NonDMUQBayesCalibration
- NonDQUESOBayesCalibration
- NonDGMPSABayesCalibration
- NonDWASABIBayesCalibration
- NonDExpansion
- NonDC3FunctionTrain
- NonDMultilevelFunctionTrain
- NonDPolynomialChaos
- NonDMultilevelPolynomialChaos
- NonDStochCollocation
- NonDMultilevelStochCollocation
- NonDSurrogateExpansion
- NonDIntegration
- NonDCubature
- NonDQuadrature
- NonDSparseGrid
- NonDInterval
- NonDGlobalInterval
- NonDGlobalEvidence
- NonDGlobalSingleInterval
- NonDLHSInterval
- NonDLHSEvidence
- NonDLHSSingleInterval
- NonDLocalInterval
### CHAPTER 10. HIERARCHICAL INDEX

<table>
<thead>
<tr>
<th>Item</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>NonDLocalEvidence</td>
<td>686</td>
</tr>
<tr>
<td>NonDLocalSingleInterval</td>
<td>700</td>
</tr>
<tr>
<td>NonDPOFDarts</td>
<td>726</td>
</tr>
<tr>
<td>NonDReliability</td>
<td>745</td>
</tr>
<tr>
<td>NonDGLocalReliability</td>
<td>661</td>
</tr>
<tr>
<td>NonDReliability</td>
<td>690</td>
</tr>
<tr>
<td>NonDRKDDarts</td>
<td>747</td>
</tr>
<tr>
<td>NonDSampling</td>
<td>750</td>
</tr>
<tr>
<td>NonDAdaptImpSampling</td>
<td>612</td>
</tr>
<tr>
<td>NonDAdaptiveSampling</td>
<td>617</td>
</tr>
<tr>
<td>NonDGPImpSampling</td>
<td>665</td>
</tr>
<tr>
<td>NonDLHSSampling</td>
<td>681</td>
</tr>
<tr>
<td>NonDMultilevelSampling</td>
<td>709</td>
</tr>
<tr>
<td>PStudyDACE</td>
<td>863</td>
</tr>
<tr>
<td>DDACEDesignCompExp</td>
<td>385</td>
</tr>
<tr>
<td>FSUDesignCompExp</td>
<td>433</td>
</tr>
<tr>
<td>ParamStudy</td>
<td>813</td>
</tr>
<tr>
<td>PSUADesignCompExp</td>
<td>865</td>
</tr>
<tr>
<td>Verification</td>
<td>1057</td>
</tr>
<tr>
<td>RichExtrapVerification</td>
<td>914</td>
</tr>
<tr>
<td>MetaIterator</td>
<td>522</td>
</tr>
<tr>
<td>CollabHybridMetaIterator</td>
<td>274</td>
</tr>
<tr>
<td>ConcurrentMetaIterator</td>
<td>278</td>
</tr>
<tr>
<td>EmbedHybridMetaIterator</td>
<td>406</td>
</tr>
<tr>
<td>SeqHybridMetaIterator</td>
<td>931</td>
</tr>
<tr>
<td>Minimizer</td>
<td>524</td>
</tr>
<tr>
<td>LeastSq</td>
<td>512</td>
</tr>
<tr>
<td>NL2SOLLeastSq</td>
<td>596</td>
</tr>
<tr>
<td>NLSSOLLLeastSq</td>
<td>601</td>
</tr>
<tr>
<td>SNLLLeastSq</td>
<td>976</td>
</tr>
<tr>
<td>Optimizer</td>
<td>784</td>
</tr>
<tr>
<td>APPSOptimizer</td>
<td>248</td>
</tr>
<tr>
<td>COLINOptimizer</td>
<td>269</td>
</tr>
<tr>
<td>CONMINOptimizer</td>
<td>281</td>
</tr>
<tr>
<td>JEGAOptimizer</td>
<td>505</td>
</tr>
<tr>
<td>NCSUOptimizer</td>
<td>577</td>
</tr>
<tr>
<td>NonlinearCGOptimizer</td>
<td>771</td>
</tr>
<tr>
<td>NOWPACOptimizer</td>
<td>777</td>
</tr>
<tr>
<td>OptDartsOptimizer</td>
<td>781</td>
</tr>
<tr>
<td>ROLOptimizer</td>
<td>917</td>
</tr>
<tr>
<td>SNLLOptimizer</td>
<td>980</td>
</tr>
<tr>
<td>SurrBasedMinimizer</td>
<td>1007</td>
</tr>
<tr>
<td>EffGlobalMinimizer</td>
<td>402</td>
</tr>
<tr>
<td>SurrBasedGlobalMinimizer</td>
<td>999</td>
</tr>
<tr>
<td>SurrBasedLocalMinimizer</td>
<td>1001</td>
</tr>
<tr>
<td>IteratorScheduler</td>
<td>499</td>
</tr>
<tr>
<td>LabelsWriter</td>
<td>511</td>
</tr>
<tr>
<td>LightWtBaseConstructor</td>
<td>518</td>
</tr>
</tbody>
</table>
10.1. CLASS HIERARCHY

<table>
<thead>
<tr>
<th>Class</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>LinearSolverBase</td>
<td>518</td>
</tr>
<tr>
<td>CholeskySolver</td>
<td>264</td>
</tr>
<tr>
<td>MatchesWC</td>
<td>520</td>
</tr>
<tr>
<td>Model</td>
<td>535</td>
</tr>
<tr>
<td>NestedModel</td>
<td>581</td>
</tr>
<tr>
<td>RecastModel</td>
<td>880</td>
</tr>
<tr>
<td>ActiveSubspaceModel</td>
<td>195</td>
</tr>
<tr>
<td>AdaptedBasisModel</td>
<td>202</td>
</tr>
<tr>
<td>DataTransformModel</td>
<td>363</td>
</tr>
<tr>
<td>ProbabilityTransformModel</td>
<td>834</td>
</tr>
<tr>
<td>RandomFieldModel</td>
<td>874</td>
</tr>
<tr>
<td>ScalingModel</td>
<td>919</td>
</tr>
<tr>
<td>WeightingModel</td>
<td>1064</td>
</tr>
<tr>
<td>SimulationModel</td>
<td>970</td>
</tr>
<tr>
<td>SurrogateModel</td>
<td>1015</td>
</tr>
<tr>
<td>DataFitSurrModel</td>
<td>312</td>
</tr>
<tr>
<td>HierarchSurrModel</td>
<td>467</td>
</tr>
<tr>
<td>MPIManager</td>
<td>572</td>
</tr>
<tr>
<td>MPIPackBuffer</td>
<td>573</td>
</tr>
<tr>
<td>MPIUnpackBuffer</td>
<td>575</td>
</tr>
<tr>
<td>NL2Res</td>
<td>596</td>
</tr>
<tr>
<td>NoDBBaseConstructor</td>
<td>603</td>
</tr>
<tr>
<td>NOWPACBlackBoxEvaluator</td>
<td>776</td>
</tr>
<tr>
<td>OutputManager</td>
<td>790</td>
</tr>
<tr>
<td>OutputWriter</td>
<td>793</td>
</tr>
<tr>
<td>ParallelConfiguration</td>
<td>794</td>
</tr>
<tr>
<td>ParallelLevel</td>
<td>798</td>
</tr>
<tr>
<td>ParallelLibrary</td>
<td>800</td>
</tr>
<tr>
<td>ParamResponsePair</td>
<td>810</td>
</tr>
<tr>
<td>partial_prp_equality</td>
<td>819</td>
</tr>
<tr>
<td>partial_prp_hash</td>
<td>820</td>
</tr>
<tr>
<td>PebblldBranching</td>
<td>820</td>
</tr>
<tr>
<td>PebblldBranchSub</td>
<td>821</td>
</tr>
<tr>
<td>PrefixingLineFilter</td>
<td>833</td>
</tr>
<tr>
<td>ProblemDescDB</td>
<td>838</td>
</tr>
<tr>
<td>NIDRProblemDescDB</td>
<td>590</td>
</tr>
<tr>
<td>ProgramOptions</td>
<td>858</td>
</tr>
<tr>
<td>QuesoJointPdf&lt;V, M&gt;</td>
<td>872</td>
</tr>
<tr>
<td>QuesoVectorRV&lt;V, M&gt;</td>
<td>873</td>
</tr>
<tr>
<td>RealScale</td>
<td>879</td>
</tr>
<tr>
<td>ReducedBasis</td>
<td>890</td>
</tr>
<tr>
<td>Response</td>
<td>894</td>
</tr>
<tr>
<td>ExperimentResponse</td>
<td>429</td>
</tr>
<tr>
<td>SimulationResponse</td>
<td>973</td>
</tr>
<tr>
<td>RestartWriter</td>
<td>902</td>
</tr>
<tr>
<td>ResultAttribute&lt;T&gt;</td>
<td>903</td>
</tr>
<tr>
<td>ResultsDBBase</td>
<td>906</td>
</tr>
<tr>
<td>ResultsDBAny</td>
<td>904</td>
</tr>
<tr>
<td>Class/Method</td>
<td>Page</td>
</tr>
<tr>
<td>----------------------------------------------------------------------------</td>
<td>------</td>
</tr>
<tr>
<td>ResultsDBHDF5</td>
<td>907</td>
</tr>
<tr>
<td>ResultsEntry&lt; StoredType &gt;</td>
<td>909</td>
</tr>
<tr>
<td>ResultsManager</td>
<td>911</td>
</tr>
<tr>
<td>ResultsNames</td>
<td>913</td>
</tr>
<tr>
<td>ScalingOptions</td>
<td>926</td>
</tr>
<tr>
<td>SensAnalysisGlobal</td>
<td>928</td>
</tr>
<tr>
<td>TrackerHTTP::Server</td>
<td>936</td>
</tr>
<tr>
<td>SharedApproxData</td>
<td>937</td>
</tr>
<tr>
<td>SharedC3ApproxData</td>
<td>943</td>
</tr>
<tr>
<td>SharedPecosApproxData</td>
<td>947</td>
</tr>
<tr>
<td>SharedSurfpackApproxData</td>
<td>954</td>
</tr>
<tr>
<td>SharedResponseData</td>
<td>950</td>
</tr>
<tr>
<td>SharedResponseDataRep</td>
<td>952</td>
</tr>
<tr>
<td>SharedVariablesData</td>
<td>956</td>
</tr>
<tr>
<td>SharedVariablesDataRep</td>
<td>964</td>
</tr>
<tr>
<td>SNLL.Base</td>
<td>974</td>
</tr>
<tr>
<td>SNLLLeastSq</td>
<td>976</td>
</tr>
<tr>
<td>SNLLOptimizer</td>
<td>980</td>
</tr>
<tr>
<td>SOLBase</td>
<td>988</td>
</tr>
<tr>
<td>NLSSOLLeastSq</td>
<td>601</td>
</tr>
<tr>
<td>StringScale</td>
<td>994</td>
</tr>
<tr>
<td>Surrogate</td>
<td>1012</td>
</tr>
<tr>
<td>GaussianProcess</td>
<td>437</td>
</tr>
<tr>
<td>PolynomialRegression</td>
<td>829</td>
</tr>
<tr>
<td>TabularReader</td>
<td>1027</td>
</tr>
<tr>
<td>TabularWriter</td>
<td>1027</td>
</tr>
<tr>
<td>TKFactoryDIPC</td>
<td>1038</td>
</tr>
<tr>
<td>TKFactoryDIPCLogit</td>
<td>1039</td>
</tr>
<tr>
<td>TPLDataTransfer</td>
<td>1040</td>
</tr>
<tr>
<td>TrackerHTTP</td>
<td>1041</td>
</tr>
<tr>
<td>TraitsBase</td>
<td>1043</td>
</tr>
<tr>
<td>AppsTraits</td>
<td>251</td>
</tr>
<tr>
<td>COLINTraits</td>
<td>273</td>
</tr>
<tr>
<td>CONMINTraits</td>
<td>288</td>
</tr>
<tr>
<td>DataFitSurrBasedLocalTraits</td>
<td>311</td>
</tr>
<tr>
<td>DLSolverTraits</td>
<td>399</td>
</tr>
<tr>
<td>DOTTraits</td>
<td>399</td>
</tr>
<tr>
<td>EffGlobalTraits</td>
<td>405</td>
</tr>
<tr>
<td>HierarchSurrBasedLocalTraits</td>
<td>466</td>
</tr>
<tr>
<td>JEGATraits</td>
<td>511</td>
</tr>
<tr>
<td>NCSUTraits</td>
<td>580</td>
</tr>
<tr>
<td>NL2SOLLLeastSqTraits</td>
<td>599</td>
</tr>
<tr>
<td>NLPQLPTraits</td>
<td>599</td>
</tr>
<tr>
<td>NLSSOLLLeastSqTraits</td>
<td>603</td>
</tr>
<tr>
<td>NomadTraits</td>
<td>604</td>
</tr>
<tr>
<td>NonlinearCGTraits</td>
<td>774</td>
</tr>
<tr>
<td>NOWPACTraits</td>
<td>779</td>
</tr>
<tr>
<td>NPSOLTraits</td>
<td>780</td>
</tr>
</tbody>
</table>
## 10.1. Class Hierarchy

<table>
<thead>
<tr>
<th>Class</th>
<th>Line No.</th>
</tr>
</thead>
<tbody>
<tr>
<td>OptDartsTraits</td>
<td>783</td>
</tr>
<tr>
<td>PebbleTraits</td>
<td>822</td>
</tr>
<tr>
<td>ROLTraits</td>
<td>918</td>
</tr>
<tr>
<td>SNLLLeastSqTraits</td>
<td>979</td>
</tr>
<tr>
<td>SNLLTraits</td>
<td>987</td>
</tr>
<tr>
<td>SurrBasedGlobalTraits</td>
<td>1000</td>
</tr>
<tr>
<td>UsageTracker</td>
<td>1045</td>
</tr>
<tr>
<td>Var_check</td>
<td>1046</td>
</tr>
<tr>
<td>Var_rcheck</td>
<td>1047</td>
</tr>
<tr>
<td>Variables</td>
<td>1047</td>
</tr>
<tr>
<td>MixedVariables</td>
<td>533</td>
</tr>
<tr>
<td>RelaxedVariables</td>
<td>892</td>
</tr>
<tr>
<td>VLint</td>
<td>1059</td>
</tr>
<tr>
<td>VLreal</td>
<td>1059</td>
</tr>
<tr>
<td>VLstr</td>
<td>1059</td>
</tr>
<tr>
<td>WorkdirHelper</td>
<td>1065</td>
</tr>
</tbody>
</table>
Chapter 11

Class Index

11.1 Class List

Here are the classes, structs, unions and interfaces with brief descriptions:

- **ActiveSet**
  Container class for active set tracking information. Contains the active set request vector and the derivative variables vector. 193

- **ActiveSubspaceModel**
  Active subspace model for input (variable space) reduction. 195

- **AdaptedBasisModel**
  Adapted basis model for input (variable space) reduction. 202

- **AddAttributeVisitor**
  Objects of this class are called by boost::appy_visitor to add attributes to HDF5 objects. 206

- **Analyzer**
  Base class for NonD, DACE, and ParamStudy branches of the iterator hierarchy. 207

- **ApplicationInterface**
  Derived class within the interface class hierarchy for supporting interfaces to simulation codes. 215

- **Approximation**
  Base class for the approximation class hierarchy. 231

- **ApproximationInterface**
  Derived class within the interface class hierarchy for supporting approximations to simulation-based results. 239

- **APPSEvalMgr**
  Evaluation manager class for APPSPACK. 246

- **APPSOptimizer**
  Wrapper class for HOPSPACK. 248

- **AppsTraits**
  HOPSPACK-specific traits class. 251

- **ApreproWriter**
  Utility used in derived write_core to write in aprepro format. 252

- **AttachScaleVisitor**
  Objects of this class are called by boost::appy_visitor to add dimension scales (RealScale or StringScale) to HDF5 datasets. 253

- **BaseConstructor**
  Dummy struct for overloading letter-envelope constructors. 253
BootstrapSampler< Data >
    Actual bootstrap sampler implementation for common data types .................................. 254
BootstrapSampler< Teuchos::SerialDenseMatrix< OrdinalType, ScalarType > >
    Bootstrap sampler that is specialized to allow for the bootstrapping of RealMatrix .................. 255
BootstrapSamplerBase< Data >
    Base class/interface for the bootstrap sampler .......................................................... 256
BootstrapSamplerWithGS< Data, Getter, Setter >
    A derived sampler to allow for user specification of the accessor methods .......................... 257
C3Approximation
    Derived approximation class for global basis polynomials .............................................. 258
C3FnTrainPtrs
    Handle for reference-counted pointer to C3FnTrainPtrsRep body ..................................... 263
callback_data
    Can be used to solve Ax = b, provided that A is a symmetric matrix .................................. 264
CholeskySolver
    Utility class which defines convenience operators for spawning processes with system calls . . 277
ConcurrentMetaIterator
    Meta-iterator for multi-start iteration or pareto set optimization ..................................... 278
CONMINOptimizer
    Wrapper class for the CONMIN optimization library ..................................................... 281
CONMINTraits
    A version of TraitsBase specialized for CONMIN optimizers ............................................ 288
DataEnvironment
    Handle class for environment specification data .......................................................... 307
DataEnvironmentRep
    Body class for environment specification data ............................................................. 308
11.1. CLASS LIST

**DataFitSurrBasedLocalTraits**
Class for provably-convergent local surrogate-based optimization and nonlinear least squares  . 311

**DataFitSurrModel**
Derived model class within the surrogate model branch for managing data fit surrogates (global and local) ........................................................... 312

**DataInterface**
Handle class for interface specification data .......................................................... 327

**DataMethod**
Handle class for method specification data ........................................................... 328

**DataMethodRep**
Body class for method specification data .............................................................. 329

**DataModel**
Handle class for model specification data ............................................................. 346

**DataModelRep**
Body class for model specification data .............................................................. 347

**DataResponses**
Handle class for responses specification data ......................................................... 354

**DataResponsesRep**
Body class for responses specification data .......................................................... 356

**DataScaler**
Computes the scaling coefficients and scales a 2D data matrix with dimensions num_samples by num_features ................................................................. 360

**DataTransformModel**
Data transformation specialization of RecastModel .................................................. 363

**DataVariables**
Handle class for variables specification data ........................................................ 369

**DataVariablesRep**
Body class for variables specification data ........................................................... 371

**DDACEDesignCompExp**
Wrapper class for the DDACE design of experiments library ................................... 385

**DerivInformedPropCovLogitTK< V, M >**
Dakota transition kernel that updates proposal covariance based on derivatives (for logit random walk case) ................................................................. 389

**DerivInformedPropCovTK< V, M >**
Dakota transition kernel that updates proposal covariance based on derivatives (for random walk case) ................................................................. 389

**DirectApplicInterface**
Derived application interface class which spawns simulation codes and testers using direct procedure calls ................................................................. 390

**DiscrepancyCorrection**
Base class for discrepancy corrections ..................................................................... 395

**DLSolverTraits**
A version of TraitsBase specialized for DLSolver .................................................... 399

**DOTTraits**
Wrapper class for the DOT optimization library ..................................................... 399

**JEGAOptimizer::Driver**
A subclass of the JEGA front end driver that exposes the individual protected methods to execute the algorithm ................................................................. 401

**EffGlobalMinimizer** ............................................................................................. 402
### CHAPTER 11. CLASS INDEX

<table>
<thead>
<tr>
<th>Class Name</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>EffGlobalTraits</td>
<td>Implementation of Efficient Global Optimization/Least Squares algorithms</td>
<td>405</td>
</tr>
<tr>
<td>EmbedHybridMetaIterator</td>
<td>Meta-iterator for closely-coupled hybrid iteration, typically involving the embedding of local search methods within global search methods</td>
<td>406</td>
</tr>
<tr>
<td>Environment</td>
<td>Base class for the environment class hierarchy</td>
<td>408</td>
</tr>
<tr>
<td>JEGAOptimizer::Evaluator</td>
<td>An evaluator specialization that knows how to interact with Dakota</td>
<td>412</td>
</tr>
<tr>
<td>NomadOptimizer::Evaluator</td>
<td>NOMAD-based Evaluator class</td>
<td>417</td>
</tr>
<tr>
<td>JEGAOptimizer::EvaluatorCreator</td>
<td>A specialization of the JEGA::FrontEnd::EvaluatorCreator that creates a new instance of a Evaluator</td>
<td>419</td>
</tr>
<tr>
<td>ExecutableEnvironment</td>
<td>Environment corresponding to execution as a stand-alone application</td>
<td>420</td>
</tr>
<tr>
<td>ExperimentData</td>
<td>Interpolation method for interpolating between experimental and model data. I need to work on inputs/outputs to this method. For now, this assumes interpolation of functional data</td>
<td>421</td>
</tr>
<tr>
<td>ExperimentResponse</td>
<td>Container class for response functions and their derivatives. ExperimentResponse provides the body class</td>
<td>429</td>
</tr>
<tr>
<td>FileReadException</td>
<td>Base class for Dakota file read exceptions (to allow catching both tabular and general file truncation issues)</td>
<td>431</td>
</tr>
<tr>
<td>ForkApplicInterface</td>
<td>Derived application interface class which spawns simulation codes using fork/execvp/waitpid</td>
<td>431</td>
</tr>
<tr>
<td>FSUDesignCompExp</td>
<td>Wrapper class for the FSUDace QMC/CVT library</td>
<td>433</td>
</tr>
<tr>
<td>FunctionEvalFailure</td>
<td>Exception class for function evaluation failures</td>
<td>437</td>
</tr>
<tr>
<td>GaussianProcess</td>
<td>The GaussianProcess constructs a Gaussian Process regressor surrogate given a matrix of data</td>
<td>437</td>
</tr>
<tr>
<td>GaussProcApproximation</td>
<td>Derived approximation class for Gaussian Process implementation</td>
<td>446</td>
</tr>
<tr>
<td>GeneralReader</td>
<td>Utility used in derived read_core to read in generic format</td>
<td>451</td>
</tr>
<tr>
<td>GeneralWriter</td>
<td>Utility used in derived write_core to write in generic format</td>
<td>452</td>
</tr>
<tr>
<td>GetLongOpt</td>
<td>GetLongOpt is a general command line utility from S. Manoharan (Advanced Computer Research Institute, Lyon, France)</td>
<td>452</td>
</tr>
<tr>
<td>GP_Objective</td>
<td>ROL objective function for the Gaussian Process (GP) surrogate</td>
<td>455</td>
</tr>
<tr>
<td>Graphics</td>
<td>Single interface to 2D (motif) and 3D (PLPLOT) graphics; there is only one instance of this OutputManager::dakotaGraphics</td>
<td>457</td>
</tr>
</tbody>
</table>
11.1. CLASS LIST

**GridApplicInterface**
Derived application interface class which spawns simulation codes using grid services such as Condor or Globus .................................................. 459

**HDF5IOHelper** ................................................................. 461

**HierarchSurrBasedLocalTraits**
Class for multilevel-multifidelity optimization algorithm ................................................. 466

**HierarchSurrModel**
Derived model class within the surrogate model branch for managing hierarchical surrogates (models of varying fidelity) .............................................. 467

**IntegerScale**
Data structure for storing int-valued dimension scale ..................................................... 473

**Interface**
Base class for the interface class hierarchy ..................................................................... 474

**Iterator**
Base class for the iterator class hierarchy ..................................................................... 483

**IteratorScheduler**
This class encapsulates scheduling operations for concurrent sub-iteration within an outer level context (e.g., meta-iteration, nested models) .................................................. 499

**JEGAOptimizer**
A version of Dakota::Optimizer for instantiation of John Eddy’s Genetic Algorithms (JEGA) .......................................................... 505

**JEGATraits**
A version of TraitsBase specialized for John Eddy’s Genetic Algorithms (JEGA) ........... 511

**LabelsWriter**
Utility used in derived write_core to write labels in tabular format .............................. 511

**LeastSq**
Base class for the nonlinear least squares branch of the iterator hierarchy ..................... 512

**LibraryEnvironment**
Environment corresponding to execution as an embedded library .................................. 515

**LightWtBaseConstructor**
Dummy struct for overloading constructors used in on-the-fly Model instantiations .......... 518

**LinearSolverBase**
Serves as an API for derived solvers ............................................................................. 518

**MatchesWC**
Predicate that returns true when the passed path matches the wild_card with which it was configured. Currently supports * and ? .................................................. 520

**MatlabInterface** ........................................................................ 521

**MetaIterator**
Base class for meta-iterators ....................................................................................... 522

**Minimizer**
Base class for the optimizer and least squares branches of the iterator hierarchy .......... 524

**MixedVarConstraints**
Derived class within the Constraints hierarchy which separates continuous and discrete variables (no domain type array merging) .................................................. 532

**MixedVariables**
Derived class within the Variables hierarchy which separates continuous and discrete variables (no domain type array merging) .................................................. 533

**Model**
Base class for the model class hierarchy ...................................................................... 535
### CHAPTER 11. CLASS INDEX

<table>
<thead>
<tr>
<th>Class Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>MPIManager</td>
<td>Class MPIManager to manage Dakota’s MPI world, which may be a subset of MPI_COMM_WORLD</td>
</tr>
<tr>
<td>MPIPackBuffer</td>
<td>Class for packing MPI message buffers</td>
</tr>
<tr>
<td>MPIUnpackBuffer</td>
<td>Class for unpacking MPI message buffers</td>
</tr>
<tr>
<td>NCSUOptimizer</td>
<td>Wrapper class for the NCSU DIRECT optimization library</td>
</tr>
<tr>
<td>NCSUTraits</td>
<td>A version of TraitsBase specialized for NCSU optimizers</td>
</tr>
<tr>
<td>NestedModel</td>
<td>Derived model class which performs a complete sub-iterator execution within every evaluation of the model</td>
</tr>
<tr>
<td>NIDRProblemDescDB</td>
<td>The derived input file database utilizing the new IDR parser</td>
</tr>
<tr>
<td>NL2Res</td>
<td>Auxiliary information passed to calcr and calcj via ur</td>
</tr>
<tr>
<td>NL2SOLLeastSq</td>
<td>Wrapper class for the NL2SOL nonlinear least squares library</td>
</tr>
<tr>
<td>NL2SOLLeastSqTraits</td>
<td>A version of TraitsBase specialized for NL2SOL nonlinear least squares library</td>
</tr>
<tr>
<td>NLQLPTraits</td>
<td>Wrapper class for the NLPQLP optimization library, Version 2.0</td>
</tr>
<tr>
<td>NLSSOLLeastSq</td>
<td>Wrapper class for the NLSSOL nonlinear least squares library</td>
</tr>
<tr>
<td>NLSSOLLeastSqTraits</td>
<td>A version of TraitsBase specialized for NLSSOL nonlinear least squares library</td>
</tr>
<tr>
<td>NoDBBaseConstructor</td>
<td>Dummy struct for overloading constructors used in on-the-fly instantiations without ProblemDescDB support</td>
</tr>
<tr>
<td>NomadTraits</td>
<td>Wrapper class for NOMAD Optimizer</td>
</tr>
<tr>
<td>NonD</td>
<td>Base class for all nondeterministic iterators (the DAKOTA/UQ branch)</td>
</tr>
<tr>
<td>NonDAdaptImpSampling</td>
<td>Class for the Adaptive Importance Sampling methods within DAKOTA</td>
</tr>
<tr>
<td>NonDAdaptiveSampling</td>
<td>Class for testing various Adaptively sampling methods using geometric, statistical, and topological information of the surrogate</td>
</tr>
<tr>
<td>NonDBayesCalibration</td>
<td>Base class for Bayesian inference: generates posterior distribution on model parameters given experimental data</td>
</tr>
<tr>
<td>NonDC3FunctionTrain</td>
<td>Nonintrusive uncertainty quantification with the C3 library</td>
</tr>
<tr>
<td>NonDCalibration</td>
<td>Derived nondeterministic class that generates N-dimensional numerical cubature points for evaluation of expectation integrals</td>
</tr>
</tbody>
</table>
11.1. CLASS LIST

NonDDREAMBayesCalibration
Bayesian inference using the DREAM approach ........................................ 640

NonDExpansion
Base class for polynomial chaos expansions (PCE), stochastic collocation (SC) and functional
tensor train (FT) ......................................................................................... 644

NonDGGlobalEvidence
Class for the Dempster-Shafer Evidence Theory methods within DAKOTA/UQ .... 656

NonDGGlobalInterval
Class for using global nongradient-based optimization approaches to calculate interval bounds
for epistemic uncertainty quantification ....................................................... 657

NonDGGlobalReliability
Class for global reliability methods within DAKOTA/UQ ............................. 661

NonDGGlobalSingleInterval
Class for using global nongradient-based optimization approaches to calculate interval bounds
for epistemic uncertainty quantification ....................................................... 664

NonDGPlmpSampling
Class for the Gaussian Process-based Importance Sampling method ............. 665

NonDGPMSEBayesCalibration
Generates posterior distribution on model parameters given experiment data .... 668

NonDIIntegration
Derived nondeterministic class that generates N-dimensional numerical integration points for
evaluation of expectation integrals .............................................................. 672

NonDInterval
Base class for interval-based methods within DAKOTA/UQ ........................ 676

NonDLHSEvidence
Class for the Dempster-Shafer Evidence Theory methods within DAKOTA/UQ .... 678

NonDLHSInterval
Class for the LHS-based interval methods within DAKOTA/UQ .................... 679

NonDLHSSampling
Performs LHS and Monte Carlo sampling for uncertainty quantification .......... 681

NonDLHSInterval
Class for pure interval propagation using LHS ........................................... 685

NonDLLocalEvidence
Class for the Dempster-Shafer Evidence Theory methods within DAKOTA/UQ .... 686

NonDLLocalInterval
Class for using local gradient-based optimization approaches to calculate interval bounds for
epistemic uncertainty quantification ........................................................... 687

NonDLLocalReliability
Class for the reliability methods within DAKOTA/UQ .................................. 690

NonDLLocalSingleInterval
Class for using local gradient-based optimization approaches to calculate interval bounds for
epistemic uncertainty quantification ........................................................... 700

NonDMultilevelFunctionTrain
Nonintrusive polynomial chaos expansion approaches to uncertainty quantification .... 702

NonDMultilevelPoinomialChaos
Nonintrusive polynomial chaos expansion approaches to uncertainty quantification .... 705

NonDMultilevelSampling
Performs Multilevel Monte Carlo sampling for uncertainty quantification ......... 709
<table>
<thead>
<tr>
<th>Class Index</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>NonDMultilevelStochCollocation</strong></td>
<td>Nonintrusive stochastic collocation approaches to uncertainty quantification</td>
</tr>
<tr>
<td><strong>NonDMUQBayesCalibration</strong></td>
<td>Dakota interface to MUQ (MIT Uncertainty Quantification) library</td>
</tr>
<tr>
<td><strong>NonDPOFDarts</strong></td>
<td>Base class for POF Dart methods within DAKOTA/UQ</td>
</tr>
<tr>
<td><strong>NonDPolynomialChaos</strong></td>
<td>Nonintrusive polynomial chaos expansion approaches to uncertainty quantification</td>
</tr>
<tr>
<td><strong>NonDQuadrature</strong></td>
<td>Derived nondeterministic class that generates N-dimensional numerical quadrature points for evaluation of expectation integrals over uncorrelated standard normals/uniforms/exponentials/betas/gammas</td>
</tr>
<tr>
<td><strong>NonDQUESTOBayesCalibration</strong></td>
<td>Bayesian inference using the QUESO library from UT Austin</td>
</tr>
<tr>
<td><strong>NonDReliability</strong></td>
<td>Base class for the reliability methods within DAKOTA/UQ</td>
</tr>
<tr>
<td><strong>NonDRKDDarts</strong></td>
<td>Base class for the Recursive k-d Dart methods within DAKOTA/UQ</td>
</tr>
<tr>
<td><strong>NonDSampling</strong></td>
<td>Base class for common code between NonDLHSSampling, NonDAdaptImpSampling, and other specializations</td>
</tr>
<tr>
<td><strong>NonDSparseGrid</strong></td>
<td>Derived nondeterministic class that generates N-dimensional Smolyak sparse grids for numerical evaluation of expectation integrals over independent standard random variables</td>
</tr>
<tr>
<td><strong>NonDStochCollocation</strong></td>
<td>Nonintrusive stochastic collocation approaches to uncertainty quantification</td>
</tr>
<tr>
<td><strong>NonDSurrogateExpansion</strong></td>
<td>Generic uncertainty quantification with Model-based stochastic expansions</td>
</tr>
<tr>
<td><strong>NonDASABIBayesCalibration</strong></td>
<td>WASABI - Weighted Adaptive Surrogate Approximations for Bayesian Inference</td>
</tr>
<tr>
<td><strong>NonlinearCGOptimizer</strong></td>
<td>Nonlinear CG Optimizer</td>
</tr>
<tr>
<td><strong>NonlinearCGTraits</strong></td>
<td>A version of TraitsBase specialized for NonlinearCG optimizers</td>
</tr>
<tr>
<td><strong>NormalizationScaler</strong></td>
<td>Normalizes the data using max and min feature values</td>
</tr>
<tr>
<td><strong>NoScaler</strong></td>
<td>Leaves the data unscaled</td>
</tr>
<tr>
<td><strong>NOWPACBlackBoxEvaluator</strong></td>
<td>Derived class for plugging Dakota evaluations into NOWPAC solver</td>
</tr>
<tr>
<td><strong>NOWPACOptimizer</strong></td>
<td>Wrapper class for the (S)NOWPAC optimization algorithms from Florian Augustin (MIT)</td>
</tr>
<tr>
<td><strong>NOWPACTraits</strong></td>
<td>A version of TraitsBase specialized for NOWPAC optimizers</td>
</tr>
<tr>
<td><strong>NPSOLTraits</strong></td>
<td>Wrapper class for the NPSOL optimization library</td>
</tr>
<tr>
<td><strong>OptDartsOptimizer</strong></td>
<td>Wrapper class for OptDarts Optimizer</td>
</tr>
<tr>
<td><strong>OptDartsTraits</strong></td>
<td>A version of TraitsBase specialized for OptDarts</td>
</tr>
</tbody>
</table>
### 11.1. CLASS LIST

<table>
<thead>
<tr>
<th>Class</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Optimizer</strong></td>
<td>Base class for the optimizer branch of the iterator hierarchy</td>
</tr>
<tr>
<td><strong>OutputManager</strong></td>
<td>Class to manage redirection of stdout/stderr, keep track of current redir state, and manage rank 0 output. Also manage tabular data output for post-processing with Matlab, Tecplot, etc. and delegate to Graphics for X Windows Graphics</td>
</tr>
<tr>
<td><strong>OutputWriter</strong></td>
<td></td>
</tr>
<tr>
<td><strong>ParallelConfiguration</strong></td>
<td>Container class for a set of ParallelLevel list iterators that collectively identify a particular multilevel parallel configuration</td>
</tr>
<tr>
<td><strong>ParallelDirectApplicInterface</strong></td>
<td>Sample derived interface class for testing parallel simulator plug-ins using assign_rep()</td>
</tr>
<tr>
<td><strong>ParallelLevel</strong></td>
<td>Container class for the data associated with a single level of communicator partitioning</td>
</tr>
<tr>
<td><strong>ParallelLibrary</strong></td>
<td>Class for partitioning multiple levels of parallelism and managing message passing within these levels</td>
</tr>
<tr>
<td><strong>ParamResponsePair</strong></td>
<td>Container class for a variables object, a response object, and an evaluation id</td>
</tr>
<tr>
<td><strong>ParamStudy</strong></td>
<td>Class for vector, list, centered, and multidimensional parameter studies</td>
</tr>
<tr>
<td><strong>partial_prp_equality</strong></td>
<td>Predicate for comparing ONLY the interfaceId and Vars attributes of PRPair</td>
</tr>
<tr>
<td><strong>partial_prp_hash</strong></td>
<td>Wrapper to delegate to the ParamResponsePair hash_value function</td>
</tr>
<tr>
<td><strong>PebbldBranching</strong></td>
<td>Main Branching class for the PEBBL-based Minimizer</td>
</tr>
<tr>
<td><strong>PebbldBranchSub</strong></td>
<td>Sub Branch class for the PEBBL-based Minimizer</td>
</tr>
<tr>
<td><strong>PebbldTraits</strong></td>
<td>Wrapper class for experimental PebbldMinimizer</td>
</tr>
<tr>
<td><strong>PecosApproximation</strong></td>
<td>Derived approximation class for global basis polynomials</td>
</tr>
<tr>
<td><strong>PolynomialRegression</strong></td>
<td>Constructs a polynomial regressor using ordinary least squares</td>
</tr>
<tr>
<td><strong>PrefixingLineFilter</strong></td>
<td></td>
</tr>
<tr>
<td><strong>ProbabilityTransformModel</strong></td>
<td>Probability transformation specialization of RecastModel</td>
</tr>
<tr>
<td><strong>ProblemDescDB</strong></td>
<td>The database containing information parsed from the DAKOTA input file</td>
</tr>
<tr>
<td><strong>ProcessApplicInterface</strong></td>
<td>Derived application interface class that spawns a simulation code using a separate process and communicates with it through files</td>
</tr>
<tr>
<td><strong>ProcessHandleApplicInterface</strong></td>
<td>Derived application interface class that spawns a simulation code using a separate process, receives a process identifier, and communicates with the spawned process through files</td>
</tr>
<tr>
<td><strong>ProgramOptions</strong></td>
<td>ProgramOptions stores options whether from the CLH or from library user; initially valid only on worldRank = 0, but then broadcast in ParallelLibrary::push_output_tag()</td>
</tr>
</tbody>
</table>


<table>
<thead>
<tr>
<th>Class Name</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>PStudyDACE</td>
<td>Base class for managing common aspects of parameter studies and design of experiments methods</td>
<td>863</td>
</tr>
<tr>
<td>PSUADEDesignCompExp</td>
<td>Wrapper class for the PSUADE library</td>
<td>865</td>
</tr>
<tr>
<td>PythonInterface</td>
<td></td>
<td>868</td>
</tr>
<tr>
<td>QMEAApproximation</td>
<td>Derived approximation class for QMEA Quadratic Multipoint Exponential Approximation (a multipoint approximation)</td>
<td>870</td>
</tr>
<tr>
<td>QuesoJointPdf $&lt;$ V, M $&gt;$</td>
<td>Dakota specialization of QUESO generic joint PDF</td>
<td>872</td>
</tr>
<tr>
<td>QuesoVectorRV $&lt;$ V, M $&gt;$</td>
<td>Dakota specialization of QUESO vector-valued random variable</td>
<td>873</td>
</tr>
<tr>
<td>RandomFieldModel</td>
<td>Random field model, capable of generating and then forward propagating</td>
<td>874</td>
</tr>
<tr>
<td>RealScale</td>
<td>Data structure for storing real-valued dimension scale</td>
<td>879</td>
</tr>
<tr>
<td>RecastModel</td>
<td>Derived model class which provides a thin wrapper around a sub-model in order to recast the form of its inputs and/or outputs</td>
<td>880</td>
</tr>
<tr>
<td>ReducedBasis</td>
<td></td>
<td>890</td>
</tr>
<tr>
<td>RelaxedVarConstraints</td>
<td>Derived class within the Constraints hierarchy which employs relaxation of discrete variables</td>
<td>891</td>
</tr>
<tr>
<td>RelaxedVariables</td>
<td>Derived class within the Variables hierarchy which employs the relaxation of discrete variables</td>
<td>892</td>
</tr>
<tr>
<td>Response</td>
<td>Container class for response functions and their derivatives. Response provides the enveloper base class</td>
<td>894</td>
</tr>
<tr>
<td>RestartWriter</td>
<td></td>
<td>902</td>
</tr>
<tr>
<td>ResultAttribute $&lt;$ T $&gt;$</td>
<td>Data structure for a single Real, String, or int valued attribute</td>
<td>903</td>
</tr>
<tr>
<td>ResultsDBAny</td>
<td></td>
<td>904</td>
</tr>
<tr>
<td>ResultsDBBase</td>
<td></td>
<td>906</td>
</tr>
<tr>
<td>ResultsDBHDF5</td>
<td>Manage interactions between ResultsManager and the low-level HDFIOHelper class</td>
<td>907</td>
</tr>
<tr>
<td>ResultsEntry $&lt;$ StoredType $&gt;$</td>
<td>Class to manage in-core vs. file database lookups</td>
<td>909</td>
</tr>
<tr>
<td>ResultsFileError</td>
<td>Exception throw for other results file read error</td>
<td>910</td>
</tr>
<tr>
<td>ResultsManager</td>
<td>Results manager for iterator final data</td>
<td>911</td>
</tr>
<tr>
<td>ResultsNames</td>
<td>List of valid names for iterator results</td>
<td>913</td>
</tr>
<tr>
<td>RichExtrapVerification</td>
<td>Class for Richardson extrapolation for code and solution verification</td>
<td>914</td>
</tr>
<tr>
<td>ROLOptimizer</td>
<td></td>
<td>917</td>
</tr>
<tr>
<td>ROLTraits</td>
<td></td>
<td>918</td>
</tr>
<tr>
<td>ScalingModel</td>
<td>Scaling specialization of RecastModel</td>
<td>919</td>
</tr>
</tbody>
</table>
11.1. CLASS LIST

ScalingOptions
Simple container for user-provided scaling data, possibly expanded by replicates through the models .................................................. 926

ScilabInterface .............................................................................. 927

SensAnalysisGlobal
Class for a utility class containing correlation calculations and variance-based decomposition 928

SeqHybridMetaIterator
Method for sequential hybrid iteration using multiple optimization and nonlinear least squares methods on multiple models of varying fidelity ........................................... 931

SerialDirectApplicInterface
Sample derived interface class for testing serial simulator plug-ins using assign_rep() .... 935

TrackerHTTP::Server
Struct to hold tracker/proxy pairs .................................................. 936

SharedApproxData
Base class for the shared approximation data class hierarchy .................. 937

SharedC3ApproxData
Derived approximation class for global basis polynomials ....................... 943

SharedPecosApproxData
Derived approximation class for global basis polynomials ....................... 947

SharedResponseData
Container class encapsulating variables data that can be shared among a set of Response instances .......................................................... 950

SharedResponseDataRep
The representation of a SharedResponseData instance. This representation, or body, may be shared by multiple SharedResponseData handle instances ........................................... 952

SharedSurfpackApproxData
Derived approximation class for Surfpack approximation classes. Interface between Surfpack and Dakota .......................................................... 954

SharedVariablesData
Container class encapsulating variables data that can be shared among a set of Variables instances .......................................................... 956

SharedVariablesDataRep
The representation of a SharedVariablesData instance. This representation, or body, may be shared by multiple SharedVariablesData handle instances ........................................... 964

SimulationModel
Derived model class which utilizes a simulation-based application interface to map variables into responses .......................................................... 970

SimulationResponse
Container class for response functions and their derivatives. SimulationResponse provides the body class ........................................................................... 973

SNLLBase
Base class for OPT++ optimization and least squares methods ............... 974

SNLLLeastSq
Wrapper class for the OPT++ optimization library .................................. 976

SNLLLeastSqTraits
A version of TraitsBase specialized for SNLLLeastSq ............................. 979

SNLLOptimizer
Wrapper class for the OPT++ optimization library .................................. 980
### CHAPTER 11. CLASS INDEX

<table>
<thead>
<tr>
<th>Class</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>SNLLTraits</td>
<td>A version of TraitsBase specialized for SNLL optimizers</td>
<td>987</td>
</tr>
<tr>
<td>SOLBase</td>
<td>Base class for Stanford SOL software</td>
<td>988</td>
</tr>
<tr>
<td>SoleilDirectApplicInterface</td>
<td>Sample derived interface class for testing serial simulator plug-ins using assign_rep()</td>
<td>990</td>
</tr>
<tr>
<td>SpawnApplicInterface</td>
<td>Derived application interface class which spawns simulation codes using spawnvp</td>
<td>992</td>
</tr>
<tr>
<td>StandardizationScaler</td>
<td>Standardizes the data so the each feature has zero mean and unit variance</td>
<td>993</td>
</tr>
<tr>
<td>StringScale</td>
<td>Data structure for storing string-valued dimension scale</td>
<td>994</td>
</tr>
<tr>
<td>SurfpackApproximation</td>
<td>Derived approximation class for Surfpack approximation classes. Interface between Surfpack and Dakota</td>
<td>995</td>
</tr>
<tr>
<td>SurrBasedGlobalMinimizer</td>
<td>The global surrogate-based minimizer which sequentially minimizes and updates a global surrogate model without trust region controls</td>
<td>999</td>
</tr>
<tr>
<td>SurrBasedGlobalTraits</td>
<td></td>
<td>1000</td>
</tr>
<tr>
<td>SurrBasedLocalMinimizer</td>
<td>Class for provably-convergent local surrogate-based optimization and nonlinear least squares</td>
<td>1001</td>
</tr>
<tr>
<td>SurrBasedMinimizer</td>
<td>Base class for local/global surrogate-based optimization/least squares</td>
<td>1007</td>
</tr>
<tr>
<td>Surrogate</td>
<td>Parent class for surrogate models</td>
<td>1012</td>
</tr>
<tr>
<td>SurrogateModel</td>
<td>Base class for surrogate models (DataFitSurrModel and HierarchSurrModel)</td>
<td>1015</td>
</tr>
<tr>
<td>SurrogatesGPApprox</td>
<td>Derived approximation class for Surrogates approximation classes</td>
<td>1020</td>
</tr>
<tr>
<td>SurrogatesPolyApprox</td>
<td>Derived approximation class for Surrogates Polynomial approximation classes</td>
<td>1021</td>
</tr>
<tr>
<td>SysCallApplicInterface</td>
<td>Derived application interface class which spawns simulation codes using system calls</td>
<td>1023</td>
</tr>
<tr>
<td>TabularDataTruncated</td>
<td>Exception thrown when data read truncated</td>
<td>1026</td>
</tr>
<tr>
<td>TabularReader</td>
<td>Utility used in derived read_core to read values in tabular format</td>
<td>1027</td>
</tr>
<tr>
<td>TabularWriter</td>
<td>Utility used in derived write_core to write values in tabular format</td>
<td>1027</td>
</tr>
<tr>
<td>TANA3Approximation</td>
<td>Derived approximation class for TANA-3 two-point exponential approximation (a multipoint approximation)</td>
<td>1028</td>
</tr>
<tr>
<td>TaylorApproximation</td>
<td>Derived approximation class for first- or second-order Taylor series (a local approximation)</td>
<td>1030</td>
</tr>
<tr>
<td>TestDriverInterface</td>
<td></td>
<td>1031</td>
</tr>
<tr>
<td>TKFactoryDIPC</td>
<td>Custom RW TKfactory: passes Dakota QUESO instance pointer to the TK at build</td>
<td>1038</td>
</tr>
<tr>
<td>TKFactoryDIPCLogit</td>
<td>Custom Logit RW TKfactory: passed Dakota QUESO instance pointer to the TK at build</td>
<td>1039</td>
</tr>
</tbody>
</table>
### 11.1. CLASS LIST

- **TPLDataTransfer**: 1040
- **TrackerHTTP**: a usage tracking module that uses HTTP/HTTPS via the curl library 1041
- **TraitsBase**: Base class for traits 1043
- **UsageTracker**: Lightweight class to manage conditionally active Curl-based HTTP tracker via PIMPL 1045
- **Var icheck**: Structure for verifying bounds and initial point for string-valued vars 1046
- **Var rcheck**: Structure for verifying bounds and initial point for real-valued vars 1047
- **Variables**: Base class for the variables class hierarchy 1047
- **Verification**: Base class for managing common aspects of verification studies 1057
- **VLint**: Structure for validating integer uncertain variable labels, bounds, values 1059
- **VLreal**: Structure for validating real uncertain variable labels, bounds, values 1059
- **VLstr**: Structure for validating string uncertain variable labels, bounds, values 1059
- **VPSApproximation**: Derived approximation class for VPS implementation 1060
- **WeightingModel**: Weighting specialization of RecastModel 1064
- **WorkdirHelper**: 1065
Chapter 12

File Index

12.1 File List

Here is a list of all documented files with brief descriptions:

- **dakota.dll_api.cpp**
  - This file contains a DakotaRunner class, which launches DAKOTA

- **dakota.dll_api.h**
  - API for DLL interactions

- **dakota_linear_algebra.hpp**
  - Dakota linear algebra utilities

- **dakota_tabular_io.hpp**
  - Utility functions for reading and writing tabular data files. Emerging utilities for tabular file I/O. For now, just extraction of capability from separate contexts to facilitate rework. These augment (and leverage) those in data_util.h

- **dll_tester.cpp**
  - Test the DLL with a DAKOTA input file

- **JEGAOptimizer.cpp**
  - Contains the implementation of the JEGAOptimizer class

- **JEGAOptimizer.hpp**
  - Contains the definition of the JEGAOptimizer class

- **library_mode.cpp**
  - File containing a mock simulator main for testing Dakota in library mode

- **library_split.cpp**
  - File containing a mock simulator main for testing DAKOTA in library mode on a split communicator

- **main.cpp**
  - File containing the main program for DAKOTA

- **QUESOImpl.hpp**

- **restart_util.cpp**
  - File containing the DAKOTA restart utility main program
Chapter 13

Namespace Documentation

13.1 dakota Namespace Reference

dakota (lowercase) namespace for new Dakota modules

Namespaces

- surrogates
  namespace for new Dakota surrogates module
- util
  namespace for new Dakota utilities module

Typedefs

- using VectorXd = Eigen::VectorXd
  Eigen generic vector of doubles in Dakota namespace.
- using MatrixXd = Eigen::MatrixXd
  Eigen generic matrix of doubles in Dakota namespace.
- using VectorXi = Eigen::VectorXi
  Eigen generic vector of integers in Dakota namespace.
- using MatrixXi = Eigen::MatrixXi
  Eigen generic matrix of integers in Dakota namespace.
- using Real = double
  Dakota real floating point type.
- using RealMatrix = Teuchos::SerialDenseMatrix< int, Real >
  Dakota matrix of reals.
- using RealVector = Teuchos::SerialDenseVector< int, Real >
  Dakota vector of reals.
- using ParameterList = Teuchos::ParameterList
  Teuchos ParameterList for options management in Dakota namespace.
Functions

- template<typename... Ts>
  
  void silence_unused_args (const Ts...)

  silence unused parameter warning; use to indicate those parameters are intentionally unused

13.1.1 Detailed Description

dakota (lowercase) namespace for new Dakota modules

13.2 Dakota Namespace Reference

The primary namespace for DAKOTA.

Classes

- class ActiveSubspaceModel
  
  Active subspace model for input (variable space) reduction.

- class AdaptedBasisModel
  
  Adapted basis model for input (variable space) reduction.

- class ApplicationInterface
  
  Derived class within the interface class hierarchy for supporting interfaces to simulation codes.

- class ApproximationInterface
  
  Derived class within the interface class hierarchy for supporting approximations to simulation-based results.

- class APPSEvalMgr
  
  Evaluation manager class for APPSPACK.

- class AppsTraits
  
  HOPSPACK-specific traits class.

- class APPSOptimizer
  
  Wrapper class for HOPSPACK.

- class BootstrapSamplerBase
  
  Base class/interface for the bootstrap sampler.

- class BootstrapSampler
  
  Actual bootstrap sampler implementation for common data types.

- class BootstrapSampler< Teuchos::SerialDenseMatrix< OrdinalType, ScalarType > >
  
  Bootstrap sampler that is specialized to allow for the bootstrapping of RealMatrix.

- class BootstrapSamplerWithGS
  
  A derived sampler to allow for user specification of the accessor methods.

- class C3FnTrainPtrs
  
  Handle for reference-counted pointer to C3FnTrainPtrsRep body.

- class C3Approximation
  
  Derived approximation class for global basis polynomials.

- class COLINApplication

- class COLINTraits
  
  A version of TraitsBase specialized for COLIN optimizers.
13.2. DAKOTA NAMESPACE REFERENCE

- class COLINOptimizer
  Wrapper class for optimizers defined using COLIN.
- class CollabHybridMetaIterator
  Meta-iterator for hybrid iteration using multiple collaborating optimization and nonlinear least squares methods.
- class GetLongOpt
  GetLongOpt is a general command line utility from S. Manoharan (Advanced Computer Research Institute, Lyon, France).
- class CommandLineHandler
  Utility class for managing command line inputs to DAKOTA.
- class CommandShell
  Utility class which defines convenience operators for spawning processes with system calls.
- class ConcurrentMetaIterator
  Meta-iterator for multi-start iteration or pareto set optimization.
- class CONMINTraits
  A version of TraitsBase specialized for CONMIN optimizers.
- class CONMINOptimizer
  Wrapper class for the CONMIN optimization library.
- class FileReadException
  base class for Dakota file read exceptions (to allow catching both tabular and general file truncation issues)
- class TabularDataTruncated
  exception thrown when data read truncated
- class ResultsFileError
  exception throw for other results file read error
- class FunctionEvalFailure
  exception class for function evaluation failures
- struct BaseConstructor
  Dummy struct for overloading letter-envelope constructors.
- struct NoDBBaseConstructor
  Dummy struct for overloading constructors used in on-the-fly instantiations without ProblemDescDB support.
- struct LightWtBaseConstructor
  Dummy struct for overloading constructors used in on-the-fly Model instantiations.
- class RealScale
  Data structure for storing real-valued dimension scale.
- struct IntegerScale
  Data structure for storing int-valued dimension scale.
- struct StringScale
  Data structure for storing string-valued dimension scale.
- struct ResultAttribute
  Data structure for a single Real, String, or int valued attribute.
- class ActiveSet
  Container class for active set tracking information. Contains the active set request vector and the derivative variables vector.
- class Analyzer
CHAPTER 13. NAMESPACE DOCUMENTATION

Base class for NonD, DACE, and ParamStudy branches of the iterator hierarchy.

- class Approximation
  Base class for the approximation class hierarchy.

- class Constraints
  Base class for the variable constraints class hierarchy.

- class Environment
  Base class for the environment class hierarchy.

- class Graphics
  The Graphics class provides a single interface to 2D (motif) and 3D (PLPLOT) graphics; there is only one instance of this OutputManager::dakotaGraphics.

- class Interface
  Base class for the interface class hierarchy.

- class Iterator
  Base class for the iterator class hierarchy.

- class LeastSq
  Base class for the nonlinear least squares branch of the iterator hierarchy.

- class Minimizer
  Base class for the optimizer and least squares branches of the iterator hierarchy.

- class Model
  Base class for the model class hierarchy.

- class NonD
  Base class for all nondeterministic iterators (the DAKOTA/UQ branch).

- class Optimizer
  Base class for the optimizer branch of the iterator hierarchy.

- class PStudyDACE
  Base class for managing common aspects of parameter studies and design of experiments methods.

- class Response
  Container class for response functions and their derivatives. Response provides the enveloper base class.

- class SurrogatesGPApprox
  Derived approximation class for Surrogates approximation classes.

- class SurrogatesPolyApprox
  Derived approximation class for Surrogates Polynomial approximation classes.

- class TPLDataTransfer

- class TraitsBase
  Base class for traits.

- class GeneralReader
  Utility used in derived read_core to read in generic format.

- class TabularReader
  Utility used in derived read_core to read values in tabular format.

- class GeneralWriter
  Utility used in derived write_core to write in generic format.

- class ApreproWriter
  Utility used in derived write_core to write in aprepro format.
• class TabularWriter
  Utility used in derived write_core to write values in tabular format.
• class LabelsWriter
  Utility used in derived write_core to write labels in tabular format.
• class Variables
  Base class for the variables class hierarchy.
• class Verification
  Base class for managing common aspects of verification studies.
• class DataEnvironmentRep
  Body class for environment specification data.
• class DataEnvironment
  Handle class for environment specification data.
• class DataFitSurrBasedLocalTraits
  Class for provably-convergent local surrogate-based optimization and nonlinear least squares.
• class DataFitSurrModel
  Derived model class within the surrogate model branch for managing data fit surrogates (global and local)
• class DataInterface
  Handle class for interface specification data.
• class DataMethodRep
  Body class for method specification data.
• class DataMethod
  Handle class for method specification data.
• class DataModelRep
  Body class for model specification data.
• class DataModel
  Handle class for model specification data.
• class DataResponsesRep
  Body class for responses specification data.
• class DataResponses
  Handle class for responses specification data.
• class DataTransformModel
  Data transformation specialization of RecastModel.
• class DataVariablesRep
  Body class for variables specification data.
• class DataVariables
  Handle class for variables specification data.
• class DDACEDesignCompExp
  Wrapper class for the DDACE design of experiments library.
• class DirectApplicInterface
  Derived application interface class which spawns simulation codes and testers using direct procedure calls.
• class DiscrepancyCorrection
  Base class for discrepancy corrections.
• class DLSolverTraits
A version of TraitsBase specialized for DLSolver.

- **class DOTTraits**
  Wrapper class for the DOT optimization library.

- **class EffGlobalTraits**
  Implementation of Efficient Global Optimization/Least Squares algorithms.

- **class EffGlobalMinimizer**

- **class EmbedHybridMetaIterator**
  Meta-iterator for closely-coupled hybrid iteration, typically involving the embedding of local search methods within global search methods.

- **class ExecutableEnvironment**
  Environment corresponding to execution as a stand-alone application.

- **class ExperimentData**
  Interpolation method for interpolating between experimental and model data. I need to work on inputs/outputs to this method. For now, this assumes interpolation of functional data.

- **class ExperimentResponse**
  Container class for response functions and their derivatives. ExperimentResponse provides the body class.

- **class ForkApplicInterface**
  Derived application interface class which spawns simulation codes using fork/execvp/waitpid.

- **class FSUDesignCompExp**
  Wrapper class for the FSUDace QMC/CVT library.

- **class GaussProcApproximation**
  Derived approximation class for Gaussian Process implementation.

- **class GridApplicInterface**
  Derived application interface class which spawns simulation codes using grid services such as Condor or Globus.

- **class HDF5IOHelper**

- **class HierarchSurrBasedLocalTraits**
  Class for multilevel-multifidelity optimization algorithm.

- **class HierarchSurrModel**
  Derived model class within the surrogate model branch for managing hierarchical surrogates (models of varying fidelity).

- **class IteratorScheduler**
  This class encapsulates scheduling operations for concurrent sub-iteration within an outer level context (e.g., meta-iteration, nested models).

- **class JEGAOptimizer**
  A version of Dakota::Optimizer for instantiation of John Eddy’s Genetic Algorithms (JEGA).

- **class JEGATraits**
  A version of TraitsBase specialized for John Eddy’s Genetic Algorithms (JEGA).

- **class LibraryEnvironment**
  Environment corresponding to execution as an embedded library.

- **class MatlabInterface**

- **class MetaIterator**
  Base class for meta-iterators.

- **class MixedVarConstraints**
Derived class within the Constraints hierarchy which separates continuous and discrete variables (no domain type array merging).

- **class MixedVariables**
  Derived class within the Variables hierarchy which separates continuous and discrete variables (no domain type array merging).

- **class MPIManager**
  Class MPIManager to manage Dakota’s MPI world, which may be a subset of MPI_COMM_WORLD.

- **class MPIPackBuffer**
  Class for packing MPI message buffers.

- **class MPIUnpackBuffer**
  Class for unpacking MPI message buffers.

- **class NCSUTraits**
  A version of TraitsBase specialized for NCSU optimizers.

- **class NCSUOptimizer**
  Wrapper class for the NCSU DIRECT optimization library.

- **class NestedModel**
  Derived model class which performs a complete sub-iterator execution within every evaluation of the model.

- **struct Var_rcheck**
  Structure for verifying bounds and initial point for real-valued vars

- **struct Var_icheck**
  Structure for verifying bounds and initial point for string-valued vars

- **struct VLreal**
  Structure for validating real uncertain variable labels, bounds, values

- **struct VLint**
  Structure for validating integer uncertain variable labels, bounds, values

- **struct VLstr**
  Structure for validating string uncertain variable labels, bounds, values

- **class NIDRProblemDescDB**
  The derived input file database utilizing the new IDR parser.

- **struct NL2Res**
  Auxiliary information passed to calcr and calcj via ur.

- **class NL2SOLLeastSqTraits**
  A version of TraitsBase specialized for NL2SOL nonlinear least squares library.

- **class NL2SOLLeastSq**
  Wrapper class for the NL2SOL nonlinear least squares library.

- **class NLPQLP Traits**
  Wrapper class for the NLPQLP optimization library, Version 2.0.

- **class NLSSOLLeastSqTraits**
  A version of TraitsBase specialized for NLSSOL nonlinear least squares library.

- **class NLSSOLLeastSq**
  Wrapper class for the NLSSOL nonlinear least squares library.

- **class NomadTraits**
  Wrapper class for NOMAD Optimizer.
• class NonDAdaptImpSampling
  Class for the Adaptive Importance Sampling methods within DAKOTA.

• class NonDAbsImpSampling
  Class for testing various Adaptively sampling methods using geometric, statistical, and topological information of the surrogate.

• class NonDBayesCalibration
  Base class for Bayesian inference: generates posterior distribution on model parameters given experimental data.

• class NonDC3FunctionTrain
  Nonintrusive uncertainty quantification with the C3 library ...

• class NonDCalibration

• class NonDCubature
  Derived nondeterministic class that generates N-dimensional numerical cubature points for evaluation of expectation integrals.

• class NonDDREAMBayesCalibration
  Bayesian inference using the DREAM approach.

• class NonDExpansion
  Base class for polynomial chaos expansions (PCE), stochastic collocation (SC) and functional tensor train (FT)

• class NonDGlobalEvidence
  Class for the Dempster-Shafer Evidence Theory methods within DAKOTA/UQ.

• class NonDGlobalInterval
  Class for using global nongradient-based optimization approaches to calculate interval bounds for epistemic uncertainty quantification.

• class NonDGlobalSingleInterval
  Class for using global nongradient-based optimization approaches to calculate interval bounds for epistemic uncertainty quantification.

• class NonDGPImpSampling
  Class for the Gaussian Process-based Importance Sampling method.

• class NonDGPMSABayesCalibration
  Generates posterior distribution on model parameters given experiment data.

• class NonDIIntegration
  Derived nondeterministic class that generates N-dimensional numerical integration points for evaluation of expectation integrals.

• class NonDInterval
  Base class for interval-based methods within DAKOTA/UQ.

• class NonDLHSEvidence
  Class for the Dempster-Shafer Evidence Theory methods within DAKOTA/UQ.

• class NonDLHSInterval
  Class for the LHS-based interval methods within DAKOTA/UQ.

• class NonDLHSSampling
  Performs LHS and Monte Carlo sampling for uncertainty quantification.

• class NonDLHSSingleInterval
  Class for pure interval propagation using LHS.
13.2. DAKOTA NAMESPACE REFERENCE

- class NonDLocalEvidence
  Class for the Dempster-Shafer Evidence Theory methods within DAKOTA/UQ.

- class NonDLocalInterval
  Class for using local gradient-based optimization approaches to calculate interval bounds for epistemic uncertainty quantification.

- class NonDLocalReliability
  Class for the reliability methods within DAKOTA/UQ.

- class NonDLocalSingleInterval
  Class for using local gradient-based optimization approaches to calculate interval bounds for epistemic uncertainty quantification.

- class NonDMultilevelFunctionTrain
  Nonintrusive polynomial chaos expansion approaches to uncertainty quantification.

- class NonDMultilevelPolynomialChaos
  Nonintrusive polynomial chaos expansion approaches to uncertainty quantification.

- class NonDMultilevelSampling
  Performs Multilevel Monte Carlo sampling for uncertainty quantification.

- class NonDMultilevelStochCollocation
  Nonintrusive stochastic collocation approaches to uncertainty quantification.

- class NonDMUQBayesCalibration
  Dakota interface to MUQ (MIT Uncertainty Quantification) library.

- class NonDPOFDarts
  Base class for POF Dart methods within DAKOTA/UQ.

- class NonDPolynomialChaos
  Nonintrusive polynomial chaos expansion approaches to uncertainty quantification.

- class NonDQuadrature
  Derived nondeterministic class that generates N-dimensional numerical quadrature points for evaluation of expectation integrals over uncorrelated standard normals/uniforms/exponentials/betas/gammas.

- class DerivInformedPropCovTK
  Dakota transition kernel that updates proposal covariance based on derivatives (for random walk case)

- class DerivInformedPropCovLogitTK
  Dakota transition kernel that updates proposal covariance based on derivatives (for logit random walk case)

- class NonDQUESThesisBayesCalibration
  Bayesian inference using the QUESO library from UT Austin.

- class NonDReliability
  Base class for the reliability methods within DAKOTA/UQ.

- class NonDRKDDarts
  Base class for the Recursive k-d Dart methods within DAKOTA/UQ.

- class NonDSampling
  Base class for common code between NonDLHSSampling, NonDAdaptImpSampling, and other specializations.

- class NonDSparseGrid
  Derived nondeterministic class that generates N-dimensional Smolyak sparse grids for numerical evaluation of expectation integrals over independent standard random variables.

- class NonDStochCollocation
  Nonintrusive stochastic collocation approaches to uncertainty quantification.
• class **NonDSurrogateExpansion**
  Generic uncertainty quantification with Model-based stochastic expansions.
• class **NonDWSABIBayesCalibration**
  WASABI - Weighted Adaptive Surrogate Approximations for Bayesian Inference.
• class **NonlinearCGTraits**
  A version of TraitsBase specialized for NonlinearCG optimizers.
• class **NonlinearCGOptimizer**
• class **NOWPACBlackBoxEvaluator**
  Derived class for plugging Dakota evaluations into NOWPAC solver.
• class **NOWPACTraits**
  A version of TraitsBase specialized for NOWPAC optimizers.
• class **NOWPACOptimizer**
  Wrapper class for the (S)NOWPAC optimization algorithms from Florian Augustin (MIT)
• class **NPSOLTraits**
  Wrapper class for the NPSOL optimization library.
• class **OptDartsTraits**
  A version of TraitsBase specialized for OptDarts.
• class **OptDartsOptimizer**
  Wrapper class for OptDarts Optimizer.
• class **OutputWriter**
• class **ConsoleRedirector**
• class **RestartWriter**
• class **OutputManager**
  Class to manage redirection of stdout/stderr, keep track of current redir state, and manage rank 0 output. Also manage tabular data output for post-processing with Matlab, Tecplot, etc. and delegate to Graphics for X Windows Graphics.
• class **ParallelLevel**
  Container class for the data associated with a single level of communicator partitioning.
• class **ParallelConfiguration**
  Container class for a set of ParallelLevel list iterators that collectively identify a particular multilevel parallel configuration.
• class **ParallelLibrary**
  Class for partitioning multiple levels of parallelism and managing message passing within these levels.
• class **ParamResponsePair**
  Container class for a variables object, a response object, and an evaluation id.
• class **ParamStudy**
  Class for vector, list, centered, and multidimensional parameter studies.
• class **PebbldBranching**
  Main Branching class for the PEBBL-based Minimizer.
• class **PebbldBranchSub**
  Sub Branch class for the PEBBL-based Minimizer.
• class **PebbldTraits**
  Wrapper class for experimental PebbldMinimizer.
• class **PecosApproximation**
13.2. DAKOTA NAMESPACE REFERENCE

- Derived approximation class for global basis polynomials.
  - class ProbabilityTransformModel
    Probability transformation specialization of RecastModel.
  - class ProblemDescDB
    The database containing information parsed from the DAKOTA input file.
  - class ProcessApplicInterface
    Derived application interface class that spawns a simulation code using a separate process and communicates with it through files.
  - class ProcessHandleApplicInterface
    Derived application interface class that spawns a simulation code using a separate process, receives a process identifier, and communicates with the spawned process through files.
  - class ProgramOptions
    ProgramOptions stores options whether from the CLH or from library user; initially valid only on worldRank = 0, but then broadcast in ParallelLibrary::push_output_tag()
  - struct partial_prp_hash
    wrapper to delegate to the ParamResponsePair hash_value function
  - struct partial_prp_equality
    predicate for comparing ONLY the interfaceId and Vars attributes of PRPair
  - class PSUADEDesignCompExp
    Wrapper class for the PSUADE library.
  - class PythonInterface
  - class QMEApproximation
    Derived approximation class for QMEA Quadratic Multipoint Exponential Approximation (a multipoint approximation).
  - class QesoJointPdf
    Dakota specialization of QUESO generic joint PDF.
  - class QuesoVectorRV
    Dakota specialization of QUESO vector-valued random variable.
  - class TKFactoryDIPC
    Custom RW TKfactory: passes Dakota QUESO instance pointer to the TK at build.
  - class TKFactoryDIPCLogit
    Custom Logit RW TKfactory: passed Dakota QUESO instance pointer to the TK at build.
  - class RandomFieldModel
    Random field model, capable of generating and then forward propagating.
  - class RecastModel
    Derived model class which provides a thin wrapper around a sub-model in order to recast the form of its inputs and/or outputs.
  - class ReducedBasis
  - class RelaxedVarConstraints
    Derived class within the Constraints hierarchy which employs relaxation of discrete variables.
  - class RelaxedVariables
    Derived class within the Variables hierarchy which employs the relaxation of discrete variables.
  - class ResultsDBAny
  - class ResultsDBBase
• class **AddAttributeVisitor**
  
  Objects of this class are called by `boost::apply_visitor` to add attributes to HDF5 objects.

• class **AttachScaleVisitor**
  
  Objects of this class are called by `boost::apply_visitor` to add dimension scales (`RealScale` or `StringScale`) to HDF5 datasets.

• class **ResultsDBHDF5**
  
  Manage interactions between `ResultsManager` and the low-level HDFIOHelper class.

• class **ResultsNames**
  
  List of valid names for iterator results.

• class **ResultsManager**
  
  Results manager for iterator final data.

• class **ResultsEntry**
  
  Class to manage in-core vs. file database lookups.

• class **RichExtrapVerification**
  
  Class for Richardson extrapolation for code and solution verification.

• class **ROLOptimizer**

• class **ROLTraits**

• class **DakotaROLObjective**

• class **DakotaROLObjectiveGrad**

• class **DakotaROLObjectiveHess**

• class **DakotaROLIneqConstraints**

• class **DakotaROLIneqConstraintsGrad**

• class **DakotaROLIneqConstraintsHess**

• class **DakotaROLEqConstraints**

• class **DakotaROLEqConstraintsGrad**

• class **DakotaROLEqConstraintsHess**

• class **PrefixingLineFilter**

• class **ScalingModel**
  
  Scaling specialization of `RecastModel`.

• class **ScalingOptions**
  
  Simple container for user-provided scaling data, possibly expanded by replicates through the models.

• class **ScilabInterface**

• class **SensAnalysisGlobal**
  
  Class for a utility class containing correlation calculations and variance-based decomposition.

• class **SeqHybridMetaIterator**
  
  Method for sequential hybrid iteration using multiple optimization and nonlinear least squares methods on multiple models of varying fidelity.

• class **SharedApproxData**
  
  Base class for the shared approximation data class hierarchy.

• class **SharedC3ApproxData**
  
  Derived approximation class for global basis polynomials.

• class **SharedPecosApproxData**
  
  Derived approximation class for global basis polynomials.

• class **SharedResponseDataRep**
The representation of a `SharedResponseData` instance. This representation, or body, may be shared by multiple `SharedResponseData` handle instances.

- **class** `SharedResponseData`
  Container class encapsulating variables data that can be shared among a set of `Response` instances.

- **class** `SharedSurfpackApproxData`
  Derived approximation class for Surfpack approximation classes. Interface between Surfpack and Dakota.

- **class** `SharedVariablesDataRep`
  The representation of a `SharedVariablesData` instance. This representation, or body, may be shared by multiple `SharedVariablesData` handle instances.

- **class** `SharedVariablesData`
  Container class encapsulating variables data that can be shared among a set of `Variables` instances.

- **class** `SimulationModel`
  Derived model class which utilizes a simulation-based application interface to map variables into responses.

- **class** `SimulationResponse`
  Container class for response functions and their derivatives. `SimulationResponse` provides the body class.

- **class** `SNLLBase`
  Base class for OPT++ optimization and least squares methods.

- **class** `SNLLLLeastSqTraits`
  A version of `TraitsBase` specialized for `SNLLLLeastSq`.

- **class** `SNLLLLeastSq`
  Wrapper class for the OPT++ optimization library.

- **class** `SNLLTraits`
  A version of `TraitsBase` specialized for SNLL optimizers.

- **class** `SNLLOptimizer`
  Wrapper class for the OPT++ optimization library.

- **class** `SOLBase`
  Base class for Stanford SOL software.

- **class** `SpawnApplicInterface`
  Derived application interface class which spawns simulation codes using spawnvp.

- **class** `SurfpackApproximation`
  Derived approximation class for Surfpack approximation classes. Interface between Surfpack and Dakota.

- **class** `SurrBasedGlobalTraits`
  The global surrogate-based minimizer which sequentially minimizes and updates a global surrogate model without trust region controls.

- **class** `SurrBasedGlobalMinimizer`

- **class** `SurrBasedLocalMinimizer`
  Class for provably-convergent local surrogate-based optimization and nonlinear least squares.

- **class** `SurrBasedMinimizer`
  Base class for local/global surrogate-based optimization/least squares.

- **class** `SurrogateModel`
  Base class for surrogate models (`DataFitSurrModel` and `HierarchSurrModel`).

- **class** `SysCallApplicInterface`
  Derived application interface class which spawns simulation codes using system calls.

- **class** `TANA3Approximation`
Derived approximation class for TANA-3 two-point exponential approximation (a multipoint approximation).

- **class TaylorApproximation**
  Derived approximation class for first- or second-order Taylor series (a local approximation).

- **class TestDriverInterface**

- **class TrackerHTTP**
  *TrackerHTTP*: a usage tracking module that uses HTTP/HTTPS via the curl library.

- **class UsageTracker**
  Lightweight class to manage conditionally active Curl-based HTTP tracker via PIMPL.

- **class VPSApproximation**
  Derived approximation class for VPS implementation.

- **class WeightingModel**
  Weighting specialization of *RecastModel*.

- **struct MatchesWC**
  *Predicate that returns true when the passed path matches the wildcard with which it was configured. Currently supports * and ?.*

- **class WorkdirHelper**

### Typedefs

- typedef boost::tuple
  
  < std::string, std::string, size_t, std::string > *ResultsKeyType*

  *Data type for results key (instance name / id, unique run, label), where data_key is a valid colon-delimited string from *ResultsNames* tuple < method_name, method_id, execution_number, data_key >*

- typedef std::string *MetaDataKeyType*

  *Data type for metadata key.*

- typedef std::vector< std::string > *MetaDataValueType*

  *Data type for metadata value.*

- typedef std::map<MetaDataKeyType,MetaDataValueType> *MetaDataType*

  *A single MetaData entry is map<string, vector<string>>. Example: pair( "Column labels", ["Mean", "Std Dev", "Skewness", "Kurtosis"] )*

- typedef boost::tuple
  
  < std::string, std::string, size_t > *StrStrSizet*

  *Iterator unique ID: <method_name, method_id, exec_num>*

- typedef std::multimap<int, boost::variant<StringScale, RealScale, IntegerScale>> *DimScaleMap*

  *Datatype to communicate scales (stored in boost::variant) and their associated dimension (the int) to the *Results-Mananger* instance.*

- typedef std::vector< boost::variant<ResultAttribute<int>, ResultAttribute<String>, ResultAttribute<Real>>> *AttributeArray*
13.2. DAKOTA NAMESPACE REFERENCE

Datatype to communicate metadata (attributes) to the ResultsManager instance.

- typedef boost::bimap<unsigned short, std::string> UShortStrBimap
  
  bimaps to convert from enums <--> strings

- typedef void(* dl_core_run_t )(void *, Optimizer1 *, char *)
- typedef void(* dl_destructor_t )(void **)
- typedef Teuchos::SerialDenseSolver <int, Real> RealSolver
- typedef Teuchos::SerialSpdDenseSolver <int, Real> RealSpdSolver
- typedef int(* start_grid_computing_t )(char *analysis_driver_script, char *params_file, char *results_file)
  
  definition of start grid computing type (function pointer)
- typedef int(* perform_analysis_t )(char *iteration_num)
  
  definition of perform analysis type (function pointer)
- typedef int(* get_jobs_completed_t )()
  
  definition of get completed jobs type (function pointer)
- typedef int(* stop_grid_computing_t )()
  
  definition of stop grid computing type (function pointer)
- typedef int MPI_Comm
- typedef void * MPI_Request
- typedef unsigned char u_char
- typedef unsigned short u_short
- typedef unsigned int u_int
- typedef unsigned long u_long
- typedef long long long_long
- typedef unsigned long UL
- typedef void(* Calcrj )(int *n, int *p, Real *x, int *nf, Real *r, int *ui, void *ur, Vf vf)
- typedef void(* Vf )()
- typedef void(* DbCallbackFunctionPtr )(Dakota::ProblemDescDB *db, void *data_ptr)
- typedef boost::tuple
  
  Triplet of filesystem paths: e.g., params, results, workdir.

- typedef
bmi::multi_index_container
< Dakota::ParamResponsePair,
bmi::indexed_by
< bmi::ordered_non_unique
< bmi::tag< ordered >
, bmi::const_mem_fun
< Dakota::ParamResponsePair,
const IntStringPair
>&,&Dakota::ParamResponsePair::eval_interface_ids >
>, bmi::hashed_non_unique
< bmi::tag< hashed >
, bmi::identity
< Dakota::ParamResponsePair >
, partial_prp_hash,
partial_prp_equality > > > PRPMultiIndexCache

Boost Multi-Index Container for globally caching ParamResponsePairs.

- typedef PRPMultiIndexCache PRPCache
- typedef PRPCache::index_iterator
  < ordered >::type PRPCacheOIter
- typedef PRPCache::index_const_iterator
  < ordered >::type PRPCacheOCIter
- typedef PRPCache::index_iterator
  < hashed >::type PRPCacheHIter
- typedef PRPCache::index_const_iterator
  < hashed >::type PRPCacheHCIter
- typedef PRPCacheOIter PRPCachelter
  
  default cache iterator <0>
  
- typedef PRPCacheOIter PRPCacheClter
  
  default cache const reverse iterator <0>
  
- typedef
  boost::reverse_iterator
  < PRPCacheClter > PRPCacheCRevIter
- typedef
13.2. _DAKOTA NAMESPACE REFERENCE_ 103

```cpp
bmi::multi_index_container
< Dakota::ParamResponsePair,
bmi::indexed_by
< bmi::ordered_unique
< bmi::tag< ordered >
, bmi::const_mem_fun
< Dakota::ParamResponsePair,
int,&Dakota::ParamResponsePair::eval_id >
>, bmi::hashed_non_unique
< bmi::tag< hashed >
, bmi::identity
< Dakota::ParamResponsePair >
, partial_prp_hash,
partial_prp_equality > > > PRPMultiIndexQueue

_Boost Multi-Index Container for locally queueing ParamResponsePairs._

- typedef PRPMultiIndexQueue **PRPQueue**
- typedef PRPQueue::index_iterator
  < ordered >::type **PRPQueueOlIter**
- typedef PRPQueue::index_const_iterator
  < ordered >::type **PRPQueueOCIter**
- typedef PRPQueue::index_iterator
  < hashed >::type **PRPQueueHIter**
- typedef PRPQueue::index_const_iterator
  < hashed >::type **PRPQueueHCIter**
- typedef std::pair< boost::any,
  MetaDataType > **ResultsValueType**

  _Core data storage type: boost::any, with optional metadata (see other types in results_types.hpp)_

- typedef boost::function< bool(const
  bfs::path &src_path, const
  bfs::path &dest_path, bool
  overwrite)> **file_op_function**

  _define a function type that operates from src to dest, with option to overwrite_

- typedef boost::filter_iterator
  < MatchesWC,
  bfs::directory_iterator > **glob_iterator**

  _a glob_iterator filters a directory_iterator based on a wildcard predicate_

**Enumerations**

- enum {
  _COBYLA, DIRECT, EA, MS, PS, SW, BETA_ }
• enum {
  VARS_ERROR = -10, RESP_ERROR = -9, APPROX_ERROR = -8, METHOD_ERROR = -7,
  MODEL_ERROR = -6, IO_ERROR = -5, INTERFACE_ERROR = -4, CONSTRUCT_ERROR = -3,
  PARSE_ERROR = -2, OTHER_ERROR = -1 }
  
  enum for Dakota abort reasons; using negative numbers to distinguish Dakota exit states from signals / uncaught signals. These need to be in range [-63, -1], so exit code (256+enum) is in [193, 255]. See RATIONALE in dakota_global_defs.cpp.

• enum { MODEL_EVAL_STORE_TOP_METHOD = 0, MODEL_EVAL_STORE_NONE, MODEL_EVAL_STORE_ALL, MODEL_EVAL_STORE_ALL_METHODS }
  
  enum for selecting the models that store evaluations

• enum { INTERF_EVAL_STORE_SIMULATION = 0, INTERF_EVAL_STORE_NONE, INTERF_EVAL_STORE_ALL }
  
  enum for selecting the interfaces that store evaluations

• enum { ABORT_EXITS, ABORT_THROWS }
  
  enum for dakota abort behaviors

• enum { CV_ID_DEFAULT = 0, MINIMUM_METRIC, RELATIVE_TOLERANCE, DECREASE_TOLERANCE }
  
  enum for active subspace cross validation identification

• enum {
  TABULAR_NONE = 0, TABULAR_HEADER = 1, TABULAR_EVAL_ID = 2, TABULAR_IFACE_ID = 4,
  TABULAR_EXPER_ANNOT = TABULAR_HEADER | TABULAR_EVAL_ID, TABULAR_ANNOTATED = TABULAR_HEADER | TABULAR_EVAL_ID | TABULAR_IFACE_ID }
  
  options for tabular columns

• enum { RESULTS_OUTPUT_TEXT = 1, RESULTS_OUTPUT_HDF5 = 2 }
  
  Results output format.

• enum { FLEXIBLE_RESULTS, LABELED_RESULTS }
  
  options for results file format

• enum {
  NO_MODEL_FORMAT =0, TEXT_ARCHIVE =1, BINARY_ARCHIVE =2, ALGEBRAIC_FILE =4, ALGEBRAIC_CONSOLE =8 }
  
  define special values for surrogateExportFormats

• enum ScaleScope { SHARED, UNSHARED }
  
  Enum to specify whether a scale shared among responses.

• enum ResultsOutputType { REAL, INTEGER, UINTeger, STRING }
  
  enum for setting type on allotted matrix for Results Output

• enum CONSTRAINT_TYPE { LINEAR, NONLINEAR }
• enum CONSTRAINT_EQUALITY_TYPE { EQUALITY, INEQUALITY }
• enum LINEAR_INEQUALITY_FORMAT { NONE, TWO_SIDED, ONE_SIDED_LOWER, ONE_SIDED_UPPER }
• enum NONLINEAR_EQUALITY_FORMAT { NONE, TRUE_EQUALITY, TWO_INEQUALITY }
• enum NONLINEAR_INEQUALITY_FORMAT { NONE, ONE_SIDED_UPPER, ONE_SIDED_LOWER, TWO_SIDED }
13.2. DAKOTA NAMESPACE REFERENCE

* enum {
  DEFAULT_INTERFACE =0, APPROX_INTERFACE, FORK_INTERFACE =PROCESS_INTERFACE_BIT, SYSTEM_INTERFACE,
  GRID_INTERFACE, TEST_INTERFACE =DIRECT_INTERFACE_BIT, MATLAB_INTERFACE, PYTHON_INTERFACE,
  SCILAB_INTERFACE }
  special values for interface type

* enum { SYNCHRONOUS_INTERFACE, ASYNCHRONOUS_INTERFACE }
  interface synchronization types

* enum { OBJECTIVE, INEQUALITY_CONSTRAINT, EQUALITY_CONSTRAINT }
  define algebraic function types

* enum {
  DEFAULT_METHOD =0, HYBRID =(META_BIT | PARALLEL_BIT), PARETO_SET, MULTI_START,
  RICHARDSON_EXTRAP =(ANALYZER_BIT | VERIF_BIT), CENTERED_PARAMETER_STUDY
  =(ANALYZER_BIT | PSTUDY_DACE_BIT), LIST_PARAMETER_STUDY, MULTIDIM PARAMETER_STUDY,
  VECTOR_PARAMETER_STUDY, DACE, FSU_CVT, FSU_HALTON,
  FSU_HAMMERSLEY, PSUADE_MOAT, LOCAL_RELIABILITY
  =(ANALYZER_BIT | NOND_BIT), GLOBAL_RELIABILITY,
  SURROGATE_BASED_UQ, POLYNOMIAL_CHAOS, MULTILEVEL_POLYNOMIAL_CHAOS,
  MULTIFIDELITY_POLYNOMIAL_CHAOS,
  STOCH_Collocation, MULTIFIDELITY_STOCH_Collocation, C3_FUNCTION_TRAIN,
  MULTILEVEL_FUNCTION_TRAIN,
  MULTIFIDELITY_FUNCTION_TRAIN, CUBATURE_INTEGRATION, SPARSE_GRID_INTEGRATION,
  BAYES_CALIBRATION, GPAIS, PO_MET_DARTS, RKD_DARTS,
  IMPORTANCE_SAMPLING, ADAPTIVE_SAMPLING, MULTILEVEL_SAMPLING, LIST_SAMPLING,
  RANDOM_SAMPLING, MUQ_SAMPLING, LOCAL_INTERVAL_EST, LOCAL_EVIDENCE,
  GLOBAL_INTERVAL_EST, GLOBAL_EVIDENCE, SURROGATE_BASED_LOCAL
  =(MINIMIZER_BIT | SURRBASED_BIT), DATA_FIT_SURROGATE_BASED_LOCAL,
  HIERARCH_SURROGATE_BASED_LOCAL, SURROGATE_BASED_GLOBAL, EFFICIENTLOBAL,
  NL2SOL =(MINIMIZER_BIT | LEASTSQ_BIT), NLSSOL_SQP, OPTPP_G_NEWTON,
  ASYNCH_PATTERN_SEARCH
  =(MINIMIZER_BIT | OPTIMIZER_BIT), OPTPP_PDS,
  COLINY_BETA, COLINY_COBYLA, COLINY_DIRECT, COLINY_MULTI_START,
  COLINY_EA, COLINY_PATTERN_SEARCH, COLINY_SOLIS_WETS, MOGA,
  SOGA, NCSU_DIRECT, MESH_ADAPTIVE_SEARCH, MIT_NOWPAC,
  MIT_SNOWPAC, GENIE_OPT_DARTS, GENIE_DIRECT, DEMO_TPL,
  NONLINEAR_CG, OPTPP_CG, OPTPP_Q_NEWTON, OPTPP_FD_NEWTON,
  OPTPP_NEWTON, NPSOL_SQP, NLPQL_SQP, DOT_BFGS,
  DOT_FRCG, DOT_MMFD, DOT_SLPS, DOT_SQP,
  CONMIN_FRCG, CONMIN_MFD, ROL, DL_SOLVER,
  BRANCH_AND_BOUND
  =(MINIMIZER_BIT | OPTIMIZER_BIT | LEASTSQ_BIT) }

* enum {
  SUBMETHOD_DEFAULT =0, SUBMETHOD_NONE, SUBMETHOD_COLLABORATIVE, SUBMETHODMBEDDED,
  SUBMETHOD_SEQUENTIAL, SUBMETHOD_LHS, SUBMETHOD_RANDOM, SUBMETHOD_-_
BOX_BEHNKEN,
SUBMETHOD_CENTRAL_COMPOSITE, SUBMETHOD_GRID, SUBMETHOD_OA_LHS, SUB-
METHOD_OAS,
SUBMETHOD_DREAM, SUBMETHOD_GPMSA, SUBMETHOD_MUQ, SUBMETHOD_QUESO,
SUBMETHOD_WASABI, SUBMETHOD_NIP, SUBMETHOD_SQP, SUBMETHOD_EA,
SUBMETHOD_EGO, SUBMETHOD_SBO, SUBMETHOD_CONVERGE_ORDER, SUBMETHOD-
CONVERGE_QOI,
SUBMETHOD_ESTIMATE_ORDER }

Sub-methods, including sampling, inference algorithm, opt algorithm types.
• enum { SILENT_OUTPUT, QUIET_OUTPUT, NORMAL_OUTPUT, VERBOSE_OUTPUT,
DEBUG_OUTPUT }
• enum { NO_RESULTS =0, REFINEMENT_RESULTS, INTERMEDIATE_RESULTS, FINAL_RES-
ULTS }
• enum { DEFAULT_SCHEDULING, MASTER_SCHEDULING, PEER_SCHEDULING, PEER_DYNAMIC-
_SCHEDULING,
PEER_STATIC_SCHEDULING, DYNAMIC_SCHEDULING, STATIC_SCHEDULING }
• enum { DEFAULT_CONFIG, PUSH_DOWN, PUSH_UP }
• enum {
STD_NORMAL_U, STD_UNIFORM_U, PARTIAL_ASKEY_U, ASKEY_U,
EXTENDED_U }
• enum { DEFAULT_COVARIANCE, NO_COVARIANCE, DIAGONAL_COVARIANCE, FULL_C-
OVARIANCE }
• enum { NO_INT_REFINE =0, IS, AIS, MMAIS }
• enum { PROBABILITIES, RELIABILITIES, GEN_RELIABILITIES }
• enum { COMPONENT =0, SYSTEM_SERIES, SYSTEM_PARALLEL }
• enum { CUMULATIVE, COMPLEMENTARY }
• enum { NO_MOMENTS =0, STANDARD_MOMENTS, CENTRAL_MOMENTS }
• enum { DEFAULT_LS =0, SVD_LS, EQ_CON_LS }
• enum {
DEFAULT_MLMF_CONTROL =0, ESTIMATOR_VARIANCE, RIP_SAMPLING, RANK_SAMP-
LING,
GREEDY_REFINEMENT }
• enum { DEFAULT_EMULATION, DISTINCT_EMULATION, RECURSIVE_EMULATION }
• enum {
NO_EMULATOR, PCE_EMULATOR, ML_PCE_EMULATOR, MF_PCE_EMULATOR,
SC_EMULATOR, MF_SC_EMULATOR, GP_EMULATOR, KRIGING_EMULATOR,
VPS_EMULATOR }
• enum {
CALIBRATE_NONE = 0, CALIBRATE_ONE, CALIBRATE_PER_EXPER, CALIBRATE_PER_R-
ESP,
CALIBRATE_BOTH }
• enum { IGNORE_RANKS, SET_RANKS, GET_RANKS, SET_GET_RANKS }
• enum {
DESIGN, UNCERTAIN, UNCERTAIN_UNIFORM, ALEATORY_UNCERTAIN,
ALEATORY_UNCERTAIN_UNIFORM, EPISODE_UNCERTAIN, EPISODE_UNCERTAIN-
_UNIFORM, STATE,
ACTIVE, ACTIVE_UNIFORM, ALL, ALL_UNIFORM }
• enum {
    ONE_SIDED_LOWER, ONE_SIDED_LOWER, ONE_SIDED_LOWER, ONE_SIDED_UPPER,
    ONE_SIDED_UPPER, ONE_SIDED_UPPER, TWO_SIDED, TWO_SIDED,
    TWO_SIDED
}
• enum { QOI_AGGREGATION_MAX, QOI_AGGREGATION_SUM }
• enum { TARGET_MEAN, TARGET_VARIANCE }
• enum {
    MV =0, AMV_X, AMV_U, AMV_PLUS_X,
    AMV_PLUS_U, TANA_X, TANA_U, QMEA_X,
    QMEA_U, NO_APPROX, EGRA_X, EGRA_U
}
• enum { BREITUNG, HOHENRACK, HONG }
• enum { ORIGINAL_PRIMARY, SINGLE_OBJECTIVE, LAGRANGIAN_OBJECTIVE, AUGMENTED_LAGRANGIAN_OBJECTIVE }
• enum { NO_CONSTRAINTS, LINEARIZED_CONSTRAINTS, ORIGINAL_CONSTRAINTS }
• enum { PENALTY_MERIT, ADAPTIVE_PENALTY_MERIT, LAGRANGIAN_MERIT, AUGMENTED_LAGRANGIAN_MERIT }
• enum { FILTER, TR_RATIO }
• enum { DEFAULT_POINTS, MINIMUM_POINTS, RECOMMENDED_POINTS, TOTAL_POINTS }

    define special values for pointsManagement

• enum {
    NO_SURROGATE =0, UNCORRECTED_SURROGATE, AUTO_CORRECTED_SURROGATE, BYPASS_SURROGATE,
    MODEL_DISCREPANCY, AGGREGATED_MODELS
}

    define special values for SurrogateModel::responseMode
• enum { NO_CORRECTION =0, ADDITIVE_CORRECTION, MULTIPLICATIVE_CORRECTION,
    COMBINED_CORRECTION }

    define special values for approxCorrectionType
• enum { RF_KARHUNEN_LOEVE =0, RF_PCA_GP, RF_ICA }

    define types of random field approximations
• enum { NOCOVAR =0, EXP_L2, EXP_L1 }

    define types of analytic covariance functions
• enum { SUBSPACE_NORM_DEFAULT =0, SUBSPACE_NORM_MEAN_VALUE, SUBSPACE_NORM_MEAN_GRAD,
    SUBSPACE_NORM_LOCAL_GRAD }

    define special values for active subspace normalizations
• enum {
    NO_PARALLEL_MODE =0, SURROGATE_MODEL_MODE, TRUTH_MODEL_MODE, SUB_MODEL_MODE,
    INTERFACE_MODE
}

    define special values for componentParallelMode (active model for parallel scheduling)
• enum { NO_DERIVS =0, ALL_DERIVS, MIXED_DERIVS }

    define special values for distParamDerivs
• enum { FT_LS, FT_RLS2 }
• enum { BASE_RESPONSE =0, SIMULATION_RESPONSE, EXPERIMENT_RESPONSE }

    special values for derived Response type
• enum { GENERIC_FNS = 0, OBJECTIVE_FNS, CALIB_TERMS }
  values for primary response types
• enum { DEFAULT_DOMAIN = 0, RELAXED_DOMAIN, MIXED_DOMAIN }
• enum {
  DEFAULT_VIEW = 0, ALL_VIEW, DESIGN_VIEW, UNCERTAIN_VIEW,
  ALEATORY_UNCERTAIN_VIEW, EPISTEMIC_UNCERTAIN_VIEW, STATE_VIEW }
• enum {
  EMPTY_VIEW = 0, RELAXED_ALL, MIXED_ALL, RELAXED_DESIGN,
  RELAXED_UNCERTAIN, RELAXED_ALEATORY_UNCERTAIN, RELAXED_EPSI-
  TEMIC_UNCERTAIN, RELAXED_STATE,
  MIXED_DESIGN, MIXED_UNCERTAIN, MIXED_ALEATORY_UNCERTAIN, MIXED_EPIS-
  TEMIC_UNCERTAIN,
  MIXED_STATE }
• enum { ALL_VARS = 0, ACTIVE_VARS, INACTIVE_VARS }
  values differentiating subsets of variables for I/O
• enum {
  EMPTY_TYPE = 0, CONTINUOUS_DESIGN, DISCRETE_DESIGN_RANGE, DISCRETE_DESI-
  GN_SET_INT,
  DISCRETE_DESIGN_SET_STRING, DISCRETE_DESIGN_SET_REAL, NORMAL_UNCERTAIN,
  LOGNORMAL_UNCERTAIN,
  UNIFORM_UNCERTAIN, LOGUNIFORM_UNCERTAIN, TRIANGULAR_UNCERTAIN, EXPONEN-
  TIAL_UNCERTAIN,
  BETA_UNCERTAIN, GAMMA_UNCERTAIN, GUMBEL_UNCERTAIN, FRECHET_UNCERTA-
  IN,
  WEIBULL_UNCERTAIN, HISTOGRAM_BIN_UNCERTAIN, POISSON_UNCERTAIN, BINOMI-
  AL_UNCERTAIN,
  NEGATIVE_BINOMIAL_UNCERTAIN, GEOMETRIC_UNCERTAIN, HYPERGEOMETRIC_U-
  NCERTAIN, HISTOGRAM_POINT_UNCERTAIN_INT,
  HISTOGRAM_POINT_UNCERTAIN_STRING, HISTOGRAM_POINT_UNCERTAIN_REAL, CO-
  Ntinuous_INTERVAL_UNCERTAIN, DISCRETE_INTERVAL_UNCERTAIN,
  DISCRETE_UNCERTAIN_SET_INT, DISCRETE_UNCERTAIN_SET_STRING, DISCRETE_UN-
  CERTAIN_SET_REAL, CONTINUOUS_STATE,
  DISCRETE_STATE_RANGE, DISCRETE_STATE_SET_INT, DISCRETE_STATE_SET_STRING,
  DISCRETE_STATE_SET_REAL }
• enum {
  TOTAL_CDV = 0, TOTAL_DDIV, TOTAL_DDSV, TOTAL_DDRV,
  TOTAL_CAUV, TOTAL_DAUV, TOTAL_DAUSV, TOTAL_DAURV,
  TOTAL_CEUV, TOTAL_DEUV, TOTAL_DEUSV, TOTAL_DEURV,
  TOTAL_CSV, TOTAL_DSV, TOTAL_DSSV, TOTAL_DSrv,
  NUM_VC_TOTALS }
• enum var_t {
  VAR_x1, VAR_x2, VAR_x3, VAR_b,
  VAR_h, VAR_P, VAR_M, VAR_Y,
  VAR_w, VAR_t, VAR_R, VAR_E,
  VAR_X, VAR_area_type, VAR_Fs, VAR_P1,
  VAR_P2, VAR_P3, VAR_B, VAR_D,
  VAR_H, VAR_F0, VAR_d, VAR_MForm,
  VAR_x, VAR_xi, VAR_Af, VAR_Ac }
### 13.2. Dakota Namespace Reference

**Enumeration of Possible Variable Types (to Index to Names)**

- **driver_t**
  - `NO_DRIVER` = 0
  - `CANTILEVER_BEAM`
  - `MOD_CANTILEVER_BEAM`
  - `CANTILEVER_BEAM_ML`
  - `CYLINDER_HEAD`
  - `EXTENDED_ROSENBOCK`
  - `GENERALIZED_ROSENBOCK`
  - `LF_ROSENBOCK`
  - `EXTRA_LF_ROSENBOCK`
  - `MF_ROSENBOCK`
  - `MODIFIED_ROSENBOCK`
  - `ROSENBOCK`
  - `LF_POLY_PROD`
  - `POLY_PROD`
  - `GERSTNER`
  - `SCALABLEGERSTNER`
  - `LOGNORMAL_RATIO`
  - `MULTIMODAL`
  - `PLUGIN_ROSENBOCK`
  - `PLUGIN_TEXTBOOK`
  - `SHORT_COLUMN`
  - `LF_SHORT_COLUMN`
  - `MF_SHORT_COLUMN`
  - `SIDE_IMPACT_COST`
  - `SIDE_IMPACT_PERFORMANCE`
  - `SOBOL_RATIONAL`
  - `SOBOL_FUNCTION`
  - `SOBOLISHIKAMI`
  - `STEEL_COLUMN_COST`
  - `STEEL_COLUMN_PERFORMANCE`
  - `TEXTBOOK`
  - `TEXTBOOK1`
  - `TEXTBOOK2`
  - `TEXTBOOK3`
  - `TEXTBOOKOUU`
  - `SCALABLEMONOMIALS`
  - `MOGATEST1`
  - `MOGATEST2`
  - `MOGATEST3`
  - `ILLUMINATION`
  - `BARNES`
  - `BARNESLF`
  - `HERBIE`
  - `SMOOTHHERBIE`
  - `SHUBERT`
  - `SALINAS`
  - `MODELCENTER`
  - `GENZ`
  - `DAMPEDOSCILLATOR`
  - `ANISOTROPICQUADRATICFORM`
  - `BAYESLINEAR`
  - `STEADYSTATE_DIFFUSION1D`
  - `SS_DIFFUSION_DISCREPANCY`
  - `TRANSIENTDIFFUSION_1D`
  - `PREDATORPREY`
  - `PROBLEM18`

**Enumeration of Possible Direct Driver Types (to Index to Names)**

- **local_data_t**
  - `VARIABLES_MAP` = 1
  - `VARIABLES_VECTOR` = 2

**Enumeration for How Local Variables Are Stored (Values Must Employ a Bit Representation)**

- **sigtype**
  - `NO_SIGMA`
  - `SCALAR_SIGMA`
  - `DIAGONAL_SIGMA`
  - `MATRIX_SIGMA`

**Special Values for SigmaType**

- **edtype**
  - `SCALAR_DATA`
  - `FUNCTIONAL_DATA`

**Special Values for Experimental Data Type**

- **CAUVar**
  - `normal` = 0
  - `lognormal` = 1
  - `uniform` = 2
  - `loguniform` = 3
  - `triangular` = 4
  - `exponential` = 5
  - `beta` = 6
  - `gamma` = 7
  - `gumbel` = 8
  - `frechet` = 9
  - `weibull` = 10
  - `histogram_bin` = 11
  - `Nkinds` = 12

- **DAUIVar**
  - `poisson` = 0
  - `binomial` = 1
  - `negative_binomial` = 2
  - `geometric` = 3
  - `hypergeometric` = 4
  - `histogram_point_int` = 5
  - `Nkinds` = 6

- **DAUSVar**
  - `histogram_point_str` = 0
  - `Nkinds` = 1

- **DEUIVar**
  - `interval` = 0
  - `set_int` = 1

- **DEUSVar**
  - `set_str` = 0
  - `Nkinds` = 1
• enum { DEURVar_set_real = 0, DEURVar_Nkinds = 1 }
• enum {
  DiscSetVar_design_set_int = 0, DiscSetVar_design_set_str = 1, DiscSetVar_design_set_real = 2, DiscSetVar_state_set_int = 3,
  DiscSetVar_state_set_str = 4, DiscSetVar_state_set_real = 5, DiscSetVar_Nkinds = 6 }
• enum { NUM_UNC_REAL_CONT = 4 }
  number of real-valued uncertain contiguous containers
• enum { NUM_UNC_INT_CONT = 2 }
  number of int-valued uncertain contiguous containers
• enum { NUM_UNC_STR_CONT = 2 }
  number of string-valued uncertain contiguous containers
• enum miAlg : unsigned short { MI_ALG_KSG1 = 0, MI_ALG_KSG2 = 1 }
• enum { FULL_TENSOR, FILTERED_TENSOR, RANDOM_TENSOR }
• enum CG_UPDATETYPE {
  CG_STEEPEST, CG_FLETCHER_REEVES, CG_POLAK_RIBIERE, CG_POLAK_RIBIERE_PLUS,
  CGHESTENES_STIEFEL }
  NonlinearCG update options.
• enum CG_LINESEARCHTYPE { CG_FIXED_STEP, CG_LS_SIMPLE, CG_LS_BRENT, CG_LS_WOLFE }
  NonlinearCG linesearch options.
• enum { AS_FUNC =1, AS_GRAD =2, AS_HESS =4 }
• enum { TYPE_U =1, TYPE_B =2, TYPE_E =3, TYPE_EB =4 }
• enum { CDV, LINEAR, LINEAR, NONLIN, FN_LSQ }
  to indicate type of object being scaled
• enum { DISALLOW, TARGET, BOUNDS }
  to restrict type of auto scaling allowed
• enum { SCALE_NONE = 0, SCALE_VALUE = 1, SCALE_LOG = 2, SCALE_AUTO = 4 }
  indicate type of scaling active for a component (bitwise)
• enum EvalType { NO_EVALUATOR, NLF_EVALUATOR, CON_EVALUATOR }
  enumeration for the type of evaluator function
• enum { APPROX_RESPONSE =1, TRUTH_RESPONSE }
• enum { CORR_APPROX_RESPONSE =1, UNCORR_APPROX_RESPONSE, CORR_TRUTH_RESPONSE, UNCORR_TRUTH_RESPONSE }
• enum {
  NEW_CANDIDATE = 1, CANDIDATE_ACCEPTED = 2, CANDIDATE_STATE = (NEW_CANDIDATE | CANDIDATE_ACCEPTED), NEW_CENTER = 8,
  CENTER_BUILT = 16, CENTER_STATE = (NEWCENTER | CENTER_BUILT), NEW_TR_FACTOR = 64, NEW_TRUST_REGION = (NEWCENTER | NEW_TR_FACTOR),
  HARD_CONVERGED = 128, SOFT_CONVERGED = 256, MIN_TR_CONVERGED = 512, MAX_ITER_CONVERGED = 1024,
  CONVERGED }
• enum {
    TH_SILENT_OUTPUT, TH_QUIET_OUTPUT, TH_NORMAL_OUTPUT, TH_VERBOSE_OUTPUT,
    TH_DEBUG_OUTPUT
}
• enum { DIR_CLEAN, DIR_PERSIST, DIR_ERROR }
  define directory creation options
• enum { FILEOP_SILENT, FILEOP_WARN, FILEOP_ERROR }
  enum indicating action on failed file operation

Functions
• void batch_means_interval (RealMatrix &mcmc_matrix, RealMatrix &interval_matrix, RealMatrix &means_matrix, int moment, Real alpha)
• void batch_means_percentile (RealMatrix &mcmc_matrix, RealMatrix &interval_matrix, RealMatrix &means_matrix, Real percentile, Real alpha)
• CommandShell & flush (CommandShell &shell)
  convenient shell manipulator function to "flush" the shell
• void read_sized_data (std::istream &s, RealVectorArray &va, size_t num_rows, int num_cols)
  istream extraction operator for configuration data of known dim and length
• void read_fixed_rowsize_data (std::istream &s, RealVectorArray &va, int num_cols, bool row_major)
  istream extraction operator for response data of known dim and unknown length
• void read_unsized_data (std::istream &s, RealVectorArray &va, bool row_major)
  istream extraction operator for coordinate data of unknown dim and unknown length
• void read_config_vars_multifile (const std::string &basename, int num_expts, int ncv, RealVectorArray &config_vars)
  file reader for configuration data supplied via multiple files
• void read_config_vars_singlefile (const std::string &basename, int num_expts, int ncv, RealVectorArray &config_vars)
  file reader for configuration data supplied via a single file
• void read_field_values (const std::string &basename, int expt_num, RealVectorArray &field_vars)
  file reader for vector field (response) data
• void read_field_values (const std::string &basename, int expt_num, RealVector &field_vars)
  file reader for scalar field (response) data
• void read_coord_values (const std::string &basename, int expt_num, RealMatrix &coords)
  file reader for experimental coordinate data
• void read_coord_values (const std::string &basename, RealMatrix &coords)
  file reader for simulation coordinate data
• void read_covariance (const std::string &basename, int expt_num, RealMatrix &cov_vals)
  file reader for CONSTANT covariance data
• void read_covariance (const std::string &basename, int expt_num, Dakota::CovarianceMatrix::FORMAT format, int num_vals, RealMatrix &cov_vals)
  file reader for VECTOR and MATRIX covariance data
• bool nearby (const RealVector &rv1, const RealVector &rv2, Real rel_tol)
  tolerance-based equality operator for RealVector
• bool operator==(const ShortArray &dsa1, const ShortArray &dsa2)
equality operator for ShortArray

- bool operator==(const StringArray &dsa1, const StringArray &dsa2)

equality operator for StringArray

- Real rel_change_L2 (const RealVector &curr_rv, const RealVector &prev_rv)
  Computes relative change between RealVectors using Euclidean L2 norm.

- Real rel_change_L2 (const RealVector &curr_rv1, const RealVector &prev_rv1, const IntVector &curr_iv, const IntVector &prev_iv, const RealVector &curr_rv2, const RealVector &prev_rv2)
  Computes relative change between Real/int/Real vector triples using Euclidean L2 norm.

- void compute_col_means (RealMatrix &matrix, RealVector &avg_vals)
  Computes means of columns of matrix.

- void compute_col_stdevs (RealMatrix &matrix, RealVector &avg_vals, RealVector &std_devs)
  Computes standard deviations of columns of matrix.

- void remove_column (RealMatrix &matrix, int index)
  Removes column from matrix.

- std::vector<std::string> strsplit (const std::string &input)
  Trim then split a string on {space, tab} and return as vector of strings.

- std::string::size_type longest_strlen (const std::vector<std::string> &vecstr)
  Return the length of the longest string in the passed vector.

- void iround (const RealVector &input_vec, IntVector &rounded_vec)
  round entries of a RealVector yielding an IntVector

- bool operator==(const IntArray &dia1, const IntArray &dia2)

equality operator for IntArray

- template<typename T >
  bool operator==(const std::vector<T> &vec, typename boost::multi_array<T, 1>::template const_array_view<1>::type mav)
  equality operator for std::vector and boost::multi_array::const_array_view

- template<typename T >
  bool operator==(typename boost::multi_array<T, 1>::template const_array_view<1>::type mav, const std::vector<T> &vec)
  equality operator for boost::multi_array::const_array_view and std::vector

- template<typename T >
  bool operator==(const boost::multi_array<T, 1> &ma, typename boost::multi_array<T, 1>::template const_array_view<1>::type mav)
  equality operator for boost::multi_array and boost::multi_array::const_array_view

- template<typename T >
  bool operator==(typename boost::multi_array<T, 1>::template const_array_view<1>::type mav, const boost::multi_array<T, 1> &ma)
  equality operator for boost::multi_array::const_array_view and boost::multi_array

- bool operator!=(const IntArray &dia1, const IntArray &dia2)
  inequality operator for IntArray

- bool operator!=(const ShortArray &dsa1, const ShortArray &dsa2)
  inequality operator for ShortArray

- bool operator!=(const StringArray &dsa1, const StringArray &dsa2)
  inequality operator for StringArray
13.2. DAKOTA NAMESPACE REFERENCE

- template<typename T>
  bool operator!= (const std::vector<T> &vec, typename boost::multi_array<T, 1>::template const_array_view<1>::type mav)
  inequality operator for std::vector and boost::multi_array::const_array_view

- template<typename T>
  bool operator!= (typename boost::multi_array<T, 1>::template const_array_view<1>::type mav, const std::vector<T> &vec)
  inequality operator for boost::multi_array::const_array_view and std::vector

- template<typename T>
  bool operator!= (const boost::multi_array<T, 1>& ma, typename boost::multi_array<T, 1>::template const_array_view<1>::type mav)
  inequality operator for boost::multi_array and boost::multi_array::const_array_view

- template<typename T>
  bool operator!= (typename boost::multi_array<T, 1>::template const_array_view<1>::type mav, const boost::multi_array<T, 1>& ma)
  inequality operator for boost::multi_array::const_array_view and boost::multi_array

- template<typename OrdinalType>
  bool non_zero (const std::vector<OrdinalType> &vec)
  checks for any non-zero value in std::vector(); useful for determining whether an array of request codes (e.g., an ASV) has any actionable content

- template<typename VectorType>
  bool is_equal (const RealVector &vec1, const VectorType &vec2)
  equality function for RealVector and a vector of arbitrary type

- template<typename MatrixType, typename VectorType>
  void apply_matrix_partial (const MatrixType &M, const VectorType &v1, VectorType &v2)
  Applies a RealMatrix to a vector (or subset of vector) v1.

- template<typename VectorType>
  void apply_matrix_transpose_partial (const RealMatrix &M, const VectorType &v1, VectorType &v2)
  Applies transpose of a RealMatrix to a vector (or subset of vector) v1.

- std::string strtolower (const std::string &s)
  Return lowercase copy of string s.

- bool strbegins (const std::string &input, const std::string &test)
  Return true if input string begins with string test.

- bool strends (const std::string &input, const std::string &test)
  Return true if input string ends with string test.

- bool strcontains (const std::string &input, const std::string &test)
  Return true if input string contains string test.

- void build_label (String &label, const String &root_label, size_t tag, const String &separator="")
  create a label by appending a numerical tag to the root_label, o

- void build_labels (StringArray &label_array, const String &root_label)
  create an array of labels by tagging root_label for each entry in label_array. Uses build_label().

- void build_labels (StringMultiArray &label_array, const String &root_label)
  create an array of labels by tagging root_label for each entry in label_array. Uses build_label().

- void build_labels_partial (StringArray &label_array, const String &root_label, size_t start_index, size_t num_items)
create a partial array of labels by tagging root label for a subset of entries in label_array. Uses build_label().

- template<typename vecType, typename valueType>
  void assign_value (vecType &target, valueType val)
  assign a value to an arbitrary vector

- template<typename vecType, typename valueType>
  void assign_value (vecType &target, valueType val, size_t start, size_t len)
  assign a value to a portion of an arbitrary vector

- template<typename OrdinalType, typename ScalarType>
  void copy_data (const Teuchos::SerialDenseVector<OrdinalType, ScalarType> &sdva, Teuchos::SerialDenseMatrix<OrdinalType, ScalarType> &sdm)
  copy Array<Teuchos::SerialDenseVector<OT,ST>> to Teuchos::SerialDenseMatrix<OT,ST> - used by read_data_tabular - RWH

- template<typename OrdinalType1, typename OrdinalType2, typename ScalarType>
  void copy_data (const Teuchos::SerialDenseVector<OrdinalType1, ScalarType> &sdv1, Teuchos::SerialDenseMatrix<OrdinalType1, ScalarType> &sdm1, OrdinalType2 nr, OrdinalType2 nc)
  copy Teuchos::SerialDenseVector<OT,ST> to Teuchos::SerialDenseMatrix<OT,ST> - used by NestedModel::update_sub_iterator - RWH

- template<typename T>
  void copy_data (const std::vector<std::vector<T>> &d2a, std::vector<T> &da)
  copy std::vector<vector<T>> to std::vector<T> (unroll vecOfvecs into vector) - used by ProcessApplicInterface::write_parameters_files - RWH

- template<typename T>
  void copy_data (const std::map<int, T> &im, std::vector<T> &da)
  copy map<int, T> to std::vector<T> (discard integer keys) - used by SurrBasedGlobalMinimizer::core_run - RWH

- template<typename OrdinalType, typename ScalarType>
  void copy_data (const Teuchos::SerialDenseVector<OrdinalType, ScalarType> &sdv1, Teuchos::SerialDenseVector<OrdinalType, ScalarType> &sdv2)
  copy Teuchos::SerialDenseVector<OrdinalType, ScalarType> to same (used in place of operator= when a deep copy is required) - used by Response - MSE

- template<typename OrdinalType, typename ScalarType>
  void copy_data (const Teuchos::SerialDenseMatrix<OrdinalType, ScalarType> &sdm1, Teuchos::SerialDenseMatrix<OrdinalType, ScalarType> &sdm2)
  copy Teuchos::SerialDenseMatrix<OrdinalType, ScalarType> to same (used in place of operator= when a deep copy is required) - used by Response - MSE

- template<typename OrdinalType, typename ScalarType>
  void copy_data (const Teuchos::SerialSymDenseMatrix<OrdinalType, ScalarType> &ssdm1, Teuchos::SerialSymDenseMatrix<OrdinalType, ScalarType> &ssdm2)
  copy Teuchos::SerialSymDenseMatrix<OrdinalType, ScalarType> to same (used in place of operator= when a deep copy is required) - used by Response - MSE

- void copy_data (const RealMatrix &source, RealMatrix &dest, int num_rows, int num_cols, int start_row=0, int start_col=0)
13.2. DAKOTA NAMESPACE REFERENCE

Taken from pecos/src/MathTools.hpp, BUT not templated because the implementation is specific to RealMatrix.

- template<typename OrdinalType, typename ScalarType, typename VecType>
  void copy_data (const Teuchos::SerialDenseVector<OrdinalType, ScalarType> &sdv, VecType &vec)

- template<typename OrdinalType , typename ScalarType1 , typename ScalarType2 >
  void copy_data (const Teuchos::SerialDenseVector<OrdinalType, ScalarType1 > &sdv, std::vector<ScalarType2> &vec)

- template<typename OrdinalType , typename ScalarType >
  void copy_data (const std::vector<ScalarType> &da, Teuchos::SerialDenseVector<OrdinalType, ScalarType> &sdv)

- template<typename OrdinalType1 , typename OrdinalType2 , typename ScalarType >
  void copy_data (const Teuchos::SerialDenseVector<OrdinalType1, ScalarType> &sdv, Teuchos::SerialDenseVector<OrdinalType1, ScalarType> &sdv2, OrdinalType2 start_index2)

- template<typename OrdinalType1 , typename OrdinalType2 , typename ScalarType >
  void copy_data (const Teuchos::SerialDenseVector<OrdinalType1, ScalarType> &sdv, Teuchos::SerialDenseVector<OrdinalType1, ScalarType> &sdv2, OrdinalType22 start_index2)

- template<typename OrdinalType1 , typename OrdinalType2 , typename ScalarType >
  void copy_data (const Teuchos::SerialDenseVector<OrdinalType1, ScalarType> &sdv, Teuchos::SerialDenseVector<OrdinalType1, ScalarType> &sdv2, OrdinalType22 start_index2)
CHAPTER 13. NAMESPACE DOCUMENTATION

copy portion of first SerialDenseVector to portion of second SerialDenseVector - used by ScalingModel::secondary_resp_scaled2native - RWH

- template<typename OrdinalType1, typename OrdinalType2, typename ScalarType>
  void copy_data_partial (const Teuchos::SerialDenseVector< OrdinalType1, ScalarType > &sdv1, std::vector< ScalarType > &da2, OrdinalType2 start_index2)

  copy all of first SerialDenseVector to portion of second SerialDenseVector - used by SharedSurfpackApproxData::merge_variable_arrays - RWH

- template<typename T>
  void copy_data_partial (const std::vector< T > &da1, size_t start_index1, size_t num_items, std::vector< T > &da2)

  copy portion of first Array<T> to all of second Array<T> - used by SharedResponseDataRep constructor - RWH

- template<typename T>
  void copy_data_partial (const std::vector< T > &da1, std::vector< T > &da2, size_t start_index2)

  copy all of first Array<T> to portion of second Array<T> - used by ParamStudy::multidim_loop - RWH

- template<typename T>
  void copy_data_partial (const std::vector< T > &da, boost::multi_array< T, 1 > &bma, size_t start_index_bma)

  copy all of first Array<T> to portion of boost::multi_array<T, 1> - used by RelaxedVariables - RWH

- template<typename VectorType>
  void copy_column_vector (const RealMatrix &m, RealMatrix::ordinalType j, VectorType &col)

  Copies a column of a Teuchos SerialDenseMatrix<int,Real> to std::vector<Real>

- template<typename VectorType>
  void copy_row_vector (const RealMatrix &m, RealMatrix::ordinalType i, VectorType &row)

  Copies a row of a Teuchos SerialDenseMatrix<int,Real> to std::vector<Real>

- template<typename ScalarType>
  void insert_row_vector (const std::vector< ScalarType > &row, RealMatrix::ordinalType i, RealMatrix &m)

    Inserts a std::vector<Real> into a row of a Teuchos SerialDenseMatrix<int,Real>

- void merge_data_partial (const IntVector &d_vec, RealVector &m_vec, size_t start_index_ma)

  merge a discrete integer vector into a single continuous vector

- void merge_data_partial (const IntVector &d_vec, RealArray &m_array, size_t start_index_ma)

  merge a discrete integer vector into a single continuous array

- template<typename OrdinalType, typename ScalarType>
  const ScalarType & set_index_to_value (OrdinalType index, const std::set< ScalarType > &values)

    retrieve the set value corresponding to the passed index

- template<typename ScalarType>
  size_t set_value_to_index (const ScalarType &value, const std::set< ScalarType > &values)

    calculate the set index corresponding to the passed value

- template<typename OrdinalType, typename KeyType, typename ValueType>
  const KeyType & map_index_to_key (OrdinalType index, const std::map< KeyType, ValueType > &pairs)

    retrieve the set value corresponding to the passed index

- template<typename OrdinalType, typename KeyType, typename ValueType>
  const ValueType & map_index_to_value (OrdinalType index, const std::map< KeyType, ValueType > &pairs)

    retrieve the set value corresponding to the passed index
13.2. DAKOTA NAMESPACE REFERENCE

- template<typename KeyType, typename ValueType>
  void map_keys_to_set (const std::map<KeyType, ValueType> &source_map, std::set<KeyType> &target_set)
  calculate the map index corresponding to the passed key

- template<typename KeyType, typename ValueType>
  size_t map_key_to_index (const KeyType &key, const std::map<KeyType, ValueType> &pairs)
  calculate the map index corresponding to the passed key

- template<typename KeyType, typename ValueType>
  size_t map_value_to_index (const ValueType &value, const std::map<KeyType, ValueType> &pairs)
  calculate the map index corresponding to the passed value

- template<typename OrdinalType, typename ScalarType>
  void x_y_pairs_to_x_set (const Teuchos::SerialDenseVector<OrdinalType, ScalarType> &xy_pairs, std::set<ScalarType> &x_set)
  convert a SerialDenseVector of head-to-tail (x,y) pairs into a std::set of (x), discarding the y values

- template<typename ContainerType>
  size_t find_index (const ContainerType &c, const typename ContainerType::value_type &search_data)
  generic find_index (inactive)

- template<typename T>
  size_t find_index (const boost::multi_array<T, 1> &bma, const T &search_data)
  compute the index of an entry within a boost::multi_array

- size_t find_index (SizetMultiArrayConstView bmacv, size_t search_data)
  compute the index of an entry within a boost::multi_array view

- size_t find_index (StringMultiArrayConstView bmacv, const String &search_data)
  compute the index of an entry within a boost::multi_array view

- template<typename ListT>
  size_t find_index (const ListT &l, const typename ListT::value_type &val)
  compute the index of an entry within a std::list

- template<typename ListT>
  ListT::const_iterator find_if (const ListT &l, bool(*test_fn)(const typename ListT::value_type &, const std::string &), const std::string &test_fn_data)
  return an iterator to the first list element satisfying the predicate test_fn w.r.t. the passed test_fn_data; end if not found

- template<typename VectorType, typename ScalarType>
  void copy_data (const std::vector<VectorType> &va, ScalarType *ptr, int ptr_len)

- void copy_data (SizetMultiArrayConstView ma, SizetArray &da)
  copy boost::multi_array view to Array - used by ActiveSet::derivative_vector - RWH

- void copy_data (StringMultiArrayConstView ma, StringArray &da)
  copy boost::multi_array view to Array - used by Pecos::copy_data - RWH

- template<typename DakContainerType>
  bool contains (const DakContainerType &v, const typename DakContainerType::value_type &val)
  return true if the item val appears in container v

- void abort_handler (int code)
  global function which handles serial or parallel aborts

- void abort_throw_or_exit (int dakota_code)
  throw or exit depending on abort_mode
• void register_signal_handlers()
  Tie various signal handlers to Dakota's abort handler function.
• void mpi_debug_hold()
  Global function to hold Dakota processes to help with MPI debugging.
• template<typename T>
  T abort_handler(int code)
• void svd(RealMatrix &matrix, RealVector &singularvals, RealMatrix &v_trans, bool compute_vectors=true)
  Compute the SVD of an arbitrary matrix \( \mathbf{A} = \mathbf{USV}^T \).
• void singular_values(RealMatrix &matrix, RealVector &singular_values)
  Compute the singular values without storing any singular vectors (A will be destroyed)
• int qr(RealMatrix &A)
  Compute an in-place QR factorization \( \mathbf{A} = \mathbf{QR} \).
• int qr_solve(const RealMatrix &qr, bool transpose, RealMatrix &rhs)
  Perform a multiple right-hand sides \( \mathbf{R}^{\pm} \mathbf{rhs} \) solve using the \( \mathbf{R} \) from a qr factorization.
• double det_AtransA(RealMatrix &A)
  Use SVD to compute \( \det(\mathbf{A}^*\mathbf{A}) \), destroying A with the SVD.
• std::string string_to_tmpfile(const std::string &dump_string)
  Utility to write an input string to a tmpfile in PWD
• std::string pyprepro_input(const std::string &template_file, const std::string &preproc_cmd="pyprepro.py")
  Run pyprepro on the user-provided input file and return generated tmp output
• ResultsKeyType make_key(const StrStrSizet &iterator_id, const std::string &data_name)
  Make a full ResultsKeyType from the passed iterator_id and data_name.
• MetaDataValueType make_metadatavalue(StringMultiArrayConstView labels)
  Create MetaDataValueType from the passed strings
• MetaDataValueType make_metadatavalue(StringMultiArrayConstView cv_labels, StringMultiArrayConstView div_labels, StringMultiArrayConstView dsv_labels, StringMultiArrayConstView drv_labels, const StringArray &resp_labels)
  Create MetaDataValueType from the passed strings
• MetaDataValueType make_metadatavalue(const StringArray &resp_labels)
  Create MetaDataValueType from the passed strings
• MetaDataValueType make_metadatavalue(const std::string &)
  Create MetaDataValueType from the passed strings
• MetaDataValueType make_metadatavalue(const std::string &, const std::string &)
  Create MetaDataValueType from the passed strings
• MetaDataValueType make_metadatavalue(const std::string &, const std::string &, const std::string &)
  Create MetaDataValueType from the passed strings
• MetaDataValueType make_metadatavalue(const std::string &, const std::string &, const std::string &, const std::string &)
  Create MetaDataValueType from the passed strings
• MetaDataValueType make_metadatavalue(StringMultiArrayConstView cv_labels, StringMultiArrayConstView div_labels, StringMultiArrayConstView dsv_labels, StringMultiArrayConstView drv_labels, const StringArray &resp_labels)
• int generate_system_seed()
  Clock microseconds-based random seed in [1, 1000000]
• std::istream & operator>>(std::istream &s, ActiveSet &set)
13.2. DAKOTA NAMESPACE REFERENCE

- std::istream extraction operator for ActiveSet. Calls read(std::istream&).
- std::ostream & operator<<(std::ostream &s, const ActiveSet &set)
- MPIUnpackBuffer & operator>>(MPIUnpackBuffer &s, ActiveSet &set)
  MPIUnpackBuffer extraction operator for ActiveSet. Calls read(MPIUnpackBuffer&).
- MPIPackBuffer & operator<<(MPIPackBuffer &s, const ActiveSet &set)
  MPIPackBuffer insertion operator for ActiveSet. Calls write(MPIPackBuffer&).
- bool operator!=(const ActiveSet &set1, const ActiveSet &set2)
  inequality operator for ActiveSet
- std::istream & operator>>(std::istream &s, Constraints &con)
- std::ostream & operator<<(std::ostream &s, const Constraints &con)
- std::string re_match(const std::string &token, const boost::regex &re)
  Global utility function to ease migration from CtelRegExp to Boost.Regex.
- bool interface_id_compare(const Interface &interface_in, const void *id)
  global comparison function for Interface
- bool method_id_compare(const Iterator &iterator, const void *id)
  global comparison function for Iterator
- bool model_id_compare(const Model &model, const void *id)
  global comparison function for Model
- bool operator==(const Model &m1, const Model &m2)
  equality operator for Envelope is true if same letter instance
- bool operator!=(const Model &m1, const Model &m2)
  inequality operator for Envelope is true if different letter instance
- template<typename VecT>
  void get_initial_values(const Model &model, VecT &values)
- template<typename VecT>
  bool get_bounds(const RealVector &lower_source, const RealVector &upper_source, VecT &lower_target, VecT &upper_target, Real big_real_bound_size, Real no_value)
- template<typename VecT>
  void get_bounds(const Model &model, VecT &lower_target, VecT &upper_target)
- template<typename SetT, typename VecT>
  void get_bounds(const SetT &source_set, VecT &lower_target, VecT &upper_target, int target_offset)
- template<typename OrdinalType, typename ScalarType, typename VectorType2, typename MaskType, typename SetArray>
  bool get_mixed_bounds(const MaskType &mask_set, const SetArray &source_set, const Teuchos::SerialDenseVector<OrdinalType, ScalarType> &lower_source, const Teuchos::SerialDenseVector<OrdinalType, ScalarType> &upper_source, VectorType2 &lower_target, VectorType2 &upper_target, ScalarType bigBoundSize, ScalarType no_value, int target_offset=0)
- template<typename AdapterT>
  bool get_variable_bounds(Model &model, Real big_real_bound_size, int big_int_bound_size, typename AdapterT::VecT &lower, typename AdapterT::VecT &upper)
- template<typename RVect, typename IVect>
  int configure_inequality_constrained_maps(const Model &model, Real big_real_bound_size, CONSTRAINT_T_TYPE ctype, IVect &map_indices, RVect &map_multipliers, RVect &map_offsets, Real scaling=1.0)
• template<typename RVecT, typename IVecT>
  void configure_equality_constraint_maps (Model &model, CONSTRAINT_TYPE ctype, IVecT &indices,
  size_t index_offset, RVecT &multipliers, RVecT &values, bool make_one_sided)
• template<typename AdapterT>
  void get_linear_constraints (Model &model, Real big_real_bound_size, typename AdapterT::VecT &lin_ineq_lower_bnds, typename AdapterT::VecT &lin_ineq_upper_bnds, typename AdapterT::VecT &lin_eq_targets, typename AdapterT::MatT &lin_ineq_coeffs, typename AdapterT::MatT &lin_eq_coeffs)
• template<typename VecT>
  void apply_linear_constraints (const Model &model, CONSTRAINT_EQUALITY_TYPE etype, const VecT &in_vals, VecT &values, bool adjoint=false)
• template<typename VecT>
  void apply_nonlinear_constraints (const Model &model, CONSTRAINT_EQUALITY_TYPE etype, const VecT &in_vals, VecT &values, bool adjoint=false)
• template<typename VecT1, typename VecT2, typename SetArray>
  void copy_variables (const VecT1 &source, const BitArray &set_bits, const SetArray &set_vars, VecT2 &dest, size_t offset, size_t len)
• template<typename VecT1, typename VecT2, typename SetArray>
  void copy_variables (const VecT1 &source, const SetArray &set_vars, VecT2 &dest, size_t offset, size_t len)
• template<typename VecT1, typename VecT2>
  void copy_variables (const VecT1 &source, VecT2 &dest, const BitArray &int_set_bits, const IntSetArray &int_set_vars, size_t offset, size_t len)
• template<typename AdapterT>
  void set_best_responses (typename AdapterT::OptT &optimizer, const Model &model, const std::vector<int> &constraint_map_indices, const std::vector<double> &constraint_map_multipliers, const std::vector<double> &constraint_map_offsets, ResponseArray &response_array)
• template<typename VectorType1, typename VectorType2>
  void set_variables (const VectorType1 &source, Model &model, Variables &vars)
• template<typename VectorType1, typename VectorType2>
  void get_variables (Model &model, VectorType1 &vec)
• template<typename vectorType>
  void get_responses (const Model &model, const RealVector &dak_fn_vals, const std::vector<int> &constraint_map_indices, const std::vector<double> &constraint_map_multipliers, const std::vector<double> &constraint_map_offsets, vectorType &f_vec, vectorType &cEqs_vec, vectorType &cIneqs_vec)
• template<typename VecT>
  void get_nonlinear_eq_constraints (const Model &model, VecT &values, Real scale, int offset=-1)
• template<typename VecT>
  void get_nonlinear_eq_constraints (Model &model, const RealVector &curr_resp_vals, VecT &values, Real scale, int offset=0)
• template<typename VecT>
  void get_nonlinear_ineq_constraints (const Model &model, VecT &values)
• template<typename VecT>
  void get_nonlinear_bounds (Model &model, VecT &nonlin_ineq_lower, VecT &nonlin_ineq_upper, VecT &nonlin_eq_targets)

Would like to combine the previous adapter with this one (based on APPSOptimizer and COLINOptimizer) and then see how much more generalization is needed to support other TPLs like JEGA.
• bool responses_id_compare (const Response &resp, const void *id)
  global comparison function for Response
13.2. DAKOTA NAMESPACE REFERENCE

- std::istream & operator>>(std::istream &s, Response &response)
  std::istream extraction operator for Response. Calls read(std::istream&).

- std::ostream & operator<<(std::ostream &s, Response &response)
  std::ostream insertion operator for Response. Calls write(std::ostream&).

- MPIUnpackBuffer & operator>>(MPIUnpackBuffer &s, Response &response)
  MPIUnpackBuffer extraction operator for Response. Calls read(MPIUnpackBuffer&).

- MPIPackBuffer & operator<<(MPIUnpackBuffer &s, const Response &response)
  MPIPackBuffer insertion operator for Response. Calls write(MPIUnpackBuffer&).

- bool operator!=(const Response &resp1, const Response &resp2)
  inequality operator for Response.

- bool variables_id_compare(const Variables &vars, const void *id)
  global comparison function for Variables.

- std::istream & operator>>(std::istream &s, Variables &vars)
  std::istream extraction operator for Variables.

- std::ostream & operator<<(std::ostream &s, const Variables &vars)
  std::ostream insertion operator for Variables.

- MPIUnpackBuffer & operator>>(MPIUnpackBuffer &s, Variables &vars)
  MPIUnpackBuffer extraction operator for Variables.

- MPIPackBuffer & operator<<(MPIPackBuffer &s, const Variables &vars)
  MPIPackBuffer insertion operator for Variables.

- bool operator!=(const Variables &vars1, const Variables &vars2)
  inequality operator for Variables.

- template<typename OrdinalType , typename ScalarType1 , typename ScalarType2 , typename ScalarType3 , typename ScalarType4>
  void write_ordered(std::ostream &s, const SizetArray &comp_totals, const Teuchos::SerialDenseVector<OrdinalType, ScalarType1> &c_vector, const Teuchos::SerialDenseVector<OrdinalType, ScalarType2> &di_vector, const Teuchos::SerialDenseVector<OrdinalType, ScalarType3> &ds_vector, const Teuchos::SerialDenseVector<OrdinalType, ScalarType4> &dr_vector)
  free function to write Variables data vectors in input spec ordering.

- template<typename OrdinalType , typename ScalarType1 , typename ScalarType2 , typename ScalarType3 , typename ScalarType4>
  void write_ordered(std::ostream &s, const SizetArray &comp_totals, const Teuchos::SerialDenseVector<OrdinalType, ScalarType1> &c_vector, const Teuchos::SerialDenseVector<OrdinalType, ScalarType2> &di_vector, const Teuchos::SerialDenseVector<OrdinalType, ScalarType3> &ds_array, const Teuchos::SerialDenseVector<OrdinalType, ScalarType4> &dr_vector)
  free function to write Variables data vectors in input spec ordering.

- template<typename ScalarType>
  void write_ordered(std::ostream &s, const SizetArray &comp_totals, const std::vector<ScalarType> &c_array, const std::vector<ScalarType> &di_array, const std::vector<ScalarType> &ds_array, const std::vector<ScalarType> &dr_array)
  free function to write Variables data vectors in input spec ordering.

- MPIPackBuffer & operator<(MPIPackBuffer &s, const DataEnvironment &data)
  MPIPackBuffer insertion operator for DataEnvironment.

- MPIUnpackBuffer & operator>>(MPIUnpackBuffer &s, DataEnvironment &data)
  MPIUnpackBuffer extraction operator for DataEnvironment.
• `std::ostream & operator<< (std::ostream &s, const DataEnvironment &data)
  `std::ostream insertion operator for DataEnvironment`.
• `MPIPackBuffer & operator<<(MPIPackBuffer &s, const DataInterface &data)
  `MPIPackBuffer insertion operator for DataInterface`.
• `MPIUnpackBuffer & operator>>(MPIUnpackBuffer &s, DataInterface &data)
  `MPIUnpackBuffer extraction operator for DataInterface`.
• `std::ostream & operator<< (std::ostream &s, const DataInterface &data)
  `std::ostream insertion operator for DataInterface`.
• `MPIPackBuffer & operator<<(MPIPackBuffer &s, const DataMethod &data)
  `MPIPackBuffer insertion operator for DataMethod`.
• `MPIUnpackBuffer & operator>>(MPIUnpackBuffer &s, DataMethod &data)
  `MPIUnpackBuffer extraction operator for DataMethod`.
•`std::ostream & operator<< (std::ostream &s, const DataMethod &data)
  `std::ostream insertion operator for DataMethod`.
• `MPIPackBuffer & operator<<(MPIPackBuffer &s, const DataModel &data)
  `MPIPackBuffer insertion operator for DataModel`.
• `MPIUnpackBuffer & operator>>(MPIUnpackBuffer &s, DataModel &data)
  `MPIUnpackBuffer extraction operator for DataModel`.
• `std::ostream & operator<< (std::ostream &s, const DataModel &data)
  `std::ostream insertion operator for DataModel`.
• `MPIPackBuffer & operator<<(MPIPackBuffer &s, const DataResponses &data)
  `MPIPackBuffer insertion operator for DataResponses`.
• `MPIUnpackBuffer & operator>>(MPIUnpackBuffer &s, DataResponses &data)
  `MPIUnpackBuffer extraction operator for DataResponses`.
• `std::ostream & operator<< (std::ostream &s, const DataResponses &data)
  `std::ostream insertion operator for DataResponses`.
• `MPIPackBuffer & operator<<(MPIPackBuffer &s, const DataVariables &data)
  `MPIPackBuffer insertion operator for DataVariables`.
• `MPIUnpackBuffer & operator>>(MPIUnpackBuffer &s, DataVariables &data)
  `MPIUnpackBuffer extraction operator for DataVariables`.
• `std::ostream & operator<< (std::ostream &s, const DataVariables &data)
  `std::ostream insertion operator for DataVariables`.
• `int dl solver option` (Opt_Info *)
• `RealVector const * continuous_lower_bounds` (Optimizer1 *o)
• `RealVector const * continuous_upper_bounds` (Optimizer1 *o)
• `RealVector const * nonlinear_ineq_constraint_lower_bounds` (Optimizer1 *o)
• `RealVector const * nonlinear_ineq_constraint_upper_bounds` (Optimizer1 *o)
• `RealVector const * nonlinear_eq_constraint_targets` (Optimizer1 *o)
• `RealVector const * linear_ineq_constraint_lower_bounds` (Optimizer1 *o)
• `RealVector const * linear_ineq_constraint_upper_bounds` (Optimizer1 *o)
• `RealVector const * linear_eq_constraint_targets` (Optimizer1 *o)
• `RealMatrix const * linear_eq_constraint_coeffs` (Optimizer1 *o)
• `RealMatrix const * linear_eq_constraint_coeffs` (Optimizer1 *o)
13.2. **DAKOTA NAMESPACE REFERENCE**

- void **ComputeResponses** (Optimizer1 o, int mode, int n, double *x)
- void **GetFuncs** (Optimizer1 o, int m0, int m1, double *f)
- void **GetGrads** (Optimizer1 o, int m0, int m1, int n, int is, int js, double *g)
- void **GetContVars** (Optimizer1 o, int n, double *x)
- void **SetBestContVars** (Optimizer1 o, int n, double *x)
- void **SetBestRespFns** (Optimizer1 o, int n, double *x)
- void **dl_constructor** (Optimizer1 *, Dakota_funcs *, dl_core_run_t *, dl Destructor_t *)
- static RealVector const * **continuous_lower_bounds1** (Optimizer1 o)
- static RealVector const * **continuous_upper_bounds1** (Optimizer1 o)
- static RealVector const * **nonlinear_ineq_constraint_lower_bounds1** (Optimizer1 o)
- static RealVector const * **nonlinear_ineq_constraint_upper_bounds1** (Optimizer1 o)
- static RealVector const * **nonlinear_eq_constraint_targets1** (Optimizer1 o)
- static RealVector const * **linear_ineq_constraint_lower_bounds1** (Optimizer1 o)
- static RealVector const * **linear_ineq_constraint_upper_bounds1** (Optimizer1 o)
- static RealVector const * **linear_eq_constraint_targets1** (Optimizer1 o)
- static RealMatrix const * **linear_eq_constraint_coeffs1** (Optimizer1 o)
- static void **ComputeResponses1** (Optimizer1 o, int mode, int n, double *x)
- static void **GetFuncs1** (Optimizer1 o, int m0, int m1, double *f)
- static void **GetGrads1** (Optimizer1 o, int m0, int m1, int n, int is, int js, double *g)
- static void **GetContVars1** (Optimizer1 o, int n, double *x)
- static void **SetBestContVars1** (Optimizer1 o, int n, double *x)
- static void **SetBestRespFns1** (Optimizer1 o, int n, double *x)
- static double **Get_Real1** (Optimizer1 o, const char *name)
- static int **Get_Int1** (Optimizer1 o, const char *name)
- static bool **Get_Bool1** (Optimizer1 o, const char *name)
- DOTOptimizer * **new_DOTOptimizer** (ProblemDescDB &problem_db)
- DOTOptimizer * **new_DOTOptimizer** (Model &model)
- DOTOptimizer * **new_DOTOptimizer** (ProblemDescDB &problem_db, Model &model)
- void **copy_field_data** (const RealVector &fn_vals, RealMatrix &fn_grad, const RealSymMatrixArray &fn_hess, size_t offset, size_t num_fns, Response &response)
- void **copy_field_data** (const RealVector &fn_vals, RealMatrix &fn_grad, const RealSymMatrixArray &fn_hess, size_t offset, size_t num_fns, short total_asv, Response &response)
- void **interpolate_simulation_field_data** (const Response &sim_res, const RealMatrix &exp_coords, size_t field_num, short total_asv, size_t interp_res_offset, Response &interp_res)
- void **linear_interpolate_1d** (const RealMatrix &build_pts, const RealVector &build_vals, const RealMatrix &build_grads, const RealSymMatrixArray &build_hessians, const RealMatrix &pred_pts, RealVector &pred_vals, RealMatrix &pred_grads, RealSymMatrixArray &pred_hessians)

*Returns the value of at 1D function f and its gradient and hessians (if available) at the points of vector pred_pts using linear interpolation. The vector build_pts specifies the coordinates of the underlying interval at which the values (build_vals) of the function f are known. The length of output pred_vals is equal to the length of pred_pts. This function assumes the build_pts is in ascending order.*

- void **symmetric_eigenvalue_decomposition** (const RealSymMatrix &matrix, RealVector &eigenvalues, RealMatrix &eigenvectors)

*Computes the eigenvalues and, optionally, eigenvectors of a real symmetric matrix A.*
- void **compute_column_means** (RealMatrix &matrix, RealVector &avg_vals)
Compute the means of each column of an arbitrary matrix.

- **void sort_vector** (const RealVector &vec, RealVector &sort_vec, IntVector &indices)
  
  Sort incoming vector with result and corresponding indices returned in passed arguments.

- **void sort_matrix_columns** (const RealMatrix &mat, RealMatrix &sort_mat, IntMatrix &indices)
  
  Sort incoming matrix columns with result and corresponding indices returned in passed arguments.

- **bool is_matrix_symmetric** (const RealMatrix &matrix)
  
  Test if incoming matrix is symmetric.

- **template<typename O, typename T>**
  
  **int binary_search** (T target, Teuchos::SerialDenseVector< O, T > &data)
  
  Find the interval containing a target value. This function assumes the data is in ascending order.

- **Real getdist** (const RealVector &x1, const RealVector &x2)

- **Real mindist** (const RealVector &x, const RealMatrix &xset, int except)

- **Real mindistidx** (const RealVector &x, const RealMatrix &xset, const IntArray &indx)

- **Real getRmax** (const RealMatrix &xset)

- **int start_grid_computing** (char *analysis_driver_script, char *params_file, char *results_file)

- **int stop_grid_computing** ()

- **int perform_analysis** (char *iteration_num)

- **int length** (const StringMultiArrayConstView &vec)
  
  Return the length of a StringMultiArrayConstView.

- **H5::DataType h5_file_dtype** (const short &)
  
  Return the HDF5 datatype to store a short.

- **H5::DataType h5_file_dtype** (const int &)
  
  Return the HDF5 datatype to store an int.

- **H5::DataType h5_file_dtype** (const unsigned int &)

- **H5::DataType h5_file_dtype** (const unsigned long &)

- **H5::DataType h5_file_dtype** (const Real &)
  
  Return the HDF5 datatype to store a Real.

- **H5::DataType h5_file_dtype** (const char *)
  
  Return the HDF5 datatype to store a string.

- **H5::DataType h5_file_dtype** (const ResultsOutputType t)
  
  Overloads for ResultsOutputType (used when creating empty datasets)

- **H5::DataType h5_file_dtype** (const String &)
  
  Return the HDF5 datatype to store a string.

- **H5::DataType h5_mem_dtype** (const Real &)
  
  Return the HDF5 datatype to read a Real in memory.

- **H5::DataType h5_mem_dtype** (const short &)
  
  Return the HDF5 datatype to read a short in memory.

- **H5::DataType h5_mem_dtype** (const int &)
  
  Return the HDF5 datatype to read an int in memory.

- **H5::DataType h5_mem_dtype** (const unsigned long &)
  
  Return the HDF5 datatype to read an unsigned long (maybe a size_t) in memory.

- **H5::DataType h5_mem_dtype** (const unsigned int &)
  
  Return the HDF5 datatype to read a uint in memory.

- **H5::DataType h5_mem_dtype** (const char *)
Return the HDF5 datatype to read a string in memory.

- H5::DataType h5_mem_dtype (const String &)
  Return the HDF5 datatype to read a string in memory.

- H5::DataType h5_mem_dtype (const ResultsOutputType t)
  Overloads for ResultsOutputType (used when creating empty datasets)

- template<typename T>
  int length (const std::vector<T> &vec)
  Return the length of several types.

- template<typename T>
  int length (const Teuchos::SerialDenseVector<int, T>& vec)
  Return the length of an SDV.

- template<typename T>
  std::vector<const char *> pointers_to_strings (const T &data)
  Return a vector of pointers to strings.

- template<typename T>
  string asstring (const T &val)
  Creates a string from the argument val using an ostringstream.

- PACKBUF (int, MPI_INT) PACKBUF(u_int)
- MPI_UNSIGNED PACKBUF (long, MPI_LONG) PACKBUF(u_long)
- MPI_UNSIGNED MPI_UNSIGNED_LONG PACKBUF (short, MPI_SHORT) PACKBUF(u_short)
- MPI_UNSIGNED MPI_UNSIGNED_LONG
  MPI_UNSIGNED_SHORT PACKBUF (char, MPI_CHAR) PACKBUF(u_char)
- MPI_UNSIGNED MPI_UNSIGNED_LONG
  MPI_UNSIGNED_SHORT
  MPI_UNSIGNED_CHAR PACKBUF (double, MPI_DOUBLE) PACKBUF(float)

- UNPACKBUF (int, MPI_INT) UNPACKBUF(u_int)
- MPI_UNSIGNED UNPACKBUF (long, MPI_LONG) UNPACKBUF(u_long)
- MPI_UNSIGNED MPI_UNSIGNED_LONG UNPACKBUF (short, MPI_SHORT) UNPACKBUF(u_short)
- MPI_UNSIGNED MPI_UNSIGNED_LONG
  MPI_UNSIGNED_SHORT UNPACKBUF (char, MPI_CHAR) UNPACKBUF(u_char)
- MPI_UNSIGNED MPI_UNSIGNED_LONG
  MPI_UNSIGNED_SHORT
  MPI_UNSIGNED_CHAR UNPACKBUF (double, MPI_DOUBLE) UNPACKBUF(float)

- PACKSIZE (int, MPI_INT) PACKSIZE(u_int)
- MPI_UNSIGNED PACKSIZE (long, MPI_LONG) PACKSIZE(u_long)
- MPI_UNSIGNED MPI_UNSIGNED_LONG PACKSIZE (short, MPI_SHORT) PACKSIZE(u_short)
- MPI_UNSIGNED MPI_UNSIGNED_LONG
  MPI_UNSIGNED_SHORT PACKSIZE (char, MPI_CHAR) PACKSIZE(u_char)
- MPI_UNSIGNED MPI_UNSIGNED_LONG
  MPI_UNSIGNED_SHORT
  MPI_UNSIGNED_CHAR PACKSIZE (double, MPI_DOUBLE) PACKSIZE(float)
- MPI_UNSIGNED MPI_UNSIGNED_LONG
  MPI_UNSIGNED_SHORT
  MPI_UNSIGNED_CHAR MPI_FLOAT
  int MPIPackSize (const bool &data, const int num=1)
  return packed size of a bool
• MPIPackBuffer & operator<<(MPIPackBuffer &buff, const int &data)
  insert an int
• MPIPackBuffer & operator<<(MPIPackBuffer &buff, const u_int &data)
  insert a u_int
• MPIPackBuffer & operator<<(MPIPackBuffer &buff, const long &data)
  insert a long
• MPIPackBuffer & operator<<(MPIPackBuffer &buff, const u_long &data)
  insert a u_long
• MPIPackBuffer & operator<<(MPIPackBuffer &buff, const short &data)
  insert a short
• MPIPackBuffer & operator<<(MPIPackBuffer &buff, const u_short &data)
  insert a u_short
• MPIPackBuffer & operator<<(MPIPackBuffer &buff, const char &data)
  insert a char
• MPIPackBuffer & operator<<(MPIPackBuffer &buff, const u_char &data)
  insert a u_char
• MPIPackBuffer & operator<<(MPIPackBuffer &buff, const double &data)
  insert a double
• MPIPackBuffer & operator<<(MPIPackBuffer &buff, const float &data)
  insert a float
• MPIPackBuffer & operator<<(MPIPackBuffer &buff, const bool &data)
  insert a bool
• MPIUnpackBuffer & operator>>(MPIUnpackBuffer &buff, int &data)
  extract an int
• MPIUnpackBuffer & operator>>(MPIUnpackBuffer &buff, u_int &data)
  extract a u_int
• MPIUnpackBuffer & operator>>(MPIUnpackBuffer &buff, long &data)
  extract a long
• MPIUnpackBuffer & operator>>(MPIUnpackBuffer &buff, u_long &data)
  extract a u_long
• MPIUnpackBuffer & operator>>(MPIUnpackBuffer &buff, short &data)
  extract a short
• MPIUnpackBuffer & operator>>(MPIUnpackBuffer &buff, u_short &data)
  extract a u_short
• MPIUnpackBuffer & operator>>(MPIUnpackBuffer &buff, char &data)
  extract a char
• MPIUnpackBuffer & operator>>(MPIUnpackBuffer &buff, u_char &data)
  extract a u_char
• MPIUnpackBuffer & operator>>(MPIUnpackBuffer &buff, double &data)
  extract a double
• MPIUnpackBuffer & operator>>(MPIUnpackBuffer &buff, float &data)
  extract a float
• MPIUnpackBuffer & operator>>(MPIUnpackBuffer &buff, bool &data)
  extract a bool
extract a bool

- template<class ContainerT >
  void container_read (ContainerT &c, MPIUnpackBuffer &s)
  Read a generic container (vector<T>, list<T>) from MPIUnpackBuffer, s.

- template<class ContainerT >
  void container_write (const ContainerT &c, MPIPackBuffer &s)
  Write a generic container to MPIPackBuffer, s.

- template<typename Block, typename Allocator >
  MPIPackBuffer & operator<< (MPIPackBuffer &s, const boost::dynamic_bitset<Block, Allocator> &bs)
  stream insertion for BitArray

- template<typename Block, typename Allocator >
  MPIUnpackBuffer & operator>> (MPIUnpackBuffer &s, const boost::dynamic_bitset<Block, Allocator> &bs)
  stream extraction for BitArray

- template<class ContainerT >
  MPIUnpackBuffer & operator>>(MPIUnpackBuffer &s, ContainerT &data)
  global MPIUnpackBuffer extraction operator for generic container

- template<class ContainerT >
  MPIPackBuffer & operator<<(MPIPackBuffer &s, const ContainerT &data)
  global MPIPackBuffer insertion operator for generic container

- int MPIPackSize (const int &data, const int num=1)
  return packed size of an int

- int MPIPackSize (const u_int &data, const int num=1)
  return packed size of a u_int

- int MPIPackSize (const long &data, const int num=1)
  return packed size of a long

- int MPIPackSize (const u_long &data, const int num=1)
  return packed size of a u_long

- int MPIPackSize (const short &data, const int num=1)
  return packed size of a short

- int MPIPackSize (const u_short &data, const int num=1)
  return packed size of a u_short

- int MPIPackSize (const char &data, const int num=1)
  return packed size of a char

- int MPIPackSize (const u_char &data, const int num=1)
  return packed size of a u_char

- int MPIPackSize (const double &data, const int num=1)
  return packed size of a double

- int MPIPackSize (const float &data, const int num=1)
  return packed size of a float

- int nidr_Parse (const char *, FILE *)

- const char ** arg_list_adjust (const char **, void **)

- int not_executable (const char *driver_name, const char *tdir)

- static void scale_chk (StringArray &ST, RealVector &S, const char *what, const char **univ)
CHAPTER 13. NAMESPACE DOCUMENTATION

- static void BuildLabels (StringArray *sa, size_t nsa, size_t n1, size_t n2, const char *stub)
- static int mixed_check (IntSet *S, int n, IntArray *iv, const char *what)
- static void mixed_check2 (size_t n, IntArray *iv, const char *what)
- static int wronglen (size_t n, RealVector *V, const char *what)
- static int wronglen (size_t n, RealVector *V, const char *what)
- static void Vcopyup (RealVector *V, RealVector *M, size_t i, size_t n)
- static void Setrv (RealVector *V, double d, size_t n)
- static void Setiv (IntVector *V, int d, size_t n)
- static void wrong_number (const char *what, const char *kind, size_t nsv, size_t m)
- static void too_small (const char *kind)
- static void not_div (const char *kind, size_t nsv, size_t m)
- static void suppressed (const char *kind, int ndup, int *ip, String *sp, Real *rp)
- static void bad_initial_ivalue (const char *kind, int val)
- static void bad_initial_svalue (const char *kind, String val)
- static void bad_initial_rvalue (const char *kind, Real val)
- static void Vgen_ContinuousDes (DataVariablesRep *dv, size_t offset)
- static void Vgen_DiscreteDesRange (DataVariablesRep *dv, size_t offset)
- static void Vgen_DiscreteState (DataVariablesRep *dv, size_t offset)
- static void Vgen_DiscreteStateRange (DataVariablesRep *dv, size_t offset)
- static void Vchk_NormalUnc (DataVariablesRep *dv, size_t offset, Var_Info *vi)
- static void Vgen_NormalUnc (DataVariablesRep *dv, size_t offset)
- static void Vchk_LognormalUnc (DataVariablesRep *dv, size_t offset, Var_Info *vi)
- static void Vgen_LognormalUnc (DataVariablesRep *dv, size_t offset)
- static void Vchk_UniformUnc (DataVariablesRep *dv, size_t offset, Var_Info *vi)
- static void Vgen_UniformUnc (DataVariablesRep *dv, size_t offset)
- static void Vchk_LoguniformUnc (DataVariablesRep *dv, size_t offset, Var_Info *vi)
- static void Vgen_LoguniformUnc (DataVariablesRep *dv, size_t offset)
- static void Vchk_TriangularUnc (DataVariablesRep *dv, size_t offset, Var_Info *vi)
- static void Vgen_TriangularUnc (DataVariablesRep *dv, size_t offset)
- static void Vchk_ExponentialUnc (DataVariablesRep *dv, size_t offset, Var_Info *vi)
- static void Vgen_ExponentialUnc (DataVariablesRep *dv, size_t offset)
- static void Vchk_BetaUnc (DataVariablesRep *dv, size_t offset, Var_Info *vi)
- static void Vgen_BetaUnc (DataVariablesRep *dv, size_t offset)
- static void Vchk_GammaUnc (DataVariablesRep *dv, size_t offset, Var_Info *vi)
- static void Vgen_GammaUnc (DataVariablesRep *dv, size_t offset)
- static void Vchk_GumbelUnc (DataVariablesRep *dv, size_t offset, Var_Info *vi)
- static void Vgen_GumbelUnc (DataVariablesRep *dv, size_t offset)
- static void Vchk_FréchetUnc (DataVariablesRep *dv, size_t offset, Var_Info *vi)
- static void Vgen_FréchetUnc (DataVariablesRep *dv, size_t offset)
- static void Vchk_WeibullUnc (DataVariablesRep *dv, size_t offset, Var_Info *vi)
- static void Vgen_WeibullUnc (DataVariablesRep *dv, size_t offset)
- static void Vchk_HistogramBinUnc (DataVariablesRep *dv, size_t offset, Var_Info *vi)

Check the histogram bin input data, normalize the counts and populate the histogramUncBinPairs map data structure; map keys are guaranteed unique since the abscissas must increase.
- static void Vgen_HistogramBinUnc (DataVariablesRep *dv, size_t offset)
13.2. DAKOTA NAMESPACE REFERENCE

Infer lower/upper bounds for histogram and set initial variable values based on initial point or moments, snapping to bounds as needed. (Histogram bin doesn’t have lower/upper bounds specification)

- static void Vchk_PoissonUnc (DataVariablesRep *dv, size_t offset, Var_Info *vi)
- static void Vgen_PoissonUnc (DataVariablesRep *dv, size_t offset)
- static void Vchk_BinomialUnc (DataVariablesRep *dv, size_t offset, Var_Info *vi)
- static void Vgen_BinomialUnc (DataVariablesRep *dv, size_t offset)
- static void Vchk_NegBinomialUnc (DataVariablesRep *dv, size_t offset, Var_Info *vi)
- static void Vgen_NegBinomialUnc (DataVariablesRep *dv, size_t offset)
- static void Vchk_GeometricUnc (DataVariablesRep *dv, size_t offset, Var_Info *vi)
- static void Vgen_GeometricUnc (DataVariablesRep *dv, size_t offset)
- static void Vchk_HyperGeomUnc (DataVariablesRep *dv, size_t offset, Var_Info *vi)
- static void Vgen_HyperGeomUnc (DataVariablesRep *dv, size_t offset)
- static void Vchk_HistogramPtIntUnc (DataVariablesRep *dv, size_t offset, Var_Info *vi)
- static void Vgen_HistogramPtIntUnc (DataVariablesRep *dv, size_t offset)

Check the histogram point integer input data, normalize the counts, and populate DataVariables::histogramUncPointIntPairs; map keys are guaranteed unique since the abscissas must increase.

- static void Vchk_HistogramPtStrUnc (DataVariablesRep *dv, size_t offset, Var_Info *vi)
- static void Vgen_HistogramPtStrUnc (DataVariablesRep *dv, size_t offset)

Use the integer-valued point histogram data to initialize the lower, upper, and initial values of the variables, using value closest to mean if no initial point.

- static void Vchk_HistogramPtRealUnc (DataVariablesRep *dv, size_t offset, Var_Info *vi)
- static void Vgen_HistogramPtRealUnc (DataVariablesRep *dv, size_t offset)

Use the real-valued point histogram data to initialize the lower, upper, and initial values of the variables, using value closest to mean if no initial point.

- static void Vchk_DiscreteIntervalUnc (DataVariablesRep *dv, size_t offset, Var_Info *vi)
- static void Vgen_DiscreteIntervalUnc (DataVariablesRep *dv, size_t offset)

Check the discrete interval uncertain input data and populate DataVariables::discreteIntervalUncBasicProbs; map keys (integer intervals) are checked for uniqueness because we don’t have a theoretically sound way to combine duplicate intervals.

- static void Vchk_ContinuousIntervalUnc (DataVariablesRep *dv, size_t offset, Var_Info *vi)
- static void Vgen_ContinuousIntervalUnc (DataVariablesRep *dv, size_t offset)

Check the continuous interval uncertain input data and populate DataVariables::continuousIntervalUncBasicProbs; map keys (real intervals) are checked for uniqueness because we don’t have a theoretically sound way to combine duplicate intervals.

- static void check_set_keys (size_t num_v, size_t ds_len, const char *kind, IntArray *input_nds, int &avg_num_ds)

validate the number of set elements (values) given the number of variables and an optional apportionment with elements_per_variable; return the average number per variable if equally distributed

- static void Vchk_DIset (size_t num_v, const char *kind, IntArray *input_nds, IntVector *input_dsi, IntSetArray &dsi_all, IntVector &dsi_init_pt)
CHAPTER 13. NAMESPACE DOCUMENTATION

check discrete sets of integers (design and state variables); error if a duplicate value is specified error if not ordered to prevent user confusion

- static void \\*Vchk\\_Dlset (size_t num_v, const char *kind, IntArray *input\_ndsi, IntVector *input\_dsi, RealVector *input\_dsip, IntRealMapArray &dss, IntVector &dsi, StringArray &sets, IntVector &L, IntVector &U, IntVector &V, bool aggregate)

check discrete sets of integers (uncertain variables); error if a duplicate value is specified error if not ordered to prevent user confusion

- static void \\*Vchk\\_DSset (size_t num_v, const char *kind, IntArray *input\_ndss, StringArray *input\_dss, RealVector *input\_dssp, StringRealMapArray &dss, StringArray &sets, StringArray &L, StringArray &U, StringArray &V, bool aggregate)

- static void \\*Vchk\\_DRset (size_t num_v, const char *kind, IntArray *input\_ndsr, RealVector *input\_dsr, RealSetArray &dss, RealMatrixArray &dda, RealVector &dsr, RealRealMapArray &dsr)

- static void \\*Vchk\\_Adjacency (size_t num_v, const char *kind, const IntArray &num_e, const IntVector &input\_dss, Real\_MatrixArray &dda)

- static bool \\*check\_LUV\_size (size_t num_v, IntVector &L, IntVector &U, IntVector &V, bool aggregate\_LUV, size_t offset)

- static bool \\*check\_LUV\_size (size_t num_v, StringArray &L, StringArray &U, StringArray &V, bool aggregate\_LUV, size_t offset)

- static bool \\*check\_LUV\_size (size_t num_v, RealVector &L, RealVector &U, RealVector &V, bool aggregate\_LUV, size_t offset)

- static void \\*Vgen\\_Dlset (size_t num_v, IntSetArray &sets, IntVector &L, IntVector &U, IntVector &V, bool aggregate\_LUV=false, size_t offset=0)

- static void \\*Vgen\\_DSset (size_t num_v, StringSetArray &sets, StringArray &L, StringArray &U, StringArray &V, bool aggregate\_LUV=false, size_t offset=0)

- static void \\*Vgen\\_DSset (size_t num_v, RealSetArray &sets, RealVector &L, RealVector &U, RealVector &V, bool aggregate\_LUV=false, size_t offset=0)

- static void \\*Vgen\\_DSset (size_t num_v, RealRealMapArray &vals\_probs, RealVector &IP, RealVector &IP, RealVector &L, RealVector &U, RealVector &V, bool aggregate\_LUV=false, size_t offset=0)

- static void \\*Vgen\\_DSset (size_t num_v, StringRealMapArray &vals\_probs, StringArray &IP, StringArray &IP, StringArray &L, StringArray &U, StringArray &V, bool aggregate\_LUV=false, size_t offset=0)

- static void \\*Vchk\\_Discrete\_DesSet\_Int (DataVariablesRep *dv, size_t offset, Var\_Info *vi)

- static void \\*Vchk\\_Discrete\_DesSet\_Int (DataVariablesRep *dv, size_t offset)

- static void \\*Vchk\\_Discrete\_DesSet\_Str (DataVariablesRep *dv, size_t offset, Var\_Info *vi)

- static void \\*Vchk\\_Discrete\_DesSet\_Real (DataVariablesRep *dv, size_t offset, Var\_Info *vi)

- static void \\*Vchk\\_Discrete\_UncSet\_Int (DataVariablesRep *dv, size_t offset, Var\_Info *vi)

- static void \\*Vchk\\_Discrete\_UncSet\_Str (DataVariablesRep *dv, size_t offset, Var\_Info *vi)

- static void \\*Vchk\\_Discrete\_UncSet\_Real (DataVariablesRep *dv, size_t offset, Var\_Info *vi)

- static void \\*Vchk\\_Discrete\_UncSet\_Str (DataVariablesRep *dv, size_t offset, Var\_Info *vi)

- static void \\*Vchk\\_Discrete\_UncSet\_Real (DataVariablesRep *dv, size_t offset, Var\_Info *vi)

- static void \\*Vchk\\_Discrete\_UncSet\_Real (DataVariablesRep *dv, size_t offset)
For real-valued variables: verify lengths of bounds and initial point, validate bounds and adjust initial point to bounds.
• static void Vchk_DiscreteStateSetInt (DataVariablesRep *dv, size_t offset, Var_Info *vi)
• static void Vchk_DiscreteStateSetInt (DataVariablesRep *dv, size_t offset, Var_Info *vi)
• static void Vchk_DiscreteStateSetStr (DataVariablesRep *dv, size_t offset, Var_Info *vi)
• static void Vchk_DiscreteStateSetStr (DataVariablesRep *dv, size_t offset, Var_Info *vi)
• static void Vchk_DiscreteStateSetReal (DataVariablesRep *dv, size_t offset, Var_Info *vi)
• static void Vchk_DiscreteStateSetReal (DataVariablesRep *dv, size_t offset, Var_Info *vi)
• static void Var_RealBoundIPCheck (DataVariablesRep *dv, Var_rcheck *b)

For integer-valued variables: verify lengths of bounds and initial point, validate bounds and initial point against bounds.
• static void Vgen_DiscreteStateSetInt (DataVariablesRep *dv, size_t offset)
• static void Vgen_DiscreteStateSetInt (DataVariablesRep *dv, size_t offset)
• static void Vgen_DiscreteStateSetStr (DataVariablesRep *dv, size_t offset, Var_Info *vi)
• static void Vgen_DiscreteStateSetStr (DataVariablesRep *dv, size_t offset, Var_Info *vi)
• static void Vgen_DiscreteStateSetReal (DataVariablesRep *dv, size_t offset)
• static void Vgen_DiscreteStateSetReal (DataVariablesRep *dv, size_t offset)
• static void Var_IntBoundIPCheck (DataVariablesRep *dv, Var_jcheck *ib)

Flatten real-valued interval uncertain variable intervals and probabilities back into separate arrays.
• static void flatten_rva (RealVectorArray *rva, RealVector **prv)
• static void flatten_iva (IntVectorArray *iva, IntVector **piv)
• static void flatten_rsm (RealSymMatrix *rsm, RealVector **prv)
• static void flatten_rsa (RealSetArray *rsa, RealVector **prv)
• static void flatten_ssa (StringSetArray **ssa, StringArray **psa)
• static void flatten_isa (IntSetArray *isa, IntVector **piv)
• static void flatten_rrma_keys (RealRealMapArray *rrma, RealVector **prv)
• static void flatten_rrma_values (RealRealMapArray *rrma, RealVector **prv)
• static void flatten_irma_keys (IntRealMapArray *irma, IntVector **piv)
• static void flatten_irma_values (IntRealMapArray *irma, RealVector **prv)
• static void flatten_srma_keys (StringRealMapArray *srma, StringArray **psa)
• static void flatten_srma_values (StringRealMapArray *srma, RealVector **prv)
• static void flatten_real_intervals (const RealRealPairRealMapArray &rrprma, RealVector **probs, RealVector **lb, RealVector **ub)

Flatten integer-valued interval uncertain variable intervals and probabilities back into separate arrays.
• static void var_julbl (const char *keyname, Values *val, VarLabel *vl)
• static Iface_mp_Rlit MP3 (failAction, recoveryFnVals, recover)
• static Iface_mp_ilit MP3 (failAction, retryLimit, retry)
• static Iface_mp_lit MP2 (failAction, abort)
• static Iface_mp_lit MP2 (failAction, continuation)
• static Iface_mp_type MP2s (analysisScheduling, MASTER_SCHEDULING)
• static Iface_mp_type MP2s (analysisScheduling, PEER_SCHEDULING)
• static Iface_mp_type MP2s (evalScheduling, MASTER_SCHEDULING)
• static Iface_mp_type MP2s (evalScheduling, PEER_DYNAMIC_SCHEDULING)
• static Iface_mp_type MP2s (evalScheduling, PEER_STATIC_SCHEDULING)
• static Iface_mp_type MP2s (asynchLocalEvalScheduling, DYNAMIC_SCHEDULING)
• static Iface_mp_type MP2s (asynchLocalEvalScheduling, STATIC_SCHEDULING)
• static Iface_mp_utype MP2s (interfaceType, TEST_INTERFACE)
• static Iface_mp_utype MP2s (interfaceType, FORK_INTERFACE)
• static Iface_mp_utype MP2s (interfaceType, GRID_INTERFACE)
- static Iface_mp_utype MP2s (interfaceType, MATLAB_INTERFACE)
- static Iface_mp_utype MP2s (interfaceType, PYTHON_INTERFACE)
- static Iface_mp_utype MP2s (interfaceType, SCILAB_INTERFACE)
- static Iface_mp_utype MP2s (interfaceType, SYSTEM_INTERFACE)
- static Iface_mp_utype MP2s (resultsFileFormat, LABELED_RESULTS)
- static String MP_ (algebraicMappings)
- static String MP_ (idInterface)
- static String MP_ (inputFilter)
- static String MP_ (outputFilter)
- static String MP_ (parametersFile)
- static String MP_ (resultsFile)
- static String MP_ (workDir)
- static String2DArray MP_ (analysisComponents)
- static StringArray MP_ (analysisDrivers)
- static StringArray MP_ (copyFiles)
- static StringArray MP_ (linkFiles)
- static bool MP_ (activeSetVectorFlag)
- static bool MP_ (allowExistingResultsFlag)
- static bool MP_ (apreproFlag)
- static bool MP_ (asynchFlag)
- static bool MP_ (batchEvalFlag)
- static bool MP_ (dirSave)
- static bool MP_ (dirTag)
- static bool MP_ (evalCacheFlag)
- static bool MP_ (fileSaveFlag)
- static bool MP_ (fileTagFlag)
- static bool MP_ (nearbyEvalCacheFlag)
- static bool MP_ (numpyFlag)
- static bool MP_ (restartFileFlag)
- static bool MP_ (templateReplace)
- static bool MP_ (verbatimFlag)
- static int MP_ (analysisServers)
- static int MP_ (asynchLocalAnalysisConcurrency)
- static int MP_ (asynchLocalEvalConcurrency)
- static int MP_ (evalServers)
- static int MP_ (procsPerAnalysis)
- static int MP_ (procsPerEval)
- static Real MP_ (nearbyEvalCacheTol)
- static IntVector MP_ (primeBase)
- static IntVector MP_ (refineSamples)
- static IntVector MP_ (sequenceLeap)
- static IntVector MP_ (sequenceStart)
- static IntVector MP_ (stepsPerVariable)
- static Method_mp_ilit2 MP3 (replacementType, numberRetained, chc)
- static Method_mp_ilit2 MP3 (replacementType, numberRetained, elitist)
- static Method_mp_lit2 MP3 (replacementType, numberRetained, random)
- static Method_mp_lit2z MP3 (crossoverType, numCrossPoints, multi_point_binary)
- static Method_mp_lit2z MP3 (crossoverType, numCrossPoints, multi_point_parameterized_binary)
- static Method_mp_lit2z MP3 (crossoverType, numCrossPoints, multi_point_real)
- static Method_mp_lit MP2 (batchSelectionType, naive)
- static Method_mp_lit MP2 (batchSelectionType, distance)
- static Method_mp_lit MP2 (batchSelectionType, topology)
- static Method_mp_lit MP2 (batchSelectionType, cl)
- static Method_mp_lit MP2 (boxDivision, all_dimensions)
- static Method_mp_lit MP2 (boxDivision, major_dimension)
- static Method_mp_lit MP2 (convergenceType, average_fitness_tracker)
- static Method_mp_lit MP2 (convergenceType, best_fitness_tracker)
- static Method_mp_lit MP2 (convergenceType, metric_tracker)
- static Method_mp_lit MP2 (crossoverType, blend)
- static Method_mp_lit MP2 (crossoverType, two_point)
- static Method_mp_lit MP2 (crossoverType, uniform)
- static Method_mp_lit MP2 (dataDistCovInputType, diagonal)
- static Method_mp_lit MP2 (dataDistCovInputType, matrix)
- static Method_mp_lit MP2 (evalSynchronize, blocking)
- static Method_mp_lit MP2 (evalSynchronize, nonblocking)
- static Method_mp_lit MP2 (exploratoryMoves, adaptive)
- static Method_mp_lit MP2 (exploratoryMoves, multi_step)
- static Method_mp_lit MP2 (exploratoryMoves, simple)
- static Method_mp_lit MP2 (fitnessType, domination_count)
- static Method_mp_lit MP2 (fitnessType, layer_rank)
- static Method_mp_lit MP2 (fitnessType, linear_rank)
- static Method_mp_lit MP2 (fitnessType, merit_function)
- static Method_mp_lit MP2 (fitnessType, proportional)
- static Method_mp_lit MP2 (fitnessMetricType, predicted_variance)
- static Method_mp_lit MP2 (fitnessMetricType, distance)
- static Method_mp_lit MP2 (fitnessMetricType, gradient)
- static Method_mp_lit MP2 (initializationType, random)
- static Method_mp_lit MP2 (initializationType, unique_random)
- static Method_mp_lit MP2 (lipschitzType, global)
- static Method_mp_lit MP2 (lipschitzType, local)
- static Method_mp_lit MP2 (meritFunction, merit_max)
- static Method_mp_lit MP2 (meritFunction, merit_max_smooth)
- static Method_mp_lit MP2 (meritFunction, merit1)
- static Method_mp_lit MP2 (meritFunction, merit1_smooth)
- static Method_mp_lit MP2 (meritFunction, merit2)
- static Method_mp_lit MP2 (meritFunction, merit2_smooth)
- static Method_mp_lit MP2 (meritFunction, merit2_squared)
- static Method_mp_lit MP2 (mcmcType, adaptive_metropolis)
- static Method_mp_lit MP2 (mcmcType, delayed_rejection)
- static Method_mp_lit MP2 (mcmcType, dram)
- static Method_mp_lit MP2 (mcmcType, metropolis_hastings)
• static Method_mp_lit MP2 (mcmcType, multilevel)
• static Method_mp_lit MP2 (modelDiscrepancyType, global_kriging)
• static Method_mp_lit MP2 (modelDiscrepancyType, global_polynomial)
• static Method_mp_lit MP2 (mutationType, bit_random)
• static Method_mp_lit MP2 (mutationType, offset_cauchy)
• static Method_mp_lit MP2 (mutationType, offset_normal)
• static Method_mp_lit MP2 (mutationType, offset_uniform)
• static Method_mp_lit MP2 (mutationType, replace_uniform)
• static Method_mp_lit MP2 (patternBasis, coordinate)
• static Method_mp_lit MP2 (patternBasis, simplex)
• static Method_mp_lit MP2 (pointReuse, all)
• static Method_mp_lit MP2 (proposalCovInputType, diagonal)
• static Method_mp_lit MP2 (proposalCovInputType, matrix)
• static Method_mp_lit MP2 (proposalCovType, derivatives)
• static Method_mp_lit MP2 (proposalCovType, prior)
• static Method_mp_lit MP2 (proposalCovType, user)
• static Method_mp_lit MP2 (reliabilityIntegration, first_order)
• static Method_mp_lit MP2 (reliabilityIntegration, second_order)
• static Method_mp_lit MP2 (replacementType, elitist)
• static Method_mp_lit MP2 (replacementType, favor_feasible)
• static Method_mp_lit MP2 (replacementType, roulette_wheel)
• static Method_mp_lit MP2 (replacementType, unique_roulette_wheel)
• static Method_mp_lit MP2 (rngName, mt19937)
• static Method_mp_lit MP2 (rngName, rnum2)
• static Method_mp_lit MP2 (searchMethod, gradient_based_line_search)
• static Method_mp_lit MP2 (searchMethod, tr_pds)
• static Method_mp_lit MP2 (searchMethod, trust_region)
• static Method_mp_lit MP2 (searchMethod, value_based_line_search)
• static Method_mp_lit MP2 (trialType, grid)
• static Method_mp_lit MP2 (trialType, halton)
• static Method_mp_lit MP2 (trialType, random)
• static Method_mp_lit MP2 (useSurrogate, inform_search)
• static Method_mp_lit MP2 (useSurrogate, optimize)
• static Method_mp_litc MP3 (crossoverType, crossoverRate, shuffle_random)
• static Method_mp_litc MP3 (crossoverType, crossoverRate, null_crossover)
• static Method_mp_litc MP3 (mutationType, mutationRate, null_mutation)
• static Method_mp_litc MP3 (mutationType, mutationRate, offset_cauchy)
• static Method_mp_litc MP3 (mutationType, mutationRate, offset_normal)
• static Method_mp_litc MP3 (mutationType, mutationRate, offset_uniform)
• static Method_mp_litc MP3 (replacementType, fitnessLimit, below_limit)
• static Method_mp_litrv MP3 (nichingType, nicheVector, distance)
• static Method_mp_litrv MP3 (nichingType, nicheVector, max_designs)
• static Method_mp_litrv MP3 (nichingType, nicheVector, radial)
• static Method_mp_litrv MP3 (postProcessorType, distanceVector, distance_postprocessor)
• static Method_mp_slit2 MP3 (initializationType, flatFile, flat_file)
• static Method_mp_utype_lit MP3s (methodName, dlDetails, DL_SOLVER)
• static Method_mp_ord MP2s (approxCorrectionOrder, 0)
• static Method_mp_ord MP2s (approxCorrectionOrder, 1)
• static Method_mp_ord MP2s (approxCorrectionOrder, 2)
• static Real MP_ (absConvTol)
• static Real MP_ (arithmeticTol)
• static Real MP_ (centeringParam)
• static Real MP_ (collocationRatio)
• static Real MP_ (collocRatioTermsOrder)
• static Real MP_ (constraintPenalty)
• static Real MP_ (constrPenalty)
• static Real MP_ (constraintTolerance)
• static Real MP_ (contractFactor)
• static Real MP_ (contractStepLength)
• static Real MP_ (convergenceTolerance)
• static Real MP_ (crossoverRate)
• static Real MP_ (falseConvTol)
• static Real MP_ (functionPrecision)
• static Real MP_ (globalBalanceParam)
• static Real MP_ (gradientTolerance)
• static Real MP_ (hybridLSProb)
• static Real MP_ (grThreshold)
• static Real MP_ (initDelta)
• static Real MP_ (initMeshSize)
• static Real MP_ (initStepLength)
• static Real MP_ (initTRRadius)
• static Real MP_ (lineSearchTolerance)
• static Real MP_ (localBalanceParam)
• static Real MP_ (maxBoxSize)
• static Real MP_ (maxStep)
• static Real MP_ (minBoxSize)
• static Real MP_ (minMeshSize)
• static Real MP_ (multilevEstimatorRate)
• static Real MP_ (mutationRate)
• static Real MP_ (mutationScale)
• static Real MP_ (percentVarianceExplained)
• static Real MP_ (priorPropCovMult)
• static Real MP_ (refinementRate)
• static Real MP_ (regressionL2Penalty)
• static Real MP_ (roundingTol)
• static Real MP_ (shrinkagePercent)
• static Real MP_ (singConvTol)
• static Real MP_ (singRadius)
• static Real MP_ (smoothFactor)
• static Real MP_ (solnTarget)
• static Real MP_ (solverTol)
• static Real MP_ (stepLenToBoundary)
- static Real MP_(threshDelta)
- static Real MP_(threshStepLength)
- static Real MP_(trustRegionContract)
- static Real MP_(trustRegionContractTrigger)
- static Real MP_(trustRegionExpand)
- static Real MP_(trustRegionExpandTrigger)
- static Real MP_(trustRegionMinSize)
- static Real MP_(vbdDropTolerance)
- static Real MP_(volBoxSize)
- static Real MP_(vns)
- static Real MP_(wilksConfidenceLevel)
- static Real MP_(xConvTol)
- static RealVector MP_(anisoDimPref)
- static RealVector MP_(concurrentParameterSets)
- static RealVector MP_(dataDistCovariance)
- static RealVector MP_(dataDistMeans)
- static RealVector MP_(finalPoint)
- static RealVector MP_(hyperPriorAlphas)
- static RealVector MP_(hyperPriorBetas)
- static RealVector MP_(listOfPoints)
- static RealVector MP_(predictionConfigList)
- static RealVector MP_(proposalCovData)
- static RealVector MP_(regressionNoiseTol)
- static RealVector MP_(stepVector)
- static RealVector MP_(trustRegionInitSize)
- static RealVectorArray MP_(genReliabilityLevels)
- static RealVectorArray MP_(probabilityLevels)
- static RealVectorArray MP_(reliabilityLevels)
- static RealVectorArray MP_(responseLevels)
- static unsigned short MP_(adaptedBasisAdvancements)
- static unsigned short MP_(cubIntOrder)
- static unsigned short MP_(expansionOrder)
- static unsigned short MP_(maxOrder)
- static unsigned short MP_(quadratureOrder)
- static unsigned short MP_(softConvLimit)
- static unsigned short MP_(sparseGridLevel)
- static unsigned short MP_(startOrder)
- static unsigned short MP_(vbdOrder)
- static unsigned short MP_(wilksOrder)
- static SizetArray MP_(collocationPointsSeq)
- static SizetArray MP_(expansionSamplesSeq)
- static SizetArray MP_(pilotSamples)
- static SizetArray MP_(randomSeedSeq)
- static SizetArray MP_(startRankSeq)
- static UShortArray MP_(expansionOrderSeq)
- static UShortArray MP_(quadratureOrderSeq)
• static UShortArray MP_ (sparseGridLevelSeq)
• static UShortArray MP_ (startOrderSeq)
• static UShortArray MP_ (tensorGridOrder)
• static UShortArray MP_ (varPartitions)
• static String MP_ (advancedOptionsFilename)
• static String MP_ (betaSolverName)
• static String MP_ (dataDistFile)
• static String MP_ (displayFormat)
• static String MP_ (exportApproxPtsFile)
• static String MP_ (exportCorrModelFile)
• static String MP_ (exportCorrVarFile)
• static String MP_ (exportDiscrepFile)
• static String MP_ (exportExpansionFile)
• static String MP_ (exportMCMCPtsFile)
• static String MP_ (historyFile)
• static String MP_ (hybridGlobalMethodName)
• static String MP_ (hybridGlobalMethodPointer)
• static String MP_ (hybridGlobalModelPointer)
• static String MP_ (hybridLocalMethodName)
• static String MP_ (hybridLocalMethodPointer)
• static String MP_ (hybridLocalModelPointer)
• static String MP_ (idMethod)
• static String MP_ (importApproxPtsFile)
• static String MP_ (importBuildPtsFile)
• static String MP_ (importCandPtsFile)
• static String MP_ (importExpansionFile)
• static String MP_ (importPredConfig)
• static String MP_ (logFile)
• static String MP_ (lowFidModelPointer)
• static String MP_ (modelPointer)
• static String MP_ (posteriorDensityExportFilename)
• static String MP_ (posteriorSamplesExportFilename)
• static String MP_ (posteriorSamplesImportFilename)
• static String MP_ (proposalCovFile)
• static String MP_ (pstudyFilename)
• static String MP_ (subMethodName)
• static String MP_ (subMethodPointer)
• static String MP_ (subModelPointer)
• static StringArray MP_ (hybridMethodName)
• static StringArray MP_ (hybridMethodPointers)
• static StringArray MP_ (hybridModelPointers)
• static StringArray MP_ (miscOptions)
• static bool MP_ (adaptExpDesign)
• static bool MP_ (adaptPosteriorRefine)
• static bool MP_ (adaptRank)
• static bool MP_ (backfillFlag)
• static bool MP_ (calModelDiscrepancy)
• static bool MP_ (chainDiagnostics)
• static bool MP_ (chainDiagnosticsCI)
• static bool MP_ (constantPenalty)
• static bool MP_ (crossValidation)
• static bool MP_ (crossValidNoiseOnly)
• static bool MP_ (dOptimal)
• static bool MP_ (evaluatePosteriorDensity)
• static bool MP_ (expansionFlag)
• static bool MP_ (exportSampleSeqFlag)
• static bool MP_ (fixedSeedFlag)
• static bool MP_ (fixedSequenceFlag)
• static bool MP_ (generatePosteriorSamples)
• static bool MP_ (gpmsaNormalize)
• static bool MP_ (importApproxActive)
• static bool MP_ (importBuildActive)
• static bool MP_ (latinizeFlag)
• static bool MP_ (logitTransform)
• static bool MP_ (mainEffectsFlag)
• static bool MP_ (methodScaling)
• static bool MP_ (methodUseDerivsFlag)
• static bool MP_ (modelEvidence)
• static bool MP_ (modelEvidLaplace)
• static bool MP_ (modelEvidMC)
• static bool MP_ (mutualInfoKSG2)
• static bool MP_ (mutationAdaptive)
• static bool MP_ (normalizedCoeffs)
• static bool MP_ (pcaFlag)
• static bool MP_ (posteriorStatsKL)
• static bool MP_ (posteriorStatsKDE)
• static bool MP_ (posteriorStatsMutual)
• static bool MP_ (printPopFlag)
• static bool MP_ (pstudyFileActive)
• static bool MP_ (randomizeOrderFlag)
• static bool MP_ (regressDiag)
• static bool MP_ (showAllEval)
• static bool MP_ (showMiscOptions)
• static bool MP_ (speculativeFlag)
• static bool MP_ (standardizedSpace)
• static bool MP_ (useTargetVarianceOptimizationFlag)
• static bool MP_ (tensorGridFlag)
• static bool MP_ (surrBasedGlobalReplacePts)
• static bool MP_ (surrBasedLocalLayerBypass)
• static bool MP_ (vbdFlag)
• static bool MP_ (volQualityFlag)
• static bool MP_ (wilksFlag)
13.2. DAKOTA NAMESPACE REFERENCE

- static short MP_ (expansionType)
- static short MP_ (nestingOverride)
- static short MP_ (refinementType)
- static short MP_ (wilksSidedInterval)
- static int MP_ (batchSize)
- static int MP_ (batchSizeExplore)
- static int MP_ (buildSamples)
- static int MP_ (burnInSamples)
- static int MP_ (chainSamples)
- static int MP_ (concurrentRandomJobs)
- static int MP_ (contractAfterFail)
- static int MP_ (covarianceType)
- static int MP_ (crossoverChainPairs)
- static int MP_ (emulatorOrder)
- static int MP_ (expandAfterSuccess)
- static int MP_ (evidenceSamples)
- static int MP_ (iteratorServers)
- static int MP_ (jumpStep)
- static int MP_ (maxCrossIterations)
- static int MP_ (maxFunctionEvaluations)
- static int MP_ (maxHifiEvals)
- static int MP_ (maxIterations)
- static int MP_ (maxRefineIterations)
- static int MP_ (maxSolverIterations)
- static int MP_ (mutationRange)
- static int MP_ (neighborOrder)
- static int MP_ (newSolnsGenerated)
- static int MP_ (numChains)
- static int MP_ (numCR)
- static int MP_ (numSamples)
- static int MP_ (numSteps)
- static int MP_ (numSymbols)
- static int MP_ (numTrials)
- static int MP_ (populationSize)
- static int MP_ (procsPerIterator)
- static int MP_ (proposalCovUpdatePeriod)
- static int MP_ (numPushforwardSamples)
- static int MP_ (randomSeed)
- static int MP_ (samplesOnEmulator)
- static int MP_ (searchSchemeSize)
- static int MP_ (subSamplingPeriod)
- static int MP_ (totalPatternSize)
- static int MP_ (verifyLevel)
- static size_t MP_ (collocationPoints)
- static size_t MP_ (expansionSamples)
- static size_t MP_ (kickRank)
• static size_t MP_(maxRank)
• static size_t MP_(numCandidateDesigns)
• static size_t MP_(numCandidates)
• static size_t MP_(numDesigns)
• static size_t MP_(numFinalSolutions)
• static size_t MP_(numGenerations)
• static size_t MP_(numOffspring)
• static size_t MP_(numParents)
• static size_t MP_(numPredConfigs)
• static size_t MP_(startRank)
• static Method_mp_type MP2s(allocationTarget, TARGET_MEAN)
• static Method_mp_type MP2s(allocationTarget, TARGET_VARIANCE)
• static Method_mp_type MP2s(covarianceControl, DIAGONAL_COVARIANCE)
• static Method_mp_type MP2s(covarianceControl, FULL_COVARIANCE)
• static Method_mp_type MP2s(distributionType, COMPLEMENTARY)
• static Method_mp_type MP2s(distributionType, CUMULATIVE)
• static Method_mp_type MP2s(emulatorType, GP_EMULATOR)
• static Method_mp_type MP2s(emulatorType, KIRIGING_EMULATOR)
• static Method_mp_type MP2s(emulatorType, MF_PCE_EMULATOR)
• static Method_mp_type MP2s(emulatorType, MF_SC_EMULATOR)
• static Method_mp_type MP2s(emulatorType, ML_PCE_EMULATOR)
• static Method_mp_type MP2s(emulatorType, PCE_EMULATOR)
• static Method_mp_type MP2s(emulatorType, SC_EMULATOR)
• static Method_mp_type MP2s(emulatorType, VPS_EMULATOR)
• static Method_mp_type MP2p(expansionBasisType, ADAPTED_BASIS_EXPANDING_FRONT)
• static Method_mp_type MP2p(expansionBasisType, ADAPTED_BASIS_GENERALIZED)
• static Method_mp_type MP2p(expansionBasisType, HIERARCHICAL_INTERPOLANT)
• static Method_mp_type MP2p(expansionBasisType, NODAL_INTERPOLANT)
• static Method_mp_type MP2p(expansionBasisType, TENSOR_PRODUCT_BASIS)
• static Method_mp_type MP2p(expansionBasisType, TOTAL_ORDER_BASIS)
• static Method_mp_type MP2s(expansionType, ASKEY_U)
• static Method_mp_type MP2s(expansionType, STD_NORMAL_U)
• static Method_mp_type MP2s(finalMomentsType, CENTRAL_MOMENTS)
• static Method_mp_type MP2s(finalMomentsType, NO_MOMENTS)
• static Method_mp_type MP2s(finalMomentsType, STANDARD_MOMENTS)
• static Method_mp_type MP2p(growthOverride, RESTRICTED)
• static Method_mp_type MP2p(growthOverride, UNRESTRICTED)
• static Method_mp_type MP2s(iteratorScheduling, MASTER_SCHEDULING)
• static Method_mp_type MP2s(iteratorScheduling, PEER_SCHEDULING)
• static Method_mp_type MP2s(lsRegressionType, EQ_CON_LS)
• static Method_mp_type MP2s(lsRegressionType, SVD_LS)
• static Method_mp_type MP2o(meritFn, ArgaezTapia)
• static Method_mp_type MP2o(meritFn, NormFmu)
• static Method_mp_type MP2o(meritFn, VanShanno)
• static Method_mp_type MP2s(methodOutput, DEBUG_OUTPUT)
• static Method_mp_type MP2s(methodOutput, NORMAL_OUTPUT)
13.2. DAKOTA NAMESPACE REFERENCE

- static Method_mp_type MP2s (methodOutput, QUIET_OUTPUT)
- static Method_mp_type MP2s (methodOutput, SILENT_OUTPUT)
- static Method_mp_type MP2s (methodOutput, VERBOSE_OUTPUT)
- static Method_mp_type MP2s (multilevAllocControl, ESTIMATOR_VARIANCE)
- static Method_mp_type MP2s (multilevAllocControl, GREEDY_REFINEMENT)
- static Method_mp_type MP2s (multilevAllocControl, RANK_SAMPLING)
- static Method_mp_type MP2s (multilevAllocControl, RIP_SAMPLING)
- static Method_mp_type MP2s (multilevDiscrepEmulation, DISTINCT_EMULATION)
- static Method_mp_type MP2s (multilevDiscrepEmulation, RECURSIVE_EMULATION)
- static Method_mp_type MP2s (nestingOverride, NESTED)
- static Method_mp_type MP2s (nestingOverride, NON_NESTED)
- static Method_mp_type MP2s (qoiAggregation, QOI_AGGREGATION_SUM)
- static Method_mp_type MP2s (qoiAggregation, QOI_AGGREGATION_MAX)
- static Method_mp_type MP2s (refinementControl, DIMENSION_ADAPTIVE_CONTROL_GENERALIZED)
- static Method_mp_type MP2s (refinementControl, DIMENSION_ADAPTIVE_CONTROL_DECAY)
- static Method_mp_type MP2s (refinementControl, DIMENSION_ADAPTIVE_CONTROL_SOBOL)
- static Method_mp_type MP2s (refinementControl, LOCAL_ADAPTIVE_CONTROL)
- static Method_mp_type MP2s (refinementControl, UNIFORM_CONTROL)
- static Method_mp_type MP2s (refinementType, P_REFINEMENT)
- static Method_mp_type MP2s (refinementType, H_REFINEMENT)
- static Method_mp_type MP2s (regressionType, BASIS_PURSUIT)
- static Method_mp_type MP2s (regressionType, BASIS_PURSUIT_DENOISING)
- static Method_mp_type MP2s (regressionType, DEFAULT_LEAST_SQ_REGRESSION)
- static Method_mp_type MP2s (regressionType, LASSO_REGRESSION)
- static Method_mp_type MP2s (regressionType, LEAST_ANGLE_REGRESSION)
- static Method_mp_type MP2s (regressionType, ORTHOG_LEAST_INTERPOLATION)
- static Method_mp_type MP2s (regressionType, ORTHOG_MATCH_PURSUIT)
- static Method_mp_type MP2s (regressionType, FT_LS)
- static Method_mp_type MP2s (regressionType, FT_RLS2)
- static Method_mp_type MP2s (responseLevelTarget, GEN_RELIABILITIES)
- static Method_mp_type MP2s (responseLevelTarget, PROBABILITIES)
- static Method_mp_type MP2s (responseLevelTarget, RELIABILITIES)
- static Method_mp_type MP2s (responseLevelTargetReduce, SYSTEM_PARALLEL)
- static Method_mp_type MP2s (responseLevelTargetReduce, SYSTEM_SERIES)
- static Method_mp_type MP2s (surrBasedLocalAcceptLogic, FILTER)
- static Method_mp_type MP2s (surrBasedLocalAcceptLogic, TR_RATIO)
- static Method_mp_type MP2s (surrBasedLocalConstrRelax, HOMOTOPY)
- static Method_mp_type MP2s (surrBasedLocalMeritFn, ADAPTIVE_PENALTY_MERIT)
- static Method_mp_type MP2s (surrBasedLocalMeritFn, AUGMENTED_LAGRANGIAN_MERIT)
- static Method_mp_type MP2s (surrBasedLocalMeritFn, LAGRANGIAN_MERIT)
- static Method_mp_type MP2s (surrBasedLocalMeritFn, PENALTY_MERIT)
- static Method_mp_type MP2s (surrBasedLocalSubProbCon, LINEARIZED_CONSTRAINTS)
- static Method_mp_type MP2s (surrBasedLocalSubProbCon, NO_CONSTRAINTS)
- static Method_mp_type MP2s (surrBasedLocalSubProbCon, ORIGINAL_CONSTRAINTS)
- static Method_mp_type MP2s (surrBasedLocalSubProbObj, AUGMENTED_LAGRANGIAN_OBJECTIVE)
<table>
<thead>
<tr>
<th>Method Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>MP2s (surrBasedLocalSubProbObj, LAGRANGIAN_OBJECTIVE)</td>
<td>static</td>
</tr>
<tr>
<td>MP2s (surrBasedLocalSubProbObj, ORIGINAL_PRIMARY)</td>
<td>static</td>
</tr>
<tr>
<td>MP2s (surrBasedLocalSubProbObj, SINGLE_OBJECTIVE)</td>
<td>static</td>
</tr>
<tr>
<td>MP2s (wilksSidedInterval, ONE_SIDED_LOWER)</td>
<td>static</td>
</tr>
<tr>
<td>MP2s (wilksSidedInterval, ONE_SIDED_UPPER)</td>
<td>static</td>
</tr>
<tr>
<td>MP2s (wilksSidedInterval, TWO_SIDED)</td>
<td>static</td>
</tr>
<tr>
<td>MP2s (calibrateErrorMode, CALIBRATE_ONE)</td>
<td>utype</td>
</tr>
<tr>
<td>MP2s (calibrateErrorMode, CALIBRATE_PER_EXPER)</td>
<td>utype</td>
</tr>
<tr>
<td>MP2s (calibrateErrorMode, CALIBRATE_PER_RESP)</td>
<td>utype</td>
</tr>
<tr>
<td>MP2s (calibrateErrorMode, CALIBRATE_BOTH)</td>
<td>utype</td>
</tr>
<tr>
<td>MP2s (exportApproxFormat, TABULAR_NONE)</td>
<td>static</td>
</tr>
<tr>
<td>MP2s (exportApproxFormat, TABULAR_HEADER)</td>
<td>static</td>
</tr>
<tr>
<td>MP2s (exportApproxFormat, TABULAR_EVAL_ID)</td>
<td>static</td>
</tr>
<tr>
<td>MP2s (exportApproxFormat, TABULAR_IFACE_ID)</td>
<td>static</td>
</tr>
<tr>
<td>MP2s (exportCorrModelFormat, TABULAR_NONE)</td>
<td>static</td>
</tr>
<tr>
<td>MP2s (exportCorrModelFormat, TABULAR_HEADER)</td>
<td>static</td>
</tr>
<tr>
<td>MP2s (exportCorrModelFormat, TABULAR_EVAL_ID)</td>
<td>static</td>
</tr>
<tr>
<td>MP2s (exportCorrModelFormat, TABULAR_IFACE_ID)</td>
<td>static</td>
</tr>
<tr>
<td>MP2s (exportCorrVarFormat, TABULAR_NONE)</td>
<td>static</td>
</tr>
<tr>
<td>MP2s (exportCorrVarFormat, TABULAR_HEADER)</td>
<td>static</td>
</tr>
<tr>
<td>MP2s (exportCorrVarFormat, TABULAR_EVAL_ID)</td>
<td>static</td>
</tr>
<tr>
<td>MP2s (exportCorrVarFormat, TABULAR_IFACE_ID)</td>
<td>static</td>
</tr>
<tr>
<td>MP2s (exportDiscrepFormat, TABULAR_NONE)</td>
<td>static</td>
</tr>
<tr>
<td>MP2s (exportDiscrepFormat, TABULAR_HEADER)</td>
<td>static</td>
</tr>
<tr>
<td>MP2s (exportDiscrepFormat, TABULAR_EVAL_ID)</td>
<td>static</td>
</tr>
<tr>
<td>MP2s (exportDiscrepFormat, TABULAR_IFACE_ID)</td>
<td>static</td>
</tr>
<tr>
<td>MP2s (exportSamplesFormat, TABULAR_NONE)</td>
<td>static</td>
</tr>
<tr>
<td>MP2s (exportSamplesFormat, TABULAR_HEADER)</td>
<td>static</td>
</tr>
<tr>
<td>MP2s (exportSamplesFormat, TABULAR_EVAL_ID)</td>
<td>static</td>
</tr>
<tr>
<td>MP2s (exportSamplesFormat, TABULAR_IFACE_ID)</td>
<td>static</td>
</tr>
<tr>
<td>MP2s (importApproxFormat, TABULAR_NONE)</td>
<td>static</td>
</tr>
<tr>
<td>MP2s (importApproxFormat, TABULAR_HEADER)</td>
<td>static</td>
</tr>
<tr>
<td>MP2s (importApproxFormat, TABULAR_EVAL_ID)</td>
<td>static</td>
</tr>
<tr>
<td>MP2s (importApproxFormat, TABULAR_IFACE_ID)</td>
<td>static</td>
</tr>
<tr>
<td>MP2s (importBuildFormat, TABULAR_NONE)</td>
<td>static</td>
</tr>
<tr>
<td>MP2s (importBuildFormat, TABULAR_HEADER)</td>
<td>static</td>
</tr>
<tr>
<td>MP2s (importBuildFormat, TABULAR_EVAL_ID)</td>
<td>static</td>
</tr>
<tr>
<td>MP2s (importBuildFormat, TABULAR_IFACE_ID)</td>
<td>static</td>
</tr>
<tr>
<td>MP2s (importCandFormat, TABULAR_NONE)</td>
<td>static</td>
</tr>
</tbody>
</table>
• static Method_mp_utype **MP2s** (importCandFormat, TABULAR_HEADER)
• static Method_mp_utype **MP2s** (importCandFormat, TABULAR_EVAL_ID)
• static Method_mp_utype **MP2s** (importCandFormat, TABULAR_IFACE_ID)
• static Method_mp_utype **MP2s** (importCandFormat, TABULAR_ANNOTATED)
• static Method_mp_utype **MP2s** (importPredConfigFormat, TABULAR_NONE)
• static Method_mp_utype **MP2s** (importPredConfigFormat, TABULAR_HEADER)
• static Method_mp_utype **MP2s** (importPredConfigFormat, TABULAR_EVAL_ID)
• static Method_mp_utype **MP2s** (importPredConfigFormat, TABULAR_IFACE_ID)
• static Method_mp_utype **MP2s** (importPredConfigFormat, TABULAR_ANNOTATED)
• static Method_mp_utype **MP2s** (integrationRefine, AIS)
• static Method_mp_utype **MP2s** (integrationRefine, IS)
• static Method_mp_utype **MP2s** (integrationRefine, MMAIS)
• static Method_mp_utype **MP2s** (methodName, ASYNCH_PATTERN_SEARCH)
• static Method_mp_utype **MP2s** (methodName, BRANCH_AND_BOUND)
• static Method_mp_utype **MP2s** (methodName, C3_FUNCTION_TRAIN)
• static Method_mp_utype **MP2s** (methodName, COLINY_BETA)
• static Method_mp_utype **MP2s** (methodName, COLINY_COBYLA)
• static Method_mp_utype **MP2s** (methodName, COLINY_DIRECT)
• static Method_mp_utype **MP2s** (methodName, COLINY_EA)
• static Method_mp_utype **MP2s** (methodName, COLINY_PATTERN_SEARCH)
• static Method_mp_utype **MP2s** (methodName, COLINY_SOLIS_WETS)
• static Method_mp_utype **MP2s** (methodName, CONMIN_FRCG)
• static Method_mp_utype **MP2s** (methodName, CONMIN_MFD)
• static Method_mp_utype **MP2s** (methodName, DACE)
• static Method_mp_utype **MP2s** (methodName, DATA_FIT_SURROGATE_BASED_LOCAL)
• static Method_mp_utype **MP2s** (methodName, DOT_BFGS)
• static Method_mp_utype **MP2s** (methodName, DOT_FRCG)
• static Method_mp_utype **MP2s** (methodName, DOT_MMFD)
• static Method_mp_utype **MP2s** (methodName, DOT_SLP)
• static Method_mp_utype **MP2s** (methodName, DOT_SQP)
• static Method_mp_utype **MP2s** (methodName, EFFICIENT_GLOBAL)
• static Method_mp_utype **MP2s** (methodName, FSU_CVT)
• static Method_mp_utype **MP2s** (methodName, FSU_HALTON)
• static Method_mp_utype **MP2s** (methodName, FSU_HAMMERSLEY)
• static Method_mp_utype **MP2s** (methodName, HIERARCH_SURROGATE_BASED_LOCAL)
• static Method_mp_utype **MP2s** (methodName, HYBRID)
• static Method_mp_utype **MP2s** (methodName, MESH_ADAPTIVE_SEARCH)
• static Method_mp_utype **MP2s** (methodName, MOGA)
• static Method_mp_utype **MP2s** (methodName, MULTI_START)
• static Method_mp_utype **MP2s** (methodName, NCSU_DIRECT)
• static Method_mp_utype **MP2s** (methodName, ROL)
• static Method_mp_utype **MP2s** (methodName, DEMO_TPL)
• static Method_mp_utype **MP2s** (methodName, NL2SOL)
• static Method_mp_utype **MP2s** (methodName, NLPQL_SQP)
• static Method_mp_utype **MP2s** (methodName, NLSSOL_SQP)
• static Method_mp_utype **MP2s** (methodName, MIT_NOWPAC)
• static Method\_mp\_u\_type MP2s (methodName, MIT\_SNOWPAC)
• static Method\_mp\_u\_type MP2s (methodName, ADAPTIVE\_SAMPLING)
• static Method\_mp\_u\_type MP2s (methodName, BAYES\_CALIBRATION)
• static Method\_mp\_u\_type MP2s (methodName, GENIE\_DIRECT)
• static Method\_mp\_u\_type MP2s (methodName, GENIE\_OPT\_DARTS)
• static Method\_mp\_u\_type MP2s (methodName, GPAIS)
• static Method\_mp\_u\_type MP2s (methodName, GLOBAL\_EVIDENCE)
• static Method\_mp\_u\_type MP2s (methodName, GLOBAL\_INTERVAL\_EST)
• static Method\_mp\_u\_type MP2s (methodName, GLOBAL\_RELIABILITY)
• static Method\_mp\_u\_type MP2s (methodName, IMPORTANCE\_SAMPLING)
• static Method\_mp\_u\_type MP2s (methodName, LOCAL\_EVIDENCE)
• static Method\_mp\_u\_type MP2s (methodName, LOCAL\_INTERVAL\_EST)
• static Method\_mp\_u\_type MP2s (methodName, LOCAL\_RELIABILITY)
• static Method\_mp\_u\_type MP2s (methodName, MULTIFIDELITY\_FUNCTION\_TRAIN)
• static Method\_mp\_u\_type MP2s (methodName, MULTIFIDELITY\_POLYNOMIAL\_CHAOS)
• static Method\_mp\_u\_type MP2s (methodName, MULTIFIDELITY\_STOCH\_COLLOCATION)
• static Method\_mp\_u\_type MP2s (methodName, MULTILEVEL\_FUNCTION\_TRAIN)
• static Method\_mp\_u\_type MP2s (methodName, MULTILEVEL\_POLYNOMIAL\_CHAOS)
• static Method\_mp\_u\_type MP2s (methodName, POF\_DARTS)
• static Method\_mp\_u\_type MP2s (methodName, RKD\_DARTS)
• static Method\_mp\_u\_type MP2s (methodName, POLYNOMIAL\_CHAOS)
• static Method\_mp\_u\_type MP2s (methodName, STOCH\_COLLOCATION)
• static Method\_mp\_u\_type MP2s (methodName, SURROGATE\_BASED\_UQ)
• static Method\_mp\_u\_type MP2s (methodName, RANDOM\_SAMPLING)
• static Method\_mp\_u\_type MP2s (methodName, NONLINEAR\_CG)
• static Method\_mp\_u\_type MP2s (methodName, NPSOL\_SQP)
• static Method\_mp\_u\_type MP2s (methodName, OPTPP\_CG)
• static Method\_mp\_u\_type MP2s (methodName, OPTPP\_FD\_NEWTON)
• static Method\_mp\_u\_type MP2s (methodName, OPTPP\_G\_NEWTON)
• static Method\_mp\_u\_type MP2s (methodName, OPTPP\_NEWTON)
• static Method\_mp\_u\_type MP2s (methodName, OPTPP\_PDS)
• static Method\_mp\_u\_type MP2s (methodName, OPTPP\_Q\_NEWTON)
• static Method\_mp\_u\_type MP2s (methodName, PARETO\_SET)
• static Method\_mp\_u\_type MP2s (methodName, PSUADE\_MOAT)
• static Method\_mp\_u\_type MP2s (methodName, RICHARDSON\_EXTRAP)
• static Method\_mp\_u\_type MP2s (methodName, SOGA)
• static Method\_mp\_u\_type MP2s (methodName, SURROGATE\_BASED\_GLOBAL)
• static Method\_mp\_u\_type MP2s (methodName, SURROGATE\_BASED\_LOCAL)
• static Method\_mp\_u\_type MP2s (methodName, VECTOR\_PARAMETER\_STUDY)
• static Method\_mp\_u\_type MP2s (methodName, LIST\_PARAMETER\_STUDY)
• static Method\_mp\_u\_type MP2s (methodName, CENTERED\_PARAMETER\_STUDY)
• static Method\_mp\_u\_type MP2s (methodName, MULTIDIM\_PARAMETER\_STUDY)
• static Method\_mp\_u\_type MP2s (preSolveMethod, SUBMETHOD\_NIP)
• static Method\_mp\_u\_type MP2s (preSolveMethod, SUBMETHOD\_NONE)
• static Method\_mp\_u\_type MP2s (preSolveMethod, SUBMETHOD\_SQP)
• static Method\_mp\_utype MP2s (pstudyFileFormat, TABULAR\_NONE)
• static Method\_mp\_utype MP2s (pstudyFileFormat, TABULAR\_HEADER)
• static Method\_mp\_utype MP2s (pstudyFileFormat, TABULAR\_EVAL\_ID)
• static Method\_mp\_utype MP2s (pstudyFileFormat, TABULAR\_INTERFACE\_ID)
• static Method\_mp\_utype MP2s (pstudyFileFormat, TABULAR\_ANNOTATED)
• static Method\_mp\_utype MP2s (reliabilitySearchType, AMV\_PLUS\_U)
• static Method\_mp\_utype MP2s (reliabilitySearchType, AMV\_PLUS\_X)
• static Method\_mp\_utype MP2s (reliabilitySearchType, AMV\_U)
• static Method\_mp\_utype MP2s (reliabilitySearchType, AMV\_X)
• static Method\_mp\_utype MP2s (reliabilitySearchType, EGRA\_U)
• static Method\_mp\_utype MP2s (reliabilitySearchType, EGRA\_X)
• static Method\_mp\_utype MP2s (reliabilitySearchType, NO\_APPROX)
• static Method\_mp\_utype MP2s (reliabilitySearchType, QMEA\_U)
• static Method\_mp\_utype MP2s (reliabilitySearchType, QMEA\_X)
• static Method\_mp\_utype MP2s (sampleType, SUBMETHOD\_LHS)
• static Method\_mp\_utype MP2s (sampleType, SUBMETHOD\_RANDOM)
• static Method\_mp\_utype MP2s (subMethod, SUBMETHOD\_COLLABORATIVE)
• static Method\_mp\_utype MP2s (subMethod, SUBMETHOD\_EMBEDDED)
• static Method\_mp\_utype MP2s (subMethod, SUBMETHOD\_SEQUENTIAL)
• static Method\_mp\_utype MP2s (subMethod, SUBMETHOD\_MUQ)
• static Method\_mp\_utype MP2s (subMethod, SUBMETHOD\_DREAM)
• static Method\_mp\_utype MP2s (subMethod, SUBMETHOD\_WASABI)
• static Method\_mp\_utype MP2s (subMethod, SUBMETHOD\_GPMSA)
• static Method\_mp\_utype MP2s (subMethod, SUBMETHOD\_QUESO)
• static Method\_mp\_utype MP2s (subMethod, SUBMETHOD\_NIP)
• static Method\_mp\_utype MP2s (subMethod, SUBMETHOD\_SQP)
• static Method\_mp\_utype MP2s (subMethod, SUBMETHOD\_EA)
• static Method\_mp\_utype MP2s (subMethod, SUBMETHOD\_EGO)
• static Method\_mp\_utype MP2s (subMethod, SUBMETHOD\_SBO)
• static Method\_mp\_utype MP2s (subMethod, SUBMETHOD\_LHS)
• static Method\_mp\_utype MP2s (subMethod, SUBMETHOD\_RANDOM)
• static Method\_mp\_utype MP2s (subMethod, SUBMETHOD\_OA\_LHS)
• static Method\_mp\_utype MP2s (subMethod, SUBMETHOD\_OAS)
• static Method\_mp\_utype MP2s (subMethod, SUBMETHOD\_BOX\_BEHNKEN)
• static Method\_mp\_utype MP2s (subMethod, SUBMETHOD\_CENTRAL\_COMPOSITE)
• static Method\_mp\_utype MP2s (subMethod, SUBMETHOD\_GRID)
• static Method\_mp\_utype MP2s (subMethod, SUBMETHOD\_CONVERGE\_ORDER)
• static Method\_mp\_utype MP2s (subMethod, SUBMETHOD\_CONVERGE\_QOI)
• static Method\_mp\_utype MP2s (subMethod, SUBMETHOD\_ESTIMATE\_ORDER)
• static IntSet MP\_ (surrogateFnIndices)
• static Model\_mp\_lit MP2 (approxPointReuse, all)
• static Model\_mp\_lit MP2 (approxPointReuse, none)
• static Model\_mp\_lit MP2 (approxPointReuse, region)
• static Model\_mp\_lit MP2 (marsInterpolation, linear)
• static Model_mp_lit MP2 (marsInterpolation, cubic)
• static Model_mp_lit MP2 (modelType, active_subspace)
• static Model_mp_lit MP2 (modelType, adapted_basis)
• static Model_mp_lit MP2 (modelType, nested)
• static Model_mp_lit MP2 (modelType, random_field)
• static Model_mp_lit MP2 (modelType, simulation)
• static Model_mp_lit MP2 (modelType, surrogate)
• static Model_mp_lit MP2 (surrogateType, hierarchical)
• static Model_mp_lit MP2 (surrogateType, global_exp_gauss_proc)
• static Model_mp_lit MP2 (surrogateType, global_exp_poly)
• static Model_mp_lit MP2 (surrogateType, global_function_train)
• static Model_mp_lit MP2 (surrogateType, global_gaussian)
• static Model_mp_lit MP2 (surrogateType, global_kriging)
• static Model_mp_lit MP2 (surrogateType, global_mars)
• static Model_mp_lit MP2 (surrogateType, global_moving_least_squares)
• static Model_mp_lit MP2 (surrogateType, global_neural_network)
• static Model_mp_lit MP2 (surrogateType, global_polynomial)
• static Model_mp_lit MP2 (surrogateType, global_radial_basis)
• static Model_mp_lit MP2 (surrogateType, global_voronoi_surrogate)
• static Model_mp_lit MP2 (surrogateType, local_taylor)
• static Model_mp_lit MP2 (surrogateType, multipoint_qmea)
• static Model_mp_lit MP2 (surrogateType, multipoint_tana)
• static Model_mp_lit MP2 (trendOrder, constant)
• static Model_mp_lit MP2 (trendOrder, linear)
• static Model_mp_lit MP2 (trendOrder, reduced_quadratic)
• static Model_mp_lit MP2 (trendOrder, quadratic)
• static Model_mp_ord MP2s (polynomialOrder, 1)
• static Model_mp_ord MP2s (polynomialOrder, 2)
• static Model_mp_ord MP2s (polynomialOrder, 3)
• static Model_mp_type MP2s (approxCorrectionType, ADDITIVE_CORRECTION)
• static Model_mp_type MP2s (approxCorrectionType, COMBINED_CORRECTION)
• static Model_mp_type MP2s (approxCorrectionType, MULTIPLICATIVE_CORRECTION)
• static Model_mp_type MP2s (pointsManagement, MINIMUM_POINTS)
• static Model_mp_type MP2s (pointsManagement, RECOMMENDED_POINTS)
• static Model_mp_type MP2s (subMethodScheduling, MASTER_SCHEDULING)
• static Model_mp_type MP2s (subMethodScheduling, PEER_SCHEDULING)
• static Model_mp_utope MP2s (analyticCovIdForm, EXP_L2)
• static Model_mp_utope MP2s (analyticCovIdForm, EXP_L1)
• static Model_mp_utope MP2s (importChallengeFormat, TABULAR_NONE)
• static Model_mp_utope MP2s (importChallengeFormat, TABULAR_HEADER)
• static Model_mp_utope MP2s (importChallengeFormat, TABULAR_EVAL_ID)
• static Model_mp_utope MP2s (importChallengeFormat, TABULAR_IFACE_ID)
• static Model_mp_utope MP2s (importChallengeFormat, TABULAR_ANNOTATED)
• static Model_mp_utope MP2s (modelExportFormat, TEXT_ARCHIVE)
• static Model_mp_utope MP2s (modelExportFormat, BINARY_ARCHIVE)
• static Model_mp_utope MP2s (modelExportFormat, ALGEBRAIC_FILE)
static Model_mptype MP2s (modelExportFormat, ALGEBRAIC_CONSOLE)
static Model_mptype MP2s (randomFieldIdForm, RF_KARHUNEN_LOEVE)
static Model_mptype MP2s (randomFieldIdForm, RF_PCA_GP)
static Model_mptype MP2s (subspaceNormalization, SUBSPACE_NORM_MEAN_VALUE)
static Model_mptype MP2s (subspaceNormalization, SUBSPACE_NORM_MEAN_GRAD)
static Model_mptype MP2s (subspaceNormalization, SUBSPACE_NORM_LOCAL_GRAD)
static Model_mptype MP2s (subspaceSampleType, SUBMETHOD_LHS)
static Model_mptype MP2s (subspaceSampleType, SUBMETHOD_RANDOM)
static Model_mptype MP2s (subspaceIdCVMethod, MINIMUM_METRIC)
static Model_mptype MP2s (subspaceIdCVMethod, RELATIVE_TOLERANCE)
static Model_mptype MP2s (subspaceIdCVMethod, DECREASE_TOLERANCE)
static Real mptype (adaptedBasisCollocRatio)
static Real mptype (decreaseTolerance)
static Real mptype (discontGradThresh)
static Real mptype (discontJumpThresh)
static Real mptype (krigingNugget)
static Real mptype (percentFold)
static Real mptype (relTolerance)
static Real mptype (truncationTolerance)
static RealVector mptype (krigingCorrelations)
static RealVector mptype (primaryRespCoeffs)
static RealVector mptype (secondaryRespCoeffs)
static RealVector mptype (solutionLevelCost)
static String mptype (actualModelPointer)
static String mptype (decompCellType)
static String mptype (idModel)
static String mptype (importChallengePtsFile)
static String mptype (interfacePointer)
static String mptype (krigingOptMethod)
static String mptype (modelExportPrefix)
static String mptype (optionalInterfRespPointer)
static String mptype (propagationModelPointer)
static String mptype (refineCVMetric)
static String mptype (responsesPointer)
static String mptype (rfDataFileName)
static String mptype (solutionLevelControl)
static String mptype (variablesPointer)
static StringArray mptype (diagMetrics)
static StringArray mptype (orderedModelPointers)
static StringArray mptype (primaryVarMaps)
static StringArray mptype (secondaryVarMaps)
static bool mptype (autoRefine)
static bool mptype (crossValidateFlag)
static bool mptype (decompDiscontDetect)
static bool mptype (exportSurrogate)
• static bool MP_ (hierarchicalTags)
• static bool MP_ (identityRespMap)
• static bool MP_ (importChallengeActive)
• static bool MP_ (importChalUseVariableLabels)
• static bool MP_ (importUseVariableLabels)
• static bool MP_ (modelUseDerivsFlag)
• static bool MP_ (domainDecomp)
• static bool MP_ (pointSelection)
• static bool MP_ (pressFlag)
• static bool MP_ (subspaceIdBingLi)
• static bool MP_ (subspaceIdConstantine)
• static bool MP_ (subspaceIdEnergy)
• static bool MP_ (subspaceBuildSurrogate)
• static bool MP_ (subspaceIdCV)
• static bool MP_ (subspaceCVIncremental)
• static unsigned short MP_ (adaptedBasisSparseGridLev)
• static unsigned short MP_ (adaptedBasisExpOrder)
• static short MP_ (annNodes)
• static short MP_ (annRandomWeight)
• static short MP_ (krigingFindNugget)
• static short MP_ (krigingMaxTrials)
• static short MP_ (marsMaxBases)
• static short MP_ (mlsWeightFunction)
• static short MP_ (polynomialOrder)
• static short MP_ (rbfBases)
• static short MP_ (rbfMaxPts)
• static short MP_ (rbfMaxSubsets)
• static short MP_ (rbfMinPartition)
• static int MP_ (decompSupportLayers)
• static int MP_ (initialSamples)
• static int MP_ (maxFunctionEvals)
• static int MP_ (numFolds)
• static int MP_ (numReplicates)
• static int MP_ (numRestarts)
• static int MP_ (pointsTotal)
• static int MP_ (refineCVFolds)
• static int MP_ (softConvergenceLimit)
• static int MP_ (subMethodProcs)
• static int MP_ (subMethodServers)
• static int MP_ (subspaceDimension)
• static int MP_ (subspaceCVMaxRank)
• static IntSet MP_ (idAnalyticGrads)
• static IntSet MP_ (idAnalyticHessians)
• static IntSet MP_ (idNumericalGrads)
• static IntSet MP_ (idNumericalHessians)
• static IntSet MP_ (idQuasiHessians)
13.2. DAKOTA NAMESPACE REFERENCE

- static IntVector MP_ (fieldLengths)
- static IntVector MP_ (numCoordsPerField)
- static RealVector MP_ (expConfigVars)
- static RealVector MP_ (expObservations)
- static RealVector MP_ (primaryRespFnWeights)
- static RealVector MP_ (nonlinearEqTargets)
- static RealVector MP_ (nonlinearIneqLowerBnds)
- static RealVector MP_ (nonlinearIneqUpperBnds)
- static RealVector MP_ (simVariance)
- static RealVector MP_ (fdGradStepSize)
- static RealVector MP_ (fdHessStepSize)
- static RealVector MP_ (primaryRespFnScales)
- static RealVector MP_ (nonlinearEqScales)
- static RealVector MP_ (nonlinearIneqScales)
- static Resp_mp_lit MP2 (gradientType, analytic)
- static Resp_mp_lit MP2 (gradientType, mixed)
- static Resp_mp_lit MP2 (gradientType, none)
- static Resp_mp_lit MP2 (gradientType, numerical)
- static Resp_mp_lit MP2 (hessianType, analytic)
- static Resp_mp_lit MP2 (hessianType, mixed)
- static Resp_mp_lit MP2 (hessianType, none)
- static Resp_mp_lit MP2 (hessianType, numerical)
- static Resp_mp_lit MP2 (hessianType, quasi)
- static Resp_mp_lit MP2 (intervalType, central)
- static Resp_mp_lit MP2 (intervalType, forward)
- static Resp_mp_lit MP2 (methodSource, dakota)
- static Resp_mp_lit MP2 (methodSource, vendor)
- static Resp_mp_lit MP2 (fdGradStepType, absolute)
- static Resp_mp_lit MP2 (fdGradStepType, bounds)
- static Resp_mp_lit MP2 (fdGradStepType, relative)
- static Resp_mp_lit MP2 (fdHessStepType, absolute)
- static Resp_mp_lit MP2 (fdHessStepType, bounds)
- static Resp_mp_lit MP2 (fdHessStepType, relative)
- static Resp_mp_lit MP2 (quasiHessianType, bfgs)
- static Resp_mp_lit MP2 (quasiHessianType, damped_bfgs)
- static Resp_mp_lit MP2 (quasiHessianType, sr1)
- static String MP_ (scalarDataFileName)
- static String MP_ (idResponses)
- static StringArray MP_ (nonlinearEqScaleTypes)
- static StringArray MP_ (nonlinearIneqScaleTypes)
- static StringArray MP_ (primaryRespFnScaleTypes)
- static StringArray MP_ (primaryRespFnSense)
- static StringArray MP_ (responseLabels)
- static bool MP_ (varianceType)
- static bool MP_ (calibrationDataFlag)
- static bool MP_ (centralHess)
• static bool MP_ (interpolateFlag)
• static bool MP_ (ignoreBounds)
• static bool MP_ (readFieldCoords)
• static size_t MP_ (numExpConfigVars)
• static size_t MP_ (numExperiments)
• static size_t MP_ (numFieldLeastSqTerms)
• static size_t MP_ (numFieldObjectiveFunctions)
• static size_t MP_ (numFieldResponseFunctions)
• static size_t MP_ (numLeastSqTerms)
• static size_t MP_ (numNonlinearEqConstraints)
• static size_t MP_ (numNonlinearIneqConstraints)
• static size_t MP_ (numObjectiveFunctions)
• static size_t MP_ (numResponseFunctions)
• static size_t MP_ (numScalarLeastSqTerms)
• static size_t MP_ (numScalarObjectiveFunctions)
• static size_t MP_ (numScalarResponseFunctions)
• static RespType MP2s (scalarDataFormat, TABULAR_NONE)
• static RespType MP2s (scalarDataFormat, TABULAR_HEADER)
• static EnvType MP2s (preRunOutputFormat, TABULAR_EVAL_ID)
• static EnvType MP2s (postRunInputFormat, TABULAR_HEADER)
• static EnvType MP2s (preRunOutputFormat, TABULAR_EVAL_ID)
• static EnvType MP2s (postRunInputFormat, TABULAR_EVAL_ANNOT)
• static EnvType MP2s (preRunOutputFormat, TABULAR_EVAL_ANNOT)
• static RespType MP2s (tabularFormat, TABULAR_NONE)
• static RespType MP2s (tabularFormat, TABULAR_HEADER)
• static EnvType MP2s (preRunOutputFormat, TABULAR_EVAL_ANNOT)
• static EnvType MP2s (tabularFormat, TABULAR_ANNOTATED)
• static EnvType MP2s (resultsOutputFormat, RESULTS_OUTPUT_TEXT)
• static EnvType MP2s (resultsOutputFormat, RESULTS_OUTPUT_HDF5)
• static EnvType MP2s (modelEvalsSelection, MODEL_EVAL_STORE_TOP_METHOD)
• static EnvType MP2s (modelEvalsSelection, MODEL_EVAL_STORE_NONE)
• static EnvType MP2s (modelEvalsSelection, MODEL_EVAL_STORE_ALL)
• static EnvType MP2s (modelEvalsSelection, MODEL_EVAL_STORE_ALL_METHODS)
• static EnvType MP2s (interfEvalsSelection, INTERF_EVAL_STORE_SIMULATION)
• static EnvType MP2s (interfEvalsSelection, INTERF_EVAL_STORE_NONE)
• static EnvType MP2s (interfEvalsSelection, INTERF_EVAL_STORE_ALL)
• static String MP_ (errorFile)
• static String MP_ (outputFile)
• static String MP_ (postRunInput)
• static String MP_ (postRunOutput)
• static String MP_ (preRunInput)
• static String MP_ (preRunOutput)
• static String MP_ (readRestart)
• static String MP_ (resultsOutputFile)
• static String MP_ (runInput)
• static String MP_ (runOutput)
• static String MP_ (tabularDataFile)
• static String MP_ (topMethodPointer)
• static String MP_ (writeRestart)
• static bool MP_ (checkFlag)
• static bool MP_ (graphicsFlag)
• static bool MP_ (postRunFlag)
• static bool MP_ (preRunFlag)
• static bool MP_ (resultsOutputFlag)
• static bool MP_ (runFlag)
• static bool MP_ (tabularDataFlag)
• static int MP_ (outputPrecision)
• static int MP_ (stopRestart)
• static size_t MP_ (numBetaUncVars)
• static size_t MP_ (numBinomialUncVars)
• static size_t MP_ (numContinuousDesVars)
• static size_t MP_ (numContinuousIntervalUncVars)
• static size_t MP_ (numContinuousStateVars)
• static size_t MP_ (numDiscreteDesRangeVars)
• static size_t MP_ (numDiscreteDesSetIntVars)
• static size_t MP_ (numDiscreteDesSetStrVars)
• static size_t MP_ (numDiscreteDesSetRealVars)
• static size_t MP_ (numDiscreteIntervalUncVars)
• static size_t MP_ (numDiscreteStateRangeVars)
• static size_t MP_ (numDiscreteStateSetIntVars)
• static size_t MP_ (numDiscreteStateSetStrVars)
• static size_t MP_ (numDiscreteStateSetRealVars)
• static size_t MP_ (numExponentialUncVars)
• static size_t MP_ (numFrechetUncVars)
• static size_t MP_ (numGammaUncVars)
• static size_t MP_ (numGeometricUncVars)
• static size_t MP_ (numGumbelUncVars)
• static size_t MP_ (numHistogramBinUncVars)
• static size_t MP_ (numHistogramPtIntUncVars)
• static size_t MP_ (numHistogramPtStrUncVars)
• static size_t MP_ (numHistogramPtRealUncVars)
- static size_t MP_ (numHyperGeomUncVars)
- static size_t MP_ (numLognormalUncVars)
- static size_t MP_ (numLoguniformUncVars)
- static size_t MP_ (numNegBinomialUncVars)
- static size_t MP_ (numNormalUncVars)
- static size_t MP_ (numPoissonUncVars)
- static size_t MP_ (numTriangularUncVars)
- static size_t MP_ (numUniformUncVars)
- static size_t MP_ (numWeibullUncVars)
- static IntVector VP_ (ddsi)
- static IntVector VP_ (DIlb)
- static IntVector MP_ (discreteDesignRangeLowerBnds)
- static IntVector MP_ (discreteDesignRangeUpperBnds)
- static IntVector MP_ (discreteDesignRangeVars)
- static IntVector MP_ (discreteDesignSetIntVars)
- static IntVector MP_ (discreteIntervalUncVars)
- static IntVector MP_ (discreteStateRangeLowerBnds)
- static IntVector MP_ (discreteStateRangeUpperBnds)
- static IntVector MP_ (discreteStateRangeVars)
- static IntVector MP_ (discreteStateSetIntVars)
- static IntVector VP_ (DIub)
- static IntVector MP_ (histogramPointIntUncVars)
- static IntVector VP_ (hpia)
- static IntVector VP_ (dssia)
- static IntVector VP_ (ddssa)
- static IntVector VP_ (ddsra)
- static IntVector VP_ (dusi)
- static IntVector VP_ (nddsi)
- static IntVector VP_ (nddss)
- static IntVector VP_ (nddsr)
- static IntVector VP_ (nddssi)
- static IntVector VP_ (nddss)
- static IntVector VP_ (nddssr)
- static IntVector VP_ (ndusi)
- static IntVector VP_ (nhbp)
- static IntVector VP_ (nhpip)
- static IntVector VP_ (nhpsp)
- static IntVector VP_ (nhprp)
- static IntVector VP_ (nCI)
- static IntVector VP_ (nDI)
- static RealVector MP_ (betaUncLowerBnds)
- static RealVector MP_ (betaUncUpperBnds)
• static RealVector MP_ (betaUncVars)
• static RealVector MP_ (binomialUncProbPerTrial)
• static RealVector MP_ (continuousDesignLowerBnds)
• static RealVector MP_ (continuousDesignUpperBnds)
• static RealVector MP_ (continuousDesignVars)
• static RealVector MP_ (continuousDesignScales)
• static RealVector MP_ (continuousIntervalUncVars)
• static RealVector MP_ (continuousStateLowerBnds)
• static RealVector MP_ (continuousStateUpperBnds)
• static RealVector MP_ (continuousStateVars)
• static RealVector MP_ (discreteDesignSetRealVars)
• static RealVector MP_ (discreteStateSetRealVars)
• static RealVector MP_ (discreteUncSetRealVars)
• static RealVector MP_ (frechetUncBetas)
• static RealVector MP_ (frechetUncVars)
• static RealVector MP_ (geometricUncProbPerTrial)
• static RealVector MP_ (gumbelUncBetas)
• static RealVector MP_ (gumbelUncVars)
• static RealVector MP_ (histogramBinUncVars)
• static RealVector MP_ (histogramPointRealUncVars)
• static RealVector MP_ (linearEqConstraintCoeffs)
• static RealVector MP_ (linearEqScales)
• static RealVector MP_ (linearEqTargets)
• static RealVector MP_ (linearIneqConstraintCoeffs)
• static RealVector MP_ (linearIneqLowerBnds)
• static RealVector MP_ (linearIneqUpperBnds)
• static RealVector MP_ (linearIneqScales)
• static RealVector MP_ (negBinomialUncProbPerTrial)
• static RealVector MP_ (normalUncLowerBnds)
• static RealVector MP_ (normalUncMeans)
• static RealVector MP_ (normalUncUpperBnds)
• static RealVector MP_ (normalUncVars)
• static RealVector MP_ (triangularUncModes)
• static RealVector MP_ (triangularUncVars)
• static RealVector MP_ (uniformUncVars)
• static RealVector MP_ (weibullUncVars)
• static RealVector VP_ (ddsr)
• static RealVector VP_ (dssr)
• static RealVector VP_ (dusr)
• static RealVector VP_ (CIlb)
• static RealVector VP_ (CIub)
• static RealVector VP_ (CIp)
• static RealVector VP_ (DIP)
• static RealVector VP_ (DSIp)
• static RealVector VP_ (DSSp)
• static RealVector VP_ (DSRp)
• static RealVector VP_ (hba)
• static RealVector VP_ (hbo)
• static RealVector VP_ (hbc)
• static RealVector VP_ (hpic)
• static RealVector VP_ (hpsc)
• static RealVector VP_ (hpra)
• static RealVector VP_ (hprrc)
• static RealVector VP_ (ucm)
• static String MP_ (idVariables)
• static StringArray MP_ (continuousDesignLabels)
• static StringArray MP_ (continuousDesignScaleTypes)
• static StringArray MP_ (continuousStateLabels)
• static StringArray MP_ (discreteDesignRangeLabels)
• static StringArray MP_ (discreteDesignSetIntLabels)
• static StringArray MP_ (discreteDesignSetStrLabels)
• static StringArray MP_ (discreteDesignSetRealLabels)
• static StringArray MP_ (discreteStateRangeLabels)
• static StringArray MP_ (discreteStateSetIntLabels)
• static StringArray MP_ (discreteStateSetStrLabels)
• static StringArray MP_ (discreteStateSetRealLabels)
• static StringArray MP_ (discreteUncSetIntLabels)
• static StringArray MP_ (discreteUncSetStrLabels)
• static StringArray MP_ (discreteUncSetRealLabels)
• static StringArray MP_ (histogramPointStrUncVars)
• static StringArray MP_ (linearEqScaleTypes)
• static StringArray MP_ (linearIneqScaleTypes)
• static StringArray VP_ (ddss)
• static StringArray VP_ (duss)
• static StringArray VP_ (dsss)
• static BitArray MP_ (discreteDesignSetIntCat)
• static BitArray MP_ (discreteDesignSetRealCat)
• static BitArray MP_ (discreteStateSetIntCat)
• static BitArray MP_ (discreteStateSetRealCat)
• static BitArray MP_ (discreteUncSetIntCat)
• static BitArray MP_ (discreteUncSetRealCat)
• static Var_brv MP2s (betaUncAlphas, 0.)
• static Var_brv MP2s (betaUncBetas, 0.)
• static Var_brv MP2s (exponentialUncBetas, 0.)
• static Var_brv MP2s (exponentialUncVars, 0.)
• static Var_brv MP2s (frechetUncAlphas, 2.)
• static Var_brv MP2s (gammaUncAlphas, 0.)
• static Var_brv MP2s (gammaUncBetas, 0.)
• static Var_brv MP2s (gammaUncVars, 0.)
• static Var_brv MP2s (gumbelUncAlphas, 0.)
• static Var_brv MP2s (lognormalUncErrFacts, 1.)
13.2. DAKOTA NAMESPACE REFERENCE

- static Var_brv MP2s (lognormalUncLambdas, 0.)
- static Var_brv MP2s (lognormalUncLowerBnds, 0.)
- static Var_brv MP2s (lognormalUncMeans, 0.)
- static Var_brv MP2s (lognormalUncStdDevs, 0.)
- static Var_brv MP2s (lognormalUncUpperBnds, std::numeric_limits<Real>::infinity())
- static Var_brv MP2s (lognormalUncVars, 0.)
- static Var_brv MP2s (loguniformUncLowerBnds, 0.)
- static Var_brv MP2s (loguniformUncUpperBnds, std::numeric_limits<Real>::infinity())
- static Var_brv MP2s (loguniformUncVars, 0.)
- static Var_brv MP2s (normalUncStdDevs, 0.)
- static Var_brv MP2s (poissonUncLambdas, 0.)
- static Var_brv MP2s (triangularUncLowerBnds, -std::numeric_limits<Real>::infinity())
- static Var_brv MP2s (triangularUncUpperBnds, std::numeric_limits<Real>::infinity())
- static Var_brv MP2s (triangularUncVars, 0.)
- static Var_brv MP2s (uniformUncLowerBnds, -std::numeric_limits<Real>::infinity())
- static Var_brv MP2s (uniformUncUpperBnds, std::numeric_limits<Real>::infinity())
- static Var_brv MP2s (uniformUncVars, 0.)
- static Var_brv MP2s (weibullUncAlphas, 0.)
- static Var_brv MP2s (weibullUncBetas, 0.)
- static Var_biv MP2s (binomialUncNumTrials, 0)
- static Var_biv MP2s (binomialUncVars, 0)
- static Var_biv MP2s (geometricUncVars, 0)
- static Var_biv MP2s (hyperGeomUncNumDrawn, 0)
- static Var_biv MP2s (hyperGeomUncSelectedPop, 0)
- static Var_biv MP2s (hyperGeomUncTotalPop, 0)
- static Var_biv MP2s (hyperGeomUncVars, 0)
- static Var_biv MP2s (negBinomialUncNumTrials, 0)
- static Var_biv MP2s (negBinomialUncVars, 0)
- static Var_biv MP2s (poissonUncVars, 0)
- static Var_mp_type Vtype (varsDomain, MIXED_DOMAIN)
- static Var_mp_type Vtype (varsDomain, RELAXED_DOMAIN)
- static Var_mp_type Vtype (varsView, ALL_VIEW)
- static Var_mp_type Vtype (varsView, DESIGN_VIEW)
- static Var_mp_type Vtype (varsView, UNCERTAIN_VIEW)
- static Var_mp_type Vtype (varsView, ALEATORY_UNCERTAIN_VIEW)
- static Var_mp_type Vtype (varsView, EPISTEMIC_UNCERTAIN_VIEW)
- static Var_mp_type Vtype (varsView, STATE_VIEW)

template<class ContainerT>
void flatten_num_array (const std::vector<ContainerT> &input_array, IntArray **pia)

Free convenience function that flatten sizes of an array of std containers; takes an array of containers and returns an IntArray containing the sizes of each container in the input array. Note: Did not specialize for vector<RealVector> as no current use cases.

- void dn2f_ (int *n, int *p, Real *x, Calcrj, int *iv, int *lv, Real *v, int *ui, void *ur, Vf)
- void dn2fb_ (int *n, int *p, Real *x, Real *b, Calcrj, int *iv, int *lv, Real *v, int *ui, void *ur, Vf)
- void dn2g_ (int *n, int *p, Real *x, Calcrj, Calcrj, int *iv, int *lv, Real *v, int *ui, void *ur, Vf)
- void dn2gb_ (int *n, int *p, Real *x, Real *b, Calcrj, Calcrj, int *iv, int *lv, Real *v, int *ui, void *ur, Vf)
• void divset (int *, int *, int *, int *, Real *)
  
• double dr7mdc (int *)
  
• static void Rsnapchk (NL2Misc *q)
  
• static int hasnaninf (const double *, int n)
  
• NLPQLPOptimizer * new_NLPQLPOptimizer (ProblemDescDB &problem_db, Model &model)
  
• NLPQLPOptimizer * new_NLPQLPOptimizer (Model &model)
  
• void print_c3_sobol_indices (double value, size_t ninteract, size_t *interactions, void *arg)
  
• static RealVector * static Lev_cost_vec (NULL)
  
• static size_t * static_qoi (NULL)
  
• static RealVector * static_Nlq_pilot (NULL)
  
• NLPQLPOptimizer * new_NLPQLPOptimizer (Model &model)
  
• void start_dakota_heartbeat (int)
  
• void dak_sigcatch (int sig)
  
• MPIUnpackBuffer & operator>>(MPIUnpackBuffer &s, ParallelLevel &pl)
  
  MPIUnpackBuffer extraction operator for ParallelLevel. Calls read(MPIUnpackBuffer&).

• MPIPackBuffer & operator<<(MPIPackBuffer &s, const ParallelLevel &pl)
  
  MPIPackBuffer insertion operator for ParallelLevel. Calls write(MPIPackBuffer&).

• std::istream & operator>>(std::istream &s, ParamResponsePair &pair)
  
  std::istream extraction operator for ParamResponsePair

• std::ostream & operator<<(std::ostream &s, const ParamResponsePair &pair)
13.2. DAKOTA NAMESPACE REFERENCE

std::ostream insertion operator for `ParamResponsePair`

- **MPIUnpackBuffer** & operator `>>` (MPIUnpackBuffer &s, ParamResponsePair &pair)
  
  *MPIUnpackBuffer* extraction operator for `ParamResponsePair`.

- **MPIPackBuffer** & operator `<<` (MPIPackBuffer &s, const ParamResponsePair &pair)
  
  *MPIPackBuffer* insertion operator for `ParamResponsePair`.

- bool operator `==` (const ParamResponsePair &pair1, const ParamResponsePair &pair2)
  
  equality operator for `ParamResponsePair`

- bool operator `!=` (const ParamResponsePair &pair1, const ParamResponsePair &pair2)
  
  inequality operator for `ParamResponsePair`

- static void `*binsearch` (void *kw, size_t kwsize, size_t n, const char *key)

- static const char * `Begins` (const String &entry_name, const char *s)

- static void `Bad_name` (String entry_name, const char *where)

- static void `Locked_db` ()

- static void `Null_rep` (const char *who)

- static void `Null_rep1` (const char *who)

- boost::regex `PARAMS_TOKEN` ("\{\{PARAMETERS\}\}"")

- boost::regex `RESULTS_TOKEN` ("\{\{RESULTS\}\}"")

- String `substitute_params_and_results` (const String &driver, const String &params, const String &results)

  Substitute parameters and results file names into driver strings.

- **MPIUnpackBuffer** & operator `>>` (MPIUnpackBuffer &s, ProgramOptions &progopt)
  
  *MPIUnpackBuffer* extraction operator.

- **MPIPackBuffer** & operator `<<` (MPIPackBuffer &s, const ProgramOptions &progopt)
  
  *MPIPackBuffer* insertion operator.

- bool set `compare` (const ParamResponsePair &database_pr, const ActiveSet &search_set)

  search function for a particular `ParamResponsePair` within a PRPList based on ActiveSet content (request vector and derivative variables vector)

- bool id `vars_exact_compare` (const ParamResponsePair &database_pr, const ParamResponsePair &search_pr)

  search function for a particular `ParamResponsePair` within a PRPMultiIndex

- std::size_t `hash_value` (const ParamResponsePair &pr)

  hash_value for `ParamResponsePairs` stored in a PRPMultiIndex

- PRPCacheHIter `hashedCacheBegin` (PRPCache &prp_cache)

  hashed definition of cache begin

- PRPCacheHIter `hashedCacheEnd` (PRPCache &prp_cache)

  hashed definition of cache end

- PRPQueueHIter `hashedQueueBegin` (PRPQueue &prp_queue)

  hashed definition of queue begin

- PRPQueueHIter `hashedQueueEnd` (PRPQueue &prp_queue)

  hashed definition of queue end

- PRPCacheHIter `lookup_by_val` (PRPMultiIndexCache &prp_cache, const ParamResponsePair &search_pr)

  find a `ParamResponsePair` based on the interface id, variables, and ActiveSet search data within search_pr.

- PRPCacheHIter `lookup_by_val` (PRPMultiIndexCache &prp_cache, const String &search_interface_id, const Variables &search_vars, const ActiveSet &search_set)
find a `ParamResponsePair` within a `PRPMultiIndexCache` based on the interface id, variables, and `ActiveSet` search data

- PRPCacheOIter `lookup_by_nearly_val` (PRPMultiIndexCache &prp_cache, const String &search_interface_id, const Variables &search_vars, const ActiveSet &search_set, Real tol)

- PRPCacheOIter `lookup_by_ids` (PRPMultiIndexCache &prp_cache, const IntStringPair &search_ids)

find a `ParamResponsePair` within a `PRPMultiIndexCache` based on search ids (i.e. `std::pair<eval_id, interface_id>`) search data

- PRPCacheOIter `lookup_by_ids` (PRPMultiIndexCache &prp_cache, const IntStringPair &search_ids, const ParamResponsePair &search_pr)

- PRPQueueHIter `lookup_by_val` (PRPMultiIndexQueue &prp_queue, const ParamResponsePair &search_pr)

find a `ParamResponsePair` based on the interface id, variables, and `ActiveSet` search data within `search_pr`.

- PRPQueueHIter `lookup_by_val` (PRPMultiIndexQueue &prp_queue, const String &search_interface_id, const Variables &search_vars, const ActiveSet &search_set)

find a `ParamResponsePair` within a `PRPMultiIndexQueue` based on interface id, variables, and `ActiveSet` search data

- PRPQueueOIter `lookup_by_eval_id` (PRPMultiIndexQueue &prp_queue, int search_id)

find a `ParamResponsePair` within a `PRPMultiIndexQueue` based on search id (i.e. integer `eval_id`) search data

- void `print_usage` (std::ostream &s)

print restart utility help message

- void `print_restart` (StringArray pos_args, String print_dest)

print a restart file

- void `print_restart_pdb` (StringArray pos_args, String print_dest)

print a restart file (PDB format)

- void `print_restart_tabular` (StringArray pos_args, String print_dest, unsigned short tabular_format, int tabular_precision)

print a restart file (tabular format)

- void `read_neutral` (StringArray pos_args)

read a restart file (neutral file format)

- void `repair_restart` (StringArray pos_args, String identifier_type)

repair a restart file by removing corrupted evaluations

- void `concatenate_restart` (StringArray pos_args)

concatenate multiple restart files

- String `method_results_hdf5_link_name` (const StrStrSizet &iterator_id)

Create a method results name (HDF5 link name) from iterator_id.

- String `method_hdf5_link_name` (const StrStrSizet &iterator_id)

Create a method name (HDF5 link name) from iterator_id.

- String `execution_hdf5_link_name` (const StrStrSizet &iterator_id)

Create an execution name (HDF5 link name) from iterator_id.

- String `object_hdf5_link_name` (const StrStrSizet &iterator_id, const StringArray &location)

- template< typename ScaleType >

String `scale_hdf5_link_name` (const StrStrSizet &iterator_id, const StringArray &location, const ScaleType &scale)

Create a scale name (HDF5 link name) for a scale from an iterator_id, the name of the result, the name of the response (can be empty), and the scale itself.
13.2. DAKOTA NAMESPACE REFERENCE

- template<typename T>
  
  ```cpp
  void expand_for_fields_sdv (const SharedResponseData &srd, const T &src_array, const String &src_desc, bool allow_by_element, T &expanded_array)
  ```

  expand primary response specs in SerialDenseVectors, e.g. scales, for fields no change on empty, expands 1 and num_groups, copies num_elements

- template<typename T>
  
  ```cpp
  void expand_for_fields_stl (const SharedResponseData &srd, const T &src_array, const String &src_desc, bool allow_by_element, T &expanded_array)
  ```

  expand primary response specs in STL containers, e.g. scale types, for fields no change on empty, expands 1 and num_groups, copies num_elements

- static HANDLE * wait_setup (std::map<pid_t, int> *M, size_t *pn)

- static int wait_for_one (size_t n, HANDLE *h, int req1, size_t *pi)

- void gauss_legendre_pts_wts_1D (int level, RealVector &result_0, RealVector &result_1)

- void lagrange_interpolation_1d (const RealVector &samples, const RealVector &abscissa, const RealVector &values, RealVector &result)

- void kronecker_product_2d (const RealMatrix &matrix1, const RealMatrix &matrix2, RealMatrix &matrix)

- void get_chebyshev_points (int order, RealVector &points)

- void chebyshev_derivative_matrix (int order, RealMatrix &derivative_matrix, RealVector &points)

- int salinas_main (int argc, char *argv[])

  subroutine interface to SALINAS simulation code

- std::string get_cwd_str ()

- std::vector<std::string> get_pathext ()

- bool contains (const bfs::path &dir_path, const std::string &file_name, boost::filesystem::path &complete_filepath)

Variables

- PRPCache data_pairs

  contains all parameter/response pairs.

- double PI = boost::math::constants::pi<double>()

  constant pi

- double HALF_LOG_2PI = std::log(2.0*PI)/2.0

  constant log(2*pi)/2.0

- short abort_mode = ABORT_EXITS

  by default Dakota exits or calls MPI_Abort on errors

- std::ostream * dakota_cout = &std::cout

  DAKOTA stdout initially points to <std::cout, but may be redirected to a tagged ofstream if there are < concurrent iterators.

- std::ostream * dakota_cerr = &std::cerr

  DAKOTA stderr initially points to <std::cerr, but may be redirected to a tagged ofstream if there are < concurrent iterators.

- ResultsManager iterator_results_db

  Global results database for iterator results.

- EvaluationStore evaluation_store_db

  Global database for evaluation storage.
• int write_precision = 10
  used in ostream data output functions < (restart_util.cpp overrides default value)

• MPIManager dummy_mpi_mgr
  dummy MPIManager for ref initialization

• ProgramOptions dummy_prg_opt
  dummy ProgramOptions for ref initialization

• OutputManager dummy_out_mgr
  dummy OutputManager for ref initialization

• ParallelLibrary dummy_lib
  dummy ParallelLibrary for ref initialization

• ProblemDescDB dummy_db
  dummy ProblemDescDB for ref initialization

• int mc_ptr_int = 0
  global pointer for ModelCenter API

• int dc_ptr_int = 0
  global pointer for ModelCenter eval DB

• ProblemDescDB *Dak_pddb
  set by ProblemDescDB, for use in parsing

• const size_t NPOS = ~ (size_t)0
  special value returned by index() when entry not found

• const double BIG_REAL_BOUND = 1.0e+30
  bound beyond which constraints are considered inactive

• static UShortStrBimap method_map
  bimap between method enums and strings; only used in this compilation unit

• static UShortStrBimap submethod_map
  bimap between sub-method enums and strings; only used in this compilation unit (using bimap for consistency, though at time of addition, only uni-directional mapping is supported)

• Interface dummy_interface
  dummy Interface object used for mandatory
  < reference initialization or default virtual < function return by reference when a real < Interface instance is unavailable

• Model dummy_model
  dummy Model object used for mandatory reference
  < initialization or default virtual function < return by reference when a real Model instance < is unavailable

• Iterator dummy_iterator
  dummy Iterator object used for mandatory
  < reference initialization or default virtual < function return by reference when a real < Iterator instance is unavailable

• Dakota_funcs *DF

• Dakota_funcs DakFuncs0

• const Real REAL_DSET_FILL_VAL = NAN

• const int INT_DSET_FILL_VAL = INT_MAX

• const String STR_DSET_FILL_VAL = ""

• const int HDF5_CHUNK_SIZE = 40000

• const char *FIELD_NAMES [ ]
13.2. DAKOTA NAMESPACE REFERENCE

- const int NUMBER_OF_FIELDS = 23
- static const int MPI_COMM_WORLD = 1
- static const int MPI_COMM_NULL = 0
- static const int MPI_COMM_SELF = 92
- static const int MPI_ANY_TAG = -1
- static void * MPI_REQUEST_NULL = NULL
  
  return
  
  FILE * nidrin

const size_t NIDR_MAX_ERROR_LEN = 8192

  maximum error length is roughly 100 lines at 80 char; using fixed error length instead of investing in converting to vsnprintf (C++11)

- static const char * auto_log_scaletypes [] = { "auto", "log", "none", 0 }

- static Var_uinfo CAUVLbl [CAUVar_Nkinds]
- static Var_uinfo DAUIVLbl [DAUIVar_Nkinds]
- static Var_uinfo DAUSVLbl [DAUSVar_Nkinds]
- static Var_uinfo DAURVLbl [DAURVar_Nkinds]
- static Var_uinfo CEUVLbl [CEUVar_Nkinds]
- static Var_uinfo DEUVLbl [DEUVar_Nkinds]
- static Var_uinfo DEUSVLbl [DEUSVar_Nkinds]
- static Var_uinfo DEURVLbl [DEURVar_Nkinds]
- static Var_uinfo DiscSetLbl [DiscSetVar_Nkinds]

- static VarLabelChk DesignAndStateLabelsCheck []

  Variables label array designations for design and state. All non-uncertain variables need to be in this array. Used in check_variables node to check lengths and make_variable_defaults to build labels.

- static VLReal VLUncertainReal [NUM_UNC_REAL_CONT]

  Variables labels/bounds/values check array for real-valued uncertain variables; one array entry per contiguous container. These associate the individual variables given by, e.g., CAUVLbl, with the contiguous container in which they are stored.

- static VLInt VLUncertainInt [NUM_UNC_INT_CONT]

  Variables labels/bounds/values check array for integer-valued uncertain variables; one array entry per contiguous container. These associate the individual variables given by, e.g., DAUIVLbl, with the contiguous container in which they are stored.

- static VLstr VLUncertainStr [NUM_UNC_STR_CONT]

  Variables labels/bounds/values check array for string-valued uncertain variables; one array entry per contiguous container. These associate the individual variables given by, e.g., DAUSVLbl, with the contiguous container in which they are stored.

- static int VLR_aleatory [NUM_UNC_REAL_CONT] = { 1, 0, 1, 0 }

  which uncertain real check array containers are aleatory (true = 1)

- static int VLI_aleatory [NUM_UNC_INT_CONT] = { 1, 0 }

  which uncertain integer check array containers are aleatory (true = 1)

- static int VLS_aleatory [NUM_UNC_STR_CONT] = { 1, 0 }

  which uncertain string check array containers are aleatory (true = 1)

- static Var_check var_mp_check_cv []
- static Var_check var_mp_check_dset []
- static Var_check var_mp_check_cau []
- static Var_check var_mp_check_daui []
• static Var_check var_mp_check_daus [ ]
• static Var_check var_mp_check_daur [ ]
• static Var_check var_mp_check_ceu [ ]
• static Var_check var_mp_check_deui [ ]
• static Var_check var_mp_check_deus [ ]
• static Var_check var_mp_check_deur [ ]
• static Var_check var_mp_cbound [ ]

This is used within check_variables_node(): \texttt{Var\_RealBoundIPCheck()} is applied to validate bounds and initial points.

• static Var_check var_mp_drange [ ]

This is used in check_variables_node(): \texttt{Var\_IntBoundIPCheck()} is applied to validate bounds and initial points, and in make_variable_defaults(): \texttt{Vgen\_*} is called to infer bounds.

• \texttt{TKFactoryDIPC tk\_factory\_dipc ("dakota\_dipc\_tk")}

Static registration of RW TK with the QUESO TK factory.

• \texttt{TKFactoryDIPCLogit tk\_factory\_dipclogit ("dakota\_dipc\_logit\_tk")}

Static registration of Logit RW TK with the QUESO TK factory.

• static time t start_time
• const double SCALING\_MIN\_SCALE = 1.0e10 + DBL\_MIN

minimum value allowed for a characteristic value when scaling; ten orders of magnitude greater than DBL\_MIN

• const double SCALING\_MIN\_LOG = SCALING\_MIN\_SCALE

lower bound on domain of logarithm function when scaling

• const double SCALING\_LOGBASE = 10.0

logarithm base to be used when scaling

• const double SCALING\_LN\_LOGBASE = std::log(SCALING\_LOGBASE)

\texttt{ln(SCALING\_LOGBASE)}; needed in transforming variables in several places

• const char * SCI\_FIELD\_NAMES [ ]
• const int SCI\_NUMBER\_OF\_FIELDS = 26

• const int LARGE\_SCALE = 100

a (perhaps arbitrary) definition of large scale; choose a large-scale algorithm if numVars \( \geq \) LARGE\_SCALE

• const double POW\_VAL = 1.0

offset used textbook exponent: 1.0 is nominal, 1.4 used for B&B testing

• const String LEV\_REF = "Dakota"

\texttt{levenshtein\_distance} computes the distance between its argument and this

13.2.1 Detailed Description

The primary namespace for DAKOTA. The Dakota namespace encapsulates the core classes of the DAKOTA framework and prevents name clashes with third-party libraries from methods and packages. The C++ source files defining these core classes reside in Dakota/src as *.\texttt{ch}pp.
13.2. DAKOTA NAMESPACE REFERENCE

13.2.2 Typedef Documentation

typedef bmi::multi_index_container<Dakota::ParamResponsePair, bmi::indexed_by<
bmi::ordered_non_unique<bmi::tag<bmi::ordered>>, bmi::const_mem_fun<Dakota::ParamResponsePair, const IntStringPair&,
&Dakota::ParamResponsePair::eval_interface_ids>>, bmi::hashed_non_unique<bmi::tag<bmi::hashed>>,
bmi::identity<Dakota::ParamResponsePair>, partial_prp_hash, partial_prp_equality>> > >
PRPMultiIndexCache

Boost Multi-Index Container for globally caching ParamResponsePairs.

For a global cache, both evaluation and interface id’s are used for tagging ParamResponsePair records.

typedef bmi::multi_index_container<Dakota::ParamResponsePair, bmi::indexed_by<
bmi::ordered_unique<bmi::tag<bmi::ordered>>, bmi::const_mem_fun<Dakota::ParamResponsePair,
int, &Dakota::ParamResponsePair::eval_id>>, bmi::hashed_non_unique<bmi::tag<bmi::hashed>>,
bmi::identity<Dakota::ParamResponsePair>, partial_prp_hash, partial_prp_equality>> > >
PRPMultiIndexQueue

Boost Multi-Index Container for locally queueing ParamResponsePairs.

For a local queue, interface id’s are expected to be consistent, such that evaluation id’s are sufficient for
tracking particular evaluations.

13.2.3 Enumeration Type Documentation

anonymous enum

Sub-methods, including sampling, inference algorithm, opt algorithm types.

Enumerator

**SUBMETHOD COLLABORATIVE**  Type of hybrid meta-iterator:

13.2.4 Function Documentation

CommandShell & flush ( CommandShell & shell )

convenient shell manipulator function to “flush” the shell

global convenience function for manipulating the shell; invokes the class member flush function.
References CommandShell::flush().

Referenced by HDF5I0Helper::append_empty(), HDF5IOHelper::create_dataset(), HDF5IOHelper::create_empty_dataset(), HDF5IOHelper::create_group(), SysCallApplicInterface::spawn_analysis_to_shell(), SysCallApplicInterface::spawn_evaluation_to_shell(), SysCallApplicInterface::spawn_input_filter_to_shell(), and SysCallApplicInterface::spawn_output_filter_to_shell().

void Dakota::apply_matrix_partial ( const MatrixType & M, const VectorType & v1, VectorType & v2 )

Applies a RealMatrix to a vector (or subset of vector) v1.

Optionally works with a subset of the passed vectors; applies the matrix M to the first M.numCols() entries in
v1, and populates the first M.numRows entries in v2.

References abort_handler().

Referenced by apply_linear_constraints(), DakotaROLIneqConstraintsHess::applyAdjointHessian(), DakotaRLOEqConstraintsHess::applyAdjointHessian(), DakotaROLIneqConstraintsGrad::applyJacobian(), DakotaRLOLEqConstraintsGrad::applyJacobian(), and DakotaROLObjectiveHess::hessVec().
void Dakota::apply_matrix_transpose_partial ( const RealMatrix & \( M \), const VectorType & \( v_1 \),
VectorType & \( v_2 \) )

Applies transpose of a RealMatrix to a vector (or subset of vector) \( v_1 \).

Optionally works with a subset of the passed vectors; applies the matrix \( M^\top \) to the first \( M\text{.numRows()} \) entries in \( v_1 \), and populates the first \( M\text{.numCols()} \) entries in \( v_2 \).

References abort_handler().

Referenced by DakotaROLIneqConstraintsGrad::applyAdjointJacobian(), and DakotaROLEqConstraintsGrad-::applyAdjointJacobian().

void abort_throw_or_exit ( int dakota_code )

throw or exit depending on abort_mode

Throw a Boost system_error or call std::exit, with \((256 \times \text{dakota\_code})\), where dakota_code < 0

RATIONALE: Avoid common "standard" exit codes and signals (signum.h) as well as uncaught signals / uncatchable SIGKILL which return 128

- \(<\text{signum}>\) on Linux = [129, 192]

Return a value in [0,255] since some operating systems only return the 8 least significant bits, leaves [193, 255] for Dakota. This should make return codes consistent cross-platform.

References abort_mode.

Referenced by abort_handler(), and ParallelLibrary::abort_helper().

void register_signal_handlers ( )

Tie various signal handlers to Dakota’s abort_handler function.

Global function to register signal handlers at top-level.

References abort_handler().

Referenced by main().

void mpi_debug_hold ( )

Global function to hold Dakota processes to help with MPI debugging.

See details in code for details, depending on MPI implementation in use.

Referenced by main().

T Dakota::abort_handler_t ( int code )

Templatized abort_handler_t method that allows for convenient return from methods that otherwise have no sensible return from error clauses. Usage: MyType& method() { return abort_handler<MyType&>(-1); } 

References abort_handler().

void svd ( RealMatrix & matrix, RealVector & singular_vals, RealMatrix & v_trans, bool compute_vectors = true )

Compute the SVD of an arbitrary matrix \( A = USV^\top \).

Uses Teuchos::LAPACK.GESVD() to compute the singular value decomposition, overwriting \( A \) with the left singular vectors \( U \) (or destroying \( A \) if compute_vectors = false); optionally returns right singular vectors in \( v\_trans \).

References abort_handler().

Referenced by ProbabilityTransformModel::acv_index_to_corr_index(), Model::assign_max_strings(), NonDBayesCalibration::augment_gradient_with_log_prior(), NonDBayesCalibration::augment_hessian_with_log_prior(),
13.2. **DAKOTA NAMESPACE REFERENCE**

NonDAdaptImpSampling::calculate_statistics(), PebblBranchSub::candidateSolution(), ActiveSubspaceModel::compute_svd(), ActiveSubspaceModel::computeBingLiCriterion(), ActiveSubspaceModel::computeConstantine-Metric(), Variables::continuous_variable_id(), Variables::continuous_variable_ids(), Variables::continuous_variable_label(), Variables::continuous_variable_labels(), Variables::continuous_variable_type(), Variables::continuous_variable_types(), SharedVariablesData::copy(), Model::discrete_int_sets(), Variables::discrete_int_variable_label(), Variables::discrete_int_variable_labels(), Variables::discrete_int_variable_type(), Variables::discrete_int_variable_types(), Variables::discrete_real_variable_label(), Variables::discrete_real_variable_labels(), Variables::discrete_real_variable_type(), Variables::discrete_real_variable_types(), Model::discrete_set_int_values(), Model::discrete_set_real_values(), Model::discrete_set_string_values(), Variables::discrete_string_variable_label(), Variables::discrete_string_variable_labels(), Variables::discrete_string_variable_type(), Variables::discrete_string_variable_types(), ParamStudy::distribute(), QMEEApproximation::find_scaled_coefficients(), NonDAdaptImpSampling::generate_samples(), DataTransformModel::get_hyperparam_vc_index(), Variables::inactive_continuous_variable_ids(), Variables::inactive_continuous_variable_labels(), Variables::inactive_continuous_variable_types(), Variables::inactive_discrete_int_variable_labels(), Variables::inactive_discrete_int_variable_types(), Variables::inactive_discrete_real_variable_labels(), Variables::inactive_discrete_real_variable_types(), Variables::inactive_discrete_string_variable_labels(), Variables::inactive_discrete_string_variable_types(), NonDLHSSampling::increment_lhs_parameter_set(), DataTransformModel::init_continuous_vars(), RecastModel::init_variables(), SimulationModel::initialize_solution_control(), NonDBayesCalibration::log_prior_density(), NonDSampling::mode_bits(), NonDSampling::mode_counts(), NestedModel::NestedModel(), NonDInterval::NonDInterval(), NonDLHSSampling::NonDLHSSampling(), Optimizer::Optimizer(), SensAnalysisGlobal::partial_corr(), ParamStudy::pre_run(), NonDBayesCalibration::prior_density(), NonDBayesCalibration::prior_sample(), ProbabilityTransformModel::ProbabilityTransformModel(), NonDAdaptImpSampling::recentered_density(), Constraints::reshape(), NonDSampling::sample_to_variables(), NonDAdaptImpSampling::select_rep_points(), singular_values(), PebblBranchSub::splitComputation(), NonDExpansion::update_final_statistics_gradients(), DataFitSurrModel::update_from_model(), DataFitSurrModel::update_model(), NestedModel::update_sub_model(), ReducedBasis::update_svd(), DataTransformModel::variables_expand(), RandomFieldModel::variables_resize(), AdaptedBasisModel::variables_resize(), ActiveSubspaceModel::variables_resize(), and NonDSampling::variables_to_sample().

```cpp
template int qr ( RealMatrix & A )
{
    Compute an in-place QR factorization A = QR.
    Uses Teuchos::LAPACK.GEQRF() to compute the QR decomposition, overwriting A with the transformations and R.
    References abort_handler() and
    References by SensAnalysisGlobal::partial_corr().
}
```

```cpp
template int qr_rsolve ( const RealMatrix & q_r, bool transpose, RealMatrix & rhs )
{
    Perform a multiple right-hand sides Rinv * rhs solve using the R from a qr factorization.
    Returns info > 0 if the matrix is singular
    Uses Teuchos::LAPACK.TRTRS() to perform a triangular backsolve
    References abort_handler() and
    References by SensAnalysisGlobal::partial_corr().
}
```

```cpp
template int generate_system_seed ( )
{
    clock microseconds-based random seed in [1, 1000000]
    Mimics DDACE timeSeed(), which returns the trailing microseconds on the time of day clock. Historically, most algorithms opted for DDACE, Utilib, std::clock(), in that order.
}```
Referenced by NonDWA5ABIBayesCalibration::calibrate(), PSUADEDesignCompExp::get_parameter_sets(), FSUDesignCompExp::get_parameter_sets(), NonDQuadrature::get_parameter_sets(), NonDSampling::initialize_lhs(), and NonDBayesCalibration::NonDBayesCalibration().

```cpp
bool Dakota::operator!= ( const ActiveSet & set1, const ActiveSet & set2 ) [inline]
```

inequality operator for ActiveSet

```cpp
bool Dakota::operator== ( const Model & m1, const Model & m2 ) [inline]
```
equality operator for Envelope is true if same letter instance

```cpp
bool Dakota::operator!= ( const Model & m1, const Model & m2 ) [inline]
```
equality operator for Envelope is true if different letter instance

```cpp
void Dakota::get_initial_values ( const Model & model, VecT & values )
```
Adapter for copying initial continuous variables values from a Dakota Model into TPL vectors

```cpp
void Dakota::get_bounds ( const RealVector & lower_source, const RealVector & upper_source, VecT & lower_target, VecT & upper_target, Real big_real_bound_size, Real no_value )
```
Adapter for copying continuous variables data from Dakota RealVector into TPL vectors

```cpp
void Dakota::get_bounds ( const Model & model, VecT & lower_target, VecT & upper_target )
```
Adapter for copying continuous variables data from a Dakota Model into TPL vectors

```cpp
void Dakota::get_bounds ( const SetT & source_set, VecT & lower_target, VecT & upper_target, int target_offset )
```
Adapter originating from (and somewhat specialized based on) APPSOptimizer for copying discrete variables from a set-based Dakota container into TPL vectors

```cpp
bool Dakota::get_mixed_bounds ( const MaskType & mask_set, const SetArray & source_set, const Teuchos::SerialDenseVector<OrdinalType, ScalarType> & lower_source, const Teuchos::SerialDenseVector<OrdinalType, ScalarType> & upper_source, VectorType2 & lower_target, VectorType2 & upper_target, ScalarType bigBoundSize, ScalarType no_value, int target_offset = 0 )
```
Adapter originating from (and somewhat specialized based on) APPSOptimizer for copying discrete integer variables data with bit masking from Dakota into TPL vectors
13.2. DAKOTA NAMESPACE REFERENCE

bool Dakota::get_variable_bounds ( Model & model, Real big_real_bound_size, int big_int_bound_size, typename AdapterT::VecT & lower, typename AdapterT::VecT & upper )

Adapter originating from (and somewhat specialized based on) APPSOptimizer for copying heterogeneous bounded data from Dakota::Variables into concatenated TPL vectors.

References Model::continuous_lower_bounds(), Model::continuous_upper_bounds(), Model::cv(), Model::discrete_int_lower_bounds(), Model::discrete_int_sets(), Model::discrete_int_upper_bounds(), Model::discrete_real_lower_bounds(), Model::discrete_real_upper_bounds(), Model::discrete_set_int_values(), Model::discrete_set_real_values(), Model::discrete_set_string_values(), Model::drv(), Model::get_bounds(), and Model::get_mixed_bounds().

int Dakota::configure_inequality_constraint_maps ( const Model & model, Real big_real_bound_size, CONSTRAINT_TYPE ctype, IVecT & map_indices, RVecT & map_multipliers, RVecT & map_offsets, Real scaling = 1.0 )

Adapter for configuring inequality constraint maps used when transferring data between Dakota and a TPL.

References Model::linear_ineq_constraint_lower_bounds(), Model::linear_ineq_constraint_upper_bounds(), Model::nonlinear_ineq_constraint_lower_bounds(), Model::nonlinear_ineq_constraint_upper_bounds(), Model::num_linear_ineq_constraints(), and Model::num_nonlinear_ineq_constraints().

Referenced by Optimizer::configure_constraint_maps().

void Dakota::configure_equality_constraint_maps ( Model & model, CONSTRAINT_TYPE ctype, IVecT & indices, size_t index_offset, RVecT & multipliers, RVecT & values, bool make_one_sided )

Adapter for configuring equality constraint maps used when transferring data between Dakota and a TPL.

References Model::linear_eq_constraint_coeffs(), Model::linear_eq_constraint_targets(), Model::num_linear_eq_constraints(), and Model::num_nonlinear_eq_constraints().

void Dakota::get_linear_constraints ( Model & model, Real big_real_bound_size, typename AdapterT::VecT & lin_ineq_lower_bnds, typename AdapterT::VecT & lin_ineq_upper_bnds, typename AdapterT::VecT & lin_eq_targets, typename AdapterT::MatT & lin_ineq_coeffss, typename AdapterT::MatT & lin_eq_coeffs )

Adapter based initially on APPSOptimizer for linear constraint maps and including matrix and bounds data; bundles a few steps together which could (should?) be broken into two or more adapters.

References copy_data(), get_bounds(), Model::linear_eq_constraint_coeffs(), Model::linear_eq_constraint_targets(), Model::linear_ineq_constraint_coeffs(), Model::linear_ineq_constraint_lower_bounds(), and Model::linear_ineq_constraint_upper_bounds().

void Dakota::apply_linear_constraints ( const Model & model, CONSTRAINT_EQUALITY_TYPE etype, const VecT & in_vals, VecT & values, bool adjoint = false )

Data adapter to transfer data from Dakota to third-party opt packages. The vector values might contain additional constraints; the first entries corresponding to linear constraints are populated by apply.

References apply_matrix_partial(), Model::linear_eq_constraint_coeffs(), Model::linear_eq_constraint_targets(), Model::linear_ineq_constraint_coeffs(), Model::num_linear_eq_constraints(), and Model::num_linear_ineq_constraints().

Referenced by DakotaROLIneqConstraints::value(), and DakotaROLEqConstraints::value().

void Dakota::apply_nonlinear_constraints ( const Model & model, CONSTRAINT_EQUALITY_TYPE etype, const VecT & in_vals, VecT & values, bool adjoint = false )

Data adapter to transfer data from Dakota to third-party opt packages.
If adjoint = false, (perhaps counter-intuitively) apply the Jacobian (transpose of the gradient) to in_vals, which should be of size num_continuous_vars: \( J^*x = G^*x \), resulting in num_nonlinear_const values getting populated (possibly a subset of the total constraint vector).

If adjoint = true, apply the adjoint Jacobian (gradient) to the nonlinear constraint portion of in_vals, which should be of size at least num_nonlinear_consts: \( J^*y = G^*y \), resulting in num_continuous_vars values getting populated.

References Model::current_response(), Model::cv(), Response::function_gradients(), Model::num_linear_eq_constraints(), Model::num_linear_ineq_constraints(), Model::num_nonlinear_eq_constraints(), and Model::num_nonlinear_ineq_constraints().

Referenced by DakotaROLIneqConstraintsGrad::applyAdjointJacobian(), DakotaROLEqConstraintsGrad::applyAdjointJacobian(), DakotaROLIneqConstraintsGrad::applyJacobian(), and DakotaROLEqConstraintsGrad::applyJacobian().

```cpp
void Dakota::set_best_responses ( typename AdapterT::OptT & optimizer, const Model & model, const std::vector<int> & constraint_map_indices, const std::vector<double> & constraint_map_multipliers, const std::vector<double> & constraint_map_offsets, ResponseArray & response_array )
```

Data adapter for use by third-party opt packages to transfer response data to Dakota

References Model::num_nonlinear_eq_constraints(), Model::num_nonlinear_ineq_constraints(), Model::primary_response_fn_sense(), and Model::response_size().

```cpp
void Dakota::set_variables ( const VectorType & source, Model & model, Variables & vars )
```

copy appropriate slices of source vector to Dakota::Variables

References Variables::continuous_variables(), copy_data_partial(), Variables::cv(), Model::discrete_int_sets(), Variables::discrete_int_variables(), Variables::discrete_real_variables(), Model::discrete_set_int_values(), Model::discrete_set_real_values(), Model::discrete_set_string_values(), Variables::discrete_string_variable(), Variables::div(), Variables::drv(), Variables::dsv(), and set_index_to_value().

Referenced by NomadOptimizer::Evaluator::eval_x().

```cpp
void Dakota::get_variables ( Model & model, VectorType & vec )
```

copy the various pieces comprising Dakota::Variables into a concatenated TPL vector

References abort_handler(), Model::continuous_variables(), copy_data_partial(), Model::cv(), Model::discrete_int_sets(), Model::discrete_int_variables(), Model::discrete_real_variables(), Model::discrete_set_int_values(), Model::discrete_set_real_values(), Model::discrete_set_string_values(), Model::discrete_string_variables(), Model::div(), Model::drv(), and Model::dsv().

```cpp
void Dakota::get_responses ( const Model & model, const RealVector & dak_fn_vals, const std::vector<int> & constraint_map_indices, const std::vector<double> & constraint_map_multipliers, const std::vector<double> & constraint_map_offsets, VectorType & f_vec, VectorType & eEqs_vec, VectorType & cEqs_vec, VectorType & cIneqs_vec )
```

Data adapter to transfer data from Dakota to third-party opt packages

References Model::num_nonlinear_eq_constraints(), and Model::primary_response_fn_sense().

Referenced by NomadOptimizer::Evaluator::eval_x(), and Optimizer::get_responses_from_dakota().

```cpp
void Dakota::get_nonlinear_eq_constraints ( const Model & model, VecT & values, Real scale, int offset = -1 )
```

Data adapter to transfer data from Dakota to third-party opt packages
References Model::current_response(), Response::function_values(), Model::nonlinear_eq_constraint_targets(), Model::num_linear_eq_constraints(), Model::num_nonlinear_eq_constraints(), and Model::num_nonlinear_ineq_constraints.

Referenced by DakotaROLEqConstraints::value().

```cpp
void Dakota::get_nonlinear_eq_constraints ( Model & model, const RealVector & curr_resp_vals, VecT & values, Real scale, int offset = 0 )
```

Data adapter to transfer data from Dakota to third-party opt packages

References Model::nonlinear_eq_constraint_targets(), and Model::num_nonlinear_eq_constraints().

```cpp
void Dakota::get_nonlinear_ineq_constraints ( const Model & model, VecT & values )
```

Data adapter to transfer data from Dakota to third-party opt packages (ROL-specific)

References copy_data_partial(), Model::current_response(), Response::function_values(), Model::num_linear_ineq_constraints(), and Model::num_nonlinear_ineq_constraints().

Referenced by DakotaROLIneqConstraints::value().

```cpp
void Dakota::get_nonlinear_bounds ( Model & model, VecT & nonlin_ineq_lower, VecT & nonlin_ineq_upper, VecT & nonlin_eq_targets )
```

Would like to combine the previous adapter with this one (based on APPSOptimizer and COLINOptimizer) and then see how much more generalization is needed to support other TPLs like JEGA.

Data adapter to transfer data from Dakota to third-party opt packages

References copy_data(), Model::nonlinear_eq_constraint_targets(), Model::nonlinear_ineq_constraint_lower_bounds(), and Model::nonlinear_ineq_constraint_upper_bounds().

Referenced by COLINApplication::set_problem().

```cpp
bool Dakota::operator!= ( const Response & resp1, const Response & resp2 ) [inline]
```

inequality operator for Response

inequality operator

```cpp
bool Dakota::operator!= ( const Variables & vars1, const Variables & vars2 ) [inline]
```

inequality operator for Variables

strict inequality operator

```cpp
void Dakota::write_ordered ( std::ostream & s, const SizetArray & comp_totals, const Teuchos::SerialDenseVector<OrdinalType, ScalarType1> & c_vector, const Teuchos::SerialDenseVector<OrdinalType, ScalarType2> & d_vector, const Teuchos::SerialDenseVector<OrdinalType, ScalarType3> & ds_vector, const Teuchos::SerialDenseVector<OrdinalType, ScalarType4> & dr_vector ) [inline]
```

free function to write Variables data vectors in input spec ordering

written for arbitrary types, but typical use will be ScalarType1 = Real, ScalarType2 = int, ScalarType3 = string, and ScalarType4 = int or Real.

Referenced by ParamStudy::pre_run().
void Dakota::write_ordered ( std::ostream & s, const SizetArray & comp_totals, const Teuchos::SerialDenseVector<OrdinalType, ScalarType1> & e_vector, const Teuchos::SerialDenseVector<OrdinalType, ScalarType2> & di_vector, const Teuchos::SerialDenseVector<OrdinalType, ScalarType3, 1> & ds_array, const Teuchos::SerialDenseVector<OrdinalType, ScalarType4> & dr_vector ) [inline]

free function to write Variables data vectors in input spec ordering
written for arbitrary types, but typical use will be ScalarType1 = Real, ScalarType2 = int, ScalarType3 = string, and ScalarType4 = int or Real.

void copy_field_data ( const RealVector & fn(vals, RealMatrix & fn_grad, const RealSymMatrixArray & fn_hess, size_t offset, size_t num_fns, Response & response )

This assumes the source gradient/Hessian are size less or equal to the destination response, and that the leading part is to be populated.
References Response::active_set_request_vector(), Response::function_gradient_view(), Response::function_hessian_view(), and Response::function_value().
Referenced by ExperimentData::scale_residuals().

void Dakota::copy_field_data ( const RealVector & fn vals, RealMatrix & fn grad, const RealSymMatrixArray & fn hess, size_t offset, size_t num fns, Response & response )

This assumes the source gradient/Hessian are size less or equal to the destination response, and that the leading part is to be populated.
References Response::function_gradient_view(), Response::function_hessian_view(), and Response::function_value().
Referenced by ExperimentData::scale_residuals().

void symmetric_eigenvalue_decomposition ( const RealSymMatrix & matrix, RealVector & eigenvalues, RealMatrix & eigenvectors )

Computes the eigenvalues and, optionally, eigenvectors of a real symmetric matrix A.
Eigenvalues are returned in ascending order.
References symmetric_eigenvalue_decomposition().
Referenced by NonDBayesCalibration::get_positive_definite_covariance_from_hessian(), and symmetric_eigenvalue_decomposition().

Real Dakota::getdist ( const RealVector & x1, const RealVector & x2 )

Gets the Euclidean distance between x1 and x2
References by mindist(), and mindistindx().

Real Dakota::mindist ( const RealVector & x, const RealMatrix & xset, int except )

Returns the minimum distance between the point x and the points in the set xset (compares against all points in xset except point "except"): if except is not needed, pass 0.
References getdist().
Referenced by getRmax().

Real Dakota::mindistindx ( const RealVector & x, const RealMatrix & xset, const IntArray & indx )

Gets the min distance between x and points in the set xset defined by the nindx values in indx.
References getdist().
Referenced by GaussProcApproximation::pointsel_add_sel().
Real Dakota::getRmax ( const RealMatrix & xset )

Gets the maximum of the min distance between each point and the rest of the set.
  References mindist().
  Referenced by GaussProcApproximation::pointsel_add_sel().

int Dakota::start_grid_computing ( char * analysis_driver_script, char * params_file, char * results_file )
sample function prototype for launching grid computing

int Dakota::stop_grid_computing ( )
sample function prototype for terminating grid computing

int Dakota::perform_analysis ( char * iteration_num )
sample function prototype for submitting a grid evaluation

string Dakota::asstring ( const T & val )

Creates a string from the argument val using an ostringstream.
  This only gets used in this file and is only ever called with ints so no error checking is in place.
Parameters

| val | The value of type T to convert to a string. |

Returns

  The string representation of val created using an ostringstream.
  Referenced by JEGAOptimizer::LoadTheConstraints().

void start_dakota_heartbeat ( int seconds )

Heartbeat function provided by dakota_filesystem-utils; pass output interval in seconds, or -1 to use $DAKOTA-
  HEARTBEAT
  Referenced by OutputManager::OutputManager().

bool Dakota::operator== ( const ParamResponsePair & pair1, const ParamResponsePair & pair2 ) [inline]
equality operator for ParamResponsePair
equality operator
  References ParamResponsePair::evalInterfaceIds, ParamResponsePair::prpResponse, and ParamResponsePair-
  ::prpVariables.

bool Dakota::operator!=( const ParamResponsePair & pair1, const ParamResponsePair & pair2 ) [inline]

inequality operator for ParamResponsePair
  inequality operator
bool Dakota::set_compare ( const ParamResponsePair & database_pr, const ActiveSet & search_set )
[inline]

search function for a particular ParamResponsePair within a PRPList based on ActiveSet content (request vector and derivative variables vector)

a global function to compare the ActiveSet of a particular database_pr (presumed to be in the global history list) with a passed in ActiveSet (search_set).
References ParamResponsePair::active_set(), ActiveSet::derivative_vector(), and ActiveSet::request_vector().
Referenced by lookup_by_val().

bool Dakota::id_vars_exact_compare ( const ParamResponsePair & database_pr, const ParamResponsePair & search_pr )
[inline]

search function for a particular ParamResponsePair within a PRPMultiIndex

a global function to compare the interface id and variables of a particular database_pr (presumed to be in the global history list) with a passed in key of interface id and variables provided by search_pr.
References ParamResponsePair::interface_id(), and ParamResponsePair::variables().
Referenced by partial_prp_equality::operator().

PRPCacheHIter Dakota::lookup_by_val ( PRPMultiIndexCache & prp_cache, const ParamResponsePair & search_pr )
[inline]

find a ParamResponsePair based on the interface id, variables, and ActiveSet search data within search_pr.

Lookup occurs in two steps: (1) PRPMultiIndexCache lookup based on strict equality in interface id and variables, and (2) set_compare() post-processing based on ActiveSet subset logic.
References ParamResponsePair::active_set(), and set_compare().
Referenced by NonDDREAMBayesCalibration::archive_acceptance_chain(), Minimizer::archive_best_results(), DataTransformModel::archive_submodel_responses(), NonDMUQBayesCalibration::cache_chain(), NonDQUE-SOBayesCalibration::cache_chain(), Model::db_lookup(), ApplicationInterface::duplication_detect(), SurrBasedLocalMinimizer::find_response(), Minimizer::local_recast_retrieve(), lookup_by_val(), SurrBasedMinimizer::print_results(), LeastSq::print_results(), Optimizer::print_results(), DiscrepancyCorrection::search_db(), and NonDLocalReliability::update_mpp_search_data().

PRPQueueHIter Dakota::lookup_by_val ( PRPMultiIndexQueue & prp_queue, const ParamResponsePair & search_pr )
[inline]

find a ParamResponsePair based on the interface id, variables, and ActiveSet search data within search_pr.

Lookup occurs in two steps: (1) PRPMultiIndexQueue lookup based on strict equality in interface id and variables, and (2) set_compare() post-processing based on ActiveSet subset logic.
References ParamResponsePair::active_set(), and set_compare().

void print_restart ( StringArray pos_args, String print_dest )

print a restart file

Usage: "dakota_restart_util print dakota.rst"
"dakota_restart_util to_neutral dakota.rst dakota.neu"

Prints all evals. in full precision to either stdout or a neutral file. The former is useful for ensuring that duplicate detection is successful in a restarted run (e.g., starting a new method from the previous best), and the latter is used for translating binary files between platforms.
References abort_handler(), ParamResponsePair::eval_id(), ParamResponsePair::write.annotated(), and write_precision.
13.2. DAKOTA NAMESPACE REFERENCE

Referenced by main().

void print_restart_pdb ( StringArray pos_args, String print_dest )

print a restart file (PDB format)

Usage: "dakota_restart_util_to_pdb dakota.rst dakota.pdb"

Unrolls all data associated with a particular tag for all evaluations and then writes this data in a tabular format
(e.g., to a PDB database or MATLAB/TECPLOT data file).

References abort_handler(), Variables::continuous_variables(), Variables::discrete_int_variables(), Variables-
::discrete_real_variables(), and Response::function_values().

Referenced by main().

void print_restart_tabular ( StringArray pos_args, String print_dest, unsigned short tabular_format, int tabular_precision )

print a restart file (tabular format)

Usage: "dakota_restart_util_to_tabular dakota.rst dakota.txt"

Unrolls all data associated with a particular tag for all evaluations and then writes this data in a tabular format
(e.g., to a PDB database or MATLAB/TECPLOT data file).

References abort_handler(), Variables::acv(), Variables::adiv(), Variables::adriv(), Variables::adsv(), Variables-
::all_continuous_variable_labels(), Variables::all_discrete_int_variable_labels(), Variables::all_discrete_real_variable-
labels(), Variables::all_discrete_string_variable_labels(), Response::function_labels(), ParamResponsePair::interface-
id(), ParamResponsePair::response(), ParamResponsePair::variables(), write_precision, ParamResponsePair::write-
tabular(), and ParamResponsePair::write_tabular_labels().

Referenced by main().

void read_neutral ( StringArray pos_args )

read a restart file (neutral file format)

Usage: "dakota_restart_util_from_neutral dakota.neu dakota.rst"

Reads evaluations from a neutral file. This is used for translating binary files between platforms.

References abort_handler(), and ParamResponsePair::read_annotated().

Referenced by main().

void repair_restart ( StringArray pos_args, String identifier_type )

repair a restart file by removing corrupted evaluations

Usage: "dakota_restart_util_remove 0.0 dakota_old.rst dakota_new.rst"
"dakota_restart_util_remove_ids 2 7 13 dakota_old.rst dakota_new.rst"

Repairs a restart file by removing corrupted evaluations. The identifier for evaluation removal can be either
a double precision number (all evaluations having a matching response function value are removed) or a list of
integers (all evaluations with matching evaluation ids are removed).

References abort_handler(), Response::active_set_request_vector(), contains(), ParamResponsePair::eval_id(),
Response::function_values(), and ParamResponsePair::response().

Referenced by main().

void concatenate_restart ( StringArray pos_args )

concatenate multiple restart files

Usage: "dakota_restart_util_cat dakota_1.rst ... dakota_n.rst dakota_new.rst"

Combines multiple restart files into a single restart database.
References abort_handler().
Referenced by main().

std::vector<std::string> Dakota::get_pathext()
Utility function for executable file search algorithms
Referenced by WorkdirHelper::which().

bool Dakota::contains(const bfs::path &dir_path, const std::string &file_name, boost::filesystem::path &complete_filepath) [inline]
Utility function for "which" sets complete_filepath from dir_path/file_name combo

13.2.5 Variable Documentation

short abort_mode = ABORT_EXITS
by default Dakota exits or calls MPI_Abort on errors
whether dakota exits/aborts or throws on errors
Referenced by abort_throw_or_exit(), Environment::exit_mode(), and PythonInterface::python_run().

UShortStrBimap submethod_map [static]
Initial value:

= boost::assign::list_of<UShortStrBimap::relation>
  (HYBRID, "hybrid")
  (SUBMETHOD_COLLABORATIVE, "collaborative")
  (SUBMETHOD_EMBEDDED, "embedded")
  (SUBMETHOD_SEQUENTIAL, "sequential")
  (SUBMETHOD_LHS, "lhs")
  (SUBMETHOD_RANDOM, "random")
  (SUBMETHOD_BOX_BEHENKEN, "box_behenken")
  (SUBMETHOD_CENTRAL_COMPOSITE, "central_composite")
  (SUBMETHOD_GRID, "grid")
  (SUBMETHOD_OA_LHS, "oa_lhs")
  (SUBMETHOD_OAS, "oas")
  (SUBMETHOD_DREAM, "dream")
  (SUBMETHOD_WASABI, "wasabi")
  (SUBMETHOD_GMMEA, "gmmea")
  (SUBMETHOD_MIQ, "miq")
  (SUBMETHOD_QUESO, "queso")
  (SUBMETHOD_NIP, "nip")
  (SUBMETHOD_SOCP, "scp")
  (SUBMETHOD_EA, "ea")
  (SUBMETHOD_EGO, "ego")
  (SUBMETHOD_SBO, "sbo")
  (SUBMETHOD_CONVERGE_ORDER, "converge_order")
  (SUBMETHOD_CONVERGE_QOI, "converge_qoi")
  (SUBMETHOD_ESTIMATE_ORDER, "estimate_order")

bimap between sub-method enums and strings; only used in this compilation unit (using bimap for consistency, though at time of addition, only uni-directional mapping is supported)
Referenced by Iterator::submethod_enum_to_string().
13.2. DAKOTA NAMESPACE REFERENCE

Dakota_funcs DakFuncs

Initial value:

= {
    fprintf,
    abort_handler,
    disolver_option,
    continuous_lower_bounds1,
    continuous_upper_bounds1,
    nonlinear_ineq_constraint_lower_bounds1,
    nonlinear_ineq_constraint_upper_bounds1,
    nonlinear_eq_constraint_targets1,
    linear_ineq_constraint_lower_bounds1,
    linear_ineq_constraint_upper_bounds1,
    linear_eq_constraint_targets1,
    linear_eq_constraint_coeffs1,
    ComputeResponses1,
    GetFuncs1,
    GetGrads1,
    GetContVars1,
    SetBestContVars1,
    SetBestDiscVars1,
    SetBestRespFns1,
    Get_Real1,
    Get_Int1,
    Get_Bool1
}

const char * FIELD_NAMES[]

Initial value:

= { "numFns", "numVars", "numACV", "numADIV",
    "numADRV", "numDerivVars", "xC", "xDI",
    "xDR", "xCLabels", "xDILabels",
    "xDRLabels", "directFnASV", "directFnDVV",
    "fnFlag", "gradFlag", "hessFlag",
    "InVals", "fnGrads", "fnHessians",
    "fnLabels", "failure", "currEvalId" }

fields to pass to Matlab in Dakota structure
Referenced by MatlabInterface::matlab_engine_run(), and MatlabInterface::MatlabInterface().

const int NUMBER_OF_FIELDS = 23

number of fields in above structure
Referenced by MatlabInterface::matlab_engine_run(), and MatlabInterface::MatlabInterface().

Var_uinfo CAUVLbl[CAUVar_Nkinds] [static]

Initial value:

= {
    VarLabelInfo(nuv., NormalUnc),
    VarLabelInfo(lnuv., LognormalUnc),
    VarLabelInfo(uuv., UniformUnc),
    VarLabelInfo(luuv., LoguniformUnc),
    VarLabelInfo(tuv., TriangularUnc),
    VarLabelInfo(euv., ExponentialUnc),
    VarLabelInfo(beuv., BetaUnc),
    VarLabelInfo(gauv., GammaUnc),
    VarLabelInfo(guuv., GumbelUnc),
    VarLabelInfo(fuv., FrechetUnc),
}
VarLabelInfo(wu\_v, WeibullUnc),
VarLabelInfo(hbuv\_v, HistogramBinUnc)

\textbf{Var\_uinfo DAUIVLbl[DAUIVar\_Nkinds] [static]}

Initial value:

\{ 
VarLabelInfo(puv\_v, PoissonUnc),
VarLabelInfo(binv\_v, BinomialUnc),
VarLabelInfo(nbuv\_v, NegBinomialUnc),
VarLabelInfo(geuv\_v, GeometricUnc),
VarLabelInfo(hguv\_v, HyperGeomUnc),
VarLabelInfo(hpiuv\_v, HistogramPtIntUnc)
\}

\textbf{Var\_uinfo DAUSVLbl[DAUSVar\_Nkinds] [static]}

Initial value:

\{ 
VarLabelInfo(hpsuv\_v, HistogramPtStrUnc)
\}

\textbf{Var\_uinfo DAURVLbl[DAURVar\_Nkinds] [static]}

Initial value:

\{ 
VarLabelInfo(hpruv\_v, HistogramPtRealUnc)
\}

\textbf{Var\_uinfo CEUVLbl[CEUVar\_Nkinds] [static]}

Initial value:

\{ 
VarLabelInfo(ciuv\_v, ContinuousIntervalUnc)
\}

\textbf{Var\_uinfo DEUIVLbl[DEUIVar\_Nkinds] [static]}

Initial value:

\{ 
VarLabelInfo(diuv\_v, DiscreteIntervalUnc),
VarLabelInfo(dusiv\_v, DiscreteUncSetInt)
\}

\textbf{Var\_uinfo DEUSVLbl[DEUSVar\_Nkinds] [static]}

Initial value:

\{ 
VarLabelInfo(dussv\_v, DiscreteUncSetStr)
\}
13.2. DAKOTA NAMESPACE REFERENCE

Var\_uinfo DEURVLbl[DEURVar\_Nkinds] [static]

**Initial value:**

```
= {
    VarLabelInfo(dusrv, DiscreteUncSetReal)
}
```

Var\_uinfo DiscSetLbl[DiscSetVar\_Nkinds] [static]

**Initial value:**

```
= {
    VarLabelInfo(ddisiv, DiscreteDesSetInt),
    VarLabelInfo(ddssv, DiscreteDesSetStr),
    VarLabelInfo(ddsrv, DiscreteDesSetReal),
    VarLabelInfo(dssiv, DiscreteStateSetInt),
    VarLabelInfo(dsssv, DiscreteStateSetStr),
    VarLabelInfo(dssrv, DiscreteStateSetReal)
}
```

VarLabelChk DesignAndStateLabelsCheck[] [static]

**Initial value:**

```
= {
    AVI numContinuousDesVars, AVI continuousDesignLabels, "cdv", "cdv\_descriptors" },
    AVI numDiscreteDesSetIntVars, AVI discreteDesignSetIntLabels, "ddsiv", "ddsiv\_descriptors" ),
    AVI numDiscreteDesSetStrVars, AVI discreteDesignSetStrLabels, "ddssv", "ddssv\_descriptors" ),
    AVI numContinuousStateVars, AVI continuousStateLabels, "csv", "csv\_descriptors" ),
    AVI numDiscreteStateRangeVars, AVI discreteStateRangeLabels, "dsriv", "dsriv\_descriptors" ),
    AVI numDiscreteStateSetIntVars, AVI discreteStateSetIntLabels, "dssiv", "dssiv\_descriptors" ),
    AVI numDiscreteStateSetStrVars, AVI discreteStateSetStrLabels, "dsssv", "dsssv\_descriptors" ),
    AVI numContinuousDesVars, AVI continuousDesignScaleTypes, 0, "cdv\_scale\_types" )
}
```

Variables label array designations for design and state. All non-uncertain variables need to be in this array. Used in check\_variables\_node to check lengths and make\_variable\_defaults to build labels.

Referenced by NIDRProblemDescDB::check\_variables\_node(), and NIDRProblemDescDB::make\_variable\_defaults().

V1real VLUncertainReal[NUM\_UNC\_REAL\_CONT] [static]

**Initial value:**

```
= {
    {CAUVar\_Nkinds, AVI CAUv, CAUVLbl, 
     DVR continuousAleatoryUncLabels, 
     DVR continuousAleatoryUncLowerBnds, 
     DVR continuousAleatoryUncUpperBnds, 
     DVR continuousAleatoryUncVars},
    {CEUVar\_Nkinds, AVI CEUv, CEUVLbl, 
     DVR continuousEpistemicUncLabels, 
     DVR continuousEpistemicUncLowerBnds, 
     DVR continuousEpistemicUncUpperBnds, 
     DVR continuousEpistemicUncVars},
    {DAURVar\_Nkinds, AVI DAURv, DAURVLbl, 
     DVR discreteRealAleatoryUncLabels, 
     DVR discreteRealAleatoryUncLowerBnds, 
     DVR discreteRealAleatoryUncUpperBnds, 
     DVR discreteRealAleatoryUncVars},
    {DSSVVar\_Nkinds, AVI DSSVv, DSSVLbl, 
     DVR discreteStateSetIntLabels, 
     DVR discreteStateSetIntLowerBnds, 
     DVR discreteStateSetIntUpperBnds, 
     DVR discreteStateSetIntVars},
```
Variables labels/bounds/values check array for real-valued uncertain variables; one array entry per contiguous container. These associate the individual variables given by, e.g., CAUVLbl, with the contiguous container in which they are stored.

Referenced by NIDRProblemDescDB::check_variables_node(), and NIDRProblemDescDB::make_variable_defaults().

VLint VLUncertainInt[NUM_UNC_INT_CONT] [static]

Initial value:

```cpp
= {
  {DAUIVarNkinds, AVI DAUIv, DAUIVLbl,
   DVR discreteIntAleatoryUncLabels,
   DVR discreteIntAleatoryUncLowerBnds,
   DVR discreteIntAleatoryUncUpperBnds,
   DVR discreteIntAleatoryUncVars},
  {DEUIVarNkinds, AVI DEUIv, DEUIVLbl,
   DVR discreteIntEpistemicUncLabels,
   DVR discreteIntEpistemicUncLowerBnds,
   DVR discreteIntEpistemicUncUpperBnds,
   DVR discreteIntEpistemicUncVars}}
```

Variables labels/bounds/values check array for integer-valued uncertain variables; one array entry per contiguous container. These associate the individual variables given by, e.g., DAUIVLbl, with the contiguous container in which they are stored.

Referenced by NIDRProblemDescDB::check_variables_node(), and NIDRProblemDescDB::make_variable_defaults().

VLstr VLUncertainStr[NUM_UNC_STR_CONT] [static]

Initial value:

```cpp
= {
  {DAUSVarNkinds, AVI DAUSv, DAUSVLbl,
   DVR discreteStrAleatoryUncLabels,
   DVR discreteStrAleatoryUncLowerBnds,
   DVR discreteStrAleatoryUncUpperBnds,
   DVR discreteStrAleatoryUncVars},
  {DEUSVarNkinds, AVI DEUSv, DEUSVLbl,
   DVR discreteStrEpistemicUncLabels,
   DVR discreteStrEpistemicUncLowerBnds,
   DVR discreteStrEpistemicUncUpperBnds,
   DVR discreteStrEpistemicUncVars}}
```

Variables labels/bounds/values check array for string-valued uncertain variables; one array entry per contiguous container. These associate the individual variables given by, e.g., DAUSVLbl, with the contiguous container in which they are stored.

Referenced by NIDRProblemDescDB::check_variables_node(), and NIDRProblemDescDB::make_variable_defaults().
13.2. DAKOTA NAMESPACE REFERENCE

Var_check var_mp_check_cv[] [static]
Initial value:

= { 
  Vchk_3(continuous_design,ContinuousDes),
  Vchk_3(continuous_state,ContinuousState) }

Var_check var_mp_check_dset[] [static]
Initial value:

= { 
  Vchk_3(discrete_design_set_integer,DiscreteDesSetInt),
  Vchk_3(discrete_design_set_string,DiscreteDesSetStr),
  Vchk_3(discrete_design_set_real,DiscreteDesSetReal),
  Vchk_3(discrete_state_set_integer,DiscreteStateSetInt),
  Vchk_3(discrete_state_set_string,DiscreteStateSetStr),
  Vchk_3(discrete_state_set_real,DiscreteStateSetReal) }

Var_check var_mp_check_cau[] [static]
Initial value:

= { 
  Vchk_3(normal_uncertain,NormalUnc),
  Vchk_3(lognormal_uncertain,LognormalUnc),
  Vchk_3(uniform_uncertain,UniformUnc),
  Vchk_3(loguniform_uncertain,LoguniformUnc),
  Vchk_3(triangular_uncertain,TriangularUnc),
  Vchk_3(exponential_uncertain,ExponentialUnc),
  Vchk_3(beta_uncertain,BetaUnc),
  Vchk_3(gamma_uncertain,GammaUnc),
  Vchk_3(gumbel_uncertain,GumbelUnc),
  Vchk_3(frechet_uncertain,FrechetUnc),
  Vchk_3(weibull_uncertain,WeibullUnc),
  Vchk_3(histogram_bin_uncertain,HistogramBinUnc) }

Var_check var_mp_check_dau[] [static]
Initial value:

= { 
  Vchk_3(poisson_uncertain,PoissonUnc),
  Vchk_3(binomial_uncertain,BinomialUnc),
  Vchk_3(negative_binomial_uncertain,NegBinomialUnc),
  Vchk_3(geometric_uncertain,GeometricUnc),
  Vchk_3(hypergeometric_uncertain,HyperGeomUnc),
  Vchk_3(histogram_point_int_uncertain,HistogramPtIntUnc) }

Var_check var_mp_check_daus[] [static]
Initial value:

= { 
  Vchk_3(histogram_point_str_uncertain,HistogramPtStrUnc) }
Var_check var_mp_check_daur[] [static]
Initial value:
= { Vchk.3(histogram_point.real_uncertain,HistogramPtRealUnc) }

Var_check var_mp_check_ceu[] [static]
Initial value:
= { Vchk.3(continuous_interval_uncertain,ContinuousIntervalUnc) }

Var_check var_mp_check_deui[] [static]
Initial value:
= { Vchk.3(discrete_interval_uncertain,DiscreteIntervalUnc),
      Vchk.3(discrete_uncertain_set_integer,DiscreteUncSetInt) }

Var_check var_mp_check_deus[] [static]
Initial value:
= { Vchk.3(discrete_uncertain_set_string,DiscreteUncSetStr) }

Var_check var_mp_check_deur[] [static]
Initial value:
= { Vchk.3(discrete_uncertain_set_real,DiscreteUncSetReal) }

Var_check var_mp_cbound[] [static]
Initial value:
= { Vchk.7(continuous_design,ContinuousDes,continuousDesign),
      Vchk.7(continuous_state,ContinuousState,continuousState),
      Vchk.5(normal_uncertain,NormalUnc,normalUnc),
      Vchk.5(lognormal_uncertain,LognormalUnc,lognormalUnc),
      Vchk.5(uniform_uncertain,UniformUnc,uniformUnc),
      Vchk.5(loguniform_uncertain,LoguniformUnc,loguniformUnc),
      Vchk.5(triangular_uncertain,TriangularUnc,triangularUnc),
      Vchk.5(beta_uncertain,BetaUnc,betaUnc) }

This is used within check_variables_node(): Var_RealBoundIPCheck() is applied to validate bounds and initial
points.
Referenced by NIDRProblemDescDB::check_variables_node().
13.3. DAKOTA::SURROGATES NAMESPACE REFERENCE

Var::check var::mp::drange[ ] [static]

Initial value:

= { Vchk?, (discrete::design::range, Discrete::Des::Range, discrete::Design::Range),
Vchk?, (discrete::state::range, Discrete::State::Range, discrete::State::Range) }

This is used in check::variables::node(): Var::Int::Bound::IP::Check() is applied to validate bounds and initial points,
and in make::variable::defaults(): Vgen::* is called to infer bounds.

Referenced by NIDR::Problem::Desc::DB::check::variables::node(), and NIDR::Problem::Desc::DB::make::variable::
    defaults().

const char* SCI_FIELD_NAMES[ ]

Initial value:

= { "dakota::type", "numFns", "numVars", "numACV", "numADIV",
    "numADRV", "numDeriv::Vars", "xC", "xDI",
    "xDR", "xLabels", "xDILabels",
    "xDRLabels", "direct::Fn::ASV", "direct::Fn::ASM",
    "direct::Fn::DVV", "direct::Fn::DVV::bool",
    "fnFlag", "gradFlag", "hessFlag",
    "fnVals", "fnGrads", "fnHessians",
    "fnLabels", "failure", "currEvalId" }

fields to pass to Scilab in Dakota structure

Referenced by ScilabInterface::scilab::engine::run().

const int SCI_NUMBER_OF_FIELDS = 26

number of fields in above structure

Referenced by ScilabInterface::scilab::engine::run().

13.3. Dakota::surrogates Namespace Reference

namespace for new Dakota surrogates module

Classes

- class GaussianProcess
  
  The GaussianProcess constructs a Gaussian Process regressor surrogate given a matrix of data.

- class GP::Objective
  
  ROL objective function for the Gaussian Process (GP) surrogate.

- class PolynomialRegression
  
  The PolynomialRegression class constructs a polynomial regressor using ordinary least squares.

- class Surrogate
  
  Parent class for surrogate models.
Typedefs
- using RolVec = ROL::Vector<double>
  \textit{Dakota} alias for \textit{ROL} \textit{Vector}.
- using RolStdVec = ROL::StdVector<double>
  \textit{Dakota} alias for \textit{ROL} \textit{StdVector}.
- using SCALER\_TYPE = util::DataScaler::SCALER\_TYPE
  alias for \textit{util} \textit{SCALER\_TYPE} enum
- using SOLVER\_TYPE = util::LinearSolverBase::SOLVER\_TYPE
  alias for \textit{util} \textit{SOLVER\_TYPE} enum

 Functions
- void compute\_next\_combination (int num\_dims, int level, VectorXi &index, bool &extend, int &h, int &t)
- void size\_level\_index\_vector (int num\_dims, int level, MatrixXi &indices)
  \textit{Compute a matrix of basis indices for given dimension and level. Each row of the matrix sums to level.}
- void compute\_hyperbolic\_subdim\_level\_indices (int num\_dims, int level, int num\_active\_dims, double p, MatrixXi &indices)
  \textit{Compute a matrix of indices for a submatrix (i.e. up to the active dimensions column) of indices produced by size\_level\_index\_vector(num\_dims, level, indices) where each feature has a component > 0 and respects the p-norm cutoff.}
- void compute\_hyperbolic\_level\_indices (int num\_dims, int level, double p, MatrixXi &indices)
  \textit{Compute the hyperbolic cross indices for a given level.}
- void compute\_hyperbolic\_indices (int num\_dims, int level, double p, MatrixXi &indices)
  \textit{Compute the hyperbolic cross indices for all levels up to level.}
- void compute\_reduced\_indices (int num\_dims, int level, MatrixXi &indices)
  \textit{Compute the reduced indices for all levels up to level.}
- void fd\_check\_gradient (Surrogate &surr, const MatrixXd &sample, MatrixXd &fd\_error, const int num\_steps=10)
  \textit{Perform a centered finite difference check of a \textit{Surrogate}'s gradient method.}
- void fd\_check\_hessian (Surrogate &surr, const MatrixXd &sample, MatrixXd &fd\_error, const int num\_steps=10)
  \textit{Perform a centered finite difference check of a \textit{Surrogate}'s Hessian method.}

13.3.1 Detailed Description
namespace for new Dakota surrogates module

13.3.2 Function Documentation
void compute\_next\_combination ( int num\_dims, int level, VectorXi &index, bool & extend, int & h, int & t )
Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>num_dims</td>
<td>int</td>
<td>Dimension of the feature space.</td>
</tr>
<tr>
<td>level</td>
<td>int</td>
<td>Total order in each row of indices. Should be &gt;= 1.</td>
</tr>
<tr>
<td>index</td>
<td>int</td>
<td>Vector of ints that specifies the powers for each term in the basis.</td>
</tr>
<tr>
<td>extend</td>
<td>bool</td>
<td>Bool for whether to continue with the computation of basis indices.</td>
</tr>
<tr>
<td>h</td>
<td>double</td>
<td>Working variable for basis enumeration.</td>
</tr>
<tr>
<td>t</td>
<td>double</td>
<td>Working variable for basis enumeration.</td>
</tr>
</tbody>
</table>

Referenced by size_level_index_vector().

```cpp
def size_level_index_vector(int num_dims, int level, MatrixXi & indices):
    Compute a matrix of basis indices for given dimension and level. Each row of the matrix sums to level.
    Parameters:
    - num_dims: Dimension of the feature space.
    - level: Total order in each row of indices. Should be >= 1.
    - indices: Matrix of indices - (num_terms by num_dims).
```

References compute_next_combination(), and dakota::util::n_choose_k().
Referenced by compute_hyperbolic_level_indices(), and compute_hyperbolic_subdim_level_indices().

```cpp
def compute_hyperbolic_subdim_level_indices(int num_dims, int level, int num_active_dims, double p, MatrixXi & indices):
    Compute a matrix of indices for a submatrix (i.e. up to the active dimensions column) of indices produced by size_level_index_vector(num_dims, level, indices) where each feature has a component > 0 and respects the p-norm cutoff.
    Parameters:
    - num_dims: Dimension of the feature space.
    - level: Total order in each row of indices. Should be >= 1.
    - num_active_dims: The # of active features and end index of the submatrix.
    - p: Real value for p-norm.
    - indices: Matrix of indices - (num_terms by num_active_dims).
```

References dakota::util::num_nonzeros(), dakota::util::p_norm(), dakota::silence_unused_args(), and size_level_index_vector().
Referenced by compute_hyperbolic_level_indices().

```cpp
def compute_hyperbolic_level_indices(int num_dims, int level, double p, MatrixXi & indices):
    Compute the hyperbolic cross indices for a given level.
    Parameters:
    - num_dims: Dimension of the feature space.
    - level: Total order in each column of indices. Should be >= 0.
    - p: Real value for p-norm.
```
CHAPTER 13. NAMESPACE DOCUMENTATION

| out         | indices | Matrix of indices - (num_dims by num_terms) |

References dakota::util::append_columns(), compute_hyperbolic_subdim_level_indices(), dakota::util::nonzero(), dakota::util::num_nonzeros(), and size_level_index_vector().

Referenced by compute_hyperbolic_indices().

void compute_hyperbolic_indices ( int num_dims, int level, double p, MatrixXi & indices )

Compute the hyperbolic cross indices for all levels up to level.

Parameters

| in            | num_dims | Dimension of the feature space. |
| in            | level    | Highest level to compute basis indices for. |
| in            | p        | Real value for p-norm. |
| out           | indices  | Matrix of indices - (num_dims by num_terms). |

References dakota::util::append_columns(), and compute_hyperbolic_level_indices().

Referenced by PolynomialRegression::build().

void compute_reduced_indices ( int num_dims, int level, MatrixXi & indices )

Compute the reduced indices for all levels up to level.

Parameters

| in            | num_dims | Dimension of the feature space. |
| in            | level    | Highest level to compute basis indices for. |
| out           | indices  | Matrix of indices - (num_dims by num_terms). |

References dakota::util::append_columns().

Referenced by PolynomialRegression::build().

void fd_check_gradient ( Surrogate & surr, const MatrixXd & sample, MatrixXd & fd_error, const int num_steps = 10 )

Perform a centered finite difference check of a Surrogate’s gradient method.

Parameters

| in            | surr     | Reference to a Surrogate. |
| in            | sample   | Point to evaluate the gradient at - (1 by numVariables). |
| out           | fd_error | Matrix of finite difference error for each component of the gradient - (num_steps by numVariables). |
| in            | num_steps | Number of increments (N) for the finite difference. The increment vector h = 10**(-i), i = 1, ..., N. |

References Surrogate::dataScaler, Surrogate::gradient(), and Surrogate::value().

void fd_check_hessian ( Surrogate & surr, const MatrixXd & sample, MatrixXd & fd_error, const int num_steps = 10 )

Perform a centered finite difference check of a Surrogate’s Hessian method.
13.4. Dakota::Util Namespace Reference

namespace for new Dakota utilities module

Classes

- class DataScaler
  
  The DataScaler class computes the scaling coefficients and scales a 2D data matrix with dimensions num_samples by num_features.

- class NormalizationScaler
  
  Normalizes the data using max and min feature values.

- class StandardizationScaler
  
  Standardizes the data so the each feature has zero mean and unit variance.

- class NoScaler
  
  Leaves the data unscaled.

- class LinearSolverBase
  
  The LinearSolverBase class serves as an API for derived solvers.

- class CholeskySolver
  
  The CholeskySolver class can be used to solve Ax = b, provided that A is a symmetric matrix.

Typedefs

- using SCALER_TYPE = DataScaler::SCALER_TYPE
  
  alias for DataScaler's SCALER_TYPE

- using BimapScalertypeStr = boost::bimap< SCALER_TYPE, std::string >
  
  alias for Boost Bimap scaler type <-> string

- using SOLVER_TYPE = LinearSolverBase::SOLVER_TYPE
  
  alias for LinearSolverBase's SOLVER_TYPE

- using BimapSolvertypeStr = boost::bimap< SOLVER_TYPE, std::string >
  
  alias for Boost Bimap solver type <-> string

Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>surv</th>
<th>Reference to a Surrogate.</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>sample</td>
<td>Point to evaluate the Hessian at - (1 by numVariables).</td>
</tr>
<tr>
<td>out</td>
<td>fd_error</td>
<td>Matrix of finite difference error for each independent component of the Hessian. There are numVariables*(numVariables+1)/2 = numInd components - (num_steps by numInd). of the Hessian - (num_steps by numVariables).</td>
</tr>
<tr>
<td>in</td>
<td>num_steps</td>
<td>Number of increments (N) for the finite difference. The increment vector h = 10**(-i), i = 1, ..., N.</td>
</tr>
</tbody>
</table>

References Surrogate::dataScaler, Surrogate::hessian(), and Surrogate::value().
Functions

- void error (const std::string &msg)
  
  Throws a std::runtime_error based on the message argument.

- bool matrix_equals (const MatrixXi &A, const MatrixXi &B)
  
  Tests whether two Eigen MatrixXi objects are equal.

- bool matrix_equals (const MatrixXd &A, const MatrixXd &B, double tol)
  
  Tests whether two Eigen MatrixXd objects are equal, within a given tolerance.

- bool matrix_equals (const RealMatrix &A, const RealMatrix &B, double tol)
  
  Tests whether two Teuchos RealMatrix objects are equal, within a given tolerance.

- double variance (const VectorXd &vec)
  
  Calculates the variance based on an Eigen VectorXd of double values.

- void populateVectorsFromFile (const std::string &filename, std::vector<VectorXd> &R, int num_datasets, int num_samples)
  
  Populate a collection of vectors read in from a text file assuming data layout is one dataset per row.

- void populateMatricesFromFile (const std::string &filename, std::vector<MatrixXd> &S, int num_datasets, int num_vars, int num_samples)
  
  Populate a collection of matrices read in from a text file assuming data layout is a "column-major" stack of num_samples by num_vars matrices.

- std::shared_ptr<DataScaler> scaler_factory (DataScaler::SCALER_TYPE scaler_type, const MatrixXd &unscaled_matrix)
  
  Free function to construct DataScaler.

- std::shared_ptr<LinearSolverBase> solver_factory (LinearSolverBase::SOLVER_TYPE type)
  
  Free function to construct LinearSolverBase.

- int n_choose_k (int n, int k)
  
  Calculate Binomial coefficient n choose k.

- template<typename T> int num_nonzeros (const T &mat)
  
  Calculate and return number of nonzero entries in vector or matrix.

- template<typename T1, typename T2> void nonzero (const T1 &v, T2 &result)
  
  Create a vector of indices based on nonzero entries in input vector.

- template<typename T1, typename T2> void append_columns (const T1 &new_cols, T2 &target)
  
  Append columns of input matrix to existing matrix.

- template<typename T> double p_norm (const T &v, double p)
  
  Calculate and return p-norm of a vector.

Variables

- static BimapScalertypeStr type_name_bimap
  
  Bimap between scaler types and names.

- static BimapSolvertypeStr type_name_bimap
  
  Bimap between solver types and names.
13.4. DAKOTA::UTIL NAMESPACE REFERENCE

13.4.1 Detailed Description

namespace for new Dakota utilities module

13.4.2 Function Documentation

void error ( const std::string & msg )

Throws a std::runtime_error based on the message argument.
Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>msg</th>
<th>The error message to throw</th>
</tr>
</thead>
</table>

Referenced by matrix_equal().

bool matrix_equal ( const MatrixXi & A, const MatrixXi & B )

Tests whether two Eigen MatrixXi objects are equal.
Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>A</th>
<th>The first matrix to test</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>B</td>
<td>The second matrix to test</td>
</tr>
</tbody>
</table>

Returns

Whether the matrices are equal

bool matrix_equal ( const MatrixXd & A, const MatrixXd & B, double tol )

Tests whether two Eigen MatrixXd objects are equal, within a given tolerance.
Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>A</th>
<th>The first matrix to test</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>B</td>
<td>The second matrix to test</td>
</tr>
<tr>
<td>in</td>
<td>tol</td>
<td>The tolerance to use when comparing double values</td>
</tr>
</tbody>
</table>

Returns

Whether the matrices are equal to within tolerance

References error().

bool matrix_equal ( const RealMatrix & A, const RealMatrix & B, double tol )

Tests whether two Teuchos RealMatrix objects are equal, within a given tolerance.
Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>A</th>
<th>The first matrix to test</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>B</td>
<td>The second matrix to test</td>
</tr>
</tbody>
</table>
### Chapter 13. Namespace Documentation

#### in
tol

The tolerance to use when comparing double values

Returns
- Whether the matrices are equal to within tolerance

References error().

#### double variance (const VectorXd & vec)

Calculates the variance based on an Eigen VectorXd of double values.

**Parameters**

| in  | vec | The vector |

**Returns**
- The calculated variance

#### void populateVectorsFromFile(const std::string & filename, std::vector<VectorXd> & R, int num_datasets, int num_samples)

Populate a collection of vectors read in a from a text file assuming data layout is one dataset per row.

**Parameters**

<table>
<thead>
<tr>
<th>in</th>
<th>filename</th>
<th>The file that contains the data</th>
</tr>
</thead>
<tbody>
<tr>
<td>out</td>
<td>R</td>
<td>The collection of vectors to be populated</td>
</tr>
<tr>
<td>in</td>
<td>num_datasets</td>
<td>The number of datasets to read in</td>
</tr>
<tr>
<td>in</td>
<td>num_samples</td>
<td>The number of data points (e.g. function values, build points) per dataset</td>
</tr>
</tbody>
</table>

#### void populateMatricesFromFile(const std::string & filename, std::vector<MatrixXd> & S, int num_datasets, int num_vars, int num_samples)

Populate a collection of matrices read in a from a text file assuming data layout is a "column-major" stack of num_samples by num_vars matrices.

**Parameters**

<table>
<thead>
<tr>
<th>in</th>
<th>filename</th>
<th>The file that contains the data</th>
</tr>
</thead>
<tbody>
<tr>
<td>out</td>
<td>S</td>
<td>The collection of vectors to be populated</td>
</tr>
<tr>
<td>in</td>
<td>num_datasets</td>
<td>The number of datasets to read in</td>
</tr>
<tr>
<td>in</td>
<td>num_samples</td>
<td>The number of data points (e.g. function values, build points) per dataset (row dim)</td>
</tr>
<tr>
<td>in</td>
<td>num_vars</td>
<td>The number of variables per dataset (column dim)</td>
</tr>
</tbody>
</table>

#### std::shared_ptr<DataScaler> scaler_factory(DataScaler::SCALER_TYPE scaler_type, const MatrixXd & unscaled_matrix)

Free function to construct DataScaler.
13.4. **DAKOTA::UTIL NAMESPACE REFERENCE**

Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>scaler_type</th>
<th>Which scaler to construct</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>unscaled_matrix</td>
<td>Unscaled data matrix - (num_samples by num_features)</td>
</tr>
</tbody>
</table>

Returns

Shared pointer to a DataScaler

Referenced by GaussianProcess::build(), and PolynomialRegression::build().

`std::shared_ptr< LinearSolverBase > solver_factory ( LinearSolverBase::SOLVER_TYPE type )`

Free function to construct LinearSolverBase.

Parameters

| in | type | Which solver to construct |

Returns

Shared pointer to a LinearSolverBase

Referenced by PolynomialRegression::build().

`int n_choose_k ( int n, int k )`

Calculate Binomial coefficient n choose k.

Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>n</th>
<th>Number of elements in set</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>k</td>
<td>Number of elements in subset k where n &gt;= k &gt;= 0</td>
</tr>
</tbody>
</table>

Returns

Number of ways to choose an (unordered) subset of k elements from a fixed set of n elements

Referenced by dakota::surrogates::size_level_index_vector().

`int dakota::util::num_nonzeros ( const T & mat )`

Calculate and return number of nonzero entries in vector or matrix.

Parameters

| in | mat | Incoming vector or matrix |

Returns

Number of nonzeros

Referenced by dakota::surrogates::compute_hyperbolic_level_indices(), dakota::surrogates::compute_hyperbolic_subdim_level_indices(), and nonzero().

`void dakota::util::nonzero ( const T1 & v, T2 & result )`

Create a vector of indices based on nonzero entries in input vector.
190

### CHAPTER 13. NAMESPACE DOCUMENTATION

**Parameters**

<table>
<thead>
<tr>
<th>in</th>
<th>v</th>
<th>Incoming vector</th>
</tr>
</thead>
<tbody>
<tr>
<td>out</td>
<td>result</td>
<td>Vector having values at nonzero locations of incoming vector and value equal to ordinal of occurrence</td>
</tr>
</tbody>
</table>

References `num_nonzeros()`.

Referenced by `dakota::surrogates::compute_hyperbolic_level_indices()`.

---

**void dakota::util::append_columns ( std::vector<T1> & new_cols, std::vector<T2> & target )**

Append columns of input matrix to existing matrix.

**Parameters**

<table>
<thead>
<tr>
<th>in</th>
<th>new_cols</th>
<th>Incoming matrix of column vectors to append</th>
</tr>
</thead>
<tbody>
<tr>
<td>out</td>
<td>target</td>
<td>Matrix to augment with appended columns</td>
</tr>
</tbody>
</table>

Referenced by `dakota::surrogates::compute_hyperbolic_indices()`, `dakota::surrogates::compute_hyperbolic_level_indices()`, and `dakota::surrogates::compute_reduced_indices()`.

---

**double dakota::util::p_norm ( const T & v, double p )**

Calculate and return p-norm of a vector.

**Parameters**

<table>
<thead>
<tr>
<th>in</th>
<th>v</th>
<th>Incoming vector</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>p</td>
<td>Order or norm to compute</td>
</tr>
</tbody>
</table>

Returns

p-norm of incoming vector

Referenced by `dakota::surrogates::compute_hyperbolic_subdim_level_indices()`.

---

### 13.4.3 Variable Documentation

**BimapScalertypeStr type_name_bimap [static]**

**Initial value:**

```cpp
boost::assign::list_of< BimapScalertypeStr::relation >
  (SCALER_TYPE::NONE, "none")
  (SCALER_TYPE::STANDARDIZATION, "standardization")
  (SCALER_TYPE::MEAN_NORMALIZATION, "mean normalization")
  (SCALER_TYPE::MINMAX_NORMALIZATION, "min-max normalization")
```

Bimap between scaler types and names.

Referenced by `DataScaler::scaler_type()`, and `LinearSolverBase::solver_type()`.

---

**BimapSolvertypeStr type_name_bimap [static]**

**Initial value:**

---
Boost::assign::list of < BimapSolverTypeStr::relation >
(SOLVER_TYPE::CHOLESKY, "cholesky")
(SOLVER_TYPE::EQ_CONS_LEAST_SQ_REGRESSION, "equality-constrained lsq regression")
(SOLVER_TYPE::LASSO_REGRESSION, "lasso regression")
(SOLVER_TYPE::LEAST_ANGLE_REGRESSION, "least angle regression")
(SOLVER_TYPE::LU, "LU")
(SOLVER_TYPE::ORTHOG_MATCH_PURSUIT, "orthogonal matching pursuit")
(SOLVER_TYPE::QR_LEAST_SQ_REGRESSION, "QR lsq regression")
(SOLVER_TYPE::SVD_LEAST_SQ_REGRESSION, "SVD")

Bimap between solver types and names.

13.5 SIM Namespace Reference

A sample namespace for derived classes that use assign_rep() to plug facilities into DAKOTA.

Classes

- class ParallelDirectApplicInterface
  
  Sample derived interface class for testing parallel simulator plug-ins using assign_rep().

- class SerialDirectApplicInterface
  
  Sample derived interface class for testing serial simulator plug-ins using assign_rep().

13.5.1 Detailed Description

A sample namespace for derived classes that use assign_rep() to plug facilities into DAKOTA. A typical use of plug-ins with assign_rep() is to publish a simulation interface for use in library mode. See Interfacing with Dakota as a Library for more information.

13.6 StanfordPSAAP Namespace Reference

A sample namespace for derived classes that use assign_rep() to plug facilities into DAKOTA.

Classes

- class SoleilDirectApplicInterface
  
  Sample derived interface class for testing serial simulator plug-ins using assign_rep().

13.6.1 Detailed Description

A sample namespace for derived classes that use assign_rep() to plug facilities into DAKOTA. A typical use of plug-ins with assign_rep() is to publish a simulation interface for use in library mode. See Interfacing with Dakota as a Library for more information.
CHAPTER 13. NAMESPACE DOCUMENTATION
Chapter 14

Class Documentation

14.1 ActiveSet Class Reference

Container class for active set tracking information. Contains the active set request vector and the derivative variables vector.

Public Member Functions

- **ActiveSet ()**
  
  *default constructor*

- **ActiveSet (size_t num_fns, size_t num_deriv_vars)**
  
  *standard constructor*

- **ActiveSet (size_t num_fns)**
  
  *partial constructor*

- **ActiveSet (const ShortArray &asv, const SizetArray &dvv)**
  
  *alt constructor*

- **ActiveSet (const ActiveSet &set)**
  
  *copy constructor*

- **~ActiveSet ()**
  
  *destructor*

- **ActiveSet & operator= (const ActiveSet &set)**
  
  *assignment operator*

- **void reshape (size_t num_fns, size_t num_deriv_vars)**
  
  *reshape requestVector and derivVarsVector*

- **void reshape (size_t num_fns)**
  
  *reshape requestVector*

- **const ShortArray & request_vector () const**
  
  *return the request vector*

- **void request_vector (const ShortArray &rv)**
  
  *set the request vector*

- **void request_values (const short rv_val)**
  
  *set all request vector values*
• short request_value (const size_t index) const
  get the value of an entry in the request vector
• void request_value (const short rv_val, const size_t index)
  set the value of an entry in the request vector
• const SizetArray & derivative_vector () const
  return the derivative variables vector
• void derivative_vector (const SizetArray &dvv)
  set the derivative variables vector from a SizetArray
• void derivative_vector (SizetMultiArrayConstView dvv)
  set the derivative variables vector from a SizetMultiArrayConstView
• void derivative_start_value (size_t dvv_start_val)
  set the derivative variables vector values
• void read (std::istream &s)
  read an active set object from an std::istream
• void write (std::ostream &s) const
  write an active set object to an std::ostream
• void write.annotated (std::ostream &s) const
  write an active set object to an std::ostream in annotated format
• void read (MPIUnpackBuffer &s)
  read an active set object from a packed MPI buffer
• void write (MPIPackBuffer &s) const
  write an active set object to a packed MPI buffer

Private Member Functions

• template<class Archive >
  void serialize (Archive &ar, const unsigned int version)
  implementation of Boost serialize for ActiveSet

Private Attributes

• ShortArray requestVector
  the vector of response requests
• SizetArray derivVarsVector
  the vector of variable ids used for computing derivatives

Friends

• class boost::serialization::access

• bool operator== (const ActiveSet &set1, const ActiveSet &set2)
  equality operator

• bool operator!=(const ActiveSet &set1, const ActiveSet &set2)
  inequality operator
14.1.1 Detailed Description

Container class for active set tracking information. Contains the active set request vector and the derivative variables vector.

The `ActiveSet` class is a small class whose initial design function is to avoid having to pass the ASV and D-VV separately. It is not part of a class hierarchy and does not employ reference-counting/representation-sharing idioms (e.g., handle-body).

14.1.2 Member Data Documentation

`ShortArray requestVector` [private]

the vector of response requests

It uses a 0 value for inactive functions and sums 1 (value), 2 (gradient), and 4 (Hessian) for active functions. Referenced by `ActiveSet::ActiveSet()`, `ActiveSet::operator=()`, `ActiveSet::read()`, `ActiveSet::request_value()`, `ActiveSet::request_values()`, `ActiveSet::request_vector()`, `ActiveSet::reshape()`, `ActiveSet::write()`, and `ActiveSet::write_annotated()`.

`SizetArray derivVarsVector` [private]

the vector of variable ids used for computing derivatives

These ids will generally identify either the active continuous variables or the inactive continuous variables. Referenced by `ActiveSet::ActiveSet()`, `ActiveSet::derivative_start_value()`, `ActiveSet::derivative_vector()`, `ActiveSet::operator=()`, `ActiveSet::read()`, `ActiveSet::reshape()`, `ActiveSet::write()`, and `ActiveSet::write_annotated()`.

The documentation for this class was generated from the following file:

- DakotaActiveSet.hpp

14.2 ActiveSubspaceModel Class Reference

Active subspace model for input (variable space) reduction.

Inheritance diagram for `ActiveSubspaceModel`:

```
Model
RecastModel
ActiveSubspaceModel
```

Public Member Functions

- `ActiveSubspaceModel (ProblemDescDB &problem_db)`
  *Problem database constructor.*
- `ActiveSubspaceModel (const Model &sub_model, unsigned int dimension, const RealMatrix &rotation_matrix, short output_level)`
  *Lightweight constructor*
- `~ActiveSubspaceModel ()`
CHAPTER 14. CLASS DOCUMENTATION

destructor

- bool initialize_mapping (ParLevLIter pl_iter)
- bool resize_pending () const
  
  return true if a potential resize is still pending, such that sizing-based initialization should be deferred

- void stop_init_mapping (ParLevLIter pl_iter)
  
  called from IteratorScheduler::init_iterator() for iteratorComm rank 0 to terminate serve_init_mapping() on other iteratorComm processors

- int serve_init_mapping (ParLevLIter pl_iter)
  
  called from IteratorScheduler::init_iterator() for iteratorComm rank != 0 to balance resize() calls on iteratorComm rank 0

Protected Member Functions

- void derived_init_communicators (ParLevLIter pl_iter, int max_eval_concurrency, bool recurse_flag)

- void derived_set_communicators (ParLevLIter pl_iter, int max_eval_concurrency, bool recurse_flag)
  
  portion of set_communicators() specific to derived model classes

- void derived_free_communicators (ParLevLIter pl_iter, int max_eval_concurrency, bool recurse_flag)
  
  portion of free_communicators() specific to derived model classes

- void derived_evaluate (const ActiveSet &set)
  
  portion of evaluate() specific to derived model classes

- void derived_evaluate_nowait (const ActiveSet &set)
  
  portion of evaluate_nowait() specific to derived model classes

- const IntResponseMap & derived_synchronize ()
  
  portion of synchronize() specific to derived model classes

- const IntResponseMap & derived_synchronize_nowait ()
  
  portion of synchronize_nowait() specific to derived model classes

- void component_parallel_mode (short mode)
  
  update component parallel mode for supporting parallelism in the offline and online phases

- void serve_run (ParLevLIter pl_iter, int max_eval_concurrency)
  
  Service the offline and online phase job requests received from the master; completes when termination message received from stop_servers().

- void stop_servers ()
  
  Executed by the master to terminate the offline and online phase server operations when iteration on the Active-SubspaceModel is complete.

- void assign_instance ()
  
  assign static pointer instance to this for use in static transformation functions

- Model get_sub_model (ProblemDescDB &problem_db)
  
  retrieve the sub-Model from the DB to pass up the constructor chain

- void init_fullspace_sampler (unsigned short sample_type)
  
  initialize the native problem space Monte Carlo sampler

- void validate_inputs ()
  
  validate the build controls and set defaults

- void build_subspace ()
  
  sample the model's gradient, computed the SVD, and form the active subspace rotation matrix.
void initialize_subspace ()
  
  helper for shared code between lightweight ctor and initialize_mapping()

void generate_fullspace_samples (unsigned int diff_samples)
  
  sample the derivative at diff_samples points and leave temporary in dace_iterator

void populate_matrices (unsigned int diff_samples)
  
  populate the derivative and vars matrices with fullspaceSampler samples

void compute_svd ()
  
  factor the derivative matrix and analyze singular values, assessing convergence and rank, returning whether tolerance met

void identify_subspace ()
  
  use the truncation methods to identify the size of an active subspace

unsigned int computeBingLiCriterion (RealVector &singular_values)
  
  compute Bing Li’s criterion to identify the active subspace

unsigned int computeConstantineMetric (RealVector &singular_values)
  
  compute Constantine’s metric to identify the active subspace

unsigned int computeEnergyCriterion (RealVector &singular_values)
  
  Compute active subspace size based on eigenvalue energy. Compatible with other truncation methods.

unsigned int computeCrossValidationMetric ()
  
  Use cross validation of a moving least squares surrogate to identify the size of an active subspace that meets an error tolerance.

Real build_cv_surrogate (Model &cv_surr_model, RealMatrix training_x, IntResponseMap training_y, RealMatrix test_x, IntResponseMap test_y)
  
  Build moving least squares surrogate over candidate active subspace.

unsigned int determine_rank_cv (const std::vector<Real> &cv_error)

unsigned int min_index (const std::vector<Real> &cv_error)

unsigned int tolerance_met_index (const std::vector<Real> &cv_error, Real tolerance, bool &tol_met)

std::vector<Real> negative_diff (const std::vector<Real> &cv_error)

void build_surrogate ()
  
  Build surrogate over active subspace.

void initialize_recast ()
  
  Initialize the base class RecastModel with reduced space variable sizes.

SizetArray variables_resize ()
  
  Create a variables components totals array with the reduced space size for continuous variables.

void uncertain-vars-to-subspace ()
  
  translate the characterization of uncertain variables in the native model to the reduced space of the transformed model

void update_linear_constraints ()
  
  transform the original bounded domain (and any existing linear constraints) into linear constraints in the reduced space

void update_var_labels ()
  
  update variable labels
CHAPTER 14. CLASS DOCUMENTATION

Static Protected Member Functions

- static void `vars_mapping` (const Variables &recast_xi_vars, Variables &sub_model_x_vars)
  map the active continuous recast variables to the active submodel variables (linear transformation)
- static void `set_mapping` (const Variables &recast_vars, const ActiveSet &recast_set, ActiveSet &sub_model_set)
  map the inbound ActiveSet to the sub-model (map derivative variables)
- static void `response_mapping` (const Variables &recast_y_vars, const Variables &sub_model_x_vars, const Response &sub_model_resp, Response &recast_resp)
  map responses from the sub-model to the recast model

Protected Attributes

- int `randomSeed`
  seed controlling all samplers
- int `initialSamples`
  initial number of samples at which to query the truth model
- int `maxFunctionEvals`
  maximum number of build evaluations
- bool `subspaceIdBingLi`
  Boolean flag signaling use of Bing Li criterion to identify active subspace dimension.
- bool `subspaceIdConstantine`
  Boolean flag signaling use of Constantine criterion to identify active subspace dimension.
- bool `subspaceIdEnergy`
  Boolean flag signaling use of eigenvalue energy criterion to identify active subspace dimension.
- bool `subspaceIdCV`
  Boolean flag signaling use of cross validation to identify active subspace dimension.
- size_t `numReplicates`
  Number of bootstrap samples for subspace identification.
- bool `transformVars`
  boolean flag to determine if variables should be transformed to u-space before active subspace initialization
- size_t `numFullspaceVars`
  Number of fullspace active continuous variables.
- unsigned int `totalSamples`
  total construction samples evaluated so far
- unsigned short `subspaceNormalization`
  Normalization to use in the case of multiple QoI's.
- unsigned int `reducedRank`
  current approximation of system rank
- RealMatrix `activeBasis`
  basis for the reduced subspace
- RealMatrix `inactiveBasis`
  basis for the inactive subspace
- RealVector `inactiveVars`
current inactive variables

- **RealMatrix derivativeMatrix**
  
  matrix of derivative data with \( numFns \) columns per fullspace sample; each column contains the gradient of one function at one sample point, so total matrix size is \( numContinuousVars \times (numFns \times numSamples) \)

- **RealMatrix leftSingularVectors**

  matrix of the left singular vectors of derivativeMatrix

- **RealVector singularValues**

  singular values of derivativeMatrix

- **RealMatrix varsMatrix**

  matrix of fullspace variable points samples size \( numContinuousVars \times (numSamples) \)

- **RealArray gradientScaleFactors**

  Gradient scaling factors to make multiple response function gradients similar orders of magnitude.

- **Real truncationTolerance**

  Truncation tolerance for eigenvalue energy subspace identification.

- **bool cvIncremental**

- **short cvIdMethod**

- **Real cvRelTolerance**

- **Real cvDecreaseTolerance**

- **unsigned int cvMaxRank**

  maximum subspace size to consider using cross validation

- **Model surrogateModel**

  model containing a surrogate built over the active subspace

- **bool buildSurrogate**

  flag specifying whether or not a surrogate is built over the subspace

- **int refinementSamples**

  Number of refinement samples to use when building a surrogate.

- **Iterator fullspaceSampler**

  Monte Carlo sampler for the full parameter space.

- **size_t miPLIndex**

  the index of the active metaiterator-iterator parallelism level (corresponding to `ParallelConfiguration::miPLIters`) used at runtime

- **int onlineEvalConcurrency**

  Concurrency to use once subspace has been built.

- **int offlineEvalConcurrency**

  Concurrency to use when building subspace.

- **IntResponseMap surrResponseMap**

  map of responses returned in buildSurrogate mode

- **IntIntMap surrIdMap**

  map from surrogateModel evaluation ids to `RecastModel` ids

### Static Protected Attributes

- **static ActiveSubspaceModel * asmInstance**

  static pointer to this class for use in static callbacks
Additional Inherited Members

14.2.1 Detailed Description

Active subspace model for input (variable space) reduction.

Specialization of a RecastModel that identifies an active subspace during build phase and creates a RecastModel in the reduced space.

14.2.2 Constructor & Destructor Documentation

ActiveSubspaceModel ( const Model & submodel, unsigned int dimension, const RealMatrix & rotation_matrix, short output_level )

lightweight constructor

An ActiveSubspaceModel will be built over all functions, without differentiating primary vs. secondary constraints. However the associated RecastModel has to differentiate. Currently identifies subspace for continuous variables only, but carries other active variables along for the ride.

References ActiveSubspaceModel::activeBasis, Model::componentParallelMode, Model::derivative_concurrency(), ActiveSubspaceModel::inactiveBasis, ActiveSubspaceModel::initialize_subspace(), ActiveSubspaceModel::initialSamples, Model::mappingInitialized, Model::modelId, Model::modelType, ActiveSubspaceModel::numFullspaceVars, ActiveSubspaceModel::offlineEvalConcurrency, ActiveSubspaceModel::onlineEvalConcurrency, Model::output_level(), Model::outputLevel, RecastModel::recast_model_id(), ActiveSubspaceModel::reducedRank, RecastModel::root_model_id(), and RecastModel::subModel.

14.2.3 Member Function Documentation

bool initialize_mapping ( ParLevLIter pl_iter ) [virtual]

May eventually take on init_comms and related operations. Also may want ide of build/update like DataFitSurfModel, eventually.

Reimplemented from Model.

References ActiveSubspaceModel::build_subspace(), ActiveSubspaceModel::component_parallel_mode(), RecastModel::initialize_mapping(), ActiveSubspaceModel::initialize_subspace(), ActiveSubspaceModel::miPLIndex, Model::modelPCIter, ActiveSubspaceModel::numFullspaceVars, Model::outputLevel, and ActiveSubspaceModel::reducedRank.

void derived_init_communicators ( ParLevLIter pl_iter, int max_eval_concurrency, bool recurse_flag ) [protected], [virtual]

This specialization is because the model is used in multiple contexts in this iterator, depending on build phase. Note that this overrides the default behavior at Iterator which recurses into any submodels.

Reimplemented from Model.

References ActiveSubspaceModel::fullspaceSampler, Iterator::init_communicators(), Model::init_communicators(), Model::mappingInitialized, ActiveSubspaceModel::onlineEvalConcurrency, and RecastModel::subModel.

Real build_cv_surrogate ( Model & cv_surr_model, RealMatrix training_x, IntResponseMap training_y, RealMatrix test_x, IntResponseMap test_y ) [protected]

Build moving least squares surrogate over candidate active subspace.

Build global moving least squares surrogate model to use in cross validation to estimate active subspace size.

References Response::active_set(), Model::continuous_variables(), Response::copy(), Model::current_response(), Model::evaluate(), Model::numFns, and Model::update_approximation().
14.2. ACTIVESUBSPACEMODEL CLASS REFERENCE

Referenced by ActiveSubspaceModel::computeCrossValidationMetric().

```cpp
void build_surrogate() [protected]
```

Build surrogate over active subspace.

References Response::active_set(), Iterator::active_set_request_values(), ActiveSubspaceModel::activeBasis, Iterator::all_responses(), Iterator::all_samples(), Model::append_approximation(), Model::assign_rep(), Model::current_response(), ActiveSubspaceModel::fullspaceSampler, ActiveSubspaceModel::miPLIndex, Model::modelPCIter, Model::outputLevel, ActiveSubspaceModel::reducedRank, ActiveSubspaceModel::refinementSamples, Iterator::run(), Iterator::sampling_reference(), Iterator::sampling_reset(), and ActiveSubspaceModel::surrogateModel.

Referenced by ActiveSubspaceModel::initialize_subspace().

```cpp
void initialize_recast() [protected]
```

Initialize the base class RecastModel with reduced space variable sizes.

Initialize the recast model based on the reduced space, with no response function mapping (for now). TODO: use a surrogate model over the inactive dimension.

References Model::current_response(), Model::cv(), Model::div(), Model::drv(), Model::dsv(), Response::function_gradients(), Response::function_hessians(), RecastModel::init_maps(), RecastModel::init_sizes(), Model::num_nonlinear_ineq_constraints(), Model::num_primary_fns(), Model::num_secondary_fns(), Model::numFns, ActiveSubspaceModel::reducedRank, ActiveSubspaceModel::response_mapping(), ActiveSubspaceModel::set_mapping(), RecastModel::subModel, ActiveSubspaceModel::variables_resize(), and ActiveSubspaceModel::vars_mapping().

Referenced by ActiveSubspaceModel::initialize_subspace().

```cpp
void uncertain_vars_to_subspace() [protected]
```

Translate the characterization of uncertain variables in the native model to the reduced space of the transformed model.

Transform and set the distribution parameters in the reduced model.

Convert the user-specified normal random variables to the appropriate reduced space variables, based on the orthogonal transformation.

TODO: Generalize to convert other random variable types (non-normal)

TODO: The translation of the correlations from full to reduced space is likely wrong for rank correlations; should be correct for covariance.

Referenced by ActiveSubspaceModel::initialize_subspace().

```cpp
void vars_mapping(const Variables & recast_y_vars, Variables & sub_model_x_vars) [static], [protected]
```

Map the active continuous recast variables to the active submodel variables (linear transformation).

Perform the variables mapping from recast reduced dimension variables y to original model x variables via linear transformation. Maps only continuous variables.

References ActiveSubspaceModel::activeBasis, ActiveSubspaceModel::asmInstance, Variables::continuous_variables(), Dakota::copy_data(), ActiveSubspaceModel::inactiveBasis, ActiveSubspaceModel::inactiveVars, and Model::outputLevel.

Referenced by ActiveSubspaceModel::initialize_recast().
void set_mapping ( const Variables & recast_vars, const ActiveSet & recast_set, ActiveSet & sub_model_set ) [static], [protected]

map the inbound ActiveSet to the sub-model (map derivative variables)
   Simplified derivative variables mapping where all continuous depend on all others. TODO: Could instead rely on a richer default in RecastModel based on varsMapIndices.
   References ActiveSubspaceModel::asmInstance, Variables::cv(), Model::cv(), ActiveSet::derivative_vector(), and RecastModel::subModel.
   Referenced by ActiveSubspaceModel::initialize_recast().

void response_mapping ( const Variables & recast_y_vars, const Variables & sub_model_x_vars, const Response & sub_model.resp, Response & recast.resp ) [static], [protected]

map responses from the sub-model to the recast model
   Perform the response mapping from submodel to recast response
   References ActiveSubspaceModel::activeBasis, ActiveSubspaceModel::asmInstance, Response::function_gradients(), Response::function_hessians(), and Response::function_values().
   Referenced by ActiveSubspaceModel::initialize_recast().

14.2.4 Member Data Documentation

ActiveSubspaceModel * asmInstance [static], [protected]

static pointer to this class for use in static callbacks
   initialization of static needed by RecastModel
   Referenced by ActiveSubspaceModel::assign_instance(), ActiveSubspaceModel::response_mapping(), ActiveSubspaceModel::set_mapping(), and ActiveSubspaceModel::vars_mapping().
   The documentation for this class was generated from the following files:
   • ActiveSubspaceModel.hpp
   • ActiveSubspaceModel.cpp

14.3 AdaptedBasisModel Class Reference

Adapted basis model for input (variable space) reduction.
   Inheritance diagram for AdaptedBasisModel:

   Model
   |
   V
   RecastModel
   |
   V
   AdaptedBasisModel

Public Member Functions

• AdaptedBasisModel (ProblemDescDB &problem_db)
  Problem database constructor.
• \texttt{~AdaptedBasisModel} ()
  destructor
• bool \texttt{initialize\_mapping} (ParLevLIter pl\_iter)
  Perform any global updates prior to individual \texttt{evaluate}() calls; returns true if the variables size has changed.
• bool \texttt{finalize\_mapping} ()
  restore state in preparation for next initialization; returns true if the variables size has changed.
• bool \texttt{mapping\_initialized} () const
• void \texttt{stop\_init\_mapping} (ParLevLIter pl\_iter)
  called from \texttt{IteratorScheduler::init\_iterator()} for iteratorComm rank 0 to terminate \texttt{serve\_init\_mapping()} on other iteratorComm processors
• int \texttt{serve\_init\_mapping} (ParLevLIter pl\_iter)
  called from \texttt{IteratorScheduler::init\_iterator()} for iteratorComm rank \( \neq 0 \) to balance \texttt{resize()} calls on iteratorComm rank 0

Protected Member Functions

• void \texttt{derived\_init\_communicators} (ParLevLIter pl\_iter, int max\_eval\_concurrency, bool recurse\_flag)
  portion of \texttt{set\_communicators()} specific to derived model classes
• void \texttt{derived\_set\_communicators} (ParLevLIter pl\_iter, int max\_eval\_concurrency, bool recurse\_flag)
  portion of \texttt{set\_communicators()} specific to derived model classes
• void \texttt{derived\_free\_communicators} (ParLevLIter pl\_iter, int max\_eval\_concurrency, bool recurse\_flag)
  portion of \texttt{free\_communicators()} specific to derived model classes
• void \texttt{derived\_evaluate} (const ActiveSet &set)
  portion of \texttt{evaluate()} specific to derived model classes
• void \texttt{derived\_evaluate\_nowait} (const ActiveSet &set)
  portion of \texttt{evaluate\_nowait()} specific to derived model classes
• const IntResponseMap & \texttt{derived\_synchronize} ()
  portion of \texttt{synchronize()} specific to derived model classes
• const IntResponseMap & \texttt{derived\_synchronize\_nowait} ()
  portion of \texttt{synchronize\_nowait()} specific to derived model classes
• void \texttt{component\_parallel\_mode} (short mode)
  update component parallel mode for supporting parallelism in the offline and online phases
• void \texttt{serve\_run} (ParLevLIter pl\_iter, int max\_eval\_concurrency)
  Service the offline and online phase job requests received from the master; completes when termination message received from \texttt{stop\_servers()}.
• void \texttt{stop\_servers} ()
  Executed by the master to terminate the offline and online phase server operations when iteration on the \texttt{AdaptedBasisModel} is complete.
• void \texttt{assign\_instance} ()
  assign static pointer instance to this for use in static transformation functions
• Model \texttt{get\_sub\_model} (ProblemDescDB &problem\_db)
  retrieve the sub-Model from the DB to pass up the constructor chain
• void \texttt{validate\_inputs} ()
  validate the build controls and set defaults
• void \texttt{identify\_subspace} ()
void initialize_recast()

Initialize the base class RecastModel with reduced space variable sizes.

SizetArray variables_resize()

Create a variables components totals array with the reduced space size for continuous variables.

void uncertain_vars_to_subspace()

translate the characterization of uncertain variables in the native_model to the reduced space of the transformed model

Static Protected Member Functions

static void vars_mapping (const Variables &recast_xi_vars, Variables &sub_model_x_vars)

map the active continuous recast variables to the active submodel variables (linear transformation)

static void set_mapping (const Variables &recast_vars, const ActiveSet &recast_set, ActiveSet &sub_model_set)

map the inbound ActiveSet to the sub-model (map derivative variables)

static void response_mapping (const Variables &recast_y_vars, const Variables &sub_model_x_vars, const Response &sub_model_resp, Response &recast_resp)

map responses from the sub-model to the recast model

Protected Attributes

Iterator * pcePilotExpRepPtr

PCE representation pointer that is initialized in get_sub_model() and then assigned into pcePilotExpansion in the constructor initializer list.

Iterator pcePilotExpansion

low-order (linear or quadratic) PCE generator for computing rotation matrices $A_i$ for each of the QoI; this is low-order and potentially high-dimension whereas a client PCE could be high-order in the reduced dimension

size_t numFullspaceVars

Number of fullspace active continuous variables.

size_t numFunctions

Total number of response functions.

bool adaptedBasisInitialized

boolean flag to determine if mapping has been fully initialized

unsigned int reducedRank

current approximation of system rank

RealMatrix rotationMatrix

basis for the reduced subspace

size_t miPLIndex

the index of the active metaiterator-iterator parallelism level (corresponding to ParallelConfiguration::miPLIteS) used at runtime

int onlineEvalConcurrency

Concurrency to use once subspace has been built.

int offlineEvalConcurrency

Concurrency to use when building subspace.
14.3. ADAPTEDBASISMODEL CLASS REFERENCE

Static Protected Attributes

- static AdaptedBasisModel * abmInstance
  static pointer to this class for use in static callbacks

Additional Inherited Members

14.3.1 Detailed Description

Adapted basis model for input (variable space) reduction.

Specialization of a RecastModel that creates an adapted basis model during build phase and creates a RecastModel in the reduced space.

14.3.2 Member Function Documentation

void derived_init_communicators ( ParLevLIter pl_iter, int max_eval_concurrency, bool recurse_flag ) [inline], [protected], [virtual]

This specialization is because the model is used in multiple contexts depending on build phase.
Reimplemented from Model.
References Iterator::init_communicators(), Model::init_communicators(), AdaptedBasisModel::onlineEvalConcurrency, AdaptedBasisModel::pcePilotExpansion, and RecastModel::subModel.

void initialize_recast ( ) [protected]

Initialize the base class RecastModel with reduced space variable sizes.
Initialize the recast model based on the reduced space, with no response function mapping (for now).
References Model::current_response(), Model::cv(), Model::div(), Model::drv(), Model::dsv(), Response::function_gradients(), Response::function_hessians(), RecastModel::init_maps(), RecastModel::init_sizes(), Model::num_nonlinear_ineq_constraints(), Model::num_primary_fns(), Model::num_secondary_fns(), AdaptedBasisModel::reduced-Rank, AdaptedBasisModel::response_mapping(), AdaptedBasisModel::set_mapping(), RecastModel::subModel, AdaptedBasisModel::variables_resize(), and AdaptedBasisModel::vars_mapping().
Referenced by AdaptedBasisModel::initialize_mapping().

void uncertain_vars_to_subspace ( ) [protected]

translate the characterization of uncertain variables in the native_model to the reduced space of the transformed model.
Define the distribution of recast reduced dimension variables. They are standard Gaussian in adapted basis model.
References Model::continuous_variable_labels(), Variables::continuous_variable_types(), Model::continuous_variables(), Model::currentVariables, Model::mvDist, Model::outputLevel, and AdaptedBasisModel::reduced-Rank.
Referenced by AdaptedBasisModel::initialize_mapping().

void vars_mapping ( const Variables & recast_eta_vars, Variables & sub_model_xi_vars ) [static], [protected]

map the active continuous recast variables to the active submodel variables (linear transformation).
Perform the variables mapping from recast reduced dimension variables to original model variables via linear transformation. Maps only continuous variables.
References AdaptedBasisModel::abmInstance, Variables::continuous_variables(), Variables::continuous_variables_view(), Model::outputLevel, and AdaptedBasisModel::rotationMatrix.
Referenced by AdaptedBasisModel::initialize_recast().

```cpp
void set_mapping ( const Variables & recast_vars, const ActiveSet & recast_set, ActiveSet & sub_model_set ) [static], [protected]
```

map the inbound `ActiveSet` to the sub-model (map derivative variables)
Simplified derivative variables mapping where all continuous depend on all others. TODO: Could instead rely on a richer default in `RecastModel` based on `varsMapIndices`.
References AdaptedBasisModel::abmInstance, Variables::cv(), Model::cv(), ActiveSet::derivative_vector(), and RecastModel::subModel.
Referenced by AdaptedBasisModel::initialize_recast().

```cpp
void response_mapping ( const Variables & recast_eta_vars, const Variables & sub_model_xi_vars, const Response & sub_model_resp, Response & recast_resp ) [static], [protected]
```

map responses from the sub-model to the recast model
Perform the response mapping from submodel to recast response
References AdaptedBasisModel::abmInstance, Response::function_gradients(), Response::function_hessians(), Response::function_values(), and AdaptedBasisModel::rotationMatrix.
Referenced by AdaptedBasisModel::initialize_recast().

### 14.3.3 Member Data Documentation

**AdaptedBasisModel ∗ abmInstance** [static], [protected]

static pointer to this class for use in static callbacks
initialization of static needed by `RecastModel`
Referenced by AdaptedBasisModel::assign_instance(), AdaptedBasisModel::response_mapping(), AdaptedBasisModel::set_mapping(), and AdaptedBasisModel::vars_mapping().
The documentation for this class was generated from the following files:

- `AdaptedBasisModel.hpp`
- `AdaptedBasisModel.cpp`

### 14.4 AddAttributeVisitor Class Reference

Objects of this class are called by boost::apply_visitor to add attributes to HDF5 objects.
Inherits static_visitor<>.

#### Public Member Functions

- **AddAttributeVisitor** (const String &location, const std::shared_ptr<HDF5IOHelper> &hdf5_stream)

  The attributes will be added to the HDF5 object at location, using the HDF5IOHelper instance hdf5_stream.

- template<typename T>

  void operator() (const ResultAttribute<T> &a) const

  Called by boost::apply_visitor to process a ResultAttribute.
**Private Attributes**

- **String location**
  
  *Link name of the HDF5 object to add attributes to.*
- **std::shared_ptr<HDF5IOHelper> hdf5Stream**
  
  *HDF5IOHelper instance.*

**14.4.1 Detailed Description**

Objects of this class are called by boost::apply_visitor to add attributes to HDF5 objects.

The documentation for this class was generated from the following file:

- ResultsDBHDF5.hpp

**14.5 Analyzer Class Reference**

Base class for NonD, DACE, and ParamStudy branches of the iterator hierarchy.

Inheritance diagram for Analyzer:

```
Analyzer
  
  Iterator
    
    NonD
      
      NonDCalibration
        
        NonDExpansion
          
          NonDIntegration
            
            NonDInterval
              
              NonDPOFDarts
                
                NonDReliability
                  
                  NonDRKDDarts
                    
                    NonDSampling
                      
                      PSStudyDACE
                        
                        DDACEDesignCompExp
                          
                          RichExtrapVerification
                            
                            FSUDesignCompExp
                              
                              ParamStudy
                                
                                PSUADEDesignCompExp

  
  Verification
```

**Public Member Functions**

- **const VariablesArray & all_variables ()**
  
  *return the complete set of evaluated variables*
- **const RealMatrix & all_samples ()**
  
  *return the complete set of evaluated samples*
- **const IntResponseMap & all_responses () const**
  
  *return the complete set of computed responses*
- **bool resize ()**
reinitializes iterator based on new variable size

- int num_samples () const
- virtual void vary_pattern (bool pattern_flag)
  sets varyPattern in derived classes that support it

Protected Member Functions

- Analyzer ()
  default constructor
- Analyzer (ProblemDescDB &problem_db, Model &model)
  standard constructor
- Analyzer (unsigned short method_name, Model &model)
  alternate constructor for instantiations "on the fly" with a Model
- Analyzer (unsigned short method_name)
  alternate constructor for instantiations "on the fly" without a Model
- ~Analyzer ()
  destructor
- virtual void get_parameter_sets (Model &model)
  Generate one block of numSamples samples (ndim * num_samples), populating allSamples; ParamStudy is the only class that specializes to use allVariables.
- virtual void get_parameter_sets (Model &model, const int num_samples, RealMatrix &design_matrix)
  Generate one block of numSamples samples (ndim * num_samples), populating design_matrix.
- virtual void update_model_from_sample (Model &model, const Real *sample_vars)
  update model’s current variables with data from sample
- virtual void update_model_from_variables (Model &model, const Variables &vars)
  update model’s current variables with data from vars
- virtual void sample_to_variables (const Real *sample_vars, Variables &vars)
  convert column of samples array to variables; derived classes may reimplement for more than active continuous variables
- void update_from_model (const Model &model)
  set inherited data attributes based on extractions from incoming model
- void initialize_run ()
  utility function to perform common operations prior to pre_run(); typically memory initialization; setting of instance pointers
- void pre_run ()
  pre-run portion of run (optional); re-implemented by Iterators which can generate all Variables (parameter sets) a priori
- void post_run (std::ostream &s)
  post-run portion of run (optional); verbose to print results; re-implemented by Iterators that can read all Variables/Responses and perform final analysis phase in a standalone way
- void finalize_run ()
  utility function to perform common operations following post_run(); deallocation and resetting of instance pointers
- void pre_output ()
- void print_results (std::ostream &s, short results_state=FINAL_RESULTS)
  print the final iterator results
• const Model & algorithm_space_model () const
• const Variables & variables_results () const
  return a single final iterator solution (variables)
• const Response & response_results () const
  return a single final iterator solution (response)
• const VariablesArray & variables_array_results ()
  return multiple final iterator solutions (variables). This should only be used if returns_multiple_points() returns true.
• const ResponseArray & response_array_results ()
  return multiple final iterator solutions (response). This should only be used if returns_multiple_points() returns true.
• void response_results_active_set (const ActiveSet &set)
  set the requested data for the final iterator response results
• bool compact_mode () const
  returns Analyzer::compactMode
• bool returns_multiple_points () const
  indicates if this iterator returns multiple final points. Default return is false. Override to return true if appropriate.
• void evaluate_parameter_sets (Model &model, bool log_resp_flag, bool log_best_flag)
  perform function evaluations to map parameter sets (allVariables) into response sets (allResponses)
• void get_vbd_parameter_sets (Model &model, int num_samples)
  generate replicate parameter sets for use in variance-based decomposition
• void compute_vbd_stats (const int num_samples, const IntResponseMap &resp_samples)
  compute VBD-based Sobol indices
• void archive sobol_indices () const
  archive VBD-based Sobol indices
• virtual void archive_model_variables (const Model &, size_t idx) const
  archive model evaluation points
• virtual void archive_model_response (const Response &, size_t idx) const
  archive model evaluation responses
• void read_variables_responses (int num_evals, size_t num_vars)
  convenience function for reading variables/responses (used in derived classes post_input)
• void print_sobol_indices (std::ostream &s) const
  Printing of VBD results.
• void samples_to_variables_array (const RealMatrix &sample_matrix, VariablesArray &vars_array)
  convert samples array to variables array; e.g., allSamples to allVariables
• virtual void variables_to_sample (const Variables &vars, Real *sample_c_vars)
  convert the active continuous variables into a column of allSamples
• void variables_array_to_samples (const VariablesArray &vars_array, RealMatrix &sample_matrix)
  convert variables array to samples array; e.g., allVariables to allSamples
### Protected Attributes

- `size_t numFunctions`  
  number of response functions
- `size_t numContinuousVars`  
  number of active continuous vars
- `size_t numDiscreteIntVars`  
  number of active discrete integer vars
- `size_t numDiscreteStringVars`  
  number of active discrete string vars
- `size_t numDiscreteRealVars`  
  number of active discrete real vars
- `bool compactMode`  
  switch for allSamples (compact mode) instead of allVariables (normal mode)
- `VariablesArray allVariables`  
  array of all variables to be evaluated in `evaluate_parameter_sets()`
- `RealMatrix allSamples`  
  compact alternative to allVariables
- `IntResponseMap allResponses`  
  array of all responses to be computed in `evaluate_parameter_sets()`
- `StringArray allHeaders`  
  array of headers to insert into output while evaluating allVariables
- `size_t numObjFns`  
  number of objective functions
- `size_t numLSqTerms`  
  number of least squares terms
- `RealPairPRPMultiMap bestVarsRespMap`  
  map which stores best set of solutions

### Private Member Functions

- `void compute_best_metrics (const Response &response, std::pair<Real, Real> &metrics)`  
  compares current evaluation to best evaluation and updates best
- `void update_best (const Variables &vars, int eval_id, const Response &response)`  
  compares current evaluation to best evaluation and updates best
- `void update_best (const Real *sample_c_vars, int eval_id, const Response &response)`  
  compares current evaluation to best evaluation and updates best

### Private Attributes

- `int writePrecision`  
  write precision as specified by the user
- `Real vbdDropTol`  
  tolerance for omitting output of small VBD indices
- `RealVectorArray S4`
14.5. ANALYZER CLASS REFERENCE

VBD main effect indices.
- RealVectorArray T4

VBD total effect indices.

Additional Inherited Members

14.5.1 Detailed Description

Base class for NonD, DACE, and ParamStudy branches of the iterator hierarchy.

The Analyzer class provides common data and functionality for various types of systems analysis, including nondeterministic analysis, design of experiments, and parameter studies.

14.5.2 Member Function Documentation

int num_samples() const [inline], [virtual]

Return current number of evaluation points. Since the calculation of samples, collocation points, etc. might be costly, provide a default implementation here that backs out from the maxEvalConcurrency.

Reimplemented from Iterator.
Reimplemented in NonDSampling, NonDQuadrature, NonDSparseGrid, NonDCubature, DDACEDesignCompExp, FSUDESIGNCompExp, and PSUADEDesignCompExp.

References Model::derivative_concurrency(), Iterator::iteratedModel, and Iterator::maxEvalConcurrency.

void sample_to_variables(const Real *sample_c_vars, Variables &vars) [protected], [virtual]

convert column of samples array to variables; derived classes may reimplement for more than active continuous variables

Default mapping that maps into continuous part of Variables only
Reimplemented in NonDSampling.

References Variables::adiv(), Variables::adriv(), Variables::all_discrete_int_variables(), Variables::all_discrete_real_variables(), Variables::continuous_variable(), Model::current_variables(), Variables::inactive_continuous_variables(), Variables::is_null(), Iterator::iteratedModel, Analyzer::numContinuousVars, and Variables::shared_data().

Referenced by NonDLHSEvidence::post_process_samples(), Analyzer::pre_output(), Analyzer::samples_to_variables_array(), and Analyzer::update_best().

void initialize_run() [protected], [virtual]

utility function to perform common operations prior to pre_run(); typically memory initialization; setting of instance pointers

Perform initialization phases of run sequence, like allocating memory and setting instance pointers. Commonly used in sub-iterator executions. This is a virtual function; when re-implementing, a derived class must call its nearest parent’s initialize_run(), typically before performing its own implementation steps.

Reimplemented from Iterator.
Reimplemented in NonD.

References Model::initialize_mapping(), Model::is_null(), Iterator::iteratedModel, Iterator::methodPCIter, Analyzer::resize(), Model::set_evaluation_reference(), and Iterator::summaryOutputFlag.

Referenced by NonD::initialize_run().
void pre_run ( ) [protected], [virtual]

pre-run portion of run (optional); re-implemented by Iterators which can generate all Variables (parameter sets) a priori

pre-run phase, which a derived iterator may optionally reimplement; when not present, pre-run is likely integrated into the derived run function. This is a virtual function; when re-implementing, a derived class must call its nearest parent’s pre_run(), if implemented, typically before performing its own implementation steps.

Reimplemented from Iterator.

Reimplemented in NonDSampling, NonDBayesCalibration, NonDLHSSampling, NonDLocalReliability, DACEDesignCompExp, NonDRKDDarts, FSUDesignCompExp, ParamStudy, NonDMultilevelSampling, PSUADEDesignCompExp, and NonDGlobalReliability.

References Analyzer::bestVarsRespMap.

Referenced by NonDGlobalReliability::pre_run(), PSUADEDesignCompExp::pre_run(), ParamStudy::pre_run(), FSUDesignCompExp::pre_run(), NonDRKDDarts::pre_run(), DDACEDesignCompExp::pre_run(), NonDLocalReliability::pre_run(), NonDBayesCalibration::pre_run(), and NonDSampling::pre_run().

void post_run ( std::ostream & s ) [protected], [virtual]

post-run portion of run (optional); verbose to print results; re-implemented by Iterators that can read all Variables/Responses and perform final analysis phase in a standalone way

Post-run phase, which a derived iterator may optionally reimplement; when not present, post-run is likely integrated into run. This is a virtual function; when re-implementing, a derived class must call its nearest parent’s post_run(), typically after performing its own implementation steps.

Reimplemented from Iterator.

Reimplemented in NonDRKDDarts, NonDLHSSampling, DDACEDesignCompExp, FSUDesignCompExp, NonDReliability, ParamStudy, PSUADEDesignCompExp, and NonDMultilevelSampling.

References Model::is_null(), Iterator::iteratedModel, Model::print_evaluation_summary(), Analyzer::print_results(), and Iterator::summaryOutputFlag.

Referenced by NonDMultilevelSampling::post_run(), PSUADEDesignCompExp::post_run(), NonDReliability::post_run(), ParamStudy::post_run(), FSUDesignCompExp::post_run(), DDACEDesignCompExp::post_run(), NonDLHSSampling::post_run(), and NonDRKDDarts::post_run().

void finalize_run ( ) [protected], [virtual]

utility function to perform common operations following post_run(); deallocation and resetting of instance pointers

Optional: perform finalization phases of run sequence, like deallocating memory and resetting instance pointers. Commonly used in sub-iterator executions. This is a virtual function; when re-implementing, a derived class must call its nearest parent’s finalize_run(), typically after performing its own implementation steps.

Reimplemented from Iterator.

Reimplemented in NonD.

References Model::finalize_mapping(), Iterator::finalize_run(), Iterator::iteratedModel, and Analyzer::resize().

Referenced by NonD::finalize_run().

void pre_output ( ) [protected], [virtual]

Generate tabular output with active variables (compactMode) or all variables with their labels and response labels, with no data. Variables are sequenced \{cv, div, drv\}

Reimplemented from Iterator.
14.5. ANALYZER CLASS REFERENCE

References Analyzer::allSamples, Analyzer::allVariables, ParallelLibrary::command_line::pre_run::output(), ParallelLibrary::command_line::user::modes(), Analyzer::compactMode, Variables::copy(), Model::current_response(), Model::current_variables(), Model::interface::id(), Iterator::iteratedModel, Iterator::outputLevel, Iterator::parallelLib, ProgramOptions::pre_run::output::format(), ParallelLibrary::program::options(), Analyzer::sample::to::variables(), Dakota::write::precision, Variables::write::tabular(), and Analyzer::write::Precision.

```cpp
void print_results ( std::ostream & s, short results::state = FINAL_RESULTS ) [protected], [virtual]
```

print the final iterator results

This virtual function provides additional iterator-specific final results outputs beyond the function evaluation summary printed in finalize::run().

Reimplemented from Iterator.

Reimplemented in NonDPolynomialChaos, NonDGPMSSABayesCalibration, NonDLHSSampling, NonDMultilevelPolynomialChaos, NonDBayesCalibration, NonDMultilevelFunctionTrain, NonDPofDarts, NonDQUESOBayesCalibration, NonDMultilevelStochCollocation, NonDAdaptiveSampling, NonDWasABI2BayesCalibration, NonDAdaptiveSampling, NonDLocalReliability, NonDExpansion, NonDGpImpSampling, NonDMUQBayesCalibration, NonDInterval, NonDMultilevelSampling, PStudy::DACE, NonDGlobalReliability, Verification, and RichExtrapolation.

References Analyzer::best::Vars::Resp::Map, ParamResponsePair::eval::id(), Response::function::values(), Analyzer::num::LSq::Terms, Analyzer::num::Obj::Fns, ParamResponsePair::response(), and ParamResponsePair::variables().

Referenced by Analyzer::post::run(), Verification::print::results(), PStudy::DACE::print::results(), and NonDLHSSampling::print::results().

```cpp
const Model & algorithm::space::model ( ) const [inline], [protected], [virtual]
```

default definition that gets redefined in selected derived Minimizers

Reimplemented from Iterator.

Reimplemented in NonDBayesCalibration, NonDExpansion, NonDReliability, and NonDGlobalInterval.

References Iterator::iteratedModel.

```cpp
void evaluate::parameter::sets ( Model & model, bool log resp flag, bool log best flag ) [protected]
```

perform function evaluations to map parameter sets (allVariables) into response sets (allResponses)

Convenience function for derived classes with sets of function evaluations to perform (e.g., NonDSampling, DDACE::Design::Comp::Exp, PSUADE::Design::Comp::Exp, ParamStudy).

References Results::Manager::active(), Iterator::active::Set, Analyzer::all::Headers, Analyzer::all::Responses, Analyzer::all::Samples, Analyzer::all::Variables, Analyzer::archive::model::response(), Analyzer::archive::model::variables(), Model::asynch::flag(), Analyzer::compact::Mode, Response::copy(), Model::current::response(), Model::current::variables(), Model::evaluate(), Model::evaluate::nowait(), Model::evaluation::id(), Iterator::results::DB, Model::synchronize(), Analyzer::update::best(), Analyzer::update::model::from::sample(), and Analyzer::update::model::from::variables().

Referenced by ParamStudy::core::run(), PSUADE::Design::Comp::Exp::core::run(), FSU::Design::Comp::Exp::core::run(), DDACE::Design::Comp::Exp::core::run(), NonDAdaptive::Sampling::core::run(), NonDLHSSampling::core::run(), NonDI::Integration::core::run(), NonDSampling::core::run(), NonDSparse::Grid::evaluate::grid::increment(), NonDQuadrature::evaluate::grid::increment(), NonDSparse::Grid::evaluate::set(), NonDMultilevel::Sampling::if::increment(), NonDMultilevel::Sampling::multilevel::control::variate::mc::Qcorr(), NonDMultilevel::Sampling::multilevel::control::variate::mc::Ycorr(), NonDMultilevel::Sampling::multilevel::mc::Qsum(), NonDMultilevel::Sampling::multilevel::mc::Ysum(), NonDMultilevel::Sampling::shared::increment(), and NonDQUESOBayes::Calibration::update::model().
void get_vbd_parameter_sets ( Model & model, int num_samples ) [protected]
generate replicate parameter sets for use in variance-based decomposition

    Generate (numvars + 2) * num_samples replicate sets for VBD, populating allSamples( numvars, (numvars + 2) * num_samples )

References Dakota::abort_handler(), Analyzer::allSamples, Analyzer::compactMode, Analyzer::get_parameter_sets(), Analyzer::num_samples(), Analyzer::numContinuousVars, Analyzer::numDiscreteIntVars, Analyzer::numDiscreteRealVars, Analyzer::numDiscreteStringVars, and Analyzer::vary_pattern().

Referenced by FSUDesignCompExp::pre_run(), DDACEDesignCompExp::pre_run(), and NonDLHSSampling::pre_run().

void compute_vbd_stats ( const int num_samples, const IntResponseMap & resp_samples ) [protected]
compute VBD-based Sobol indices

    Calculation of sensitivity indices obtained by variance based decomposition. These indices are obtained by the Saltelli version of the Sobol VBD which uses (K+2)*N function evaluations, where K is the number of dimensions (uncertain vars) and N is the number of samples.

References Dakota::abort_handler(), Analyzer::allSamples, Analyzer::num_samples(), Analyzer::numContinuousVars, Analyzer::numDiscreteIntVars, Analyzer::numDiscreteRealVars, Analyzer::numDiscreteStringVars, Analyzer::numFunctions, Analyzer::S4, and Analyzer::T4.

Referenced by FSUDesignCompExp::post_run(), DDACEDesignCompExp::post_run(), and NonDLHSSampling::post_run().

void archive_sobol_indices ( ) const [protected]
archive VBD-based Sobol indices

    printing of variance based decomposition indices.

References ResultsManager::active(), Model::continuous_variable_labels(), Model::discrete_int_variable_labels(), Model::discrete_real_variable_labels(), ResultsManager::insert(), Iterator::iteratedModel, main(), Analyzer::numContinuousVars, Analyzer::numDiscreteIntVars, Analyzer::numDiscreteRealVars, Analyzer::numDiscreteStringVars, Analyzer::numFunctions, Model::response_labels(), Iterator::resultsDB, Iterator::run_identifier(), Analyzer::S4, Analyzer::T4, and Analyzer::vbdDropTol.

Referenced by NonDLHSSampling::post_run().

void read_variables_responses ( int num_evals, size_t num_vars ) [protected]
convenience function for reading variables/responses (used in derived classes post_input)

    read num_evals variables/responses from file

References Dakota::abort_handler(), Analyzer::allResponses, Analyzer::allSamples, Analyzer::allVariables, ParallelLibrary::command_line_post_run_input(), ParallelLibrary::command_line_user_modes(), Analyzer::compactMode, Response::copy(), Variables::copy(), Model::current_response(), Model::current_variables(), Dakota::data_pairs, ParamResponsePair::eval_id(), Model::evaluation_cache(), Iterator::iteratedModel, Model::manage_data_recastings(), Analyzer::numLSqTerms, Analyzer::numObjFns, Iterator::outputLevel, Iterator::parallelLib, ProgramOptions::post_run_input_format(), ParallelLibrary::program_options(), ParamResponsePair::response(), Model::restart_file(), Analyzer::update_best(), Model::user_space_to_iterator_space(), ParamResponsePair::variables(), Analyzer::variables_to_sample(), and ParallelLibrary::write_restart().

Referenced by PSUADEDesignCompExp::post_input(), ParamStudy::post_input(), FSUDesignCompExp::post_input(), DDACEDesignCompExp::post_input(), and NonDLHSSampling::post_input().
void print_sobol_indices ( std::ostream & s ) const [protected]

Printing of VBD results.
    printing of variance based decomposition indices.
    References Model::continuous_variable_labels(), Model::discrete_int_variable_labels(), Model::discrete_real_variable_labels(), Iterator::iteratedModel, main(), Analyzer::numContinuousVars, Analyzer::numDiscreteIntVars, Analyzer::numDiscreteRealVars, Analyzer::numFunctions, Model::response_labels(), Analyzer::S4, Analyzer::-T4, Analyzer::vbdDropTol, and Dakota::write_precision.
    Referenced by PStudyDACE::print_results(), and NonDLHSSampling::print_results().

void variables_to_sample ( const Variables & vars, Real * sample_c vars ) [protected], [virtual]

convert the active continuous variables into a column of allSamples
    Default implementation maps active continuous variables only
    Reimplemented in NonDSampling.
    References Variables::continuous_variables(), and Analyzer::numContinuousVars.
    Referenced by Analyzer::read_variables_responses(), and Analyzer::variables_array_to_samples().
    The documentation for this class was generated from the following files:
    • DakotaAnalyzer.hpp
    • DakotaAnalyzer.cpp

14.6 ApplicationInterface Class Reference

Derived class within the interface class hierarchy for supporting interfaces to simulation codes.
    Inheritance diagram for ApplicationInterface:

![Inheritance Diagram]

Public Member Functions

• ApplicationInterface (const ProblemDescDB &problem_db)
CHAPTER 14. CLASS DOCUMENTATION

constructor
• ~ApplicationInterface ()

destructor

Protected Member Functions

• void init_communicators (const IntArray &message_lengths, int max_eval_concurrency)
  allocate communicator partitions for concurrent evaluations within an iterator and concurrent multiprocessor analyses within an evaluation.

• void set_communicators (const IntArray &message_lengths, int max_eval_concurrency)
  set the local parallel partition data for an interface (the partitions are already allocated in ParallelLibrary).

• void init_serial ()
• int async_local_evaluation_concurrency () const
  return asyncLocalEvalConcurrency

•short interface_synchronization () const
  return interfaceSynchronization

• bool evaluation_cache () const
  return evalCacheFlag

• bool restart_file () const
  return evalCacheFlag

• String final_eval_id_tag (int fn_eval_id)
  form and return the final evaluation ID tag, appending iface ID if needed

• void map (const Variables &vars, const ActiveSet &set, Response &response, bool asynch_flag=false)
  Provides a "mapping" of variables to responses using a simulation. Protected due to Interface letter-envelope idiom.

• void manage_failure (const Variables &vars, const ActiveSet &set, Response &response, int failed_eval_id)
  manages a simulation failure using abort/retry/recover/continuation

• const IntResponseMap & synchronize ()
  executes a blocking schedule for asynchronous evaluations in the beforeSynchCorePRPQueue and returns all jobs

• const IntResponseMap & synchronize_nowait ()
  executes a nonblocking schedule for asynchronous evaluations in the beforeSynchCorePRPQueue and returns a partial set of completed jobs

• void serve_evaluations ()
  run on evaluation servers to serve the iterator master

• void stop_evaluation_servers ()
  used by the iterator master to terminate evaluation servers

• bool check_multiprocessor_analysis (bool warn)
  checks on multiprocessor analysis configuration

• bool check_asynchronous (bool warn, int max_eval_concurrency)
  checks on asynchronous configuration (for direct interfaces)

• bool check_multiprocessor_asynchronous (bool warn, int max_eval_concurrency)
  checks on asynchronous settings for multiprocessor partitions

• String final_batch_id_tag ()
  form and return the final batch ID tag
14.6. APPLICATIONINTERFACE CLASS REFERENCE

- virtual void derived_map (const Variables &vars, const ActiveSet &set, Response &response, int fn_eval_id)
  Called by map() and other functions to execute the simulation in synchronous mode. The portion of performing an evaluation that is specific to a derived class.

- virtual void derived_map_asynch (const ParamResponsePair &pair)
  Called by map() and other functions to execute the simulation in asynchronous mode. The portion of performing an asynchronous evaluation that is specific to a derived class.

- virtual void wait_local_evaluations (PRPQueue &prp_queue)
  For asynchronous function evaluations, this method is used to detect completion of jobs and process their results. It provides the processing code that is specific to derived classes. This version waits for at least one completion.

- virtual void test_local_evaluations (PRPQueue &prp_queue)
  For asynchronous function evaluations, this method is used to detect completion of jobs and process their results. It provides the processing code that is specific to derived classes. This version is nonblocking and will return without any completions if none are immediately available.

- virtual void init_communicators_checks (int max_eval_concurrency)
  perform construct-time error checks on the parallel configuration

- virtual void set_communicators_checks (int max_eval_concurrency)
  perform run-time error checks on the parallel configuration

- void master_dynamic_schedule_analyses ()
  blocking dynamic schedule of all analyses within a function evaluation using message passing

- void serve_analyses_synch ()
  serve the master analysis scheduler and manage one synchronous analysis job at a time

- virtual int synchronous_local_analysis (int analysis_id)
  Execute a particular analysis (identified by analysis_id) synchronously on the local processor. Used for the derived class specifics within ApplicationInterface::serve_analyses_synch().

Protected Attributes

- ParallelLibrary & parallelLib
  reference to the ParallelLibrary object used to manage MPI partitions for the concurrent evaluations and concurrent analyses parallelism levels

- bool batchEval
  flag indicating usage of batch evaluation facilities, where a set of jobs is launched and scheduled as a unit rather than individually

- bool asynchFlag
  flag indicating usage of asynchronous evaluation

- int batchIdCntr
  maintain a count of the batches

- bool suppressOutput
  flag for suppressing output on slave processors

- int evalCommSize
  size of evalComm

- int evalCommRank
  processor rank within evalComm

- int evalServerId
  evaluation server identifier
CHAPTER 14. CLASS DOCUMENTATION

- **bool eaDedMasterFlag**
  - flag for dedicated master partitioning at ea level

- **int analysisCommSize**
  - size of analysisComm

- **int analysisCommRank**
  - processor rank within analysisComm

- **int analysisServerId**
  - analysis server identifier

- **int numAnalysisServers**
  - current number of analysis servers

- **bool multiProcAnalysisFlag**
  - flag for multiprocessor analysis partitions

- **bool asynchLocalAnalysisFlag**
  - flag for asynchronous local parallelism of analyses

- **int asynchLocalAnalysisConcurrency**
  - limits the number of concurrent analyses in asynchronous local scheduling and specifies hybrid concurrency when message passing

- **int asynchLocalEvalConcSpec**
  - user specification for asynchronous local evaluation concurrency

- **int asynchLocalAnalysisConcSpec**
  - user specification for asynchronous local analysis concurrency

- **int numAnalysisDrivers**
  - the number of analysis drivers used for each function evaluation (from the analysis_drivers interface specification)

- **IntSet completionSet**
  - the set of completed fn_eval_id’s populated by wait_local_evaluations() and test_local_evaluations()

- **String failureMessage**
  - base message for managing failed evals; will be followed with more details in screen output

**Private Member Functions**

- **bool duplication_detect (const Variables &vars, Response &response, bool asynch_flag)**
  - checks data_pairs and beforeSynchCorePRPQueue to see if the current evaluation request has already been performed or queued

- **void init_default_asv (size_t num_fns)**
  - initialize default ASV if needed; this is done at run time due to post-construct time Response size changes.

- **void master_dynamic_schedule_evaluations ()**
  - blocking dynamic schedule of all evaluations in beforeSynchCorePRPQueue using message passing on a dedicated master partition; executes on iteratorComm master

- **void peer_static_schedule_evaluations ()**
  - blocking static schedule of all evaluations in beforeSynchCorePRPQueue using message passing on a peer partition; executes on iteratorComm master

- **void peer_dynamic_schedule_evaluations ()**
  - blocking dynamic schedule of all evaluations in beforeSynchCorePRPQueue using message passing on a peer partition; executes on iteratorComm master
14.6. APPLICATION INTERFACE CLASS REFERENCE

- `void asynchronous_local_evaluations (PRPQueue &prp_queue)`
  *perform all jobs in prp_queue using asynchronous approaches on the local processor*

- `void synchronous_local_evaluations (PRPQueue &prp_queue)`
  *perform all jobs in prp_queue using synchronous approaches on the local processor*

- `void master_dynamic_schedule_evaluations_nowait ()`
  *execute a nonblocking dynamic schedule in a master-slave partition*

- `void peer_static_schedule_evaluations_nowait ()`
  *execute a nonblocking static schedule in a peer partition*

- `void peer_dynamic_schedule_evaluations_nowait ()`
  *execute a nonblocking dynamic schedule in a peer partition*

- `void asynchronous_local_evaluations_nowait (PRPQueue &prp_queue)`
  *launch new jobs in prp_queue asynchronously (if capacity is available), perform nonblocking query of all running jobs, and process any completed jobs (handles both local master- and local peer-scheduling cases)*

- `void broadcast_evaluation (const ParamResponsePair &pair)`
  *convenience function for broadcasting an evaluation over an evalComm*

- `void broadcast_evaluation (int fn_eval_id, const Variables &vars, const ActiveSet &set)`
  *convenience function for broadcasting an evaluation over an evalComm*

- `void send_evaluation (PRPQueueIter &prp_it, size_t buff_index, int server_id, bool peer_flag)`
  *helper function for sending sendBuffers[buff_index] to server*

- `void receive_evaluation (PRPQueueIter &prp_it, size_t buff_index, int server_id, bool peer_flag)`
  *helper function for processing recvBuffers[buff_index] within scheduler*

- `void launch_asynch_local (PRPQueueIter &prp_it)`
  *launch an asynchronous local evaluation from a queue iterator*

- `void launch_asynch_local (MPIUnpackBuffer &recv_buffer, int fn_eval_id)`
  *launch an asynchronous local evaluation from a receive buffer*

- `void process_asynch_local (int fn_eval_id)`
  *process a completed asynchronous local evaluation*

- `void process_synch_local (PRPQueueIter &prp_it)`
  *process a completed synchronous local evaluation*

- `void assign_asynch_local_queue (PRPQueue &local_prp_queue, PRPQueueIter &local_prp_iter)`
  *helper function for creating an initial active local queue by launching asynch local jobs from local_prp_queue, as limited by server capacity*

- `void assign_asynch_local_queue_nowait (PRPQueue &local_prp_queue, PRPQueueIter &local_prp_iter)`
  *helper function for updating an active local queue by backfilling asynch local jobs from local_prp_queue, as limited by server capacity*

- `size_t test_local_backfill (PRPQueue &assign_queue, PRPQueueIter &assign_iter)`
  *helper function for testing active asynch local jobs and then backfilling*

- `size_t test_receives_backfill (PRPQueueIter &assign_iter, bool peer_flag)`
  *helper function for testing receive requests and then backfilling jobs*

- `void serve_evaluations_synch ()`
  *serve the evaluation message passing schedulers and perform one synchronous evaluation at a time*

- `void serve_evaluations_synch_peer ()`
  *serve the evaluation message passing schedulers and perform one synchronous evaluation at a time as part of the 1st peer*
void serve_evaluations_asynch ()
serve the evaluation message passing schedulers and manage multiple asynchronous evaluations

void serve_evaluations_asynch_peer ()
serve the evaluation message passing schedulers and perform multiple asynchronous evaluations as part of the 1st peer

void set_evaluation_communicators (const IntArray &message_lengths)
convenience function for updating the local evaluation partition data following ParallelLibrary::init_evaluation_communicators().

void set_analysis_communicators ()
convenience function for updating the local analysis partition data following ParallelLibrary::init_analysis_communicators().

void init_serial_evaluations ()
set concurrent evaluation configuration for serial operations

void init_serial_analyses ()
set concurrent analysis configuration for serial operations (e.g., for local executions on a dedicated master)

const ParamResponsePair & get_source_pair (const Variables &target_vars)
convenience function for the continuation approach in manage_failure() for finding the nearest successful "source" evaluation to the failed "target"

void continuation (const Variables &target_vars, const ActiveSet &set, Response &response, const ParamResponsePair &source_pair, int failed_eval_id)
performs a 0th order continuation method to step from a successful "source" evaluation to the failed "target". Invoked by manage_failure() for failAction == "continuation".

void common_input_filtering (const Variables &vars)
common input filtering operations, e.g. mesh movement

void common_output_filtering (Response &response)
common output filtering operations, e.g. data filtering

Private Attributes

- int worldSize
  size of MPI_COMM_WORLD
- int worldRank
  processor rank within MPI_COMM_WORLD
- int iteratorCommSize
  size of iteratorComm
- int iteratorCommRank
  processor rank within iteratorComm
- bool ieMessagePass
  flag for message passing at ie scheduling level
- int numEvalServers
  current number of evaluation servers
- int numEvalServersSpec
  user specification for number of evaluation servers
- int procsPerEvalSpec
  user specification for processors per analysis servers
14.6. APPLICATIONINTERFACE CLASS REFERENCE

- **bool eaMessagePass**
  flag for message passing at ea scheduling level
- **int numAnalysisServersSpec**
  user spec for number of analysis servers
- **int procsPerAnalysisSpec**
  user specification for processors per analysis servers
- **int lenVarsMessage**
  length of a MPIPackBuffer containing a Variables object; computed in Model::init_communicators()
- **int lenVarsActSetMessage**
  length of a MPIPackBuffer containing a Variables object and an ActiveSet object; computed in Model::init_communicators()
- **int lenResponseMessage**
  length of a MPIPackBuffer containing a Response object; computed in Model::init_communicators()
- **int lenPRPairMessage**
  length of a MPIPackBuffer containing a ParamResponsePair object; computed in Model::init_communicators()
- **short evalScheduling**
  user specification of evaluation scheduling algorithm: \{DEFAULT,MASTER,PEER\_DYNAMIC,PEER\_STATIC\}.
- **short analysisScheduling**
  user specification of analysis scheduling algorithm: \{DEFAULT,MASTER,PEER\}.
- **int asynchLocalEvalConcurrency**
  limits the number of concurrent evaluations in asynchronous local scheduling and specifies hybrid concurrency when message passing
- **bool asynchLocalEvalStatic**
  whether the asynchronous local evaluations are to be performed with a static schedule (default false)
- **BitArray localServerAssigned**
  array with one bit per logical "server" indicating whether a job is currently running on the server (used for async local static schedules)
- **short interfaceSynchronization**
  interface synchronization specification: synchronous (default) or asynchronous
- **bool headerFlag**
  used by synchronize\_nowait to manage header output frequency (since this function may be called many times prior to any completions)
- **bool asvControlFlag**
  used to manage a user request to deactivate the active set vector control. true = modify the ASV each evaluation as appropriate (default); false = ASV values are static so that the user need not check them on each evaluation.
- **bool evalCacheFlag**
  used to manage a user request to deactivate the function evaluation cache (i.e., queries and insertions using the data\_pairs cache).
- **bool nearbyDuplicateDetect**
  flag indicating optional usage of tolerance-based duplication detection (less efficient, but helpful when experiencing restart cache misses)
- **Real nearbyTolerance**
  tolerance value for tolerance-based duplication detection
- **bool restartFileFlag**
  used to manage a user request to deactivate the restart file (i.e., insertions into write_restart).

- **SharedResponseData sharedRespData**
  *SharedResponseData of associated Response.*

- **String gradientType**
  *type of gradients present in associated Response*

- **String hessianType**
  *type of Hessians present in associated Response*

- **IntSet gradMixedAnalyticIds**
  *IDs of analytic gradients when mixed gradients present.*

- **IntSet hessMixedAnalyticIds**
  *IDs of analytic gradients when mixed gradients present.*

- **ShortArray defaultASV**
  *the static ASV values used when the user has selected asvControl = off*

- **String failAction**
  *mitigation action for captured simulation failures: abort, retry, recover, or continuation*

- **int failRetryLimit**
  *limit on the number of retries for the retry failAction*

- **RealVector failRecoveryFnVals**
  *the dummy function values used for the recover failAction*

- **IntResponseMap historyDuplicateMap**
  *used to bookkeep asynchronous evaluations which duplicate data_pairs evaluations. Map key is evalIdCntr, map value is corresponding response.*

- **std::map<int, std::pair<PRPQueueHitIter, Response>> beforeSynchDuplicateMap**
  *used to bookkeep evalIdCntr, beforeSynchCorePRPQueue iterator, and response of asynchronous evaluations which duplicate queued beforeSynchCorePRPQueue evaluations*

- **PRPQueue beforeSynchCorePRPQueue**
  *used to bookkeep vars/set/response of nonduplicate asynchronous core evaluations. This is the queue of jobs populated by asynchronous map() that is later scheduled in synchronize() or synchronize_nowait().*

- **PRPQueue beforeSynchAlgPRPQueue**
  *used to bookkeep vars/set/response of asynchronous algebraic evaluations. This is the queue of algebraic jobs populated by asynchronous map() that is later evaluated in synchronize() or synchronize_nowait().*

- **PRPQueue asyncLocalActivePRPQueue**
  *used by nonblocking asynchronous local schedulers to bookkeep active local jobs*

- **std::map<int, IntSizetPair> msgPassRunningMap**
  *used by nonblocking message passing schedulers to bookkeep which jobs are running remotely*

- **int nowaitEvalIdRef**
  *fnEvalId reference point for preserving modulo arithmetic-based job assignment in case of peer static nonblocking schedulers*

- **MPIPackBuffer *sendBuffers**
  *array of pack buffers for evaluation jobs queued to a server*

- **MPIUnpackBuffer *recvBuffers**
  *array of unpack buffers for evaluation jobs returned by a server*

- **MPI_Request *recvRequests**
  *array of requests for nonblocking evaluation receives*
14.6. APPLICATIONINTERFACE CLASS REFERENCE

14.6.1 Detailed Description

Derived class within the interface class hierarchy for supporting interfaces to simulation codes.

ApplicationInterface provides an interface class for performing parameter to response mappings using simulation code(s). It provides common functionality for a number of derived classes and contains the majority of all of the scheduling algorithms in DAKOTA. The derived classes provide the specifics for managing code invocations using system calls, forks, direct procedure calls, or distributed resource facilities.

14.6.2 Member Function Documentation

void init_serial ( ) [inline], [protected], [virtual]

DataInterface.cpp defaults of 0 servers are needed to distinguish an explicit user request for 1 server (serialization of a parallelism level) from no user request (use parallel auto-config). This default causes problems when init_communicators() is not called for an interface object (e.g., static scheduling fails in DirectApplicInterface::derived_map() for NestedModel::optionallInterface). This is the reason for this function: to reset certain defaults for interface objects that are used serially.

Reimplemented from Interface.

References ApplicationInterface::init_serial_analyses(), and ApplicationInterface::init_serial_evaluations().

void map ( const Variables & vars, const ActiveSet & set, Response & response, bool asynch_flag = false ) [protected], [virtual]

Provides a "mapping" of variables to responses using a simulation. Protected due to Interface letter-envelope idiom.

The function evaluator for application interfaces. Called from derived_evaluate() and derived_evaluate_nowait() in derived Model classes. If asynch_flag is not set, perform a blocking evaluation (using derived_map()). If asynch_flag is set, add the job to the beforeSynchCorePRPQueue queue for execution by one of the scheduler routines in synchronize() or synchronize_nowait(). Duplicate function evaluations are detected with duplication_detect().

Reimplemented from Interface.

References Response::active_set(), Interface::algebraic_mappings(), Interface::algebraicMappings, Interface::asv_mapping(), ApplicationInterface::asvControlFlag, ApplicationInterface::batchEval, ApplicationInterface::beforeSynchAlgPRPQueue, ApplicationInterface::beforeSynchCorePRPQueue, ApplicationInterface::broadcast_evaluation(), Response::copy(), Interface::coreMappings, Interface::currEvalId, Dakota::data_pairs, ApplicationInterface::defaultASV, ApplicationInterface::derived_map(), ApplicationInterface::duplication_detect(), ApplicationInterface::evalCacheFlag, Interface::evalIdCntr, Interface::fineGrainEvalCounters, Interface::fnGradCounter, Interface::fnHessCounter, Interface::fnLabels, Interface::fnValCounter, Response::function_labels(), Interface::init_algebraic_mappings(), ApplicationInterface::init_default_asv(), Interface::init_evaluation_counters(), Interface::interfaceId, ApplicationInterface::manage_failure(), Interface::multiProcEvalFlag, Interface::newEvalIdCntr, Interface::newFnGradCounter, Interface::newFnHessCounter, Interface::newFnValCounter, Interface::outputLevel, ApplicationInterface::parallelLib, ActiveSet::request_vector(), Interface::response_mapping(), ApplicationInterface::restartFileFlag, ApplicationInterface::sharedRespData, and ParallelLibrary::write_restart().

const IntResponseMap & synchronize ( ) [protected], [virtual]

executes a blocking schedule for asynchronous evaluations in the beforeSynchCorePRPQueue and returns all jobs

This function provides blocking synchronization for all cases of asynchronous evaluations, including the local asynchronous case (background system call, nonblocking fork, & multithreads), the message passing case, and the hybrid case. Called from derived_synchronize() in derived Model classes.

Reimplemented from Interface.
References Interface::algebraic_mappings(), Interface::algebraicMappings, Interface::asv_mapping(), ApplicationInterface::asynchLocalEvalStatic, ApplicationInterface::asynchronous_local_evaluations(), ApplicationInterface::beforeSynchAlgPRPQueue, ApplicationInterface::beforeSynchCorePRPQueue, ApplicationInterface::beforeSynchDuplicateMap, Interface::cachedResponseMap, Interface::coreMappings, ApplicationInterface::evalScheduling, ApplicationInterface::historyDuplicateMap, Interface::ieDedMasterFlag, ApplicationInterface::ieMessagePass, Interface::interfaceId, Interface::interfaceType, ApplicationInterface::master_dynamic_schedule_evaluations(), Interface::multiProcEvalFlag, Interface::outputLevel, ApplicationInterface::peer_dynamic_schedule_evaluations(), ApplicationInterface::peer_static_schedule_evaluations(), Interface::rawResponseMap, Interface::response_mapping(), and ApplicationInterface::sharedRespData.

const IntResponseMap & synchronize_nowait( ) [protected], [virtual]
executes a nonblocking schedule for asynchronous evaluations in the beforeSynchCorePRPQueue and returns a partial set of completed jobs
This function provides nonblocking synchronization for the local asynchronous case and selected nonblocking message passing schedulers. Called from derived synchronize_nowait() in derived Model classes.
Reimplemented from Interface.
References Interface::algebraic_mappings(), Interface::algebraicMappings, Interface::asv_mapping(), ApplicationInterface::asynchLocalEvalStatic, ApplicationInterface::asynchronous_local_evaluations_nowait(), ApplicationInterface::beforeSynchAlgPRPQueue, ApplicationInterface::beforeSynchCorePRPQueue, ApplicationInterface::beforeSynchDuplicateMap, Interface::cachedResponseMap, Interface::coreMappings, ParamResponsePair::evalid(), ApplicationInterface::evalScheduling, ApplicationInterface::headerFlag, ApplicationInterface::historyDuplicateMap, Interface::ieDedMasterFlag, ApplicationInterface::ieMessagePass, Interface::interfaceId, Interface::interfaceType, Dakota::lookup_by_evalid(), ApplicationInterface::master_dynamic_schedule_evaluations_nowait(), Interface::multiProcEvalFlag, Interface::outputLevel, ApplicationInterface::peer_dynamic_schedule_evaluations_nowait(), ApplicationInterface::peer_static_schedule_evaluations_nowait(), Interface::rawResponseMap, ParamResponsePair::response(), Interface::response_mapping(), ApplicationInterface::sharedRespData, and Response::update().

void serve_evaluations( ) [protected], [virtual]
run on evaluation servers to serve the iterator master
Invoked by the serve() function in derived Model classes. Passes control to serve_evaluations_synch(), serve_evaluations_asynch(), serve_evaluations_synch_peer(), or serve_evaluations_asynch_peer() according to specified concurrency, partition, and scheduler configuration.
Reimplemented from Interface.
References ApplicationInterface::asynchLocalEvalConcurrency, ApplicationInterface::evalServerId, Interface::ieDedMasterFlag, ApplicationInterface::serve_evaluations_asynch(), ApplicationInterface::serve_evaluations_asynch_peer(), ApplicationInterface::serve_evaluations_synch(), and ApplicationInterface::serve_evaluations_synch_peer().

void stop_evaluation_servers( ) [protected], [virtual]
used by the iterator master to terminate evaluation servers
This code is executed on the iteratorComm rank 0 processor when iteration on a particular model is complete. It sends a termination signal (tag = 0 instead of a valid fn_eval_id) to each of the slave analysis servers. NOTE: This function is called from the Strategy layer even when in serial mode. Therefore, use iteratorCommSize to provide appropriate fall through behavior.
Reimplemented from Interface.
References ParallelLibrary::bcast_e(), ParallelLibrary::free(), ParallelConfiguration::ie_parallel_level(), Interface::ieDedMasterFlag, ParallelLibrary::isend_ie(), ApplicationInterface::iteratorCommSize, Interface::multiProcEval-
14.6. APPLICATIONINTERFACE CLASS REFERENCE

Flag, ApplicationInterface::numEvalServers, Interface::outputLevel, ParallelLibrary::parallel_configuration(), and ApplicationInterface::parallelLib.

`void init_communicators() [protected], [virtual]`
perform construct-time error checks on the parallel configuration

Override DirectApplicInterface definition if plug-in to allow batch processing in Plugin{Serial,Parallel} DirectApplicInterface.cpp

Reimplemented in DirectApplicInterface, ProcessHandleApplicInterface, and SysCallApplicInterface.

Referenced by ApplicationInterface::init_communicators().

`void set_communicators() [protected], [virtual]`
perform run-time error checks on the parallel configuration

Override DirectApplicInterface definition if plug-in to allow batch processing in Plugin{Serial,Parallel} DirectApplicInterface.cpp

Reimplemented in DirectApplicInterface, SerialDirectApplicInterface, SoleilDirectApplicInterface, ParallelDirectApplicInterface, ProcessHandleApplicInterface, and SysCallApplicInterface.

Referenced by ApplicationInterface::set_communicators().

`void master_dynamic_schedule_analyses() [protected]`
blocking dynamic schedule of all analyses within a function evaluation using message passing

This code is called from derived classes to provide the master portion of a master-slave algorithm for the dynamic scheduling of analyses among slave servers. It is patterned after master_dynamic_schedule_evaluations(). It performs no analyses locally and matches either serve_analyses_synch() or serve_analyses_asynch() on the slave servers, depending on the value of asynchLocalAnalysisConcurrency. Dynamic scheduling assigns jobs in 2 passes. The 1st pass gives each server the same number of jobs (equal to asynchLocalAnalysisConcurrency). The 2nd pass assigns the remaining jobs to slave servers as previous jobs are completed. Single- and multilevel parallel use intra- and inter-communicators, respectively, for send/receive. Specific syntax is encapsulated within ParallelLibrary.

References ApplicationInterface::asynchLocalAnalysisConcurrency, ParallelLibrary::free(), ParallelLibrary::irecv_ea(), ParallelLibrary::isend_ea(), ApplicationInterface::numAnalysisDrivers, ApplicationInterface::numAnalysisServers, ApplicationInterface::parallelLib, ParallelLibrary::waitall(), and ParallelLibrary::waitsome().

Referenced by ProcessHandleApplicInterface::create_evaluation_process(), SysCallApplicInterface::create_evaluation_process(), and DirectApplicInterface::derived_map().

`void serve_analyses_synch() [protected]`
serve the master analysis scheduler and manage one synchronous analysis job at a time

This code is called from derived classes to run synchronous analyses on slave processors. The slaves receive requests (blocking receive), do local derived_map_ac’s, and return codes. This is done continuously until a termination signal is received from the master. It is patterned after serve_evaluations_synch().

References ApplicationInterface::analysisCommRank, ParallelLibrary::bcast_ea(), ParallelLibrary::isend_ea(), ApplicationInterface::multiProcAnalysisFlag, ApplicationInterface::parallelLib, ParallelLibrary::recv_ea(), ApplicationInterface::synchronous_local_analysis(), and ParallelLibrary::wait().

Referenced by ProcessHandleApplicInterface::create_evaluation_process(), SysCallApplicInterface::create_evaluation_process(), and DirectApplicInterface::derived_map().
bool duplication_detect ( const Variables & vars, Response & response, bool asynch_flag ) [private]

checks data_pairs and beforeSynchCorePRPQueue to see if the current evaluation request has already been performed or queued.

Called from map() to check incoming evaluation request for duplication with content of data_pairs and beforeSynchCorePRPQueue. If duplication is detected, return true, else return false. Manage bookkeeping with history-DuplicateMap and beforeSynchDuplicateMap. Note that the list searches can get very expensive if a long list is searched on every new function evaluation (either from a large number of previous jobs, a large number of pending jobs, or both). For this reason, a user request for deactivation of the evaluation cache results in a complete bypass of duplication_detect(), even though a beforeSynchCorePRPQueue search would still be meaningful. Since the intent of this request is to streamline operations, both list searches are bypassed.

References Response::active_set(), ApplicationInterface::beforeSynchCorePRPQueue, ApplicationInterface::beforeSynchDuplicateMap, Response::copy(), Dakota::data_pairs, ParamResponsePair::eval_id(), Interface::evalIdCntr, ApplicationInterface::historyDuplicateMap, Interface::interfaceId, Dakota::lookup_by_val(), ApplicationInterface::nearbyDuplicateDetect, ApplicationInterface::nearbyTolerance, and Response::update().

Referenced by ApplicationInterface::map().

void init_default_asv ( size_t num_fns ) [private]

initialize default ASV if needed; this is done at run time due to post-construct time Response size changes.

If the user has specified active_set_vector as off, then map() uses a default ASV which is constant for all function evaluations (so that the user need not check the content of the ASV on each evaluation). Only initialized if needed and not already sized.

References ApplicationInterface::asvControlFlag, ApplicationInterface::defaultASV, ApplicationInterface::gradientType, ApplicationInterface::gradMixedAnalyticIds, ApplicationInterface::hessianType, and ApplicationInterface::hessMixedAnalyticIds.

Referenced by ApplicationInterface::map().

void master_dynamic_schedule_evaluations ( ) [private]

blocking dynamic schedule of all evaluations in beforeSynchCorePRPQueue using message passing on a dedicated master partition; executes on iteratorComm master.

This code is called from synchronize() to provide the master portion of a master-slave algorithm for the dynamic scheduling of evaluations among slave servers. It performs no evaluations locally and matches either serve_evaluations_synch() or serve_evaluations_asynch() on the slave servers, depending on the value of asynchLocalEvalConcurrency. Dynamic scheduling assigns jobs in 2 passes. The 1st pass gives each server the same number of jobs (equal to asynchLocalEvalConcurrency). The 2nd pass assigns the remaining jobs to slave servers as previous jobs are completed and returned. Single- and multilevel parallel use intra- and inter-communicators, respectively, for send/receive. Specific syntax is encapsulated within ParallelLibrary::peer.

References ApplicationInterface::asynchLocalEvalConcurrency, ApplicationInterface::beforeSynchCorePRPQueue, Dakota::lookup_by_eval_id(), ApplicationInterface::numEvalServers, Interface::outputLevel, ApplicationInterface::parallelLib, ApplicationInterface::receive_evaluation(), ApplicationInterface::recvBuffers, ApplicationInterface::recvRequests, ApplicationInterface::send_evaluation(), ApplicationInterface::sendBuffers, ParallelLibrary::waitall(), and ParallelLibrary::waitsome().

Referenced by ApplicationInterface::synchronize().

void peer_static_schedule_evaluations ( ) [private]

blocking static schedule of all evaluations in beforeSynchCorePRPQueue using message passing on a peer partition; executes on iteratorComm master.
This code runs on the iteratorCommRank 0 processor (the iterator) and is called from `synchronize()` in order to manage a static schedule for cases where peer 1 must block when evaluating its local job allocation (e.g., single or multiprocessor direct interface evaluations). It matches `serve_evaluations.peer()` for any other processors within the first evaluation partition and `serve_evaluations.{synch,async}()` for all other evaluation partitions (depending on `asyncLocalEvalConcurrency`). It performs function evaluations locally for its portion of the job allocation using either `asynchronous_local_evaluations()` or `synchronous_local_evaluations()`. Single-level and multilevel parallel use intra- and inter-communicators, respectively, for send/receive. Specific syntax is encapsulated within `ParallelLibrary`. The iteratorCommRank 0 processor assigns the static schedule since it is the only processor with access to `beforeSynchCorePRPQueue` (it runs the iterator and calls synchronize). The alternate design of each peer selecting its own jobs using the modulus operator would be applicable if execution of this function (and therefore the job list) were distributed.

References: ApplicationInterface::asynchLocalEvalConcurrency, ApplicationInterface::asynchronous_local_evaluations(), ApplicationInterface::beforeSynchCorePRPQueue, ApplicationInterface::numEvalServers, Interface::outputLevel, ApplicationInterface::parallelLib, ApplicationInterface::receive_evaluation(), ApplicationInterface::recvBuffers, ApplicationInterface::recvRequests, ApplicationInterface::send_evaluation(), ApplicationInterface::sendBuffers, ApplicationInterface::synchronous_local_evaluations(), and ParallelLibrary::waitall().

Referenced by ApplicationInterface::synchronize().

```c++
void peer_dynamic_schedule_evaluations() [private]
blocking dynamic schedule of all evaluations in beforeSynchCorePRPQueue using message passing on a peer partition; executes on iteratorComm master
```

This code runs on the iteratorCommRank 0 processor (the iterator) and is called from `synchronize()` in order to manage a dynamic schedule, as enabled by nonblocking management of local asynchronous jobs. It matches `serve_evaluations.{synch,async}()` for other evaluation partitions, depending on `asyncLocalEvalConcurrency`; it does not match `serve_evaluations.peer()` since, for local asynchronous jobs, the first evaluation partition cannot be multiprocessor. It performs function evaluations locally for its portion of the job allocation using `asynchronous_local_evaluations_nowait()`. Single-level and multilevel parallel use intra- and inter-communicators, respectively, for send/receive. Specific syntax is encapsulated within `ParallelLibrary`.

References: ApplicationInterface::assign_asynch_local_queue(), ApplicationInterface::asynchLocalEvalConcurrency, ApplicationInterface::beforeSynchCorePRPQueue, ApplicationInterface::msgPassRunningMap, ApplicationInterface::numEvalServers, Interface::outputLevel, ApplicationInterface::recvBuffers, ApplicationInterface::recvRequests, ApplicationInterface::send_evaluation(), ApplicationInterface::sendBuffers, ApplicationInterface::synchronous_local_evaluations(), and ParallelLibrary::waitall().

Referenced by ApplicationInterface::synchronize().

```c++
void asynchronous_local_evaluations(PRPQueue & local_prp_queue) [private]
perform all jobs in prp_queue using asynchronous approaches on the local processor
```

This function provides blocking synchronization for the local async case (background system call, nonblocking fork, or threads). It can be called from `synchronize()` for a complete local scheduling of all asynchronous jobs or from `peer_{static, dynamic}_schedule_evaluations()` to perform a local portion of the total job set. It uses `derived_map_asynch()` to initiate asynchronous evaluations and `wait_local_evaluations()` to capture completed jobs, and mirrors the `master_dynamic_schedule_evaluations()` message passing scheduler as much as possible (`wait_local_evaluations()` is modeled after MPI_Waitsome()).

References: ApplicationInterface::assign_asynch_local_queue(), ApplicationInterface::asynchLocalActivePRPQueue, ApplicationInterface::asynchLocalEvalConcurrency, ApplicationInterface::asynchLocal EvalStatic, ApplicationInterface::batchEval, ApplicationInterface::completionSet, ApplicationInterface::launch_asynch_local(), ApplicationInterface::localServerAssigned, Dakota::lookup_by_eval_id(), ApplicationInterface::numEvalServers, Interface-
CHAPTER 14. CLASS DOCUMENTATION

::outputLevel, ApplicationInterface::process_asynch_local(), Interface::rawResponseMap, and ApplicationInterface::wait_local_evaluations().

Referenced by ApplicationInterface::peer_static_schedule_evaluations(), and ApplicationInterface::synchronize().

void synchronous_local_evaluations ( PRPQueue & local_prp_queue ) [private]

perform all jobs in prp_queue using synchronous approaches on the local processor.

This function provides blocking synchronization for the local synchronous case (foreground system call, blocking fork, or procedure call from derived_map()). It is called from peer_static_schedule_evaluations() to perform a local portion of the total job set.

References ApplicationInterface::broadcast_evaluation(), Interface::currEvalId, ApplicationInterface::derived_map(), ApplicationInterface::manage_failure(), Interface::multiProcEvalFlag, and ApplicationInterface::process_synch_local().

Referenced by ApplicationInterface::peer_static_schedule_evaluations(), and ApplicationInterface::peer_static_schedule_evaluations_nowait().

void master_dynamic_schedule_evaluations_nowait ( ) [private]

execute a nonblocking dynamic schedule in a master-slave partition.

This code is called from synchronize_nowait() to provide the master portion of a nonblocking master-slave algorithm for the dynamic scheduling of evaluations among slave servers. It performs no evaluations locally and matches either serve_evaluations_synch() or serve_evaluations_asynch() on the slave servers, depending on the value of asynchLocalEvalConcurrency. Dynamic scheduling assigns jobs in 2 passes. The 1st pass gives each server the same number of jobs (equal to asynchLocalEvalConcurrency). The 2nd pass assigns the remaining jobs to slave servers as previous jobs are completed. Single- and multilevel parallel use intra- and inter-communicators, respectively, for send/receive. Specific syntax is encapsulated within ParallelLibrary.

References Dakota::abort_handler(), ApplicationInterface::asynchLocalEvalConcurrency, ApplicationInterface::beforeSynchCorePRPQueue, ApplicationInterface::headerFlag, ApplicationInterface::msgPassRunningMap, ApplicationInterface::numEvalServers, ApplicationInterface::recvBuffers, ApplicationInterface::recvRequests, ApplicationInterface::send_evaluation(), ApplicationInterface::sendBuffers, and ApplicationInterface::test_receives_backfill().

Referenced by ApplicationInterface::synchronize_nowait().

void peer_static_schedule_evaluations_nowait ( ) [private]

execute a nonblocking static schedule in a peer partition.

This code runs on the iteratorCommRank 0 processor (the iterator) and is called from synchronize_nowait() in order to manage a nonblocking static schedule. It matches serve_evaluations_synch() for other evaluation partitions (asynchLocalEvalConcurrency == 1). It performs blocking local function evaluations, one at a time, for its portion of the static schedule and checks for remote completions in between each local completion. Therefore, unlike peer_dynamic_schedule_evaluations_nowait(), this scheduler will always return at least one job. Single-level and multilevel parallel use intra- and inter-communicators, respectively, for send/receive, with specific syntax as encapsulated within ParallelLibrary. The iteratorCommRank 0 processor assigns the static schedule since it is the only processor with access to beforeSynchCorePRPQueue (it runs the iterator and calls synchronize). The alternate design of each peer selecting its own jobs using the modulus operator would be applicable if execution of this function (and therefore the job list) were distributed.

References Dakota::abort_handler(), ApplicationInterface::assign_asynch_local_queue(), ApplicationInterface::assign_asynch_local_queue_nowait(), ApplicationInterface::asynchLocalActivePRPQueue, ApplicationInterface::asynchLocalEvalConcurrency, ApplicationInterface::beforeSynchCorePRPQueue, ApplicationInterface::headerFlag, Interface::interfaceType, Dakota::lookup_by_eval_id(), ApplicationInterface::msgPassRunningMap, Interface::multiProcEvalFlag, ApplicationInterface::nowaitEvalIdRef, ApplicationInterface::numEvalServers, ApplicationInterface::wait_local_evaluations().

Referenced by ApplicationInterface::peer_static_schedule_evaluations().
Interface::recvBuffers, ApplicationInterface::recvRequests, ApplicationInterface::send_evaluation(), Application-Interface::sendBuffers, ApplicationInterface::synchronous_local_evaluations(), ApplicationInterface::test_local_backfill(), and ApplicationInterface::test_receives_backfill.

Referenced by ApplicationInterface::synchronize_nowait().

void peer_dynamic_schedule_evaluations_nowait ( ) [private]
execute a nonblocking dynamic schedule in a peer partition

This code runs on the iteratorCommRank 0 processor (the iterator) and is called from synchronize_nowait() in order to manage a nonblocking static schedule. It matches serve_evaluations_{synch,asynch}() for other evaluation partitions (depending on asynchLocalEvalConcurrency). It performs nonblocking local function evaluations for its portion of the static schedule using asynchronous_local_evaluations(). Single-level and multilevel parallel use intra- and inter-communicators, respectively, for send/receive, with specific syntax as encapsulated within ParallelLibrary. The iteratorCommRank 0 processor assigns the dynamic schedule since it is the only processor with access to beforeSynchCorePRPQueue (it runs the iterator and calls synchronize). The alternate design of each peer selecting its own jobs using the modulus operator would be applicable if execution of this function (and therefore the job list) were distributed.

References Dakota::abort_handler(), ApplicationInterface::assign_asynch_local_queue(), ApplicationInterface-::assign_asynch_local_queue_nowait(), ApplicationInterface::asynchLocalActivePRPQueue, ApplicationInterface-::asynchLocalEvalConcurrency, ApplicationInterface::beforeSynchCorePRPQueue, ApplicationInterface::headerFlag, Dakota::lookup_by_eval_id(), ApplicationInterface::msgPassRunningMap, ApplicationInterface::numEval-Servers, ApplicationInterface::recvBuffers, ApplicationInterface::recvRequests, ApplicationInterface::send_evaluation(), ApplicationInterface::sendBuffers, ApplicationInterface::test_local_backfill(), and ApplicationInterface::test_receives_backfill().

Referenced by ApplicationInterface::synchronize_nowait().

void asynchronous_local_evaluations_nowait ( PRPQueue & local_prp_queue ) [private]
launch new jobs in prp_queue asynchronously (if capacity is available), perform nonblocking query of all running jobs, and process any completed jobs (handles both local master- and local peer-scheduling cases)

This function provides nonblocking synchronization for the local async case (background system call, non-blocking fork, or threads). It is called from synchronize_nowait() and passed the complete set of all asynchronous jobs (beforeSynchCorePRPQueue). It uses derived_map_asynch() to initiate asynchronous evaluations and test_local_evaluations() to capture completed jobs in nonblocking mode. It mirrors a nonblocking message passing scheduler as much as possible (test_local_evaluations() modeled after MPI_Testsome()). The result of this function is rawResponseMap, which uses eval_id as a key. It is assumed that the incoming local_prp_queue contains only active and new jobs - i.e., all completed jobs are cleared by synchronize_nowait().

Also supports asynchronous local evaluations with static scheduling. This scheduling policy specifically ensures that a completed asynchronous evaluation eval_id is replaced with an equivalent one, modulo asynchLocalEvalConcurrency. In the nowait case, this could render some servers idle if evaluations don’t come in eval_id order or some evaluations are cancelled by the caller in between calls. If this function is called with unlimited local eval concurrency, the static scheduling request is ignored.

References ApplicationInterface::assign_asynch_local_queue_nowait(), ApplicationInterface::asynchLocalActive-PRPQueue, ApplicationInterface::asynchLocalEvalConcurrency, ApplicationInterface::asynchLocalEvalStatic, Application-Interface::headerFlag, and ApplicationInterface::test_local_backfill().

Referenced by ApplicationInterface::synchronize_nowait().

void serve_evaluations_synch ( ) [private]
serve the evaluation message passing schedulers and perform one synchronous evaluation at a time
CHAPTER 14. CLASS DOCUMENTATION

This code is invoked by `serve_evaluations()` to perform one synchronous job at a time on each slave/peer server. The servers receive requests (blocking receive), do local synchronous maps, and return results. This is done continuously until a termination signal is received from the master (sent via `stop_evaluation_servers()`).

References `ParallelLibrary::bcast()`, `Interface::currEvalId`, `ApplicationInterface::derived_map()`, `ApplicationInterface::evalCommRank`, `ParallelLibrary::isend()`, `ApplicationInterface::lenResponseMessage`, `ApplicationInterface::lenVarsActSetMessage`, `ApplicationInterface::manage_failure()`, `Interface::multiProcEvalFlag`, `ApplicationInterface::parallelLib`, `ParallelLibrary::recv()`, `MPIPackBuffer::reset()`, `ApplicationInterface::sharedRespData`, and `ParallelLibrary::wait()`.

Referenced by `ApplicationInterface::serve_evaluations()`.

```c
void serve_evaluations_synch_peer() [private]
```

serve the evaluation message passing schedulers and perform one synchronous evaluation at a time as part of the 1st peer

This code is invoked by `serve_evaluations()` to perform a synchronous evaluation in coordination with the iteratorCommRank 0 processor (the iterator) for static schedules. The `bcast()` matches either the `bcast()` in `synchronous_local_evaluations()`, which is invoked by `peer_static_schedule_evaluations()`, or the `bcast()` in `map()`.

References `ParallelLibrary::bcast()`, `Interface::currEvalId`, `ApplicationInterface::derived_map()`, `ApplicationInterface::lenVarsActSetMessage`, `ApplicationInterface::manage_failure()`, `ApplicationInterface::parallelLib`, and `ApplicationInterface::sharedRespData`.

Referenced by `ApplicationInterface::serve_evaluations()`.

```c
void serve_evaluations_asynch() [private]
```

serve the evaluation message passing schedulers and manage multiple asynchronous evaluations

This code is invoked by `serve_evaluations()` to perform multiple asynchronous jobs on each slave/peer server. The servers test for any incoming jobs, launch any new jobs, process any completed jobs, and return any results. Each of these components is nonblocking, although the server loop continues until a termination signal is received from the master (sent via `stop_evaluation_servers()`). In the master-slave case, the master maintains the correct number of jobs on each slave. In the static scheduling case, each server is responsible for limiting concurrency (since the entire static schedule is sent to the peers at start up).

References `Dakota::abort_handler()`, `ApplicationInterface::asynchLocalActivePRPQueue`, `ApplicationInterface::asynchLocalEvalConcurrency`, `ParallelLibrary::bcast()`, `ApplicationInterface::completionSet`, `ApplicationInterface::evalCommRank`, `ParallelLibrary::irecv()`, `ApplicationInterface::launch_asynch_local()`, `ApplicationInterface::lenResponseMessage`, `ApplicationInterface::lenVarsActSetMessage`, `Dakota::lookup_by_eval_id()`, `Interface::multiProcEvalFlag`, `ApplicationInterface::parallelLib`, `ParallelLibrary::recv()`, `ParallelLibrary::send()`, `ParallelLibrary::test()`, and `ApplicationInterface::test_local_evaluations()`.

Referenced by `ApplicationInterface::serve_evaluations()`.

```c
void serve_evaluations_asynch_peer() [private]
```

serve the evaluation message passing schedulers and perform multiple asynchronous evaluations as part of the 1st peer

This code is invoked by `serve_evaluations()` to perform multiple asynchronous jobs on multiprocessor slave/peer servers. It matches the mulitProcEvalFlag bcasts in `ApplicationInterface::asynchronous_local_evaluations()`.

References `Dakota::abort_handler()`, `ApplicationInterface::asynchLocalActivePRPQueue`, `ApplicationInterface::asynchLocalEvalConcurrency`, `ParallelLibrary::bcast()`, `ApplicationInterface::completionSet`, `ApplicationInterface::launch_asynch_local()`, `ApplicationInterface::lenVarsActSetMessage`, `Dakota::lookup_by_eval_id()`, `ApplicationInterface::parallelLib`, and `ApplicationInterface::test_local_evaluations()`.

Referenced by `ApplicationInterface::serve_evaluations()`.
14.7 APPROXIMATION CLASS REFERENCE

The documentation for this class was generated from the following files:

- ApplicationInterface.hpp
- ApplicationInterface.cpp

14.7 Approximation Class Reference

Base class for the approximation class hierarchy.

Inheritance diagram for Approximation:

```
Approximation
  C3Approximation
  GaussProcApproximation
  PecosApproximation
  QMEApproximation
  SurfpackApproximation
  SurrogatesGPApprox
  SurrogatesPolyApprox
  TANA3Approximation
  TaylorApproximation
  VPSApproximation
```

Public Member Functions

- **Approximation ()**
  default constructor

- **Approximation (ProblemDescDB &problem_db, const SharedApproxData &shared_data, const String &approx_label)**
  standard constructor for envelope

- **Approximation (const SharedApproxData &shared_data)**
  alternate constructor

- **Approximation (const Approximation &approx)**

```
copy constructor
• virtual ~Approximation ()

destructor
• Approximation operator= (const Approximation &approx)

assignment operator
• virtual void active_model_key (const UShortArray &sd_key)

activate an approximation state based on its multi-index key
• virtual void clear_model_keys ()

reset initial state by removing all model keys for an approximation
• virtual void build ()

builds the approximation from scratch
• virtual void export_model (const String &fn_label="", const String &export_prefix="", const unsigned short export_format=NO_MODEL_FORMAT)

exports the approximation
• virtual void rebuild ()

rebuilds the approximation incrementally
• virtual void pop_coefficients (bool save_data)

removes entries from end of SurrogateData::{vars,resp}Data (last points appended, or as specified in args)
• virtual void push_coefficients ()

restores state prior to previous pop()
• virtual void finalize_coefficients ()

finalize approximation by applying all remaining trial sets
• virtual void clear_current_active_data ()

clear current build data in preparation for next build
• virtual void combine_coefficients ()

combine all level approximations into a single aggregate approximation
• virtual void combined_to_active_coefficients (bool clear_combined=true)

promote combined approximation into active approximation
• virtual void clear_inactive_coefficients ()

prune inactive coefficients following combination and promotion to active
• virtual Real value (const Variables &vars)

retrieve the approximate function value for a given parameter vector
• virtual const RealVector & gradient (const Variables &vars)

retrieve the approximate function gradient for a given parameter vector
• virtual const RealSymMatrix & hessian (const Variables &vars)

retrieve the approximate function Hessian for a given parameter vector
• virtual Real prediction_variance (const Variables &vars)

retrieve the variance of the predicted value for a given parameter vector
• virtual Real value (const RealVector &c_vars)

retrieve the approximate function value for a given parameter vector
• virtual const RealVector & gradient (const RealVector &c_vars)

retrieve the approximate function gradient for a given parameter vector
• virtual const RealSymMatrix & hessian (const RealVector &c_vars)
14.7. APPROXIMATION CLASS REFERENCE

retrieve the approximate function Hessian for a given parameter vector

- virtual Real prediction_variance (const RealVector &c_vars)
  retrieve the variance of the predicted value for a given parameter vector

- virtual Real mean ()
  return the mean of the expansion, where all active vars are random

- virtual Real mean (const RealVector &x)
  return the mean of the expansion for a given parameter vector, where a subset of the active variables are random

- virtual Real combined_mean ()
  return the mean of the combined expansion, where all active vars are random

- virtual Real combined_mean (const RealVector &x)
  return the mean of the combined expansion for a given parameter vector, where a subset of the active variables are random

- virtual const RealVector & mean_gradient ()
  return the gradient of the expansion mean

- virtual const RealVector & mean_gradient (const RealVector &x, const SizetArray &dvv)
  return the gradient of the expansion mean

- virtual Real variance ()
  return the variance of the expansion, where all active vars are random

- virtual Real variance (const RealVector &x)
  return the variance of the expansion for a given parameter vector, where a subset of the active variables are random

- virtual const RealVector & variance_gradient ()
  return the gradient of the expansion mean

- virtual const RealVector & variance_gradient (const RealVector &x, const SizetArray &dvv)
  return the gradient of the expansion mean

- virtual Real covariance (Approximation &approx_2)
  return the covariance between two response expansions, treating all variables as random

- virtual Real covariance (const RealVector &x, Approximation &approx_2)
  return the covariance between two response expansions, treating a subset of the variables as random

- virtual Real combined_covariance (Approximation &approx_2)
  return the covariance between two combined response expansions, where all active variables are random

- virtual Real combined_covariance (const RealVector &x, Approximation &approx_2)
  return the covariance between two combined response expansions, where a subset of the active variables are random

- virtual void compute_moments (bool full_stats=true, bool combined_stats=false)
- virtual void compute_moments (const RealVector &x, bool full_stats=true, bool combined_stats=false)
- virtual const RealVector & moments () const
- virtual const RealVector & expansion_moments () const
- virtual const RealVector & numerical_integration_moments () const
- virtual const RealVector & combined_moments () const
- virtual Real moment (size_t i) const
- virtual void moment (Real mom, size_t i)
  virtual Real combined_moment (size_t i) const
- virtual void combined_moment (Real mom, size_t i)
- virtual void clear_component_effects ()
- virtual void compute_component_effects ()
- virtual void compute_total_effects ()
• virtual const RealVector & sobol_indices() const
• virtual const RealVector & total_sobol_indices() const
• virtual ULongULongMap sparse_sobol_index_map() const
• virtual bool diagnostics_available() const
  check if diagnostics are available for this approximation type
• virtual Real diagnostic(const String &metric_type)
  retrieve a single diagnostic metric for the diagnostic type specified
• virtual RealArray cv_diagnostic(const StringArray &metric_types, unsigned num_folds)
  retrieve diagnostic metrics for the diagnostic types specified, applying
• virtual void primary_diagnostics(int fn_index)
  compute and print all requested diagnostics and cross-validation
• virtual RealArray challenge_diagnostic(const StringArray &metric_types, const RealMatrix &challenge_points, const RealVector &challenge_responses)
  compute requested diagnostics for user provided challenge pts
• virtual void challenge_diagnostics(int fn_index, const RealMatrix &challenge_points, const RealVector &challenge_responses)
  compute and print all requested diagnostics for user provided challenge pts
• virtual RealVector approximation_coefficients(bool normalized) const
  return the coefficient array computed by build()/rebuild()
• virtual void approximation_coefficients(const RealVector &approx_coeffs, bool normalized)
  set the coefficient array from external sources, rather than computing with build()/rebuild()
• virtual void coefficient_labels(std::vector<std::string> &coeff_labels) const
  print the coefficient array computed in build()/rebuild()
• virtual void print_coefficients(std::ostream &s, bool normalized)
  print the coefficient array computed in build()/rebuild()
• virtual int min_coefficients() const
  return the minimum number of samples (unknowns) required to build the derived class approximation type in numVars dimensions
• virtual int recommended_coefficients() const
  return the recommended number of samples (unknowns) required to build the derived class approximation type in numVars dimensions
• virtual int num_constraints() const
  return the number of constraints to be enforced via an anchor point
• virtual void expansion_coefficient_flag(bool)
• virtual bool expansion_coefficient_flag() const
• virtual void expansion_gradient_flag(bool)
• virtual bool expansion_gradient_flag() const
• virtual void clear_computed_bits() const
  clear tracking of computed moments, due to (expansion) change that invalidates previous results
• int min_points(bool constraint_flag) const
  return the minimum number of points required to build the approximation type in numVars dimensions. Uses *coefficients() and num_constraints().
• int recommended_points(bool constraint_flag) const
  return the recommended number of samples to build the approximation type in numVars dimensions (default same as min_points)
14.7. APPROXIMATION CLASS REFERENCE

- void **pop_data** (bool save_data)
  
  removes entries from end of SurrogateData::{vars,resp}Data (last points appended, or as specified in args)

- void **push_data** ()
  
  restores SurrogateData state prior to previous pop()

- void **finalize_data** ()
  
  finalize SurrogateData by applying all remaining trial sets

- const Pecos::SurrogateData & **surrogate_data** () const
  
  return approxData

- Pecos::SurrogateData & **surrogate_data** ()
  
  return approxData

- void **add** (const Pecos::SurrogateDataVars &sdv, bool anchor_flag, bool deep_copy, size_t key_index=NP-OS)
  
  append to SurrogateData::varsData

- void **add** (const Variables &vars, bool anchor_flag, bool deep_copy, size_t key_index=NP-OS)
  
  extract the relevant vectors from Variables and invoke add(RealVector&, IntVector&, RealVector&)

- void **add** (const RealVector &c_vars, const IntVector &di_vars, const RealVector &dr_vars, bool anchor_flag, bool deep_copy, size_t key_index=NP-OS)
  
  adds a new data point by appending to SurrogateData::varsData

- void **add** (const Real *sample_c_vars, bool anchor_flag, bool deep_copy, size_t key_index=NP-OS)
  
  create a RealVector view and invoke add(SurrogateDataVars&)

- void **add** (const Pecos::SurrogateDataResp &sdr, bool anchor_flag, bool deep_copy, size_t key_index=NP-OS)
  
  append to SurrogateData::respData

- void **add** (const Response &response, int fn_index, bool anchor_flag, bool deep_copy, size_t key_index=NP-OS)
  
  adds a new data point by appending to SurrogateData::respData

- void **add_array** (const RealMatrix &sample_vars, const RealVector &sample_resp, bool deep_copy=true, size_t key_index=NP-OS)
  
  add surrogate data from the provided sample and response data, assuming continuous variables and function values only

- void **pop_count** (size_t count, size_t key_index)
  
  appends to SurrogateData::popCountStack (number of entries to pop from end of SurrogateData::{vars,resp}Data, based on size of last data append)

- void **clear_data** ()
  
  clear SurrogateData::{vars,resp}Data for all approxDataKeys

- void **clear_active_data** ()
  
  clear active approximation data

- void **clear_inactive_data** ()
  
  clear inactive approximation data

- void **clear_active_popped** ()
  
  clear SurrogateData::popped{Vars,Resp}Trials,popCountStack for active key

- void **clear_popped** ()
  
  clear SurrogateData::popped{Vars,Resp}Trials,popCountStack for all keys
void set_bounds (const RealVector &c_l_bnds, const RealVector &c_u_bnds, const IntVector &di_l_bnds, const IntVector &di_u_bnds, const RealVector &dr_l_bnds, const RealVector &dr_u_bnds)

set approximation lower and upper bounds (currently only used by graphics)

• Approximation * approx_rep () const

returns approxRep for access to derived class member functions that are not mapped to the top Approximation level

Protected Member Functions

• Approximation (BaseConstructor, const ProblemDescDB &problem_db, const SharedApproxData &shared_data, const String &approx_label)

constructor initializes the base class part of letter classes (BaseConstructor overloading avoids infinite recursion in the derived class constructors - Coplien, p. 139)

• Approximation (NoDBBaseConstructor, const SharedApproxData &shared_data)

constructor initializes the base class part of letter classes (BaseConstructor overloading avoids infinite recursion in the derived class constructors - Coplien, p. 139)

• void check_points (size_t num_build_pts)

Check number of build points against minimum required.

Protected Attributes

• Pecos::SurrogateData approxData

contains the variables/response data for constructing a single approximation model (one response function). There is only one SurrogateData instance per Approximation, although it may contain keys for different model forms/resolutions and aggregations (e.g., discrepancies) among forms/resolutions.

• RealVector approxGradient

gradient of the approximation returned by gradient()

• RealSymMatrix approxHessian

Hessian of the approximation returned by hessian()

• String approxLabel

label for approximation, if applicable

• SharedApproxData * sharedDataRep

contains the approximation data that is shared among the response set

Private Member Functions

• Approximation * get_approx (ProblemDescDB &problem_db, const SharedApproxData &shared_data, const String &approx_label)

Used only by the standard envelope constructor to initialize approxRep to the appropriate derived type.

• Approximation * get_approx (const SharedApproxData &shared_data)

Used only by the alternate envelope constructor to initialize approxRep to the appropriate derived type.

Private Attributes

• Approximation * approxRep

pointer to the letter (initialized only for the envelope)

• int referenceCount

number of objects sharing approxRep
14.7. APPROXIMATION CLASS REFERENCE

14.7.1 Detailed Description

Base class for the approximation class hierarchy.

The Approximation class is the base class for the response data fit approximation class hierarchy in DAKOTA. One instance of an Approximation must be created for each function to be approximated (a vector of Approximations is contained in ApproximationInterface). For memory efficiency and enhanced polymorphism, the approximation hierarchy employs the "letter/envelope idiom" (see Coplien "Advanced C++", p. 133), for which the base class (Approximation) serves as the envelope and one of the derived classes (selected in Approximation::get_approx()) serves as the letter.

14.7.2 Constructor & Destructor Documentation

Approximation()

default constructor

The default constructor is used in Array<Approximation> instantiations and by the alternate envelope constructor. approxRep is NULL in this case (problem_db is needed to build a meaningful Approximation object). This makes it necessary to check for NULL in the copy constructor, assignment operator, and destructor.

Approximation(ProblemDescDB & problem_db, const SharedApproxData & shared_data, const String & approx_label)

standard constructor for envelope

Envelope constructor only needs to extract enough data to properly execute get_approx, since Approximation(-BaseConstructor, problem_db) builds the actual base class data for the derived approximations.

References Dakota::abort_handler(), Approximation::approxRep, and Approximation::get_approx().

Approximation(const SharedApproxData & shared_data)

alternate constructor

This is the alternate envelope constructor for instantiations on the fly. Since it does not have access to problem_db, it utilizes the NoDBBaseConstructor constructor chain.

References Dakota::abort_handler(), Approximation::approxRep, and Approximation::get_approx().

Approximation(const Approximation & approx)

copy constructor

Copy constructor manages sharing of approxRep and incrementing of referenceCount.

References Approximation::approxRep, and Approximation::referenceCount.

Approximation (BaseConstructor, const ProblemDescDB & problem_db, const SharedApproxData & shared_data, const String & approx_label) [protected]

constructor initializes the base class part of letter classes (BaseConstructor overloading avoids infinite recursion in the derived class constructors - Coplien, p. 139)
This constructor is the one which must build the base class data for all derived classes. `get_approx()` instantiates a derived class letter and the derived constructor selects this base class constructor in its initialization list (to avoid recursion in the base class constructor calling `get_approx()` again). Since the letter IS the representation, its rep pointer is set to NULL (an uninitialized pointer causes problems in ~Approximation).

**Approximation ( NoDBBaseConstructor, const SharedApproxData & shared_data ) [protected]**

constructor initializes the base class part of letter classes (BaseConstructor overloading avoids infinite recursion in the derived class constructors - Coplien, p. 139)

This constructor is the one which must build the base class data for all derived classes. `get_approx()` instantiates a derived class letter and the derived constructor selects this base class constructor in its initialization list (to avoid recursion in the base class constructor calling `get_approx()` again). Since the letter IS the representation, its rep pointer is set to NULL (an uninitialized pointer causes problems in ~Approximation).

### 14.7.3 Member Function Documentation

**Approximation operator= ( const Approximation & approx )**

assignment operator


References Approximation::approxRep, and Approximation::referenceCount.

**void build ( ) [virtual]**

builds the approximation from scratch

This is the common base class portion of the virtual fn and is insufficient on its own; derived implementations should explicitly invoke (or reimplement) this base class contribution.

Reimplemented in PecosApproximation, C3Approximation, VPSApproximation, GaussProcApproximation, SurfpackApproximation, SurrogatesGPApprox, SurrogatesPolyApprox, TaylorApproximation, QMEApproximation, and TANA3Approximation.

References Approximation::approxData, SharedApproxData::approxDataKeys, Approximation::approxRep, Approximation::build(), Approximation::check_points(), and Approximation::sharedDataRep.

Referenced by QMEApproximation::build(), TANA3Approximation::build(), TaylorApproximation::build(), SurfpackApproximation::build(), GaussProcApproximation::build(), Approximation::build(), VPSApproximation::build(), C3Approximation::build(), PecosApproximation::build(), and Approximation::rebuild().

**void clear_current_active_data ( ) [inline], [virtual]**

clear current build data in preparation for next build

Clear current but preserve hisory for active key (virtual function redefined by \{TANA3,QMEA\} Approximation to denote current while preserving previous points).

Reimplemented in QMEApproximation, and TANA3Approximation.

References Approximation::approxRep, Approximation::clear_active_data(), and Approximation::clear_current_active_data().

Referenced by Approximation::clear_current_active_data().
14.8. APPROXIMATIONINTERFACE CLASS REFERENCE

```cpp
void add_array ( const RealMatrix & sample_vars, const RealVector & sample_resp, bool deep_copy = true, size_t key_index = NPOS )
```

add surrogate data from the provided sample and response data, assuming continuous variables and function values only

- Short cut function (not used by ApproximationInterface).
- References Dakota::abort_handler(), Approximation::add(), Approximation::add_array(), and Approximation::approxRep.
- Referenced by Approximation::add_array().

```cpp
void clear_data() [inline]
```

clear SurrogateData::{vars,resp}Data for all approxDataKeys

- Clears out current + history for each tracked key (not virtual).
- References Approximation::approxData, Approximation::approxRep, and Approximation::clear_data().
- Referenced by Approximation::clear_data().

```cpp
Approximation * get_approx ( ProblemDescDB & problem_db, const SharedApproxData & shared_data, const String & approx_label ) [private]
```

Used only by the standard envelope constructor to initialize approxRep to the appropriate derived type.

- Used only by the envelope constructor to initialize approxRep to the appropriate derived type.
- References SharedApproxData::approxType, SharedApproxData::data_rep(), ProblemDescDB::get_bool(), and Dakota::strends().
- Referenced by Approximation::Approximation().

```cpp
Approximation * get_approx ( const SharedApproxData & shared_data ) [private]
```

Used only by the alternate envelope constructor to initialize approxRep to the appropriate derived type.

- Used only by the envelope constructor to initialize approxRep to the appropriate derived type.
- References SharedApproxData::approxType, SharedApproxData::data_rep(), and Dakota::strends().
- The documentation for this class was generated from the following files:
  - DakotaApproximation.hpp
  - DakotaApproximation.cpp

### 14.8 ApproximationInterface Class Reference

Derived class within the interface class hierarchy for supporting approximations to simulation-based results.

Inheritance diagram for ApproximationInterface:

```
  Interface

  ApproximationInterface
```
Public Member Functions

- `ApproximationInterface (ProblemDescDB &problem_db, const Variables &am_vars, bool am_cache, const String &am_interface_id, const StringArray &fn_labels)`
  primary constructor
- `ApproximationInterface (const String &approx_type, const UShortArray &approx_order, const Variables &am_vars, bool am_cache, const String &am_interface_id, size_t num_fns, short data_order, short output_level)`
  alternate constructor for instantiations on the fly
- `~ApproximationInterface ()`
  destructor

Protected Member Functions

- `void map (const Variables &vars, const ActiveSet &set, Response &response, bool asynch_flag=false)`
  the function evaluator: provides an approximate “mapping” from the variables to the responses using function-Surfaces
- `int minimum_points (bool constraint_flag) const`
  returns the minimum number of samples required to build the functionSurfaces
- `int recommended_points (bool constraint_flag) const`
  returns the recommended number of samples recommended to build the functionSurfaces
- `void active_model_key (const UShortArray &key)`
  activate an approximation state based on its multi-index key
- `void clear_model_keys ()`
  reset initial state by removing all model keys for an approximation
- `void approximation_function_indices (const IntSet &approx_fn_indices)`
  set the (currently active) approximation function index set
- `void update_approximation (const Variables &vars, const IntResponsePair &response_pr)`
- `void update_approximation (const RealMatrix &samples, const IntResponseMap &resp_map)`
- `void append_approximation (const VariablesArray &vars_array, const IntResponseMap &resp_map)`
- `void build_approximation (const RealVector &c_lbnds, const RealVector &c_ubnds, const IntVector &di_lbnds, const IntVector &di_u_bnds, const RealVector &dr_lbnds, const RealVector &dr_u_bnds)`
- `void export_approximation ()`
- `void rebuild_approximation (const BitArray &rebuild_fns)`
- `void pop_approximation (bool save_data)`
- `bool push_approximation ()`
  queries the approximation for the ability to retrieve a previous increment
- `void finalize_approximation ()`
  finalizes the approximation by applying all trial increments
- `void combine_approximation ()`
  combine the current approximation with previously stored data sets
- `void combined_to_active (bool clear_combined=true)`
14.8. APPROXIMATIONINTERFACE CLASS REFERENCE

promote the combined approximation to the currently active one

- `void clear_inactive()`
  - clear inactive approximation data
- `void clear_current_active_data()`
  - clears current data from an approximation interface
- `void clear_active_data()`
  - clears all data from an approximation interface
- `SharedApproxData & shared_approximation()`
  - retrieve the SharedApproxData within an ApproximationInterface
- `std::vector< Approximation > & approximations()`
  - retrieve the Approximations within an ApproximationInterface
- `const Pecos::SurrogateData & approximation_data(size_t fn_index)`
  - retrieve the approximation data from a particular Approximation within an ApproximationInterface
- `const RealVectorArray & approximation_coefficients(bool normalized=false)`
  - retrieve the approximation coefficients from each Approximation within an ApproximationInterface
- `void approximation_coefficients(const RealVectorArray &approx_coeffs, bool normalized=false)`
  - set the approximation coefficients within each Approximation within an ApproximationInterface
- `const RealVector & approximation_variances(const Variables &vars)`
  - retrieve the approximation variances from each Approximation within an ApproximationInterface
- `bool formulation_updated() const`
  - query for change in approximation formulation
- `Real2DArray cv_diagnostics(const StringArray &metrics, unsigned num_folds)`
  - approximation cross-validation quality metrics per response function
- `Real2DArray challenge_diagnostics(const StringArray &metric_types, const RealMatrix &challenge_pts, const RealVector &challenge_resps)`
- `const IntResponseMap & synchronize()`
  - recovers data from a series of asynchronous evaluations (blocking)
- `const IntResponseMap & synchronize_nowait()`
  - recovers data from a series of asynchronous evaluations (nonblocking)

Private Member Functions

- `void mixed_add(const Variables &vars, const Response &response, bool anchor)`
  - add variables/response data to functionSurfaces using a mixture of shallow and deep copies
- `void mixed_add(const Real *c_vars, const Response &response, bool anchor)`
  - add variables/response data to functionSurfaces using a mixture of shallow and deep copies
- `void shallow_add(const Variables &vars, const Response &response, bool anchor)`
  - add variables/response data to functionSurfaces using a shallow copy
- `void sample_to_variables(const Real *sample_c_vars, size_t num_cv, Variables &vars)`
  - populate continuous variables within vars from sample_c_vars
- `void update_pop_counts(const IntResponsePair &response_pr)`
  - append to the stack of pop counts within each of the functionSurfaces based on the active set definition within a single incoming response
- `void update_pop_counts(const IntResponseMap &resp_map)`
append to the stack of pop counts within each of the functionSurfaces based on the active set definitions within a map of incoming responses.

- void restore_data_key ()
  following Approximation::add() and Approximation::pop_count() operations, which may enumerate multiple approxDataKeys, restore the active approxData to the nominal key.

- void read_challenge_points ()
  Load approximation test points from user challenge points file.

Private Attributes

- IntSet approxFnIndices
  for incomplete approximation sets, this array specifies the response function subset that is approximated.

- SharedApproxData sharedData
  data that is shared among all functionSurfaces.

- std::vector< Approximation > functionSurfaces
  list of approximations, one per response function.

- RealVectorArray functionSurfaceCoeffs
  array of approximation coefficient vectors, one per response function.

- RealVector functionSurfaceVariances
  vector of approximation variances, one value per response function.

- String challengeFile
  data file for user-supplied challenge data (per interface, since may contain multiple responses).

- unsigned short challengeFormat
  tabular format of the challenge points file.

- bool challengeUseVarLabels
  whether to validate variable labels in header.

- bool challengeActiveOnly
  whether to import active only.

- RealMatrix challengePoints
  container for the challenge points data (variables only).

- RealMatrix challengeResponses
  container for the challenge points data (responses only).

- Variables actualModelVars
  copy of the actualModel variables object used to simplify conversion among differing variable views.

- bool actualModelCache
  indicates usage of an evaluation cache by the actualModel.

- String actualModelInterfaceId
  the interface id from the actualModel used for ordered PRPCache lookups.

- IntResponseMap beforeSynchResponseMap
  bookkeeping map to catalogue responses generated in map() for use in synchronize() and synchronize_nowait().
  This supports pseudo-asynchronous operations (approximate responses are always computed synchronously, but asynchronous virtual functions are supported through bookkeeping).
Static Private Attributes

- static size_t approxIdNum = 0
  
  counter for giving unique names to approximation interfaces

Additional Inherited Members

14.8.1 Detailed Description

Derived class within the interface class hierarchy for supporting approximations to simulation-based results. ApproximationInterface provides an interface class for building a set of global/local/multipoint approximations and performing approximate function evaluations using them. It contains a list of Approximation objects, one for each response function.

14.8.2 Member Function Documentation

void update_approximation ( const Variables & vars, const IntResponsePair & response_pr ) [protected], [virtual]

This function populates/replaces each Approximation::anchorPoint with the incoming variables/response data point.

  Reimplemented from Interface.

  References ApproximationInterface::actualModelCache, ApproximationInterface::actualModelInterfaceId, Dakota::data_pairs, Dakota::lookup_by_ids(), ApproximationInterface::mixed_add(), ApproximationInterface::restore_data_key(), and ApproximationInterface::shallow_add().

void update_approximation ( const RealMatrix & samples, const IntResponseMap & resp_map ) [protected], [virtual]

This function populates/replaces each Approximation::currentPoints with the incoming variables/response arrays.

  Reimplemented from Interface.

  References Dakota::abort_handler(), ApproximationInterface::actualModelCache, ApproximationInterface::actualModelInterfaceId, ApproximationInterface::actualModelVars, ApproximationInterface::approxFnIndices, ApproximationInterface::clear_active_data(), Dakota::data_pairs, ApproximationInterface::functionSurfaces, Dakota::lookup_by_ids(), ApproximationInterface::mixed_add(), ApproximationInterface::restore_data_key(), ApproximationInterface::sample_to_variables(), and ApproximationInterface::shallow_add().

void update_approximation ( const VariablesArray & vars_array, const IntResponseMap & resp_map ) [protected], [virtual]

This function populates/replaces each Approximation::currentPoints with the incoming variables/response arrays.

  Reimplemented from Interface.

  References Dakota::abort_handler(), ApproximationInterface::actualModelCache, ApproximationInterface::actualModelInterfaceId, ApproximationInterface::approxFnIndices, ApproximationInterface::clear_active_data(), Dakota::data_pairs, ApproximationInterface::functionSurfaces, Dakota::lookup_by_ids(), ApproximationInterface::mixed_add(), ApproximationInterface::restore_data_key(), and ApproximationInterface::shallow_add().
void append_approximation ( const Variables & vars, const IntResponsePair & response )
[protected], [virtual]

This function appends to each Approximation::currentPoints with one incoming variables/response data point.

Reimplemented from Interface.

References ApproximationInterface::actualModelCache, ApproximationInterface::actualModelInterfaceId, Dakota::data_pairs, Dakota::lookup_by_ids(), ApproximationInterface::mixed_add(), ApproximationInterface::restore_data_key(), ApproximationInterface::shallow_add(), and ApproximationInterface::update_pop_counts().

void append_approximation ( const RealMatrix & samples, const IntResponseMap & resp_map )
[protected], [virtual]

This function appends to each Approximation::currentPoints with multiple incoming variables/response data points.

Reimplemented from Interface.

References Dakota::abort_handler(), ApproximationInterface::actualModelCache, ApproximationInterface::actual-ModelInterfaceId, ApproximationInterface::actualModelVars, Dakota::data_pairs, Dakota::lookup_by_ids(), Approximation-Interface::mixed_add(), ApproximationInterface::restore_data_key(), ApproximationInterface::sample_to_variables(), ApproximationInterface::shallow_add(), and ApproximationInterface::update_pop_counts().

void append_approximation ( const VariablesArray & vars_array, const IntResponseMap & resp_map )
[protected], [virtual]

This function appends to each Approximation::currentPoints with multiple incoming variables/response data points.

Reimplemented from Interface.

References Dakota::abort_handler(), ApproximationInterface::actualModelCache, ApproximationInterface::actual-ModelInterfaceId, Dakota::data_pairs, Dakota::lookup_by_ids(), ApproximationInterface::mixed_add(), Approximation-Interface::restore_data_key(), ApproximationInterface::shallow_add(), and ApproximationInterface::update_pop_counts().

void build_approximation ( const RealVector & clbnds, const RealVector & cubnds, const IntVector & dlbnds, const IntVector & du_bnds, const RealVector & dlbnds, const IntVector & du_bnds, const RealVector & drlbnds, const RealVector & drubnds )
[protected], [virtual]

This function finds the coefficients for each Approximation based on the data passed through update_approximation() calls. The bounds are used only for graphics visualization.

Reimplemented from Interface.

References ApproximationInterface::approxFnIndices, SharedApproxData::build(), ApproximationInterface::challengeFile, ApproximationInterface::challengePoints, ApproximationInterface::challengeResponses, Approximation-Interface::functionSurfaces, ApproximationInterface::read_challenge_points(), SharedApproxData::set_bounds(), and ApproximationInterface::sharedData.

void export_approximation ()
[protected], [virtual]

This function calls export on each approximation

Reimplemented from Interface.

References ApproximationInterface::approxFnIndices, and ApproximationInterface::functionSurfaces.
void rebuild_approximation ( const BitArray & rebuild_fns ) [protected], [virtual]
This function updates the coefficients for each Approximation based on data increments provided by \{update.append\}_{-approximation}().
Reimplemented from Interface.
References ApproximationInterface::approxFnIndices, ApproximationInterface::functionSurfaces, SharedApproxData::rebuild(), and ApproximationInterface::sharedData.

void pop_approximation ( bool save_data ) [inline], [protected], [virtual]
This function removes data provided by a previous append_approximation() call, possibly different numbers for each function, or as specified in pop_count, which is assumed to be the same for all functions.
Reimplemented from Interface.
References ApproximationInterface::approxFnIndices, ApproximationInterface::functionSurfaces, SharedApproxData::pop(), Approximation::pop_coefficients(), Approximation::pop_data(), and ApproximationInterface::sharedData.

void push_approximation ( ) [inline], [protected], [virtual]
This function updates the coefficients for each Approximation based on data increments provided by \{update.append\}_{-approximation}().
Reimplemented from Interface.
References ApproximationInterface::approxFnIndices, ApproximationInterface::functionSurfaces, SharedApproxData::post_push(), SharedApproxData::pre_push(), Approximation::push_coefficients(), Approximation::push_data(), and ApproximationInterface::sharedData.

void restore_data_key ( ) [inline], [private]
following Approximation::add() and Approximation::pop_count() operations, which may enumerate multiple approxDataKeys, restore the active approxData to the nominal key
Restore active key in approxData using shared key.
References ApproximationInterface::active_model_key(), SharedApproxData::active_model_key(), ApproximationInterface::approxFnIndices, ApproximationInterface::functionSurfaces, and ApproximationInterface::sharedData.
Referenced by ApproximationInterface::append_approximation(), and ApproximationInterface::update_approximation().

void read_challenge_points ( ) [private]
Load approximation test points from user challenge points file.
Challenge data defaults to active/inactive, but user can override to active only.
References ApproximationInterface::actualModelVars, ApproximationInterface::challengeActiveOnly, ApproximationInterface::challengeFile, ApproximationInterface::challengeFormat, ApproximationInterface::challengePoints, ApproximationInterface::challengeResponses, ApproximationInterface::challengeUseVarLabels, Variables::copy(), ApproximationInterface::functionSurfaces, Interface::interface_id(), and Interface::outputLevel.
Referenced by ApproximationInterface::build_approximation().

14.8.3 Member Data Documentation

std::vector<Approximation> functionSurfaces [private]
list of approximations, one per response function
This formulation allows the use of mixed approximations (i.e., different approximations used for different response functions), although the input specification is not currently general enough to support it.

Referenced by ApproximationInterface::active_model_key(), ApproximationInterface::approximation_coefficients(), ApproximationInterface::approximation_data(), ApproximationInterface::approximation_variances(), ApproximationInterface::approximations(), ApproximationInterface::build_approximation(), ApproximationInterface::clear_active_data(), ApproximationInterface::clear_current_active_data(), ApproximationInterface::clear_inactive(), ApproximationInterface::clear_model_keys(), ApproximationInterface::combine_approximation(), ApproximationInterface::combined_to_active(), ApproximationInterface::cv_diagnostics(), ApproximationInterface::export_approximation(), ApproximationInterface::finalize_approximation(), ApproximationInterface::map(), ApproximationInterface::minimum_points(), ApproximationInterface::mixed_add(), ApproximationInterface::pop_approximation(), ApproximationInterface::push_approximation(), ApproximationInterface::read_challenge_points(), ApproximationInterface::rebuild_approximation(), ApproximationInterface::recommended_points(), ApproximationInterface::restore_data_key(), ApproximationInterface::shallow_add(), ApproximationInterface::update_approximation(), and ApproximationInterface::update_pop_counts().

The documentation for this class was generated from the following files:

- ApproximationInterface.hpp
- ApproximationInterface.cpp

14.9 APPSEvalMgr Class Reference

Evaluation manager class for APPSPACK.
Inherits Executor.

Public Member Functions

- APPSEvalMgr (Optimizer &, Model &model)
  constructor
- ~APPSEvalMgr ()
  destructor
- bool isReadyForWork () const
tells APPS whether or not there is a processor available to perform a function evaluation
- bool submit (const int apps_tag, const HOPSPACK::Vector &apps_xtrial, const HOPSPACK::EvalRequest-Type apps_request)
  performs a function evaluation at APPS-provided x_in
- int recv (int &apps_tag, HOPSPACK::Vector &apps_f, HOPSPACK::Vector &apps_cEqs, HOPSPACK::Vector &apps_cIneqs, string &apps_msg)
  returns a function value to APPS
- std::string getEvaluatorType () const
  return the type of the Dakota linked evaluator
- void printDebugInfo () (void) const
  empty implementation of debug info needed to complete the interface
- void printTimingInfo () (void) const
  empty implementation of timing info needed to complete the interface
- void set_asynch_flag (const bool dakotaAsynchFlag)
  publishes whether or not to do asynchronous evaluations
• void `set_blocking_synch` (const bool blockingSynchFlag)  
  publishes whether or not APPS is operating synchronously
• void `set_total_workers` (const int numDakotaWorkers)  
  publishes the number of processors available for function evaluations

Private Attributes

• `Optimizer & dakOpt`  
  reference to the DakotaOptimizer
• `Model & iteratedModel`  
  reference to the APPSOptimizer’s model passed in the constructor
• `bool modelAsynchFlag`  
  flag for asynchronous function evaluations
• `bool blockingSynch`  
  flag for APPS synchronous behavior
• `int numWorkersUsed`  
  number of processors actively performing function evaluations
• `int numWorkersTotal`  
  total number of processors available for performing function evaluations
• `RealVector xTrial`  
  trial iterate
• `std::map< int, int > tagList`  
  map of DAKOTA eval id to APPS eval id (for asynchronous evaluations)
• `std::map< int, RealVector > functionList`  
  map of APPS eval id to responses (for synchronous evaluations)
• `IntResponseMap dakotaResponseMap`  
  map of DAKOTA responses returned by `synchronize_nowait()`

14.9.1 Detailed Description

Evaluation manager class for APPSPACK.

The `APPSEvalMgr` class is derived from APPSPACK’s Executor class. It implements the methods of that class in such a way that allows DAKOTA to manage the computation of responses instead of APPS. Iterate and response values are passed between Dakota and APPSPACK via this interface.

14.9.2 Constructor & Destructor Documentation

`APPSEvalMgr ( Optimizer & opt, Model & model )`  
constructor

Evaluation manager class for APPSPACK.

The `APPSEvalMgr` class is derived from APPSPACK’s Executor class. It implements the methods of that class in such a way that allows DAKOTA to manage the computation of responses instead of APPS. Iterate and response values are passed between Dakota and APPSPACK via this interface.
14.9.3 Member Function Documentation

bool isReadyForWork() const

tells APPS whether or not there is a processor available to perform a function evaluation.
Check to see if all processors available for function evaluations are being used. If not, tell APPS that one is available.

References APPSEvalMgr::numWorkersTotal, and APPSEvalMgr::numWorkersUsed.

bool submit(const int apps_tag, const HOPSPACK::Vector &apps_xtrial, const HOPSPACK::EvalRequestType apps_request)

performs a function evaluation at APPS-provided x_in.
Convert APPSPACK vector of variables to DAKOTA vector of variables and perform function evaluation asynchronously or not as specified in the DAKOTA input deck. If evaluation is asynchronous, map the dakota id to the APPS tag. If evaluation is synchronous, map the responses to the APPS tag.

References Model::current_response(), Model::current_variables(), Model::evaluate(), Model::evaluate_nowait(), Model::evaluation_id(), Response::function_values(), APPSEvalMgr::functionList, APPSEvalMgr::iteratedModel, APPSEvalMgr::modelAsynchFlag, APPSEvalMgr::numWorkersTotal, APPSEvalMgr::numWorkersUsed, and APPSEvalMgr::tagList.

int recv(int &apps_tag, HOPSPACK::Vector &apps_f, HOPSPACK::Vector &apps_cEqs, HOPSPACK::Vector &apps_cIneqs, string &apps_msg)

returns a function value to APPS.
Retrieve a set of response values, convert to APPS data structures, and return them to APPS. APPS tags are tied to corresponding responses using the appropriate (i.e., asynchronous or synchronous) map.

References APPSEvalMgr::blockingSynch, APPSEvalMgr::dakOpt, APPSEvalMgr::dakotaResponseMap, APPSEvalMgr::functionList, Optimizer::get_responses_from_dakota(), APPSEvalMgr::iteratedModel, APPSEvalMgr::modelAsynchFlag, APPSEvalMgr::numWorkersUsed, Model::synchronize(), Model::synchronize_nowait(), and APPSEvalMgr::tagList.

The documentation for this class was generated from the following files:

- APPSEvalMgr.hpp
- APPSEvalMgr.cpp

14.10 APPSOptimizer Class Reference

Wrapper class for HOPSPACK.
The inheritance diagram for APPSOptimizer:
Public Member Functions

- `APPSOptimizer (ProblemDescDB &problem_db, Model &model)`
  constructor
- `APPSOptimizer (Model &model)`
  alternate constructor for on-the-fly instantiation without ProblemDescDB
- `APPSOptimizer ()`
  alternate constructor for even more rudimentary on-the-fly instantiation
- `~APPSOptimizer ()`
  destructor
- `void initialize_run ()`
- `void core_run ()`
  compute the optimal solution

Protected Member Functions

- `void set_apps_parameters ()`
  sets options for specific methods based on user specifications
- `void set_apps_traits ()`
  sets traits for specific TPL
- `void initialize_variables_and_constraints ()`
  initializes problem variables and constraints

Protected Attributes

- `int numTotalVars`
  Total across all types of variables.
- `HOPSPACK::ParameterList params`
  Pointer to APPS parameter list.
- `HOPSPACK::ParameterList * problemParams`
  Pointer to APPS problem parameter sublist.
- `HOPSPACK::ParameterList * linearParams`
  Pointer to APPS linear constraint parameter sublist.
- `HOPSPACK::ParameterList * mediatorParams`
Chapter 14. Class Documentation

Pointer to APPS mediator parameter sublist.

- HOPSPACK::ParameterList * citizenParams

Pointer to APPS citizen/algorithm parameter sublist.

- APPSEvalMgr * evalMgr

Pointer to the APPS evaluation manager object.

Additional Inherited Members

14.10.1 Detailed Description

Wrapper class for HOPSPACK.

The APPSOptimizer class provides a wrapper for HOPSPACK, a Sandia-developed C++ library for generalized pattern search. HOPSPACK defaults to a coordinate pattern search but also allows for augmented search patterns. It can solve problems with bounds, linear constraints, and general nonlinear constraints. APPSOptimizer uses an APPSEvalMgr object to manage the function evaluations.

The user input mappings are as follows: output max_function_evaluations, constraint_tol initial_delta, contraction_factor, variable_tolerance, solution_target, synchronization, merit_function, constraint_penalty, and smoothing_factor are mapped into HOPS’s "Display", "Maximum Evaluations", "Active Tolerance"/"Nonlinear Active Tolerance", "Initial Step", "Contraction Factor", "Step Tolerance", "Objective Target", "Synchronous Evaluations", "Penalty Function", "Penalty Parameter", and "Penalty Smoothing Value" data attributes. Refer to the HOPS web site [https://software.sandia.gov/trac/hopspack](https://software.sandia.gov/trac/hopspack) for additional information on HOPS objects and controls.

14.10.2 Member Function Documentation

void initialize_run() [virtual]

Allows us to initialize nonlinear equality constraint maps before inequality ones, ie a workaround in need of traits to specify constraint maps packing order - RWH

Reimplemented from Iterator.

References Optimizer::initialize_run(), and Minimizer::numNonlinearIneqConstraints.

void core_run() [virtual]

compute the optimal solution

core_run redefines the Optimizer virtual function to perform the optimization using HOPS. It first sets up the problem data, then executes minimize() on the HOPS optimizer, and finally catalogues the results.

Reimplemented from Iterator.

References Model::asynch_flag(), Iterator::bestResponseArray, Iterator::bestVariablesArray, Optimizer::constraintMapIndices, Optimizer::constraintMapMultipliers, Optimizer::constraintMapOffsets, APPSOptimizer::evalMgr, Model::evaluation_capacity(), APPSOptimizer::initialize_variables_and_constraints(), Iterator::iteratedModel, Optimizer::localObjectiveRecast, APPSOptimizer::numTotalVars, APPSOptimizer::params, APPSEvalMgr::set_asynch_flag(), and APPSEvalMgr::set_total_workers().

void set_apps_parameters() [protected]

sets options for specific methods based on user specifications

Set all of the HOPS algorithmic parameters as specified in the DAKOTA input deck. This is called at construction time.
References APPSOptimizer::citizenParams, Minimizer::constraintTol, APPSOptimizer::evalMgr, ProblemDescDB::get_real(), ProblemDescDB::get_string(), ProblemDescDB::is_null(), APPSOptimizer::linearParams, Iterator::maxEvalConcurrency, Iterator::maxFunctionEvals, APPSOptimizer::mediatorParams, Minimizer::numContinuousVars, Minimizer::numNonlinearConstraints, Iterator::outputLevel, APPSOptimizer::params, Iterator::probDescDB, APPSOptimizer::problemParams, and APPSEvalMgr::set_blocking_synch().

Referenced by APPSOptimizer::APPSOptimizer().

```cpp
void initialize_variables_and_constraints() [protected]
```

initializes problem variables and constraints

Set the variables and constraints as specified in the DAKOTA input deck. This is done at run time.

References Optimizer::constraintMapIndices, Iterator::iteratedModel, APPSOptimizer::linearParams, Minimizer::numContinuousVars, Minimizer::numDiscreteIntVars, Minimizer::numDiscreteRealVars, Minimizer::numDiscreteStringVars, Minimizer::numLinearEqConstraints, Minimizer::numLinearIneqConstraints, Minimizer::numNonlinearEqConstraints, APPSOptimizer::numTotalVars, and APPSOptimizer::problemParams.

Referenced by APPSOptimizer::core_run().

The documentation for this class was generated from the following files:

- APPSOptimizer.hpp
- APPSOptimizer.cpp

14.11 AppsTraits Class Reference

HOPSPACK-specific traits class.

Inheritance diagram for AppsTraits:

```
TraitsBase

AppsTraits
```

Public Types

- typedef HOPSPACK::Hopspack OptT
- typedef HOPSPACK::Vector VecT
- typedef HOPSPACK::Matrix MatT

Public Member Functions

- AppsTraits()
  
  default constructor

- virtual ~AppsTraits()
  
  destructor

- virtual bool is_derived() (A temporary query used in the refactor)

- bool supports_continuous_variables()


Return the flag indicating whether method supports continuous variables.

- bool supports_discrete_variables()

Return the flag indicating whether method supports discrete variables.

- bool supports_linear_equality()

Return the flag indicating whether method supports linear equalities.

- bool supports_linear_inequality()

Return the flag indicating whether method supports linear inequalities.

- bool supports_nonlinear_equality()

Return the flag indicating whether method supports nonlinear equalities.

- bool supports_nonlinear_inequality()

Return the format used for nonlinear equality constraints.

- NONLINEAR_EQUALITY_FORMAT nonlinear_equality_format()

Return the format used for nonlinear inequality constraints.

- NONLINEAR_INEQUALITY_FORMAT nonlinear_inequality_format()

Return the flag indicating whether method supports nonlinear inequalities.

// Static Public Member Functions

- static double noValue()
- static double getBestObj(const OptT &)
- static void copy_matrix_data(const RealMatrix &source, HOPSPACK::Matrix &target)

14.11.1 Detailed Description

HOPSPACK-specific traits class.

AppsTraits specializes some traits accessors by over-riding the default accessors in TraitsBase.

The documentation for this class was generated from the following files:

- APPSOptimizer.hpp
- APPSOptimizer.cpp

14.12 ApreproWriter Class Reference

Utility used in derived write_core to write in a prepro format.

Public Member Functions

- template<typename ArrayType>
  void operator() (std::ostream &, size_t start_index, size_t num_items, const ArrayType &array_data, StringMultiArrayConstView label_array)

14.12.1 Detailed Description

Utility used in derived write_core to write in a prepro format.

The documentation for this class was generated from the following file:

- DakotaVariables.hpp
14.13 AttachScaleVisitor Class Reference

Objects of this class are called by boost::apply_visitor to add dimension scales (RealScale or StringScale) to HDF5 datasets.

Inherits static visitor<>.

Public Member Functions

- AttachScaleVisitor (const StrStrSizet &iterator_id, const StringArray &location, const int &dim, const String &dset_name, const std::shared_ptr<HDF5IOHelper> &hdf5_stream)
  Construct with context for attaching the scale, including the iterator and location used to construct the scale, the dimension and name of the dataset to attach the scale to, the HDF5IOHelper instance.

- void operator() (const RealScale &scale)
  Called by boost::apply_visitor to process a RealScale.

- void operator() (const StringScale &scale)
  Called by boost::apply_visitor to process a StringScale.

- void operator() (const IntegerScale &scale)
  Called by boost::apply_visitor to process an IntegerScale.

Private Attributes

- StrStrSizet iteratorID
  Iterator ID for the method and execution.

- StringArray location
  Location used to create the dataset.

- int dimension
  Dimension of the dataset to attach the scale to.

- String dsetName
  Name of the dataset to attach the scale to.

- std::shared_ptr<HDF5IOHelper> hdf5Stream
  Instance of HDF5IOHelper.

14.13.1 Detailed Description

Objects of this class are called by boost::apply_visitor to add dimension scales (RealScale or StringScale) to HDF5 datasets.

The documentation for this class was generated from the following file:

- ResultsDBHDF5.hpp

14.14 BaseConstructor Struct Reference

Dummy struct for overloading letter-envelope constructors.

Public Member Functions

- BaseConstructor (int=0)
  C++ structs can have constructors.
14.14.1 Detailed Description

Dummy struct for overloading letter-envelope constructors.

BaseConstructor is used to overload the constructor for the base class portion of letter objects. It avoids infinite recursion (Coplien p.139) in the letter-envelope idiom by preventing the letter from instantiating another envelope. Putting this struct here avoids circular dependencies.

The documentation for this struct was generated from the following file:
- dakota_global_defs.hpp

14.15 BootstrapSampler< Data > Class Template Reference

Actual bootstrap sampler implementation for common data types.

Inheritance diagram for BootstrapSampler< Data >:

```
BootstrapSamplerBase< Data >
```

```
BootstrapSampler< Data >
```

```
BootstrapSamplerWithGS< Data, Getter, Setter >
```

Public Member Functions

- **BootstrapSampler** (const Data &orig_data, size_t block_size=1)
  Constructor for the sampler.
- **virtual ~BootstrapSampler** ()
  Destructor.
- **virtual void operator()** (size_t num_samp, Data &bootstrapped_sample)
  Generate and store a new boostrapped sample into bootstrapped_sample.

Protected Attributes

- **size_t blockSize**
  Size of the block defining a sample.

Additional Inherited Members

14.15.1 Detailed Description

template<typename Data> class Dakota::BootstrapSampler< Data >

Actual bootstrap sampler implementation for common data types.

Template requires the given type to support an STL-like interface, including a size method and begin and end methods returning random access iterators.

The documentation for this class was generated from the following file:
- BootstrapSampler.hpp
14.16. BOOTSTRAPSAMPLER< TEUCHOS::SERIALDENSEMATRIX< ORDINALTYPE, SCALARTYPE > > CLASS TEMPLATE REFERENCE

Bootstrap sampler that is specialized to allow for the bootstrapping of RealMatrix.

Inheritance diagram for BootstrapSampler< Teuchos::SerialDenseMatrix< OrdinalType, ScalarType > >:

```
  BootstrapSamplerBase< Teuchos::SerialDenseMatrix< OrdinalType, ScalarType > >
  BootstrapSampler< Teuchos::SerialDenseMatrix< OrdinalType, ScalarType > >
```

Public Types

- typedef Teuchos::SerialDenseMatrix< OrdinalType, ScalarType > MatType
  
  *Convenience definition.*

Public Member Functions

- **BootstrapSampler** (const MatType &orig_data, size_t block_size=1)
  
  *Constructor for the sampler.*

- virtual **~BootstrapSampler** ()
  
  *Destructor.*

- virtual void **operator()** (size_t num_samp, MatType &bootstrapped_sample)
  
  *Generate and store a new boostrapped sample into bootstrapped_sample.*

Protected Attributes

- size_t blockSize
  
  *Size of the block defining a sample.*

Additional Inherited Members

14.16.1 Detailed Description

```
template< typename OrdinalType, typename ScalarType >
class Dakota::BootstrapSampler< Teuchos::::SerialDenseMatrix< OrdinalType, ScalarType > >
```

Bootstrap sampler that is specialized to allow for the bootstrapping of RealMatrix.

The documentation for this class was generated from the following file:

- BootstrapSampler.hpp
14.17 **BootstrapSamplerBase< Data >** Class Template Reference

Base class/interface for the bootstrap sampler.

Inheritance diagram for BootstrapSamplerBase< Data >:

```
  BootstrapSamplerBase< Data >
  |
  BootstrapSampler< Data >
  |
  BootstrapSamplerWithGS< Data, Getter, Setter >
```

**Public Member Functions**

- **BootstrapSamplerBase** (size_t data_size, Data orig_data)
  
  Constructor for the bootstrap functor base.

- **virtual ~BootstrapSamplerBase ()**
  
  Destructor.

- **virtual void operator() (size_t num_samp, Data &bootstrapped_sample)=0**
  
  Generate and store a new bootstrapped sample into bootstrapped_sample.

- **virtual size_t getDataSize ()**
  
  Obtain the number of samples used in the empirical distribution.

- **virtual void operator() (Data &bootstrapped_sample)**
  
  Generate and store an dataSize out of dataSize bootstrap sample.

- **virtual Data operator() ()**
  
  Return bootstrapped sample.

**Static Public Member Functions**

- **static void set_seed (size_t seed)**

**Protected Attributes**

- **boost::random::uniform_int_distribution sampler**
  
  Uniform distribution to provide samples from the empirical distribution.

- **const size_t data_size**
  
  Size of the dataset defining the empirical distribution.

- **Data origData**
  
  Original data defining the empirical distribution TODO: Consider if it should be const (breaks Teuchos)

**Static Protected Attributes**

- **static boost::random::mt19937 bootstrapRNG bootstrapRNG**
  
  Random number generator to use for sampling.
14.18. **BOOTSTRAPSAMPLERWITHGS< DATA, GETTER, SETTER > CLASS TEMPLATE REFERENCE**

### 14.17.1 Detailed Description

**template<typename Data> class Dakota::BootstrapSamplerBase< Data >**

Base class/interface for the bootstrap sampler.

*BootstrapSamplerBase* defines the minimum interface for a bootstrap sampler and handles initialization of the random variate generation used by the bootstrap. Functor is templated on the data type, but does not actually define a data member.

#### 14.17.2 Member Data Documentation

**boost::random::mt19937 bootstrapRNG [static], [protected]**

Random number generator to use for sampling.

The bootstrap random number generator.

Referenced by BootstrapSampler< Data >::operator(), BootstrapSampler< Teuchos::SerialDenseMatrix< OrdinalType, ScalarType > >::operator(), and BootstrapSamplerWithGS< Data, Getter, Setter >::operator().

The documentation for this class was generated from the following file:

- BootstrapSampler.hpp

### 14.18 BootstrapSamplerWithGS< DATA, GETTER, SETTER > Class Template Reference

A derived sampler to allow for user specification of the accessor methods.

Inheritance diagram for BootstrapSamplerWithGS< Data, Getter, Setter >:

```
  BootstrapSamplerBase< Data >
     
  BootstrapSampler< Data >
     
  BootstrapSamplerWithGS< Data, Getter, Setter >
```

#### Public Member Functions

- **BootstrapSamplerWithGS** (const Data &orig_data, Getter getter_method, Setter setter_method)
  
  Constructor with extra arguments for the accessor methods.

- **virtual ~BootstrapSamplerWithGS ()**
  
  Destructor.

- **virtual void operator() (size_t num_samp, Data &bootstrapped_sample)**
  
  Generate and store a new bootstrapped sample into bootstrapped_sample TODO: bounds checking.
Protected Attributes

- Getter `getterMethod`
  
  Function to obtain a single sample from a Data object. Function should take a Data object and an unsigned integer corresponding to a sample index and return the sample.

- Setter `setterMethod`
  
  Function to place a single sample into a Data object. Function should take a Data object and an unsigned integer corresponding to the sample index to set.

Additional Inherited Members

14.18.1 Detailed Description

```cpp
template<typename Data, typename Getter, typename Setter>
class Dakota::BootstrapSamplerWithGS<Data, Getter, Setter>
```

A derived sampler to allow for user specification of the accessor methods.

The documentation for this class was generated from the following file:

- BootstrapSampler.hpp

14.19 C3Approximation Class Reference

Derived approximation class for global basis polynomials.

Inheritance diagram for C3Approximation:

```
Approximation

C3Approximation
```

Public Member Functions

- `C3Approximation()`
  
  default constructor

- `C3Approximation(ProblemDescDB &problem_db, const SharedApproxData &shared_data, const String &approx_label)`
  
  standard ProblemDescDB-driven constructor

- `C3Approximation(const SharedApproxData &shared_data)`
  
  alternate constructor

- `C3FnTrainPtrs & active ftp()`
  
  return the active C3FnTrainPtrs instance in levelApprox

- `C3FnTrainPtrs & combined ftp()`
  
  return combinedC3FTPtrs

- `size_t regression_size()`

- `size_t regression_size(const SizetVector &ranks, const UShortArray &orders)`

- `void expansion_coefficient_flag(bool coeff_flag)`
• bool expansion_coefficient_flag () const
• void expansion_gradient_flag (bool grad_flag)
• bool expansion_gradient_flag () const
• void compute_moments (bool full_stats=true, bool combined_stats=false)
• void compute_moments (const Pecos::RealVector &x, bool full_stats=true, bool combined_stats=false)
• const RealVector & moments () const
• const RealVector & expansion_moments () const
• const RealVector & numerical_integration_moments () const
• const RealVector & combined_moments () const
• Real moment (size_t i) const
• void moment (Real mom, size_t i)
• Real combined_moment (size_t i) const
• void combined_moment (Real mom, size_t i)
• void compute_component_effects ()
• void compute_total_effects ()
• void compute_all_sobol_indices (size_t)
• Real total_sobol_index (size_t)
• Real main_sobol_index (size_t)
• void sobol_iterate_apply (void (*)(double, size_t, size_t *, void *), void *)
• Real mean ()
  return the mean of the expansion, where all active vars are random
• Real mean (const RealVector &)
  return the mean of the expansion for a given parameter vector, where a subset of the active variables are random
• const RealVector & mean_gradient ()
  return the gradient of the expansion mean
• const RealVector & mean_gradient (const RealVector &, const SizetArray &)
  return the gradient of the expansion mean...
• Real variance ()
  return the variance of the expansion, where all active vars are random
• Real variance (const RealVector &)
  return the variance of the expansion for a given parameter vector, where a subset of the active variables are random
• const RealVector & variance_gradient ()
• const RealVector & variance_gradient (const RealVector &, const SizetArray &)
• Real covariance (Approximation &approx,)
  return the covariance between two response expansions, treating all variables as random
• Real covariance (const RealVector &, Approximation &approx,)
  return the covariance between two response expansions, treating a subset of the variables as random
• Real skewness ()
• Real kurtosis ()
• Real third_central ()
• Real fourth_central ()
• Real combined_mean ()
  return the mean of the combined expansion, where all active vars are random
• Real combined_mean (const RealVector &)
return the mean of the combined expansion for a given parameter vector, where a subset of the active variables are random

- Real combined_variance()
- Real combined_variance(const RealVector &)
- Real combined_covariance(Approximation &approx, 2)
  return the covariance between two combined response expansions, where all active variables are random
- Real combined_covariance(const RealVector &x, Approximation &approx, 2)
  return the covariance between two combined response expansions, where a subset of the active variables are random
- Real combined_third_central()
- Real combined_fourth_central()
- void synchronize_surrogate_data()
  update surrData to define aggregated data from raw data, when indicated by an active aggregated key
- void generate_synthetic_data(Pecos::SurrogateData &surr_data, const UShortArray &active_key, short combine_type)
  generate synthetic data for the surrogate QoI prediction corresponding to the level key preceding active key; for use in surplus estimation for new level data relative to a previous level's surrogate prediction

Protected Member Functions

- void active_model_key(const UShortArray &key)
  activate an approximation state based on its multi-index key
- void clear_model_keys()
  reset initial state by removing all model keys for an approximation
- Real value(const Variables &vars)
  retrieve the approximate function value for a given parameter vector
- const RealVector & gradient(const Variables &vars)
  retrieve the approximate function gradient for a given parameter vector
- const RealSymMatrix & hessian(const Variables &vars)
  retrieve the approximate function Hessian for a given parameter vector
- void build()
  builds the approximation from scratch
- void rebuild()
  rebuilds the approximation incrementally
- void pop_coefficients(bool save_data)
  removes entries from end of SurrogateData::{vars,resp}Data (last points appended, or as specified in args)
- void push_coefficients()
  restores state prior to previous pop()
- void combine_coefficients()
  combine all level approximations into a single aggregate approximation
- void combined_to_active_coefficients(bool clear_combined=true)
  promote combined approximation into active approximation
- void clear_inactive_coefficients()
  prune inactive coefficients following combination and promotion to active
- int min_coefficients() const
  return the minimum number of samples (unknowns) required to build the derived class approximation type in num- Vars dimensions
Private Member Functions

- Real **stored_value** (const RealVector &c_vars, const UShortArray &key)
- void **compute_derived_statistics** (C3FnTrainPtrs &ftp, size_t num_mom, bool overwrite=false)
- void **compute_derived_statistics_av** (C3FnTrainPtrs &ftp, size_t num_mom, bool overwrite=false)
- void **check_function_gradient** ()
  
  differentiate the ft to form its gradient, if not previously performed
- void **check_function_hessian** ()
  
  differentiate the fig to form the ft Hessian, if not previously performed
- Real **mean** (C3FnTrainPtrs &ftp)
  
  compute mean corresponding to the passed FT expansion
- Real **mean** (const RealVector &x, C3FnTrainPtrs &ftp)
  
  compute mean corresponding to the passed FT expansion
- Real **variance** (C3FnTrainPtrs &ftp)
  
  compute variance corresponding to the passed FT expansion
- Real **variance** (const RealVector &x, C3FnTrainPtrs &ftp)
  
  compute variance corresponding to the passed FT expansion
- Real **covariance** (C3FnTrainPtrs &ftp1, C3FnTrainPtrs &ftp2)
  
  compute covariance corresponding to the passed FT expansion
- Real **covariance** (const RealVector &x, C3FnTrainPtrs &ftp1, C3FnTrainPtrs &ftp2)
  
  compute covariance corresponding to the passed FT expansion
- Real **third_central** (C3FnTrainPtrs &ftp)
  
  compute 3rd central moment corresponding to the passed FT expansion
- Real **fourth_central** (C3FnTrainPtrs &ftp)
  
  compute 4th central moment corresponding to the passed FT expansion
- Real **skewness** (C3FnTrainPtrs &ftp)
  
  compute skewness corresponding to the passed FT expansion
- Real **kurtosis** (C3FnTrainPtrs &ftp)
  
  compute excess kurtosis corresponding to the passed FT expansion

Private Attributes

- std::map< UShortArray, C3FnTrainPtrs > levelApprox
  
  set of pointers to QoI approximation data for each model key
- std::map< UShortArray, C3FnTrainPtrs >::iterator levApproxIter
  
  iterator to active levelApprox
- C3FnTrainPtrs prevC3FTPtrs
  
  the previous approximation, cached for restoration
- std::deque< C3FnTrainPtrs > poppedLevelApprox
  
  bookkeeping for previously evaluated FT approximations that may be restored
- C3FnTrainPtrs combinedC3FTPtrs
  
  the combined approximation, summed across model keys
• bool expansionCoeffFlag
  flag indicating need to build a fn train approximation for this QoI
• bool expansionCoeffGradFlag
  flag indicating need to build a fn train gradient approx for this QoI
• std::map< UShortArray, RealVector > primaryMoments
  mean and central moments 2/3/4 computed from either the expansion form
• std::map< UShortArray, RealVector >::iterator primaryMomIter
  iterator to active entry in primaryMoments
• RealVector secondaryMoments
  secondary (numerical) moments: inactive
• RealVector combinedMoments
  combined moments from multilevel-multifidelity FT rollup

Additional Inherited Members

14.19.1 Detailed Description

Derived approximation class for global basis polynomials.

The PecosApproximation class provides a global approximation based on basis polynomials. This includes orthogonal polynomials used for polynomial chaos expansions and interpolation polynomials used for stochastic collocation.

14.19.2 Member Function Documentation

size_t regression_size ( const SizetVector & ranks, const UShortArray & orders )

compute the regression size (number of unknowns) for ranks per dimension and (polynomial) orders per dimension

References Dakota::abort_handler(), SharedApproxData::numVars, and Approximation::sharedDataRep.

void build ( ) [protected], [virtual]

builds the approximation from scratch

This is the common base class portion of the virtual fn and is insufficient on its own; derived implementations should explicitly invoke (or reimplement) this base class contribution.

Reimplemented from Approximation.

References Dakota::abort_handler(), SharedC3ApproxData::adaptConstruct, SharedC3ApproxData::adaptRank, Approximation::approxData, Approximation::build(), SharedC3ApproxData::c3Verbosity, SharedC3ApproxData::crossVal, C3FnTrainPtrs::free_all(), C3FnTrainPtrs::function_train(), SharedC3ApproxData::kickRank, C3Approximation::levApproxIter, SharedC3ApproxData::maxRank, SharedC3ApproxData::maxSolverIterations, SharedC3ApproxData::multiApproxOpts, SharedApproxData::numVars, SharedC3ApproxData::oneApproxOpts, SharedApproxData::outputLevel, SharedC3ApproxData::regressRegParam, SharedC3ApproxData::regressType, SharedC3ApproxData::roundingTol, Approximation::sharedDataRep, SharedC3ApproxData::solverTol, SharedC3ApproxData::start_rank(), C3Approximation::synchronize surrogate data(), and Dakota::write precision.

Referenced by C3Approximation::rebuild().

The documentation for this class was generated from the following files:

• C3Approximation.hpp
• C3Approximation.cpp
14.20  C3FnTrainPtrs Class Reference

Handle for reference-counted pointer to C3FnTrainPtrsRep body.

Public Member Functions

- `C3FnTrainPtrs () default constructor`
- `C3FnTrainPtrs (const C3FnTrainPtrs &ftp) copy constructor`
- `~C3FnTrainPtrs () destructor`
- `C3FnTrainPtrs & operator= (const C3FnTrainPtrs &ftp) assignment operator`
- `C3FnTrainPtrs copy () const perform a deep copy (copy ctor and operator= use shallow copies)`
- `void swap (C3FnTrainPtrs &ftp) swap ftpReps between two envelopes`
- `void free_ft () free FT storage for value, gradient, and Hessian expansions`
- `void free_all () augment free_ft() with derived functions and global sensitivities`
- `void ftDerivedFunctions_init_null () initialize derived funtions pointers to NULL`
- `void ftDerivedFunctions_create (struct MultiApproxOpts *opts, size_t num_mom, Real round_tol) allocate derived funtions pointers (standard mode)`
- `void ftDerivedFunctions_create_av (struct MultiApproxOpts *opts, const SizetArray &rand_indices, Real round_tol) allocate derived funtions pointers (all variables mode)`
- `void ftDerivedFunctions_free () deallocate derived funtions pointers`
- `struct FunctionTrain * function_train () get pointer to the FunctionTrain approximation`
- `void function_train (struct FunctionTrain *ft) set pointer to the FunctionTrain approximation`
- `struct FT1DArray * ftGradient () get pointer to the FunctionTrain gradient`
- `void ftGradient (struct FT1DArray *ftg) set pointer to the FunctionTrain gradient`
- `struct FT1DArray * ftHessian () get pointer to the FunctionTrain Hessian`
- `void ftHessian (struct FT1DArray *fth) set pointer to the FunctionTrain Hessian`
- `struct FTDerivedFunctions & derived_functions ()`
return reference to the FTDerivedFunctions instance
• struct C3SobolSensitivity * sobol ()
  get pointer to the Sobol' indices object
• void sobol (struct C3SobolSensitivity *ss)
  set pointer to the Sobol' indices object

Public Attributes
• std::shared_ptr<C3FnTrainPtrsRep> ftpRep
  (shared) pointer to body instance

14.20.1 Detailed Description
Handle for reference-counted pointer to C3FnTrainPtrsRep body.

The documentation for this class was generated from the following files:

• C3Approximation.hpp
• C3Approximation.cpp

14.21 callback_data Struct Reference

Public Attributes
• double rosen_cdv_upper_bd
  upper bound value to pass through parser to callback function

14.21.1 Detailed Description
Data structure to pass application-specific values through Dakota back to the callback function, for example to convey late updates to bounds, initial points, etc., to Dakota.

The documentation for this struct was generated from the following file:

• library_mode.cpp

14.22 CholeskySolver Class Reference

The CholeskySolver class can be used to solve $Ax = b$, provided that $A$ is a symmetric matrix.

Inheritance diagram for CholeskySolver:

```
  LinearSolverBase
    |
    +-- CholeskySolver
```
Public Member Functions

- **CholeskySolver ()**
  Constructor.
- **~CholeskySolver ()**
  Destructor.
- **bool is_factorized () const override**
  Query to determine if the matrix of the solver has been factored.
- **void factorize (const MatrixXd &A) override**
  Perform the matrix factorization for the linear solver matrix.
- **void solve (const MatrixXd &A, const MatrixXd &b, MatrixXd &x) override**
  Find a solution to Ax = b.
- **void solve (const MatrixXd &b, MatrixXd &x) override**
  Find a solution to Ax = b when A is already factorized.

Private Attributes

- **std::shared_ptr<Eigen::LDLT<MatrixXd>> LDLT_Ptr**
  Cached LDLᵀ factorization.

Additional Inherited Members

14.22.1 Detailed Description

The CholeskySolver class can be used to solve Ax = b, provided that A is a symmetric matrix.

14.22.2 Member Function Documentation

**void factorize ( const MatrixXd & A ) [override], [virtual]**

Perform the matrix factorization for the linear solver matrix.

<table>
<thead>
<tr>
<th>in</th>
<th>A</th>
<th>The incoming matrix to factorize.</th>
</tr>
</thead>
</table>

Reimplemented from LinearSolverBase.
References CholeskySolver::LDLT_Ptr.
Referenced by CholeskySolver::solve().

**void solve ( const MatrixXd & A, const MatrixXd & b, MatrixXd & x ) [override], [virtual]**

Find a solution to Ax = b.

<table>
<thead>
<tr>
<th>in</th>
<th>A</th>
<th>The linear system left-hand-side matrix.</th>
</tr>
</thead>
</table>
CHAPTER 14. CLASS DOCUMENTATION

<table>
<thead>
<tr>
<th>in</th>
<th>b</th>
<th>The linear system right-hand-side (multi-)vector.</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>x</td>
<td>The linear system solution (multi-)vector.</td>
</tr>
</tbody>
</table>

Reimplemented from LinearSolverBase.
References CholeskySolver::factorize().

```cpp
void solve ( const MatrixXd & b, MatrixXd & x ) [override], [virtual]
```

Find a solution to Ax = b when A is already factorized.

Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>b</th>
<th>The linear system right-hand-side (multi-)vector.</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>x</td>
<td>The linear system solution (multi-)vector.</td>
</tr>
</tbody>
</table>

Reimplemented from LinearSolverBase.
References CholeskySolver::LDLT_Ptr.

The documentation for this class was generated from the following files:

- LinearSolvers.hpp
- LinearSolvers.cpp

14.23 COLINApplication Class Reference

Inherits Application< colin::MO_MINLP2_problem >.

Public Member Functions

- COLINApplication ()
  Default constructor. Requred by COLIN's ApplicationHandle creation.
- COLINApplication (Model &model)
  Constructor with Model (not presently used).
- ~COLINApplication ()
  Destructor.
- void set_problem (Model &model)
  Helper function called after default construction to extract problem information from the Model and set it for COLIN.
- void set_blocking_synch (const bool blockingSynchFlag)
  Publishes whether or not COLIN is operating synchronously
- virtual utilib::Any spawn_evaluation_impl (const utilib::Any &domain, const colin::AppRequest::request_map_t &requests, utilib::seed_t &seed)
  Schedule one or more requests at specified domain point, returning a DAKOTA-specific evaluation tracking ID.
- virtual bool evaluation_available ()
  Check to see if there are any function values ready to be collected.
- virtual void perform_evaluation_impl (const utilib::Any &domain, const colin::AppRequest::request_map_t &requests, utilib::seed_t &seed, colin::AppResponse::response_map_t &colin_responses)
  Perform a function evaluation at t given point.
- virtual utilib::Any collect_evaluation_impl (colin::AppResponse::response_map_t &responses, utilib::seed_t &seed)
Collect a completed evaluation from DAKOTA.

- virtual void colin_request_to_dakota_request (const utilib::Any &domain, const colin::AppRequest::request_map_t &requests, utilib::seed_t &seed)
  
  Helper function to convert evaluation request data from COLIN structures to DAKOTA structures.

- virtual void dakota_response_to_colin_response (const Response &dakota_response, colin::AppResponse::response_map_t &colin_responses)
  
  Helper function to convert evaluation response data from DAKOTA structures to COLIN structures.

- virtual bool map_domain (const utilib::Any &src, utilib::Any &native, bool forward=true) const
  
  Map the domain point into data type desired by this application context.

Protected Attributes

- Model iteratedModel
  
  Shallow copy of the model on which COLIN will iterate.

- bool blockingSynch
  
  Flag for COLIN synchronous behavior (Pattern Search only).

- ActiveSet activeSet
  
  Local copy of model’s active set for convenience.

- std::vector< int > requestedEvals
  
  Evaluations queued for async evaluation.

- IntResponseMap dakota_responses
  
  eval_id to response mapping to cache completed jobs.

14.23.1 Detailed Description

COLINApplication is a DAKOTA class that is derived from COLIN’s Application hierarchy. It redefines a variety of virtual COLIN functions to use the corresponding DAKOTA functions. This is a more flexible algorithm library interfacing approach than can be obtained with the function pointer approaches used by NPSOLOptimizer and SNLLOptimizer.

14.23.2 Member Function Documentation

void set_problem ( Model & model )

Helper function called after default construction to extract problem information from the Model and set it for COLIN.

Set variable bounds and linear and nonlinear constraints. This avoids using probDescDB, so it is called by both the standard and the on-the-fly COLINOptimizer constructors.

References Response::active_set(), COLINApplication::activeSet, Model::continuous_lower_bounds(), Model::continuous_upper_bounds(), Dakota::copy_data_partial(), Model::current_response(), Model::cv(), Model::discrete_int_lower_bounds(), Model::discrete_int_sets(), Model::discrete_int_upper_bounds(), Model::discrete_set_int_values(), Model::discrete_set_real_values(), Model::discrete_set_string_values(), Model::div(), Model::drv(), Model::dsv(), Dakota::get_nonlinear_bounds(), COLINApplication::iteratedModel, Model::linear_eq_constraint_coeff(), Model::linear_eq_constraint_targets(), Model::linear_ineq_constraint_coeffs(), Model::linear_ineq_constraint_lower_bounds(), Model::linear_ineq_constraint_upper_bounds(), Model::num_linear_eq_constraints(), Model::num_linear_ineq_constraints(), Model::num_primary_fns(), Model::num_secondary_fns(), and Model::primary_response_fn_sense().

Referenced by COLINApplication::COLINApplication().
utilib::Any spawn_evaluation_impl ( const utilib::Any & domain, const colin::AppRequest::request_map_t & requests, utilib::seed_t & seed ) [virtual]

Schedule one or more requests at specified domain point, returning a DAKOTA-specific evaluation tracking ID.

Schedule one or more requests at specified domain point, returning a DAKOTA-specific evaluation tracking ID. This is only called by COLIN’s concurrent evaluator, which is only instantiated when the Model supports asynch evals. The domain point is guaranteed to be compatible with data type specified by map_domain(...)

References COLINApplication::colin_request_to_dakota_request(), Model::evaluate_nowait(), Model::evaluation_id(), and COLINApplication::iteratedModel.

bool evaluation_available ( ) [virtual]

Check to see if there are any function values ready to be collected.

Check to see if any asynchronous evaluations have finished. This is only called by COLIN’s concurrent evaluator, which is only instantiated when the Model supports asynch evals.

References COLINApplication::blockingSynch, COLINApplication::dakota_responses, COLINApplication::iteratedModel, Model::synchronize(), and Model::synchronize_nowait().

void perform_evaluation_impl ( const utilib::Any & domain, const colin::AppRequest::request_map_t & requests, utilib::seed_t & seed, colin::AppResponse::response_map_t & colin_responses ) [virtual]

Perform a function evaluation at t given point.

Perform an evaluation at a specified domain point. Wait for and return the response. This is only called by COLIN’s serial evaluator, which is only instantiated when the Model does not support asynch evals. The domain point is guaranteed to be compatible with data type specified by map_domain(...)

References COLINApplication::colin_request_to_dakota_request(), Model::current_response(), COLINApplication::dakota_response_to_colin_response(), Model::evaluate(), and COLINApplication::iteratedModel.

utilib::Any collect_evaluation_impl ( colin::AppResponse::response_map_t & colin_responses, utilib::seed_t & seed ) [virtual]

Collect a completed evaluation from DAKOTA.

Collect the next completed evaluation from DAKOTA. Always returns the evalid of the response returned.

References COLINApplication::dakota_response_to_colin_response(), and COLINApplication::dakota_responses.

void colin_request_to_dakota_request ( const utilib::Any & domain, const colin::AppRequest::request_map_t & requests, utilib::seed_t & seed ) [virtual]

Helper function to convert evaluation request data from COLIN structures to DAKOTA structures.

Map COLIN info requests to DAKOTA objectives and constraints.

References Model::continuous_variables(), Model::discrete_int_sets(), Model::discrete_int_variable(), Model::discrete_real_variable(), Model::discrete_set_int_values(), Model::discrete_set_real_values(), Model::discrete_set_string_values(), Model::discrete_string_variable(), Model::div(), Model::drv(), Model::dsv(), COLINApplication::iteratedModel, Model::response_size(), and Dakota::set_index_to_value().

Referenced by COLINApplication::perform_evaluation_impl(), and COLINApplication::spawn_evaluation_impl().

void dakota_response_to_colin_response ( const Response & dakota_response, colin::AppResponse::response_map_t & colin_responses ) [virtual]

Gelper function to convert evaluation response data from DAKOTA structures to COLIN structures.
14.24. COLINOPTIMIZER CLASS REFERENCE

Map DAKOTA objective and constraint values to COLIN response.
References Response::active_set_request_vector(), and Response::function_value().
Referenced by COLINAapplication::collect_evaluation_impl(), and COLINAapplication::perform_evaluation_impl().

bool map_domain ( const utilib::Any & src, utilib::Any & native, bool forward = true ) const [virtual]
Map the domain point into data type desired by this application context.
Map the domain point into data type desired by this application context (utilib::MixedIntVars). This data type can be exposed from the Any &domain presented to spawn and collect.
The documentation for this class was generated from the following files:
• COLINAapplication.hpp
• COLINAapplication.cpp

14.24 COLINOptimizer Class Reference

Wrapper class for optimizers defined using COLIN.
Inheritance diagram for COLINOptimizer:

Public Member Functions

• COLINOptimizer (ProblemDescDB &problem_db, Model &model)
  *standard constructor*
• COLINOptimizer (const String &method_name, Model &model, int seed, int max_iter, int max_eval)
  *alternate constructor for on-the-fly instantiations*
• COLINOptimizer (const String &method_name, Model &model)
  *alternate constructor for Iterator instantiations by name*
• ~COLINOptimizer ()
  *destructor*
• void reset ()
  *clears internal optimizer state*
• void core_run ()
  *iterates the COLIN solver to determine the optimal solution*
• bool returns_multiple_points () const
  *some COLIN methods can return multiple points*
Protected Member Functions

- void solver_setup (unsigned short method_name)
  convenience function for setting up the particular COLIN solver and appropriate Application
- void set_rng (int seed)
  sets up the random number generator for stochastic methods
- void set_solver_parameters ()
  sets construct-time options for specific methods based on user specifications, including calling method-specific set
  functions
- void post_run (std::ostream &s)
  Get the final set of points from the solver Look up responses and sort, first according to constraint violation, then
  according to function value.
- std::pair<bool, bool> colin_cache_lookup (const colin::AppResponse &colinResponse, Response &tmp-
  ResponseHolder)
  Retrieve response from Colin AppResponse, return pair indicating success for <objective, constraints>
- double constraintViolation (const Response &tmpResponseHolder)
  Compute constraint violation, based on nonlinear constraints in iteratedModel and provided Response data.

Protected Attributes

- short solverType
  COLIN solver sub-type as enumerated in COLINOptimizer.cpp.
- colin::SolverHandle colinSolver
  handle to the COLIN solver
- std::pair<colin::ApplicationHandle, COLINApplication *> colinProblem
  handle and pointer to the COLINApplication object
- colin::EvaluationManager_Base * colinEvalMgr
  pointer to the COLIN evalutaion manager object
- utilib::RNG * rng
  random number generator pointer
- bool blockingSynch
  the synchronization setting: true if blocking, false if nonblocking
- Real constraint_penalty
  Buffer to hold problem constraint_penalty parameter.
- bool constant_penalty
  Buffer to hold problem constant_penalty parameter.

Additional Inherited Members

14.24.1 Detailed Description

Wrapper class for optimizers defined using COLIN.

The COLINOptimizer class wraps COLIN, a Sandia-developed C++ optimization interface library. A variety of
COLIN optimizers are defined in COLIN and its associated libraries, including SCOLIB which contains the
optimization components from the old COLINY (formerly SGOPT) library. COLIN contains optimizers such as genetic algorithms, pattern search methods, and other nongradient-based techniques. COLINOptimizer uses a COLINApplication object to perform the function evaluations.

The user input mappings are as follows: max_iterations, max_function_evaluations, convergence_tolerance, and solution_accuracy are mapped into COLIN’s max_iterations, max_function_evaluations, this_trial, function_value.tolerance, sufficient.objective.value properties. An outputLevel is mapped to COLIN’s output_level property and a setting of debug activates output of method initialization and sets the COLIN debug attribute to 10000 for the DEBUG output level. Refer to [Hart, W.E., 2006] for additional information on COLIN objects and controls.

14.24.2 Constructor & Destructor Documentation

COLINOptimizer ( ProblemDescDB & problem_db, Model & model )
standard constructor
Standard constructor.
References ProblemDescDB::get_int(), ProblemDescDB::get_ushort(), Iterator::probDescDB, COLINOptimizer::set_rng(), COLINOptimizer::set_solver_parameters(), and COLINOptimizer::solver_setup().

COLINOptimizer ( const String & method_name, Model & model, int seed, int max_iter, int max_eval )
alternate constructor for on-the-fly instantiations
Alternate constructor for on-the-fly instantiations.
References Iterator::maxFunctionEvals, Iterator::maxIterations, Iterator::method_string_to_enum(), COLINOptimizer::set_rng(), COLINOptimizer::set_solver_parameters(), and COLINOptimizer::solver_setup().

COLINOptimizer ( const String & method_name, Model & model )
alternate constructor for Iterator instantiations by name
Alternate constructor for Iterator instantiations by name.
References Iterator::method_string_to_enum(), COLINOptimizer::set_solver_parameters(), and COLINOptimizer::solver_setup().

14.24.3 Member Function Documentation

void core_run ( ) [virtual]
iterates the COLIN solver to determine the optimal solution
core_run redefines the Optimizer virtual function to perform the optimization using COLIN. It first sets up the problem data, then executes optimize() on the COLIN solver and finally catalogues the results.
Reimplemented from Iterator.
References Dakota::NPOS, Dakota::abort_handler(), Model::async_flag(), COLINOptimizer::blockingSynch, COLINOptimizer::colinEvalMgr, COLINOptimizer::colinProblem, COLINOptimizer::colinSolver, COLINOptimizer::constant_penalty, COLINOptimizer::constraint_penalty, Model::continuous_variables(), Model::discrete_int_sets(), Model::discrete_int_variables(), Model::discrete_real_variables(), Model::discrete_set_int_values(), Model::discrete_set_real_values(), Model::discrete_set_string_values(), Model::discrete_string_variables(), Model::evaluation_capacity(), Iterator::iteratedModel, Minimizer::numDiscreteIntVars, Minimizer::numDiscreteRealVars, Minimizer::numDiscreteStringVars, Iterator::outputLevel, Dakota::set_value_to_index(), and COLINOptimizer::solverType.
bool returns_multiple_points() const [virtual]

some COLIN methods can return multiple points
    Designate which solvers can return multiple final points.
    Reimplemented from Iterator.
    References COLINOptimizer::solverType.

void solver_setup(unsigned short method_name) [protected]

convenience function for setting up the particular COLIN solver and appropriate Application
    This convenience function is called by the constructors in order to instantiate the solver.
    References COLINOptimizer::colinProblem, COLINOptimizer::colinSolver, COLINOptimizer::constant_penalty,
    COLINOptimizer::constraint_penalty, ProblemDescDB::get_string(), Iterator::method_enum_to_string(), Iterator-
    ::probDescDB, and COLINOptimizer::solverType.
    Referenced by COLINOptimizer::COLINOptimizer().

void set_rng(int seed) [protected]

sets up the random number generator for stochastic methods
    Instantiate random number generator (RNG).
    References COLINOptimizer::colinSolver, and COLINOptimizer::rng.
    Referenced by COLINOptimizer::COLINOptimizer().

void set Solver parameters() [protected]

sets construct-time options for specific methods based on user specifications, including calling method-specific
    set functions
    Sets solver properties based on user specifications. Called at construction time.
    References Model::asynch_flag(), COLINOptimizer::blockingSynch, COLINOptimizer::colinSolver, COLI-
    NOptimizer::constant_penalty, COLINOptimizer::constraint_penalty, Iterator::convergenceTol, ProblemDescDB-
    ::get_bool(), ProblemDescDB::get_int(), ProblemDescDB::get_real(), ProblemDescDB::get_sa(), ProblemDescD-
    B::get_string(), ProblemDescDB::is_null(), Iterator::iteratedModel, Iterator::maxEvalConcurrency, Iterator::max-
    FunctionEvals, Iterator::maxIterations, Minimizer::numContinuousVars, Iterator::outputLevel, Iterator::probDesc-
    DB, and COLINOptimizer::solverType.
    Referenced by COLINOptimizer::COLINOptimizer().

void post_run(std::ostream & s) [protected, [virtual]

Get the final set of points from the solver Look up responses and sort, first according to constraint violation, then
    according to function value.
    Supplement Optimizer::post_run to first retrieve points from the Colin cache (or possibly the Dakota DB)
    and rank them. When complete, this function will populate bestVariablesArray and bestResponsesArray with
    iterator-space data, that is, in the context of the solver, leaving any further untransformation to Optimizer.
    Reimplemented from Iterator.
    References Iterator::bestResponseArray, Iterator::bestVariablesArray, COLINOptimizer::colin_cache_lookup(),
    COLINOptimizer::colinProblem, COLINOptimizer::colinSolver, COLINOptimizer::constraint_violation(), Variables-
    ::continuous_variables(), Response::copy(), Variables::copy(), Model::current_response(), Model::current_variables(),
    Model::discrete_int_sets(), Variables::discrete_int_variable(), Variables::discrete_real_variable(), Model::discrete-_set_int_values(), Model::discrete_set_real_values(), Model::discrete_set_string_values(), Variables::discrete_string-
    _variable(), Response::function_values(), Iterator::iteratedModel, Optimizer::localObjectiveRecast, Minimizer::num-
    DiscreteIntVars, Minimizer::numDiscreteRealVars, Minimizer::numDiscreteStringVars, Iterator::numFinalSolutions,
Optimizer::numObjectiveFns, Minimizer::objective(), Optimizer::post_run(), Model::primary_response_fn_sense(), Model::primary_response_fn_weights(), Minimizer::resize_best_resp_array(), Minimizer::resize_best_vars_array(), Dakota::set_index_to_value(), COLINOptimizer::solverType, and Model::subordinate_model().

std::pair<bool, bool> colin_cache_lookup ( const colin::AppResponse & colinResponse, Response & tmpResponseHolder ) [protected]
Retrieve response from Colin AppResponse, return pair indicating success for <objective, constraints>
Encapsulated Colin Cache response extraction, which will ultimately become the default lookup. Might want to return separate vectors of function values and constraints for use in the sort, but not for now (least change). Return true if not needed or successful lookup.
References Response::function_value(), Minimizer::numNonlinearConstraints, and Optimizer::numObjectiveFns.
Referenced by COLINOptimizer::post_run().

double constraint_violation ( const Response & tmpResponseHolder ) [protected]
Compute constraint violation, based on nonlinear constraints in iteratedModel and provided Response data.
BMA TODO: incorporate constraint tolerance, possibly via elevating SurrBasedMinimizer::constraint_violation(). Always use iteratedModel to get the constraints; they are in the right space.
References Response::function_values(), Iterator::iteratedModel, Model::nonlinear_eq_constraint_targets(), Model::nonlinear_ineq_constraint_lower_bounds(), Model::nonlinear_ineq_constraint_upper_bounds(), Model::num_nonlinear_eq_constraints(), Model::num_nonlinear_ineq_constraints(), and Minimizer::numIterPrimaryFns.
Referenced by COLINOptimizer::post_run().
The documentation for this class was generated from the following files:

- COLINOptimizer.hpp
- COLINOptimizer.cpp

14.25 COLINTraits Class Reference
A version of TraitsBase specialized for COLIN optimizers.
Inheritance diagram for COLINTraits:

```
                        TraitsBase
                          |
                          v
  COLINTraits
```

Public Member Functions
- COLINTraits ()
  default constructor
- virtual ~COLINTraits ()
  destructor
- virtual bool is_derived ()
  A temporary query used in the refactor.
• bool supports_continuous_variables ()
  Return the flag indicating whether method supports continuous variables.
• bool supports_nonlinear_equality ()
  Return the flag indicating whether method supports nonlinear equalities.
• bool supports_nonlinear_inequality ()
  Return the flag indicating whether method supports nonlinear inequalities.
• NONLINEAR_INEQUALITY_FORMAT nonlinear_inequality_format ()
  Return the format used for nonlinear inequality constraints.

14.25.1 Detailed Description

A version of TraitsBase specialized for COLIN optimizers.

The documentation for this class was generated from the following file:

• COLINOptimizer.hpp

14.26 CollabHybridMetaIterator Class Reference

Meta-iterator for hybrid iteration using multiple collaborating optimization and nonlinear least squares methods. Inheritance diagram for CollabHybridMetaIterator:

```
  Iterator
   |
   MetaIterator
   |
CollabHybridMetaIterator
```

Public Member Functions

• CollabHybridMetaIterator (ProblemDescDB &problem_db)
  standard constructor
• CollabHybridMetaIterator (ProblemDescDB &problem_db, Model &model)
  alternate constructor
• ~CollabHybridMetaIterator ()
  destructor

Protected Member Functions

• void core_run ()
  Performs the collaborative hybrid iteration.
• void derived_init_communicators (ParLevLIter pl_iter)
  derived class contributions to initializing the communicators associated with this Iterator instance
• void derived_set_communicators (ParLevLIter pl_iter)
derived class contributions to setting the communicators associated with this *Iterator* instance

- void derived_free_communicators (ParLevLIter pl_iter)
  
  derived class contributions to freeing the communicators associated with this *Iterator* instance

- IntIntPair estimate_partition_bounds ()
  
  estimate the minimum and maximum partition sizes that can be utilized by this *Iterator*

- const Variables & variables_results () const
  
  return the final solution from the collaborative iteration (variables)

- const Response & response_results () const
  
  return the final solution from the collaborative iteration (response)

Private Attributes

- String hybridCollabType
  
  *abo* or *hops*

- StringArray methodStrings
  
  the list of method pointer or method name identifiers

- StringArray modelStrings
  
  the list of model pointer identifiers for method identification by name

- bool lightwtMethodCtor
  
  use of lightweight *Iterator* construction by name

- bool singlePassedModel
  
  use of constructor that enforces use of a single passed *Model*

- IteratorArray selectedIterators
  
  the set of iterators, one for each entry in methodStrings

- ModelArray selectedModels
  
  the set of models, one for each iterator

- Variables bestVariables
  
  best variables found in collaborative iteration

- Response bestResponse
  
  best response found in collaborative iteration

Additional Inherited Members

14.26.1 Detailed Description

Meta-iterator for hybrid iteration using multiple collaborating optimization and nonlinear least squares methods.

This meta-iterator has two approaches to hybrid iteration: (1) agent-based using the ABO framework; (2) nonagent-based using the HOPSPACK framework.

The documentation for this class was generated from the following files:

- CollabHybridMetaIterator.hpp
- CollabHybridMetaIterator.cpp
14.27 CommandLineHandler Class Reference

Utility class for managing command line inputs to DAKOTA.

Inheritance diagram for CommandLineHandler:

```
CommandLineHandler
      | GetLongOpt
```

Public Member Functions

- **CommandLineHandler ()**
  
  Default constructor, requires `check_usage()` call for parsing

- **CommandLineHandler (int argc, char **argv, int world_rank)**
  
  Constructor with parsing

- **~CommandLineHandler ()**
  
  Destructor

- **void check_usage (int argc, char **argv)**
  
  Verifies that DAKOTA is called with the correct command usage. Prints a descriptive message and exits the program if incorrect.

- **int read_restart_evals () const**
  
  Returns the number of evaluations to be read from the restart file (as specified on the DAKOTA command line) as an integer instead of a const char*.

- **void usage (std::ostream &outfile=Cout) const**
  
  Print usage information to outfile, conditionally on rank.

Private Member Functions

- **void initialize_options ()**
  
  Enrolls the supported command line inputs.

- **void output_helper (const std::string &message, std::ostream &os) const**
  
  Output only on Dakota worldRank 0 if possible

Private Attributes

- **int worldRank**
  
  Rank of this process within Dakota's allocation; manages conditional output.

Additional Inherited Members

14.27.1 Detailed Description

Utility class for managing command line inputs to DAKOTA.

*CommandLineHandler* provides additional functionality that is specific to DAKOTA’s needs for the definition and parsing of command line options. Inheritance is used to allow the class to have all the functionality of the base class, *GetLongOpt*. 
14.28. COMMANDSHELL CLASS REFERENCE

14.27.2 Member Function Documentation

`void output_helper ( const std::string & message, std::ostream & os ) const` [private]

output only on Dakota worldRank 0 if possible

When there is a valid ParallelLibrary, output only on rank 0

References CommandLineHandler::worldRank.

Referenced by CommandLineHandler::check_usage().

The documentation for this class was generated from the following files:

- CommandLineHandler.hpp
- CommandLineHandler.cpp

14.28 CommandShell Class Reference

Utility class which defines convenience operators for spawning processes with system calls.

Public Member Functions

- `CommandShell ()`
  constructor
- `~CommandShell ()`
  destructor
- `CommandShell & operator<< (const char *cmd)`
  appends cmd to sysCommand
- `CommandShell & operator<< (const std::string &cmd)`
  convenient operator: appends string to the commandString to be executed
- `CommandShell & operator<< (CommandShell & (f)(CommandShell &))`
  allows passing of the flush function to the shell using <<
- `CommandShell & flush ()`
  "flushes" the shell; i.e. executes the sysCommand
- `void asynch_flag (const bool flag)`
  set the asynchFlag
- `bool asynch_flag () const`
  get the asynchFlag
- `void suppress_output_flag (const bool flag)`
  set the suppressOutputFlag
- `bool suppress_output_flag () const`
  get the suppressOutputFlag

Private Attributes

- `std::string sysCommand`
  The command string that is constructed through one or more << insertions and then executed by flush.
- `bool asynchFlag`
  flags nonblocking operation (background system calls)
- `bool suppressOutputFlag`
  flags suppression of shell output (no command echo)
14.28.1 Detailed Description

Utility class which defines convenience operators for spawning processes with system calls.

The `CommandShell` class wraps the C system() utility and defines convenience operators for building a command string and then passing it to the shell.

14.28.2 Member Function Documentation

`CommandShell & operator<< (const char * cmd) [inline]`

appends cmd to sysCommand

convenient operator: appends string to the commandString to be executed

References `CommandShell::sysCommand`.

`CommandShell & operator<<(CommandShell &(=)(CommandShell & f)) [inline]`

allows passing of the flush function to the shell using `<<`

convenience operator: allows passing of the flush func to the shell via `<<`

`CommandShell & flush()`

"flushes" the shell; i.e. executes the sysCommand

Executes the sysCommand by passing it to system(). Appends an ";" if asynchFlag is set (background system call) and echoes the sysCommand to Cout if suppressOutputFlag is not set.

References Dakota::abort_handler(), CommandShell::asynchFlag, CommandShell::suppressOutputFlag, and CommandShell::sysCommand.

Referenced by Dakota::flush().

The documentation for this class was generated from the following files:

- `CommandShell.hpp`
- `CommandShell.cpp`

14.29 ConcurrentMetaIterator Class Reference

Meta-iterator for multi-start iteration or pareto set optimization.

Inheritance diagram for ConcurrentMetaIterator:

```
   Iterator
     
   MetaIterator
     
ConcurrentMetaIterator
```
14.29. **CONCURRENTMETA_ITERATOR CLASS REFERENCE**

**Public Member Functions**

- **ConcurrentMetaIterator** (ProblemDescDB &problem_db)
  
  *standard constructor*

- **ConcurrentMetaIterator** (ProblemDescDB &problem_db, Model &model)
  
  *alternate constructor*

- **~ConcurrentMetaIterator** ()
  
  *destructor*

**Protected Member Functions**

- **void pre_run ()**
  
  *pre-run portion of run (optional); re-implemented by Iterators which can generate all Variables (parameter sets) a priori*

- **void core_run ()**
  
  *Performs the concurrent iteration by executing selectedIterator on iteratedModel multiple times in parallel for different parameter sets.*

- **void print_results (std::ostream &s, short results_state=FINAL_RESULTS)**
  
  *print the final iterator results*

- **void derived_init_communicators (ParLevLIter pl_iter)**
  
  *derived class contributions to initializing the communicators associated with this Iterator instance*

- **void derived_set_communicators (ParLevLIter pl_iter)**
  
  *derived class contributions to setting the communicators associated with this Iterator instance*

- **void derived_free_communicators (ParLevLIter pl_iter)**
  
  *derived class contributions to freeing the communicators associated with this Iterator instance*

- **IntIntPair estimate_partition_bounds ()**
  
  *estimate the minimum and maximum partition sizes that can be utilized by this Iterator*

- **void initialize_iterator (int job_index)**
  
  *used by IteratorScheduler to set the starting data for a run*

- **void pack_parameters_buffer (MPIPackBuffer &send_buffer, int job_index)**
  
  *used by IteratorScheduler to pack starting data for an iterator run*

- **void unpack_parameters_initialize (MPIUnpackBuffer &recv_buffer, int job_index)**
  
  *used by IteratorScheduler to unpack starting data and initialize an iterator run*

- **void pack_results_buffer (MPIPackBuffer &send_buffer, int job_index)**
  
  *used by IteratorScheduler to pack results data from an iterator run*

- **void unpack_results_buffer (MPIUnpackBuffer &recv_buffer, int job_index)**
  
  *used by IteratorScheduler to unpack results data from an iterator run*

- **void update_local_results (int job_index)**
  
  *used by IteratorScheduler to update local results arrays*

- **const Model & algorithm_space_model () const**
  
  *return the result of any recasting or surrogate model recursion layered on top of iteratedModel by the derived Iterator ctor chain*

- **virtual void declare_sources ()**
  
  *Declare sources to the evaluations database.*
Private Member Functions

- **void initialize_iterator** (const RealVector &param_set)
  
  called by unpack_parameters.initialize(MPIUnpackBuffer) and initialize_iterator(int) to update iteratedModel and selectedIterator

- **void initialize_model** ()
  
  initialize the iterated Model prior to Iterator instantiation and define param_set_len

Private Attributes

- **Iterator selectedIterator**
  
  the iterator selected for concurrent iteration

- **RealVector initialPt**
  
  the initial continuous variables for restoring the starting point in the Pareto set minimization

- **RealVectorArray parameterSets**
  
  an array of parameter set vectors (either multistart variable sets or pareto multi-objective/least squares weighting sets) to be performed.

- **int paramSetLen**
  
  length of each of the parameter sets associated with an iterator job (number of continuous variables for MULTI_START, number of objective fun for PARETO_SET)

- **int numRandomJobs**
  
  number of randomly-generated parameter sets to evaluate

- **int randomSeed**
  
  seed for random number generator for random samples

- **PRPArray prpResults**
  
  1-d array of ParamResponsePair results corresponding to numIteratorJobs

Friends

- **class IteratorScheduler**
  
  protect scheduler callback functions from general access

Additional Inherited Members

14.29.1 Detailed Description

Meta-iterator for multi-start iteration or pareto set optimization.

This meta-iterator maintains two concurrent iterator capabilities. First, a general capability for running an iterator multiple times from different starting points is provided (often used for multi-start optimization, but not restricted to optimization). Second, a simple capability for mapping the "pareto frontier" (the set of optimal solutions in multiobjective formulations) is provided. This pareto set is mapped through running an optimizer multiple times for different sets of multiobjective weightings.
14.29.2 Member Function Documentation

```cpp
void pre_run() [protected], [virtual]
```

pre-run portion of run (optional); re-implemented by Iterators which can generate all Variables (parameter sets) a priori.

pre-run phase, which a derived iterator may optionally reimplement; when not present, pre-run is likely integrated into the derived run function. This is a virtual function; when re-implementing, a derived class must call its nearest parent’s `pre_run()`, if implemented, typically before performing its own implementation steps.

Reimplemented from Iterator.

References Analyzer::all_samples(), Iterator::all_samples(), ParallelLibrary::bcast_hs(), Model::continuous_lower_bounds(), Model::continuous_upper_bounds(), Model::continuous_variables(), Dakota::copy_data(), Model::estimate_message_lengths(), ConcurrentMetaIterator::initialPt, Iterator::iteratedModel, IteratorScheduler::iteratedModel, IteratorScheduler::iniitialCommRank, IteratorScheduler::iteratorScheduling, IteratorScheduler::iteratorServerId, Model::message_lengths(), Iterator::method_Name, Iterator::methodPCIter, IteratorScheduler::miPLIndex, IteratorScheduler::numIteratorJobs, IteratorScheduler::numIteratorServers, ConcurrentMetaIterator::numRandomJobs, Iterator::parallelLib, ConcurrentMetaIterator::parameterSets, ConcurrentMetaIterator::paramSetLen, ConcurrentMetaIterator::prpResults, ConcurrentMetaIterator::randomSeed, and MPIPackBuffer::size().

```cpp
void print_results(std::ostream & s, short results_state = FINAL_RESULTS) [protected], [virtual]
```

print the final iterator results.

This virtual function provides additional iterator-specific final results outputs beyond the function evaluation summary printed in finalize.run().

Reimplemented from Iterator.

References ResultsManager::active(), ResultsManager::allocate_matrix(), Variables::continuous_variables(), Variables::discrete_int_variables(), Variables::discrete_real_variables(), Variables::discrete_string_variables(), ParamResponsePair::eval_id(), ResultsManager::insert_into(), Iterator::methodName, ConcurrentMetaIterator::parameterSets, ConcurrentMetaIterator::paramSetLen, ConcurrentMetaIterator::prpResults, ParamResponsePair::response(), Iterator::resultsDB, Iterator::run_identifier(), ParamResponsePair::variables(), and Response::write_tabular().

The documentation for this class was generated from the following files:

- ConcurrentMetaIterator.hpp
- ConcurrentMetaIterator.cpp

14.30 CONMINOptimizer Class Reference

Wrapper class for the CONMIN optimization library.

Inheritance diagram for CONMINOptimizer:
Public Member Functions

- **CONMINOptimizer (ProblemDescDB &problem_db, Model &model)**
  
  standard constructor
- **CONMINOptimizer (const String &method_string, Model &model)**
  
  alternate constructor; construct without ProblemDescDB
- **~CONMINOptimizer ()**
  
  destructor
- **void core_run ()**
  
  core portion of run; implemented by all derived classes and may include pre/post steps in lieu of separate pre/post

Protected Member Functions

- **void initialize_run ()**
  
  performs run-time set up

Private Member Functions

- **void initialize ()**
  
  Shared constructor code.
- **void allocate_workspace ()**
  
  Allocates workspace for the optimizer.
- **void deallocate_workspace ()**
  
  Releases workspace memory.
- **void allocate_constraints ()**
  
  Allocates constraint mappings.

Private Attributes

- **int conminInfo**
  
  INFO from CONMIN manual.
- **int printControl**
  
  IPRINT from CONMIN manual (controls output verbosity)
- **Real objFnValue**
  
  value of the objective function passed to CONMIN
- RealVector `constraintValues`
  array of nonlinear constraint values passed to CONMIN
- `int numConminNlnConstr`
  total number of nonlinear constraints seen by CONMIN
- `int numConminLinConstr`
  total number of linear constraints seen by CONMIN
- `int numConminConstr`
  total number of linear and nonlinear constraints seen by CONMIN
- `int N1`
  Size variable for CONMIN arrays. See CONMIN manual.
- `int N2`
  Size variable for CONMIN arrays. See CONMIN manual.
- `int N3`
  Size variable for CONMIN arrays. See CONMIN manual.
- `int N4`
  Size variable for CONMIN arrays. See CONMIN manual.
- `int N5`
  Size variable for CONMIN arrays. See CONMIN manual.
- `int NFDG`
  Finite difference flag.
- `int IPRINT`
  Flag to control amount of output data.
- `int ITMAX`
  Flag to specify the maximum number of iterations.
- `double FDCH`
  Relative finite difference step size.
- `double FDCHM`
  Absolute finite difference step size.
- `double CT`
  Constraint thickness parameter.
- `double CMIN`
  Minimum absolute value of CT used during optimization.
- `double CTL`
  Constraint thickness parameter for linear and side constraints.
- `double CTLMIN`
  Minimum value of CTL used during optimization.
- `double DELFUN`
  Relative convergence criterion threshold.
- `double DABFUN`
  Absolute convergence criterion threshold.
- `double * conminDesVars`
  Array of design variables used by CONMIN (length N1 = numdv+2)
- `double * conminLowerBnds`
Array of lower bounds used by CONMIN (length $N1 = numdv+2$)

- `double * conminLowerBnds`

Array of upper bounds used by CONMIN (length $N1 = numdv+2$)

- `double * S`
  - Internal CONMIN array.

- `double * G1`
  - Internal CONMIN array.

- `double * G2`
  - Internal CONMIN array.

- `double * B`
  - Internal CONMIN array.

- `double * C`
  - Internal CONMIN array.

- `int * MS1`
  - Internal CONMIN array.

- `double * SCAL`
  - Internal CONMIN array.

- `double * DF`
  - Internal CONMIN array.

- `double * A`
  - Internal CONMIN array.

- `int * ISC`
  - Internal CONMIN array.

- `int * IC`
  - Internal CONMIN array.

Additional Inherited Members

14.30.1 Detailed Description

Wrapper class for the CONMIN optimization library.

The `CONMINOptimizer` class provides a wrapper for CONMIN, a Public-domain Fortran 77 optimization library written by Gary Vanderplaats under contract to NASA Ames Research Center. The CONMIN User’s Manual is contained in NASA Technical Memorandum X-62282, 1978. CONMIN uses a reverse communication mode, which avoids the static member function issues that arise with function pointer designs (see NPSOL-Optimizer and SNLLOptimizer).

The user input mappings are as follows: `max_iterations` is mapped into CONMIN’s `ITMAX` parameter, `max_function_evaluations` is implemented directly in the `core_run()` loop since there is no CONMIN parameter equivalent, `convergence_tolerance` is mapped into CONMIN’s `DELFUN` and `DABFUN` parameters, `output_verbosity` is mapped into CONMIN’s `IPRINT` parameter (verbose: `IPRINT = 4`; quiet: `IPRINT = 2`), gradient mode is mapped into CONMIN’s `NFDG` parameter, and finite difference step size is mapped into CONMIN’s `FDCH` and `FDCHM` parameters. Refer to [Vanderplaats, 1978] for additional information on CONMIN parameters.
14.30. **CONMINOPTIMIZER CLASS REFERENCE**

### 14.30.2 Member Function Documentation

```cpp
void core_run() [virtual]
```

Core portion of run; implemented by all derived classes and may include pre/post steps in lieu of separate pre/post

Virtual run function for the iterator class hierarchy. All derived classes need to redefine it.

Reimplemented from **Iterator**.

References `CONMINOptimizer::A, Iterator::activeSet, CONMINOptimizer::B, Iterator::bestResponseArray, Iterator::bestVariablesArray, Minimizer::bigRealBoundSize, CONMINOptimizer::C, CONMINOptimizer::conminDesVars, CONMINOptimizer::conminInfo, CONMINOptimizer::conminLowerBnds, CONMINOptimizer::conminUpperBnds, Optimizer::constraintMapIndices, Optimizer::constraintMapMultipliers, Optimizer::constraintMapOffsets, CONMINOptimizer::constraintValues, Model::continuous_variables(), Dakota::copy_data(), CONMINOptimizer::CT, CONMINOptimizer::CTL, CONMINOptimizer::CTLMIN, CONMINOptimizer::CTMIN, Model::current_response(), CONMINOptimizer::DABFUN, CONMINOptimizer::deallocate_workspace(), CONMINOptimizer::DELFUN, CONMINOptimizer::DF, Model::evaluate(), CONMINOptimizer::FDCH, CONMINOptimizer::FDCHM, Response::function_gradients(), Response::function_values(), CONMINOptimizer::G1, CONMINOptimizer::G2, Model::gradient_type(), CONMINOptimizer::IC, CONMINOptimizer::IPRINT, CONMINOptimizer::IS-C, Iterator::iteratedModel, CONMINOptimizer::ITMAX, Model::linear_eq_constraint_coeffs(), Model::linear_ineq_constraint_coeffs(), Optimizer::localObjectiveRecast, Iterator::maxFunctionEvals, CONMINOptimizer::MS1, CONMINOptimizer::N1, CONMINOptimizer::N2, CONMINOptimizer::N3, CONMINOptimizer::N4, CONMINOptimizer::N5, CONMINOptimizer::NFDG, Model::num_linear_eq_constraints(), Model::num_linear_ineq_constraints(), CONMINOptimizer::numConminConstr, CONMINOptimizer::numConminNlnConstr, Minimizer::numContinuousVars, Minimizer::numFunctions, Optimizer::numObjectiveFns, CONMINOptimizer::objFnValue, Iterator::outputLevel, Model::primary_response_fn_sense(), ActiveSet::request_value(), ActiveSet::request_values(), CONMINOptimizer::S, CONMINOptimizer::SCAL, Minimizer::speculativeFlag, and Minimizer::vendorNumericalGradFlag.

### 14.30.3 Member Data Documentation

```cpp
int conminInfo [private]
```

INFO from CONMIN manual.

Information requested by CONMIN: 1 = evaluate objective and constraints, 2 = evaluate gradients of objective and constraints.

Referenced by `CONMINOptimizer::core_run()` and `CONMINOptimizer::initialize()`.

```cpp
int printControl [private]
```

IPRINT from CONMIN manual (controls output verbosity)

Values range from 0 (nothing) to 4 (most output). 0 = nothing, 1 = initial and final function information, 2 = all of #1 plus function value and design vars at each iteration, 3 = all of #2 plus constraint values and direction vectors, 4 = all of #3 plus gradients of the objective function and constraints, 5 = all of #4 plus proposed design vector, plus objective and constraint functions from the 1-D search

Referenced by `CONMINOptimizer::initialize()`.

```cpp
RealVector constraintValues [private]
```

Array of nonlinear constraint values passed to CONMIN

This array must be of nonzero length and must contain only one-sided inequality constraints which are <= 0 (which requires a transformation from 2-sided inequalities and equalities).

Referenced by `CONMINOptimizer::allocate_workspace()` and `CONMINOptimizer::core_run()`.
int N1 [private]
Size variable for CONMIN arrays. See CONMIN manual.
   N1 = number of variables + 2
   Referenced by CONMINOptimizer::allocate_workspace(), CONMINOptimizer::core_run(), and CONMINOptimizer::initialize_run().

int N2 [private]
Size variable for CONMIN arrays. See CONMIN manual.
   N2 = number of constraints + 2*(number of variables)
   Referenced by CONMINOptimizer::allocate_workspace(), and CONMINOptimizer::core_run().

int N3 [private]
Size variable for CONMIN arrays. See CONMIN manual.
   N3 = Maximum possible number of active constraints.
   Referenced by CONMINOptimizer::allocate_workspace(), and CONMINOptimizer::core_run().

int N4 [private]
Size variable for CONMIN arrays. See CONMIN manual.
   N4 = Maximum(N3,number of variables)
   Referenced by CONMINOptimizer::allocate_workspace(), and CONMINOptimizer::core_run().

int N5 [private]
Size variable for CONMIN arrays. See CONMIN manual.
   N5 = 2*(N4)
   Referenced by CONMINOptimizer::allocate_workspace(), and CONMINOptimizer::core_run().

double CT [private]
Constraint thickness parameter.
   The value of CT decreases in magnitude during optimization.
   Referenced by CONMINOptimizer::core_run(), and CONMINOptimizer::initialize().

double* S [private]
Internal CONMIN array.
   Move direction in N-dimensional space.
   Referenced by CONMINOptimizer::allocate_workspace(), CONMINOptimizer::core_run(), and CONMINOptimizer::deallocate_workspace().

double* G1 [private]
Internal CONMIN array.
   Temporary storage of constraint values.
   Referenced by CONMINOptimizer::allocate_workspace(), CONMINOptimizer::core_run(), and CONMINOptimizer::deallocate_workspace().
**double* G2 [private]**

Internal CONMIN array.
- Temporary storage of constraint values.
- Referenced by CONMINOptimizer::allocate_workspace(), CONMINOptimizer::core_run(), and CONMIN-Optimizer::deallocate_workspace().

**double* B [private]**

Internal CONMIN array.
- Temporary storage for computations involving array S.
- Referenced by CONMINOptimizer::allocate_workspace(), CONMINOptimizer::core_run(), and CONMIN-Optimizer::deallocate_workspace().

**double* C [private]**

Internal CONMIN array.
- Temporary storage for use with arrays B and S.
- Referenced by CONMINOptimizer::allocate_workspace(), CONMINOptimizer::core_run(), and CONMIN-Optimizer::deallocate_workspace().

**int* MS1 [private]**

Internal CONMIN array.
- Temporary storage for use with arrays B and S.
- Referenced by CONMINOptimizer::allocate_workspace(), CONMINOptimizer::core_run(), and CONMIN-Optimizer::deallocate_workspace().

**double* SCAL [private]**

Internal CONMIN array.
- Vector of scaling parameters for design parameter values.
- Referenced by CONMINOptimizer::allocate_workspace(), CONMINOptimizer::core_run(), and CONMIN-Optimizer::deallocate_workspace().

**double* DF [private]**

Internal CONMIN array.
- Temporary storage for analytic gradient data.
- Referenced by CONMINOptimizer::allocate_workspace(), CONMINOptimizer::core_run(), and CONMIN-Optimizer::deallocate_workspace().

**double* A [private]**

Internal CONMIN array.
- Temporary 2-D array for storage of constraint gradients.
- Referenced by CONMINOptimizer::allocate_workspace(), CONMINOptimizer::core_run(), and CONMIN-Optimizer::deallocate_workspace().
```plaintext
int* ISC [private]

Internal CONMIN array.
Array of flags to identify linear constraints. (not used in this implementation of CONMIN)
Referenced by CONMINOptimizer::allocate_workspace(), CONMINOptimizer::core_run(), CONMINOptimizer-:
deallocate_workspace(), and CONMINOptimizer::initialize_run().

int* IC [private]

Internal CONMIN array.
Array of flags to identify active and violated constraints
Referenced by CONMINOptimizer::allocate_workspace(), CONMINOptimizer::core_run(), CONMINOptimizer-:
deallocate_workspace(), and CONMINOptimizer::initialize_run().
The documentation for this class was generated from the following files:
• CONMINOptimizer.hpp
• CONMINOptimizer.cpp

14.31 CONMINTraits Class Reference

A version of TraitsBase specialized for CONMIN optimizers.
Inheritance diagram for CONMINTraits:

```
```

Public Member Functions

- CONMINTraits ()
  default constructor
- virtual ~CONMINTraits ()
  destructor
- virtual bool isDerived ()
  A temporary query used in the refactor.
- bool supportsContinuousVariables ()
  Return the flag indicating whether method supports continuous variables.
- bool supportsLinearEquality ()
  Return the flag indicating whether method supports linear equalities.
- bool supportsLinearInequality ()
  Return the flag indicating whether method supports linear inequalities.
- LINEAR_INEQUALITY_FORMAT linearInequalityFormat ()
  Return the format used for linear inequality constraints.
- bool supportsNonlinearEquality ()
  Return the flag indicating whether method supports nonlinear equalities.
```
14.32. CONOLEREDIRECTOR CLASS REFERENCE

- bool supports_nonlinear_inequality ()
  Return the flag indicating whether method supports nonlinear inequalities.
- NONLINEAR_INEQUALITY_FORMAT nonlinear_inequality_format ()
  Return the format used for nonlinear inequality constraints.

14.31.1 Detailed Description

A version of TraitsBase specialized for CONMIN optimizers.

The documentation for this class was generated from the following file:
- CONMINOptimizer.hpp

14.32 ConsoleRedirector Class Reference

Public Member Functions
- ConsoleRedirector (std::ostream *&dakota_stream, std::ostream *default_dest)
  Constructor taking a reference to the Dakota Cout/Cerr handle and a default destination to use when no redirection (or destruct)
- ~ConsoleRedirector ()
  when the redirector stack is destroyed, it will rebind the output handle to the default ostream, then destroy open files
- void push_back ()
  push back the default or repeat the last pushed file stream
- void push_back (const String &filename)
  push back a new output filestream, or repeat the last one if no filename change
- void pop_back ()
  pop the last redirection

Protected Attributes
- std::ostream *& oststreamHandle
  The handle (target ostream) through which output is sent; typically dakota_cout or dakota_cerr. Will be rebound to specific streams as they are pushed or popped.
- std::ostream * defaultOStream
  initial stream to reset to when redirections are done (typically std::cout or std::cerr)
- std::vector<std::shared_ptr<OutputWriter>> ostreamDestinations
  stack of redirections to OutputWriters; shared pointers are used to potentially share the same ostream at multiple levels

Private Member Functions
- ConsoleRedirector ()
  default constructor is disallowed
- ConsoleRedirector (const ConsoleRedirector &)
  copy constructor is disallowed due
- const ConsoleRedirector & operator= (const ConsoleRedirector &)
  assignment is disallowed
14.32.1 Detailed Description

Component to manage a set of output or error redirections. Push operations may present a new filename, or none in order to preserve current binding to cout/cerr or file, but place an entry on the stack. Cout/Cerr are rebound as needed when a stream is destroyed on pop.

The documentation for this class was generated from the following files:

- OutputManager.hpp
- OutputManager.cpp

14.33 Constraints Class Reference

Base class for the variable constraints class hierarchy.

Inheritance diagram for Constraints:

```
Constraints
   \--- MixedVarConstraints
       \--- RelaxedVarConstraints
```

Public Member Functions

- **Constraints ()**
  - default constructor

- **Constraints (const ProblemDescDB &prob_db, const SharedVariablesData &svd)**
  - standard constructor

- **Constraints (const SharedVariablesData &svd)**
  - alternate constructor for instantiations on the fly

- **Constraints (const Constraints &con)**
  - copy constructor

- **virtual ~Constraints ()**
  - destructor

- **Constraints operator= (const Constraints &con)**
  - assignment operator

- **virtual void write (std::ostream &s) const**
  - write a variable constraints object to an std::ostream

- **virtual void read (std::istream &s)**
  - read a variable constraints object from an std::istream

- **const RealVector & continuous_lower_bounds () const**
  - return the active continuous variable lower bounds

- **Real continuous_lower_bound (size_t i) const**
  - return an active continuous variable lower bound

- **void continuous_lower_bounds (const RealVector &cl_bnds)**
  - set the active continuous variable lower bounds
14.33. CONSTRAINTS CLASS REFERENCE

- void **continuous_lower_bound** (Real cl_bnd, size_t i)
  set an active continuous variable lower bound
- const RealVector & **continuous_upper_bounds** () const
  return the active continuous variable upper bounds
- Real **continuous_upper_bound** (size_t i) const
  return an active continuous variable upper bound
- void **continuous_upper_bounds** (const RealVector &cu_bnds)
  set the active continuous variable upper bounds
- void **continuous_upper_bound** (Real cu_bnd, size_t i)
  set an active continuous variable upper bound
- const IntVector & **discrete_int_lower_bounds** () const
  return the active discrete variable lower bounds
- int **discrete_int_lower_bound** (size_t i) const
  return an active discrete variable lower bound
- void **discrete_int_lower_bounds** (const IntVector &dil_bnds)
  set the active discrete variable lower bounds
- void **discrete_int_lower_bound** (int dil_bnd, size_t i)
  set an active discrete variable lower bound
- const IntVector & **discrete_int_upper_bounds** () const
  return the active discrete variable upper bounds
- int **discrete_int_upper_bound** (size_t i) const
  return an active discrete variable upper bound
- void **discrete_int_upper_bounds** (const IntVector &diu_bnds)
  set the active discrete variable upper bounds
- void **discrete_int_upper_bound** (int diu_bnd, size_t i)
  set an active discrete variable upper bound
- const RealVector & **discrete_real_lower_bounds** () const
  return the active discrete variable lower bounds
- Real **discrete_real_lower_bound** (size_t i) const
  return an active discrete variable lower bound
- void **discrete_real_lower_bounds** (const RealVector &drl_bnds)
  set the active discrete variable lower bounds
- void **discrete_real_lower_bound** (Real drl_bnd, size_t i)
  set an active discrete variable lower bound
- const RealVector & **discrete_real_upper_bounds** () const
  return the active discrete variable upper bounds
- Real **discrete_real_upper_bound** (size_t i) const
  return an active discrete variable upper bound
- void **discrete_real_upper_bounds** (const RealVector &dru_bnds)
  set the active discrete variable upper bounds
- void **discrete_real_upper_bound** (Real dru_bnd, size_t i)
  set an active discrete variable upper bound
- const RealVector & **inactive_continuous_lower_bounds** () const
  set the inactive continuous variable lower bounds
return the inactive continuous lower bounds

- void inactive_continuous_lower_bounds (const RealVector &icl_bnds)
  set the inactive continuous lower bounds

- const RealVector & inactive_continuous_upper_bounds () const
  return the inactive continuous upper bounds

- void inactive_continuous_upper_bounds (const RealVector &icu_bnds)
  set the inactive continuous upper bounds

- const IntVector & inactive_discrete_int_lower_bounds () const
  return the inactive discrete lower bounds

- void inactive_discrete_int_lower_bounds (const IntVector &idil_bnds)
  set the inactive discrete lower bounds

- const IntVector & inactive_discrete_int_upper_bounds () const
  return the inactive discrete upper bounds

- void inactive_discrete_int_upper_bounds (const IntVector &idiu_bnds)
  set the inactive discrete upper bounds

- const RealVector & inactive_discrete_real_lower_bounds () const
  return the inactive discrete lower bounds

- void inactive_discrete_real_lower_bounds (const RealVector &idrl_bnds)
  set the inactive discrete lower bounds

- const RealVector & inactive_discrete_real_upper_bounds () const
  return the inactive discrete upper bounds

- void inactive_discrete_real_upper_bounds (const RealVector &idru_bnds)
  set the inactive discrete upper bounds

- const RealVector & all_continuous_lower_bounds () const
  returns a single array with all continuous lower bounds

- void all_continuous_lower_bounds (const RealVector &acl_bnds)
  sets all continuous lower bounds using a single array

- void all_continuous_lower_bound (Real acl_bnd, size_t i)
  set a lower bound within the all continuous lower bounds array

- const RealVector & all_continuous_upper_bounds () const
  returns a single array with all continuous upper bounds

- void all_continuous_upper_bounds (const RealVector &acu_bnds)
  sets all continuous upper bounds using a single array

- void all_continuous_upper_bound (Real acu_bnd, size_t i)
  set an upper bound within the all continuous upper bounds array

- const IntVector & all_discrete_int_lower_bounds () const
  returns a single array with all discrete lower bounds

- void all_discrete_int_lower_bounds (const IntVector &adil_bnds)
  sets all discrete lower bounds using a single array

- void all_discrete_int_lower_bound (int adil_bnd, size_t i)
  set a lower bound within the all discrete lower bounds array

- const IntVector & all_discrete_int_upper_bounds () const
  returns a single array with all discrete upper bounds

- void all_discrete_int_upper_bounds (const IntVector &adil_bnds)
  sets all discrete upper bounds using a single array

- void all_discrete_int_upper_bound (int adil_bnd, size_t i)
  set an upper bound within the all discrete upper bounds array
• void all_discrete_int_upper_bounds (const IntVector &adiu_bnds)
  sets all discrete upper bounds using a single array
• void all_discrete_int_upper_bound (int adiu_bnd, size_t i)
  set an upper bound within the all discrete upper bounds array
• const RealVector & all_discrete_real_lower_bounds () const
  returns a single array with all discrete lower bounds
• void all_discrete_real_lower_bounds (const RealVector &adrl_bnds)
  sets all discrete lower bounds using a single array
• void all_discrete_real_lower_bound (Real adrl_bnd, size_t i)
  set a lower bound within the all discrete lower bounds array
• const RealVector & all_discrete_real_upper_bounds () const
  returns a single array with all discrete upper bounds
• void all_discrete_real_upper_bounds (const RealVector &adru_bnds)
  sets all discrete upper bounds using a single array
• void all_discrete_real_upper_bound (Real adru_bnd, size_t i)
  set an upper bound within the all discrete upper bounds array
• size_t num_linear_ineq_constraints () const
  return the number of linear inequality constraints
• size_t num_linear_eq_constraints () const
  return the number of linear equality constraints
• const RealMatrix & linear_ineq_constraint_coeffs () const
  return the linear inequality constraint coefficients
• void linear_ineq_constraint_coeffs (const RealMatrix &lin_ineq_coeffs)
  set the linear inequality constraint coefficients
• const RealVector & linear_ineq_constraint_lower_bounds () const
  return the linear inequality constraint lower bounds
• void linear_ineq_constraint_lower_bounds (const RealVector &lin_ineq_l_bnds)
  set the linear inequality constraint lower bounds
• const RealVector & linear_ineq_constraint_upper_bounds () const
  return the linear inequality constraint upper bounds
• void linear_ineq_constraint_upper_bounds (const RealVector &lin_ineq_u_bnds)
  set the linear inequality constraint upper bounds
• const RealMatrix & linear_eq_constraint_coeffs () const
  return the linear equality constraint coefficients
• void linear_eq_constraint_coeffs (const RealMatrix &lin_eq_coeffs)
  set the linear equality constraint coefficients
• const RealVector & linear_eq_constraint_targets () const
  return the linear equality constraint targets
• void linear_eq_constraint_targets (const RealVector &lin_eq_targets)
  set the linear equality constraint targets
• size_t num_nonlinear_ineq_constraints () const
  return the number of nonlinear inequality constraints
• size_t num_nonlinear_eq_constraints () const
return the number of nonlinear equality constraints

- const RealVector & nonlinear_ineq_constraint_lower_bounds () const
  return the nonlinear inequality constraint lower bounds

- void nonlinear_ineq_constraint_lower_bounds (const RealVector & nln_ineq_l_bnds)
  set the nonlinear inequality constraint lower bounds

- const RealVector & nonlinear_ineq_constraint_upper_bounds () const
  return the nonlinear inequality constraint upper bounds

- void nonlinear_ineq_constraint_upper_bounds (const RealVector & nln_ineq_u_bnds)
  set the nonlinear inequality constraint upper bounds

- const RealVector & nonlinear_eq_constraint_targets () const
  return the nonlinear equality constraint targets

- void nonlinear_eq_constraint_targets (const RealVector & nln_eq_targets)
  set the nonlinear equality constraint targets

- Constraints copy () const
  for use when a deep copy is needed (the representation is not shared)

- void shape ()
  shape the lower/upper bound arrays based on sharedVarsData

- void reshape (size_t num_nln_ineq_cons, size_t num_nln_eq_cons, size_t num_lin_ineq_cons, size_t num_lin_eq_cons, const SharedVariablesData & svd)
  reshape the linear/nonlinear/bound constraint arrays and the lower/upper bound arrays

- void reshape ()
  reshape the lower/upper bound arrays based on sharedVarsData

- void reshape (size_t num_nln_ineq_cons, size_t num_nln_eq_cons, size_t num_nln_ineq_cons, size_t num_lin_ineq_cons)
  reshape the linear/nonlinear constraint arrays

- void inactive_view (short view2)
  sets the inactive view based on higher level (nested) context

- bool is_null () const
  function to check constraintsRep (does this envelope contain a letter)

Protected Member Functions

- Constraints (BaseConstructor, const ProblemDescDB & problem_db, const SharedVariablesData & svd)
  constructor initializes the base class part of letter classes (BaseConstructor overloading avoids infinite recursion in the derived class constructors - Coplien, p. 139)

- Constraints (BaseConstructor, const SharedVariablesData & svd)
  constructor initializes the base class part of letter classes (BaseConstructor overloading avoids infinite recursion in the derived class constructors - Coplien, p. 139)

- void build_views ()
  construct active/inactive views of all variables arrays

- void build_active_views ()
  construct active views of all variables bounds arrays

- void build_inactive_views ()
  construct inactive views of all variables bounds arrays

- void manage_linear_constraints (const ProblemDescDB & problem_db)
  perform checks on user input, convert linear constraint coefficient input to matrices, and assign defaults
Protected Attributes

- `SharedVariablesData sharedVarsData`
  - configuration data shared from a `Variables` instance
- `RealVector allContinuousLowerBnds`
  - a continuous lower bounds array combining continuous design, uncertain, and continuous state variable types (all view).
- `RealVector allContinuousUpperBnds`
  - a continuous upper bounds array combining continuous design, uncertain, and continuous state variable types (all view).
- `IntVector allDiscreteIntLowerBnds`
  - a discrete lower bounds array combining discrete design and discrete state variable types (all view).
- `IntVector allDiscreteIntUpperBnds`
  - a discrete upper bounds array combining discrete design and discrete state variable types (all view).
- `RealVector allDiscreteRealLowerBnds`
  - a discrete lower bounds array combining discrete design and discrete state variable types (all view).
- `RealVector allDiscreteRealUpperBnds`
  - a discrete upper bounds array combining discrete design and discrete state variable types (all view).
- `size_t numNonlinearIneqCons`
  - number of nonlinear inequality constraints
- `size_t numNonlinearEqCons`
  - number of nonlinear equality constraints
- `RealVector nonlinearIneqConLowerBnds`
  - nonlinear inequality constraint lower bounds
- `RealVector nonlinearIneqConUpperBnds`
  - nonlinear inequality constraint upper bounds
- `RealVector nonlinearEqConTargets`
  - nonlinear equality constraint targets
- `size_t numLinearIneqCons`
  - number of linear inequality constraints
- `size_t numLinearEqCons`
  - number of linear equality constraints
- `RealMatrix linearIneqConCoefs`
  - linear inequality constraint coefficients
- `RealMatrix linearEqConCoefs`
  - linear equality constraint coefficients
- `RealVector linearIneqConLowerBnds`
  - linear inequality constraint lower bounds
- `RealVector linearIneqConUpperBnds`
  - linear inequality constraint upper bounds
- `RealVector linearEqConTargets`
  - linear equality constraint targets
- `RealVector continuousLowerBnds`
  - the active continuous lower bounds array view
CHAPTER 14. CLASS DOCUMENTATION

- RealVector continuousUpperBnds
  the active continuous upper bounds array view
- IntVector discreteIntLowerBnds
  the active discrete lower bounds array view
- IntVector discreteIntUpperBnds
  the active discrete upper bounds array view
- RealVector discreteRealLowerBnds
  the active discrete lower bounds array view
- RealVector discreteRealUpperBnds
  the active discrete upper bounds array view
- RealVector inactiveContinuousLowerBnds
  the inactive continuous lower bounds array view
- RealVector inactiveContinuousUpperBnds
  the inactive continuous upper bounds array view
- IntVector inactiveDiscreteIntLowerBnds
  the inactive discrete lower bounds array view
- IntVector inactiveDiscreteIntUpperBnds
  the inactive discrete upper bounds array view
- RealVector inactiveDiscreteRealLowerBnds
  the inactive discrete lower bounds array view
- RealVector inactiveDiscreteRealUpperBnds
  the inactive discrete upper bounds array view

Private Member Functions

- Constraints * get_constraints (const ProblemDescDB &problem_db, const SharedVariablesData &svd)
  Used only by the constructor to initialize constraintsRep to the appropriate derived type.
- Constraints * get_constraints (const SharedVariablesData &svd) const
  Used by copy() to initialize constraintsRep to the appropriate derived type.

Private Attributes

- Constraints * constraintsRep
  pointer to the letter (initialized only for the envelope)
- int referenceCount
  number of objects sharing constraintsRep

14.33.1 Detailed Description

Base class for the variable constraints class hierarchy.

The Constraints class is the base class for the class hierarchy managing bound, linear, and nonlinear constraints. Using the variable lower and upper bounds arrays from the input specification, different derived classes define different views of this data. The linear and nonlinear constraint data is consistent in all views and is managed at the base class level. For memory efficiency and enhanced polymorphism, the variable constraints hierarchy employs the "letter/envelope idiom" (see Coplien "Advanced C++", p. 133), for which the base class (Constraints) serves as the envelope and one of the derived classes (selected in Constraints::get_constraints()) serves as the letter.
14.33. CONSTRAINTS CLASS REFERENCE

14.33.2 Constructor & Destructor Documentation

**Constraints ( )**

default constructor

The default constructor: constraintsRep is NULL in this case (a populated problem_db is needed to build a meaningful Constraints object). This makes it necessary to check for NULL in the copy constructor, assignment operator, and destructor.

**Constraints ( const ProblemDescDB & problem_db, const SharedVariablesData & svd )**

standard constructor

The envelope constructor only needs to extract enough data to properly execute get_constraints, since the constructor overloaded with BaseConstructor builds the actual base class data inherited by the derived classes.

References Dakota::abort_handler(), Constraints::constraintsRep, and Constraints::get_constraints().

**Constraints ( const SharedVariablesData & svd )**

alternate constructor for instantiations on the fly

Envelope constructor for instantiations on the fly. This constructor executes get_constraints(view), which invokes the default derived/base constructors, followed by a reshape() based on vars_comps.

References Dakota::abort_handler(), Constraints::constraintsRep, and Constraints::get_constraints().

**Constraints ( const Constraints & con )**

copy constructor

Copy constructor manages sharing of constraintsRep and incrementing of referenceCount.

References Constraints::constraintsRep, and Constraints::referenceCount.

**Constraints ( BaseConstructor, const ProblemDescDB & problem_db, const SharedVariablesData & svd ) [protected]**

constructor initializes the base class part of letter classes (BaseConstructor overloading avoids infinite recursion in the derived class constructors - Coplien, p. 139)

This constructor is the one which must build the base class data for all derived classes. get_constraints() instantiates a derived class letter and the derived constructor selects this base class constructor in its initialization list (to avoid recursion in the base class constructor calling get_constraints() again). Since the letter IS the representation, its rep pointer is set to NULL (an uninitialized pointer causes problems in ~Constraints).

References Constraints::build_views(), Constraints::manage_linear_constraints(), and Constraints::shape().

**Constraints ( BaseConstructor, const SharedVariablesData & svd ) [protected]**

constructor initializes the base class part of letter classes (BaseConstructor overloading avoids infinite recursion in the derived class constructors - Coplien, p. 139)

This constructor is the one which must build the base class data for all derived classes. get_constraints() instantiates a derived class letter and the derived constructor selects this base class constructor in its initialization list (to
avoid recursion in the base class constructor calling `get_constraints()` again). Since the letter IS the representation, its rep pointer is set to NULL (an uninitialized pointer causes problems in ~Constraints).

References Constraints::build_views(), and Constraints::shape().

### 14.33.3 Member Function Documentation

**Constraints operator= ( const Constraints & con )**

assignment operator


References Constraints::constraintsRep, and Constraints::referenceCount.

**Constraints copy ( ) const**

for use when a deep copy is needed (the representation is not shared)

Deep copies are used for history mechanisms that catalogue permanent copies (should not change as the representation within userDefinedConstraints changes).

References Constraints::allContinuousLowerBnds, Constraints::allContinuousUpperBnds, Constraints::allDiscreteIntLowerBnds, Constraints::allDiscreteIntUpperBnds, Constraints::allDiscreteRealLowerBnds, Constraints::allDiscreteRealUpperBnds, Constraints::build_views(), Constraints::constraintsRep, Constraints::get_constraints(), Constraints::linearEqConCoeffs, Constraints::linearEqConTargets, Constraints::linearIneqConCoeffs, Constraints::linearIneqConLowerBnds, Constraints::linearIneqConUpperBnds, Constraints::nonlinearEqConCoeffs, Constraints::nonlinearEqConTargets, Constraints::nonlinearIneqConCoeffs, Constraints::nonlinearIneqConLowerBnds, Constraints::nonlinearIneqConUpperBnds, Constraints::numLinearEqCons, Constraints::numNonlinearEqCons, Constraints::numNonlinearIneqCons, and Constraints::sharedVarsData.

Referenced by SurrogateModel::force_rebuild(), and RecastModel::init_constraints().

**void shape ( )**

shape the lower/upper bound arrays based on sharedVarsData

Resizes the derived bounds arrays.

References SharedVariablesData::all_counts(), Constraints::allContinuousLowerBnds, Constraints::allContinuousUpperBnds, Constraints::allDiscreteIntLowerBnds, Constraints::allDiscreteIntUpperBnds, Constraints::allDiscreteRealLowerBnds, Constraints::allDiscreteRealUpperBnds, Constraints::constraintsRep, Constraints::shape(), and Constraints::sharedVarsData.

Referenced by Constraints::Constraints(), and Constraints::shape().

**void reshape ( size_t num_nln_ineq_cons, size_t num_nln_eq_cons, size_t num_lin_ineq_cons, size_t num_lin_eq_cons )**

reshape the linear/nonlinear constraint arrays

Resizes the linear and nonlinear constraint arrays at the base class. Does NOT currently resize the derived bounds arrays.

References Constraints::constraintsRep, Constraints::continuousLowerBnds, Constraints::discreteIntLowerBnds, Constraints::discreteRealLowerBnds, Constraints::linearEqConCoeffs, Constraints::linearEqConTargets, Constraints::linearIneqConCoeffs, Constraints::linearIneqConLowerBnds, Constraints::linearIneqConUpperBnds, Constraints::nonlinearEqConCoeffs, Constraints::nonlinearEqConLowerBnds, Constraints::nonlinearEqConTargets, Constraints::nonlinearIneqConLowerBnds, Constraints::nonlinearIneqConUpperBnds, Constraints::numLinearEqCons, Constraints::numLinearIneqCons, Constraints::numNonlinearEqCons, Constraints::numNonlinearIneqCons, and Constraints::reshape().
void manage_linear_constraints ( const ProblemDescDB & problem_db ) [protected]
perform checks on user input, convert linear constraint coefficient input to matrices, and assign defaults

Convenience function called from derived class constructors. The number of variables active for applying linear constraints is currently defined to be the number of active continuous variables plus the number of active discrete variables (the most general case), even though very few optimizers can currently support mixed variable linear constraints.

References Dakota::abort_handler(), Constraints::continuousLowerBnds, Dakota::copy_data(), Constraints::discreteIntLowerBnds, Constraints::discreteRealLowerBnds, ProblemDescDB::get_rv(), Constraints::linearEqConCoeffs, Constraints::linearEqConTargets, Constraints::linearIneqConCoeffs, Constraints::linearIneqConLowerBnds, Constraints::linearIneqConUpperBnds, Constraints::numLinearEqCons, and Constraints::numLinearIneqCons.

Referenced by Constraints::Constraints().

Constraints * get_constraints ( const ProblemDescDB & problem_db, const SharedVariablesData & svd ) [private]
Used only by the constructor to initialize constraintsRep to the appropriate derived type.

Initializes constraintsRep to the appropriate derived type, as given by the variables view.

References SharedVariablesData::view().

Referenced by Constraints::Constraints(), and Constraints::copy().

Constraints * get_constraints ( const SharedVariablesData & svd ) const [private]
Used by copy() to initialize constraintsRep to the appropriate derived type.

Initializes constraintsRep to the appropriate derived type, as given by the variables view. The default derived class constructors are invoked.

References SharedVariablesData::view().

The documentation for this class was generated from the following files:

- DakotaConstraints.hpp
- DakotaConstraints.cpp

14.34 DakotaROLEqConstraints Class Reference

Inheritance diagram for DakotaROLEqConstraints:

```
DakotaROLEqConstraints
```

```
DakotaROLEqConstraintsGrad
```

```
DakotaROLEqConstraintsHess
```

Public Member Functions

- DakotaROLEqConstraints (Model &model)
  Constructor.
void value (std::vector<Real> &c, const std::vector<Real> &x, Real &tol) override

Function to return the constraint value to ROL.

Protected Attributes

- Model & dakotaModel
  Dakota problem data provided by user.
- bool haveNlnConst
  Whether or not problem has nonlinear equality constraints.

14.34.1 Detailed Description

DakotaROLEqConstraints is derived from the ROL constraint class. It overrides the member functions to provide Dakota-specific implementations of equality constraint evaluation and the application of the equality constraint Jacobian to a vector.

14.34.2 Constructor & Destructor Documentation

DakotaROLEqConstraints (Model &model)

Constructor.

Implementation of the DakotaROLEqConstraints class.
References DakotaROLEqConstraints::haveNlnConst, and Model::num_nonlinear_eq_constraints().

The documentation for this class was generated from the following files:

- ROLOptimizer.hpp
- ROLOptimizer.cpp

14.35 DakotaROLEqConstraintsGrad Class Reference

Inheritance diagram for DakotaROLEqConstraintsGrad:

Public Member Functions

- DakotaROLEqConstraintsGrad (Model &model)
  Constructor.
- virtual ~DakotaROLEqConstraintsGrad ()
  Destructor.
• void **applyJacobian** (std::vector< Real > &jv, const std::vector< Real > &v, const std::vector< Real > &x, Real &tol) override

  *Function to return the result of applying the constraint gradient to an arbitrary vector to ROL.*

• void **applyAdjointJacobian** (std::vector< Real > &ajv, const std::vector< Real > &v, const std::vector< Real > &x, Real &tol) override

  *Function to return the result of applying the constraint adjoint to an arbitrary vector to ROL.*

**Additional Inherited Members**

**14.35.1 Detailed Description**

DakotaROLEqConstraintsGrad is derived from DakotaROLEqConstraints. It implements overrides of ROL member functions to provide a Dakota-specific application of the inequality constraint Jacobian to a vector. This separate class is needed to allow for the option of utilizing ROL’s finite-differenced gradients.

**14.35.2 Constructor & Destructor Documentation**

DakotaROLEqConstraintsGrad ( Model & model )

Constructor.

Implementation of the DakotaROLEqConstraintsGrad class.

The documentation for this class was generated from the following files:

- ROLOptimizer.hpp
- ROLOptimizer.cpp

**14.36 DakotaROLEqConstraintsHess Class Reference**

Inheritance diagram for DakotaROLEqConstraintsHess:

```
DakotaROLEqConstraintsHess
<table>
<thead>
<tr>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
</tr>
<tr>
<td>DakotaROLEqConstraintsGrad</td>
</tr>
<tr>
<td>----------------</td>
</tr>
<tr>
<td>DakotaROLEqConstraints</td>
</tr>
</tbody>
</table>
```

**Public Member Functions**

- **DakotaROLEqConstraintsHess (Model &model)**

  *Constructor.*

- virtual ~DakotaROLEqConstraintsHess ()

  *Destructor.*

- void **applyAdjointHessian** (std::vector< Real > &ahuv, const std::vector< Real > &u, const std::vector< Real > &v, const std::vector< Real > &x, Real &tol) override

  *Function to return the result of applying the constraint adjoint Hessian to an arbitrary vector to ROL.*
Additional Inherited Members

14.36.1 Detailed Description

DakotaROLEqConstraintsHess is derived from DakotaROLEqConstraintsGrad. It implements overrides of ROL member functions to provide a Dakota-specific implementation of an adjoint Hessian-vector product for equality constraints. This separate class is needed (rather than putting the product into DakotaROLEqConstraints) because logic in ROL does not always protect against calling the adjoint Hessian-vector product in cases where there is not actually a Hessian provided.

14.36.2 Constructor & Destructor Documentation

DakotaROLEqConstraintsHess (Model & model)

Constructor.

Implementation of the DakotaROLEqConstraintsHess class.

The documentation for this class was generated from the following files:

- ROLOptimizer.hpp
- ROLOptimizer.cpp

14.37 DakotaROLIneqConstraints Class Reference

Inheritance diagram for DakotaROLIneqConstraints:

```
DakotaROLIneqConstraints
\downarrow
DakotaROLIneqConstraintsGrad
\downarrow
DakotaROLIneqConstraintsHess
```

Public Member Functions

- DakotaROLIneqConstraints (Model &model)
  
  Constructor.

- void value (std::vector<Real> &c, const std::vector<Real> &x, Real &tol) override

  Function to return the constraint value to ROL.

Protected Attributes

- Model & dakotaModel

  Dakota problem data provided by user.

- bool haveNlnConst

  Whether or not problem has nonlinear inequality constraints.
14.38 DakotaROLIneqConstraintsGrad Class Reference

**Public Member Functions**

- **DakotaROLIneqConstraintsGrad (Model &model)**
  *Constructor.*

- **virtual ~DakotaROLIneqConstraintsGrad ()**
  *Destructor.*

- **void applyJacobian (std::vector<Real> &jv, const std::vector<Real> &v, const std::vector<Real> &x, Real &tol) override**
  *Function to return the result of applying the constraint gradient on an arbitrary vector to ROL.*

- **void applyAdjointJacobian (std::vector<Real> &ajv, const std::vector<Real> &v, const std::vector<Real> &x, Real &tol) override**
  *Function to return the result of applying the constraint adjoint to an arbitrary vector to ROL.*

**Additional Inherited Members**

14.38.1 Detailed Description

DakotaROLIneqConstraintsGrad is derived from DakotaROLIneqConstraints. It implements overrides of ROL member functions to provide a Dakota-specific application of the inequality constraint Jacobian to a vector. This separate class is needed to allow for the option of utilizing ROL’s finite-differenced gradients.
### 14.38.2 Constructor & Destructor Documentation

DakotaROLIneqConstraintsGrad ( Model & model )

Constructor.  
Implementation of the DakotaROLIneqConstraintsGrad class.  
The documentation for this class was generated from the following files:
- ROLOptimizer.hpp
- ROLOptimizer.cpp

### 14.39 DakotaROLIneqConstraintsHess Class Reference

Inheritance diagram for DakotaROLIneqConstraintsHess:

```
DakotaROLIneqConstraints
    |
    v
DakotaROLIneqConstraintsGrad
    |
    v
DakotaROLIneqConstraintsHess
```

**Public Member Functions**

- DakotaROLIneqConstraintsHess (Model &model)  
  
  Constructor.
- virtual ~DakotaROLIneqConstraintsHess ()  
  
  Destructor.
- void applyAdjointHessian (std::vector< Real > &ahuv, const std::vector< Real > &u, const std::vector< Real > &v, const std::vector< Real > &x, Real &tol) override  
  
  Function to return the result of applying the constraint adjoint Hessian to an arbitrary vector to ROL.

**Additional Inherited Members**

### 14.39.1 Detailed Description

DakotaROLIneqConstraintsHess is derived from DakotaROLIneqConstraintsGrad. It implements overrides of ROL member functions to provide a Dakota-specific implementation of a adjoint Hessian-vector product for inequality constraints. This separate class is needed (rather than putting the product into DakotaROLIneqConstraints) because logic in ROL does not always protect against calling the adjoint Hessian-vector product in cases where there is not actually a Hessian provided.

### 14.39.2 Constructor & Destructor Documentation

DakotaROLIneqConstraintsHess ( Model & model )

Constructor.  
Implementation of the DakotaROLIneqConstraintsGrad class.  
The documentation for this class was generated from the following files:
14.40 DakotaROLObjective Class Reference

Inheritance diagram for DakotaROLObjective:

```
DakotaROLObjective
  └── DakotaROLObjectiveGrad
        └── DakotaROLObjectiveHess
```

Public Member Functions

- **DakotaROLObjective (Model &model)**
  
  Constructor.

- **Real value (const std::vector<Real> &x, Real &tol) override**
  
  Function to return the objective value (response) to ROL.

Public Attributes

- **Model & dakotaModel**
  
  Dakota problem data provided by user.

14.40.1 Detailed Description

DakotaROLObjective is derived from the ROL objective class. It overrides the member functions to provide Dakota-specific implementations of function evaluations.

14.40.2 Constructor & Destructor Documentation

DakotaROLObjective ( Model & model )

Constructor.

Implementation of the DakotaROLObjective class.

The documentation for this class was generated from the following files:

- ROLOptimizer.hpp
- ROLOptimizer.cpp
14.41 DakotaROLObjectiveGrad Class Reference

Inheritance diagram for DakotaROLObjectiveGrad:

```
DakotaROLObjectiveGrad
  |__ DakotaROLObjective
  |  |__ DakotaROLObjectiveHess
```

Public Member Functions

- **DakotaROLObjectiveGrad (Model &model)**  
  Constructor.

- **virtual ~DakotaROLObjectiveGrad ()**  
  Destructor.

- **void gradient (std::vector<Real> &g, const std::vector<Real> &x, Real &tol) override**  
  Function to return the response gradient to ROL.

Additional Inherited Members

14.41.1 Detailed Description

DakotaROLObjectiveGrad is derived from DakotaROLObjective. It implements overrides of ROL member functions to provide a Dakota-specific Gradient support for the objective function. This separate class is needed to allow for the option of utilizing ROL’s finite-differenced gradients.

14.41.2 Constructor & Destructor Documentation

DakotaROLObjectiveGrad ( Model & model )

Constructor.

Implementation of the DakotaROLObjectiveGrad class.

The documentation for this class was generated from the following files:

- ROLOptimizer.hpp
- ROLOptimizer.cpp

14.42 DakotaROLObjectiveHess Class Reference

Inheritance diagram for DakotaROLObjectiveHess:
Public Member Functions

- **DakotaROLObjectiveHess (Model &model)**
  Constructor.
- **virtual ~DakotaROLObjectiveHess ()**
  Destructor.
- **void hessVec (std::vector<Real> &hv, const std::vector<Real> &v, const std::vector<Real> &x, Real &tol) override**
  Function to return Hessian-vector product needed by ROL when using user/Dakota-supplied Hessians.
- **void invHessVec (std::vector<Real> &hv, const std::vector<Real> &v, const std::vector<Real> &x, Real &tol) override**
  This function is not used by ROL algorithms currently supported by Dakota but is included to protect against unexpected behavior.

Additional Inherited Members

14.42.1 Detailed Description

**DakotaROLObjectiveHess** is derived from **DakotaROLObjectiveGrad**. It implements overrides of ROL member functions to provide a Dakota-specific implementation of a Hessian-vector product. This separate class is needed (rather than putting the product into **DakotaROLObjective**) because logic in ROL does not always protect against calling the Hessian-vector product in cases where there is not actually a Hessian provided.

14.42.2 Constructor & Destructor Documentation

**DakotaROLObjectiveHess ( Model & model )**

Constructor.

Implementation of the **DakotaROLObjectiveHess** class.

The documentation for this class was generated from the following files:

- ROLOptimizer.hpp
- ROLOptimizer.cpp

14.43 DataEnvironment Class Reference

Handle class for environment specification data.
Public Member Functions

- DataEnvironment ()
  constructor
- DataEnvironment (const DataEnvironment &)
  copy constructor
- ~DataEnvironment ()
  destructor
- DataEnvironment & operator= (const DataEnvironment &)
  assignment operator
- void write (std::ostream &s) const
  write a DataEnvironment object to an std::ostream
- void read (MPIUnpackBuffer &s)
  read a DataEnvironment object from a packed MPI buffer
- void write (MPIPackBuffer &s) const
  write a DataEnvironment object to a packed MPI buffer
- DataEnvironmentRep * data_rep ()
  return dataEnvRep

Private Attributes

- DataEnvironmentRep * dataEnvRep
  pointer to the body (handle-body idiom)

Friends

- class ProblemDescDB
- class NIDRProblemDescDB

14.43.1 Detailed Description

Handle class for environment specification data.

The DataEnvironment class is used to provide a memory management handle for the data in DataEnvironmentRep. It is populated by IDRProblemDescDB::environment_kwhandler() and is queried by the ProblemDescDB::get_\langle\text{datatype}\rangle() functions. A single DataEnvironment object is maintained in ProblemDescDB::environment-Spec.

The documentation for this class was generated from the following files:

- DataEnvironment.hpp
- DataEnvironment.cpp

14.44 DataEnvironmentRep Class Reference

Body class for environment specification data.
Public Attributes

- bool checkFlag
  flag for whether to run in check only mode (default false)
- String outputFile
  file name for output redirection (overrides command-line)
- String errorFile
  file name for error redirection (overrides command-line)
- String readRestart
  file name for restart read (overrides command-line)
- int stopRestart
  record at which to stop reading restart
- String writeRestart
  file name for restart write (overrides command-line)
- bool preRunFlag
  flags invocation with command line option -pre_run
- bool runFlag
  flags invocation with command line option -run
- bool postRunFlag
  flags invocation with command line option -post_run
- String preRunInput
  filename for pre_run input
- String preRunOutput
  filename for pre_run output
- String runInput
  filename for run input
- String runOutput
  filename for run output
- String postRunInput
  filename for post_run input
- String postRunOutput
  filename for post_run output
- unsigned short preRunOutputFormat
  tabular format for pre_run output
- unsigned short postRunInputFormat
  tabular format for post_run input
- bool graphicsFlag
  flags use of graphics by the environment (from the graphics specification in EnvIndControl)
- bool tabularDataFlag
  flags tabular data collection by the environment (from the tabular_graphics_data specification in EnvIndControl)
- String tabularDataFile
  the filename used for tabular data collection by the environment (from the tabular_graphics_file specification in EnvIndControl)
- unsigned short tabularFormat
CHAPTER 14. CLASS DOCUMENTATION

format for tabular data files (see enum)

- `int outputPrecision`
  output precision for tabular and screen output
- `bool resultsOutputFlag`
  flags use of results output to default file
- `String resultsOutputFile`
  named file for results output
- `unsigned short resultsOutputFormat`
  Results output format.
- `unsigned short modelEvalsSelection`
  Model selection for eval storage.
- `unsigned short interfEvalsSelection`
  Interface selection for eval storage.
- `String topMethodPointer`
  method identifier for the environment (from the top_method_pointer specification)

Private Member Functions

- `DataEnvironmentRep ()`
  constructor
- `~DataEnvironmentRep ()`
  destructor
- `void write (std::ostream &s) const`
  write a `DataEnvironmentRep` object to an `std::ostream`
- `void read (MPIUnpackBuffer &s)`
  read a `DataEnvironmentRep` object from a packed MPI buffer
- `void write (MPIPackBuffer &s) const`
  write a `DataEnvironmentRep` object to a packed MPI buffer

Private Attributes

- `int referenceCount`
  number of handle objects sharing this `dataEnvironmentRep`

Friends

- `class DataEnvironment`
  the handle class can access attributes of the body class directly
14.44.1 Detailed Description

Body class for environment specification data.

The DataEnvironmentRep class is used to contain the data from the environment keyword specification. Default values are managed in the DataEnvironmentRep constructor. Data is public to avoid maintaining set/get functions, but is still encapsulated within ProblemDescDB since ProblemDescDB::environmentSpec is private.

The documentation for this class was generated from the following files:

- DataEnvironment.hpp
- DataEnvironment.cpp

14.45 DataFitSurrBasedLocalTraits Class Reference

Class for provably-convergent local surrogate-based optimization and nonlinear least squares.

Inheritance diagram for DataFitSurrBasedLocalTraits:

```
TraitsBase
```

```
DataFitSurrBasedLocalTraits
```

Public Member Functions

- **DataFitSurrBasedLocalTraits ()**
  
  *default constructor*

- **virtual ~DataFitSurrBasedLocalTraits ()**
  
  *destructor*

- **virtual bool is_derived ()**
  
  *A temporary query used in the refactor.*

- **bool supports_continuous_variables ()**
  
  *Return the flag indicating whether method supports continuous variables.*

- **bool supports_linear_equality ()**
  
  *Return the flag indicating whether method supports linear equalities.*

- **bool supports_linear_inequality ()**
  
  *Return the flag indicating whether method supports linear inequalities.*

- **bool supports_nonlinear_equality ()**
  
  *Return the flag indicating whether method supports nonlinear equalities.*

- **bool supports_nonlinear_inequality ()**
  
  *Return the flag indicating whether method supports nonlinear inequalities.*
14.45.1 Detailed Description

Class for provably-convergent local surrogate-based optimization and nonlinear least squares.

This minimizer uses a SurrogateModel to perform minimization based on local, global, or hierarchical surrogates. It achieves provable convergence through the use of a sequence of trust regions and the application of surrogate corrections at the trust region centers. A version of TraitsBase specialized for local surrogate-based minimizer.

The documentation for this class was generated from the following file:

- DataFitSurrBasedLocalMinimizer.hpp

14.46 DataFitSurrModel Class Reference

Derived model class within the surrogate model branch for managing data fit surrogates (global and local)

Inheritance diagram for DataFitSurrModel:

```
Model

SurrogateModel

DataFitSurrModel
```

Public Member Functions

- **DataFitSurrModel (ProblemDescDB &problem_db)**
  
  constructor

- **DataFitSurrModel (Iterator &dace_iterator, Model &actual_model, const ActiveSet &set, const String &approx_type, const UShortArray &approx_order, short corr_type, short corr_order, short data_order, short output_level, const String &point_reuse, const String &import_build_points_file=String(), unsigned short import_build_format=TABULAR_ANNOTATED, bool import_build_active_only=false, const String &export_approx_points_file=String(), unsigned short export_approx_format=TABULAR_ANNOTATED)**

  alternate constructor for instantiations on the fly

- **~DataFitSurrModel ()**

  destructor

- **void total_points (int points)**

  set pointsTotal and pointsManagement mode

- **int required_points ()**

  return points required for build according to pointsManagement mode

- **void declare_sources ()**

  Declare a model’s sources to the evaluationsDB.
Protected Member Functions

- size_t qoi () const
  
  return number of unique response functions (managing any aggregations)

- DiscrepancyCorrection & discrepancy_correction ()
  
  return the DiscrepancyCorrection object used by SurrogateModels

- short correction_type ()
  
  return the correction type from the DiscrepancyCorrection object used by SurrogateModels

- void correction_type (short corr_type)
  
  set the correction type from the DiscrepancyCorrection object used by SurrogateModels

- bool initialize_mapping (ParLevLIter pl_iter)
  
  Perform any global updates prior to individual evaluate() calls; returns true if the variables size has changed.

- bool finalize_mapping ()

- void nested_variable_mappings (const SizetArray &c_index1, const SizetArray &di_index1, const SizetArray &ds_index1, const SizetArray &dr_index1, const ShortArray &c_target2, const ShortArray &di_target2, const ShortArray &ds_target2, const ShortArray &dr_target2)
  
  set primaryA(C,DI,DS,DR)VarMapIndices, secondaryA(C,DLDS,DR)VarMapTargets (coming from a higher-level NestedModel context to inform derivative est.)

- const SizetArray & nested_acv1_indices () const
  
  return primaryACVarMapIndices

- const ShortArray & nested_acv2_targets () const
  
  return secondaryACVarMapTargets

- short query_distribution_parameter_derivatives () const
  
  calculate and return derivative composition of final results w.r.t. distribution parameters (none, all, or mixed)

- void check_submodel_compatibility (const Model &sub_model)
  
  verify compatibility between SurrogateModel attributes and attributes of the submodel (DataFitSurrModel::actualModel or HierarchSurrModel::highFidelityModel)

- void derived_evaluate (const ActiveSet &set)

- void derived_evaluate_nowait (const ActiveSet &set)

- const IntResponseMap & derived_synchronize ()

- const IntResponseMap & derived_synchronize_nowait ()

- void asv_inflate_build (const ShortArray &orig_asv, ShortArray &actual_asv)
  
  map incoming ASV into actual request for surrogate construction, managing any mismatch in sizes due to response aggregation modes in actualModel

- void asv_split_eval (const ShortArray &orig_asv, ShortArray &actual_asv, ShortArray &approx_asv)
  
  split incoming ASV into actual and approximate evaluation requests, managing any mismatch in sizes due to response aggregation modes in actualModel

- Iterator & subordinate_iterator ()
  
  return dacelIterator

- void active_model_key (const UShortArray &mi_key)
  
  set active model key within approxInterface

- void clear_model_keys ()
  
  remove all model keys within approxInterface

- Model & surrogate_model ()
  
  return this model instance
• const Model & surrogate_model () const
  return this model instance
• Model & truth_model ()
  return actualModel
• const Model & truth_model () const
  return actualModel
• void derived_subordinate_models (ModelList &ml, bool recurse_flag)
  return actualModel (and optionally its sub-models)
• void resize_from_subordinate_model (size_t depth=std::numeric_limits<size_t>::max())
  pass request to actualModel if recursing
• void update_from_subordinate_model (size_t depth=std::numeric_limits<size_t>::max())
  pass request to actualModel if recursing and then update from it
• Interface & derived_interface ()
  return approxInterface
• void primary_response_fn_weights (const RealVector &wts, bool recurse_flag=true)
  set the relative weightings for multiple objective functions or least squares terms and optionally recurses into actualModel
• void surrogate_response_mode (short mode)
  set responseMode and pass any bypass request on to actualModel for any lower-level surrogates.
• void surrogate_function_indices (const IntSet &surr_fn_indices)
  (re)set the surrogate index set in SurrogateModel::surrogateFnIndices and ApproximationInterface::approxFnIndices
• void build_approximation ()
  Builds the local/multipoint/global approximation using daceIterator/actualModel to generate new data points.
• bool build_approximation (const Variables &vars, const IntResponsePair &response_pr)
  Builds the local/multipoint/global approximation using daceIterator/actualModel to generate new data points that augment the passed vars/response anchor point.
• void rebuild_approximation ()
  Rebuilds the local/multipoint/global approximation using daceIterator/actualModel to generate an increment of appended data.
• void rebuild_approximation (const IntResponsePair &response_pr)
  Rebuilds the local/multipoint/global approximation using the passed response data for a single sample.
• void rebuild_approximation (const IntResponseMap &allResp)
  Rebuilds the local/multipoint/global approximation using the passed response data for a set of samples.
• void update_approximation (bool rebuild_flag)
  replaces the approximation data with daceIterator results and rebuilds the approximation if requested
• void update_approximation (const Variables &vars, const IntResponsePair &response_pr, bool rebuild_flag)
  replaces the anchor point, and rebuilds the approximation if requested
• void update_approximation (const VariablesArray &vars_array, const IntResponseMap &resp_map, bool rebuild_flag)
  replaces the current points array and rebuilds the approximation if requested
• void update_approximation (const RealMatrix &samples, const IntResponseMap &resp_map, bool rebuild_flag)
  replaces the current points array and rebuilds the approximation if requested
void append_approximation (bool rebuild_flag)
- appends daceIterator results to a global approximation and rebuilds it if requested

void append_approximation (const Variables &vars, const IntResponsePair &response_pr, bool rebuild_flag)
- appends a point to a global approximation and rebuilds it if requested

void append_approximation (const VariablesArray &vars_array, const IntResponseMap &resp_map, bool rebuild_flag)
- appends an array of points to a global approximation and rebuilds it if requested

void append_approximation (const RealMatrix &samples, const IntResponseMap &resp_map, bool rebuild_flag)
- appends a matrix of points to a global approximation and rebuilds it if requested

void pop_approximation (bool save_surr_data, bool rebuild_flag=false)
- remove approximation data added on previous append_approximation() call or a specified number of points

void push_approximation ()
- retrieve a previous approximation data state

bool push_available ()
- query for whether a trial increment can be retrieved

void finalize_approximation ()
- finalize data fit by applying all previous trial increments

void combine_approximation ()
- combine all level approximations into a separate composite approximation

void combined_to_active (bool clear_combined=true)
- promote the combined approximation into the active one

void clear_inactive ()
- clear inactive data stored in the approxInterface

void run_dace ()
- execute the DACE iterator to generate build data

SharedApproxData & shared_approximation ()
- retrieve the SharedApproxData from approxInterface

std::vector< Approximation > & approximations ()
- retrieve the set of Approximations from approxInterface

const RealVectorArray & approximation_coefficients (bool normalized=false)
- return the approximation coefficients from each Approximation (request forwarded to approxInterface)

void approximation_coefficients (const RealVectorArray &approx_coeffs, bool normalized=false)
- set the approximation coefficients within each Approximation (request forwarded to approxInterface)

const RealVector & approximation_variance (const Variables &vars)
- return the approximation variance from each Approximation (request forwarded to approxInterface)

const Pecos::SurrogateData & approximation_data (size_t fn_index)
- return the approximation data from a particular Approximation (request forwarded to approxInterface)

void component_parallel_mode (short mode)
- update component parallel mode for supporting parallelism in actualModel

IntIntPair estimate_partition_bounds (int max_eval_concurrency)
- estimate the minimum and maximum partition sizes that can be utilized by this Model

void derived_init_communicators (ParLevLIter pl_iter, int max_eval_concurrency, bool recurse_flag=true)
set up actualModel for parallel operations

- void derived_init_serial()
  set up actualModel for serial operations.
- void derived_set_communicators(ParLevLIter pl_iter, int max_eval_concurrency, bool recurse_flag=true)
  set active parallel configuration within actualModel

- void derived_free_communicators(ParLevLIter pl_iter, int max_eval_concurrency, bool recurse_flag=true)
  deallocate communicator partitions for the DataFitSurrModel (request forwarded to actualModel)
- void serve_run(ParLevLIter pl_iter, int max_eval_concurrency)
  Service actualModel job requests received from the master. Completes when a termination message is received from stop_servers().
- void stop_servers()
  Executed by the master to terminate actualModel server operations when DataFitSurrModel iteration is complete.

- void inactive_view(short view, bool recurse_flag=true)
  update the Model’s inactive view based on higher level (nested) context and optionally recurse into actualModel

- const String & interface_id() const
  return the approxInterface identifier
- bool evaluation_cache(bool recurse_flag=true) const
  if recurse_flag, return the actualModel evaluation cache usage
- bool restart_file(bool recurse_flag=true) const
  if recurse_flag, return the actualModel restart file usage
- void set_evaluation_reference()
  set the evaluation counter reference points for the DataFitSurrModel (request forwarded to approxInterface and actualModel)
- void fine_grained_evaluation_counters()
  request fine-grained evaluation reporting within approxInterface and actualModel
- void print_evaluation_summary(std::ostream &s, bool minimal_header=false, bool relative_count=true) const
  print the evaluation summary for the DataFitSurrModel (request forwarded to approxInterface and actualModel)
- void warm_start_flag(const bool flag)
  set the warm start flag, including actualModel

Protected Attributes

- const bool exportSurrogate
  whether to export the surrogate to file
- const bool autoRefine
  whether to automatically refine the surrogate during the build phase
- const int maxIterations
  Maximum number of times to refine the surrogate.
- const int maxFuncEvals
  Maximum number of evaluations while refining a surrogate.
- const Real convergenceTolerance
  Convergence criterion, compared to CV score for specified metric.
- const int softConvergenceLimit
  Max number of iterations for which there is no average improvement.
- const String refineCVMetric
  Type of error metric to test for surrogate refinement convergence.
- const int refineCVFolds
  Number of cross validation folds for surrogate refinement.

Private Member Functions

- void import_points (unsigned short tabular_format, bool use_var_labels, bool active_only)
  optionally read surrogate data points from provided file
- void initialize_export ()
  initialize file stream for exporting surrogate evaluations
- void finalize_export ()
  finalize file stream for exporting surrogate evaluations
- void export_point (int eval_id, const Variables &vars, const Response &resp)
  initialize file stream for exporting surrogate evaluations
- void derived_synchronize_approx (bool block, IntResponseMap &approx_resp_map_rekey)
  Common code for processing of approximate response maps shared by derived_synchronize() and derived_synchronize_nowait()
- void update_local_reference ()
  Updates fit arrays for local or multipoint approximations.
- void build_local_multipoint ()
  Builds a local or multipoint approximation using actualModel.
- void build_local_multipoint (const Variables &vars, const IntResponsePair &response_pr)
  Builds a local or multipoint approximation using actualModel.
- void update_global_reference ()
  Updates fit arrays for global approximations.
- void build_global ()
  Builds a global approximation using daceIterator.
- void rebuild_global ()
  Rebuilds a global approximation by generating new data using daceIterator and appending to approxInterface.
- void refine_surrogate ()
  Refine the built surrogate until convergence criteria are met.
- void clear_approx_interface ()
  clear current data from approxInterface
- void update_approx_interface (const Variables &vars, const IntResponsePair &response_pr)
  update anchor data in approxInterface
- void build_approx_interface ()
  build the approxInterface surrogate, passing variable bounds
- void init_model (Model &model)
  update actualModel with data from constraints/labels/sets
- void update_model (Model &model)
  update actualModel with data from current variables/bounds
• void update_from_model (const Model &model)
  update current variables/labels/bounds/targets with data from actualModel
• bool consistent (const Variables &vars) const
  test if inactive state is consistent
• bool inside (const Variables &vars) const
  test if active vars are within [l_bnds, u_bnds]
• bool active_vars_compare (const Variables &vars, const Pecos::SurrogateDataVars &sdv) const
  test for exact equality in values between active vars and sdv

Private Attributes

• DiscrepancyCorrection deltaCorr
  manages construction and application of correction functions that are applied to a surrogate model (DataFitSurr or HierarchSurr) in order to reproduce high fidelity data.
• int pointsTotal
  total points the user specified to construct the surrogate
• short pointsManagement
  configuration for points management in build_global()
• String pointReuse
  type of point reuse for approximation builds: all, region (default if points file), or none (default if no points file)
• String importPointsFile
  file name from import_build_points_file specification
• String exportPointsFile
  file name from export_approx_points_file specification
• unsigned short exportFormat
  file export format for variables and approximate responses
• std::ofstream exportFileStream
  output file stream for export_approx_points_file specification
• Interface approxInterface
  manages the building and subsequent evaluation of the approximations (required for both global and local)
• Model actualModel
  the truth model which provides evaluations for building the surrogate (optional for global, required for local)
• Iterator daceIterator
  selects parameter sets on which to evaluate actualModel in order to generate the necessary data for building global approximations (optional for global since restart data may also be used)

Additional Inherited Members

14.46.1 Detailed Description

Derived model class within the surrogate model branch for managing data fit surrogates (global and local)

The DataFitSurrModel class manages global or local approximations (surrogates that involve data fits) that are used in place of an expensive model. The class contains an approxInterface (required for both global and local) which manages the approximate function evaluations, an actualModel (optional for global, required for local) which provides truth evaluations for building the surrogate, and a daceIterator (optional for global, not used for local) which selects parameter sets on which to evaluate actualModel in order to generate the necessary data for building global approximations.
14.46.2 Constructor & Destructor Documentation

\~DataFitSurrModel( ) [inline]
destructor

Virtual destructor handles referenceCount at base Model level.
References DataFitSurrModel::exportPointsFile, and DataFitSurrModel::finalize_export().

14.46.3 Member Function Documentation

bool finalize_mapping ( ) [protected], [virtual]

Inactive variables must be propagated when a HierarchSurrModel is employed by a sub-iterator (e.g., OUU with MLMC or MLPCE). In current use cases, this can occur once per sub-iterator execution within Model::initialize_mapping().

Reimplemented from Model.
References DataFitSurrModel::actualModel, and Model::finalize_mapping().

void derived_evaluate ( const ActiveSet & set ) [protected], [virtual]

Compute the response synchronously using actualModel, approxInterface, or both (mixed case). For the approxInterface portion, build the approximation if needed, evaluate the approximate response, and apply correction (if active) to the results.

Reimplemented from Model.
References Response::active_set(), DataFitSurrModel::actualModel, SurrogateModel::aggregate_response(), Interface::analysis_components(), DiscrepancyCorrection::apply(), SurrogateModel::approxBuilds, DataFitSurrModel::approxInterface, DataFitSurrModel::asv_split_eval(), DataFitSurrModel::build_approximation(), DataFitSurrModel::component_parallel_mode(), DiscrepancyCorrection::compute(), Response::copy(), Model::current_response(), Model::currentResponse, Model::currentVariables, DataFitSurrModel::deltaCorr, Model::eval_tag_prefix(), Model::eval_tag_prefix(), Model::evalTagPrefix, Model::evaluate(), Interface::evaluation_id(), Model::evaluationsDB, DataFitSurrModel::export_point(), DataFitSurrModel::exportPointsFile, SurrogateModel::force_rebuild(), Model::hierarchicalTagging, Interface::interface_id(), Model::interEvaluationsDBState, Interface::map(), Model::modelId, Model::outputLevel, ActiveSet::request_vector(), SurrogateModel::response_combine(), SurrogateModel::responseMode, SurrogateModel::surrModelEvalCntr, Response::update(), and DataFitSurrModel::update_model().

void derived_evaluate_nowait ( const ActiveSet & set ) [protected], [virtual]

Compute the response asynchronously using actualModel, approxInterface, or both (mixed case). For the approxInterface portion, build the approximation if needed and evaluate the approximate response in a quasi-asynchronous approach (ApproximationInterface::map() performs the map synchronously and bookkeeps the results for return in derived_synchronize() below).

Reimplemented from Model.
References DataFitSurrModel::actualModel, Interface::analysis_components(), SurrogateModel::approxBuilds, DataFitSurrModel::approxInterface, DataFitSurrModel::asv_split_eval(), DataFitSurrModel::build_approximation(), Variables::copy(), Model::currentResponse, Model::currentVariables, Model::eval_tag_prefix(), Model::eval_tag_prefix(), Model::evalTagPrefix, Model::evaluate_nowait(), Interface::evaluation_id(), Model::evaluation_id(), Model::evaluationsDB, DataFitSurrModel::exportPointsFile, SurrogateModel::force_rebuild(), Model::hierarchicalTagging, Interface::interface_id(), Model::interEvaluationsDBState, Interface::map(), Model::modelId, SurrogateModel::rawVarsMap, ActiveSet::request_vector(), SurrogateModel::responseMode, SurrogateModel::surrIdMap, SurrogateModel::surrModelEvalCntr, SurrogateModel::truthIdMap, and DataFitSurrModel::update_model().
const IntResponseMap & derived_synchronize()  [protected], [virtual]

Blocking retrieval of asynchronous evaluations from actualModel, approxInterface, or both (mixed case). For the approxInterface portion, apply correction (if active) to each response in the array. derived_synchronize() is designed for the general case where derived_evaluate_nowait() may be inconsistent in its use of actual evaluations, approximate evaluations, or both.

Reimplemented from Model.

References DataFitSurrModel::actualModel, SurrogateModel::aggregate_response(), SurrogateModel::check_key(), DataFitSurrModel::component_parallel_mode(), DiscrepancyCorrection::compute(), DataFitSurrModel::delta-Corr, DataFitSurrModel::derived_synchronize_approx(), Model::outputLevel, Model::rekey_synch(), SurrogateModel::response_combine(), SurrogateModel::responseMode, SurrogateModel::surrIdMap, SurrogateModel::surrResponseMap, and SurrogateModel::truthIdMap.

const IntResponseMap & derived_synchronize_nowait()  [protected], [virtual]

Nonblocking retrieval of asynchronous evaluations from actualModel, approxInterface, or both (mixed case). For the approxInterface portion, apply correction (if active) to each response in the map. derived_synchronize_nowait() is designed for the general case where derived_evaluate_nowait() may be inconsistent in its use of actual evals, approx evals, or both.

Reimplemented from Model.

References Dakota::abort_handler(), DataFitSurrModel::actualModel, SurrogateModel::aggregate_response(), SurrogateModel::cachedApproxRespMap, DataFitSurrModel::component_parallel_mode(), DiscrepancyCorrection::compute(), DataFitSurrModel::deltaCorr, DataFitSurrModel::derived_synchronize_approx(), Model::outputLevel, Model::rekey_synch(), SurrogateModel::response_combine(), SurrogateModel::responseMode, SurrogateModel::surrIdMap, SurrogateModel::surrResponseMap, and SurrogateModel::truthIdMap.

void build_approximation()  [protected], [virtual]

Builds the local/multipoint/global approximation using daceIterator/actualModel to generate new data points.

This function constructs a new approximation, discarding any previous data. It constructs any required data for SurrogateData::{vars,resp}Data and does not define an anchor point for SurrogateData::anchor{Vars,Resp}, so is an unconstrained build.

Reimplemented from Model.

References DataFitSurrModel::actualModel, DataFitSurrModel::build_global(), DataFitSurrModel::build_local_multipoint(), DataFitSurrModel::clear_approx_interface(), Dakota::strbegins(), Model::surrogateType, DataFitSurrModel::update_global_reference(), DataFitSurrModel::update_local_reference(), and DataFitSurrModel::update_model().

Referenced by DataFitSurrModel::derived_evaluate(), and DataFitSurrModel::derived_evaluate_nowait().

bool build_approximation(const Variables & vars, const IntResponsePair & response_pr )  [protected], [virtual]

Builds the local/multipoint/global approximation using daceIterator/actualModel to generate new data points that augment the passed vars/response anchor point.

This function constructs a new approximation, discarding any previous data. It uses the passed data to populate SurrogateData::anchor{Vars,Resp} and constructs any required data points for SurrogateData::{vars,resp}Data.

Reimplemented from Model.

References DataFitSurrModel::actualModel, DataFitSurrModel::build_global(), DataFitSurrModel::build_local_multipoint(), Dakota::strbegins(), Model::surrogateType, DataFitSurrModel::update_approx_interface(), DataFitSurrModel::update_global_reference(), DataFitSurrModel::update_local_reference(), and DataFitSurrModel::update_model().
void rebuild_approximation( ) [protected], [virtual]

Rebuilds the local/multipoint/global approximation using daceIterator/actualModel to generate an increment of appended data.

This function updates an existing approximation, by appending new data. It does not define an anchor point, so is an unconstrained build.

Reimplemented from Model.

References DataFitSurrModel::actualModel, DataFitSurrModel::build_local_multipoint(), Model::outputLevel, DataFitSurrModel::rebuild_global(), Dakota::strbegins(), Model::surrogateType, and DataFitSurrModel::update_model().

Referenced by DataFitSurrModel::append_approximation(), and DataFitSurrModel::update_approximation().

void update_approximation( bool rebuild_flag ) [protected], [virtual]

replaces the approximation data with daceIterator results and rebuilds the approximation if requested.

This function populates/replaces SurrogateData::anchor{Vars,Resp} and rebuilds the approximation, if requested. It does not clear other data (i.e., SurrogateData::{vars,resp}Data) and does not update the actualModel with revised bounds, labels, etc. Thus, it updates data from a previous call to build_approximation(), and is not intended to be used in isolation.

Reimplemented from Model.

References Iterator::all_responses(), Iterator::all_samples(), Iterator::all_variables(), DataFitSurrModel::approxInterface, Iterator::compact_mode(), DataFitSurrModel::daceIterator, Model::outputLevel, DataFitSurrModel::rebuild_approximation(), Model::surrogateType, and Interface::update_approximation().

void update_approximation( const Variables & vars, const IntResponsePair & response_prior, bool rebuild_flag ) [protected], [virtual]

replaces the anchor point, and rebuilds the approximation if requested.

This function populates/replaces SurrogateData::anchor{Vars,Resp} and rebuilds the approximation, if requested. It does not clear other data (i.e., SurrogateData::{vars,resp}Data) and does not update the actualModel with revised bounds, labels, etc. Thus, it updates data from a previous call to build_approximation(), and is not intended to be used in isolation.

Reimplemented from Model.

References DataFitSurrModel::approxInterface, Model::outputLevel, DataFitSurrModel::rebuild_approximation(), Model::surrogateType, and Interface::update_approximation().

void update_approximation( const VariablesArray & vars_array, const IntResponseMap & response_map, bool rebuild_flag ) [protected], [virtual]

replaces the current points array and rebuilds the approximation if requested.

This function populates/replaces SurrogateData::{vars,resp}Data and rebuilds the approximation, if requested. It does not clear other data (i.e., SurrogateData::anchor{Vars,Resp}) and does not update the actualModel with revised bounds, labels, etc. Thus, it updates data from a previous call to build_approximation(), and is not intended to be used in isolation.

Reimplemented from Model.

References DataFitSurrModel::approxInterface, Model::outputLevel, DataFitSurrModel::rebuild_approximation(), Model::surrogateType, and Interface::update_approximation().
void update_approximation ( const RealMatrix& samples, const IntResponseMap& resp_map, bool rebuild_flag ) [protected], [virtual]

replaces the current points array and rebuilds the approximation if requested

This function populates/replaces SurrogateData::{vars,resp}Data and rebuilds the approximation, if requested. It does not clear other data (i.e., SurrogateData::anchor{Vars,Resp}) and does not update the actualModel with revised bounds, labels, etc. Thus, it updates data from a previous call to build_approximation(), and is not intended to be used in isolation.

Reimplemented from Model.
References DataFitSurrModel::approxInterface, Model::outputLevel, DataFitSurrModel::rebuild_approximation(), Model::surrogateType, and Interface::update_approximation().

void append_approximation ( bool rebuild_flag ) [protected], [virtual]

appends daceIterator results to a global approximation and rebuilds it if requested

This function appends all{Samples, Variables, Responses} to SurrogateData::{vars,resp}Data and rebuilds the approximation, if requested.

Reimplemented from Model.
References Iterator::all_responses(), Iterator::all_samples(), Iterator::all_variables(), Interface::append_approximation(), DataFitSurrModel::approxInterface, Iterator::compact_mode(), DataFitSurrModel::daceIterator, Model::outputLevel, DataFitSurrModel::rebuild_approximation(), and Model::surrogateType.

Referenced by DataFitSurrModel::build_global(), DataFitSurrModel::rebuild_global(), and DataFitSurrModel::refine_surrogate().

void append_approximation ( const Variables& vars, const IntResponsePair& response_pr, bool rebuild_flag ) [protected], [virtual]

appends a point to a global approximation and rebuilds it if requested

This function appends one point to SurrogateData::{vars,resp}Data and rebuilds the approximation, if requested. It does not modify other data (i.e., SurrogateData::anchor{Vars,Resp}) and does not update the actualModel with revised bounds, labels, etc. Thus, it appends to data from a previous call to build_approximation(), and is not intended to be used in isolation.

Reimplemented from Model.
References Interface::append_approximation(), DataFitSurrModel::approxInterface, Model::outputLevel, DataFitSurrModel::rebuild_approximation(), and Model::surrogateType.

void append_approximation ( const VariablesArray& vars_array, const IntResponseMap& resp_map, bool rebuild_flag ) [protected], [virtual]

appends an array of points to a global approximation and rebuilds it if requested

This function appends multiple points to SurrogateData::{vars,resp}Data and rebuilds the approximation, if requested. It does not modify other data (i.e., SurrogateData::anchor{Vars,Resp}) and does not update the actualModel with revised bounds, labels, etc. Thus, it appends to data from a previous call to build_approximation(), and is not intended to be used in isolation.

Reimplemented from Model.
References Interface::append_approximation(), DataFitSurrModel::approxInterface, Model::outputLevel, DataFitSurrModel::rebuild_approximation(), and Model::surrogateType.
void append_approximation ( const RealMatrix & samples, const IntResponseMap & resp_map, bool rebuild_flag ) [protected], [virtual]

appends a matrix of points to a global approximation and rebuilds it if requested

This function appends multiple points to SurrogateData::{vars,resp}Data and rebuilds the approximation, if requested. It does not modify other data (i.e., SurrogateData::anchor::{Vars,Resp}) and does not update the actual-Model with revised bounds, labels, etc. Thus, it appends to data from a previous call to build_approximation(), and is not intended to be used in isolation.

Reimplemented from Model.

References Interface::append_approximation(), DataFitSurrModel::approxInterface, Model::outputLevel, DataFitSurrModel::rebuild_approximation(), and Model::surrogateType.

void derived_init_communicators ( ParLevLIter pl_iter, int max_eval_concurrency, bool recurse_flag = true ) [protected], [virtual]

set up actualModel for parallel operations

asynchronous flags need to be initialized for the sub-models. In addition, max_eval_concurrency is the outer level iterator concurrency, not the DACE concurrency that actualModel will see, and recomputing the message_lengths on the sub-model is probably not a bad idea either. Therefore, recompute everything on actualModel using init_communicators.

Reimplemented from Model.

References DataFitSurrModel::actualModel, DataFitSurrModel::approxInterface, DataFitSurrModel::daceIterator, Model::derivative_concurrency(), ProblemDescDB::get_db_method_node(), ProblemDescDB::get_db_model_node(), Iterator::init_communicators(), Model::init_communicators(), Iterator::is_null(), Model::is_null(), Iterator::iterated_model(), Iterator::maximum_evaluation_concurrency(), Iterator::method_id(), Interface::minimum_points(), Model::model_id(), Model::probDescDB, ProblemDescDB::set_db_list_nodes(), ProblemDescDB::set_db_method_node(), and ProblemDescDB::set_db_model_nodes().

void import_points ( unsigned short tabular_format, bool use_var_labels, bool active_only ) [private]

optionally read surrogate data points from provided file

Constructor helper to read the points file once, if provided, and then reuse its data as appropriate within build_global(). Surrogate data imports default to active/inactive variables, but user can override to active only process arrays of data from TabularIO::read_data_tabular() above

References DataFitSurrModel::actualModel, Response::copy(), Variables::copy(), Model::current_response(), Model::current_variables(), Model::currentResponse, Model::currentVariables, Dakota::data_pairs, ParamResponsePair::eval_id(), Model::evaluation_cach(), DataFitSurrModel::importPointsFile, ParamResponsePair::interface_id(), Model::interface_id(), Model::is_null(), Model::model_id(), Model::numFns, Model::outputLevel, Model::parallelLib, Model::restart_file(), Variables::total_active(), Variables::tv(), and ParallelLibrary::write_restart().

Referenced by DataFitSurrModel::DataFitSurrModel().

void initialize_export ( ) [private]

initialize file stream for exporting surrogate evaluations

Constructor helper to export approximation-based evaluations to a file.

References Model::currentResponse, Model::currentVariables, DataFitSurrModel::exportFileStream, DataFitSurrModel::exportFormat, and DataFitSurrModel::exportPointsFile.

Referenced by DataFitSurrModel::DataFitSurrModel().
void finalize_export( ) [private]

finalize file stream for exporting surrogate evaluations
   Constructor helper to export approximation-based evaluations to a file.
   References DataFitSurrModel::exportFileStream and DataFitSurrModel::exportPointsFile.
   Referenced by DataFitSurrModel::~DataFitSurrModel().

void export_point ( int eval_id, const Variables & vars, const Response & resp ) [private]

initialize file stream for exporting surrogate evaluations
   Constructor helper to export approximation-based evaluations to a file. Exports all variables, so it’s clear at
what values of inactive it was built at
   References DataFitSurrModel::exportFileStream, DataFitSurrModel::exportFormat, DataFitSurrModel::interface_id(), Model::iterator_space_to_user_space(), and Model::recastings().
   Referenced by DataFitSurrModel::derived_evaluate(), and DataFitSurrModel::derived_synchronize_approx().

void build_local_multipoint ( ) [private]

Builds a local or multipoint approximation using actualModel.
   Evaluate the value, gradient, and possibly Hessian needed for a local or multipoint approximation using actualModel.
   References Response::active_set(), DataFitSurrModel::actualModel, DataFitSurrModel::asv_inflate_build(), DataFitSurrModel::component_parallel_mode(), Model::continuous_variable_ids(), Model::current_response(), Model::current_variables(), ActiveSet::derivative_vector(), Model::evaluate(), Model::evaluation_id(), Model::hessian_type(), Model::numFns, ActiveSet::request_vector(), Dakota::strbegins(), SurrogateModel::surrogateFnIndices, and Model::surrogateType.
   Referenced by DataFitSurrModel::build_approximation(), and DataFitSurrModel::rebuild_approximation().

void build_global ( ) [private]

Builds a global approximation using daceIterator.
   Determine points to use in building the approximation and then evaluate them on actualModel using daceIterator. Any changes to the bounds should be performed by setting them at a higher level (e.g., SurrBasedOptStrategy).
   References Dakota::abort_handler(), DataFitSurrModel::active_vars_compare(), DataFitSurrModel::actualModel, Interface::append_approximation(), DataFitSurrModel::append_approximation(), SurrogateModel::approxBuilds, Interface::approximation_data(), DataFitSurrModel::approxInterface, DataFitSurrModel::autoRefine, DataFitSurrModel::build_approx_interface(), DataFitSurrModel::component_parallel_mode(), DataFitSurrModel::consistent(), Model::currentVariables, Variables::cv(), Model::cv(), DataFitSurrModel::daceIterator, Dakota::data_pairs, Variables::div(), Model::div(), Variables::drv(), Model::drv(), DataFitSurrModel::inside(), Model::interface_id(), Iterator::is_null(), Model::is_null(), Interface::minimum_points(), Iterator::num_samples(), Model::outputLevel, DataFitSurrModel::pointReuse, Model::recastings(), DataFitSurrModel::refine_surrogate(), DataFitSurrModel::required_points(), DataFitSurrModel::run_dace(), Iterator::sampling_reset(), SurrogateModel::surrogateFnIndices, and Model::user_space_to_iterator_space().
   Referenced by DataFitSurrModel::build_approximation().

void rebuild_global ( ) [private]

Rebuilds a global approximation by generating new data using daceIterator and appending to approxInterface.
   Determine points to use in rebuilding the approximation and then evaluate them on actualModel using daceIterator. Assumes data imports/reuse have been handled previously within build_global().
References Dakota::abort_handler(), DataFitSurrModel::append_approximation(), SurrogateModel::approxBuilds, Interface::approximation_data(), DataFitSurrModel::approxInterface, DataFitSurrModel::component_parallel_mode(), DataFitSurrModel::daceIterator, Interface::formulation_updated(), Iterator::is_null(), Interface::minimum_points(), Iterator::num_samples(), Model::outputLevel, Interface::rebuild_approximation(), DataFitSurrModel::required_points(), DataFitSurrModel::run_dace(), Iterator::sampling_reset(), and SurrogateModel::surrogateFnIndices.

Referenced by DataFitSurrModel::rebuild_approximation().

```cpp
void init_model ( Model & model ) [private]
update actualModel with data from constraints/labels/sets
Update variables and constraints data within model using values and labels from currentVariables and bound/linear/nonlinear constraints from userDefinedConstraints.
References Dakota::abort_handler(), Variables::all_continuous_variable_labels(), Model::all_continuous_variable_labels(), Variables::all_discrete_int_variable_labels(), Model::all_discrete_int_variable_labels(), Variables::all_discrete_real_variable_labels(), Model::all_discrete_real_variable_labels(), SurrogateModel::approxBuilds, Variables::continuous_variable_labels(), Model::continuous_variable_labels(), Model::currentVariables, Variables::cv(), Model::cv(), Variables::discrete_int_variable_labels(), Model::discrete_int_variable_labels(), Variables::div(), Model::div(), Variables::drv(), Model::drv(), Response::function_labels(), Variables::inactive_continuous_variable_labels(), Model::inactive_continuous_variable_labels(), Variables::inactive_discrete_int_variable_labels(), Model::inactive_discrete_int_variable_labels(), Variables::inactive_discrete_real_variable_labels(), Model::inactive_discrete_real_variable_labels(), Model::is_null(), Constraints::linear_eq_constraint_coeffis(), Model::linear_eq_constraint_coeffis(), Constraints::linear_ineq_constraint_lower_bounds(), Model::linear_ineq_constraint_lower_bounds(), Constraints::linear_ineq_constraint_upper_bounds(), Model::linear_ineq_constraint_upper_bounds(), Constraints::nonlinear_eq_constraint_targets(), Model::nonlinear_eq_constraint_targets(), Constraints::nonlinear_ineq_constraint_lower_bounds(), Model::nonlinear_ineq_constraint_lower_bounds(), Constraints::nonlinear_ineq_constraint_upper_bounds(), Model::nonlinear_ineq_constraint_upper_bounds(), Constraints::num_linear_eq_constraints(), Constraints::num_linear_ineq_constraints(), Constraints::num_nonlinear_eq_constraints(), Constraints::num_nonlinear_ineq_constraints(), Model::response_labels(), Model::userDefinedConstraints, and Variables::view().
Referenced by DataFitSurrModel::initialize.
```

```cpp
void update_model ( Model & model ) [private]
update actualModel with data from currentVariables/bounds
Update variables and constraints data within model using values and labels from currentVariables and bound/linear/nonlinear constraints from userDefinedConstraints.
References Dakota::abort_handler(), Constraints::all_continuous_lower_bounds(), Model::all_continuous_lower_bounds(), Constraints::all_continuous_upper_bounds(), Model::all_continuous_upper_bounds(), Variables::all_continuous_variables(), Model::all_continuous_variables(), Constraints::all_discrete_int_lower_bounds(), Model::all_discrete_int_lower_bounds(), Constraints::all_discrete_int_upper_bounds(), Model::all_discrete_int_upper_bounds(), Variables::all_discrete_int_variables(), Model::all_discrete_int_variables(), Constraints::all_discrete_real_lower_bounds(), Model::all_discrete_real_lower_bounds(), Constraints::all_discrete_real_upper_bounds(), Model::all_discrete_real_upper_bounds(), Variables::all_discrete_real_variables(), Model::all_discrete_real_variables(), SharedVariablesData::assemble_all_labels(), Constraints::continuous_lower_bounds(), Model::continuous_lower_bounds(), Constraints::continuous_upper_bounds(), Model::continuous_upper_bounds(), Variables::continuous_variables(), Model::continuous_variables(), Model::current_variables(), Model::currentVariables, Constraints::discrete_int_lower_bounds(), Model::discrete_int_lower_bounds(), Constraints::discrete_int_upper_bounds(), Model::discrete_int_upper_bounds(), Variables::discrete_int_variables(), Model::discrete_int_variables(), Constraints::discrete_real_lower_bounds(), Model::discrete_real_lower_bounds(), Constraints::discrete_real_upper_bounds(), Model::discrete_real_upper_bounds(), Variables::discrete_real_variables(), Model::discrete_real_variables(), Constraints::dace(), Iterator::sampling, and SurrogateModel::approxBuilds.
```
void update_from_model ( const Model & model ) [private]
update current variables/labels/bounds/targets with data from actualModel

Update values and labels in currentVariables and bound/linear/nonlinear constraints in userDefinedConstraints from variables and constraints data within model.

References Dakota::abort_handler(), Constraints::all_continuous_lower_bounds(), Model::all_continuous_lower_bounds(), Constraints::all_continuous_upper_bounds(), Model::all_continuous_upper_bounds(), Variables::all_continuous_variable_labels(), Model::all_continuous_variable_labels(), Variables::all_continuous_variables(), Model::all_continuous_variables(), Constraints::all_discrete_int_lower_bounds(), Model::all_discrete_int_lower_bounds(), Constraints::all_discrete_int_upper_bounds(), Model::all_discrete_int_upper_bounds(), Variables::all_discrete_int_variable_labels(), Model::all_discrete_int_variable_labels(), Variables::all_discrete_int_variables(), Model::all_discrete_int_variables(), Constraints::all_discrete_real_lower_bounds(), Model::all_discrete_real_lower_bounds(), Constraints::all_discrete_real_upper_bounds(), Model::all_discrete_real_upper_bounds(), Variables::all_discrete_real_variable_labels(), Model::all_discrete_real_variable_labels(), Variables::all_discrete_real_variables(), Model::all_discrete_real_variables(), Variables::all_discrete_string_variable_labels(), Model::all_discrete_string_variable_labels(), Variables::all_discrete_string_variable_labels(), Model::all_discrete_string_variable_labels(), SurrogateModel::approxBuilds, SharedVariablesData::assemble_all_labels(), Model::current_variables(), Model::currentResponse, Model::currentVariables, Variables::cv(), Model::cv(), Variables::div(), Model::div(), Variables::drv(), Model::drv(), Response::function_labels(), SharedVariablesData::id(), Constraints::linear_eq_constraint_coeffs(), Model::linear_eq_constraint_coeffs(), Constraints::linear_eq_constraint_targets(), Model::linear_eq_constraint_targets(), Constraints::linear_ineq_constraint_coeffs(), Constraints::linear_ineq_constraint_lower_bounds(), Model::linear_ineq_constraint_lower_bounds(), Constraints::linear_ineq_constraint_upper_bounds(), Model::linear_ineq_constraint_upper_bounds(), Model::multivariate_distribution(), Model::mvDist, Constraints::nonlinear_eq_constraint_targets(), Model::nonlinear_eq_constraint_targets(), Constraints::nonlinear_ineq_constraint_lower_bounds(), Model::nonlinear_ineq_constraint_lower_bounds(), Constraints::nonlinear_ineq_constraint_upper_bounds(), Model::nonlinear_ineq_constraint_upper_bounds(), Model::num_linear_eq_constraints(), Model::num_linear_eq_constraints(), Model::num_linear_ineq_constraints(), Model::num_linear_ineq_constraints(), Model::primary_response_fn_sense(), Model::primary_response_fn_weights(), Model::primaryRespFnSense, Model::primaryRespFnWts, Model::response_labels(), Variables::shared_data(), Dakota::svd(), and Model::userDefinedConstraints.

Referenced by DataFitSurrModel::DataFitSurrModel(), and DataFitSurrModel::update_from_subordinate_model().

14.46.4 Member Data Documentation

Model actualModel [private]

the truth model which provides evaluations for building the surrogate (optional for global, required for local)

actualModel is unrestricted in type; arbitrary nestings are possible.

Referenced by DataFitSurrModel::active_model_key(), DataFitSurrModel::asv_inflate_build(), DataFitSurrModel::asv_split_eval(), DataFitSurrModel::build_approx_interface(), DataFitSurrModel::build_approximation(), DataFitSurrModel::build_global(), DataFitSurrModel::build_local_multipoint(), DataFitSurrModel::consistent(), DataFitSurrModel::DataFitSurrModel(), DataFitSurrModel::declare_sources(), DataFitSurrModel::derived_evaluate(), DataFitSurrModel::derived_evaluate_nowait(), DataFitSurrModel::derived_free_communicators(), DataFitSurrModel::derived_init_communicators(), DataFitSurrModel::derived_init_serial(), DataFitSurrModel::derived_set_communicators(), DataFitSurrModel::derived_update(), Dakota::svd(), and Model::userDefinedConstraints.
14.47. DATAINTERFACE CLASS REFERENCE

DataFitSurrModel::derived_subordinate_models(), DataFitSurrModel::derived_synchronize(), DataFitSurrModel::derived_synchronize_nowait(), DataFitSurrModel::estimate_partition_bounds(), DataFitSurrModel::evaluation_cache(), DataFitSurrModel::finalize_mapping(), DataFitSurrModel::fine_grained_evaluation_counters(), DataFitSurrModel::import_points(), DataFitSurrModel::inactive_view(), DataFitSurrModel::initialize_mapping(), DataFitSurrModel::inside(), DataFitSurrModel::nested_acv1_indices(), DataFitSurrModel::nested_acv2_targets(), DataFitSurrModel::nested_variable_mappings(), DataFitSurrModel::partition_bounds(), DataFitSurrModel::primary_response_fn_weights(), DataFitSurrModel::print_evaluation_summary(), DataFitSurrModel::qoi(), DataFitSurrModel::query_distribution_parameter_derivatives(), DataFitSurrModel::restart_file(), DataFitSurrModel::run_dace(), DataFitSurrModel::serve_run(), DataFitSurrModel::stop_servers(), DataFitSurrModel::surrogate_response_mode(), DataFitSurrModel::truth_model(), DataFitSurrModel::update_from_subordinate_model(), DataFitSurrModel::update_global_reference(), DataFitSurrModel::update_local_reference(), and DataFitSurrModel::warm_start_flag().

The documentation for this class was generated from the following files:

- DataFitSurrModel.hpp
- DataFitSurrModel.cpp

14.47 DataInterface Class Reference

Handle class for interface specification data.

Public Member Functions

- **DataInterface ()**
  
  *constructor*

- **DataInterface (const DataInterface &)**
  
  *copy constructor*

- **~DataInterface ()**
  
  *destructor*

- **DataInterface & operator= (const DataInterface &)**
  
  *assignment operator*

- **void write (std::ostream &s) const**
  
  *write a DataInterface object to an std::ostream*

- **void read (MPIUnpackBuffer &s)**
  
  *read a DataInterface object from a packed MPI buffer*

- **void write (MPIPackBuffer &s) const**
  
  *write a DataInterface object to a packed MPI buffer*

- **DataInterfaceRep * data_rep ()**
  
  *return datafaceRep*

Static Public Member Functions

- **static bool id_compare (const DataInterface &di, const std::string &id)**
  
  *compares the idInterface attribute of DataInterface objects*
Private Attributes

- DataInterfaceRep * dataIfaceRep
  pointer to the body (handle-body idiom)

Friends

- class ProblemDescDB
- class NIDRProblemDescDB

14.47.1 Detailed Description

Handle class for interface specification data.

The DataInterface class is used to provide a memory management handle for the data in DataInterface-Rep. It is populated by IDRProblemDescDB::interface_kwhandler() and is queried by the ProblemDescDB::get_<datatype>() functions. A list of DataInterface objects is maintained in ProblemDescDB::dataInterfaceList, one for each interface specification in an input file.

The documentation for this class was generated from the following files:

- DataInterface.hpp
- DataInterface.cpp

14.48 DataMethod Class Reference

Handle class for method specification data.

Public Member Functions

- DataMethod ()
  constructor
- DataMethod (const DataMethod &)
  copy constructor
- ~DataMethod ()
  destructor
- DataMethod & operator= (const DataMethod &)
  assignment operator
- void write (std::ostream &s) const
  write a DataMethod object to an std::ostream
- void read (MPIUnpackBuffer &s)
  read a DataMethod object from a packed MPI buffer
- void write (MPIPackBuffer &s) const
  write a DataMethod object to a packed MPI buffer
- DataMethodRep * data_rep ()
  return dataMethodRep
Static Public Member Functions

- static bool id_compare (const DataMethod &dm, const std::string &id)
  
  compares the idMethod attribute of DataMethod objects

Private Attributes

- DataMethodRep * dataMethodRep
  
  pointer to the body (handle-body idiom)

Friends

- class ProblemDescDB
- class NIDRProblemDescDB

14.48.1 Detailed Description

Handle class for method specification data.

The DataMethod class is used to provide a memory management handle for the data in DataMethodRep. It is populated by IDRProblemDescDB::method_kwhandler() and is queried by the ProblemDescDB::get_<datatype>() functions. A list of DataMethod objects is maintained in ProblemDescDB::dataMethodList, one for each method specification in an input file.

The documentation for this class was generated from the following files:

- DataMethod.hpp
- DataMethod.cpp

14.49 DataMethodRep Class Reference

Body class for method specification data.

Public Attributes

- String idMethod
  
  string identifier for the method specification data set (from the idMethod specification in MethodIndControl)

- String modelPointer
  
  string pointer to the model specification to be used by this method (from the model.pointer specification in MethodIndControl)

- String lowFidModelPointer
  
  string to point to the low fidelity model for Bayesian experimental design

- short methodOutput
  
  method verbosity control: {SILENT, QUIET, NORMAL, VERBOSE, DEBUG}_OUTPUT (from the output specification in MethodIndControl)

- int maxIterations
  
  maximum number of iterations allowed for the method (from the max.iterations specification in MethodIndControl)

- int maxRefineIterations
maximum number of refinement iterations allowed for a uniform/adaptive refinement approach (from the \texttt{max\_refinement\_iterations} specification in \texttt{MethodIndControl})

- int \texttt{maxSolverIterations}
  maximum number of internal solver iterations allowed for the method (from the \texttt{max\_solver\_iterations} specification in \texttt{MethodIndControl})

- int \texttt{maxFunctionEvaluations}
  maximum number of function evaluations allowed for the method (from the \texttt{max\_function\_evaluations} specification in \texttt{MethodIndControl})

- bool \texttt{speculativeFlag}
  flag for use of speculative gradient approaches for maintaining parallel load balance during the line search portion of optimization algorithms (from the \texttt{speculative} specification in \texttt{MethodIndControl})

- bool \texttt{methodUseDerivsFlag}
  flag for usage of derivative data to enhance the computation of surrogate models (PCE/SC expansions, GP models for EGO/EGRA/EGIE) based on the \texttt{use\_derivatives} specification

- Real \texttt{convergenceTolerance}
  iteration convergence tolerance for the method (from the \texttt{convergence\_tolerance} specification in \texttt{MethodIndControl})

- Real \texttt{constraintTolerance}
  tolerance for controlling the amount of infeasibility that is allowed before an active constraint is considered to be violated (from the \texttt{constraint\_tolerance} specification in \texttt{MethodIndControl})

- bool \texttt{methodScaling}
  flag indicating scaling status (from the \texttt{scaling} specification in \texttt{MethodIndControl})

- \texttt{size\_t numFinalSolutions}
  number of final solutions returned from the iterator

- unsigned short \texttt{methodName}
  the method selection: one of the optimizer, least squares, nond, dace, or parameter study methods

- unsigned short \texttt{subMethod}
  enum value for a sub-method type

- String \texttt{subMethodName}
  string identifier for a sub-method name within a multi-option method specification (e.g., from meta-iterators)

- String \texttt{subModelPointer}
  string pointer for a sub-model specification used by a meta-iterator

- String \texttt{subMethodPointer}
  string pointer for a sub-method specification used by a meta-iterator

- int \texttt{iteratorServers}
  number of servers for concurrent iterator parallelism (from the \texttt{iterator\_servers} specification)

- int \texttt{procsPerIterator}
  number of processors for each concurrent iterator partition (from the \texttt{processors\_per\_iterator} specification)

- short \texttt{iteratorScheduling}
  type of scheduling (\{\texttt{DEFAULT,MASTER,PEER}\} \texttt{SCHEDULING}) used in concurrent iterator parallelism (from the \texttt{iterator\_scheduling} specification)

- StringArray \texttt{hybridMethodNames}
  array of methods for the sequential and collaborative hybrid meta-iterations (from the \texttt{method\_name\_list} specification)
14.49. DATAMETHODREP CLASS REFERENCE

- **StringArray hybridModelPointers**
  array of models for the sequential and collaborative hybrid meta-iterators (from the `model_pointer_list` specification)

- **StringArray hybridMethodPointers**
  array of methods for the sequential and collaborative hybrid meta-iterators (from the `method_pointer_list` specification)

- **String hybridGlobalMethodName**
  global method name for embedded hybrids (from the `global_method_name` specification)

- **String hybridGlobalModelPointer**
  global model pointer for embedded hybrids (from the `global_model_pointer` specification)

- **String hybridGlobalMethodPointer**
  global method pointer for embedded hybrids (from the `global_method_pointer` specification)

- **String hybridLocalMethodName**
  local method name for embedded hybrids (from the `local_method_name` specification)

- **String hybridLocalModelPointer**
  local model pointer for embedded hybrids (from the `local_model_pointer` specification)

- **String hybridLocalMethodPointer**
  local method pointer for embedded hybrids (from the `local_method_pointer` specification)

- **Real hybridLSProb**
  local search probability for embedded hybrids (from the `local_search_probability` specification)

- **int concurrentRandomJobs**
  number of random jobs to perform in the pareto_set and multi_start meta-iterators (from the `random_starts` and `random_weight_sets` specifications)

- **RealVector concurrentParameterSets**
  user-specified (i.e., nonrandom) parameter sets to evaluate in the pareto_set and multi_start meta-iterators (from the `starting_points` and `weight_sets` specifications)

- **unsigned short softConvLimit**
  number of consecutive iterations with change less than convergenceTolerance required to trigger convergence

- **bool surrBasedLocalLayerBypass**
  flag to indicate user-specification of a bypass of any/all layerings in evaluating truth response values in SBL.

- **RealVector trustRegionInitSize**
  initial trust region sizes in the surrogate-based local method (from the `initial_size` specification in MethodSBL), one size per surrogate model (notes: no trust region for the truth model; sizes are relative values, e.g., 0.1 = 10% of range of global bounds for each variable)

- **Real trustRegionMinSize**
  minimum trust region size in the surrogate-based local method (from the `minimum_size` specification in MethodSBL), if the trust region size falls below this threshold the SBL iterations are terminated (note: if kriging is used with SBL, the min trust region size is set to 1.0e-3 in attempt to avoid ill-conditioned matrixes that arise in kriging over small trust regions)

- **Real trustRegionContractTrigger**
  trust region minimum improvement level (ratio of actual to predicted decrease in objective fcn) in the surrogate-based local method (from the `contract_threshold` specification in MethodSBL), the trust region shrinks or is rejected if the ratio is below this value (“eta_1” in the Conn-Gould-Toint trust region book)

- **Real trustRegionExpandTrigger**
  ...
trust region sufficient improvement level (ratio of actual to predicted decrease in objective fn) in the surrogate-based local method (from the expand_threshold specification in MethodSBL), the trust region expands if the ratio is above this value ("eta_2" in the Conn-Gould-Toint trust region book)

- **Real trustRegionContract**
  trust region contraction factor in the surrogate-based local method (from the contraction_factor specification in MethodSBL)

- **Real trustRegionExpand**
  trust region expansion factor in the surrogate-based local method (from the expansion_factor specification in MethodSBL)

- **short surrBasedLocalSubProbObj**
  SBL approximate subproblem objective: ORIGINAL, PRIMARY, SINGLE_OBJECTIVE, LAGRANGIAN_OBJECTIVE, or AUGMENTED_LAGRANGIAN_OBJECTIVE.

- **short surrBasedLocalSubProbCon**
  SBL approximate subproblem constraints: NO_CONSTRAINTS, LINEARIZED_CONSTRAINTS, or ORIGINAL_CONSTRAINTS.

- **short surrBasedLocalMeritFn**
  SBL merit function type: BASIC_PENALTY, ADAPTIVE_PENALTY, BASIC_LAGRANGIAN, or AUGMENTED_LAGRANGIAN.

- **short surrBasedLocalAcceptLogic**
  SBL iterate acceptance logic: TR_RATIO or FILTER.

- **short surrBasedLocalConstrRelax**
  SBL constraint relaxation method: NO_RELAX or HOMOTOPY.

- **bool surrBasedGlobalReplacePts**
  user-specified method for adding points to the set upon which the next surrogate is based in the surrogate-based global method.

- **String dlDetails**
  string of options for a dynamically linked solver

- **void *dlLib**
  handle to dynamically loaded library

- **int verifyLevel**
  the verify_level specification in MethodNPSOLDC

- **Real functionPrecision**
  the function_precision specification in MethodNPSOLDC and the EPSILON specification in NOMAD

- **Real lineSearchTolerance**
  the linesearch_tolerance specification in MethodNPSOLDC

- **Real absConvTol**
  absolute function convergence tolerance

- **Real xConvTol**
  x-convergence tolerance

- **Real singConvTol**
  singular convergence tolerance

- **Real singRadius**
  radius for singular convergence test

- **Real falseConvTol**
  false-convergence tolerance
• Real initTRRadius
  initial trust radius
• int covarianteType
  kind of covariance required
• bool regressDiag
  whether to print the regression diagnostic vector
• String searchMethod
  the search_method specification for Newton and nonlinear interior-point methods in MethodOPTPPDC
• Real gradientTolerance
  the gradient_tolerance specification in MethodOPTPPDC
• Real maxStep
  the max_step specification in MethodOPTPPDC
• short meritFn
  the merit_function specification for nonlinear interior-point methods in MethodOPTPPDC
• Real stepLenToBoundary
  the steplength_to_boundary specification for nonlinear interior-point methods in MethodOPTPPDC
• Real centeringParam
  the centering parameter specification for nonlinear interior-point methods in MethodOPTPPDC
• int searchSchemeSize
  the search_scheme_size specification for PDS methods in MethodOPTPPDC
• Real initStepLength
  the initStepLength choice for nonlinearly constrained APPS in MethodAPPSDC
• Real contractStepLength
  the contractStepLength choice for nonlinearly constrained APPS in MethodAPPSDC
• Real threshStepLength
  the threshStepLength choice for nonlinearly constrained APPS in MethodAPPSDC
• String meritFunction
  the meritFunction choice for nonlinearly constrained APPS in MethodAPPSDC
• Real constrPenalty
  the constrPenalty choice for nonlinearly constrained APPS in MethodAPPSDC
• Real smoothFactor
  the initial smoothFactor value for nonlinearly constrained APPS in MethodAPPSDC
• Real constraintPenalty
  the initial constraint_penalty for COLINY methods in MethodAPPS, MethodSCOLIBDIR, MethodSCOLIBPS, MethodSCOLIBSW and MethodSCOLIBEA
• bool constantPenalty
  the constant_penalty flag for COLINY methods in MethodSCOLIBPS and MethodSCOLIBSW
• Real globalBalanceParam
  the global_balance_parameter for the DIRECT method in MethodSCOLIBDIR
• Real localBalanceParam
  the local_balance_parameter for the DIRECT method in MethodSCOLIBDIR
• Real maxBoxSize
  the max_boxsize_limit for the DIRECT method in MethodSCOLIBDIR
• Real minBoxSize
  the min.boxsize_limit for the DIRECT method in MethodSCOLIBDIR and MethodNCSUDC
• String boxDivision
  the division setting (major_dimension or all_dimensions) for the DIRECT method in MethodSCOLIBDIR
• bool mutationAdaptive
  the non_adaptive specification for the coliny_ea method in MethodSCOLIBEA
• bool showMiscOptions
  the show_misc_options specification in MethodSCOLIBDC
• StringArray miscOptions
  the misc_options specification in MethodSCOLIBDC
• Real solnTarget
  the solution_target specification in MethodSCOLIBDC
• Real crossoverRate
  the crossover_rate specification for EA methods in MethodSCOLIBEA
• Real mutationRate
  the mutation_rate specification for EA methods in MethodSCOLIBEA
• Real mutationScale
  the mutation_scale specification for EA methods in MethodSCOLIBEA
• Real mutationMinScale
  the min_scale specification for mutation in EA methods in MethodSCOLIBEA
• Real initDelta
  the initial_delta specification for APPS/COBYLA/PS/SW methods in MethodAPPS, MethodSCOLIBCOB, MethodSCOLIBPS, and MethodSCOLIBSW
• Real threshDelta
  the variable_tolerance specification for APPS/COBYLA/PS/SW methods in MethodAPPS, MethodSCOLIBCOB, MethodSCOLIBPS, and MethodSCOLIBSW
• Real contractFactor
  the contraction_factor specification for APPS/PS/SW methods in MethodAPPS, MethodSCOLIBPS, and MethodSCOLIBSW
• int newSolnsGenerated
  the new_solutions_generated specification for GA/EPSA methods in MethodSCOLIBEA
• int numberRetained
  the integer assignment to random, chc, or elitist in the replacement_type specification for GA/EPSA methods in MethodSCOLIBEA
• bool expansionFlag
  the no_expansion specification for APPS/PS/SW methods in MethodAPPS, MethodSCOLIBPS, and MethodSCOLIBSW
• int expandAfterSuccess
  the expand_after_success specification for PS/SW methods in MethodSCOLIBPS and MethodSCOLIBSW
• int contractAfterFail
  the contract_after_failure specification for the SW method in MethodSCOLIBSW
• int mutationRange
  the mutation_range specification for the pga_int method in MethodSCOLIBEA
• int totalPatternSize
  the total_pattern_size specification for PS methods in MethodSCOLIBPS
• bool randomizeOrderFlag
  the stochastic specification for the PS method in MethodSCOLIBPS
• String selectionPressure
  the fitness_type specification for EA methods in MethodSCOLIBEA
• String replacementType
  the replacement_type specification for EA methods in MethodSCOLIBEA
• String crossoverType
  the crossover_type specification for EA methods in MethodSCOLIBEA
• String mutationType
  the mutation_type specification for EA methods in MethodSCOLIBEA
• String exploratoryMoves
  the exploratory_moves specification for the PS method in MethodSCOLIBPS
• String patternBasis
  the pattern_basis specification for APPS/PS methods in MethodAPPS and MethodSCOLIBPS
• String betaSolverName
  beta solvers don’t need documentation
• String evalSynchronize
  the synchronization setting for parallel pattern search methods in MethodSCOLIBPS and MethodAPPS
• size_t numCrossPoints
  The number of crossover points or multi-point schemes.
• size_t numParents
  The number of parents to use in a crossover operation.
• size_t numOffspring
  The number of children to produce in a crossover operation.
• String fitnessType
  the fitness assessment operator to use.
• String convergenceType
  The means by which this JEGA should converge.
• Real percentChange
  The minimum percent change before convergence for a fitness tracker converger.
• size_t numGenerations
  The number of generations over which a fitness tracker converger should track.
• Real fitnessLimit
  The cutoff value for survival in fitness limiting selectors (e.g., below_limit selector).
• Real shrinkagePercent
  The minimum percentage of the requested number of selections that must take place on each call to the selector (0, 1).
• String nichingType
  The niching type.
• RealVector nicheVector
  The discretization percentage along each objective.
• size_t numDesigns
  The maximum number of designs to keep when using the max_designs nicher.
• String postProcessorType
  The post processor type.
• RealVector distanceVector
  The discretization percentage along each objective.
• String initializationType
  The means by which the JEGA should initialize the population.
• String flatFile
  The filename to use for initialization.
• String logFile
  The filename to use for logging.
• int populationSize
  The population size specification for GA methods in MethodSCOLIBEA
• bool printPopFlag
  The print each pop flag to set the printing of the population at each generation.
• Real volBoxSize
  the volume boxsize limit for the DIRECT method in MethodNCSUDC
• int numSymbols
  the symbols specification for DACE methods
• bool mainEffectsFlag
  the main effects specification for sampling methods in MethodDDACE)
• bool latinizeFlag
  the latinize specification for FSU QMC and CVT methods in MethodFSUDACE
• bool volQualityFlag
  the quality metrics specification for sampling methods (FSU QMC and CVT methods in MethodFSUDACE)
• IntVector sequenceStart
  the sequenceStart specification in MethodFSUDACE
• IntVector sequenceLeap
  the sequenceLeap specification in MethodFSUDACE
• IntVector primeBase
  the primeBase specification in MethodFSUDACE
• int numTrials
  the numTrials specification in MethodFSUDACE
• String trialType
  the trial type specification in MethodFSUDACE
• int randomSeed
  the seed specification for COLINY, NonD, & DACE methods
• SizetArray randomSeedSeq
  the seed sequence specification for multilevel UQ methods
• Real initMeshSize
  the initMeshSize choice for NOMAD in MethodNOMADDC
• Real minMeshSize
the minMeshSize choice for NOMAD in MethodNOMADDCC

- String historyFile
  the HISTORY_FILE specification for NOMAD

- String displayFormat
  the DISPLAY_STATS specification for NOMAD

- Real vns
  the VNS specification for NOMAD

- int neighborOrder
  the NEIGHBOR_ORDER specification for NOMAD

- bool showAllEval
  the DISPLAY_ALL_EVAL specification for NOMAD

- String useSurrogate
  the HAS_SGTE specification for NOMAD

- int maxCrossIterations
  maximum number of cross iterations

- Real solverTol
  optimization tolerance for FT regression

- Real roundingTol
  Rounding tolerance for FT regression.

- Real arithmeticTol
  arithmetic (rounding) tolerance for FT sums and products

- unsigned short startOrder
  starting polynomial order

- unsigned short maxOrder
  maximum order of basis polynomials

- size_t startRank
  starting rank

- size_t kickRank
  rank increase increment

- size_t maxRank
  maximum rank

- bool adaptRank
  whether or not to adapt rank

- UShortArray startOrderSeq
  starting polynomial order

- SizetArray startRankSeq
  starting rank

- int numSamples
  the samples specification for NonD & DACE methods

- bool fixedSeedFlag
  flag for fixing the value of the seed among different NonD/DACE sample sets. This results in the use of the same sampling stencil/pattern throughout an execution with repeated sampling.

- bool fixedSequenceFlag
Flag for fixing the sequence for Halton or Hammersley QMC sample sets. This results in the use of the same sampling stencil/pattern throughout an execution with repeated sampling.

- **bool vbdFlag**
  *the var_based_decomp specification for a variety of sampling methods*

- **Real vbdDropTolerance**
  *the var_based_decomp tolerance for omitting index output*

- **bool backfillFlag**
  *the backfill option allows one to augment in LHS sample by enforcing the addition of unique discrete variables to the sample*

- **bool pcaFlag**
  *Flag to specify the calculation of principal components when using LHS.*

- **Real percentVarianceExplained**
  *The percentage of variance explained by using a truncated number of principal components in PCA.*

- **bool wilksFlag**
  *Flag to specify use of Wilks formula to calculate num samples.*

- **unsigned short wilksOrder**
  *Wilks order parameter.*

- **Real wilksConfidenceLevel**
  *Wilks confidence interval parameter.*

- **short wilksSidedInterval**
  *Wilks sided interval type.*

- **unsigned short vbdOrder**
  *a sub-specification of vbdFlag: interaction order limit for calculation/output of component VBD indices*

- **short covarianceControl**
  *restrict the calculation of a full response covariance matrix for high dimensional outputs: {DEFAULT,DIAGONAL,FULL}_COVARIANCE*

- **String rngName**
  *the basic random-number generator for NonD*

- **short refinementType**
  *refinement type for stochastic expansions from dimension refinement keyword group*

- **short refinementControl**
  *refinement control for stochastic expansions from dimension refinement keyword group*

- **short nestingOverride**
  *override for default point nesting policy: NO_NESTING_OVERRIDE, NESTED, or NON_NESTED*

- **short growthOverride**
  *override for default point growth restriction policy: NO_GROWTH_OVERRIDE, RESTRICTED, or UNRESTRICTED*

- **short expansionType**
  *enumeration for u-space type that defines u-space variable targets for probability space transformations: EXTENDED_U (default), ASKEY_U, PARTIALASKEY_U, STD_NORMAL_U, or STD_UNIFORM_U*

- **bool piecewiseBasis**
  *boolean indicating presence of piecewise keyword*

- **short expansionBasisType**
enumeration for type of basis in sparse grid interpolation (Pecos::{NODAL,HIERARCHICAL},INTERPOLANT) or regression (Pecos::{TENSOR_PRODUCT,TOTAL_ORDER,ADAPTED},BASIS).

- **UShortArray** quadratureOrderSeq
  
  _the quadrature order sequence specification in MethodNonDPCE and MethodNonDSC_

- **UShortArray** sparseGridLevelSeq
  
  _the sparse grid level specification in MethodNonDPCE and MethodNonDSC_

- **UShortArray** expansionOrderSeq
  
  _the expansion order sequence specification in MethodNonDPCE_

- **SizetArray** collocationPointsSeq
  
  _the collocation points sequence specification in MethodNonDPCE_

- **SizetArray** expansionSamplesSeq
  
  _the expansion samples sequence specification in MethodNonDPCE_

- **unsigned short** quadratureOrder
  
  _the quadrature order specification in MethodNonDPCE and MethodNonDSC_

- **unsigned short** sparseGridLevel
  
  _the sparse grid level specification in MethodNonDPCE and MethodNonDSC_

- **unsigned short** expansionOrder
  
  _the expansion order specification in MethodNonDPCE_

- **size_t** collocationPoints
  
  _the collocation points specification in MethodNonDPCE_

- **size_t** expansionSamples
  
  _the expansion samples specification in MethodNonDPCE_

- **RealVector** anisoDimPref
  
  _the dimension preference specification for tensor and sparse grids and expansion orders in MethodNonDPCE and MethodNonDSC_

- **unsigned short** cubIntOrder
  
  _the cubature integrand specification in MethodNonDPCE_

- **Real** collocationRatio
  
  _the collocation ratio specification in MethodNonDPCE_

- **Real** collocRatioTermsOrder
  
  _order applied to the number of expansion terms when applying or computing the collocation ratio within regression PCE; based on the ratio order specification in MethodNonDPCE_

- **short** regressionType
  
  _type of regression: LS, OMP, BP, BPDN, LARS, or LASSO_

- **short** lsRegressionType
  
  _type of least squares regression: SVD or EQ,QR_

- **RealVector** regressionNoiseTol
  
  _noise tolerance(s) for OMP, BPDN, LARS, and LASSO_

- **Real** regressionL2Penalty
  
  _L2 regression penalty for a variant of LASSO known as the elastic net method (default of 0 gives standard LASSO)_

- **bool** crossValidation
  
  _flag indicating the use of cross-validation across expansion orders (given a prescribed maximum order) and, for some methods, noise tolerances_

- **bool** crossValidNoiseOnly
flag indicating the restriction of cross-validation to estimate only the most effective noise tolerance; used to reduce cost from performing CV over both noise tolerances and expansion orders

- unsigned short adaptedBasisAdvancements
  initial grid level for the ADAPTED_BASIS_EXPANDING_FRONT approach to defining the candidate basis for sparse recovery (compressed sensing)

- bool normalizedCoeffs
  flag indicating the output of PCE coefficients corresponding to normalized basis polynomials

- String pointReuse
  allows PCE construction to reuse points from previous sample sets or data import using the reuse_points specification in MethodNonDPCE

- bool tensorGridFlag
  flag for usage of a sub-sampled set of tensor-product grid points within regression PCE; based on the tensor_grid specification in MethodNonDPCE

- UShortArray tensorGridOrder
  order of tensor-product grid points that are sub-sampled within orthogonal least interpolation PCE; based on the tensor_grid specification in MethodNonDPCE

- String importExpansionFile
  the import_expansion_file specification in MethodNonDPCE

- String exportExpansionFile
  the export_expansion_file specification in MethodNonDPCE

- unsigned short sampleType
  the sample_type specification in MethodNonDMC, MethodNonDPCE, and MethodNonDSC

- bool dOptimal
  whether to generate D-optimal designs

- size_t numCandidateDesigns
  number of candidate designs in D-optimal design selection

- unsigned short reliabilitySearchType
  the type of limit state search in MethodNonDLocalRel (x_taylor_mean, x_taylor_mpp, x_two_point, u_taylor_mean, u_taylor_mpp, u_two_point, or no_approx) or MethodNonDGlobalRel (x_gaussian_process or u_gaussian_process)

- String reliabilityIntegration
  the first_order or second_order integration selection in MethodNonDLocalRel

- unsigned short integrationRefine
  the import, adapt_import, or mm_adapt_import integration refinement selection in MethodNonDLocalRel, MethodNonDPCE, and MethodNonDSC

- IntVector refineSamples
  Sequence of refinement samples, e.g., the size of the batch (e.g. number of supplemental points added) to be added to the build points for an emulator at each iteration.

- SizetArray pilotSamples
  the pilot_samples selection in MethodMultilevelMC

- short allocationTarget
  the allocationTarget selection in MethodMultilevelMC

- bool useTargetVarianceOptimizationFlag
  the useTargetVarianceOptimizationFlag selection in MethodMultilevelMC

- short qoiAggregation
the qoi aggregation norm selection in MethodMultilevelMC

- short multilevAllocControl
  the allocation control selection in MethodMultilevelPCE
- Real multilevEstimatorRate
  the estimator rate selection in MethodMultilevelPCE
- short multilevDiscrepEmulation
  type of discrepancy emulation in multilevel methods: distinct or recursive
- short finalMomentsType
  the final moments specification in MethodNonD
- short distributionType
  the distribution cumulative or complementary specification in MethodNonD
- short responseLevelTarget
  the compute probabilities, reliabilities, or gen_reliabilities specification in MethodNonD
- short responseLevelTargetReduce
  the system series or parallel specification in MethodNonD
- RealVectorArray responseLevels
  the response levels specification in MethodNonD
- RealVectorArray probabilityLevels
  the probability levels specification in MethodNonD
- RealVectorArray reliabilityLevels
  the reliability levels specification in MethodNonD
- RealVectorArray genReliabilityLevels
  the gen_reliability levels specification in MethodNonD
- int chainSamples
  the number of MCMC chain samples
- int buildSamples
  the number of samples to construct an emulator, e.g., for Bayesian calibration methods
- int samplesOnEmulator
  number of samples to perform on emulator
- int emulatorOrder
  The total order to be used in construction of a VPS surrogate.
- short emulatorType
  the emulator specification in MethodNonDBayesCalib
- String mcmcType
  the mcmc type specification in MethodNonDBayesCalib
- bool standardizedSpace
  use of standardized probability spaces for MCMC within Bayesian inference
- bool adaptPosteriorRefine
  flag indicating adaptive refinement of the emulator in regions of high posterior probability
- bool logitTransform
  flag indicating user activation of logit transform option within QUESO
- bool gpmsaNormalize
  whether to apply GPMSA-internal normalization
- bool posteriorStatsKL
  flag indicating the calculation of KL divergence between prior and posterior in Bayesian methods
- bool posteriorStatsMutual
  flag indicating the calculation of mutual information between prior and posterior in Bayesian methods
- bool posteriorStatsKDE
  flag indicating calculation of kernel density estimate of posterior distributions
- bool chainDiagnostics
  flag indicating calculation of chain diagnostics
- bool chainDiagnosticsCI
  flag indicating calculation of confidence intervals as a chain diagnostic
- bool modelEvidence
  flag indicating calculation of the evidence of the model
- bool modelEvidMC
  flag indicating use of Monte Carlo approximation for evidence calc.
- int evidenceSamples
  number of prior samples to use in model evidence calculation
- bool modelEvidLaplace
  flag indicating use of Laplace approximation for evidence calc.
- unsigned short preSolveMethod
  the method used for performing a pre-solve for the MAP point
- String proposalCovType
  the type of proposal covariance: user, derivatives, or prior
- Real priorPropCovMult
  optional multiplier for prior-based proposal covariance
- int proposalCovUpdatePeriod
  number of samples after which to update the proposal covariance from misfit Hessian (using residual values and derivatives)
- String proposalCovInputType
  the format of proposal covariance input: diagonal or matrix
- RealVector proposalCovData
  raw list of real data for the proposal covariance
- String proposalCovFile
  file from which to read proposal covariance in diagonal or matrix format
- String advancedOptionsFilename
  file containing advanced ROL option overrides
- String quesoOptionsFilename
  file containing advanced QUESO option overrides
- String fitnessMetricType
  the fitness metric type specification in MethodNonDAdaptive
- String batchSelectionType
  the batch selection type specification in MethodNonDAdaptive
- String lipschitzType
  the Lipschitz type specification in MethodNonDPOFDarts (e.g. either local or global estimation)
• unsigned short calibrateErrorMode
  calibration mode for observation error multipliers (CALIBRATE_*)
• RealVector hyperPriorAlphas
  hyperparameters inverse gamma prior alphas
• RealVector hyperPriorBetas
  hyperparameters inverse gamma prior alphas
• int burnInSamples
  number of MCMC samples to discard from acceptance chain
• int subSamplingPeriod
  period or skip in post-processing the acceptance chain
• bool calModelDiscrepancy
  flag to calculate model discrepancy
• size_t numPredConfigs
  number of prediction configurations at which to calculate model discrepancy
• RealVector predictionConfigList
  list of prediction configurations at which to calculate model discrepancy
• String importPredConfigs
  whether to import prediction configurations at which to calculate model discrepancy
• unsigned short importPredConfigFormat
  tabular format for prediction configurations import file
• String modelDiscrepancyType
  type of model discrepancy emulation
• short approxCorrectionOrder
  correction order for either gaussian process or polynomial model discrepancy calculations: 0 (=constant), 1 (=linear), 2 (=quadratic)
• String exportCorrModelFile
  specify the name of file to which corrected model (model+discrepancy) calculations are output
• unsigned short exportCorrModelFormat
  tabular format for corrected model (model+discrepancy) export file
• String exportCorrVarFile
  specify the name of file to which corrected model variance calculations are output
• unsigned short exportCorrVarFormat
  tabular format for corrected model variance export file
• String exportDiscrepFile
  specify the name of file to which discrepancy calculations are output
• unsigned short exportDiscrepFormat
  tabular format for model discrepancy export file
• bool adaptExpDesign
  whether to perform adaptive Bayesian design of experiments
• String importCandPtsFile
  whether to import candidate design points for adaptive Bayesian experimental design
• unsigned short importCandFormat
  tabular format for the candidate design points import file
• size_t numCandidates
  number of candidate designs for adaptive Bayesian experimental design
• int maxHifiEvals
  maximum number of highfidelity model runs to be used for adaptive Bayesian experimental design
• int batchSize
  number of optimal designs selected per iteration of experimental design algorithm; also number of concurrent GP refinement points for EGO
• int batchSizeExplore
  portion of batchSize earmarked for exploration rather than acquisition
• bool mutualInfoKSG2
  indicate that the KSG2 algorithm is to be employed in the calculation of the mutual information
• int numChains
  number of concurrent chains
• int numCR
  number of CR-factors
• int crossoverChainPairs
  number of crossover chain pairs
• Real grThreshold
  threshold for the Gelman-Rubin statistic
• int jumpStep
  how often to perform a long jump in generations
• int numPushforwardSamples
  Number of samples from the prior that is pushed forward through the model to obtain the initial set of pushforward samples.
• String dataDistType
  the type of data distribution: kde, or gaussian
• String dataDistCovInputType
  the format of data distribution gaussian covariance input: diagonal or matrix
• RealVector dataDistMeans
  raw list of real data for the data distribution gaussian means
• RealVector dataDistCovariance
  raw list of real data for the data distribution gaussian covariance
• String dataDistFile
  file from which to read data distribution data (covariance or samples)
• String posteriorDensityExportFilename
  The filename of the export file containing an arbitrary set of samples and their corresponding density values.
• String posteriorSamplesExportFilename
  The filename of the export file containing samples from the posterior and their corresponding density values.
• String posteriorSamplesImportFilename
  The filename of the import file containing samples at which the posterior will be evaluated.
• bool generatePosteriorSamples
  Flag specifying whether to generate random samples from the posterior.
• bool evaluatePosteriorDensity
Flag specifying whether to evaluate the posterior density at a set of samples.

- RealVector `finalPoint`
  the final point specification in `MethodPSVPS`

- RealVector `stepVector`
  the step vector specification in `MethodPSVPS` and `MethodPSCPS`

- int `numSteps`
  the num steps specification in `MethodPSVPS`

- IntVector `stepsPerVariable`
  the deltas per variable specification in `MethodPSCPS`

- RealVector `listOfPoints`
  the list of points specification in `MethodPSLPS`

- String `pstudyFilename`
  the import points file spec for a file-based parameter study

- unsigned short `pstudyFileFormat`
  tabular format for the parameter study points file

- bool `pstudyFileActive`
  whether to import active variables only

- UShortArray `varPartitions`
  the partitions specification for `PStudy` method in `MethodPSMPS`

- Real `refinementRate`
  rate of mesh refinement in Richardson extrapolation

- String `importBuildPtsFile`
  the file name from the import build points file specification

- unsigned short `importBuildFormat`
  tabular format for the build point import file

- bool `importBuildActive`
  whether to import active variables only

- String `importApproxPtsFile`
  the file name from the import approx points file specification

- unsigned short `importApproxFormat`
  tabular format for the approx point import file

- bool `importApproxActive`
  whether to import active variables only

- String `exportApproxPtsFile`
  the file name from the export approx points file specification

- unsigned short `exportApproxFormat`
  tabular format for the approx point export file

- String `exportMCMCPtsFile`
  the file name from the export mcmc points file specification

- bool `exportSampleSeqFlag`
  flag for exporting the sequence of sample increments within multilevel sampling from the export sample sequence specification

- unsigned short `exportSamplesFormat`
  tabular format for the MCMC chain and MLMC sample sequence exports
Private Member Functions

- **DataMethodRep ()**
  
  constructor
- **~DataMethodRep ()**
  
  destructor
- void **write (std::ostream &s) const**
  write a DataInterfaceRep object to an std::ostream
- void **read (MPIUnpackBuffer &s)**
  read a DataInterfaceRep object from a packed MPI buffer
- void **write (MPIPackBuffer &s) const**
  write a DataInterfaceRep object to a packed MPI buffer

Private Attributes

- int **referenceCount**
  number of handle objects sharing this dataMethodRep

Friends

- class **DataMethod**
  the handle class can access attributes of the body class directly

14.49.1 Detailed Description

Body class for method specification data.

The DataMethodRep class is used to contain the data from a method keyword specification. Default values are managed in the DataMethodRep constructor. Data is public to avoid maintaining set/get functions, but is still encapsulated within ProblemDescDB since ProblemDescDB::dataMethodList is private.

The documentation for this class was generated from the following files:

- DataMethod.hpp
- DataMethod.cpp

14.50 DataModel Class Reference

Handle class for model specification data.

Public Member Functions

- **DataModel ()**
  
  constructor
- **DataModel (const DataModel &)**
  copy constructor
- **~DataModel ()**
  destructor
- **DataModel & operator= (const DataModel &)**
14.51. DATAMODELREP CLASS REFERENCE

assignment operator

1. void write (std::ostream &s) const
   write a DataModel object to an std::ostream
2. void read (MPIUnpackBuffer &s)
   read a DataModel object from a packed MPI buffer
3. void write (MPIPackBuffer &s) const
   write a DataModel object to a packed MPI buffer
4. DataModelRep * data_rep ()
   return dataModelRep

Static Public Member Functions

1. static bool id_compare (const DataModel &dm, const std::string &id)
   compares the idModel attribute of DataModel objects

Private Attributes

1. DataModelRep * dataModelRep
   pointer to the body (handle-body idiom)

Friends

1. class ProblemDescDB
2. class NIDRProblemDescDB

14.50.1 Detailed Description

Handle class for model specification data.

The DataModel class is used to provide a memory management handle for the data in DataModelRep. It is populated by IDRProblemDescDB::model_kwhandler() and is queried by the ProblemDescDB::get_<datatype>() functions. A list of DataModel objects is maintained in ProblemDescDB::dataModelList, one for each model specification in an input file.

The documentation for this class was generated from the following files:

1. DataModel.hpp
2. DataModel.cpp

14.51 DataModelRep Class Reference

Body class for model specification data.

Public Attributes

1. String idModel
   string identifier for the model specification data set (from the id_model specification in ModelIndControl)
2. String modelType
   model type selection: single, surrogate, or nested (from the model type specification in ModelIndControl)
• String $\text{variablesPointer}$
  
  string pointer to the variables specification to be used by this model (from the $\text{variables\_pointer}$ specification in ModelIndControl)

• String $\text{interfacePointer}$
  
  string pointer to the interface specification to be used by this model (from the $\text{interface\_pointer}$ specification in ModelSingle and the optional $\text{interface\_pointer}$ specification in ModelNested)

• String $\text{responsesPointer}$
  
  string pointer to the responses specification to be used by this model (from the $\text{responses\_pointer}$ specification in ModelIndControl)

• bool $\text{hierarchicalTags}$
  
  whether this model and its children will add hierarchy-based tags to eval ids

• String $\text{subMethodPointer}$
  
  pointer to a sub-iterator used for global approximations (from the $\text{dace\_method\_pointer}$ specification in ModelSurrG) or by nested models (from the $\text{sub\_method\_pointer}$ specification in ModelNested)

• RealVector $\text{solutionLevelCost}$
  
  array of relative simulation costs corresponding to each of the solution levels (from the $\text{solution\_level\_cost}$ specification; see also $\text{solution\_level\_control}$ specification; for defining a secondary hierarchy of fidelity within the scope of a single model form (from the $\text{solution\_level\_control}$ specification; see also $\text{ordered\_model\_fidelities}$)

• IntSet $\text{surrogateFnIndices}$
  
  array specifying the response function set that is approximated

• String $\text{surrogateType}$
  
  the selected surrogate type: local_taylor, multipoint_tana, global_neural_network, mars, orthogonal_polynomial, gaussian, polynomial, kriging, or hierarchical

• String $\text{actualModelPointer}$
  
  pointer to the model specification for constructing the truth model used in constructing surrogates (from the $\text{actual\_model\_pointer}$ specification in ModelSurrL and ModelSurrMP)

• StringArray $\text{orderedModelPointers}$
  
  an ordered list of model pointers (low to high) corresponding to a hierarchy of modeling fidelity (from the $\text{ordered\_model\_fidelities}$ specification in ModelSurrH)

• int $\text{pointsTotal}$
  
  user-specified lower bound on total points with which to build the model (if $\text{reuse\_points} < \text{pointsTotal}$, new samples will make up the difference)

• short $\text{pointsManagement}$
  
  points management configuration for DataFitSurrModel: DEFAULT_POINTS, MINIMUM_POINTS, or RECOMMENDED_POINTS

• String $\text{approxPointReuse}$
  
  sample reuse selection for building global approximations: none, all, region, or file (from the $\text{reuse\_samples}$ specification in ModelSurrG)

• String $\text{importBuildPtsFile}$
  
  the file name from the $\text{import\_build\_points\_file}$ specification in ModelSurrG

• unsigned short $\text{importBuildFormat}$
  
  tabular format for the build point import file
• bool importUseVariableLabels
  whether to parse/validate variable labels from header
• bool importBuildActive
  whether to import active variables only
• String exportApproxPtsFile
  the file name from the export_approx_points_file specification in ModelSurrG
• unsigned short exportApproxFormat
  tabular format for the approx point export file
• bool exportSurrogate
  Option to turn on surrogate model export (export_model)
• String modelExportPrefix
  the filename prefix for export_model
• unsigned short modelExportFormat
  Format selection for export_model.
• short approxCorrectionType
  correction type for global and hierarchical approximations: NO_CORRECTION, ADDITIVE_CORRECTION, MULTIPLICATIVE_CORRECTION, or COMBINED_CORRECTION (from the correction specification in ModelSurrG and ModelSurrH)
• short approxCorrectionOrder
  correction order for global and hierarchical approximations: 0, 1, or 2 (from the correction specification in ModelSurrG and ModelSurrH)
• bool modelUseDerivsFlag
  flags the use of derivatives in building global approximations (from the use_derivatives specification in ModelSurrG)
• short polynomialOrder
  scalar integer indicating the order of the polynomial approximation (1=linear, 2=quadratic, 3=cubic; from the polynomial specification in ModelSurrG)
• RealVector krigingCorrelations
  vector of correlations used in building a kriging approximation (from the correlations specification in ModelSurrG)
• String krigingOptMethod
  optimization method to use in finding optimal correlation parameters: none, sampling, local, global
• short krigingMaxTrials
  maximum number of trials in optimization of kriging correlations
• RealVector krigingMaxCorrelations
  upper bound on kriging correlation vector
• RealVector krigingMinCorrelations
  lower bound on kriging correlation vector
• Real krigingNugget
  nugget value for kriging
• short krigingFindNugget
  option to have Kriging find the best nugget value to use
• short mlsWeightFunction
  weight function for moving least squares approximation
• short rbfBases
  bases for radial basis function approximation
• short rbfMaxPts
  maximum number of points for radial basis function approximation
• short rbfMaxSubsets
  maximum number of subsets for radial basis function approximation
• short rbfMinPartition
  minimum partition for radial basis function approximation
• short marsMaxBases
  maximum number of bases for MARS approximation
• String marsInterpolation
  interpolation type for MARS approximation
• short annRandomWeight
  random weight for artificial neural network approximation
• short annNodes
  number of nodes for artificial neural network approximation
• Real annRange
  range for artificial neural network approximation
• int numRestarts
  number of restarts for gradient-based optimization in GP
• bool domainDecomp
  whether domain decomposition is enabled
• String decompCellType
  type of local cell of domain decomp
• int decompSupportLayers
  number of support layers for each local basis function
• bool decompDiscontDetect
  whether discontinuity detection is enabled
• Real discontJumpThresh
  function value (jump) threshold for discontinuity detection in domain decomp
• Real discontGradThresh
  gradient threshold for discontinuity detection in domain decomp
• String trendOrder
  scalar integer indicating the order of the Gaussian process mean (0= constant, 1=linear, 2=quadratic, 3=cubic); from the gaussian process specification in ModelSurrG)
• bool pointSelection
  flag indicating the use of point selection in the Gaussian process
• StringArray diagMetrics
  List of diagnostic metrics the user requests to assess the goodness of fit for a surrogate model.
• bool crossValidateFlag
  flag indicating the use of cross validation on the metrics specified
• int numFolds
  number of folds to perform in cross validation
14.51. DATAMODELREP CLASS REFERENCE

- **Real** percentFold
  - percentage of data to withhold for cross validation process
- **bool** pressFlag
  - flag indicating the use of PRESS on the metrics specified
- **String** importChallengePtsFile
  - the file name from the challenge_points_file specification in ModelSurrG
- **unsigned short** importChallengeFormat
  - tabular format of the challenge data file
- **bool** importChalUseVariableLabels
  - whether to parse/validate variable labels from header
- **bool** importChallengeActive
  - whether to import active variables only
- **String** advancedOptionsFilename
  - file containing advanced surrogate option overrides
- **String** optionalInterfRespPointer
  - string pointer to the responses specification used by the optional interface in nested models (from the optional_interface_responses_pointer specification in ModelNested)
- **StringArray** primaryVarMaps
  - the primary variable mappings used in nested models for identifying the lower level variable targets for inserting top level variable values (from the primary_variable_mapping specification in ModelNested)
- **StringArray** secondaryVarMaps
  - the secondary variable mappings used in nested models for identifying the (distribution) parameter targets within the lower level variables for inserting top level variable values (from the secondary_variable_mapping specification in ModelNested)
- **RealVector** primaryRespCoeffs
  - the primary response mapping matrix used in nested models for weighting contributions from the sub-iterator responses in the top level (objective) functions (from the primary_response_mapping specification in ModelNested)
- **RealVector** secondaryRespCoeffs
  - the secondary response mapping matrix used in nested models for weighting contributions from the sub-iterator responses in the top level (constraint) functions (from the secondary_response_mapping specification in ModelNested)
- **bool** identityRespMap
  - whether an identity response map is requested in lieu of explicit maps
- **int** subMethodServers
  - number of servers for concurrent sub-iterator parallelism
- **int** subMethodProcs
  - number of processors for each concurrent sub-iterator partition
- **short** subMethodScheduling
  - scheduling approach for concurrent sub-iterator parallelism: {DEFAULT,MASTER,PEER}_SCHEDULING
- **int** initialSamples
  - initial samples to build the subspace model
- **unsigned short** subspaceSampleType
  - sampling method for building the subspace model
- **IntVector** refineSamples
refinement samples to add in each batch

- int maxIterations
  maximum number of subspace build iterations
- Real convergenceTolerance
  convergence tolerance on build process
- bool subspaceIdBingLi
  Flag to use Bing Li method to identify active subspace dimension.
- bool subspaceIdConstantine
  Flag to use Constantine method to identify active subspace dimension.
- bool subspaceIdEnergy
  Flag to use eigenvalue energy method to identify active subspace dimension.
- bool subspaceBuildSurrogate
  Flag to build surrogate over active subspace.
- int subspaceDimension
  Size of subspace.
- unsigned short subspaceNormalization
  Normalization to use when forming a subspace with multiple response functions.
- int numReplicates
  Number of bootstrap samples for subspace identification.
- bool subspaceIdCV
  Flag to use cross validation to identify active subspace dimension.
- Real relTolerance
  relative tolerance used by cross validation subspace dimension id method
- Real decreaseTolerance
  decrease tolerance used by cross validation subspace dimension id method
- int subspaceCVMaxRank
  maximum rank considered by cross validation subspace dimension id method
- bool subspaceCVIncremental
  flag to use incremental dimension estimation in the cross validation metric
- unsigned short subspaceIdCVMethod
  Contains which cutoff method to use in the cross validation metric.
- short regressionType
  type of (regularized) regression: FT_LS or FT_RLS2
- Real regressionL2Penalty
  penalty parameter for regularized regression (FT_RLS2)
- int maxSolverIterations
  max iterations for optimization solver used in FT regression
- int maxCrossIterations
  maximum number of cross iterations
- Real solverTol
  optimization tolerance for FT regression
- Real roundingTol
  Rounding tolerance for FT regression.
• Real `arithmeticTol`
  arithmetic (rounding) tolerance for FT sums and products
• bool `tensorGridFlag`
  sub-sample a tensor grid for generating regression data
• unsigned short `startOrder`
  starting polynomial order
• unsigned short `maxOrder`
  maximum order of basis polynomials
• size_t `startRank`
  starting rank
• size_t `kickRank`
  rank increase increment
• size_t `maxRank`
  maximum rank
• bool `adaptRank`
  whether or not to adapt rank
• size_t `collocationPoints`
  number of data points used in FT construction by regression
• Real `collocationRatio`
  ratio of number of points to number of unknowns
• short `refinementType`
  type of adaptive refinement (p-, h-, hp-)
• short `refinementControl`
  type of control for adaptive refinement (uniform, anisotropic, ...)
• bool `autoRefine`
  whether automatic surrogate refinement is enabled
• int `maxFunctionEvals`
  maximum evals in refinement
• String `refineCVMetric`
  metric to use in cross-validation guided refinement
• int `softConvergenceLimit`
  max number of iterations in refinement without improvement
• int `refineCVFolds`
  number of cross-validation folds in guided refinement
• unsigned short `adaptedBasisSparseGridLev`
  sparse grid level for low-order PCE used to compute rotation matrix within adapted basis approach to dimension reduction
• unsigned short `adaptedBasisExpOrder`
  expansion order for low-order PCE used to compute rotation matrix within adapted basis approach to dimension reduction
• Real `adaptedBasisCollocRatio`
  collocation ratio for low-order PCE used to compute rotation matrix within adapted basis approach to dimension reduction
• unsigned short `randomFieldIdForm`
Contains which type of random field model.

- unsigned short **analyticCovIdForm**
  Contains which type of analytic covariance function.
- Real **truncationTolerance**
  truncation tolerance on build process: percent variance explained
- String **propagationModelPointer**
  pointer to the model through which to propagate the random field
- String **rfDataFileName**
  File from which to build the random field.

Private Member Functions

- **DataModelRep ()**
  constructor
- **~DataModelRep ()**
  destructor
- void **write (std::ostream &s) const**
  write a DataModelRep object to an std::ostream
- void **read (MPIUnpackBuffer &s)**
  read a DataModelRep object from a packed MPI buffer
- void **write (MPIPackBuffer &s) const**
  write a DataModelRep object to a packed MPI buffer

Private Attributes

- int **referenceCount**
  number of handle objects sharing this dataModelRep

Friends

- class **DataModel**
  the handle class can access attributes of the body class directly

14.51.1 Detailed Description

Body class for model specification data.

The **DataModelRep** class is used to contain the data from a model keyword specification. Default values are managed in the **DataModelRep** constructor. Data is public to avoid maintaining set/get functions, but is still encapsulated within **ProblemDescDB** since **ProblemDescDB::dataModelList** is private.

The documentation for this class was generated from the following files:

- DataModel.hpp
- DataModel.cpp

14.52 DataResponses Class Reference

Handle class for responses specification data.
Public Member Functions

- **DataResponses ()**
  constructor
- **DataResponses (const DataResponses &)**
  copy constructor
- **∼DataResponses ()**
  destructor
- **DataResponses & operator= (const DataResponses &)**
  assignment operator
- **void write (std::ostream &s) const**
  write a DataResponses object to an std::ostream
- **void read (MPIUnpackBuffer &s)**
  read a DataResponses object from a packed MPI buffer
- **void write (MPIPackBuffer &s) const**
  write a DataResponses object to a packed MPI buffer
- **DataResponsesRep * data_rep ()**
  return dataRespRep

Static Public Member Functions

- **static bool id_compare (const DataResponses &dr, const std::string &id)**
  compares the idResponses attribute of DataResponses objects

Private Attributes

- **DataResponsesRep * dataRespRep**
  pointer to the body (handle-body idiom)

Friends

- class ProblemDescDB
- class NIDRProblemDescDB

14.52.1 Detailed Description

Handle class for responses specification data.

The DataResponses class is used to provide a memory management handle for the data in DataResponses-Rep. It is populated by IDRProblemDescDB::responses_kwhandler() and is queried by the ProblemDescDB::get_-<datatype>() functions. A list of DataResponses objects is maintained in ProblemDescDB::dataResponsesList, one for each responses specification in an input file.

The documentation for this class was generated from the following files:

- DataResponses.hpp
- DataResponses.cpp
14.53 DataResponsesRep Class Reference

Body class for responses specification data.

Public Attributes

- String idResponses
  string identifier for the responses specification data set (from the id.responses specification in RespSetId)
- StringArray responseLabels
  the response labels array (from the response.descriptors specification in RespLabels)
- size_t numObjectiveFunctions
  number of objective functions (from the num_objective_functions specification in RespFnOpt)
- size_t numLeastSqTerms
  number of least squares terms (from the num_calibration_terms specification in RespFnLS)
- size_t numNonlinearIneqConstraints
  number of nonlinear inequality constraints (from the num_nonlinear_inequality_constraints specification in RespFnOpt)
- size_t numNonlinearEqConstraints
  number of nonlinear equality constraints (from the num_nonlinear Equality constraints specification in RespFnOpt)
- size_t numResponseFunctions
  number of generic response functions (from the num_response_functions specification in RespFnGen)
- size_t numScalarObjectiveFunctions
  scalar_objectives: number of objective functions which are scalar
- size_t numScalarLeastSqTerms
  scalar_calibration_terms: number of calibration terms which are scalar
- size_t numScalarNonlinearIneqConstraints
  number of scalar nonlinear inequality constraints (from the num_scalar_nonlinear_inequality_constraints specification in RespFnOpt)
- size_t numScalarNonlinearEqConstraints
  number of scalar nonlinear equality constraints (from the num_scalar_nonlinear_latitude_constraints specification in RespFnOpt)
- size_t numScalarResponseFunctions
  scalar_responses: number of response functions which are scalar
- size_t numFieldObjectiveFunctions
  field_objectives: number of objective functions which are field-valued
- size_t numFieldLeastSqTerms
  field_calibration_terms: number of calibration terms which are field-valued
- size_t numFieldNonlinearIneqConstraints
  number of field nonlinear inequality constraints (from the num_field_nonlinear_inequality_constraints specification in RespFnOpt)
- size_t numFieldNonlinearEqConstraints
  number of field nonlinear equality constraints (from the num_field_nonlinear_latitude_constraints specification in RespFnOpt)
- size_t numFieldResponseFunctions
**field_responses**: number of response functions which are field-valued

- **StringArray primaryRespFnSense**
  - optimization sense for each objective function: minimize or maximize

- **RealVector primaryRespFnWeights**
  - vector of weightings for multiobjective optimization or weighted nonlinear least squares (from the `multi_objective_weights` specification in RespFnOpt and the `least_squares_weights` specification in RespFnLS)

- **RealVector nonlinearIneqLowerBnds**
  - vector of nonlinear inequality constraint lower bounds (from the `nonlinear_inequality_lower_bounds` specification in RespFnOpt)

- **RealVector nonlinearIneqUpperBnds**
  - vector of nonlinear inequality constraint upper bounds (from the `nonlinear_inequality_upper_bounds` specification in RespFnOpt)

- **RealVector nonlinearEqTargets**
  - vector of nonlinear equality constraint targets (from the `nonlinear_equality_targets` specification in RespFnOpt)

- **StringArray primaryRespFnScaleTypes**
  - vector of primary response function scaling types (from the `objective_function_scale_types` specification in RespFnOpt and the `least_squares_term_scale_types` specification in RespFnLS)

- **RealVector primaryRespFnScales**
  - vector of primary response function scaling factors (from the `objective_function_scales` specification in RespFnOpt and the `least_squares_term_scales` specification in RespFnLS)

- **StringArray nonlinearIneqScaleTypes**
  - vector of nonlinear inequality constraint scaling types (from the `nonlinear_inequality_scale_types` specification in RespFnOpt)

- **RealVector nonlinearIneqScales**
  - vector of nonlinear inequality constraint scaling factors (from the `nonlinear_inequality_scales` specification in RespFnOpt)

- **StringArray nonlinearEqScaleTypes**
  - vector of nonlinear equality constraint scaling types (from the `nonlinear_equality_scale_types` specification in RespFnOpt)

- **RealVector nonlinearEqScales**
  - vector of nonlinear equality constraint scaling factors (from the `nonlinear_equality_scales` specification in RespFnOpt)

- **bool calibrationDataFlag**
  - whether calibration data was specified

- **size_t numExperiments**
  - number of distinct experiments in experimental data

- **size_t numExpConfigVars**
  - number of experimental configuration vars (state variables) in each row of data

- **RealVector expConfigVars**
  - list of `num_experiments x num_config_vars` configuration variable values

- **RealVector simVariance**
  - list of variances of errors to be added to simulation responses

- **bool interpolateFlag**
whether one should interpolate between the experiment and simulation field data

- RealVector expObservations
  list of num_calibration_terms observation data
- RealVector expStdDeviations
  list of 1 or num_calibration_terms observation standard deviations
- String scalarDataFileName
  name of experimental data file containing response data (with optional state variable and sigma data) to read
- unsigned short scalarDataFormat
  tabular format of the scalar data file
- String gradientType
  gradient type: none, numerical, analytic, or mixed (from the no_gradients, numerical_gradients, analytic_gradients, and mixed_gradients specifications in RespGrad)
- String hessianType
  Hessian type: none, numerical, quasi, analytic, or mixed (from the no_hessians, numerical_hessians, quasi_hessians, analytic_hessians, and mixed_hessians specifications in RespHess)
- bool ignoreBounds
  option to ignore bounds when doing finite differences (default is to honor bounds)
- bool centralHess
  Temporary(?) option to use old 2nd-order diffs when computing finite-difference Hessians; default is forward differences.
- String quasiHessianType
  quasi-Hessian type: bfgs, damped_bfgs, or sr1 (from the bfgs and sr1 specifications in RespHess)
- String methodSource
  numerical gradient method source: dakota or vendor (from the method_source specification in RespGradNum and RespGradMixed)
- String intervalType
  numerical gradient interval type: forward or central (from the interval_type specification in RespGradNum and RespGradMixed)
- RealVector fdGradStepSize
  vector of finite difference step sizes for numerical gradients, one step size per active continuous variable, used in computing 1st-order forward or central differences (from the fd_gradient_step_size specification in RespGradNum and RespGradMixed)
- String fdGradStepType
  type of finite difference step to use for numerical gradient: relative - step length is relative to x absolute - step length is what is specified bounds - step length is relative to range of x
- RealVector fdHessStepSize
  vector of finite difference step sizes for numerical Hessians, one step size per active continuous variable, used in computing 1st-order gradient-based differences and 2nd-order function-based differences (from the fd_hessian_step_size specification in RespHessNum and RespHessMixed)
- String fdHessStepType
  type of finite difference step to use for numerical Hessian: relative - step length is relative to x absolute - step length is what is specified bounds - step length is relative to range of x
- IntSet idNumericalGrads
  mixed gradient numerical identifiers (from the id_numerical_gradients specification in RespGradMixed)
- IntSet idAnalyticGrads
mixed gradient analytic identifiers (from the id_analytic_gradients specification in RespGradMixed)

- IntSet idNumericalHessians
  mixed Hessian numerical identifiers (from the id_numerical_hessians specification in RespHessMixed)

- IntSet idQuasiHessians
  mixed Hessian quasi identifiers (from the id_quasi_hessians specification in RespHessMixed)

- IntSet idAnalyticHessians
  mixed Hessian analytic identifiers (from the id_analytic_hessians specification in RespHessMixed)

- IntVector fieldLengths
  number of entries in each field

- IntVector numCoordsPerField
  number of coordinates per field

- bool readFieldCoords
  Field data related storage: whether to read simulation field coordinates.

- StringArray varianceType
  Array which specifies the sigma type per response (none, one constant value, one per response (vector) or a full covariance matrix

Private Member Functions

- DataResponsesRep ()
  constructor

- ~DataResponsesRep ()
  destructor

- void write (std::ostream &s) const
  write a DataResponsesRep object to an std::ostream

- void read (MPIUnpackBuffer &s)
  read a DataResponsesRep object from a packed MPI buffer

- void write (MPIPackBuffer &s) const
  write a DataResponsesRep object to a packed MPI buffer

Private Attributes

- int referenceCount
  number of handle objects sharing this dataResponsesRep

Friends

- class DataResponses
  the handle class can access attributes of the body class directly
14.53.1 Detailed Description

Body class for responses specification data.

The DataResponsesRep class is used to contain the data from a responses keyword specification. Default values are managed in the DataResponsesRep constructor. Data is public to avoid maintaining set/get functions, but is still encapsulated within ProblemDescDB since ProblemDescDB::dataResponsesList is private.

The documentation for this class was generated from the following files:

- DataResponses.hpp
- DataResponses.cpp

14.54 DataScaler Class Reference

The DataScaler class computes the scaling coefficients and scales a 2D data matrix with dimensions num_samples by num_features.

Inheritance diagram for DataScaler:

```
  DataScaler
     |     |     |
  NormalizationScaler   NoScaler   StandardizationScaler
```

Public Types

- enum SCALER_TYPE { NONE, STANDARDIZATION, MEAN_NORMALIZATION, MINMAX_NORMALIZATION }

  Enumeration for supported types of DataScalers.

Public Member Functions

- std::shared_ptr<MatrixXd> scale_samples (const MatrixXd &unscaled_samples)
  Apply scaling to a set of unscaled samples.
- void scale_samples (const MatrixXd &unscaled_samples, MatrixXd &scaled_samples)
  Apply scaling to a set of unscaled samples.
- const VectorXd & get_scaler_features_offsets () const
  Get the vector of offsets.
- const VectorXd & get_scaler_features_scale_factors () const
  Get the vector of scaling factors.
- const MatrixXd & get_scaled_features () const
  Get the the scaled data matrix.
- bool check_for_zero_scaler_factor (int index)
  Checks an individual scaler feature scale factor for being close to zero; If it is near zero, we can potentially run into a divide-by-zero error if not handled appropriately.
### Static Public Member Functions

- static `SCALER_TYPE scaler_type (const std::string &scaler_name)`
  
  Convert scaler name to enum type.

### Protected Attributes

- `bool hasScaling`
  
  Bool for whether or not the the scaling coefficients have been computed.
- `VectorXd scalerFeaturesOffsets`
  
  Vector of offsets - (num_features)
- `VectorXd scalerFeaturesScaleFactors`
  
  Vector of scaling factors - (num_features)
- `MatrixXd scaledFeatures`
  
  Scaled surrogate data matrix - (num_samples by num_features)

### Detailed Description

The DataScaler class computes the scaling coefficients and scales a 2D data matrix with dimensions `num_samples` by `num_features`.

There are currently 3 scaling options for the DataScaler class:

1. **StandardizationScaler** - transform each feature to have zero mean and unit variance.
2. **NormalizationScaler** - normalizes each feature uses the max and min value divided by either the mean value (mean_normalization = true) or min value (mean_normalization = false) Also allows for a norm_factor scaling, required for the direct neural network.
3. **NoScaler** - scaling coefficients amount to an identity operation

### Member Function Documentation

#### `std::shared_ptr< MatrixXd > scale_samples ( const MatrixXd & unscaled_samples )`

Apply scaling to a set of unscaled samples.

<table>
<thead>
<tr>
<th>Input</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>unscaled_samples</code></td>
<td>Unscaled matrix of samples</td>
</tr>
</tbody>
</table>

Returns

A shared_ptr to the scaled matrix of samples

References DataScaler::check_for_zero_scaler_factor(), DataScaler::scalerFeaturesOffsets, and DataScaler::scalerFeaturesScaleFactors.

#### `void scale_samples ( const MatrixXd & unscaled_samples, MatrixXd & scaled_samples )`

Apply scaling to a set of unscaled samples.
Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>unscaled_samples</th>
<th>Unscaled matrix of samples</th>
</tr>
</thead>
<tbody>
<tr>
<td>out</td>
<td>scaled_samples</td>
<td>Scaled matrix of samples</td>
</tr>
</tbody>
</table>

References DataScaler::check_for_zero_scaler_factor(), DataScaler::scalerFeaturesOffsets, and DataScaler::scalerFeaturesScaleFactors.

```cpp
const VectorXd& get_scaler_features_offsets() const [inline]
```

Get the vector of offsets.

Returns

Vector of scaler offsets - (num_features)

References DataScaler::scalerFeaturesOffsets.

```cpp
const VectorXd& get_scaler_features_scale_factors() const [inline]
```

Get the vector of scaling factors.

Returns

Vector of scaling factors - (num_features)

References DataScaler::scalerFeaturesScaleFactors.

```cpp
const MatrixXd& get_scaled_features() const [inline]
```

Get the the scaled data matrix.

Returns

Scaled features - (num_samples by num_features)

References DataScaler::scaledFeatures.

```cpp
bool check_for_zero_scaler_factor(int index)
```

Checks an individual scaler feature scale factor for being close to zero; If it is near zero, we can potentially run into a divide-by-zero error if not handled appropriately.

Parameters

| in  | index | The scaler feature index to check |

Returns

True if the value is near zero

References DataScaler::scalerFeaturesScaleFactors.

Referenced by NormalizationScaler::NormalizationScaler(), DataScaler::scale_samples(), and StandardizationScaler::StandardizationScaler().

```cpp
SCALER_TYPE scaler_type(const std::string & scaler_name) [static]
```

Convert scaler name to enum type.
14.55 DataTransformModel Class Reference

Data transformation specialization of RecastModel.

Inheritance diagram for DataTransformModel:

```
Model
  RecastModel
    DataTransformModel
```

Public Member Functions

- **DataTransformModel** (const Model &sub_model, const ExperimentData &exp_data, size_t num_hyper=0, unsigned short mult_mode=CALIBRATE_NONE, short recast_resp_deriv_order=1)
  - *standard constructor*
- **~DataTransformModel** ()
  - *destructor*
- void data_transform_response (const Variables &sub_model_vars, const Response &sub_model_resp, Response &residual_resp)
  - *Convenience function to help recover a residual response from the submodel.*
- void data_resize ()
  - *The size of the ExperimentData changed; update the residualModel size.*
- void print_best_responses (std::ostream &, const Variables &best_submodel_vars, const Response &best_submodel_resp, size_t num_best, size_t best_ind)
  - *manage best responses including residuals and model responses per config*
- void archive_best_responses (const ResultsManager &results_db, const StrStrSizet iterator_id, const Variables &best_submodel_vars, const Response &best_submodel_resp, size_t num_best, size_t best_ind)
  - *archive best responses*
- int num_config_vars () const
  - *return number of configuration variables*
Protected Types

- typedef std::map<int, IntResponseMap> IntIntResponseMapMap

Protected Member Functions

- void assign_instance()
  
  assign static pointer instance to this for use in static transformation functions

- void update_from_subordinate_model (size_t depth=std::numeric_limits<size_t>::max())
  
  propagate vars/labels/bounds/targets from the bottom up

- void gen_primary_resp_map (const SharedResponseData &srd, Sizet2DArray &primary_resp_map_indices, BoolDequeArray &nonlinear_resp_map) const
  
  compute the primary response map for a data transform RecastModel

- void derived_evaluate (const ActiveSet &set)
  
  specialization of evaluate that iterates over configuration variables

- void derived_evaluate_nowait (const ActiveSet &set)
  
  specialization of evaluate that iterates over configuration variables

- const IntResponseMap & derived_synchronize ()
  
  synchronize all evaluations (all residuals for all experiment configurations)

- const IntResponseMap & derived_synchronize_nowait ()
  
  return any evaluations for which all experiment configurations have completed

- const IntResponseMap & filter_submodel_responses ()

- void cache_submodel_responses (const IntResponseMap &sm_resp_map, bool deep_copy)
  
  cache the subModel responses into a per-RecastModel eval ID map

- void collect_residuals (bool collect_all)
  
  collect any (or force all) completed subModel evals and populate recastResponseMap with residuals for those that are fully completed

- void transform_response_map (const IntResponseMap &submodel_resp, const Variables &recast_vars, Response &residual_resp)
  
  transform a set of per-configuration subModel Responses to a single evaluation's residuals

- void scale_response (const Variables &submodel_vars, const Variables &recast_vars, Response &recast_response)
  
  scale the populated residual response by any covariance information, including hyper-parameter multipliers

- void init_continuous_vars ()
  
  Initialize continuous variable values/labels.

- template<typename T>
  
  void expand_primary_array (size_t submodel_size, const T &submodel_array, size_t recast_size, T &recast_array) const
  
  (if non-empty) expand submodel_array by replicates to populate a recast_array

- void print_residual_response (const Response &resid_resp)

- void recover_submodel_responses (std::ostream &s, const Variables &best_submodel_vars, size_t num_best, size_t best_ind, Response &residual_resp)

- void archive_submodel_responses (const ResultsManager &results_db, const StrStrSizet &iterator_id, const Variables &best_submodel_vars, size_t num_best, size_t best_ind, Response &residual_resp)
  
  archive original model responses
void archive_best_original (const ResultsManager &results_db, const StrSizet &iterator_id, const RealVector &function_values, const int &exp_index, const int &num_best, const int &best_index)

Archive the best model responses (undifferenced with experimental data) for experiment exp_index and final solution soln_index.

void archive_best_config_variables (const ResultsManager &results_db, const StrSizet &iterator_id, const Variables &vars, const int &exp_index, const int &num_best, const int &best_index)

Archive the best configuration variables associated with each model response.

void archive_best_residuals (const ResultsManager &results_db, const StrSizet &iterator_id, const int num_fns, const RealVector &best_terms, const Real wssr, const int num_points, const int point_index)

Archive the best residuals.

Static Protected Member Functions

static SizetArray variables_expand (const Model &sub_model, size_t num_hyper)

expand the variable counts to account for hyper-parameters

static int get_hyperparam_vc_index (const Model &sub_model)

determine the index into vc_totals corresponding to where the hyper-parameters go

static short response_order (const Model &sub_model, short recast_resp_order=1)

helper to compute the recast response order during member initialization; recast_resp_order passed is the minimum request client needs

static void vars_mapping (const Variables &recastVars, Variables &submodel_vars)

map the inbound expanded variables to the sub-model, discarding hyperparams (assumes hyper-parameters are at end of active continuous variables)

static void set_mapping (const Variables &recast_vars, const ActiveSet &recast_set, ActiveSet &sub_model_set)

map the inbound ActiveSet to the sub-model (map derivative variables)

static void primary_resp_differencer (const Variables &submodel_vars, const Variables &recast_vars, const Response &submodel_response, Response &recast_response)

Recast callback function to difference residuals with observed data.

Protected Attributes

const ExperimentData & expData

Reference to the experiment data used to construct this Model.

size_t numHyperparams

Number of calibrated variance multipliers.

unsigned short obsErrorMultiplierMode

Calibration mode for the hyper-parameters.

IntIntResponseMapMap cachedResp

Static Protected Attributes

static DataTransformModel * dtModelInstance

static pointer to this class for use in static callbacks
Additional Inherited Members

14.55.1 Detailed Description

Data transformation specialization of RecastModel.

Specialization of RecastModel to create a residual model that maps (1) from an augmented set of calibration parameters (including hyper-parameters) to those needed by the underlying simulation model and (2) from the simulation model response to a set of residuals, whose overall size may differ from the simulation (sub-model) response. The residuals may be scaled by experiment covariance information. This class provides a simple constructor that forwards to the more complicated RecastModel API.

14.55.2 Constructor & Destructor Documentation

DataTransformModel (const Model &sub_model, const ExperimentData &exp_data, size_t num_hyper = 0, unsigned short mult_mode = CALIBRATE_NONE, short recast_resp_deriv_order = 1 )

standard constructor

This constructor computes various indices and mappings, then updates the properties of the RecastModel. Hyper-parameters are assumed to trail the active continuous variables when presented to this RecastModel.

References Dakota::abort_handler(), Model::current_response(), Model::cv(), Model::div(), Model::drv(), Model::dsv(), DataTransformModel::expand_primary_array(), DataTransformModel::expData, DataTransformModel::gen_primary_resp_map(), Model::cv(), Model::div(), Model::drv(), Model::dsv(), Model::inactive_view(), DataTransformModel::init_continuous_vars(), RecastModel::init_maps(), Model::modelId, DataTransformModel::num_config_vars(), ExperimentData::num_config_vars(), Model::num_primary_fns(), Model::num_secondary_fns(), ExperimentData::num_total_exppoints(), DataTransformModel::numHyperparams, DataTransformModel::primary_resp_differencer(), Model::primary_response_fn_sense(), Model::primary_response_fn_weights(), Model::primaryRespFnSense, Model::primaryRespFnWts, ScalingOptions::priScales, ScalingOptions::priScaleTypes, RecastModel::recast_model_id(), RecastModel::root_model_id(), Model::scaling_options(), Model::scalingOpts, DataTransformModel::set_mapping(), Response::shared_data(), RecastModel::subModel, and DataTransformModel::vars_mapping().

14.55.3 Member Function Documentation

void update_from_subordinate_model ( size_t depth = std::numeric_limits<size_t>::max() ) [inline], [protected], [virtual]

propagate vars/labels/bounds/targets from the bottom up

used only for instantiate-on-the-fly model recursions (all RecastModel instantiations and alternate DataFitSurrModel instantiations). Simulation, Hierarchical, and Nested Models do not redefine the function since they do not support instantiate-on-the-fly. This means that the recursion will stop as soon as it encounters a Model that was instantiated normally, which is appropriate since ProblemDescDB-constructed Models use top-down information flow and do not require bottom-up updating.

Reimplemented from Model.

References DataTransformModel::numHyperparams, RecastModel::subModel, Model::update_from_subordinate_model(), RecastModel::update_response_from_model(), RecastModel::update_variables_active_complement_from_model(), and RecastModel::update_variables_from_model().

SizetArray variables_expand (const Model &sub_model, size_t num_hyper ) [static], [protected]

expand the variable counts to account for hyper-parameters
Incorporate the hyper parameters into Variables, assuming they are at the end of the active continuous variables. For example, append them to continuous design or continuous aleatory uncertain.

References SharedVariablesData::components_totals(), Model::current_variables(), DataTransformModel::get_hyperparam_vc_index(), Variables::shared_data(), and Dakota::svd().

```cpp
void derived_evaluate ( const ActiveSet & set ) [protected], [virtual]
```
specialization of evaluate that iterates over configuration variables

Blocking evaluation over all experiment configurations to compute a single set of expanded residuals. If the subModel supports asynchronous evaluate_nowait(), do the configuration evals concurrently and then synchronize.

Reimplemented from Model.

```cpp
void derived_evaluate_nowait ( const ActiveSet & set ) [protected], [virtual]
```
specialization of evaluate that iterates over configuration variables

Non-blocking evaluation (scheduling) over all experiment configurations. Assumes that if this model supports nowait, its subModel does too and schedules them all.

Reimplemented from Model.

```cpp
const IntResponseMap & derived_synchronize() [protected], [virtual]
```
synchronize all evaluations (all residuals for all experiment configurations)

Collect all the subModel evals and build the residual sets for all evaluations. Like rekey functions in DakotaModel, but many sub-model to one recast-model. For the blocking synchronize case, we force the subModel to synch and have all needed data.

Reimplemented from Model.

```cpp
const IntResponseMap & derived_synchronize_nowait() [protected], [virtual]
```
return any evaluations for which all experiment configurations have completed

Collect any completed subModel evals and build the residual sets for any fully completed evaluations. Like rekey functions in DakotaModel, but many sub-model to one recast-model. We do not force the subModel to synch.

Reimplemented from Model.
References DataTransformModel::cache_submodel_responses(), DataTransformModel::collect_residuals(), ExperimentData::config_vars(), RecastModel::derived_synchronize_nowait(), DataTransformModel::expData, RecastModel::recastResponseMap, RecastModel::subModel, and Model::synchronize_nowait().

```cpp
const IntResponseMap & filter_submodel_responses() [protected]
```

(We don’t quite want the rekey behavior since multiple subModel evals map to one recast eval.)

References Model::cache_unmatched_response(), RecastModel::recastIdMap, RecastModel::subModel, and Model::synchronize().

Referenced by DataTransformModel::derived_evaluate().

```cpp
void transform_response_map(const IntResponseMap & submodel, const Variables & recast_vars, Response & residual) [protected]
```

transform a set of per-configuration subModel Responses to a single evaluation’s residuals

This transformation assumes the residuals are in submodel eval id order.

References Dakota::abort_handler(), Model::current_variables(), DataTransformModel::expData, ExperimentData::form_residuals(), ExperimentData::num_experiments(), DataTransformModel::scale_response(), and RecastModel::subModel.

Referenced by DataTransformModel::collect_residuals(), and DataTransformModel::derived_evaluate().

```cpp
void set_mapping(const Variables & recast_vars, const ActiveSet & recast_set, ActiveSet & submodel) [static], [protected]
```

map the inbound ActiveSet to the sub-model (map derivative variables)

RecastModel sets up a default set mapping before calling this update, so focus on updating the derivative variables vector

References Model::cv(), ActiveSet::derivative_vector(), DataTransformModel::dtModelInstance, DataTransformModel::numHyperparams, ActiveSet::request_vector(), and RecastModel::subordinate_model().

Referenced by DataTransformModel::DataTransformModel().

```cpp
void init_continuous_vars() [protected]
```

Initialize continuous variable values/labels.

Pull up the continuous variable values and labels into the RecastModel, inserting the hyper-parameter values/labels

References Model::all_continuous_lower_bound(), Model::all_continuous_lower_bounds(), Model::all_continuous_upper_bound(), Model::all_continuous_upper_bounds(), Model::all_continuous_variable(), Model::all_continuous_variable_label(), Model::all_continuous_variable_labels(), Model::all_continuous_variables(), SharedVariablesData::components_totals(), Model::current_variables(), DataTransformModel::expData, DataTransformModel::get_hyperparam_vc_index(), ExperimentData::hyperparam_labels(), DataTransformModel::numHyperparams, DataTransformModel::obsErrorMultiplierMode, Variables::shared_data(), RecastModel::subModel, and Dakota::svd().

Referenced by DataTransformModel::DataTransformModel().

```cpp
void expand_primary_array(size_t submodel_size, const T & submodel_array, size_t recast_size, T & recast_array) const [protected]
```

(if non-empty) expand submodel_array by replicates to populate a recast_array

If size greater than 1, expand submodel_array by replicates to populate a pre-sized recast_array, otherwise copy

Passing the inbound array size so we can use one function for Teuchos and std containers (size vs. length)
14.56. DATAVARIABLES CLASS REFERENCE

References DataTransformModel::expData, and ExperimentData::num_experiments().
Referenced by DataTransformModel::DataTransformModel().

```cpp
void archive_best_config_variables ( const ResultsManager & results_db, const StrStrSizet & iterator_id,
const Variables & vars, const int & exp_index, const int & num_best, const int & best_index )
[protected]
```

Archive the best configuration variables associated with each model response.

Archive best configuration variables.

References ResultsManager::active(), Variables::inactive_continuous_variable_labels(), Variables::inactive_continuous_variables(), Variables::inactive_discrete_int_variable_labels(), Variables::inactive_discrete_int_variables(), Variables::inactive_discrete_real_variable_labels(), Variables::inactive_discrete_real_variables(), Variables::inactive_discrete_string_variable_labels(), Variables::inactive_discrete_string_variables(), and ResultsManager::insert().

Referenced by DataTransformModel::archive_submodel_responses().

14.55.4 Member Data Documentation

DataTransformModel * dtModelInstance [static], [protected]

static pointer to this class for use in static callbacks
initialization of static needed by RecastModel

Referenced by DataTransformModel::assign_instance(), DataTransformModel::primary_resp_differencer(), and DataTransformModel::set_mapping().

The documentation for this class was generated from the following files:

- DataTransformModel.hpp
- DataTransformModel.cpp

14.56 DataVariables Class Reference

Handle class for variables specification data.

Public Member Functions

- **DataVariables ()**
  
  constructor

- **DataVariables (const DataVariables &)**
  
  copy constructor

- **~DataVariables ()**
  
  destructor

- **DataVariables operator= (const DataVariables &)**
  
  assignment operator

- **bool operator== (const DataVariables &)**
  
  equality operator

- **void write (std::ostream &s) const**
  
  write a DataVariables object to an std::ostream

- **void read (MPIUnpackBuffer &s)**
  
  read a DataVariables object from a packed MPI buffer
• void write (MPIPackBuffer &s) const
  
  write a DataVariables object to a packed MPI buffer

• DataVariablesRep * data_rep ()
  
  return dataVarsRep

• size_t continuous_design ()
  
  return total number of design variables

• size_t discrete_design ()
  
  return total number of design variables

• size_t continuous_aleatory_uncertain ()
  
  return total number of continuous aleatory uncertain variables

• size_t discrete_aleatory_uncertain ()
  
  return total number of continuous aleatory uncertain variables

• size_t continuous_epistemic_uncertain ()
  
  return total number of epistemic uncertain variables

• size_t discrete_epistemic_uncertain ()
  
  return total number of epistemic uncertain variables

• size_t continuous_state ()
  
  return total number of state variables

• size_t discrete_state ()
  
  return total number of state variables

• size_t design ()
  
  return total number of design variables

• size_t aleatory_uncertain ()
  
  return total number of aleatory uncertain variables

• size_t epistemic_uncertain ()
  
  return total number of epistemic uncertain variables

• size_t uncertain ()
  
  return total number of uncertain variables

• size_t state ()
  
  return total number of state variables

• size_t continuous_variables ()
  
  return total number of continuous variables

• size_t discrete_variables ()
  
  return total number of discrete variables

• size_t total_variables ()
  
  return total number of variables

**Static Public Member Functions**

• static bool id_compare (const DataVariables &dv, const std::string &id)
  
  compares the idVariables attribute of DataVariables objects
14.57. DATAVARIABLESREP CLASS REFERENCE

Private Attributes

- **DataVariablesRep * dataVarsRep**
  pointer to the body (handle-body idiom)

Friends

- class ProblemDescDB
- class NIDRProblemDescDB

14.56.1 Detailed Description

Handle class for variables specification data.

The **DataVariables** class is used to provide a memory management handle for the data in DataVariables-Rep. It is populated by IDRProblemDescDB::variables_khandler() and is queried by the ProblemDescDB::get_<datatype>() functions. A list of DataVariables objects is maintained in ProblemDescDB::dataVariablesList, one for each variables specification in an input file.

The documentation for this class was generated from the following files:

- DataVariables.hpp
- DataVariables.cpp

14.57 DataVariablesRep Class Reference

Body class for variables specification data.

Public Attributes

- **String idVariables**
  string identifier for the variables specification data set (from the id_variables specification in VarSetId)

- **short varsView**
  user selection/override of variables view: {DEFAULT, ALL, DESIGN, UNCERTAIN, ALEATORY, UNCERTAIN, EPISTEMIC, UNCERTAIN, STATE} VIEW

- **short varsDomain**
  user selection/override of variables domain: {DEFAULT, MIXED, RELAXED} DOMAIN

- **bool uncertainVarsInitPt**
  flag indicating user specification of initial points (for local optimization-based UQ methods) for at least one uncertain variable type

- **size_t numContinuousDesVars**
  number of continuous design variables (from the continuous_design specification in VarDV)

- **size_t numDiscreteDesRangeVars**
  number of discrete design variables defined by an integer range (from the discrete_design_range specification in VarDV)

- **size_t numDiscreteDesSetIntVars**
  number of discrete design variables defined by a set of integers (from the discrete_design_set integer specification in VarDV)

- **size_t numDiscreteDesSetStrVars**
  number of discrete design variables defined by a set of strings (from the discrete_design_set_string specification in VarDV)
number of discrete design variables defined by a set of strings (from the discrete_design_set string specification in VarDV)

- `size_t numDiscreteDesSetRealVars`
  number of discrete design variables defined by a set of reals (from the discrete_design_set real specification in VarDV)

- `size_t numNormalUncVars`
  number of normal uncertain variables (from the normal_uncertain specification in VarAUV)

- `size_t numLognormalUncVars`
  number of lognormal uncertain variables (from the lognormal_uncertain specification in VarAUV)

- `size_t numUniformUncVars`
  number of uniform uncertain variables (from the uniform_uncertain specification in VarAUV)

- `size_t numLoguniformUncVars`
  number of loguniform uncertain variables (from the loguniform_uncertain specification in VarAUV)

- `size_t numTriangularUncVars`
  number of triangular uncertain variables (from the triangular_uncertain specification in VarAUV)

- `size_t numExponentialUncVars`
  number of exponential uncertain variables (from the exponential_uncertain specification in VarAUV)

- `size_t numBetaUncVars`
  number of beta uncertain variables (from the beta_uncertain specification in VarAUV)

- `size_t numGammaUncVars`
  number of gamma uncertain variables (from the gamma_uncertain specification in VarAUV)

- `size_t numGumbelUncVars`
  number of gumbel uncertain variables (from the gumbel_uncertain specification in VarAUV)

- `size_t numFrechetUncVars`
  number of frechet uncertain variables (from the frechet_uncertain specification in VarAUV)

- `size_t numWeibullUncVars`
  number of weibull uncertain variables (from the weibull_uncertain specification in VarAUV)

- `size_t numHistogramBinUncVars`
  number of histogram bin uncertain variables (from the histogram_bin_uncertain specification in VarAUV)

- `size_t numPoissonUncVars`
  number of Poisson uncertain variables (from the poisson_uncertain specification in VarAUV)

- `size_t numBinomialUncVars`
  number of binomial uncertain variables (from the binomial_uncertain specification in VarAUV)

- `size_t numNegBinomialUncVars`
  number of negative binomial uncertain variables (from the negative_binomial_uncertain specification in VarAUV)

- `size_t numGeometricUncVars`
  number of geometric uncertain variables (from the geometric_uncertain specification in VarAUV)

- `size_t numHyperGeomUncVars`
  number of hypergeometric uncertain variables (from the hypergeometric_uncertain specification in VarAUV)

- `size_t numHistogramPtIntUncVars`
  number of integer-valued histogram point uncertain variables (from the histogram_point_int_uncertain specification in VarAUV)
• size_t numHistogramPtStrUncVars
  number of string-valued histogram point uncertain variables (from the histogram_point_uncertain specification in VarAUV)
• size_t numHistogramPtRealUncVars
  number of real-valued histogram point uncertain variables (from the histogram_point_uncertain specification in VarAUV)
• size_t numContinuousIntervalUncVars
  number of continuous epistemic interval uncertain variables (from the continuous_interval_uncertain specification in VarEUV)
• size_t numDiscreteIntervalUncVars
  number of discrete epistemic interval uncertain variables (from the discrete_interval_uncertain specification in VarEUV)
• size_t numDiscreteUncSetIntVars
  number of discrete epistemic uncertain integer set variables (from the discrete_uncertain_set integer specification in VarEUV)
• size_t numDiscreteUncSetStrVars
  number of discrete epistemic uncertain string set variables (from the discrete_uncertain_set string specification in VarEUV)
• size_t numDiscreteUncSetRealVars
  number of discrete epistemic uncertain real set variables (from the discrete_uncertain_set real specification in VarEUV)
• size_t numContinuousStateVars
  number of continuous state variables (from the continuous_state specification in VarSV)
• size_t numDiscreteStateRangeVars
  number of discrete state variables defined by an integer range (from the discrete_state_range specification in VarDV)
• size_t numDiscreteStateSetIntVars
  number of discrete state variables defined by a set of integers (from the discrete_state_set integer specification in VarDV)
• size_t numDiscreteStateSetStrVars
  number of discrete state variables defined by a set of strings (from the discrete_state_set string specification in VarDV)
• size_t numDiscreteStateSetRealVars
  number of discrete state variables defined by a set of reals (from the discrete_state_set real specification in VarDV)
• RealVector continuousDesignVars
  initial values for the continuous design variables array (from the continuous_design_initial_point specification in VarDV)
• RealVector continuousDesignLowerBnds
  lower bounds array for the continuous design variables (from the continuous_design_lower_bounds specification in VarDV)
• RealVector continuousDesignUpperBnds
  upper bounds array for the continuous design variables (from the continuous_design_upper_bounds specification in VarDV)
• StringArray continuousDesignScaleTypes
scale types array for the continuous design variables (from the \texttt{continuous\_design scale\_types specification} in \texttt{VarDV})

- \texttt{RealVector continuousDesignScales}
  scales array for the continuous design variables (from the \texttt{continuous\_design scales specification} in \texttt{VarDV})

- \texttt{IntVector discreteDesignRangeVars}
  initial values for the discrete design variables defined by an integer range (from the \texttt{discrete\_design_range initial\_point specification} in \texttt{VarDV})

- \texttt{IntVector discreteDesignRangeLowerBnds}
  lower bounds array for the discrete design variables defined by an integer range (from the \texttt{discrete\_design\_range lower\_bounds specification} in \texttt{VarDV})

- \texttt{IntVector discreteDesignRangeUpperBnds}
  upper bounds array for the discrete design variables defined by an integer range (from the \texttt{discrete\_design\_range upper\_bounds specification} in \texttt{VarDV})

- \texttt{BitArray discreteDesignRangeCat}
  is each ddr var strictly categorical (true) or relaxable (false)

- \texttt{IntVector discreteDesignSetIntVars}
  initial values for the discrete design variables defined by an integer set (from the \texttt{discrete\_design\_set integer initial\_point specification} in \texttt{VarDV})

- \texttt{StringArray discreteDesignSetStrVars}
  initial values for the discrete design variables defined by a string set (from the \texttt{discrete\_design\_set string initial\_point specification} in \texttt{VarDV})

- \texttt{RealVector discreteDesignSetRealVars}
  initial values for the discrete design variables defined by a real set (from the \texttt{discrete\_design\_set real initial\_point specification} in \texttt{VarDV})

- \texttt{IntSetArray discreteDesignSetInt}
  complete set of admissible values for each of the discrete design variables defined by an integer set (from the \texttt{discrete\_design\_set integer set\_values specification} in \texttt{VarDV})

- \texttt{StringSetArray discreteDesignSetStr}
  complete set of admissible values for each of the discrete design variables defined by a string set (from the \texttt{discrete\_design\_set string set\_values specification} in \texttt{VarDV})

- \texttt{RealSetArray discreteDesignSetReal}
  complete set of admissible values for each of the discrete design variables defined by a real set (from the \texttt{discrete\_design\_set real set\_values specification} in \texttt{VarDV})

- \texttt{BitArray discreteDesignSetIntCat}
  is each ddsi var strictly categorical (true) or relaxable (false)

- \texttt{BitArray discreteDesignSetRealCat}
  is each ddsr var strictly categorical (true) or relaxable (false)

- \texttt{RealMatrixArray discreteDesignSetIntAdj}
  Adjacency matrices for each of the discrete design variables defined by an integer set (from the \texttt{discrete\_design\_set integer categorical adjacency specification} in \texttt{VarDV})

- \texttt{RealMatrixArray discreteDesignSetStrAdj}
  Adjacency matrices for each of the discrete design variables defined by a string set (from the \texttt{discrete\_design\_set string adjacency specification} in \texttt{VarDV})

- \texttt{RealMatrixArray discreteDesignSetRealAdj}
Adjacency matrices for each of the discrete design variables defined by a real set (from the discrete_design_set real categorical adjacency specification in VarDV).

- **StringArray `continuousDesignLabels`**
  labels array for the continuous design variables (from the continuous_design descriptors specification in VarDV)

- **StringArray `discreteDesignRangeLabels`**
  labels array for the discrete design variables defined by an integer range (from the discrete_design_range descriptors specification in VarDV)

- **StringArray `discreteDesignSetIntLabels`**
  labels array for the discrete design variables defined by an integer set (from the discrete_design_set int descriptors specification in VarDV)

- **StringArray `discreteDesignSetStrLabels`**
  labels array for the discrete design variables defined by a string set (from the discrete_design_set string descriptors specification in VarDV)

- **StringArray `discreteDesignSetRealLabels`**
  labels array for the discrete design variables defined by a real set (from the discrete_design_set real descriptors specification in VarDV)

- **RealVector `normalUncMeans`**
  means of the normal uncertain variables (from the means specification in VarCAUV_Normal)

- **RealVector `normalUncStdDevs`**
  standard deviations of the normal uncertain variables (from the std_deviations specification in VarCAUV_Normal)

- **RealVector `normalUncLowerBnds`**
  distribution lower bounds for the normal uncertain variables (from the lower_bounds specification in VarCAUV_Normal)

- **RealVector `normalUncUpperBnds`**
  distribution upper bounds for the normal uncertain variables (from the upper_bounds specification in VarCAUV_Normal)

- **RealVector `normalUncVars`**
  initial values of the normal uncertain variables (from the initial_point specification in VarCAUV_Normal)

- **RealVector `lognormalUncLambdas`**
  lambdas (means of the corresponding normals) of the lognormal uncertain variables (from the lambdas specification in VarCAUV_Lognormal)

- **RealVector `lognormalUncZetas`**
  zetas (standard deviations of the corresponding normals) of the lognormal uncertain variables (from the zetas specification in VarCAUV_Lognormal)

- **RealVector `lognormalUncMeans`**
  means of the lognormal uncertain variables (from the means specification in VarCAUV_Lognormal)

- **RealVector `lognormalUncStdDevs`**
  standard deviations of the lognormal uncertain variables (from the std_deviations specification in VarCAUV_Lognormal)

- **RealVector `lognormalUncErrFacts`**
  error factors for the lognormal uncertain variables (from the error_factors specification in VarCAUV_Lognormal)

- **RealVector `lognormalUncLowerBnds`**
  distribution lower bounds for the lognormal uncertain variables (from the lower_bounds specification in VarCAUV_Lognormal)
• RealVector lognormalUncUpperBnds
distribution upper bounds for the lognormal uncertain variables (from the upper_bounds specification in VarC-AUV_Lognormal)

• RealVector lognormalUncVars
initial values of the lognormal uncertain variables (from the initial_point specification in VarCAUV_Lognormal)

• RealVector uniformUncLowerBnds
distribution lower bounds for the uniform uncertain variables (from the lower_bounds specification in VarCAUV_Uniform)

• RealVector uniformUncUpperBnds
distribution upper bounds for the uniform uncertain variables (from the upper_bounds specification in VarCAUV_Uniform)

• RealVector uniformUncVars
initial values of the uniform uncertain variables (from the initial_point specification in VarCAUV_Uniform)

• RealVector loguniformUncLowerBnds
distribution lower bounds for the loguniform uncertain variables (from the lower_bounds specification in VarCAUV_Loguniform)

• RealVector loguniformUncUpperBnds
distribution upper bounds for the loguniform uncertain variables (from the upper_bounds specification in VarCAUV_Loguniform)

• RealVector loguniformUncVars
initial values of the loguniform uncertain variables (from the initial_point specification in VarCAUV_Loguniform)

• RealVector triangularUncModes
modes of the triangular uncertain variables (from the modes specification in VarCAUV_Triangular)

• RealVector triangularUncLowerBnds
distribution lower bounds for the triangular uncertain variables (from the lower_bounds specification in VarCAUV_Triangular)

• RealVector triangularUncUpperBnds
distribution upper bounds for the triangular uncertain variables (from the upper_bounds specification in VarCAUV_Triangular)

• RealVector triangularUncVars
initial values of the triangular uncertain variables (from the initial_point specification in VarCAUV_Triangular)

• RealVector exponentialUncBetas
beta factors for the exponential uncertain variables (from the betas specification in VarCAUV_Exponential)

• RealVector exponentialUncVars
initial values of the exponential uncertain variables (from the initial_point specification in VarCAUV_Exponential)

• RealVector betaUncAlphas
alpha factors for the beta uncertain variables (from the means specification in VarCAUV_Beta)

• RealVector betaUncBetas
beta factors for the beta uncertain variables (from the std.deviations specification in VarCAUV_Beta)

• RealVector betaUncLowerBnds
distribution lower bounds for the beta uncertain variables (from the lower_bounds specification in VarCAUV_Beta)

• RealVector betaUncUpperBnds
distribution upper bounds for the beta uncertain variables (from the `upper_bounds` specification in VarCAUV.
- Beta)

- RealVector `betaUncVars`
  initial values of the beta uncertain variables (from the `initial_point` specification in VarCAUV.Beta)

- RealVector `gammaUncAlphas`
  alpha factors for the gamma uncertain variables (from the `alphas` specification in VarCAUV.Gamma)

- RealVector `gammaUncBetas`
  beta factors for the gamma uncertain variables (from the `betas` specification in VarCAUV.Gamma)

- RealVector `gammaUncVars`
  initial values of the gamma uncertain variables (from the `initial_point` specification in VarCAUV.Gamma)

- RealVector `gumbelUncAlphas`
  alpha factors for the gumbel uncertain variables (from the `alphas` specification in VarCAUV.Gumbel)

- RealVector `gumbelUncBetas`
  beta factors for the gumbel uncertain variables (from the `betas` specification in VarCAUV.Gumbel)

- RealVector `gumbelUncVars`
  initial values of the gumbel uncertain variables (from the `initial_point` specification in VarCAUV.Gumbel)

- RealVector `frechetUncAlphas`
  alpha factors for the frechet uncertain variables (from the `alphas` specification in VarCAUV.Frechet)

- RealVector `frechetUncBetas`
  beta factors for the frechet uncertain variables (from the `betas` specification in VarCAUV.Frechet)

- RealVector `frechetUncVars`
  initial values of the frechet uncertain variables (from the `initial_point` specification in VarCAUV.Frechet)

- RealVector `weibullUncAlphas`
  alpha factors for the weibull uncertain variables (from the `alphas` specification in VarCAUV.Weibull)

- RealVector `weibullUncBetas`
  beta factors for the weibull uncertain variables (from the `betas` specification in VarCAUV.Weibull)

- RealVector `weibullUncVars`
  initial values of the weibull uncertain variables (from the `initial_point` specification in VarCAUV.Weibull)

- RealRealMapArray `histogramUncBinPairs`
  An array for each real-valued bin-based histogram uncertain variable. Each array entry is a map from a real value to its probability (from the `histogram_bin_uncertain` specification in VarCAUV.Bin.Histogram; see also `continuous_linear` variable type in LHS manual). Note: bin widths may be unequal and any (x,c) count specifications are converted to (x,y) ordinates (probability densities) within `Vchk.HistogramBinUnc()` in the NIDR parser.

- RealVector `histogramBinUncVars`
  initial values of the histogram bin uncertain variables (from the `initial_point` specification in VarCAUV.Bin.
- Histogram)

- RealVector `poissonUncLambdas`
  lambdas (rate parameter) for the poisson uncertain variables (from the `lambdas` specification in VarDAUV.
- Poisson)

- IntVector `poissonUncVars`
  initial values of the poisson uncertain variables (from the `initial_point` specification in VarDAUV.Poisson)

- BitArray `poissonUncCat`
  is each poisson var strictly categorical (true) or relaxable (false)
• RealVector binomialUncProbPerTrial
  probabilities per each trial (p) for the binomial uncertain variables from the prob_per_trial specification in VarDAUV_Binomial)
• IntVector binomialUncNumTrials
  Number of trials (N) for the binomial uncertain variables from the num_trials specification in VarDAUV_Binomial)
• IntVector binomialUncVars
  initial values of the binomial uncertain variables (from the initial_point specification in VarDAUV_Binomial)
• BitArray binomialUncCat
  is each binomial var strictly categorical (true) or relaxable (false)
• RealVector negBinomialUncProbPerTrial
  probabilities per each trial (p) for the negative binomial uncertain variables from the prob_per_trial specification in VarDAUV_Negative_Binomial)
• IntVector negBinomialUncNumTrials
  Number of trials (N) for the negative binomial uncertain variables from the num_trials specification in VarDAUV_Negative_Binomial)
• IntVector negBinomialUncVars
  initial values of the negative binomial uncertain variables (from the initial_point specification in VarDAUV_Negative_Binomial)
• BitArray negBinomialUncCat
  is each negbinomial var strictly categorical (true) or relaxable (false)
• RealVector geometricUncProbPerTrial
  probabilities per each trial (p) for the geometric uncertain variables from the prob_per_trial specification in VarDAUV_Geometric)
• IntVector geometricUncVars
  initial values of the geometric uncertain variables (from the initial_point specification in VarDAUV_Geometric)
• BitArray geometricUncCat
  is each geometric var strictly categorical (true) or relaxable (false)
• IntVector hyperGeomUncTotalPop
  Size of total populations (N) for the hypergeometric uncertain variables from the total_population specification in VarDAUV_Hypergeometric)
• IntVector hyperGeomUncSelectedPop
  Size of selected populations for the hypergeometric uncertain variables from the selected_population specification in VarDAUV_Hypergeometric)
• IntVector hyperGeomUncNumDrawn
  Number failed in the selected populations for the hypergeometric variables from the num_drawn specification in VarDAUV_Hypergeometric)
• IntVector hyperGeomUncVars
  initial values of the hypergeometric uncertain variables (from the initial_point specification in VarDAUV_Hypergeometric)
• BitArray hyperGeomUncCat
  is each hypergeom var strictly categorical (true) or relaxable (false)
• IntRealMapArray histogramUncPointIntPairs
  An array for each integer-valued point-based histogram uncertain variable. Each array entry is a map from an integer value to its probability. (See discrete histogram in LHS manual; from the histogram_point_uncertain specification in VarDAUV_Point_Histogram)
• IntVector `histogramPointIntUncVars`
  initial values of the real-valued histogram point uncertain variables (from the `initial_point` specification in VarDAUV_Point_Histogram)

• BitArray `histogramUncPointIntCat`
  is each hupi var strictly categorical (true) or relaxable (false)

• StringRealMapArray `histogramUncPointStrPairs`
  An array for each string-valued point-based histogram uncertain variable. Each array entry is a map from a string value to its probability. (See discrete histogram in LHS manual; from the `histogram_point_uncertain` specification in VarDAUV_Point_Histogram)

• StringArray `histogramPointStrUncVars`
  initial values of the real-valued histogram point uncertain variables (from the `initial_point` specification in VarDAUV_Point_Histogram)

• RealRealMapArray `histogramUncPointRealPairs`
  An array for each real-valued point-based histogram uncertain variable. Each array entry is a map from a real value to its probability. (See discrete histogram in LHS manual; from the `histogram_point_uncertain` specification in VarDAUV_Point_Histogram)

• RealVector `histogramUncPointRealVars`
  initial values of the real-valued histogram point uncertain variables (from the `initial_point` specification in VarDAUV_Point_Histogram)

• BitArray `histogramUncPointRealCat`
  is each hupr var strictly categorical (true) or relaxable (false)

• RealSymMatrix `uncertainCorrelations`
  correlation matrix for all uncertain variables (from the `uncertain_correlation_matrix` specification in VarAUV_Correlations). This matrix specifies rank correlations for LHS sampling and correlation coefficients ($\rho_{ij} = $ normalized covariance matrix) for other methods.

• RealRealPairRealMapArray `continuousIntervalUncBasicProbs`
  Probability values per interval cell per epistemic interval uncertain variable (from the `continuous_interval_uncertain_interval_probs` specification in VarCEUV_Interval)

• RealVector `continuousIntervalUncVars`
  initial values of the continuous interval uncertain variables (from the `initial_point` specification in VarCEUV_Interval)

• IntIntPairRealMapArray `discreteIntervalUncBasicProbs`
  Probability values per interval cell per epistemic interval uncertain variable (from the `discrete_interval_uncertain_interval_probs` specification in VarDIUV)

• IntVector `discreteIntervalUncVars`
  initial values of the discrete interval uncertain variables (from the `initial_point` specification in VarDIUV)

• BitArray `discreteIntervalUncCat`
  is each diu var strictly categorical (true) or relaxable (false)

• IntRealMapArray `discreteUncSetIntValuesProbs`
  complete set of admissible values with associated basic probability assignments for each of the discrete epistemic uncertain variables defined by an integer set (from the `discrete_uncertain_set_integer_set_values` specification in VarDUSIV)

• IntVector `discreteUncSetIntVars`
  initial values of the discrete uncertain set integer variables (from the `initial_point` specification in VarDUSIV)

• BitArray `discreteUncSetIntCat`
  is each dusi var strictly categorical (true) or relaxable (false)
• StringRealMapArray \texttt{discreteUncSetStrValuesProbs}  
  complete set of admissible values with associated basic probability assignments for each of the discrete epistemic uncertain variables defined by a string set (from the \texttt{discrete.uncertain.set string set.values specification} in VarDUSIV)  
  
• StringArray \texttt{discreteUncSetStrVars}  
  initial values of the discrete uncertain set integer variables (from the \texttt{initial_point specification} in VarDUSIV)  
  
• RealRealMapArray \texttt{discreteUncSetRealValuesProbs}  
  complete set of admissible values with associated basic probability assignments for each of the discrete epistemic uncertain variables defined by a real set (from the \texttt{discrete.uncertain.set real set.values specification} in VarDUSRV)  
  
• RealVector \texttt{discreteUncSetRealVars}  
  initial values of the discrete uncertain set real variables (from the \texttt{initial_point specification} in VarDUSRV)  
  
• BitArray \texttt{discreteUncSetRealCat}  
  is each dusr var strictly categorical (true) or relaxable (false)  
  
• RealVector \texttt{continuousStateVars}  
  initial values for the continuous state variables array (from the \texttt{continuous.state initial_point specification} in VarSV)  
  
• RealVector \texttt{continuousStateLowerBnds}  
  lower bounds array for the continuous state variables (from the \texttt{continuous.state lower.bounds specification} in VarSV)  
  
• RealVector \texttt{continuousStateUpperBnds}  
  upper bounds array for the continuous state variables (from the \texttt{continuous.state upper.bounds specification} in VarSV)  
  
• IntVector \texttt{discreteStateRangeVars}  
  initial values for the discrete state variables defined by an integer range (from the \texttt{discrete.state.range initial_point specification} in VarSV)  
  
• IntVector \texttt{discreteStateRangeLowerBnds}  
  lower bounds array for the discrete state variables defined by an integer range (from the \texttt{discrete.state.range lower.bounds specification} in VarSV)  
  
• IntVector \texttt{discreteStateRangeUpperBnds}  
  upper bounds array for the discrete state variables defined by an integer range(from the \texttt{discrete.state.range upper.bounds specification} in VarSV)  
  
• BitArray \texttt{discreteStateRangeCat}  
  is each dsr var strictly categorical (true) or relaxable (false)  
  
• IntVector \texttt{discreteStateSetIntVars}  
  initial values for the discrete state variables defined by an integer set (from the \texttt{discrete.state.set integer initial_point specification} in VarSV)  
  
• StringArray \texttt{discreteStateSetStrVars}  
  initial values for the discrete state variables defined by a string set (from the \texttt{discrete.state.set string initial_point specification} in VarSV)  
  
• RealVector \texttt{discreteStateSetRealVars}  
  initial values for the discrete state variables defined by a real set (from the \texttt{discrete.state.set real initial_point specification} in VarSV)  
  
• IntSetArray \texttt{discreteStateSetInt}  
  complete set of admissible values for each of the discrete state variables defined by an integer set (from the \texttt{discrete.state.set integer set.values specification} in VarSV)
• **StringSetArray** `discreteStateSetStr`  
  complete set of admissible values for each of the discrete state variables defined by a string set (from the `discrete-state_set string set_values` specification in VarSV)

• **RealSetArray** `discreteStateSetReal`  
  complete set of admissible values for each of the discrete state variables defined by a real set (from the `discrete_state_set real set_values` specification in VarSV)

• **BitArray** `discreteStateSetIntCat`  
  is each dssi var strictly categorical (true) or relaxable (false)

• **BitArray** `discreteStateSetRealCat`  
  is each dssr var strictly categorical (true) or relaxable (false)

• **StringArray** `continuousStateLabels`  
  labels array for the continuous state variables (from the `continuous_state descriptors` specification in VarSV)

• **StringArray** `discreteStateRangeLabels`  
  labels array for the discrete state variables defined by an integer range (from the `discrete_state_range descriptors` specification in VarSV)

• **StringArray** `discreteStateSetIntLabels`  
  labels array for the discrete state variables defined by an integer set (from the `discrete_state_set descriptors` specification in VarSV)

• **StringArray** `discreteStateSetStrLabels`  
  labels array for the discrete state variables defined by a string set (from the `discrete_state_set descriptors` specification in VarSV)

• **StringArray** `discreteStateSetRealLabels`  
  labels array for the discrete state variables defined by a real set (from the `discrete_state_set descriptors` specification in VarSV)

• **IntVector** `discreteDesignSetIntLowerBnds`  
  discrete design integer set lower bounds inferred from set values

• **IntVector** `discreteDesignSetIntUpperBnds`  
  discrete design integer set upper bounds inferred from set values

• **StringArray** `discreteDesignSetStrLowerBnds`  
  discrete design string set lower bounds inferred from set values

• **StringArray** `discreteDesignSetStrUpperBnds`  
  discrete design string set upper bounds inferred from set values

• **RealVector** `discreteDesignSetRealLowerBnds`  
  discrete design real set lower bounds inferred from set values

• **RealVector** `discreteDesignSetRealUpperBnds`  
  discrete design real set upper bounds inferred from set values

• **RealVector** `continuousAleatoryUncVars`  
  array of values for all continuous aleatory uncertain variables

• **RealVector** `continuousAleatoryUncLowerBnds`  
  distribution lower bounds for all continuous aleatory uncertain variables (collected from `nuv_lower_bounds`, `lnuv_lower_bounds`, `uuv_lower_bounds`, `luuv_lower_bounds`, `tuv_lower_bounds`, and `buv_lower_bounds` specifications in VarAUV, and derived for gamma, gumbel, frechet, weibull and histogram bin specifications)

• **RealVector** `continuousAleatoryUncUpperBnds`
CHAPTER 14. CLASS DOCUMENTATION

distribution upper bounds for all continuous aleatory uncertain variables (collected from nuv_upper_bounds, lnuv_upper_bounds, uuv_upper_bounds, luvv_upper_bounds, tvv_lower_bounds, and buv_upper_bounds specifications in VarAUV, and derived for gamma, gumbel, frechet, weibull and histogram bin specifications)

- **StringArray** continuousAleatoryUncLabels
  labels for all continuous aleatory uncertain variables (collected from nuv_descriptors, lnuv_descriptors, uuv_descriptors, luvv_descriptors, tvv_descriptors, buv_descriptors, gauv_descriptors, guuv_descriptors, fuv_descriptors, wuv_descriptors, and hbuv_descriptors specifications in VarAUV)

- **IntVector** discreteIntAleatoryUncVars
  array of values for all discrete integer aleatory uncertain variables

- **IntVector** discreteIntAleatoryUncLowerBnds
  distribution lower bounds for all discrete integer aleatory uncertain variables

- **IntVector** discreteIntAleatoryUncUpperBnds
  distribution upper bounds for all discrete integer aleatory uncertain variables

- **StringArray** discreteIntAleatoryUncLabels
  labels for all discrete integer aleatory uncertain variables

- **StringArray** discreteIntAleatoryUncVars
  array of values for all discrete integer aleatory uncertain variables

- **StringArray** discreteIntAleatoryUncLowerBnds
  distribution lower bounds for all discrete integer aleatory uncertain variables

- **StringArray** discreteIntAleatoryUncUpperBnds
  distribution upper bounds for all discrete integer aleatory uncertain variables

- **StringArray** discreteIntAleatoryUncLabels
  labels for all discrete integer aleatory uncertain variables

- **RealVector** discreteRealAleatoryUncVars
  array of values for all discrete real aleatory uncertain variables

- **RealVector** discreteRealAleatoryUncLowerBnds
  distribution lower bounds for all discrete real aleatory uncertain variables

- **RealVector** discreteRealAleatoryUncUpperBnds
  distribution upper bounds for all discrete real aleatory uncertain variables

- **StringArray** discreteRealAleatoryUncLabels
  labels for all discrete real aleatory uncertain variables

- **RealVector** continuousEpistemicUncVars
  array of values for all continuous epistemic uncertain variables

- **RealVector** continuousEpistemicUncLowerBnds
  distribution lower bounds for all continuous epistemic uncertain variables

- **RealVector** continuousEpistemicUncUpperBnds
  distribution upper bounds for all continuous epistemic uncertain variables

- **StringArray** continuousEpistemicUncLabels
  labels for all continuous epistemic uncertain variables

- **IntVector** discreteIntEpistemicUncVars
  array of values for all discrete integer epistemic uncertain variables

- **IntVector** discreteIntEpistemicUncLowerBnds
  distribution lower bounds for all discrete integer epistemic uncertain variables

- **IntVector** discreteIntEpistemicUncUpperBnds
  distribution upper bounds for all discrete integer epistemic uncertain variables

- **StringArray** discreteIntEpistemicUncLabels
  labels for all discrete integer epistemic uncertain variables
distribution lower bounds for all discrete integer epistemic uncertain variables

- IntVector discreteIntEpistemicUncUpperBnds
distribution upper bounds for all discrete integer epistemic uncertain variables

- StringArray discreteIntEpistemicUncLabels
labels for all discrete integer epistemic uncertain variables

- StringArray discreteStrEpistemicUncVars
array of values for all discrete string epistemic uncertain variables

- StringArray discreteStrEpistemicUncLowerBnds
distribution lower bounds for all discrete string epistemic uncertain variables

- StringArray discreteStrEpistemicUncUpperBnds
distribution upper bounds for all discrete string epistemic uncertain variables

- StringArray discreteStrEpistemicUncLabels
labels for all discrete string epistemic uncertain variables

- RealVector discreteRealEpistemicUncVars
array of values for all discrete real epistemic uncertain variables

- RealVector discreteRealEpistemicUncLowerBnds
distribution lower bounds for all discrete real epistemic uncertain variables

- RealVector discreteRealEpistemicUncUpperBnds
distribution upper bounds for all discrete real epistemic uncertain variables

- StringArray discreteRealEpistemicUncLabels
labels for all discrete real epistemic uncertain variables

- IntVector discreteStateSetIntLowerBnds
discrete state integer set lower bounds inferred from set values

- IntVector discreteStateSetIntUpperBnds
discrete state integer set upper bounds inferred from set values

- StringArray discreteStateSetStrLowerBnds
discrete state string set lower bounds inferred from set values

- StringArray discreteStateSetStrUpperBnds
discrete state string set upper bounds inferred from set values

- RealVector discreteStateSetRealLowerBnds
discrete state real set lower bounds inferred from set values

- RealVector discreteStateSetRealUpperBnds
discrete state real set upper bounds inferred from set values

- RealVector linearIneqConstraintCoeffs
coefficient matrix for the linear inequality constraints (from the linear_inequality_constraint_matrix specification in MethodIndControl)

- RealVector linearIneqLowerBnds
lower bounds for the linear inequality constraints (from the linear_inequality_lower_bounds specification in MethodIndControl)

- RealVector linearIneqUpperBnds
upper bounds for the linear inequality constraints (from the linear_inequality_upper_bounds specification in MethodIndControl)

- StringArray linearIneqScaleTypes
CHAPTER 14. CLASS DOCUMENTATION

scaling types for the linear inequality constraints (from the linear_inequality_scale_types specification in MethodIndControl)

- RealVector linearIneqScales
  scaling factors for the linear inequality constraints (from the linear_inequality_scales specification in MethodIndControl)

- RealVector linearEqConstraintCoeffs
  coefficient matrix for the linear equality constraints (from the linear_equality_constraint_matrix specification in MethodIndControl)

- RealVector linearEqTargets
  targets for the linear equality constraints (from the linear_equality_targets specification in MethodIndControl)

- StringArray linearEqScaleTypes
  scaling types for the linear equality constraints (from the linear_equality_scale_types specification in MethodIndControl)

- RealVector linearEqScales
  scaling factors for the linear equality constraints (from the linear_equality_scales specification in MethodIndControl)

Private Member Functions

- DataVariablesRep()
  default constructor

- ~DataVariablesRep()
  destructor

- void write(std::ostream &s) const
  write a DataVariablesRep object to an std::ostream

- void read(MPIUnpackBuffer &s)
  read a DataVariablesRep object from a packed MPI buffer

- void write(MPIPackBuffer &s) const
  write a DataVariablesRep object to a packed MPI buffer

Private Attributes

- int referenceCount
  number of handle objects sharing dataVarsRep

Friends

- class DataVariables
  the handle class can access attributes of the body class directly
14.57.1 Detailed Description

Body class for variables specification data.

The `DataVariablesRep` class is used to contain the data from a variables keyword specification. Default values are managed in the `DataVariablesRep` constructor. Data is public to avoid maintaining set/get functions, but is still encapsulated within `ProblemDescDB` since `ProblemDescDB::dataVariablesList` is private.

The documentation for this class was generated from the following files:

- `DataVariables.hpp`
- `DataVariables.cpp`

14.58 DDACEDesignCompExp Class Reference

Wrapper class for the DDACE design of experiments library.

Inheritance diagram for DDACEDesignCompExp:

```
Iterator
   /\    
  /  
Analyzer
   /\    
  /  
PStudyDACE
   /\    
  /  
DDACEDesignCompExp
```

Public Member Functions

- `DDACEDesignCompExp (ProblemDescDB &problem_db, Model &model)`
  
  primary constructor for building a standard DACE iterator

- `DDACEDesignCompExp (Model &model, int samples, int symbols, int seed, unsigned short sampling_method)`
  
  alternate constructor used for building approximations

- `~DDACEDesignCompExp ()`
  
  destructor

- `bool resize ()`
  
  reinitializes iterator based on new variable size

Protected Member Functions

- `void pre_run ()`
  
  pre-run portion of run (optional); re-implemented by Iterators which can generate all Variables (parameter sets) a priori

- `void core_run ()`
  
  core portion of run; implemented by all derived classes and may include pre/post steps in lieu of separate pre/post

- `void post_input ()`
read tabular data for post-run mode

- void post_run (std::ostream &s)
  post-run portion of run (optional); verbose to print results; re-implemented by Iterators that can read all Variables/Responses and perform final analysis phase in a standalone way

- int num_samples () const
- void sampling_reset (int min_samples, bool all_data_flag, bool stats_flag)
  reset sampling iterator to use at least min_samples

- unsigned short sampling_scheme () const
  return sampling name

- void vary_pattern (bool pattern_flag)
  sets varyPattern in derived classes that support it

- void get_parameter_sets (Model &model)
  Generate one block of numSamples samples (ndim * num_samples), populating allSamples; ParamStudy is the only class that specializes to use allVariables.

- void get_parameter_sets (Model &model, const int num_samples, RealMatrix &design_matrix)
  Generate one block of numSamples samples (ndim * num_samples), populating design_matrix.

**Private Member Functions**

- void compute_main_effects ()
  builds a DDaceMainEffects::OneWayANOVA if mainEffectsFlag is set

- boost::shared_ptr < DDaceSamplerBase > create_sampler (Model &model)
  create a DDACE sampler

- void resolve_samples_symbols ()
  convenience function for resolving number of samples and number of symbols from input.

**Private Attributes**

- unsigned short daceMethod
  oas, lhs, oa_lhs, random, box_behnken, central composite, or grid

- int samplesSpec
  initial specification of number of samples

- int symbolsSpec
  initial specification of number of symbols

- int numSamples
  current number of samples to be evaluated

- int numSymbols
  current number of symbols to be used in generating the sample set (inversely related to number of replications)

- const int seedSpec
  the user seed specification for the random number generator (allows repeatable results)

- int randomSeed
  current seed for the random number generator

- bool allDataFlag
  flag which triggers the update of allVars/allResponses for use by Iterator::all_variables() and Iterator::all_responses()
• size_t numDACERuns
  counter for number of executions for this object
• bool varyPattern
  flag for continuing the random number sequence from a previous execution (e.g., for surrogate-based optimization)
  so that multiple executions are repeatable but not correlated.
• bool mainEffectsFlag
  flag which specifies main effects
• std::vector<std::vector<int>> symbolMapping
  mapping of symbols for main effects calculations

Additional Inherited Members

14.58.1 Detailed Description

Wrapper class for the DDACE design of experiments library.

The DDACEDesignCompExp class provides a wrapper for DDACE, a C++ design of experiments library from the Computational Sciences and Mathematics Research (CSMR) department at Sandia’s Livermore CA site. This class uses design and analysis of computer experiments (DACE) methods to sample the design space spanned by the bounds of a Model. It returns all generated samples and their corresponding responses as well as the best sample found.

14.58.2 Constructor & Destructor Documentation

DDACEDesignCompExp ( ProblemDescDB & problem_db, Model & model )

primary constructor for building a standard DACE iterator
  This constructor is called for a standard iterator built with data from probDescDB.
  References Dakota::abort_handler(), DDACEDesignCompExp::daceMethod, DDACEDesignCompExp::mainEffectsFlag, Iterator::maxEvalConcurrency, Analyzer::numContinuousVars, Analyzer::numDiscreteIntVars, Analyzer::numDiscreteRealVars, Analyzer::numDiscreteStringVars, and DDACEDesignCompExp::numSamples.

DDACEDesignCompExp ( Model & model, int samples, int symbols, int seed, unsigned short sampling_method )

alternate constructor used for building approximations
  This alternate constructor is used for instantiations on-the-fly, using only the incoming data. No problem description database queries are used.
  References Dakota::abort_handler(), Iterator::maxEvalConcurrency, Analyzer::numDiscreteIntVars, Analyzer::numDiscreteRealVars, Analyzer::numDiscreteStringVars, DDACEDesignCompExp::numSamples, and DDACEDesignCompExp::resolve_samples_symbols().

14.58.3 Member Function Documentation

void pre_run () [protected], [virtual]

pre-run portion of run (optional); re-implemented by Iterators which can generate all Variables (parameter sets) a priori
  pre-run phase, which a derived iterator may optionally reimplement; when not present, pre-run is likely integrated into the derived run function. This is a virtual function; when re-implementing, a derived class must call its nearest parent’s pre_run(), if implemented, typically before performing its own implementation steps.
Reimplemented from Analyzer.

References DDACEDesignCompExp::get_parameter_sets(), Analyzer::get_vbd_parameter_sets(), Iterator::iteratedModel, DDACEDesignCompExp::numSamples, Analyzer::pre_run(), DDACEDesignCompExp::resolve_samples_symbols(), and PStudyDACE::varBasedDecompFlag.

void core_run() [protected], [virtual]
core portion of run; implemented by all derived classes and may include pre/post steps in lieu of separate pre/post
Virtual run function for the iterator class hierarchy. All derived classes need to redefine it.
Reimplemented from Iterator.
References DDACEDesignCompExp::allDataFlag, Analyzer::evaluate_parameter_sets(), Iterator::iteratedModel, DDACEDesignCompExp::mainEffectsFlag, Analyzer::numLSqTerms, Analyzer::numObjFns, and Iterator::subIteratorFlag.

void post_run(std::ostream &s) [protected], [virtual]
post-run portion of run (optional); verbose to print results; re-implemented by Iterators that can read all Variables/Responses and perform final analysis phase in a standalone way
Post-run phase, which a derived iterator may optionally reimplement; when not present, post-run is likely integrated into run. This is a virtual function; when re-implementing, a derived class must call its nearest parent’s post_run(), typically after performing its own implementation steps.
Reimplemented from Analyzer.
References Dakota::abort_handler(), Analyzer::allResponses, Analyzer::allSamples, SensAnalysisGlobal::compute_correlations(), DDACEDesignCompExp::compute_main_effects(), Analyzer::compute_vbd_stats(), DDACEDesignCompExp::create_sampler(), Iterator::iteratedModel, DDACEDesignCompExp::mainEffectsFlag, DDACEDesignCompExp::numSamples, Analyzer::post_run(), PStudyDACE::pStudyDACEsensGlobal, DDACEDesignCompExp::seedSpec, Iterator::subIteratorFlag, DDACEDesignCompExp::symbolMapping, and PStudyDACE::varBasedDecompFlag.

int num_samples() const [inline], [protected], [virtual]
Return current number of evaluation points. Since the calculation of samples, collocation points, etc. might be costly, provide a default implementation here that backs out from the maxEvalConcurrency.
Reimplemented from Analyzer.
References DDACEDesignCompExp::numSamples.
Referenced by DDACEDesignCompExp::get_parameter_sets().

void resolve_samples_symbols() [private]
convenience function for resolving number of samples and number of symbols from input.
This function must define a combination of samples and symbols that is acceptable for a particular sampling algorithm. Users provide requests for these quantities, but this function must enforce any restrictions imposed by the sampling algorithms.
References Dakota::abort_handler(), DDACEDesignCompExp::daceMethod, Analyzer::numContinuousVars, DDACEDesignCompExp::numSamples, DDACEDesignCompExp::numSymbols, and Iterator::submethod_enum_to_string().
Referenced by DDACEDesignCompExp::DDACEDesignCompExp(), DDACEDesignCompExp::post_input(), and DDACEDesignCompExp::pre_run().
The documentation for this class was generated from the following files:
• DDACEDesignCompExp.hpp
14.59 DerivInformedPropCovLogitTK < V, M > Class Template Reference

Dakota transition kernel that updates proposal covariance based on derivatives (for logit random walk case)
Inherits TransformedScaledCovMatrixTKGroup< V, M >.

Public Member Functions

- DerivInformedPropCovLogitTK (const char *prefix, const QUESO::VectorSet< V, M >& vectorSet, const std::vector<double>& scales, const M &covMatrix, NonDQUESOBayesCalibration *queso_instance)
  Constructor for derivative-informed logit proposal covariance.
- virtual ~DerivInformedPropCovLogitTK ()
  Destructor for derivative-informed logit proposal covariance.
- virtual void updateTK () override
- virtual bool covMatrixIsDirty () override
  whether the covariance matrix has been updated
- virtual void cleanCovMatrix () override
  dependent algorithms have taken necessary cleanup actions

Private Attributes

- const QUESO::VectorSet< V, M >& m_vectorSet
  calibration parameter vector set (note: hides base class member)
- bool covIsDirty
  whether we’ve updated the proposal covariance
- unsigned int chainIndex
  index into current chain position
- NonDQUESOBayesCalibration * nonDQUESOInstance
  Dakota QUESO instance for callbacks.

14.59.1 Detailed Description

template<class V, class M> class Dakota::DerivInformedPropCovLogitTK< V, M >

Dakota transition kernel that updates proposal covariance based on derivatives (for logit random walk case)
The documentation for this class was generated from the following files:
- NonDQUESOBayesCalibration.hpp
- QUESOImpl.hpp
- QUESOImpl.cpp

14.60 DerivInformedPropCovTK < V, M > Class Template Reference

Dakota transition kernel that updates proposal covariance based on derivatives (for random walk case)
Inherits ScaledCovMatrixTKGroup< V, M >.
Public Member Functions

- **DerivInformedPropCovTK** (const char *prefix, const QUESO::VectorSpace<V, M> &vectorSpace, const std::vector<double> &scales, const M &covMatrix, NonDQUESOBayesCalibration *queso_instance)
  
  Constructor for derivative-informed proposal covariance.

- **virtual ~DerivInformedPropCovTK ()**
  
  Destructor for derivative-informed proposal covariance.

- **virtual void updateTK () override**
  
  update the transition kernel with new covariance information

- **virtual bool covMatrixIsDirty () override**
  
  whether the covariance matrix has been updated

- **virtual void cleanCovMatrix () override**
  
  dependent algorithms have taken necessary cleanup actions

Private Attributes

- **const QUESO::VectorSpace<V, M> & m_vectorSpace**
  
  calibration parameter vector space (note: hides base class member)

- **bool covIsDirty**
  
  whether we've updated the proposal covariance

- **unsigned int chainIndex**
  
  index into current chain position

- **NonDQUESOBayesCalibration * nonDQUESOInstance**
  
  Dakota QUESO instance for callbacks.

14.60.1 Detailed Description

template<class V, class M> class Dakota::DerivInformedPropCovTK<V, M>

Dakota transition kernel that updates proposal covariance based on derivatives (for random walk case)

The documentation for this class was generated from the following files:

- NonDQUESOBayesCalibration.hpp
- QUESOImpl.hpp
- QUESOImpl.cpp

14.61 DirectApplicInterface Class Reference

Derived application interface class which spawns simulation codes and testers using direct procedure calls.

Inheritance diagram for DirectApplicInterface:
Public Member Functions

- **DirectApplicInterface** (const ProblemDescDB &problem_db)
  - constructor
- **~DirectApplicInterface**()
  - destructor
- void **derived_map** (const Variables &vars, const ActiveSet &set, Response &response, int fn_eval_id)
  - Called by map() and other functions to execute the simulation in synchronous mode. The portion of performing an evaluation that is specific to a derived class.
- void **derived_map_asynch** (const ParamResponsePair &pair)
  - Called by map() and other functions to execute the simulation in asynchronous mode. The portion of performing an asynchronous evaluation that is specific to a derived class.
- void **wait_local_evaluations** (PRPQueue &prp_queue)
  - For asynchronous function evaluations, this method is used to detect completion of jobs and process their results. It provides the processing code that is specific to derived classes. This version waits for at least one completion.
- void **test_local_evaluations** (PRPQueue &prp_queue)
  - For asynchronous function evaluations, this method is used to detect completion of jobs and process their results. It provides the processing code that is specific to derived classes. This version is nonblocking and will return without any completions if none are immediately available.
- int **synchronous_local_analysis** (int analysis_id)
- const StringArray & **analysis_drivers** () const
  - retrieve the analysis drivers specification for application interfaces
- void **init_communicators_checks** (int max_eval_concurrency)
- void **set_communicators_checks** (int max_eval_concurrency)

Protected Member Functions

- virtual int **derived_map_if** (const Dakota::String &if_name)
  - execute the input filter portion of a direct evaluation invocation
- virtual int **derived_map_ac** (const Dakota::String &ac_name)
  - execute an analysis code portion of a direct evaluation invocation
- virtual int **derived_map_of** (const Dakota::String &of_name)
  - execute the output filter portion of a direct evaluation invocation
- virtual void **set_local_data** (const Variables &vars, const ActiveSet &set)
  - convenience function for local test simulators which sets per-evaluation variable and active set attributes; derived classes reimplementing this likely need to invoke the base class API
- virtual void **set_local_data** (const Response &response)
  - convenience function for local test simulators which sets per-evaluation response attributes; derived classes reimplementing this likely need to invoke the base class API
- virtual void **set_local_data** (const Variables &vars, const ActiveSet &set, const Response &response)
  - convenience function for local test simulators which sets per-evaluation variable, active set, and response attributes; derived classes reimplementing this likely need to invoke the base class API
- void **overlay_response** (Response &response)
  - convenience function for local test simulators which overlays response contributions from multiple analyses using MPI_Reduce
Protected Attributes

- String iFilterName
  name of the direct function input filter
- String oFilterName
  name of the direct function output filter
- driver_t iFilterType
  enum type of the direct function input filter
- driver_t oFilterType
  enum type of the direct function output filter
- bool gradFlag
  signals use of fnGrads in direct simulator functions
- bool hessFlag
  signals use of fnHessians in direct simulator functions
- size_t numFns
  number of functions in fnVals
- size_t numVars
  total number of continuous and discrete variables
- size_t numACV
  total number of continuous variables
- size_t numADIV
  total number of discrete integer variables
- size_t numADRV
  total number of discrete real variables
- size_t numADSV
  total number of discrete string variables
- size_t numDerivVars
  number of active derivative variables
- unsigned short localDataView
  bit-wise record of which local data views are active; see enum local_data_t
- RealVector xC
  continuous variables used within direct simulator fns
- IntVector xDI
  discrete int variables used within direct simulator fns
- RealVector xDR
  discrete real variables used within direct simulator fns
- StringMultiArray xDS
  discrete string variables used within direct simulator fns
- StringMultiArray xCLabels
  continuous variable labels
- StringMultiArray xDILabels
  discrete integer variable labels
- StringMultiArray xDRLabels
discrete real variable labels
- `StringMultiArray xDSLLabels`

discrete string variable labels
- `std::map< String, var_t > varTypeMap`
  - `map from variable label to enum`
- `std::map< String, driver_t > driverTypeMap`
  - `map from driver name to enum`
- `std::map< var_t, Real > xCM`
  - `map from var_t enum to continuous value`
- `std::map< var_t, int > xDIM`
  - `map from var_t enum to discrete int value`
- `std::map< var_t, Real > xDRM`
  - `map from var_t enum to discrete real value`
- `std::map< var_t, String > xDSM`
  - `map from var_t enum to discrete string value`
- `std::vector< var_t > varTypeDVV`
  - `var_t enumerations corresponding to DVV components`
- `std::vector< var_t > xCMLabels`
  - `var_t enumerations corresponding to continuous variable labels`
- `std::vector< var_t > xDIMLabels`
  - `var_t enumerations corresponding to discrete integer variable labels`
- `std::vector< var_t > xDRMLabels`
  - `var_t enumerations corresponding to discrete real variable labels`
- `std::vector< var_t > xDSMLabels`
  - `var_t enumerations corresponding to discrete string variable labels`
- `ShortArray directFnASV`
  - `class scope active set vector`
- `SizetArray directFnDVV`
  - `class scope derivative variables vector`
- `RealVector fnVals`
  - `response fn values within direct simulator fns`
- `RealMatrix fnGrads`
  - `response fn gradients w/i direct simulator fns`
- `RealSymMatrixArray fnHessians`
  - `response fn Hessians within direct fns`
- `StringArray analysisDrivers`
  - `the set of analyses within each function evaluation (from the analysis_drivers interface specification)`
- `std::vector< driver_t > analysisDriverTypes`
  - `conversion of analysisDrivers to driver_t`
- `size_t analysisDriverIndex`
  - `the index of the active analysis driver within analysisDrivers`
Private Member Functions

- void map_labels_to_enum (StringMultiArrayConstView &src, std::vector< var_t >&dest)
  
  map labels in src to var_t in dest

14.61.1 Detailed Description

Derived application interface class which spawns simulation codes and testers using direct procedure calls. 

DirectApplicInterface uses a few linkable simulation codes and several internal member functions to perform parameter to response mappings.

14.61.2 Member Function Documentation

int synchronous_local_analysis ( int analysis_id ) [inline], [virtual]

This code provides the derived function used by ApplicationInterface::serve_analyses_synch().

Reimplemented from ApplicationInterface.

References DirectApplicInterface::analysisDriverIndex, DirectApplicInterface::analysisDrivers, and DirectApplicInterface::derived_map_ac().

void init_communicators_checks ( int max_eval_concurrency ) [inline], [virtual]

Process init issues as warnings since some contexts (e.g., HierarchSurrModel) initialize more configurations than will be used and DirectApplicInterface allows override by derived plug-ins.

Reimplemented from ApplicationInterface.

References ApplicationInterface::check_asynchronous(), and ApplicationInterface::check_multiprocessor_asynchronous().

void set_communicators_checks ( int max_eval_concurrency ) [inline], [virtual]

Process run-time issues as hard errors.

Reimplemented from ApplicationInterface.

Reimplemented in SerialDirectApplicInterface, SoleilDirectApplicInterface, and ParallelDirectApplicInterface.

References Dakota::abort_handler(), ApplicationInterface::check_asynchronous(), and ApplicationInterface::check_multiprocessor_asynchronous().

int derived_map_ac ( const Dakota::String & ac_name ) [protected], [virtual]

execute an analysis code portion of a direct evaluation invocation

When a direct analysis/filter is a member function, the (vars,set,response) data does not need to be passed through the API. If, however, non-member analysis/filter functions are added, then pass (vars,set,response) through to the non-member fn:

// API declaration
int sim(const Variables& vars, const ActiveSet& set, Response& response);
// use of API within derived_map_ac()
if (ac_name == "sim")
  fail_code = sim(directFnVars, directFnActSet, directFnResponse);

Reimplemented in SerialDirectApplicInterface, SoleilDirectApplicInterface, ParallelDirectApplicInterface, and TestDriverInterface.

References Dakota::abort_handler(), and ApplicationInterface::analysisServerId.

Referenced by DirectApplicInterface::derived_map(), and DirectApplicInterface::synchronous_local_analysis().

The documentation for this class was generated from the following files:
DirectApplicInterface.hpp
DirectApplicInterface.cpp

14.62 DiscrepancyCorrection Class Reference

Base class for discrepancy corrections.

Public Member Functions

- DiscrepancyCorrection()
  \textit{default constructor}

- DiscrepancyCorrection (Model &surr_model, const IntSet &surr_fn_indices, short corr_type, short corr_order)
  \textit{standard constructor}

- DiscrepancyCorrection (const IntSet &surr_fn_indices, size_t num_fns, size_t num_vars, short corr_type, short corr_order)
  \textit{alternate constructor}

- \texttt{~DiscrepancyCorrection()}
  \textit{destructor}

- void initialize (Model &surr_model, const IntSet &surr_fn_indices, short corr_type, short corr_order)
  \textit{initialize the DiscrepancyCorrection data}

- void initialize (const IntSet &surr_fn_indices, size_t num_fns, size_t num_vars, short corr_type, short corr_order)
  \textit{initialize the DiscrepancyCorrection data}

- void initialize (const IntSet &surr_fn_indices, size_t num_fns, size_t num_vars, short corr_type, short corr_order, const String &approx_type)
  \textit{initialize the DiscrepancyCorrection data}

- void compute (const Variables &vars, const Response &truth_response, const Response &approx_response, bool quiet_flag=false)
  \textit{compute the correction required to bring approx\_response into agreement with truth\_response and store in \{add,mult\} Corrections}

- void compute (const Response &truth_response, const Response &approx_response, Response &discrepancy_response, bool quiet_flag=false)
  \textit{compute the correction required to bring approx\_response into agreement with truth\_response as a function of the variables and store in discrepancy\_response}

- void compute (const VariablesArray &vars_array, const ResponseArray &truth_response_array, const ResponseArray &approx_response, bool quiet_flag=false)
  \textit{compute the correction required to bring approx\_response into agreement with truth\_response as a function of the variables and store in \{add,mult\} Corrections}

- void apply (const Variables &vars, Response &approx_response, bool quiet_flag=false)
  \textit{apply the correction computed in compute() to approx\_response}

- void compute\_variance (const VariablesArray &vars_array, RealMatrix &approx\_variance, bool quiet_flag=false)
  \textit{compute the variance of approx\_response}

- void correction\_type (short corr\_type)
  \textit{update correctionType}

- short correction\_type () const
return correctionType

- void correction_order (short order)
  update correctionOrder
- short correction_order () const
  return correctionOrder
- void data_order (short order)
  update dataOrder
- short data_order () const
  return dataOrder
- bool computed () const
  return correctionComputed
- bool initialized () const
  return initializedFlag

Protected Attributes

- IntSet surrogateFnIndices
  for mixed response sets, this array specifies the response function subset that is approximated
- bool initializedFlag
  indicates that discrepancy correction instance has been initialized following construction
- short correctionType
  approximation correction approach to be used: NO_CORRECTION, ADDITIVE_CORRECTION, MULTIPLICATIVE_CORRECTION, or COMBINED_CORRECTION.
- short correctionOrder
  approximation correction order to be used: 0, 1, or 2
- short dataOrder
  order of correction data in 3-bit format: overlay of 1 (value), 2 (gradient), and 4 (Hessian)
- bool correctionComputed
  flag indicating whether or not a correction has been computed and is available for application
- size_t numFns
  total number of response functions (of which surrogateFnIndices may define a subset)
- size_t numVars
  number of continuous variables active in the correction

Private Member Functions

- void initialize_corrections ()
  internal convenience function shared by overloaded initialize() variants
- bool check_multiplicative (const RealVector &truth_fns, const RealVector &approx_fns)
  define badScalingFlag
- void compute_additive (const Response &truth_response, const Response &approx_response, int index, Real &discrep_fn, RealVector &discrep_grad, RealSymMatrix &discrep_hess)
  internal convenience function for computing additive corrections between truth and approximate responses
- void compute_multiplicative (const Response &truth_response, const Response &approx_response, int index, Real &discrep_fn, RealVector &discrep_grad, RealSymMatrix &discrep_hess)
internal convenience function for computing multiplicative corrections between truth and approximate responses

- void apply_additive (const Variables &vars, Response &approx_response)
  
  internal convenience function for applying additive corrections to an approximate response

- void apply_multiplicative (const Variables &vars, Response &approx_response)
  
  internal convenience function for applying multiplicative corrections to an approximate response

- void apply_additive (const Variables &vars, RealVector &approx_fns)
  
  internal convenience function for applying additive corrections to a set of response functions

- void apply_multiplicative (const Variables &vars, RealVector &approx_fns)
  
  internal convenience function for applying multiplicative corrections to a set of response functions

- const Response & search_db (const Variables &search_vars, const ShortArray &search_asv)
  search data pairs for missing approximation data

Private Attributes

- Pecos::DiscrepancyCalculator discrepCalc
  helper utility for calculating discrepancies
- bool badScalingFlag
  flag used to indicate function values near zero for multiplicative corrections; triggers an automatic switch to additive corrections
- bool computeAdditive
  flag indicating the need for additive correction calculations
- bool computeMultiplicative
  flag indicating the need for multiplicative correction calculations
- String approxType
  string indicating the discrepancy approximation type
- SharedApproxData sharedData
  data that is shared among all correction Approximations
- std::vector< Approximation > addCorrections
  array of additive corrections; surrogate models of a model discrepancy function (formed from model differences)
- std::vector< Approximation > multCorrections
  array of multiplicative corrections; surrogate models of a model discrepancy function (formed from model ratios)
- Model surrModel
  shallow copy of the surrogate model instance as returned by Model::surrogate_model() (the DataFitSurrModel or HierarchSurrModel::lowFidelityModel instance)
- RealVector combineFactors
  factors for combining additive and multiplicative corrections. Each factor is the weighting applied to the additive correction and 1.-factor is the weighting applied to the multiplicative correction. The factor value is determined by an additional requirement to match the high fidelity function value at the previous correction point (e.g., previous trust region center). This results in a multipoint correction instead of a strictly local correction.
- Variables correctionPrevCenterPt
  copy of center point from the previous correction cycle
- RealVector truthFnsCenter
  truth function values at the current correction point
- RealVector approxFnsCenter
Surrogate function values at the current correction point.

- RealMatrix approxGradsCenter

Surrogate gradient values at the current correction point.

- RealVector truthFnsPrevCenter
  
  copy of truth function values at center of previous correction cycle

- RealVector approxFnsPrevCenter
  
  copy of approximate function values at center of previous correction cycle

## 14.62.1 Detailed Description

Base class for discrepancy corrections.

The **DiscrepancyCorrection** class provides common functions for computing and applying corrections to approximations.

## 14.62.2 Member Function Documentation

```cpp
void compute ( const Variables & vars,
               const Response & truth_response,
               const Response & approx_response,
               bool quiet_flag = false )
```

compute the correction required to bring **approx_response** into agreement with **truth_response** and store in \{add,mult\}Corrections

Compute an additive or multiplicative correction that corrects the **approx_response** to have 0th-order consistency (matches values), 1st-order consistency (matches values and gradients), or 2nd-order consistency (matches values, gradients, and Hessians) with the **truth_response** at a single point (e.g., the center of a trust region). The 0th-order, 1st-order, and 2nd-order corrections use scalar values, linear scaling functions, and quadratic scaling functions, respectively, for each response function.

References
  - Response::active_set(), DiscrepancyCorrection::addCorrections, DiscrepancyCorrection::apply(), DiscrepancyCorrection::apply_additive(), DiscrepancyCorrection::apply_multiplicative(), DiscrepancyCorrection::approxFnsCenter, DiscrepancyCorrection::approxFnsPrevCenter, DiscrepancyCorrection::approxGradsCenter, DiscrepancyCorrection::approxType, DiscrepancyCorrection::badScalingFlag, DiscrepancyCorrection::combineFactors, DiscrepancyCorrection::computeAdditive, DiscrepancyCorrection::computeMultiplicative, Variables::continuous_variables(), Response::copy(), DiscrepancyCorrection::correctionComputed, DiscrepancyCorrection::correctionOrder, DiscrepancyCorrection::correctionPrevCenterPt, DiscrepancyCorrection::correctionType, DiscrepancyCorrection::dataOrder, DiscrepancyCorrection::disrepCalc, Variables::discrete_int_variables(), Variables::discrete_real_variables(), Response::function_gradient_view(), Response::function_gradients(), Response::function_hessian(), Response::function_value(), Response::function_values(), Model::is_null(), DiscrepancyCorrection::multCorrections, DiscrepancyCorrection::numFns, DiscrepancyCorrection::numVars, ActiveSet::request_values(), DiscrepancyCorrection::sharedData, DiscrepancyCorrection::surrModel, DiscrepancyCorrection::surrFnsIndices, DiscrepancyCorrection::truthFnsCenter, and DiscrepancyCorrection::truthFnsPrevCenter.

Referenced by
  - DiscrepancyCorrection::compute(), HierarchSurrModel::compute_apply_delta(), DataFitSurrModel::compute_apply_delta(), DataFitSurrModel::comput_evaluate(), DataFitSurrModel::compute_synchronize(). HierarchSurrModel::deried_synchronize(), HierarchSurrModel::derived_synchronize_combine(), DataFitSurrModel::derived_synchronize_nowait(), and HierarchSurrModel::single_apply().

The documentation for this class was generated from the following files:

- DiscrepancyCorrection.hpp
- DiscrepancyCorrection.cpp
14.63 DLSolverTraits Class Reference

A version of TraitsBase specialized for DLSolver.

Inheritance diagram for DLSolverTraits:

```
TraitsBase
  DLSolverTraits
```

Public Member Functions

- **DLSolverTraits ()**
  
  *default constructor*

- **virtual ~DLSolverTraits ()**
  
  *destructor*

- **virtual bool is_derived ()**
  
  *A temporary query used in the refactor.*

- **bool supports_continuous_variables ()**
  
  *Return the flag indicating whether method supports continuous variables.*

- **bool supports_linear_equality ()**
  
  *Return the flag indicating whether method supports linear equalities.*

- **bool supports_linear_inequality ()**
  
  *Return the flag indicating whether method supports linear inequalities.*

- **bool supports_nonlinear_equality ()**
  
  *Return the flag indicating whether method supports nonlinear equalities.*

- **bool supports_nonlinear_inequality ()**
  
  *Return the flag indicating whether method supports nonlinear inequalities.*

- **NONLINEAR_INEQUALITY_FORMAT nonlinear_inequality_format ()**
  
  *Return the format used for nonlinear inequality constraints.*

14.63.1 Detailed Description

A version of TraitsBase specialized for DLSolver.

The documentation for this class was generated from the following file:

- DLSolver.hpp

14.64 DOTTraits Class Reference

Wrapper class for the DOT optimization library.

Inheritance diagram for DOTTraits:

```
DOTTraits
```
Public Member Functions

- **DOTTraits ()**
  
  *default constructor*

- **virtual ~DOTTraits ()**
  
  *destructor*

- **virtual bool is_derived ()**
  
  *A temporary query used in the refactor.*

- **bool supports_continuous_variables ()**
  
  *Return the flag indicating whether method supports continuous variables.*

- **bool supports_linear_equality ()**
  
  *Return the flag indicating whether method supports linear equalities.*

- **bool supports_linear_inequality ()**
  
  *Return the flag indicating whether method supports linear inequalities.*

- **LINEAR_INEQUALITY_FORMAT linear_inequality_format ()**
  
  *Return the format used for linear inequality constraints.*

- **bool supports_nonlinear_equality ()**
  
  *Return the flag indicating whether method supports nonlinear equalities.*

- **bool supports_nonlinear_inequality ()**
  
  *Return the flag indicating whether method supports nonlinear inequalities.*

- **NONLINEAR_INEQUALITY_FORMAT nonlinear_inequality_format ()**
  
  *Return the format used for nonlinear inequality constraints.*

14.64.1 Detailed Description

Wrapper class for the DOT optimization library.

The DOTOptimizer class provides a wrapper for DOT, a commercial Fortran 77 optimization library from Vanderplaats Research and Development. It uses a reverse communication mode, which avoids the static member function issues that arise with function pointer designs (see NPSOLOptimizer and SNLLOptimizer).

The user input mappings are as follows: max_iterations is mapped into DOT’s ITMAX parameter within its IPRM array, max_function_evaluations is implemented directly in the core_run() loop since there is no DOT parameter equivalent, convergence_tolerance is mapped into DOT’s DELOBJ parameter (the relative convergence tolerance) within its RPRM array, output verbosity is mapped into DOT’s IPRINT parameter within its function call parameter list (verbose: IPRINT = 7; quiet: IPRINT = 3), and optimization_type is mapped into DOT’s MINMAX parameter within its function call parameter list. Refer to [Vanderplaats Research and Development, 1995] for information on IPRM, RPRM, and the DOT function call parameter list. A version of TraitsBase specialized for DOT optimizers

The documentation for this class was generated from the following file:

- DOTOptimizer.hpp
14.65 JEGAOptimizer::Driver Class Reference

A subclass of the JEGA front end driver that exposes the individual protected methods to execute the algorithm. Inherits Driver.

Public Member Functions

- GeneticAlgorithm * ExtractAllData (const AlgorithmConfig &algConfig)
  
  Reads all required data from the problem description database stored in the supplied algorithm config.

- DesignOFSortSet PerformIterations (GeneticAlgorithm *theGA)
  
  Performs the required iterations on the supplied GA.

- void DestroyAlgorithm (GeneticAlgorithm *theGA)
  
  Deletes the supplied GA.

- Driver (const ProblemConfig &probConfig)
  
  Default constructs a Driver.

14.65.1 Detailed Description

A subclass of the JEGA front end driver that exposes the individual protected methods to execute the algorithm.

This is necessary because DAKOTA requires that all problem information be extracted from the problem description DB at the time of Optimizer construction and the front end does it all in the execute algorithm method which must be called in core_run.

14.65.2 Constructor & Destructor Documentation

Driver ( const ProblemConfig & probConfig ) [inline]

Default constructs a Driver.

Parameters

| probConfig | The definition of the problem to be solved by this Driver whenever ExecuteAlgorithm is called. |

The problem can be solved in multiple ways by multiple algorithms even using multiple different evaluators by issuing multiple calls to ExecuteAlgorithm with different AlgorithmConfigs.

14.65.3 Member Function Documentation

GeneticAlgorithm* ExtractAllData ( const AlgorithmConfig & algConfig ) [inline]

Reads all required data from the problem description database stored in the supplied algorithm config.

The returned GA is fully configured and ready to be run. It must also be destroyed at some later time. You MUST call DestroyAlgorithm for this purpose. Failure to do so could result in a memory leak and an eventual segmentation fault! Be sure to call DestroyAlgorithm prior to destroying the algorithm config that was used to create it!

This is just here to expose the base class method to users.
Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>algConfig</td>
<td>The fully loaded configuration object containing the database of parameters for the algorithm to be run on the known problem.</td>
</tr>
</tbody>
</table>

Returns

The fully configured and loaded GA ready to be run using the PerformIterations method.

Referenced by JEGAOptimizer::core_run().

DesignOFSortSet PerformIterations ( GeneticAlgorithm *theGA ) [inline]

Performs the required iterations on the supplied GA.

This includes the calls to AlgorithmInitialize and AlgorithmFinalize and logs some information if appropriate.

This is just here to expose the base class method to users.

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>theGA</td>
<td>The GA on which to perform iterations. This parameter must be non-null.</td>
</tr>
</tbody>
</table>

Returns

The final solutions reported by the supplied GA after all iterations and call to AlgorithmFinalize.

Referenced by JEGAOptimizer::core_run().

void DestroyAlgorithm ( GeneticAlgorithm *theGA ) [inline]

Deletes the supplied GA.

Use this method to destroy a GA after all iterations have been run. This method knows if the log associated with the GA was created here and needs to be destroyed as well or not.

This is just here to expose the base class method to users.

Be sure to use this prior to destroying the algorithm config object which contains the target. The GA destructor needs the target to be in tact.

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>theGA</td>
<td>The algorithm that is no longer needed and thus must be destroyed.</td>
</tr>
</tbody>
</table>

Referenced by JEGAOptimizer::core_run().

The documentation for this class was generated from the following file:

- JEGAOptimizer.cpp

### 14.66 EffGlobalMinimizer Class Reference

Inheritance diagram for EffGlobalMinimizer:
Public Member Functions

- **EffGlobalMinimizer (ProblemDescDB &problem_db, Model &model)**
  standard constructor
- **EffGlobalMinimizer ()**
  alternate constructor for instantiations "on the fly"
- **void core_run ()**
  destructor
- **const Model & algorithm_space_model () const**
- **void declare_sources ()**
  Declare sources to the evaluations database.

Private Member Functions

- **void minimize_surrogates_on_model ()**
  called by minimize_surrogates for setUpType == "model"
- **void get_best_sample ()**
  called by minimize_surrogates for setUpType == "user functions"
- **void initialize ()**
  initialize
- **Real get_augmented_lagrangian (const RealVector &mean, const RealVector &c_vars, const Real &eif_star)**
  augmented Lagrangian augmented Lagrangian
- **void check_convergence (const Real &eif_star, const RealVector &c_vars, RealVector prev_cv_star, unsigned short eif_convergence_cntr, unsigned short dist_convergence_cntr)**
  check convergence if EGO has converged
- **void debug_print_values ()**
  print mean and variance if debug flag is ON
- **void debug_print_counter (unsigned short globalIterCount, const Real &eif_star, Real distCStar, unsigned short dist_convergence_cntr)**
  print counter if debug flag is ON
- **void debug_plots ()**
- **Real expected_improvement (const RealVector &means, const RealVector &variances)**
  expected improvement function for the GP
- **RealVector expected_violation (const RealVector &means, const RealVector &variances)**
expected violation function for the constraint functions

- **void update_penalty ()**
  initialize and update the penaltyParameter

**Static Private Member Functions**

- **static void EIF_objective_eval (const Variables &sub_model_vars, const Variables &recast_vars, const Response &sub_model_response, Response &recast_response)**
  static function used as the objective function in the Expected Improvement (EIF) problem formulation for PMA

**Private Attributes**

- **String setUpType**
  controls iteration mode: "model" (normal usage) or "user functions" (user-supplied functions mode for "on the fly" instantiations).

- **Real distanceTol**
  convergence tolerance on distance between predicted best points

- **Real convergenceTol**
  convergence tolerances in objectives

- **Model fHatModel**
  GP model of response, one approximation per response function.

- **Model eifModel**
  recast model which assimilates mean and variance to solve the max(EIF) sub-problem

- **Real meritFnStar**
  minimum penalized response from among true function evaluations

- **RealVector truthFnStar**
  true function values corresponding to the minimum penalized response

- **RealVector varStar**
  point that corresponds to the optimal value meritFnStar

- **short dataOrder**
  order of the data used for surrogate construction, in ActiveSet request vector 3-bit format; user may override responses spec

- **Real distCStar**
  declare check convergence variables relative distance change in input measured in L2

- **int BatchSizeAcquisition**
  declare batch size for BatchSizeAcquisition sampling point located at maximum acquisition function

- **int BatchSizeExploration**
  declare batch size for BatchSizeExploration sampling point located at maximum posterior variance

**Static Private Attributes**

- **static EffGlobalMinimizer * effGlobalInstance**
  pointer to the active object instance used within the static evaluator functions in order to avoid the need for static data
Additional Inherited Members

14.66.1 Detailed Description

The **EffGlobalMinimizer** class provides an implementation of the Efficient Global Optimization algorithm developed by Jones, Schonlau, & Welch as well as adaptation of the concept to nonlinear least squares.

14.66.2 Member Function Documentation

**const Model & algorithm.space_model( ) const** [inline], [virtual]

default definition that gets redefined in selected derived Minimizers

- Reimplemented from **Minimizer**.
- References **EffGlobalMinimizer::fHatModel**.

**void get_best_sample( ) [private]**

called by minimize_surrogates for setUpType == "user_functions"

determine best solution from among sample data for expected improvememt function

- References **Model::approximation_data()**, **SurfBasedMinimizer::augmented_lagrangian_merit()**, **Model::continuous_variables()**, **Dakota::copy_data()**, **Model::current_response()**, **Model::evaluate()**, **EffGlobalMinimizer::fHatModel**, **Response::function_values()**, **Iterator::iteratedModel**, **EffGlobalMinimizer::meritFnStar**, **Minimizer::numFunctions**, **SurfBasedMinimizer::origNonlinEqTargets**, **SurfBasedMinimizer::origNonlinIneqLowerBnds**, **SurfBasedMinimizer::origNonlinIneqUpperBnds**, **Model::primary_response_fn_sense()**, **Model::primary_response_fn_weights()**, **EffGlobalMinimizer::truthFnStar**, and **EffGlobalMinimizer::varStar**.

- Referenced by **EffGlobalMinimizer::minimize_surrogates_on_model()**.

**void EIF_objective_eval( const Variables & sub_model_vars, const Variables & recast_vars, const Response & sub_model_response, Response & recast_response ) [static], [private]**

static function used as the objective function in the Expected Improvement (EIF) problem formulation for PMA

To maximize expected improvement, the approxSubProbMinimizer will minimize -(expected improvement).

- References **Response::active_set_request_vector()**, **Model::approximation_variances()**, **EffGlobalMinimizer::effGlobalInstance**, **EffGlobalMinimizer::expected_improvement()**, **EffGlobalMinimizer::fHatModel**, **Response::function_value()**, and **Response::function_values()**.

- Referenced by **EffGlobalMinimizer::minimize_surrogates_on_model()**.

The documentation for this class was generated from the following files:

- **EffGlobalMinimizer.hpp**
- **EffGlobalMinimizer.cpp**

14.67 **EffGlobalTraits Class Reference**

Implementation of Efficient Global Optimization/Least Squares algorithms.

Inheritance diagram for EffGlobalTraits:

```
        TraitsBase
           |
           v
EffGlobalTraits
```
Public Member Functions

- **EffGlobalTraits ()**
  
  *default constructor*

- **virtual ~EffGlobalTraits ()**
  
  *destructor*

- **virtual bool is_derived ()**
  
  *A temporary query used in the refactor.*

- **bool supports_continuous_variables ()**
  
  *Return the flag indicating whether method supports continuous variables.*

- **bool supports_nonlinear_equality ()**
  
  *Return the flag indicating whether method supports nonlinear equalities.*

- **bool supports_nonlinear_inequality ()**
  
  *Return the flag indicating whether method supports nonlinear inequalities.*

### 14.67.1 Detailed Description

Implementation of Efficient Global Optimization/Least Squares algorithms.

A version of TraitsBase specialized for efficient global minimizer

The documentation for this class was generated from the following file:

- EffGlobalMinimizer.hpp

### 14.68 EmbedHybridMetaIterator Class Reference

Meta-iterator for closely-coupled hybrid iteration, typically involving the embedding of local search methods within global search methods.

Inheritance diagram for EmbedHybridMetaIterator:

```
   Iterator
     |     |
     |     |
     v     v
MetaIterator
     |     |
     |     |
     v     v
EmbedHybridMetaIterator
```

Public Member Functions

- **EmbedHybridMetaIterator (ProblemDescDB &problem_db)**
  
  *standard constructor*

- **EmbedHybridMetaIterator (ProblemDescDB &problem_db, Model &model)**
  
  *alternate constructor*

- **~EmbedHybridMetaIterator ()**
  
  *destructor*
14.68. EMBEDHYBRIDMETAITERATOR CLASS REFERENCE

Protected Member Functions

- void core_run ()
  Performs the hybrid iteration by executing global and local iterators, using a set of models that may vary in fidelity.
- void derived_init_communicators (ParLevLIter pl_iter)
  derived class contributions to initializing the communicators associated with this Iterator instance
- void derived_set_communicators (ParLevLIter pl_iter)
  derived class contributions to setting the communicators associated with this Iterator instance
- void derived_free_communicators (ParLevLIter pl_iter)
  derived class contributions to freeing the communicators associated with this Iterator instance
- IntIntPair estimate_partition_bounds ()
  estimate the minimum and maximum partition sizes that can be utilized by this Iterator
- const Variables & variables_results () const
  return the final solution from the embedded hybrid (variables)
- const Response & response_results () const
  return the final solution from the embedded hybrid (response)

Private Attributes

- Iterator globalIterator
  the top-level outer iterator (e.g., global minimizer)
- Model globalModel
  the model employed by the top-level outer iterator
- Iterator localIterator
  the inner iterator (e.g., local minimizer)
- Model localModel
  the model employed by the inner iterator
- bool singlePassedModel
  use of constructor that enforces use of a single passed Model
- Real localSearchProb
  the probability of running a local search refinement within phases of the global minimization for tightly-coupled hybrids

Additional Inherited Members

14.68.1 Detailed Description

Meta-iterator for closely-coupled hybrid iteration, typically involving the embedding of local search methods within global search methods.

This meta-iterator uses multiple methods in close coordination, generally using a local search minimizer repeatedly within a global minimizer (the local search minimizer refines candidate minima which are fed back to the global minimizer).

The documentation for this class was generated from the following files:

- EmbedHybridMetaIterator.hpp
- EmbedHybridMetaIterator.cpp
14.69 Environment Class Reference

Base class for the environment class hierarchy.

Inheritance diagram for Environment:

```
Environment
    └── ExecutableEnvironment
    └── LibraryEnvironment
```

Public Member Functions

- **Environment ()**
  
  default constructor: empty envelope

- **Environment (int argc, char ∗argv[])**
  
  envelope constructor for ExecutableEnvironment letter

- **Environment (ProgramOptions prog_opts)**

- **Environment (MPI_Comm dakota_mpi_comm, ProgramOptions prog_opts=ProgramOptions())**

- **Environment (const String &env_type)**
  
  envelope constructor for letter type identified by String

- **Environment (const Environment &env)**
  
  copy constructor

- **virtual ~Environment ()**
  
  destructor

- **Environment operator= (const Environment &env)**
  
  assignment operator

- **virtual void execute ()**
  
  the run function for the environment: invoke the iterator(s) on the model(s). Called from main.cpp.

- **bool check () const**
  
  Print status of check and return true if in a "check" mode, including version and help. Return false if proceeding to a run mode.

- **MPIManager & mpi_manager ()**
  
  return mpiManager

- **ProgramOptions & program_options ()**
  
  return programOptions

- **OutputManager & output_manager ()**
  
  return outputManager

- **ParallelLibrary & parallel_library ()**
  
  return parallelLib

- **ProblemDescDB & problem_description_db ()**
  
  return probDescDB

- **const Variables & variables_results () const**
  
  return the final environment solution (variables)
const Response & response_results () const
return the final environment solution (response)

void exit_mode (const String &mode="exit")
allow environment clients to set Dakota exit behavior (throw vs. exit)

Protected Member Functions

Environment (BaseConstructor)
constructor initializes the base class part of default-constructed letters

Environment (BaseConstructor, int argc, char *argv[])
constructor initializes the base class part of executable letter classes

Environment (BaseConstructor, ProgramOptions prog_opts, MPI_Comm dakota_mpi_comm=MPI_COMM_M_WORLD)
constructor initializes the base class part of library letter classes

void parse (bool check_bcast_database=true, DbCallbackFunctionPtr callback=NULL, void *callback_data=NULL)
parse inputs, callbacks, and optionally check and broadcast

void construct ()
Instantiate topLevelIterator.

void destruct ()
Deallocate parallel partitioning for topLevelIterator.

bool assign_model_pointer () const

Protected Attributes

MPIManager mpiManager
the MPI manager instance

ProgramOptions programOptions
the command line options manager

OutputManager outputManager
>tagged output stream manager

ParallelLibrary parallelLib
the parallel library instance

ProblemDescDB probDescDB
the parser database instance

Iterator topLevelIterator
the top level (meta-)iterator

UsageTracker usageTracker
tool for Dakota usage tracking (this is a thin wrapper class)

Private Member Functions

Environment * get_environment (const String &env_type)
Used by the envelope to instantiate the correct letter class.
CHAPTER 14. CLASS DOCUMENTATION

Private Attributes

- `Environment * environmentRep`
  pointer to the letter (initialized only for the envelope)
- `int referenceCount`
  number of objects sharing environmentRep

14.69.1 Detailed Description

Base class for the environment class hierarchy.

The `Environment` class is the base class for the class hierarchy providing the top level control in DAKOTA. The environment is responsible for creating and managing iterators and models. For memory efficiency and enhanced polymorphism, the environment hierarchy employs the "letter/envelope idiom" (see Coplien "Advanced C++", p. 133), for which the base class (`Environment`) serves as the envelope and one of the derived classes (selected in `Environment::get_environment()`) serves as the letter.

14.69.2 Constructor & Destructor Documentation

`Environment ()`

default constructor: empty envelope

Default envelope constructor. `environmentRep` is NULL in this case, which makes it necessary to check for NULL in the copy constructor, assignment operator, and destructor.

`Environment ( int argc, char * argv[] )`
envelope constructor for `ExecutableEnvironment` letter

Envelope constructor for `ExecutableEnvironment`. Selection of derived type by `get_environment()` is not necessary in this case.

References `Dakota::abort_handler()`, and `Environment::environmentRep`.

`Environment ( ProgramOptions prog_opts )`

Envelope constructor for `LibraryEnvironment`. Selection of derived type by `get_environment()` is not necessary in this case.

References `Dakota::abort_handler()`, and `Environment::environmentRep`.

`Environment ( MPI_Comm dakota_mpi_comm, ProgramOptions prog_opts = ProgramOptions() )`

Envelope constructor for `LibraryEnvironment`. Selection of derived type by `get_environment()` is not necessary in this case.

References `Dakota::abort_handler()`, and `Environment::environmentRep`.

`Environment ( const String & env_type )`

envelope constructor for letter type identified by String

Alternate construction by String. Envelope constructor invokes `get_environment()` which instantiates a derived class letter; the derived constructor selects a `BaseConstructor` constructor in its initialization list to avoid the recursion of a base class constructor calling `get_environment()` again.

References `Dakota::abort_handler()`, `Environment::environmentRep`, and `Environment::get_environment()`.
Environment (const Environment & env)

copy constructor
  Copy constructor manages sharing of environmentRep and incrementing of referenceCount.
  References Environment::environmentRep, and Environment::referenceCount.

~Environment() [virtual]
destructor
  Destructor decrements referenceCount and only deletes environmentRep when referenceCount reaches zero.
  References Environment::destruct(), Environment::environmentRep, and Environment::referenceCount.

Environment(BaseConstructor) [protected]
constructor initializes the base class part of default-constructed letters
  This letter constructor initializes base class data for inherited environments that are default constructed. Since
  the letter IS the representation, its representation pointer is set to NULL (an uninitialized pointer causes problems
  in ~Environment).
  Use cases: library with no options, no MPI comm
  References ProgramOptions::exit_mode(), Environment::exit_mode(), WorkdirHelper::initialize(), and Environment-
  ::programOptions.

Environment(BaseConstructor, int argc, char *argv[]) [protected]
constructor initializes the base class part of executable letter classes
  This letter constructor initializes base class data for inherited environments: instantiate/initialize the environ-
  ment, options, parallel library, and problem description database objects. Since the letter IS the representation, its
  representation pointer is set to NULL (an uninitialized pointer causes problems in ~Environment).
  Use cases: executable with command-line args
  References ProgramOptions::exit_mode(), Environment::exit_mode(), WorkdirHelper::initialize(), and Environment-
  ::programOptions.

Environment(BaseConstructor, ProgramOptions prog_opts, MPI_Comm dakota_mpi_comm = MPI_COMM_WORLD) [protected]
constructor initializes the base class part of library letter classes
  This letter constructor initializes base class data for inherited environments. Since the letter IS the representa-
  tion, its representation pointer is set to NULL (an uninitialized pointer causes problems in ~Environment).
  Use cases: library with program options library with program options and MPI comm
  References ProgramOptions::exit_mode(), Environment::exit_mode(), WorkdirHelper::initialize(), and Environment-
  ::programOptions.

14.69.3 Member Function Documentation

Environment operator=(const Environment & env)
assignment operator
  Assignment operator decrements referenceCount for old environmentRep, assigns new environmentRep, and
  References Environment::environmentRep, and Environment::referenceCount.
void exit_mode ( const String & mode = "exit" )
allow environment clients to set Dakota exit behavior (throw vs. exit)
Set the global variable controlling Dakota's exit behavior. Call with no arguments to reset to default behavior.
References Dakota::abort_handler(), and Dakota::abort_mode.
Referenced by Environment::Environment(), and run_dakota_data().

void parse ( bool check_bcast_database = true, DbCallbackFunctionPtr callback = NULL, void * callback_data = NULL ) [protected]
parse inputs, callbacks, and optionally check and broadcast
Parse input file and invoked any callbacks, then optionally check and sync database if check_bcast_database =
true
References ProblemDescDB::check_and_broadcast(), ProgramOptions::input_file(), ProgramOptions::input_-string(), ProblemDescDB::parse_inputs(), Environment::probDescDB, and Environment::programOptions.
Referenced by ExecutableEnvironment::ExecutableEnvironment(), and LibraryEnvironment::LibraryEnvironment().

Environment * get_environment ( const String & env_type ) [private]
Used by the envelope to instantiate the correct letter class.
Used only by the envelope constructor to initialize environmentRep to the appropriate derived type, as given
by the environmentName attribute.
Referenced by Environment::Environment().
The documentation for this class was generated from the following files:
• DakotaEnvironment.hpp
• DakotaEnvironment.cpp

14.70 JEGAOptimizer::Evaluator Class Reference
An evaluator specialization that knows how to interact with Dakota.
Inherits GeneticAlgorithmEvaluator.

Public Member Functions
• virtual bool Evaluate (DesignGroup &group)
  Does evaluation of each design in group.
• virtual bool Evaluate (Design &des)
  This method cannot be used!!
• virtual std::string GetName () const
  Returns the proper name of this operator.
• virtual std::string GetDescription () const
  Returns a full description of what this operator does and how.
• virtual GeneticAlgorithmOperator * Clone (GeneticAlgorithm &algorithm) const
  Creates and returns a pointer to an exact duplicate of this operator.
• Evaluator (GeneticAlgorithm &algorithm, Model &model)
  Constructs a Evaluator for use by algorithm.
• Evaluator (const Evaluator &copy)
Copy constructs a Evaluator.

- Evaluator (const Evaluator &copy, GeneticAlgorithm &algorithm, Model &model)
  
  Copy constructs a Evaluator for use by algorithm.

**Static Public Member Functions**

- static const std::string & Name ()
  
  Returns the proper name of this operator.

- static const std::string & Description ()
  
  Returns a full description of what this operator does and how.

**Protected Member Functions**

- void SeparateVariables (const Design &from, RealVector &intoCont, IntVector &intoDiscInt, RealVector &intoDiscReal, StringMultiArray &intoDiscString) const
  
  This method fills intoCont, intoDiscInt and intoDiscReal appropriately using the values of from.

- void RecordResponses (const RealVector &from, Design &into) const
  
  Records the computed objective and constraint function values into into.

- std::size_t GetNumberNonLinearConstraints () const
  
  Returns the number of non-linear constraints for the problem.

- std::size_t GetNumberLinearConstraints () const
  
  Returns the number of linear constraints for the problem.

**Private Member Functions**

- Evaluator (GeneticAlgorithm &algorithm)
  
  This constructor has no implementation and cannot be used.

**Private Attributes**

- Model & _model
  
  The Model known by this evaluator.

### 14.70.1 Detailed Description

An evaluator specialization that knows how to interact with Dakota.

This evaluator knows how to use the model to do evaluations both in synchronous and asynchronous modes.

### 14.70.2 Constructor & Destructor Documentation

Evaluator ( GeneticAlgorithm & algorithm, Model & model ) [inline]

Constructs a Evaluator for use by algorithm.

The optimizer is needed for purposes of variable scaling.
CHAPTER 14. CLASS DOCUMENTATION

Parameters

<table>
<thead>
<tr>
<th>parameter</th>
<th>description</th>
</tr>
</thead>
<tbody>
<tr>
<td>algorithm</td>
<td>The GA for which the new evaluator is to be used.</td>
</tr>
<tr>
<td>model</td>
<td>The model through which evaluations will be done.</td>
</tr>
</tbody>
</table>

Evaluator ( const Evaluator & copy ) [inline]

Copy constructs a Evaluator.

Parameters

<table>
<thead>
<tr>
<th>parameter</th>
<th>description</th>
</tr>
</thead>
<tbody>
<tr>
<td>copy</td>
<td>The evaluator from which properties are to be duplicated into this.</td>
</tr>
</tbody>
</table>

Evaluator ( const Evaluator & copy, GeneticAlgorithm & algorithm, Model & model ) [inline]

Copy constructs a Evaluator for use by algorithm.

The optimizer is needed for purposes of variable scaling.

Parameters

<table>
<thead>
<tr>
<th>parameter</th>
<th>description</th>
</tr>
</thead>
<tbody>
<tr>
<td>copy</td>
<td>The existing Evaluator from which to retrieve properties.</td>
</tr>
<tr>
<td>algorithm</td>
<td>The GA for which the new evaluator is to be used.</td>
</tr>
<tr>
<td>model</td>
<td>The model through which evaluations will be done.</td>
</tr>
</tbody>
</table>

Evaluator ( GeneticAlgorithm & algorithm ) [private]

This constructor has no implementation and cannot be used.

This constructor can never be used. It is provided so that this operator can still be registered in an operator registry even though it can never be instantiated from there.

Parameters

<table>
<thead>
<tr>
<th>parameter</th>
<th>description</th>
</tr>
</thead>
<tbody>
<tr>
<td>algorithm</td>
<td>The GA for which the new evaluator is to be used.</td>
</tr>
</tbody>
</table>

14.70.3 Member Function Documentation

static const std::string& Name() [inline], [static]

Returns the proper name of this operator.

Returns

The string "DAKOTA JEGA Evaluator".

static const std::string& Description() [inline], [static]

Returns a full description of what this operator does and how.

The returned text is:

This evaluator uses Sandia’s DAKOTA optimization software to evaluate the passed in Designs. This makes it possible to take advantage of the fact that DAKOTA is designed to run on massively parallel machines.
Returns

A description of the operation of this operator.

```cpp
void SeparateVariables ( const Design & from, RealVector & intoCont, IntVector & intoDiscInt, RealVector & intoDiscReal, StringMultiArray & intoDiscString ) const [protected]
```

This method fills `intoCont`, `intoDiscInt` and `intoDiscReal` appropriately using the values of `from`.

The discrete integer design variable values are placed in `intoDiscInt`, the discrete real design variable values are placed in `intoDiscReal`, and the continuum are placed into `intoCont`. The values are written into the vectors from the beginning so any previous contents of the vectors will be overwritten.

Parameters

<table>
<thead>
<tr>
<th>from</th>
<th>The Design class object from which to extract the discrete design variable values.</th>
</tr>
</thead>
<tbody>
<tr>
<td>intoDiscInt</td>
<td>The vector into which to place the extracted discrete integer values.</td>
</tr>
<tr>
<td>intoDiscReal</td>
<td>The vector into which to place the extracted discrete real values.</td>
</tr>
<tr>
<td>intoCont</td>
<td>The vector into which to place the extracted continuous values.</td>
</tr>
</tbody>
</table>

References JEGAOptimizer::Evaluator::model, Model::cv(), Model::discrete_int_sets(), Model::discrete_set_string_values(), Model::div(), Model::drv(), Model::dsv(), and Dakota::set_index_to_value().

```cpp
void RecordResponses ( const RealVector & from, Design & into ) const [protected]
```

Records the computed objective and constraint function values into `into`.

This method takes the response values stored in `from` and properly transfers them into the `into` design.

The response vector `from` is expected to contain values for each objective function followed by values for each non-linear constraint in the order in which the info objects were loaded into the target by the optimizer class.

Parameters

<table>
<thead>
<tr>
<th>from</th>
<th>The vector of responses to install into <code>into</code>.</th>
</tr>
</thead>
<tbody>
<tr>
<td>into</td>
<td>The Design to which the responses belong and into which they must be written.</td>
</tr>
</tbody>
</table>

```cpp
std::size_t GetNumberNonLinearConstraints ( ) const [inline], [protected]
```

Returns the number of non-linear constraints for the problem.

This is computed by adding the number of non-linear equality constraints to the number of non-linear inequality constraints. These values are obtained from the model.

Returns

The total number of non-linear constraints.

```cpp
std::size_t GetNumberLinearConstraints ( ) const [inline], [protected]
```

Returns the number of linear constraints for the problem.

This is computed by adding the number of linear equality constraints to the number of linear inequality constraints. These values are obtained from the model.

Returns

The total number of linear constraints.
bool Evaluate ( DesignGroup & group ) [virtual]

Does evaluation of each design in group.

This method uses the Model known by this class to get Designs evaluated. It properly formats the Design class information in a way that Dakota will understand and then interprets the Dakota results and puts them back into the Design class object. It respects the asynchronous flag in the Model so evaluations may occur synchronously or asynchronously.

Prior to evaluating a Design, this class checks to see if it is marked as already evaluated. If it is, then the evaluation of that Design is not carried out. This is not strictly necessary because Dakota keeps track of evaluated designs and does not re-evaluate. An exception is the case of a population read in from a file complete with responses where Dakota is unaware of the evaluations.

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>group</td>
<td>The group of Design class objects to be evaluated.</td>
</tr>
</tbody>
</table>

Returns

ture if all evaluations completed and false otherwise.

virtual bool Evaluate ( Design & des ) [inline], [virtual]

This method cannot be used!!

This method does nothing and cannot be called. This is because in the case of asynchronous evaluation, this method would be unable to conform. It would require that each evaluation be done in a synchronous fashion.

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>des</td>
<td>A Design that would be evaluated if this method worked.</td>
</tr>
</tbody>
</table>

Returns

Would return true if the Design were evaluated and false otherwise. Never actually returns here. Issues a fatal error. Otherwise, it would always return false.

virtual std::string GetName ( ) const [inline], [virtual]

Returns the proper name of this operator.

Returns

See Name().

virtual std::string GetDescription ( ) const [inline], [virtual]

Returns a full description of what this operator does and how.

Returns

See Description().

virtual GeneticAlgorithmOperator * Clone ( GeneticAlgorithm & algorithm ) const [inline], [virtual]

Creates and returns a pointer to an exact duplicate of this operator.
Parameters

| algorithm | The GA for which the clone is being created. |

Returns

A clone of this operator.

14.70.4 Member Data Documentation

Model& _model  [private]
The Model known by this evaluator.
It is through this model that evaluations will take place.
Referenced by JEGAOptimizer::Evaluator::SeparateVariables().
The documentation for this class was generated from the following file:
- JEGAOptimizer.cpp

14.71 NomadOptimizer::Evaluator Class Reference

NOMAD-based Evaluator class.
Inherits Evaluator.

Public Member Functions

- Evaluator (const NOMAD::Parameters &p, Model &model)
  Constructor.
- ~Evaluator (void)
  Destructor.
- bool eval_x (NOMAD::Eval_Point &x, const NOMAD::Double &h_max, bool &count_eval) const
  Main Evaluation Method.
- bool eval_x (std::list<NOMAD::Eval_Point*> &x, const NOMAD::Double &h_max, std::list<bool> &count_eval) const
  multi-point variant of evaluator
- void set_constraint_map (int numNomadNonlinearIneqConstraints, int numNomadNonlinearEqConstraints,
  std::vector<int> constraintMapIndices, std::vector<double> constraintMapMultipliers, std::vector<double> constraintMapOffsets)
  publishes constraint transformation
- void set_surrogate_usage (std::string useSurrogate)
  publishes surrogate usage

Private Member Functions

- void set_variables (const NOMAD::Eval_Point &x) const
  map NOMAD evaluation point to Dakota model
- void eval_model (bool allow_asynch, const NOMAD::Eval_Point &x) const
  evaluate the Dakota model (block or not, but don’t collect response)
- void get_responses (const RealVector &ftn_vals, NOMAD::Eval_Point &x) const
  map Dakota model responses to NOMAD evaluation point
Private Attributes

- `Model & _model`
- `int n_cont`
- `int n_disc_int`
- `int n_disc_real`
- `int numNomadNonlinearIneqConstr`
  
  Number of nonlinear constraints after put into Nomad format.
- `int numNomadNonlinearEqConstr`
- `std::vector<int> constrMapIndices`
  
  map from Dakota constraint number to Nomad constraint number
- `std::vector<double> constrMapMultipliers`
  
  multipliers for constraint transformations
- `std::vector<double> constrMapOffsets`
  
  offsets for constraint transformations
- `std::string useSgte`
  
  defines use of surrogate in NOMAD

14.71.1 Detailed Description

NOMAD-based Evaluator class.

The NOMAD process requires an evaluation step, which calls the Simulation program. In the simplest version of this call, NOMAD executes the black box executable, which proceeds to write a file in a NOMAD-compatible format, which NOMAD reads to continue the process.

Because DAKOTA files are different form NOMAD files, and the simulations processed by DAKOTA already produce DAKOTA-compatible files, we cannot use this method for NOMAD. Instead, we implement the Nomad-Evaluator class, which takes the NOMAD inputs and passes them to DAKOTA’s Interface for processing. The evaluator then passes the evaluation Responses into the NOMAD objects for further analysis.

14.71.2 Constructor & Destructor Documentation

Evaluator ( const NOMAD::Parameters & p, Model & model )

Constructor.

<table>
<thead>
<tr>
<th>p</th>
<th>NOMAD Parameters object</th>
</tr>
</thead>
<tbody>
<tr>
<td>model</td>
<td>DAKOTA Model object</td>
</tr>
</tbody>
</table>

14.71.3 Member Function Documentation

bool eval_x ( NOMAD::Eval_Point & x, const NOMAD::Double & h_max, bool & count_eval ) const

Main Evaluation Method.

Method that handles the communication between the NOMAD search process and the Black Box Evaluation managed by DAKOTA’s Interface.
Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>$x$</td>
<td>Object that contains the points that need to be evaluated. Once the evaluation is completed, this object also stores the output back to be read by NOMAD.</td>
</tr>
<tr>
<td>$h_{\text{max}}$</td>
<td>Current value of the barrier parameter. Not used in this implementation.</td>
</tr>
<tr>
<td>$\text{count_eval}$</td>
<td>Flag that indicates whether this evaluation counts towards the max number of evaluations, often set to false when the evaluation does not meet certain costs during expensive evaluations. Not used in this implementation.</td>
</tr>
</tbody>
</table>

Returns

`true` if the evaluation was successful; `false` otherwise.

References Dakota::get_responses(), and Dakota::set_variables().
The documentation for this class was generated from the following files:

- NomadOptimizer.hpp
- NomadOptimizer.cpp

### 14.72 JEGA::JEGAOptimizer::EvaluatorCreator Class Reference

A specialization of the JEGA::FrontEnd::EvaluatorCreator that creates a new instance of a `Evaluator`. Inherits `EvaluatorCreator`.

#### Public Member Functions

- virtual GeneticAlgorithmEvaluator * CreateEvaluator (GeneticAlgorithm &alg)
  
  Overriden to return a newly created `Evaluator`.
- EvaluatorCreator (Model &theModel)
  
  Constructs an `EvaluatorCreator` using the supplied model.

#### Private Attributes

- Model & _theModel
  
  The user defined model to be passed to the constructor of the `Evaluator`.

### 14.72.1 Detailed Description

A specialization of the JEGA::FrontEnd::EvaluatorCreator that creates a new instance of a `Evaluator`.

### 14.72.2 Constructor & Destructor Documentation

EvaluatorCreator ( Model & theModel ) [inline]

Constructs an `EvaluatorCreator` using the supplied model.
Parameters

| `theModel` | The Dakota::Model this creator will pass to the created evaluator. |

### 14.72.3 Member Function Documentation

```cpp
virtual GeneticAlgorithmEvaluator* CreateEvaluator ( GeneticAlgorithm & alg ) [inline], [virtual]
```

Overridden to return a newly created `Evaluator`.

The GA will assume ownership of the evaluator so we needn’t worry about keeping track of it for destruction. The additional parameters needed by the `Evaluator` are stored as members of this class at construction time.

**Parameters**

| `alg` | The GA for which the evaluator is to be created. |

**Returns**

A pointer to a newly created `Evaluator`.

The documentation for this class was generated from the following file:

- JEGAOptimizer.cpp

### 14.73 ExecutableEnvironment Class Reference

`Environment` corresponding to execution as a stand-alone application.

Inheritance diagram for `ExecutableEnvironment`:

```
Environment
  |
  v
ExecutableEnvironment
```

**Public Member Functions**

- `ExecutableEnvironment ()`
  
  *default constructor*

- `ExecutableEnvironment (int argc, char *argv[])`
  
  *constructor*

- `~ExecutableEnvironment ()`
  
  *destructor*

- `void execute ()`
  
  *the run function for the environment: invoke the iterator(s) on the model(s). Called from main.cpp.*
Additional Inherited Members

14.73.1 Detailed Description

**Environment** corresponding to execution as a stand-alone application.

This environment corresponds to a stand-alone executable program, e.g., `main.cpp`. It sets up the **Parallel-Library**, **ProgramOptions**, and **ProblemDescDB** objects based on access to command line arguments.

The documentation for this class was generated from the following files:

- ExecutableEnvironment.hpp
- ExecutableEnvironment.cpp

14.74 ExperimentData Class Reference

Interpolation method for interpolating between experimental and model data. I need to work on inputs/outputs to this method. For now, this assumes interpolation of functional data.

Public Member Functions

- **ExperimentData()**
  default constructor
- **ExperimentData(const ProblemDescDB &prob_desc_db, const SharedResponseData &srd, short output_level)**
  typical DB-based constructor
- **ExperimentData(size_t num_experiments, size_t num_config_vars, const boost::filesystem::path &data_prefix, const SharedResponseData &srd, const StringArray &variance_types, short output_level, std::string scalarDataFilename="")**
  temporary? constructor for testing
- **ExperimentData(size_t num_experiments, const SharedResponseData &srd, const RealMatrix &configVars, const IntResponseMap &all_responses, short output_level)**
- **void load_data(const std::string &context_message)**
  Load experiments from data files (simple scalar or field)
- **void add_data(const RealVector &one_configvars, const Response &one_response)**
  Add one data point to the experimental data set.
- **size_t num_experiments() const**
  retrieve the number of experiments
- **size_t num_total_exp_points() const**
  retrieve the total number of experimental data points over all experiments
- **size_t num_scalar_primary() const**
  retrieve the number of scalars (applies to all experiments)
- **size_t num_fields() const**
  retrieve the number of fields (applies to all experiments)
- **size_t num_config_vars() const**
  number of configuration variables
- **const std::vector<RealVector> &config_vars() const**
  values of the configuration variables, 1 RealVector per experiment
• const RealVector & all_data (size_t experiment)
  return contiguous vector of all data (scalar, followed by field) for the specified experiment
• const Response & response (size_t experiment)
  return response for the specified experiment
• void per_exp_length (IntVector &per_length) const
  return the individual sizes of the experimental data lengths (all function values, scalar and field)
• const IntVector & field_lengths (size_t experiment) const
  return the field lengths for specified experiment index
• Real scalar_data (size_t response, size_t experiment) const
  retrieve the data value for the given response, for the given experiment
• RealVector field_data_view (size_t response, size_t experiment) const
  retrieve a view of the field data for the given response, for the given experiment
• RealMatrix field_coords_view (size_t response, size_t experiment) const
  retrieve a view of the field data coordinates for the given response, for the given experiment
• bool variance_type_active (short variance_type) const
  whether the specified variance type (enum value) is present and active
• bool variance_active () const
  whether any variance type is active
• Real apply_covariance (const RealVector &residuals, size_t experiment) const
  apply the covariance responses to compute the triple product v' * inv(C) * v for the given experiment
• void apply_covariance_inv_sqrt (const RealVector &residuals, size_t experiment, RealVector &weighted_residuals) const
  apply inverse sqrt of the covariance to compute weighted residuals
• void apply_covariance_inv_sqrt (const RealMatrix &gradients, size_t experiment, RealMatrix &weighted_gradients) const
  apply inverse sqrt of the covariance to compute weighted gradients
• void apply_covariance_inv_sqrt (const RealSymMatrixArray &hessians, size_t experiment, RealSymMatrixArray &weighted_hessians) const
  apply inverse sqrt of the covariance to compute weighted Hessians
• void apply_simulation_error (const RealVector &simulation_error, size_t experiment)
  apply simulation error to experiment data
• void get_main_diagonal (RealVector &diagonal, size_t experiment) const
  return a (copy) vector containing the main diagonal entries of a specified experimental covariance matrix
• void cov_std_deviation (RealVectorArray &std_deviation) const
  get the standard deviation of the observation error process, one vector per experiment
• void cov_as_correlation (RealSymMatrixArray &corr_matrix) const
  get the observation error covariance as a correlation matrix, one vector per experiment
• void covariance (int exp_ind, RealSymMatrix &cov_mat) const
  retrieve an individual covariance entry as a dense matrix
• void form_residuals (const Response &simResp, Response &residual_resp) const
  form residuals for all experiments, interpolating if necessary; one simulation response maps to all experiments
• void form_residuals (const Response &simResp, const size_t curr_exp, Response &residual_resp) const
  Populate the portion of residual_resp corresponding to experiment curr_exp; the passed simulation response maps only to the specified experiment.
void form_residuals (const Response &sim_resp, size_t exp_num, const ShortArray &total_asv, size_t residual RESP_offset, Response &residual_resp) const

form residuals for an individual experiment, interpolating if necessary

void recover_model (size_t num_pri_fns, RealVector &model_fns) const

calculate the original model from the first experiment block in a full set of residuals; works in no interpolation case only

void interpolate_flag () const

flag for interpolation. If 0, no interpolation. If 1, interpolate.

void interpolate_simulation_data (const Response &sim_resp, size_t exp_num, const ShortArray &total_asv, size_t exp_offset, Response &interp_resp) const

Interpolate simulation data (values, gradients and Hessians) onto the coordinates of the experimental data.

void scale_residuals (const Response &residual_response, RealVector &scaled_residuals) const

Apply the experiment data covariance to the residual data (scale functions by Gamma_d^{-1/2}), returning in scaled_residuals.

void scale_residuals (Response &residual_response) const

Apply the experiment data covariance to the residual data in-place (scale functions, gradients, and Hessians by Gamma_d^{-1/2}).

void build_gradient_of_sum_square_residuals (const Response &resp, RealVector &ssr_gradient)

Build the gradient of the ssr from residuals and function gradients based on the response’s active set request vector.

void build_gradient_of_sum_square_residuals (const Response &resp, const ShortArray &asrv, RealVector &ssr_gradient)

Build the gradient of the ssr from residuals and function gradients using the passed active set request vector (overrides the response’s request vector).

void build_gradient_of_sum_square_residuals_from_response (const Response &resp, const ShortArray &asrv, int exp_ind, RealVector &ssr_gradient)

Update the gradient of ssr with the values from the gradient associated with a single experiment.

void build_gradient_of_sum_square_residuals_from_function_data (const RealMatrix &func_gradients, const RealVector &residuals, RealVector &ssr_gradient, const ShortArray &asrv)

Construct the gradient of the sum of squares of residuals.

void build_hessian_of_sum_square_residuals (const Response &resp, RealSymMatrix &ssr_hessian)

Build the hessian of the ssr from residuals, function gradients and function Hessians based on the response’s active set request vector.

void build_hessian_of_sum_square_residuals (const Response &resp, const ShortArray &asrv, RealSymMatrix &ssr_hessian)

Build the hessian of the ssr from residuals, function gradients and function Hessians using the passed active set request vector (overrides the response’s request vector).

void build_hessian_of_sum_square_residuals_from_response (const Response &resp, const ShortArray &asrv, int exp_ind, RealSymMatrix &ssr_hessian)

Update the hessian of ssr with the values from the hessian associated with a single experiment.

void build_hessian_of_sum_square_residuals_from_function_data (const RealSymMatrixArray &func_hessians, const RealMatrix &func_gradients, const RealVector &residuals, RealSymMatrix &ssr_hessian, const ShortArray &asrv)

Construct the hessian of the sum of squares of residuals.

void scale_residuals (const RealVector &multipliers, unsigned short multiplier_mode, size_t num_calib_params, Response &residual_response) const
in-place scale the residual response (functions, gradients, Hessians) by sqrt(multipliers), according to blocks indicated by multiplier mode

- Real **cov_determinant** (const RealVector &multipliers, unsigned short multiplier_mode) const
  returns the determinant of (covariance block-scaled by the passed multipliers)

- Real **half_log_cov_determinant** (const RealVector &multipliers, unsigned short multiplier_mode) const
  returns the log of the determinant of (covariance block-scaled by the passed multipliers)

- void **half_log_cov_det_gradient** (const RealVector &multipliers, unsigned short multiplier_mode, size_t hyper_offset, RealVector &gradient) const
  populated the passed gradient with derivatives w.r.t. the hyper-parameter multipliers, starting at hyper_offset (must be sized)

- void **half_log_cov_det_hessian** (const RealVector &multipliers, unsigned short multiplier_mode, size_t hyper_offset, RealSymMatrix &hessian) const
  populated the passed Hessian with derivatives w.r.t. the hyper-parameter multipliers, starting at hyper_offset (must be sized)

- StringArray **hyperparam_labels** (unsigned short multiplier_mode) const
  generate variable labels for the covariance (error) multiplier hyperparams

Protected Member Functions

- ShortArray **determine_active_request** (const Response &resid RESP) const
  Perform check on the active request vector to make sure it is amenable to interpolation of simulation data and application of apply covariance.

- SizetArray **residuals_per_multiplier** (unsigned short multiplier_mode) const
  count the number of residuals influenced by each multiplier

- void **generate_multipliers** (const RealVector &multipliers, unsigned short multiplier_mode, RealVector &expanded_multipliers) const
  Generate a set of multipliers commensurate with the residual size for the total experiment data set. Instead of repeating the loops all over the place, generate an expanded set of multipliers; the conditionals get too complicated otherwise.

- void **resid2mult_map** (unsigned short multiplier_mode, IntVector &resid2mult_indices) const
  return the index of the multiplier that affects each residual

Private Member Functions

- void **initialize** (const StringArray &variance_types, const SharedResponseData &srd)
  shared body of constructor initialization

- void **parse_sigma_types** (const StringArray &sigma_types)
  parse user-provided sigma type strings and populate enums

- void **load_experiment** (size_t exp_index, std::ifstream &scalar_data_stream, size_t num_field_sigma_matrices, size_t num_field_sigma_diagonals, size_t num_field_sigma_scalars, size_t num_field_sigma_none, Response &exp RESP)
  Load a single experiment exp_index into exp_RESP.

- void **read_scalar_sigma** (std::ifstream &scalar_data_stream, RealVector &sigma_scalars, IntVector &scalar_map_indices)
  read or default populate the scalar sigma

- RealVector **residuals_view** (const RealVector &residuals, size_t experiment) const
Return a view (to allowing updating in place) of the residuals associated with a given experiment, from a vector containing residuals from all experiments.

- **RealMatrix gradients_view** (const RealMatrix &gradients, size_t experiment) const
  
  Return a view (to allowing updating in place) of the gradients associated with a given experiment, from a matrix containing gradients from all experiments.

- **RealSymMatrixArray hessians_view** (const RealSymMatrixArray &hessians, size_t experiment) const
  
  Return a view (to allowing updating in place) of the hessians associated with a given experiment, from an array containing the hessians from all experiments.

### Private Attributes

- **bool calibrationDataFlag**
  
  whether the user specified a calibration data block

- **size_t numExperiments**
  
  the total number of experiments

- **size_t numConfigVars**
  
  number of configuration (state) variables to read for each experiment

- **UShortArray varianceTypes**
  
  type of variance specified for each variable, one per response group; empty varianceType indicates none specified by user

- **Real covarianceDeterminant**
  
  cached product of each experiment covariance’s determinant

- **Real logCovarianceDeterminant**
  
  cached sum of each experiment covariance’s log determinant

- **boost::filesystem::path dataPathPrefix**
  
  path to prepend to any data file names

- **String scalarDataFilename**
  
  the user-specified scalar data filename

- **unsigned short scalarDataFormat**
  
  tabular format of the simple scalar data file; supports TABULAR_NONE, TABULAR_HEADER, TABULAR_EVAL_ID, TABULAR_EXPERANNOT

- **size_t scalarSigmaPerRow**
  
  number of sigma values to read from each row in simple data file format (calculated from variance types strings

- **bool readSimFieldCoords**
  
  whether to read coordinate data files for simulation fields

- **SharedResponseData simulationSRD**
  
  archived shared data for use in sizing fields, total functions (historically we read all functions, including constraints, which might not be correct)

- **bool interpolateFlag**
  
  flag for interpolation.

- **short outputLevel**
  
  output verbosity level

- **std::vector< Response > allExperiments**
  
  Vector of numExperiments ExperimentResponses, holding the observed data and error (sigma/covariance) for each experiment.
• std::vector<RealVector> allConfigVars

  Vector of numExperiments configurations at which data were gathered; empty if no configurations specified.

• IntVector experimentLengths

  Length of each experiment.

• IntVector expOffsets

  function index offsets for individual experiment data sets

14.74.1 Detailed Description

Interpolation method for interpolating between experimental and model data. I need to work on inputs/outputs to this method. For now, this assumes interpolation of functional data.

As Brian suggested, this class has the experimental data (coordinates and RealVectorArray interpolated-Results; The ExperimentData class is used to read and populate data (currently from user-specified files and/or the input spec) relating to experimental (physical observations) data for the purposes of calibration. Such data may include (for example): number of experiments, configuration variables, type of data (scalar vs. functional), treatment of sigma (experimental uncertainties). This class also provides an interpolation capability to interpolate between simulation or experimental data so that the differencing between simulation and experimental data may be performed properly.

14.74.2 Member Function Documentation

void form_residuals ( const Response & sim_resp, Response & residual_resp ) const

form residuals for all experiments, interpolating if necessary; one simulation response maps to all experiments

  This assumes the source gradient/Hessian are size less or equal to the destination response, and that the leading part is to be populated.

  References ExperimentData::determine_active_request(), ExperimentData::numExperiments, and ExperimentData::per_exp_length().

  Referenced by DataTransformModel::archive_submodel_responses(), DataTransformModel::derived_evaluate(), ExperimentData::form_residuals(), DataTransformModel::primary_resp_differencer(), and DataTransformModel::transform_response_map().

void form_residuals ( const Response & sim_resp, size_t exp_ind, const ShortArray & total_asv, size_t exp_offset, Response & residual_resp ) const

form residuals for an individual experiment, interpolating if necessary

  This assumes the source gradient/Hessian are size less or equal to the destination response, and that the leading part is to be populated.

  References ExperimentData::allExperiments, ExperimentData::field_data_view(), Response::field_lengths(), Response::function_gradients_view(), Response::function_gradients(), Response::function_hessians_view(), Response::function_hessians(), Response::function_values_view(), Response::function_values(), ExperimentData::gradients_view(), ExperimentData::hessians_view(), ExperimentData::interpolate_simulation_data(), ExperimentData::interpolateFlag, ExperimentData::num_fields(), ExperimentData::num_scalar_primary(), and ExperimentData::outputLevel.

void recover_model ( size_t num_pri_fns, RealVector & best_fns ) const

recover original model from the first experiment block in a full set of residuals; works in no interpolation case only (sizes same)
Add the data back to the residual to recover the model, for use in surrogated-based LSQ where DB lookup will fail (need approx eval DB). best_fns contains primary and secondary responses.

References Dakota::abort_handler(), ExperimentData::allExperiments, Response::function_value(), ExperimentData::interpolateFlag, and Response::num_functions().

```cpp
void build_gradient_of_sum_square_residuals_from_function_data ( const RealMatrix & func_gradients, const RealVector & residuals, RealVector & ssr_gradient, const ShortArray & asrv )
```

Construct the gradient of the sum of squares of residuals.

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>func_gradients</code></td>
<td>A matrix containing the gradients of the residual vector</td>
</tr>
<tr>
<td><code>residuals</code></td>
<td>A vector of residuals (mismatch between experimental data and the corresponding function values)</td>
</tr>
<tr>
<td><code>asrv</code></td>
<td>The active set request vector</td>
</tr>
</tbody>
</table>

Referenced by ExperimentData::build_gradient_of_sum_square_residuals_from_response().

```cpp
void build_hessian_of_sum_square_residuals_from_function_data ( const RealSymMatrixArray & func_hessians, const RealMatrix & func_gradients, const RealVector & residuals, RealSymMatrix & ssr_hessian, const ShortArray & asrv )
```

Construct the hessian of the sum of squares of residuals.

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>func_hessians</code></td>
<td>A list of matrices containing the Hessians of the function elements in the residual vector</td>
</tr>
<tr>
<td><code>func_gradients</code></td>
<td>A matrix containing the gradients of the residual vector</td>
</tr>
<tr>
<td><code>residuals</code></td>
<td>A vector of residuals (mismatch between experimental data and the corresponding function values)</td>
</tr>
<tr>
<td><code>asrv</code></td>
<td>The active set request vector</td>
</tr>
</tbody>
</table>

Referenced by ExperimentData::build_hessian_of_sum_square_residuals_from_response().

```cpp
void scale_residuals ( const RealVector & multipliers, unsigned short multiplier_mode, size_t num_calib_params, Response & residual_response ) const
```

In-place scale the residual response (functions, gradients, Hessians) by sqrt(multipliers), according to blocks indicated by multiplier mode.

In-place scaling of residual response by hyper-parameter multipliers.

References Dakota::abort_handler(), Response::active_set_request_vector(), Response::function_gradient_view(), Response::function_hessian_view(), Response::function_value(), Response::function_value_view(), ExperimentData::num_total_exppoints(), and ExperimentData::resid2mult_map().

```cpp
Real cov_determinant ( const RealVector & multipliers, unsigned short multiplier_mode ) const
```

returns the determinant of (covariance block-scaled by the passed multipliers)

Determinant of the total covariance used in inference, which has blocks mult_i * I * Cov_i.

References Dakota::abort_handler(), ExperimentData::covarianceDeterminant, ExperimentData::generate_multipliers(), and ExperimentData::num_total_exppoints().
Real half_log_cov_determinant ( const RealVector & multipliers, unsigned short multiplier_mode ) const

returns the log of the determinant of (covariance block-scaled by the passed multipliers)

- Determinant of half the log of total covariance used in inference, which has blocks mult_i * I * Cov_i.
- References Dakota::abort_handler(), ExperimentData::generate_multipliers(), ExperimentData::logCovariance-Determinant, and ExperimentData::num_total_exppoints().
- Referenced by NonDBayesCalibration::log_likelihood(), NonDMUQBayesCalibration::print_results(), and NonDQUESOBayesCalibration::print_results().

void half_log_cov_det_gradient ( const RealVector & multipliers, unsigned short multiplier_mode, size_t hyper_offset, RealVector & gradient ) const

populated the passed gradient with derivatives w.r.t. the hyper-parameter multipliers, starting at hyper_offset (must be sized)

- Compute the gradient of scalar f(m) 0.5*log(det(mult*Cov)) w.r.t. mults. Since this is the only use case, we include the 0.5 factor and perform an update in-place.
- References ExperimentData::num_total_exppoints(), and ExperimentData::residuals_per_multiplier().
- Referenced by NonDBayesCalibration::neg_log_post_resp_mapping().

void half_log_cov_det_hessian ( const RealVector & multipliers, unsigned short multiplier_mode, size_t hyper_offset, RealSymMatrix & hessian ) const

populated the passed Hessian with derivatives w.r.t. the hyper-parameter multipliers, starting at hyper_offset (must be sized)

- Compute the gradient of scalar f(m) log(det(mult*Cov)) w.r.t. mults
- References ExperimentData::num_total_exppoints(), and ExperimentData::residuals_per_multiplier().
- Referenced by NonDBayesCalibration::calculate_evidence(), and NonDBayesCalibration::neg_log_post_resp_mapping().

SizetArray residuals_per_multiplier ( unsigned short multiplier_mode ) const [protected]

count the number of residuals influenced by each multiplier

- Calculate how many residuals each multiplier affects
- References ExperimentData::allExperiments, SharedResponseData::num_field_response_groups(), ExperimentData::num_fields(), SharedResponseData::num_response_groups(), SharedResponseData::num_scalar_primary(), ExperimentData::numExperiments, and ExperimentData::simulationSRD.
- Referenced by ExperimentData::half_log_cov_det_gradient(), and ExperimentData::half_log_cov_det_hessian().

void parse_sigma_types ( const StringArray & sigma_types ) [private]

parse user-provided sigma type strings and populate enums

- Validate user-provided sigma specification. User can specify 0, 1, or num_response_groups sigmas. If specified, sigma types must be the same for all scalar responses.
- References Dakota::abort_handler(), SharedResponseData::num_response_groups(), SharedResponseData::num_scalar_primary(), ExperimentData::scalarDataFilename, ExperimentData::scalarSigmaPerRow, ExperimentData::simulationSRD, and ExperimentData::varianceTypes.
- Referenced by ExperimentData::initialize().
void load_experiment ( size_t exp_index, std::ifstream & scalar_data_stream, size_t num_field_sigma_matrices, size_t num_field_sigma_diagonals, size_t num_field_sigma_scalars, size_t num_field_sigma_none, Response & exp_resp ) [private]

Load a single experiment exp_index into exp_resp.

Load an experiment from a mixture of legacy format data and field data format files

References Dakota::abort_handler(), ExperimentData::dataPathPrefix, Response::field_coords(), Response::field_group_labels(), ExperimentData::field_lengths(), Response::field_lengths(), Response::field_values(), Response::function_labels(), Response::function_value(), Dakota::is_matrix_symmetric(), SharedResponseData::num_field_response_groups(), ExperimentData::num_fields(), SharedResponseData::num_scalar_primary(), Dakota::read_coord_values(), Dakota::read_covariance(), Dakota::read_field_values(), ExperimentData::read_scalar_sigma(), ExperimentData::scalarDataFilename, ExperimentData::scalarSigmaPerRow, Response::set_full_covariance(), ExperimentData::simulationSRD, and ExperimentData::varianceTypes.

Referenced by ExperimentData::load_data().

The documentation for this class was generated from the following files:

- ExperimentData.hpp
- ExperimentData.cpp

14.75 ExperimentResponse Class Reference

Container class for response functions and their derivatives. ExperimentResponse provides the body class.

Inheritance diagram for ExperimentResponse:

```
Response
   
ExperimentResponse
```

Public Member Functions

- **ExperimentResponse ()**
  
  *default constructor*

- **ExperimentResponse (const Variables &vars, const ProblemDescDB &problem_db)**
  
  *standard constructor built from problem description database*

- **ExperimentResponse (const SharedResponseData &srd, const ActiveSet &set)**
  
  *alternate constructor that shares a SharedResponseData instance*

- **ExperimentResponse (const SharedResponseData &srd)**
  
  *alternate constructor that shares a SharedResponseData instance*

- **ExperimentResponse (const ActiveSet &set)**
  
  *alternate constructor using limited data*

- **~ExperimentResponse ()**
  
  *destructor*

- **void set_scalar_covariance (RealVector &scalars)**
  
  *method to set the covariance matrix defined for ExperimentResponse*

- **const ExperimentCovariance & experiment_covariance ()** const
430  CHAPTER 14.  CLASS DOCUMENTATION

retrieve the ExperimentCovariance structure

- void set_full_covariance (std::vector<RealMatrix> &matrices, std::vector<RealVector> &diagonals, RealVector &scalars, IntVector matrix_map_indices, IntVector diagonal_map_indices, IntVector scalar_map_indices)
  method to set the full covariance matrices for ExperimentResponse

- Real apply_covariance (const RealVector &residual) const
  method to compute the triple product v'*inv(C)*v.

- void apply_covariance_inv_sqrt (const RealVector &residuals, RealVector &weighted_residuals) const
  method to compute (v'*inv(C)' 1/2), to compute weighted residual

- void apply_covariance_inv_sqrt (const RealMatrix &gradients, RealMatrix &weighted_gradients) const

- void apply_covariance_inv_sqrt (const RealSymMatrixArray &hessians, RealSymMatrixArray &weighted_hessians) const

- void get_covariance_diagonal (RealVector &diagonal) const

- Real covariance_determinant () const
covariance determinant for this experiment (default 1.0)

- Real log_covariance_determinant () const
  log covariance determinant for this experiment (default 0.0)

Protected Member Functions

- void copy_rep (Response *source_resp_rep)
  Specialization of copy_rep; pulls base class data as well as derived specific data from the source rep into the this object.

Private Attributes

- ExperimentCovariance expDataCovariance
  sigma terms...

Additional Inherited Members

14.75.1 Detailed Description

Container class for response functions and their derivatives. ExperimentResponse provides the body class.

The ExperimentResponse class is the "representation" of the response container class. It is the "body" portion of the "handle-body idiom" (see Coplien "Advanced C++", p. 58). The handle class (Response) provides for memory efficiency in management of multiple response objects through reference counting and representation sharing. The body class (ExperimentResponse) actually contains the response data (functionValues, functionGradients, functionHessians, etc.). The representation is hidden in that an instance of ExperimentResponse may only be created by Response. Therefore, programmers create instances of the Response handle class, and only need to be aware of the handle/body mechanisms when it comes to managing shallow copies (shared representation) versus deep copies (separate representation used for history mechanisms).

The documentation for this class was generated from the following files:

- ExperimentResponse.hpp
- ExperimentResponse.cpp
14.76 FileReadException Class Reference

base class for Dakota file read exceptions (to allow catching both tabular and general file truncation issues)

Inheritance diagram for FileReadException:

```
FileReadException
   |       |
   |       |
ResultsFileError  TabularDataTruncated
```

Public Member Functions

- FileReadException (const std::string &msg)

14.76.1 Detailed Description

base class for Dakota file read exceptions (to allow catching both tabular and general file truncation issues)

The documentation for this class was generated from the following file:

- dakota_global_defs.hpp

14.77 ForkApplicInterface Class Reference

Derived application interface class which spawns simulation codes using fork/execvp/waitpid.

Inheritance diagram for ForkApplicInterface:

```
Interface
   |       |
   |       |
ApplicationInterface
   |       |
   |       |
ProcessApplicInterface
   |       |
   |       |
ProcessHandleApplicInterface
   |       |
   |       |
ForkApplicInterface
```

Public Member Functions

- ForkApplicInterface (const ProblemDescDB &problem_db)
  
  constructor

- ~ForkApplicInterface ()
  
  destructor
CHAPTER 14. CLASS DOCUMENTATION

Protected Member Functions

- void wait_local_evaluation_sequence (PRPQueue &prp_queue)
  version of wait_local_evaluations() managing of set of individual asynchronous evaluations
- void test_local_evaluation_sequence (PRPQueue &prp_queue)
  version of test_local_evaluations() managing of set of individual asynchronous evaluations
- pid_t create_analysis_process (bool block_flag, bool new_group)
  spawn a child process for an analysis component within an evaluation using fork()/execvp() and wait for completion using waitpid() if block_flag is true
- size_t wait_local_analyses ()
  wait for asynchronous analyses on the local processor, completing at least one job
- size_t test_local_analyses_send (int analysis_id)
  test for asynchronous analysis completions on the local processor and return results for any completions by sending messages
- void join_evaluation_process_group (bool new_group)
  create (if new_group) and join the process group for asynch evaluations
- void join_analysis_process_group (bool new_group)
  create (if new_group) and join the process group for asynch analyses
- void evaluation_process_group_id (pid_t pgid)
  set evalProcGroupId
- pid_t evaluation_process_group_id () const
  return evalProcGroupId
- void analysis_process_group_id (pid_t pgid)
  set analysisProcGroupId
- pid_t analysis_process_group_id () const
  return analysisProcGroupId
- pid_t wait_evaluation (bool block_flag)
  process all available completions within the evaluation process group; if block_flag = true, wait for at least one completion
- pid_t wait_analysis (bool block_flag)
  process all available completions within the analysis process group; if block_flag = true, wait for at least one completion
- void check_group (int err, pid_t proc_group_id)
  check the exit status of setpgid and abort if an error code was returned

Private Member Functions

- pid_t wait (pid_t proc_group_id, std::map< pid_t, int > &process_id_map, bool block_flag)
  core code used by wait_{evaluation,analysis}()
- void join_process_group (pid_t &process_group_id, bool new_group)
  core code used by join_{evaluation,analysis}_process_group()
14.78. FSUDESIGNCOMPEXP CLASS REFERENCE

Private Attributes

- pid_t evalProcGroupId
  the process group id used to identify a set of child evaluation processes used by this interface instance (to distinguish from other interface instances that could be running at the same time)

- pid_t analysisProcGroupId
  the process group id used to identify a set of child analysis processes used by this interface instance (to distinguish from other interface instances that could be running at the same time)

Additional Inherited Members

14.77.1 Detailed Description

Derived application interface class which spawns simulation codes using fork/execvp/waitpid. 
ForkApplicInterface is used on Unix systems and is a peer to SpawnApplicInterface for Windows systems. 
The documentation for this class was generated from the following files:

- ForkApplicInterface.hpp
- ForkApplicInterface.cpp

14.78 FSUDesignCompExp Class Reference

Wrapper class for the FSUDace QMC/CVT library.

Inheritance diagram for FSUDesignCompExp:

```
  FSUDesignCompExp
     └─ PStudyDACE
       └─ Analyzer
         └─ Iterator
```

Public Member Functions

- FSUDesignCompExp (ProblemDescDB &problem_db, Model &model)
  primary constructor for building a standard DACE iterator

- FSUDesignCompExp (Model &model, int samples, int seed, unsigned short sampling_method)
  alternate constructor for building a DACE iterator on-the-fly

- ~FSUDesignCompExp ()
  destructor

- bool resize ()
  reinitializes iterator based on new variable size
Protected Member Functions

- `void pre_run ()`
  pre-run portion of run (optional); re-implemented by Iterators which can generate all Variables (parameter sets) a priori
- `void core_run ()`
  core portion of run; implemented by all derived classes and may include pre/post steps in lieu of separate pre/post
- `void post_input ()`
  read tabular data for post-run mode
- `void post_run (std::ostream &s)`
  post-run portion of run (optional); verbose to print results; re-implemented by Iterators that can read all Variables/Responses and perform final analysis phase in a standalone way
- `int num_samples () const`
- `void sampling_reset (int min_samples, bool all_data_flag, bool stats_flag)`
  reset sampling iterator to use at least min_samples
- `unsigned short sampling_scheme () const`
  return sampling name
- `void vary_pattern (bool pattern_flag)`
  sets varyPattern in derived classes that support it
- `void get_parameter_sets (Model &model)`
  Generate one block of numSamples samples (ndim * num_samples), populating allSamples; ParamStudy is the only class that specializes to use allVariables.
- `void get_parameter_sets (Model &model, const int num_samples, RealMatrix &design_matrix)`
  Generate one block of numSamples samples (ndim * num_samples), populating design_matrix.

Private Member Functions

- `void enforce_input_rules ()`
  enforce sanity checks/modifications for the user input specification

Private Attributes

- `int samplesSpec`
  initial specification of number of samples
- `int numSamples`
  current number of samples to be evaluated
- `bool allDataFlag`
  flag which triggers the update of allVars/allResponses for use by Iterator::all_variables() and Iterator::all_responses()
- `size_t numDACERuns`
  counter for number of executions for this object
- `bool latinizeFlag`
  flag which specifies latinization of QMC or CVT sample sets
- `IntVector sequenceStart`
  Integer vector defining a starting index into the sequence for random variable sampled. Default is 0 0 0 (e.g. for three random variables).
• IntVector sequenceLeap
  Integer vector defining the leap number for each sequence being generated. Default is 1 1 1 (e.g. for three random vars.)

• IntVector primeBase
  Integer vector defining the prime base for each sequence being generated. Default is 2 3 5 (e.g., for three random vars.)

• int seedSpec
  the user seed specification for the random number generator (allows repeatable results)

• int randomSeed
  current seed for the random number generator

• bool varyPattern
  flag for continuing the random number or QMC sequence from a previous execution (e.g., for surrogate-based optimization) so that multiple executions are repeatable but not identical.

• int numCVTTrials
  specifies the number of sample points taken at internal CVT iteration

• int trialType
  Trial type in CVT. Specifies where the points are placed for consideration relative to the centroids. Choices are grid (2), halton (1), uniform (0), or random (-1). Default is random.

### Additional Inherited Members

#### 14.78.1 Detailed Description

Wrapper class for the FSUDace QMC/CVT library.

The FSUDesignCompExp class provides a wrapper for FSUDace, a C++ design of experiments library from Florida State University. This class uses quasi Monte Carlo (QMC) and Centroidal Voronoi Tessellation (CVT) methods to uniformly sample the parameter space spanned by the active bounds of the current Model. It returns all generated samples and their corresponding responses as well as the best sample found.

#### 14.78.2 Constructor & Destructor Documentation

**FSUDesignCompExp ( ProblemDescDB & problem_db, Model & model )**

primary constructor for building a standard DACE iterator

This constructor is called for a standard iterator built with data from probDescDB.

References Dakota::abort_handler(), ProblemDescDB::get_bool(), ProblemDescDB::get_int(), ProblemDescDB::get_iv(), ProblemDescDB::get_string(), Iterator::maxEvalConcurrency, Iterator::methodName, Analyzer::numContinuousVars, FSUDesignCompExp::numCVTTrials, Analyzer::numDiscreteIntVars, Analyzer::numDiscreteRealVars, Analyzer::numDiscreteStringVars, FSUDesignCompExp::numSamples, FSUDesignCompExp::primeBase, Iterator::probDescDB, FSUDesignCompExp::randomSeed, FSUDesignCompExp::seedSpec, FSUDesignCompExp::sequenceLeap, FSUDesignCompExp::sequenceStart, FSUDesignCompExp::trialType, and FSUDesignCompExp::varyPattern.

**FSUDesignCompExp ( Model & model, int samples, int seed, unsigned short sampling_method )**

alternate constructor for building a DACE iterator on-the-fly

This alternate constructor is used for instantiations on-the-fly, using only the incoming data. No problem description database queries are used.
14.78.3 Member Function Documentation

void prerun() [protected], [virtual]

pre-run portion of run (optional); re-implemented by Iterators which can generate all Variables (parameter sets) a priori

pre-run phase, which a derived iterator may optionally reimplement; when not present, pre-run is likely integrated into the derived run function. This is a virtual function; when re-implementing, a derived class must call its nearest parent’s prerun(), if implemented, typically before performing its own implementation steps.

Reimplemented from Analyzer.

References Dakota::abort_handler(), Iterator::maxEvalConcurrency, Analyzer::numContinuousVars, FSUDesignCompExp::numCVTTrials, Analyzer::numDiscreteIntVars, Analyzer::numDiscreteRealVars, Analyzer::numDiscreteStringVars, FSUDesignCompExp::numSamples, FSUDesignCompExp::primeBase, FSUDesignCompExp::randomSeed, FSUDesignCompExp::seedSpec, FSUDesignCompExp::sequenceLeap, FSUDesignCompExp::sequenceStart, and FSUDesignCompExp::trialType.

void corerun() [protected], [virtual]

core portion of run; implemented by all derived classes and may include pre/post steps in lieu of separate pre/post Virtual run function for the iterator class hierarchy. All derived classes need to redefine it.

Reimplemented from Iterator.

References FSUDesignCompExp::allDataFlag, Analyzer::evaluateParameterSets(), Iterator::iteratedModel, Analyzer::numLSqTerms, Analyzer::numObjFns, and Iterator::subIteratorFlag.

void postrun(std::ostream & s) [protected], [virtual]

post-run portion of run (optional); verbose to print results; re-implemented by Iterators that can read all Variables/Responses and perform final analysis phase in a standalone way

Post-run phase, which a derived iterator may optionally reimplement; when not present, post-run is likely integrated into run. This is a virtual function; when re-implementing, a derived class must call its nearest parent’s postrun(), typically after performing its own implementation steps.

Reimplemented from Analyzer.

References Analyzer::allResponses, Analyzer::allSamples, SensAnalysisGlobal::compute_correlations(), Analyzer::compute_vbd_stats(), FSUDesignCompExp::enforce_input_rules(), FSUDesignCompExp::numSamples, Analyzer::postrun(), PStudyDACE::pStudyDACESENSGlobal, Iterator::subIteratorFlag, and PStudyDACE::varBasedDecompFlag.

int num_samples() const [inline], [protected], [virtual]

Return current number of evaluation points. Since the calculation of samples, collocation points, etc. might be costly, provide a default implementation here that backs out from the maxEvalConcurrency.

Reimplemented from Analyzer.

References FSUDesignCompExp::numSamples.

Referenced by FSUDesignCompExp::get_parameter_sets().
void enforce_input_rules() [private]

enforce sanity checks/modifications for the user input specification

Users may input a variety of quantities, but this function must enforce any restrictions imposed by the sampling algorithms.

References Dakota::abort_handler(), Iterator::methodName, Analyzer::numContinuousVars, FSUDesignCompExp::numSamples, and FSUDesignCompExp::primeBase.

Referenced by FSUDesignCompExp::post_input(), FSUDesignCompExp::post_run(), and FSUDesignCompExp::pre_run().

The documentation for this class was generated from the following files:

- FSUDesignCompExp.hpp
- FSUDesignCompExp.cpp

### 14.79 FunctionEvalFailure Class Reference

exception class for function evaluation failures

Inherits runtime_error.

#### Public Member Functions

- FunctionEvalFailure (const std::string &msg)

### 14.79.1 Detailed Description

exception class for function evaluation failures

The documentation for this class was generated from the following file:

- dakota_global_defs.hpp

### 14.80 GaussianProcess Class Reference

The GaussianProcess constructs a Gaussian Process regressor surrogate given a matrix of data.

Inheritance diagram for GaussianProcess:

```
GaussianProcess
<table>
<thead>
<tr>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td>Surrogate</td>
</tr>
<tr>
<td>------------------</td>
</tr>
</tbody>
</table>

GaussianProcess
```

#### Public Member Functions

- GaussianProcess ()
  
  Constructor that uses defaultConfigOptions and does not build.

- GaussianProcess (const Teuchos::ParameterList &param_list)
  
  Constructor that sets configOptions but does not build.
**GaussianProcess** (const `MatrixXd` &samples, const `MatrixXd` &response, const Teuchos::ParameterList &param_list)

Constructor for the `GaussianProcess` that sets configOptions and builds the GP.

**~GaussianProcess** ()

Default destructor.

- **void build** (const `MatrixXd` &samples, const `MatrixXd` &response) override

  Build the GP using specified build data.

- **void value** (const `MatrixXd` &samples, `MatrixXd` &approx_values) override

  Evaluate the Gaussian Process at a set of prediction points.

- **void gradient** (const `MatrixXd` &samples, `MatrixXd` &gradient, const int qoi=0) override

  Evaluate the gradient of the Gaussian process at a set of prediction points.

- **void hessian** (const `MatrixXd` &sample, `MatrixXd` &hessian, const int qoi=0) override

  Evaluate the Hessian of the Gaussian process at a single point.

- **void negative_marginal_log_likelihood** (double &obj_value, `VectorXd` &obj_gradient)

  Evaluate the negative marginal loglikelihood and its gradient.

- **const VectorXd & get_posterior_std_dev** () const

  Get the standard deviations for the set of prediction points passed to value.

- **const MatrixXd & get_posterior_covariance** () const

  Get the covariance matrix for the set of prediction points passed to value.

- **int get_num_opt_variables** ()

  Get the number of optimization variables.

- **int get_num_variables** () const

  Get the dimension of the feature space.

- **void set_opt_params** (const std::vector<double> &opt_params)

  Update the vector of optimization parameters.

**Private Member Functions**

- **void default_options** () override

  Construct and populate the defaultConfigOptions.

- **void compute_build_dists** ()

  Compute squared distances between the scaled build points.

- **void compute_pred_dists** (const `MatrixXd` &scaled_pred_pts)

  Compute distances between build and prediction points. This includes build-prediction and prediction-prediction distance matrices.

- **void compute_gram** (const std::vector<`MatrixXd`> &dists2, bool add_nugget, bool compute_derivs, `MatrixXd` &gram)

  Compute a Gram matrix given a vector of squared distances and optionally compute its derivatives and/or adds nugget terms.

- **void compute_first_deriv_pred_gram** (const `MatrixXd` &pred_gram, const int index, `MatrixXd` &first_deriv_pred_gram)

  Compute the first derivative of the prediction matrix for a given component.

- **void compute_second_deriv_pred_gram** (const `MatrixXd` &second_deriv_pred_gram)
Compute the second derivative of the prediction matrix for a pair of components.

- void generate_initial_guesses (const VectorXd &sigma_bounds, const MatrixXd &length_scale_bounds, const VectorXd &nugget_bounds, const int num_restarts, MatrixXd &initial_guesses)
  Randomly generate initial guesses for the optimization routine.

- void setup_default_optimization_params (Teuchos::RCP<Teuchos::ParameterList> rol_params)
  Set the default optimization parameters for ROL for GP hyperparameter estimation.

Private Attributes

- double fixedNuggetValue
  Small constant added to the diagonal of the Gram matrix to avoid ill-conditioning.

- MatrixXd eyeMatrix
  Identity matrix for the build points space.

- MatrixXd basisMatrix
  Basis matrix for the sample points in polynomial regression.

- MatrixXd targetValues
  Target values for the surrogate dataset.

- VectorXd thetaValues
  Vector of log-space hyperparameters.

- VectorXd betaValues
  Vector of polynomial coefficients.

- double estimatedNuggetValue
  Estimated nugget term.

- VectorXd bestThetaValues
  Vector of best hyperparameters from MLE with restarts.

- VectorXd bestBetaValues
  Vector of best polynomial coefficients from MLE with restarts.

- double bestEstimatedNuggetValue
  Best estimated nugget value from MLE with restarts.

- VectorXd objectiveFunctionHistory
  Final objective function values for each optimization run.

- MatrixXd GramMatrix
  Gram matrix for the build points.

- std::vector< MatrixXd > GramMatrixDerivs
  Derivatives of the Gram matrix w.r.t. the hyperparameters.

- std::vector< MatrixXd > cwiseDists2
  Squared component-wise distances between points in the surrogate dataset.

- std::vector< MatrixXd > cwiseMixedDists
  Component-wise distances between prediction and build points.

- std::vector< MatrixXd > cwiseMixedDists2
  Squared component-wise distances between prediction and build points.

- std::vector< MatrixXd > cwisePredDists2
  Component-wise distances between prediction points.

- Eigen::LDLT< MatrixXd > CholFact
Pivoted Cholesky factorization.

- **MatrixXd posteriorCov**
  Posterior covariance matrix for prediction points.

- **VectorXd posteriorStdDev**
  Vector of posterior standard deviation at prediction points.

- **std::shared_ptr < PolynomialRegression > polyRegression**
  PolynomialRegression for trend function.

- **const double betaBound = 1.0e20**
  Large constant for polynomial coefficients upper/lower bounds.

- **bool estimateTrend**
  Bool for polynomial trend (i.e. semi-parametric GP) estimation.

- **int numPolyTerms = 0**
  Number of terms in polynomial trend.

- **int numNuggetTerms = 0**
  Number of terms for the (estimated) nugget parameter.

- **bool estimateNugget**
  Bool for nugget estimation.

- **double bestObjFunValue = std::numeric_limits<double>::max()**
  Final objective function value.

- **const double PI = 3.14159265358979323846**
  Numerical constant – needed for negative marginal log-likelihood.

**Additional Inherited Members**

**14.80.1 Detailed Description**

The GaussianProcess constructs a Gaussian Process regressor surrogate given a matrix of data.

The Gaussian Process (GP) uses an anisotropic squared exponential kernel with a constant multiplicative scaling factor. This yields a total of num_features + 1 kernel hyperparameters. These parameters are internally transformed to a log-space vector (theta) for optimization and evaluation of the GP. Polynomial trend and nugget estimation are optional.

The GP’s parameters are determined through maximum likelihood estimation (MLE) via minimization of the negative marginal log-likelihood function. ROL’s implementation of L-BFGS-B is used to solve the optimization problem, and the algorithm may be run from multiple random initial guesses to increase the chance of finding the global minimum.

Once the GP is constructed its mean, standard deviation, and covariance matrix can be computed for a set of prediction points. Gradients and Hessians are available.

**14.80.2 Constructor & Destructor Documentation**

GaussianProcess ( const Teuchos::ParameterList & param_list )

Constructor that sets configOptions but does not build.
14.80. GAUSSIANPROCESS CLASS REFERENCE

Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>param_list</th>
<th>List that overrides entries in defaultConfigOptions.</th>
</tr>
</thead>
</table>

References Surrogate::configOptions, and GaussianProcess::default_options().

GaussianProcess ( const MatrixXd & samples, const MatrixXd & response, const Teuchos::ParameterList & param_list )

Constructor for the GaussianProcess that sets configOptions and builds the GP.
Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>samples</th>
<th>Matrix of data for surrogate construction - (num_samples by num_features)</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>response</td>
<td>Vector of targets for surrogate construction - (num_samples by num_qoi = 1; only 1 response is supported currently).</td>
</tr>
<tr>
<td>in</td>
<td>param_list</td>
<td>List that overrides entries in defaultConfigOptions</td>
</tr>
</tbody>
</table>

References GaussianProcess::build(), Surrogate::configOptions, and GaussianProcess::default_options().

14.80.3 Member Function Documentation

void build ( const MatrixXd & samples, const MatrixXd & response ) [override], [virtual]

Build the GP using specified build data.
Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>samples</th>
<th>Matrix of data for surrogate construction - (num_samples by num_features)</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>response</td>
<td>Vector of targets for surrogate construction - (num_samples by num_qoi = 1; only 1 response is supported currently).</td>
</tr>
</tbody>
</table>

Implements Surrogate.
References GaussianProcess::basisMatrix, GaussianProcess::bestBetaValues, GaussianProcess::bestEstimatedNuggetValue, GaussianProcess::bestObjFunValue, GaussianProcess::bestThetaValues, GaussianProcess::betaBound, GaussianProcess::bestValues, GaussianProcess::compute_build_dists(), Surrogate::configOptions, Surrogate::dataScaler, Surrogate::defaultConfigOptions, GaussianProcess::estimatedNuggetValue, GaussianProcess::estimateNugget, GaussianProcess::estimateTrend, GaussianProcess::eyeMatrix, GaussianProcess::fixedNuggetValue, GaussianProcess::generate_initial_guesses(), GaussianProcess::GramMatrix, GaussianProcess::GramMatrixDerivs, GaussianProcess::negative_marginal_log_likelihood(), GaussianProcess::numNuggetTerms, GaussianProcess::numPolyTerms, GaussianProcess::numQOI, Surrogate::numSamples, Surrogate::numVariables, GaussianProcess::objectiveFunctionHistory, GaussianProcess::polyRegression, dakota::util::scaler_factory(), DataScaler::scaler_type(), GaussianProcess::setup_default_optimization_params(), GaussianProcess::targetValues, and GaussianProcess::thetaValues.

Referenced by GaussianProcess::GaussianProcess().

void value ( const MatrixXd & samples, MatrixXd & approx_values ) [override], [virtual]

Evaluate the Gaussian Process at a set of prediction points.
Parameters

| in | samples | Matrix of prediction points - (num_pts by num_features). |
**void gradient** ( const MatrixXd & samples, MatrixXd & gradient, const int qoi = 0 ) [override], [virtual]

Evaluate the gradient of the Gaussian process at a set of prediction points.

Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>samples</th>
<th>Coordinates of the prediction points - (num_pts by num_features).</th>
</tr>
</thead>
<tbody>
<tr>
<td>out</td>
<td>gradient</td>
<td>Matrix of gradient vectors at the prediction points - (num_pts by num_features).</td>
</tr>
<tr>
<td>out</td>
<td>qoi</td>
<td>Index of response/QOI for which to compute derivatives</td>
</tr>
</tbody>
</table>

Reimplemented from Surrogate.

References GaussianProcess::basisMatrix, GaussianProcess::betaValues, GaussianProcess::CholFact, GaussianProcess::compute_gram(), GaussianProcess::compute_pred_dists(), GaussianProcess::cwiseDists2, GaussianProcess::cwiseMixedDists2, Surrogate::dataScaler, GaussianProcess::estimateTrend, GaussianProcess::GramMatrix, Surrogate::numVariables, GaussianProcess::polyRegression, GaussianProcess::posteriorCov, GaussianProcess::posteriorStdDev, and GaussianProcess::targetValues.

**void hessian** ( const MatrixXd & sample, MatrixXd & hessian, const int qoi = 0 ) [override], [virtual]

Evaluate the Hessian of the Gaussian process at a single point.

Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>sample</th>
<th>Coordinates of the prediction point - (num_samples by num_features).</th>
</tr>
</thead>
<tbody>
<tr>
<td>out</td>
<td>hessian</td>
<td>Hessian matrix at the prediction point - (num_features by num_features).</td>
</tr>
<tr>
<td>out</td>
<td>qoi</td>
<td>Index of response/QOI for which to compute derivatives</td>
</tr>
</tbody>
</table>

Reimplemented from Surrogate.

References GaussianProcess::basisMatrix, GaussianProcess::betaValues, GaussianProcess::CholFact, GaussianProcess::compute_first_deriv_pred_gram(), GaussianProcess::compute_gram(), GaussianProcess::compute_pred_dists(), GaussianProcess::cwiseDists2, GaussianProcess::cwiseMixedDists2, Surrogate::dataScaler, GaussianProcess::estimateTrend, GaussianProcess::GramMatrix, Surrogate::numVariables, GaussianProcess::polyRegression, dakota::silence_unused_args(), and GaussianProcess::targetValues.

**void negative_marginal_log_likelihood** ( double & obj_value, VectorXd & obj_gradient )

Evaluate the negative marginal loglikelihood and its gradient.
Parameters

<table>
<thead>
<tr>
<th>out</th>
<th>obj_value</th>
<th>Value of the objection function.</th>
</tr>
</thead>
<tbody>
<tr>
<td>out</td>
<td>obj_grad</td>
<td>Gradient of the objective function.</td>
</tr>
</tbody>
</table>

References GaussianProcess::basisMatrix, GaussianProcess::betaValues, GaussianProcess::CholFact, GaussianProcess::compute_gram(), GaussianProcess::cwiseDists2, GaussianProcess::estimatedNuggetValue, GaussianProcess::estimateNugget, GaussianProcess::estimateTrend, GaussianProcess::eyeMatrix, GaussianProcess::GramMatrix, GaussianProcess::GramMatrixDerivs, GaussianProcess::numPolyTerms, Surrogate::numSamples, Surrogate::numVariables, GaussianProcess::PI, and GaussianProcess::targetValues.

Referenced by GaussianProcess::build(), GP_Objective::gradient(), and GP_Objective::value().

```cpp
const VectorXd & get_posterior_std_dev ( ) const
```

Get the standard deviations for the set of prediction points passed to value.

Returns

Vector of standard deviations for the prediction points.

References GaussianProcess::posteriorStdDev.

```cpp
const MatrixXd & get_posterior_covariance ( ) const
```

Get the covariance matrix for the set of prediction points passed to value.

Returns

Covariance matrix for the prediction points.

References GaussianProcess::posteriorCov.

```cpp
int get_num_opt_variables ( )
```

Get the number of optimization variables.

Returns

Number of total optimization variables (hyperparameters + trend coefficients + nugget)

References GaussianProcess::numNuggetTerms, GaussianProcess::numPolyTerms, and Surrogate::numVariables. Referenced by GP_Objective::GP_Objective().

```cpp
int get_num_variables ( ) const
```

Get the dimension of the feature space.

Returns

numVariables The dimension of the feature space.

References Surrogate::numVariables.

```cpp
void set_opt_params ( const std::vector< double > & opt_params )
```

Update the vector of optimization parameters.
Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>opt_params</code></td>
<td>Vector of optimization parameter values.</td>
</tr>
</tbody>
</table>

References GaussianProcess::betaValues, GaussianProcess::estimatedNuggetValue, GaussianProcess::estimateNugget, GaussianProcess::estimateTrend, GaussianProcess::numPolyTerms, Surrogate::numVariables, and GaussianProcess::thetaValues.

Referenced by GP::Objective::gradient(), and GP::Objective::value().

```cpp
void compute_pred_dists ( const MatrixXd & scaled_pred_pts ) [private]
```

Compute distances between build and prediction points. This includes build-prediction and prediction-prediction distance matrices.

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>scaled_pred_pts</code></td>
<td>Matrix of scaled prediction points.</td>
</tr>
</tbody>
</table>

References GaussianProcess::cwiseMixedDists, GaussianProcess::cwiseMixedDists2, GaussianProcess::cwisePredDists2, Surrogate::dataScaler, Surrogate::numSamples, and Surrogate::numVariables.

Referenced by GaussianProcess::gradient(), GaussianProcess::hessian(), and GaussianProcess::value().

```cpp
void compute_gram ( const std::vector<MatrixXd> & dists2, bool add_nugget, bool compute_derivs, MatrixXd & gram ) [private]
```

Compute a Gram matrix given a vector of squared distances and optionally compute its derivatives and/or adds nugget terms.

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>dists2</code></td>
<td>Vector of squared distance matrices.</td>
</tr>
<tr>
<td><code>add_nugget</code></td>
<td>Bool for whether or add nugget terms.</td>
</tr>
<tr>
<td><code>compute_derivs</code></td>
<td>Bool for whether or not to compute the derivatives of the Gram matrix.</td>
</tr>
<tr>
<td><code>gram</code></td>
<td>Gram matrix.</td>
</tr>
</tbody>
</table>

References GaussianProcess::estimatedNuggetValue, GaussianProcess::estimateNugget, GaussianProcess::fixedNuggetValue, GaussianProcess::GramMatrixDerivs, Surrogate::numVariables, and GaussianProcess::thetaValues.

Referenced by GaussianProcess::gradient(), GaussianProcess::hessian(), GaussianProcess::negative_marginal_log_likelihood(), and GaussianProcess::value().

```cpp
void compute_first_deriv_pred_gram ( const MatrixXd & pred_gram, const int index, MatrixXd & first_deriv_pred_gram ) [private]
```

Compute the first derivative of the prediction matrix for a given component.

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>pred_gram</code></td>
<td>Prediction Gram matrix - Rectangular matrix of kernel evaluations between the surrogate and prediction points.</td>
</tr>
<tr>
<td><code>index</code></td>
<td>Specifies the component of the derivative.</td>
</tr>
<tr>
<td><code>first_deriv_pred_gram</code></td>
<td>First derivative of the prediction matrix for a given component.</td>
</tr>
</tbody>
</table>

References GaussianProcess::cwiseMixedDists, and GaussianProcess::thetaValues.

Referenced by GaussianProcess::gradient().
void compute_second_deriv_pred_gram ( const MatrixXd & pred_gram, const int index_i, const int index_j, MatrixXd & second_deriv_pred_gram ) [private]

Compute the second derivative of the prediction matrix for a pair of components.
Parameters

<table>
<thead>
<tr>
<th>Type</th>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td><code>pred_gram</code></td>
<td>Prediction Gram matrix - Rectangular matrix of kernel evaluations between the surrogate and prediction points.</td>
</tr>
<tr>
<td>in</td>
<td><code>index_i</code></td>
<td>Specifies the first component of the second derivative.</td>
</tr>
<tr>
<td>in</td>
<td><code>index_j</code></td>
<td>Specifies the second component of the second derivative.</td>
</tr>
<tr>
<td>out</td>
<td><code>second_deriv_pred_gram</code></td>
<td>Second derivative of the prediction matrix for a pair of components.</td>
</tr>
</tbody>
</table>

References GaussianProcess::cwiseMixedDists, and GaussianProcess::thetaValues.
Referenced by GaussianProcess::hessian().

```cpp
void generate_initial_guesses ( const VectorXd & sigma_bounds, const MatrixXd & length_scale_bounds, const VectorXd & nugget_bounds, const int num_restarts, MatrixXd & initial_guesses ) [private]
```
Randomly generate initial guesses for the optimization routine.

Parameters

<table>
<thead>
<tr>
<th>Type</th>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td><code>sigma_bounds</code></td>
<td>Bounds for the scaling hyperparameter (sigma).</td>
</tr>
<tr>
<td>in</td>
<td><code>length_scale_bounds</code></td>
<td>Bounds for the length scales for each feature (l).</td>
</tr>
<tr>
<td>in</td>
<td><code>nugget_bounds</code></td>
<td>Bounds for the nugget term.</td>
</tr>
<tr>
<td>in</td>
<td><code>num_restarts</code></td>
<td>Number of restarts for the optimizer.</td>
</tr>
<tr>
<td>out</td>
<td><code>initial_guesses</code></td>
<td>Matrix of initial guesses.</td>
</tr>
</tbody>
</table>

References GaussianProcess::estimateNugget, GaussianProcess::estimateTrend, GaussianProcess::numNuggetTerms, GaussianProcess::numPolyTerms, and Surrogate::numVariables.
Referenced by GaussianProcess::build().

```cpp
void setup_default_optimization_params ( Teuchos::RCP<Teuchos::ParameterList> rol_params ) [private]
```
Set the default optimization parameters for ROL for GP hyperparameter estimation.

Parameters

<table>
<thead>
<tr>
<th>Type</th>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td><code>rol_params</code></td>
<td>RCP to a Teuchos::ParameterList of ROL's options.</td>
</tr>
</tbody>
</table>

Referenced by GaussianProcess::build().

The documentation for this class was generated from the following files:

- GaussianProcess.hpp
- GaussianProcess.cpp

### 14.81 GaussProcApproximation Class Reference

Derived approximation class for Gaussian Process implementation.

Inheritance diagram for GaussProcApproximation:
Public Member Functions

- GaussProcApproximation ()
  default constructor
- GaussProcApproximation (const SharedApproxData &shared_data)
  alternate constructor
- GaussProcApproximation (const ProblemDescDB &problem_db, const SharedApproxData &shared_data, const String &approx_label)
  standard constructor
- ~GaussProcApproximation ()
  destructor

Protected Member Functions

- int min_coefficients () const
  return the minimum number of samples (unknowns) required to build the derived class approximation type in n_vars dimensions
- void build ()
  find the covariance parameters governing the Gaussian process response
- Real value (const Variables &vars)
  retrieve the function value for a given parameter set
- const RealVector & gradient (const Variables &vars)
  retrieve the function gradient at the predicted value for a given parameter set
- Real prediction_variance (const Variables &vars)
  retrieve the variance of the predicted value for a given parameter set

Private Member Functions

- void GPmodel_build ()
  Function to compute hyperparameters governing the GP.
- void GPmodel_apply (const RealVector &new_x, bool variance_flag, bool gradients_flag)
  Function returns a response value using the GP surface.
- void normalize_training_data ()
  Normalizes the initial inputs upon which the GP surface is based.
- void get_trend ()
  Gets the trend (basis) functions for the calculation of the mean of the GP If the order = 0, the trend is a constant, if the order = 1, trend is linear, if order = 2, trend is quadratic.
- void get_beta_coefficients ()
  Gets the beta coefficients for the calculation of the mean of the GP.
- int get_cholesky_factor ()
  Gets the Cholesky factorization of the covariance matrix, with error checking.
- void get_process_variance ()
  Gets the estimate of the process variance given the values of beta and the correlation lengthscales.
- void get_cov_matrix ()
  calculates the covariance matrix for a given set of input points
void get_cov_vector ()

calculates the covariance vector between a new point x and the set of inputs upon which the GP is based

void optimize_theta_global ()

sets up and performs the optimization of the negative log likelihood to determine the optimal values of the covariance parameters using NCSUDirect

void optimize_theta_multipoint ()

sets up and performs the optimization of the negative log likelihood to determine the optimal values of the covariance parameters using a gradient-based solver and multiple starting points

void predict (bool variance_flag, bool gradients_flag)

Calculates the predicted new response value for x in normalized space.

Real calc_nll ()

calculates the negative log likelihood function (based on covariance matrix)

void calc_grad_nll ()

Gets the gradient of the negative log likelihood function with respect to the correlation lengthscales, theta.

void get_grad_cov_vector ()

Calculate the derivatives of the covariance vector, with respect to each component of x.

void run_point_selection ()

Runs the point selection algorithm, which will choose a subset of the training set with which to construct the GP model, and estimate the necessary parameters.

void initialize_point_selection ()

Initializes the point selection routine by choosing a small initial subset of the training points.

void pointsel_get_errors (RealArray &delta)

Uses the current GP model to compute predictions at all of the training points and find the errors.

int addpoint (int, IntArray &added_index)

Adds a point to the effective training set. Returns 1 on success.

int pointsel_add_sel (const RealArray &delta)

Accepts a vector of unsorted prediction errors, determines which points should be added to the effective training set, and adds them.

Real maxval (const RealArray &) const

Return the maximum value of the elements in a vector.

void pointsel_write_points ()

Writes out the training set before and after point selection.

void lhood_2d_grid_eval ()

For problems with 2D input, evaluates the negative log likelihood on a grid.

void writex (const char []]

Writes out the current training set (in original units) to a specified file.

void writeCovMat (char []]

Writes out the covariance matrix to a specified file.
Static Private Member Functions

- static void negloglik (int mode, int n, const Teuchos::SerialDenseVector<int, double>& X, Real &fx, Teuchos::SerialDenseVector<int, double>& grad_x, int &result_mode)
  
  static function used by OPT++ as the objective function to optimize the hyperparameters in the covariance of the GP by minimizing the negative log likelihood

- static void constraint_eval (int mode, int n, const Teuchos::SerialDenseVector<int, double>& X, Teuchos::SerialDenseVector<int, double>& g, Teuchos::SerialDenseMatrix<int, double>& gradC, int &result_mode)
  
  static function used by OPT++ as the constraint function in the optimization of the negative log likelihood. Currently this function is empty: it is an unconstrained optimization.

- static double negloglikNCSU (const RealVector &x)
  
  function used by NCSUOptimizer to optimize negloglik objective

Private Attributes

- Real approxValue
  
  value of the approximation returned by value()

- Real approxVariance
  
  value of the approximation returned by prediction_variance()

- RealMatrix trainPoints
  
  A 2-D array (num sample sites = rows, num vars = columns) used to create the Gaussian process.

- RealMatrix trainValues
  
  An array of response values; one response value per sample site.

- RealVector trainMeans
  
  The mean of the input columns of trainPoints.

- RealVector trainStdvs
  
  The standard deviation of the input columns of trainPoints.

- RealMatrix normTrainPoints
  
  Current working set of normalized points upon which the GP is based.

- RealMatrix trendFunction
  
  matrix to hold the trend function

- RealMatrix betaCoeffs
  
  matrix to hold the beta coefficients for the trend function

- RealSymMatrix covMatrix
  
  The covariance matrix where each element (i,j) is the covariance between points Xi and Xj in the initial set of samples.

- RealMatrix covVector
  
  The covariance vector where each element (j,0) is the covariance between a new point X and point Xj from the initial set of samples.

- RealMatrix approxPoint
  
  Point at which a prediction is requested. This is currently a single point, but it could be generalized to be a vector of points.

- RealMatrix gradNegLogLikTheta
  
  matrix to hold the gradient of the negative log likelihood with respect to the theta correlation terms
• Teuchos::SerialSpdDenseSolver
  < int, Real > covSlvr
  The global solver for all computations involving the inverse of the covariance matrix.
• RealMatrix gradCovVector
  A matrix, where each column is the derivative of the covVector with respect to a particular component of X.
• RealMatrix normTrainPointsAll
  Set of all original samples available.
• RealMatrix trainValuesAll
  All original samples available.
• RealMatrix trendFunctionAll
  Trend function values corresponding to all original samples.
• RealMatrix Rinv
  Matrix for storing inverse of correlation matrix Rinv\ast(Y-FB)
• size_t numObs
  The number of observations on which the GP surface is built.
• size_t numObsAll
  The original number of observations.
• short trendOrder
  The number of variables in each X variable (number of dimensions of the problem).
• RealVector thetaParams
  Theta is the vector of covariance parameters for the GP. We determine the values of theta by optimization Currently, the covariance function is \theta[0]\ast\exp(-0.5\ast\mathrm{sume})+\delta\ast\text{pow}(\sigma_\text{e}, 2). \ \mathrm{sume} is the sum squared of weighted distances; it involves a sum of \theta[1](X_i(1)-X_j(1))^2 + \theta[2](X_i(2)-X_j(2))^2 + ... where X_i(1) is the first dimension value of multi-dimensional variable X_i. \delta\ast\text{pow}(\sigma_\text{e}, 2) is a jitter term used to improve matrix computations. \delta is zero for the covariance between different points and 1 for the covariance between the same point. \sigma_\text{e} is the underlying process error.
• Real procVar
  The process variance, the multiplier of the correlation matrix.
• IntArray pointsAddedIndex
  Used by the point selection algorithm, this vector keeps track all points which have been added.
• int cholFlag
  A global indicator for success of the Cholesky factorization.
• bool usePointSelection
  A flag to indicate the use of point selection

Static Private Attributes
• static GaussProcApproximation * GPinstance
  pointer to the active object instance used within the static evaluator

Additional Inherited Members

14.81.1 Detailed Description
Derived approximation class for Gaussian Process implementation.
The GaussProcApproximation class provides a global approximation (surrogate) based on a Gaussian process. The Gaussian process is built after normalizing the function values, with zero mean. Opt++ is used to determine the optimal values of the covariance parameters, those which minimize the negative log likelihood function.
14.81.2 Constructor & Destructor Documentation

GaussProcApproximation ( ) [inline]
default constructor
alternate constructor used by EffGlobalOptimization and NonDGlobalReliability that does not use a problem
database defaults here are no point selectin and quadratic trend function.

14.81.3 Member Function Documentation

void GPmodel_apply ( const RealVector & new_x, bool variance_flag, bool gradients_flag ) [private]
Function returns a response value using the GP surface.
The response value is computed at the design point specified by the RealVector function argument.
References Dakota::abort_handler(), GaussProcApproximation::approxPoint, GaussProcApproximation::get_cov_vector(), SharedApproxData::numVars, GaussProcApproximation::predict(), Approximation::sharedDataRep, GaussProcApproximation::trainMeans, and GaussProcApproximation::trainStdvs.
Referenced by GaussProcApproximation::gradient(), GaussProcApproximation::pointsel_get_errors(), GaussProcApproximation::prediction_variance(), and GaussProcApproximation::value().

14.81.4 Member Data Documentation

short trendOrder [private]
The number of variables in each X variable (number of dimensions of the problem).
The order of the basis function for the mean of the GP If the order = 0, the trend is a constant, if the order = 1,
trend is linear, if order = 2, trend is quadratic.
Referenced by GaussProcApproximation::GaussProcApproximation(), GaussProcApproximation::get_beta_coefficients(), GaussProcApproximation::get_trend(), GaussProcApproximation::GPmodel_build(), and GaussProcApproximation::predict().
The documentation for this class was generated from the following files:
- GaussProcApproximation.hpp
- GaussProcApproximation.cpp

14.82 GeneralReader Class Reference

Utility used in derived read_core to read in generic format.

Public Member Functions

- template<typename ArrayType>
  void operator() (std::istream &s, size_t start_index, size_t num_items, ArrayType &array_data, StringMultiArrayView label_array)

14.82.1 Detailed Description

Utility used in derived read_core to read in generic format.
The documentation for this class was generated from the following file:
- DakotaVariables.hpp
14.83 GeneralWriter Class Reference

Utility used in derived write core to write in generic format.

Public Member Functions

- template<typename ArrayType>
  void operator() (std::ostream &s, size_t start_index, size_t num_items, const ArrayType &array_data, StringMultiArrayConstView label_array)

14.83.1 Detailed Description

Utility used in derived write core to write in generic format.

The documentation for this class was generated from the following file:
- DakotaVariables.hpp

14.84 GetLongOpt Class Reference

GetLongOpt is a general command line utility from S. Manoharan (Advanced Computer Research Institute, Lyon, France).

Inheritance diagram for GetLongOpt:

```
GetLongOpt
    CommandLineHandler
```

Public Types

- enum OptType { Valueless, OptionalValue, MandatoryValue }
  
  _enum for different types of values associated with command line options._

Public Member Functions

- GetLongOpt (const char optmark= '-')
  Constructor.
- ~GetLongOpt ()
  Destructor.
- int parse (int argc, char *const *argv)
  
  _parse the command line args (argc, argv)._  
- int parse (char *const str, char *const p)
  
  _parse a string of options (typically given from the environment)._  
- int enroll (const char *const opt, const OptType t, const char *const desc, const char *const val)
  
  _Add an option to the list of valid command options._
- const char * retrieve (const char *const opt) const
Retrieve value of option.
- void `usage (std::ostream &outfile=Cout) const`
  
  Print usage information to outfile.
- void `usage (const char ∗str)`
  
  Change header of usage output to str.
- void `store (const char ∗name, const char ∗value)`
  
  Store a specified option value.

**Private Member Functions**

- char ∗ `basename (char ∗const p) const`
  
  extract the base name from a string as delimited by ‘/’
- int `setcell (Cell ∗c, char ∗valtoken, char ∗nexttoken, const char ∗p)`
  
  internal convenience function for setting Cell::value

**Private Attributes**

- Cell ∗ `table`
  
  option table
- const char ∗ `ustring`
  
  usage message
- char ∗ `pname`
  
  program basename
- char `optmarker`
  
  option marker
- int `enroll_done`
  
  finished enrolling
- Cell ∗ `last`
  
  last entry in option table

### 14.84.1 Detailed Description

`GetLongOpt` is a general command line utility from S. Manoharan (Advanced Computer Research Institute, Lyon, France).

`GetLongOpt` manages the definition and parsing of "long options." Command line options can be abbreviated as long as there is no ambiguity. If an option requires a value, the value should be separated from the option either by whitespace or an "=".

### 14.84.2 Member Enumeration Documentation

**enum OptType**

enum for different types of values associated with command line options.

**Enumerator**

- `Valueless` option that may never have a value
- `OptionalValue` option with optional value
- `MandatoryValue` option with required value
14.84.3 Constructor & Destructor Documentation

GetLongOpt ( const char optmark = '-' )

Constructor.

Constructor for GetLongOpt takes an optional argument: the option marker. If unspecified, this defaults to '-', the standard (?) Unix option marker.


14.84.4 Member Function Documentation

int parse ( int argc, char *const *argv )

parse the command line args (argc, argv).

A return value < 1 represents a parse error. Appropriate error messages are printed when errors are seen.
parse returns the the optind (see getopt(3)) if parsing is successful.


Referenced by CommandLineHandler::check_usage().

int parse ( char *const str, char *const p )

parse a string of options (typically given from the environment).

A return value < 1 represents a parse error. Appropriate error messages are printed when errors are seen.
parse takes two strings: the first one is the string to be parsed and the second one is a string to be prefixed to the parse errors.


int enroll ( const char *const opt, const OptType t, const char *const desc, const char *const val )

Add an option to the list of valid command options.

enroll adds option specifications to its internal database. The first argument is the option sting. The second is an enum saying if the option is a flag (Valueless), if it requires a mandatory value (MandatoryValue) or if it takes an optional value (OptionalValue). The third argument is a string giving a brief description of the option. This description will be used by GetLongOpt::usage. GetLongOpt, for usage-printing, uses {${val}} to represent values needed by the options. {<${val}>} is a mandatory value and {{${val}}} is an optional value. The final argument to enroll is the default string to be returned if the option is not specified. For flags (options with Valueless), use """" (empty string, or in fact any arbitrary string) for specifying TRUE and 0 (null pointer) to specify FALSE.


Referenced by CommandLineHandler::initialize_options().

const char * retrieve ( const char *const opt ) const

Retrieve value of option.

The values of the options that are enrolled in the database can be retrieved using retrieve. This returns a string and this string should be converted to whatever type you want. See atoi, atof, atol, etc. If a "parse" is not done before retrieving all you will get are the default values you gave while enrolling! Ambiguities while retrieving (may happen when options are abbreviated) are resolved by taking the matching option that was enrolled last. For example, -{v} will expand to {{-verify}}. If you try to retrieve something you didn’t enroll, you will get a warning message.
References GetLongOpt::optmarker, and GetLongOpt::table.
Referenced by CommandLineHandler::check_usage(), ProgramOptions::manage_run_modes(), ProgramOptions-:
ProgramOptions(), and CommandLineHandler::read_restart_evals().

\textbf{void usage ( const char ∗ str ) [inline]}

Change header of usage output to str.
\texttt{GetLongOpt::usage} is overloaded. If passed a string "str", it sets the internal usage string to "str". Otherwise it simply prints the command usage.
References GetLongOpt::ustring.
The documentation for this class was generated from the following files:
• CommandLineHandler.hpp
• CommandLineHandler.cpp

\subsection{14.85 GP\_Objective Class Reference}

ROL objective function for the Gaussian Process (GP) surrogate.
Inherits Objective< double >.

\textbf{Public Member Functions}

• \textbf{GP\_Objective (GaussianProcess &gp\_model)}
  Constructor for \texttt{GP\_Objective}.
• double \textbf{value (const ROL::Vector< double > &p, double &tol)}
  Get the value of the objective function at a point.
• void \textbf{gradient (ROL::Vector< double > &g, const ROL::Vector< double > &p, double &tol)}
  Get the gradient of the objective function at a point.

\textbf{Private Member Functions}

• bool \textbf{pdiff (const std::vector< double > &pnew)}
  Compute the l2 norm of the difference between new and old parameter vectors.
• ROL::Ptr< const std::vector< double > > \textbf{getVector (const ROL::Vector< double > &vec)}
  Convert a const ROL Vector to a ROL::Ptr< const std::vector< > >.
• ROL::Ptr< std::vector< double > > \textbf{getVector (ROL::Vector< double > &vec)}
  Convert a ROL Vector to a ROL::Ptr< std::vector< > >.

\textbf{Private Attributes}

• GaussianProcess & \textbf{gp}
  Pointer to the GaussianProcess surrogate.
• int \textbf{nopt}
  Number of optimization variables.
• double \textbf{Jold}
  Previously computed value of the objective function.
• VectorXd grad_old
  
  Previously computed gradient of the objective function.
• VectorXd pold
  
  Previous value of the parameter vector.
• double difftol = 1.0e-15
  
  Tolerance for $l_2$ difference between parameter vectors. Should smaller than ROL’s stepnorm but bigger than machine epsilon.

### 14.85.1 Detailed Description

ROL objective function for the Gaussian Process (GP) surrogate.

### 14.85.2 Constructor & Destructor Documentation

**GP_Objective ( GaussianProcess & gp_model )**

Constructor for GP_Objective.

**Parameters**

| in     | gp_model | Reference to the GaussianProcess surrogate. |

References GaussianProcess::get_num_opt_variables(), GP_Objective::gp, GP_Objective::grad_old, GP_Objective::Jold, GP_Objective::nopt, and GP_Objective::pold.

### 14.85.3 Member Function Documentation

**double value ( const ROL::Vector< double > & p, double & tol )**

Get the value of the objective function at a point.

**Parameters**

| in     | p         | ROL vector of parameters. |
| in     | tol       | Tolerance for inexact evaluation (not used here). |

References GP_Objective::getVector(), GP_Objective::gp, GP_Objective::grad_old, GP_Objective::Jold, GaussianProcess::negative_marginal_log_likelihood(), GP_Objective::nopt, GP_Objective::pdiff(), GaussianProcess::set_opt_params(), and dakota::silence_unused_args().

**void gradient ( ROL::Vector< double > & g, const ROL::Vector< double > & p, double & tol )**

Get the gradient of the objective function at a point.

**Parameters**

| out    | g         | Gradient of the objective function. |
| in     | p         | ROL vector of parameters. |
| in     | tol       | Tolerance for inexact evaluation (not used here). |

References GP_Objective::getVector(), GP_Objective::gp, GP_Objective::grad_old, GP_Objective::Jold, GaussianProcess::negative_marginal_log_likelihood(), GP_Objective::nopt, GP_Objective::pdiff(), GaussianProcess::set_opt_params(), and dakota::silence_unused_args().

**bool pdiff ( const std::vector< double > & pnew ) [private]**

Compute the $l_2$ norm of the difference between new and old parameter vectors.
Parameters

<table>
<thead>
<tr>
<th></th>
<th>pnew</th>
<th>New value of the parameter vector.</th>
</tr>
</thead>
</table>

References GP::Objective::difftol, GP::Objective::nopt, and GP::Objective::pold.
Referenced by GP::Objective::gradient(), and GP::Objective::value().

ROL::Ptr<const std::vector<double> > getVector ( const ROL::Vector<double> & vec ) [inline], [private]
Convert a const ROL Vector to a ROL::Ptr<const std::vector<double> >
Parameters

<table>
<thead>
<tr>
<th></th>
<th>vec</th>
<th>const ROL vector</th>
</tr>
</thead>
</table>

Referenced by GP::Objective::getVector(), GP::Objective::gradient(), and GP::Objective::value().

ROL::Ptr<std::vector<double> > getVector ( ROL::Vector<double> & vec ) [inline], [private]
Convert a ROL Vector to a ROL::Ptr<std::vector<double> >
Parameters

<table>
<thead>
<tr>
<th></th>
<th>vec</th>
<th>ROL vector</th>
</tr>
</thead>
</table>

References GP::Objective::getVector().
The documentation for this class was generated from the following files:

- GP::Objective.hpp
- GP::Objective.cpp

## 14.86 Graphics Class Reference

The Graphics class provides a single interface to 2D (motif) and 3D (PLPLOT) graphics; there is only one instance of this OutputManager::dakotaGraphics.

### Public Member Functions

- **Graphics ()**
  
  *constructor*

- **~Graphics ()**
  
  *destructor*

- void create_plots_2d (const Variables &vars, const Response &response)
  
  creates the 2d graphics window and initializes the plots

- void add_datapoint (int graphics, const Variables &vars, const Response &response)
  
  adds data to each window in the 2d graphics based on the results of a model evaluation

- void add_datapoint (int i, double x, double y)
  
  adds data to a single window in the 2d graphics

- void new_dataset (int i)
  
  creates a separate line graphic for subsequent data points for a single window in the 2d graphics

- void close ()
CHAPTER 14. CLASS DOCUMENTATION

- close graphics windows
  - void set_x_labels2d (const char *x_label)
    
    set x label for each plot equal to x_label
  - void set_y_labels2d (const char *y_label)
    
    set y label for each plot equal to y_label
  - void set_x_label2d (int i, const char *x_label)
    
    set x label for ith plot equal to x_label
  - void set_y_label2d (int i, const char *y_label)
    
    set y label for ith plot equal to y_label

Private Attributes

- Graphics2D * graphics2D
  
  pointer to the 2D graphics object
- bool win2dOn
  
  flag to indicate if 2D graphics window is active

14.86.1 Detailed Description

The Graphics class provides a single interface to 2D (motif) and 3D (PLPLOT) graphics; there is only one instance of this OutputManager::dakotaGraphics.

14.86.2 Member Function Documentation

void create_plots_2d ( const Variables & vars, const Response & response )

creates the 2d graphics window and initializes the plots

Sets up a single event loop for duration of the dakotaGraphics object, continuously adding data to a single window. There is no reset. To start over with a new data set, you need a new object (delete old and instantiate new).

References Variables::continuous_variable_labels(), Variables::cv(), Variables::discrete_int_variable_labels(), Variables::discrete_real_variable_labels(), Variables::div(), Variables::drv(), Response::function_labels(), Graphics::graphics2D, Response::num_functions(), Dakota::re_match(), and Graphics::win2dOn.

Referenced by SurrBasedMinimizer::initialize_graphics(), NonDReliability::initialize_graphics(), and Iterator::initialize_graphics().

void add_datapoint ( int graphics_cntr, const Variables & vars, const Response & response )

adds data to each window in the 2d graphics based on the results of a model evaluation

Adds data to each 2d plot and each tabular data column (one for each active variable and for each response function). graphicsCntr is used for the x axis in the graphics and the first column in the tabular data.

References Response::active_set_request_vector(), Variables::continuous_variables(), Variables::discrete_int_variables(), Variables::discrete_real_variables(), Response::function_values(), Graphics::graphics2D, and Graphics::win2dOn.

Referenced by OutputManager::add_datapoint(), NonDLocalReliability::mean_value(), and NonDLocalReliability::update_level_data().
void add_datapoint ( int i, double x, double y )
adds data to a single window in the 2d graphics
   Adds data to a single 2d plot. Allows complete flexibility in defining other kinds of x-y plotting in the 2D graphics.
   References Graphics::graphics2D, and Graphics::win2dOn.

void new_dataset ( int i )
creates a separate line graphic for subsequent data points for a single window in the 2d graphics
   Used for displaying multiple data sets within the same plot.
   References Graphics::graphics2D, and Graphics::win2dOn.
   Referenced by NonDLocalReliability::update_level_data().
The documentation for this class was generated from the following files:
   • DakotaGraphics.hpp
   • DakotaGraphics.cpp

14.87 GridApplicInterface Class Reference

Derived application interface class which spawns simulation codes using grid services such as Condor or Globus.

Inheritance diagram for GridApplicInterface:

```
  Interface
   ApplicationInterface
      ProcessApplicInterface
         SysCallApplicInterface
            GridApplicInterface
```

Public Member Functions

- **GridApplicInterface** (const ProblemDescDB &problem_db)
  
  *constructor*
- **~GridApplicInterface** ()
  
  *destructor*
- void derived_map (const Variables &vars, const ActiveSet &set, Response &response, int fn_eval_id)
  
  Called by map() and other functions to execute the simulation in synchronous mode. The portion of performing an evaluation that is specific to a derived class.
- void derived_map_asynch (const ParamResponsePair &pair)
  
  Called by map() and other functions to execute the simulation in asynchronous mode. The portion of performing an asynchronous evaluation that is specific to a derived class.
• **void wait_local_evaluation_sequence** (PRPQueue &prp_queue)
  
  version of wait_local_evaluations() managing of set of individual asynchronous evaluations

• **void test_local_evaluation_sequence** (PRPQueue &prp_queue)
  
  Convenience function for common code between wait and nowait case.

• **int synchronous_local_analysis** (int analysis_id)

Protected Member Functions

• **bool grid_file_test** (const String &root_file)
  
  test file(s) for existence based on root_file name

Protected Attributes

• **IntSet idSet**
  
  Set of function evaluation id’s for active asynchronous system call evaluations.

• **IntShortMap failCountMap**
  
  map linking function evaluation id’s to number of response read failures

• **start_grid_computing_t start_grid_computing**
  
  handle to dynamically linked start_grid_computing function

• **perform_analysis_t perform_analysis**
  
  handle to dynamically linked perform_analysis grid function

• **get_jobs_completed_t get_jobs_completed**
  
  handle to dynamically linked get_jobs_completed grid function

• **stop_grid_computing_t stop_grid_computing**
  
  handle to dynamically linked stop_grid_computing function

14.87.1 Detailed Description

Derived application interface class which spawns simulation codes using grid services such as Condor or Globus.

This class is currently a modified copy of SysCallApplicInterface adapted for use with an external grid dervices library which was dynamically linked using dlopen() services.

14.87.2 Member Function Documentation

**int synchronous_local_analysis** ( int analysis_id ) [inline], [virtual]

This code provides the derived function used by ApplicationInterface::serve_analyses_synch(). TODO - allow local analyses?????

Reimplemented from ApplicationInterface.

References SysCallApplicInterface::spawn_analysis_to_shell().

The documentation for this class was generated from the following files:

• GridApplicInterface.hpp

• GridApplicInterface.cpp
14.88 HDF5IOHelper Class Reference

Public Member Functions

- HDF5IOHelper (const std::string &file_name, bool overwrite=false)
- template<typename T>
  void store_scalar (const std::string &dset_name, const T &val)
    
    Store scalar data to a data set.
- void store_scalar (const std::string &dset_name, const String &val)
    
    Store string scalar data to a data set.
- template<typename T>
  void store_vector (const std::string &dset_name, const std::vector<T> &array) const
    
    Store vector (1D) information to a dataset.
- template<typename T>
  void store_vector (const std::string &dset_name, const Teuchos::SerialDenseVector<int, T> &vec)
    
    Store vector (1D) information to a dataset.
- void store_vector (const std::string &dset_name, const StringMultiArrayConstView &vec)
    
    Store vector (1D) information to a dataset.
- void store_vector (const std::string &dset_name, const SizetMultiArrayConstView &vec)
    
    Store vector (1D) information to a dataset.
- template<typename T>
  void set_scalar (const String &dset_name, const T &data, const int &index)
    
    Set a scalar in a 1D dataset at index using its name.
- template<typename T>
  void set_scalar (const String &dset_name, H5::DataSet &ds, const T &data, const int &index)
    
    Set a scalar in a 1D dataset at index using the dataset object.
- void set_scalar (const String &dset_name, H5::DataSet &ds, const String &data, const int &index)
    
    Set a scalar in a 1D dataset at index using the dataset object.
- template<typename T>
  void set_vector (const String &dset_name, const T &data, const int &index, const bool &row=true)
    
    Set a row or column in a 2D dataset at index using its name.
- void set_vector (const String &dset_name, H5::DataSet &ds, const StringMultiArrayConstView &data, const int &index, const bool &row=true)
    
    Set a row or column of Strings in a 2D dataset at index using the dataset object.
• void set_vector (const String &dset_name, H5::DataSet &ds, const std::vector<String> &data, const int &index, const bool &row=true)
  
  Set a row or column of Strings in a 2D dataset at index using the dataset object.

• template<typename T>
  void set_vector (const String &dset_name, H5::DataSet &ds, const T &data, const int &index, const bool &row=true)

  Set a row or column in a 2D dataset at index using the dataset object.

• template<typename T>
  void set_matrix (const String &dset_name, const Teuchos::SerialDenseMatrix<int, T> &data, const int &index, const bool &transpose=false)

  Set a matrix in a 3D dataset at the index into the 0th dimension by name. Dims of matrix must match those of the trailing dimensions of the dataset.

• template<typename T>
  void set_matrix (const String &dset_name, H5::DataSet &ds, const Teuchos::SerialDenseMatrix<int, T> &data, const int &index, const bool &transpose=false)

  Set a matrix in a 3D dataset at the index into the 0th dimension using a dataset object. Dims of matrix must match those of the trailing dimensions of the dataset.

• template<typename T>
  void set_vector_matrix (const String &dset_name, const std::vector<Teuchos::SerialDenseMatrix<int, T> > &data, const int &index, const bool &transpose=false)

  Set a 3D slab in a 4D dataset at the index into the 0th dimension by name. The length of the vector must match the 1st dimension of the dataset, and the dimensions of the matrices must all match the 2nd and 3rd dimensions.

• template<typename T>
  void set_vector_matrix (const String &dset_name, H5::DataSet &ds, const std::vector<Teuchos::SerialDenseMatrix<int, T> > &data, const int &index, const bool &transpose=false)

  Set a 3D slab in a 4D dataset at the index into the 0th dimension using a dataset object. The length of the vector must match the 1st dimension of the dataset, and the dimensions of the matrices must all match the 2nd and 3rd dimensions.

• template<typename T>
  void set_vector_scalar_field (const String &dset_name, const T &data, const String &field_name)

  Set a scalar field on all elements of a 1D dataset of compound type using a ds name.

• template<typename T>
  void set_vector_scalar_field (const String &dset_name, H5::DataSet &ds, const std::vector<T> &data, const String &field_name)

  Set a scalar field on all elements of a 1D dataset of compound type using a ds object.

• template<typename T>
  void set_vector_vector_field (const String &dset_name, const T &data, const size_t length, const String &field_name)

  Set a vector field on all elements of a 1D dataset of compound type using a ds name.

• template<typename T>
  void set_vector_vector_field (const String &dset_name, H5::DataSet &ds, const std::vector<T> &data, const size_t length, const String &field_name)

  Set a vector field on all elements of a 1D dataset of compound type using a ds object.

• void set_vector_vector_field (const String &dset_name, H5::DataSet &ds, const std::vector<String> &data, const size_t length, const String &field_name)

  Set a vector field on all elements of a 1D dataset of compound type using a ds object.
• int `append_empty` (const String &dset_name)
  Append an empty "layer" to the 0th dimension and return its index.
• template<typename T >
  void `append_scalar` (const String &dset_name, const T &data)
  Append a scalar to a 1D dataset.
• void `append_scalar` (const String &dset_name, const String &data)
  Append a scalar to a 1D dataset.
• template<typename T >
  void `append_vector` (const String &dset_name, const T &data, const bool &row=true)
  Append a vector as a row or column to a 2D dataset.
• void `append_vector` (const String &dset_name, const std::vector<String> &data, const bool &row=true)
  Append a vector as a row or column to a 2D dataset.
• void `append_vector` (const String &dset_name, const StringMultiArrayConstView &data, const bool &row=true)
  Append a vector as a row or column to a 2D dataset.
• template<typename T >
  void `append_matrix` (const String &dset_name, const Teuchos::SerialDenseMatrix<int, T> &data, const bool &transpose=false)
  Append a SerialDenseMatrix to a 3D dataset. The dataset will be expanded along the 0th dimension. By default, the shape of the matrix, (nrows, ncols), must match the size of the 1st and 2nd dimensions of the dataset. For transpose=true, the reverse must be true.
• template<typename T >
  void `append_vector_matrix` (const String &dset_name, const std::vector<Teuchos::SerialDenseMatrix<int, T>> &data, const bool &transpose=false)
  Append a std::vector of SerialDenseMatrix's to a 4D dataset. The dataset will be expanded along the 0th dimension. By default, the size of the vector must equal the size of the 1st dimension of the dataset, and the shape of the SDMs (nrows, ncols), must match the sizes of the 2nd and 2nd dimensions of the dataset. For transpose=true, the reverse must be true of the SDMs.
• template<typename T >
  void `read_scalar` (const std::string &dset_name, T &val)
  Read scalar data from a dataset.
• void `read_scalar` (const std::string &dset_name, String &val)
  Read scalar data from a dataset.
• template<typename T >
  void `read_vector` (const std::string &dset_name, T &array) const
  Read vector (1D) information from a dataset.
• void `read_vector` (const std::string &dset_name, StringArray &array) const
  Read a vector of Strings from a dataset.
• template<typename T >
  void `read_matrix` (const std::string &dset_name, Teuchos::SerialDenseMatrix<int, T> &matrix, const bool &transpose=false) const
  Read matrix (2D) information from a dataset. Currently this involves a wasteful copy to do the transpose and is intended only for purposes of testing.
• template<typename T >
  void `get_matrix` (const std::string &dset_name, Teuchos::SerialDenseMatrix<int, T> &matrix, const int &index, const bool &transpose=false) const
Get the matrix (2D) at the index into the 0th dimension of the 3D dataset at dsetname. Currently this involves a wasteful copy to do the transpose and is intended only for purposes of testing.

- template<typename T>
  void get_vector_matrix (const std::string &dset_name, std::vector<Teuchos::SerialDenseMatrix<int, T> >&data, const int &index, const bool &transpose=false) const

Read the 3D slice at the index into the 0th dimension of the 4D dataset at ds_name. Currently this involves a wasteful copy to do the transpose and is intended only for purposes of testing.

- void report_num_open ()

  Report the number of open descriptors of each type; just for debugging.

- void create_empty_dataset (const String &dset_name, const IntArray &dims, ResultsOutputType stored_type, int chunk_size=0, const void *fill_val=NULL)

  Create an empty dataset. Setting the first element of dims to 0 makes the dataset unlimited in that dimension. DSs unlimited in other dimensions currently are unsupported.

- void create_empty_dataset (const String &dset_name, const IntArray &dims, const std::vector<VariableParametersField>&fields)

  Create a dataset with compound type.

- void attach_scale (const String &dset_name, const String &scale_name, const String &label, const int &dim) const

  Attach a dimension scale to a dataset

- template<typename T>
  void add_attribute (const String &location, const String &label, const T &value)

  Add an attribute to a group or dataset.

- void add_attribute (const String &location, const String &label, const String &value)

  Add an attribute to a group or dataset.

- bool exists (const String location_name) const

  Does a group or dataset exist?

- bool is_scale (const H5::DataSet dset) const

  Is the dataset a dimensions scale?

- H5::Group create_groups (const std::string &name, bool includes_dset=true) const

  Create a group hierarchy (final token optionally a dataset name)

- H5::DataSet create_dataset (const H5::H5Location &loc, const std::string &name, const H5::DataType &type, const H5::DataSpace &space, const H5::DSetCreatPropList &create_plist=H5::DSetCreatPropList(), const H5::DSetAccPropList &access_plist=H5::DSetAccPropList()) const

  Create a dataset with a custom CreatPropList.

- H5::Group create_group (const H5::H5Location &loc, const std::string &name) const

  Create a group.

  void create_softlink (const String &link_location, const String &source_location)

  Create a soft link.

  void flush () const

  Flush cache to file.
Public Attributes

- H5::LinkCreatPropList linkCreatePL
  Global link creation property list.
- H5::DSetCreatPropList datasetCompactPL
  Global Dataset creation property list for compact datasets.
- H5::DSetCreatPropList datasetContiguousPL
  Global Dataset creation property list for contiguous datasets.

Protected Member Functions

- template<typename T>
  H5::Attribute create_attribute (const String &location, const String &label, const T &data)
  create an attribute at the location and return it
- template<typename T>
  void store_vector (const String &dset_name, const T *data, const int &len) const
  Store vector data using a pointer to the first element and length.
- void store_vector (const String &dset_name, const String *data, const int &len) const
  Store vector of Strings using a pointer to the first element and length.

Protected Attributes

- std::string fileName
  Name of the HDF5 file.
- H5::H5File h5File
  HDF5 file object.
- std::map<String, H5::DataSet> datasetCache
  Cache open datasets that have unlimited dimension This is an optimization to prevent eval-related datasets being repeatedly flushed and reopened, which is very costly.

14.88.1 Detailed Description

This helper class provides wrapper functions that perform low-level access operations in HDF5 databases.
Authors: J. Adam Stephens, Russell Hooper, Elliott Ridgway

14.88.2 Member Function Documentation

void set_scalar ( const String &dset_name, H5::DataSet &ds, const String &data, const int &index )

Set a scalar in a 1D dataset at index using the dataset object.
Set a scalar in a 1D dataset at index using an object.

void set_vector_scalar_field ( const String &dset_name, H5::DataSet &ds, const std::vector<T> &data, const String &field_name )

Set a scalar field on all elements of a 1D dataset of compound type using a ds object.
Set a field on all elements of a 1D dataset of compound type using a ds object.
References Dakota::h5_mem_dtype().
void set_vector_vector_field ( const std::string & dset_name, H5::DataSet & ds, const std::vector<T> & data, const size_t length, const std::string & field_name )

Set a vector field on all elements of a 1D dataset of compound type using a ds object.
Set a field on all elements of a 1D dataset of compound type using a ds object.
References Dakota::h5_mem_dtype().

void set_vector_vector_field ( const std::string & dset_name, H5::DataSet & ds, const std::vector<String> & data, const size_t length, const std::string & field_name )

Set a vector field on all elements of a 1D dataset of compound type using a ds object.
Set a field on all elements of a 1D dataset of compound type using a ds object.
References Dakota::pointers_to_strings().

void read_vector ( const std::string & dset_name, StringArray & array ) const

Read a vector of Strings from a dataset.
Read vector (1D) String information from a dataset.
References Dakota::abort_handler(), and Dakota::h5_mem_dtype().
The documentation for this class was generated from the following files:

- HDF5_IO.hpp
- HDF5_IO.cpp

14.89 HierarchSurrBasedLocalTraits Class Reference

Class for multilevel-multifidelity optimization algorithm.
Inheritance diagram for HierarchSurrBasedLocalTraits:

```
TraitsBase

HierarchSurrBasedLocalTraits
```

Public Member Functions

- **HierarchSurrBasedLocalTraits ()**
  *default constructor*
- **virtual ~HierarchSurrBasedLocalTraits ()**
  *destructor*
- **virtual bool is_derived ()**
  *A temporary query used in the refactoring.*
- **bool supports_continuous_variables ()**
  *Return the flag indicating whether method supports continuous variables.*
- **bool supports_linear_inequality ()**
  *Return the flag indicating whether method supports linear equalities.*
- **bool supports_linear_inequality ()**
14.90. HIERARCHSURRMODEL CLASS REFERENCE

Return the flag indicating whether method supports linear inequalities.

- `bool supports_nonlinear_equality()`

Return the flag indicating whether method supports nonlinear equalities.

- `bool supports_nonlinear_inequality()`

Return the flag indicating whether method supports nonlinear inequalities.

14.89.1 Detailed Description

Class for multilevel-multifidelity optimization algorithm.

This minimizer uses SurrogateModel(s) to perform minimization leveraging multiple model forms and discretization levels. A version of TraitsBase specialized for multilevel-multifidelity minimizer.

The documentation for this class was generated from the following file:

- HierarchSurrBasedLocalMinimizer.hpp

14.90 HierarchSurrModel Class Reference

Derived model class within the surrogate model branch for managing hierarchical surrogates (models of varying fidelity).

Inheritance diagram for HierarchSurrModel:

```
Model
|-- SurrogateModel
|   |-- HierarchSurrModel
```

Public Member Functions

- `HierarchSurrModel(ProblemDescDB &problem_db)`
  constructor

- `~HierarchSurrModel()`
  destructor

- `const unsigned short correction_mode() const`
  return correctionMode

- `void correction_mode(unsigned short corr_mode)`
  set correctionMode

Protected Member Functions

- `size_t qoi() const`
  return number of unique response functions (managing any aggregations)

- `DiscrepancyCorrection & discrepancy_correction()`
  return the DiscrepancyCorrection object used by SurrogateModels
• short correction_type ()
  
  return the correction type from the DiscrepancyCorrection object used by SurrogateModels
• void correction_type (short corr_type)
  
  set the correction type from the DiscrepancyCorrection object used by SurrogateModels
• bool initialize_mapping (ParLevLIter plIter)
• bool finalize_mapping ()
• void nested_variable_mappings (const SizetArray &c_index1, const SizetArray &di_index1, const SizetArray &ds_index1, const SizetArray &dr_index1, const ShortArray &c_target2, const ShortArray &di_target2, const ShortArray &ds_target2, const ShortArray &dr_target2)
  
  set primaryA{C,DI,DS,DR}VarMapIndices, secondaryA{C,DI,DS,DR}VarMapTargets (coming from a higher-level NestedModel context to inform derivative est.)
• const SizetArray & nested_acv1_indices () const
  
  return primaryACVarMapIndices
• const ShortArray & nested_acv2_targets () const
  
  return secondaryACVarMapTargets
• short query_distribution_parameter_derivatives () const
  
  calculate and return derivative composition of final results w.r.t. distribution parameters (none, all, or mixed)
• void check_submodel_compatibility (const Model &subModel)
  
  verify compatibility between SurrogateModel attributes and attributes of the submodel (DataFitSurrModel::actualModel or HierarchSurrModel::highFidelityModel)
• void derived_evaluate (const ActiveSet &set)
• void derived_evaluate_nowait (const ActiveSet &set)
• const IntResponseMap & derived_synchronize ()
• const IntResponseMap & derived_synchronize_nowait ()
• Model & surrogate_model ()
  
  return the active low fidelity model
• const Model & surrogate_model () const
  
  return the active low fidelity model
• Model & truth_model ()
  
  return the active high fidelity model
• const Model & truth_model () const
  
  return the active high fidelity model
• void active_model_key (const UShortArray &key)
  
  define the active model key and associated {truth,surr}ModelKey pairing
• void derived_subordinate_models (ModelList &ml, bool recurse_flag)
  
  return orderedModels and, optionally, their sub-model recursions
• void resize_from_subordinate_model (size_t depth=std::numeric_limits<size_t>::max())
  
  resize currentResponse if needed when one of the subordinate models has been resized
• void update_from_subordinate_model (size_t depth=std::numeric_limits<size_t>::max())
  
  update currentVariables using non-active data from the passed model (one of the ordered models)
• void primary_response_fn_weights (const RealVector &wts, bool recurse_flag=true)
  
  set the relative weightings for multiple objective functions or least squares terms and optionally recurses into LF/HF models
• void surrogate_response_mode (short mode)
set responseMode and pass any bypass request on to the high fidelity model for any lower-level surrogate recursions

- void surrogate_function_indices (const IntSet &surr_fn_indices)
  (re)set the surrogate index set in SurrogateModel::surrogateFnIndices

- void build_approximation ()
  use the high fidelity model to compute the truth values needed for correction of the low fidelity model results

- void component_parallel_mode (short mode)
  update component parallel mode for supporting parallelism in the low ad high fidelity models

- IntIntPair estimate_partition_bounds (int max_eval_concurrency)
  estimate the minimum and maximum partition sizes that can be utilized by this Model

- void derived_init_communicators (ParLevLIter pl_iter, int max_eval_concurrency, bool recurse_flag=true)
  set up parallel operations for the array of ordered model fidelities

- void derived_init_serial ()
  set up serial operations for the array of ordered model fidelities

- void derived_set_communicators (ParLevLIter pl_iter, int max_eval_concurrency, bool recurse_flag=true)
  set active parallel configuration within the current low and high fidelity models identified by (low,high) FidelityKey

- void derived_free_communicators (ParLevLIter pl_iter, int max_eval_concurrency, bool recurse_flag=true)
  deallocate communicator partitions for the HierarchSurrModel (request forwarded to the the array of ordered model fidelities)

- void serve_run (ParLevLIter pl_iter, int max_eval_concurrency)
  Service the low and high fidelity model job requests received from the master; completes when termination message received from stop_servers().

- void stop_servers ()
  Executed by the master to terminate the low and high fidelity model server operations when iteration on the HierarchSurrModel is complete.

- void inactive_view (short view, bool recurse_flag=true)
  update the Model’s inactive view based on higher level (nested) context and optionally recurse into

- bool evaluation_cache (bool recurse_flag=true) const
  if recurse_flag, return true if orderedModels evaluation cache usage

- bool restart_file (bool recurse_flag=true) const
  if recurse_flag, return true if orderedModels restart file usage

- void set_evaluation_reference ()
  set the evaluation counter reference points for the HierarchSurrModel (request forwarded to the low and high fidelity models)

- void fine_grained_evaluation_counters ()
  request fine-grained evaluation reporting within the low and high fidelity models

- void print_evaluation_summary (std::ostream &s, bool minimal_header=false, bool relative_count=true) const
  print the evaluation summary for the HierarchSurrModel (request forwarded to the low and high fidelity models)

- void warm_start_flag (const bool flag)
  set the warm start flag, including the orderedModels
Private Member Functions

- void **key_updates** (unsigned short model_index, unsigned short soln_lev_index)
  
  utility for propagating new key values

- void **check_model_interface_instance** ()
  
  update sameInterfaceInstance based on interface ids for models identified by current {low,high}FidelityKey

- void **init_model** (Model &model)
  
  update the passed model (one of the ordered models) with data that could change once per set of evaluations
  (e.g., an outer iterator execution), including active variable labels, inactive variable values/bounds/labels, and
  linear/nonlinear constraint coeff/bounds

- void **update_model** (Model &model)
  
  update the passed model (one of the ordered models) with data that could change per function evaluation (active
  variable values/bounds)

- void **update_from_model** (Model &model)
  
  update currentVariables using non-active data from the passed model (one of the ordered models)

- void **derived_synchronize_segmental** (IntResponseMap &hf_res_map_rekey, IntResponseMap &lf_res_map_rekey, bool block)
  
  called from derived_synchronize() and derived_synchronize_nowait() to extract and rekey response maps using
  blocking or nonblocking synchronization on the LF and HF models

- void **derived_synchronize_segmental** ()
  
  called from derived_synchronize() for case of distinct models/interfaces with competing LF/HF job queues

- void **derived_synchronize_combine** (const IntResponseMap &hf_res_map, IntResponseMap &lf_res_map, IntResponseMap &combined_res_map)
  
  combine the HF and LF response maps into a combined response map

- void **derived_synchronize_combine_nowait** (const IntResponseMap &hf_res_map, IntResponseMap &lf_res_map, IntResponseMap &combined_res_map)
  
  combine the available components from HF and LF response maps into a combined response map

- void **resize_response** (bool use_virtual_counts=true)
  
  resize currentResponse based on responseMode

- void **compute_apply_delta** (IntResponseMap &lf_res_map)
  
  helper function used in the AUTO_CORRECTED_SURROGATE responseMode for computing a correction and
  applying it to lf_res_map

- void **single_apply** (const Variables &vars, Response &resp, const UShortArray &paired_key)
  
  helper function for applying a single response correction corresponding to deltaCorr[paired_key]

- void **recursive_apply** (const Variables &vars, Response &resp)
  
  helper function for applying a correction across a sequence of model forms or discretization levels

- void **stop_model** (size_t ordered_model_index)
  
  stop the servers for the orderedModels instance identified by the passed index

Private Attributes

- std::map< UShortArray, DiscrepancyCorrection > deltaCorr
  
  manages construction and application of correction functions that are applied to a surrogate model (DataFitSurr
  or HierarchSurr) in order to reproduce high fidelity data.

- short corrOrder
order of correction: 0, 1, or 2
- unsigned short correctionMode
- ModelArray orderedModels
  Ordered sequence (low to high) of model fidelities. Models are of arbitrary type and supports recursions.
- bool sameModelInstance
  flag indicating that the \{low,high\} FidelityKey correspond to the same model instance, requiring modifications to updating and evaluation scheduling processes
- bool sameInterfaceInstance
  flag indicating that the models identified by \{low,high\} FidelityKey employ the same interface instance, requiring modifications to evaluation scheduling processes
- UShortArray componentParallelKey
  store \{LF, HF\} model key that is active in component_parallel_mode()
- std::map<UShortArray, Response> truthResponseRef
  map of reference truth (high fidelity) responses computed in build_approximation() and used for calculating corrections
- IntResponseMap cachedTruthRespMap
  map of truth (high-fidelity) responses retrieved in derived_synchronize_nowait() that could not be returned since corresponding low-fidelity response portions were still pending

Additional Inherited Members

14.90.1 Detailed Description

Derived model class within the surrogate model branch for managing hierarchical surrogates (models of varying fidelity).

The HierarchSurrModel class manages hierarchical models of varying fidelity. The class contains an ordered array of model forms (fidelity ordered from low to high), where each model form may also contain a set of solution levels (space/time discretization, convergence tolerances, etc.). At run time, one of these combinations is activated as the low fidelity model and used to perform approximate function evaluations, while another of these combinations is activated as the high fidelity model and used to provide truth evaluations for computing corrections to the low fidelity results.

14.90.2 Member Function Documentation

bool initialize_mapping ( ParLevLIter plIter ) [protected], [virtual]

Inactive variables must be propagated when a HierarchSurrModel is employed by a sub-iterator (e.g., OUU with MLMC or MLPCE). In current use cases, this can occur once per sub-iterator execution within Model::initialize_mapping().

Reimplemented from Model.
References HierarchSurrModel::init_model(), Model::initialize_mapping(), and HierarchSurrModel::orderedModels.

bool finalize_mapping ( ) [protected], [virtual]

Inactive variables must be propagated when a HierarchSurrModel is employed by a sub-iterator (e.g., OUU with MLMC or MLPCE). In current use cases, this can occur once per sub-iterator execution within Model::initialize_mapping().

Reimplemented from Model.
References Model::finalize_mapping(), and HierarchSurrModel::orderedModels.
void derived_evaluate ( const ActiveSet & set ) [protected], [virtual]

Compute the response synchronously using LF model, HF model, or both (mixed case). For the LF model portion, compute the high fidelity response if needed with build_approximation(), and, if correction is active, correct the low fidelity results.

Reimplemented from Model.

References Response::active_set(), SurrogateModel::activeKey, SurrogateModel::aggregate_response(), SurrogateModel::approxBuilds, SurrogateModel::asv_split(), HierarchSurrModel::build_approximation(), HierarchSurrModel::component_parallel_mode(), Response::copy(), Model::current_response(), Model::currentResponse, Model::current_variables, HierarchSurrModel::deltaCorr, ActiveSet::derivative_vector(), Dakota::dummy_model, Model::eval_tag_prefix(), Model::evalTagPrefix, Model::evaluate(), SurrogateModel::force_build(), Model::hierarchicalTagging, Model::outputLevel, HierarchSurrModel::recursive_apply(), ActiveSet::request_vector(), SurrogateModel::response_combine(), SurrogateModel::responseMode, HierarchSurrModel::sameModelInstance, Model::solution_level_index(), SurrogateModel::surrModelEvalCntr, SurrogateModel::surrogate_level_index(), HierarchSurrModel::surrogate_model(), SurrogateModel::truth_level_index(), HierarchSurrModel::truth_model(), Response::update(), and HierarchSurrModel::update_model().

void derived_evaluate_nowait ( const ActiveSet & set ) [protected], [virtual]

Compute the response asynchronously using LF model, HF model, or both (mixed case). For the LF model portion, compute the high fidelity response with build_approximation() (for correcting the low fidelity results in derived_synchronize() and derived_synchronize_nowait()) if not performed previously.

Reimplemented from Model.

References SurrogateModel::approxBuilds, SurrogateModel::asv_split(), Model::async_flag(), HierarchSurrModel::build_approximation(), SurrogateModel::cachedApproxRespMap, HierarchSurrModel::cachedTruthRespMap, HierarchSurrModel::component_parallel_mode(), Response::copy(), Variables::copy(), Model::current_response(), Model::currentVariables, ActiveSet::derivative_vector(), Dakota::dummy_model, Model::eval_tag_prefix(), Model::evalTagPrefix, Model::evaluate(), Model::evaluate_nowait(), Model::evaluation_id(), SurrogateModel::force_build(), Model::hierarchicalTagging, SurrogateModel::rawVarsMap, HierarchSurrModel::recursive_apply(), ActiveSet::request_vector(), SurrogateModel::responseMode, HierarchSurrModel::sameModelInstance, Model::solution_level_index(), SurrogateModel::surrIdMap, SurrogateModel::surrModelEvalCntr, SurrogateModel::surrogate_level_index(), HierarchSurrModel::surrogate_model(), SurrogateModel::truth_level_index(), HierarchSurrModel::truth_model(), SurrogateModel::truthIdMap, and HierarchSurrModel::update_model().

const IntResponseMap & derived_synchronize ( ) [protected], [virtual]

Blocking retrieval of asynchronous evaluations from LF model, HF model, or both (mixed case). For the LF model portion, apply correction (if active) to each response in the array. derived_synchronize() is designed for the general case where derived_evaluate_nowait() may be inconsistent in its use of low fidelity evaluations, high fidelity evaluations, or both.

Reimplemented from Model.

References HierarchSurrModel::derived_synchronize_combine(), HierarchSurrModel::derived_synchronize_competing(), HierarchSurrModel::derived_synchronize_sequential(), HierarchSurrModel::sameInterfaceInstance, HierarchSurrModel::sameModelInstance, SurrogateModel::surrIdMap, SurrogateModel::surrResponseMap, and SurrogateModel::truthIdMap.

const IntResponseMap & derived_synchronize_nowait ( ) [protected], [virtual]

Nonblocking retrieval of asynchronous evaluations from LF model, HF model, or both (mixed case). For the LF model portion, apply correction (if active) to each response in the map. derived_synchronize_nowait() is designed
for the general case where `derived_evaluate_nowait()` may be inconsistent in its use of actual evals, approx evals, or both.

Reimplemented from `Model`.
References `HierarchSurrModel::derived_synchronize_combine_nowait()`, `HierarchSurrModel::derived_synchronize_sequential()`, and `SurrogateModel::surrResponseMap`.
Referenced by `HierarchSurrModel::derived_synchronize_competing()`.

The documentation for this class was generated from the following files:

- `HierarchSurrModel.hpp`
- `HierarchSurrModel.cpp`

## 14.91 IntegerScale Struct Reference

Data structure for storing int-valued dimension scale.

### Public Member Functions

- `IntegerScale (const std::string &label, const IntVector &items, ScaleScope scope=ScaleScope::UNSHARED)`
  
  Constructor that takes an `IntVector`.

- `IntegerScale (const std::string &label, const IntArray &items, ScaleScope scope=ScaleScope::UNSHARED)`
  
  Constructor that takes an `IntArray`.

- `IntegerScale (const std::string &label, const int *items, const int len, ScaleScope scope=ScaleScope::UNSHARED)`
  
  Constructor that takes a pointer to int and length.

- `IntegerScale (const std::string &label, std::initializer_list<int> items, ScaleScope in_scope=ScaleScope::UNSHARED)`
  
  Constructor that takes an `initializer_list`.

### Public Attributes

- `std::string label`
- `ScaleScope scope`
- `IntVector items`
- `int numCols`
  
  Number of columns; equals length of scale when 1D.

- `bool isMatrix`
  
  2D or 1D?

### 14.91.1 Detailed Description

Data structure for storing int-valued dimension scale.

The documentation for this struct was generated from the following file:

- `dakota_results_types.hpp`
14.92 Interface Class Reference

Base class for the interface class hierarchy.

Inheritance diagram for Interface:

```
  Interface
     |     |
  ApplicationInterface ApproximationInterface
     |     |
  DirectApplicInterface  ProcessApplicInterface
     |     |
  MatlabInterface        ProcessHandleApplicInterface
  PythonInterface        SysCallApplicInterface
  ScilabInterface
  TestDriverInterface
  ParallelDirectApplicInterface
  SerialDirectApplicInterface
  SoleilDirectApplicInterface
```

Public Member Functions

- **Interface ()**
  default constructor
- **Interface (ProblemDescDB &problem_db)**
  standard constructor for envelope
- **Interface (const Interface &interface_in)**
  copy constructor
- **virtual ~Interface ()**
  destructor
- **Interface operator= (const Interface &interface_in)**
  assignment operator
- **virtual void map (const Variables &vars, const ActiveSet &set, Response &response, bool asynch_flag=false)**
  the function evaluator: provides a “mapping” from the variables to the responses.
- **virtual const IntResponseMap & synchronize ()**
  recovers data from a series of asynchronous evaluations (blocking)
- **virtual const IntResponseMap & synchronize_nowait ()**
  recovers data from a series of asynchronous evaluations (nonblocking)
- **virtual void serve_evaluations ()**
  evaluation server function for multiprocessor executions
- **virtual void stop_evaluation_servers ()**
  send messages from iterator rank 0 to terminate evaluation servers
virtual void init_communicators (const IntArray &message_lengths, int max_eval_concurrency)
    allocate communicator partitions for concurrent evaluations within an iterator and concurrent multiprocessor analyses within an evaluation.

virtual void set_communicators (const IntArray &message_lengths, int max_eval_concurrency)
    set the local parallel partition data for an interface (the partitions are already allocated in ParallelLibrary).

virtual void init_serial ()
    reset certain defaults for serial interface objects.

virtual int asynch_local_evaluation_concurrency () const
    return the user-specified concurrency for asynch local evaluations

virtual short interface_synchronization () const
    return the user-specified interface synchronization

virtual int minimum_points (bool constraint_flag) const
    returns the minimum number of points required to build a particular ApproximationInterface (used by DataFitSurfModels).

virtual int recommended_points (bool constraint_flag) const
    returns the recommended number of points required to build a particular ApproximationInterface (used by DataFitSurfModels).

virtual void active_model_key (const UShortArray &mi_key)
    activate an approximation state based on its multi-index key

virtual void clear_model_keys ()
    reset initial state by removing all model keys for an approximation

virtual void approximation_function_indices (const IntSet &approx_fn_indices)
    set the (currently active) approximation function index set

virtual void update_approximation (const Variables &vars, const IntResponsePair &response_pr)
    updates the anchor point for an approximation

virtual void update_approximation (const RealMatrix &samples, const IntResponseMap &resp_map)
    updates the current data points for an approximation

virtual void update_approximation (const VariablesArray &vars_array, const IntResponseMap &resp_map)
    updates the current data points for an approximation

virtual void append_approximation (const Variables &vars, const IntResponsePair &response_pr)
    appends a single point to an existing approximation

virtual void append_approximation (const RealMatrix &samples, const IntResponseMap &resp_map)
    appends multiple points to an existing approximation

virtual void append_approximation (const VariablesArray &vars_array, const IntResponseMap &resp_map)
    appends multiple points to an existing approximation

virtual void build_approximation (const RealVector &c_lbnds, const RealVector &c_ubnds, const IntVector &di_lbnds, const IntVector &di_ubnds, const RealVector &dr_lbnds, const RealVector &dr_ubnds)
    builds the approximation

virtual void export_approximation ()
    export the approximation to disk

virtual void rebuild_approximation (const BitArray &rebuild_fns)
    rebuilds the approximation after a data update

virtual void pop_approximation (bool save_data)
removes data from last append from the approximation

- virtual void push_approximation ()
  retrieves approximation data from a previous state (negates pop)

- virtual bool push_available ()
  queries the approximation for the ability to retrieve a previous increment

- virtual void finalize_approximation ()
  finalizes the approximation by applying all trial increments

- virtual void combine_approximation ()
  combine the current approximation with previously stored data sets

- virtual void combined_to_active (bool clear_combined=true)
  promote the combined approximation to the currently active one

- virtual void clear_inactive ()
  clear inactive approximation data

- virtual bool formulation_updated () const
  query for change in approximation formulation

- virtual Real2DArray cv_diagnostics (const StringArray &metric_types, unsigned num_folds)
  approximation cross-validation quality metrics per response function

- virtual RealArray challenge_diagnostics (const String &metric_type, const RealMatrix &challenge_pts)
  approximation challenge data metrics per response function

- virtual void clear_current_active_data ()
  clears current data from an approximation interface

- virtual void clear_active_data ()
  clears all data from an approximation interface

- virtual SharedApproxData & shared_approximation ()
  retrieve the SharedApproxData within an ApproximationInterface

- virtual std::vector<Approximation> & approximations ()
  retrieve the Approximations within an ApproximationInterface

- virtual const Pecos::SurrogateData & approximation_data (size_t fn_index)
  retrieve the approximation data from a particular Approximation within an ApproximationInterface

- virtual const RealVectorArray & approximation_coefficients (bool normalized=false)
  retrieve the approximation coefficients from each Approximation within an ApproximationInterface

- virtual void approximation_coefficients (const RealVectorArray &approx(coeffs, bool normalized=false)
  set the approximation coefficients within each Approximation within an ApproximationInterface

- virtual const RealVector & approximation_variances (const Variables &vars)
  retrieve the approximation variances from each Approximation within an ApproximationInterface

- virtual const StringArray & analysis_drivers () const
  retrieve the analysis drivers specification for application interfaces

- virtual const String2DArray & analysis_components () const
  retrieve the analysis components, if available

- virtual bool evaluation_cache () const
  return flag indicating usage of the global evaluation cache
• virtual bool restart_file() const
  return flag indicating usage of the restart file
• virtual void file_cleanup() const
  clean up any interface parameter/response files when aborting
• void cache_unmatched_response(int raw_id)
  migrate an unmatched response record from rawResponseMap to cachedResponseMap
• void assign_rep(Interface *interface_rep, bool ref_count_incr=true)
  assign letter or replace existing letter with a new one
• unsigned short interface_type() const
  returns the interface type
• const String & interface_id() const
  returns the interface identifier
• int evaluation_id() const
  returns the value of the (total) evaluation id counter for the interface
• void fine_grained_evaluation_counters(size_t num_fns)
  set fineGrainEvalCounters to true and initialize counters if needed
• void init_evaluation_counters(size_t num_fns)
  initialize fine grained evaluation counters, sizing if needed
• void set_evaluation_reference()
  set evaluation count reference points for the interface
• void print_evaluation_summary(std::ostream &s, bool minimal_header, bool relative_count) const
  print an evaluation summary for the interface
• bool multi_proc_eval() const
  returns a flag signaling the use of multiprocessor evaluation partitions
• bool iterator_eval_dedicated_master() const
  returns a flag signaling the use of a dedicated master processor at the iterator-evaluation scheduling level
• bool is_null() const
  function to check interfaceRep (does this envelope contain a letter?)
• void eval_tag_prefix(const String &eval_id_str, bool append_iface_id=true)
  set the evaluation tag prefix (does not recurse)

Protected Member Functions

• Interface (BaseConstructor, const ProblemDescDB &problem_db)
  constructor initializes the base class part of letter classes (BaseConstructor overloading avoids infinite recursion in the derived class constructors - Coplien, p. 139)
• Interface (NoDBBaseConstructor, size_t num_fns, short output_level)
  constructor initializes the base class part of letter classes (NoDBBaseConstructor used for on the fly instantiations without a DB)
• void init_algebraic_mappings(const Variables &vars, const Response &response)
  Define algebraicACVIndices, algebraicACVIds, and algebraicFnIndices.
• void asv_mapping(const ActiveSet &total_set, ActiveSet &algebraic_set, ActiveSet &core_set)
  define the evaluation requirements for algebraic_mappingst() (algebraic_set) and the core Application/Approximation mapping (core_set) from the total Interface evaluation requirements (total_set)
• void asv_mapping (const ActiveSet &algebraic_set, ActiveSet &total_set)
  map an algebraic ASV back to original total ordering for async recovery

• void algebraic_mappings (const Variables &vars, const ActiveSet &algebraic_set, Response &algebraic_response)
  evaluate the algebraic_response using the AMPL solver library and the data extracted from the algebraic_mappings file

• void response_mapping (const Response &algebraic_response, const Response &core_response, Response &total_response)
  combine the response from algebraic_mappings() with the response from derived_map() to create the total response

• virtual String final_eval_id_tag (int fn_eval_id)
  form and return the final evaluation ID tag, appending iface ID if needed

Protected Attributes

• unsigned short interface_type
  the interface type: enum for system, fork, direct, grid, or approximation

• String interfaceId
  the interface specification identifier string from the DAKOTA input file

• bool algebraicMappings
  flag for the presence of algebraic_mappings that define the subset of an Interface’s parameter to response mapping that is explicit and algebraic.

• bool coreMappings
  flag for the presence of non-algebraic mappings that define the core of an Interface’s parameter to response mapping (using analysis_drivers for ApplicationInterface or functionSurfaces for ApproximationInterface).

• short outputLevel
  output verbosity level: {SILENT, QUIET, NORMAL, VERBOSE, DEBUG, OUTPUT}

• int currEvalId
  identifier for the current evaluation, which may differ from the evaluation counters in the case of evaluation scheduling; used on iterator master as well as server processors. Currently, this is set prior to all invocations of derived_map() for all processors.

• bool fineGrainEvalCounters
  controls use of fn val/grad/hess counters for detailed evaluation report

• int evalIdCntr
  total interface evaluation counter

• int newEvalIdCntr
  new (non-duplicate) interface evaluation counter

• int evalIdRefPt
  iteration reference point for evalIdCntr

• int newEvalIdRefPt
  iteration reference point for newEvalIdCntr

• IntArray fnValCounter
  number of value evaluations by resp fn

• IntArray fnGradCounter
  number of gradient evaluations by resp fn

• IntArray fnHessCounter
14.92. INTERFACE CLASS REFERENCE

number of Hessian evaluations by resp fn

- **IntArray newFnValCounter**
  number of new value evaluations by resp fn
- **IntArray newFnGradCounter**
  number of new gradient evaluations by resp fn
- **IntArray newFnHessCounter**
  number of new Hessian evaluations by resp fn
- **IntArray fnValRefPt**
  iteration reference point for fnValCounter
- **IntArray fnGradRefPt**
  iteration reference point for fnGradCounter
- **IntArray fnHessRefPt**
  iteration reference point for fnHessCounter
- **IntArray newFnValRefPt**
  iteration reference point for newFnValCounter
- **IntArray newFnGradRefPt**
  iteration reference point for newFnGradCounter
- **IntResponseMap rawResponseMap**
  Set of responses returned by either a blocking or nonblocking schedule.
- **IntResponseMap cachedResponseMap**
  Set of available asynchronous responses completed within a blocking or nonblocking scheduler that cannot be processed in a higher level context and need to be stored for later.
- **StringArray fnLabels**
  response function descriptors (used in print_evaluation_summary() and derived direct interface classes); initialized in map() functions due to potential updates after construction
- **bool multiProcEvalFlag**
  flag for multiprocessor evaluation partitions (evalComm)
- **bool ieDedMasterFlag**
  flag for dedicated master partitioning at the iterator level
- **String evalTagPrefix**
  set of period-delimited evaluation ID tags to use in evaluation tagging
- **bool append_ifaceId**
  whether to append the interface ID to the prefix during map (default true)
- **String2DArray analysisComponents**
  Analysis components for interface types that support them.

Private Member Functions

- **Interface * get_interface (ProblemDescDB &problem_db)**
  Used by the envelope to instantiate the correct letter class.
- **int algebraic_function_type (String)**
  Used by algebraic mappings to determine the correct AMPL function evaluation call to make.
CHAPTER 14. CLASS DOCUMENTATION

Static Private Member Functions

- static String user_auto_id ()
  return the next available interface ID for no-ID user methods
- static String no_spec_id ()
  return the next available interface ID for on-the-fly methods

Private Attributes

- StringArray algebraicVarTags
  set of variable tags from AMPL stub.col
- SizetArray algebraicACVIndices
  set of indices mapping AMPL algebraic variables to DAKOTA all continuous variables
- SizetArray algebraicACVIDs
  set of ids mapping AMPL algebraic variables to DAKOTA all continuous variables
- StringArray algebraicFnTags
  set of function tags from AMPL stub.row
- IntArray algebraicFnTypes
  function type: > 0 = objective, < 0 = constraint |value| - 1 is the objective (constraint) index when making AMPL objval (conival) calls
- SizetArray algebraicFnIndices
  set of indices mapping AMPL algebraic objective functions to DAKOTA response functions
- RealArray algebraicConstraintWeights
  set of weights for computing Hessian matrices for algebraic constraints;
- int numAlgebraicResponses
  number of algebraic responses (objectives+constraints)
- Interface * interfaceRep
  pointer to the letter (initialized only for the envelope)
- int referenceCount
  number of objects sharing interfaceRep
- ASL * asl
  pointer to an AMPL solver library (ASL) object

Static Private Attributes

- static size_t noSpecIdNum = 0
  the last used interface ID number for on-the-fly instantiations (increment before each use)

14.92.1 Detailed Description

Base class for the interface class hierarchy.

The Interface class hierarchy provides the part of a Model that is responsible for mapping a set of Variables into a set of Responses. The mapping is performed using either a simulation-based application interface or a surrogate-based approximation interface. For memory efficiency and enhanced polymorphism, the interface hierarchy employs the "letter/envelope idiom" (see Coplien "Advanced C++", p. 133), for which the base class (Interface) serves as the envelope and one of the derived classes (selected in Interface::get_interface()) serves as the letter.
14.92. INTERFACE CLASS REFERENCE

14.92.2 Constructor & Destructor Documentation

**Interface ( )**

default constructor
    used in Model envelope class instantiations

**Interface ( ProblemDescDB & problem_db )**

standard constructor for envelope
    Used in Model instantiation to build the envelope. This constructor only needs to extract enough data to properly execute get_interface, since Interface::Interface(BaseConstructor, problem_db) builds the actual base class data inherited by the derived interfaces.
    References Dakota::abort_handler(), Interface::get_interface(), and Interface::interfaceRep.

**Interface ( const Interface & interface_in )**

copy constructor
    Copy constructor manages sharing of interfaceRep and incrementing of referenceCount.
    References Interface::interfaceRep, and Interface::referenceCount.

**∼Interface ( ) [virtual]**

destructor
    Destructor decrements referenceCount and only deletes interfaceRep if referenceCount is zero.
    References Interface::interfaceRep, and Interface::referenceCount.

**Interface ( BaseConstructor, const ProblemDescDB & problem_db ) [protected]**

constructor initializes the base class part of letter classes (BaseConstructor overloading avoids infinite recursion in the derived class constructors - Coplien, p. 139)
    This constructor is the one which must build the base class data for all inherited interfaces. get_interface() instantiates a derived class letter and the derived constructor selects this base class constructor in its initialization list (to avoid the recursion of the base class constructor calling get_interface() again). Since this is the letter and the letter IS the representation, interfaceRep is set to NULL (an uninitialized pointer causes problems in ∼Interface).
    References Dakota::abort_handler(), Interface::algebraic_function_type(), Interface::algebraicConstraintWeights, Interface::algebraicFnTags, Interface::algebraicFnTypes, Interface::algebraicMappings, Interface::algebraicVarTags, Interface::asl, ProblemDescDB::get_string(), Interface::interfaceId, Interface::outputLevel, Dakota::strends(), and Interface::user_auto_id().

14.92.3 Member Function Documentation

**Interface operator= ( const Interface & interface_in )**

assignment operator
    References Interface::interfaceRep, and Interface::referenceCount.
void assign_rep ( Interface * interface_rep, bool ref_count_incr = true )

assign letter or replace existing letter with a new one

Similar to the assignment operator, the assign_rep() function decrements referenceCount for the old interface-Rep and assigns the new interfaceRep. It is different in that it is used for publishing derived class letters to existing envelopes, as opposed to sharing representations among multiple envelopes (in particular, assign_rep is passed a letter object and operator= is passed an envelope object). Letter assignment supports two models as governed by ref_count_incr:

- ref_count_incr = true (default): the incoming letter belongs to another envelope. In this case, increment the reference count in the normal manner so that deallocation of the letter is handled properly.

- ref_count_incr = false: the incoming letter is instantiated on the fly and has no envelope. This case is modeled after get_interface(): a letter is dynamically allocated using new and passed into assign_rep, the letter’s reference count is not incremented, and the letter is not remotely deleted (its memory management is passed over to the envelope).

References Dakota::abort_handler(), Interface::interfaceRep, and Interface::referenceCount.
Referenced by DataFitSurrModel::DataFitSurrModel(), parallel_interface_plugin(), LibraryEnvironment::plugin_interface(), and run_dakota().

void eval_tag_prefix ( const String & eval_id_str, bool append_iface_id = true )

set the evaluation tag prefix (does not recurse)

default implementation just sets the list of eval ID tags; derived classes containing additional models or interfaces should override (currently no use cases)

References Interface::appendIfaceId, Interface::eval_tag_prefix(), Interface::evalTagPrefix, and Interface::interface-Rep.
Referenced by NestedModel::derived_evaluate(), SimulationModel::eval_tag_prefix(), and Interface::eval_tag_prefix().

void response_mapping ( const Response & algebraic_response, const Response & core_response, Response & total_response ) [protected]

combine the response from algebraic_mappings() with the response from derived_map() to create the total response

This function will get invoked even when only algebraic mappings are active (no core mappings from derived_map), since the AMPL algebraic_response may be ordered differently from the total_response. In this case, the core_response object is unused.

References Dakota::NPOS, Dakota::abort_handler(), Response::active_set_derivative_vector(), Response::active_set_request_vector(), Interface::algebraicACVIds, Interface::algebraicFnIndices, Interface::coreMappings, Dakota::find_index(), Response::function_gradient(), Response::function_gradient_view(), Response::function_gradients(), Response::function_hessian(), Response::function_hessian_view(), Response::function_hessians(), Response::function_value(), Response::function_values(), Response::function_values_view(), Interface::outputLevel, Response::reset(), and Response::reset_inactive().
Referenced by ApproximationInterface::map(), ApplicationInterface::map(), ApplicationInterface::synchronize(), and ApplicationInterface::synchronize_nowait().

Interface * get_interface ( ProblemDescDB & problem_db ) [private]

Used by the envelope to instantiate the correct letter class.
used only by the envelope constructor to initialize interfaceRep to the appropriate derived type.
String user_auto_id() [static], [private]

return the next available interface ID for no-ID user methods

Rationale: The parser allows multiple user-specified interfaces with empty (unspecified) ID. However, only a single Interface with empty ID can be constructed (if it’s the only one present, or the "last one parsed"). Therefore decided to prefer NO_ID over NO_ID_<num> for consistency with interface NO_ID convention. Additionally, NO_ID is preferred over NO_INTERFACE_ID (contrast with Iterator and Model) to preserve backward compatibility

References ProblemDescDB::get_string(), ProblemDescDB::get_ushort(), and Interface::interface_type().

Referenced by Interface::Interface().

String no_spec_id() [static], [private]

return the next available interface ID for on-the-fly methods

Rationale: For now NOSPEC_ID is chosen due to historical id="NO_SPECIFICATION" used for internally-constructed Iterators. Longer-term, consider auto-generating an ID that includes the context from which the method is constructed, e.g., the parent method or model’s ID, together with its name.

References Interface::noSpecIdNum.

14.92.4 Member Data Documentation

IntResponseMap rawResponseMap [protected]

Set of responses returned by either a blocking or nonblocking schedule.

The map is a full/partial set of completions which are identified through their evalIdCntr key. The raw set is postprocessed (i.e., finite diff grads merged) in Model::synchronize() where it becomes responseMap.

Referenced by ApplicationInterface::asynchronous_local_evaluations(), Interface::cache_unmatched_response(), ApplicationInterface::process_async_local(), ApplicationInterface::process_synch_local(), ApplicationInterface::receive_evaluation(), ApplicationInterface::synchronize(), ApproximationInterface::synchronize(), ApplicationInterface::synchronize_nowait(), ApproximationInterface::synchronize_nowait(), ApplicationInterface::test_local_backfill(), and ApplicationInterface::test_receives_backfill().

The documentation for this class was generated from the following files:

- DakotaInterface.hpp
- DakotaInterface.cpp

14.93 Iterator Class Reference

Base class for the iterator class hierarchy.

Inheritance diagram for Iterator:
Public Member Functions

- **Iterator** (std::shared_ptr< TraitsBase > traits=std::shared_ptr< TraitsBase >)(new TraitsBase()))
  
  *default constructor*

- **Iterator** (Iterator *iterator_rep, bool ref_count_incr=true, std::shared_ptr< TraitsBase > traits=std::shared_ptr< TraitsBase >)(new TraitsBase()))
  
  *alternate envelope constructor that assigns a representation pointer*

- **Iterator** (ProblemDescDB &problem_db, std::shared_ptr< TraitsBase > traits=std::shared_ptr< TraitsBase >)(new TraitsBase()))
  
  *standard envelope constructor, which constructs its own model(s)*

- **Iterator** (ProblemDescDB &problem_db, Model &model, std::shared_ptr< TraitsBase > traits=std::shared_ptr< TraitsBase >)(new TraitsBase()))
  
  *alternate envelope constructor which uses the ProblemDescDB but accepts a model from a higher level (meta-iterator) context, instead of constructing its own*

- **Iterator** (const String &method_string, Model &model, std::shared_ptr< TraitsBase > traits=std::shared_ptr< TraitsBase >)(new TraitsBase()))
  
  *alternate envelope constructor for instantiations by name without the ProblemDescDB*

- **Iterator** (const Iterator &iterator)
  
  *copy constructor*

- virtual ~Iterator ()
  
  *destructor*

- **Iterator operator=(const Iterator &iterator)
  
  *assignment operator*

- virtual void derived_set_communicators (ParLevLIter pl_iter)
  
  *derived class contributions to setting the communicators associated with this Iterator instance*

- virtual void derived_free_communicators (ParLevLIter pl_iter)
  
  *derived class contributions to freeing the communicators associated with this Iterator instance*

- virtual void initialize_run ()
  
  *utility function to perform common operations prior to pre_run(); typically memory initialization; setting of instance pointers*

- virtual void pre_run ()
  
  *pre-run portion of run (optional); re-implemented by Iterators which can generate all Variables (parameter sets) a priori*

- virtual void core_run ()
  
  *core portion of run; implemented by all derived classes and may include pre/post steps in lieu of separate pre/post*

- virtual void post_run (std::ostream &s)
post-run portion of run (optional); verbose to print results; re-implemented by Iterators that can read all Variables/Responses and perform final analysis phase in a standalone way

- virtual void finalize ()
  utility function to perform common operations following post_run(); deallocation and resetting of instance pointers

- virtual void pre_output ()
  write variables to file, following pre-run

- virtual void post_input ()
  read tabular data for post-run mode

- virtual void reset ()
  restore initial state for repeated sub-iterator executions

- virtual void nested_variable_mappings (const SizetArray &c_index1, const SizetArray &di_index1, const SizetArray &ds_index1, const SizetArray &dr_index1, const ShortArray &c_target2, const ShortArray &di_target2, const ShortArray &ds_target2, const ShortArray &dr_target2)
  set primary $\{CV, DIV, DRV\}$ MapIndices, secondary $\{CV, DIV, DRV\}$ MapTargets within derived Iterators; supports computation of higher-level sensitivities in nested contexts (e.g., derivatives of statistics w.r.t. inserted design variables)

- virtual void initialize_iterator (int job_index)
  used by IteratorScheduler to set the starting data for a run

- virtual void pack_parameters_buffer (MPIPackBuffer &send_buffer, int job_index)
  used by IteratorScheduler to pack starting data for an iterator run

- virtual void unpack_parameters_buffer (MPIUnpackBuffer &recv_buffer, int job_index)
  used by IteratorScheduler to unpack starting data for an iterator run

- virtual void unpack_parameters_initialize (MPIUnpackBuffer &recv_buffer, int job_index)
  used by IteratorScheduler to unpack starting data and initialize an iterator run

- virtual void pack_results_buffer (MPIPackBuffer &send_buffer, int job_index)
  used by IteratorScheduler to pack results data from an iterator run

- virtual void unpack_results_buffer (MPIUnpackBuffer &recv_buffer, int job_index)
  used by IteratorScheduler to unpack results data from an iterator run

- virtual void update_local_results (int job_index)
  used by IteratorScheduler to update local results arrays

- virtual const Variables & variables_results () const
  return a single final iterator solution (variables)

- virtual const Response & response_results () const
  return a single final iterator solution (response)

- virtual const VariablesArray & variables_array_results ()
  return multiple final iterator solutions (variables). This should only be used if returns_multiple_points() returns true.

- virtual const ResponseArray & response_array_results ()
  return multiple final iterator solutions (response). This should only be used if returns_multiple_points() returns true.

- virtual void response_results_active_set (const ActiveSet &set)
  set the requested data for the final iterator response results

- virtual const RealVector & response_error_estimates () const
  return error estimates associated with the final iterator solution

- virtual bool accepts_multiple_points () const
CHAPTER 14. CLASS DOCUMENTATION

indicates if this iterator accepts multiple initial points. Default return is false. Override to return true if appropriate.

virtual bool returns_multiple_points () const

indicates if this iterator returns multiple final points. Default return is false. Override to return true if appropriate.

virtual void initial_points (const VariablesArray &pts)
sets the multiple initial points for this iterator. This should only be used if accepts_multiple_points() returns true.

virtual void initialize_graphics (int iterator_server_id=1)
initialize the 2D graphics window and the tabular graphics data

virtual void print_results (std::ostream &s, short results_state=FINAL_RESULTS)
print the final iterator results

virtual const Model & algorithm_space_model () const
return the result of any recasting or surrogate model recursion layered on top of iteratedModel by the derived Iterator ctor chain

virtual unsigned short uses_method () const
return name of any enabling iterator used by this iterator

virtual void method_recourse ()
perform a method switch, if possible, due to a detected conflict

virtual const VariablesArray & all_variables ()
return the complete set of evaluated variables

virtual const RealMatrix & all_samples ()
return the complete set of evaluated samples

virtual const IntResponseMap & all_responses () const
return the complete set of computed responses

virtual int num_samples () const
get the current number of samples

virtual void sampling_reset (int min_samples, bool all_data_flag, bool stats_flag)
reset sampling iterator to use at least min_samples

virtual void sampling_reference (int samples_ref)
set reference number of samples, which is a lower bound during reset

virtual void sampling_increment ()
increment to next in sequence of refinement samples

virtual void random_seed (int seed)
set randomSeed, if present

virtual unsigned short sampling_scheme () const
return sampling name

virtual bool compact_mode () const
returns Analyzer::compactMode

virtual IntIntPair estimate_partition_bounds ()
estimate the minimum and maximum partition sizes that can be utilized by this Iterator

virtual bool resize ()
reinitializes iterator based on new variable size

virtual void declare_sources ()
Declare sources to the evaluations database.

void init_communicators (ParLevLIter pl_iter)
• initialize the communicators associated with this Iterator instance
  - void set_communicators (ParLevLIter pl_iter)
    set the communicators associated with this Iterator instance
• void free_communicators (ParLevLIter pl_iter)
  free the communicators associated with this Iterator instance
• void resize_communicators (ParLevLIter pl_iter, bool reinit_comms)
  Resize the communicators. This is called from the letter’s resize()
• void parallel_configuration_iterator (ParConfigLIter pc_iter)
  set methodPCIter
• ParConfigLIter parallel_configuration_iterator () const
  return methodPCIter
• void run (ParLevLIter pl_iter)
  invoke set_communicators(pl_iter) prior to run()
• void run ()
  orchestrate initialize/pre/core/post/finalize phases
• void assign_rep (Iterator *iterator_rep, bool ref_count_incr=true)
  replaces existing letter with a new one
• void iterated_model (const Model &model)
  set the iteratedModel (iterators and meta-iterators using a single model instance)
• Model & iterated_model ()
  return the iteratedModel (iterators & meta-iterators using a single model instance)
• ProblemDescDB & problem_description_db () const
  return the problem description database (probDescDB)
• ParallelLibrary & parallel_library () const
  return the parallel library (parallelLib)
• void method_name (unsigned short m_name)
  set the method name to an enumeration value
• unsigned short method_name () const
  return the method name via its native enumeration value
• void method_string (const String &m_str)
  set the method name by string
• String method_string () const
  return the method name by string
• String method_enum_to_string (unsigned short method_enum) const
  convert a method name enumeration value to a string
• unsigned short method_string_to_enum (const String &method_str) const
  convert a method name string to an enumeration value
• String submethod_enum_to_string (unsigned short submethod_enum) const
  convert a sub-method name enumeration value to a string
• const String & method_id () const
  return the method identifier (methodId)
• int maximum_evaluation_concurrency () const
  return the maximum evaluation concurrency supported by the iterator
void maximum_evaluation_concurrency (int max_conc)
set the maximum evaluation concurrency supported by the iterator

int maximum_iterations () const
return the maximum iterations for this iterator

void maximum_iterations (int max_iter)
set the maximum iterations for this iterator

void convergence_tolerance (Real conv_tol)
set the method convergence tolerance (convergenceTol)

Real convergence_tolerance () const
return the method convergence tolerance (convergenceTol)

void output_level (short out_lev)
set the method output level (outputLevel)

short output_level () const
return the method output level (outputLevel)

void summary_output (bool summary_output_flag)
Set summary output control; true enables evaluation/results summary.

size_t num_final_solutions () const
return the number of solutions to retain in best variables/response arrays

void num_final_solutions (size_t num_final)
set the number of solutions to retain in best variables/response arrays

void active_set (const ActiveSet &set)
set the default active set (for use with iterators that employ evaluate parameter_sets())

const ActiveSet & active_set () const
return the default active set (used by iterators that employ evaluate parameter_sets())

void active_set_request_vector (const ShortArray &asv)
return the default active set request vector (used by iterators that employ evaluate parameter_sets())

const ShortArray & active_set_request_vector () const
return the default active set request vector (used by iterators that employ evaluate parameter_sets())

void active_set_request_values (short asv_val)
return the default active set request vector (used by iterators that employ evaluate parameter_sets())

void sub_iterator_flag (bool si_flag)
set subIteratorFlag (and update summaryOutputFlag if needed)

bool is_null () const
function to check iteratorRep (does this envelope contain a letter?)

Iterator * iterator_rep () const
returns iteratorRep for access to derived class member functions that are not mapped to the top Iterator level

virtual void eval_tag_prefix (const String &eval_id_str)
set the hierarchical eval ID tag prefix

std::shared_ptr< TraitsBase > traits () const
returns methodTraits for access to derived class member functions that are not mapped to the top TraitsBase level

bool top_level ()
Return whether the iterator is the top level iterator.

void top_level (const bool &tflag)
Set the iterator's top level flag.
Protected Member Functions

- **Iterator** (BaseConstructor, ProblemDescDB &problem_db, std::shared_ptr<TraitsBase> traits=std::shared_ptr<TraitsBase>(new TraitsBase()))
  
  The constructor initializes the base class part of iterator classes. (BaseConstructor overloading avoids infinite recursion in the derived class constructors - Coplien, p. 139)

- **Iterator** (NoDBBaseConstructor, unsigned short method_name, Model &model, std::shared_ptr<TraitsBase> traits=std::shared_ptr<TraitsBase>(new TraitsBase()))
  
  Alternate constructor for base iterator classes constructed on the fly.

- **Iterator** (NoDBBaseConstructor, unsigned short method_name, std::shared_ptr<TraitsBase> traits=std::shared_ptr<TraitsBase>(new TraitsBase()))
  
  Alternate constructor for base iterator classes constructed on the fly.

- **virtual void derived_init_communicators** (ParLevIter pl_iter)
  
  Derived class contributions to initializing the communicators associated with this Iterator instance.

- **virtual void update_from_model** (const Model &model)
  
  Set inherited data attributes based on extractions from incoming model.

- **virtual const VariablesArray & initial_points** () const
  
  Gets the multiple initial points for this iterator. This will only be meaningful after a call to initial_points mutator.

- **StrStrSizeT run_identifier** () const
  
  Gets the unique run identifier based on method name, id, and number of executions.

Static Protected Member Functions

- **static void gnewton_set_recast** (const Variables &recast_vars, const ActiveSet &recast_set, ActiveSet &sub_model_set)
  
  Conversion of request vector values for the Gauss-Newton Hessian approximation.

Protected Attributes

- **ProblemDescDB & probDescDB**
  
  Class member reference to the problem description database.

- **ParallelLibrary & parallelLib**
  
  Class member reference to the parallel library.

- **ParConfigIter methodPCIter**
  
  The active ParallelConfiguration used by this Iterator instance.

- **Model iteratedModel**
  
  The model to be iterated (for iterators and meta-iterators employing a single model instance).

- **size_t myModelLayers**
  
  Number of Models locally (in Iterator or derived classes) wrapped around the initially passed in Model.

- **unsigned short methodName**
  
  Name of the iterator (the user's method spec).

- **Real convergenceTol**
  
  Iteration convergence tolerance.

- **int maxIterations**
  
  Maximum number of iterations for the iterator.
CHAPTER 14. CLASS DOCUMENTATION

- int maxFunctionEvals
  maximum number of function evaluations for the iterator
- int maxEvalConcurrency
  maximum number of concurrent model evaluations
- ActiveSet activeSet
  the response data requirements on each function evaluation
- size_t numFinalSolutions
  number of solutions to retain in best variables/response arrays
- VariablesArray bestVariablesArray
  collection of N best solution variables found during the study; always in context of Model originally passed to the Iterator (any in-flight Recasts must be undone)
- ResponseArray bestResponseArray
  collection of N best solution responses found during the study; always in context of Model originally passed to the Iterator (any in-flight Recasts must be undone)
- bool subIteratorFlag
  flag indicating if this Iterator is a sub-iterator (NestedModel::subIterator or DataFitSurrModel::daceIterator)
- short outputLevel
  output verbosity level: [SILENT, QUIET, NORMAL, VERBOSE, DEBUG]_OUTPUT
- bool summaryOutputFlag
  flag for summary output (evaluation stats, final results); default true, but false for on-the-fly (helper) iterators and sub-iterator use cases
- ResultsManager & resultsDB
  reference to the global iterator results database
- EvaluationStore & evaluationsDB
  reference to the global evaluation database
- EvaluationsDBState evaluationsDBState
  State of evaluations DB for this iterator.
- ResultsNames resultsNames
  valid names for iterator results
- std::shared_ptr<TraitsBase> methodTraits
  pointer that retains shared ownership of a TraitsBase object, or child thereof
- bool topLevel
  Whether this is the top level iterator.

Private Member Functions

- Iterator * get_iterator (ProblemDescDB &problem_db)
  Used by the envelope to instantiate the correct letter class.
- Iterator * get_iterator (ProblemDescDB &problem_db, Model &model)
  Used by the envelope to instantiate the correct letter class.
- Iterator * get_iterator (const String &method_string, Model &model)
  Used by the envelope to instantiate the correct letter class.
14.93. **ITERATOR CLASS REFERENCE**

**Static Private Member Functions**
- static String `user_auto_id()`  
  *return the next available method ID for no-ID user methods*
- static String `no_spec_id()`  
  *return the next available method ID for on-the-fly methods*

**Private Attributes**
- String `methodId`
  *method identifier string from the input file, or an auto-generated ID, such that each instance of an Iterator has a unique ID*
- size_t `execNum`
  *An execution number for this instance of the class. Now that each instance has a unique methodId, this is just a simple counter.*
- std::map< size_t, ParConfigIter > `methodPCIterMap`
  *track the available configurations that have been created (init Communications) and are available for activation at run time (set Communications)*
- Iterator * `iteratorRep`
  *pointer to the letter (initialized only for the envelope)*
- int `referenceCount`
  *number of objects sharing iteratorRep*

**Static Private Attributes**
- static size_t `noSpecIdNum = 0`
  *the last used method ID number for on-the-fly instantiations (increment before each use)*

### 14.93.1 Detailed Description

Base class for the iterator class hierarchy. The `Iterator` class is the base class for one of the primary class hierarchies in DAKOTA. The iterator hierarchy contains all of the iterative algorithms which use repeated execution of simulations as function evaluations. For memory efficiency and enhanced polymorphism, the iterator hierarchy employs the "letter/envelope idiom" (see Coplien "Advanced C++", p. 133), for which the base class (Iterator) serves as the envelope and one of the derived classes (selected in `Iterator::get_iterator()`) serves as the letter.

### 14.93.2 Constructor & Destructor Documentation

**Iterator ( std::shared_ptr< TraitsBase > traits = std::shared_ptr< TraitsBase > (new TraitsBase()) )**

default constructor

The default constructor is used in `Vector<Iterator>` instantiations and for initialization of `Iterator` objects contained in meta-Iterators and `Model` recursions. `iteratorRep` is NULL in this case, making it necessary to check for NULL pointers in the copy constructor, assignment operator, and destructor.
Iterator ( Iterator * iterator_rep, bool ref_count_incr = true, std::shared_ptr<TraitsBase> traits = std::shared_ptr<TraitsBase>(new TraitsBase()) )

alternate envelope constructor that assigns a representation pointer

This constructor assigns a representation pointer and optionally increments its reference count. It behaves the same as a default construction followed by assign_rep().

References Iterator::iteratorRep, and Iterator::referenceCount.

Iterator ( ProblemDescDB & problem_db, std::shared_ptr<TraitsBase> traits = std::shared_ptr<TraitsBase>(new TraitsBase()) )

standard envelope constructor, which constructs its own model(s)

Envelope constructor only needs to extract enough data to properly execute get_iterator(), since letter holds the actual base class data. This version is used for top-level ProblemDescDB-driven construction of all Iterators and MetaIterators, which construct their own Model instances.

References Dakota::abort_handler(), Iterator::get_iterator(), and Iterator::iteratorRep.

Iterator ( ProblemDescDB & problem_db, Model & model, std::shared_ptr<TraitsBase> traits = std::shared_ptr<TraitsBase>(new TraitsBase()) )

alternate envelope constructor which uses the ProblemDescDB but accepts a model from a higher level (meta-iterator) context, instead of constructing its own

Envelope constructor only needs to extract enough data to properly execute get_iterator(), since letter holds the actual base class data. This version is used for ProblemDescDB-driven construction of Iterators that are passed a Model from a higher-level context (e.g., a MetaIterator instantiates its sub-iterator(s) by name instead of pointer and passes in its iteratedModel, since these sub-iterators lack their own model pointers).

References Dakota::abort_handler(), Iterator::get_iterator(), and Iterator::iteratorRep.

Iterator ( const String & method_string, Model & model, std::shared_ptr<TraitsBase> traits = std::shared_ptr<TraitsBase>(new TraitsBase()) )

alternate envelope constructor for instantiations by name without the ProblemDescDB

Used in sub-iterator instantiations within iterator constructors. Envelope constructor only needs to extract enough data to properly execute get_iterator(), since letter holds the actual base class data. This version is used for lightweight constructions without the ProblemDescDB.

References Dakota::abort_handler(), Iterator::get_iterator(), and Iterator::iteratorRep.

Iterator ( const Iterator & iterator )

copy constructor

Copy constructor manages sharing of iteratorRep and incrementing of referenceCount.

References Iterator::iteratorRep, and Iterator::referenceCount.

~Iterator ( ) [virtual]

destructor

Destructor decrements referenceCount and only deletes iteratorRep when referenceCount reaches zero.

References Iterator::iteratorRep, and Iterator::referenceCount.
14.93. **ITERATOR CLASS REFERENCE**

**Iterator (BaseConstructor, ProblemDescDB & problem_db, std::shared_ptr<TraitsBase> traits = std::shared_ptr<TraitsBase>(new TraitsBase()) ) [protected]**

Constructor initializes the base class part of letter classes (BaseConstructor overloading avoids infinite recursion in the derived class constructors - Coplien, p. 139)

This constructor builds the base class data for all inherited iterators, including meta-iterators. get_iterator() instantiates a derived class and the derived class selects this base class constructor in its initialization list (to avoid the recursion of the base class constructor calling get_iterator() again). Since the letter IS the representation, its representation pointer is set to NULL (an uninitialized pointer causes problems in ~Iterator).

References Iterator::method_enum_to_string(), Iterator::methodId, Iterator::methodName, Iterator::outputLevel, and Iterator::user_auto_id().

**Iterator (NoDBBaseConstructor, unsigned short method_name, Model & model, std::shared_ptr<TraitsBase> traits = std::shared_ptr<TraitsBase>(new TraitsBase()) ) [protected]**

Alternate constructor for base iterator classes constructed on the fly

This alternate constructor builds base class data for inherited iterators. It is used for on-the-fly instantiations for which DB queries cannot be used, and is not used for construction of meta-iterators.

14.93.3 **Member Function Documentation**

**Iterator operator=( const Iterator & iterator )**

Assignment operator


References Iterator::iteratorRep, and Iterator::referenceCount.

**void initialize_run() [virtual]**

Utility function to perform common operations prior to pre_run(); typically memory initialization; setting of instance pointers

Perform initialization phases of run sequence, like allocating memory and setting instance pointers. Commonly used in sub-iterator executions. This is a virtual function; when re-implementing, a derived class must call its nearest parent's initialize_run(), typically before performing its own implementation steps.

Reimplemented in Optimizer, SNLLOptimizer, APPSOptimizer, SNLLLeastSq, Minimizer, CONMINOptimizer, Analyzer, NonD, ROLOptimizer, and LeastSq.

References Iterator::initialize_run(), and Iterator::iteratorRep.

Referenced by Iterator::initialize_run(), Iterator::run(), and SeqHybridMetaIterator::run_sequential_adaptive().
void pre_run() [virtual]

pre-run portion of run (optional); re-implemented by Iterators which can generate all Variables (parameter sets) a priori

pre-run phase, which a derived iterator may optionally reimplement; when not present, pre-run is likely integrated into the derived run function. This is a virtual function; when re-implementing, a derived class must call its nearest parent’s pre_run(), if implemented, typically before performing its own implementation steps.


References Iterator::iteratorRep, and Iterator::pre_run().

Referenced by NonDBayesCalibration::build_designs(), Iterator::pre_run(), and Iterator::run().

void core_run() [virtual]

core portion of run; implemented by all derived classes and may include pre/post steps in lieu of separate pre/post Virtual run function for the iterator class hierarchy. All derived classes need to redefine it.

Reimplemented in JEGAOptimizer, NonDSampling, NOWPACOptimizer, APPSOptimizer, SNLLOptimizer, NonDIntegration, COLINOptimizer, SNLLEstSq, CONMINOptimizer, NLSSOLLeastSq, SurrBasedGlobalMinimizer, NonDBayesCalibration, NonDLHSSampling, NCSUOptimizer, EffGlobalMinimizer, NonDMultilevelPolynomialChaos, NonlinearCGOptimizer, ROLOptimizer, NL2SOLLeastSq, OptDartsOptimizer, NonDAdaptImpSampling, NonDMultilevelFunctionTrain, SeqHybridMetaIterator, NonDLocalReliability, NonDAdaptiveSampling, ConcurrentMetaIterator, DDACEDesignCompExp, NonDExpansion, NonDGPImpSampling, NonDMultilevelStochCollocation, NonDPOFDarts, FSUDesignCompExp, NonDGlobalInterval, NonDLocalInterval, ParamStudy, PSUADesignCompExp, EmbedHybridMetaIterator, NonDMultilevelSampling, SurrBasedLocalMinimizer, NonDGlobalReliability, CollabHybridMetaIterator, NonDLHSInterval, NonDRKDDarts, RichExtrapVerification, and NonDSurrogateExpansion.

References Dakota::abort_handler(), Iterator::core_run(), and Iterator::iteratorRep.

Referenced by Iterator::core_run(), and Iterator::run().

void post_run(std::ostream &s) [virtual]

post-run portion of run (optional); verbose to print results; re-implemented by Iterators that can read all Variables/-Responses and perform final analysis phase in a standalone way

Post-run phase, which a derived iterator may optionally reimplement; when not present, post-run is likely integrated into run. This is a virtual function; when re-implementing, a derived class must call its nearest parent’s post_run(), typically after performing its own implementation steps.

Reimplemented in Optimizer, SNLLOptimizer, COLINOptimizer, NonDRKDDarts, Minimizer, Analyzer, NonDLHSSampling, DDACEDesignCompExp, FSUDesignCompExp, MetaIterator, NonDReliability, ParamStudy, PSUADesignCompExp, SurrBasedLocalMinimizer, NonDMultilevelSampling, and LeastSq.

References Iterator::iteratorRep, and Iterator::post_run().

Referenced by Iterator::post_run(), and Iterator::run().

void finalize_run() [virtual]

utility function to perform common operations following post_run(); deallocation and resetting of instance pointers

Optional: perform finalization phases of run sequence, like dealocating memory and resetting instance pointers. Commonly used in sub-iterator executions. This is a virtual function; when re-implementing, a derived class must call its nearest parent’s finalize_run(), typically after performing its own implementation steps.

Reimplemented in Optimizer, SNLLOptimizer, SNLLEstSq, Minimizer, Analyzer, NonD, and LeastSq.

References Iterator::finalize_run(), and Iterator::iteratorRep.
Referenced by Iterator::finalize_run, Analyzer::finalize_run, Minimizer::finalize_run, Iterator::run, and SeqHybridMetaIterator::run_sequential_adaptive.

```cpp
void initialize_graphics ( int iterator_server_id = 1 ) [virtual]
```

initialize the 2D graphics window and the tabular graphics data

This is a convenience function for encapsulating graphics initialization operations. It is overridden by derived classes that specialize the graphics display.

Reimplemented in NonDReliability, and SurrBasedMinimizer.

References Model::auto_graphics(), Graphics::create_plots_2d(), OutputManager::create_tabular_datastream(), Model::current_response(), Model::current_variables(), OutputManager::graph2DFlag, OutputManager::graphics(), Iterator::initialize_graphics(), Iterator::iteratedModel, Iterator::iteratorRep, ParallelLibrary::output_manager(), Iterator::parallelLib, and OutputManager::tabularDataFlag.

Referenced by CollabHybridMetaIterator::core_run(), EmbedHybridMetaIterator::core_run(), ConcurrentMetaIterator::core_run(), Environment::execute(), Iterator::initialize_graphics(), SeqHybridMetaIterator::run_sequential(), and SeqHybridMetaIterator::run_sequential_adaptive.

```cpp
void print_results ( std::ostream & s, short results_state = FINAL_RESULTS ) [virtual]
```

print the final iterator results

This virtual function provides additional iterator-specific final results outputs beyond the function evaluation summary printed in finalize_run.

Reimplemented in Optimizer, NonDPolynomialChaos, Analyzer, NonDGPMSSABayesCalibration, NonDLHSSampling, NonDMultilevelPolynomialChaos, NonDBayesCalibration, NonDMultilevelFunctionTrain, NonDPOFDarts, NonDQUESOBayesCalibration, NonDMultilevelStochCollocation, NonDAdaptImpSampling, NonDWasabiBayesCalibration, SeqHybridMetaIterator, NonDAdaptiveSampling, NonDLocalReliability, ConcurrentMetaIterator, NonDExpansion, NonDGPImpSampling, NonDMUQBayesCalibration, NonDInterval, NonDMultilevelSampling, PStudyDACE, NonDGlobalReliability, LeastSq, Verification, SurrBasedMinimizer, and RichExtrapVerification.

References Iterator::iteratorRep, and Iterator::print_results().

Referenced by MetaIterator::post_run, Minimizer::post_run, and Iterator::print_results().

```cpp
unsigned short uses_method ( ) const [virtual]
```

return name of any enabling iterator used by this iterator

This is used to avoid clashes in state between non-object-oriented (i.e., F77, C) iterator executions, when such iterators could potentially be executing simultaneously (e.g., nested execution). It is not an issue (and a used method is not reported) in cases where a helper execution is completed before a lower level one could be initiated; an example of this is DIRECT for maximization of expected improvement: the EIF maximization is completed before a new point evaluation (which could include nested iteration) is performed.

Reimplemented in NonDLocalReliability, and NonDLocalInterval.

References Iterator::iteratorRep, and Iterator::uses_method().

Referenced by NCSUOptimizer::initialize(), CONMINOptimizer::initialize(), NonDLocalInterval::NonDLocalInterval(), NonDLocalReliability::NonDLocalReliability(), SOLBase::SOLBase(), and Iterator::uses_method().

```cpp
void run ( )
```

orchestrate initialize/pre/core/post/finalize phases

Iterator supports a construct/initialize-run/pre-run/core-run/post-run/finalize-run/destruct progression. This member (non-virtual) function sequences these run phases.
References ParallelLibrary::command_line_post_run(), ParallelLibrary::command_line_pre_run(), ParallelLibrary::command_line_run(), iterator::core_run(), iterator::declare_sources(), iterator::evaluationsDB, iterator::evaluationsDBState, iterator::execNum, iterator::finalize_run(), resultsManager::flush(), iterator::initialize_run(), iterator::iteratorRep, iterator::method_enum_to_string(), iterator::method_id(), iterator::method_string(), iterator::methodName, iterator::outputLevel, iterator::parallelLib, iterator::post_input(), iterator::post_run(), iterator::pre_output(), iterator::pre_run(), iterator::resultsDB, iterator::run(), iterator::summaryOutputFlag, and iterator::top_level().

Referenced by iterator::run().

void assign_rep ( iterator * iterator_rep, bool ref_count_incr = true )

replaces existing letter with a new one

Similar to the assignment operator, the assign_rep() function decrements referenceCount for the old iterator-Rep and assigns the new iteratorRep. It is different in that it is used for publishing derived class letters to existing envelopes, as opposed to sharing representations among multiple envelopes (in particular, assign_rep is passed a letter object and operator= is passed an envelope object). Letter assignment supports two models as governed by ref_count_incr:

- ref_count_incr = true (default): the incoming letter belongs to another envelope. In this case, increment the reference count in the normal manner so that deallocation of the letter is handled properly.

- ref_count_incr = false: the incoming letter is instantiated on the fly and has no envelope. This case is modeled after get_iterator(): a letter is dynamically allocated using new and passed into assign_rep, the letter’s reference count is not incremented, and the letter is not remotely deleted (its memory management is passed over to the envelope).

References Dakota::abort_handler(), iterator::iterator_rep(), iterator::iteratorRep, and iterator::referenceCount. Referenced by NonDBayesCalibration::build_designs(), NonDExpansion::construct_cubature(), NonDExpansion::construct_expansion_sampler(), NonDAAdaptiveSampling::construct_fsu_sampler(), NonD::construct_lhs(), NonD::BayesCalibration::construct_mcmc_model(), NonDExpansion::construct_quadrature(), NonDExpansion::construct_sparse_grid(), Minimizer::data_transform_model(), EffGlobalMinimizer::EffGlobalMinimizer(), ActiveSubspaceModel::init_fullspace_sampler(), NonDLocalInterval::method_recourse(), NonDLocalReliability::method_recourse(), NonDAAdaptiveSampling::NonDAAdaptiveSampling(), NonD::BayesCalibration::NonD::BayesCalibration(), NonDGlobalInterval::NonDGlobalInterval(), NonDGlobalReliability::NonDGlobalReliability(), NonDGPMSA::NonDGPMSA, NonDGPMSABayesCalibration::NonDGPMSABayesCalibration(), NonDLHSInterval::NonDLHSInterval(), NonDLocalInterval::NonDLocalInterval(), NonDLocalReliability::NonDLocalReliability(), GaussProcApproximation::optimize_theta_global(), GaussProcApproximation::optimize_theta_multipoint(), and SurrBasedLocalMinimizer::relax_constraints().

void eval_tag_prefix ( const String & eval_id_str ) [virtual]

set the hierarchical eval ID tag prefix

This prepend may need to become a virtual function if the tagging should propagate to other subModels or helper Iterators an Iterator may contain.

References iterator::eval_tag_prefix(), Model::eval_tag_prefix(), iterator::iteratedModel, and iterator::iteratorRep.

Referenced by NestedModel::derived_evaluate(), iterator::eval_tag_prefix(), iterator::init_communicators(), NestedModel::initialize_iterator(), and DataFitSurrModel::run_dace().
void gnewton_set_recast ( const Variables & recast_vars, const ActiveSet & recast_set, ActiveSet & sub_model_set ) [static], [protected]

conversion of request vector values for the Gauss-Newton Hessian approximation
   For Gauss-Newton Hessian requests, activate the 2 bit and mask the 4 bit.
   References ActiveSet::request_value(), and ActiveSet::request_vector().
   Referenced by NonDBayesCalibration::construct_map_model(), and Optimizer::reduce_model().

Iterator * get_iterator ( ProblemDescDB & problem_db ) [private]

Used by the envelope to instantiate the correct letter class.
   Used only by the envelope constructor to initialize iteratorRep to the appropriate derived type, as given by
   the DB’s method_name. Supports all iterators and meta-iterators. These instantiations will NOT recurse on the
   Iterator(problem_db) constructor due to the use of BaseConstructor.
   References ProblemDescDB::get_model(), ProblemDescDB::get_ushort(), Iterator::method_enum_to_string(),
   Iterator::method_name(), and Dakota::SUBMETHOD_COLLABORATIVE.
   Referenced by Iterator::Iterator().

Iterator * get_iterator ( ProblemDescDB & problem_db, Model & model ) [private]

Used by the envelope to instantiate the correct letter class.
   Used only by the envelope constructor to initialize iteratorRep to the appropriate derived type. Alternate
   construction of meta-iterators is supported to enable use of meta-iterators as components. These instantiations
   will NOT recurse on the Iterator(problem_db, model) constructor due to the use of BaseConstructor.
   References ProblemDescDB::get_ushort(), Iterator::method_enum_to_string(), Iterator::method_name(), Iterator-::probDescDB, Dakota::SUBMETHOD_COLLABORATIVE, Iterator::submethod_enum_to_string(), and Model-::surrogate_type().

Iterator * get_iterator ( const String & method_string, Model & model ) [private]

Used by the envelope to instantiate the correct letter class.
   Used only by the envelope constructor to initialize iteratorRep to the appropriate derived type, as given by
   the passed method_string. Lightweight instantiations by name are supported by a subset of Iterators (primarily
   Minimizers).
   References Dakota::strbegins(), and Dakota::strends().

String user_auto_id ( ) [static], [private]

return the next available method ID for no-ID user methods
   Rationale: The parser allows multiple user-specified methods with empty (unspecified) ID. However, only a
   single Iterator with empty ID can be constructed (if it’s the only one present, or the "last one parsed"). Therefore
decided to prefer NO_METHOD_ID over NO_METHOD_ID.<num> for (partial) consistency with interface N- O_ID convention. The addition of METHOD is it distinguish methods, models and interfaces in the HDF5 output.
   Referenced by Iterator::Iterator().

String no_spec_id ( ) [static], [private]

return the next available method ID for on-the-fly methods
   Rationale: For now NOSPEC_METHOD_ID is chosen due to historical id="NO_SPECIFICATION" used for internally-constructed Iterators. Longer-term, consider auto-generating an ID that includes the context from which the method is constructed, e.g., the parent method or model’s ID, together with its name.
14.93.4 Member Data Documentation

ProblemDescDB& probDescDB  [protected]

class member reference to the problem description database

Referenced by MetaIterator::allocate_by_name(), MetaIterator::allocate_by_pointer(), Analyzer::Analyzer(),
MetaIterator::check_model(), COLINOptimizer::COLINOptimizer(), NonDC3FunctionTrain::config_regression(),
NonDBayesCalibration::construct_mcmc_model(), Minimizer::data_transform_model(), SurrBasedMinimizer::derived_init_communicators(),
EmbedHybridMetaIterator::derived_init_communicators(), ConcurrentMetaIterator::derived_init_communicators(),
EffGlobalMinimizer::EffGlobalMinimizer(), MetaIterator::estimate_by_name(), MetaIterator::estimate_by_pointer(),
CollabHybridMetaIterator::estimate_partition_bounds(), EmbedHybridMetaIterator::estimate_partition_bounds(),
SeqHybridMetaIterator::estimate_partition_bounds(), FSUDesignCompExp::FSUDesignCompExp(), Optimizer::get_common_stopping_criteria(),
Iterator::get_iterator(), ConcurrentMetaIterator::initialize_model(), NOWPACOptimizer::initialize_options(),
SurrBasedLocalMinimizer::initialize_sub_minimizer(), JEGAOptimizer::JEGAOptimizer(), NLSSOLLeastSq::NLSSOLLeastSq(),
NonDAdaptImpSampling::NonDAdaptImpSampling(), NonDAdaptiveSampling::NonDAdaptiveSampling(),
NonDBayesCalibration::NonDBayesCalibration(), NonDC3FunctionTrain::NonDC3FunctionTrain(), NonDGralGlobalInterval::NonDGralGlobalInterval(),
NonDGralLocalReliability::NonDGralLocalReliability(), NonDGPImpSampling::NonDGPImpSampling(),
NonDGPMSABayesCalibration::NonDGPMSABayesCalibration(), NonDInterval::NonDInterval(),
NonDLocalInterval::NonDLocalInterval(), NonDLocalReliability::NonDLocalReliability(),
NonDMultilevelFunctionTrain::NonDMultilevelFunctionTrain(), NonDMultilevelPolynomialChaos::NonDMultilevelPolynomialChaos(),
NonDMultilevelStochCollocation::NonDMultilevelStochCollocation(), NonDQuadrature::NonDQuadrature(),
NonDSampling::NonDSampling(), NonDSparseGrid::NonDSparseGrid(), NonDStochCollocation::NonDStochCollocation(),
NonDSurrogateExpansion::NonDSurrogateExpansion(), OptDartsOptimizer::OptDartsOptimizer(),
ParamStudy::ParamStudy(), NonlinearCGOptimizer::parse_options(),
Iterator::problem_description_db(), NonDC3FunctionTrain::push_c3_db_options(),
APPSOptimizer::set_apps_parameters(), ROLOptimizer::set_rol_parameters(), POLINOptimizer::set_solver_parameters(),
SNLLLeastSq::SNLLLeastSq(), SNLLOptimizer::SNLLOptimiser(),
COLINOptimizer::solver_setup(), and SurrBasedGlobalMinimizer::SurrBasedGlobalMinimizer().

int maxEvalConcurrency  [protected]

maximum number of concurrent model evaluations

This is important for parallel configuration init/set/free and may be set within empty envelope instances.
Therefore, it cannot be pushed down into Analyzer/Minimizer derived classes.

Referenced by NonDPolynomialChaos::config_expectation(), NonDC3FunctionTrain::config_regression(),
NonDPolynomialChaos::config_regression(), DDACEEDesignCompExp::DDACEEDesignCompExp(), SurrBasedMinimizer::derived_free_communicators(),
NonDGralReliability::derived_free_communicators(), NonDLocalInterval::derived_free_communicators(),
NonDGralInterval::derived_free_communicators(), NonDExpansion::derived_free_communicators(),
NonDGPImpSampling::derived_free_communicators(), NonDAdaptiveSampling::derived_free_communicators(),
NonDBayesCalibration::derived_free_communicators(), MetaIterator::derived_free_communicators(),
IterateDep::derived_free_communicators(), NonDGralReliability::derived_init_communicators(),
NonDLocalInterval::derived_init_communicators(), NonDGralInterval::derived_init_communicators(),
NonDExpansion::derived_init_communicators(), NonDGPImpSampling::derived_init_communicators(),
NonDAdaptiveSampling::derived_init_communicators(), NonDBayesCalibration::derived_init_communicators(),
NonDLocalReliability::derived_init_communicators(), NonDBayesCalibration::derived_init_communicators(),
NonD-
14.94 IteratorScheduler Class Reference

This class encapsulates scheduling operations for concurrent sub-iteration within an outer level context (e.g., meta-iteration, nested models).

Public Member Functions

- **IteratorScheduler (ParallelLibrary &parallel_lib, bool peer_assign_jobs, int num_servers=0, int procs_per_iterator=0, short scheduling=DEFAULT_SCHEDULING)**
  
  constructor

- **~IteratorScheduler ()**
  
  destructor

- **void construct_sub_iterator (ProblemDescDB &problem_db, Iterator &sub_iterator, Model &sub_model, const String &method_ptr, const String &method_name, const String &model_ptr)***

  instantiate sub_iterator on the current rank if not already constructed

- **IntIntPair configure (ProblemDescDB &problem_db, Iterator &sub_iterator, Model &sub_model)**

  performs sufficient initialization to define partitioning controls (min and max processors per iterator server)

- **IntIntPair configure (ProblemDescDB &problem_db, const String &method_string, Iterator &sub_iterator, Model &sub_model)**

  performs sufficient initialization to define partitioning controls (min and max processors per iterator server)

- **IntIntPair configure (ProblemDescDB &problem_db, Iterator &sub_iterator)**

  performs sufficient initialization to define partitioning controls (min and max processors per iterator server)

- **void partition (int max_iterator_concurrency, IntIntPair &ppi_pr)**

  convenience function for initializing iterator communicators, setting parallel configuration attributes, and managing outputs and restart.

- **void init_iterator (ProblemDescDB &problem_db, Iterator &sub_iterator, Model &sub_model)**

  invokes static version of this function with appropriate parallelism level
• void init_iterator (ProblemDescDB &problem_db, const String &method_string, Iterator &sub_iterator, Model &sub_model)
  invokes static version of this function with appropriate parallelism level
• void set_iterator (Iterator &sub_iterator)
  invokes static version of this function with appropriate parallelism level
• void run_iterator (Iterator &sub_iterator)
  invokes static version of this function with appropriate parallelism level
• void free_iterator (Iterator &sub_iterator)
  invokes static version of this function with appropriate parallelism level
• void free_iterator.parallelism ()
  convenience function for deallocating the concurrent iterator parallelism level
• template<typename MetaType >
  void schedule_iterators (MetaType &meta_object, Iterator &sub_iterator)
  short convenience function for distributing control among master_dynamic_schedule_iterators(), serve_iterators(), and peer_static_schedule_iterators()
• template<typename MetaType >
  void master_dynamic_schedule_iterators (MetaType &meta_object)
  executed by the scheduler master to manage a dynamic schedule of iterator jobs among slave iterator servers
• void stop_iterator_servers ()
  executed by the scheduler master to terminate slave iterator servers
• template<typename MetaType >
  void serve_iterators (MetaType &meta_object, Iterator &sub_iterator)
  executed on the slave iterator servers to perform iterator jobs assigned by the scheduler master
• template<typename MetaType >
  void peer_static_schedule_iterators (MetaType &meta_object, Iterator &sub_iterator)
  executed on iterator peers to manage a static schedule of iterator jobs
• void update (ParConfigLIter pc_iter)
  update schedPCIter
• void update (size_t index)
  update miPLIndex as well as associated settings for concurrent iterator scheduling from the corresponding Parallel-Level
• void update (ParConfigLIter pc_iter, size_t index)
  invoke update(ParConfigLIter) and update(size_t) in sequence
• void iterator_message_lengths (int params_msg_len, int results_msg_len)
  update paramsMsgLen and resultsMsgLen
• bool lead_rank () const
  determines if current processor is rank 0 of the parent comm

Static Public Member Functions

• static void init_iterator (ProblemDescDB &problem_db, Iterator &sub_iterator, ParLevLIter pl_iter)
  convenience function for allocation of an iterator and (parallel) initialization of its comms
• static void init_iterator (ProblemDescDB &problem_db, Iterator &sub_iterator, Model &sub_model, ParLevLIter pl_iter)
  convenience function for allocation of an iterator and (parallel) initialization of its comms
• static void init_iterator (ProblemDescDB &problem_db, const String &method_string, Iterator &sub_iterator, Model &sub_model, ParLevLIter pl_iter)
  convenience function for lightweight allocation of an iterator and (parallel) initialization of its comms
• static void set_iterator (Iterator &sub_iterator, ParLevLIter pl_iter)
  convenience function for setting comms prior to running an iterator
• static void run_iterator (Iterator &sub_iterator, ParLevLIter pl_iter)
  Convenience function for invoking an iterator and managing parallelism. This version omits communicator repartitioning. Function must be public due to use by MINLPNode.
• static void free_iterator (Iterator &sub_iterator, ParLevLIter pl_iter)
  convenience function for deallocating comms after running an iterator

Public Attributes

• ParallelLibrary & parallelLib
  reference to the ParallelLibrary instance
• int numIteratorJobs
  number of iterator executions to schedule
• int numIteratorServers
  number of concurrent iterator partitions
• int procsPerIterator
  partition size request
• int iteratorCommRank
  processor rank in iteratorComm
• int iteratorCommSize
  number of processors in iteratorComm
• int iteratorServerId
  identifier for an iterator server
• bool messagePass
  flag for message passing among iterator servers
• short iteratorScheduling
  {DEFAULT,MASTER,PEER}_SCHEDULING
• bool peerAssignJobs
  flag indicating need for peer 1 to assign jobs to peers 2-n
• ParConfigLIter schedPCIter
  iterator for active parallel configuration
• size_t miPLIndex
  index of active parallel level (corresponding to ParallelConfiguration::miPLIter) to use < for parallelLib send/recv

Private Attributes

• int paramsMsgLen
  length of MPI buffer for parameter input instance(s)
• int resultsMsgLen
  length of MPI buffer for results output instance(s)
14.94.1 Detailed Description

This class encapsulates scheduling operations for concurrent sub-iteration within an outer level context (e.g., meta-iteration, nested models).

In time, a Scheduler class hierarchy is envisioned, but for now, this class is not part of a hierarchy.

14.94.2 Constructor & Destructor Documentation

IteratorScheduler ( ParallelLibrary & parallel_lib, bool peer_assign_jobs, int num_servers = 0, int procs_per_iterator = 0, short scheduling = DEFAULT_SCHEDULING )

Constructor

Current constructor parameters are the input specification components, which are requests subject to override by ParallelLibrary::init_iterator_communicators().

14.94.3 Member Function Documentation

void init_iterator ( ProblemDescDB & problem_db, Iterator & sub_iterator, ParLevLIter pl_iter ) [static]

Convenience function for allocation of an iterator and (parallel) initialization of its comms

This is a convenience function for encapsulating the allocation of communicators prior to running an iterator.

References ProblemDescDB::get_iterator(), ProblemDescDB::get_model(), ProblemDescDB::get_ushort(), Model::init_comms_bcast_flag(), Iterator::init_communicators(), Iterator::is_null(), Model::is_null(), Iterator::iterated_model(), Iterator::maximum_evaluation_concurrency(), Iterator::method_name(), Model::serve_init_communicators(), and Model::stop_init_communicators().

Referenced by MetaIterator::allocate_by_name(), MetaIterator::allocate_by_pointer(), Environment::construct(), ConcurrentMetaIterator::derived_init_communicators(), NestedModel::derived_init_communicators(), and IteratorScheduler::init_iterator().

void init_iterator ( ProblemDescDB & problem_db, Iterator & sub_iterator, Model & sub_model, ParLevLIter pl_iter ) [static]

Convenience function for allocation of an iterator and (parallel) initialization of its comms

This is a convenience function for encapsulating the allocation of communicators prior to running an iterator.

References ProblemDescDB::get_iterator(), ProblemDescDB::get_ushort(), Model::init_comms_bcast_flag(), Iterator::init_communicators(), Iterator::is_null(), Iterator::iterated_model(), Iterator::maximum_evaluation_concurrency(), Iterator::method_name(), Model::serve_init_communicators(), and Model::stop_init_communicators().

void init_iterator ( ProblemDescDB & problem_db, const String & method_string, Iterator & sub_iterator, Model & sub_model, ParLevLIter pl_iter ) [static]

Convenience function for lightweight allocation of an iterator and (parallel) initialization of its comms

This is a convenience function for encapsulating the allocation of communicators prior to running an iterator.

References ProblemDescDB::get_iterator(), Model::init_comms_bcast_flag(), Iterator::init_communicators(), Iterator::is_null(), Iterator::iterated_model(), Iterator::maximum_evaluation_concurrency(), Iterator::method_string(), Model::serve_init_communicators(), and Model::stop_init_communicators().
void set_iterator ( Iterator & sub_iterator, ParLevIter pl_iter ) [static]

convenience function for setting comms prior to running an iterator

This is a convenience function for encapsulating the deallocation of communicators after running an iterator.
References Iterator::derived_set_communicators(), and Iterator::set_communicators().
References by CollabHybridMetaIterator::derived_set_communicators(), EmbedHybridMetaIterator::derived_set_communicators(), ConcurrentMetaIterator::derived_set_communicators(), SeqHybridMetaIterator::derived_set_communicators(), NestedModel::derived_set_communicators(), and IteratorScheduler::set_iterator().

void run_iterator ( Iterator & sub_iterator, ParLevIter pl_iter ) [static]

Convenience function for invoking an iterator and managing parallelism. This version omits communicator repartitioning. Function must be public due to use by MINLPNode.
This is a convenience function for encapsulating the parallel features (run/serve) of running an iterator. This function omits allocation/deallocation of communicators to provide greater efficiency in approaches that involve multiple iterator executions but only require communicator allocation/deallocation to be performed once.
References Model::finalize_mapping(), Model::initialize_mapping(), Iterator::iterated_model(), Iterator::maximum_evaluation_concurrency(), Iterator::method_name(), Iterator::resize(), Iterator::resize_communicators(), Iterator::run(), Model::serve_finalize_mapping(), Model::serve_init_mapping(), Model::serve_run(), Model::stop_finalize_mapping(), Model::stop_init_mapping(), and Model::stop_servers().
References by NestedModel::derived_evaluate(), Environment::execute(), IteratorScheduler::peer_static_schedule_iterators(), IteratorScheduler::run_iterator(), and IteratorScheduler::serve_iterators().

void free_iterator ( Iterator & sub_iterator, ParLevIter pl_iter ) [static]

convenience function for deallocating comms after running an iterator

This is a convenience function for encapsulating the deallocation of communicators after running an iterator.
References Iterator::derived_free_communicators(), Iterator::free_communicators(), and Iterator::method_name().
References by CollabHybridMetaIterator::derived_free_communicators(), EmbedHybridMetaIterator::derived_free_communicators(), ConcurrentMetaIterator::derived_free_communicators(), SeqHybridMetaIterator::derived_free_communicators(), NestedModel::derived_free_communicators(), Environment::destruct(), and IteratorScheduler::free_iterator().

IntIntPair configure ( ProblemDescDB & problem_db, Iterator & sub_iterator, Model & sub_model )

performs sufficient initialization to define partitioning controls (min and max processors per iterator server)
This is a convenience function for computing the minimum and maximum partition size prior to concurrent iterator partitioning.
References ProblemDescDB::get_iterator(), IteratorScheduler::schedPCLiter, and ParallelLevel::server_communicator_rank().
References by IteratorScheduler::configure(), ConcurrentMetaIterator::derived_init_communicators(), NestedModel::derived_init_communicators(), MetaIterator::estimate_by_name(), and MetaIterator::estimate_by_pointer().

IntIntPair configure ( ProblemDescDB & problem_db, const String & method_string, Iterator & sub_iterator, Model & sub_model )

performs sufficient initialization to define partitioning controls (min and max processors per iterator server)
This is a convenience function for computing the minimum and maximum partition size prior to concurrent iterator partitioning.
References IteratorScheduler::configure(), ProblemDescDB::get_iterator(), IteratorScheduler::schedPCLiter, and ParallelLevel::server_communicator_rank().
CHAPTER 14. CLASS DOCUMENTATION

**IntIntPair configure ( ProblemDescDB & problem_db, Iterator & sub_iterator )**

performs sufficient initialization to define partitioning controls (min and max processors per iterator server)

This is a convenience function for computing the minimum and maximum partition size prior to concurrent iterator partitioning.

References ParallelLibrary::bcast(), Iterator::estimate_partition_bounds(), ProblemDescDB::get_db_method_node(), ProblemDescDB::get_db_model_node(), IteratorScheduler::parallelLib, IteratorScheduler::schedPCIter, ParallelLevel::server_communicator_rank(), ParallelLevel::server_communicator_size(), ProblemDescDB::set_db_method_node(), ProblemDescDB::set_db_model_nodes(), and MPIPackBuffer::size().

**void partition ( int max_iterator_concurrency, IntIntPair & ppi )**

convenience function for initializing iterator communicators, setting parallel configuration attributes, and managing outputs and restart.

Called from derived class constructors once maxIteratorConcurrency is defined but prior to instantiating Iterators and Models.

References ParallelLibrary::init_iterator_communicators(), IteratorScheduler::iteratorScheduling, IteratorScheduler::numIteratorServers, ParallelLibrary::parallel_configuration_iterator(), IteratorScheduler::parallelLib, IteratorScheduler::procsPerIterator, ParallelLibrary::push_output_tag(), and IteratorScheduler::update().

Referenced by CollabHybridMetaIterator::derived_init_communicators(), EmbedHybridMetaIterator::derived_init_communicators(), ConcurrentMetaIterator::derived_init_communicators(), SeqHybridMetaIterator::derived_init_communicators(), and NestedModel::derived_init_communicators().

**void schedule_iterators ( MetaType & meta_object, Iterator & sub_iterator )**

short convenience function for distributing control among master_dynamic_schedule_iterators(), serve_iterators(), and peer_static_schedule_iterators()

This implementation supports the scheduling of multiple jobs using a single iterator/model pair. Additional future (overloaded) implementations could involve independent iterator instances.

References IteratorScheduler::iteratorScheduling, IteratorScheduler::iteratorServerId, IteratorScheduler::lead_rank(), IteratorScheduler::master_dynamic_schedule_iterators(), IteratorScheduler::numIteratorServers, ParallelLibrary::parallel_configuration_iterator(), IteratorScheduler::parallelLib, IteratorScheduler::peer_static_schedule_iterators(), IteratorScheduler::serve_iterators(), and IteratorScheduler::stop_iterator_servers().

Referenced by CollabHybridMetaIterator::core_run(), EmbedHybridMetaIterator::core_run(), ConcurrentMetaIterator::core_run(), NestedModel::derived_synchronize(), SeqHybridMetaIterator::run_sequential(), and NestedModel::serve_run().

**void master_dynamic_schedule_iterators ( MetaType & meta_object )**

executed by the scheduler master to manage a dynamic schedule of iterator jobs among slave iterator servers

This function is adapted from ApplicationInterface::master_dynamic_schedule_evaluations().

References ParallelLibrary::free(), ParallelLibrary::irecv_mti(), ParallelLibrary::isend_mti(), IteratorScheduler::miPLIndex, IteratorScheduler::numIteratorJobs, IteratorScheduler::numIteratorServers, IteratorScheduler::parallelLib, MPIPackBuffer::reset(), MPIUnpackBuffer::resize(), IteratorScheduler::resultsMsgLen, ParallelLibrary::waitall(), and ParallelLibrary::waitsome().

Referenced by IteratorScheduler::schedule_iterators().

**void serve_iterators ( MetaType & meta_object, Iterator & sub_iterator )**

executed on the slave iterator servers to perform iterator jobs assigned by the scheduler master

This function is similar in structure to ApplicationInterface::serve_evaluations_synch().
14.95 JEGAOptimizer Class Reference

A version of Dakota::Optimizer for instantiation of John Eddy’s Genetic Algorithms (JEGA).

Inheritance diagram for JEGAOptimizer:

```
    JEGAOptimizer
       |        |
       v        v
    Optimizer
       |        |
       v        v
    Minimizer
       |        |
       v        v
    Iterator
```

**Classes**

- class Driver
  
  A subclass of the JEGA front end driver that exposes the individual protected methods to execute the algorithm.

- class Evaluator
  
  An evaluator specialization that knows how to interact with Dakota.

- class EvaluatorCreator
  
  A specialization of the JEGA::FrontEnd::EvaluatorCreator that creates a new instance of a Evaluator.

**Public Member Functions**

- virtual void `core_run()`
  
  Performs the iterations to determine the optimal set of solutions.

- virtual bool `accepts_multiple_points()` const
  
  Overridden to return true since JEGA algorithms can accept multiple initial points.

- virtual bool `returns_multiple_points()` const
  
  Overridden to return true since JEGA algorithms can return multiple final points.

- virtual void `initial_points(const VariablesArray &pts)`
  
  Overridden to assign the _initPts member variable to the passed in collection of Dakota::Variables.

- virtual const VariablesArray & `initial_points()` const
CHAPTER 14. CLASS DOCUMENTATION

Overridden to return the collection of initial points for the JEGA algorithm created and run by this JEGAOptimizer.

- **JEGAOptimizer (ProblemDescDB &problem_db, Model &model)**
  
  Constructs a JEGAOptimizer class object.

- ∼JEGAOptimizer ()
  
  Destroys a JEGAOptimizer.

Protected Member Functions

- void LoadDakotaResponses (const JEGA::Utilities::Design &from, Variables &vars, Response &resp) const
  
  Loads the JEGA-style Design class into equivalent Dakota-style Variables and Response objects.

- void ReCreateTheParameterDatabase ()
  
  Destroys the current parameter database and creates a new empty one.

- void LoadTheParameterDatabase ()
  
  Reads information out of the known Dakota::ProblemDescDB and puts it into the current parameter database.

- void LoadAlgorithmConfig (JEGA::FrontEnd::AlgorithmConfig &aConfig)
  
  Completely initializes the supplied algorithm configuration.

- void LoadProblemConfig (JEGA::FrontEnd::ProblemConfig &pConfig)
  
  Completely initializes the supplied problem configuration.

- void LoadTheDesignVariables (JEGA::FrontEnd::ProblemConfig &pConfig)
  
  Adds DesignVariableInfo objects into the problem configuration object.

- void LoadTheObjectiveFunctions (JEGA::FrontEnd::ProblemConfig &pConfig)
  
  Adds ObjectiveFunctionInfo objects into the problem configuration object.

- void LoadTheConstraints (JEGA::FrontEnd::ProblemConfig &pConfig)
  
  Adds ConstraintInfo objects into the problem configuration object.

- void GetBestSolutions (const JEGA::Utilities::DesignOFSortSet &from, const JEGA::Algorithms::GeneticAlgorithm &theGA, std::multimap< RealRealPair, JEGA::Utilities::Design * > &designSortMap)
  
  Returns up to numBest designs sorted by DAKOTA’s fitness (L2 constraint violation, then utopia or objective), taking into account the algorithm type. The front of the returned map can be viewed as a single ”best”.

- void GetBestMOSolutions (const JEGA::Utilities::DesignOFSortSet &from, const JEGA::Algorithms::GeneticAlgorithm &theGA, std::multimap< RealRealPair, JEGA::Utilities::Design * > &designSortMap)
  
  Retreive the best Designs from a set of solutions assuming that they are generated by a multi objective algorithm.

- void GetBestSOSolutions (const JEGA::Utilities::DesignOFSortSet &from, const JEGA::Algorithms::GeneticAlgorithm &theGA, std::multimap< RealRealPair, JEGA::Utilities::Design * > &designSortMap)
  
  Retreive the best Designs from a set of solutions assuming that they are generated by a single objective algorithm.

- JEGA::DoubleMatrix ToDoubleMatrix (const VariablesArray &variables) const
  
  Converts the items in a VariablesArray into a DoubleMatrix whereby the items in the matrix are the design variables.

Private Attributes

- **EvaluatorCreator * _theEvalCreator**
  
  A pointer to an EvaluatorCreator used to create the evaluator used by JEGA in Dakota (a JEGAEvaluator).

- JEGA::Utilities::ParameterDatabase * _theParamDB
  
  A pointer to the ParameterDatabase from which all parameters are retrieved by the created algorithms.

- VariablesArray _initPts
  
  An array of initial points to use as an initial population.
Additional Inherited Members

14.95.1 Detailed Description

A version of Dakota::Optimizer for instantiation of John Eddy’s Genetic Algorithms (JEGA).

This class encapsulates the necessary functionality for creating and properly initializing the JEGA algorithms (MOGA and SOGA).

14.95.2 Constructor & Destructor Documentation

JEGAOptimizer ( ProblemDescDB & problem_db, Model & model )

Constructs a JEGAOptimizer class object.

This method does some of the initialization work for the algorithm. In particular, it initialized the JEGA core.

Parameters

<table>
<thead>
<tr>
<th>problem_db</th>
<th>The Dakota::ProblemDescDB with information on how the algorithm controls should be set.</th>
</tr>
</thead>
<tbody>
<tr>
<td>model</td>
<td>The Dakota::Model that will be used by this optimizer for problem information, etc.</td>
</tr>
</tbody>
</table>

References JEGAOptimizer::theEvalCreator, ProblemDescDB::get_int(), ProblemDescDB::get_short(), Iterator::iteratedModel, JEGAOptimizer::LoadTheParameterDatabase(), Iterator::maxEvalConcurrency, Iterator::method_Name, Iterator::numFinalSolutions, and Iterator::probDescDB.

14.95.3 Member Function Documentation

void LoadDakotaResponses ( const JEGA::Utilities::Design & from, Dakota::Variables & vars, Dakota::Response & resp ) const [protected]

Loads the JEGA-style Design class into equivalent Dakota-style Variables and Response objects.

This version is meant for the case where a Variables and a Response object exist and just need to be loaded.

Parameters

<table>
<thead>
<tr>
<th>from</th>
<th>The JEGA Design class object from which to extract the variable and response information for Dakota.</th>
</tr>
</thead>
<tbody>
<tr>
<td>vars</td>
<td>The Dakota::Variables object into which to load the design variable values of from.</td>
</tr>
<tr>
<td>resp</td>
<td>The Dakota::Response object into which to load the objective function and constraint values of from.</td>
</tr>
</tbody>
</table>

References Variables::continuous_variables(), Variables::discrete_int_variables(), Variables::discrete_real_variables(), Variables::discrete_string_variable(), Response::function_values(), and Dakota::set_index_to_value().

void LoadTheParameterDatabase ( ) [protected]

Reads information out of the known Dakota::ProblemDescDB and puts it into the current parameter database.

This should be called from the JEGAOptimizer constructor since it is the only time when the problem description database is certain to be configured to supply data for this optimizer.

Referenced by JEGAOptimizer::JEGAOptimizer().

void LoadAlgorithmConfig ( JEGA::FrontEnd::AlgorithmConfig & aConfig ) [protected]

Completely initializes the supplied algorithm configuration.

This loads the supplied configuration object with appropriate data retrieved from the parameter database.
CHAPTER 14. CLASS DOCUMENTATION

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>aConfig</td>
<td>The algorithm configuration object to load.</td>
</tr>
</tbody>
</table>

`void LoadProblemConfig ( JEGA::FrontEnd::ProblemConfig & pConfig ) [protected]`

Completely initializes the supplied problem configuration. This loads the fresh configuration object using the LoadTheDesignVariables, LoadTheObjectiveFunctions, and LoadTheConstraints methods.

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>pConfig</td>
<td>The problem configuration object to load.</td>
</tr>
</tbody>
</table>

`void LoadTheDesignVariables ( JEGA::FrontEnd::ProblemConfig & pConfig ) [protected]`

Adds DesignVariableInfo objects into the problem configuration object. This retrieves design variable information from the ParameterDatabase and creates DesignVariableInfo’s from it.

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>pConfig</td>
<td>The problem configuration object to load.</td>
</tr>
</tbody>
</table>

`void LoadTheObjectiveFunctions ( JEGA::FrontEnd::ProblemConfig & pConfig ) [protected]`

Adds ObjectiveFunctionInfo objects into the problem configuration object. This retrieves objective function information from the ParameterDatabase and creates ObjectiveFunctionInfo’s from it.

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>pConfig</td>
<td>The problem configuration object to load.</td>
</tr>
</tbody>
</table>

`void LoadTheConstraints ( JEGA::FrontEnd::ProblemConfig & pConfig ) [protected]`

Adds ConstraintInfo objects into the problem configuration object. This retrieves constraint function information from the ParameterDatabase and creates ConstraintInfo’s from it.

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>pConfig</td>
<td>The problem configuration object to load.</td>
</tr>
</tbody>
</table>

References Dakota::asstring(), and Dakota::copy_row_vector().

`void GetBestSolutions ( const JEGA::Utilities::DesignOFSortSet & from, const JEGA::Algorithms::GeneticAlgorithm & theGA, std::multimap< RealRealPair, JEGA::Utilities::DesignOFSortSet::const_iterator > & designSortMap ) [protected]`

Returns up to `numBest` designs sorted by DAKOTA’s fitness (L2 constraint violation, then utopia or objective), taking into account the algorithm type. The front of the returned map can be viewed as a single "best".
Parameters

<table>
<thead>
<tr>
<th>from</th>
<th>The full set of designs returned by the solver.</th>
</tr>
</thead>
<tbody>
<tr>
<td>theGA</td>
<td>The GA used to generate this set; needed for its weights in the SO case, provided to both for consistency</td>
</tr>
<tr>
<td>designSortMap</td>
<td>Map of best solutions with key pair &lt;constraintViolation, fitness&gt;</td>
</tr>
</tbody>
</table>

eventually this functionality must be moved into a separate post-processing application for MO datasets.

```cpp
def GetBestMOSolutions(const JEGA::Utilities::DesignOFSortSet & from, const JEGA::Algorithms::GeneticAlgorithm & theGA, std::multimap<RealRealPair, JEGA::Utilities::Design *> & designSortMap) [protected]
```
Retrieves the best Designs from a set of solutions assuming that they are generated by a multi objective algorithm.

eventually this functionality must be moved into a separate post-processing application for MO datasets.

```cpp
def GetBestSOSolutions(const JEGA::Utilities::DesignOFSortSet & from, const JEGA::Algorithms::GeneticAlgorithm & theGA, std::multimap<RealRealPair, JEGA::Utilities::Design *> & designSortMap) [protected]
```
Retrieves the best Designs from a set of solutions assuming that they are generated by a single objective algorithm.

eventually this functionality must be moved into a separate post-processing application for MO datasets.

References Dakota::abort handler().

```cpp
def JEGA::DoubleMatrix ToDoubleMatrix(const VariablesArray & variables) const [protected]
```
Converts the items in a VariablesArray into a DoubleMatrix whereby the items in the matrix are the design variables.

The matrix will not contain responses but when being used by Dakota, this doesn’t matter. JEGA will attempt to re-evaluate these points but Dakota will recognize that they do not require re-evaluation and thus it will be a cheap operation.

Parameters

| variables | The array of DakotaVariables objects to use as the contents of the returned matrix. |

Returns

The matrix created using the supplied VariablesArray.

```cpp
def core_run() [virtual]
```
Performs the iterations to determine the optimal set of solutions.

Override of pure virtual method in Optimizer base class.

The extraction of parameter values actually occurs in this method when the JEGA::FrontEnd::Driver::ExecuteAlgorithm is called. Also the loading of the problem and algorithm configurations occurs in this method. That way, if it is called more than once and the algorithm or problem has changed, it will be accounted for.

Reimplemented from Iterator.

References JEGAOptimizer::Driver::DestroyAlgorithm(), JEGAOptimizer::Driver::ExtractAllData(), and JEGAOptimizer::Driver::PerformIterations().
bool accepts_multiple_points( ) const [virtual]
Overridden to return true since JEGA algorithms can accept multiple initial points.

Returns
true, always.

Reimplemented from Iterator.

bool returns_multiple_points( ) const [virtual]
Overridden to return true since JEGA algorithms can return multiple final points.

Returns
true, always.

Reimplemented from Iterator.

void initial_points( const VariablesArray & pts ) [virtual]
Overridden to assign the _initPts member variable to the passed in collection of Dakota::Variables.

Parameters

| pts | The array of initial points for the JEGA algorithm created and run by this JEGAOptimizer. |

Reimplemented from Iterator.

const VariablesArray & initial_points( ) const [virtual]
Overridden to return the collection of initial points for the JEGA algorithm created and run by this JEGA-Optimizer.

Returns
The collection of initial points for the JEGA algorithm created and run by this JEGAOptimizer.

Reimplemented from Iterator.

14.95.4 Member Data Documentation

VariablesArray _initPts [private]
An array of initial points to use as an initial population.

This member is here to help support the use of JEGA algorithms in Dakota strategies. If this array is populated, then whatever initializer is specified will be ignored and the DoubleMatrix initializer will be used instead on a matrix created from the data in this array.

The documentation for this class was generated from the following files:

- JEGAOptimizer.hpp
- JEGAOptimizer.cpp
14.96 JEGATraits Class Reference

A version of TraitsBase specialized for John Eddy’s Genetic Algorithms (JEGA).

Inheritance diagram for JEGATraits:

```
  TraitsBase
   ^
   |  TraitsBase
   |
  JEGATraits
```

Public Member Functions

- **JEGATraits ()**
  
  *default constructor*

- **virtual ~JEGATraits ()**
  
  *destructor*

- **virtual bool is_derived ()**
  
  A temporary query used in the refactor.

- **bool supports_continuous_variables ()**
  
  *Return the flag indicating whether method supports continuous variables.*

- **bool supports_discrete_variables ()**
  
  *Return the flag indicating whether method supports continuous variables.*

- **bool supports_linear_equality ()**
  
  *Return the flag indicating whether method supports linear equalities.*

- **bool supports_linear_inequality ()**
  
  *Return the flag indicating whether method supports linear inequalities.*

- **bool supports_nonlinear_equality ()**
  
  *Return the flag indicating whether method supports nonlinear equalities.*

- **bool supports_nonlinear_inequality ()**
  
  *Return the flag indicating whether method supports nonlinear inequalities.*

- **NONLINEAR_INEQUALITY_FORMAT nonlinear_inequality_format ()**
  
  *Return the format used for nonlinear inequality constraints.*

14.96.1 Detailed Description

A version of TraitsBase specialized for John Eddy’s Genetic Algorithms (JEGA).

The documentation for this class was generated from the following file:

- **JEGAOptimizer.hpp**

14.97 LabelsWriter Class Reference

Utility used in derived write_core to write labels in tabular format.
Public Member Functions

- template<typename ArrayType >
  
  void operator() (std::ostream &s, size_t start_index, size_t num_items, const ArrayType &array_data, StringMultiArrayConstView label_array)

14.97.1 Detailed Description

Utility used in derived write_core to write labels in tabular format.

14.97.2 Member Function Documentation

void operator() ( std::ostream &s, size_t start_index, size_t num_items, const ArrayType & array_data, StringMultiArrayConstView label_array ) [inline]

The tabular labels writer only forwards the label arrays.

The documentation for this class was generated from the following file:

- DakotaVariables.hpp

14.98 LeastSq Class Reference

Base class for the nonlinear least squares branch of the iterator hierarchy.

Inheritance diagram for LeastSq:

```
  Iterator
    Minimizer
      LeastSq
        NL2SOLLeastSq NLSSOLLeastSq SNLLLeastSq
```

Protected Member Functions

- LeastSq (std::shared_ptr<TraitsBase> traits)
  default constructor
- LeastSq (ProblemDescDB &problem_db, Model &model, std::shared_ptr<TraitsBase> traits)
  standard constructor
- LeastSq (unsigned short method_name, Model &model, std::shared_ptr<TraitsBase> traits)
  alternate "on the fly" constructor
- ~LeastSq ()
  destructor
- void initialize_run ()
- void post_run (std::ostream &s)
• void finalize_run ()
  utility function to perform common operations following post_run(); deallocation and resetting of instance pointers
• void print_results (std::ostream &s, short results_state=FINAL_RESULTS)
• void get_confidence_intervals (const Variables &native_vars, const Response &iter_resp)
  Calculate confidence intervals on estimated parameters.

Protected Attributes
• size_t numLeastSqTerms
  number of least squares terms
• LeastSq * prevLSqInstance
  pointer containing previous value of leastSqInstance
• bool weightFlag
  flag indicating whether weighted least squares is active
• RealVector confBoundsLower
  lower bounds for confidence intervals on calibration parameters
• RealVector confBoundsUpper
  upper bounds for confidence intervals on calibration parameters
• RealVector bestIterPriFns
  storage for iterator best primary functions (which shouldn’t be stored in bestResponseArray when there are transformations)
• bool retrievedIterPriFns
  whether final primary iterator space functions have been retrieved (possibly by a derived class)

Static Protected Attributes
• static LeastSq * leastSqInstance
  pointer to LeastSq instance used in static member functions

Private Member Functions
• void weight_model ()
  Wrap iteratedModel in a RecastModel that weights the residuals.
• void archive_best_results ()
  top-level archival method

Additional Inherited Members
14.98.1 Detailed Description
Base class for the nonlinear least squares branch of the iterator hierarchy.

The LeastSq class provides common data and functionality for least squares solvers (including NL2OL, NLSSOLLeastSq, and SNLLLeastSq.
14.98.2 Constructor & Destructor Documentation

LeastSq (ProblemDescDB & problem_db, Model & model, std::shared_ptr<TraitsBase> traits) [protected]

standard constructor

This constructor extracts the inherited data for the least squares branch and performs sanity checking on gradient and constraint settings.

References Dakota::abort_handler(), Iterator::bestVariablesArray, Minimizer::calibrationDataFlag, Variables::copy(), Model::current_variables(), Minimizer::data_transform_model(), Iterator::iteratedModel, Iterator::method_name, LeastSq::numLeastSqTerms, Minimizer::numTotalCalibTerms, Minimizer::optimizationFlag, Model::primary_fn_type(), Minimizer::scale_model(), Minimizer::scaleFlag, LeastSq::weight_model(), and LeastSq::weightFlag.

14.98.3 Member Function Documentation

void initialize_run() [protected], [virtual]

This function should be invoked (or reimplemented) by any derived implementations of initialize_run() (which would otherwise hide it).

Reimplemented from Iterator.
Reimplemented in SNLLLeastSq.

References LeastSq::bestIterPriFns, Minimizer::initialize_run(), Iterator::iteratedModel, LeastSq::leastSqInstance, Iterator::myModelLayers, LeastSq::prevLSqInstance, LeastSq::retrievedIterPriFns, and Model::update from subordinate_model().

Referenced by SNLLLeastSq::initialize_run().

void post_run(std::ostream & s) [protected], [virtual]

Implements portions of post_run specific to LeastSq for scaling back to native variables and functions. This function should be invoked (or reimplemented) by any derived implementations of post_run() (which would otherwise hide it).

Reimplemented from Iterator.
Reimplemented in SNLLLeastSq.

References Dakota::abort_handler(), Iterator::activeSet, LeastSq::bestIterPriFns, Iterator::bestResponseArray, Iterator::bestVariablesArray, Minimizer::calibrationDataFlag, Variables::continuous_variables(), Model::continuous_variables(), Response::copy(), Variables::copy(), Dakota::copy_data_partial(), Model::current_response(), ScalingModel::cv_scaled2native(), Model::db_lookup(), Model::evaluate(), Response::function_gradients(), Response::function_values(), Response::function_values_view(), LeastSq::get_confidence_intervals(), Iterator::iteratedModel, Model::model_rep(), Minimizer::numContinuousVars, LeastSq::numLeastSqTerms, Minimizer::numNonlinearConstraints, Minimizer::numUserPrimaryFns, Minimizer::original_model(), Iterator::outputLevel, Minimizer::post_run(), ActiveSet::request_value(), ActiveSet::request_values(), ActiveSet::request_vector(), LeastSq::retrievedIterPriFns, Minimizer::scaleFlag, Minimizer::scalingModel, Minimizer::vendorNumericalGradFlag, and LeastSq::weightFlag.

void finalize_run() [inline], [protected], [virtual]

utility function to perform common operations following post_run(); deallocation and resetting of instance pointers

Optional: perform finalization phases of run sequence, like deallocating memory and resetting instance pointers. Commonly used in sub-iterator executions. This is a virtual function; when re-implementing, a derived class must call its nearest parent’s finalize_run(), typically after performing its own implementation steps.

Reimplemented from Iterator.
Reimplemented in SNLLLeastSq.

References Minimizer::finalize_run(), LeastSq::leastSqInstance, and LeastSq::prevLSqInstance.
void print_results ( std::ostream & s, short results_state = FINAL_RESULTS ) [protected], [virtual]

Redefines default iterator results printing to include nonlinear least squares results (residual terms and constraints).
Reimplemented from Iterator.

References Iterator::activeSet, Iterator::bestResponseArray, Iterator::bestVariablesArray, Minimizer::calibrationDataFlag, LeastSq::confBoundsLower, LeastSq::confBoundsUpper, ExperimentData::config_vars(), Model::continuous_variable_labels(), Dakota::data_pairs, Minimizer::dataTransformModel, Minimizer::expData, Model::interface_id(), Iterator::iteratedModel, Dakota::lookup_by_val(), Model::model_rep(), ExperimentData::num_experiments(), Minimizer::numContinuousVars, Minimizer::numNonlinearConstraints, Minimizer::numUserPrimaryFns, Minimizer::original_model(), DataTransformModel::print_best_responses(), Minimizer::print_residuals(), ActiveSet::request_values(), Model::response_size(), Minimizer::scaleFlag, LeastSq::weightFlag, Variables::write(), and Dakota::write_precision.

void get_confidence_intervals ( const Variables & native_vars, const Response & iter_resp ) [protected]

Calculate confidence intervals on estimated parameters.
Calculate individual confidence intervals for each parameter, based on a linear approximation of the nonlinear model. native_cv are needed for transformations and final reporting. iter_resp must contain the final differenced, scaled, weighted residuals and gradients.

References LeastSq::confBoundsLower, LeastSq::confBoundsUpper, Variables::continuous_variables(), Response::copy(), Response::function_gradients_view(), Response::function_values(), Model::model_rep(), Minimizer::numContinuousVars, LeastSq::numLeastSqTerms, ScalingModel::response_modify_s2n(), Minimizer::scaleFlag, Minimizer::scalingModel, and Minimizer::vendorNumericalGradFlag.

Referenced by LeastSq::post_run().

void weight_model( ) [private]

Wrap iteratedModel in a RecastModel that weights the residuals.
Setup Recast for weighting model. The weighting transformation doesn’t resize, and makes no vars, active set or secondary mapping. All indices are one-to-one mapped (no change in counts).

References Dakota::abort_handler(), Model::assign_rep(), Iterator::iteratedModel, Iterator::myModelLayers, Iterator::outputLevel, and Model::primary_response_fn_weights().

Referenced by LeastSq::LeastSq().

The documentation for this class was generated from the following files:

- DakotaLeastSq.hpp
- DakotaLeastSq.cpp

14.99 LibraryEnvironment Class Reference

Environment corresponding to execution as an embedded library.

Inheritance diagram for LibraryEnvironment:
Public Member Functions

- **LibraryEnvironment ()**
  
  *default constructor*

- **LibraryEnvironment (ProgramOptions prog_opts, bool check_bcast_construct=true, DbCallbackFunctionPtr callback=NULL, void *callback_data=NULL)**

  *Primary constructor: program options typically specifies an input file or input string. Optionally specify a callback function to be invoked after parsing. Set check_bcast_construct if performing late updates and later calling done_modifying_db().*

- **LibraryEnvironment (MPI_Comm dakota_mpi_comm, ProgramOptions prog_opts=ProgramOptions(), bool check_bcast_construct=true, DbCallbackFunctionPtr callback=NULL, void *callback_data=NULL)**

  *Alternate constructor accepting communicator, same options as primary.*

- **∼LibraryEnvironment ()**

  *destructor*

- **void insert_nodes (Dakota::DataMethod &dme, Dakota::DataModel &dmo, Dakota::DataVariables &dv, Dakota::DataInterface &di, Dakota::DataResponses &dr)**

  *Insert DB nodes for a {Method,Model,Variables,Interface,Responses} set.*

- **void done_modifying_db ()**

  *Check database contents, broadcast, and construct iterators.*

- **bool plugin_interface (const String &model_type, const String &interf_type, const String &an_driver, Interface *plugin_iface)**

  *Plug-in the passed interface into any interface matching the specified (possibly empty) model, interface, and driver strings; returns true if a plugin was performed.*

- **InterfaceList filtered_interface_list (const String &interf_type, const String &an_driver)**

  *Filter the available Interface instances based on matching interface type and analysis drivers (empty String matches any)*

- **ModelList filtered_model_list (const String &model_type, const String &interf_type, const String &an_driver)**

  *Filter the available Model instances based on matching model type, interface type, and analysis drivers (empty String matches any)*

Additional Inherited Members

14.99.1 Detailed Description

**Environment** corresponding to execution as an embedded library.

This environment corresponds to use of Dakota as a library within another application, e.g., within library_mode.cpp. It sets up the ParallelLibrary and ProblemDescDB objects without access to command line arguments.
14.99. Constructor & Destructor Documentation

LibraryEnvironment ( ProgramOptions prog_opts, bool check_bcast_construct = true, DbCallbackFunctionPtr callback = NULL, void * callback_data = NULL )

Primary constructor: program options typically specifies an input file or input string. Optionally specify a callback function to be invoked after parsing. Set check_bcast_construct if performing late updates and later calling done_modifying_db().

Construct library environment, optionally performing check/bcast of database and iterator construction

References Environment::construct(), OutputManager::output_startup_message(), Environment::outputManager, and Environment::parse().

LibraryEnvironment ( MPI_Comm dakota_mpi_comm, ProgramOptions prog_opts = ProgramOptions (), bool check_bcast_construct = true, DbCallbackFunctionPtr callback = NULL, void * callback_data = NULL )

Alternate constructor accepting communicator, same options as primary.

Construct library environment on passed MPI Comm, optionally performing check/bcast of database and iterator construction. MPI Comm is first argument so client doesn’t have to pass all args

References Environment::construct(), OutputManager::output_startup_message(), Environment::outputManager, and Environment::parse().

14.99.3 Member Function Documentation

InterfaceList filtered_interface_list ( const String & interf_type, const String & an_driver )

filter the available Interface instances based on matching interface type and analysis drivers (empty String matches any)

This convenience function helps clients locate and plugin to the right Interface instance for simple cases. Pass an empty string to match any instead of a specific instance

References Interface::analysis_drivers(), Dakota::contains(), Interface::interface_type(), ProblemDescDB::model_list(), and Environment::probDescDB.

ModelList filtered_model_list ( const String & model_type, const String & interf_type, const String & an_driver )

filter the available Model instances based on matching model type, interface type, and analysis drivers (empty String matches any)

This convenience function helps clients locate and plugin to the right Interface instance for cases where the parallel configuration is needed in constructing a parallel plugin. Pass an empty string to match any instead of a specific instance

References Interface::analysis_drivers(), Dakota::contains(), Interface::interface_type(), ProblemDescDB::model_list(), and Environment::probDescDB.

Referenced by parallel_interface_plugin(), LibraryEnvironment::plugin_interface(), run_dakota(), and run_dakota_mixed().

The documentation for this class was generated from the following files:

- LibraryEnvironment.hpp
- LibraryEnvironment.cpp
14.100 LightWtBaseConstructor Struct Reference

Dummy struct for overloading constructors used in on-the-fly Model instantiations.

Public Member Functions

- LightWtBaseConstructor (int=0)
  
  \textit{C++ structs can have constructors.}

14.100.1 Detailed Description

Dummy struct for overloading constructors used in on-the-fly Model instantiations.
  
  \textbf{LightWtBaseConstructor} is used to overload the constructor used for on-the-fly Model instantiations. Putting this struct here avoids circular dependencies.
  
  The documentation for this struct was generated from the following file:
  
  - dakota\_global\_defs.hpp

14.101 LinearSolverBase Class Reference

The \texttt{LinearSolverBase} class serves as an API for derived solvers.
  
  Inheritance diagram for LinearSolverBase:

\begin{center}
\begin{tikzpicture}
  \node[concept] {LinearSolverBase};
  \node[concept, below of=LinearSolverBase] {CholeskySolver};
\end{tikzpicture}
\end{center}

Public Types

- enum SOLVER\_TYPE {
  
  \texttt{CHOLESKY, EQ\_CONS\_LEAST\_SQ\_REGRESSION, LASSO\_REGRESSION, LEAST\_ANGLE\_REGRESSION, LU, ORTHOG\_MATCH\_PURSUIT, QR\_LEAST\_SQ\_REGRESSION, SVD\_LEAST\_SQ\_REGRESSION} }

  \textit{How best to Doxygenate class enums? RWH.}

Public Member Functions

- LinearSolverBase ()
  
  \texttt{Constructor.}

- \~LinearSolverBase ()
  
  \texttt{Destructor.}

- virtual bool is\_factorized () const
  
  \texttt{Query to determine if the matrix of the solver has been factored.}

- virtual void factorize (const MatrixXd &mat)
Perform the matrix factorization for the linear solver matrix.

- virtual void solve (const MatrixXd &lhs, const MatrixXd &rhs, MatrixXd &x)
  Find a solution to linear problem.
- virtual void solve (const MatrixXd &rhs, MatrixXd &x)
  Find a solution to linear problem where the LHS is already factorized.

Static Public Member Functions

- static SOLVER_TYPE solver_type (const std::string &solver_name)
  Convert solver name to enum type.

### 14.101.1 Detailed Description

The LinearSolverBase class serves as an API for derived solvers.

### 14.101.2 Member Function Documentation

**SOLVER_TYPE solver_type (const std::string &solver_name) [static]**

Convert solver name to enum type.

**Parameters**

<table>
<thead>
<tr>
<th>in</th>
<th>solver_name</th>
<th>LinearSolverBase name to map</th>
</tr>
</thead>
</table>

**Returns**

Corresponding LinearSolverBase enum

References dakota::util::type_name_bimap.

Referenced by PolynomialRegression::build().

**void factorize (const MatrixXd &mat) [virtual]**

Perform the matrix factorization for the linear solver matrix.

**Parameters**

<table>
<thead>
<tr>
<th>in</th>
<th>mat</th>
<th>The incoming matrix to factorize.</th>
</tr>
</thead>
</table>

Reimplemented in CholeskySolver.

References dakota::silence_unused_args().

**void solve (const MatrixXd &lhs, const MatrixXd &rhs, MatrixXd &x) [virtual]**

Find a solution to linear problem.

**Parameters**

<table>
<thead>
<tr>
<th>in</th>
<th>lhs</th>
<th>The linear system left-hand-side matrix.</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>rhs</td>
<td>The linear system right-hand-side (multi-)vector.</td>
</tr>
</tbody>
</table>
The linear system solution (multi-)vector.

Reimplemented in CholeskySolver.
References dakota::silence_unused_args().

void solve ( const MatrixXd & rhs, MatrixXd & x ) [virtual]

Find a solution to linear problem where the LHS is already factorized.

Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>rhs</th>
<th>The linear system right-hand-side (multi-)vector.</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>x</td>
<td>The linear system solution (multi-)vector.</td>
</tr>
</tbody>
</table>

Reimplemented in CholeskySolver.
References dakota::silence_unused_args().

The documentation for this class was generated from the following files:

- LinearSolvers.hpp
- LinearSolvers.cpp

### 14.102 MatchesWC Struct Reference

Predicate that returns true when the passed path matches the wild_card with which it was configured. Currently supports * and ?.

#### Public Member Functions

- **MatchesWC (const bfs::path & wild_card)**
  
  *ctor that builds and stores the regular expression*

- **bool operator() (const bfs::path & dir_entry)**
  
  *return true is dir_entry matches wildCardRegEx*

#### Public Attributes

- **boost::basic_regex**
  
  `<bfs::path::value_type> wildCardRegEx`

  *archived RegEx; wchar-based on Windows*

### 14.102.1 Detailed Description

Predicate that returns true when the passed path matches the wild_card with which it was configured. Currently supports * and ?.

The documentation for this struct was generated from the following file:

- WorkdirHelper.hpp
14.103 MatlabInterface Class Reference

Inheritance diagram for MatlabInterface:

```
      Interface
       /   \
  ApplicationInterface
      /   \
DirectApplicInterface
      \    
MatlabInterface
```

Public Member Functions

- `MatlabInterface (const ProblemDescDB &problem_db)`
  Constructor: start Matlab engine.
- `~MatlabInterface ()`
  Destructor: close Matlab engine.

Protected Member Functions

- `virtual int derived_map_ac (const String &ac_name)`
  execute an analysis code portion of a direct evaluation invocation
- `int matlab_engine_run (const Dakota::String &ac_name)`
  Helper function supporting derived_map_ac. Sends data to Matlab, executes analysis, collects return data.
- `int matlab_field_prep (mxArray *dakota_matlab, const char *field_name)`
  check that the dakota_matlab structure has the specified field_name and add if necessary; free structure memory in preparation for new alloc

Protected Attributes

- `engine * matlabEngine`
  pointer to the MATLAB engine used for direct evaluations

14.103.1 Detailed Description

Specialization of DirectApplicInterface to link to Matlab analysis drivers. Includes convenience functions to map data to/from Matlab

14.103.2 Member Function Documentation

`int derived_map_ac (const String &ac_name)` [protected], [virtual]

execute an analysis code portion of a direct evaluation invocation
Matlab specialization of dervied analysis components.
References ApplicationInterface::analysisServerId, and MatlabInterface::matlab_engine_run().
int matlab_engine_run ( const Dakota::String & ac_name ) [protected]

Helper function supporting derived_map_ac. Sends data to Matlab, executes analysis, collects return data.

Direct interface to Matlab through Mathworks external API. m-file executed is specified through analysis._drivers, extra strings through analysis_components. (Original BMA 11/28/2005)

Special thanks to Lee Peterson for substantial enhancements 12/15/2007: Added output buffer for the MATLAB command response and error messages Made the Dakota variable persistent in the MATLAB engine workspace Added robustness to the user deleting required Dakota fields

References Dakota::abort_handler(), Interface::analysisComponents, DirectApplicInterface::analysisDriverIndex, Interface::currEvalId, DirectApplicInterface::directFnASV, DirectApplicInterface::directFnDVV, Dakota::FIELD_NAMES, DirectApplicInterface::fnGrads, DirectApplicInterface::fnHessians, Interface::fnLabels, DirectApplicInterface::fnVals, DirectApplicInterface::gradFlag, DirectApplicInterface::hessFlag, MatlabInterface::matlab_field_prep(), MatlabInterface::matlabEngine, DirectApplicInterface::numACV, DirectApplicInterface::numADIV, DirectApplicInterface::numADR, Dakota::NUMBER_OF_FIELDS, DirectApplicInterface::numDerivVars, DirectApplicInterface::numFns, DirectApplicInterface::numVars, Interface::outputLevel, DirectApplicInterface::xC, DirectApplicInterface::xCLabels, DirectApplicInterface::xDI, DirectApplicInterface::xDILabels, DirectApplicInterface::xDRLabels.

Referenced by MatlabInterface::derived_map_ac().

The documentation for this class was generated from the following files:

- MatlabInterface.hpp
- MatlabInterface.cpp

### 14.104 MetaIterator Class Reference

Base class for meta-iterators.

Inheritance diagram for MetaIterator:

```
  Iterator
  |     |
  |     |
  MetaIterator
  |     |
  CollabHybridMetaIterator ConcurrentMetaIterator EmbedHybridMetaIterator SeqHybridMetaIterator
```

**Public Member Functions**

- `bool resize ()`

  `reinitializes iterator based on new variable size`

**Protected Member Functions**

- `MetaIterator (ProblemDescDB &problem_db)`

  `standard constructor`

- `MetaIterator (ProblemDescDB &problem_db, Model &model)`

  `alternate constructor`

- `~MetaIterator ()`
14.104. METAITTERATOR CLASS REFERENCE

**Destructor**

- void **post_run** (std::ostream &s)

  *post-run portion of run (optional); verbose to print results; re-implemented by Iterators that can read all Variables/Responses and perform final analysis phase in a standalone way*

- void **check_model** (const String &method_ptr, const String &model_ptr)

  *check that a model identified by pointer has the same id as the iteratedModel passed through the ctor chain*

- void **allocate_by_pointer** (const String &method_ptr, Iterator &the_iterator, Model &the_model)

  *initialize the iterator and the model based on method_ptr*

- void **allocate_by_name** (const String &method_string, const String &model_ptr, Iterator &the_iterator, Model &the_model)

  *initialize the iterator based on method_string*

- std::pair<int, int> **estimate_by_pointer** (const String &method_ptr, Iterator &the_iterator, Model &the_model)

  *estimate minimum and maximum processors per iterator needed for init_iterator_parallelism(); instantiates the_iterator and the_model as needed, but on minimal processor ranks (is later augmented by allocate_by_pointer())*

- std::pair<int, int> **estimate_by_name** (const String &method_string, const String &model_ptr, Iterator &the_iterator, Model &the_model)

  *estimate minimum and maximum processors per iterator needed for init_iterator_parallelism(); instantiates the_iterator and the_model as needed, but on minimal processor ranks (is later augmented by allocate_by_name())*

**Protected Attributes**

- IteratorScheduler iterSched
  *scheduler for concurrent execution of Iterators*

- int maxIteratorConcurrency
  *maximum number of concurrent sub-iterator executions*

**Additional Inherited Members**

14.104.1 Detailed Description

Base class for meta-iterators.

This base class shares code for concurrent and hybrid meta-iterators, where the former supports multi-start and Pareto set iteration and the latter supports sequential, embedded, and collaborative hybrids.

14.104.2 Member Function Documentation

void **post_run** (std::ostream &s) [protected], [virtual]

*post-run portion of run (optional); verbose to print results; re-implemented by Iterators that can read all Variables/Responses and perform final analysis phase in a standalone way*

Post-run phase, which a derived iterator may optionally reimplement; when not present, post-run is likely integrated into run. This is a virtual function; when re-implementing, a derived class must call its nearest parent’s **post_run()**, typically after performing its own implementation steps.

Reimplemented from Iterator.

References MetaIterator::iterSched, IteratorScheduler::lead_rank(), and Iterator::print_results().

The documentation for this class was generated from the following files:

- MetaIterator.hpp
- MetaIterator.cpp
14.105 Minimizer Class Reference

Base class for the optimizer and least squares branches of the iterator hierarchy. Inheritance diagram for Minimizer:

```
Minimizer
  Iterator
  LeastSq
    NL2SOLLeastSq
    NLSSOLLeastSq
    SNLLLeastSq
  Optimizer
    APPSOptimizer
    COLNDOptimizer
    CONMINOptimizer
    JEGAOptimizer
    NCSUOptimizer
    NonlinearCGOptimizer
    NOWPACOptimizer
    OptDartsOptimizer
    ROLOptimizer
    SNLLOptimizer
  SurrBasedMinimizer
    EffGlobalMinimizer
    SurrBasedGlobalMinimizer
    SurrBasedLocalMinimizer
```

Public Member Functions

- void `constraint_tolerance` (Real constr_tol)
  
  set the method constraint tolerance (constraintTol)

- Real `constraint_tolerance` () const
  
  return the method constraint tolerance (constraintTol)

- std::shared_ptr<TPLDataTransfer> `get_data_transfer_helper` () const

- bool `resize` ()
  
  reinitializes iterator based on new variable size

Static Public Member Functions

- static Real `sum_squared_residuals` (size_t num_pri_fns, const RealVector &residuals, const RealVector &weights)
  
  return weighted sum of squared residuals

- static void `print_residuals` (size_t num_terms, const RealVector &best_terms, const RealVector &weights, size_t num_best, size_t best_index, std::ostream &s)
  
  print num_terms residuals and misfit for final results

- static void `print_model_resp` (size_t num_pri_fns, const RealVector &best_fns, size_t num_best, size_t best_index, std::ostream &s)
  
  print the original user model resp in the case of data transformations
14.105. MINIMIZER CLASS REFERENCE

Protected Member Functions

- **Minimizer** (std::shared_ptr< TraitsBase > traits=std::shared_ptr< TraitsBase >(new TraitsBase()))
  default constructor
- **Minimizer** (ProblemDescDB &problem_db, Model &model, std::shared_ptr< TraitsBase > traits=std::shared_ptr< TraitsBase >(new TraitsBase()))
  standard constructor
- **Minimizer** (unsigned short method_name, Model &model, std::shared_ptr< TraitsBase > traits=std::shared_ptr< TraitsBase >(new TraitsBase()))
  alternate constructor for "on the fly" instantiations
- **Minimizer** (unsigned short method_name, size_t num_lin_ineq, size_t num_lin_eq, size_t num_nln_ineq, size_t num_nln_eq, std::shared_ptr< TraitsBase > traits=std::shared_ptr< TraitsBase >(new TraitsBase()))
  alternate constructor for "on the fly" instantiations
- **~Minimizer** ()
  destructor
- **void update_from_model** (const Model &model)
  set inherited data attributes based on extractions from incoming model
- **void initialize_run** ()
  utility function to perform common operations prior to pre_run(); typically memory initialization; setting of instance pointers
- **void post_run** (std::ostream &s)
  post-run portion of run (optional); verbose to print results; re-implemented by Iterators that can read all Variables/Responses and perform final analysis phase in a standalone way
- **void finalize_run** ()
  utility function to perform common operations following post_run(); deallocation and resetting of instance pointers
- **const Model & algorithm_space_model** () const
- **Model original_model** (unsigned short recasts_left=0) const
  Return a shallow copy of the original model this Iterator was originally passed, optionally leaving recasts_left on top of it.
- **void data_transform_model** ()
  Wrap iteratedModel in a RecastModel that subtracts provided observed data from the primary response functions (variables and secondary responses are unchanged)
- **void scale_model** ()
  Wrap iteratedModel in a RecastModel that performs variable and/or response scaling.
- **Real objective** (const RealVector &fn_vals, const BoolDeque &max_sense, const RealVector &primary_wts) const
  compute a composite objective value from one or more primary functions
- **Real objective** (const RealVector &fn_vals, size_t num_fns, const BoolDeque &max_sense, const RealVector &primary_wts) const
  compute a composite objective with specified number of source primary functions, instead of userPrimaryFns
- **void objective_gradient** (const RealVector &fn_vals, const RealMatrix &fn_grads, const BoolDeque &max_sense, const RealVector &primary_wts, RealVector &obj_grad) const
  compute the gradient of the composite objective function
- **void objective_gradient** (const RealVector &fn_vals, size_t num_fns, const RealMatrix &fn_grads, const BoolDeque &max_sense, const RealVector &primary_wts, RealVector &obj_grad) const
compute the gradient of the composite objective function

- `void objective_hessian` (const RealVector &fn_vals, const RealMatrix &fn_grads, const RealSymMatrixArray &fn_hessians, const BoolDeque &max_sense, const RealVector &primary_wts, RealSymMatrix &obj_hess) const

compute the Hessian of the composite objective function

- `void objective_hessian` (const RealVector &fn_vals, const RealMatrix &fn_grads, const RealSymMatrixArray &fn_hessians, const BoolDeque &max_sense, const RealVector &primary_wts, RealSymMatrix &obj_hess) const

- `compute the Hessian of the composite objective function`

- `virtual void archive_best_results ()` top-level archival method

- `void archive_best_variables` (const bool active_only=false) const

archive best variables for the index'th final solution

- `void archive_best_objective_functions ()` const

archive the index'th set of objective functions

- `void archive_best_constraints ()` const

archive the index'th set of constraints

- `void archive_best_residuals ()` const

Archive residuals when calibration terms are used.

- `void resize_best_vars_array` (size_t newsize)

Safely resize the best variables array to newsize taking into account the envelope-letter design pattern and any recasting.

- `void resize_best_resp_array` (size_t newsize)

Safely resize the best response array to newsize taking into account the envelope-letter design pattern and any recasting.

- `void local_recast_retrieve` (const Variables &vars, Response &response) const

infers MOO/NLS solution from the solution of a single-objective optimizer

Protected Attributes

- `size_t numFunctions`

  number of response functions

- `size_t numContinuousVars`

  number of active continuous vars

- `size_t numDiscreteIntVars`

  number of active discrete integer vars

- `size_t numDiscreteStringVars`

  number of active discrete string vars

- `size_t numDiscreteRealVars`

  number of active discrete real vars

- `Real constraintTol`

  optimizer/least squares constraint tolerance

- `Real bigRealBoundSize`

  cutoff value for inequality constraint and continuous variable bounds
- `int bigIntBoundSize`
  cutoff value for discrete variable bounds

- `size_t numNonlinearIneqConstraints`
  number of nonlinear inequality constraints

- `size_t numNonlinearEqConstraints`
  number of nonlinear equality constraints

- `size_t numLinearIneqConstraints`
  number of linear inequality constraints

- `size_t numLinearEqConstraints`
  number of linear equality constraints

- `size_t numNonlinearConstraints`
  total number of nonlinear constraints

- `size_t numLinearConstraints`
  total number of linear constraints

- `size_t numConstraints`
  total number of linear and nonlinear constraints

- `bool optimizationFlag`
  flag for use where optimization and NLS must be distinguished

- `size_t numUserPrimaryFns`
  number of objective functions or least squares terms in the inbound model; always initialize at Minimizer, even if overridden later

- `size_t numIterPrimaryFns`
  number of objective functions or least squares terms in iterator’s view, after transformations; always initialize at Minimizer, even if overridden later

- `bool boundConstraintFlag`
  convenience flag for denoting the presence of user-specified bound constraints. Used for method selection and error checking.

- `bool speculativeFlag`
  flag for speculative gradient evaluations

- `bool calibrationDataFlag`
  flag indicating whether user-supplied calibration data is active

- `ExperimentData expData`
  Container for experimental data to which to calibrate model using least squares or other formulations which minimize SSE.

- `size_t numExperiments`
  number of experiments

- `size_t numTotalCalibTerms`
  number of total calibration terms (sum over experiments of number of experimental data per experiment, including field data)

- `Model dataTransformModel`
  Shallow copy of the data transformation model, when present (cached in case further wrapped by other transformations)

- `bool scaleFlag`
  whether Iterator-level scaling is active
• Model scalingModel
  Shallow copy of the scaling transformation model, when present (cached in case further wrapped by other transformations)
• Minimizer * prevMinInstance
  pointer containing previous value of minimizerInstance
• bool vendorNumericalGradFlag
  convenience flag for gradient type == numerical && method source == vendor
• std::shared_ptr<TPLDataTransfer> dataTransferHandler
  Emerging helper class for handling data transfers to/from Dakota and the underlying TPL.

Static Protected Attributes
• static Minimizer * minimizerInstance
  pointer to Minimizer used in static member functions

Friends
• class SOLBase
  the SOLBase class is not derived the iterator hierarchy but still needs access to iterator hierarchy data (to avoid attribute replication)
• class SNLLBase
  the SNLLBase class is not derived the iterator hierarchy but still needs access to iterator hierarchy data (to avoid attribute replication)

Additional Inherited Members

14.105.1 Detailed Description
Base class for the optimizer and least squares branches of the iterator hierarchy.

The Minimizer class provides common data and functionality for Optimizer and LeastSq.

14.105.2 Constructor & Destructor Documentation
Minimizer ( ProblemDescDB & problem_db, Model & model, std::shared_ptr<TraitsBase> traits = std::shared_ptr<TraitsBase>(new TraitsBase()) ) [protected]

standard constructor
This constructor extracts inherited data for the optimizer and least squares branches and performs sanity checking on constraint settings.

References Iterator::iteratedModel, Iterator::maxIterations, Iterator::methodName, Iterator::numFinalSolutions, and Minimizer::update_from_model().

14.105.3 Member Function Documentation
void initialize_run( ) [protected], [virtual]
utility function to perform common operations prior to pre_run(); typically memory initialization; setting of instance pointers
Perform initialization phases of run sequence, like allocating memory and setting instance pointers. Commonly used in sub.iterator executions. This is a virtual function; when re-implementing, a derived class must call its nearest parent’s \texttt{initialize\_run()}, typically before performing its own implementation steps.

Reimplemented from \texttt{Iterator}.

Reimplemented in \texttt{Optimizer, SNLLOptimizer, SNLLLeastSq, and ROLOptimizer}.

References \texttt{Model::all\_continuous\_variables(), Model::all\_discrete\_int\_variables(), Model::all\_discrete\_real\_variables(), Iterator::bestVariablesArray, Model::initialize\_mapping(), Model::is\_null(), Iterator::iteratedModel, Iterator::method\_PCIter, Minimizer::minimizer\_Instance, Iterator::my\_Model\_Layers, Minimizer::prev\_Min\_Instance, Minimizer::resize(), Model::set\_evaluation\_reference(), Iterator::sub\_iterator\_Flag, Model::subordinate\_model(), and Iterator::summary\_output\_Flag.}

Referenced by \texttt{LeastSq::initialize\_run()}, and \texttt{Optimizer::initialize\_run()}.

\begin{verbatim}
void post\_run ( std::ostream & s ) [protected], [virtual]
\end{verbatim}

post-run portion of run (optional); verbose to print results; re-implemented by Iterators that can read all Variables/Responses and perform final analysis phase in a standalone way.

Post-run phase, which a derived iterator may optionally reimplement; when not present, post-run is likely integrated into run. This is a virtual function; when re-implementing, a derived class must call its nearest parent’s \texttt{post\_run()}, typically after performing its own implementation steps.

Reimplemented from \texttt{Iterator}.

Reimplemented in \texttt{Optimizer, SNLLOptimizer, and SurrBasedLocalMinimizer}.

References Minimizer::archive\_best\_results(), Model::is\_null(), Iterator::iteratedModel, Model::print\_evaluation\_summary(), Iterator::print\_results(), and Iterator::summary\_output\_Flag.

Referenced by \texttt{LeastSq::post\_run()}, \texttt{SurrBasedLocalMinimizer::post\_run()}, and \texttt{Optimizer::post\_run()}.\n
\begin{verbatim}
void finalize\_run ( ) [protected], [virtual]
\end{verbatim}

utility function to perform common operations following \texttt{post\_run()}; deallocation and resetting of instance pointers.

Optional: perform finalization phases of run sequence, like deallocating memory and resetting instance pointers. Commonly used in sub.iterator executions. This is a virtual function; when re-implementing, a derived class must call its nearest parent’s \texttt{finalize\_run()}, typically after performing its own implementation steps.

Reimplemented from \texttt{Iterator}.

Reimplemented in \texttt{Optimizer, SNLLOptimizer, and SNLLLeastSq}.

References \texttt{Model::finalize\_mapping()}, \texttt{Iterator::finalize\_run()}, \texttt{Iterator::iteratedModel}, \texttt{Minimizer::minimizer\_Instance, Minimizer::prev\_Min\_Instance}, and \texttt{Minimizer::resize().}

Referenced by \texttt{LeastSq::finalize\_run()}, and \texttt{Optimizer::finalize\_run()}.\n
\begin{verbatim}
const Model & algorithm\_space\_model( ) const [inline], [protected], [virtual]
\end{verbatim}

default definition that gets redefined in selected derived Minimizers.

Reimplemented from \texttt{Iterator}.

Reimplemented in \texttt{EffGlobalMinimizer}.

References \texttt{Iterator::iteratedModel}.

\begin{verbatim}
void data\_transform\_model ( ) [protected]
\end{verbatim}

Wrap \texttt{iteratedModel} in a \texttt{RecastModel} that subtracts provided observed data from the primary response functions (variables and secondary responses are unchanged)

Reads observation data to compute least squares residuals. Does not change size of responses, and is the first wrapper, therefore sizes are based on \texttt{iteratedModel}.
References Dakota::abort_handler(), Iterator::activeSet, Iterator::assign_rep(), Minimizer::dataTransformModel, Minimizer::expData, ProblemDescDB::get_size(), Iterator::iteratedModel, ExperimentData::load_data(), Iterator::myModelLayers, Model::num_primary_fns(), Minimizer::numExperiments, Minimizer::numFunctions, Minimizer::numIterPrimaryFns, Minimizer::numTotalCalibTerms, Iterator::outputLevel, Iterator::probDescDB, ActiveSet::request_vector(), and Model::response_size().

Referenced by LeastSq::LeastSq(), and Optimizer::Optimizer().

```cpp
void scale_model ( ) [protected]
```

Wrap iteratedModel in a RecastModel that performs variable and/or response scaling.

Wrap the iteratedModel in a scaling transformation, such that iteratedModel now contains a scaling recast model. Potentially affects variables, primary, and secondary responses

References Model::assign_rep(), Iterator::iteratedModel, Iterator::myModelLayers, and Minimizer::scaling-Model.

Referenced by LeastSq::LeastSq(), and Optimizer::Optimizer().

```cpp
Real objective ( const RealVector & fn_vals, const BoolDeque & max_sense, const RealVector & primary_wts ) const [protected]
```

compute a composite objective value from one or more primary functions

The composite objective computation sums up the contributions from one of more primary functions using the primary response fn weights.

References Minimizer::numUserPrimaryFns.

Referenced by SurrBasedLocalMinimizer::approx_subprob_objective_eval(), SurrBasedMinimizer::augmented_lagrangian_merit(), SurrBasedLocalMinimizer::compute_trust_region_ratio(), EffGlobalMinimizer::expected_improvement(), SurrBasedMinimizer::initialize_filter(), SurrBasedMinimizer::lagrangian_merit(), Optimizer::objective_reduction(), SurrBasedMinimizer::penalty_merit(), COLINOptimizer::post_run(), SurrBasedMinimizer::update_filter(), and SurrBasedLocalMinimizer::update_penalty().

```cpp
Real objective ( const RealVector & fn_vals, size_t num_fns, const BoolDeque & max_sense, const RealVector & primary_wts ) const [protected]
```

compute a composite objective with specified number of source primary functions, instead of userPrimaryFns

This "composite" objective is a more general case of the previous objective(), but doesn’t presume a reduction map from user to iterated space. Used to apply weights and sense in COLIN results sorting. Leaving as a duplicate implementation pending resolution of COLIN lookups.

References Minimizer::optimizationFlag.

```cpp
void objective_gradient ( const RealVector & fn_vals, size_t num_fns, const RealMatrix & fn_grads, const BoolDeque & max_sense, const RealVector & primary_wts, RealVector & obj_grad ) const [protected]
```

compute the gradient of the composite objective function

The composite objective gradient computation combines the contributions from one of more primary function gradients, including the effect of any primary function weights. In the case of a linear mapping (MOO), only the primary function gradients are required, but in the case of a nonlinear mapping (NLS), primary function values are also needed. Within RecastModel::set_mapping(), the active set requests are automatically augmented to make values available when needed, based on nonlinearRespMapping settings.

References Minimizer::numContinuousVars, and Minimizer::optimizationFlag.
void objective_hessian ( const RealVector & fn_vals, size_t num_fns, const RealMatrix & fn_grads, const RealSymMatrixArray & fn_hessians, const BoolDeque & max_sense, const RealVector & primary_wts, RealSymMatrix & obj_hess ) const  
  [protected]

compute the Hessian of the composite objective function

The composite objective Hessian computation combines the contributions from one of more primary function Hessians, including the effect of any primary function weights. In the case of a linear mapping (MOO), only the primary function Hessians are required, but in the case of a nonlinear mapping (NLS), primary function values and gradients are also needed in general (gradients only in the case of a Gauss-Newton approximation). Within the default RecastModel::set_mapping(), the active set requests are automatically augmented to make values and gradients available when needed, based on nonlinearRespMapping settings.

References Dakota::abort_handler(), Minimizer::numContinuousVars, and Minimizer::optimizationFlag.

void resize_best_vars_array ( size_t newsize )  
  [protected]

Safely resize the best variables array to newsize taking into account the envelope-letter design pattern and any recasting.

Uses data from the innermost model, should any Minimizer recasts be active. Called by multipoint return solvers. Do not directly call resize on the bestVariablesArray object unless you intend to share the internal content (letter) with other objects after assignment.

References Iterator::bestVariablesArray, Variables::copy(), Model::current_variables(), and Minimizer::original_model().

Referenced by COLINOptimizer::post_run().

void resize_best_resp_array ( size_t newsize )  
  [protected]

Safely resize the best response array to newsize taking into account the envelope-letter design pattern and any recasting.

Uses data from the innermost model, should any Minimizer recasts be active. Called by multipoint return solvers. Do not directly call resize on the bestResponseArray object unless you intend to share the internal content (letter) with other objects after assignment.

References Iterator::bestResponseArray, Response::copy(), Model::current_response(), and Minimizer::original_model().

Referenced by COLINOptimizer::post_run().

void local_recast_retrieve ( const Variables & vars, Response & response ) const  
  [protected]

infers MOO/NLS solution from the solution of a single-objective optimizer

Retrieve a MOO/NLS response based on the data returned by a single objective optimizer by performing a data_pairs search. This may get called even for a single user-specified function, since we may be recasting a single NLS residual into a squared objective. Always returns best data in the space of the original inbound Model.

References Response::active_set(), Dakota::data_pairs, Model::interface_id(), Iterator::iteratedModel, Dakota::lookup_by_val(), and Response::update().

Referenced by Minimizer::archive_best_results(), and Optimizer::post_run().

The documentation for this class was generated from the following files:

- DakotaMinimizer.hpp
- DakotaMinimizer.cpp
14.106 **MixedVarConstraints Class Reference**

Derived class within the Constraints hierarchy which separates continuous and discrete variables (no domain type array merging).

Inheritance diagram for MixedVarConstraints:

```
Constraints
   
MixedVarConstraints
```

### Public Member Functions

- **MixedVarConstraints** (const SharedVariablesData &svd)
  
  *lightweight constructor*

- **MixedVarConstraints** (const ProblemDescDB &problem_db, const SharedVariablesData &svd)
  
  *standard constructor*

- **~MixedVarConstraints** ()
  
  *destructor*

- **void** write (std::ostream &s) const
  
  *write a variable constraints object to an std::ostream*

- **void** read (std::istream &s)
  
  *read a variable constraints object from an std::istream*

### Additional Inherited Members

14.106.1 **Detailed Description**

Derived class within the Constraints hierarchy which separates continuous and discrete variables (no domain type array merging).

Derived variable constraints classes take different views of the design, uncertain, and state variable types and the continuous and discrete domain types. The MixedVarConstraints derived class separates the continuous and discrete domain types (see Variables::get_variables(problem_db) for variables type selection; variables type is passed to the Constraints constructor in Model).

14.106.2 **Constructor & Destructor Documentation**

MixedVarConstraints ( const ProblemDescDB & problem_db, const SharedVariablesData & svd )

*standard constructor*

In this class, mixed continuous/discrete variables are used. Most iterators/strategies use this approach, which is the default in Constraints::get_constraints().

References Constraints::allContinuousLowerBnds, Constraints::allContinuousUpperBnds, Constraints::allDiscrete-IntLowerBnds, Constraints::allDiscreteIntUpperBnds, Constraints::allDiscreteRealLowerBnds, Constraints::all-DiscreteRealUpperBnds, Dakota::copy_data_partial(), ProblemDescDB::get_iv(), ProblemDescDB::get_rv(), Constraints-::sharedVarsData, and SharedVariablesData::view().

The documentation for this class was generated from the following files:
14.107 MixedVariables Class Reference

Derived class within the Variables hierarchy which separates continuous and discrete variables (no domain type array merging).

Inheritance diagram for MixedVariables:

```
Variables
  MixedVariables
```

Public Member Functions

- **MixedVariables** (const ProblemDescDB &problem_db, const std::pair< short, short > &view)
  *standard constructor*
- **MixedVariables** (const SharedVariablesData &svd)
  *lightweight constructor*
- **~MixedVariables** ()
  *destructor*

Protected Member Functions

- **void read** (std::istream &s)
  *read a variables object from an std::istream*
- **void write** (std::ostream &s, unsigned short vars_part=ALL_VARS) const
  *write a variables object to an std::ostream, e.g., the console, optionally specifying which partition (all/active/inactive)*
- **void write_aprepro** (std::ostream &s) const
  *write a variables object to an std::ostream in aprepro format, e.g., a parameters file*
- **void read_tabular** (std::istream &s, unsigned short vars_part=ALL_VARS)
- **void write_tabular** (std::ostream &s, unsigned short vars_part=ALL_VARS) const
  *write a variables object in tabular format to an std::ostream, optionally specifying which partition (all/active/inactive)*
- **void write_tabular_labels** (std::ostream &s, unsigned short vars_part=ALL_VARS) const
  *write the labels in input spec order to a std::ostream, optionally specifying which partition (all/active/inactive)*
- **template<typename Reader > void read_core** (std::istream &s, Reader read_handler, unsigned short vars_part)
  *Implementation of reading various formats using the specified read handler.*
- **template<typename Writer > void write_core** (std::ostream &s, Writer write_handler, unsigned short vars_part) const
  *Implementation of writing various formats using the specified write handler.*
CHAPTER 14. CLASS DOCUMENTATION

Additional Inherited Members

14.107.1 Detailed Description

Derived class within the Variables hierarchy which separates continuous and discrete variables (no domain type array merging).

Derived variables classes take different views of the design, uncertain, and state variable types and the continuous and discrete domain types. The MixedVariables derived class separates the continuous and discrete domain types (see Variables::get_variables(problem_db)).

14.107.2 Constructor & Destructor Documentation

MixedVariables ( const ProblemDescDB & problem_db, const std::pair< short, short > & view )

standard constructor

In this class, the distinct approach is used (design, uncertain, and state variable types and continuous and discrete domain types are distinct). Most iterators/strategies use this approach.

References Variables::allContinuousVars, Variables::allDiscreteIntVars, Variables::allDiscreteRealVars, Variables::allDiscreteStringVars, Dakota::copy_data_partial(), ProblemDescDB::get_iv(), ProblemDescDB::get_rv(), ProblemDescDB::get_sa(), Variables::sharedVarsData, and SharedVariablesData::view().

14.107.3 Member Function Documentation

void read_tabular ( std::istream & s, unsigned short vars_part = ALL_VARS ) [protected], [virtual]

Tabular reader that reads data in order design, aleatory, epistemic, state according to counts in vc_tots (extract in order: cdv/ddv/ddrv, cauv/dauiv/daurv, ceuv/deuiv/deurv, csv/dsiv/dsrv, which might reflect active or all depending on context. Assumes container sized, since might be a view into a larger array.

Reimplemented from Variables.

References MixedVariables::read_core().

void read_core ( std::istream & s, Reader read_handler, unsigned short vars_part ) [protected]

Implementation of reading various formats using the specified read handler.

Reordering is required in all read/write cases that will be visible to the user since all derived vars classes should use the same CDV/DDV/UV/CSV/DSV ordering for clarity. Neutral file I/O, binary streams, and packed buffers do not need to reorder (so long as read/write are consistent) since this data is not intended for public consumption.

References SharedVariablesData::active_components_tots(), Variables::all_continuous_variable_labels(), Variables::all_discrete_int_variable_labels(), Variables::all_discrete_real_variable_labels(), Variables::all_discrete_string_variable_labels(), Variables::allContinuousVars, Variables::allDiscreteIntVars, Variables::allDiscreteRealVars, Variables::allDiscreteStringVars, SharedVariablesData::components_tots(), SharedVariablesData::cv_start(), SharedVariablesData::dv_start(), SharedVariablesData::drv_start(), SharedVariablesData::dsv_start(), SharedVariablesData::icv_start(), SharedVariablesData::idiv_start(), SharedVariablesData::idrv_start(), SharedVariablesData::idsv_start(), SharedVariablesData::inactive_components_tots(), and Variables::sharedVarsData.

Referenced by MixedVariables::read(), and MixedVariables::read_tabular().

The documentation for this class was generated from the following files:

- MixedVariables.hpp
- MixedVariables.cpp
14.108 Model Class Reference

Base class for the model class hierarchy.

Inheritance diagram for Model:

```
Model
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>NestedModel</td>
<td>RecastModel</td>
</tr>
<tr>
<td>SimulationModel</td>
<td>SurrogateModel</td>
</tr>
<tr>
<td>ActiveSubspaceModel</td>
<td>DataFitSurModel</td>
</tr>
<tr>
<td>AdaptedBasisModel</td>
<td>HierarchyModel</td>
</tr>
<tr>
<td>DataTransformModel</td>
<td></td>
</tr>
<tr>
<td>ProbabilityTransformModel</td>
<td></td>
</tr>
<tr>
<td>RandomFieldModel</td>
<td></td>
</tr>
<tr>
<td>ScalingModel</td>
<td></td>
</tr>
<tr>
<td>WeightingModel</td>
<td></td>
</tr>
</tbody>
</table>
```

Public Member Functions

- **Model ()**
  
  *default constructor*

- **Model (ProblemDescDB &problem_db)**
  
  *standard constructor for envelope*

- **Model (const Model &model)**
  
  *copy constructor*

- **virtual ~Model ()**
  
  *destructor*

- **Model operator= (const Model &model)**
  
  *assignment operator*

- **virtual Iterator & subordinate_iterator ()**
  
  *return the sub-iterator in nested and surrogate models*

- **virtual Model & subordinate_model ()**
  
  *return a single sub-model defined from subModel in nested and recast models and truth_model() in surrogate models; used for a directed dive through model recursions that may bypass some components.*

- **virtual void active_model_key (const UShortArray &mi_key)**
  
  *set the active multi-index key within surrogate data, grid driver, and approximation classes that support the management of multiple approximation states within surrogate models*

- **virtual void clear_model_keys ()**
  
  *reset by removing all multi-index keys within surrogate data, grid driver, and approximation classes that support the management of multiple approximation states within surrogate models*

- **virtual size_t qoi () const**
  
  *return number of unique response functions (managing any aggregations)*

- **virtual Model & surrogate_model ()**
return the active approximation sub-model in surrogate models

- virtual const Model & surrogate_model() const
  return the active approximation sub-model in surrogate models

- virtual Model & truth_model()
  return the active truth sub-model in surrogate models

- virtual const Model & truth_model() const
  return the active truth sub-model in surrogate models

- virtual void derived_subordinate_models(ModelList &ml, bool recurse_flag)
  portion of subordinate_models() specific to derived model classes

- virtual void resize_from_subordinate_model(size_t depth=std::numeric_limits<size_t>::max())
  resize vars/resp if needed from the bottom up

- virtual void update_from_subordinate_model(size_t depth=std::numeric_limits<size_t>::max())
  propagate vars/labels/bounds/targets from the bottom up

- virtual Interface & derived_interface()
  return the interface employed by the derived model class, if present: SimulationModel::userDefinedInterface, Data-FitSurrModel::approxInterface, or NestedModel::optionalInterface

- virtual size_t solution_levels(bool lwr_bnd=true) const
  number of discrete levels within solution control (SimulationModel)

- virtual void solution_level_index(unsigned short index)
  activate a particular level within the solution level control and return the cost estimate (SimulationModel)

- virtual unsigned short solution_level_index() const
  return currently active level within the solution level control (SimulationModel)

- virtual RealVector solution_level_costs() const
  return ordered cost estimates across solution levels (SimulationModel)

- virtual Real solution_level_cost() const
  return currently active cost estimate from solution level control (SimulationModel)

- virtual void primary_response_fn_weights(const RealVector &wts, bool recurse_flag=true)
  set the relative weightings for multiple objective functions or least squares terms

- virtual void surrogate_function_indices(const IntSet &surr_fn_indices)
  set the (currently active) surrogate function index set

- virtual Pecos::ProbabilityTransformation & probability_transformation()
  return probability transformation employed by the Model (forwarded along to ProbabilityTransformModel recasting)

- virtual bool initialize_mapping(ParLevLIter pl_iter)
  Perform any global updates prior to individual evaluate() calls; returns true if the variables size has changed.

- virtual bool finalize_mapping()
  restore state in preparation for next initialization; returns true if the variables size has changed

- virtual bool resize.pending() const
  return true if a potential resize is still pending, such that sizing-based initialization should be deferred

- virtual void nested_variable_mappings(const SizetArray &c_index1, const SizetArray &di_index1, const SizetArray &ds_index1, const SizetArray &dr_index1, const ShortArray &c_target2, const ShortArray &di_target2, const ShortArray &ds_target2, const ShortArray &dr_target2)
set primaryA {C, D, L, S, DR} VarMapIndices, secondaryA {C, D, L, S, DR} VarMapTargets (coming from a higher-level NestedModel context to inform derivative est.)

- virtual const SizetArray & nested_acv1_indices() const
  return primaryACVarMapIndices
- virtual const ShortArray & nested_acv2_targets() const
  return secondaryACVarMapTargets
- virtual short query_distribution_parameter_derivatives() const
  calculate and return derivative composition of final results w.r.t. distribution parameters (none, all, or mixed)
- virtual void activate_distribution_parameter_derivatives() ()
  activate derivative setting w.r.t. distribution parameters
- virtual void deactivate_distribution_parameter_derivatives() ()
  deactivate derivative setting w.r.t. distribution parameters
- virtual void trans_grad_X_to_U(const RealVector &fn_grad_x, RealVector &fn_grad_u, const RealVector &x_vars)
  transform x-space gradient vector to u-space
- virtual void trans_grad_U_to_X(const RealVector &fn_grad_u, RealVector &fn_grad_x, const RealVector &x_vars)
  transform u-space gradient vector to x-space
- virtual void trans_grad_X_to_S(const RealVector &fn_grad_x, RealVector &fn_grad_s, const RealVector &x_vars)
  transform x-space gradient vector to gradient with respect to inserted distribution parameters
- virtual void trans_hess_X_to_U(const RealSymMatrix &fn_hess_x, RealSymMatrix &fn_hess_u, const RealVector &x_vars, const RealVector &fn_grad_x)
  transform x-space Hessian matrix to u-space
- virtual void build_approximation() ()
  build a new SurrogateModel approximation
- virtual bool build_approximation(const Variables &vars, const IntResponsePair &response_pr)
  build a new SurrogateModel approximation using/enforcing anchor response at vars; rebuild if needed
- virtual void rebuild_approximation() ()
  update an existing SurrogateModel approximation
- virtual void update_approximation(bool rebuild_flag)
  replace the approximation data within an existing surrogate based on data updates propagated elsewhere
- virtual void update_approximation(const Variables &vars, const IntResponsePair &response_pr, bool rebuild_flag)
  replace the anchor point data within an existing surrogate
- virtual void update_approximation(const VariablesArray &vars_array, const IntResponseMap &resp_map, bool rebuild_flag)
  replace the data points within an existing surrogate
- virtual void update_approximation(const RealMatrix &samples, const IntResponseMap &resp_map, bool rebuild_flag)
  replace the data points within an existing surrogate
- virtual void append_approximation(bool rebuild_flag)
  append to the existing approximation data within a surrogate based on data updates propagated elsewhere
virtual void append_approximation (const Variables &vars, const IntResponsePair &response, bool rebuild_flag)
  append a single point to an existing surrogate’s data

virtual void append_approximation (const VariablesArray &vars_array, const IntResponseMap &resp_map, bool rebuild_flag)
  append multiple points to an existing surrogate’s data

virtual void append_approximation (const RealMatrix &samples, const IntResponseMap &resp_map, bool rebuild_flag)
  append multiple points to an existing surrogate’s data

virtual void pop_approximation (bool save_surr_data, bool rebuild_flag=false)
  remove the previous data set addition to a surrogate (e.g., due to a previous append_approximation() call); flag manages storing of surrogate data for use in a subsequent push_approximation()

virtual void push_approximation ()
  push a previous approximation data state; reverse of pop_approximation

virtual bool push_available ()
  query for whether a trial increment is restorable within a surrogate

virtual void finalize_approximation ()
  finalize an approximation by applying all previous trial increments

virtual void combine_approximation ()
  combine the current approximation with previously stored data sets

virtual void combined_to_active (bool clear_combined=true)
  promote the combined approximation into the active approximation

virtual void clear_inactive ()
  clear inactive approximations (finalization + combination completed)

virtual void run_dace ()
  execute the DACE iterator (prior to building/appending the approximation)

virtual bool force_rebuild ()
  determine whether a surrogate model rebuild should be forced based on changes in the inactive data

virtual SharedApproxData & shared_approximation ()
  retrieve the shared approximation data within the ApproximationInterface of a DataFitSurrModel

virtual std::vector<Approximation> & approximations ()
  retrieve the set of Approximations within the ApproximationInterface of a DataFitSurrModel

virtual const Pecos::SurrogateData & approximation_data (size_t fn_index)
  retrieve a SurrogateData instance from a particular Approximation instance within the ApproximationInterface of a DataFitSurrModel

virtual const RealVectorArray & approximation_coefficients (bool normalized=false)
  retrieve the approximation coefficients from each Approximation within a DataFitSurrModel

virtual void approximation_coefficients (const RealVectorArray &approx_coeffs, bool normalized=false)
  set the approximation coefficients for each Approximation within a DataFitSurrModel

virtual const RealVector & approximation_variances (const Variables &vars)
  retrieve the prediction variances from each Approximation within a DataFitSurrModel

virtual void surrogate_response_mode (short mode)
set response computation mode used in SurrogateModels for forming currentResponse

- virtual short surrogate_response_mode () const
  return response computation mode used in SurrogateModels for forming currentResponse

- virtual const RealVector & error_estimates ()
  retrieve error estimates corresponding to the Model’s response (could be surrogate error for SurrogateModels, statistical MSE for NestedModels, or adjoint error estimates for SimulationModels). Errors returned correspond to most recent evaluate().

- virtual DiscrepancyCorrection & discrepancy_correction ()
  return the DiscrepancyCorrection object used by SurrogateModels

- virtual void correction_type (short corr_type)
  set the correction type from the DiscrepancyCorrection object used by SurrogateModels

- virtual short correction_type ()
  return the correction type from the DiscrepancyCorrection object used by SurrogateModels

- virtual void single_apply (const Variables &vars, Response &resp, const UShortArray &paired_key)
  apply a DiscrepancyCorrection to correct an approximation within a HierarchSurrModel

- virtual void recursive_apply (const Variables &vars, Response &resp)
  apply a sequence of DiscrepancyCorrections to recursively correct an approximation within a HierarchSurrModel

- virtual void component_parallel_mode (short mode)
  update componentParallelMode for supporting parallelism in model sub-components

- virtual IntIntPair estimate_partition_bounds (int max_eval_concurrency)
  estimate the minimum and maximum partition sizes that can be utilized by this Model

- virtual size_t mi_parallel_level_index () const
  return the index for the metaiterator-iterator parallelism level within ParallelConfiguration::miPLIter that is active for use in a particular Model at runtime

- virtual void cache_unmatched_response (int raw_id)
  migrate an unmatched response record from active response map (computed by synchronize() or synchronize_nowait()) to cached response map

- virtual short local_eval_synchronization ()
  return derived model synchronization setting

- virtual int local_eval_concurrency ()
  return derived model asynchronous evaluation concurrency

- virtual void serve_run (ParLevLIter pl_iter, int max_eval_concurrency)
  Service job requests received from the master. Completes when a termination message is received from stop_servers().

- virtual void stop_servers ()
  Executed by the master to terminate all server operations for a particular model when iteration on the model is complete.

- virtual bool derived_master_overload () const
  Return a flag indicating the combination of multiprocessor evaluations and a dedicated master iterator scheduling. Used in synchronous evaluate functions to prevent the error of trying to run a multiprocessor job on the master.

- virtual void inactive_view (short view, bool recurse_flag=true)
  update the Model’s inactive view based on higher level (nested) context

- virtual const String & interface_id () const
  return the interface identifier
virtual int derived_evaluation_id () const

    Return the value of the evaluation id counter for the Model.

virtual bool evaluation_cache (bool recurse_flag=true) const

    Indicates the usage of an evaluation cache by the Model.

virtual bool restart_file (bool recurse_flag=true) const

    Indicates the usage of a restart file by the Model.

virtual void set_evaluation_reference ()

    Set the reference points for the evaluation counters within the Model.

virtual void fine_grained_evaluation_counters ()

    Request fine-grained evaluation reporting within the Model.

virtual void print_evaluation_summary (std::ostream &s, bool minimal_header=false, bool relative_count=true) const

    Print an evaluation summary for the Model.

virtual void eval_tag_prefix (const String &eval_id_str)

    Set the hierarchical eval ID tag prefix

virtual bool db_lookup (const Variables &search_vars, const ActiveSet &search_set, Response &found_resp)

    search the eval database (during derivative estimation); derived may need to reimplement due to problem transformations (RecastModel); return true if found in DB

virtual void stop_init_mapping (ParLevLIter pl_iter)

    called from IteratorScheduler::run_iterator() for iteratorComm rank 0 to terminate serve_init_mapping() on other iteratorComm processors

virtual int serve_init_mapping (ParLevLIter pl_iter)

    called from IteratorScheduler::run_iterator() for iteratorComm rank != 0 to balance resize() calls on iteratorComm rank 0

virtual void stop_finalize_mapping (ParLevLIter pl_iter)

    called from IteratorScheduler::run_iterator() for iteratorComm rank 0 to terminate serve_finalize_mapping() on other iteratorComm processors

virtual int serve_finalize_mapping (ParLevLIter pl_iter)

    called from IteratorScheduler::run_iterator() for iteratorComm rank != 0 to balance resize() calls on iteratorComm rank 0

virtual void warm_start_flag (const bool flag)

    set the warm start flag (warmStartFlag)

virtual void declare_sources ()

    Declare a model’s sources to the evaluationsDB.

ModelList & subordinate_models (bool recurse_flag=true)

    return the sub-models in nested and surrogate models

void evaluate ()

    Compute the Response at currentVariables (default ActiveSet).

void evaluate (const ActiveSet &set)

    Compute the Response at currentVariables (specified ActiveSet).

void evaluate_nowait ()

    Spawn an asynchronous job (or jobs) that computes the value of the Response at currentVariables (default ActiveSet).

void evaluate_nowait (const ActiveSet &set)
14.108. MODEL CLASS REFERENCE

Spawn an asynchronous job (or jobs) that computes the value of the Response at currentVariables (specified Active-Set).

- `const IntResponseMap & synchronize ()`
  Execute a blocking scheduling algorithm to collect the complete set of results from a group of asynchronous evaluations.

- `const IntResponseMap & synchronize_nowait ()`
  Execute a nonblocking scheduling algorithm to collect all available results from a group of asynchronous evaluations.

- `int evaluation_id () const`
  return Model’s (top-level) evaluation counter, not to be confused with derived counter returned by derived_evaluation_id()

- `bool mapping_initialized () const`

- `void init_communicators (ParLevLIter pl_iter, int max_eval_concurrency, bool recurse_flag=true)`
  allocate communicator partitions for a model and store configuration in modelPCIterMap

- `void init_serial ()`
  for cases where init_communicators() will not be called, modify some default settings to behave properly in serial.

- `void set_communicators (ParLevLIter pl_iter, int max_eval_concurrency, bool recurse_flag=true)`
  set active parallel configuration for the model (set modelPCIter from modelPCIterMap)

- `void free_communicators (ParLevLIter pl_iter, int max_eval_concurrency, bool recurse_flag=true)`
  deallocate communicator partitions for a model

- `MPI_Comm analysis_comm () const`
  retrieve the MPI communicator on which this model is configured to conduct function evaluation analyses (provided for library clients)

- `void stop_init_communicators (ParLevLIter pl_iter)`
  called from IteratorScheduler::init_iterator() for iteratorComm rank 0 to terminate serve_init_communicators() on other iteratorComm processors

- `int serve_init_communicators (ParLevLIter pl_iter)`
  called from IteratorScheduler::init_iterator() for iteratorComm rank != 0 to balance init_communicators() calls on iteratorComm rank 0

- `void estimate_message_lengths ()`
  estimate messageLengths for a model

- `size_t response_size () const`
  return (potentially aggregated) size of response vector in currentResponse

- `bool manage_data_recastings ()`
  initialize modelList and recastFlags for data import/export

- `bool recastings () const`
  return true if recastFlags is defined

- `void user_space_to_iterator_space (const Variables &user_vars, const Response &user.resp, Variables &iter_vars, Response &iterResp)`
  employ the model recursion to transform from bottom level user-space data to top level iterator-space data

- `void iterator_space_to_user_space (const Variables &iter_vars, const Response &iter_resp, Variables &user_vars, Response &user Resp)`
  employ the model recursion to transform from top level iterator-space data to bottom level user-space data

- `void assign_rep (Model *model_rep, bool ref_count_incr=true)`
  replaces existing letter with a new one
• size_t tv () const
  returns total number of vars

• size_t cv () const
  returns number of active continuous variables

• size_t div () const
  returns number of active discrete integer vars

• size_t dsv () const
  returns number of active discrete string vars

• size_t drv () const
  returns number of active discrete real vars

• size_t icv () const
  returns number of inactive continuous variables

• size_t idiv () const
  returns number of inactive discrete integer vars

• size_t idsv () const
  returns number of inactive discrete string vars

• size_t idrv () const
  returns number of inactive discrete real vars

• size_t acv () const
  returns total number of continuous variables

• size_t adiv () const
  returns total number of discrete integer vars

• size_t adsv () const
  returns total number of discrete string vars

• size_t adrv () const
  returns total number of discrete real vars

• void active_variables (const Variables &vars)
  set the active variables in currentVariables

• const RealVector & continuous_variables () const
  return the active continuous variables from currentVariables

• Real continuous_variable (size_t i) const
  return an active continuous variable from currentVariables

• void continuous_variables (const RealVector &c_vars)
  set the active continuous variables in currentVariables

• void continuous_variable (Real c_var, size_t i)
  set an active continuous variable in currentVariables

• const IntVector & discrete_int_variables () const
  return the active discrete integer variables from currentVariables

• int discrete_int_variable (size_t i) const
  return an active discrete integer variable from currentVariables

• void discrete_int_variables (const IntVector &d_vars)
  set the active discrete integer variables in currentVariables

• void discrete_int_variable (int d_var, size_t i)
set an active discrete integer variable in currentVariables

- StringMultiArrayConstView discrete_string_variables() const
  return the active discrete string variables from currentVariables
- const String & discrete_string_variable(size_t i) const
  return an active discrete string variable from currentVariables
- void discrete_string_variables(StringMultiArrayConstView d_vars)
  set the active discrete string variables in currentVariables
- void discrete_string_variable(const String &d_var, size_t i)
  set an active discrete string variable in currentVariables
- const RealVector & discrete_real_variables() const
  return the active discrete real variables from currentVariables
- Real discrete_real_variable(size_t i) const
  return an active discrete real variable from currentVariables
- void discrete_real_variables(const RealVector &d_vars)
  set the active discrete real variables in currentVariables
- void discrete_real_variable(Real d_var, size_t i)
  set an active discrete real variable in currentVariables
- UShortMultiArrayConstView continuous_variable_types() const
  return the active continuous variable types from currentVariables
- void continuous_variable_types(UShortMultiArrayConstView cv_types)
  set the active continuous variable types in currentVariables
- void continuous_variable_type(unsigned short cv_type, size_t i)
  set an active continuous variable type in currentVariables
- UShortMultiArrayConstView discrete_int_variable_types() const
  return the active discrete variable types from currentVariables
- void discrete_int_variable_types(UShortMultiArrayConstView div_types)
  set the active discrete variable types in currentVariables
- void discrete_int_variable_type(unsigned short div_type, size_t i)
  set an active discrete variable type in currentVariables
- UShortMultiArrayConstView discrete_string_variable_types() const
  return the active discrete variable types from currentVariables
- void discrete_string_variable_types(UShortMultiArrayConstView drv_types)
  set the active discrete variable types in currentVariables
- void discrete_string_variable_type(unsigned short drv_type, size_t i)
  set an active discrete variable type in currentVariables
- UShortMultiArrayConstView discrete_real_variable_types() const
  return the active discrete variable types from currentVariables
- void discrete_real_variable_types(UShortMultiArrayConstView drv_types)
  set the active discrete variable types in currentVariables
- void discrete_real_variable_type(unsigned short drv_type, size_t i)
  set an active discrete variable type in currentVariables
- SizetMultiArrayConstView continuous_variable_ids() const
  return the active continuous variable identifiers from currentVariables
• void continuous_variable_ids (SizetMultiArrayConstView cv_ids)
  set the active continuous variable identifiers in currentVariables

• void continuous_variable_id (size_t cv_id, size_t i)
  set an active continuous variable identifier in currentVariables

• const RealVector & inactive_continuous_variables () const
  return the inactive continuous variables in currentVariables

• void inactive_continuous_variables (const RealVector &i_c_vars)
  set the inactive continuous variables in currentVariables

• const IntVector & inactive_discrete_int_variables () const
  return the inactive discrete variables in currentVariables

• void inactive_discrete_int_variables (const IntVector &i_d_vars)
  set the inactive discrete variables in currentVariables

• StringMultiArrayConstView inactive_discrete_string_variables () const
  return the inactive discrete variables in currentVariables

• void inactive_discrete_string_variables (StringMultiArrayConstView i_d_vars)
  set the inactive discrete variables in currentVariables

• const RealVector & inactive_discrete_real_variables () const
  return the inactive discrete variables in currentVariables

• void inactive_discrete_real_variables (const RealVector &i_d_vars)
  set the inactive discrete variables in currentVariables

• UShortMultiArrayConstView inactive_continuous_variable_types () const
  return the inactive continuous variable types from currentVariables

• SizetMultiArrayConstView inactive_continuous_variable_ids () const
  return the inactive continuous variable identifiers from currentVariables

• const RealVector & all_continuous_variables () const
  return all continuous variables in currentVariables

• void all_continuous_variables (const RealVector &a_c_vars)
  set all continuous variables in currentVariables

• void all_continuous_variable (Real a_c_var, size_t i)
  set a variable within the all continuous variables in currentVariables

• const IntVector & all_discrete_int_variables () const
  return all discrete variables in currentVariables

• void all_discrete_int_variables (const IntVector &a_d_vars)
  set all discrete variables in currentVariables

• void all_discrete_int_variable (int a_d_var, size_t i)
  set a variable within the all discrete variables in currentVariables

• StringMultiArrayConstView all_discrete_string_variables () const
  return all discrete variables in currentVariables

• void all_discrete_string_variables (StringMultiArrayConstView a_d_vars)
  set all discrete variables in currentVariables

• void all_discrete_string_variable (const String &a_d_var, size_t i)
  set a variable within the all discrete variables in currentVariables

• const RealVector & all_discrete_real_variables () const
return all discrete variables in currentVariables

- void **all_discrete_real_variables** (const RealVector &a_dvars)
  set all discrete variables in currentVariables

- void **all_discrete_real_variable** (Real a_d_var, size_t i)
  set a variable within the all discrete variables in currentVariables

- UShortMultiArrayConstView **all_continuous_variable_types** () const
  return all continuous variable types from currentVariables

- UShortMultiArrayConstView **all_discrete_int_variable_types** () const
  return all discrete variable types from currentVariables

- UShortMultiArrayConstView **all_discrete_string_variable_types** () const
  return all discrete variable types from currentVariables

- UShortMultiArrayConstView **all_discrete_real_variable_types** () const
  return all discrete variable types from currentVariables

- SizetMultiArrayConstView **all_continuous_variable_ids** () const
  return all continuous variable identifiers from currentVariables

- const BitArray & **discrete_int_sets** ()
  define and return discreteIntSets using active view from currentVariables

- const BitArray & **discrete_int_sets** (short active_view)
  define and return discreteIntSets using passed active view

- const IntSetArray & **discrete_set_int_values** ()
  return the sets of values available for each of the active discrete set integer variables (aggregated in activeDiscSetIntValues)

- const IntSetArray & **discrete_set_int_values** (short active_view)
  return the sets of values available for each of the active discrete set integer variables (aggregated in activeDiscSetIntValues)

- const StringSetArray & **discrete_set_string_values** ()
  return the sets of values available for each of the active discrete set string variables (aggregated in activeDiscSetStringValues)

- const StringSetArray & **discrete_set_string_values** (short active_view)
  return the sets of values available for each of the active discrete set string variables (aggregated in activeDiscSetStringValues)

- const RealSetArray & **discrete_set_real_values** ()
  return the sets of values available for each of the active discrete set real variables (aggregated in activeDiscSetRealValues)

- const RealSetArray & **discrete_set_real_values** (short active_view)
  return the sets of values available for each of the active discrete set real variables (aggregated in activeDiscSetRealValues)

- Pecos::MultivariateDistribution & **multivariate_distribution** ()
  return mvDist

- const Pecos::MultivariateDistribution & **multivariate_distribution** () const
  return mvDist

- StringMultiArrayConstView **continuous_variable_labels** () const
  return the active continuous variable labels from currentVariables
• void continuous_variable_labels (StringMultiArrayConstView c_v_labels)
  set the active continuous variable labels in currentVariables
• StringMultiArrayConstView discrete_int_variable_labels () const
  return the active discrete variable labels from currentVariables
• void discrete_int_variable_labels (StringMultiArrayConstView d_v_labels)
  set the active discrete variable labels in currentVariables
• StringMultiArrayConstView discrete_string_variable_labels () const
  return the active discrete variable labels from currentVariables
• void discrete_string_variable_labels (StringMultiArrayConstView d_v_labels)
  set the active discrete variable labels in currentVariables
• StringMultiArrayConstView discrete_real_variable_labels () const
  return the active discrete variable labels from currentVariables
• void discrete_real_variable_labels (StringMultiArrayConstView d_v_labels)
  set the active discrete variable labels in currentVariables
• StringMultiArrayConstView inactive_continuous_variable_labels () const
  return the inactive continuous variable labels in currentVariables
• void inactive_continuous_variable_labels (StringMultiArrayConstView i_c_v_labels)
  set the inactive continuous variable labels in currentVariables
• StringMultiArrayConstView inactive_discrete_int_variable_labels () const
  return the inactive discrete variable labels in currentVariables
• void inactive_discrete_int_variable_labels (StringMultiArrayConstView i_d_v_labels)
  set the inactive discrete variable labels in currentVariables
• StringMultiArrayConstView inactive_discrete_string_variable_labels () const
  return the inactive discrete variable labels in currentVariables
• void inactive_discrete_string_variable_labels (StringMultiArrayConstView i_d_v_labels)
  set the inactive discrete variable labels in currentVariables
• StringMultiArrayConstView inactive_discrete_real_variable_labels () const
  return the inactive discrete variable labels in currentVariables
• void inactive_discrete_real_variable_labels (StringMultiArrayConstView i_d_v_labels)
  set the inactive discrete variable labels in currentVariables
• StringMultiArrayConstView all_continuous_variable_labels () const
  return all continuous variable labels in currentVariables
• void all_continuous_variable_labels (StringMultiArrayConstView a_c_v_labels)
  set all continuous variable labels in currentVariables
• void all_continuous_variable_label (const String &a_c_v_label, size_t i)
  set a label within the all continuous labels in currentVariables
• StringMultiArrayConstView all_discrete_int_variable_labels () const
  return all discrete variable labels in currentVariables
• void all_discrete_int_variable_labels (StringMultiArrayConstView a_d_v_labels)
  set all discrete variable labels in currentVariables
• void all_discrete_int_variable_label (const String &a_d_v_label, size_t i)
  set a label within the all discrete labels in currentVariables
• StringMultiArrayConstView all_discrete_string_variable_labels () const
  return all discrete variable labels in currentVariables
• void all_discrete_string_variable_labels (StringMultiArrayConstView a_d_v_labels)
  set all discrete variable labels in currentVariables
• void all_discrete_string_variable_label (const String &a_d_v_label, size_t i)
  set a label within the all discrete labels in currentVariables
• StringMultiArrayConstView all_discrete_real_variable_labels () const
  return all discrete variable labels in currentVariables
• void all_discrete_real_variable_labels (StringMultiArrayConstView a_d_v_labels)
  set all discrete variable labels in currentVariables
• void all_discrete_real_variable_label (const String &a_d_v_label, size_t i)
  set a label within the all discrete labels in currentVariables
return all discrete variable labels in currentVariables

- void all_discrete_string_variable_labels (StringMultiArrayConstView a_d_v_labels)
  set all discrete variable labels in currentVariables

- void all_discrete_string_variable_label (const String &a_d_v_label, size_t i)
  set a label within the all discrete labels in currentVariables

- StringMultiArrayConstView all_discrete_real_variable_labels () const
  return all discrete variable labels in currentVariables

- void all_discrete_real_variable_labels (StringMultiArrayConstView a_d_v_labels)
  set all discrete variable labels in currentVariables

- void all_discrete_real_variable_label (const String &a_d_v_label, size_t i)
  set a label within the all discrete labels in currentVariables

- const StringArray & response_labels () const
  return the response labels from currentResponse

- void response_labels (const StringArray &resp_labels)
  set the response labels in currentResponse

- const RealVector & continuous_lower_bounds () const
  return the active continuous lower bounds from userDefinedConstraints

- Real continuous_lower_bound (size_t i) const
  return an active continuous lower bound from userDefinedConstraints

- void continuous_lower_bounds (const RealVector &c_l_bnds)
  set the active continuous lower bounds in userDefinedConstraints

- void continuous_lower_bound (Real c_l_bnd, size_t i)
  set the i-th active continuous lower bound in userDefinedConstraints

- const RealVector & continuous_upper_bounds () const
  return the active continuous upper bounds from userDefinedConstraints

- Real continuous_upper_bound (size_t i) const
  return an active continuous upper bound from userDefinedConstraints

- void continuous_upper_bounds (const RealVector &c_u_bnds)
  set the active continuous upper bounds in userDefinedConstraints

- void continuous_upper_bound (Real c_u_bnd, size_t i)
  set the i-th active continuous upper bound from userDefinedConstraints

- const IntVector & discrete_int_lower_bounds () const
  return the active discrete int lower bounds from userDefinedConstraints

- int discrete_int_lower_bound (size_t i) const
  return an active discrete int lower bound from userDefinedConstraints

- void discrete_int_lower_bounds (const IntVector &d_l_bnds)
  set the active discrete int lower bounds in userDefinedConstraints

- void discrete_int_lower_bound (int d_l_bnd, size_t i)
  set the i-th active discrete int lower bound in userDefinedConstraints

- const IntVector & discrete_int_upper_bounds () const
  return the active discrete int upper bounds from userDefinedConstraints

- int discrete_int_upper_bound (size_t i) const
  return an active discrete int upper bound from userDefinedConstraints
• void discrete_int_upper_bounds (const IntVector &d_u_bnds)
  
  set the active discrete int upper bounds in userDefinedConstraints

• void discrete_int_upper_bound (int d_u_bnd, size_t i)
  
  set the i-th active discrete int upper bound in userDefinedConstraints

• const RealVector & discrete_real_lower_bounds () const
  
  return the active discrete real lower bounds from userDefinedConstraints

• Real discrete_real_lower_bound (size_t i) const
  
  return an active discrete real lower bound from userDefinedConstraints

• void discrete_real_lower_bounds (const RealVector &d_l_bnds)
  
  set the active discrete real lower bounds in userDefinedConstraints

• void discrete_real_lower_bound (Real d_l_bnd, size_t i)
  
  set the i-th active discrete real lower bound in userDefinedConstraints

• const RealVector & discrete_real_upper_bounds () const
  
  return the active discrete real upper bounds from userDefinedConstraints

• Real discrete_real_upper_bound (size_t i) const
  
  return an active discrete real upper bound from userDefinedConstraints

• void discrete_real_upper_bounds (const RealVector &d_u_bnds)
  
  set the active discrete real upper bounds in userDefinedConstraints

• void discrete_real_upper_bound (Real d_u_bnd, size_t i)
  
  set the i-th active discrete real upper bound in userDefinedConstraints

• const RealVector & inactive_continuous_lower_bounds () const
  
  return the inactive continuous lower bounds in userDefinedConstraints

• void inactive_continuous_lower_bounds (const RealVector &i_c_l_bnds)
  
  set the inactive continuous lower bounds in userDefinedConstraints

• const RealVector & inactive_continuous_upper_bounds () const
  
  return the inactive continuous upper bounds in userDefinedConstraints

• void inactive_continuous_upper_bounds (const RealVector &i_c_u_bnds)
  
  set the inactive continuous upper bounds in userDefinedConstraints

• const IntVector & inactive_discrete_int_lower_bounds () const
  
  return the inactive discrete lower bounds in userDefinedConstraints

• void inactive_discrete_int_lower_bounds (const IntVector &i_d_l_bnds)
  
  set the inactive discrete lower bounds in userDefinedConstraints

• const IntVector & inactive_discrete_int_upper_bounds () const
  
  return the inactive discrete upper bounds in userDefinedConstraints

• void inactive_discrete_int_upper_bounds (const IntVector &i_d_u_bnds)
  
  set the inactive discrete upper bounds in userDefinedConstraints

• const RealVector & inactive_discrete_real_lower_bounds () const
  
  return the inactive discrete lower bounds in userDefinedConstraints

• void inactive_discrete_real_lower_bounds (const RealVector &i_d_l_bnds)
  
  set the inactive discrete lower bounds in userDefinedConstraints

• const RealVector & inactive_discrete_real_upper_bounds () const
  
  return the inactive discrete upper bounds in userDefinedConstraints

• void inactive_discrete_real_upper_bounds (const RealVector &i_d_u_bnds)
set the inactive discrete upper bounds in userDefinedConstraints

- const RealVector & all_continuous_lower_bounds() const
  return all continuous lower bounds in userDefinedConstraints
- void all_continuous_lower_bounds(const RealVector &a_c_l_bnds)
  set all continuous lower bounds in userDefinedConstraints
- void all_continuous_lower_bound(Real a_c_l_bnd, size_t i)
  set a lower bound within continuous lower bounds in userDefinedConstraints
- const RealVector & all_continuous_upper_bounds() const
  return all continuous upper bounds in userDefinedConstraints
- void all_continuous_upper_bounds(const RealVector &a_c_u_bnds)
  set all continuous upper bounds in userDefinedConstraints
- void all_continuous_upper_bound(Real a_c_u_bnd, size_t i)
  set an upper bound within all continuous upper bounds in userDefinedConstraints
- const IntVector & all_discrete_int_lower_bounds() const
  return all discrete lower bounds in userDefinedConstraints
- void all_discrete_int_lower_bounds(const IntVector &a_d_l_bnds)
  set all discrete lower bounds in userDefinedConstraints
- void all_discrete_int_lower_bound(int a_d_l_bnd, size_t i)
  set a lower bound within all discrete lower bounds in userDefinedConstraints
- const IntVector & all_discrete_int_upper_bounds() const
  return all discrete upper bounds in userDefinedConstraints
- void all_discrete_int_upper_bounds(const IntVector &a_d_u_bnds)
  set all discrete upper bounds in userDefinedConstraints
- void all_discrete_int_upper_bound(int a_d_u_bnd, size_t i)
  set an upper bound within all discrete upper bounds in userDefinedConstraints
- const RealVector & all_discrete_real_lower_bounds() const
  return all discrete lower bounds in userDefinedConstraints
- void all_discrete_real_lower_bounds(const RealVector &a_d_l_bnds)
  set all discrete lower bounds in userDefinedConstraints
- void all_discrete_real_lower_bound(Real a_d_l_bnd, size_t i)
  set a lower bound within all discrete lower bounds in userDefinedConstraints
- const RealVector & all_discrete_real_upper_bounds() const
  return all discrete upper bounds in userDefinedConstraints
- void all_discrete_real_upper_bounds(const RealVector &a_d_u_bnds)
  set all discrete upper bounds in userDefinedConstraints
- void all_discrete_real_upper_bound(Real a_d_u_bnd, size_t i)
  set an upper bound within all discrete upper bounds in userDefinedConstraints
- void reshape_constraints(size_t num_nln_ineq_cons, size_t num_nln_eq_cons, size_t num_lin_ineq_cons, size_t num_lin_eq_cons)
  reshape the linear/nonlinear constraint arrays
- size_t num_linear_ineq_constraints() const
  return the number of linear inequality constraints
- size_t num_linear_eq_constraints() const
• const RealMatrix & linear_ineq_constraint_coeffs () const
  return the linear inequality constraint coefficients
• void linear_ineq_constraint_coeffs (const RealMatrix &lin_ineq_coeffs)
  set the linear inequality constraint coefficients
• const RealVector & linear_ineq_constraint_lower_bounds () const
  return the linear inequality constraint lower bounds
• void linear_ineq_constraint_lower_bounds (const RealVector &lin_ineq_l_bnds)
  set the linear inequality constraint lower bounds
• const RealVector & linear_ineq_constraint_upper_bounds () const
  return the linear inequality constraint upper bounds
• void linear_ineq_constraint_upper_bounds (const RealVector &lin_ineq_u_bnds)
  set the linear inequality constraint upper bounds
• const RealMatrix & linear_eq_constraint_coeffs () const
  return the linear equality constraint coefficients
• void linear_eq_constraint_coeffs (const RealMatrix &lin_eq_coeffs)
  set the linear equality constraint coefficients
• const RealVector & linear_eq_constraint_targets () const
  return the linear equality constraint targets
• void linear_eq_constraint_targets (const RealVector &lin_eq_targets)
  set the linear equality constraint targets
• size_t num_nonlinear_ineq_constraints () const
  return the number of nonlinear inequality constraints
• size_t num_nonlinear_eq_constraints () const
  return the number of nonlinear equality constraints
• const RealVector & nonlinear_ineq_constraint_lower_bounds () const
  return the nonlinear inequality constraint lower bounds
• void nonlinear_ineq_constraint_lower_bounds (const RealVector &nln_ineq_l_bnds)
  set the nonlinear inequality constraint lower bounds
• const RealVector & nonlinear_ineq_constraint_upper_bounds () const
  return the nonlinear inequality constraint upper bounds
• void nonlinear_ineq_constraint_upper_bounds (const RealVector &nln_ineq_u_bnds)
  set the nonlinear inequality constraint upper bounds
• const RealVector & nonlinear_eq_constraint_targets () const
  return the nonlinear equality constraint targets
• void nonlinear_eq_constraint_targets (const RealVector &nln_eq_targets)
  set the nonlinear equality constraint targets
• const Variables & current_variables () const
  return the current variables (currentVariables) as const reference (preferred)
• Variables & current_variables ()
  return the current variables (currentVariables) in mutable form (special cases)
• const Constraints & user_defined_constraints () const
  return the user-defined constraints (userDefinedConstraints)
const Response & current_response() const
return the current response (currentResponse)

ProblemDescDB & problem_description_db() const
return the problem description database (probDescDB)

ParallelLibrary & parallel_library() const
return the parallel library (parallelLib)

const String & model_type() const
return the model type (modelType)

const String & surrogate_type() const
return the surrogate type (surrogateType)

const String & model_id() const
return the model identifier (modelId)

size_t num_primary_fns() const
return number of primary functions (total less nonlinear constraints)

size_t num_secondary_fns() const
return number of secondary functions (number of nonlinear constraints)

const String & gradient_type() const
return the gradient evaluation type (gradientType)

const String & method_source() const
return the numerical gradient evaluation method source (methodSource)

const String & interval_type() const
return the numerical gradient evaluation interval type (intervalType)

bool ignore_bounds() const
option for ignoring bounds when numerically estimating derivatives

bool central_hess() const
option for using old 2nd-order scheme when computing finite-diff Hessian

const RealVector & fd_gradient_step_size() const
return the finite difference gradient step size (fdGradStepSize)

const String & fd_gradient_step_type() const
return the finite difference gradient step type (fdGradStepType)

const IntSet & gradient_id_analytic() const
return the mixed gradient analytic IDs (gradIdAnalytic)

const IntSet & gradient_id_numerical() const
return the mixed gradient numerical IDs (gradIdNumerical)

const String & hessian_type() const
return the Hessian evaluation type (hessianType)

const String & quasi_hessian_type() const
return the Hessian evaluation type (quasiHessType)

const RealVector & fd_hessian_by_grad_step_size() const
return gradient-based finite difference Hessian step size (fdHessByGradStepSize)

const RealVector & fd_hessian_by_fn_step_size() const
return function-based finite difference Hessian step size (fdHessByFnStepSize)

const String & fd_hessian_step_type() const
return the finite difference Hessian step type (fdHessStepType)

- const IntSet & hessian_id_analytic () const
  return the mixed Hessian analytic IDs (hessIdAnalytic)

- const IntSet & hessian_id_numerical () const
  return the mixed Hessian analytic IDs (hessIdNumerical)

- const IntSet & hessian_id_quasi () const
  return the mixed Hessian analytic IDs (hessIdQuasi)

- void primary_response_fn_sense (const BoolDeque &sense)
  set the optimization sense for multiple objective functions

- const BoolDeque & primary_response_fn_sense () const
  get the optimization sense for multiple objective functions

- const RealVector & primary_response_fn_weights () const
  get the relative weightings for multiple objective functions or least squares terms

- const ScalingOptions & scaling_options () const
  user-provided scaling options

- short primary_fn_type () const
  get the primary response function type (generic, objective, calibration)

- void primary_fn_type (short type)
  set the primary response function type, e.g., when recasting

- bool derivative_estimation ()
  indicates potential usage of estimate_derivatives() based on gradientType/hessianType

- void supports_derivative_estimation (bool sed_flag)
  set whether this model should perform or pass on derivative estimation

- void init_comms_bcast_flag (bool icb_flag)
  set initCommsBcastFlag

- int evaluation_capacity () const
  return the evaluation capacity for use in iterator logic

- int derivative_concurrency () const
  return the gradient concurrency for use in parallel configuration logic

- bool asynch_flag () const
  return the asynchronous evaluation flag (asynchEvalFlag)

- void asynch_flag (const bool flag)
  set the asynchronous evaluation flag (asynchEvalFlag)

- short output_level () const
  return the outputLevel

- void output_level (const short level)
  set the outputLevel

- const IntArray & message_lengths () const
  return the array of MPI packed message buffer lengths (messageLengths)

- void parallel_configuration_iterator (ParConfigLIter pc_iter)
  set modelPCIter

- ParConfigLIter parallel_configuration_iterator () const
  return modelPCIter
• void **auto** graphics** (const bool flag)
  set modelAutoGraphicsFlag to activate posting of graphics data within evaluate/synchronize functions (automatic graphics posting in the model as opposed to graphics posting at the strategy level).

• bool **auto** graphics** () const
  get modelAutoGraphicsFlag to activate posting of graphics data within evaluate/synchronize functions (automatic graphics posting in the model as opposed to graphics posting at the strategy level).

• bool **is_null** () const
  function to check modelRep (does this envelope contain a letter)

• Model * **model_rep** () const
  returns modelRep for access to derived class member functions that are not mapped to the top Model level

• virtual String root model id ()
  Return the model ID of the “innermost” model. For all derived Models except RecastModels, return modelId. The RecastModel override returns the root model id() of the subModel.

• virtual ActiveSet default_active_set ()

Static Public Member Functions

• static void **active** variables (const RealVector &config vars, Model &model)
  set the specified configuration to the Model’s inactive vars, converting from real to integer or through index to string value as needed

• static void **inactive** variables (const RealVector &config vars, Model &model)
  set the specified configuration to the Model’s inactive vars, converting from real to integer or through index to string value as needed

• static void **inactive** variables (const RealVector &config vars, Model &model, Variables &updated vars)

• static void evaluate (const RealMatrix &samples matrix, Model &model, RealMatrix &resp matrix)
  Bulk synchronously evaluate the model for each column in the samples matrix and return as columns of the response matrix.

Protected Member Functions

• Model (BaseConstructor, ProblemDescDB &problem db)
  constructor initializing the base class part of letter classes (BaseConstructor overloading avoids infinite recursion in the derived class constructors - Coplien, p. 139)

• Model (LightWtBaseConstructor, ProblemDescDB &problem db, ParallelLibrary &parallel_lib, const SharedVariablesData &svd, bool share_svd, const SharedResponseData &srd, bool share_srd, const ActiveSet &set, short output_level)
  constructor initializing base class for derived model class instances constructed on the fly

• Model (LightWtBaseConstructor, ProblemDescDB &problem db, ParallelLibrary &parallel_lib)
  constructor initializing base class for recast model instances

• virtual void derived evaluate (const ActiveSet &set)
  portion of evaluate() specific to derived model classes

• virtual void derived evaluate_nowait (const ActiveSet &set)
  portion of evaluate_nowait() specific to derived model classes

• virtual const IntResponseMap & derived synchronize ()
  portion of synchronize() specific to derived model classes

• virtual const IntResponseMap & derived synchronize_nowait ()
CHAPTER 14. CLASS DOCUMENTATION

- portion of synchronize_nowait() specific to derived model classes
- virtual void derived_init_communicators (ParLevLIter pl_iter, int max_eval_concurrency, bool recurse_flag=true) portion of init_communicators() specific to derived model classes
- virtual void derived_init_serial () portion of init_serial() specific to derived model classes
- virtual void derived_set_communicators (ParLevLIter pl_iter, int max_eval_concurrency, bool recurse_flag=true) portion of set_communicators() specific to derived model classes
- virtual void derived_free_communicators (ParLevLIter pl_iter, int max_eval_concurrency, bool recurse_flag=true) portion of free_communicators() specific to derived model classes
- void initialize_distribution (Pecos::MultivariateDistribution &mv_dist, bool active_only=false) initialize distribution types from problemDescDB
- void initialize_distribution_parameters (Pecos::MultivariateDistribution &mv_dist, bool active_only=false) initialize distribution parameters from problemDescDB
- void set_ie_asynchronous_mode (int max_eval_concurrency) default logic for defining asyncEvalFlag and evaluationCapacity based on ie_pl settings
- void assign_max_strings (const Pecos::MultivariateDistribution &mv_dist, Variables &vars) assign all of the longest possible string values into vars
- SSCIter max_string (const StringSet &ss) return iterator for longest string value found in string set
- SRMCIter max_string (const StringRealMap &srm) return iterator for longest string value found in string map
- SizetMultiArrayConstView initialize_x0_bounds (const SizetArray &original_dvv, bool &active_derivs, bool &inactive_derivs, RealVector &fd_lb, RealVector &fd_ub) const Initialize data needed for computing finite differences (active/inactive, center point, and bounds)
- Real forward_grad_step (size_t num_deriv_vars, size_t xj_index, Real x0_j, Real lb_j, Real ub_j) Computes the forward step for a finite difference gradient; updates shortStep.
- EvaluationsDBState evaluations_db_state (const Interface &interface) Return the interface flag for the EvaluationsDB state.
- EvaluationsDBState evaluations_db_state (const Model &model) Return the model flag for the EvaluationsDB state.
- void asynch_eval_store (const Interface &interface, const int &id, const Response &response) Store the response portion of an interface evaluation. Called from rekey_response_map.
- void asynch_eval_store (const Model &model, const int &id, const Response &response) Exists to support storage of interface evaluations. No-op so that rekey_response_map<Model> can be generated.
- template<typename MetaType > void rekey_response_map (MetaType &meta_object, const IntResponseMap &resp_map, IntIntMap &id_map, IntResponseMap &resp_map_rekey, bool deep_copy resp=false) rekey returned jobs matched in id_map into resp_map_rekey; unmatched jobs are cached within the meta_object
- template<typename MetaType > void rekey_synch (MetaType &meta_object, bool block, IntIntMap &id_map, IntResponseMap &resp_map_rekey, bool deep_copy resp=false) synchronize via meta_object and rekey returned jobs matched in id_map into resp_map_rekey; unmatched jobs are cached within the meta_object
**Static Protected Member Functions**

- static String `user_auto_id()`  
  return the next available model ID for no-ID user methods
- static String `no_spec_id()`  
  return the next available model ID for on-the-fly methods

**Protected Attributes**

- EvaluationsDBState `modelEvaluationsDBState`  
  Whether to write model evals to the evaluations DB.
- EvaluationsDBState `interfEvaluationsDBState`  
  Whether to write interface evals to the evaluations DB.
- Variables `currentVariables`  
  the set of current variables used by the model for performing function evaluations
- size_t `numDerivVars`  
  the number of active continuous variables used in computing most response derivatives (i.e., in places such as quasi-Hessians and response corrections where only the active continuous variables are supported)
- Response `currentResponse`  
  the set of current responses that holds the results of model function evaluations
- size_t `numFns`  
  the number of functions in `currentResponse`
- Constraints `userDefinedConstraints`  
  Explicit constraints on variables are maintained in the Constraints class hierarchy. Currently, this includes linear constraints and bounds, but could be extended in the future to include other explicit constraints which (1) have their form specified by the user, and (2) are not catalogued in Response since their form and coefficients are published to an iterator at startup.
- String `modelId`  
  model identifier string from the input file
- String `modelType`  
  type of model: simulation, nested, or surrogate
- String `surrogateType`  
  type of surrogate model: local, multipoint, global, or hierarchical
- String `gradientType`  
  type of gradient data: analytic, numerical, mixed, or none
- String `methodSource`  
  source of numerical gradient routine: dakota or vendor
- String `intervalType`  
  type of numerical gradient interval: central or forward
- String `hessianType`  
  type of Hessian data: analytic, numerical, quasi, mixed, or none
- RealVector `fdGradStepSize`  
  relative finite difference step size for numerical gradients
- String `fdGradStepType`
type of finite difference step to use for numerical gradient: relative - step length is relative to x absolute - step length is what is specified bounds - step length is relative to range of x

- RealVector fdHessByGradStepSize
  relative finite difference step size for numerical Hessians estimated using first-order differences of gradients

- RealVector fdHessByFnStepSize
  relative finite difference step size for numerical Hessians estimated using second-order differences of function values

- String fdHessStepType
  type of finite difference step to use for numerical Hessian: relative - step length is relative to x absolute - step length is what is specified bounds - step length is relative to range of x

- bool ignoreBounds
  option to ignore bounds when computing finite diffs

- bool centralHess
  option to use old 2nd-order finite diffs for Hessians

- bool warmStartFlag
  if in warm-start mode, don’t reset accumulated data (e.g., quasiHessians)

- bool supportsEstimDerivs
  whether model should perform or forward derivative estimation

- String quasiHessType
  quasi-Hessian type: bfgs, damped_bfgs, sr1

- IntSet gradIdAnalytic
  analytic id’s for mixed gradients

- IntSet gradIdNumerical
  numerical id’s for mixed gradients

- IntSet hessIdAnalytic
  analytic id’s for mixed Hessians

- IntSet hessIdNumerical
  numerical id’s for mixed Hessians

- IntSet hessIdQuasi
  quasi id’s for mixed Hessians

- IntArray messageLengths
  length of packed MPI buffers containing vars, vars/set, response, and PRPair

- bool mappingInitialized
  track use of initialize_mapping() and finalize_mapping()

- ProblemDescDB & probDescDB
  class member reference to the problem description database

- ParallelLibrary & parallelLib
  class member reference to the parallel library

- ParConfigIter modelPCIter
  the ParallelConfiguration node used by this Model instance

- short componentParallelMode
  the component parallelism mode: NO_PARALLEL_MODE, SURROGATE_MODE, MODE

- bool asynchEvalFlag
  flags asynch evaluations (local or distributed)
14.108. MODEL CLASS REFERENCE

- **int evaluationCapacity**
  
  capacity for concurrent evaluations supported by the Model

- **short outputLevel**
  
  output verbosity level: \{SILENT,QUIET,NORMAL,VERBOSE,DEBUG\}_OUTPUT

- **Pecos::MultivariateDistribution mvDist**
  
  the multivariate random variable distribution (in probability space corresponding to currentVariables)

- **BoolDeque primaryRespFnSense**
  
  array of flags (one per primary function) for switching the sense to maximize the primary function (default is minimize)

- **RealVector primaryRespFnWts**
  
  primary response function weightings (either weights for multiobjective optimization or weighted least squares)

- **bool hierarchicalTagging**
  
  whether to perform hierarchical evalID tagging of params/results

- **ScalingOptions scalingOpts**
  
  user-provided scaling data from the problem DB, possibly modified by Recasting

- **String evalTagPrefix**
  
  cached evalTag Prefix from parents to use at evaluate time

- **EvaluationStore & evaluationsDB**
  
  reference to the global evaluation database

### Private Member Functions

- **Model * get_model (ProblemDescDB &problem_db)**
  
  Used by the envelope to instantiate the correct letter class.

- **int estimate_derivatives (const ShortArray &map_asv, const ShortArray &fd_grad_asv, const ShortArray &fd_hess_asv, const ShortArray &quasi_hess_asv, const ActiveSet &original_set, const bool asynch_flag)**
  
  evaluate numerical gradients using finite differences. This routine is selected with "method_source dakota" (the default method_source) in the numerical gradient specification.

- **void synchronize_derivatives (const Variables &vars, const IntResponseMap &fd_responses, Response &new_response, const ShortArray &fd_grad_asv, const ShortArray &fd_hess_asv, const ShortArray &quasi_hess_asv, const ActiveSet &original_set)**
  
  combine results from an array of finite difference response objects (fd_grad_responses) into a single response (new_response)

- **void update_response (const Variables &vars, Response &new_response, const ShortArray &fd_grad_asv, const ShortArray &fd_hess_asv, const ShortArray &quasi_hess_asv, const ActiveSet &original_set, Response &initial_map_response, const RealMatrix &new_fn_grads, const RealSymMatrixArray &new_fn_hessians)**
  
  overlay results to update a response object

- **void update_quasi_hessians (const Variables &vars, Response &new_response, const ActiveSet &original_set)**
  
  perform quasi-Newton Hessian updates

- **bool manage_asv (const ActiveSet &original_set, ShortArray &map_asv_out, ShortArray &fd_grad_asv_out, ShortArray &fd_hess_asv_out, ShortArray &quasi_hess_asv_out)**
  
  Coordinates usage of estimate_derivatives() calls based on asv_in.

- **Real initialize_h (Real x_j, Real lb_j, Real ub_j, Real step_size, String step_type) const**
CHAPTER 14. CLASS DOCUMENTATION

- **Real FDstep1** (Real x0, Real lb, Real ub, Real h_mag)
  - function to determine initial finite difference h (before step length adjustment) based on type of step desired

- **Real FDstep2** (Real x0, Real lb, Real ub, Real h)
  - function returning finite-difference step size (affected by bounds)

**Private Attributes**

- **int modelEvalCntr**
  - evaluation counter for top-level `evaluate()` and `evaluate_nowait()` calls. Differs from lower level counters in case of numerical derivative estimation (several lower level evaluations are assimilated into a single higher level evaluation)

- **bool estDerivsFlag**
  - flags presence of estimated derivatives within a set of calls to `evaluate_nowait()`

- **bool shortStep**
  - flags finite-difference step size adjusted by bounds

- **std::map< SizetIntPair, ParConfigLIter > modelPCIterMap**
  - map<> used for tracking modelPCIter instances using depth of parallelism level and max evaluation concurrency as the lookup keys

- **bool initCommsBcastFlag**
  - flag for determining need to bcast the max concurrency from `init_communicators();` set from `IteratorScheduler::init_iterator()`

- **bool modelAutoGraphicsFlag**
  - flag for posting of graphics data within `evaluate()` (automatic graphics posting in the model as opposed to graphics posting at the strategy level)

- **IntVariablesMap varsMap**
  - history of vars populated in `evaluate_nowait()` and used in `synchronize()`.

- **std::list< ShortArray > asvList**
  - if `estimate_derivatives()` is used, transfers ASVs from `evaluate_nowait()` to `synchronize()`

- **std::list< ActiveSet > setList**
  - if `estimate_derivatives()` is used, transfers ActiveSets from `evaluate_nowait()` to `synchronize()`

- **BoolList initialMapList**
  - transfers initial map flag values from `estimate_derivatives()` to `synchronize_derivatives()`

- **BoolList dbCaptureList**
  - transfers db_capture flag values from `estimate_derivatives()` to `synchronize_derivatives()`

- **ResponseList dbResponseList**
  - transfers database captures from `estimate_derivatives()` to `synchronize_derivatives()`

- **RealList deltaList**
  - transfers deltas from `estimate_derivatives()` to `synchronize_derivatives()`

- **IntIntMap numFDEvalsMap**
  - tracks the number of evaluations used within `estimate_derivatives()`. Used in `synchronize()` as a key for combining finite difference responses into numerical gradients.

- **IntIntMap rawEvalIdMap**
maps from the raw evaluation ids returned by \texttt{derived\_synchronize()} and \texttt{derived\_synchronize\_nowait()} to the corresponding modelEvalCntr id. Used for rekeying responseMap.

- \texttt{RealVectorArray xPrev}
  previous parameter vectors used in computing \( s \) for quasi-Newton updates
- \texttt{RealMatrix fnGradsPrev}
  previous gradient vectors used in computing \( y \) for quasi-Newton updates
- \texttt{RealSymMatrixArray quasiHessians}
  quasi-Newton Hessian approximations
- \texttt{SizetArray numQuasiUpdates}
  number of quasi-Newton Hessian updates applied
- \texttt{IntResponseMap responseMap}
  used to return a map of responses for asynchronous evaluations in final concatenated form. The similar map in \texttt{Interface} contains raw responses.
- \texttt{IntResponseMap cachedResponseMap}
  caching of responses returned by \texttt{derived\_synchronize\_nowait()} but not matched within current rawEvalIdMap
- \texttt{IntResponseMap graphicsRespMap}
  used to cache the data returned from \texttt{derived\_synchronize\_nowait()} prior to sequential input into the graphics
- \texttt{IntSetArray activeDiscSetIntValues}
  aggregation of the admissible value sets for all active discrete set integer variables
- \texttt{StringSetArray activeDiscSetStringValues}
  aggregation of the admissible value sets for all active discrete set string variables
- \texttt{RealSetArray activeDiscSetRealValues}
  aggregation of the admissible value sets for all active discrete set real variables
- \texttt{BitArray discreteIntSets}
  key for identifying discrete integer set variables within the active discrete integer variables
- \texttt{short prevDSIView}
  previous view used in \texttt{discrete\_set\_int\_values}(view): avoids recomputation of activeDiscSetIntValues
- \texttt{short prevDSSView}
  previous view used in \texttt{discrete\_set\_string\_values}(view): avoids recomputation of activeDiscSetStringValues
- \texttt{short prevDSRView}
  previous view used in \texttt{discrete\_set\_real\_values}(view): avoids recomputation of activeDiscSetRealValues
- \texttt{ModelList modelList}
  used to collect sub-models for \texttt{subordinate\_models()}
- \texttt{BoolDeque recastFlags}
  a key indicating which models within a model recursion involve recasting
- \texttt{Model * modelRep}
  pointer to the letter (initialized only for the envelope)
- \texttt{int referenceCount}
  number of objects sharing modelRep

\textbf{Static Private Attributes}

- \texttt{static size\_t noSpecIdNum = 0}
  the last used model ID number for on-the-fly instantiations (increment before each use)
14.108.1 Detailed Description

Base class for the model class hierarchy.

The Model class is the base class for one of the primary class hierarchies in DAKOTA. The model hierarchy contains a set of variables, an interface, and a set of responses, and an iterator operates on the model to map the variables into responses using the interface. For memory efficiency and enhanced polymorphism, the model hierarchy employs the "letter/envelope idiom" (see Coplien "Advanced C++", p. 133), for which the base class (Model) serves as the envelope and one of the derived classes (selected in Model::get_model()) serves as the letter.

14.108.2 Constructor & Destructor Documentation

Model ( )

default constructor

The default constructor is used in vector<Model> instantiations and for initialization of Model objects contained in Iterator and derived Strategy classes. modelRep is NULL in this case (a populated problem_db is needed to build a meaningful Model object). This makes it necessary to check for NULL in the copy constructor, assignment operator, and destructor.

Model ( ProblemDescDB & problem_db )

standard constructor for envelope

Used in model instantiations within strategy constructors. Envelope constructor only needs to extract enough data to properly execute get_model, since Model(BaseConstructor, problem_db) builds the actual base class data for the derived models.

References Dakota::abort_handler(), Model::get_model(), and Model::modelRep.

Model ( const Model & model )

copy constructor

Copy constructor manages sharing of modelRep and incrementing of referenceCount.

References Model::modelRep, and Model::referenceCount.

~Model ( ) [virtual]

destructor

Destructor decrements referenceCount and only deletes modelRep when referenceCount reaches zero.

References Model::modelRep, and Model::referenceCount.
Model (BaseConstructor, ProblemDescDB & problem_db) [protected]

constructor initializing the base class part of letter classes (BaseConstructor overloading avoids infinite recursion in the derived class constructors - Coplien, p. 139)

This constructor builds the base class data for all inherited models. get_model() instantiates a derived class and the derived class selects this base class constructor in its initialization list (to avoid the recursion of the base class constructor calling get_model() again). Since the letter IS the representation, its representation pointer is set to NULL (an uninitialized pointer causes problems in ~Model).

References Dakota::abort_handler(), Model::currentResponse, Dakota::expand_for_fields_sdv(), Model::fdGradStepSize, Model::fdHessByFnStepSize, ProblemDescDB::get_rv(), ProblemDescDB::get_sa(), Model::gradIdNumerical, Model::gradientType, Model::hessianType, Model::hessIdNumerical, Model::initialize_distribution(), Model::initialize_distribution_parameters(), Model::modelId, Model::mvDist, Model::num-primary_fns(), Model::primaryRespFnSense, Model::primaryRespFnWts, Model::probDescDB, Response::shared_data(), Dakota::strbegins(), Dakota::strtolower(), and Model::user_auto_id().

Model (LightWtBaseConstructor, ProblemDescDB & problem_db, ParallelLibrary & parallel_lib) [protected]

constructor initializing base class for recast model instances

This constructor also builds the base class data for inherited models. However, it is used for recast models which are instantiated on the fly. Therefore it only initializes a small subset of attributes.

14.108.3 Member Function Documentation

Model operator= (const Model & model)

assignment operator


References Model::modelRep, and Model::referenceCount.

Iterator & subordinate_iterator () [virtual]

return the sub-iterator in nested and surrogate models

return by reference requires use of dummy objects, but is important to allow use of assign_rep() since this operation must be performed on the original envelope object.

Reimplemented in RecastModel, DataFitSurrModel, and NestedModel.

References Dakota::dummy_iterator, Model::modelRep, and Model::subordinate_iterator().

Referenced by NonDExpansion::append_expansion(), NonDMultilevelStochCollocation::assign_specification_sequence(), NonDMultilevelPolynomialChaos::assign_specification_sequence(), NonDExpansion::compute_expansion(), SurrBasedGlobalMinimizer::core_run(), NonDExpansion::decrement_grid(), NonDExpansion::decrement_order_and_grid(), NonDExpansion::finalize_sets(), NonDGlobalReliability::get_best_sample(), NonDExpansion::increment_grid(), NonDExpansion::increment_order_and_grid(), NonDExpansion::increment_sets(), NonDMultilevelStochCollocation::increment_specification_sequence(), NonDMultilevelPolynomialChaos::increment_specification_sequence(), NCSUOptimizer::initialize(), CONMINOptimizer::initialize(), NonDExpansion::initialize_expansion(), NonDExpansion::initialize_ml_regression(), NonDExpansion::initialize_u_space_grid(), NonDStochCollocation::initialize_u_space_model(), NonDMultilevelPolynomialChaos::initialize_u_space_model(), NonDExpansion::merge_grid(), NonDLocalInterval::NonDLocalInterval(), NonDLocalReliability::NonDLocalReliability(), NonDExpansion::pop_increment(), NonDExpansion::pre_refinement(), NonDExpansion::print_results(), NonDExpansion::push_increment(), NonDExpansion::select_candidate(), NonDExpansion::select_index_set_candidate(), SOLBase::SOLBase(), Model::subordinate-
iterator(), RecastModel::subordinate_iterator(), NonDExpansion::update_expansion(), NonDQUESOBayesCalibration::update_model(), NonDExpansion::update_model_from_samples(), and NonDExpansion::update_u_space_sampler().

**Model & subordinate_model ( ) [virtual]**

return a single sub-model defined from subModel in nested and recast models and truth_model() in surrogate models; used for a directed dive through model recursions that may bypass some components.

return by reference requires use of dummy objects, but is important to allow use of assign_rep() since this operation must be performed on the original envelope object.

Reimplemented in RecastModel, NestedModel, and SurrogateModel.

References Dakota::dummy_model, Model::modelRep, and Model::subordinate_model().

Referenced by NonDGlobalReliability::expected_feasibility(), NonDGlobalReliability::expected_improvement(), SurrogateModel::force_rebuild(), AdaptedBasisModel::get_sub_model(), Minimizer::initialize_run(), NonDExpansion::initialize_u_space_grid(), NonDGlobalReliability::optimize_gaussian_process(), Minimizer::original_model(), C-OLINOptimizer::post_run(), Optimizer::primary_resp_reducer(), Model::subordinate_model(), and DataFitSurrModel::update_global_reference().

**Model & surrogate_model ( ) [virtual]**

return the active approximation sub-model in surrogate models

return by reference requires use of dummy objects, but is important to allow use of assign_rep() since this operation must be performed on the original envelope object.

Reimplemented in RecastModel, DataFitSurrModel, and HierarchSurrModel.

References Dakota::dummy_model, Model::modelRep, and Model::surrogate_model().

Referenced by NonDAdaptiveSampling::calc_score_delta_y(), NonDAdaptiveSampling::calc_score_topo_alm_hybrid(), NonDAdaptiveSampling::calc_score_topo_avg_persistence(), NonDAdaptiveSampling::calc_score_topo_bottleneck(), NonDBayesCalibration::construct_mcmc_model(), NonDMultilevelSampling::control_variate_mc(), SurrBasedGlobalMinimizer::core_run(), SurrBasedLocalMinimizer::find_approx_response(), NonDMultilevelSampling::lf_increment(), NonDMultilevelSampling::multilevel_control_variate_mc_Ycorr(), NonDMultilevelSampling::multilevel_control_variate_mc_Ysum(), NonDAdaptiveSampling::output_round_data(), Model::surrogate_model(), and RecastModel::surrogate_model().

**Model & truth_model ( ) [virtual]**

return the active truth sub-model in surrogate models

return by reference requires use of dummy objects, but is important to allow use of assign_rep() since this operation must be performed on the original envelope object.

Reimplemented in RecastModel, DataFitSurrModel, and HierarchSurrModel.

References Model::modelRep, and Model::truth_model().

Referenced by SurrogateModel::activate_distribution_parameter_derivatives(), NonDMultilevelSampling::compute_error_estimates(), NonDMultilevelSampling::control_variate_mc(), SurrBasedGlobalMinimizer::core_run(), SurrogateModel::deactivate_distribution_parameter_derivatives(), SurrBasedLocalMinimizer::find_truth_response(), SurrogateModel::force_rebuild(), SurrBasedMinimizer::initialize_graphics(), NonDC3FunctionTrain::initialize_u_space_model(), NonDMultilevelFunctionTrain::initialize_u_space_model(), NonDMultilevelSampling::multilevel_control_variate_mc_Qcorr(), NonDMultilevelSampling::multilevel_control_variate_mc_Ycorr(), NonDMultilevelSampling::multilevel_mc_Qsum(), NonDMultilevelSampling::multilevel_mc_Ysum(), NonDBayesCalibration::NonDBayesCalibration(), NonDLocalReliability::NonDLocalReliability(), SurrBasedMinimizer::print_results(), NonDMultilevelSampling::print_results(), SurrogateModel::probability_transformation(), NonDMultilevelSampling::shared_increment(), SurrogateModel::subordinate_model(), SurrBasedGlobalMinimizer::SurrBasedGlobalMinimizer(), SurrBasedLocalMinimizer::SurrBasedLocalMinimizer(), SurrogateModel::trans_grad_U_to_X(), SurrogateModel::trans_grad_X_to_S(), Surrogate-
void update_from_subordinate_model ( size_t depth = std::numeric_limits<
size_t>::max() )
[virtual]

propagate vars/labels/bounds/targets from the bottom up

used only for instantiate-on-the-fly model recursions (all RecastModel instantiations and alternate DataFit-
SurrModel instantiations). Simulation, Hierarchical, and Nested Models do not redefine the function since they
do not support instantiate-on-the-fly. This means that the recursion will stop as soon as it encounters a Model
that was instantiated normally, which is appropriate since ProblemDescDB-constructed Models use top-down
information flow and do not require bottom-up updating.

Reimplemented in RecastModel, DataFitSurrModel, HierarchSurrModel, DataTransformModel, and Probability-
TransformModel.

References Model::modelRep, and Model::update_from_subordinate_model().

Interface & derived_interface ( ) [virtual]

return the interface employed by the derived model class, if present: SimulationModel::userDefinedInterface,
DataFitSurrModel::approxInterface, or NestedModel::optionalInterface

return by reference requires use of dummy objects, but is important to allow use of assign_rep() since this
operation must be performed on the original envelope object.

Reimplemented in RecastModel, DataFitSurrModel, NestedModel, and SimulationModel.

References Model::derived_interface(), Dakota::dummy_interface, and Model::modelRep.

Referenced by SurrBasedGlobalMinimizer::core_run(), Model::derived_interface(), and RecastModel::derived-
derived_interface().

size_t solution_levels ( bool lwr_bnd = true ) const [virtual]

number of discrete levels within solution control (SimulationModel)

return the number of levels within a solution / discretization hierarchy.

Reimplemented in SimulationModel.

References Model::modelRep, and Model::solution_levels().

Referenced by NonDMultilevelSampling::compute_error_estimates(), NonDMultilevelSampling::multilevel-
control_variate_mc_Qcorr(), NonDMultilevelSampling::multilevel_control_variate_mc_Ycorr(), NonDMultilevel-
Sampling::multilevel_mc_Qsum(), NonDMultilevelSampling::multilevel_mc_Ysum(), HierarchSurrModel::recursive-
apply(), Model::solution_levels(), and RecastModel::solution_levels().

void solution_level_index ( unsigned short index ) [virtual]

activate a particular level within the solution level control and return the cost estimate (SimulationModel)

activate a particular level within a solution / discretization hierarchy.

Reimplemented in RecastModel, and SimulationModel.

References Dakota::abort_handler(), Model::modelRep, and Model::solution_level_index().
Referenced by HierarchSurrModel::build_approximation(), HierarchSurrModel::derived_evaluate(), HierarchSurrModel::derived_evaluate_nowait(), Model::solution_level_index(), and RecastModel::solution_level_index().

short local_eval_synchronization( ) [virtual]
return derived model synchronization setting

SimulationModels and HierarchSurrModels redefine this virtual function. A default value of "synchronous" prevents async local operations for:

- NestedModels: a subiterator can support message passing parallelism, but not async local.
- DataFitSurrModels: while async evals on approximations will work due to some added bookkeeping, avoiding them is preferable.

Reimplemented in RecastModel, NestedModel, and SimulationModel.
References Model::local_eval_synchronization(), and Model::modelRep.

Referenced by Model::init_serial(), Model::local_eval_synchronization(), RecastModel::local_eval_synchronization(), and Model::set_ie_asyncynchronous_mode().

int local_eval_concurrency( ) [virtual]
return derived model asynchronous evaluation concurrency

SimulationModels and HierarchSurrModels redefine this virtual function.
Reimplemented in RecastModel, NestedModel, and SimulationModel.
References Model::local_eval_concurrency(), and Model::modelRep.

Referenced by Model::local_eval_concurrency(), RecastModel::local_eval_concurrency(), and Model::set_ie_asyncynchronous_mode().

const String & interface_id( ) const [virtual]
return the interface identifier

return by reference requires use of dummy objects, but is important to allow use of assign_rep() since this operation must be performed on the original envelope object.

Reimplemented in RecastModel, DataFitSurrModel, NestedModel, and SimulationModel.
References Dakota::dummy_interface, Interface::interface_id(), Model::interface_id(), and Model::modelRep.

Referenced by NonDDREAMBayesCalibration::archive_acceptance_chain(), Minimizer::archive_best_results(), DataTransformModel::archive_submodel_responses(), DataFitSurrModel::build_global(), NonDMUQBayesCalibration::cache_chain(), NonDQUESOBayesCalibration::cache_chain(), HierarchSurrModel::check_model_interface_instance(), DataFitSurrModel::DataFitSurrModel(), Model::db_lookup(), Model::estimate_message_lengths(), Model::evaluate(), NonDMultilevelSampling::export_all_samples(), NonDBayesCalibration::export_chain(), NonDBayesCalibration::export_discrepancy(), NonDBayesCalibration::export_field_discrepancy(), SurrBasedLocalMinimizer::find_approx_response(), SurrBasedLocalMinimizer::find_truth_response(), DataFitSurrModel::import_points(), Model::interface_id(), RecastModel::interface_id(), Minimizer::local_recast_retrieve(), Analyzer::pre_output(), SurrBasedMinimizer::print_results(), LeastSq::print_results(), Optimizer::print_results(), SeqHybridMetaIterator::run_sequential(), DiscrepancyCorrection::search_db(), Model::synchronize(), Model::synchronize_nowait(), Analyzer::update_best(), ConcurrentMetaIterator::update_local_results(), SeqHybridMetaIterator::update_local_results(), and NonDLocalReliability::update_mpp_search_data().
bool evaluation_cache ( bool recurse_flag = true ) const [virtual]

Indicates the usage of an evaluation cache by the Model.

Only Models including ApplicationInterfaces support an evaluation cache: surrogate, nested, and recast mappings are not stored in the cache. Possible exceptions: HierarchSurrModel, NestedModel::optionalInterface.

Reimplemented in RecastModel, DataFitSurrModel, HierarchSurrModel, and SimulationModel.

References Model::evaluation_cache(), and Model::modelRep.

Referenced by DataFitSurrModel::DataFitSurrModel(), DataFitSurrModel::evaluation_cache(), Model::evaluation_cache(), RecastModel::evaluation_cache(), DataFitSurrModel::import_points(), and Analyzer::read_variables_responses().

bool restart_file ( bool recurse_flag = true ) const [virtual]

Indicates the usage of a restart file by the Model.

Only Models including ApplicationInterfaces interact with the restart file: surrogate, nested, and recast mappings are not stored in restart. Possible exceptions: DataFitSurrModel::import_points(), NestedModel::optionalInterface.

Reimplemented in RecastModel, DataFitSurrModel, HierarchSurrModel, and SimulationModel.

References Model::modelRep, and Model::restart_file().

Referenced by DataFitSurrModel::import_points(), Analyzer::read_variables_responses(), DataFitSurrModel::restart_file(), Model::restart_file(), and RecastModel::restart_file().

void eval_tag_prefix ( const String & eval_id_str ) [virtual]

set the hierarchical eval ID tag prefix

Derived classes containing additional models or interfaces should implement this function to pass along to their sub Models/Interfaces.

Reimplemented in RecastModel, and SimulationModel.

References Model::eval_tag_prefix(), Model::evalTagPrefix, and Model::modelRep.

Referenced by HierarchSurrModel::build_approximation(), HierarchSurrModel::derived_evaluate(), DataFitSurrModel::derived_evaluate(), HierarchSurrModel::derived_evaluate_nowait(), DataFitSurrModel::derived_evaluate_nowait(), Iterator::eval_tag_prefix(), Model::eval_tag_prefix(), and RecastModel::eval_tag_prefix().

ModelList & subordinate_models ( bool recurse_flag = true )

return the sub-models in nested and surrogate models

since modelList is built with list insertions (using envelope copies), these models may not be used for model::assign_rep() since this operation must be performed on the original envelope object. They may, however, be used for letter-based operations (including assign_rep() on letter contents such as an interface).

References Model::derived_subordinate_models(), Model::modelList, Model::modelRep, and Model::subordinate_models().

Referenced by NonDExpansion::configure_cost(), NonDExpansion::configure_sequence(), NCSUOptimizer::initialize(), CONMINOptimizer::initialize(), Model::manage_data_recastings(), NonDLocalInterval::NonDLocalInterval(), NonDLocalReliability::NonDLocalReliability(), NonDMultilevelSampling::NonDMultilevelSampling(), SOLBase::SOLBase(), and Model::subordinate_models().

void init_communicators ( ParLevLIter pl_iter, int max_eval_concurrency, bool recurse_flag = true )

allocate communicator partitions for a model and store configuration in modelPIterMap
The `init_communicators()` and `derived_init_communicators()` functions are structured to avoid performing the messageLengths estimation more than once. `init_communicators()` (not virtual) performs the estimation and then forwards the results to `derived_init_communicators` (virtual) which uses the data in different contexts.

References `ParallelLibrary::bcast()`, `Model::derived_init_communicators()`, `Model::estimate_message_lengths()`, `ParallelLibrary::increment_parallel_configuration()`, `Model::init_communicators()`, `Model::initCommsBcastFlag`, `Model::messageLengths`, `Model::modelPCIter`, `Model::modelPCIterMap`, `Model::modelRep`, `ParallelLibrary::parallel_configuration_iterator()`, `ParallelLibrary::parallel_level_index()`, and `Model::parallelLib`.

Referenced by `SurrBasedMinimizer::derived_init_communicators()`, `NonDGlobalReliability::derived_init_communicators()`, `NonDGlobalInterval::derived_init_communicators()`, `NonDExpansion::derived_init_communicators()`, `NonDAdaptImpSampling::derived_init_communicators()`, `NonDGPImpSampling::derived_init_communicators()`, `AdaptedBasisModel::derived_init_communicators()`, `NonDAdaptiveSampling::derived_init_communicators()`, `NonDBayesCalibration::derived_init_communicators()`, `NonDPolynomialChaos::derived_init_communicators()`, `HierarchSurrModel::derived_init_communicators()`, `DataFitSurrModel::derived_init_communicators()`, `RecastModel::derived_init_communicators()`, `Iterator::derived_init_communicators()`, `DataFitSurrModel::derived_set_communicators()`, `Model::init_communicators()`, and `Model::serve_init_communicators()`.

```cpp
void init_serial()
```

for cases where `init_communicators()` will not be called, modify some default settings to behave properly in serial.

The `init_serial()` and `derived_init_serial()` functions are structured to separate base class (common) operations from derived class (specialized) operations.

References `Model::asynchEvalFlag`, `Model::derived_init_serial()`, `Model::init_serial()`, `Model::local_eval_synchronization()`, and `Model::modelRep`.

Referenced by `NestedModel::derived_init_serial()`, `HierarchSurrModel::derived_init_serial()`, `DataFitSurrModel::derived_init_serial()`, `RecastModel::derived_init_serial()`, `Model::init_serial()`.

```cpp
void estimate_message_lengths()
```

estimate messageLengths for a model

This functionality has been pulled out of `init_communicators()` and defined separately so that it may be used in those cases when messageLengths is needed but `model.init_communicators()` is not called, e.g., for the master processor in the self-scheduling of a concurrent iterator strategy.

References `Response::active_set_derivative_vector()`, `Model::assign_max_strings()`, `Response::copy()`, `Variables::copy()`, `Model::currentResponse`, `Model::currentVariables`, `Variables::cv()`, `Model::estimate_message_lengths()`, `Variables::icv()`, `Model::interface_id()`, `Model::messageLengths`, `Model::modelRep`, `ParallelLibrary::mpirun_flag()`, `Model::mvDist`, `Model::numFns`, `Model::parallelLib`, `MPIPackBuffer::reset()`, and `MPIPackBuffer::size()`.

Referenced by `Model::estimate_message_lengths()`, `Model::init_communicators()`, `RandomFieldModel::initialize_mapping()`, `RecastModel::initialize_mapping()`, `ConcurrentMetaIterator::pre_run()`, `Iterator::resize_communicators()`, `AdaptedBasisModel::serve_init_mapping()`, and `ActiveSubspaceModel::serve_init_mapping()`.

```cpp
bool manage_data_recastings()
```

initialize modelList and recastFlags for data import/export

Constructor helper to manage model recastings for data import/export.

References `Model::manage_data_recastings()`, `Model::modelRep`, `Model::recastFlags`, and `Model::subordinate_models()`.

Referenced by `DataFitSurrModel::DataFitSurrModel()`, `Model::manage_data_recastings()`, and `Analyzer::read_variables_responses()`.
void assign_rep ( Model * model_rep, bool ref_count_incr = true )

replaces existing letter with a new one

Similar to the assignment operator, the assign_rep() function decrements referenceCount for the old model-Rep and assigns the new modelRep. It is different in that it is used for publishing derived class letters to existing envelopes, as opposed to sharing representations among multiple envelopes (in particular, assign_rep is passed a letter object and operator= is passed an envelope object). Letter assignment supports two models as governed by ref_count_incr:

- ref_count_incr = true (default): the incoming letter belongs to another envelope. In this case, increment the reference count in the normal manner so that deallocation of the letter is handled properly.

- ref_count_incr = false: the incoming letter is instantiated on the fly and has no envelope. This case is modeled after get_model(): a letter is dynamically allocated using new and passed into assign_rep, the letter’s reference count is not incremented, and the letter is not remotely deleted (its memory management is passed over to the envelope).

References Dakota::abort_handler(), Model::model_rep(), Model::modelRep, and Model::referenceCount.

Referenced by ActiveSubspaceModel::build_surrogate(), NonDBayesCalibration::calibrate_to_hif(), Active-SubspaceModel::computeCrossValidationMetric(), NonDBayesCalibration::construct_map_model(), NonDBayesCalibration::construct_mcmc_model(), DataFitSurrModel::DataFitSurrModel(), EffGlobalMinimizer::EffGlobal-Reductizer(), ActiveSubspaceModel::get_sub_model(), SurrBasedLocalMinimizer::initialize_sub_model(), Non-DAdaptImpSampling::NonDAdaptImpSampling(), NonDAdaptiveSampling::NonDAdaptiveSampling(), Non-DBayesCalibration::NonDBayesCalibration(), NonDC3FunctionTrain::NonDC3FunctionTrain(), NonDGlobalInterval-NonDGlobalInterval(), NonDGlobalReliability::NonDGlobalReliability(), NonDGPImpSampling::NonDGPImp-NonDGPImpSampling(), NonDGPlocalTrain::NonDGPlocalTrain(), NonDLocalReliability::NonDLocalReliability(), NonD-LocalFunctionTrain::NonDLocalFunctionTrain(), NonDLocalPolynomialChaos::NonDLocalPolynomialChaos(), NonDStochCollocation::NonDStochCollocation(), NonDStochCollocation::NonDStochCollocation(), Optimizer::reduce_model(), Non-DPolynomialChaos::NonDPolynomialChaos(), and NonDStochCalibration::NonDStochCalibration::weight_model().

int derivative_concurrency ( ) const

return the gradient concurrency for use in parallel configuration logic

This function assumes derivatives with respect to the active continuous variables. Therefore, concurrency with respect to the inactive continuous variables is not captured.

References Dakota::contains(), Model::derivative_concurrency(), Model::gradIdAnalytic, Model::gradientType, Model::hessianType, Model::hessIdNumerical, Model::intervalType, Model::methodSource, Model::modelRep, and Model::numDerivVars.

Referenced by ActiveSubspaceModel::ActiveSubspaceModel(), Model::derivative_concurrency(), Hierarch-SurrModel::derived_free_communicators(), HierarchSurrModel::derived_init_communicators(), DataFitSurrModel::derived_init_communicators(), HierarchSurrModel::derived_set_communicators(), DataFitSurrModel::estimate_partition_bounds(), NonDExpansion::initialize_u_space_grid(), Analyzer::num_samples(), HierarchSurrModel::serve_run(), and Iterator::update_from_model().

void active_variables ( const RealVector & config_vars, Model & model ) [static]

set the specified configuration to the Model’s inactive vars, converting from real to integer or through index to string value as needed

config_vars consists of [continuous, integer, string, real].
References Model::continuous_variables(), Model::current_variables(), Model::cv(), Model::discrete_int_variables(), Model::discrete_real_variables(), Model::discrete_set_string_values(), Variables::discrete_string_variable(), Model::div(), Model::drv(), Model::dsv(), Dakota::iround(), and Dakota::set_index_to_value().

void inactive_variables ( const RealVector & config_vars, Model & model ) [static]
set the specified configuration to the Model's inactive vars, converting from real to integer or through index to string value as needed
  config_vars consists of [continuous, integer, string, real].
  References Model::current_variables().
  Referenced by DataTransformModel::archive_submodel_responses(), NonDBayesCalibration::calibrate_to_hifi(), DataTransformModel::derived_evaluate(), DataTransformModel::derived_evaluate_nowait(), NonDBayesCalibration::export_discrepancy(), and NonDBayesCalibration::export_field_discrepancy().

void inactive_variables ( const RealVector & config_vars, Model & model, Variables & vars ) [static]
config_vars consists of [continuous, integer, string, real].
  References Model::current_variables(), Model::discrete_set_string_values(), Model::icv(), Model::idiv(), Model::idrv(), Model::idsv(), Variables::inactive_continuous_variables(), Variables::inactive_discrete_int_variables(), Variables::inactive_discrete_real_variables(), Variables::inactive_discrete_string_variable(), Dakota::iround(), Dakota::set_index_to_value(), and Variables::view().

String user_auto_id ( ) [static], [protected]
return the next available model ID for no-ID user methods
  Rationale: The parser allows multiple user-specified models with empty (unspecified) ID. However, only a single Model with empty ID can be constructed (if it's the only one present, or the "last one parsed"). Therefore decided to prefer NO_MODEL_ID over NO MODEL_ID <num> for (some) consistency with interface NO_ID convention. MODEL was inserted in the middle to distinguish "anonymous" MODELS from methods and interfaces in the hdf5 output. Note that this function is not used to name recast models; see their constructors for how it's done.
  Referenced by Model::Model().

String no_spec_id ( ) [static], [protected]
return the next available model ID for on-the-fly methods
  Rationale: For now NOSPEC_MODEL_ID is chosen due to historical id="NO_SPECIFICATION" used for internally-constructed Models. Longer-term, consider auto-generating an ID that includes the context from which the method is constructed, e.g., the parent method or model's ID, together with its name. Note that this function is not used to name recast models; see their constructors for how it's done.
  References Model::noSpecIdNum.

void initialize_distribution ( Pecos::MultivariateDistribution & mv_dist, bool active_only = false ) [protected]
initialize distribution types from problemDescDB
  Build random variable distribution types and active subset. This function is used when the Model variables are in x-space.
  References SharedVariablesData::active_subsets(), Dakota::assign_value(), Model::currentVariables, Variables::cv(), Variables::div(), Variables::drv(), Variables::dsv(), Model::dsv(), ProblemDescDB::get_rv(), ProblemDescDB::get_sizet(), Model::probDescDB, Variables::shared_data(), and Variables::tv().

References Model::continuous_variables(), Model::current_variables(), Model::cv(), Model::discrete_int_variables(), Model::discrete_real_variables(), Model::discrete_set_string_values(), Variables::discrete_string_variable(), Model::div(), Model::drv(), Model::dsv(), Dakota::iround(), and Dakota::set_index_to_value().
Referenced by Model::Model().

Model * get_model ( ProblemDescDB & problem_db ) [private]

Used by the envelope to instantiate the correct letter class.
Used only by the envelope constructor to initialize modelRep to the appropriate derived type, as given by the modelType attribute.
References ProblemDescDB::get_string(), Model::model_type(), and Model::modelType.
Referenced by Model::Model().

int estimate_derivatives ( const ShortArray & asv, const ShortArray & fd_grad_asv, const ShortArray & fd_hess_asv, const ShortArray & quasi_hess_asv, const ActiveSet & original_set, const bool asynch_flag ) [private]

evaluate numerical gradients using finite differences. This routine is selected with ”method_source dakota” (the default method_source) in the numerical gradient specification.
Estimate derivatives by computing finite difference gradients, finite difference Hessians, and/or quasi-Newton Hessians. The total number of finite difference evaluations is returned for use by synchronize() to track response arrays, and it could be used to improve management of max_function_evaluations within the iterators. ! new logic
References Variables::all_continuous_variables(), Model::centralHess, Variables::continuous_variables(), Model::currentResponse, Model::currentVariables, Model::dbLookup(), Model::dbCaptureList, Model::dbResponseList, Model::deltaList, ActiveSet::derivative_vector(), Model::derived_evaluate(), Model::derived_evaluate_nowait(), Model::fdHessByFnStepSize, Model::fdHessByGradStepSize, Model::fdHessType, Model::FDstep1(), Model::FDstep2(), Dakota::find_index(), Model::forward_grad_step(), Response::function_gradients(), Response::function_values(), Model::ignoreBounds, Variables::inactive_continuous_variables(), Model::initialize_h(), Model::initialize_x0_bounds(), Model::initializeMapList, Model::intervalType, Model::numFns, Model::outputLevel, ActiveSet::request_vector(), Response::shared_data(), Model::shortStep, and Model::update_response.
Referenced by Model::evaluate(), and Model::evaluate_nowait().

void synchronize_derivatives ( const Variables & vars, const IntResponseMap & fd_responses, Response & new_response, const ShortArray & fd_grad_asv, const ShortArray & fd_hess_asv, const ShortArray & quasi_hess_asv, const ActiveSet & original_set ) [private]

combine results from an array of finite difference response objects (fd_grad_responses) into a single response (new_response)
Merge an array of fd_responses into a single new_response. This function is used both by synchronous evaluate() for the case of asynchronous estimate_derivatives() and by synchronize() for the case where one or more evaluate_nowait() calls has employed asynchronous estimate_derivatives(). !
References Model::acv(), Variables::all_continuous_variable_ids(), Model::centralHess, Variables::continuous_variable_ids(), Model::currentResponse, Model::currentVariables, Model::cv(), Model::dbCaptureList, Model::dbResponseList, Model::deltaList, ActiveSet::derivative_vector(), Dakota::find_index(), Response::function_gradients(), Response::function_values(), Model::icv(), Variables::inactive_continuous_variable_ids(), Model::initializeMapList, Model::intervalType, Model::numFns, Response::shared_data(), and Model::update_response().
Referenced by Model::evaluate(), and Model::synchronize().

void update_response ( const Variables & vars, Response & new_response, const ShortArray & fd_grad_asv, const ShortArray & fd_hess_asv, const ShortArray & quasi_hess_asv, const ActiveSet & original_set, Response & initial_map_response, const RealMatrix & new_fn_grads, const RealSymMatrixArray & new_fn_hessians ) [private]

overlay results to update a response object
Overlay the initial_map_response with numerically estimated new_fn_grads and new_fn_hessians to populate new_response as governed by asv vectors. Quasi-Newton secant Hessian updates are also performed here, since this is where the gradient data needed for the updates is first consolidated. Convenience function used by estimate_derivatives() for the synchronous case and by synchronize_derivatives() for the asynchronous case.

References Response::active_set.request_vector(), Variables::continuous_variable_ids(), Response::copy(), Model::currentResponse, Model::currentVariables, ActiveSet::derivative_vector(), Response::function_gradients(), Response::function_values(), Model::hessianType, Model::hessIdQuasi, Response::is_null(), Model::numFns, Model::outputLevel, Model::quasiHessians, ActiveSet::request_vector(), Response::reset_inactive(), Model::supportsEstimDerivs, Model::surrogate_response_mode(), and Model::update_quasi_hessians().

Referenced by Model::estimate_derivatives(), and Model::synchronize_derivatives().

```cpp
void update_quasi_hessians ( const Variables & vars, Response & new_response, const ActiveSet & original_set ) [private]
```

perform quasi-Newton Hessian updates

quasi-Newton updates are performed for approximating response function Hessians using BFGS or SR1 formulations. These Hessians are supported only for the active continuous variables, and a check is performed on the DVV prior to invoking the function.

References Dakota::contains(), Variables::continuous_variables(), Dakota::copy_data(), Model::fnGradsPrev, Response::function_gradients(), Model::hessianType, Model::hessIdQuasi, Model::modelType, Model::numDerivVars, Model::numFns, Model::numQuasiUpdates, Model::outputLevel, Model::quasiHessians, Model::quasiHessType, ActiveSet::request_vector(), and Model::xPrev.

Referenced by Model::update_response().

```cpp
bool manage_asv ( const ActiveSet & original_set, ShortArray & map_asv_out, ShortArray & fd_grad_asv_out, ShortArray & fd_hess_asv_out, ShortArray & quasi_hess_asv_out ) [private]
```

Coordinates usage of estimate_derivatives() calls based on asv_in.

Splits asv_in total request into map_asv_out, fd_grad_asv_out, fd_hess_asv_out, and quasi_hess_asv_out as governed by the responses specification. If the returned use_est_deriv is true, then these asv outputs are used by estimate_derivatives() for the initial map, finite difference gradient evals, finite difference Hessian evals, and quasi-Hessian updates, respectively. If the returned use_est_deriv is false, then only map_asv_out is used.

References Dakota::abort_handler(), Dakota::contains(), ActiveSet::derivative_vector(), Model::FDstep2(), Dakota::find_index(), Model::forward_grad_step(), Model::gradIdAnalytic, Model::gradIdNumerical, Model::gradientType, Model::hessianType, Model::hessIdAnalytic, Model::hessIdNumerical, Model::hessIdQuasi, Model::ignoreBounds, Model::initialize_x0_bounds(), Model::intervalType, Model::methodSource, Model::model_id(), ActiveSet::request_vector(), Model::shortStep, Model::supportsEstimDerivs, and Model::surrogate_response_mode().

Referenced by Model::evaluate(), and Model::evaluate_nowait().

```cpp
Real initialize_h ( Real x_j, Real lb_j, Real ub_j, Real step_size, String step_type ) const [private]
```

function to determine initial finite difference h (before step length adjustment) based on type of step desired

Auxiliary function to determine initial finite difference h (before step length adjustment) based on type of step desired.

Referenced by Model::estimate_derivatives(), and Model::forward_grad_step().

```cpp
Real FDstep1 ( Real x0_j, Real lb_j, Real ub_j, Real h_mag ) [private]
```

function returning finite-difference step size (affected by bounds)
Auxiliary function to compute forward or first central-difference step size, honoring bounds. The first step is away from zero, when possible. Flips the direction or updates shortStep if can’t take the full requested step h_mag.

References Model::ignoreBounds, and Model::shortStep.
Referenced by Model::estimate_derivatives(), and Model::forward_grad_step().

Real FDstep2 ( Real x0_j, Real lb_j, Real ub_j, Real h ) [private]
function returning second central-difference step size (affected by bounds)
Auxiliary function to compute the second central-difference step size, honoring bounds.
References Model::ignoreBounds, and Model::shortStep.
Referenced by Model::estimate_derivatives(), and Model::manage_asv().

14.108.4 Member Data Documentation

RealVector fdGradStepSize [protected]
relative finite difference step size for numerical gradients
A scalar value (instead of the vector fd_gradient_step_size spec) is used within the iterator hierarchy since this attribute is only used to publish a step size to vendor numerical gradient algorithms.

Referenced by DataFitSurrModel::DataFitSurrModel(), Model::fd_gradient_step_size(), Model::forward_grad_step(), RecastModel::initialize_data_from_submodel(), and Model::Model().

RealVector fdHessByGradStepSize [protected]
relative finite difference step size for numerical Hessians estimated using first-order differences of gradients
For vendor numerical Hessian algorithms, a scalar value is used.

Referenced by DataFitSurrModel::DataFitSurrModel(), Model::estimate_derivatives(), Model::fd_hessian_by_grad_step_size(), RecastModel::initialize_data_from_submodel(), and Model::Model().

RealVector fdHessByFnStepSize [protected]
relative finite difference step size for numerical Hessians estimated using second-order differences of function values
For vendor numerical Hessian algorithms, a scalar value is used.

Referenced by DataFitSurrModel::DataFitSurrModel(), Model::estimate_derivatives(), Model::fd_hessian_by_fn_step_size(), RecastModel::initialize_data_from_submodel(), and Model::Model().

ProblemDescDB& probDescDB [protected]
class member reference to the problem description database

Iterator and Model cannot use a shallow copy of ProblemDescDB due to circular destruction dependency (reference counts can’t get to 0), since ProblemDescDB contains {iterator,model} List.

Referenced by ActiveSubspaceModel::ActiveSubspaceModel(), NestedModel::derived_init_communicators(), HierarchSurrModel::derived_init_communicators(), DataFitSurrModel::derived_init_communicators(), NestedModel::derived_init_serial(), SimulationModel::estimate_partition_bounds(), NestedModel::estimate_partition_bounds(), HierarchSurrModel::estimate_partition_bounds(), DataFitSurrModel::estimate_partition_bounds(), NestedModel::init_sub_iterator(), Model::initialize_distribution(), Model::initialize_distribution_parameters(), Model::Model(), and Model::problem_description_db().

The documentation for this class was generated from the following files:
14.109 MPIManager Class Reference

Class MPIManager to manage Dakota’s MPI world, which may be a subset of MPI_COMM_WORLD.

Public Member Functions

- **MPIManager ()**
  Default constructor; Dakota will not call MPI_Init.
- **MPIManager (int &argc, char **&argv)**
  Command-line constructor; parses MPI arguments during call to MPI_Init.
- **MPIManager (MPI_Comm dakota_mpi_comm)**
  Construct on specified MPI_Comm.
- **~MPIManager ()**
  Destructor: calls finalize if Dakota owns MPI
- **MPI_Comm dakota_mpi_comm () const**
  Get the MPI Comm on which Dakota is running
- **int world_rank () const**
  Get the rank of this process in Dakota’s MPI Comm
- **int world_size () const**
  Get the size of the MPI Comm on which Dakota is running
- **bool mpirun_flag () const**
  True when Dakota is running in MPI mode

Static Public Member Functions

- **static bool detect_parallel_launch (int &argc, char **&argv)**
  Detect parallel launch of Dakota using mpirun/mpiexec/poe/etc. based on command line arguments and environment variables

Private Attributes

- **MPI_Comm dakotaMPIComm**
  MPI Comm on which DAKOTA is running.
- **int dakotaWorldRank**
  Rank in MPI Comm in which DAKOTA is running
- **int dakotaWorldSize**
  Size of MPI Comm in which DAKOTA is running
- **bool mpirunFlag**
  Flag for a parallel mpirun/yod launch
- **bool ownMPIFlag**
  Flag for ownership of MPI_Init/MPI_Finalize
14.109.1 Detailed Description

Class MPIManager to manage Dakota’s MPI world, which may be a subset of MPI_COMM_WORLD.

The documentation for this class was generated from the following files:

- MPIManager.hpp
- MPIManager.cpp

14.110 MPIPackBuffer Class Reference

Class for packing MPI message buffers.

Public Member Functions

- **MPIPackBuffer (int size=1024)**
  
  Constructor, which allows the default buffer size to be set.

- **~MPIPackBuffer ()**
  
  Destructor.

- **const char ∗ buf ()**
  
  Returns a pointer to the internal buffer that has been packed.

- **int size ()**
  
  The number of bytes of packed data.

- **int capacity ()**
  
  The allocated size of Buffer.

- **void reset ()**
  
  Resets the buffer index in order to reuse the internal buffer.

- **void pack (const int ∗ data, const int num=1)**
  
  Pack one or more int's.

- **void pack (const u_int ∗ data, const int num=1)**
  
  Pack one or more unsigned int's.

- **void pack (const long ∗ data, const int num=1)**
  
  Pack one or more long's.

- **void pack (const u_long ∗ data, const int num=1)**
  
  Pack one or more unsigned long's.

- **void pack (const short ∗ data, const int num=1)**
  
  Pack one or more short's.

- **void pack (const u_short ∗ data, const int num=1)**
  
  Pack one or more unsigned short's.

- **void pack (const char ∗ data, const int num=1)**
  
  Pack one or more char's.

- **void pack (const u_char ∗ data, const int num=1)**
  
  Pack one or more unsigned char's.

- **void pack (const double ∗ data, const int num=1)**
  
  Pack one or more double's.

- **void pack (const float ∗ data, const int num=1)**
  
  Pack one or more float's.
Pack one or more float's.
- void pack (const bool *data, const int num=1)

Pack one or more bool's.
- void pack (const int &data)

Pack a int.
- void pack (const u_int &data)

Pack a unsigned int.
- void pack (const long &data)

Pack a long.
- void pack (const u_long &data)

Pack a unsigned long.
- void pack (const short &data)

Pack a short.
- void pack (const u_short &data)

Pack a unsigned short.
- void pack (const char &data)

Pack a char.
- void pack (const u_char &data)

Pack a unsigned char.
- void pack (const double &data)

Pack a double.
- void pack (const float &data)

Pack a float.
- void pack (const bool &data)

Pack a bool.

Protected Member Functions
- void resize (const int newsize)
  
  Resizes the internal buffer.

Protected Attributes
- char * Buffer
  
  The internal buffer for packing.
- int Index
  
  The index into the current buffer.
- int Size
  
  The total size that has been allocated for the buffer.
14.110.1 Detailed Description

Class for packing MPI message buffers.

A class that provides a facility for packing message buffers using the MPI_Pack facility. The MPIPackBuffer class dynamically resizes the internal buffer to contain enough memory to pack the entire object. When deleted, the MPIPackBuffer object deletes this internal buffer. This class is based on the Dakota_Version_3.0 version of utilib::PackBuffer from utilib/src/io/PackBuf.[cpp,h]

The documentation for this class was generated from the following files:

- MPIPackBuffer.hpp
- MPIPackBuffer.cpp

14.111 MPIUnpackBuffer Class Reference

Class for unpacking MPI message buffers.

Public Member Functions

- void setup (char *buf, int size, bool flag=false)
  
  Method that does the setup for the constructors.

- MPIUnpackBuffer ()
  
  Default constructor.

- MPIUnpackBuffer (int size)
  
  Constructor that specifies the size of the buffer.

- MPIUnpackBuffer (char *buf, int size, bool flag=false)
  
  Constructor that sets the internal buffer to the given array.

- ~MPIUnpackBuffer ()
  
  Destructor.

- void resize (const int newsize)
  
  Resize the internal buffer.

- const char *buf ()
  
  Returns a pointer to the internal buffer.

- int size ()
  
  Returns the length of the buffer.

- int curr ()
  
  Returns the number of bytes that have been unpacked from the buffer.

- void reset ()
  
  Resets the index of the internal buffer.

- void unpack (int *data, const int num=1)
  
  Unpack one or more int's.

- void unpack (u_int *data, const int num=1)
  
  Unpack one or more unsigned int's.

- void unpack (long *data, const int num=1)
  
  Unpack one or more long's.

- void unpack (u_long *data, const int num=1)
Unpack one or more unsigned long's.
• void unpack (short *data, const int num=1)
  Unpack one or more short's.
• void unpack (u_short *data, const int num=1)
  Unpack one or more unsigned short's.
• void unpack (char *data, const int num=1)
  Unpack one or more char's.
• void unpack (u_char *data, const int num=1)
  Unpack one or more unsigned char's.
• void unpack (double *data, const int num=1)
  Unpack one or more double's.
• void unpack (float *data, const int num=1)
  Unpack one or more float's.
• void unpack (bool *data, const int num=1)
  Unpack one or more bool's.
• void unpack (int &data)
  Unpack a int.
• void unpack (u_int &data)
  Unpack a unsigned int.
• void unpack (long &data)
  Unpack a long.
• void unpack (u_long &data)
  Unpack a unsigned long.
• void unpack (short &data)
  Unpack a short.
• void unpack (u_short &data)
  Unpack a unsigned short.
• void unpack (char &data)
  Unpack a char.
• void unpack (u_char &data)
  Unpack a unsigned char.
• void unpack (double &data)
  Unpack a double.
• void unpack (float &data)
  Unpack a float.
• void unpack (bool &data)
  Unpack a bool.
 Protected Attributes

- char * Buffer
  The internal buffer for unpacking.
- int Index
  The index into the current buffer.
- int Size
  The total size that has been allocated for the buffer.
- bool ownFlag
  If TRUE, then this class owns the internal buffer.

14.111.1 Detailed Description

Class for unpacking MPI message buffers.
A class that provides a facility for unpacking message buffers using the MPI_Unpack facility. This class is based on the Dakota_Version_3_0 version of utilib::UnPackBuffer from utilib/src/io/PackBuf.[cpp,h].
The documentation for this class was generated from the following files:
- MPIPackBuffer.hpp
- MPIPackBuffer.cpp

14.112 NCSUOptimizer Class Reference

Wrapper class for the NCSU DIRECT optimization library.
Inheritance diagram for NCSUOptimizer:

```
  NCSUOptimizer
     |___________
        |        
     Minimizer
     |___________
        |        
     Optimizer
     |___________
        |        
   Iterator
```

Public Member Functions

- NCSUOptimizer (ProblemDescDB &problem db, Model &model)
  standard constructor
- NCSUOptimizer (Model &model, const int &max_iter, const int &max_eval, double min_box_size=-1.,
  double vol_box_size=-1., double solution_target=-DBL_MAX)
  alternate constructor for instantiations "on the fly"
- NCSUOptimizer (Model &model)
  alternate constructor for Iterator instantiations by name
• NCSUOptimizer (const RealVector &var_lbnds, const RealVector &var_ubnds, const int &max_iter, const int &max_eval, double(*user_obj_eval)(const RealVector &x), double min_box_size=-1., double vol_box_size=-1., double solution_target=-DBL_MAX)
  alternate constructor for instantiations "on the fly"
• ∼NCSUOptimizer ()
  destructor
• void core_run ()
  core portion of run; implemented by all derived classes and may include pre/post steps in lieu of separate pre/post
• void declare_sources ()
  Declare sources to the evaluations database.

Private Member Functions
• void initialize ()
  shared code among model-based constructors
• void check_inputs ()
  verify problem respects NCSU DIRECT Fortran limits

Static Private Member Functions
• static int objective_eval (int *n, double c[], double l[], double u[], int point[], int *maxI, int *start, int *maxfunc, double fvec[], int idata[], int isize, double ddata[], int idsize, char cdata[], int icsize)
  'fep' in Griffin-modified NCSUDirect: computes the value of the objective function (potentially at multiple points, passed by function pointer to NCSUDirect). Include unscaling from DIRECT.

Private Attributes
• short setUpType
  controls iteration mode: SETUP_MODEL (normal usage) or SETUP_USERFUNC (user-supplied functions mode for "on the fly" instantiations). see enum in NCSUOptimizer.cpp NonDGlobalReliability currently uses the model mode. GaussProcApproximation currently uses the user_functions mode.
• Real minBoxSize
  holds the minimum boxsize
• Real volBoxSize
  hold the minimum volume boxsize
• Real solutionTarget
  holds the solution target minimum to drive towards
• RealVector lowerBounds
  holds variable lower bounds passed in for "user_functions" mode.
• RealVector upperBounds
  holds variable upper bounds passed in for "user_functions" mode.
• double(* userObjectiveEval ) (const RealVector &x)
  holds function pointer for objective function evaluator passed in for "user_functions" mode.
Static Private Attributes

- static NCSUOptimizer* ncsudirectInstance

  pointer to the active object instance used within the static evaluator functions in order to avoid the need for static data

Additional Inherited Members

14.112.1 Detailed Description

Wrapper class for the NCSU DIRECT optimization library.

The NCSUOptimizer class provides a wrapper for a Fortran 77 implementation of the DIRECT algorithm developed at North Carolina State University. It uses a function pointer approach for which passed functions must be either global functions or static member functions. Any attribute used within static member functions must be either local to that function or accessed through a static pointer.

The user input mappings are as follows:

14.112.2 Constructor & Destructor Documentation

NCSUOptimizer (ProblemDescDB & problemDb, Model & model)

standard constructor

  This is the standard constructor with method specification support.
  References NCSUOptimizer::check_inputs(), and NCSUOptimizer::initialize().

NCSUOptimizer (Model & model, const int & max_iter, const int & maxeval, double min_box_size = -1., double vol_box_size = -1., double solution_target = -DBL_MAX)

alternate constructor for instantiations “on the fly”

  This is an alternate constructor for instantiations on the fly using a Model but no ProblemDescDB.
  References NCSUOptimizer::check_inputs(), NCSUOptimizer::initialize(), Iterator::maxFunctionEvals, and Iterator::maxIterations.

NCSUOptimizer (Model & model)

alternate constructor for Iterator instantiations by name

  This is an alternate constructor for Iterator instantiations by name using a Model but no ProblemDescDB.
  References NCSUOptimizer::check_inputs(), and NCSUOptimizer::initialize().

NCSUOptimizer (const RealVector & var_l_bnds, const RealVector & var_u_bnds, const int & max_iter, const int & maxeval, double(*)(const RealVector &x) user_obj_eval, double min_box_size = -1., double vol_box_size = -1., double solution_target = -DBL_MAX)

alternate constructor for instantiations “on the fly”

  This is an alternate constructor for performing an optimization using the passed in objective function pointer.
  References NCSUOptimizer::check_inputs(), Iterator::maxFunctionEvals, and Iterator::maxIterations.
14.112.3 Member Function Documentation

```cpp
void core_run ( ) [virtual]
```
core portion of run; implemented by all derived classes and may include pre/post steps in lieu of separate pre/post
Virtual run function for the iterator class hierarchy. All derived classes need to redefine it.
Reimplemented from Iterator.
References Dakota::abort_handler(), Iterator::bestResponseArray, Iterator::bestVariablesArray, Model::continuous_lower_bounds(), Model::continuous_upper_bounds(), Model::continuousVariables(), Iterator::convergenceTol, Dakota::copy_data(), Iterator::iteratedModel, Optimizer::localObjectiveRecast, NCSUOptimizer::lowerBounds, Iterator::maxFunctionEvals, Iterator::maxIterations, NCSUOptimizer::minBoxSize, NCSUOptimizer::ncsudirectInstance, Minimizer::numContinuousVars, Minimizer::numFunctions, NCSUOptimizer::objective_eval(), Iterator::outputLevel, Model::primary_response_fn_sense(), NCSUOptimizer::setUptype, NCSUOptimizer::solutionTarget, NCSUOptimizer::upperBounds, and NCSUOptimizer::volBoxSize.

```cpp
int objective_eval ( int * n, double c[], double l[], double u[], int * point[], int * maxI[], int * start, int * maxfunc, double fvec[], int iidata[], int * isize, double ddata[], int * idsize, char cdata[], int * icsize ) [static], [private]
```
'fep' in Griffin-modified NCSUDirect: computes the value of the objective function (potentially at multiple points, passed by function pointer to NCSUDirect). Include unscaling from DIRECT.
Modified batch evaluator that accepts multiple points and returns corresponding vector of functions in fvec. Must be used with modified DIRECT src (DIRbatch.f).
References Model::asynch_flag(), Model::continuous_variables(), Model::current_response(), Model::evaluate(), Model::evaluate_nowait(), Response::function_value(), Iterator::iteratedModel, NCSUOptimizer::ncsudirectInstance, Model::primary_response_fn_sense(), NCSUOptimizer::setUptype, Model::synchronize(), and NCSUOptimizer::userObjectiveEval.
Referenced by NCSUOptimizer::core_run().
The documentation for this class was generated from the following files:

- NCSUOptimizer.hpp
- NCSUOptimizer.cpp

14.113 NCSUTraits Class Reference

A version of TraitsBase specialized for NCSU optimizers.
Inheritance diagram for NCSUTraits:

```
TraitsBase

NCSUTraits
```

Public Member Functions

- NCSUTraits()
  
  default constructor
- virtual ~NCSUTraits()
14.114. NESTEDMODEL CLASS REFERENCE

** destructor

- virtual bool is_derived ()
  
  *A temporary query used in the refactor.*
- bool supports_continuous_variables ()
  
  *Return the flag indicating whether method supports continuous variables.*

14.113.1 Detailed Description

A version of TraitsBase specialized for NCSU optimizers.

The documentation for this class was generated from the following file:

- NCSUOptimizer.hpp

14.114 NestedModel Class Reference

Derived model class which performs a complete sub-iterator execution within every evaluation of the model.

Inheritance diagram for NestedModel:

```
  Model
   
   NestedModel
```

**Public Member Functions**

- NestedModel (ProblemDescDB &problem_db)
  
  *constructor*
- ~NestedModel ()
  
  *destructor*
- void declare_sources ()
  
  *Declare a model’s sources to the evaluationsDB.*

**Protected Member Functions**

- void derived_evaluate (const ActiveSet &set)
  
  *portion of evaluate() specific to NestedModel*
- void derived_evaluate_nowait (const ActiveSet &set)
  
  *portion of evaluate_nowait() specific to NestedModel*
- const IntResponseMap & derived_synchronize ()
  
  *portion of synchronize() specific to NestedModel*
- Iterator & subordinate_iterator ()
  
  *return subIterator*
- Model & subordinate_model ()
  
  *return subModel*
- **void derived_subordinate_models** (ModelList &ml, bool recurse_flag)
  
  return subModel
- **Interface & derived_interface** ()
  
  return optionalInterface
- **const RealVector & error_estimates** ()
  
  retrieve error estimates corresponding to the subIterator's response results (e.g., statistical MSE for subordinate UQ).
- **void surrogate_response_mode** (short mode)
  
  pass a bypass request on to the subModel for any lower-level surrogates
- **void component_parallel_mode** (short mode)
  
  update component parallel mode for supporting parallelism in optionalInterface and subModel
- **size_t mi_parallel_level_index** () const
  
  return subIteratorSched.miPLIndex
- **short local_eval_synchronization** ()
  
  return optionalInterface synchronization setting
- **int local_eval_concurrency** ()
  
  return optionalInterface asynchronous evaluation concurrency
- **bool derived_master_overload** () const
  
  flag which prevents overloading the master with a multiprocessor evaluation (forwarded to optionalInterface)
- **IntIntPair estimate_partition_bounds** (int max_eval_concurrency)
  
  estimate the minimum and maximum partition sizes that can be utilized by this Model
- **void derived_init_communicators** (ParLevLIter pl_iter, int max_eval_concurrency, bool recurse_flag=true)
  
  set up optionalInterface and subModel for parallel operations
- **void derived_init_serial** ()
  
  set up optionalInterface and subModel for serial operations.
- **void derived_set_communicators** (ParLevLIter pl_iter, int max_eval_concurrency, bool recurse_flag=true)
  
  set active parallel configuration within subModel
- **void derived_free_communicators** (ParLevLIter pl_iter, int max_eval_concurrency, bool recurse_flag=true)
  
  deallocate communicator partitions for the NestedModel (forwarded to optionalInterface and subModel)
- **void serve_run** (ParLevLIter pl_iter, int max_eval_concurrency)
  
  Service optionalInterface and subModel job requests received from the master. Completes when a termination message is received from stop_servers().
- **void stop_servers** ()
  
  Executed by the master to terminate server operations for subModel and optionalInterface when iteration on the NestedModel is complete.
- **const String & interface_id** () const
  
  return the optionalInterface identifier
- **int derived_evaluation_id** () const
  
  Return the current evaluation id for the NestedModel.
- **void set_evaluation_reference** ()
  
  set the evaluation counter reference points for the NestedModel (request forwarded to optionalInterface and subModel)
- **void fine_grained_evaluation_counters** ()
  
  request fine-grained evaluation reporting within optionalInterface and subModel
• void print_evaluation_summary (std::ostream &s, bool minimal_header=false, bool relative_count=true) const
  
  print the evaluation summary for the NestedModel (request forwarded to optionalInterface and subModel)
• void warm_start_flag (const bool flag)
  
  set the warm start flag, including actualModel
• void initialize_iterator (int job_index)
• void pack_parameters_buffer (MPIPackBuffer &send_buffer, int job_index)
• void unpack_parameters_buffer (MPIUnpackBuffer &recv_buffer, int job_index)
• void unpack_parameters_initialize (MPIUnpackBuffer &recv_buffer, int job_index)
• void pack_results_buffer (MPIPackBuffer &send_buffer, int job_index)
• void unpack_results_buffer (MPIUnpackBuffer &recv_buffer, int job_index)
• void update_local_results (int job_index)
• ActiveSet default_interface_active_set ()

Private Member Functions

• void init_sub_iterator ()
  
  init subIterator-based counts and init subModel with mapping data
• PRPQueueLite job_index_to_queue_iterator (int job_index)
  
  convert job_index to an eval_id through subIteratorIdMap and eval_id to a subIteratorPRPQueue queue iterator
• void initialize_iterator (const Variables &vars, const ActiveSet &set, int eval_id)
  
  lower level function shared by initialize_iterator(int) and unpack_parameters_initialize()
• void unpack (MPIUnpackBuffer &recv_buffer, int job_index, Variables &vars, ActiveSet &set, int &eval_id)
  
  lower level function shared by unpack_parameters_buffer() and unpack_parameters_initialize()
• void resolve_map1 (const String &map1, size_t &ac_index1, size_t &adi_index1, size_t &ads_index1, size_t &adr_index1, size_t curr_index, short &inactive_sm_view)
  
  compute variable mapping indices corresponding to map1 and update inactive view if necessary
• void resolve_real_variable_mapping (const String &map1, const String &map2, size_t curr_index, short &inactive_sm_view)
  
  for a named real mapping, resolve primary index and secondary target
• void resolve_integer_variable_mapping (const String &map1, const String &map2, size_t curr_index, short &inactive_sm_view)
  
  for a named integer mapping, resolve primary index and secondary target
• void resolve_string_variable_mapping (const String &map1, const String &map2, size_t curr_index, short &inactive_sm_view)
  
  for a named string mapping, resolve primary index and secondary target
• void real_variable_mapping (Real r_var, size_t av_index, short svm_target)
  
  insert r_var into appropriate recipient
• void integer_variable_mapping (int i_var, size_t av_index, short svm_target)
  
  insert i_var into appropriate recipient
• void string_variable_mapping (const String &str_var, size_t av_index, short svm_target)
  
  insert str_var into appropriate recipient
• void set_mapping (const ActiveSet &mapped_set, ActiveSet &interface_set, bool &opt_interface_map, ActiveSet &sub_iterator_set, bool &sub_iterator_map)
define the evaluation requirements for the optionalInterface (interface_set) and the subIterator (sub_iterator_set) from the total model evaluation requirements (mapped_set)

- void response_mapping (const Response &interface_response, const Response &sub_iterator_response, Response &mapped_response)
  - combine the response from the optional interface evaluation with the response from the sub-iteration using the primaryCoeffs/secondaryCoeffs mappings to create the total response for the model

- void interface_response_overlay (const Response &opt_interface_response, Response &mapped_response)
  - assign the response from the optional interface evaluation within the total response for the model

- void iterator_response_overlay (const Response &sub_iterator_response, Response &mapped_response)
  - overlay the sub-iteration response within the total response for the model using the primaryCoeffs/secondaryCoeffs mappings

- void iterator_error_estimation (const RealVector &sub_iterator_errors, RealVector &mapped_errors)
  - combine error estimates from the sub-iteration to define mappedErrorEstimates

- Response & nested_response (int nested_cntr)
  - locate existing or allocate new entry in nestedResponseMap

- void check_response_map (const ShortArray &mapped_asv)
  - check function counts for the mapped_asv

- void update_inactive_view (short new_view, short &view)
  - update inactive variables view for subIterator based on new_view

- void update_inactive_view (unsigned short type, short &view)
  - update inactive variables view for subIterator based on type

- void update_sub_model (const Variables &vars, const Constraints &cons)
  - update subModel with current variable values/bounds/labels

Private Attributes

- int nestedModelEvalCntr
  - number of calls to derived_evaluate()/derived_evaluate_nowait()

- bool firstUpdate
  - boolean to trigger one-time updates on first call to update_sub_model()

- IntResponseMap nestedResponseMap
  - used to return a map of nested responses (including subIterator and optionalInterface contributions) for aggregation and rekeying at the base class level

- RealVector mappedErrorEstimates
  - mapping of subIterator.response_error_estimates() through primary and secondary mappings

- size_t outerMIPLIndex
  - the miPLIndex for the outer parallelism context, prior to any subIterator partitioning

- Iterator subIterator
  - the sub-iterator that is executed on every evaluation of this model

- Model subModel
  - the sub-model used in sub-iterator evaluations

- PRPQueue subIteratorPRPQueue
  - job queue for asynchronous execution of subIterator jobs

- IteratorScheduler subIteratorSched
scheduling object for concurrent iterator parallelism

- **String subMethodPointer**
  the sub-method pointer from the nested model specification

- **int subIteratorJobCntr**
  subIterator job counter since last \texttt{synchronize()}

- **IntIntMap subIteratorIdMap**
  mapping from subIterator evaluation counter to nested model counter (different when subIterator evaluations do not occur on every nested model evaluation due to variable ASV content)

- **size_t numSubInterFns = 0**
  number of sub-iterator response functions prior to mapping

- **size_t numSubInterMappedIneqCon = 0**
  number of top-level inequality constraints mapped from the sub-iteration results

- **size_t numSubInterMappedEqCon = 0**
  number of top-level equality constraints mapped from the sub-iteration results

- **Interface optionalInterface**
  the optional interface contributes nonnested response data to the total model response

- **String optInterfacePointer**
  the optional interface pointer from the nested model specification

- **Response optInterfaceResponse**
  the response object resulting from optional interface evaluations

- **IntIntMap optInterfaceIdMap**
  mapping from optionalInterface evaluation counter to nested model counter (different when optionalInterface evaluations do not occur on every nested model evaluation due to variable ASV content)

- **size_t numOptInterfPrimary = 0**
  number of primary response functions (objective/least squares/generic functions) resulting from optional interface evaluations

- **size_t numOptInterfIneqCon = 0**
  number of inequality constraints resulting from optional interface evaluations

- **size_t numOptInterfEqCon = 0**
  number of equality constraints resulting from the optional interface evaluations

- **IntSet optInterfGradIdAnalytic**
  analytic IDs for mixed gradients on the optional interface

- **IntSet optInterfHessIdAnalytic**
  analytic IDs for mixed Hessians on the optional interface

- **String optInterfGradientType**
  Gradient type for the optional interface.

- **String optInterfHessianType**
  Hessian type for the optional interface.

- **SizetArray active1ACVarMapIndices**
  "primary" variable mappings for inserting active continuous currentVariables within all continuous subModel variables. If there are no secondary mappings defined, then the insertions replace the subModel variable values.

- **SizetArray active1ADIVarMapIndices**
  "primary" variable mappings for inserting active discrete int currentVariables within all discrete int subModel variables. No secondary mappings are defined for discrete int variables, so the active variables replace the sub-Model variable values.
• SizetArray active1ADSVarMapIndices
  "primary" variable mappings for inserting active discrete string currentVariables within all discrete string subModel variables. No secondary mappings are defined for discrete string variables, so the active variables replace the subModel variable values.
• SizetArray active1ADRVarMapIndices
  "primary" variable mappings for inserting active discrete real currentVariables within all discrete real subModel variables. No secondary mappings are defined for discrete real variables, so the active variables replace the subModel variable values.
• ShortArray active2ACVarMapTargets
  "secondary" variable mappings for inserting active continuous currentVariables into sub-parameters (e.g., distribution parameters for uncertain variables or bounds for continuous design/state variables) within all continuous subModel variables.
• ShortArray active2ADIVarMapTargets
  "secondary" variable mappings for inserting active discrete int currentVariables into sub-parameters (e.g., bounds for discrete design/state variables) within all discrete int subModel variables.
• ShortArray active2ADSVarMapTargets
  "secondary" variable mappings for inserting active discrete string currentVariables into sub-parameters (e.g., bounds for discrete design/state variables) within all discrete string subModel variables.
• ShortArray active2ADRVarMapTargets
  "secondary" variable mappings for inserting active discrete real currentVariables into sub-parameters (e.g., bounds for discrete design/state variables) within all discrete real subModel variables.
• SizetArray complement1ACVarMapIndices
  "primary" variable mappings for inserting the complement of the active continuous currentVariables within all continuous subModel variables.
• SizetArray complement1ADIVarMapIndices
  "primary" variable mappings for inserting the complement of the active discrete int currentVariables within all discrete int subModel variables.
• SizetArray complement1ADSVarMapIndices
  "primary" variable mappings for inserting the complement of the active discrete string currentVariables within all discrete string subModel variables.
• SizetArray complement1ADRVarMapIndices
  "primary" variable mappings for inserting the complement of the active discrete real currentVariables within all discrete real subModel variables.
• BitArray extraCVarsData
  flags for updating subModel continuous bounds and labels, one for each active continuous variable in currentVariables.
• BitArray extraDIVarsData
  flags for updating subModel discrete int bounds and labels, one for each active discrete int variable in currentVariables.
• BitArray extraDSVarsData
  flags for updating subModel discrete string labels, one for each active discrete string variable in currentVariables.
• BitArray extraDRVarsData
  flags for updating subModel discrete real bounds and labels, one for each active discrete real variable in currentVariables.
• bool identityRespMap = false
  whether identity response mapping is active.
- `size_t subIterMappedPri = 0`
  
  number of sub-iterator results functions mapped to nested model primary functions (cached for use with identity case)

- `size_t subIterMappedSec = 0`
  
  number of sub-iterator results functions mapped to nested model secondary functions (cached for use with identity case)

- `RealMatrix primaryRespCoeffs`
  
  "primary" response mapping matrix applied to the sub-iterator response functions. For OUU, the matrix is applied to UQ statistics to create contributions to the top-level objective functions/least squares/generic response terms.

- `RealMatrix secondaryRespCoeffs`
  
  "secondary" response mapping matrix applied to the sub-iterator response functions. For OUU, the matrix is applied to UQ statistics to create contributions to the top-level inequality and equality constraints.

**Friends**

- `class IteratorScheduler`
  protect scheduler callback functions from general access

**Additional Inherited Members**

### 14.114.1 Detailed Description

Derived model class which performs a complete sub-iterator execution within every evaluation of the model.

The `NestedModel` class nests a sub-iterator execution within every model evaluation. This capability is most commonly used for optimization under uncertainty, in which a nondeterministic iterator is executed on every optimization function evaluation. The `NestedModel` also contains an optional interface, for portions of the model evaluation which are independent from the sub-iterator, and a set of mappings for combining sub-iterator and optional interface data into a top level response for the model.

### 14.114.2 Member Function Documentation

```cpp
void derived_evaluate ( const ActiveSet & set ) [protected], [virtual]
```

Update subModel’s inactive variables with active variables from currentVariables, compute the optional interface and sub-iterator responses, and map these to the total model response.

Reimplemented from `Model`.

References `NestedModel::active2ACVarMapTargets`, `Response::active_set()`, `Interface::analysis_components()`, `ParallelLibrary::bcast()`, `ParallelLibrary::bcast_hs()`, `NestedModel::component_parallel_mode()`, `Model::currentResponse`, `Model::currentVariables`, `Interface::eval_tag_prefix()`, `Iterator::eval_tag_prefix()`, `Model::evalTagPrefix`, `Interface::evaluation_id()`, `Model::evaluationsDB`, `Model::hierarchicalTagging`, `NestedModel::interface_id()`, `NestedModel::interface_response_overlay()`, `Model::interfEvaluationsDBState`, `NestedModel::iterator_response.overlay()`, `IteratorScheduler::iteratorCommSize`, `IteratorScheduler::iteratorScheduling`, `Interface::map()`, `IteratorScheduler::messagePass`, `IteratorScheduler::miPLIndex`, `Model::modelId`, `Model::modelPCIter`, `NestedModel::nestedModelEvalCntr`, `NestedModel::optInterfaceResponse`, `NestedModel::optionalInterface`, `Model::outputLevel`, `ParallelLibrary::parallel_configuration_iterator()`, `Model::parallelLib`, `IteratorScheduler::peerAssignJobs`, `Response::reset()`, `Iterator::response_results()`, `Iterator::response_results_active_set()`, `Iterator::run()`, `IteratorScheduler::run_iterator()`, `NestedModel::set_mapping()`, `IteratorScheduler::stop_iterator_servers()`, `NestedModel::subIterator`, `NestedModel::subIteratorSched`, `NestedModel::update_sub_model()`, and `Model::userDefinedConstraints`. 

void derived_evaluate_nowait ( const ActiveSet & set ) [protected], [virtual]

portion of evaluate_nowait() specific to NestedModel

Asynchronous execution of subIterator on subModel and, optionally, optionalInterface.
Reimplemented from Model.
References Response::active_set(), Interface::analysis_components(), Model::currentResponse, Model::currentVariables, Interface::evaluation_id(), Model::evaluationsDB, NestedModel::interface_id(), Model::interfEvaluationsDBState, Interface::map(), Iterator::method_id(), Model::modelId, NestedModel::nestedModelEvalCntr, NestedModel::optInterfaceIdMap, NestedModel::optInterfaceResponse, NestedModel::optionalInterface, Iterator::response_results(), Iterator::response_results_active_set(), NestedModel::set_mapping(), NestedModel::subIterator, NestedModel::subIteratorIdMap, NestedModel::subIteratorJobCntr, and NestedModel::subIteratorPRPQueue.

const IntResponseMap & derived_synchronize ( ) [protected], [virtual]

portion of synchronize() specific to NestedModel

Recovery of asynchronous subIterator executions and, optionally, asynchronous optionalInterface mappings.
Reimplemented from Model.
References Interface::cache_unmatched_response(), NestedModel::component_parallel_mode(), NestedModel::interface_response_overlay(), NestedModel::iterator_response_overlay(), Model::modelPCIter, NestedModel::nested_response(), NestedModel::nestedResponseMap, IteratorScheduler::numIteratorJobs, NestedModel::optInterfaceIdMap, NestedModel::optInterfacePointer, NestedModel::optionalInterface, ParallelLibrary::parallel_configuration_iterator(), Model::parallelLib, IteratorScheduler::schedule_iterators(), NestedModel::subIterator, NestedModel::subIteratorIdMap, NestedModel::subIteratorJobCntr, NestedModel::subIteratorPRPQueue, NestedModel::subIteratorSched, and Interface::synchronize().

short local_eval_synchronization ( ) [inline], [protected], [virtual]

return optionalInterface synchronization setting

Used in setting Model::asynchEvalFlag. subModel synchronization is used for setting asynchEvalFlag within subModel.
Reimplemented from Model.
References Interface::asynch_local_evaluation_concurrency(), Interface::interface_synchronization(), NestedModel::optInterfacePointer, and NestedModel::optionalInterface.

int local_eval_concurrency ( ) [inline], [protected], [virtual]

return optionalInterface asynchronous evaluation concurrency

Used in setting Model::evaluationCapacity. subModel concurrency is used for setting evaluationCapacity within subModel.
Reimplemented from Model.
References Interface::asynch_local_evaluation_concurrency(), NestedModel::optInterfacePointer, and NestedModel::optionalInterface.

bool derived_master_overload ( ) const [inline], [protected], [virtual]

flag which prevents overloading the master with a multiprocessor evaluation (forwarded to optionalInterface)

Derived master overload for subModel is handled separately in subModel.evaluate() within subIterator.run().
Reimplemented from Model.
References Iterator::is_null(), Interface::iterator_eval_dedicated_master(), IteratorScheduler::iteratorScheduling, Interface::multi_proc_eval(), NestedModel::optInterfacePointer, NestedModel::optionalInterface, IteratorScheduler::procsPerIterator, NestedModel::subIterator, and NestedModel::subIteratorSched.
void derived_init_communicators ( ParLevIter pl_iter, int max_eval_concurrency, bool recurse_flag = true) [protected], [virtual]

set up optionalInterface and subModel for parallel operations

Asynchronous flags need to be initialized for the subModel. In addition, max_eval_concurrency is the outer
level iterator concurrency, not the sub_iterator concurrency that subModel will see, and recomputing the message-
lengths on the subModel is probably not a bad idea either. Therefore, recompute everything on subModel using
init_communicators()

Reimplemented from Model.
References Response::active_set(), IteratorScheduler::configure(), Model::currentVariables, ProblemDescDB-
::get_db_method_node(), ProblemDescDB::get_db_model_node(), Interface::init_communicators(), IteratorScheduler-
::init_iterator(), NestedModel::init_sub_iterator(), Iterator::is_null(), IteratorScheduler::iterator_message_lengths(),
IteratorScheduler::iteratorServerId, Model::messageLengths, IteratorScheduler::messagePass, Model::modelPC-
Iter, IteratorScheduler::numIteratorServers, NestedModel::optInterfacePointer, NestedModel::optionalInterface,
ParallelLibrary::parallel_configuration_iterator(), Model::parallelLib, IteratorScheduler::partition(), Model::prob-
DescDB, MPIPackBuffer::reset(), Iterator::response_results(), ProblemDescDB::set_db_list_nodes(), ProblemDesc-
DB::set_db_method_node(), ProblemDescDB::set_db_model_nodes(), MPIPackBuffer::size(), NestedModel::sub-
Iterator, NestedModel::subIteratorSched, NestedModel::subMethodPointer, NestedModel::subModel, and Iterator-
Scheduler::update().

int derived_evaluation_id () const [inline], [protected], [virtual]

Return the current evaluation id for the NestedModel.

return the top level nested evaluation count. To get the lower level eval count, the subModel must be explicitly
queried. This is consistent with the eval counter definitions in surrogate models.

Reimplemented from Model.
References NestedModel::nestedModelEvalCntr.

void response_mapping ( const Response & opt_interface_response, const Response & sub_iterator_response, Response & mapped_response ) [inline], [private]

combine the response from the optional interface evaluation with the response from the sub-iteration using the
primaryCoeffs/secondaryCoeffs mappings to create the total response for the model

In the OUU case,

optionalInterface fns = {f}, {g} (deterministic primary functions, constraints)
subIterator fns = {S} (UQ response statistics)

Problem formulation for mapped functions:
minimize (f) + [W]{S}
subject to
{g_l} <= {g} <= {g_u}
{a_l} <= [A]{S} <= {a_u}
{g} == {g_t}
[A]{S} == {a_t}

where [W] is the primary_mapping_matrix user input (primaryRespCoeffs class attribute), [A] is the secondary-
_mapping_matrix user input (secondaryRespCoeffs class attribute), {{g_l},{a_l}} are the top level inequality con-
straint lower bounds, {{g_u},{a_u}} are the top level inequality constraint upper bounds, and {{g_t},{a_t}} are
the top level equality constraint targets.

NOTE: optionalInterface/subIterator primary fns (obj/lsq/generic fns) overlap but optionalInterface/subIterator
secondary fns (ineq/eq constraints) do not. The [W] matrix can be specified so as to allow
CHAPTER 14. CLASS DOCUMENTATION

- some purely deterministic primary functions and some combined: \([W]\) filled and \([W].\text{num\_rows}() < \{f\}.\text{length}()\) [combined first] or \([W].\text{num\_rows}() == \{f\}.\text{length}()\) and \([W]\) contains rows of zeros [combined last]
- some combined and some purely stochastic primary functions: \([W]\) filled and \([W].\text{num\_rows}() > \{f\}.\text{length}()\)
- separate deterministic and stochastic primary functions: \([W].\text{num\_rows}() > \{f\}.\text{length}()\) and \([W]\) contains \{f\}.\text{length}() rows of zeros.

If the need arises, could change constraint definition to allow overlap as well: \(\{g\_l\} \subseteq \{g\} + [A]\{S\} \subseteq \{g\_u\}\) with \([A]\) usage the same as for \([W]\) above.

In the UOO case, things are simpler, just compute statistics of each optimization response function: \([W] = [I], \{f\}/\{g\}/[A]\) are empty.

References Response::active_set_request_vector(), NestedModel::check_response_map(), NestedModel::interface_response_overlay(), and NestedModel::iterator_response_overlay().

14.114.3 Member Data Documentation

**Model subModel** [private]

the sub-model used in sub-iterator evaluations

There are no restrictions on subModel, so arbitrary nestings are possible. This is commonly used to support surrogate-based optimization under uncertainty by having NestedModels contain SurrogateModels and vice versa.

Referenced by NestedModel::component_parallel_mode(), NestedModel::derived_init_communicators(), NestedModel::derived_init_serial(), NestedModel::derived_subordinate_models(), NestedModel::estimate_partition_bounds(), NestedModel::fine_grained_evaluation_counts(), NestedModel::integer_variable_mapping(), NestedModel::NestedModel(), NestedModel::print_evaluation_summary(), NestedModel::real_variable_mapping(), NestedModel::resolve_integer_variable_mapping(), NestedModel::resolve_map1(), NestedModel::resolve_real_variable_mapping(), NestedModel::resolve_string_variable_mapping(), NestedModel::serve_run(), NestedModel::set_mapping(), NestedModel::string_variable_mapping(), NestedModel::subordinate_model(), NestedModel::surrogate_response_mode(), NestedModel::update_inactive_view(), NestedModel::update_sub_model(), and NestedModel::warm_start_flag().

The documentation for this class was generated from the following files:

- NestedModel.hpp
- NestedModel.cpp

14.115 NIDRProblemDescDB Class Reference

The derived input file database utilizing the new IDR parser.

Inheritance diagram for NIDRProblemDescDB:

```
ProblemDescDB
NIDRProblemDescDB
```

Public Member Functions

- **NIDRProblemDescDB** (ParallelLibrary &parallel_lib)
  
  *constructor*
• ~NIDRProblemDescDB ()
  destructor
• void derived_parse_inputs (const std::string &dakota_input_file, const std::string &dakota_input_string, const std::string &parser_options)
  parses the input file and populates the problem description database using NIDR.
• void derived_broadcast ()
  perform any data processing that must be coordinated with DB buffer broadcasting (performed prior to broadcasting the DB buffer on rank 0 and after receiving the DB buffer on other processor ranks)
• void derived_post_process ()
  perform any additional data post-processing

  • KWH (iface_Real)
  • KWH (iface_Rlit)
  • KWH (iface_false)
  • KWH (iface_ilit)
  • KWH (iface_int)
  • KWH (iface_lit)
  • KWH (iface_start)
  • KWH (iface_stop)
  • KWH (iface_str)
  • KWH (iface_str2D)
  • KWH (iface_strL)
  • KWH (iface_true)
  • KWH (iface_type)
  • KWH (method_ili)
  • KWH (method_Real)
  • KWH (method_Real01)
  • KWH (method_RealDL)
  • KWH (method_RealLit)
  • KWH (method_Realp)
  • KWH (method_Realz)
  • KWH (method_Ri)
  • KWH (method_false)
  • KWH (method_szarray)
  • KWH (method_ilit2)
  • KWH (method_ilit2p)
  • KWH (method_int)
  • KWH (method_ivec)
  • KWH (method_lit)
  • KWH (method_litc)
  • KWH (method_lit1)
  • KWH (method_litp)
  • KWH (method_litr)
  • KWH (method_litz)
  • KWH (method_order)
  • KWH (method_num_resplevs)
  • KWH (method_piecewise)
• KWH (method_resplevs)
• KWH (method_resplevs01)
• KWH (method_shint)
• KWH (method_sizet)
• KWH (method_slit2)
• KWH (method_start)
• KWH (method_stop)
• KWH (method_str)
• KWH (method_strL)
• KWH (method_true)
• KWH (method_tr_final)
• KWH (method_type)
• KWH (method_usharray)
• KWH (method_ushint)
• KWH (method_utype)
• KWH (method_augment_utype)
• KWH (method_utype_lit)
• KWH (model_Real)
• KWH (model_RealDL)
• KWH (model_ivec)
• KWH (model_false)
• KWH (model_int)
• KWH (model_intsetm1)
• KWH (model_lit)
• KWH (model_order)
• KWH (model_shint)
• KWH (model_sizet)
• KWH (model_start)
• KWH (model_stop)
• KWH (model_str)
• KWH (model_strL)
• KWH (model_true)
• KWH (model_type)
• KWH (model_usharray)
• KWH (model_ushint)
• KWH (model_utype)
• KWH (model_augment_utype)
• KWH (resp_RealDL)
• KWH (resp_RealL)
• KWH (resp_false)
• KWH (resp_intset)
• KWH (resp_ivec)
• KWH (resp_lit)
• KWH (resp_sizet)
• KWH (resp_start)
• KWH (resp_stop)
• KWH (resp_str)
• KWH (resp_strL)
• KWH (resp_true)
• KWH (resp_utype)
• KWH (resp_augment_utype)
• KWH (env_int)
• KWH (env_start)
• KWH (env_str)
• KWH (env_strL)
• KWH (env_true)
• KWH (env_utype)
• KWH (env_augment_utype)
• KWH (var_RealLb)
• KWH (var_RealUb)
• KWH (var_IntLb)
• KWH (var_categorical)
• KWH (var_caulbl)
• KWH (var_dauilbl)
• KWH (var_dauslbl)
• KWH (var_daurlbl)
• KWH (var_ceulbl)
• KWH (var_deulbl)
• KWH (var_deuslbl)
• KWH (var_deurlbl)
• KWH (var_sizet)
• KWH (var_start)
• KWH (var_stop)
• KWH (var_str)
• KWH (var_strL)
• KWH (var_true)
• KWH (var_newiarray)
• KWH (var_newsarray)
• KWH (var_newivec)
• KWH (var_newrvec)
• KWH (var_ivec)
• KWH (var_svec)
• KWH (var_rvec)
• KWH (var_type)

Static Public Member Functions

• static void botch (const char *fmt,...)

  print and error message and immediately abort

• static void check_variables (std::list< DataVariables > *)

  check each node in a list of DataVariables, first mapping DataVariables members back to flat NIDR arrays if needed.

• static void check_responses (std::list< DataResponses > *)
• static void check_descriptor_format (const StringArray &labels)
  Validate format user-supplied descriptors.
• static void check_descriptors_for_repeats (const StringArray &labels)
  Ensure no response descriptors are repeated.
• static void check_descriptors_for_repeats (const StringArray &cd_labels, const StringArray &ddr_labels,
  const StringArray &ddsi_labels, const StringArray &ddss_labels, const StringArray &ddsr_labels, const
  StringArray &cs_labels, const StringArray &dss_labels, const StringArray &dsr_labels, const StringArray
  &dssi_labels, const StringArray &dsss_labels, const StringArray &dssr_labels, const StringArray &dsau_labels,
  const StringArray &dsau_labels, const StringArray &dau_labels, const StringArray &ceu_labels, const
  StringArray &dceu_labels, const StringArray &ceu_labels, const StringArray &dceu_labels, const
  StringArray &dceu_labels, const StringArray &ceu_labels)
  Ensure no variable descriptors are repeated.
• static void make_variable_defaults (std::list<DataVariables> *)
  Bounds and initial point check and inferred bounds generation.
• static void make_response_defaults (std::list<DataResponses> *)
• static void squawk (const char *fmt,...)
  print an error message and increment nerr, but continue
• static void warn (const char *fmt,...)
  print a warning

Static Public Attributes

• static NIDRProblemDescDB * pDDBInstance
  Pointer to the active object instance used within the static kwhandler functions in order to avoid the need for static
data. Only initialized when parsing an input file; will be NULL for cases of direct DB population only.
• static int nerr = 0
  number of parse error encountered

Static Private Member Functions

• static void check_variables_node (void *)
  check a single variables node; input argument v is Var_Info*
• static int check_driver (const String &an_driver, const StringArray &link_files, const StringArray &copy_
  -files)
  tokenize and try to validate the presence of an analysis driver, potentially included in the linked or copied template
  files

Private Attributes

• std::list<void *> VIL
  List of Var_Info pointers, one per Variables instance.
Additional Inherited Members

14.115.1 Detailed Description

The derived input file database utilizing the new IDR parser.

The NIDRProblemDescDB class is derived from ProblemDescDB for use by the NIDR parser in processing DAKOTA input file data. For information on modifying the NIDR input parsing procedures, refer to Dakota/docs/Dev_Spec_Change.doc. For more on the parsing technology, see "Specifying and Reading Program Input with NIDR" by David M. Gay (report SAND2008-2261P, which is available in PDF form as http://dakota.sandia.gov/papers/nidr08.pdf). Source for the routines declared herein is NIDRProblemDescDB.cpp, in which most routines are so short that a description seems unnecessary.

14.115.2 Member Function Documentation

void derived_parse_inputs ( const std::string & dakota_input_file, const std::string & dakota_input_string, const std::string & parser_options ) [virtual]

parses the input file and populates the problem description database using NIDR.

Parse the input file using the Input Deck Reader (IDR) parsing system. IDR populates the IDRProblemDescDB object with the input file data.

Reimplemented from ProblemDescDB.

References Dakota::abort_handler(), NIDRProblemDescDB::botch(), ProblemDescDB::dataMethodList, DataMethodRep::dlDetails, DataMethodRep::dlLib, NIDRProblemDescDB::nerr, ProblemDescDB::parallel_library(), NIDRProblemDescDB::pDDBInstance, and NIDRProblemDescDB::squawk().

int check_driver ( const String & an_driver, const StringArray & link_files, const StringArray & copy_files ) [static], [private]

tokenize and try to validate the presence of an analysis driver, potentially included in the linked or copied template files

returns 1 if not found, 2 if found, but not executable, 0 if found (no error) in case we want to return to error on not found...

References WorkdirHelper::find_driver(), NIDRProblemDescDB::squawk(), WorkdirHelper::tokenize_driver(), NIDRProblemDescDB::warn(), and WorkdirHelper::which().

void make_variable_defaults ( std::list< DataVariables > * dvl ) [static]

Bounds and initial point check and inferred bounds generation.

Size arrays for contiguous storage of aggregated uncertain types. For each variable type, call Vgen_* to generate inferred bounds and initial point, repairing initial if needed. size the aggregate arrays for uncertain (design and state are stored separately

References Dakota::DesignAndStateLabelsCheck, NIDRProblemDescDB::squawk(), Dakota::var_mp_drange, Dakota::VLUncertainInt, Dakota::VLUncertainReal, and Dakota::VLUncertainStr.

Referenced by NIDRProblemDescDB::derived_post_process().

The documentation for this class was generated from the following files:

- NIDRProblemDescDB.hpp
- NIDRProblemDescDB.cpp
14.116  NL2Res Struct Reference

Auxiliary information passed to calcr and calcj via ur.

Public Attributes

- \texttt{Real * r}
  
  \texttt{residual r = r(x)}

- \texttt{Real * J}
  
  \texttt{Jacobian J = J(x)}

- \texttt{Real * x}
  
  \texttt{corresponding parameter vector}

- \texttt{int nf}
  
  \texttt{function invocation count for r(x)}

14.116.1  Detailed Description

Auxiliary information passed to calcr and calcj via ur.

The documentation for this struct was generated from the following file:

- \texttt{NL2SOLLeastSq.cpp}

14.117  NL2SOLLeastSq Class Reference

Wrapper class for the NL2SOL nonlinear least squares library.

Inheritance diagram for NL2SOLLeastSq:

```
NL2SOLLeastSq
  \downarrow
LeastSq
  \downarrow
Minimizer
  \downarrow
Iterator
```

Public Member Functions

- \texttt{NL2SOLLeastSq (ProblemDescDB &problem_db, Model &model)}
  
  \texttt{standard constructor}

- \texttt{NL2SOLLeastSq (Model &model)}
  
  \texttt{alternate constructor}

- \texttt{\sim NL2SOLLeastSq ()}
  
  \texttt{destructor}

- \texttt{void core_run ()}
  
  \texttt{core portion of run; implemented by all derived classes and may include pre/post steps in lieu of separate pre/post}
Static Private Member Functions

- static void calcr (int *np, int *pp, Real *x, int *nfp, Real *r, int *ui, void *ur, Vf vf)
  evaluator function for residual vector
- static void calcj (int *np, int *pp, Real *x, int *nfp, Real *J, int *ui, void *ur, Vf vf)
  evaluator function for residual Jacobian

Private Attributes

- int auxprt
  auxiliary printing bits (see Dakota Ref Manual): sum of
  < 1 = x0prt (print initial guess) < 2 = solprt (print final solution) < 4 = statpr (print solution statistics) < 8 = parprt (print nondefault parameters) < 16 = dradpr (print bound constraint drops/adds) < debug/verbose/normal use default = 31 (everything), < quiet uses 3, silent uses 0.
- int outlev
  frequency of output summary lines in number of iterations
  < (debug/verbose/normal/quiet use default = 1, silent uses 0)
- Real dltfdj
  finite-diff step size for computing Jacobian approximation
  < (fd,gradient_step_size)
- Real delta0
  finite-diff step size for gradient differences for H
  < (a component of some covariance approximations, if desired) < (fd,hessian_step_size)
- Real dltfdc
  finite-diff step size for function differences for H
  < (fd,hessian_step_size)
- int mxfcal
  function-evaluation limit (max_function_evaluations)
- int mxiter
  iteration limit (max_iterations)
- Real rfctol
  relative fn convergence tolerance (convergence.tolerance)
- Real afctol
  absolute fn convergence tolerance (absolute_conv_tol)
- Real xctol
  x-convergence tolerance (x_conv_tol)
- Real sctol
  singular convergence tolerance (singular_conv_tol)
- Real lmaxs
  radius for singular-convergence test (singular.radius)
- Real xftol
  false-convergence tolerance (false_conv_tol)
- int covreq
  kind of covariance required (\c covariance):
  < 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 < 1 or 2 ==> use gradient diffs to estimate H < -1 or -2 ==> use function diffs to estimate H < default = 0 (no covariance)
• int rdreq
  
  whether to compute the regression diagnostic vector
  <(regression_diagnostics)
• Real fprec
  
  expected response function precision (function_precision)
• Real lmax0
  
  initial trust-region radius (initial_trust_radius)

Static Private Attributes

• static NL2SOLLeastSq* nl2solInstance
  
  pointer to the active object instance used within the static evaluator functions

Additional Inherited Members

14.117.1 Detailed Description

Wrapper class for the NL2SOL nonlinear least squares library.

The NL2SOLLeastSq class provides a wrapper for NL2SOL (TOMS Algorithm 573), in the updated form
of Port Library routines dn[fg][b ] from Bell Labs; see http://www.netlib.org/port/readme. The
Fortran from Port has been turned into C by f2c. NL2SOL uses a function pointer approach for which passed
functions must be either global functions or static member functions.

14.117.2 Member Function Documentation

void core_run ( ) [virtual]

core portion of run; implemented by all derived classes and may include pre/post steps in lieu of separate pre/post
Virtual run function for the iterator class hierarchy. All derived classes need to redefine it. Details on the fol-
lowing subscript values appear in "Usage Summary for Selected Optimization Routines" by David M. Gay, Com-
com/cm/cs/cstr/153.ps.gz

Reimplemented from Iterator.

References NL2SOLLeastSq::afctol, NL2SOLLeastSq::auxprt, LeastSq::bestIterPriFns, Iterator::bestVariables-
Array, Minimizer::boundConstraintFlag, NL2SOLLeastSq::calcj(), NL2SOLLeastSq::calcr(), Model::continuous-
_lower_bounds(), Model::continuous_upper_bounds(), Model::continuous_variables(), Dakota::copy_data(), NL2-
SOLLeastSq::covreq, NL2SOLLeastSq::delta0, NL2SOLLeastSq::dltf, NL2SOLLeastSq::dltj, NL2SOLLeast-
Sq::fprec, Model::gradient_type(), Iterator::iteratedModel, NL2SOLLeastSq::lmax0, NL2SOLLeastSq::lmaxs, N-
L2SOLLeastSq::mxfcal, NL2SOLLeastSq::mxiter, NL2SOLLeastSq::mxsolInstance, Minimizer::numContinuous-
Vars, LeastSq::numLeastSqTerms, NL2SOLLeastSq::outlev, NL2SOLLeastSq::rdreq, LeastSq::retrievedIterPri-
Fns, NL2SOLLeastSq::rfctol, NL2SOLLeastSq::sctol, Minimizer::speculativeFlag, Minimizer::vendorNumerical-
GradFlag, NL2SOLLeastSq::xtol, and NL2SOLLeastSq::xf toll.

The documentation for this class was generated from the following files:

• NL2SOLLeastSq.hpp
• NL2SOLLeastSq.cpp
14.118 NL2SOLLeastSqTraits Class Reference

A version of TraitsBase specialized for NL2SOL nonlinear least squares library. Inheritance diagram for NL2SOLLeastSqTraits:

```
NL2SOLLeastSqTraits
    TraitsBase
```

Public Member Functions

- **NL2SOLLeastSqTraits ()**
  
  *default constructor*

- **virtual ~NL2SOLLeastSqTraits ()**
  
  *destructor*

- **virtual bool is_derived ()**
  
  *A temporary query used in the refactor.*

- **bool supports_continuous_variables ()**
  
  *Return the flag indicating whether method supports continuous variables.*

14.118.1 Detailed Description

A version of TraitsBase specialized for NL2SOL nonlinear least squares library. The documentation for this class was generated from the following file:

- NL2SOLLeastSq.hpp

14.119 NLPQLPTraits Class Reference

Wrapper class for the NLPQLP optimization library, Version 2.0. Inheritance diagram for NLPQLPTraits:

```
NLPQLPTraits
    TraitsBase
```

Public Member Functions

- **NLPQLPTraits ()**
  
  *default constructor*

- **virtual ~NLPQLPTraits ()**
  
  *destructor*
• virtual bool is_derived ()
  A temporary query used in the refactor.
• bool supports_continuous_variables ()
  Return the flag indicating whether method supports continuous variables.
• bool supports_linear_equality ()
  Return the flag indicating whether method supports linear equalities.
• bool supports_linear_inequality ()
  Return the flag indicating whether method supports linear inequalities.
• bool supports_nonlinear_equality ()
  Return the flag indicating whether method supports nonlinear equalities.
• bool supports_nonlinear_inequality ()
  Return the flag indicating whether method supports nonlinear inequalities.
• NONLINEAR_INEQUALITY_FORMAT nonlinear_inequality_format ()
  Return the format used for nonlinear inequality constraints.

14.119.1 Detailed Description

Wrapper class for the NLPQLP optimization library, Version 2.0.

AN IMPLEMENTATION OF A SEQUENTIAL QUADRATIC PROGRAMMING METHOD FOR SOLVING NONLINEAR OPTIMIZATION PROBLEMS BY DISTRIBUTED COMPUTING AND NON-MONOTONE LINE SEARCH

This subroutine solves the general nonlinear programming problem

\[
\begin{align*}
\text{minimize} & \quad F(X) \\
\text{subject to} & \quad G(j,X) = 0, \quad j=1,\ldots,ME \\
& \quad G(j,X) \geq 0, \quad j=ME+1,\ldots,M \\
& \quad XL \leq X \leq XU
\end{align*}
\]

and is an extension of the code NLPQLD. NLPQLP is specifically tuned to run under distributed systems. A new input parameter L is introduced for the number of parallel computers, that is the number of function calls to be executed simultaneously. In case of L=1, NLPQLP is identical to NLPQLD. Otherwise the line search is modified to allow L parallel function calls in advance. Moreover the user has the opportunity to used distributed function calls for evaluating gradients.

The algorithm is a modification of the method of Wilson, Han, and Powell. In each iteration step, a linearly constrained quadratic programming problem is formulated by approximating the Lagrangian function quadratically and by linearizing the constraints. Subsequently, a one-dimensional line search is performed with respect to an augmented Lagrangian merit function to obtain a new iterate. Also the modified line search algorithm guarantees convergence under the same assumptions as before.

For the new version, a non-monotone line search is implemented which allows to increase the merit function in case of instabilities, for example caused by round-off errors, errors in gradient approximations, etc.

The subroutine contains the option to predetermine initial guesses for the multipliers or the Hessian of the Lagrangian function and is called by reverse communication. A version of TraitsBase specialized for NLPQLP optimizers

The documentation for this class was generated from the following file:

• NLPQLPOptimizer.hpp
14.120  NLSSOLLeastSq Class Reference

Wrapper class for the NLSSOL nonlinear least squares library.

Inheritance diagram for NLSSOLLeastSq:

```
  Iterator
  |
  V
Minimizer
  |
  V
LeastSq   SOLBase
  |
  V
  NLSSOLLeastSq
```

Public Member Functions

- `NLSSOLLeastSq (ProblemDescDB &problem_db, Model &model)`
  *standard constructor*
- `NLSSOLLeastSq (Model &model)`
  *alternate constructor*
- `~NLSSOLLeastSq ()`
  *destructor*
- `void core_run ()`
  *core portion of run; implemented by all derived classes and may include pre/post steps in lieu of separate pre/post*

Static Private Member Functions

- static void `least_sq_eval (int &mode, int &m, int &n, int &nrowfj, double *x, double *f, double *gradf, int &nstate)`
  *Evaluator for NLSSOL: computes the values and first derivatives of the least squares terms (passed by function pointer to NLSSOL).*

Static Private Attributes

- static `NLSSOLLeastSq * nlssolInstance`
  *pointer to the active object instance used within the static evaluator functions in order to avoid the need for static data*

Additional Inherited Members

14.120.1  Detailed Description

Wrapper class for the NLSSOL nonlinear least squares library.

The NLSSOLLeastSq class provides a wrapper for NLSSOL, a Fortran 77 sequential quadratic programming library from Stanford University marketed by Stanford Business Associates. It uses a function pointer approach
for which passed functions must be either global functions or static member functions. Any nonstatic attribute used within static member functions must be either local to that function or accessed through a static pointer.

The user input mappings are as follows: max function evaluations is implemented directly in NLSSOLLeastSq's evaluator functions since there is no NLSSOL parameter equivalent, and max iterations, convergence tolerance, output verbosity, verify level, function precision, and linesearch tolerance are mapped into NLSSOL's "Major Iteration Limit", "Optimality Tolerance", "Major Print Level" (verbose: Major Print Level = 20; quiet: Major Print Level = 10), "Verify Level", "Function Precision", and "Linesearch Tolerance" parameters, respectively, using NLSSOL's npoptn() subroutine (as wrapped by npoptn2() from the npoptn_wrapper.f file). Refer to [Gill, P.E., Murray, W., Saunders, M.A., and Wright, M.H., 1986] for information on NLSSOL's optional input parameters and the npoptn() subroutine.

14.120.2 Constructor & Destructor Documentation

NLSSOLLeastSq (ProblemDescDB & problem_db, Model & model)

standard constructor
This is the primary constructor. It accepts a Model reference.
References Minimizer::constraintTol, Iterator::convergenceTol, Model::fd_gradient_step_size(), ProblemDescDB::get_int(), ProblemDescDB::get_real(), Model::gradient_type(), Iterator::iteratedModel, Iterator::maxIterations, Iterator::outputLevel, Iterator::probDescDB, SOLBase::set_options(), Minimizer::speculativeFlag, and Minimizer::vendorNumericalGradFlag.

alternate constructor
This is an alternate constructor which accepts a Model but does not have a supporting method specification from the ProblemDescDB.
References Minimizer::constraintTol, Iterator::convergenceTol, Model::fd_gradient_step_size(), Model::gradient_type(), Iterator::iteratedModel, Iterator::maxIterations, Iterator::outputLevel, SOLBase::set_options(), Minimizer::speculativeFlag, and Minimizer::vendorNumericalGradFlag.

14.120.3 Member Function Documentation

void core_run ( ) [virtual]
core portion of run; implemented by all derived classes and may include pre/post steps in lieu of separate pre/post run function for the iterator class hierarchy. All derived classes need to redefine it.
Reimplemented from Iterator.
References SOLBase::allocate_arrays(), SOLBase::allocate_workspace(), SOLBase::augment_bounds(), LeastSq::bestIterPriFns, Iterator::bestResponseArray, Iterator::bestVariablesArray, SOLBase::constraint_eval(), SOLBase::constraintJacMatrixF77, SOLBase::constraintState, SOLBase::constrOffset, Model::continuous_lower_bounds(), Model::continuous_upper_bounds(), Model::continuous_variables(), Dakota::copy_data(), Dakota::copy_data_partial(), SOLBase::deallocate_arrays(), SOLBase::fnEvalCntr, SOLBase::informResult, SOLBase::intWorkSpace, SOLBase::intWorkSpaceSize, Iterator::iteratedModel, NLSSOLLeastSq::least_sq_eval(), SOLBase::linConstraintArraySize, SOLBase::linConstraintMatrixF77, Model::linear_eq_constraint_coeffs(), Model::linear_ineq_constraint_coeffs(), SOLBase::linConstraintArraySize, NLSSOLLeastSq::nlssolInstance, SOLBase::numberIterations, Minimizer::numContinuousVars, LeastSq::numLeastSqTerms, Minimizer::numLinearConstraints, Minimizer::numNonLinearConstraints, Minimizer::numUserPrimaryFns, SOLBase::optLsqInstance, Minimizer::prevMinInstance, SOLBase::realWorkSpace, SOLBase::realWorkSpaceSize, LeastSq::retrievedIterPriFns, SOLBase::sollInstance, and SOLBase::upperFactorHessianF77.

The documentation for this class was generated from the following files:
14.121 NLSSOLLeastSqTraits Class Reference

A version of TraitsBase specialized for NLSSOL nonlinear least squares library.

Inheritance diagram for NLSSOLLeastSqTraits:

```
+----------------+-------------------+
| TraitsBase     | TraitsBase        |
|                | NLSSOLLeastSqTraits |
```

Public Member Functions

- **NLSSOLLeastSqTraits ()**
  
  *default constructor*

- **virtual ~NLSSOLLeastSqTraits ()**
  
  *destructor*

- **virtual bool is_derived ()**
  
  *A temporary query used in the refactor.*

- **bool supports_continuous_variables ()**
  
  *Return the flag indicating whether method supports continuous variables.*

- **bool supports_linear_equality ()**
  
  *Return the flag indicating whether method supports linear equalities.*

- **bool supports_linear_inequality ()**
  
  *Return the flag indicating whether method supports linear inequalities.*

- **bool supports_nonlinear_equality ()**
  
  *Return the flag indicating whether method supports nonlinear equalities.*

- **bool supports_nonlinear_inequality ()**
  
  *Return the flag indicating whether method supports nonlinear inequalities.*

14.121.1 Detailed Description

A version of TraitsBase specialized for NLSSOL nonlinear least squares library.

The documentation for this class was generated from the following file:

- NLSSOLLeastSq.hpp

14.122 NoDBBaseConstructor Struct Reference

Dummy struct for overloading constructors used in on-the-fly instantiations without ProblemDescDB support.
Public Member Functions

- **NoDBBaseConstructor** (int=0)
  
  *C++ structs can have constructors.*

14.122.1 Detailed Description

Dummy struct for overloading constructors used in on-the-fly instantiations without ProblemDescDB support. 

`NoDBBaseConstructor` is used to overload the constructor used for on-the-fly instantiations in which ProblemDescDB queries cannot be used. Putting this struct here avoids circular dependencies.

The documentation for this struct was generated from the following file:

- dakota_global_defs.hpp

14.123 NomadTraits Class Reference

Wrapper class for NOMAD Optimizer.

Inheritance diagram for NomadTraits:

```
TraitsBase
     └─ NomadTraits
```

Public Member Functions

- **NomadTraits ()**
  
  *default constructor*

- **virtual ~NomadTraits ()**
  
  *destructor*

- **virtual bool is_derived ()**
  
  *A temporary query used in the refactor*

- **bool supports_continuous_variables ()**
  
  *Return the flag indicating whether method supports continuous variables.*

- **bool supports_discrete_variables ()**
  
  *Return the flag indicating whether method supports discrete variables.*

- **bool supports_nonlinear_equality ()**
  
  *Return the flag indicating whether method supports nonlinear equalities.*

- **NONLINEAR_EQUALITY_FORMAT nonlinear_equality_format ()**
  
  *Return the format used for nonlinear equality constraints.*

- **bool supports_nonlinear_inequality ()**
  
  *Return the flag indicating whether method supports nonlinear inequalities.*

- **NONLINEAR_INEQUALITY_FORMAT nonlinear_inequality_format ()**
  
  *Return the format used for nonlinear inequality constraints.*
 Wrapper class for NOMAD \texttt{Optimizer}.

 NOMAD (is a Nonlinear Optimization by Mesh Adaptive Direct search) is a simulation-based optimization package designed to efficiently explore a design space using Mesh Adaptive Search.

 Mesh Adaptive Direct Search uses Meshes, discretizations of the domain space of variables. It generates multiple meshes, and as its name implies, it also adapts the refinement of the meshes in order to find the best solution of a problem.

 The objective of each iteration is to find points in a mesh that improves the current solution. If a better solution is not found, the next iteration is done over a finer mesh.

 Each iteration is composed of two steps: Search and Poll. The Search step finds any point in the mesh in an attempt to find an improvement; while the Poll step generates trial mesh points surrounding the current best current solution.

 The \texttt{NomadOptimizer} is a wrapper for the NOMAD library. It features the following attributes: \texttt{max\_function\_evaluations}, \texttt{display\_format}, \texttt{display\_all\_evaluations}, \texttt{function\_precision}, \texttt{max\_iterations}. A version of \texttt{TraitsBase} specialized for Nomad

 The documentation for this class was generated from the following file:

 - \texttt{NomadOptimizer.hpp}

---

\textbf{14.124 NonD Class Reference}

 Base class for all nondeterministic iterators (the DAKOTA/UQ branch).

 Inheritance diagram for \texttt{NonD}:
CHAPTER 14. CLASS DOCUMENTATION

Public Member Functions

- void `requested_levels` (const RealVectorArray &reqRespLevels, const RealVectorArray &reqProbLevels, const RealVectorArray &reqRelLevels, const RealVectorArray &reqGenRelLevels, short respLevTgt, short respLevTgtReduce, bool cdfFlag, bool pdfOutput)
  
  set requestedRespLevels, requestedProbLevels, requestedRelLevels, requestedGenRelLevels, respLevelTarget, cdfFlag, and pdfOutput (used in combination with alternate ctors)

- void `print_level_mappings` (std::ostream &s) const
  
  prints the z/p/beta/beta* mappings reflected in {requested,computed}{Resp,Prob,Rel,GenRel}Levels for default qoi type and qoi labels

- void `print_level_mappings` (std::ostream &s, String qoiType, const StringArray &qoiLabels) const
  
  prints the z/p/beta/beta* mappings reflected in {requested,computed}{Resp,Prob,Rel,GenRel}Levels

- void `print_level_mappings` (std::ostream &s, const RealVector &levelMaps, bool momentOffset, const String &prepend="")
  
  print level mapping statistics using optional pre-pend

- bool `resize`() 
  
  reinitializes iterator based on new variable size

- bool `pdf_output`() const 
  
  get pdfOutput

- void `pdf_output`(bool output)
set pdfOutput

- short final_moments_type () const
  get finalMomentsType
- void final_moments_type (short type)
  set finalMomentsType

Protected Member Functions

- NonD (ProblemDescDB &problem_db, Model &model)
  constructor
- NonD (unsigned short method_name, Model &model)
  alternate constructor for sample generation and evaluation "on the fly"
- NonD (unsigned short method_name, const RealVector &lower_bnds, const RealVector &upper_bnds)
  alternate constructor for sample generation "on the fly"
- ~NonD ()
  destructor
- void derived_set_communicators (ParLevLIter pl_iter)
  derived class contributions to setting the communicators associated with this Iterator instance
- void initialize_run ()
  utility function to perform common operations prior to pre_run(); typically memory initialization; setting of instance pointers
- void finalize_run ()
  utility function to perform common operations following post_run(); deallocation and resetting of instance pointers
- const Response & response_results () const
  return the final statistics from the nondeterministic iteration
- void response_results.active_set (const ActiveSet &set)
  set the active set within finalStatistics
- virtual void initialize_response_covariance ()
  initializes respCovariance
- virtual void initialize_final_statistics ()
  initializes finalStatistics for storing NonD final results
- virtual void update_final_statistics ()
  update finalStatistics::functionValues
- void pull_level_mappings (RealVector &level_maps)
  concatenate computed {Resp,Prob,Rel,GenRel} Levels into level_maps
- void push_level_mappings (const RealVector &level_maps)
  update computed {Resp,Prob,Rel,GenRel} Levels from level_maps
- void load_pilot_sample (const SizetArray &pilot_spec, SizetArray &delta_N_l)
  distribute pilot sample specification across model levels
- void load_pilot_sample (const SizetArray &pilot_spec, const Sizet3DArray &N_l, SizetArray &delta_N_l)
  distribute pilot sample specification across model forms or levels
- void load_pilot_sample (const SizetArray &pilot_spec, const Sizet3DArray &N_l, Sizet2DArray &delta_N_l)
  distribute pilot sample specification across model forms and levels
• void resizefinal_statistics_gradients ()
  resizes finalStatistics::functionGradients based on finalStatistics ASV

• void update_aleatory_final_statistics ()
  update finalStatistics::functionValues from momentStats and computed{Prob,Rel,GenRel,Resp}Levels

• void update_system_final_statistics ()
  update system metrics from component metrics within finalStatistics

• void update_system_final_statistics_gradients ()
  update finalStatistics::functionGradients

• void initialize_level_mappings ()
  size computed{Resp,Prob,Rel,GenRel}Levels

• void compute_densities (const RealRealPairArray &min_max_fns, bool prob_refinement=false, bool all_levels_computed=false)
  compute the PDF bins from the CDF/CCDF values and store in computedPDF{Abscissas,Ordinates}

• void print_densities (std::ostream &s) const
  output the PDFs reflected in computedPDF{Abscissas,Ordinates} using default qoi_type and pdf_labels

• void print_densities (std::ostream &s, String qoi_type, const StringArray &pdf_labels) const
  output the PDFs reflected in computedPDF{Abscissas,Ordinates}

• void print_system_mappings (std::ostream &s) const
  print system series/parallel mappings for response levels

• void print_multilevel_evaluation_summary (std::ostream &s, const SizetArray &N_samp)
  print evaluation summary for multilevel sampling across 1D profile

• void print_multilevel_evaluation_summary (std::ostream &s, const Sizet2DArray &N_samp)
  print evaluation summary for multilevel sampling across 2D profile

• void print_multilevel_evaluation_summary (std::ostream &s, const Sizet3DArray &N_samp)
  print evaluation summary for multilevel sampling across 3D profile

• void construct_lhs (Iterator &u_space_sampler, Model &u_model, unsigned short sample_type, int num_samples, int seed, const String &rng, bool vary_pattern, short sampling_vars_mode=ACTIVE)
  assign a NonDLHSSampling instance within u_space_sampler

• size_t oned_sided_delta (Real current, Real target)
  compute a one-sided sample increment for multilevel methods to move current sampling level to a new target

• void archive_allocate_mappings ()
  allocate results array storage for distribution mappings

• void archive_from resp (size_t fn_index, size_t inc_id=0)
  archive the mappings from specified response levels for specified fn

• void archive_to resp (size_t fn_index, size_t inc_id=0)
  archive the mappings to computed response levels for specified fn and (optional) increment id.

• void archive_allocate_pdf ()
  allocate results array storage for pdf histograms

• void archive_pdf (size_t fn_index, size_t inc_id=0)
  archive a single pdf histogram for specified function

• void archive_equiv_hf_evals (const Real equiv_hf_evals)
  archive the equivalent number of HF evals (used by ML/MF methods)
Protected Attributes

- **NonD * prevNondInstance**
  pointer containing previous value of nondInstance
- **size_t startCAUV**
  starting index of continuous aleatory uncertain variables within active continuous variables (convenience for managing offsets)
- **size_t numCAUV**
  number of active continuous aleatory uncertain variables
- **bool epistemicStats**
  flag for computing interval-type metrics instead of integrated metrics. If any epistemic vars are active in a metric evaluation, then flag is set.
- **RealMatrix momentStats**
  standardized or central resp moments, as determined by finalMomentsType. Calculated in compute_moments() and indexed as (moment,fn).
- **RealVectorArray requestedRespLevels**
  requested response levels for all response functions
- **RealVectorArray computedProbLevels**
  output probability levels for all response functions resulting from requestedRespLevels
- **RealVectorArray computedRelLevels**
  output reliability levels for all response functions resulting from requestedRespLevels
- **RealVectorArray computedGenRelLevels**
  output generalized reliability levels for all response functions resulting from requestedRespLevels
- **short respLevelTarget**
  indicates mapping of $z \rightarrow p$ (PROBABILITIES), $z \rightarrow \beta$ (RELIABILITIES), or $z \rightarrow \beta^*$ (GEN.RELIABILITIES)
- **short respLevelTargetReduce**
  indicates component or system series/parallel failure metrics
- **RealVectorArray requestedProbLevels**
  requested probability levels for all response functions
- **RealVectorArray requestedRelLevels**
  requested reliability levels for all response functions
- **RealVectorArray requestedGenRelLevels**
  requested generalized reliability levels for all response functions
- **RealVectorArray computedRespLevels**
  output response levels for all response functions resulting from requestedProbLevels, requestedRelLevels, or requestedGenRelLevels
- **size_t totalLevelRequests**
  total number of levels specified within requestedRespLevels, requestedProbLevels, and requestedRelLevels
- **bool cdfFlag**
  flag for type of probabilities/reliabilities used in mappings: cumulative/CDF (true) or complementary/CCDF (false)
- **bool pdfOutput**
  flag for managing output of response probability density functions (PDFs)
- **RealVectorArray computedPDFAbscissas**
  sorted response PDF intervals bounds extracted from min/max sample and requested/computedRespLevels (vector lengths = num bins + 1)
CHAPTER 14. CLASS DOCUMENTATION

- **RealVectorArray computedPDFOrdinates**
  response PDF densities computed from bin counts divided by (unequal) bin widths (vector lengths = num bins)

- **Response finalStatistics**
  final statistics from the uncertainty propagation used in strategies: response means, standard deviations, and probabilities of failure

- **short finalMomentsType**
  type of moments logged within finalStatistics: none, central, standard

- **size_t miPLIndex**
  index for the active ParallelLevel within ParallelConfiguration::miPLIters

- **BitArray pdfComputed**
  Whether PDF was computed for function i; used to determine whether a pdf should be archived.

**Static Protected Attributes**

- **static NonD * nondInstance**
  pointer to the active object instance used within static evaluator functions in order to avoid the need for static data

**Private Member Functions**

- **void initialize_counts ()**
  initialize data based on variable counts

- **void distribute_levels (RealVectorArray &levels, bool ascending=true)**
  convenience function for distributing a vector of levels among multiple response functions if a short-hand specification is employed.

- **void level_mappings_file (size_t fn_index, const String &qoi_label) const**
  Write level mappings to a file for a single response.

- **void print_level_map (std::ostream &s, size_t fn_index, const String &qoi_label) const**
  Print level mapping for a single response function to ostream.

- **bool homogeneous (const SizetArray &N_l) const**
  return true if N_l has consistent values

**Additional Inherited Members**

14.124.1 Detailed Description

Base class for all nondeterministic iterators (the DAKOTA/UQ branch).

The base class for nondeterministic iterators consolidates uncertain variable data and probabilistic utilities for inherited classes.

14.124.2 Member Function Documentation

**void print_level_mappings ( std::ostream &s, String qoi_type, const StringArray &qoi_labels ) const**

prints the z/p/beta/beta+ mappings reflected in {requested,computed} {Resp,Prob,Rel,GenRel} Levels

Print distribution mappings, including to file per response.

References NonD::level_mappings_file(), Analyzer::numFunctions, Iterator::outputLevel, NonD::print_densities(), NonD::print_level_map(), NonD::requestedGenRelLevels, NonD::requestedProbLevels, NonD::requestedRelLevels, NonD::requestedRespLevels, and Dakota::write_precision.
void print_level_mappings ( std::ostream & s, const RealVector & level_maps, bool moment_offset, const String & prepend = ""
)
print level mapping statistics using optional pre-pend
This version differs in its use of a concatenated vector of level mappings, rather than computed{Resp,Prob,Real,GenRel}Levels.
References NonD::cdfFlag, Iterator::iteratedModel, Analyzer::numFunctions, NonD::requestedGenRelLevels, NonD::requestedProbLevels, NonD::requestedRelLevels, NonD::requestedRespLevels, NonD::respLevelTarget, Model::response_labels(), and Dakota::write_precision.

void initialize_run ( ) [inline], [protected], [virtual]
utility function to perform common operations prior to pre_run(); typically memory initialization; setting of instance pointers
Perform initialization phases of run sequence, like allocating memory and setting instance pointers. Commonly used in sub-iterator executions. This is a virtual function; when re-implementing, a derived class must call its nearest parent’s initialize_run(), typically before performing its own implementation steps.
Reimplemented from Analyzer.
References Analyzer::initialize_run(), NonD::nondInstance, and NonD::prevNondInstance.

void finalize_run ( ) [inline], [protected], [virtual]
utility function to perform common operations following post_run(); deallocation and resetting of instance pointers
Optional: perform finalization phases of run sequence, like deallocating memory and resetting instance pointers. Commonly used in sub-iterator executions. This is a virtual function; when re-implementing, a derived class must call its nearest parent’s finalize_run(), typically after performing its own implementation steps.
Reimplemented from Analyzer.
References Analyzer::finalize_run(), NonD::nondInstance, and NonD::prevNondInstance.

void initialize_final_statistics ( ) [protected], [virtual]
initializes finalStatistics for storing NonD final results
Default definition of virtual function (used by sampling, reliability, and stochastic expansion methods) defines the set of statistical results to include the first two moments and level mappings for each QoI.
Reimplemented in NonDInterval.
References Dakota::abort_handlert(), NonD::cdfFlag, Model::cv(), ActiveSet::derivative_vector(), NonD::epistemicStats, NonD::finalMomentsType, NonD::finalStatistics, Response::function_labels(), Model::inactive_continuous_variable_ids(), Iterator::iteratedModel, Dakota::length(), Analyzer::numFunctions, NonD::requestedGenRelLevels, NonD::requestedProbLevels, NonD::requestedRelLevels, NonD::requestedRespLevels, NonD::respLevelTarget, NonD::respLevelTargetReduce, and NonD::totalLevelRequests.
Referenced by NonDExpansion::NonDExpansion(), NonDIntegration::NonDIntegration(), NonDReliability::NonDReliability(), NonDSampling::NonDSampling(), NonD::requested_levels(), and NonDReliability::resize().

void compute_densities ( const RealRealPairArray & min_max_fns, bool prob_refinement = false, bool all_levels_computed = false ) [protected]
compute the PDF bins from the CDF/CCDF values and store in computedPDF{Abscissas,Ordiates}
This function infers PDFs from the CDF/CCDF level mappings, in order to enable PDF computation after CDF/CCDF probability level refinement (e.g., from importance sampling).
prob_refinement alerts the routine to exclude inverse mappings from the PDF, since refinement only applies to z→p forward mappings and mixing refined and unrefined probability mappings results in an inconsistency (potentially manifesting as negative density values).

all_levels_computed is an option used by reliability methods where computed+Levels are defined across the union of all requested levels.

References NonD::archive_allocate_pdf(), NonD::cdfFlag, NonD::computedGenRelLevels, NonD::computedPDFAbscissas, NonD::computedPDFOrdinates, NonD::computedProbLevels, NonD::computedRespLevels, Analyzer::numFunctions, NonD::pdfComputed, NonD::pdfOutput, NonD::requestedGenRelLevels, NonD::requestedProbLevels, NonD::requestedRelLevels, NonD::requestedRespLevels, and NonD::respLevelTarget.

Referenced by NonDSampling::compute_level_mappings(), NonDExpansion::compute_numerical_statistics(), NonDLocalReliability::core_run(), NonDAdaptImpSampling::core_run(), and NonDGlobalReliability::importance_sampling().

```cpp
void level_mappings_file ( size_t fn_index, const String & qoi_label ) const [private]
```

Write level mappings to a file for a single response.
Write distribution mappings to a file for a single response.
References NonD::print_level_map(), and Dakota::write_precision.
Referenced by NonD::print_level_mappings().

```cpp
void print_level_map ( std::ostream & s, size_t fn_index, const String & qoi_label ) const [private]
```

Print level mapping for a single response function to ostream.
Print the distribution mapping for a single response function to the passed output stream. This base class version maps from one requested level type to one computed level type; some derived class implementations (e.g., local and global reliability) output multiple computed level types.
References NonD::cdfFlag, NonD::computedGenRelLevels, NonD::computedProbLevels, NonD::computedRelLevels, NonD::computedRespLevels, NonD::requestedGenRelLevels, NonD::requestedProbLevels, NonD::requestedRelLevels, NonD::requestedRespLevels, NonD::respLevelTarget, and Dakota::write_precision.
Referenced by NonD::level_mappings_file(), and NonD::print_level_mappings().
The documentation for this class was generated from the following files:

- DakotaNonD.hpp
- DakotaNonD.cpp

14.125 NonDAdaptImpSampling Class Reference

Class for the Adaptive Importance Sampling methods within DAKOTA.
Inheritance diagram for NonDAdaptImpSampling:
Public Member Functions

- **NonDAdaptImpSampling (ProblemDescDB &problem_db, Model &model)**
  
  *standard constructor*

- **NonDAdaptImpSampling (Model &model, unsigned short sample_type, int samples, int seed, const String &rng, bool vary_pattern, unsigned short is_type, bool cdf_flag, bool x_space_model, bool use_model_bounds, bool track_extreme)**
  
  *alternate constructor for on-the-fly instantiations*

- **~NonDAdaptImpSampling ()**
  
  *destructor*

- **bool resize ()**
  
  *reinitializes iterator based on new variable size*

- **void derived_init_communicators (ParLevLIter pl_iter)**
  
  *derived class contributions to initializing the communicators associated with this Iterator instance*

- **void derived_set_communicators (ParLevLIter pl_iter)**
  
  *derived class contributions to setting the communicators associated with this Iterator instance*

- **void derived_free_communicators (ParLevLIter pl_iter)**
  
  *derived class contributions to freeing the communicators associated with this Iterator instance*

- **void nested_variable_mappings (const SizetArray &c_index1, const SizetArray &di_index1, const SizetArray &ds_index1, const SizetArray &dr_index1, const ShortArray &c_target2, const ShortArray &di_target2, const ShortArray &ds_target2)**
  
  *set primaryA{CV,DIV,DRV}MapIndices, secondaryA{CV,DIV,DRV}MapTargets within derived Iterators; supports computation of higher-level sensitivities in nested contexts (e.g., derivatives of statistics w.r.t. inserted design variables)*

- **void core_run ()**
  
  *performs adaptive importance sampling and computes probability of failure*

- **void print_results (std::ostream &s, short results_state=FINAL_RESULTS)**
  
  *print the final statistics*

- **unsigned short sampling_scheme () const**
  
  *return importanceSamplingType*

- **int refinement_samples () const**
  
  *return refineSamples*
• void initialize (const RealVectorArray &full_points, bool x_space_data, size_t resp_index, Real initial_prob, Real failure_threshold)
  initializes data needed for importance sampling: an initial set of points around which to sample, a failure threshold, an initial probability to refine, and flags to control transformations

• void initialize (const RealMatrix &full_points, bool x_space_data, size_t resp_index, Real initial_prob, Real failure_threshold)
  initializes data needed for importance sampling: an initial set of points around which to sample, a failure threshold, an initial probability to refine, and flags to control transformations

• void initialize (const RealVector &full_point, bool x_space_data, size_t resp_index, Real initial_prob, Real failure_threshold)
  initializes data needed for importance sampling: an initial point around which to sample, a failure threshold, an initial probability to refine, and flags to control transformations

• Real final_probability ()
  returns the final probability calculated by the importance sampling

• const RealRealPairArray & extreme_values () const
  return extremeValues

Private Member Functions

• void select_rep_points (const RealVectorArray &var_samples_u, const RealVector &fn_samples)
  select representative points from a set of samples

• void converge_statistics (bool cov_flag)
  iteratively generate samples and select representative points until probability and (optionally) coefficient of variation converge

• void generate_samples (RealVectorArray &var_samples_u)
  generate a set of samples based on multimodal sampling density

• void evaluate_samples (const RealVectorArray &var_samples_u, RealVector &fn_samples)
  evaluate the model at the sample points and store the responses

• void calculate_statistics (const RealVectorArray &var_samples_u, const RealVector &fn_samples, size_t total_samples, Real &sum_prob, Real &prob, bool compute_cov, Real &sum_var, Real &cov)
  calculate the probability of exceeding the failure threshold and the coefficient of variation (if requested)

• Real distance (const RealVector &a, const RealVector &b)
  compute Euclidean distance between points a and b

• Real recentered_density (const RealVector &sample_point)
  compute density between a representative point and a sample point, assuming standard normal

Private Attributes

• Model uSpaceModel
  importance sampling is performed in standardized probability space. This u-space model is either passed in (alternate constructor for helper AIS) or constructed using ProbabilityTransformModel (standard constructor for stand-alone AIS)

• unsigned short importanceSamplingType
  integration type (is, ais, mmais) provided by input specification

• bool initLHS
  flag to identify if initial points are generated from an LHS sample
14.125. NONDADAPTIMPSAMPLING CLASS REFERENCE

- bool useModelBounds
  flag to control if the sampler should respect the model bounds
- bool invertProb
  flag for inversion of probability values using 1.-p
- bool trackExtremeValues
  flag for tracking min/max values encountered when evaluating samples
- int refineSamples
  size of sample batch within each refinement iteration
- size_t respFnIndex
  the active response function index in the model to be sampled
- RealVector designPoint
  design subset for which uncertain subset is being sampled
- RealVectorArray initPointsU
  the original set of u-space samples passed in initialize()
- RealVectorArray repPointsU
  the set of representative points in u-space around which to sample
- RealVector repWeights
  the weight associated with each representative point
- Real probEstimate
  the probability estimate that is iteratively refined by importance sampling
- Real failThresh
  the failure threshold (z-bar) for the problem.

Additional Inherited Members

14.125.1 Detailed Description

Class for the Adaptive Importance Sampling methods within DAKOTA.

The NonDAdaptImpSampling implements the multi-modal adaptive importance sampling used for reliability calculations. (eventually we will want to broaden this). Need to add more detail to this description.

14.125.2 Constructor & Destructor Documentation

NonDAdaptImpSampling ( ProblemDescDB & problem_db, Model & model )

standard constructor

This is the primary constructor. It accepts a Model reference. It will perform refinement for all response QOI and all probability levels.

References Dakota::abort_handler(), Model::assign_rep(), NonD::finalMomentsType, ProblemDescDB::get-_iv(), Iterator::iteratedModel, NonDSampling::numSamples, Iterator::probDescDB, NonDAdaptImpSampling::refine-Samples, NonDSampling::sampleType, NonDSampling::statsFlag, NonDAdaptImpSampling::useModelBounds, and NonDAdaptImpSampling::uSpaceModel.
CHAPTER 14. CLASS DOCUMENTATION

NonDAdaptImpSampling ( Model & \textit{model}, unsigned short \textit{sample\_type}, int \textit{refine\_samples}, int \textit{refine\_seed}, const String & \textit{rng}, bool \textit{vary\_pattern}, unsigned short \textit{is\_type}, bool \textit{cdf\_flag}, bool \textit{x\_space\_model}, bool \textit{use\_model\_bounds}, bool \textit{track\_extreme} )

alternate constructor for on-the-fly instantiations

This is an alternate constructor for instantiations on the fly using a \textit{Model} but no \textit{ProblemDescDB}. It will perform refinement for one response QOI and one probability level (passed in \textit{initialize}).

References \textit{Model}\texttt{::assign\_rep()}, NonD::cdf\_Flag, NonDSampling::extreme\_Values, NonD::final\_Moments\_Type, Iterator::max\_Eval\_Concurrency, Analyzer::num\_Functions, NonDAdaptImpSampling::refine\_Samples, NonDAdaptImpSampling::track\_Extreme\_Values, NonDAdaptImpSampling::use\_Model\_Bounds, and NonDAdaptImpSampling::uSpace\_Model.

14.125.3 Member Function Documentation

\begin{description}
\item [void \textit{initialize} ( \texttt{const RealVector\_Array & \textit{acv\_points}, bool \textit{x\_space\_data}, size\_t \textit{resp\_index}, Real \textit{initial\_prob}, Real \textit{failure\_threshold} )] \end{description}

initializes data needed for importance sampling: an initial set of points around which to sample, a failure threshold, an initial probability to refine, and flags to control transformations

Initializes data using a vector array of starting points.

References NonDAdaptImpSampling::design\_Point, NonDAdaptImpSampling::fail\_Thresh, NonDAdaptImpSampling::init\_Points\_U, NonDAdaptImpSampling::invert\_Prob, NonD::num\_CAUV, Model::probability\_transformation(), NonDAdaptImpSampling::prob\_Estimate, NonDAdaptImpSampling::resp\_Fn\_Index, NonD::start\_CAUV, and NonDAdaptImpSampling::uSpace\_Model.

Referenced by NonDExpansion::compute\_numerical\_stat\_refinements(), NonDAdaptImpSampling::core\_run(), and NonDGlobalReliability::importance\_sampling().

\begin{description}
\item [void \textit{initialize} ( \texttt{const Real\_Matrix & \textit{acv\_points}, bool \textit{x\_space\_data}, size\_t \textit{resp\_index}, Real \textit{initial\_prob}, Real \textit{failure\_threshold} )] \end{description}

initializes data needed for importance sampling: an initial set of points around which to sample, a failure threshold, an initial probability to refine, and flags to control transformations

Initializes data using a matrix of starting points.

References NonDAdaptImpSampling::design\_Point, NonDAdaptImpSampling::fail\_Thresh, NonDAdaptImpSampling::init\_Points\_U, NonDAdaptImpSampling::invert\_Prob, NonD::num\_CAUV, Analyzer::num\_Continuous\_Vars, Model::probability\_transformation(), NonDAdaptImpSampling::prob\_Estimate, NonDAdaptImpSampling::resp\_Fn\_Index, NonD::start\_CAUV, and NonDAdaptImpSampling::uSpace\_Model.

\begin{description}
\item [void \textit{initialize} ( \texttt{const RealVector & \textit{acv\_point}, bool \textit{x\_space\_data}, size\_t \textit{resp\_index}, Real \textit{initial\_prob}, Real \textit{failure\_threshold} )] \end{description}

initializes data needed for importance sampling: an initial point around which to sample, a failure threshold, an initial probability to refine, and flags to control transformations

Initializes data using only one starting point.

References NonDAdaptImpSampling::design\_Point, NonDAdaptImpSampling::fail\_Thresh, NonDAdaptImpSampling::init\_Points\_U, NonDAdaptImpSampling::invert\_Prob, NonD::num\_CAUV, Model::probability\_transformation(), NonDAdaptImpSampling::prob\_Estimate, NonDAdaptImpSampling::resp\_Fn\_Index, NonD::start\_CAUV, and NonDAdaptImpSampling::uSpace\_Model.

The documentation for this class was generated from the following files:

- NonDAdaptImpSampling.hpp
- NonDAdaptImpSampling.cpp
14.126 NonDAdaptiveSampling Class Reference

Class for testing various Adaptively sampling methods using geometric, statistical, and topological information of the surrogate.

Inheritance diagram for NonDAdaptiveSampling:

```
NonDAdaptiveSampling
  NonDSampling
    NonD
      Analyzer
        Iterator
```

Public Member Functions

- **NonDAdaptiveSampling** (ProblemDescDB &problem_db, Model &model)
  - standard constructor
- **~NonDAdaptiveSampling** ()
  - alternate constructor for sample generation and evaluation "on the fly" has not been implemented
- **bool resize ()**
  - reinitializes iterator based on new variable size

Protected Member Functions

- **void derived_init_communicators** (ParLevLIter pl_iter)
  - derived class contributions to initializing the communicators associated with this Iterator instance
- **void derived_set_communicators** (ParLevLIter pl_iter)
  - derived class contributions to setting the communicators associated with this Iterator instance
- **void derived_free_communicators** (ParLevLIter pl_iter)
  - derived class contributions to freeing the communicators associated with this Iterator instance
- **void core_run ()**
  - core portion of run; implemented by all derived classes and may include pre/post steps in lieu of separate pre/post
- **Real final_probability ()**
- **void print_results** (std::ostream &s, short results_state=FINAL_RESULTS)
  - print the final iterator results
Private Member Functions

- void calc score alm()
  
  Function to compute the ALM scores for the candidate points. ALM score is the variance computed by the surrogate at the point.

- void calc score delta x()
  
  Function to compute the Distance scores for the candidate points. Distance score is the shortest distance between the candidate and an existing training point.

- void calc score delta y()
  
  Function to compute the Gradient scores for the candidate points. Gradient score is the function value difference between a candidate's surrogate response and its nearest evaluated true response from the training set.

- void calc score topo bottleneck()
  
  Function to compute the Bottleneck scores for the candidate points. Bottleneck score is computed by determining the bottleneck distance between the persistence diagrams of two approximate Morse-Smale complices. The complices used include one built from only the training data, and another built from the training data and the single candidate.

- void calc score topo avg persistence(int respFnCount)
  
  Function to compute the Average Change in Persistence scores for the candidate points. Avg Persistence score is computed as the average change in persistence each point undergoes between two approximate Morse-Smale complices. The complices used include one built from only the training data, and another built from the training data and the single candidate.

- void calc score topo highest persistence(int respFnCount)
  
  Function to compute the Highest Persistence scores for the candidate points. Highest Persistence score is calculated as a ranking of a set of candidates by constructing an approximate Morse-Smale complex over the entire set of candidates, using their surrogate responses, and the training data, using their true responses, and ranking points based on the most topological significance as measured by their persistence values. In the case where there are no topologically significant points, the point will be chosen randomly. TODO: It may be wiser to fall back to a scheme that ranks points based on proximity to extrema, or the most significant extremal point?

- void calc score topo alm hybrid(int respFnCount)
  
  Function to compute the Hybrid scores for the candidate points. Hybrid score is computed the same as Avg Persistence score except that instead of computing one score, three scores are computed: a mean surface, a mean +/- std. dev. surfaces and then averaging the three separate scores. The hope is that you strike a balance between selecting points in topologically important areas and areas of high uncertainty.

- Real calc score alm(int respFnCount, RealVector &test point)
  
  Same as the other function of the same name, only this allows the user to specify the location of the candidate.

- Real calc score delta x(int respFnCount, RealVector &test point)
  
  Same as the other function of the same name, only this allows the user to specify the location of the candidate.

- Real calc score delta y(int respFnCount, RealVector &test point)
  
  Same as the other function of the same name, only this allows the user to specify the location of the candidate.

- Real calc score topo bottleneck(int respFnCount, RealVector &test point)
  
  Same as the other function of the same name, only this allows the user to specify the location of the candidate.

- Real calc score topo avg persistence(int respFnCount, RealVector &test point)
  
  Same as the other function of the same name, only this allows the user to specify the location of the candidate.

- Real calc score topo alm hybrid(int respFnCount, RealVector &test point)
  
  Same as the other function of the same name, only this allows the user to specify the location of the candidate.

- Real compute rmspe()
  
  Using the validationSet, compute the RMSE over the surface.

- void compare complices(int dim, std::ostream &output)
Using the validationSet, compute the approximate Morse-Smale complices of the true model over the validationSet as well as the surrogate model over the validationSet, and output some topological comparisons.

- **void parse_options ()**
  Parse misc options specified in a user input deck.

- **RealVectorArray drawNewX (int this.k, int respFnCount=0)**
  Function to pick the next X value to be evaluated by the Iterated model.

- **void output_round_data (int round, int respFnCount=0)**
  Temporary function for dumping validation data to output files to be visualized in TopoAS.

- **void update_amsc (int respFnCount=0)**
  Update the approximate Morse-Smale complex based on the training points and selected candidates. Uses surrogate function responses.

- **void construct_fsu_sampler (Iterator &u_space_sampler, Model &u_model, int num_samples, int seed, unsigned short sample_type)**
  Copy of construct_lhs only it allows for the construction of FSU sample designs. This can break the fsu_cvt, so it is not used at the moment, and these designs only affect the initial sample build not the candidate sets constructed at each round.

- **void output_for_optimization (int dim)**
  This function will write an input deck for a multi-start global optimization run of DAKOTA by extracting all of the local minima off the approximate Morse-Smale complex created from the validation set of the surrogate model.

- **Real median (const RealVector &sorted_data)**
  Compute the median of the sorted values passed in.

- **void pick_new_candidates ()**
  Pick new candidates from Emulator.

- **void score_new_candidates ()**
  Score new candidates based on the chosen metrics.

**Private Attributes**

- **Iterator gpBuild**
  LHS iterator for building the initial GP.

- **Iterator gpEval**
  LHS iterator for sampling on the GP.

- **Iterator gpFinalEval**
  LHS iterator for sampling on the final GP.

- **Model gpModel**
  GP model of response, one approximation per response function.

- **int numRounds**
  The number of rounds of additions of size batchSize to add to the original set of LHS samples.

- **int numPtsTotal**
  The total number of points.

- **int numEmulEval**
  The number of points evaluated by the GP each iteration.

- **int numFinalEmulEval**
  Number of points evaluated on the final GP.

- **int scoringMethod**
the type of scoring metric to use for sampling

- **Real** `finalProb`
  the final calculated probability ($p$)

- **RealVectorArray** `gpCvars`
  Vector to hold the current values of the current sample inputs on the GP.

- **RealVectorArray** `gpMeans`
  Vector to hold the current values of the current mean estimates for the sample values on the GP.

- **RealVectorArray** `gpVar`
  Vector to hold the current values of the current variance estimates for the sample values on the GP.

- **RealVector** `emulEvalScores`
  Vector to hold the scored values for the current GP samples.

- **RealVector** `predictionErrors`
  Vector to hold the RMSE after each round of adaptively fitting the model.

- **RealVectorArray** `validationSet`
  Validation point set used to determine `predictionErrors` above.

- **RealVector** `yTrue`
  True function responses at the values corresponding to `validationSet`.

- **RealVector** `yModel`
  Surrogate function responses at the values corresponding to `validationSet`.

- **int** `validationSetSize`
  Number of points used in the `validationSet`.

- **int** `batchSize`
  Number of points to add each round, default = 1.

- **String** `batchStrategy`
  String describing the type of batch addition to use. Allowable values are naive, distance, topology.

- **String** `outputDir`
  Temporary string for dumping validation files used in TopoAS visualization.

- **String** `scoringMetric`
  String describing the method for scoring candidate points. Options are: alm, distance, gradient, highest_persistence, avg_persistence, bottleneck, alm_topo_hybrid. Note: `alm` and `alm_topo_hybrid` will fail when used with surrogates other than `global_kriging` as it is based on the variance of the surrogate. At the time of implementation, `global_kriging` is the only surrogate capable of yielding this information.

- **unsigned short** `sampleDesign`
  Enum describing the initial sample design. Options are: RANDOM_SAMPLING, FSU_CVT, FSU_HALTON, FSU_HAMMERSLEY

- **String** `approx_type`
  String describing type of surrogate is used to fit the data. Options are: global_kriging, global_mars, global_neural_network, global_polynomial, global_moving_least_squares, global_radial_basis.

- **MS_Complex** `AMSC`
  The approximate Morse-Smale complex data structure.

- **int** `numKneighbors`
  The number of approximate nearest neighbors to use in computing the AMSC.

- **bool** `outputValidationData`
  Temporary variable for toggling writing of data files to be used by TopoAS.
14.126. NONDADAPTIVESAMPLING CLASS REFERENCE

Additional Inherited Members

14.126.1 Detailed Description

Class for testing various Adaptively sampling methods using geometric, statistical, and topological information of the surrogate.

_NonDAdaptiveSampling_ implements an adaptive sampling method based on the work presented in Adaptive Sampling with Topological Scores by Dan Maljovec, Bei Wang, Ana Kupresanin, Gardar Johannesson, Valerio Pascucci, and Peer-Timo Bremer presented in IJUQ (insert issue). The method computes scores based on the topology of the known data and the topology of the surrogate model. A number of alternate adaption strategies are offered as well.

14.126.2 Constructor & Destructor Documentation

NonDAdaptiveSampling ( ProblemDescDB & problem_db, Model & model )

standard constructor

_This constructor is called for a standard letter-envelope iterator instantiation_. In this case, set_db_list_nodes has been called and probDescDB can be queried for settings from the method specification.

References Dakota::abort_handler(), Response::active_set(), NonDAdaptiveSampling::AMSC, NonDAdaptiveSampling::approx_type, Iterator::assign_rep(), Model::assign_rep(), NonDAdaptiveSampling::batchSize, NonDAdaptiveSampling::batchStrategy, NonDAdaptiveSampling::construct_fsu_sampler(), NonD::construct_lhs(), Model::current_response(), ProblemDescDB::get_bool(), ProblemDescDB::get_int(), ProblemDescDB::get-svg(), ProblemDescDB::get_sa(), ProblemDescDB::get_string(), ProblemDescDB::get_ushort(), NonDAdaptiveSampling::gpBuild, NonDAdaptiveSampling::gpEval, NonDAdaptiveSampling::gpFinalEval, NonDAdaptiveSampling::gpModel, Model::gradient_type(), Model::hessian_type(), Iterator::iteratedModel, Iterator::maxIterations, NonDAdaptiveSampling::numEmulEval, NonDAdaptiveSampling::numFinalEmulEval, NonDAdaptiveSampling::numKneighbors, NonDAdaptiveSampling::numSamples, NonDAdaptiveSampling::outputDir, Iterator::outputLevel, NonDAdaptiveSampling::outputValidationData, NonDAdaptiveSampling::parse_options(), Iterator::probDescDB, NonDSampling::randomSeed, ActiveSet::request_values(), NonDSampling::rngName, NonDAdaptiveSampling::sampleDesign, NonDSampling::sampleType, NonDAdaptiveSampling::scoringMetric, NonDSampling::vary_pattern(), and NonDSampling::varyPattern.

~NonDAdaptiveSampling ( )

alternate constructor for sample generation and evaluation "on the fly" has not been implemented

destructor

14.126.3 Member Function Documentation

void core_run ( ) [protected], [virtual]

core portion of run; implemented by all derived classes and may include pre/post steps in lieu of separate pre/post

_Virtual run function for the iterator class hierarchy. All derived classes need to redefine it._

Reimplemented from _Iterator_.

References Iterator::all_responses(), Model::append_approximation(), Model::approximation_data(), NonDAdaptiveSampling::batchSize, Model::build_approximation(), NonDAdaptiveSampling::compare_complices(), NonDAdaptiveSampling::compute_rmspe(), NonD::computedProbLevels, Model::continuous_variables(), Model::current_response(), Model::current_variables(), NonDAdaptiveSampling::drawNewX(), Model::evaluate(), Model::evaluation_id(), NonDAdaptiveSampling::gpCvars, NonDAdaptiveSampling::gpFinalEval, NonDAdaptiveSampling::gpMeans,
void print_results ( std::ostream & s, short results_state = FINAL_RESULTS ) [protected], [virtual]

print the final iterator results
This virtual function provides additional iterator-specific final results outputs beyond the function evaluation
summary printed in finalize_run().
Reimplemented from Analyzer.
References NonD::print_level_mappings(), and NonDSampling::statsFlag.
The documentation for this class was generated from the following files:

- NonDAdaptiveSampling.hpp
- NonDAdaptiveSampling.cpp

14.127 NonDBayesCalibration Class Reference

Base class for Bayesian inference: generates posterior distribution on model parameters given experimental data. Inheritance diagram for NonDBayesCalibration:

```
  NonDDREAMBayesCalibration
  NonDMUQBayesCalibration
  NonDQUESOBayesCalibration
  NonDWASABIBayesCalibration
  NonDGPMSABayesCalibration

  NonDBayesCalibration
  NonDCalibration
  NonD
  Analyzer
  Iterator
```

Public Member Functions

- **NonDBayesCalibration (ProblemDescDB & problem_db, Model & model)**
  `standard constructor`
- **~NonDBayesCalibration ()**
  `destructor`
- **template<typename VectorType>**
  `Real prior_density (const VectorType & vec)`
  `compute the prior PDF for a particular MCMC sample`
14.127. nondbayesCalibration Class Reference

- template<typename VectorType >
  Real log_prior_density (const VectorType &vec)
  compute the log prior PDF for a particular MCMC sample
- template<typename Engine >
  void prior_sample (Engine &rng, RealVector &prior_samples)
  draw a multivariate sample from the prior distribution
- template<typename VectorType >
  void prior_mean (VectorType &mean_vec) const
  return the mean of the prior distribution
- template<typename MatrixType >
  void prior_variance (MatrixType &var_mat) const
  return the covariance of the prior distribution
- template<>
  Real prior_density (const RealVector &vec)
- template<>
  Real log_prior_density (const RealVector &vec)

Static Public Member Functions

- static void get_positive_definite_covariance_from_hessian (const RealSymMatrix &hessian, const RealMatrix &prior_chol_fact, RealSymMatrix &covariance, short output_lev)
  Compute the proposal covariance C based on low-rank approximation to the prior-preconditioned misfit Hessian.
- static Real knn_kl_div (RealMatrix &distX_samples, RealMatrix &distY_samples, size_t dim)
- static Real knn_mutual_info (RealMatrix &Xmatrix, int dimX, int dimY, unsigned short alg)

Protected Member Functions

- void pre_run ()
  pre-run portion of run (optional); re-implemented by Iterators which can generate all Variables (parameter sets) a priori
- void core_run ()
  core portion of run; implemented by all derived classes and may include pre/post steps in lieu of separate pre/post
- void derived_init_communicators (ParLevLIter pliter)
  derived class contributions to initializing the communicators associated with this Iterator instance
- void derived_set_communicators (ParLevLIter pl_iter)
  derived class contributions to setting the communicators associated with this Iterator instance
- void derived_free_communicators (ParLevLIter pl_iter)
  derived class contributions to freeing the communicators associated with this Iterator instance
- void print_results (std::ostream &s, short results_state=FINAL_RESULTS)
  print the final iterator results
- const Model & algorithm_space_model () const
- virtual void calibrate ()=0
  Perform Bayesian calibration (all derived classes must implement)
- void construct_mcmc_model ()
  construct mcmcModel (no emulation, GP, PCE, or SC) that wraps inbound Model
• **void init\_hyper\_parameters ()**
  
  *initialize the hyper-parameter priors*

• **void init\_map\_optimizer ()**
  
  *initialize the MAP optimizer selection*

• **void construct\_map\_model ()**
  
  *construct the negative log posterior RecastModel (negLogPostModel)*

• **void construct\_map\_optimizer ()**
  
  *construct the MAP optimizer for minimization of negLogPostModel*

• **void initialize\_model ()**
  
  *initialize emulator model and probability space transformations*

• **void calibrate\_to\_hifi ()**
  
  *calibrate the model to a high-fidelity data source, using mutual information-guided design of experiments (adaptive experimental design)*

• **bool eval\_hi2lo\_stop (bool stop\_metric, double prev\_MI, double max\_MI, int num\_it, int num\_hifi, int max\_hifi, int num\_candidates)**
  
  *evaluate stopping criteria for calibrate\_to\_hifi*

• **void print\_hi2lo\_file (std::ostream &out\_file, int num\_it, int batch\_Evals, RealMatrix &optimal\_config\_matrix, const RealVector &MI\_vec, int max\_hifi, RealMatrix &resp\_matrix, const RealVector &optimal\_config, double max\_MI)**
  
  *print calibrate\_to\_hifi progress to file*

• **void print\_hi2lo\_begin (int num\_it)**
  
  *print calibrate\_to\_hifi progress*

• **void print\_hi2lo\_status (int num\_it, int i, const RealVector &xi\_i, double MI)**
  
  *print selected points*

• **void add\_lhs\_hifi\_data ()**
  
  *supplement high-fidelity data with LHS samples*

• **void apply\_error\_vec (const RealVector &error\_vec, int &seed, int experiment)**
  
  *apply simulation error vector*

• **void build\_error\_matrix (const RealVector &sim\_error\_vec, RealMatrix &sim\_error\_matrix, int &seed)**
  
  *build matrix of errors*

• **void build\_designs (RealMatrix &design\_matrix)**
  
  *build matrix of candidate points*

• **void build\_hi2lo\_xmatrix (RealMatrix &Xmatrix, int i, const RealMatrix &mi\_chain, RealMatrix &sim\_\_error\_matrix)**
  
  *build matrix to calculate mutual information for calibrate\_to\_hifi*

• **void run\_hifi (RealMatrix &optimal\_config\_matrix, RealMatrix &resp\_matrix)**
  
  *run high-fidelity model at several configs and add to experiment data*

• **void build\_model\_discrepancy ()**
  
  *calculate model discrepancy with respect to experimental data*

• **void build\_scalar\_discrepancy ()**

• **void build\_field\_discrepancy ()**
void build_GP_field (const RealMatrix &t, RealMatrix &t_pred, const RealVector &concat_disc, RealVector &disc_pred, RealVector &disc_var)

void calculate_kde ()
    calculate a Kernel Density Estimate (KDE) for the posterior samples

void calculate_evidence ()
    calculate the model evidence

void extract_selected_posterior_samples (const std::vector<int> &points_to_keep, const RealMatrix &samples_for_posterior_eval, const RealVector &posterior_density, RealMatrix &posterior_data) const

void export_posterior_samples_to_file (const std::string filename, const RealMatrix &posterior_data) const

template<typename VectorType1 , typename VectorType2 >
void augment_gradient_with_log_prior (VectorType1 &log_grad, const VectorType2 &vec)
    compute the (approximate) gradient of the negative log posterior by augmenting the (approximate) gradient of the negative log likelihood with the gradient of the negative log prior

template<typename MatrixType , typename VectorType >
void augment_hessian_with_log_prior (MatrixType &log_hess, const VectorType &vec)
    compute the (approximate) Hessian of the negative log posterior by augmenting the (approximate) Hessian of the negative log likelihood with the Hessian of the negative log prior

Real log_likelihood (const RealVector &residuals, const RealVector &hyper_params)
    calculate log-likelihood from the passed residuals (assuming they are already sized and scaled by covariance / hyperparams...)

void prior_cholesky_factorization ()
    compute priorCovCholFactor based on prior distributions for random variables and any hyperparameters

void get_positive_definite_covariance_from_hessian (const RealSymMatrix &hessian, RealSymMatrix &covariance)
    member version forwards member data to static function

void scale_model ()
    Wrap iteratedModel in a RecastModel that performs response scaling.

void weight_model ()
    Wrap iteratedModel in a RecastModel that weights the residuals.

void export_discrepancy (RealMatrix &pred_config_mat)
    print tabular files containing model+discrepancy responses and variances

void export_field_discrepancy (RealMatrix &pred_vars_mat)
    print tabular files containing model+discrepancy responses and variances for field responses

void compute_statistics ()
    Compute final stats for MCMC chains.

void export_chain ()
    export the acceptance chain in user space

void export_chain (RealMatrix &filtered_chain, RealMatrix &filtered_fn_vals)
    Print filtered posterior and function values (later: credibility and prediction intervals)

void filter_chain (const RealMatrix &acceptance_chain, RealMatrix &filtered_chain, int target_length)
    Perform chain filtering based on target chain length.

void filter_chain (const RealMatrix &acceptance_chain, RealMatrix &filtered_chain)
    Perform chain filtering with burn-in and sub-sampling.

void filter_fnvals (const RealMatrix &accepted_fn_vals, RealMatrix &filtered_fn_vals)
• void filter_matrix_cols (const RealMatrix &orig_matrix, int start_index, int stride, RealMatrix &filtered_matrix)
  return a newly allocated filtered matrix including start_index and every stride-th index after; for burn-in cases, start_index is the number of burn-in discards

• void compute_intervals ()
• void compute_prediction_vals (RealMatrix &filtered_fn_vals, RealMatrix &PredVals, int num_filtered, size_t num_exp, size_t num_concatenated)

• void print_intervals_file (std::ostream &stream, RealMatrix &functionvalsT, RealMatrix &predvalsT, int length, size_t aug_length)
• void print_intervals_screen (std::ostream &stream, RealMatrix &functionvalsT, RealMatrix &predvalsT, int length)

• void kl_post_prior (RealMatrix &acceptanceChain)
  Compute information metrics.
• void prior_sample_matrix (RealMatrix &prior_dist_samples)
• void mutual_info_buildX ()
• void print_kl (std::ostream &stream)
• void print_chain_diagnostics (std::ostream &s)
• void print_batch_means_intervals (std::ostream &s)

Static Protected Member Functions
• static void neg_log_post_resp_mapping (const Variables &model_vars, const Variables &nlpost_vars, const Response &model_resp, Response &nlpost_resp)
  static function passed by pointer to negLogPostModel recast model

• static void ann_dist (const ANNpointArray matrix1, const ANNpointArray matrix2, RealVector &distances, int NX, int NY, int dim2, IntVector &k, double eps)

Protected Attributes
• String scalarDataFilename
• short emulatorType
  the emulator type: NO_EMULATOR, GP_EMULATOR, PCE_EMULATOR, SC_EMULATOR, ML_PCE_EMULATOR, MF_PCE_EMULATOR, or MF_SC_EMULATOR
• Model mcmcModel
  Model instance employed in the likelihood function; provides response function values from Gaussian processes, stochastic expansions (PCE/SC), or direct access to simulations (no surrogate option)
• bool mcmcModelHasSurrogate
  whether the MCMC Model is a surrogate, or a thin transformation around a surrogate, so can be cheaply re-evaluated in chain recovery
• Model residualModel
  DataTransformModel wrapping the mcmcModel.
• Iterator mapOptimizer
  SQP or NIP optimizer for pre-solving for the MAP point prior to MCMC. This is restricted to emulator cases for now, but as for derivative preconditioning, could be activated for no-emulator cases with a specification option (not active by default).
- Model `negLogPostModel`:
  - *RecastModel* for solving for MAP: reduces `residualModel` to scalar negative log posterior.
- unsigned short `mapOptAlgOverride`:
  - user setting of the MAP optimization algorithm type
- Iterator `stochExpIterator`:
  - *NonDPolynomialChaos* or *NonDStochCollocation* instance for defining a PCE/SC-based `mcmcModel`.
- int `chainSamples`:
  - number of samples in the chain (e.g. number of MCMC samples); for iterative update cycles, number of samples per update cycle
- int `randomSeed`:
  - random seed for MCMC process
- short `mcmcDerivOrder`:
  - order of derivatives used in MCMC process (bitwise like ASV)
- RealVector `mapSoln`:
  - solution for most recent MAP pre-solve; also serves as initial guess for initializing the first solve and warm-starting the next solve (posterior emulator refinement)
- bool `adaptExpDesign`:
  - whether to perform iterative design of experiments with high-fidelity model
- size_t `numCandidates`:
  - number of candidate designs for adaptive Bayesian experimental design
- String `importCandPtsFile`:
  - whether to import candidate design points for adaptive Bayesian experimental design
- unsigned short `importCandFormat`:
  - tabular format for the candidate design points import file
- int `maxHifiEvals`:
  - maximum number of high-fidelity model runs to be used for adaptive Bayesian experimental design
- int `batchEvals`:
  - number of optimal designs selected per iteration of experimental design algorithm
- unsigned short `mutualInfoAlg`:
  - algorithm to employ in calculating mutual information
- bool `readFieldCoords`:
  - need field coordinates for model discrepancy
- bool `calModelDiscrepancy`:
  - flag whether to calculate model discrepancy
- String `discrepancyType`:
  - set discrepancy type
- String `exportCorrModelFile`:
  - filename for corrected model (model+discrepancy) calculations output
- String `exportDiscrepFile`:
  - filename for discrepancy calculations output
- String `exportCorrVarFile`:
  - filename for corrected model variance calculations
- unsigned short `exportCorrModelFormat`
format options for corrected model output

• unsigned short exportDiscrepFormat
  format options for discrepancy output

• unsigned short exportCorrVarFormat
  format options for corrected model variance output

• short approxCorrectionOrder
  specify polynomial or trend order for discrepancy correction

• size_t numPredConfigs
  number of prediction configurations at which to calculate model discrepancy

• RealVector configLowerBnds
  lower bounds on configuration domain

• RealVector configUpperBnds
  upper bounds on configuration domain

• ResponseArray discrepancyResponses
  array containing predicted of model+discrepancy

• ResponseArray correctedResponses
  array containing predicted of model+discrepancy

• RealMatrix correctedVariances
  matrix containing variances of model+discrepancy

• RealVector predictionConfigList
  list of prediction configurations at which to calculate model discrepancy

• String importPredConfigs
  whether to import prediction configurations at which to calculate model discrepancy

• unsigned short importPredConfigFormat
  tabular format for prediction configurations import file

• RealVector discrepancyFieldResponses
  array containing predicted of model+discrepancy

• RealVector correctedFieldResponses
  array containing predicted of model+discrepancy

• RealVector correctedFieldVariances
  matrix containing variances of model+discrepancy

• Model hifiModel
  a high-fidelity model data source (given by pointer in input)

• int initHifiSamples
  initial high-fidelity model samples

• Iterator hifiSampler
  LHS iterator to generate hi-fi model data.

• RealMatrix priorCovCholFactor
  the Cholesky factor of the prior covariance

• unsigned short obsErrorMultiplierMode
  mode for number of observation error multipliers to calibrate (default none)

• int numHyperparams
calculated number of hyperparameters augmenting the calibration parameter set, e.g., due to calibrate observation error multipliers

- RealVector `invGammaAlphas`
  - alphas for inverse gamma distribution on hyper-params
- RealVector `invGammaBetas`
  - alphas for inverse gamma distribution on hyper-params
- std::vector `< Pecos::RandomVariable > invGammaDists`
  - distributions for hyper-params
- bool `standardizedSpace`
  - flag indicating use of a variable transformation to standardized probability space for the model or emulator
- bool `posteriorStatsKL`
  - flag indicating the calculation of KL divergence between prior and posterior
- bool `posteriorStatsMutual`
  - flag indicating the calculation of mutual information between prior and posterior
- bool `posteriorStatsKDE`
  - flag indicating the calculation of the kernel density estimate of the posteriors
- bool `chainDiagnostics`
  - flag indicating calculation of chain diagnostics
- bool `chainDiagnosticsCI`
  - flag indicating calculation of confidence intervals as a chain diagnostic
- bool `calModelEvidence`
  - flag indicating calculation of the evidence of the model
- bool `calModelEvidMC`
  - flag indicating use of Monte Carlo approximation to calculate evidence
- bool `calModelEvidLaplace`
  - flag indicating use of Laplace approximation to calculate evidence
- int `evidenceSamples`
  - number of samples to be used in model evidence calculation
- bool `adaptPosteriorRefine`
  - flag indicating usage of adaptive posterior refinement; currently makes sense for unstructured grids in GP and PCE least squares/CS
- String `proposalCovarType`
  - approach for defining proposal covariance
- RealVector `proposalCovarData`
  - data from user input of proposal covariance
- String `proposalCovarFilename`
  - filename for user-specified proposal covariance
- String `proposalCovarInputType`
  - approach for defining proposal covariance
- RealMatrix `acceptanceChain`
  - Post-processing-related controls.
- RealMatrix `acceptedFnVals`
cached function values corresponding to acceptanceChain for final statistics reporting

- std::map<Real, RealVector> bestSamples
  container for managing best MCMC samples (points and associated log posterior) collected across multiple (restarted) chains

- int burnInSamples
  number of MCMC samples to discard from acceptance chain

- int subSamplingPeriod
  period or skip in post-processing the acceptance chain

- RealMatrix chainStats
- RealMatrix fnStats
- RealMatrix predVals
  Compute credibility and prediction intervals of final chain.

- RealMatrix filteredFnVals
  cached filtered function values for printing (may be a view of acceptedFnVals)

- String exportMCMCFilename
  output filename for the MCMC chain

- short exportMCMCFormat
  output formatting options for MCMC export

- short filteredMCMCFormat
- Real kl_est
- bool scaleFlag
  whether response scaling is active

- bool weightFlag
  whether weight scaling is active

Static Protected Attributes

- static NonDBayesCalibration * nonDBayesInstance
  Pointer to current class instance for use in static callback functions.

14.127.1 Detailed Description

Base class for Bayesian inference: generates posterior distribution on model parameters given experimental data.

This class will eventually provide a general-purpose framework for Bayesian inference. In the short term, it only collects shared code between QUESO and GPMSA implementations.

14.127.2 Constructor & Destructor Documentation

NonDBayesCalibration ( ProblemDescDB & problem_db, Model & model )

standard constructor

This constructor is called for a standard letter-envelope iterator instantiation. In this case, set_db_list_nodes has been called and probDescDB can be queried for settings from the method specification.

References Dakota::abort_handler(), NonDBayesCalibration::adaptExpDesign, Iterator::assign_rep(), Model::assign_rep(), NonDBayesCalibration::burnInSamples, NonDCalibration::calibrationData, NonDBayesCalibration::chainSamples, NonDBayesCalibration::construct_map_model(), NonDBayesCalibration::construct_mcmc_model(),
Model::continuous::lower_bound(), Model::continuous::upper_bound(), Model::continuous::variables(), Dakota::copy_data::partial(), Model::cv(), NonDBayesCalibration::emulatorType, NonDCalibration::expData, Dakota::generate_system_seed(), ProblemDescDB::get_bool(), NonDBayesCalibration::hifiModel, NonDBayesCalibration::hifiSampler, NonDBayesCalibration::init_hyper_parameters(), NonDBayesCalibration::init_map_optimizer(), NonDBayesCalibration::initHifiSamples, NonDBayesCalibration::invGammaDists, Iterator::iteratedModel, NonDBayesCalibration::map::Soln, Iterator::maxEvalConcurrency, NonDBayesCalibration::mcmcDerivOrder, NonDBayesCalibration::mcmcModel, Model::model_type(), ExperimentData::num_experiments(), Analyzer::numContinuousVars, NonDBayesCalibration::numHyperparams, NonDBayesCalibration::obsErrorMultiplierMode, Iterator::probDescDB, NonDBayesCalibration::randomSeed, NonDBayesCalibration::residualModel, NonDBayesCalibration::scale_model(), NonDBayesCalibration::scaleFlag, NonDBayesCalibration::standardizedSpace, NonDBayesCalibration::subSamplingPeriod, Model::truth_model(), Analyzer::vary_pattern(), NonDBayesCalibration::weight_model(), and NonDBayesCalibration::weightFlag.

14.127.3 Member Function Documentation

void prior_mean ( VectorType & mean_vec ) const

return the mean of the prior distribution
Assume the target mean_vec is sized by client
References NonDBayesCalibration::invGammaDists, Iterator::iteratedModel, NonDBayesCalibration::mcmcModel, Model::multivariate_distribution(), Analyzer::numContinuousVars, NonDBayesCalibration::numHyperparams, and NonDBayesCalibration::standardizedSpace.

void prior_variance ( MatrixType & var_mat ) const

return the covariance of the prior distribution
Assumes the target var_mat is sized by client
References NonDBayesCalibration::invGammaDists, Iterator::iteratedModel, NonDBayesCalibration::mcmcModel, Model::multivariate_distribution(), Analyzer::numContinuousVars, NonDBayesCalibration::numHyperparams, and NonDBayesCalibration::standardizedSpace.

void pre_run ( ) [protected], [virtual]

pre-run portion of run (optional); re-implemented by Iterators which can generate all Variables (parameter sets) a priori
pre-run phase, which a derived iterator may optionally reimplement; when not present, pre-run is likely integrated into the derived run function. This is a virtual function; when re-implementing, a derived class must call its nearest parent’s pre_run(), if implemented, typically before performing its own implementation steps.
Reimplemented from Analyzer.
References NonDBayesCalibration::construct_map_optimizer(), Model::is_null(), NonDBayesCalibration::negLogPostModel, Analyzer::pre_run(), NonDBayesCalibration::residualModel, and Model::update_from_subordinate_model().

void core_run ( ) [protected], [virtual]

core portion of run; implemented by all derived classes and may include pre/post steps in lieu of separate pre/post Virtual run function for the iterator class hierarchy. All derived classes need to redefine it.
Reimplemented from Iterator.
References NonDBayesCalibration::adaptExpDesign, NonDBayesCalibration::build_model_discrepancy(), NonDBayesCalibration::calibrate(), NonDBayesCalibration::calibrate_to_hifi(), NonDBayesCalibration::calModelDiscrepancy, and NonDBayesCalibration::nonDBayesInstance.
void print_results ( std::ostream & s, short results_state = FINAL_RESULTS ) [protected], [virtual]

print the final iterator results

This virtual function provides additional iterator-specific final results outputs beyond the function evaluation summary printed in finalize_run().

Reimplemented from Analyzer.

Reimplemented in NonDGPMSEBayesCalibration, NonDQUESTOBayesCalibration, NonDWSABIBayesCalibration, and NonDMUQBayesCalibration.

References NonDBayesCalibration::chainDiagnostics, Model::continuous_variable_labels(), Dakota::copy_data(), NonDBayesCalibration::filtered_fnVals, Response::function_labels(), Dakota::length(), NonDBayesCalibration::mcmcModel, Iterator::outputLevel, NonDBayesCalibration::posteriorStatsKL, NonDSampling::print_moments(), NonD::requestedProbLevels, and NonDBayesCalibration::residualModel.

Referenced by NonDMUQBayesCalibration::print_results(), NonDQUESTOBayesCalibration::print_results(), NonDGPMSABayesCalibration::print_results(), and NonDGPMSEBayesCalibration::print_results().

const Model & algorithm_space_model ( ) const [inline], [protected], [virtual]

default definition that gets redefined in selected derived Minimizers

Reimplemented from Analyzer.

References NonDBayesCalibration::mcmcModel.

void init_map_optimizer ( ) [protected]

initialize the MAP optimizer selection

Construct optimizer for MAP pre-solve Emulator: on by default; can be overridden with "pre_solve none" No emulator: off by default; can be activated with "pre_solve {sqp,nip}" relies on mapOptimizer ctor to enforce min derivative support Calculation of model evidence using Laplace approximation: this requires a MAP solve.

References Dakota::abort_handler(), NonDBayesCalibration::calModelEvidLaplace, NonDBayesCalibration::emulatorType, and NonDBayesCalibration::mapOptAlgOverride.

Referenced by NonDBayesCalibration::NonDBayesCalibration().

Real log_likelihood ( const RealVector & residuals, const RealVector & all_params ) [protected]

calculate log-likelihood from the passed residuals (assuming they are already sized and scaled by covariance / hyperparams...)

Calculate the log-likelihood, accounting for contributions from covariance and hyperparameters, as well as constant term:

\[
\log(L) = \frac{-1}{2}N_r \log(2\pi) - \frac{1}{2} \log(\det(Cov)) - \frac{1}{2} r'(Cov^{-1}) r
\]

The passed residuals must already be size-adjusted, differenced with any data, if present, and scaled by covariance^{-1/2}.

References NonDCalibration::expData, Dakota::HALF_LOG_2PI, ExperimentData::half_log_cov_determinant(), Analyzer::numContinuousVars, NonDBayesCalibration::numHyperparams, and NonDBayesCalibration::obsErrorMultiplierMode.

Referenced by NonDBayesCalibration::calculate_evidence(), NonDQUESTOBayesCalibration::dakotaLogLikelihood(), NonDBayesCalibration::neg_log_post_res_mapping(), and NonDDREAMBayesCalibration::sample_likelihood().

void neg_log_post_res_mapping ( const Variables & residual_vars, const Variables & nlpost_vars, const Response & residual_resp, Response & nlpost_resp ) [static], [protected]

static function passed by pointer to negLogPostModel recast model
Response mapping callback used within RecastModel for MAP pre-solve. Computes
\[
-\log(\text{post}) = -\log(\text{like}) - \log(\text{prior}); \quad \text{where} \quad -\log(\text{like}) = 1/2*Nr*\log(2\pi) + 1/2*\log(\det(\text{Cov})) + 1/2*r'(\text{Cov}^{-1})^T r = 1/2*Nr*\log(2\pi) + 1/2*\log(\det(\text{Cov})) + \text{misfit}
\]
(misfit defined as 1/2 r^T (\text{mult}^2\Gamma_{\text{d}})^{-1} r) The passed residual \( \text{resp} \) has been differenced, interpolated, and covariance-scaled

References Response::active_set_request_vector(), NonDBayesCalibration::augment_gradient_with_log_prior(), NonDBayesCalibration::augment_hessian_with_log_prior(), ExperimentData::build_gradient_of_sum_square_residuals(), ExperimentData::build_hessian_of_sum_square_residuals(), Variables::continuous_variables(), NonDCalibration::expData, Response::function_gradient_view(), Response::function_hessian_view(), Response::function_value(), Response::function_values(), ExperimentData::half_log_cov_det_gradient(), ExperimentData::half_log_cov_det_hessian(), NonDBayesCalibration::log_likelihood(), NonDBayesCalibration::log_prior_density(), NonDBayesCalibration::nonDBayesInstance, Analyzer::numContinuousVars, NonDBayesCalibration::numHyperparams, NonDBayesCalibration::obsErrorMultiplierMode, and Iterator::outputLevel.

Referenced by NonDBayesCalibration::calculate_evidence(), and NonDBayesCalibration::construct_map_model().

```cpp
void scale_model ( ) [protected]
```

Wrap iteratedModel in a RecastModel that performs response scaling.

Wrap the residualModel in a scaling transformation, such that residualModel now contains a scaling recast model.

References Model::assign_rep(), Iterator::outputLevel, and NonDBayesCalibration::residualModel.

Referenced by NonDBayesCalibration::NonDBayesCalibration().

```cpp
void weight_model ( ) [protected]
```

Wrap iteratedModel in a RecastModel that weights the residuals.

Setup Recast for weighting model. The weighting transformation doesn’t resize, and makes no vars, active set or secondary mapping. All indices are one-to-one mapped (no change in counts).

References Dakota::abort_handler(), Model::assign_rep(), Iterator::outputLevel, Model::primary_response_fn_weights(), and NonDBayesCalibration::residualModel.

Referenced by NonDBayesCalibration::NonDBayesCalibration().

```cpp
void export_chain ( RealMatrix & filtered_chain, RealMatrix & filtered_fn_vals ) [protected]
```

Print filtered posterior and function values (later: credibility and prediction intervals)

Print tabular file with filtered chain, function values, and pred values

References Variables::continuous_variables(), Variables::copy(), Model::current_response(), Model::current_variables(), NonDBayesCalibration::exportMCMCFilename, NonDBayesCalibration::exportMCMCFormat, Response::function_labels(), Model::interface_id(), NonDBayesCalibration::mcmcModel, Analyzer::numFunctions, NonDBayesCalibration::residualModel, Dakota::write_precision, and Variables::write_tabular().

14.127.4 Member Data Documentation

RealMatrix acceptanceChain [protected]

Post-processing-related controls.

accumulation of acceptance chain across restarts (stored in user-space) TO DO: retire once restarts are retired; optimize to convert to user-space only in final results
Referenced by NonDDREAMBayesCalibration::archive_acceptance_chain(), NonDGPMASBayesCalibration::cache_acceptance_chain(), NonDMUQBayesCalibration::cache_chain(), NonDDREAMBayesCalibration::cache_chain(), NonDQUESTOBayesCalibration::cache_chain(), NonDBayesCalibration::calibrate_to_hifi(), and NonDBayesCalibration::compute_statistics().

The documentation for this class was generated from the following files:

- NonDBayesCalibration.hpp
- NonDBayesCalibration.cpp

### 14.128  NonDC3FunctionTrain Class Reference

Nonintrusive uncertainty quantification with the C3 library ...

Inheritance diagram for NonDC3FunctionTrain:

```
  NonDC3FunctionTrain
  |       |       |
  |       |       |
  |       |       |
  |       |       |
  |       |       |
  |       |       |
  Iterator
  |
  Analyzer
  |
  NonD
  |
  NonDExpansion
  |
  NonDC3FunctionTrain
  |
  NonDMultilevelFunctionTrain
```

### Public Member Functions

- **NonDC3FunctionTrain (ProblemDescDB &problem_db, Model &model)**  
  *standard constructor*

- **~NonDC3FunctionTrain ()**  
  *destructor*

### Protected Member Functions

- **NonDC3FunctionTrain (unsigned short method_name, ProblemDescDB &problem_db, Model &model)**  
  *base constructor for DB construction of multilevel/multifidelity PCE (method_name is not necessary, rather it is just a convenient overload allowing the derived ML FT class to bypass the standard FT ctor)*

- **void resolve_inputs (short &u_space_type, short &data_order)**  
  *perform error checks and mode overrides*

- **void initialize_u_space_model ()**  
  *initialize uSpaceModel polynomial approximations with PCE/SC data*

- **size_t collocation_points () const**
return specification for number of collocation points (may be part of a sequence specification)

- **void push_increment()**
  
  helper function to manage different push increment cases

- **void update_samples_from_order_increment()**
  
  update numSamplesOnModel after an order increment

- **void sample_allocation_metric(Real &regress_metric, Real power)**

- **void print_moments(std::ostream &s)**
  
  override certain print functions

- **void print_sobol_indices(std::ostream &s)**
  
  print global sensitivity indices

- **bool config_regression(size_t colloc pts, size_t regress size, int seed, Iterator &u_space_sampler, Model &g_u_model)**
  
  configure u_space_sampler and approx_type based on regression specification

- **void push_C3_core_rank(size_t start_rank)**
  
  Publish configuration data for initial function train cores, prior to any adaptation.

- **void push_C3_core_orders(const UShortArray &start_orders)**
  
  Publish configuration data for initial function train cores, prior to any adaptation.

- **void push_C3_db_options()**
  
  Publish options from C3 input specification (not needed if model-driven specification: already extracted by iterated-Model)

### Protected Attributes

- **String importBuildPointsFile**
  
  user-specified file for importing build points

- **size_t startRankSpec**
  
  scalar specification for initial rank (prior to adapt_rank)

- **unsigned short startOrderSpec**
  
  scalar specification for initial basis order (prior to uniform refinement)

### Private Attributes

- **size_t collocPtsSpec**
  
  user specification for collocation points

### Additional Inherited Members

**14.128.1 Detailed Description**

Nonintrusive uncertainty quantification with the C3 library ...

The NonDC3FunctionTrain class uses ...
14.128.2 Constructor & Destructor Documentation

NonDC3FunctionTrain ( ProblemDescDB & problem_db, Model & model )

standard constructor

This constructor is called for a standard letter-envelope iterator instantiation using the ProblemDescDB.

References Dakota::abort_handler(), Response::active_set(), Model::assign_rep(), NonDC3FunctionTrain::colloc_PtsSpec, NonDC3FunctionTrain::config_regression(), NonDExpansion::configure_expansion_orders(), NonDExpansion::construct_expansion_sampler(), Model::current_response(), NonDExpansion::dimPrefSpec, ProblemDescDB::get_bool(), ProblemDescDB::get_iv(), ProblemDescDB::get_string(), ProblemDescDB::get ushort(), NonDC3FunctionTrain::importBuildPointsFile, NonDC3FunctionTrain::initialize_u_space_model(), Iterator::iteratedModel, Model::model_type(), Analyzer::numContinuousVars, Iterator::outputLevel, Iterator::probDescDB, NonDExpansion::randomSeed, SharedC3ApproxData::regression_size(), ActiveSet::request_values(), NonDC3FunctionTrain::resolve_inputs(), NonDC3FunctionTrain::startOrderSpec, NonDC3FunctionTrain::startRankSpec, Model::surrogate_type(), and NonDExpansion::uSpaceModel.

NonDC3FunctionTrain ( unsigned short method_name, ProblemDescDB & problem_db, Model & model ) [protected]

base constructor for DB construction of multilevel/multifidelity PCE (method_name is not necessary, rather it is just a convenient overload allowing the derived ML FT class to bypass the standard FT ctor)

This constructor is called by derived class constructors.

References Dakota::abort_handler(), Iterator::iteratedModel, Model::model_type(), and Model::surrogate_type().

14.128.3 Member Function Documentation

void sample_allocation_metric ( Real & regress_metric, Real power ) [protected], [virtual]

inconvenient to recompute: store previous samples rather than previous ranks

void NonDC3FunctionTrain::update_samples_from_order_decrement() { numSamplesOnModel = prevSamplesOnModel; }//requires level mgmt for persistence

Reimplemented from NonDExpansion.

References Model::approximations(), Analyzer::numFunctions, Iterator::outputLevel, and NonDExpansion::uSpaceModel.

Referenced by NonDC3FunctionTrain::update_samples_from_order_increment().

The documentation for this class was generated from the following files:

- NonDC3FunctionTrain.hpp
- NonDC3FunctionTrain.cpp

14.129 NonDCalibration Class Reference

Inheritance diagram for NonDCalibration:
Public Member Functions

- **NonDCalibration** (ProblemDescDB &problem_db, Model &model)
  
  Standard constructor
- **~NonDCalibration** ()
  
  Destructor
- **bool resize ()**
  
  Reinitializes iterator based on new variable size

Protected Attributes

- **bool calibrationData**
  
  Flag indicating whether there is calibration data present
- **ExperimentData expData**
  
  Container for experimental data to which to calibrate model.

Additional Inherited Members

14.129.1 Detailed Description

This class ...

14.129.2 Constructor & Destructor Documentation

**NonDCalibration ( ProblemDescDB &problem_db, Model &model )**

Standard constructor

This constructor is called for a standard letter-envelope iterator instantiation. In this case, set_db_list_nodes has been called and probDescDB can be queried for settings from the method specification.

References NonDCalibration::calibrationData, NonDCalibration::expData, ExperimentData::load_data(), and Iterator::outputLevel.

The documentation for this class was generated from the following files:

- NonDCalibration.hpp
- NonDCalibration.cpp
14.130 NonDCubature Class Reference

Derived nondeterministic class that generates N-dimensional numerical cubature points for evaluation of expectation integrals.

Inheritance diagram for NonDCubature:

```
NonDCubature
    NonDIntegration
        NonD
            Analyzer
                Iterator
                NonDCubature
```

Public Member Functions

- NonDCubature (Model &model, unsigned short cub_int_order)
- unsigned short integrand_order () const
  
Protected Member Functions

- NonDCubature (ProblemDescDB &problem_db, Model &model)
  constructor
- ~NonDCubature ()
  destructor
- void initialize_grid (const std::vector<Pecos::BasisPolynomial> &poly_basis)
  initialize integration grid by drawing from polynomial basis settings
- void get_parameter_sets (Model &model)
  Generate one block of numSamples samples (ndim * num_samples), populating allSamples; ParamStudy is the only class that specializes to use allVariables.
- void sampling_reset (int min_samples, bool all_data_flag, bool stats_flag)
- void increment_grid ()
  increment SSG level/TPQ order
- void increment_grid_preference (const RealVector &dim_pref)
- void increment_grid_preference ()
- void decrement_grid ()
  decrement SSG level/TPQ order
- void reset ()
  restore initial state for repeated sub-iterator executions
- int num_samples () const
Private Member Functions

- void assign_rule (const Pecos::MultivariateDistribution &mvd)
  
  define cubIntRule from random variable type

Private Attributes

- Pecos::CubatureDriver * cubDriver
  
  convenience pointer to the numIntDriver representation

- unsigned short cubIntOrderRef
  
  reference point for Pecos::CubatureDriver::cubIntOrder: the original user specification for the number of Gauss points per dimension, plus any refinements posted by increment_grid()

- unsigned short cubIntRule
  
  the isotropic cubature integration rule

Additional Inherited Members

14.130.1 Detailed Description

Derived nondeterministic class that generates N-dimensional numerical cubature points for evaluation of expectation integrals.

This class is used by NonDPolynomialChaos, but could also be used for general numerical integration of moments. It employs Stroud cubature rules and extensions by D. Xiu.

14.130.2 Constructor & Destructor Documentation

NonDCubature ( Model & model, unsigned short cub_int_order )

This alternate constructor is used for on-the-fly generation and evaluation of numerical cubature points.

References NonDCubature::assign_rule(), NonDCubature::cubDriver, NonDCubature::cubIntOrderRef, Model::multivariate_distribution(), and NonDIntegration::numIntDriver.

NonDCubature ( ProblemDescDB & problem_db, Model & model ) [protected]

canstructor

This constructor is called for a standard letter-envelope iterator instantiation. In this case, set_db_list_nodes has been called and probDescDB can be queried for settings from the method specification. It is not currently used, as there is not yet a separate nond_cubature method specification.

References NonDCubature::assign_rule(), NonDCubature::cubDriver, NonDCubature::cubIntOrderRef, NonDCubature::cubIntRule, Iterator::maxEvalConcurrency, Model::multivariate_distribution(), and NonDIntegration::numIntDriver.

14.130.3 Member Function Documentation

void sampling_reset ( int min_samples, bool all_data_flag, bool stats_flag ) [protected], [virtual]

used by DataFitSurrModel::build_global() to publish the minimum number of points needed from the cubature routine in order to build a particular global approximation.

Reimplemented from Iterator.

References NonDCubature::cubDriver.
### void increment_grid_preference (const RealVector &dim_pref) [inline], [protected], [virtual]

Should not be used, but one of virtual function pair must be defined.
- Reimplemented from NonDIntegration.
- References NonDCubature::increment_grid().

### void increment_grid_preference () [inline], [protected], [virtual]

Should not be used, but one of virtual function pair must be defined.
- Reimplemented from NonDIntegration.
- References NonDCubature::increment_grid().

### int num_samples () const [inline], [protected], [virtual]

Return current number of evaluation points. Since the calculation of samples, collocation points, etc. might be costly, provide a default implementation here that backs out from the maxEvalConcurrency.
- Reimplemented from Analyzer.
- References NonDCubature::cubDriver.

The documentation for this class was generated from the following files:
- NonDCubature.hpp
- NonDCubature.cpp

### 14.131 NonDDREAMBayesCalibration Class Reference

Bayesian inference using the DREAM approach.

Inheritance diagram for NonDDREAMBayesCalibration:

```
    Iterator
     |       
    /       
Analyzer
     |       
  NonD
     |       
NonDCalibration
     |       
NonDBayesCalibration
     |       
NonDDREAMBayesCalibration
```

#### Public Member Functions

- **NonDDREAMBayesCalibration (ProblemDescDB &problem_db, Model &model)**
  
  *standard constructor*
~~ NonDDREAMBayesCalibration ()

destructor

Static Public Member Functions

- static void problem_size (int &chain_num, int &cr_num, int &gen_num, int &pair_num, int &par_num)
  
  *initializer for problem size characteristics in DREAM*

- static void problem_value (std::string *chain_filename, std::string *gr_filename, double &gr_threshold, int &jumpstep, double limits[], int par_num, int &printstep, std::string *restart_read_filename, std::string *restart_write_filename)

  *Filename and data initializer for DREAM.*

- static double prior_density (int par_num, double zp[])  

  *Compute the prior density at specified point zp.*

- static double * prior_sample (int par_num)

  *Sample the prior and return an array of parameter values.*

- static double sample_likelihood (int par_num, double zp[])

  *Likelihood function for call-back from DREAM to DAKOTA for evaluation.*

Protected Member Functions

- void calibrate ()

- void archive_acceptance_chain ()

  *save the final x-space acceptance chain and corresponding function values*

Static Protected Member Functions

- static void cache_chain (const double *const z)

  *Callback to archive the chain from DREAM, potentially leaving it in u-space.*

Protected Attributes

- RealVector paramMins

  *lower bounds on calibrated parameters*

- RealVector paramMaxs

  *upper bounds on calibrated parameters*

- int numChains

  *number of concurrent chains*

- int numGenerations

  *number of generations*

- int numCR

  *number of CR-factors*

- int crossoverChainPairs

  *number of crossover chain pairs*

- Real grThreshold

  *threshold for the Gelmin-Rubin statistic*
• int jumpStep
  how often to perform a long jump in generations
• boost::mt19937 rnumGenerator
  random number engine for sampling the prior

Static Private Attributes

• static NonDDREAMBayesCalibration * nonDDREAMInstance
  Pointer to current class instance for use in static callback functions.

Additional Inherited Members

14.131.1 Detailed Description
Bayesian inference using the DREAM approach.

This class performed Bayesian calibration using the DREAM (Markov Chain Monte Carlo acceleration by Differential Evolution) implementation of John Burkhardt (FSU), adapted from that of Guannan Zhang (ORNL)

14.131.2 Constructor & Destructor Documentation
NonDDREAMBayesCalibration ( ProblemDescDB & problem_db, Model & model )
standard constructor

This constructor is called for a standard letter-envelope iterator instantiation. In this case, set_db_list_nodes has been called and probDescDB can be queried for settings from the method specification.

References NonDBayesCalibration::chainSamples, NonDDREAMBayesCalibration::crossoverChainPairs, NonDDREAMBayesCalibration::grThreshold, NonDDREAMBayesCalibration::jumpStep, NonDDREAMBayesCalibration::numChains, NonDDREAMBayesCalibration::numCR, and NonDDREAMBayesCalibration::numGenerations.

14.131.3 Member Function Documentation

void problem_size ( int & chain_num, int & cr_num, int & gen_num, int & pair_num, int & par_num ) [static]
initializer for problem size characteristics in DREAM

See documentation in DREAM examples

References NonDDREAMBayesCalibration::crossoverChainPairs, NonDDREAMBayesCalibration::nonDDREAMInstance, NonDDREAMBayesCalibration::numChains, Analyzer::numContinuousVars, NonDDREAMBayesCalibration::numCR, NonDDREAMBayesCalibration::numGenerations, and NonDBayesCalibration::numHyperparams.

void problem_value ( std::string * chain_filename, std::string * gr_filename, double & gr_threshold, int & jumpstep, double limits[], int par_num, int & printstep, std::string * restart_read_filename, std::string * restart_write_filename ) [static]
Filename and data initializer for DREAM.

See documentation in DREAM examples

References NonDDREAMBayesCalibration::grThreshold, NonDDREAMBayesCalibration::jumpStep, NonDDREAMBayesCalibration::nonDDREAMInstance, NonDDREAMBayesCalibration::numChains, NonDDREAMBayesCalibration::paramMaxs, and NonDDREAMBayesCalibration::paramMins.
double prior_density ( int par_num, double zp[] ) [static]

Compute the prior density at specified point zp.
See documentation in DREAM examples
References NonDBayesCalibration::nonDBayesInstance, and NonDBayesCalibration::prior_density().

double * prior_sample ( int par_num ) [static]

Sample the prior and return an array of parameter values.
See documentation in DREAM examples
References NonDBayesCalibration::nonDBayesInstance, NonDDREAMBayesCalibration::nonDDREAMInstance, and NonDDREAMBayesCalibration::numGenerator.

double sample_likelihood ( int par_num, double zp[] ) [static]

Likelihood function for call-back from DREAM to DAKOTA for evaluation.
Static callback function to evaluate the likelihood
References Model::continuous_variables(), Model::current_response(), Model::evaluate(), Response::function_values(), NonDBayesCalibration::log_likelihood(), NonDDREAMBayesCalibration::nonDDREAMInstance, Iterator::outputLevel, and NonDBayesCalibration::residualModel.

void calibrate ( ) [protected], [virtual]

Perform the uncertainty quantification DREAM will callback to cache_chain to store the chain
Implements NonDBayesCalibration.
References Dakota::abort_handler(), NonDDREAMBayesCalibration::archive_acceptance_chain(), NonDDREAMBayesCalibration::cache_chain(), NonDCalibration::calibrationData, NonDBayesCalibration::chainSamples, NonDBayesCalibration::compute_statistics(), Model::continuous_variables(), NonDBayesCalibration::initialize_model(), NonDBayesCalibration::mcmcModel, Model::multivariate_distribution(), NonDDREAMBayesCalibration::nonDDREAMInstance, Analyzer::numContinuousVars, NonDBayesCalibration::numHyperparams, NonDBayesCalibration::obsErrorMultiplierMode, NonDDREAMBayesCalibration::paramMaxs, NonDDREAMBayesCalibration::paramMins, NonDBayesCalibration::randomSeed, NonDDREAMBayesCalibration::numGenerator, and NonDBayesCalibration::standardizedSpace.

void cache_chain ( const double * const z ) [static], [protected]

Callback to archive the chain from DREAM, potentially leaving it in u-space.
Archive the chain from DREAM. This default implementation is aggregating from the parallel chains in a round-robin fashion.
References NonDBayesCalibration::acceptanceChain, NonDDREAMBayesCalibration::nonDDREAMInstance, Analyzer::num_samples(), NonDDREAMBayesCalibration::numChains, Analyzer::numContinuousVars, NonDDREAMBayesCalibration::numGenerations, and NonDBayesCalibration::numHyperparams.
Referenced by NonDDREAMBayesCalibration::calibrate().
The documentation for this class was generated from the following files:

- NonDDREAMBayesCalibration.hpp
- NonDDREAMBayesCalibration.cpp
14.132 NonDExpansion Class Reference

Base class for polynomial chaos expansions (PCE), stochastic collocation (SC) and functional tensor train (FT)

Inheritance diagram for NonDExpansion:

```
  NonDExpansion
     | NonD
     | Analyzer
     | Iterator

NonDC3FunctionTrain  NonDPolynomialChaos  NonDStochCollocation  NonDSurrogateExpansion
NonDMultilevelFunctionTrain  NonDMultilevelPolynomialChaos  NonDMultilevelStochCollocation
```

Public Member Functions

- **NonDExpansion (ProblemDescDB &problem_db, Model &model)**
  - standard constructor
- **NonDExpansion (unsigned short method_name, Model &model, short exp_coeffs_approach, const RealVector &dim_pref, int seed, short refine_type, short refine_control, short covar_control, Real colloc_ratio, short rule_nest, short rule_growth, bool piecewise_basis, bool use_derivs)**
  - alternate constructor
- **~NonDExpansion ()**
  - destructor
- **bool resize ()**
  - reinitializes iterator based on new variable size
- **void derived_init_communicators (ParLevLitIter pl_iter)**
  - derived class contributions to initializing the communicators associated with this Iterator instance
- **void derived_set_communicators (ParLevLitIter pl_iter)**
  - derived class contributions to setting the communicators associated with this Iterator instance
- **void derived_free_communicators (ParLevLitIter pl_iter)**
  - derived class contributions to freeing the communicators associated with this Iterator instance
- **void nested_variable_mappings (const SizetArray &c_index1, const SizetArray &di_index1, const SizetArray &ds_index1, const SizetArray &dr_index1, const ShortArray &c_target2, const ShortArray &di_target2, const ShortArray &ds_target2, const ShortArray &dr_target2)**
  - set primaryA\{CV,DIV,DRV\}MapIndices, secondaryA\{CV,DIV,DRV\}MapTargets within derived Iterators; supports computation of higher-level sensitivities in nested contexts (e.g., derivatives of statistics w.r.t. inserted design variables)
- **void core_run ()**
  - perform a forward uncertainty propagation using PCE/SC methods
- **void print_results (std::ostream &s, short results_state=FINAL_RESULTS)**
  - print the final statistics
- **const Model & algorithm_space_model () const**
- **virtual size_t collocation_points () const**
14.132. NONDEXPANSION CLASS REFERENCE

- virtual int random_seed() const
  return specification for random seed (may be part of a sequence specification)
- virtual int first_seed() const
  return first seed in sequence specification (defaults to random_seed())
- virtual void select_refinement_points(const RealVectorArray &candidate_samples, unsigned short batch_size, RealMatrix &best_samples)
  evaluate allSamples for inclusion in the (PCE regression) approximation and retain the best set (well spaced) of size batch_size
- virtual void append_expansion()
  generate numSamplesOnModel, append to approximation data, and update QoI expansions
- virtual void append_expansion(const RealMatrix &samples, const IntResponseMap &resp_map)
  append new data to uSpaceModel and, when appropriate, update expansion order
- virtual void assign_discrepancy_mode()
  verify supported and define default discrepancy emulation mode
- virtual void assign_hierarchical_response_mode()
  define the surrogate response mode for a hierarchical model in multilevel/multifidelity expansions
- virtual void infer_pilot_sample(SizetArray &delta_N_l)
- virtual bool saturated() const
  returns true if current model key has no additional refinements available
- int maximum_refinement_iterations() const
  return maxRefineIterations
- void maximum_refinement_iterations(int max_refine_iter)
  set maxRefineIterations

Protected Member Functions

- virtual void resolve_inputs(short &u_space_type, short &data_order)
  perform error checks and mode overrides
- virtual void initialize_u_space_model()
  initialize uSpaceModel polynomial approximations with PCE/SC data
- virtual void initialize_expansion()
  initialize random variable definitions and final stats arrays
- virtual void compute_expansion()
  form the expansion by calling uSpaceModel.build_approximation()
- virtual void finalize_expansion()
  finalize mappings for the uSpaceModel
- virtual void assign_specification_sequence()
  assign the current values from the input specification sequence
- virtual void increment_specification_sequence()
  increment the input specification sequence and assign values
- virtual void update_expansion()
  update an expansion; avoids overhead in compute_expansion()
- virtual void combined_to_active()
• virtual void archive_coefficients ()
  archive expansion coefficients, as supported by derived instance

• virtual void metric_roll_up ()
  perform any required expansion roll-ups prior to metric computation

• virtual void push_increment ()
  helper function to manage different push increment cases

• virtual void pop_increment ()
  helper function to manage different pop increment cases

• virtual Real compute_covariance_metric (bool revert, bool print_metric)
  compute 2-norm of change in response covariance

• virtual Real compute_level_mappings_metric (bool revert, bool print_metric)
  compute 2-norm of change in final statistics

• virtual void compute_statistics (short results_state=FINAL_RESULTS)
  calculate analytic and numerical statistics from the expansion, supporting {REFINEMENT,INTERMEDIATE,FINAL}_RESULTS modes

• virtual void pull_candidate (RealVector &stats_star)
  extract statistics from native stats arrays for a selected candidate

• virtual void push_candidate (const RealVector &stats_star)
  restore statistics into native stats arrays for a selected candidate

• virtual void initialize_ml_regression (size_t num_lev, bool &import_pilot)
  initializations for multilevel_regression()

• virtual void increment_sample_sequence (size_t new_samp, size_t total_samp, size_t step)
  increment sequence in numSamplesOnModel for multilevel_regression()

• virtual void sample_allocation_metric (Real &metric, Real power)
  accumulate one of the level metrics for {RIP,RANK}_SAMPLING cases

• virtual void compute_sample_increment (const RealVector &lev_metrics, const SizetArray &N_l, SizetArray &delta_N_l)
  compute delta_N_l for {RIP,RANK}_SAMPLING cases

• virtual void finalize_ml_regression ()
  finalizations for multilevel_regression()

• virtual void update_samples_from_order_increment ()
  update numSamplesOnModel after an order increment

• virtual void update_samples_from_order_decrement ()
  update (restore previous) numSamplesOnModel after an order decrement

• virtual void print_sobol_indices (std::ostream &s)
  print global sensitivity indices

• void initialize_response_covariance ()
  set covarianceControl defaults and shape respCovariance

• void update_final_statistics ()
  update function values within finalStatistics

• void update_final_statistics_gradients ()
  update function gradients within finalStatistics
• void initialize_u_space_grid()  
  helper for initializing a numerical integration grid
• void check_dimension_preference (const RealVector &dim_pref) const
  check length and content of dimension preference vector
• void construct_cubature (Iterator &u_space_sampler, Model &g_u_model, unsigned short cub_int_order)
  assign a NonDCubature instance within u_space_sampler
• void construct_quadrature (Iterator &u_space_sampler, Model &g_u_model, unsigned short quad_order, const RealVector &dim_pref)
  assign a NonDQuadrature instance within u_space_sampler based on a quad_order specification
• void construct_quadrature (Iterator &u_space_sampler, Model &g_u_model, unsigned short quad_order, const RealVector &dim_pref, int filtered_samples)
  assign a NonDQuadrature instance within u_space_sampler that generates a filtered tensor product sample set
• void construct_quadrature (Iterator &u_space_sampler, Model &g_u_model, unsigned short quad_order, const RealVector &dim_pref, int random_samples, int seed)
  assign a NonDQuadrature instance within u_space_sampler that samples randomly from a tensor product multi-index
• void construct_sparse_grid (Iterator &u_space_sampler, Model &g_u_model, unsigned short ssg_level, const RealVector &dim_pref)
  assign a NonDSparseGrid instance within u_space_sampler
• void configure_expansion_orders (unsigned short exp_order, const RealVector &dim_pref, UShortArray &exp_orders)
  configure exp_orders from inputs
• void configure_pecos_options ()
  configure expansion and basis configuration options for Pecos polynomial approximations
• void construct_expansion_sampler (unsigned short sample_type, const String &rng, unsigned short integration_refine=NO_INT_REFINE, const IntVector &refine_samples=IntVector(), const String &import_approx_file=String(), unsigned short import_approx_format=TABULAR_ANNOTATED, bool import_approx_active_only=false)
  construct the expansionSampler for evaluating samples on uSpaceModel
• void multifidelity_expansion (short refine_type, bool to_active=true)
  construct a multifidelity expansion, across model forms or discretization levels
• void greedy_multifidelity_expansion ()
  construct a multifidelity expansion, across model forms or discretization levels
• void multilevel_regression ()
  allocate a multilevel expansion based on some approximation to an optimal resource allocation across model forms/discretization levels
• void configure_sequence (unsigned short &num_steps, unsigned short &fixed_index, bool &multilevel, bool mf_precedence)
  configure fidelity/level counts from model hierarchy
• void configure_cost (unsigned short num_steps, bool multilevel, RealVector &cost)
  configure fidelity/level counts from model hierarchy
• void configure_indices (unsigned short group, unsigned short form, unsigned short lev, unsigned short s_index)
  configure response mode and active/truth/surrogate model keys within a hierarchical model. s_index is the sequence index that defines the active dimension for a model sequence.
• Real \texttt{sequence\_cost} (unsigned short step, const RealVector &cost)
  
  return aggregate cost (one or more models) for a level sample

• void \texttt{compute\_equivalent\_cost} (const SizetArray &N, const RealVector &cost)
  
  compute equivHFEvals from samples per level and cost per evaluation

• void \texttt{compute\_sample\_increment} (const RealVector &agg\_var, const RealVector &cost, Real sum_root_var\_cost, Real eps_sq_div_2, const SizetArray &N, SizetArray &delta_N)
  
  compute increment in samples for \texttt{multilevel\_regression()} based on \texttt{ESTIMATOR\_VARIANCE}

• size\_t \texttt{collocation\_points} (size\_t index) const
  
  return number of collocation points for index within model sequence

• int \texttt{random\_seed} (size\_t index) const
  
  return random seed for index within model sequence

• void \texttt{refine\_expansion} ()
  
  refine the reference expansion found by \texttt{compute\_expansion()} using uniform/adaptive p-/h-refinement strategies

• void \texttt{pre\_refinement} ()
  
  initialization of expansion refinement, if necessary

• size\_t \texttt{core\_refinement} (Real &metric, bool revert=false, bool print_metric=true)
  
  advance the refinement strategy one step

• void \texttt{post\_refinement} (Real &metric, bool reverted=false)
  
  finalization of expansion refinement, if necessary

• void \texttt{increment\_grid} (bool update_anisotropy=true)
  
  helper function to manage different grid increment cases

• void \texttt{decrement\_grid} ()
  
  helper function to manage different grid decrement cases

• void \texttt{merge\_grid} ()
  
  helper function to manage different grid merge cases

• void \texttt{increment\_order\_and\_grid} ()
  
  uniformly increment the expansion order and structured/unstructured grid (PCE and FT)

• void \texttt{decrement\_order\_and\_grid} ()
  
  uniformly decrement the expansion order and structured/unstructured grid (PCE and FT)

• void \texttt{update\_model\_from\_samples} ()
  
  publish numSamplesOnModel update to the \texttt{DataFitSurrModel} instance

• void \texttt{update\_u\_space\_sampler} (size\_t sequence\_index, const UShortArray &approx_orders)
  
  perform sampler updates after a change to numSamplesOnModel (shared code from ML/MF updaters)

• void \texttt{statistics\_type} (short stats\_type, bool clear\_bits=true)
  
  update stats\_type, here and in Pecos::ExpansionConfigOptions

• void \texttt{aggregate\_variance} (Real &agg\_var)
  
  Aggregate variance across the set of QoI for a particular model level.

• void \texttt{compute\_covariance} ()
  
  calculate the response covariance (diagonal or full matrix) for the expansion indicated by stats\_Type

• void \texttt{compute\_active\_covariance} ()
  
  calculate the response covariance of the active expansion

• void \texttt{compute\_combined\_covariance} ()
  
  calculate the response covariance of the combined expansion
• void compute_active_diagonal_variance ()
  calculate the diagonal response variance of the active expansion
• void compute_combined_diagonal_variance ()
  calculate the diagonal response variance of the combined expansion
• void compute_off_diagonal_covariance ()
  calculate off diagonal terms in respCovariance(i,j) for j<i for the expansion indicated by statsType
• void compute_active_off_diagonal_covariance ()
  calculate off diagonal terms in respCovariance(i,j) for j<i using the active expansion coefficients
• void compute_combined_off_diagonal_covariance ()
  calculate off diagonal terms in respCovariance(i,j) for j<i using the combined expansion coefficients
• void compute_moments ()
  compute expansion moments; this uses a lightweight approach for incremental statistics (no additional moments; no finalStatistics update)
• void compute_level_mappings ()
  compute all analytic/numerical level mappings; this uses a lightweight approach for incremental statistics (no derivatives, no finalStatistics update)
• void compute_numerical_level_mappings ()
  compute only numerical level mappings; this uses a lightweight approach for incremental statistics (no derivatives, no finalStatistics update)
• void compute_sobol_indices ()
  compute Sobol’ indices for main, interaction, and total effects; this is intended for incremental statistics
• void print_covariance (std::ostream &s)
  print resp\{Variance, Covariance\}
• void print_variance (std::ostream &s, const RealVector &resp_var, const String &prepend="")
  print resp\_var (response variance vector) using optional pre-pend
• void print_covariance (std::ostream &s, const RealSymMatrix &resp_covar, const String &prepend="")
  print resp\_covar (response covariance matrix) using optional pre-pend
• void archive_moments ()
  archive the central moments (numerical and expansion) to ResultsDB
• void archive_sobol_indices ()
  archive the Sobol’ indices to the resultsDB
• void pull_reference (RealVector &stats_ref)
• void push_reference (const RealVector &stats_ref)
• void pull_lower_triangle (const RealSymMatrix &mat, RealVector &vec, size_t offset=0)
  pull lower triangle of symmetric matrix into vector
• void push_lower_triangle (const RealVector &vec, RealSymMatrix &mat, size_t offset=0)
  push vector into lower triangle of symmetric matrix
• int terms_ratio_to_samples (size_t num_exp_terms, Real colloc_ratio)
  convert number of regression terms and collocation ratio to a number of collocation samples
• Real terms_samples_to_ratio (size_t num_exp_terms, int samples)
  convert number of regression terms and number of collocation samples to a collocation ratio
Protected Attributes

- **Model uSpaceModel**
  
  Model representing the approximate response function in u-space, after u-space recasting and polynomial data fit recursions.

- **Iterator expansionSampler**
  
  Iterator used for sampling on the uSpaceModel to generate approximate probability/reliability/response level statistics. Currently this is an LHS sampling instance, but AIS could also be used.

- **Iterator importanceSampler**
  
  Iterator used to refine the approximate probability estimates generated by the expansionSampler using importance sampling.

- **short expansionCoeffsApproach**
  
  Method for collocation point generation and subsequent calculation of the expansion coefficients

- **short expansionBasisType**
  
  Type of expansion basis: DEFAULT BASIS or Pecos::{NODAL,HIERARCHICAL},INTERPOLANT for SC or Pecos::{TENSOR_PRODUCT,TOTAL_ORDER,ADAPTED} BASIS for PCE regression.

- **short statsType**
  
  Type of statistical metric: NO_EXPANSION_STATS, ACTIVE_EXPANSION_STATS, or COMBINED_EXPANSION_STATS.

- **RealVector dimPrefSpec**
  
  User specification for dimension preference.

- **SizetArray collocPtsSeqSpec**
  
  User specification of number of initial samples per model instance, including adaptive cases where an optimal sample profile is the target of iteration (e.g., multilevel_regression()).

- **Real collocRatio**
  
  Factor applied to terms’ termsOrder in computing number of regression points, either user-specified or inferred.

- **Real termsOrder**
  
  Exponent applied to number of expansion terms for computing number of regression points (usually 1).

- **int randomSeed**
  
  Seed for random number generator (used for regression with LHS and sub-sampled tensor grids, as well as for expansionSampler).

- **SizetArray randomSeedSeqSpec**
  
  User specification for seed sequence.

- **bool fixedSeed**
  
  Don’t continue an existing random number sequence, rather reset seed each time within some sampling-based iteration.

- **size_t mlmfIter**
  
  Top level iteration counter in adaptive NonExpansion ML/MF algorithms, allowing special updating logic for some sequence handlers.

- **bool allVars**
  
  Flag for combined variable expansions which include a non-probabilistic subset (design, epistemic, state).

- **bool tensorRegression**
  
  Option for regression FT using a filtered set of tensor-product quadrature points.

- **short multilevAllocControl**
  
  Type of sample allocation scheme for discretization levels / model forms within multilevel / multifidelity methods.
• short multilevDiscrepEmulation
  emulation approach for multilevel / multifidelity discrepancy: distinct or recursive
• SizetArray NLev
  number of samples allocated to each level of a discretization/model hierarchy within multilevel/multifidelity methods
• Real equivHFEvals
  equivalent number of high fidelity evaluations accumulated using samples across multiple model forms and/or discretization levels
• Real kappaEstimatorRate
  rate parameter for allocation by ESTIMATOR_VARIANCE in multilevel_regression()
• Real gammaEstimatorScale
  scale parameter for allocation by ESTIMATOR_VARIANCE in multilevel_regression()
• int numSamplesOnModel
  number of truth samples performed on g_\mu_model to form the expansion
• int numSamplesOnExpansion
  number of approximation samples performed on the polynomial expansion in order to estimate probabilities
• bool relativeMetric
  flag indicating the use of relative scaling in refinement metrics
• bool nestedRules
  flag for indicating state of nested and non_nested overrides of default rule nesting, which depends on the type of integration driver; this is defined in construct\{quadrature,sparse_grid\}( ), such that override attributes (defined in ctors) must be used upstream
• short ruleNestingOverride
  user override of default rule nesting: NO_NESTING_OVERRIDE, NESTED, or NON_NESTED
• short ruleGrowthOverride
  user override of default rule growth: NO_GROWTH OVERRIDE, RESTRICTED, or UNRESTRICTED
• bool piecewiseBasis
  flag for piecewise specification, indicating usage of local basis polynomials within the stochastic expansion
• bool useDerivs
  flag for use derivatives specification, indicating usage of derivative data (with respect to expansion variables) to enhance the calculation of the stochastic expansion.
• RealVector initialPtU
  stores the initial variables data in u-space
• short refineType
  refinement type: NO_REFINEMENT, P_REFINEMENT, or H_REFINEMENT
• short refineControl
  refinement control: NO_CONTROL, UNIFORM_CONTROL, LOCAL_ADAPTIVE_CONTROL, DIMENSION_ADAPTIVE_CONTROL_SOBOL, DIMENSION_ADAPTIVE_CONTROL_DECAY, or DIMENSION_ADAPTIVE_CONTROL_GENERALIZED
• short refineMetric
  refinement metric: NO_METRIC, COVARIANCE_METRIC, LEVELSTATS_METRIC, or MIXED_STATS_METRIC
• short covarianceControl
  enumeration for controlling response covariance calculation and output: \{DEFAULT,DIAGONAL,FULL\}_COVARIANCE
• unsigned short softConvLimit
number of consecutive iterations within tolerance required to indicate soft convergence

- `RealSymMatrix respCovariance`  
  symmetric matrix of analytic response covariance (full response covariance option)

- `RealVector respVariance`  
  vector of response variances (diagonal response covariance option)

- `RealVector statsStar`  
  stats of the best refinement candidate for the current model indices

- `size_t numUncertainQuant`  
  number of invocations of `core_run()`

**Private Member Functions**

- `void initialize_counts()`  
  initialize data based on variable counts

- `void aggregated_models_mode()`  
  set response mode to `AGGREGATED_MODELS` and recur response size updates

- `void bypass_surrogate_mode()`  
  set response mode to `BYPASS_SURROGATE` and recur response size updates

- `void reduce_total_sobol_sets(RealVector &avg_sobol)`  
  compute average of total Sobol' indices (from VBD) across the response set for use as an anisotropy indicator

- `void reduce_decay_rate_sets(RealVector &min_decay)`  
  compute minimum of spectral coefficient decay rates across the response set for use as an anisotropy indicator

- `size_t increment_sets(Real &delta_star, bool revert, bool print_metric)`  
  perform an adaptive refinement increment using generalized sparse grids

- `void finalize_sets(bool converged_within_tol, bool reverted=false)`  
  finalization of adaptive refinement using generalized sparse grids

- `void select_candidate(size_t best_candidate)`  
  promote selected candidate into reference grid + expansion

- `void select_index_set_candidate(std::set<UShortArray>::const_iterator cit_star)`  
  promote selected index set candidate into reference grid + expansion

- `void select_increment_candidate()`  
  promote selected refinement increment candidate into reference grid + expansion

- `void compute_analytic_statistics()`  
  analytic portion of `compute_statistics()` from post-processing of expansion coefficients (used for `FINAL_RESULTS`)

- `void compute_numerical_statistics()`  
  numerical portion of `compute_statistics()` from sampling on the expansion (used for `FINAL_RESULTS`)

- `void compute_numerical_stat_refinements(RealVectorArray &imp_sampler_stats, RealRealPairArray &min_max_fns)`  
  refinements to numerical probability statistics from `importanceSampler`

- `void define_sampler_asv(ShortArray &sampler_asv)`  
  helper to define the expansionSampler’s data requests when sampling on the expansion

- `void run_sampler(const ShortArray &sampler_asv, RealVector &exp_sampler_stats)`  
  helper to run the expansionSampler and compute its statistics

- `void refine_sampler(RealVectorArray &imp_sampler_stats, RealRealPairArray &min_max_fns)`
helper to refine the results from expansionSampler with importance sampling (for probability levels) or bounds post-processing (for PDFs)

- void print_moments (std::ostream &s)
  print expansion and numerical moments
- void print_local_sensitivity (std::ostream &s)
  print local sensitivities evaluated at initialPtU

**Private Attributes**

- RealMatrix expGradsMeanX
  derivative of the expansion with respect to the x-space variables evaluated at the means (used as uncertainty importance metrics)
- int maxRefineIterations
  maximum number of uniform/adaptive refinement iterations (specialization of maxIterations)
- int maxSolverIterations
  maximum number of regression solver iterations (specialization of maxIterations)
- bool vbdFlag
  flag indicating the activation of variance-bsaed decomposition for computing Sobol’ indices
- unsigned short vbdOrderLimit
  limits the order of interactions within the component Sobol’ indices
- Real vbdDropTol
  tolerance for omitting output of small VBD indices

**Additional Inherited Members**

14.132.1 Detailed Description

Base class for polynomial chaos expansions (PCE), stochastic collocation (SC) and functional tensor train (FT)

The NonDExpansion class provides a base class for methods that use polynomial expansions to approximate the effect of parameter uncertainties on response functions of interest.

14.132.2 Member Function Documentation

**const Model & algorithm_space_model ( ) const [inline], [virtual]**

default definition that gets redefined in selected derived Minimizers
  Reimplemented from Analyzer.
  References NonDExpansion::uSpaceModel.

**void infer_pilot_sample ( SizetArray & delta_N_I ) [virtual]**

Default implementation redefined by Multilevel derived classes.
  Reimplemented in NonDMultilevelPolynomialChaos, and NonDMultilevelFunctionTrain.
  References Dakota::abort_handler().
  Referenced by NonDExpansion::multilevel_regression().
void increment_specification_sequence() [protected], [virtual]

increment the input specification sequence and assign values
Default implementation redefined by Multilevel derived classes.
Reimplemented in NonDMultilevelPolynomialChaos, NonDMultilevelFunctionTrain, and NonDMultilevelStochCollocation.
References Dakota::abort_handler().
Referenced by NonDExpansion::multifidelity_expansion().

void update_expansion() [protected], [virtual]
update an expansion; avoids overhead in compute_expansion()
leave sampler_set, expansion flags, and distribution parameter settings as set previously by compute_expansion();
there should be no need to update these for an expansion refinement.
References Model::append_approximation(), NonDIntegration::evaluate_grid_increment(), NonDExpansion::expansionCoeffsApproach, NonDExpansion::increment_grid(), Iterator::iterator_rep(), Model::push_approximation(), Model::push_available(), NonDIntegration::push_grid_increment(), Model::rebuild_approximation(), Model::subordinate_iterator(), Model::update_approximation(), and NonDExpansion::uSpaceModel.
Referenced by NonDExpansion::core_refinement(), and NonDExpansion::multilevel_regression().

Real compute_covariance_metric(bool revert, bool print_metric) [protected], [virtual]
compute 2-norm of change in response covariance
computes the default refinement metric based on change in respCovariance
Reimplemented in NonDStochCollocation.
References NonDExpansion::compute_moments(), NonDExpansion::compute_off_diagonal_covariance(), NonDExpansion::covarianceControl, NonDExpansion::print_covariance(), NonDExpansion::relativeMetric, NonDExpansion::respCovariance, and NonDExpansion::respVariance.
Referenced by NonDStochCollocation::compute_covariance_metric(), NonDExpansion::core_refinement(), and NonDExpansion::increment_sets().

Real compute_level_mappings_metric(bool revert, bool print_metric) [protected], [virtual]
compute 2-norm of change in final statistics
computes a "goal-oriented" refinement metric employing computed*Levels
Reimplemented in NonDStochCollocation.
References NonDExpansion::compute_level_mappings(), Analyzer::numFunctions, NonD::print_level_mappings(), NonD::pull_level_mappings(), NonD::push_level_mappings(), NonDExpansion::relativeMetric, NonD::requestedGenRelLevels, NonD::requestedProbLevels, NonD::requestedRelLevels, and NonD::requestedRespLevels.
Referenced by NonDStochCollocation::compute_level_mappings_metric(), NonDExpansion::core_refinement(), and NonDExpansion::increment_sets().

void compute_statistics(short results_state = FINAL_RESULTS) [protected], [virtual]
calculate analytic and numerical statistics from the expansion, supporting {REFINEMENT,INTERMEDIATE,FINAL} RESULTS modes
Calculate analytic and numerical statistics from the expansion and log results within final_stats for use in OOU.
References ResultsManager::active(), NonDExpansion::allVars, NonDExpansion::archive_coefficients(), NonDExpansion::archive_moments(), NonDExpansion::archive_sobol_indices(), NonDExpansion::compute_analytic_statistics(), NonDExpansion::compute_level_mappings(), NonDExpansion::compute_moments(), NonDExpansion::compute_numerical_statistics(), NonDExpansion::compute_off_diagonal_covariance(), Model::continuous_variable_labels(), Model::continuous_variables(), NonDExpansion::covarianceControl, NonDExpansion::initialPtU, ResultsManager::insert(), Iterator::iteratedModel, NonDExpansion::refineMetric, Model::response_labels(), ResultsManager::DB, Iterator::resultsNames, Iterator::run_identifier(), NonD::totalLevelRequests, NonDExpansion::update_final_statistics(), NonDExpansion::uSpaceModel, and NonDExpansion::vbdFlag.

Referenced by NonDExpansion::core_refinement(), NonDSurrogateExpansion::core_run(), NonDMultilevelStochCollocation::core_run(), NonDExpansion::core_run(), NonDMultilevelFunctionTrain::core_run(), NonDMultilevelPolynomialChaos::core_run(), NonDExpansion::greedy_multifidelity_expansion(), NonDExpansion::increment_sets(), NonDExpansion::multifidelity_expansion(), and NonDExpansion::refine_expansion().

```cpp
void update_samples_from_order_decrement( ) [protected], [virtual]
```
update (restore previous) numSamplesOnModel after an order decrement
Default implementation: increment/decrement update process is identical
References NonDExpansion::update_samples_from_order_increment().
Referenced by NonDExpansion::decrement_order_and_grid().

```cpp
int random_seed( size_t index ) const [inline], [protected]
```
return random seed for index within model sequence
extract an active seed from a seed sequence
References NonDExpansion::fixedSeed, NonDExpansion::mlmfIter, and NonDExpansion::randomSeedSeqSpec.

```cpp
void increment_order_and_grid( ) [protected]
```
uniformly increment the expansion order and structured/unstructured grid (PCE and FT)
Used for uniform refinement of regression-based PCE / FT.
References NonDQuadrature::increment_grid(), SharedApproxData::increment_order(), Iterator::iterator_rep(), NonDQuadrature::mode(), NonDExpansion::numSamplesOnModel, NonDQuadrature::samples(), Model::shared_approximation(), Model::subordinate_iterator(), NonDExpansion::tensorRegression, NonDQuadrature::update(), NonDExpansion::update_model_from_samples(), NonDExpansion::update_samples_from_order_increment(), and NonDExpansion::uSpaceModel.
Referenced by NonDExpansion::increment_grid().

```cpp
void decrement_order_and_grid( ) [protected]
```
uniformly decrement the expansion order and structured/unstructured grid (PCE and FT)
Used for uniform de-refinement of regression-based PCE / FT.
References SharedApproxData::decrement_order(), Iterator::iterator_rep(), NonDExpansion::numSamplesOnModel, NonDQuadrature::samples(), Model::shared_approximation(), Model::subordinate_iterator(), NonDExpansion::tensorRegression, NonDQuadrature::update(), NonDExpansion::update_model_from_samples(), NonDExpansion::update_samples_from_order_decrement(), and NonDExpansion::uSpaceModel.
Referenced by NonDExpansion::decrement_grid().
14.132.3 Member Data Documentation

bool useDerivs [protected]

flag for use derivatives specification, indicating usage of derivative data (with respect to expansion variables) to enhance the calculation of the stochastic expansion.

This is part of the method specification since the instantiation of the global data fit surrogate is implicit with no user specification. This behavior is distinct from the usage of response derivatives with respect to auxiliary variables (design, epistemic) for computing derivatives of aleatory expansion statistics with respect to these variables.

Referenced by NonDExpansion::compute expansion(), NonDExpansion::configure pecos options(), NonDPolynomialChaos::ratio samples to order(), NonDStochCollocation::resolve inputs(), NonDPolynomialChaos::resolve inputs(), NonDExpansion::terms ratio to samples(), and NonDExpansion::terms samples to ratio().

The documentation for this class was generated from the following files:

- NonDExpansion.hpp
- NonDExpansion.cpp

14.133 NonDGlobalEvidence Class Reference

Class for the Dempster-Shafer Evidence Theory methods within DAKOTA/UQ.

Inheritance diagram for NonDGlobalEvidence:

```
Iterator

Analyzer

NonD

NonDInterval

NonDGlobalInterval

NonDGlobalEvidence
```

Public Member Functions

- NonDGlobalEvidence (ProblemDescDB &problem_db, Model &model)
  
  *constructor*

- ~NonDGlobalEvidence ()
  
  *destructor*

- void initialize ()
  
  *perform any required initialization*

- void set_cell_bounds ()
  
  *set the optimization variable bounds for each cell*
### Additional Inherited Members

#### 14.133.1 Detailed Description

Class for the Dempster-Shafer Evidence Theory methods within DAKOTA/UQ.

The NonDEvidence class implements the propagation of epistemic uncertainty using Dempster-Shafer theory of evidence. In this approach, one assigns a set of basic probability assignments (BPA) to intervals defined for the uncertain variables. Input interval combinations are calculated, along with their BPA. Currently, the response function is evaluated at a set of sample points, then a response surface is constructed which is sampled extensively to find the minimum and maximum within each input interval cell, corresponding to the belief and plausibility within that cell, respectively. This data is then aggregated to calculate cumulative distribution functions for belief and plausibility.

The documentation for this class was generated from the following files:

- NonDGlobalEvidence.hpp
- NonDGlobalEvidence.cpp

### 14.134 NonDGlobalInterval Class Reference

Class for using global nongradient-based optimization approaches to calculate interval bounds for epistemic uncertainty quantification.

Inheritance diagram for NonDGlobalInterval:

- Iterator
- Analyzer
- NonD
- NonDInterval
- NonDGlobalInterval
- NonDGlobalEvidence
- NonDGlobalSingleInterval
Public Member Functions

- **NonDGlobalInterval** (ProblemDescDB &problem_db, Model &model)
  
  constructor

- **NonDGlobalInterval** ()
  
  destructor

- **void derived_init_communicators** (ParLevLIter pl_iter)
  
  derived class contributions to initializing the communicators associated with this Iterator instance

- **void derived_set_communicators** (ParLevLIter pl_iter)
  
  derived class contributions to setting the communicators associated with this Iterator instance

- **void derived_free_communicators** (ParLevLIter pl_iter)
  
  derived class contributions to freeing the communicators associated with this Iterator instance

- **void core_run** ()
  
  Performs an optimization to determine interval bounds for an entire function or interval bounds on a particular statistical estimator.

- **const Model & algorithm_space_model** () const

Protected Member Functions

- **virtual void initialize** ()
  
  perform any required initialization

- **virtual void set_cell_bounds** ()
  
  set the optimization variable bounds for each cell

- **virtual void get_best_sample** (bool maximize, bool eval_approx)
  
  determine truthFnStar and approxFnStar

- **virtual void post_process_cell_results** (bool maximize)
  
  post-process a cell minimization/maximization result

- **virtual void post_process_response_fn_results** ()
  
  post-process the interval computed for a response function

- **virtual void post_process_final_results** ()
  
  perform final post-processing

- **void post_process_run_results** (bool maximize)
  
  post-process an optimization execution: output results, update convergence controls, and update GP approximation

- **void evaluate_response_star_truth** ()
  
  evaluate the truth response at the optimal variables solution and update the GP with the new data

Protected Attributes

- **Iterator daceIterator**
  
  LHS iterator for constructing initial GP for all response functions.

- **Model fHatModel**
  
  GP model of response, one approximation per response function.

- **Iterator intervalOptimizer**
  
  optimizer for solving surrogate-based subproblem: NCSU DIRECT optimizer for maximizing expected improvement or mixed EA if discrete variables.
• Model intervalOptModel
  recast model which formulates the surrogate-based optimization subproblem (recasts as design problem; may as-
similate mean and variance to enable max(expected improvement))
• Real approxFinStar
  approximate response corresponding to minimum/maximum truth response
• Real truthFinStar
  minimum/maximum truth response function value

Static Private Member Functions
• static void EIF_objective_min (const Variables &sub_model_vars, const Variables &recast_vars, const Re-
sponse &sub_model_response, Response &recast_response)
  static function used as the objective function in the Expected Improvement Function (EIF) for minimizing the GP
• static void EIF_objective_max (const Variables &sub_model_vars, const Variables &recast_vars, const Re-
sponse &sub_model_response, Response &recast_response)
  static function used as the objective function in the Expected Improvement Function (EIF) for maximizing the GP
• static void extract_objective (const Variables &sub_model_vars, const Variables &recast_vars, const Re-
sponse &sub_model_response, Response &recast_response)
  static function used to extract the active objective function when optimizing for an interval lower or upper bound
  (non(EIF formulations). The sense of the optimization is set separately.

Private Attributes
• const int seedSpec
  the user seed specification (default is 0)
• int numSamples
  the number of samples used in the surrogate
• String rngName
  name of the random number generator
• bool gpModelFlag
  flag indicating use of GP surrogate emulation
• bool eifFlag
  flag indicating use of maximized expected improvement for GP iterate selection
• unsigned short improvementConvergeCntr
  counter for number of successive iterations that the iteration improvement is less than the convergenceTol
• unsigned short improvementConvergeLimit
  counter for number of successive iterations that the iteration improvement is less than the convergenceTol
• Real distanceTol
  tolerance for L_2 change in optimal solution
• unsigned short distanceConvergeCntr
  counter for number of successive iterations that the L_2 change in optimal solution is less than the convergenceTol
• unsigned short distanceConvergeLimit
  counter for number of successive iterations that the L_2 change in optimal solution is less than the convergenceTol
• RealVector prevCVStar
  stores previous optimal point for continuous variables; used for assessing convergence
CHAPTER 14. CLASS DOCUMENTATION

- IntVector `prevDIVStar`
  stores previous optimal point for discrete integer variables; used for assessing convergence
- RealVector `prevDRVStar`
  stores previous optimal point for discrete real variables; used for assessing convergence
- Real `prevFnStar`
  stores previous solution value for assessing convergence
- size_t `globalIterCntr`
  global iteration counter for number of surrogate-based min/max solves
- bool `boundConverged`
  flag indicating convergence of a minimization or maximization cycle
- bool `allResponsesPerIter`
  flag for maximal response extraction (all response values obtained on each function call)
- short `dataOrder`
  order of the data used for surrogate construction, in ActiveSet request vector 3-bit format; user may override

Static Private Attributes

- static NonDGlobalInterval * `nondGIInstance`
  pointer to the active object instance used within the static evaluator functions in order to avoid the need for static data

Additional Inherited Members

14.134.1 Detailed Description

Class for using global nongradient-based optimization approaches to calculate interval bounds for epistemic uncertainty quantification.

The NonDGlobalInterval class supports global nongradient-based optimization approaches to determining interval bounds for epistemic UQ. The interval bounds may be on the entire function in the case of pure interval analysis (e.g., intervals on input = intervals on output), or the intervals may be on statistics of an “inner loop” aleatory analysis such as intervals on means, variances, or percentile levels. The preliminary implementation will use a Gaussian process surrogate to determine interval bounds.

14.134.2 Member Function Documentation

`const Model & algorithm_space_model ( ) const [inline], [virtual]`

default definition that gets redefined in selected derived Minimizers
Reimplemented from Analyzer.
References NonDGlobalInterval::fHatModel.
The documentation for this class was generated from the following files:

- NonDGlobalInterval.hpp
- NonDGlobalInterval.cpp
14.135 NonDGlobalReliability Class Reference

Class for global reliability methods within DAKOTA/UQ.

Inheritance diagram for NonDGlobalReliability:

```
    Iterator
     |       
    Operator
     |       
    NonD
     |       
    NonDReliability
```

### Public Member Functions

- **NonDGlobalReliability (ProblemDescDB &problem_db, Model &model)**
  
  *constructor*

- **~NonDGlobalReliability ()**
  
  *destructor*

- **bool resize ()**
  
  reinitializes iterator based on new variable size

- **void derived_init_communicators (ParLevLIter pl_iter)**
  
  derived class contributions to initializing the communicators associated with this Iterator instance

- **void derived_set_communicators (ParLevLIter pl_iter)**
  
  derived class contributions to setting the communicators associated with this Iterator instance

- **void derived_free_communicators (ParLevLIter pl_iter)**
  
  derived class contributions to freeing the communicators associated with this Iterator instance

- **void pre_run ()**
  
  pre-run portion of run (optional); re-implemented by Iterators which can generate all Variables (parameter sets) a priori

- **void core_run ()**
  
  core portion of run; implemented by all derived classes and may include pre/post steps in lieu of separate pre/post

- **void print_results (std::ostream &s, short results_state=FINAL_RESULTS)**
  
  print the final iterator results

### Private Member Functions

- **void optimize_gaussian_process ()**
  
  construct the GP using EGO/SKO

- **void importance_sampling ()**
  
  perform multimodal adaptive importance sampling on the GP
• **void get_best_sample ()**
  determine current best solution from among sample data for expected improvement function in Performance Measure Approach (PMA)

• **Real constraint_penalty (const Real &constraint, const RealVector &c_variables)**
  calculate the penalty to be applied to the PMA constraint value

• **Real expected_improvement (const RealVector &expected_values, const Variables &recast_vars)**
  expected improvement function for the GP

• **Real expected_feasibility (const RealVector &expected_values, const Variables &recast_vars)**
  expected feasibility function for the GP

• **void x_truth_evaluation (short mode)**
  evaluate iteratedModel at current point to collect x-space truth data

• **void x_truth_evaluation (const RealVector &c_vars_u, short mode)**
  evaluate iteratedModel at specified point to collect x-space truth data

• **void u_truth_evaluation (const RealVector &c_vars_u, short mode)**
  evaluate uSpaceModel in BYPASS_SURROGATE mode to collect u-space truth data at specified point

• **void u_evaluation (const RealVector &c_vars_u, short mode)**
  evaluate uSpaceModel to collect u-space surrogate data at specified point

**Static Private Member Functions**

• **static void EIF_objective_eval (const Variables &sub_model_vars, const Variables &recast_vars, const Response &sub_model_response, Response &recast_response)**
  static function used as the objective function in the Expected Improvement (EIF) problem formulation for PMA

• **static void EFF_objective_eval (const Variables &sub_model_vars, const Variables &recast_vars, const Response &sub_model_response, Response &recast_response)**
  static function used as the objective function in the Expected Feasibility (EFF) problem formulation for RIA

**Private Attributes**

• **Real fnStar**
  minimum penalized response from among true function evaluations

• **short meritFunctionType**
  type of merit function used to penalize sample data

• **Real lagrangeMult**
  Lagrange multiplier for standard Lagrangian merit function.

• **Real augLagrangeMult**
  Lagrange multiplier for augmented Lagrangian merit function.

• **Real penaltyParameter**
  penalty parameter for augmented Lagrangian merit function

• **Real lastConstraintViolation**
  constraint violation at last iteration, used to determine if the current iterate should be accepted (must reduce violation)

• **bool lastIterateAccepted**
  flag to determine if last iterate was accepted this controls update of parameters for augmented Lagrangian merit fn

• **short dataOrder**
  order of the data used for surrogate construction, in ActiveSet request vector 3-bit format; user may override responses spec
Static Private Attributes

- static NonDGlobalReliability * nondGlobRelInstance
  
  pointer to the active object instance used within the static evaluator functions in order to avoid the need for static data

Additional Inherited Members

14.135.1 Detailed Description

Class for global reliability methods within DAKOTA/UQ.

The NonDGlobalReliability class implements EGO/SKO for global MPP search, which maximizes an expected improvement function derived from Gaussian process models. Once the limit state has been characterized, a multimodal importance sampling approach is used to compute probabilities.

14.135.2 Member Function Documentation

void pre_run() [virtual]

pre-run portion of run (optional); re-implemented by Iterators which can generate all Variables (parameter sets) a priori
  
  pre-run phase, which a derived iterator may optionally reimplement; when not present, pre-run is likely integrated into the derived run function. This is a virtual function; when re-implementing, a derived class must call its nearest parent’s pre_run(), if implemented, typically before performing its own implementation steps.
  
  Reimplemented from Analyzer.
  
  References Model::initialize_mapping(), Iterator::methodPCIter, NonD::miPLIndex, NonDReliability::mpp-Model, Analyzer::pre_run(), and Model::update_from_subordinate_model().

void core_run() [virtual]

core portion of run; implemented by all derived classes and may include pre/post steps in lieu of separate pre/post Virtual run function for the iterator class hierarchy. All derived classes need to redefine it.
  
  Reimplemented from Iterator.
  
  References NonDGlobalReliability::importance_sampling(), NonDGlobalReliability::nondGlobRelInstance, and NonDGlobalReliability::optimize_gaussian_process().

void print_results ( std::ostream & s, short results_state = FINAL_RESULTS ) [virtual]

print the final iterator results
  
  This virtual function provides additional iterator-specific final results outputs beyond the function evaluation summary printed in finalize_run().
  
  Reimplemented from Analyzer.
  
  References NonD::cdfFlag, NonD::computedGenRelLevels, NonD::computedProbLevels, NonD::computedRespLevels, Iterator::iteratedModel, Analyzer::numFunctions, NonD::print_densities(), Model::response_labels(), and Dakota::write_precision.
  
  The documentation for this class was generated from the following files:

- NonDGlobalReliability.hpp
- NonDGlobalReliability.cpp
14.136 NonDGlobalSingleInterval Class Reference

Class for using global nongradient-based optimization approaches to calculate interval bounds for epistemic uncertainty quantification.

Inheritance diagram for NonDGlobalSingleInterval:

- NonDGlobalSingleInterval
- NonDGlobalInterval
- NonDInterval
- NonD
- Analyzer
- Iterator

Public Member Functions

- NonDGlobalSingleInterval (ProblemDescDB &problem_db, Model &model)
  constructor
- ~NonDGlobalSingleInterval ()
  destructor

Protected Member Functions

- void initialize ()
  perform any required initialization
- void post_process_cell_results (bool maximize)
  post-process a cell minimization/maximization result
- void get_best_sample (bool maximize, bool eval_approx)
  determine truthFnStar and approxFnStar

Private Attributes

- size_t statCntr
  counter for finalStatistics

Additional Inherited Members

14.136.1 Detailed Description

Class for using global nongradient-based optimization approaches to calculate interval bounds for epistemic uncertainty quantification.
The **NonDGlobalSingleInterval** class supports global nongradient-based optimization approaches to determining interval bounds for epistemic UQ. The interval bounds may be on the entire function in the case of pure interval analysis (e.g. intervals on input = intervals on output), or the intervals may be on statistics of an “inner loop” aleatory analysis such as intervals on means, variances, or percentile levels. The preliminary implementation will use a Gaussian process surrogate to determine interval bounds.

The documentation for this class was generated from the following files:

- NonDGlobalSingleInterval.hpp
- NonDGlobalSingleInterval.cpp

### 14.137 NonDGPImpSampling Class Reference

Class for the Gaussian Process-based Importance Sampling method.

**Inheritance diagram for NonDGPImpSampling:**

```
NonDGPImpSampling
  |    |
  v    v
NonDSampling
  |    |
  v    v
NonD
  |    |
  v    v
Analyzer
  |    |
  v    v
Iterator
```

**Public Member Functions**

- **NonDGPImpSampling** (**ProblemDescDB** &problem_db, **Model** &model)
  - *standard constructor*
- **~NonDGPImpSampling** ()
  - *destructor*
- **bool resize** ()
  - *reinitializes iterator based on new variable size*
- **void derived_init_communicators** (**ParLevLIter** pl_iter)
  - *derived class contributions to initializing the communicators associated with this Iterator instance*
- **void derived_set_communicators** (**ParLevLIter** pl_iter)
  - *derived class contributions to setting the communicators associated with this Iterator instance*
- **void derived_free_communicators** (**ParLevLIter** pl_iter)
  - *derived class contributions to freeing the communicators associated with this Iterator instance*
- **void core_run** ()
  - *perform the GP importance sampling and return probability of failure*
- **void print_results** (**std::ostream** &s, short results_state=FINAL_RESULTS)
  - *print the final statistics*
• Real final_probability ()
  returns the probability calculated by the importance sampling

Private Member Functions
• RealVector calcExpIndicator (const int respFnCount, const Real respThresh)
  function to calculate the expected indicator probabilities
• Real calcExpIndPoint (const int respFnCount, const Real respThresh, const RealVector this_mean, const RealVector this_var)
  function to calculate the expected indicator probabilities for one point
• void calcRhoDraw ()
  function to update the rhoDraw data, adding x values and rho draw values
• RealVector drawNewX (int this_k)
  function to pick the next X value to be evaluated by the Iterated model

Private Attributes
• Iterator gpBuild
  LHS iterator for building the initial GP.
• Iterator gpEval
  LHS iterator for sampling on the GP.
• Model gpModel
  GP model of response, one approximation per response function.
• Iterator sampleRhoOne
  LHS iterator for sampling from the rhoOneDistribution.
• int numPtsAdd
  the number of points added to the original set of LHS samples
• int numPtsTotal
  the total number of points
• int numEmulEval
  the number of points evaluated by the GP each iteration
• Real finalProb
  the final calculated probability (p)
• RealVectorArray gpCvars
  Vector to hold the current values of the current sample inputs on the GP.
• RealVectorArray gpMeans
  Vector to hold the current values of the current mean estimates for the sample values on the GP.
• RealVectorArray gpVar
  Vector to hold the current values of the current variance estimates for the sample values on the GP.
• RealVector expIndicator
  Vector to hold the expected indicator values for the current GP samples.
• RealVector rhoDraw
  Vector to hold the rhoDraw values for the current GP samples.
• RealVector normConst
Vector to hold the normalization constant calculated for each point added.

- RealVector `indicator`
  Integer Vector to hold indicator for actual simulation values vs. threshold.

- RealVectorArray `xDrawThis`
  `xDrawThis`, appended locally to hold the X values of emulator points chosen

- RealVector `expIndThis`
  `expIndThis`, appended locally to hold the expected indicator

- RealVector `rhoDrawThis`
  `rhoDrawThis`, appended locally to hold the rhoDraw density for calculating draws

- RealVector `rhoMix`
  rhoMix, mixture density

- RealVector `rhoOne`
  rhoOne, original importance density

Additional Inherited Members

14.137.1 Detailed Description

Class for the Gaussian Process-based Importance Sampling method.

The `NonDGPImpSampling` implements a method developed by Keith Dalbey that uses a Gaussian process surrogate in the calculation of the importance density. Specifically, the mean and variance of the GP prediction are used to calculate an expected value that a particular point fails, and that is used as part of the computation of the "draw distribution." The normalization constants and the mixture distribution used are defined in (need to get SAND report).

14.137.2 Constructor & Destructor Documentation

**NonDGPImpSampling ( ProblemDescDB & problem_db, Model & model )**

standard constructor

This constructor is called for a standard letter-envelope iterator instantiation. In this case, set_db_list_nodes has been called and probDescDB can be queried for settings from the method specification.

References Response::active_set(), Iterator::assign_rep(), Model::assign_rep(), NonD::construct_lhs(), Model::current_response(), ProblemDesc::get_bool(), ProblemDescDB::get_int(), ProblemDescDB::get_string(), ProblemDescDB::get_ushort(), NonDGPImpSampling::gpBuild, NonDGPImpSampling::gpEval, NonDGPImpSampling::gpModel, Model::gradient_type(), Model::hessian_type(), Iterator::iteratedModel, Iterator::maxIterations, NonDGPImpSampling::numEmulEval, NonDGPImpSampling::numPtsAdd, NonDSampling::numSamples, Iterator::outputLevel, Iterator::probDesc, NonDSampling::randomSeed, ActiveSet::request_values(), NonDSampling::rngName, NonDGPImpSampling::sampleRhoOne, NonDSampling::sampleType, NonDSampling::samplingVarsMode, NonDSampling::statsFlag, NonDSampling::vary_pattern(), and NonDSampling::varyPattern.

14.137.3 Member Function Documentation

**void core_run ( ) [virtual]**

perform the GP importance sampling and return probability of failure

Calculate the failure probabilities for specified probability levels using Gaussian process based importance sampling.

Reimplemented from `Iterator`.
References Model::acv(), Iterator::all_responses(), Analyzer::all_samples(), Model::all_samples(), Model::append_approximation(), Model::approximation_data(), Model::approximation_variances(), Model::build_approximation(), NonDGPImpSampling::calcExpIndicator(), NonDGPImpSampling::calcExpIndPoint(), NonDGPImpSampling::calcRhoDraw(), NonD::cdfFlag, NonD::computedProbLevels, Model::continuous_lower_bounds(), Model::continuous_upper_bounds(), Model::continuous_variables(), Model::current_response(), Model::current_variables(), NonDGPImpSampling::drawNewX(), Model::evaluate(), Model::evaluate_id(), NonDGPImpSampling::expIndicator, NonDGPImpSampling::expIndThis, NonDGPImpSampling::finalProb, Response::function_values(), NonDGPImpSampling::gpCvars, NonDGPImpSampling::gpEval, NonDGPImpSampling::gpMeans, NonDGPImpSampling::gpModel, NonDGPImpSampling::gpVar, NonDGPImpSampling::indicator, NonD::initialize_level_mappings(), Iterator::iteratedModel, Iterator::methodPCIter, NonD::miPLIndex, NonDGPImpSampling::normConst, NonDGPImpSampling::numEmulEval, Analyzer::numFunctions, NonDGPImpSampling::numPtsAdd, NonDGPImpSampling::numPtsTotal, NonDSampling::numSamples, Iterator::outputLevel, Model::pop_approximation(), NonD::requestedRespLevels, NonDGPImpSampling::rhoDraw, NonDGPImpSampling::rhoDrawThis, NonDGPImpSampling::rhoMix, NonDGPImpSampling::rhoOne, Iterator::run(), NonDGPImpSampling::sampleRhoOne, and NonDGPImpSampling::xDrawThis.

The documentation for this class was generated from the following files:

- NonDGPImpSampling.hpp
- NonDGPImpSampling.cpp

### 14.138 NonDGPMsABayesCalibration Class Reference

Generates posterior distribution on model parameters given experiment data.

Inheritance diagram for NonDGPMsABayesCalibration:

```
  NonDGPMsABayesCalibration
   `-- NonDBayesCalibration
        `-- NonDQUESTOBayesCalibration
```

#### Public Member Functions

- **NonDGPMsABayesCalibration (ProblemDescDB &problem_db, Model &model)**
  *constructor*

- **~NonDGPMsABayesCalibration ()**
  *destructor*
Protected Member Functions

- `void derived_init_communicators (ParLevLIter pl_iter)`
  derived class contributions to initializing the communicators associated with this `Iterator` instance
- `void derived_set_communicators (ParLevLIter pl_iter)`
  derived class contributions to setting the communicators associated with this `Iterator` instance
- `void derived_free_communicators (ParLevLIter pl_iter)`
  derived class contributions to freeing the communicators associated with this `Iterator` instance
- `void calibrate ()`
  performs a forward uncertainty propagation by using GPM/SA to generate a posterior distribution on parameters given a set of simulation parameter/response data, a set of experimental data, and additional variables to be specified here.
- `void init_queso_solver ()`
  specialization to initialize the inverse problem and posterior
- `void acquire_simulation_data (RealMatrix &sim_data)`
  Populate simulation data (run design of experiments or load build data)
- `void overlay_proposal_covariance (QUESO::GslMatrix &full_prop_cov) const`
  fill the full proposal covariance, including hyperparameter entries with user-specified or default theta covariance information
- `void fill_simulation_data ()`
  populate the simulation data, calculating and applying scaling if needed
- `void fill_experiment_data ()`
  populate the experiment data, applying scaling if needed
- `void overlay_initial_params (QUESO::GslVector &full_param_initials)`
  overlay the Dakota user’s initial parameters on the full GPMSA vector of parameters
- `void cache_acceptance_chain ()`
  retrieve the chain from QUESO and populate acceptanceChain / acceptedFnVals
- `void print_results (std::ostream &s, short results_state=FINAL_RESULTS)`
  print the final iterator results

Protected Attributes

- `int buildSamples`
  number of samples of the simulation to construct the GP
- `String approxImportFile`
  name of file from which to import build points to build GP
- `unsigned short approxImportFormat`
  build data import tabular format
- `bool approxImportActiveOnly`
  import active variables only
- `unsigned int userConfigVars`
  number of user-specified configuration (scenario) vars
- `unsigned int gpmsaConfigVars`
  number of config vars presented to GPMSA (minimum 1)
- `bool gpmsaNormalize`
whether to apply GPMSA-internal variable and data normalization

- boost::shared_ptr<QUESO::VectorSpace>
  - boost::shared_ptr<QUESO::GslMatrix> > configSpace
    vector space defining the scenario (configuration) variables

- boost::shared_ptr<QUESO::VectorSpace>
  - boost::shared_ptr<QUESO::GslMatrix> > nEtaSpace
    vector space defining the output (response) space for the simulations

- boost::shared_ptr<QUESO::VectorSpace>
  - boost::shared_ptr<QUESO::GslMatrix> > experimentSpace
    vector space defining the output (response) space for the experiments

- boost::shared_ptr<QUESO::GPMSAOptions> > gpmsaOptions
  Configuration options for the GPMSA solver.

- boost::shared_ptr<QUESO::GPMSAFactory>
  - boost::shared_ptr<QUESO::GslMatrix> > gpmsaFactory
    core factory that manages a GP-based likelihood

Private Attributes

- Iterator lhsIter
  LHS iterator for generating samples for GP.

Static Private Attributes

- static NonDGPMSABayesCalibration* nonDGPMSAInstance
  Pointer to current class instance for use in static callback functions.

Additional Inherited Members

14.138.1 Detailed Description

Generates posterior distribution on model parameters given experiment data.

This class provides a wrapper for the functionality provided in the Los Alamos National Laboratory code called GPM/SA (Gaussian Process Models for Simulation Analysis). Although this is a code that provides input/output mapping, it DOES NOT provide the mapping that we usually think of in the NonDeterministic class hierarchy in DAKOTA, where uncertainty in parameter inputs are mapped to uncertainty in simulation responses. Instead, this class takes a pre-existing set of simulation data as well as experimental data, and maps priors on input parameters to posterior distributions on those input parameters, according to a likelihood function. The goal of the MCMC sampling is to produce posterior values of parameter estimates which will produce simulation response values that "match well" to the experimental data. The MCMC is an integral part of the calibration. The data
structures in GPM/SA are fairly detailed and nested. Part of this prototyping exercise is to determine what data structures need to be specified and initialized in DAKOTA and sent to GPM/SA, and what data structures will be returned.

14.138.2 Constructor & Destructor Documentation

NonDGPMSABayesCalibration (ProblemDescDB & problem_db, Model & model) constructor

This constructor is called for a standard letter-envelope iterator instantiation. In this case, set_db_list_nodes has been called and probDescDB can be queried for settings from the method specification. References Dakota::abort_handler(), NonDGPM-SABayesCalibration:approxImportActiveOnly, NonDGPM-SABayesCalibration::approxImportFile, Iterator::assign_rep(), NonDGPM-SABayesCalibration::buildSamples, Model::current_response(), NonDBayesCalibration::emulatorType, NonDCalibration::expData, ProblemDescDB::get_string(), NonDGPM-SABayesCalibration::lhsIter, NonDBayesCalibration::mcmcModel, ExperimentData::num_experiments(), SharedResponseData::num_field_response_groups(), Iterator::outputLevel, Iterator::probDescDB, NonDBayesCalibration::randomSeed, Response::shared_data(), and NonDGPM-SABayesCalibration::userConfigVars.

14.138.3 Member Function Documentation

void calibrate( ) [protected], [virtual]

performs a forward uncertainty propagation by using GPM/SA to generate a posterior distribution on parameters given a set of simulation parameter/response data, a set of experimental data, and additional variables to be specified here.

Perform the uncertainty quantification

Implements NonDBayesCalibration.

References NonDQUESOBayesCalibration::advancedOptionsFile, ExperimentData::all_data(), NonDGPM-SABayesCalibration::buildSamples, NonDGPM-SABayesCalibration::cache_acceptance_chain(), NonDQUESOBayesCalibration::allDataMhOptionsValues, NonDBayesCalibration::compute_statistics(), NonDGPM-SABayesCalibration::configSpace, NonDCalibration::expData, NonDGPM-SABayesCalibration::experimentSpace, NonDGPM-SABayesCalibration::fill_experiment_data(), NonDGPM-SABayesCalibration::fill_simulation_data(), NonDGPM-SABayesCalibration::gpmsaConfigVars, NonDGPM-SABayesCalibration::gpmsaFactory, NonDGPM-SABayesCalibration::gpmsaNormalize, NonDGPM-SABayesCalibration::gpmsaOptions, NonDQUESOBayesCalibration::init_parameter_domain(), NonDGPM-SABayesCalibration::init_queso_solver(), NonDBayesCalibration::initialize_model(), NonDQUESOBayesCalibration::inverseProb, NonDQUESOBayesCalibration::mcmcType, NonDGPM-SABayesCalibration::nEtaSpace, NonDGPM-SABayesCalibration::nHyperparams, NonDQUESOBayesCalibration::nonDQUESOInstance, ExperimentData::num_experiments(), Analyzer::numContinuousVars, Analyzer::numFunctions, NonDBayesCalibration::numHyperparams, Iterator::outputLevel, NonDGPM-SABayesCalibration::overlay_initial_params(), NonDGPM-SABayesCalibration::overlay_proposal_covariance(), NonDQUESOBayesCalibration::paramSpace, NonDQUESOBayesCalibration::priorRv, NonDQUESOBayesCalibration::quesoEnv, and NonDBayesCalibration::standardizedSpace.

void fill_simulation_data( ) [protected]

populate the simulation data, calculating and applying scaling if needed

simulation data, one row per simulation build sample, columns for calibration variables, configuration variables, function values (duplicates storage, but unifies import vs. DOE cases)
References NonDGPMSABayesCalibration::acquire_simulation_data(), NonDGPMSABayesCalibration::buildSamples, NonDGPMSABayesCalibration::configSpace, NonDGPMSABayesCalibration::gpmsaConfigVars, NonDGPMSABayesCalibration::gpmsaFactory, NonDGPMSABayesCalibration::nEtaSpace, Analyzer::numContinuousVars, Analyzer::numFunctions, NonDQUESOBayesCalibration::paramSpace, and NonDGPMSABayesCalibration::userConfigVars.

Referenced by NonDGPMSABayesCalibration::calibrate().

```cpp
void cache_acceptance_chain() [protected]
```

retrieve the chain from QUESO and populate acceptanceChain / acceptedFnVals

This is a subset of the base class retrieval, but we can’t do the fn value lookups. Eventually should be able to retrieve them from GPMSA.

References NonDBayesCalibration::acceptanceChain, NonDBayesCalibration::acceptedFnVals, NonDBayesCalibration::chainSamples, NonDQUESOBayesCalibration::copy_gsl_partial(), NonDQUESOBayesCalibration::inverseProb, NonDBayesCalibration::mcmcModel, Analyzer::numContinuousVars, Analyzer::numFunctions, NonDBayesCalibration::numHyperparams, Iterator::outputLevel, NonDQUESOBayesCalibration::postRv, Model::probability_transformation(), and NonDBayesCalibration::standardizedSpace.

Referenced by NonDGPMSABayesCalibration::calibrate().

```cpp
void print_results(std::ostream &s, short results_state = FINAL_RESULTS) [protected], [virtual]
```

print the final iterator results

This virtual function provides additional iterator-specific final results outputs beyond the function evaluation summary printed in finalize_run().

Reimplemented from NonDBayesCalibration.

References NonDBayesCalibration::print_results().

The documentation for this class was generated from the following files:

- NonDGPMSABayesCalibration.hpp
- NonDGPMSABayesCalibration.cpp

### 14.139 NonDIntegration Class Reference

Derived nondeterministic class that generates N-dimensional numerical integration points for evaluation of expectation integrals.

Inheritance diagram for NonDIntegration:
Public Member Functions

- virtual void initialize_grid (const std::vector<Pecos::BasisPolynomial> &poly_basis)=0
  initialize integration grid by drawing from polynomial basis settings
- virtual void increment_grid ()=0
  increment SSG level/TPQ order
- virtual void increment_grid_preference (const RealVector &dim_pref)
  increment SSG level/TPQ order and update anisotropy
- virtual void increment_grid_preference ()
  increment SSG level/TPQ order and preserve anisotropy
- virtual void increment_grid_weights (const RealVector &aniso_wts)
  increment SSG level/TPQ order and update anisotropy
- virtual void increment_grid_weights ()
  increment SSG level/TPQ order and preserve anisotropy
- virtual void decrement_grid ()=0
  decrement SSG level/TPQ order
- virtual void evaluate_grid_increment ()
  computes a grid increment and evaluates the new parameter sets
- virtual void push_grid_increment ()
  restores a previously computed grid increment (no new evaluations)
- virtual void pop_grid_increment ()
  removes a previously computed grid increment
- virtual void merge_grid_increment ()
  merges a grid increment into the reference grid
- virtual void update_reference ()
  update reference grid within adaptive grid refinement procedures
- const std::vector<Pecos::BasisPolynomial> &polynomial_basis () const
  return IntegrationDriver::polynomialBasis
- std::vector<Pecos::BasisPolynomial> &polynomial_basis ()
return IntegrationDriver::polynomialBasis
• const Pecos::IntegrationDriver & driver () const
  return numIntDriver
• bool resize ()
  reinitializes iterator based on new variable size

Static Public Member Functions

• static void dimension_preference_to_anisotropic_order (unsigned short scalar_order_spec, const RealVector &dim_pref_spec, size_t num_v, UShortArray &aniso_order)
  convert scalar_order_spec and vector dim_pref_spec to vector aniso_order
• static void anisotropic_order_to_dimension_preference (const UShortArray &aniso_order, unsigned short &scalar_order, RealVector &dim_pref)
  convert vector aniso_order to scalar_order and vector dim_pref

Protected Member Functions

• NonDIntegration (ProblemDescDB &problem_db, Model &model)
  constructor
• NonDIntegration (unsigned short method_name, Model &model)
  alternate constructor for instantiations "on the fly"
• NonDIntegration (unsigned short method_name, Model &model, const RealVector &dim_pref)
  alternate constructor for instantiations "on the fly"
• ~NonDIntegration ()
  destructor
• void core_run ()
  core portion of run; implemented by all derived classes and may include pre/post steps in lieu of separate pre/post
• void print_points_weights (const String &tabular_name)
  output integration points and weights to a tabular file

Protected Attributes

• Pecos::IntegrationDriver numIntDriver
  Pecos utility class for managing interface to tensor-product grids and VPISparseGrid utilities for Smolyak sparse grids and cubature.
• size_t numIntegrations
  counter for number of integration executions for this object
• RealVector dimPrefSpec
  the user specification for anisotropic dimension preference

Additional Inherited Members

14.139.1 Detailed Description

Derived nondeterministic class that generates N-dimensional numerical integration points for evaluation of expectation integrals.
This class provides a base class for shared code among NonDQuadrature and NonDSparseGrid.
14.139.2 Constructor & Destructor Documentation

NonDIntegration ( ProblemDescDB & problem_db, Model & model ) [protected]

constructor

This constructor is called for a standard letter-envelope iterator instantiation. In this case, set_db_list_nodes has been called and probDescDB can be queried for settings from the method specification. It is not currently used, as there are not yet separate nond_quadrature/nond_sparse_grid method specifications.

References NonD::initialize_final_statistics().

NonDIntegration ( unsigned short method_name, Model & model ) [protected]

alternate constructor for instantiations "on the fly"

This alternate constructor is used for on-the-fly generation and evaluation of numerical integration points.

NonDIntegration ( unsigned short method_name, Model & model, const RealVector & dim_pref ) [protected]

alternate constructor for instantiations "on the fly"

This alternate constructor is used for on-the-fly generation and evaluation of numerical integration points.

14.139.3 Member Function Documentation

void dimension_preference_to_anisotropic_order ( unsigned short scalar_order_spec, const RealVector & dim_pref_spec, size_t num_v, UShortArray & aniso_order ) [static]

convert scalar_order_spec and vector dim_pref_spec to vector aniso_order

Converts a scalar order specification and a vector anisotropic dimension preference into an anisotropic order vector. It is used for initialization and does not enforce a reference lower bound (see also NonDQuadrature::update_anisotropic_order()).

Referenced by NonDExpansion::configure_expansion_orders(), NonDQuadrature::initialize_dimension_quadrature_order(), and SharedC3ApproxData::SharedC3ApproxData().

void anisotropic_order_to_dimension_preference ( const UShortArray & aniso_order, unsigned short & scalar_order, RealVector & dim_pref ) [static]

convert vector aniso_order to scalar_order and vector dim_pref

Converts a vector anisotropic order into a scalar order and vector anisotropic dimension preference.

Referenced by NonDC3FunctionTrain::config_regression(), and NonDPolynomialChaos::config_regression().

void core_run ( ) [protected], [virtual]

core portion of run; implemented by all derived classes and may include pre/post steps in lieu of separate pre/post Virtual run function for the iterator class hierarchy. All derived classes need to redefine it.

Reimplemented from Iterator.

References Analyzer::evaluate_parameter_sets(), Analyzer::get_parameter_sets(), Iterator::iteratedModel, and NonDIntegration::numIntegrations.
**void print_points_weights(const String & tabular_name)** [protected]

output integration points and weights to a tabular file

Virtual function called from probDescDB-based constructors and from NonDIntegration::core_run()

**void NonDIntegration:: check_variables(const std::vector<Pecos::RandomVariable>& x_ran_vars)**

```cpp
{ base class
default definition of virtual function
bool err_flag = false;
numContDesVars = numContIntervalVars = numContStateVars = 0; size_t i, num_v = x_ran_vars.size(); short x_type; for (i=0; i<num_v; ++i) { x_type = x_ran_vars[i].type(); if (x_type == Pecos::CONTINUOUS_DESIGN) ++numContDesVars; else if (x_type == Pecos::CONTINUOUS_INTERVAL) ++numContIntervalVars; else if (x_type == Pecos::CONTINUOUS_STATE) ++numContStateVars; } if (x_ran_vars.size() != numContinuousVars || numContEpistUncVars != numContIntervalVars || numContinuous-Vars != numContDesVars + numContAleatUncVars + numContEpistUncVars + numContStateVars) { Cerr << "Error: mismatch in active variable counts in NonDIntegration::" << "check variables()." << std::endl; err_flag = true; }
if (err_flag) abort_handler(-1); }
```

References Analyzer::allSamples, Model::continuous_variable_labels(), Iterator::iteratedModel, NonDIntegration::numIntDriver, and Dakota::write_precision.

Referenced by NonDCubature::get_parameter_sets(), NonDQuadrature::get_parameter_sets(), and NonDSparse-Grid::get_parameter_sets().

The documentation for this class was generated from the following files:
- NonDIntegration.hpp
- NonDIntegration.cpp

## 14.140 NonDInterval Class Reference

Base class for interval-based methods within DAKOTA/UQ.

Inheritance diagram for NonDInterval:

```
NonDInterval
  |     
  |     
Iterator   Analyst
           NonD
            NonDInterval
           |     |
NonDGlobalInterval  NonDLHSInterval  NonDLocalInterval
            |     |
NonDGlobalEvidence  NonDGlobalSingleInterval  NonDLocalEvidence
                  |     |
NonDLHSEvidence    NonDLHSInterval      NonDLocalSingleInterval
```

### Public Member Functions

- **NonDInterval (ProblemDescDB &problem_db, Model &model)**

  constructor

- **~NonDInterval ()**

  destructor

- **void print_results (std::ostream &s, short results_state=FINAL_RESULT)**

  print the cumulative distribution functions for belief and plausibility
Protected Member Functions

- void initialize_final_statistics ()
  initialize finalStatistics for belief/plausibility results sets
- void compute_evidence_statistics ()
  method for computing belief and plausibility values for response levels or vice-versa
- void calculate_cells_and_bpas ()
  computes the interval combinations (cells) and their bpas replaces CBPIIC_F77 from wrapper calculate_basic_prob_intervals()
- void calculate_cbf_cpf (bool complementary=true)
  function to compute (complementary) distribution functions on belief and plausibility replaces CCBFPF_F77 from wrapper calculate_cum_belief_plaus()

Protected Attributes

- size_t numContIntervalVars
  number of variables of type CONTINUOUS_INTERVAL_UNCERTAIN
- size_t numDiscIntervalVars
  number of variables of type DISCRETE_INTERVAL_UNCERTAIN
- size_t numDiscSetIntUncVars
  number of variables of type DISCRETE_UNCERTAIN_SET_INT
- size_t numDiscSetRealUncVars
  number of variables of type DISCRETE_UNCERTAIN_SET_REAL
- bool singleIntervalFlag
  flag for SingleInterval derived class
- RealVectorArray ccBelFn
  Storage array to hold CCBF values.
- RealVectorArray ccPlausFn
  Storage array to hold CCPF values.
- RealVectorArray ccBelVal
  Storage array to hold CCB response values.
- RealVectorArray ccPlausVal
  Storage array to hold CCP response values.
- RealVectorArray cellContLowerBounds
  Storage array to hold cell lower bounds for continuous variables.
- RealVectorArray cellContUpperBounds
  Storage array to hold cell upper bounds for continuous variables.
- IntVectorArray cellIntRangeLowerBounds
  Storage array to hold cell lower bounds for discrete int range variables.
- IntVectorArray cellIntRangeUpperBounds
  Storage array to hold cell upper bounds for discrete int range variables.
• IntVectorArray `cellIntSetBounds`  
  Storage array to hold cell values for discrete integer set variables.

• IntVectorArray `cellRealSetBounds`  
  Storage array to hold cell value for discrete real set variables.

• RealVectorArray `cellFnLowerBounds`  
  Storage array to hold cell min.

• RealVectorArray `cellFnUpperBounds`  
  Storage array to hold cell max.

• RealVector `cellBPA`  
  Storage array to hold cell bpa.

• `size_t respFnCntr`  
  response function counter

• `size_t cellCntr`  
  cell counter

• `size_t numCells`  
  total number of interval combinations

**Additional Inherited Members**

**14.140.1 Detailed Description**

Base class for interval-based methods within DAKOTA/UQ.

The NonDInterval class implements the propagation of epistemic uncertainty using either pure interval propagation or Dempster-Shafer theory of evidence. In the latter approach, one assigns a set of basic probability assignments (BPA) to intervals defined for the uncertain variables. Input interval combinations are calculated, along with their BPA. Currently, the response function is evaluated at a set of sample points, then a response surface is constructed which is sampled extensively to find the minimum and maximum within each input interval cell, corresponding to the belief and plausibility within that cell, respectively. This data is then aggregated to calculate cumulative distribution functions for belief and plausibility.

The documentation for this class was generated from the following files:

• NonDInterval.hpp
• NonDInterval.cpp

**14.141 NonDLHSEvidence Class Reference**

Class for the Dempster-Shafer Evidence Theory methods within DAKOTA/UQ.

Inheritance diagram for NonDLHSEvidence:
Public Member Functions

- **NonDLHSEvidence** (ProblemDescDB & problem_db, Model & model)
  - *constructor*
- ~**NonDLHSEvidence** ()
  - *destructor*
- void **initialize** ()
  - *perform any required initialization*
- void **post_process_samples** ()
  - *post-process the output from executing lhsSampler*

Additional Inherited Members

14.141.1 Detailed Description

Class for the Dempster-Shafer Evidence Theory methods within DAKOTA/UQ.

The NonDEvidence class implements the propagation of epistemic uncertainty using Dempster-Shafer theory of evidence. In this approach, one assigns a set of basic probability assignments (BPA) to intervals defined for the uncertain variables. Input interval combinations are calculated, along with their BPA. Currently, the response function is evaluated at a set of sample points, then a response surface is constructed which is sampled extensively to find the minimum and maximum within each input interval cell, corresponding to the belief and plausibility within that cell, respectively. This data is then aggregated to calculate cumulative distribution functions for belief and plausibility.

The documentation for this class was generated from the following files:

- NonDLHSEvidence.hpp
- NonDLHSEvidence.cpp

14.142 NonDLHSInterval Class Reference

Class for the LHS-based interval methods within DAKOTA/UQ.

Inheritance diagram for NonDLHSInterval:
CHAPTER 14. CLASS DOCUMENTATION

```
NonDLHSInterval
NonDInterval
NonD
Analyzer
Iterator

Public Member Functions

• NonDLHSInterval (ProblemDescDB &problem_db, Model &model)
  constructor

• ~NonDLHSInterval ()
  destructor

• void derived_init_communicators (ParLevLIter pl_iter)
  derived class contributions to initializing the communicators associated with this Iterator instance

• void derived_set_communicators (ParLevLIter pl_iter)
  derived class contributions to setting the communicators associated with this Iterator instance

• void derived_free_communicators (ParLevLIter pl_iter)
  derived class contributions to freeing the communicators associated with this Iterator instance

• void core_run ()
  performs an epistemic uncertainty propagation using LHS samples

Protected Member Functions

• virtual void initialize ()
  perform any required initialization

• virtual void post_process_samples ()=0
  post-process the output from executing lhsSampler

Protected Attributes

• Iterator lhsSampler
  the LHS sampler instance

• const int seedSpec
  the user seed specification (default is 0)

• int numSamples
  the number of samples used
```
14.143. NONDLHSSAMPLING CLASS REFERENCE

- String **rngName**
  
  *name of the random number generator*

Additional Inherited Members

14.142.1 Detailed Description

Class for the LHS-based interval methods within DAKOTA/UQ.

The **NonDLHSInterval** class implements the propagation of epistemic uncertainty using LHS-based methods. The documentation for this class was generated from the following files:

- NonDLHSInterval.hpp
- NonDLHSInterval.cpp

14.143 NonDLHSSampling Class Reference

Performs LHS and Monte Carlo sampling for uncertainty quantification.

Inheritance diagram for NonDLHSSampling:

```
NonDLHSSampling
  NonDSampling
    NonD
      NonDSampling
        NonDLHSSampling
```

Public Member Functions

- **NonDLHSSampling** (ProblemDescDB &problem_db, Model &model)
  
  *standard constructor*

- **NonDLHSSampling** (Model &model, unsigned short sample_type, int samples, int seed, const String &rng, bool vary_pattern=true, short sampling_vars_mode=ACTIVE)

  *alternate constructor for sample generation and evaluation "on the fly"*

- **NonDLHSSampling** (unsigned short sample_type, int samples, int seed, const String &rng, const RealVector &lower_bnds, const RealVector &upper_bnds)

  *alternate constructor for uniform sample generation "on the fly"*

- **NonDLHSSampling** (unsigned short sample_type, int samples, int seed, const String &rng, const RealVector &means, const RealVector &std_devs, const RealVector &lower_bnds, const RealVector &upper_bnds, RealSymMatrix &correl)

  *alternate constructor for sample generation of correlated normals "on the fly"*

- **~NonDLHSSampling** ()

  *destructor*
Protected Member Functions

- void sampling_increment ()
  increment to next in sequence of refinement samples
- void pre_run ()
  generate LHS samples in non-VBD cases
- void core_run ()
  perform the evaluate parameter sets portion of run
- void post_run (std::ostream &s)
  generate statistics for LHS runs in non-VBD cases
- void post_input ()
  read tabular data for post-run mode
- void update_final_statistics ()
  update finalStatistics and (if MC sampling) finalStatErrors
- void compute_pca (std::ostream &s)
  compute a principal components analysis on the sample set
- void print_results (std::ostream &s, short results_state=FINAL_RESULTS)
  print the final statistics
- void d_optimal_parameter_set (int previous_samples, int new_samples, RealMatrix &full_samples)
  generate a d-optimal parameter set, leaving the first previous_samples columns intact and adding new_samples new columns following them
- void initial_increm_lhs_set (int new_samples, RealMatrix &full_samples, IntMatrix &full_ranks)
  Populate the first new_samples columns of allSamples with an LHS design and update the stored ranks.
- void increm_lhs_parameter_set (int previous_samples, int new_samples, RealMatrix &full_samples, IntMatrix &all_ranks)
  generate a new batch that is Latin w.r.t. the previous samples
- void store_ranks (const RealMatrix &initial_values, const RealMatrix &increm_values)
  store the ranks of the last generated sample for continuous (based on sampleRanks) and calculate/store discrete ranks
- void store_ranks (IntMatrix &full_ranks)
  store the combined ranks from sampleRanks to leading submatrix local cached ranks matrix
- void combine_discrete_ranks (const RealMatrix &initial_values, const RealMatrix &increm_values)
  merge the discrete ranks into a submatrix of sampleRanks
- void print_header_and_statistics (std::ostream &s, const int &num_samples)
  Print a header and summary statistics.
- void archive_results (int num_samples, size_t ind_inc=0)
  Archive all results.

Static Protected Member Functions

- static bool rank_sort (const int &x, const int &y)
  sort algorithm to compute ranks for rank correlations
14.143. NONDLHSSAMPLING CLASS REFERENCE

Private Attributes

- size_t numResponseFunctions
  number of response functions; used to distinguish NonD from opt/NLS usage
- IntVector refineSamples
  list of refinement sample batch sizes
- bool dOptimal
  whether to generate d-optimal point sets
- size_t numCandidateDesigns
  number of candidate designs to generate for classical D-optimal designs
- Real oversampleRatio
  oversampling ratio for Leja D-optimal candidate set generation
- bool varBasedDecompFlag
  flags computation of variance-based decomposition indices
- bool pcaFlag
  flag to specify the calculation of principal components
- Real percentVarianceExplained
  Threshold to keep number of principal components that explain this much variance.

Static Private Attributes

- static RealArray rawData
  static data used by static rank_sort() fn

Additional Inherited Members

14.143.1 Detailed Description

Performs LHS and Monte Carlo sampling for uncertainty quantification.

The Latin Hypercube Sampling (LHS) package from Sandia Albuquerque’s Risk and Reliability organization provides comprehensive capabilities for Monte Carlo and Latin Hypercube sampling within a broad array of user-specified probabilistic parameter distributions. It enforces user-specified rank correlations through use of a mixing routine. The NonDLHSSampling class provides a C++ wrapper for the LHS library and is used for performing forward propagations of parameter uncertainties into response statistics.

Batch generation options, including D-Optimal and incremental LHS are provided.

The incremental LHS sampling capability allows one to supplement an initial sample of size n to size 2n while maintaining the correct stratification of the 2n samples and also maintaining the specified correlation structure. The incremental version of LHS will return a sample of size n, which when combined with the original sample of size n, allows one to double the size of the sample.

14.143.2 Constructor & Destructor Documentation

NonDLHSSampling ( ProblemDescDB & problem_db, Model & model )

standard constructor

This constructor is called for a standard letter-envelope iterator instantiation. In this case, set_db_list_nodes has been called and probDescDB can be queried for settings from the method specification.
References Dakota::abort_handler(), SharedVariablesData::active_components_totals(), Model::current_variables(), NonDLHSSampling::dOptimal, Model::num_primary_fns(), NonDLHSSampling::numCandidateDesigns, Analyzer::numDiscreteIntVars, Analyzer::numDiscreteRealVars, Analyzer::numDiscreteStringVars, NonDLHSSampling::numResponseFunctions, Iterator::outputLevel, NonDLHSSampling::oversampleRatio, Model::primary_fn_type(), NonDLHSSampling::refineSamples, NonDSampling::sampleType, Variables::shared_data(), and Dakota::svd().

NonDLHSSampling ( Model & model, unsigned short sample_type, int samples, int seed, const String & rng, bool vary_pattern = true, short sampling_vars_mode = ACTIVE )

alternate constructor for sample generation and evaluation "on the fly"

This alternate constructor is used for generation and evaluation of Model-based sample sets. A set_db_list_nodes has not been performed so required data must be passed through the constructor. It’s purpose is to avoid the need for a separate LHS specification within methods that use LHS sampling.

NonDLHSSampling ( unsigned short sample_type, int samples, int seed, const String & rng, const RealVector & lower_bnds, const RealVector & upper_bnds )

alternate constructor for uniform sample generation "on the fly"

This alternate constructor is used by ConcurrentStrategy for generation of uniform, uncorrelated sample sets. It is not a letter-envelope instantiation and a set_db_list_nodes has not been performed. It is called with all needed data passed through the constructor and is designed to allow more flexibility in variables set definition (i.e., relax connection to a variables specification and allow sampling over parameter sets such as multiobjective weights). In this case, a Model is not used and the object must only be used for sample generation (no evaluation).

References NonDSampling::get_parameter_sets().

NonDLHSSampling ( unsigned short sample_type, int samples, int seed, const String & rng, const RealVector & means, const RealVector & std_devs, const RealVector & lower_bnds, const RealVector & upper_bnds, RealSymMatrix & correl )

alternate constructor for sample generation of correlated normals "on the fly"

This alternate constructor is used to generate correlated normal sample sets. It is not a letter-envelope instantiation and a set_db_list_nodes has not been performed. It is called with all needed data passed through the constructor. In this case, a Model is not used and the object must only be used for sample generation (no evaluation).

References NonDSampling::get_parameter_sets().

14.143.3 Member Function Documentation

void core_run() [protected], [virtual]

perform the evaluate parameter sets portion of run

Loop over the set of samples and compute responses. Compute statistics on the set of responses if statsFlag is set.

Reimplemented from Iterator.

References NonDSampling::allDataFlag, Analyzer::evaluate_parameter_sets(), Iterator::iteratedModel, NonDLHSSampling::numResponseFunctions, and NonDSampling::statsFlag.
void d_optimal_parameter_set ( int previous_samples, int new_samples, RealMatrix & full_samples )
[protected]

generate a d-optimal parameter set, leaving the first previous_samples columns intact and adding new_samples new columns following them

For now, when this function is called, numSamples is the number of new samples to generate.

References Model::current_variables(), Dakota::det_AtransA(), NonDSampling::get_parameter_sets(), Probability-TransformModel::initialize_distribution_types(), Iterator::iteratedModel, NonDSampling::mode_counts(), Model-::multivariate_distribution(), NonDLHSSampling::numCandidateDesigns, Iterator::outputLevel, NonDLHSSampling-::oversampleRatio, and NonDSampling::transform_samples().

Referenced by NonDLHSSampling::pre_run().

The documentation for this class was generated from the following files:

• NonDLHSSampling.hpp
• NonDLHSSampling.cpp

14.144 NonDLHSSingleInterval Class Reference

Class for pure interval propagation using LHS.

Inheritance diagram for NonDLHSSingleInterval:

```
<table>
<thead>
<tr>
<th>Iterator</th>
</tr>
</thead>
<tbody>
<tr>
<td>Analyzer</td>
</tr>
<tr>
<td>NonD</td>
</tr>
<tr>
<td>NonDInterval</td>
</tr>
<tr>
<td>NonDLHSSInterval</td>
</tr>
<tr>
<td>NonDLHSSingleInterval</td>
</tr>
</tbody>
</table>
```

Public Member Functions

- NonDLHSSingleInterval (ProblemDescDB &problem_db, Model &model)
  - constructor
- ~NonDLHSSingleInterval ()
  - destructor

Protected Member Functions

- void initialize ()
  - perform any required initialization
- void post_process_samples ()
post-process the output from executing lhsSampler

Private Attributes

- size_t statCntr
  counter for finalStatistics

Additional Inherited Members

14.144.1 Detailed Description

Class for pure interval propagation using LHS.

The NonDSingleInterval class implements the propagation of epistemic uncertainty using ...

The documentation for this class was generated from the following files:

- NonDLHSSingleInterval.hpp
- NonDLHSSingleInterval.cpp

14.145 NonDLocalEvidence Class Reference

Class for the Dempster-Shafer Evidence Theory methods within DAKOTA/UQ.

Inheritance diagram for NonDLocalEvidence:

![Inheritance Diagram]

Public Member Functions

- NonDLocalEvidence (ProblemDescDB &problem_db, Model &model)
  constructor
- ~NonDLocalEvidence ()
  destructor
Protected Member Functions

- void initialize ()
  *perform any required initialization*
- void set_cell_bounds ()
  *set the optimization variable bounds for each cell*
- void truncate_to_cell_bounds (RealVector &initial_pt)
  *truncate initial pt to respect current cell lower/upper bounds*
- void post_process_cell_results (bool maximize)
  *post-process a cell minimization/maximization result*
- void post_process_response_fn_results ()
  *post-process the interval computed for a response function*
- void post_process_final_results ()
  *perform final post-processing*

Additional Inherited Members

14.145.1 Detailed Description

Class for the Dempster-Shafer Evidence Theory methods within DAKOTA/UQ.

The NonDEvidence class implements the propagation of epistemic uncertainty using Dempster-Shafer theory of evidence. In this approach, one assigns a set of basic probability assignments (BPA) to intervals defined for the uncertain variables. Input interval combinations are calculated, along with their BPA. Currently, the response function is evaluated at a set of sample points, then a response surface is constructed which is sampled extensively to find the minimum and maximum within each input interval cell, corresponding to the belief and plausibility within that cell, respectively. This data is then aggregated to calculate cumulative distribution functions for belief and plausibility.

The documentation for this class was generated from the following files:

- NonDLocalEvidence.hpp
- NonDLocalEvidence.cpp

14.146 NonDLocalInterval Class Reference

Class for using local gradient-based optimization approaches to calculate interval bounds for epistemic uncertainty quantification.

Inheritance diagram for NonDLocalInterval:
Public Member Functions

- **NonDLocalInterval** (ProblemDescDB &problem_db, Model &model)
  
  *constructor*
- **~NonDLocalInterval** ()
  *destructor*
- void derived_init_communicators (ParLevLIter pl_iter)
  *derived class contributions to initializing the communicators associated with this Iterator instance*
- void derived_set_communicators (ParLevLIter pl_iter)
  *derived class contributions to setting the communicators associated with this Iterator instance*
- void derived_free_communicators (ParLevLIter pl_iter)
  *derived class contributions to freeing the communicators associated with this Iterator instance*
- void core_run ()
  *Performs a gradient-based optimization to determine interval bounds for an entire function or interval bounds on a particular statistical estimator.*
- unsigned short uses_method () const
  *return name of active optimizer method*
- void method_recourse ()
  *perform an MPP optimizer method switch due to a detected conflict*

Protected Member Functions

- virtual void initialize ()
  *perform any required initialization*
- virtual void set_cell_bounds ()
  *set the optimization variable bounds for each cell*
- virtual void truncate_to_cell_bounds (RealVector &initial_pt)
  *truncate initial_pt to respect current cell lower/upper bounds*
- virtual void post_process_cell_results (bool maximize)
  *post-process a cell minimization/maximization result*
virtual void post_process_response_fn_results ()
post-process the interval computed for a response function

virtual void post_process_final_results ()
perform final post-processing

Protected Attributes

- Iterator minMaxOptimizer
  local gradient-based optimizer
- Model minMaxModel
  recast model which extracts the active objective function

Static Private Member Functions

- static void extract_objective (const Variables &sub_model_vars, const Variables &recast_vars, const Response &sub_model_response, Response &recast_response)
  static function used to extract the active objective function when optimizing for an interval lower or upper bound

Private Attributes

- bool npsolFlag
  flag representing the gradient-based optimization algorithm selection (NPSOL SQP or OPT++ NIP)

Static Private Attributes

- static NonDLocalInterval * nondLIInstance
  pointer to the active object instance used within the static evaluator functions in order to avoid the need for static data

Additional Inherited Members

14.146.1 Detailed Description

Class for using local gradient-based optimization approaches to calculate interval bounds for epistemic uncertainty quantification.

The NonDLocalInterval class supports local gradient-based optimization approaches to determining interval bounds for epistemic UQ. The interval bounds may be on the entire function in the case of pure interval analysis (e.g. intervals on input = intervals on output), or the intervals may be on statistics of an "inner loop" aleatory analysis such as intervals on means, variances, or percentile levels.

The documentation for this class was generated from the following files:

- NonDLocalInterval.hpp
- NonDLocalInterval.cpp
14.147 NonDLocalReliability Class Reference

Class for the reliability methods within DAKOTA/UQ.

Inheritance diagram for NonDLocalReliability:

```
  public:
  NonDLocalReliability (ProblemDescDB &problem_db, Model &model)
  ~NonDLocalReliability ()
  void derived_init.communicators (ParLevLIter pl_iter)
  void derived_set.communicators (ParLevLIter pl_iter)
  void derived_free.communicators (ParLevLIter pl_iter)
  void pre_run ()
  void core_run ()
  void print_results (std::ostream &s, short results_state=FINAL_RESULTS)
  unsigned short uses_method () const
  void method_recourse ()
```

Public Member Functions

- NonDLocalReliability (ProblemDescDB &problem_db, Model &model)
  
  constructor
- ~NonDLocalReliability ()
  
  destructor
- void derived_init.communicators (ParLevLIter pl_iter)
  
  derived class contributions to initializing the communicators associated with this Iterator instance
- void derived_set.communicators (ParLevLIter pl_iter)
  
  derived class contributions to setting the communicators associated with this Iterator instance
- void derived_free.communicators (ParLevLIter pl_iter)
  
  derived class contributions to freeing the communicators associated with this Iterator instance
- void pre_run ()
  
  pre-run portion of run (optional); re-implemented by Iterators which can generate all Variables (parameter sets) a priori
- void core_run ()
  
  core portion of run; implemented by all derived classes and may include pre/post steps in lieu of separate pre/post
- void print_results (std::ostream &s, short results_state=FINAL_RESULTS)
  
  print the final iterator results
- unsigned short uses_method () const
  
  return name of active MPP optimizer
- void method_recourse ()
  
  perform an MPP optimizer method switch due to a detected conflict
Private Member Functions

- void initial_taylor_series()
  convenience function for performing the initial limit state Taylor-series approximation
- void mean_value()
  convenience function for encapsulating the simple Mean Value computation of approximate statistics and importance factors
- void mpp_search()
  convenience function for encapsulating the reliability methods that employ a search for the most probable point (AMV, AMV+, FORM, SORM)
- void initialize_class_data()
  convenience function for initializing class scope arrays
- void initialize_level_data()
  convenience function for initializing/warm starting MPP search data for each response function prior to level 0
- void initialize_mpp_search_data()
  convenience function for initializing/warm starting MPP search data for each z/p/beta level for each response function
- void update_mpp_search_data(const Variables &vars_star, const Response &resp_star)
  convenience function for updating MPP search data for each z/p/beta level for each response function
- void update_level_data()
  convenience function for updating z/p/beta level data and final statistics following MPP convergence
- void update_pma_maximize(const RealVector &mpp_u, const RealVector &fn_grad_u, const RealSymMatrix &fn_hess_u)
  update pmaMaximizeG from prescribed probabilities or prescribed generalized reliabilities by inverting second-order integrations
- void update_limit_state_surrogate()
  convenience function for passing the latest variables/response data to the data fit embedded within uSpaceModel
- void assign_mean_data()
  update mostProbPointX/U, computedrespLevel, fnGradX/U, and fnHessX/U from ranVarMeansX/U, fnValsMeanX, fnGradsMeanX, fnHessiansMeanX
- void dg_ds_eval(const RealVector &x_vars, const RealVector &fn_grad_x, RealVector &final_stat_grad)
  convenience function for evaluating dg/ds
- Real dp2_dbeta_factor(Real beta, bool cdf_flag)
  compute factor for derivative of second-order probability with respect to reliability index (from differentiating BR-EITUNG or HOHENRACK expressions)
- void truth_evaluation(short mode)
  perform an evaluation of the actual model and store value.grad.Hessian data in X.U spaces
- Real signed_norm(const RealVector &mpp_u, const RealVector &fn_grad_u, bool cdf_flag)
  convert norm of mpp_u (u-space solution) to a signed reliability index
- Real signed_norm(Real norm_mpp_u)
  convert norm of u-space vector to a signed reliability index
- Real signed_norm(Real norm_mpp_u, const RealVector &mpp_u, const RealVector &fn_grad_u, bool cdf_flag)
  shared helper function
- Real probability(Real beta)
Convert reliability to probability using a first-order integration.

- **Real probability** (bool cdf_flag, const RealVector &mpp_u, const RealVector &fn_grad_u, const RealSymMatrix &fn_hess_u)

Convert computed reliability to probability using either a first-order or second-order integration.

- **Real probability** (Real beta, bool cdf_flag, const RealVector &mpp_u, const RealVector &fn_grad_u, const RealSymMatrix &fn_hess_u)

Convert provided reliability to probability using either a first-order or second-order integration.

- **Real reliability** (Real p)

Convert probability to reliability using the inverse of a first-order integration.

- **Real reliability** (Real p, bool cdf_flag, const RealVector &mpp_u, const RealVector &fn_grad_u, const RealSymMatrix &fn_hess_u)

- **bool reliability_residual** (const Real &p, const Real &beta, const RealVector &kappa, Real &res)
  - compute the residual for inversion of second-order probability corrections using Newton’s method (called by reliability(p))

- **Real reliability_residual_derivative** (const Real &p, const Real &beta, const RealVector &kappa)
  - compute the residual derivative for inversion of second-order probability corrections using Newton’s method (called by reliability(p))

- **void principal_curvatures** (const RealVector &mpp_u, const RealVector &fn_grad_u, const RealSymMatrix &fn_hess_u, RealVector &kappa_u)
  - Compute the kappaU vector of principal curvatures from fnHessU.

- **void scale_curvature** (Real beta, bool cdf_flag, const RealVector &kappa, RealVector &scaled_kappa)
  - scale copy of principal curvatures by -1 if needed; else take a view

### Static Private Member Functions

- **static void RIA_objective_eval** (const Variables &sub_model_vars, const Variables &recast_vars, const Response &sub_model_response, Response &recast_response)
  - static function used as the objective function in the Reliability Index Approach (RIA) problem formulation. This equality-constrained optimization problem performs the search for the most probable point (MPP) with the objective function of \((\text{norm } u)^2\).

- **static void RIA_constraint_eval** (const Variables &sub_model_vars, const Variables &recast_vars, const Response &sub_model_response, Response &recast_response)
  - static function used as the constraint function in the Reliability Index Approach (RIA) problem formulation. This equality-constrained optimization problem performs the search for the most probable point (MPP) with the constraint of \(G(u) = \text{response level}\).

- **static void PMA_objective_eval** (const Variables &sub_model_vars, const Variables &recast_vars, const Response &sub_model_response, Response &recast_response)
  - static function used as the objective function in the Performance Measure Approach (PMA) problem formulation. This equality-constrained optimization problem performs the search for the most probable point (MPP) with the objective function of \(G(u)\).

- **static void PMA_constraint_eval** (const Variables &sub_model_vars, const Variables &recast_vars, const Response &sub_model_response, Response &recast_response)
  - static function used as the constraint function in the first-order Performance Measure Approach (PMA) problem formulation. This optimization problem performs the search for the most probable point (MPP) with the equality constraint of \((\text{norm } u)^2 = (\text{beta-bar})^2\).
• static void `PMA2_constraint_eval` (const Variables &sub_model_vars, const Variables &recast_vars, const Response &sub_model_response, Response &recast_response)
  static function used as the constraint function in the second-order Performance Measure Approach (PMA) problem formulation. This optimization problem performs the search for the most probable point (MPP) with the equality constraint of $\beta^* = \beta^* - \bar{\beta}$.

• static void `PMA2_set_mapping` (const Variables &recast_vars, const ActiveSet &recast_set, ActiveSet &sub_model_set)
  static function used to augment the sub-model ASV requests for second-order PMA

**Private Attributes**

- Real `computedRespLevel`
  output response level calculated
- Real `computedRelLevel`
  output reliability level calculated for RIA and 1st-order PMA
- Real `computedGenRelLevel`
  output generalized reliability level calculated for 2nd-order PMA
- RealVector `fnGradX`
  actual x-space gradient for current function from most recent response evaluation
- RealVector `fnGradU`
  u-space gradient for current function updated from fnGradX and Jacobian $\frac{dx}{du}$
- RealSymMatrix `fnHessX`
  actual x-space Hessian for current function from most recent response evaluation
- RealSymMatrix `fnHessU`
  u-space Hessian for current function updated from fnHessX and Jacobian $\frac{dx}{du}$
- RealVector `kappaU`
  principal curvatures derived from eigenvalues of orthonormal transformation of fnHessU
- RealVector `fnValsMeanX`
  response function values evaluated at mean $x$
- RealMatrix `fnGradsMeanX`
  response function gradients evaluated at mean $x$
- RealSymMatrixArray `fnHessiansMeanX`
  response function Hessians evaluated at mean $x$
- RealVector `ranVarMeansX`
  vector of means for all uncertain random variables in x-space
- RealVector `ranVarStdDevsX`
  vector of std deviations for all uncertain random variables in x-space
- RealVector `ranVarMeansU`
  vector of means for all uncertain random variables in u-space
- bool `initialPtUserSpec`
  flag indicating user specification of (any portion of) initialPtU
- RealVector `initialPtUSpec`
  user specification or default initial guess for local optimization
- RealVector `initialPtU`
CHAPTER 14. CLASS DOCUMENTATION

- **RealVector** mostProbPointX
  location of MPP in x-space
- **RealVector** mostProbPointU
  location of MPP in u-space
- **RealVectorArray** prevMPPULev0
  array of converged MPP’s in u-space for level 0. Used for warm-starting initialPtU within RBDO.
- **RealMatrix** prevFnGradDLev0
  matrix of limit state sensitivities w.r.t. inactive/design variables for level 0. Used for warm-starting initialPtU within RBDO.
- **RealMatrix** prevFnGradULev0
  matrix of limit state sensitivities w.r.t. active/uncertain variables for level 0. Used for warm-starting initialPtU within RBDO.
- **RealVector** prevICVars
  previous design vector. Used for warm-starting initialPtU within RBDO.
- **ShortArray** prevCumASVLev0
  accumulation (using \(|\) of all previous design ASV’s from requested finalStatistics. Used to detect availability of prevFnGradDLev0 data for warm-starting initialPtU within RBDO.
- **bool** npsolFlag
  flag representing the optimization MPP search algorithm selection (NPSOL SQP or OPT++ NIP)
- **bool** warmStartFlag
  flag indicating the use of warm starts
- **bool** nipModeOverrideFlag
  flag indicating the use of move overrides within OPT++ NIP
- **bool** curvatureDataAvailable
  flag indicating that sufficient data (i.e., fnGradU, fnHessU, mostProbPointU) is available for computing principal curvatures
- **bool** kappaUpdated
  track when kappaU requires updating via principal_curvatures()
- **short** integrationOrder
  integration order (1 or 2) provided by integration specification
- **short** secondOrderIntType
  type of second-order integration: Breitung, Hohenbichler-Rackwitz, or Hong
- **Real** curvatureThresh
  cut-off value for 1/sqrt() term in second-order probability corrections.
- **short** taylorOrder
  order of Taylor series approximations (1 or 2) in MV/AMV/AMV+ derived from hessian type
- **RealMatrix** impFactor
  importance factors predicted by MV
- **int** npsolDerivLevel
  derivative level for NPSOL executions (1 = analytic grads of objective fn, 2 = analytic grads of constraints, 3 = analytic grads of both).
- **unsigned short** warningBits
  set of warnings accumulated during execution
Static Private Attributes

- static NonDLocalReliability * nondLocRelInstance
  
  pointer to the active object instance used within the static evaluator functions in order to avoid the need for static data

Additional Inherited Members

14.147.1 Detailed Description

Class for the reliability methods within DAKOTA/UQ.

The NonDLocalReliability class implements the following reliability methods through the support of different limit state approximation and integration options: mean value (MVFSOM/MVSOSM), advanced mean value method (AMV, AMV^2) in x- or u-space, iterated advanced mean value method (AMV+, AMV^2+) in x- or u-space, two-point adaptive nonlinearity approximation (TANA) in x- or u-space, first order reliability method (FORM), and second order reliability method (SORM). All options except mean value employ an optimizer (currently NPSOL SQP or OPT++ NIP) to solve an equality-constrained optimization problem for the most probable point (MPP). The MPP search may be formulated as the reliability index approach (RIA) for mapping response levels to reliabilities/probabilities or as the performance measure approach (PMA) for performing the inverse mapping of reliability/probability levels to response levels.

14.147.2 Member Function Documentation

**void pre_run( ) [virtual]**

pre-run portion of run (optional); re-implemented by Iterators which can generate all Variables (parameter sets) a priori

  pre-run phase, which a derived iterator may optionally reimplement; when not present, pre-run is likely integrated into the derived run function. This is a virtual function; when re-implementing, a derived class must call its nearest parent’s pre_run(), if implemented, typically before performing its own implementation steps.

  Reimplemented from Analyzer.

  References Model::initialize_mapping(), Iterator::methodPCIter, NonD::milPIndex, NonDReliability::mpp-Model, NonDReliability::mppSearchType, Analyzer::pre_run(), and Model::update_from_subordinate_model().

**void core_run( ) [virtual]**

core portion of run; implemented by all derived classes and may include pre/post steps in lieu of separate pre/post steps

  Virtual run function for the iterator class hierarchy. All derived classes need to redefine it.

  Reimplemented from Iterator.

  References NonD::compute_densities(), NonDAdaptImpSampling::extreme_values(), NonDReliability::importance-Sampler, NonDReliability::integrationRefinement, Iterator::iterator_rep(), NonDLocalReliability::mean_value(), NonDLocalReliability::mpp_search(), NonDReliability::mppSearchType, NonD::pdfOutput, and NonD::resize-_final_statistics_gradients().

**void print_results( std::ostream & s, short results_state = FINAL_RESULTS ) [virtual]**

print the final iterator results

  This virtual function provides additional iterator-specific final results outputs beyond the function evaluation summary printed in finalize_run().

  Reimplemented from Analyzer.
References NonD::cdfFlag, NonD::computedGenRelLevels, NonD::computedProbLevels, NonD::computedRelLevels, NonD::computedRespLevels, Model::continuous_variable_labels(), NonD::finalMomentsType, NonDLocalReliability::impFactor, Iterator::iteratedModel, NonD::momentStats, NonDReliability::mppSearchType, Model::multivariate_distribution(), Analyzer::numContinuousVars, Analyzer::numFunctions, NonD::print_densities(), Model::response_labels(), NonDLocalReliability::warningBits, and Dakota::write_precision.

```cpp
void RIA_objective_eval ( const Variables & sub_model_vars, const Variables & recast_vars, const Response & sub_model_response, Response & recast_response ) [static], [private]
```

static function used as the objective function in the Reliability Index Approach (RIA) problem formulation. This equality-constrained optimization problem performs the search for the most probable point (MPP) with the objective function of $(\text{norm } u)^2$. This function recasts a $G(u)$ response set (already transformed and approximated in other recursions) into an RIA objective function.

References Response::active_set_request_vector(), Variables::continuous_variables(), Response::function_gradient_view(), Response::function_hessian_view(), and Response::function_value().

Referenced by NonDLocalReliability::mpp_search().

```cpp
void RIA_constraint_eval ( const Variables & sub_model_vars, const Variables & recast_vars, const Response & sub_model_response, Response & recast_response ) [static], [private]
```

static function used as the constraint function in the Reliability Index Approach (RIA) problem formulation. This equality-constrained optimization problem performs the search for the most probable point (MPP) with the constraint of $G(u) = \text{response level}$. This function recasts a $G(u)$ response set (already transformed and approximated in other recursions) into an RIA equality constraint.

References Response::active_set_request_vector(), Response::function_gradient(), Response::function_gradient_view(), Response::function_hessian(), Response::function_value(), NonDLocalReliability::nondLocRelInstance, NonDReliability::requestedTargetLevel, and NonDReliability::respFnCount.

Referenced by NonDLocalReliability::mpp_search().

```cpp
void PMA_objective_eval ( const Variables & sub_model_vars, const Variables & recast_vars, const Response & sub_model_response, Response & recast_response ) [static], [private]
```

static function used as the objective function in the Performance Measure Approach (PMA) problem formulation. This equality-constrained optimization problem performs the search for the most probable point (MPP) with the objective function of $G(u)$. This function recasts a $G(u)$ response set (already transformed and approximated in other recursions) into a PMA objective function.

References Response::active_set_request_vector(), Variables::continuous_variables(), NonDLocalReliability::curvatureDataAvailable, Response::function_gradient(), Response::function_gradient_view(), Response::function_hessian(), Response::function_hessian_view(), Response::function_value(), NonDLocalReliability::integrationOrder, NonDLocalReliability::kappaUpdated, NonDReliability::mppSearchType, NonDLocalReliability::nondLocRelInstance, NonDReliability::pmaMaximizeG, NonDReliability::respFnCount, and NonDLocalReliability::update_pma_maximize().

Referenced by NonDLocalReliability::mpp_search().
void PMA_constraint_eval ( const Variables & sub_model_vars, const Variables & recast_vars, const Response & sub_model_response, Response & recast_response ) [static], [private]

static function used as the constraint function in the first-order Performance Measure Approach (PMA) problem formulation. This optimization problem performs the search for the most probable point (MPP) with the equality constraint of \((\text{norm } u) \wedge 2 = (\beta - \beta_{\text{bar}}) \wedge 2\).

This function recasts a \(G(u)\) response set (already transformed and approximated in other recursions) into a first-order PMA equality constraint on reliability index \(\beta\).

References Response::active_set_request_vector(), Variables::continuous_variables(), Response::function_gradient_view(), Response::function_hessian_view(), Response::function_value(), NonDLocalReliability::nondLocRelInstance, and NonDReliability::requestedTargetLevel.

Referenced by NonDLocalReliability::mpp_search().

void PMA2_constraint_eval ( const Variables & sub_model_vars, const Variables & recast_vars, const Response & sub_model_response, Response & recast_response ) [static], [private]

static function used as the constraint function in the second-order Performance Measure Approach (PMA) problem formulation. This optimization problem performs the search for the most probable point (MPP) with the equality constraint of \(\beta^* = \beta^*_\text{bar}\).

This function recasts a \(G(u)\) response set (already transformed and approximated in other recursions) into a second-order PMA equality constraint on generalized reliability index \(\beta^*\).

References Dakota::abort_handler(), Response::active_set_request_vector(), NonD::cdfFlag, NonDLocalReliability::computedGenRelLevel, NonDLocalReliability::computedRelLevel, Variables::continuous_variables(), NonDLocalReliability::dp2_dbeta_factor(), NonDLocalReliability::fnGradU, NonDLocalReliability::fnHessU, Response::function_gradient_view(), Response::function_hessian(), Response::function_value(), NonDLocalReliability::mostProbPointU, NonD::mppSearchType, NonDLocalReliability::nondLocRelInstance, NonDLocalReliability::probability(), NonDLocalReliability::reliability(), NonDReliability::requestedTargetLevel, NonD::respFnCount, and NonDLocalReliability::signed_norm().

Referenced by NonDLocalReliability::mpp_search().

void initial_taylor_series ( ) [private]

convenience function for performing the initial limit state Taylor-series approximation

An initial first- or second-order Taylor-series approximation is required for MV/AMV/AMV+/TANA or for the case where momentStats (from MV) are required within finalStatistics for subIterator usage of NonDLocalReliability.

References Response::active_set_request_vector(), Iterator::activeSet, Model::component_parallel_mode(), Model::continuous_variables(), Model::current_response(), Model::evaluate(), NonD::finalMomentsType, NonD::finalStatistics, NonDLocalReliability::fnGradsMeanX, NonDLocalReliability::fnHessiansMeanX, NonDLocalReliability::fnValsMeanX, Response::function_gradients(), Response::function_hessians(), Response::function_values(), Model::hessian_type(), Iterator::iteratedModel, NonD::momentStats, NonDReliability::mppSearchType, Model::multivariate_distribution(), Analyzer::numContinuousVars, Analyzer::numFunctions, NonDLocalReliability::ranVarMeansX, NonDLocalReliability::ranVarStdDevsX, ActiveSet::request_vector(), NonD::requestedGenRelLevels, NonD::requestedProbLevels, NonD::requestedRelLevels, NonD::requestedRespLevels, Iterator::subIteratorFlag, NonDLocalReliability::taylorOrder, and NonD::uSpaceModel.

Referenced by NonDLocalReliability::mean_value(), and NonDLocalReliability::mpp_search().

void initialize_class_data ( ) [private]

convenience function for initializing class scope arrays
Initialize class-scope arrays and perform other start-up activities, such as evaluating median limit state responses.

References Response::active_set_derivative_vector(), NonD::finalStatistics, Analyzer::numContinuousVars, Analyzer::numFunctions, NonD::numRelAnalyses, NonD::numASVLev0, NonD::numUASVLev0, Model::probability_transformation(), NonD::uSpaceModel, and NonD::warmStartFlag.

Referenced by NonD::mpp_search().

```cpp
void initialize_level_data() [private]
```

Convenience function for initializing/warm starting MPP search data for each response function prior to level 0

For a particular response function prior to the first z/p/beta level, initialize/warm-start optimizer initial guess (initialPtU), expansion point (mostProbPointX/U), and associated response data (computedRespLevel, fnGradX/U, and fnHessX/U).

References NonD::assign_mean_data(), NonD::computedRespLevel, Model::inactive_continuous_variables(), NonD::initialPtU, NonD::initialPtUSpec, NonD::iterationModel, NonD::mostProbPointU, NonD::mppSearchType, Analyzer::numContinuousVars, NonD::numRelAnalyses, NonD::prevCumASVLev0, NonD::prevNULLLev0, NonD::prevMPPULev0, NonD::prevMPPULev0, NonD::prevMPPUlev0, NonD::prevMPPUlev0, NonD::prevMPPUlev0, NonD::requestedRespLevels, NonD::respFnCount, NonD::taylorOrder, NonD::truth_evaluation(), NonD::update_limit_state surrogate(), NonD::uSpaceModel, and NonD::warmStartFlag.

Referenced by NonD::mpp_search().

```cpp
void initialize_mpp_search_data() [private]
```

Convenience function for initializing/warm starting MPP search data for each z/p/beta level for each response function

For a particular response function at a particular z/p/beta level, warm-start or reset the optimizer initial guess (initialPtU), expansion point (mostProbPointX/U), and associated response data (computedRespLevel, fnGradX/U, and fnHessX/U).

References NonD::assign_mean_data(), NonD::computedGenRelLevels, NonD::computedRelLevels, NonD::fnGradU, Model::hessian_type(), NonD::initialPtU, NonD::iterationModel, NonD::mostProbPointU, NonD::mppSearchType, NonD::levelCount, NonD::numContinuousVars, NonD::requestedRespLevels, NonD::requestedReslLevels, NonD::requestedTargetLevel, NonD::respFnCount, NonD::taylorOrder, and NonD::warmStartFlag.

Referenced by NonD::mpp_search().

```cpp
void update_mpp_search_data(const Variables & vars_star, const Response & resp_star) [private]
```

Convenience function for updating MPP search data for each z/p/beta level for each response function

Includes case-specific logic for updating MPP search data for the AMV/AMV+/TANA/NO APPROX methods.

References Response::active_set(), Response::active_set_request_vector(), NonD::approxConverged, NonD::approxIters, NonD::computedRelLevel, NonD::computedRespLevel, Model::continuous_variable_ids(), Variables::continuous_variables(), Iteration::convergenceTol, Variables::
void update_level_data() [private]

convenience function for updating z/p/beta level data and final statistics following MPP convergence

Upgrades computedRespLevels/computedRelLevels, finalStatistics, warm start, and graphics data.

References Response::active_set_derivative_vector(), Response::active_set_request_vector(), Graphics::add datapoint(), NonD::cdfFlag, NonDLocalReliability::computedGenRelLevel, NonD::computedGenRelLevels, NonD::computedRespLevels, NonDLocalReliability::computedRelLevel, NonD::computedRelLevels, NonDLocalReliability::computedRespLevel, NonD::computedRespLevels, NonDLocalReliability::taylorOrder, Dakota::length(), NonDLocalReliability::levelCount, NonDLocalReliability::mostProbPointU, NonDLocalReliability::mostProbPointX, Graphics::new dataset(), Analyzer::numContinuousVars, ParallelLibrary::output_manager(), Iterator::parallelLib, NonDLocalReliability::prevCumASVLev0, NonDLocalReliability::prevFnGradDLev0, NonDLocalReliability::prevFnGradULevel0, NonDLocalReliability::prevMPPULevel0, NonDLocalReliability::probLevel(), NonDLocalReliability::probability(), NonD::requestedGenRelLevels, NonD::requestedRelLevels, NonD::requestedRespLevels, NonD::respLevelTarget, NonD::respLevelTargetReduce, NonDReliability::statCount, Iterator::subIteratorFlag, NonD::totalLevelRequests, and NonDLocalReliability::warmStartFlag.

Referenced by NonDLocalReliability::mpp_search().

void dg_ds_eval (const RealVector &x_vars, const RealVector &fn_grad_x, RealVector &final_stat_grad)

[private]

convenience function for evaluating dg/ds

Computes dg/ds where s = design variables. Supports potentially overlapping cases of design variable augmentation and insertion.

References Response::active_set_derivative_vector(), Iterator::activeSet, Model::component_parallel_mode(), Model::continuous_variables(), Model::current_response(), ActiveSet::derivative_vector(), Model::evaluate(), NonD::finalStatistics, Response::function_gradient_copy(), Response::function_gradients(), Model::inactive_continuous_variable_ids(), Iterator::iteratedModel, NonDReliability::mppSearchType, Model::nested_actv2_targets(), Model::query_distribution_parameter_derivatives(), ActiveSet::request_value(), ActiveSet::request_values(), NonDReliability::respFnCount, Model::trans_grad_X_to_S(), and NonDReliability::uSpaceModel.

Referenced by NonDLocalReliability::mean_value(), NonDLocalReliability::mpp_search(), and NonDLocalReliability::update_level_data().
Real dp2_dbeta_factor ( Real beta, bool cdf_flag ) [private]

compute factor for derivative of second-order probability with respect to reliability index (from differentiating BREITUNG or HOHENRACK expressions)

Compute sensitivity of second-order probability w.r.t. beta for use in derivatives of $p_2$ or beta* w.r.t. auxiliary parameters s (design, epistemic) or derivatives of beta* w.r.t. u in PMA2_constraint_eval().

References Dakota::abort_handler(), NonDLocalReliability::curvatureDataAvailable, NonDLocalReliability::curvatureThresh, NonDLocalReliability::kappaU, Analyzer::numContinuousVars, NonDLocalReliability::probability(), NonDLocalReliability::scale_curvature(), NonDLocalReliability::secondOrderIntType, and NonDLocalReliability::warningBits.

Referenced by NonDLocalReliability::PMA2_constraint_eval(), and NonDLocalReliability::update_level_data().

Real probability ( Real beta, bool cdf_flag, const RealVector & mpp_u, const RealVector & fn_grad_u, const RealSymMatrix & fn_hess_u ) [private]

Convert provided reliability to probability using either a first-order or second-order integration.

Converts beta into a probability using either first-order (FORM) or second-order (SORM) integration. The SORM calculation first calculates the principal curvatures at the MPP (using the approach in Ch. 8 of Haldar & Mahadevan), and then applies correction formulations from the literature (Breitung, Hohenbichler-Rackwitz, or Hong).

References NonDLocalReliability::curvatureDataAvailable, NonDLocalReliability::curvatureThresh, NonDAdaptImpSampling::final_probability(), NonDReliability::importanceSampler, NonDLocalReliability::integrationOrder, NonDReliability::integrationRefinement, Iterator::iterator_rep(), NonDLocalReliability::kappaU, NonDLocalReliability::kappaUpdated, Dakota::length(), NonDReliability::levelCount, Iterator::methodPCIter, NonD::miPLIndex, Analyzer::numContinuousVars, Iterator::outputLevel, NonDLocalReliability::principal_curvatures(), NonDLocalReliability::probability(), NonD::requestedRespLevels, NonDReliability::requestedTargetLevel, NonDReliability::respFnCount, Iterator::run(), NonDLocalReliability::scale_curvature(), NonDLocalReliability::secondOrderIntType, NonDLocalReliability::warningBits, and Dakota::write_precision.

The documentation for this class was generated from the following files:

- NonDLocalReliability.hpp
- NonDLocalReliability.cpp

14.148 NonDLocalSingleInterval Class Reference

Class for using local gradient-based optimization approaches to calculate interval bounds for epistemic uncertainty quantification.

Inheritance diagram for NonDLocalSingleInterval:
Public Member Functions

- `NonDLocalSingleInterval (ProblemDescDB &problem_db, Model &model)`
  
  *constructor*

- `~NonDLocalSingleInterval ()`
  
  *destructor*

Protected Member Functions

- `void initialize ()`
  
  *perform any required initialization*

- `void post_process_cell_results (bool maximize)`
  
  *post-process a cell minimization/maximization result*

Private Attributes

- `size_t statCntr`
  
  *counter for finalStatistics*

Additional Inherited Members

14.148.1 Detailed Description

Class for using local gradient-based optimization approaches to calculate interval bounds for epistemic uncertainty quantification.

The `NonDLocalSingleInterval` class supports local gradient-based optimization approaches to determining interval bounds for epistemic UQ. The interval bounds may be on the entire function in the case of pure interval analysis (e.g. intervals on input = intervals on output), or the intervals may be on statistics of an "inner loop" aleatory analysis such as intervals on means, variances, or percentile levels.

The documentation for this class was generated from the following files:

- `NonDLocalSingleInterval.hpp`
- `NonDLocalSingleInterval.cpp`
14.149 NonDMultilevelFunctionTrain Class Reference

Nonintrusive polynomial chaos expansion approaches to uncertainty quantification.

Inheritance diagram for NonDMultilevelFunctionTrain:

```
NonDMultilevelFunctionTrain
|___ NonDC3FunctionTrain
|___ NonDExpansion
|___ NonD
|___ Analyzer
|___ Iterator
```

Public Member Functions

- **NonDMultilevelFunctionTrain (ProblemDescDB &problem_db, Model &model)**
  standard constructor

- **~NonDMultilevelFunctionTrain ()**
  destructor

Protected Member Functions

- void initialize_u_space_model ()
  initialize uSpaceModel polynomial approximations with PCE/SC data

- void core_run ()
  perform a forward uncertainty propagation using PCE/SC methods

- void assign_specification_sequence ()
  assign the current values from the input specification sequence

- void increment_specification_sequence ()
  increment the input specification sequence and assign values

- size_t collocation_points () const
  return specification for number of collocation points (may be part of a sequence specification)

- int random_seed () const
  return specification for random seed (may be part of a sequence specification)

- int first_seed () const
  return first seed in sequence specification (defaults to random_seed())

- void initialize_ml_regression (size_t num_lev, bool &import_pilot)
  initializations for multilevel_regression()
• void infer_pilot_sample (SizetArray &delta_N,l)
• void increment_sample_sequence (size_t new_samp, size_t total_samp, size_t step)
  increment sequence in numSamplesOnModel for multilevel_regression()
• void compute_sample_increment (const RealVector &regress_metrics, const SizetArray &N,l, SizetArray &delta_N,l)
  compute delta_N,l for {RIP,RANK}_SAMPLING cases
• void print_results (std::ostream &s, short results_state=FINAL_RESULTS)
  print the final statistics
• void assign_allocation_control ()
  assign defaults related to allocation control (currently for ML regression approaches)

Private Member Functions

• size_t start_rank (size_t index) const
• size_t start_rank () const
• unsigned short start_order (size_t index) const
• unsigned short start_order () const

Private Attributes

• SizetArray startRankSeqSpec
  user specification for start_rank_sequence
• UShortArray startOrderSeqSpec
  user specification for start_order_sequence
• size_t sequenceIndex
  sequence index for start{Rank,Order}SeqSpec

Additional Inherited Members

14.149.1 Detailed Description

Nonintrusive polynomial chaos expansion approaches to uncertainty quantification.

The NonDMultilevelFunctionTrain class uses a set of function train (FT) expansions, one per model fidelity or resolution, to approximate the effect of parameter uncertainties on response functions of interest.

14.149.2 Constructor & Destructor Documentation

NonDMultilevelFunctionTrain ( ProblemDescDB & problem_db, Model & model )

standard constructor

This constructor is called for a standard letter-envelope iterator instantiation using the ProblemDescDB.

References Dakota::abort_handler(), Response::active_set(), NonDMultilevelFunctionTrain::assign_allocation_control(), NonDExpansion::assign_discrepancy_mode(), NonDExpansion::assign_hierarchical_response_mode(), Model::assign_rep(), NonDMultilevelFunctionTrain::collocation_points(), ParallelLibrary::command_line_check(), NonDC3FunctionTrain::config_regression(), NonDExpansion::configure_expansion_orders(), NonDExpansion::construct_expansion_sampler(), Model::current_response(), ActiveSet::derivative_vector(), NonDExpansion::dimPrefSpec, ProblemDescDB::get_bool(), ProblemDescDB::get_iv(), ProblemDescDB::get_string(), ProblemDescDB::get_sza(),
ProblemDescDB::get_ushort(), NonDC3FunctionTrain::importBuildPointsFile, NonDMultilevelFunctionTrain::initialize_u_space_model(), Iterator::iteratedModel, Analyzer::numContinuousVars, NonDExpansion::numSamplesOnModel, Iterator::outputLevel, Iterator::parallelLib, Iterator::probDescDB, Model::qoi(), NonDMultilevelFunctionTrain::random_seed(), NonDExpansion::randomSeedSeqSpec, SharedC3ApproxData::regression_size(), NonDC3FunctionTrain::resolve_inputs(), and NonDExpansion::uSpaceModel.

~NonDMultilevelFunctionTrain()

destructor

This constructor is used for helper iterator instantiation on the fly that employ regression.

NonDMultilevelFunctionTrain::NonDMultilevelFunctionTrain(unsigned short method_name, Model& model, const SizetArray& colloc_pts_seq, const RealVector& dim_pref, Real colloc_ratio, const SizetArray& seed_seq, short u_space_type, short refine_type, short refine_control, short ml_alloc_control, short ml_discrep, short rule_nest, short rule_growth, bool piecewise_basis, bool use_derivs, bool cv_flag, const String& import_build_pts_file, unsigned short import_build_format, bool import_build_active_only): NonDC3FunctionTrain(method_name, model, exp_coeffs_approach, dim_pref, u_space_type, refine_type, refine_control, covar_control, colloc_pts_seq, colloc_ratio, ml_alloc_control, ml_discrep, //rule_nest, rule_growth, piecewise_basis, use_derivs, 0, cv_flag), expOrderSeqSpec(exp_order_seq), sequenceIndex(0) {
    randomSeedSeqSpec = seed_seq;
    assign_discrepancy_mode(); assign_hierarchical_response_mode();

Resolve settings

short data_order; resolve_inputs(uSpaceType, data_order);

Recast $g(x)$ to $G(u)$

Model g_u_model; g_u_model.assign_rep(new ProbabilityTransformModel(iteratedModel, uSpaceType), false); // retain dist bounds

Construct $u$-space sampler

Iterator u_space_sampler; UShortArray approx_orders; extract (initial) values from sequences configure_expansion_orders(start_order(), dimPrefSpec, approx_orders); size_t colloc_pts = collocation_points(), regress_size = SharedC3ApproxData::regression_size(numContinuousVars, start_rank(), approx_orders); if (!config_regression(colloc_pts, regress_size, random_seed(), u_space_sampler, g_u_model)) { Cerr << "Error: incomplete configuration in NonDMultilevelFunctionTrain" << "constructor." << std::endl; abort_handler(METHOD_ERROR); }

Construct $G$-hat($u$) = $u$-SpaceModel

$G$-hat($u$) uses an orthogonal polynomial approximation over the active/uncertain variables (using same view as iteratedModel/g_u_model: not the typical All view for DACE). No correction is employed. Note: for PCB-DO with polynomials over $\{u\}+\{d\}$, change view to All. short corr_order = -1, corr_type = NO_CORRECTION; if (!import_build_pts_file.empty()) pt_reuse = "all"; const ActiveSet& recast_set = g_u_model.current_response().active_set(); DFSModel: consume any QoI aggregation. Helper mode: support approx Hessians ShortArray asv(g_u_model.qoi(), 7); // TO DO: consider passing in data_mode ActiveSet pce_set(asv, recast_set.derivative_vector()); uSpaceModel.assign_rep(new DataFitSurrModel(u_space_sampler, g_u_model, pce_set, approx_type, approx_orders, corr_type, corr_order, data_order, outputLevel, pt_reuse, import_build_pts_file, import_build_format, import_build_active_only), false); initialize_u_space_model();
Configure settings for ML allocation (requires uSpaceModel) assign_allocation_control();
no expansionSampler, no numSamplesOnExpansion }

14.149.3 Member Function Documentation

void increment_specification_sequence() [protected], [virtual]

increment the input specification sequence and assign values
Default implementation redefined by Multilevel derived classes.
Reimplemented from NonDExpansion.
References NonDMultilevelFunctionTrain::assign_specification_sequence(), NonDExpansion::collocPtsSeqSpec, NonDExpansion::randomSeedSeqSpec, NonDMultilevelFunctionTrain::sequenceIndex, NonDMultilevelFunctionTrain::startOrderSeqSpec, and NonDMultilevelFunctionTrain::startRankSeqSpec.

void infer_pilot_sample(SizetArray & delta_Nj) [protected], [virtual]

Default implementation redefined by Multilevel derived classes.
Reimplemented from NonDExpansion.
References NonDExpansion::collocRatio, NonDExpansion::configure_expansion_orders(), NonDExpansion::dimPrefSpec, Analyzer::numContinuousVars, and SharedC3ApproxData::regression_size().
The documentation for this class was generated from the following files:
• NonDMultilevelFunctionTrain.hpp
• NonDMultilevelFunctionTrain.cpp

14.150 NonDMultilevelPolynomialChaos Class Reference

Nonintrusive polynomial chaos expansion approaches to uncertainty quantification.
Inheritance diagram for NonDMultilevelPolynomialChaos:

```
  Iterator
  |
  Analyzer
  |
  NonD
  |
  NonDExpansion
  |
  NonDPolynomialChaos
  |
  NonDMultilevelPolynomialChaos
```

Public Member Functions

• NonDMultilevelPolynomialChaos (ProblemDescDB &problem_db, Model &model)
standard constructor

- \texttt{NonDMultilevelPolynomialChaos (Model \&model, short exp\_coeffs\_approach, const UShortArray \&num\_int\_seq, const RealVector \&dim\_pref, short u\_space\_type, short refine\_type, short refine\_control, short covar\_control, short ml\_alloc\_cntl, short ml\_discrep, short rule\_nest, short rule\_growth, bool piecewise\_basis, bool use\_derivs)}

alternate constructor for numerical integration (tensor, sparse, cubature)

- \texttt{NonDMultilevelPolynomialChaos (unsigned short method\_name, Model \&model, short exp\_coeffs\_approach, const UShortArray \&exp\_order\_seq, const RealVector \&dim\_pref, const SizetArray \&colloc\_pts\_seq, Real colloc\_ratio, const SizetArray \&seed\_seq, short u\_space\_type, short refine\_type, short refine\_control, short covar\_control, short ml\_alloc\_cntl, short ml\_discrep, bool piecewise\_basis, bool use\_derivs, bool cv\_flag, const String \&import\_build\_pts\_file, unsigned short import\_build\_format, bool import\_build\_active\_only)}

alternate constructor for regression (least squares, CS, OLI)

- \texttt{\sim NonDMultilevelPolynomialChaos ()}

destructor

Protected Member Functions

- \texttt{void core\_run ()}
  perform a forward uncertainty propagation using PCE/SC methods
- \texttt{void assign\_specification\_sequence ()}
  assign the current values from the input specification sequence
- \texttt{void increment\_specification\_sequence ()}
  increment the input specification sequence and assign values
- \texttt{size\_t collocation\_points () const}
  return specification for number of collocation points (may be part of a sequence specification)
- \texttt{int random\_seed () const}
  return specification for random seed (may be part of a sequence specification)
- \texttt{int first\_seed () const}
  return first seed in sequence specification (defaults to random\_seed())
- \texttt{void initialize\_ml\_regression (size\_t num\_lev, bool \&import\_pilot)}
  initializations for multilevel\_regression()
- \texttt{void infer\_pilot\_sample (SizetArray \&delta\_N\_l)}
- \texttt{void increment\_sample\_sequence (size\_t new\_samp, size\_t total\_samp, size\_t step)}
  increment sequence in numSamplesOnModel for multilevel\_regression()
- \texttt{void compute\_sample\_increment (const RealVector \&sparsity, const SizetArray \&N\_I, SizetArray \&delta\_N\_I)}
  compute delta\_N\_I for \texttt{\{RIP,RANK\}_SAMPLING} cases
- \texttt{void print\_results (std::ostream \&s, short results\_state=FINAL\_RESULTS)}
  print the final statistics
- \texttt{void assign\_allocation\_control ()}
  assign defaults related to allocation control (currently for ML regression approaches)
Nonintrusive polynomial chaos expansion approaches to uncertainty quantification. The NonDMultilevelPolynomialChaos class uses a polynomial chaos expansion (PCE) approach to approximate the effect of parameter uncertainties on response functions of interest. It utilizes the OrthogPolyApproximation class to manage multiple types of orthogonal polynomials within a Wiener-Askey scheme to PCE. It supports PCE coefficient estimation via sampling, quadrature, point-collocation, and file import.

14.150.2 Constructor & Destructor Documentation

NonDMultilevelPolynomialChaos (ProblemDescDB & problem_db, Model & model)

standard constructor

This constructor is called for a standard letter-envelope iterator instantiation using the ProblemDescDB.

References Dakota::abort_handler(), NonDMultilevelPolynomialChaos::assign_allocation_control(), NonDExpansion::assign_discrepancy_mode(), NonDExpansion::assign_hierarchical_response_mode(), Model::assign_rep(), NonDMultilevelPolynomialChaos::collocation_points(), ParallelLibrary::command_line_check(), NonDPolynomialChaos::config_expectation(), NonDPolynomialChaos::config_integration(), NonDPolynomialChaos::config_regression(), NonDExpansion::configure_expansion_orders(), NonDExpansion::construct_expansion_sampler(), NonDPolynomialChaos::cubIntSpec, ActiveSet::derivative_vector(), NonDExpansion::dimPrefSpec, ProblemDescDB::get_bool(),
ProblemDescDB::get_iv(), ProblemDescDB::get_real(), ProblemDescDB::get_short(), ProblemDescDB::get_string(), ProblemDescDB::get_sza(), ProblemDescDB::get_usa(), ProblemDescDB::get_ushort(), NonDPolynomialChaos::importBuildPointsFile, NonDPolynomialChaos::initialize_u_space_model(), Iterator::iteratedModel, NonDExpansion::numSamplesOnModel, Iterator::outputLevel, Iterator::parallelLib, Iterator::probDescDB, NonDMultilevelPolynomialChaos::random_seed(), NonDExpansion::randomSeedSeqSpec, NonDPolynomialChaos::resolve_inputs(), NonDExpansion::uSpaceModel, and NonDPolynomialChaos::uSpaceType.

NonDMultilevelPolynomialChaos ( Model & model, short exp_coeffs_approach, const UShortArray & num_int_seq, const RealVector & dim_pref, short u_space_type, short refine_type, short refine_control, short covar_control, short ml_alloc_control, short ml_discrep, short rule_nest, short rule_growth, bool piecewise_basis, bool use_derivs )

alternate constructor for numerical integration (tensor, sparse, cubature)

This constructor is used for helper iterator instantiation on the fly that employ numerical integration (quadrature, sparse grid, cubature).

References Dakota::abort_handler(), Response::active_set(), NonDMultilevelPolynomialChaos::assign_allocation_control(), NonDExpansion::assign_discrepancy_mode(), NonDExpansion::assign_hierarchical_response_mode(), Model::assign_rep(), NonDPolynomialChaos::config_integration(), NonDPolynomialChaos::cubIntSpec, Model::current_response(), ActiveSet::derivative_vector(), NonDPolynomialChaos::initialize_u_space_model(), Iterator::iteratedModel, Iterator::outputLevel, Model::qoi(), NonDMultilevelPolynomialChaos::quadOrderSeqSpec, NonDPolynomialChaos::resolve_inputs(), NonDMultilevelPolynomialChaos::sequenceIndex, NonDMultilevelPolynomialChaos::ssgLevelSeqSpec, NonDExpansion::uSpaceModel, and NonDPolynomialChaos::uSpaceType.

NonDMultilevelPolynomialChaos ( unsigned short method_name, Model & model, short exp_coeffs_approach, const UShortArray & exp_order_seq, const RealVector & dim_pref, const SizetArray & colloc_pts_seq, Real colloc_ratio, const SizetArray & seed_seq, short u_space_type, short refine_type, short refine_control, short covar_control, short ml_alloc_control, short ml_discrep, bool piecewise_basis, bool use_derivs, bool cv_flag, const String & import_build_pts_file, unsigned short import_build_format, bool import_build_active_only )

alternate constructor for regression (least squares, CS, OLI)

This constructor is used for helper iterator instantiation on the fly that employ regression (least squares, CS, OLI).

References Response::active_set(), NonDMultilevelPolynomialChaos::assign_allocation_control(), NonDExpansion::assign_discrepancy_mode(), NonDExpansion::assign_hierarchical_response_mode(), Model::assign_rep(), NonDMultilevelPolynomialChaos::collocation_points(), NonDPolynomialChaos::config_regression(), NonDExpansion::configure_expansion_orders(), Model::current_response(), ActiveSet::derivative_vector(), NonDExpansion::dimPrefSpec, NonDPolynomialChaos::initialize_u_space_model(), Iterator::iteratedModel, Iterator::outputLevel, Model::qoi(), NonDMultilevelPolynomialChaos::random_seed(), NonDExpansion::randomSeedSeqSpec, NonDPolynomialChaos::resolve_inputs(), NonDExpansion::uSpaceModel, and NonDPolynomialChaos::uSpaceType.

14.150.3 Member Function Documentation

void increment_specification_sequence ( ) [protected], [virtual]

increment the input specification sequence and assign values

Default implementation redefined by Multilevel derived classes.

Reimplemented from NonDExpansion.

References Dakota::abort_handler(), NonDExpansion::collocPtsSeqSpec, NonDExpansion::expOrderSeqSpec, NonDMultilevelPolynomialChaos::expCoeffsApproach, NonDMultilevelPolynomialChaos::expOrderSeqSpec, NonDMultilevelPolynomialChaos::expSamples-
void infer_pilot_sample ( SizetArray & delta_N ) [protected], [virtual]

Default implementation redefined by Multilevel derived classes.

Reimplemented from NonDExpansion.

References NonDExpansion::collocRatio, NonDExpansion::configure_expansion_orders(), NonDExpansion::dimPrefSpec, NonDExpansion::expansionBasisType, and NonDExpansion::terms_ratio_to_samples().

The documentation for this class was generated from the following files:

- NonDMultilevelPolynomialChaos.hpp
- NonDMultilevelPolynomialChaos.cpp

## 14.151 NonDMultilevelSampling Class Reference

Performs Multilevel Monte Carlo sampling for uncertainty quantification.

Inheritance diagram for NonDMultilevelSampling:

```plaintext
NonDMultilevelSampling
  NonDSampling
    NonD
      NonDAnalyzer
        Iterator
```

### Public Member Functions

- **NonDMultilevelSampling** (ProblemDescDB &problem_db, Model &model)
  
  *standard constructor*

- ~**NonDMultilevelSampling** ()
  
  *destructor*

- **bool resize ()**
  
  *reinitializes iterator based on new variable size*
Protected Member Functions

- void **pre_run** ()
  
  *pre-run portion of run (optional); re-implemented by Iterators which can generate all Variables (parameter sets) a priori*

- void **core_run** ()

- void **post_run** (std::ostream &s)
  
  *post-run portion of run (optional); verbose to print results; re-implemented by Iterators that can read all Variables/Responses and perform final analysis phase in a standalone way*

- void **print_results** (std::ostream &s, short results_state=FINAL_RESULTS)
  
  *print the final iterator results*

Private Member Functions

- void **multilevel_mc_Ysum** (unsigned short model_form)
  
  *Perform multilevel Monte Carlo across the discretization levels for a particular model form using discrepancy accumulators (sum_Y)*

- void **multilevel_mc_Qsum** (unsigned short model_form)
  
  *Perform multilevel Monte Carlo across the discretization levels for a particular model form using QoI accumulators (sum_Q)*

- void **control_variate_mc** (const UShortArray &active_key)
  
  *Perform control variate Monte Carlo across two model forms.*

- void **multilevel_control_variate_mc_Ycorr** (unsigned short lf_model_form, unsigned short hf_model_form)
  
  *Perform multilevel Monte Carlo across levels in combination with control variate Monte Carlo across model forms at each level; CV computes correlations for Y (LH correlations for level discrepancies)*

- void **multilevel_control_variate_mc_Qcorr** (unsigned short lf_model_form, unsigned short hf_model_form)
  
  *Perform multilevel Monte Carlo across levels in combination with control variate Monte Carlo across model forms at each level; CV computes correlations for Q (LH correlations for QoI)*

- void **shared_increment** (size_t iter, size_t lev)
  
  *perform a shared increment of LF and HF samples for purposes of computing/updating the evaluation ratio and the MSE ratio*

- bool **lf_increment** (Real avg_eval_ratio, const SizetArray &N_lf, const SizetArray &N_hf, size_t iter, size_t lev)
  
  *perform final LF sample increment as indicated by the evaluation ratio*

- void **aggregated_models_mode** ()
  
  *synchronize iteratedModel and activeSet on AGGREGATED_MODELS mode*

- void **bypass_surrogate_mode** ()
  
  *synchronize iteratedModel and activeSet on BYPASS_SURROGATE mode*

- void **uncorrected_surrogate_mode** ()
  
  *synchronize iteratedModel and activeSet on UNCORRECTED_SURROGATE mode*

- Real **level_cost** (const RealVector &cost, unsigned short step)
  
  *return (aggregate) level cost*

- void **assign_specification_sequence** (size_t index)
  
  *advance any sequence specifications*

- int **random_seed** (size_t index) const
  
  *extract current random seed from randomSeedSeqSpec*
• **void configure_indices** (unsigned short group, unsigned short form, unsigned short lev, unsigned short s_index)

  manage response mode and active model key from \{group,form,lev\} triplet. s_index is the sequence index that defines the active dimension for a model sequence.

• **void initialize_ml_Ysums** (IntRealMatrixMap &sum_Y, size_t num_lev)

  initialize the ML accumulators for computing means, variances, and covariances across fidelity levels

• **void initialize_ml_Qsums** (IntRealMatrixMap &sum_Ql, IntRealMatrixMap &sum_Qlm1, IntIntPairRealMatrixMap &sum_QlQlm1, size_t num_lev)

  initialize the ML accumulators for computing means, variances, and covariances across fidelity levels

• **void initialize_cv_sums** (IntRealVectorMap &sum_L_shared, IntRealVectorMap &sum_L_refined, IntRealVectorMap &sum_H, IntRealVectorMap &sum_LL, IntRealVectorMap &sum_LH)

  initialize the CV accumulators for computing means, variances, and covariances across fidelity levels


  initialize the MLCV accumulators for computing means, variances, and covariances across fidelity levels

• **void accumulate_offsets** (RealVector &mu)

  accumulate initial approximation to mean vector, for use as offsets in subsequent accumulations

• **void accumulate_ml_Qsums** (IntRealMatrixMap &sum_Q, size_t lev, const RealVector &offset, SizetArray &num_Q)

  update running QoI sums for one model (sum_Q) using set of model evaluations within allResponses; used for level 0 from other accumulators

• **void accumulate_ml_Ysums** (IntRealMatrixMap &sum_Y, RealMatrix &sum YY, size_t lev, const RealVector &offset, SizetArray &num_Y)

  update accumulators for multilevel telescoping running sums using set of model evaluations within allResponses

• **void accumulate_ml_Qsums** (IntRealMatrixMap &sum_Ql, IntRealMatrixMap &sum_Qlm1, IntIntPairRealMatrixMap &sum_QlQlm1, size_t lev, const RealVector &offset, SizetArray &num_Q)

  update running QoI sums for one model at two levels (sum_Ql, sum_Qlm1) using set of model evaluations within allResponses

• **void accumulate_cv_sums** (IntRealVectorMap &sum_L, const RealVector &offset, SizetArray &num_L)

  update running sums for one model (sum_L) using set of model evaluations within allResponses


  update running sums for two models (sum_L, sum_H, and sum_LH) from set of low/high fidelity model evaluations within allResponses

• **void accumulate_mlcv_Qsums** (IntRealMatrixMap &sum_Ql, IntRealMatrixMap &sum_Qlm1, size_t lev, const RealVector &offset, SizetArray &num_Q)

  update running QoI sums for one model at two levels (sum_Ql, sum_Qlm1) using set of model evaluations within allResponses
• void accumulate_mlcv_Ysums (IntRealMatrixMap &sum_Y, size_t lev, const RealVector &offset, SizetArray &num_Y)
  update running discrepancy sums for one model (sum_Y) using set of model evaluations within allResponses

  update running QoI sums for two models (sum_L, sum_H, sum_LL, sum_LH, and sum_HH) from set of low/high fidelity model evaluations within \{lf,hf\} resp_map; used for level 0 from other accumulators

  update running two-level discrepancy sums for two models and two levels from set of low/high fidelity model evaluations within \{lf,hf\} resp_map

• Real eval_ratio (const RealVector &sum_L_shared, const RealVector &sum_H, const RealVector &sum_LL, RealMatrix &sum_L_refined, RealMatrix &sum_H_refined, const RealMatrix &sum_LH, const RealMatrix &sum_HH, Real cost_ratio, const SizetArray &N_shared, RealMatrix &var_H, RealMatrix &rho2_LH)
  compute the LF/HF evaluation ratio, averaged over the QoI

• Real eval_ratio (RealMatrix &sum_L_shared, RealMatrix &sum_H, RealMatrix &sum_LL, RealMatrix &sum_LH, RealMatrix &sum_HH, Real cost_ratio, size_t lev, const SizetArray &N_shared, RealMatrix &var_H, RealMatrix &rho2_LH)
  compute the LF/HF evaluation ratio, averaged over the QoI

• Real eval_ratio (RealMatrix &sum_L_shared, RealMatrix &sum_H, RealMatrix &sum_LL, RealMatrix &sum_Lm1, RealMatrix &sum_Hm1, RealMatrix &sum_Llm1, RealMatrix &sum_Hlm1, RealMatrix &sum_L2m1, RealMatrix &sum_H2m1, RealMatrix &sum_L2m1, RealMatrix &sum_H2m1, RealMatrix &sum_H2m1, RealMatrix &sum_H2m1, size_t lev, const SizetArray &N_shared, const RealMatrix &var_YHL, const RealMatrix &rho2_LH)
  compute the LF/HF evaluation ratio, averaged over the QoI

• Real MSE_ratio (Real avg_eval_ratio, const RealVector &var_H, const RealVector &rho2_LH, size_t iter, const SizetArray &N_hf)
  compute ratio of MC and CVMC mean squared errors, averaged over the QoI

  compute control variate parameters for CVMC and estimate raw moments
  apply control variate parameters for MLCVMC to estimate raw moment contributions

• void cv\_raw\_moments (IntRealMatrixMap &sum\_L\_1, IntRealMatrixMap &sum\_L\_lm1, IntRealMatrixMap &sum\_H\_1, IntRealMatrixMap &sum\_H\_lm1, IntRealMatrixMap &sum\_L\_1\_lm1, IntRealMatrixMap &sum\_H\_1\_lm1, const SizetArray &N\_shared, IntRealMatrixMap &sum\_L\_1\_refined, IntRealMatrixMap &sum\_H\_1\_refined, const RealMatrix &rho\_dot2\_LH, size\_t lev, RealMatrix &H\_raw\_mom)  
  apply control variate parameters for MLCVMC to estimate raw moment contributions

• void compute\_control (Real sum\_L, Real sum\_H, Real sum\_LL, Real sum\_LH, size\_t N\_shared, Real &beta)  
  compute scalar control variate parameters

• void compute\_control (Real sum\_L, Real sum\_H, Real sum\_LL, Real sum\_LH, Real sum\_HH, size\_t N\_shared, Real &var\_H, Real &rho\_dot2\_LH)  
  compute scalar variance and correlation parameters for control variates

• void compute\_control (Real sum\_L\_1, Real sum\_L\_lm1, Real sum\_H\_1, Real sum\_H\_lm1, Real sum\_L\_1\_lm1, Real sum\_L\_1\_lm1, Real sum\_H\_1\_lm1, Real sum\_H\_1\_lm1, Real sum\_H\_1\_lm1, Real sum\_H\_1\_lm1, Real sum\_H\_1\_lm1, const SizetArray &N\_shared, Real &var\_YH, Real &rho\_dot2\_LH, Real &beta\_dot, Real &gamma)  
  compute scalar control variate parameters

• void compute\_control (const RealVector &sum\_L, const RealVector &sum\_H, const RealVector &sum\_LL, const RealVector &sum\_LH, const RealVector &sum\_HH, const SizetArray &N\_shared, RealVector &beta)  
  compute vector control variate parameters

• void compute\_control (const RealVector &sum\_L, const RealVector &sum\_H, const RealVector &sum\_LL, const RealVector &sum\_LH, const RealVector &sum\_HH, const SizetArray &N\_shared, RealVector &var\_H, RealVector &rho\_dot2\_LH)  
  compute vector variance and correlation parameters for control variates

• void compute\_control (const RealMatrix &sum\_L, const RealMatrix &sum\_H, const RealMatrix &sum\_LL, const RealMatrix &sum\_LH, const SizetArray &N\_shared, size\_t lev, RealVector &beta)  
  compute matrix control variate parameters

• void compute\_control (const RealMatrix &sum\_L\_1, const RealMatrix &sum\_L\_lm1, const RealMatrix &sum\_H\_1, const RealMatrix &sum\_H\_lm1, const RealMatrix &sum\_L\_1\_lm1, const RealMatrix &sum\_L\_1\_lm1, const RealMatrix &sum\_H\_1\_lm1, const RealMatrix &sum\_H\_1\_lm1, const RealMatrix &sum\_H\_1\_lm1, const RealMatrix &sum\_H\_1\_lm1, const RealMatrix &sum\_H\_1\_lm1, const SizetArray &N\_shared, size\_t lev, RealVector &beta\_dot, RealVector &gamma)  
  compute matrix control variate parameters

• void apply\_control (Real sum\_H, Real sum\_L\_shared, size\_t N\_shared, Real sum\_L\_refined, size\_t N\_refined, Real beta, Real &H\_raw\_mom)  
  apply scalar control variate parameter (beta) to approximate HF moment

• void apply\_control (Real sum\_H\_1, Real sum\_H\_lm1, Real sum\_L\_1, Real sum\_L\_lm1, size\_t N\_shared, Real sum\_L\_1\_refined, Real sum\_L\_1\_refined, size\_t N\_refined, Real beta\_dot, Real gamma, Real &H\_raw\_mom)  
  apply scalar control variate parameter (beta) to approximate HF moment
• void **apply** control (const RealVector &sum H, const RealVector &sum L_shared, const SizetArray &N_shared, const RealVector &sum L_refined, const SizetArray &N_refined, const RealVector &beta, RealVector &H_raw_mom)

  apply vector control variate parameter (beta) to approximate HF moment

• void **apply** control (const RealMatrix &sum H, const RealMatrix &sum L_shared, const SizetArray &N_shared, const RealMatrix &sum L_refined, const SizetArray &N_refined, const RealVector &beta, RealVector &H_raw_mom)

  apply matrix control variate parameter (beta) to approximate HF moment

• void **apply** control (const RealMatrix &sum Hl, const RealMatrix &sum Hlm1, const RealMatrix &sum Ll, const RealMatrix &sum Llm1, const SizetArray &N_shared, const RealMatrix &sum Ll_refined, const RealMatrix &sum Llm1_refined, const SizetArray &N_refined, size_t lev, const RealVector &beta dot, const RealVector &gamma, RealVector &H_raw_mom)

  apply matrix control variate parameter (beta) to approximate HF moment

• void **export** all samples (String root prepend, const Model &model, size_t iter, size_t lev)

  export allSamples to tagged tabular file

• void **convert** moments (const RealMatrix &raw mom, RealMatrix &final mom)

  convert uncentered raw moments (multilevel expectations) to standardized moments

• void **compute** error estimates (IntRealMatrixMap &sum Ql, IntRealMatrixMap &sum Qlm1, IntIntPairRealMatrixMap &sum QlQlm1, Sizet2DArray &num Q)

  populate finalStatErrors for MLMC based on Q sums

• Real variance **Ysum** (Real sum Y, Real sum YY, size_t Nlq)

  compute variance from sum accumulators

• Real variance **Qsum** (Real sum Ql, Real sum Qlm1, Real sum QI/Ql, Real sum QI/Qlm1, Real sum Qlm1-Qlm1, size_t Nlq)

  compute variance from sum accumulators

• Real aggregate variance **Ysum** (const Real *sum Y, const Real *sum YY, const SizetArray &N_l)

  sum up variances across QoI (using sum YY with means from sum Y)

• Real aggregate variance **Qsum** (const Real *sum Ql, const Real *sum Qlm1, const Real *sum QI/Ql, const Real *sum QI/Qlm1, const Real *sum Qlm1/Qlm1, const SizetArray &N_l, const size_t &lev)

  sum up variances across QoI (using sum QI/Ql with means from sum Y)

• Real aggregate variance **Qsum** (const Real *sum Ql, const Real *sum Qlm1, const Real *sum QI/Ql, const Real *sum QI/Qlm1, const Real *sum Qlm1/Qlm1, const SizetArray &N_l, const size_t &lev, const size_t &qoi)

• Real aggregate **mse** Yvar (const Real *var Y, const SizetArray &N_l)

  sum up Monte Carlo estimates for mean squared error (MSE) across QoI using discrepancy variances

• Real aggregate **mse** Ysum (const Real *sum Y, const Real *sum YY, const SizetArray &N_l)

  sum up Monte Carlo estimates for mean squared error (MSE) across QoI using discrepancy sums

• Real aggregate **mse** Qsum (const Real *sum Ql, const Real *sum Qlm1, const Real *sum QI/Ql, const Real *sum QI/Qlm1, const Real *sum Qlm1/Qlm1, const SizetArray &N_l, const size_t &lev)

  sum up Monte Carlo estimates for mean squared error (MSE) across QoI using discrepancy sums

• Real aggregate **mse** Qsum (const Real *sum Ql, const Real *sum Qlm1, const Real *sum QI/Ql, const Real *sum QI/Qlm1, const Real *sum Qlm1/Qlm1, const SizetArray &N_l, const size_t &lev, const size_t &qoi)

• Real **sum** (const Real *vec, size_t vec_len) const

  compute sum of a set of observations
14.151. NONDMULTILEVELSAMPLING CLASS REFERENCE

- Real `average` (const Real `*` vec, size_t vec_len) const
  compute average of a set of observations
- Real `average` (const RealVector &vec) const
  compute average of a set of observations
- Real `average` (const SizetArray &sa) const
  compute average of a set of observations
- void `assign_static_member` (Real &conv_tol, size_t &qoi, RealVector &level_cost_vec, IntRealMatrixMap &sum_Ql, IntRealMatrixMap &sum_Qlm1, IntIntPairRealMatrixMap &sum_QlQlm1, RealVector &pilot_samples) const
- void `assign_static_member_problem18` (Real &var_L_exact, Real &var_H_exact, Real &mu_four_L_exact, Real &mu_four_H_exact, Real &Ax, RealVector &level_cost_vec) const

Static Private Member Functions

- static void `uncentered_to_centered` (Real rm1, Real rm2, Real rm3, Real rm4, Real &cm1, Real &cm2, Real &cm3, Real &cm4, size_t Nlq)
  convert uncentered (raw) moments to centered moments; biased estimators
- static void `uncentered_to_centered` (Real rm1, Real rm2, Real rm3, Real rm4, Real &cm1, Real &cm2, Real &cm3, Real &cm4)
  convert uncentered (raw) moments to centered moments; unbiased estimators
- static void `centered_to_standard` (Real cm1, Real cm2, Real cm3, Real cm4, Real &sm1, Real &sm2, Real &sm3, Real &sm4)
  convert centered moments to standardized moments
- static void `check_negative` (Real &cm)
  detect, warn, and repair a negative central moment (for even orders)
- static Real `unbiased_mean_product_pair` (const Real &sumQ1, const Real &sumQ2, const Real &sumQ1Q2, const size_t &Nlq)
  compute the unbiased product of two sampling means
- static Real `unbiased_mean_product_triplet` (const Real &sumQ1, const Real &sumQ2, const Real &sumQ3, const Real &sumQ1Q2, const Real &sumQ1Q3, const Real &sumQ2Q3, const Real &sumQ1Q2Q3, const size_t &Nlq)
  compute the unbiased product of three sampling means
- static Real `unbiased_mean_product_pairpair` (const Real &sumQ1, const Real &sumQ2, const Real &sumQ1Q2, const Real &sumQ1sq, const Real &sumQ2sq, const Real &sumQ1sqQ2, const Real &sumQ1sqQ2sq, const size_t &Nlq)
  compute the unbiased product of two pairs of products of sampling means
- static Real `var_of_var_ml_L0` (IntRealMatrixMap sum_Ql, IntRealMatrixMap sum_Qlm1, IntIntPairRealMatrixMap sum_QlQlm1, const size_t &Nlq, pilot, const Real &Nlq, const size_t &qoi, const bool &compute_gradient, Real &grad_g)
- static Real `var_of_var_ml_Lmax` (IntRealMatrixMap sum_Ql, IntRealMatrixMap sum_Qlm1, IntIntPairRealMatrixMap sum_QlQlm1, const size_t &Nlq, pilot, const Real &Nlq, const size_t &qoi, const bool &compute_gradient, Real &grad_g)
- static Real `var_of_var_ml_L` (IntRealMatrixMap sum_Ql, IntRealMatrixMap sum_Qlm1, IntIntPairRealMatrixMap sum_QlQlm1, const size_t &Nlq, pilot, const Real &Nlq, const size_t &qoi, const size_t &lev, const bool &compute_gradient, Real &grad_g)
static void target_var_objective_eval_optpp (int mode, int n, const RealVector &x, double &f, RealVector &grad_f, int &result_mode)

OPTPP definition.

static void target_var_constraint_eval_optpp (int mode, int n, const RealVector &x, RealVector &g, RealMatrix &grad_g, int &result_mode)

static void target_var_constraint_eval_logsparse_optpp (int mode, int n, const RealVector &x, RealVector &g, RealMatrix &grad_g, int &result_mode)

static void target_var_objective_eval_npsol (int &mode, int &n, double *x, double &f, double *gradf, int &nstate)

NPSOL definition (Wrapper using OPTPP implementation above under the hood)

static void target_var_constraint_eval_npsol (int &mode, int &m, int &n, int &ldJ, int *needc, double *x, double *g, double *grad_g, int &nstate)

static void target_var_constraint_eval_logsparse_npsol (int &mode, int &m, int &n, int &ldJ, int *needc, double *x, double *g, double *grad_g, int &nstate)

Private Attributes

Sizet3DArray NLev

total number of successful sample evaluations (excluding faults) for each model form, discretization level, and QoI

SizetArray pilotSamples

store the pilot_samples input specification, prior to run-time invocation of load_pilot_sample()

SizetArray randomSeedSeqSpec

user specification for seed_sequence

size_t mlmfIter

major iteration counter

short allocationTarget

store the allocation_target input specification, prior to run-time Options right now:

bool useTargetVarianceOptimizationFlag

option to switch on numerical optimization for solution of sample allocation of allocationTarget Variance

short qoiAggregation

store the qoi_aggregation_norm input specification, prior to run-time Options right now:

RealVector mcMSEIter0

mean squared error of mean estimator from pilot sample MC on HF model

Real equivHFEvals

equivalent number of high fidelity evaluations accumulated using samples across multiple model forms and/or discretization levels

bool finalCVRefinement

if defined, complete the final CV refinement when terminating MLCV based on maxIter (the total number of refinements beyond the pilot sample will be one more for CV than for ML). This approach is consistent with normal termination based on \( l_1 \)norm(\( \delta_N_{hf} \)) = 0.

bool exportSampleSets

if defined, export each of the sample increments in ML, CV, MLCV using tagged tabular files

unsigned short exportSamplesFormat

format for exporting sample increments using tagged tabular files
Additional Inherited Members

14.151.1 Detailed Description

Performs Multilevel Monte Carlo sampling for uncertainty quantification.

Multilevel Monte Carlo (MLMC) is a variance-reduction technique that utilizes lower fidelity simulations that have response QoI that are correlated with the high-fidelity response QoI.

14.151.2 Constructor & Destructor Documentation

NonDMultilevelSampling (ProblemDescDB & problem_db, Model & model)

standard constructor

This constructor is called for a standard letter-envelope iterator instantiation. In this case, set_db_list_nodes has been called and probDescDB can be queried for settings from the method specification.

References Dakota::abort_handler(), NonDMultilevelSampling::aggregated_models_mode(), Iterator::iterated-Model, Iterator::maxEvalConcurrency, NonDMultilevelSampling::NLev, NonDMultilevelSampling::pilotSamples, NonDMultilevelSampling::random_seed(), NonDSampling::randomSeed, NonDSampling::sampleType, NonDSampling::seedSpec, Model::subordinate_models(), and Model::surrogate_type().

14.151.3 Member Function Documentation

void pre_run () [protected], [virtual]

pre-run portion of run (optional); re-implemented by Iterators which can generate all Variables (parameter sets) a priori

pre-run phase, which a derived iterator may optionally reimplement; when not present, pre-run is likely integrated into the derived run function. This is a virtual function; when re-implementing, a derived class must call its nearest parent’s pre_run(), if implemented, typically before performing its own implementation steps.

Reimplemented from Analyzer.

References NonDMultilevelSampling::NLev, Analyzer::numFunctions, and NonDSampling::pre_run().

void core_run () [protected], [virtual]

The primary run function manages the general case: a hierarchy of model forms (from the ordered model fidelities within a HierarchSurrModel), each of which may contain multiple discretization levels.

Reimplemented from Iterator.

References NonDMultilevelSampling::control_variate_mc(), NonDMultilevelSampling::multilevel_control_variate_-mc_Qcorr(), NonDMultilevelSampling::multilevel_control_variate_mc_Ycorr(), NonDMultilevelSampling::multilevel_-mc_Qsum(), NonMultilevelSampling::multilevel_mc_Ysum(), and NonDMultilevelSampling::NLev.

void post_run (std::ostream & s) [protected], [virtual]

post-run portion of run (optional); verbose to print results; re-implemented by Iterators that can read all Variables_-Responses and perform final analysis phase in a standalone way

Post-run phase, which a derived iterator may optionally reimplement; when not present, post-run is likely integrated into run. This is a virtual function; when re-implementing, a derived class must call its nearest parent’s post_run(), typically after performing its own implementation steps.

Reimplemented from Analyzer.

References Analyzer::post_run(), and NonDSampling::update_final_statistics().
void print_results ( std::ostream & s, short results_state = FINAL_RESULTS ) [protected], [virtual]
print the final iterator results
This virtual function provides additional iterator-specific final results outputs beyond the function evaluation summary printed in finalize_run().
Reimplemented from Analyzer.
References NonD::archive.equiv_hf_evals(), NonDSampling::archive_moments(), NonDMultilevelSampling::equivHFEvals, Iterator::iteratedModel, NonDMultilevelSampling::NLev, NonDSampling::print_moments(), NonD::print_multilevel_evaluation_summary(), Model::response_labels(), NonDSampling::statsFlag, and Model::truth_model().

void multilevel_mc_Ysum ( unsigned short model_form ) [private]
Perform multilevel Monte Carlo across the discretization levels for a particular model form using discrepancy accumulators (sum_Y)
This function performs "geometrical" MLMC on a single model form with multiple discretization levels.
References NonDMultilevelSampling::accumulate_ml_Ysums(), Model::active_model_key(), NonDMultilevelSampling::aggregate_mse_Ysum(), NonDMultilevelSampling::aggregate_variance_Ysum(), NonDMultilevelSampling::assign_specification_sequence(), NonDMultilevelSampling::average(), NonDMultilevelSampling::configure_indices(), Iterator::convergenceTol, NonDMultilevelSampling::convert_moments(), NonDMultilevelSampling::equivHFEvals, Analyzer::evaluate_parameter_sets(), NonDMultilevelSampling::export_all_samples(), NonDMultilevelSampling::exportSampleSets, NonDSampling::get_parameter_sets(), NonDMultilevelSampling::initialize_ml_Ysums(), Iterator::iteratedModel, NonDMultilevelSampling::level_cost(), NonD::load_pilot_sample(), Iterator::maxIterations, NonDMultilevelSampling::mlmfFilter, NonD::momentStats, NonDMultilevelSampling::NLev, Analyzer::numFunctions, NonDSampling::numSamples, NonD::one_sided_delta(), Iterator::outputLevel, NonDMultilevelSampling::pilot_Samples, Model::solution_level_costs(), Model::solution_levels(), and Model::truth_model().
Referenced by NonDMultilevelSampling::core_run().

void multilevel_mc_Qsum ( unsigned short model_form ) [private]
Perform multilevel Monte Carlo across the discretization levels for a particular model form using QoI accumulators (sum_Q)
This function performs "geometrical" MLMC on a single model form with multiple discretization levels.
References Dakota::abort_handler(), NonDMultilevelSampling::accumulate_ml_Qsums(), Model::active_model_key(), NonDMultilevelSampling::aggregate_mse_Qsum(), NonDMultilevelSampling::aggregate_variance_Qsum(), NonDMultilevelSampling::allocationTarget, NonDMultilevelSampling::assign_specification_sequence(), NonDMultilevelSampling::average(), NonDMultilevelSampling::centered_to_standard(), NonDMultilevelSampling::check_negative(), NonDMultilevelSampling::compute_error_estimates(), NonDMultilevelSampling::configure_indices(), Iterator::convergenceTol, NonDMultilevelSampling::equivHFEvals, Analyzer::evaluate_parameter_sets(), NonDMultilevelSampling::export_all_samples(), NonDMultilevelSampling::exportSampleSets, NonD::finalMomentsType, NonDSampling::get_parameter_sets(), NonDMultilevelSampling::initialize_ml_Qsums(), Iterator::iteratedModel, NonDMultilevelSampling::level_cost(), NonD::load_pilot_sample(), Iterator::maxIterations, NonDMultilevelSampling::mlmfFilter, NonD::momentStats, NonDMultilevelSampling::NLev, Analyzer::numFunctions, NonDSampling::numSamples, NonD::one_sided_delta(), Iterator::outputLevel, NonDMultilevelSampling::pilot_Samples, NonD::truth_model(), NonDMultilevelSampling::uncentered_to_centered(), and NonDMultilevelSampling::useTargetVarianceOptimizationFlag.
Referenced by NonDMultilevelSampling::core_run().
void control_variate_mc ( const UShortArray & active_key ) [private]

Perform control variate Monte Carlo across two model forms.

This function performs control variate MC across two combinations of model form and discretization level.

References NonDMultilevelSampling::accumulate_cv_sums(), Model::active_model_key(), NonDMultilevelSampling::aggregated_models_mode(), Iterator::convergenceTol, NonDMultilevelSampling::convert_moments(), NonDMultilevelSampling::cv_raw_moments(), NonDMultilevelSampling::equivHFEvals, NonDMultilevelSampling::eval_ratio(), NonDMultilevelSampling::initialize_cv_sums(), Iterator::iteratedModel, NonDMultilevelSampling::lf_increment(), NonD::load_pilot_sample(), Iterator::maxIterations, NonDMultilevelSampling::mlmIter, NonD::momentStats, NonDMultilevelSampling::MSE_ratio(), NonDMultilevelSampling::NLev, Analyzer::numFunctions, NonDSampling::numSamples, NonDMultilevelSampling::pilotSamples, NonDMultilevelSampling::shared_increment(), Model::solution_level_cost(), Model::surrogate_model(), Model::truth_model(), and NonDMultilevelSampling::uncorrected_surrogate_mode().

Referenced by NonDMultilevelSampling::core_run().

void multilevel_control_variate_mc_Ycorr ( unsigned short lf_model_form, unsigned short hf_model_form ) [private]

Perform multilevel Monte Carlo across levels in combination with control variate Monte Carlo across model forms at each level; CV computes correlations for Y (LH correlations for level discrepancies).

This function performs "geometrical" MLMC across discretization levels for the high fidelity model form where CVMC is employed across two model forms to exploit correlation in the discrepancies at each level (Y_l).

References NonDMultilevelSampling::accumulate_ml_Ysums(), NonDMultilevelSampling::accumulate_mlcv_Ysums(), Model::active_model_key(), NonDMultilevelSampling::aggregate_mse_Ysum(), NonDMultilevelSampling::aggregate_mse_Yvar(), NonDMultilevelSampling::aggregate_variance_Ysum(), Analyzer::allResponses, NonDMultilevelSampling::assign_specification_sequence(), NonDMultilevelSampling::average(), NonDMultilevelSampling::configure_indices(), Iterator::convergenceTol, NonDMultilevelSampling::convert_moments(), NonDMultilevelSampling::cv_raw_moments(), NonDMultilevelSampling::equivHFEvals, NonDMultilevelSampling::eval_ratio(), Analyzer::evaluate_parameter_sets(), NonDMultilevelSampling::export_all_samples(), NonDMultilevelSampling::export_sampleSets(), NonDSampling::get_parameter_sets(), NonDMultilevelSampling::initialize_mlcv_sums(), Iterator::iteratedModel, NonDMultilevelSampling::level_cost(), NonDMultilevelSampling::lf_increment(), NonD::load_pilot_sample(), Iterator::maxIterations, NonDMultilevelSampling::mlmIter, NonD::momentStats, NonDMultilevelSampling::NLev, Analyzer::numFunctions, NonDSampling::numSamples, NonD::one_sided_delta(), Iterator::outputLevel, NonDMultilevelSampling::pilotSamples, Model::solution_level_costs(), Model::solution_levels(), NonDMultilevelSampling::sum(), Model::surrogate_model(), and Model::truth_model().

Referenced by NonDMultilevelSampling::core_run().

void multilevel_control_variate_mc_Qcorr ( unsigned short lf_model_form, unsigned short hf_model_form ) [private]

Perform multilevel Monte Carlo across levels in combination with control variate Monte Carlo across model forms at each level; CV computes correlations for Q (LH correlations for QoI)

This function performs "geometrical" MLMC across discretization levels for the high fidelity model form where CVMC is employed across two model forms. It generalizes the Y_l correlation case to separately target correlations for each QoI level embedded within the level discrepancies.

References NonDMultilevelSampling::accumulate_ml_Ysums(), NonDMultilevelSampling::accumulate_mlcv_Qsums(), Model::active_model_key(), NonDMultilevelSampling::aggregate_mse_Ysum(), NonDMultilevelSampling::aggregate_mse_Yvar(), NonDMultilevelSampling::aggregate_variance_Ysum(), Analyzer::allResponses, NonDMultilevelSampling::assign_specification_sequence(), NonDMultilevelSampling::average(), NonDMultilevelSampling::configure_indices(), Iterator::convergenceTol, NonDMultilevelSampling::convert_moments(), NonDMultilevelSampling::
CHAPTER 14. CLASS DOCUMENTATION

Sampling::cv_raw_moments(), NonDMultilevelSampling::equivHFEns, NonDMultilevelSampling::eval_ratio(), Analyzer::evaluate_parameter_sets(), NonDMultilevelSampling::export_all_samples(), NonDMultilevelSampling::exportSampleSets, NonDSampling::get_parameter_sets(), NonDMultilevelSampling::initialize_mlcv_sums(), Iterator::iteratedModel, NonDMultilevelSampling::level_cost(), NonDMultilevelSampling::lf_increment(), NonD::load_pilot_sample(), Iterator::maxIterations, NonDMultilevelSampling::mlmfIter, NonD::momentStats, NonDMultilevelSampling::NLev, Analyzer::numFunctions, NonDSampling::numSamples, NonD::one_sided_delta(), Iterator::outputLevel, NonDMultilevelSampling::pilotSamples, Model::solution_level_costs(), Model::solution_levels(), NonDMultilevelSampling::sum(), Model::surrogate_model(), and Model::truth_model.

Referenced by NonDMultilevelSampling::core_run().

int random_seed( size_t index) const [inline], [private]

extract current random seed from randomSeedSeqSpec
extract an active seed from a seed sequence
References NonDMultilevelSampling::mlmfIter, NonDMultilevelSampling::randomSeedSeqSpec, and NonDSampling::varyPattern.
Referenced by NonDMultilevelSampling::assign_specification_sequence(), and NonDMultilevelSampling::NonDMultilevelSampling().

void uncentered_to_centered ( Real rm1, Real rm2, Real rm3, Real rm4, Real & cm1, Real & cm2, Real & cm3, Real & cm4, size_t Nlq ) [inline], [static], [private]

convert uncentered (raw) moments to centered moments; biased estimators
For single-level moment calculations with a scalar Nlq.
Referenced by NonDMultilevelSampling::compute_error_estimates(), NonDMultilevelSampling::convert_moments(), and NonDMultilevelSampling::multilevel_mc_Qsum().

void uncentered_to_centered ( Real rm1, Real rm2, Real rm3, Real rm4, Real & cm1, Real & cm2, Real & cm3, Real & cm4 ) [inline], [static], [private]

convert uncentered (raw) moments to centered moments; unbiased estimators
For single-level moment calculations with a scalar Nlq.

14.151.4 Member Data Documentation

short allocationTarget [private]

store the allocation_target input specification, prior to run-time Options right now:

• Mean = First moment (Mean)
• Variance = Second moment (Variance or standard deviation depending on moments central or standard)

Referenced by NonDMultilevelSampling::multilevel_mc_Qsum().

short qoiAggregation [private]

store the qoi_aggregation_norm input specification, prior to run-time Options right now:

• sum = aggregate the variance over all QoIs, compute samples from that
• max = take maximum sample allocation over QoIs for each level
Nonintrinsic stochastic collocation approaches to uncertainty quantification.

Inheritance diagram for NonDMultilevelStochCollocation:

```
NonDMultilevelStochCollocation
  NonDStochCollocation
    NonDExpansion
      NonD
        Analyzer
          Iterator
```

**Public Member Functions**

- **NonDMultilevelStochCollocation (ProblemDescDB &problem_db, Model &model)**
  - *standard constructor*
- **NonDMultilevelStochCollocation (Model &model, short exp_coeffs, approach, const UShortArray &num_int_seq, const RealVector &dim pref, short u_space_type, short refine_type, short refine_control, short covar_control, short ml_alloc_cntl, short ml_discrep, short rule_nest, short rule_growth, bool piecewise_basis, bool use derivs)**
  - *alternate constructor*
- **∼NonDMultilevelStochCollocation ()**
  - *destructor*
- **bool resize ()**
  - *reinitializes iterator based on new variable size*

**Protected Member Functions**

- **void core_run ()**
  - *perform a forward uncertainty propagation using PCE/SC methods*
- **int random_seed () const**
  - *return specification for random seed (may be part of a sequence specification)*
• int first_seed () const
  
  return first seed in sequence specification (defaults to random_seed())

• void assign_specification_sequence ()
  
  assign the current values from the input specification sequence

• void increment_specification_sequence ()
  
  increment the input specification sequence and assign values

• void print_results (std::ostream &s, short results_state=FINAL_RESULTS)
  
  print the final statistics

**Private Attributes**

• UShortArray quadOrderSeqSpec
  
  user request of quadrature order

• UShortArray ssgLevelSeqSpec
  
  user request of sparse grid level

• size_t sequenceIndex
  
  sequence index for {quadOrder, ssgLevel}SeqSpec

**Additional Inherited Members**

14.152.1 Detailed Description

Nonintrusive stochastic collocation approaches to uncertainty quantification.

The NonDmultilevelStochCollocation class uses a stochastic collocation (SC) approach to approximate the
effect of parameter uncertainties on response functions of interest. It utilizes the InterpPolyApproximation class
to manage multidimensional Lagrange polynomial interpolants.

14.152.2 Constructor & Destructor Documentation

NonDmultilevelStochCollocation ( ProblemDescDB & problem_db, Model & model )

standard constructor

This constructor is called for a standard letter-envelope iterator instantiation using the ProblemDescDB.

References Response::active_set(), NonDExpansion::assign_discrepancy_mode(), NonDExpansion::assign待ち hysterical_response_mode(), Model::assign_rep(), ParallelLibrary::command_line_check(), NonDStochCollocation::config_approximation_type(), NonDStochCollocation::config_integration(), NonDExpansion::construct_expansion_sampler(), Model::current_response(), ActiveSet::derivative_vector(), ProblemDescDB::get_bool(), ProblemDescDB::get_iv(), ProblemDescDB::get_rv(), ProblemDescDB::get_short(), ProblemDescDB::get_string(), ProblemDescDB::get_uSpaceModel(), NonDStochCollocation::initialize_uSpaceModel(), Iterator::iteratedModel, NonDExpansion::numSamplesOnModel, Iterator::outputLevel, Iterator::parallelLib, Iterator::probDescDB, Model::qoi(), NonDMultilevelStochCollocation::quadOrderSeqSpec, NonDStochCollocation::resolve_inputs(), NonDmultilevelStochCollocation::sequenceIndex, NonDMultilevelStochCollocation::ssgLevelSeqSpec, and NonDExpansion::uSpaceModel.
NonDMultilevelStochCollocation ( Model & model, short exp_coeffs_approach, const UShortArray & num_int_seq, const RealVector & dim_pref, short u_space_type, short refine_type, short refine_control, short covar_control, short ml_alloc_cntl, short ml_discrep, short rule_nest, short rule_growth, bool piecewise_basis, bool use_derivs )

alternate constructor

This constructor is used for helper iterator instantiation on the fly.

References Response::active_set(), NonDExpansion::assign_discrepancy_mode(), NonDExpansion::assign_hierarchical_response_mode(), Model::assign_rep(), NonDStochCollocation::config_approximation_type(), NonDStochCollocation::config_integration(), Model::current_response(), ActiveSet::derivative_vector(), NonDExpansion::expansionCoeffsApproach, NonDStochCollocation::initialize_u_space_model(), Iterator::iteratedModel, Iterator::outputLevel, Model::qoi(), NonDMultilevelStochCollocation::quadOrderSeqSpec, NonDStochCollocation::resolve_inputs(), NonDMultilevelStochCollocation::sequenceIndex, NonDMultilevelStochCollocation::ssgLevelSeqSpec, and NonDExpansion::uSpaceModel.

14.152.3 Member Function Documentation

```c++
void increment_specification_sequence ( ) [protected], [virtual]
```

increment the input specification sequence and assign values

Default implementation redefined by Multilevel derived classes.

Reimplemented from NonDExpansion.

References Dakota::abort_handler(), NonDExpansion::expansionCoeffsApproach, Iterator::iterator_rep(), NonDMultilevelStochCollocation::quadOrderSeqSpec, NonDQuadrature::quadrature_order(), NonDSparseGrid::reset(), NonDQuadrature::reset(), NonDMultilevelStochCollocation::sequenceIndex, NonDSparseGrid::sparse_grid_level(), NonDMultilevelStochCollocation::ssgLevelSeqSpec, Model::subordinate_iterator(), and NonDExpansion::uSpaceModel.

The documentation for this class was generated from the following files:

- NonDMultilevelStochCollocation.hpp
- NonDMultilevelStochCollocation.cpp

14.153 NonDMUQBayesCalibration Class Reference

Dakota interface to MUQ (MIT Uncertainty Quantification) library.

Inheritance diagram for NonDMUQBayesCalibration:
Public Member Functions

- `NonDMUQBayesCalibration (ProblemDescDB &problem_db, Model &model)`
  *standard constructor*
- `~NonDMUQBayesCalibration ()`
  *destructor*

Protected Member Functions

- `void calibrate ()`
- `void print_results (std::ostream &s, short results_state=FINAL_RESULTS)`
  *print the final iterator results*
- `void print_variables (std::ostream &s, const RealVector &c_vars)`
  *convenience function to print calibration parameters, e.g., for MAP / best parameters*
- `void cache_chain ()`
  *cache the chain to acceptanceChain and acceptedFnVals*
- `void log_best ()`
  *log at most batchSize best chain points into bestSamples*

Protected Attributes

- `std::shared_ptr<muq::Modeling::WorkGraph> workGraph`
- `std::shared_ptr<muq::Modeling::IdentityOperator> parameterPtr`
- `std::shared_ptr<muq::Modeling::Distribution> distPtr`
- `std::shared_ptr<muq::Modeling::DensityProduct> posteriorPtr`
- `std::shared_ptr<muq::Likhelihood> MUQLikelihoodPtr`
- `std::shared_ptr<muq::Prior> MUQPriorPtr`
14.153. NONDMUQBayesCalibration Class Reference

- `std::shared_ptr<muq::SamplingAlgorithms::SingleChainMCMC> mcmc`
- `std::shared_ptr<muq::SamplingAlgorithms::SampleCollection> samps`
- `String mcmcType`
  
  MCMC type ("dram" or "delayed_rejection" or "adaptive_metropolis" or "metropolis_hastings" or "multilevel", within QUESO)
- `unsigned int numBestSamples`

  number of best samples (max log posterior values) to keep

Static Protected Attributes

- `static NonDMUQBayesCalibration * nonDMUQInstance`

  Pointer to current class instance for use in static callback functions.

Friends

- `class MUQLikelihood`
- `class MUQPrior`

Additional Inherited Members

14.153.1 Detailed Description

Dakota interface to MUQ (MIT Uncertainty Quantification) library.

This class performs Bayesian calibration using the MUQ library.

14.153.2 Constructor & Destructor Documentation

NonDMUQBayesCalibration (ProblemDescDB & problem_db, Model & model)

standard constructor

This constructor is called for a standard letter-envelope iterator instantiation. In this case, set_db_list_nodes has been called and probDescDB can be queried for settings from the method specification.

14.153.3 Member Function Documentation

void calibrate() [protected], [virtual]

Perform the uncertainty quantification

Implements NonDBayesCalibration.

References NonDMUQBayesCalibration::cache_chain(), NonDBayesCalibration::chainSamples, NonDBayesCalibration::compute_statistics(), WorkdirHelper::create_directory(), NonDBayesCalibration::initialize_model(), NonDMUQBayesCalibration::log_best(), NonDMUQBayesCalibration::mcmcType, NonDMUQBayesCalibration::nonDMUQInstance, Analyzer::numContinuousVars, Iterator::outputLevel, NonDBayesCalibration::randomSeed, and WorkdirHelper::rel_to_abs().
void print_results ( std::ostream & s, short results_state = FINAL_RESULTS ) [protected], [virtual]

print the final iterator results

This virtual function provides additional iterator-specific final results outputs beyond the function evaluation summary printed in finalize_run().

Reimplemented from NonDBayesCalibration.

References NonDBayesCalibration::bestSamples, NonDCalibration::expData, Dakota::HALF_LOG_2PI, ExperimentData::half_log_cov_determinant(), NonDBayesCalibration::log_prior_density(), Model::num_primary_fns(), NonDBayesCalibration::numHyperparams, NonDBayesCalibration::obsErrorMultiplierMode, NonDBayesCalibration::print_results(), NonDMUQBayesCalibration::print_variables(), NonDBayesCalibration::residualModel, and Dakota::write_precision.

void cache_chain ( ) [protected]

cache the chain to acceptanceChain and acceptedFnVals

Populate all of acceptanceChain(num_params, chainSamples) acceptedFnVals(numFunctions, chainSamples)

References Dakota::abort_handler(), NonDBayesCalibration::acceptanceChain, NonDBayesCalibration::acceptedFnVals, Response::active_set(), Model::active_variables(), NonDBayesCalibration::chainSamples, Variables::continuous_variables(), Response::copy(), Variables::copy(), Model::current_response(), Model::current_variables(), Dakota::data_pairs, Model::evaluate(), Response::function_values(), Model::interface_id(), Dakota::lookup_by_val(), NonDBayesCalibration::mcmcModel, NonDBayesCalibration::mcmcModelHasSurrogate, Model::model_type(), NonDMUQBayesCalibration::nonDMUQInstance, Analyzer::numContinuousVars, Analyzer::numFunctions, NonDBayesCalibration::numHyperparams, Iterator::outputLevel, Model::probability_transformation(), ActiveSet::request_values(), NonDBayesCalibration::standardizedSpace, and ParamResponsePair::variables().

Referenced by NonDMUQBayesCalibration::calibrate().

The documentation for this class was generated from the following files:

- NonDMUQBayesCalibration.hpp
- NonDMUQBayesCalibration.cpp

14.154 NonDPOFDarts Class Reference

Base class for POF Dart methods within DAKOTA/UQ.

Inheritance diagram for NonDPOFDarts:

```
  NonDPOFDarts
  |     |
  |     |  Iterator
  |     |
  |     |  Analyzer
  |     |  NonD
  |     |
  |     |  NonDPOFDarts
```
Public Member Functions

- NonDPOFDarts (ProblemDescDB &problem_db, Model &model)
  constructor
- ~NonDPOFDarts ()
  destructor
- bool resize ()
  reinitializes iterator based on new variable size
- void core_run ()
  perform POFDart analysis and return probability of failure

Protected Member Functions

- void initiate_random_number_generator (unsigned long x)
  POF Darts Methods.
- double generate_a_random_number ()
- void init_pof_darts ()
- void exit_pof_darts ()
- void execute (size_t kd)
- void print_results (std::ostream &s, short results_state=FINAL_RESULTS)
  print the final statistics
- void classical_dart_throwing_games (size_t game_index)
- void line_dart_throwing_games (size_t game_index)
- bool valid_dart (double *x)
- bool valid_line_flat (size_t flat_dim, double *flat_dart)
- void add_point (double *x)
- void compute_response (double *x)
- void verify_neighbor_consistency ()
- bool add_neighbor (size_t ipoint, size_t inneighbor)
- void retrieve_neighbors (size_t ipoint, bool update_point_neighbors)
- void sample_furthest_vertex (size_t ipoint, double *fv)
- void update_global_L ()
- void assign_sphere_radius_POF (size_t isample)
- void shrink_big_spheres ()
- double area_triangle (double x1, double y1, double x2, double y2, double x3, double y3)
- void initialize_surrogates ()
- void add_surrogate_data (const Variables &vars, const Response &resp)
- void build_surrogate ()
- double eval_surrogate (size_t fn_index, double *vin)
- void estimate_pof_surrogate ()
- bool trim_line_using_Hyperplane (size_t num_dim, double *st, double *end, double *qH, double *nH)
- double f_true (double *x)
- void plot_vertices_2d (bool plot_true_function, bool plot_suurogate)
- void plot_neighbors ()
Protected Attributes

- int samples
- int seed
- int emulatorSamples
- String lipschitzType
- RealRealPairArray extremeValues
- double Q[1220]
- int indx
- double cc
- double c
- double zc
- double zx
- double zy
- size_t qlen
- bool _eval_error
- size_t _test_function
- size_t _n_dim
- double * _xmin
- double * _xmax
- double _diag
- double _failure_threshold
- double _num_darts
- double _num_successive_misses_p
- double _num_successive_misses_m
- double _max_num_successive_misses
- double _accepted_void_ratio
- size_t _num_inserted_points
- size_t _total_budget
- double ** _sample_points
- size_t ** _sample_neighbors
- double * _sample_vsize
- double _max_vsize
- double * _dart
- size_t _flat_dim
- size_t * _line_flat
- size_t _num_flat_segments
- double * _line_flat_start
- double * _line_flat_end
- double * _line_flat_length
- double _safety_factor
- double * _Lip
- double ** _fval
- size_t _active_response_function
- bool _use_local_L
Additional Inherited Members

14.154.1 Detailed Description

Base class for POF Dart methods within DAKOTA/UQ.

The NonDPOFDart class implements the calculation of a failure probability for a specified threshold for a specified response function using the concepts developed by Mohamed Ebeida. The approach works by throwing down a number of Poisson disk samples of varying radii, and identifying each disk as either in the failure or safe region. The center of each disk represents a "true" function evaluation. kd-darts are used to place additional points, in such a way to target the failure region. When the disks cover the space sufficiently, Monte Carlo methods or a box volume approach is used to calculate both the lower and upper bounds on the failure probability.

The documentation for this class was generated from the following files:

- NonDPOFDarts.hpp
- NonDPOFDarts.cpp

14.155 NonDPolynomialChaos Class Reference

Nonintrusive polynomial chaos expansion approaches to uncertainty quantification.

Inheritance diagram for NonDPolynomialChaos:

```
  NonDPolynomialChaos
  |__________________________|
  |                            |
  v                            v
  Analyzer                     NonD
  |__________________________|
  |                            |
  v                            v
  NonDExpansion                NonDPolynomialChaos
  |__________________________|
  |                            |
  v                            v
  NonDMultilevelPolynomialChaos
```

Public Member Functions

- **NonDPolynomialChaos** (ProblemDescDB &problem_db, Model &model)
  
  *standard constructor*

- **NonDPolynomialChaos** (Model &model, short exp coeffs approach, unsigned short num_int, const RealVector &dim_pref, short u_space_type, short refine_type, short refine_control, short covar_control, short rule nest, short rule_growth, bool piecewise_basis, bool use derivs)
  
  *alternate constructor for numerical integration (tensor, sparse, cubature)*

- **NonDPolynomialChaos** (Model &model, short exp coeffs approach, unsigned short exp order, const RealVector &dim_pref, size_t colloc pts, Real colloc ratio, int seed, short u_space_type, short refine type, short refine_control, short covar_control, bool piecewise basis, bool use derivs, bool cv_flag, const String &import build pts file, unsigned short import build format, bool import build active only)
alternate constructor for regression (least squares, CS, OLI)

- NonDPolynomialChaos ()
  destructor
- bool resize ()
  reinitializes iterator based on new variable size

Protected Member Functions

- NonDPolynomialChaos (unsigned short method_name, ProblemDescDB &problem_db, Model &model)
  base constructor for DB construction of multilevel/multifidelity PCE (method_name is not necessary, rather it is just a convenient overload allowing the derived ML PCE class to bypass the standard PCE ctor)
- NonDPolynomialChaos (unsigned short method_name, Model &model, short exp_coeffs_approach, const RealVector &dim_pref, short u_space_type, short refine_type, short refine_control, short covar_control, short ml_alloc_control, short ml_discrep, short rule_nest, short rule_growth, bool piecewise_basis, bool use_derivs)
  base constructor for lightweight construction of multifidelity PCE using numerical integration
- NonDPolynomialChaos (unsigned short method_name, Model &model, short exp_coeffs_approach, const RealVector &dim_pref, short u_space_type, short refine_type, short refine_control, short covar_control, const SizetArray &colloc_pts_seq, Real colloc_ratio, short ml_alloc_control, short ml_discrep, bool piecewise_basis, bool use_derivs, bool cv_flag)
  base constructor for lightweight construction of multilevel PCE using regression
- void derived_init_communicators (ParLevLIter pl_iter)
  derived class contributions to initializing the communicators associated with this Iterator instance
- void derived_set_communicators (ParLevLIter pl_iter)
  derived class contributions to setting the communicators associated with this Iterator instance
- void derived_free_communicators (ParLevLIter pl_iter)
  derived class contributions to freeing the communicators associated with this Iterator instance
- void resolve_inputs (short &u_space_type, short &data_order)
  perform error checks and mode overrides
- void initialize_u_space_model ()
  initialize uSpaceModel polynomial approximations with PCE/SC data
- size_t collocation_points () const
  return specification for number of collocation points (may be part of a sequence specification)
- void compute_expansion ()
  form the expansion by calling uSpaceModel.build_approximation()
- void select_refinement_points (const RealVectorArray &candidate_samples, unsigned short batch_size, RealMatrix &best_samples)
  evaluate allSamples for inclusion in the (PCE regression) approximation and retain the best set (well spaced) of size batch_size
- void select_refinement_points_deprecated (const RealVectorArray &candidate_samples, unsigned short batch_size, RealMatrix &best_samples)
- void append_expansion (const RealMatrix &samples, const IntResponseMap &resp_map)
  append new data to uSpaceModel and, when appropriate, update expansion order
- void update_samples_from_order_increment ()
  update numSamplesOnModel after an order increment
- void sample_allocation_metric (Real &sparsity_metric, Real power)
accumulate one of the level metrics for \{RIPRANK\} SAMPLING cases

- void print_results (std::ostream &s, short results_state=FINAL_RESULTS)
  print the final coefficients and final statistics
- void print_coefficients (std::ostream &s)
  print the PCE coefficient array for the orthogonal basis
- void export_coefficients ()
  export the PCE coefficient array to expansionExportFile
- void archive_coefficients ()
  archive the PCE coefficient array for the orthogonal basis

- bool config_integration (unsigned short quad_order, unsigned short ssg_level, unsigned short cub_int, Iterator &u_space_sampler, Model &g_u_model, String &approx_type)
  configure u_space_sampler and approx_type based on numerical integration specification
- bool config_expectation (size_t exp_samples, unsigned short sample_type, int seed, const String &rng, Iterator &u_space_sampler, Model &g_u_model, String &approx_type)
  configure u_space_sampler and approx_type based on expansion_samples specification
- bool config_regression (const UShortArray &exp_orders, size_t colloc_pts, Real colloc_ratio_order, short regress_type, short ls_regress_type, const UShortArray &tensor_grid_order, unsigned short sample_type, int seed, const String &rng, const String &pt_reuse, Iterator &u_space_sampler, Model &g_u_model, String &approx_type)
  configure u_space_sampler and approx_type based on regression specification
- void increment_order_from_grid ()
  define an expansion order that is consistent with an advancement in structured/unstructured grid level/density
- void ratio_samples_to_order (Real colloc_ratio, int num_samples, UShortArray &exp_order, bool less_than_or_equal)
  convert collocation ratio and number of samples to expansion order

Protected Attributes

- short uSpaceType
  user requested expansion type
- unsigned short cubIntSpec
  cubature integrand
- bool crossValidation
  flag for use of cross-validation for selection of parameter settings in regression approaches
- bool crossValidNoiseOnly
  flag to restrict cross-validation to only estimate the noise tolerance in order to manage computational cost
- String importBuildPointsFile
  user-specified file for importing build points
- String expansionImportFile
  filename for import of chaos coefficients
- String expansionExportFile
  filename for export of chaos coefficients
Private Member Functions

- void order_to_dim_preference (const UShortArray &order, unsigned short &p, RealVector &dim_pref)
  convert an isotropic/anisotropic expansion order vector into a scalar plus a dimension preference vector

Private Attributes

- RealVector noiseTols
  noise tolerance for compressive sensing algorithms; vector form used in cross-validation
- Real l2Penalty
  L2 penalty for LASSO algorithm (elastic net variant)
- unsigned short numAdvance
  number of frontier expansions per iteration with the ADAPTED_BASIS_EXPANDING_FRONT approach
- unsigned short expOrderSpec
  user specification for expansion order (array for multifidelity)
- size_t collocPtsSpec
  user specification for collocation points (array for multifidelity)
- size_t expSamplesSpec
  user specification for expansion samples (array for multifidelity)
- unsigned short quadOrderSpec
  user request of quadrature order
- unsigned short ssgLevelSpec
  user request of sparse grid level
- RealMatrix pceGradsMeanX
  derivative of the PCE with respect to the x-space variables evaluated at the means (used as uncertainty importance metrics)
- bool normalizedCoeffOutput
  user request for use of normalization when outputting PCE coefficients

Additional Inherited Members

14.155.1 Detailed Description

Nonintrusive polynomial chaos expansion approaches to uncertainty quantification.

The NonDPolynomialChaos class uses a polynomial chaos expansion (PCE) approach to approximate the
effect of parameter uncertainties on response functions of interest. It utilizes the OrthogPolyApproximation class
to manage multiple types of orthogonal polynomials within a Wiener-Askey scheme to PCE. It supports PCE
coefficient estimation via sampling, quadrature, point-collocation, and file import.

14.155.2 Constructor & Destructor Documentation

NonDPolynomialChaos (ProblemDescDB & problem_db, Model & model)

standard constructor

This constructor is called for a standard letter-envelope iterator instantiation using the ProblemDescDB.
NonDPolynomialChaos ( Model & model, short exp_coeffs_approach, unsigned short num_int, const RealVector & dim_pref, short u_space_type, short refine_type, short refine_control, short rule_nest, short rule_growth, bool piecewise_basis, bool use_derivs )

alternate constructor for numerical integration (tensor, sparse, cubature)

This constructor is used for helper iterator instantiation on the fly that employ numerical integration (quadrature, sparse grid, cubature).

References Dakota::abort_handler(), Response::active_set(), Model::assign_rep(), NonDPolynomialChaos::collocPtsSpec, Parallel::command_line_check(), NonDPolynomialChaos::config_expectation(), NonDPolynomialChaos::config_integration(), NonDPolynomialChaos::config_regression(), NonDExpansion::configure_expansion_orders(), NonDExpansion::construct_expansion_sampler(), NonPolynomialChaos::cubIntSpec, ActiveSet::derivative_vector(), NonDExpansion::dimPrefSpec, NonDPolynomialChaos::expOrderSpec, NonDPolynomialChaos::expSamplesSpec, ProblemDescDB::get_bool(), ProblemDescDB::get_iv(), ProblemDescDB::get_real(), ProblemDescDB::get_short(), ProblemDescDB::get_string(), ProblemDescDB::get_usa(), ProblemDescDB::get_ushort(), NonDPolynomialChaos::importBuildPointsFile, NonDPolynomialChaos::initialize_u_space_model(), Iterator::iteratedModel, NonDExpansion::numSamplesOnModel, Iterator::outputLevel, Iterator::parallelLib, NonDPolynomialChaos::quadOrderSpec, NonDExpansion::randomSeed, NonDPolynomialChaos::resolve_inputs(), NonDPolynomialChaos::ssgLevelSpec, NonDExpansion::uSpaceModel, and NonDPolynomialChaos::uSpaceType.

NonDPolynomialChaos ( Model & model, short exp_coeffs_approach, unsigned short exp_order, const RealVector & dim_pref, size_t colloc_pts, Real colloc_ratio, int seed, short u_space_type, short refine_control, short covar_control, bool piecewise_basis, bool use_derivs, bool cv_flag, const String & import_build_pts_file, unsigned short import_build_format, bool import_build_active_only )

alternate constructor for regression (least squares, CS, OLI)

This constructor is used for helper iterator instantiation on the fly that employ regression (least squares, CS, OLI).

References Response::active_set(), Model::assign_rep(), NonDPolynomialChaos::collocPtsSpec, NonDPolynomialChaos::config_regression(), NonDExpansion::configure_expansion_orders(), Model::current_response(), ActiveSet::derivative_vector(), NonDPolynomialChaos::initialize_u_space_model(), Iterator::iteratedModel, Iterator::outputLevel, Model::qoi(), NonDPolynomialChaos::resolve_inputs(), NonDExpansion::uSpaceModel, and NonDPolynomialChaos::uSpaceType.

NonDPolynomialChaos ( unsigned short method_name, ProblemDescDB & problem_db, Model & model ) [protected]

base constructor for DB construction of multilevel/multifidelity PCE (method_name is not necessary, rather it is just a convenient overload allowing the derived ML PCE class to bypass the standard PCE ctor)

This constructor is called by derived class constructors that customize the object construction.
NonDPolynomialChaos (unsigned short method name, Model & model, short exp_coeffs_approach, const RealVector & dim_pref, short u_space_type, short refine_type, short refine_control, short covar_control, short ml_alloc_control, short ml_discrep, short rule_nest, short rule_growth, bool piecewise_basis, bool use_derivs) [protected]

base constructor for lightweight construction of multifidelity PCE using numerical integration

This constructor is called by derived class constructors for lightweight instantiations that employ numerical integration (quadrature, sparse grid, cubature).

References NonDExpansion::multilevAllocControl, and NonDExpansion::multilevDiscrepEmulation.

NonDPolynomialChaos (unsigned short method name, Model & model, short exp_coeffs_approach, const RealVector & dim_pref, short u_space_type, short refine_type, short refine_control, short covar_control, const SizetArray & colloc_pts_seq, Real colloc_ratio, short ml_alloc_control, short ml_discrep, bool piecewise_basis, bool use_derivs, bool cv_flag) [protected]

base constructor for lightweight construction of multilevel PCE using regression

This constructor is called by derived class constructors for lightweight instantiations that employ regression (least squares, CS, OLI).

References NonDExpansion::collocPtsSeqSpec, NonDExpansion::multilevAllocControl, and NonDExpansion::multilevDiscrepEmulation.

14.155.3 Member Function Documentation

void increment_order_from_grid() [protected]

define an expansion order that is consistent with an advancement in structured/unstructured grid level/density

Used for uniform refinement of regression-based PCE.

References NonDExpansion::collocRatio, SharedApproxData::data_rep(), SharedPecosApproxData::expansion_order(), NonDExpansion::numSamplesOnModel, NonDPolynomialChaos::ratio_samples_to_order(), Model::shared_approximation(), and NonDExpansion::uSpaceModel.

Referenced by NonDPolynomialChaos::append_expansion().

The documentation for this class was generated from the following files:

- NonDPolynomialChaos.hpp
- NonDPolynomialChaos.cpp

14.156 NonDQuadrature Class Reference

Derived nondeterministic class that generates N-dimensional numerical quadrature points for evaluation of expectation integrals over uncorrelated standard normals/uniforms/exponentials/betas/gammas.

Inheritance diagram for NonDQuadrature:
Public Member Functions

- **NonDQuadrature (Model &model, unsigned short quad_order, const RealVector &dim_pref, short driver_mode)**
  
  *alternate constructor for instantiations "on the fly" based on a quadrature order specification*

- **NonDQuadrature (Model &model, unsigned short quad_order, const RealVector &dim_pref, short driver_mode, int num_filt_samples)**

  *alternate constructor for instantiations "on the fly" that filter a tensor product sample set to include points with highest sample weights*

- **NonDQuadrature (Model &model, unsigned short quad_order, const RealVector &dim_pref, short driver_mode, int num_sub_samples, int seed)**

  *alternate constructor for instantiations "on the fly" that sub-sample quadrature rules by sampling randomly from a tensor product multi-index*

- **void increment_grid ()**

  *increment SSG level/TPQ order*

- **void decrement_grid ()**

  *decrement SSG level/TPQ order*

- **void evaluate_grid_increment ()**

  *computes a grid increment and evaluates the new parameter sets*

- **void update ()**

  *propagate any numSamples updates and/or grid updates/increments*

- **void reset ()**

  *set dimQuadOrderRef to dimension orders indicated by quadOrderSpec and dimPrefSpec, following refinement or sequence advancement*

- **const Pecos::UShortArray & quadrature_order () const**

  *return Pecos::TensorProductDriver::quadOrder*

- **void quadrature_order (const Pecos::UShortArray &dim_quad_order)**

  *set dimQuadOrderRef and map to Pecos::TensorProductDriver::quadOrder*

- **void quadrature_order (unsigned short quad_order)**

  *set quadOrderSpec and map to Pecos::TensorProductDriver::quadOrder*

- **void samples (size_t samples)**

  *set numSamples*

- **short mode () const**

  *return quadMode*
CHAPTER 14. CLASS DOCUMENTATION

Protected Member Functions

- NonDQuadrature (ProblemDescDB &problem_db, Model &model)
  
  * NonDQuadrature ()
    constructor
  
  * NonDQuadrature ()
    destructor
  
  * void initialize_grid (const std::vector<Pecos::BasisPolynomial> &poly_basis)
    
  * void get_parameter_sets (Model &model)
    Generate one block of numSamples samples (ndim * num_samples), populating allSamples; ParamStudy is the only class that specializes to use allVariables.
  
  * void sampling_reset (int min_samples, bool all_data_flag, bool stats_flag)
  
  * void increment_grid_preference (const RealVector &dim_pref)
    increment SSG level/TPQ order and update anisotropy
  
  * void increment_grid_preference ()
    increment SSG level/TPQ order and preserve anisotropy
  
  * void decrement_grid (const std::vector<Pecos::BasisPolynomial> &poly_basis)
  
  * int num_samples () const
  
  * void random_seed (int seed)
    set randomSeed, if present

Private Member Functions

- void increment_grid (UShortArray &dim_quad_order)
  convenience function used to make increment_grid() more modular
  
- void increment_grid_preference (const RealVector &dim_pref, UShortArray &dim_quad_order)
    
- void decrement_grid (UShortArray &dim_quad_order)
    
- void compute_minimum_quadrature_order (size_t min_samples, const RealVector &dim_pref, UShortArray &dim_quad_order)
    calculate smallest dim_quad_order with at least min_samples
  
- void filter_parameter_sets ()
    prune allSamples back to size numSamples, retaining points with highest product weight
  
- void update_anisotropic_order (const RealVector &dim_pref, UShortArray &quad_order_ref)
    update quad_order_ref based on an updated dimension preference, enforcing previous values as a lower bound
  
- void initialize_dimension_quadrature_order (unsigned short quad_order_spec, const RealVector &dim_pref_spec, UShortArray &dim_quad_order)
    initialize dim_quad_order from quad_order_spec and dim_pref_spec
  
- void increment_dimension_quadrature_order (UShortArray &dim_quad_order)
    increment each dim_quad_order entry by 1
  
- void increment_dimension_quadrature_order (const RealVector &dim_pref, UShortArray &dim_quad_order)
    increment the dim_quad_order entry with maximum preference by 1 and then rebalance
  
- void decrement_dimension_quadrature_order (UShortArray &dim_quad_order)
    decrement each dim_quad_order entry by 1
14.156. NONDQUADRATURE CLASS REFERENCE

**Private Attributes**
- Pecos::TensorProductDriver * tpqDriver
  - convenience pointer to the numIntDriver representation
- bool nestedRules
  - for studies involving refinement strategies, allow for use of nested quadrature rules such as Gauss-Patterson
- unsigned short quadOrderSpec
  - scalar quadrature order, rendered anisotropic via dimPrefSpec
- UShortArray dimQuadOrderRef
  - reference point for Pecos::TensorProductDriver::quadOrder: the original user specification for the number of Gauss points per dimension, plus any refinements posted by increment_grid()
- UShortArray dimQuadOrderPrev
  - value of dimQuadOrderRef prior to increment_grid(), for restoration in decrement_grid() since increment must induce a change in grid size and this adaptive increment in not reversible
- short quadMode
  - point generation mode: FULL_TENSOR, FILTERED_TENSOR, RANDOM_TENSOR
- size_t numSamples
  - size of a subset of tensor quadrature points (filtered based on product weight or sampled uniformly from the tensor multi-index); used by the regression PCE approach known as “probabilistic collocation”
- int randomSeed
  - seed for the random number generator used in sampling of the tensor multi-index

**Additional Inherited Members**

14.156.1 Detailed Description

Derived nondeterministic class that generates N-dimensional numerical quadrature points for evaluation of expectation integrals over uncorrelated standard normals/uniforms/exponentials/betas/gammas.

This class is used by NonDPolynomialChaos, but could also be used for general numerical integration of moments. It employs Gauss-Hermite, Gauss-Legendre, Gauss-Laguerre, Gauss-Jacobi and generalized Gauss-Laguerre quadrature for use with normal, uniform, exponential, beta, and gamma density functions and integration bounds. The abscissas and weights for one-dimensional integration are extracted from the appropriate Orthogonal-Polynomial class and are extended to n-dimensions using a tensor product approach.

14.156.2 Constructor & Destructor Documentation

NonDQuadrature ( Model & model, unsigned short quad_order, const RealVector & dim_pref, short driver_mode )

alternate constructor for instantiations "on the fly" based on a quadrature order specification

This alternate constructor is used for on-the-fly generation and evaluation of numerical quadrature points.

References NonDIntegration::numIntDriver, and NonDQuadrature::tpqDriver.

NonDQuadrature ( Model & model, unsigned short quad_order, const RealVector & dim_pref, short driver_mode, int num_filt_samples )

alternate constructor for instantiations "on the fly" that filter a tensor product sample set to include points with highest sample weights

This alternate constructor is used for on-the-fly generation and evaluation of filtered tensor quadrature points.

References NonDIntegration::numIntDriver, and NonDQuadrature::tpqDriver.
NonDQuadrature ( Model & model, unsigned short quad_order, const RealVector & dim_pref, short driver_mode, int num_sub_samples, int seed )

alternate constructor for instantiations "on the fly" that sub-sample quadrature rules by sampling randomly from a tensor product multi-index

This alternate constructor is used for on-the-fly generation and evaluation of random sampling from a tensor quadrature multi-index.

References NonDIntegration::numIntDriver, and NonDQuadrature::tpqDriver.

NonDQuadrature ( ProblemDescDB & problem_db, Model & model ) [protected]

caracterized for a standard letter-envelope iterator instantiation. In this case, set_db_list_nodes has been called and probDescDB can be queried for settings from the method specification. It is not currently used, as there is not yet a separate non_d_quadrature method specification.

References Iterator::convergenceTol, Model::correction_type(), ProblemDescDB::get_bool(), ProblemDescDB::get_int(), ProblemDescDB::get_short(), Iterator::iteratedModel, Iterator::maxEvalConcurrency, Model::multivariate_distribution(), NonDQuadrature::nestedRules, NonDIntegration::numIntDriver, Iterator::outputLevel, Iterator::probDescDB, NonDQuadrature::reset(), and NonDQuadrature::tpqDriver.

14.156.3 Member Function Documentation

void initialize_grid ( const std::vector< Pecos::BasisPolynomial > & polybasis ) [protected], [virtual]

Used in combination with alternate NonDQuadrature constructor.

Implements NonDIntegration.

References Iterator::iteratedModel, Iterator::maxEvalConcurrency, Model::multivariate_distribution(), NonDQuadrature::nestedRules, Analyzer::numContinuousVars, NonDQuadrature::numSamples, NonDQuadrature::quadMode, NonDQuadrature::reset(), NonDQuadrature::tpqDriver, and NonDQuadrature::update().

void sampling_reset ( int min_samples, bool all_data_flag, bool stats_flag ) [protected], [virtual]

used by DataFitSurrModel::build_global() to publish the minimum number of points needed from the quadrature routine in order to build a particular global approximation.

Reimplemented from Iterator.

References NonDIntegration::dimPrefSpec, NonDQuadrature::increment_grid(), NonDQuadrature::increment_grid_preference(), and NonDQuadrature::tpqDriver.

Referenced by NonDQuadrature::update().

int num_samples ( ) const [inline], [protected], [virtual]

Return current number of evaluation points. Since the calculation of samples, collocation points, etc. might be costly, provide a default implementation here that backs out from the maxEvalConcurrency.

Reimplemented from Analyzer.

References NonDQuadrature::numSamples, NonDQuadrature::quadMode, and NonDQuadrature::tpqDriver.

The documentation for this class was generated from the following files:

- NonDQuadrature.hpp
- NonDQuadrature.cpp
14.157 NonDQUESOBayesCalibration Class Reference

Bayesian inference using the QUESO library from UT Austin.

Inheritance diagram for NonDQUESOBayesCalibration:

```
NonDQUESOBayesCalibration
  NonDBayesCalibration
    NonDCalibration
      NonD
        Analyzer
          Iterator
```

Public Member Functions

- **NonDQUESOBayesCalibration** (ProblemDescDB &problem_db, Model &model)
  
  *standard constructor*
- **~NonDQUESOBayesCalibration**()
  
  *destructor*

Protected Member Functions

- void **calibrate**()
- void **print_results**(std::ostream &s, short results_state=FINAL_RESULTS)
  
  *print the final iterator results*
- void **print_variables**(std::ostream &s, const RealVector &c_vars)
  
  *convenience function to print calibration parameters, e.g., for MAP / best parameters*
- void **init_queso_environment**()
  
  *initialize the QUESO FullEnvironment on the Dakota MPIComm*
- void **init_precond_request_value**()
  
  *initialize the ASV value for preconditioned cases*
- virtual void **init_queso_solver**()
  
  *define solver options, likelihood callback, posterior RV, and inverse problem*
- void **precondition_proposal**(unsigned int chain_index)
  
  *use derivative information from the emulator to define the proposal covariance (inverse of misfit Hessian)*
- void **run_queso_solver**()
perform the MCMC process

- **void map_pre_solve ()**
- **void run_chain ()**
  
  short term option to restart the MCMC chain with updated proposal density computed from the emulator at a new starting point
- **void cache_chain ()**
  
  cache the chain to acceptanceChain and acceptedFnVals
- **void log_best ()**
  
  log at most batchSize best chain points into bestSamples
- **void filter_chain_by_conditioning ()**
  
  extract batchSize points from the MCMC chain and store final aggregated set within allSamples; unique points with best conditioning are selected, as determined by pivoted LU
- **void best_to_all ()**
  
  copy bestSamples to allSamples to use in surrogate update
- **void update_model ()**
  
  evaluates allSamples on iteratedModel and update the mcmcModel emulator with all\{Samples,Responses\}
- **Real assess_emulator_convergence ()**
  
  compute the L2 norm of the change in emulator coefficients
- **void init_parameter_domain ()**
  
  initialize the QUESO parameter space, min, max, initial, and domain
- **void init_proposal_covariance ()**
- **void prior_proposal_covariance ()**
  
  use covariance of prior distribution for setting proposal covariance
- **void user_proposal_covariance (const String &input_fmt, const RealVector &cov_data, const String &cov_filename)**
  
  set proposal covariance from user-provided diagonal or matrix
- **void validate_proposal ()**
- **void set_ip_options ()**
  
  set inverse problem options callIpOptionsValues common to all solvers
- **void set_mh_options ()**
  
  set MH-specific inverse problem options callIpMhOptionsValues
- **void update_chain_size (unsigned int size)**
  
  update MH-specific inverse problem options callIpMhOptionsValues
- **void copy_gsl (const QUESO::GslVector &qv, RealVector &rv)**
  
  local copy data utility from GslVector to RealVector
- **void copy_gsl (const RealVector &rv, QUESO::GslVector &qv)**
  
  local copy data utility from RealVector to GslVector
- **void copy_gsl_partial (const QUESO::GslVector &qv, size_t start, RealVector &rv)**
  
  local copy_data utility from portion of GslVector to RealVector
- **void copy_gsl_partial (const RealVector &rv, QUESO::GslVector &qv, size_t start)**
  
  local copy_data utility from RealVector to portion of GslVector
- **void copy_gsl (const QUESO::GslVector &qv, RealMatrix &rm, int i)**
  
  local copy_data utility from GslVector to column in RealMatrix
- **bool equal_gsl (const QUESO::GslVector &qv1, const QUESO::GslVector &qv2)**
  
  equality tester for two GslVectors
Static Protected Member Functions

- static double dakotaLogLikelihood (const QUESO::GslVector &paramValues, const QUESO::GslVector *paramDirection, const void *functionDataPtr, QUESO::GslVector *gradVector, QUESO::GslMatrix *hessianMatrix, QUESO::GslVector *hessianEffect)

  Log Likelihood function for call-back from QUESO to DAKOTA for evaluation.

Protected Attributes

- String mcmcType
  
  MCMC type ("dram" or "delayed_rejection" or "adaptive_metropolis" or "metropolis_hastings" or "multilevel", within QUESO)

- int propCovUpdatePeriod

  period (number of accepted chain samples) for proposal covariance update

- unsigned int batchSize

  number of points to add to surrogate at each iteration

- short precondRequestValue

  the active set request value to use in proposal preconditioning

- bool logitTransform

  flag indicating user activation of logit transform option

- boost::shared_ptr < QUESO::EnvOptionsValues > envOptionsValues

  options for setting up the QUESO Environment

- boost::shared_ptr < QUESO::FullEnvironment > quesoEnv

  top-level QUESO Environment

- boost::shared_ptr < QUESO::VectorSpace>
  < QUESO::GslVector, QUESO::GslMatrix > > paramSpace

  QUESO parameter space based on number of calibrated parameters.

- boost::shared_ptr < QUESO::BoxSubset>
  < QUESO::GslVector, QUESO::GslMatrix > > paramDomain

  QUESO parameter domain: hypercube based on min/max values.

- boost::shared_ptr < QUESO::GslVector > paramInitials

  initial parameter values at which to start chain

- boost::shared_ptr < QUESO::BaseVectorRV>
  < QUESO::GslVector, QUESO::GslMatrix > > priorRv

  random variable for the prior

- boost::shared_ptr < QUESO::GslMatrix > proposalCovMatrix
CHAPTER 14. CLASS DOCUMENTATION

proposal covariance for DRAM
• double priorPropCovMult
  optional multiplier to scale prior-based proposal covariance
• boost::shared_ptr
  < QUESO::SipOptionsValues > callIpOptionsValues
  general inverse problem options
• boost::shared_ptr
  < QUESO::MhOptionsValues > callIpMhOptionsValues
  MH-specific inverse problem options.
• boost::shared_ptr
  < QUESO::GenericScalarFunction
  < QUESO::GslVector,
  QUESO::GslMatrix > > likelihoodFunctionObj
• boost::shared_ptr
  < QUESO::GenericVectorRV
  < QUESO::GslVector,
  QUESO::GslMatrix > > postRv
  random variable for the posterior
• boost::shared_ptr
  < QUESO::StatisticalInverseProblem
  < QUESO::GslVector,
  QUESO::GslMatrix > > inverseProb
  QUESO inverse problem solver.
• String advancedOptionsFile
  advanced options file name (GPMSA only); settings from this file override any C++ / Dakota input file settings

Static Protected Attributes
• static NonDQUESOBayesCalibration * nonDQUESOInstance
  Pointer to current class instance for use in static callback functions.

Private Attributes
• RealVectorArray prevCoeffs
  cache previous expansion coefficients for assessing convergence of emulator refinement process

Friends
• class DerivInformedPropCovTK< QUESO::GslVector, QUESO::GslMatrix >
  Random walk transition kernel needs callback access to QUESO details.
• class DerivInformedPropCovLogitTK< QUESO::GslVector, QUESO::GslMatrix >
  Logit random walk transition kernel needs callback access to QUESO details.
14.157. NONDQUESTOBAYESCALIBRATION CLASS REFERENCE

Additional Inherited Members

14.157.1 Detailed Description

Bayesian inference using the QUESO library from UT Austin.

This class wraps the Quantification of Uncertainty for Estimation, Simulation, and Optimization (QUESO) library, developed as part of the Predictive Science Academic Alliance Program (PSAAP)-funded Predictive Engineering and Computational Sciences (PECOS) Center at UT Austin.

14.157.2 Constructor & Destructor Documentation

NonDQUESTOBayesCalibration (ProblemDescDB & problem_db, Model & model)

standard constructor

This constructor is called for a standard letter-envelope iterator instantiation. In this case, set_db_list_nodes has been called and probDescDB can be queried for settings from the method specification.

References Dakota::abort_handler(), NonDBayesCalibration::adaptPosteriorRefine, NonDQUESTOBayesCalibration::advancedOptionsFile, NonDQUESTOBayesCalibration::batchSize, NonD Calibration::calibrationData, NonDBayesCalibration::chainSamples, NonDBayesCalibration::emulatorType, NonD QUESTOBayesCalibration::init_queso_environment(), Iterator::maxIterations, NonDBayesCalibration::obsErrorMultiplierMode, Iterator::outputLevel, NonDQUESTOBayesCalibration::priorPropCovMult, NonDQUESTOBayesCalibration::propCovUpdatePeriod, and NonDBayesCalibration::proposalCovarType.

14.157.3 Member Function Documentation

void calibrate ( ) [protected], [virtual]

Perform the uncertainty quantification

Implements NonDBayesCalibration.

References Dakota::abort_handler(), NonDBayesCalibration::adaptPosteriorRefine, NonDQUESTOBayesCalibration::assess_emulator_convergence(), Analyzer::compactMode, NonDBayesCalibration::compute_statistics(), Iterator::convergenceTol, NonDBayesCalibration::emulatorType, NonDQUESTOBayesCalibration::init_parameter_domain(), NonDQUESTOBayesCalibration::init_precond_request_value(), NonDQUESTOBayesCalibration::init_queso_solver(), NonDBayesCalibration::initialize_model(), Iterator::maxIterations, NonDQUESTOBayesCalibration::nonDQUESTOInstance, NonDBayesCalibration::proposalCovarType, NonDQUESTOBayesCalibration::run_chain(), TKFactory-DIPC::set_callback(), TKFactoryDIPCLogit::set_callback(), Dakota::tk_factory_dipc, Dakota::tk_factory_dipclogit, and NonDQUESTOBayesCalibration::update_model().

void print_results ( std::ostream & s, short results_state = FINAL_RESULTS ) [protected], [virtual]

print the final iterator results

This virtual function provides additional iterator-specific final results outputs beyond the function evaluation summary printed in finalize_run().

Reimplemented from NonDBayesCalibration.

References NonDBayesCalibration::bestSamples, NonDQUESTOBayesCalibration::copy_gsl(), NonDQUESTOBayesCalibration::copy_gsl_partial(), NonD Calibration::expData, Dakota::HALF_LOG_2PI, ExperimentData::half_log_cov_determinant(), NonDBayesCalibration::log_prior_density(), Model::num_primary_fns(), Analyzer::numContinuousVars, NonDBayesCalibration::obsErrorMultiplierMode, NonDQUESTOBayesCalibration::paramSpace, NonDBayesCalibration::print_results(), NonDQUESTOBayesCalibration::print_variables(), NonDBayesCalibration::residualModel, and Dakota::write_precision.
void cache_chain ( ) [protected]
cache the chain to acceptanceChain and acceptedFnVals
Populate all of acceptanceChain(numparams, chainSamples) acceptedFnVals(numFunctions, chainSamples)
References Dakota::abort_handler(), NonDBayesCalibration::acceptanceChain, NonDBayesCalibration::accepted-
FnVals, Response::active_set(), Model::active_variables(), NonDBayesCalibration::chainSamples, Variables::continuous-
_variables(), Response::copy(), Variables::copy(), NonDQUESTOBayesCalibration::copy_gsl_partial(), Model::current-
_response(), Model::current_variables(), Dakota::data_pairs, Model::evaluate(), Response::function_values(), Model-
::interface_id(), NonDQUESTOBayesCalibration::inverseProb, Dakota::lookup_by_val(), NonDBayesCalibration-
::mcmcModel, NonDBayesCalibration::mcmcModelHasSurrogate, Model::model_type(), Analyzer::numContinuous-
Vars, Analyzer::numFunctions, NonDBayesCalibration::numHyperparams, Iterator::outputLevel, NonDQUESTO-
BayesCalibration::postRv, Model::probability_transformation(), ActiveSet::request_values(), NonDBayesCalibration-
::standardizedSpace, and ParamResponsePair::variables().
Referenced by NonDQUESTOBayesCalibration::run_chain().

void init_parameter_domain ( ) [protected]
initialize the QUESO parameter space, min, max, initial, and domain
Initialize the calibration parameter domain (paramSpace, paramMins/paramMaxs, paramDomain, param-
Initials, priorRV)
References NonDQUESTOBayesCalibration::copy_gsl(), NonDBayesCalibration::mapSoln, NonDBayesCalibration-
::mcmcModel, Model::multivariate_distribution(), NonDQUESTOBayesCalibration::nonDQUESTOInstance, Analyzer-
::numContinuousVars, NonDBayesCalibration::numHyperparams, Iterator::outputLevel, NonDQUESTOBayesCalibration-
::paramDomain, NonDQUESTOBayesCalibration::paramInitials, NonDQUESTOBayesCalibration::paramSpace, Non-
DQUESTOBayesCalibration::priorRv, and NonDQUESTOBayesCalibration::quesoEnv.
Referenced by NonDQUESTOBayesCalibration::calibrate(), and NonDGPSMSABayesCalibration::calibrate().

void prior_proposal_covariance ( ) [protected]
use covariance of prior distribution for setting proposal covariance
Must be called after paramMins/paramMaxs set above
References NonDBayesCalibration::mcmcModel, Model::multivariate_distribution(), Analyzer::numContinuous-
Vars, Iterator::outputLevel, NonDQUESTOBayesCalibration::priorPropCovMult, NonDQUESTOBayesCalibration-
::proposalCovMatrix, and NonDBayesCalibration::standardizedSpace.

void user_proposal_covariance ( const String & input_fmt, const RealVector & cov_data, const String & cov_filename ) [protected]
set proposal covariance from user-provided diagonal or matrix
This function will convert user-specified cov_type = "diagonal" | "matrix" data from either cov_data or cov_-
filename and populate a full QUESO::GslMatrix* in proposalCovMatrix with the covariance.
References Dakota::length(), Analyzer::numContinuousVars, NonDQUESTOBayesCalibration::proposalCov-
Matrix, Dakota::read_unsigned_data(), and NonDBayesCalibration::standardizedSpace.

void set_ip_options ( ) [protected]
set inverse problem options callIpOptionsValues common to all solvers
set inverse problem options common to all solvers
References NonDQUESTOBayesCalibration::advancedOptionsFile, NonDQUESTOBayesCalibration::callIpOptions-
Values, Iterator::outputLevel, and NonDQUESTOBayesCalibration::quesoEnv.
14.158. NONDRELIABILITY CLASS REFERENCE

Referenced by NonDGPMSSBayesCalibration::init_queso_solver(), and NonDQUESOBayesCalibration::init_queso_solver().

14.157.4 Member Data Documentation

bool logitTransform [protected]
flag indicating user activation of logit transform option
this option is useful for preventing rejection or resampling for out-of-bounds samples by transforming bounded domains to [-inf,inf].
Referenced by NonDQUESOBayesCalibration::set_mh_options().
The documentation for this class was generated from the following files:
- NonDQUESOBayesCalibration.hpp
- NonDQUESOBayesCalibration.cpp

14.158 NonDReliability Class Reference

Base class for the reliability methods within DAKOTA/UQ.
Inheritance diagram for NonDReliability:

```
<table>
<thead>
<tr>
<th>Iterator</th>
</tr>
</thead>
<tbody>
<tr>
<td>Analyzer</td>
</tr>
<tr>
<td>NonD</td>
</tr>
<tr>
<td>NonDReliability</td>
</tr>
<tr>
<td>NonDGlobalReliability</td>
</tr>
<tr>
<td>NonDLocalReliability</td>
</tr>
</tbody>
</table>
```

Protected Member Functions

- NonDReliability (ProblemDescDB &problem_db, Model &model)
constructor
- ~NonDReliability ()
destructor
- void nested_variable_mappings (const SizetArray &c_index1, const SizetArray &di_index1, const SizetArray &ds_index1, const SizetArray &dr_index1, const ShortArray &c_target2, const ShortArray &di_target2, const ShortArray &ds_target2, const ShortArray &dr_target2)
  set primaryA{CV,DIV,DRV}MapIndices, secondaryA{CV,DIV,DRV}MapTargets within derived Iterators; supports computation of higher-level sensitivities in nested contexts (e.g., derivatives of statistics w.r.t. inserted design variables)
- bool resize ()
  reinitializes iterator based on new variable size
• void post_run (std::ostream &s)
  post-run portion of run (optional); verbose to print results; re-implemented by Iterators that can read all Variables/-
  Responses and perform final analysis phase in a standalone way
• void initialize_graphics (int iterator_server_id=1)
  initialize graphics customized for reliability methods
• const Model & algorithm_space_model () const

Protected Attributes

• Model uSpaceModel
  Model representing the limit state in u-space, after any recastings and data fits.
• Model mppModel
  RecastModel which formulates the optimization subproblem: RIA, PMA, EGO.
• Iterator mppOptimizer
  Iterator which optimizes the mppModel.
• unsigned short mppSearchType
  the MPP search type selection: MV, x/u-space AMV, x/u-space AMV+, x/u-space TANA, x/u-space EGO, or NO_A-
  APPROX
• Iterator importanceSampler
  importance sampling instance used to compute/refine probabilities
• unsigned short integrationRefinement
  integration refinement type (NO_INT_REFINE, IS, AIS, or MMAIS) provided by refinement specification
• size_t numRelAnalyses
  number of invocations of core run()
• size_t approxIters
  number of approximation cycles for the current respFnCount/levelCount
• bool approxConverged
  indicates convergence of approximation-based iterations
• int respFnCount
  counter for which response function is being analyzed
• size_t levelCount
  counter for which response/probability level is being analyzed
• size_t statCount
  counter for which final statistic is being computed
• bool pmaMaximizeG
  flag indicating maximization of G(u) within PMA formulation
• Real requestedTargetLevel
  the \{response, reliability, generalized reliability\} level target for the current response function

Additional Inherited Members

14.158.1 Detailed Description

Base class for the reliability methods within DAKOTA/UQ.

The NonDReliability class provides a base class for NonDLocalReliability, which implements traditional
MPP-based reliability methods, and NonDGlobalReliability, which implements global limit state search using
Gaussian process models in combination with multimodal importance sampling.
14.158.2 Member Function Documentation

```cpp
void post_run ( std::ostream & s ) [protected], [virtual]
```

Post-run portion of run (optional); verbose to print results; re-implemented by Iterators that can read all Variables/Responses and perform final analysis phase in a standalone way.

Post-run phase, which a derived iterator may optionally reimplement; when not present, post-run is likely integrated into run. This is a virtual function; when re-implementing, a derived class must call its nearest parent’s `post_run()`, typically after performing its own implementation steps.

Reimplemented from Analyzer.

References Model::finalize_mapping(), Model::is_null(), NonDReliability::mppModel, NonDReliability::numRelAnalyses, and Analyzer::post_run().

```cpp
const Model & algorithm_space_model ( ) const [inline], [protected], [virtual]
```

default definition that gets redefined in selected derived Minimizers

Reimplemented from Analyzer.

References NonDReliability::uSpaceModel.

The documentation for this class was generated from the following files:

- NonDReliability.hpp
- NonDReliability.cpp

14.159 NonDRKDDarts Class Reference

Base class for the Recursive k-d Dart methods within DAKOTA/UQ.

Inheritance diagram for NonDRKDDarts:

```
  NonDRKDDarts
     |    
     |    
  NonD  
     |    
  Analyzer  
     |    
  Iterator
```

Public Member Functions

- **NonDRKDDarts** *(ProblemDescDB &problem_db, Model &model)*  
  *constructor*

- **~NonDRKDDarts** ()  
  *destructor*

- **bool resize ()**  
  *reinitializes iterator based on new variable size*

- **void core_run ()**
Protected Member Functions

- void pre_run()
  
- void initiate_random_number_generator(unsigned long x)
- double generate_a_random_number()
- void init_rkd_darts()
- void create_rkd_containers(size_t expected_num_samples)
- void execute()
- void create_initial_children(size_t parent)
- void create_new_sample(size_t parent, size_t left, size_t right, double position)
- void improve_parent_evaluation(size_t parent)
- void evaluate_1d_surrogate(size_t parent)
- double get_surrogate_interp_error(size_t parent)
- double estimate_surrogate_evaluation_err(size_t parent)
- void get_children(size_t parent, size_t *children)
- void get_neighbors(size_t sample, size_t &num_neighbors, size_t *neighbors)
- double interpolate_lagrange(size_t num_data_points, double *data_x, double *data_f, double x)
- double integrate_legendre_gauss(double xmin, double xmax, size_t num_data_points, double *data_x, double *data_f, double &err_est)
- double f_true(double *x)
- void initialize_surrogates()
- void compute_response(double *x)
- void add_surrogate_data(const Variables &vars, const Response &resp)
- void build_surrogate()
- double eval_surrogate(size_t fn_index, double *vin)
- void estimate_rkd_surrogate()
- void post_run(std::ostream &s)

- void print_integration_results(std::ostream &s)
- void exit_rkd_darts()
- void destroy_rkd_containers()

Protected Attributes

- int samples
- int seed
- int emulatorSamples
  
- double Q[1220]
- int index
- double cc
- double c
- double zn
- double zx
- double zy
- size_t qlen
14.159. NONDRKDDARTS CLASS REFERENCE

Private Attributes

- double * _I_RKD
- bool _eval_error
- size_t _test_function
- size_t _num_inserted_points
- size_t _num_dim
- size_t _num_samples
- size_t _max_num_samples
- size_t _num_evaluations
- size_t _evaluation_budget
- size_t _max_num_neighbors
- double _bounding_box_volume
- double _discont_jump_threshold
- double * _xmin
- double * _xmax
- double ** _fval
- size_t * _sample_dim
- size_t * _sample_parent
- size_t * _sample_first_child
- size_t * _sample_num_children
- size_t * _sample_left
- size_t * _sample_right
- double * _sample_coord
- double * _sample_value
- double * _sample_left_interp_err
- double * _sample_right_interp_err
- double * _sample_left_ev_err
- double * _sample_right_ev_err

Additional Inherited Members

14.159.1 Detailed Description

Base class for the Recursive k-d Dart methods within DAKOTA/UQ.

The NonDRKDDart class recursively implements the numerical integration of a domain based on k-d flat samples.

14.159.2 Member Function Documentation

void core_run ( ) [virtual]

Loop over the set of samples and compute responses.
Reimplemented from Iterator.

void pre_run ( ) [protected],[virtual]

generate samples
Generate Parameter Sets.
Reimplemented from Analyzer.
References Analyzer::pre_run().
void post_run ( std::ostream & s ) [protected], [virtual]

generate statistics
Print function evaluation summary, and integration results.
Reimplemented from Analyzer.
References Iterator::iteratedModel, Analyzer::post_run(), and Model::print_evaluation_summary().
The documentation for this class was generated from the following files:

- NonDRKDDarts.hpp
- NonDRKDDarts.cpp

14.160 NonDSampling Class Reference

Base class for common code between NonDLHSSampling, NonDAdaptImpSampling, and other specializations.
Inheritance diagram for NonDSampling:

```
NonDSampling
    NonDAnalyzer
    Iterator
    NonDAdaptImpSampling
    NonDAdaptiveSampling
    NonDGPImpSampling
    NonDLHSSampling
    NonDMultilevelSampling
```

Public Member Functions

- **NonDSampling** (Model &model, const RealMatrix &sample_matrix)
  alternate constructor for evaluating and computing statistics for the provided set of samples
- void **compute_statistics** (const RealMatrix &vars_samples, const IntResponseMap &resp_samples)
  For the input sample set, computes mean, standard deviation, and probability/reliability/response levels (aleatory uncertainties) or intervals (epistemic or mixed uncertainties)
- void **compute_intervals** (RealRealPairArray &extreme_fns)
  called by compute_statistics() to calculate min/max intervals using allResponses
- void **compute_intervals** (const IntResponseMap &samples)
  called by compute_statistics() to calculate extremeValues from samples
- void **compute_intervals** (RealRealPairArray &extreme_fns, const IntResponseMap &samples)
  called by compute_statistics() to calculate min/max intervals using samples
- void **compute_moments** (const RealVectorArray &fn_samples)
  calculates sample moments from a matrix of observations for a set of QoI
- void **compute_moments** (const IntResponseMap &samples)
  calculate sample moments and confidence intervals from a map of response observations
- void **compute_moments** (const IntResponseMap &samples, RealMatrix &moment_stats, RealMatrix &moment_grads, RealMatrix &moment_conf_ints, short moments_type, const StringArray &labels)
  convert IntResponseMap to RealVectorArray and invoke helpers
• void `compute_moment_gradients` (const RealVectorArray &fn_samples, const RealMatrixArray &grad_samples, const RealMatrix &moment_stats, RealMatrix &moment_grads, short moments_type)
  
  `compute moment grads from function and gradient samples`

• void `compute_moment_confidence_intervals` (const RealMatrix &moment_stats, RealMatrix &moment_conf ints, const SizetArray &sample_counts, short moments_type)
  
  `compute moment confidence intervals from moment values`

• void `archive_moments` (size_t inc_id=0)
  
  `archive moment statistics in results DB`

• void `archive_moment_confidence_intervals` (size_t inc_id=0)
  
  `archive moment confidence intervals in results DB`

• void `archive_extreme_responses` (size_t inc_id=0)
  
  `archive extreme values (epistemic result) in results DB`

• void `compute_level_mappings` (const IntResponseMap &samples)
  
  `called by compute_statistics() to calculate CDF/CCDF mappings of z to p/beta and of p/beta to z as well as PDFs`

• void `print_statistics` (std::ostream &s) const
  
  `prints the statistics computed in compute_statistics()`

• void `print_intervals` (std::ostream &s) const
  
  `prints the intervals computed in compute_intervals() with default qoi_type and moment_labels`

• void `print_intervals` (std::ostream &s, String qoi_type, const StringArray &interval_labels) const
  
  `prints the intervals computed in compute_intervals()`

• void `print_moments` (std::ostream &s) const
  
  `prints the moments computed in compute_moments()`

• void `print_moments` (std::ostream &s, String qoi_type, const StringArray &moment_labels) const
  
  `prints the moments computed in compute_moments()`

• void `print_wilks_statistics` (std::ostream &s) const
  
  `prints the Wilks statistics`

• void `update_final_statistics` ()
  
  `update finalStatistics from minValues/maxValues, momentStats, and computedProbLevels/computedRelLevels/computedRespLevels`

• void `transform_samples` (Pecos::ProbabilityTransformation &nataf, bool x_to_u=true)
  
  `transform allSamples imported by alternate constructor. This is needed since random variable distribution parameters are not updated until run time and an imported sample matrix is typically in x-space.`

• void `transform_samples` (Pecos::ProbabilityTransformation &nataf, RealMatrix &sample_matrix, int num_samples=0, bool x_to_u=true)
  
  `transform the specified samples matrix from x to u or u to x`

• unsigned short `sampling_scheme` () const
  
  `return sampleType`

• const String & `random_number_generator` () const
  
  `return rngName`
CHAPTER 14. CLASS DOCUMENTATION

Static Public Member Functions

- static void compute_moments (const RealVectorArray &fn_samples, SizetArray &sample_counts, RealMatrix &moment_stats, short moments_type, const StringArray &labels)
  
  core compute_moments() implementation with all data as inputs

- static void compute_moments (const RealVectorArray &fn_samples, RealMatrix &moment_stats, short moments_type)
  
  core compute_moments() implementation with all data as inputs

- static void compute_moments (const RealMatrix &fn_samples, RealMatrix &moment_stats, short moments_type)
  
  alternate RealMatrix samples API for use by external clients

- static void print_moments (std::ostream &s, const RealMatrix &moment_stats, const RealMatrix moment_cis, String qoi_type, short moments_type, const StringArray &moment_labels, bool print_cis)
  
  core print moments that can be called without object

- static int compute_wilks_sample_size (unsigned short order, Real alpha, Real beta, bool twosided=false)
  
  calculates the number of samples using the Wilks formula Static so I can test without instantiating a NonDSampling object - RWH

- static Real compute_wilks_residual (unsigned short order, int nsamples, Real alpha, Real beta, bool twosided)
  
  Helper function - calculates the Wilks residual.

- static Real compute_wilks_alpha (unsigned short order, int nsamples, Real beta, bool twosided=false)
  
  calculates the alpha parameter given number of samples using the Wilks formula Static so I can test without instantiating a NonDSampling object - RWH

- static Real compute_wilks_beta (unsigned short order, int nsamples, Real alpha, bool twosided=false)
  
  calculates the beta parameter given number of samples using the Wilks formula Static so I can test without instantiating a NonDSampling object - RWH

- static Real get_wilks_alpha_min ()
  
  Get the lower and upper bounds supported by Wilks bisection solves.

- static Real get_wilks_alpha_max ()

- static Real get_wilks_beta_min ()

- static Real get_wilks_beta_max ()

Protected Member Functions

- NonDSampling (ProblemDescDB &problem_db, Model &model)
  
  constructor

- NonDSampling (unsigned short method_name, Model &model, unsigned short sample_type, int samples, int seed, const String &rng, bool vary_pattern, short sampling_vars_mode)
  
  alternate constructor for sample generation and evaluation "on the fly"

- NonDSampling (unsigned short sample_type, int samples, int seed, const String &rng, const RealVector &lower_bnds, const RealVector &upper_bnds)
  
  alternate constructor for sample generation "on the fly"

- NonDSampling (unsigned short sample_type, int samples, int seed, const String &rng, const RealVector &means, const RealVector &std_devs, const RealVector &lower_bnds, const RealVector &upper_bnds, RealSymMatrix &correl)
  
  alternate constructor for sample generation of correlated normals "on the fly"

- ~NonDSampling ()
14.160. NONDSAMPLING CLASS REFERENCE

• destructor

• void pre_run ()
  
  pre-run portion of run (optional); re-implemented by Iterators which can generate all Variables (parameter sets) a priori

• void core_run ()

• int num_samples () const

• void sampling_reset (int min_samples, bool all_data_flag, bool stats_flag)
  
  resets number of samples and sampling flags

• void sampling_reference (int samples_ref)
  
  set reference number of samples, which is a lower bound during reset

• void random_seed (int seed)
  
  assign randomSeed

• void vary_pattern (bool pattern_flag)
  
  set varyPattern

• void get_parameter_sets (Model &model)
  
  Uses lhsDriver to generate a set of samples from the distributions/bounds defined in the incoming model.

• void get_parameter_sets (Model &model, const int num_samples, RealMatrix &design_matrix)
  
  Uses lhsDriver to generate a set of samples from the distributions/bounds defined in the incoming model and populates the specified design matrix.

• void get_parameter_sets (Model &model, const int num_samples, RealMatrix &design_matrix, bool write_msg)
  
  core of get_parameter_sets that accepts message print control

• void get_parameter_sets (const RealVector &lower_bnds, const RealVector &upper_bnds)
  
  Uses lhsDriver to generate a set of uniform samples over lower_bnds/upper_bnds.

• void get_parameter_sets (const RealVector &means, const RealVector &std_devs, const RealVector &lower_bnds, const RealVector &upper_bnds, RealSymMatrix &correl)
  
  Uses lhsDriver to generate a set of normal samples.

• void update_model_from_sample (Model &model, const Real *sample_vars)
  
  Override default update of continuous vars only.

• void sample_to_variables (const Real *sample_vars, Variables &vars)
  
  override default mapping of continuous variables only

• void variables_to_sample (const Variables &vars, Real *sample_vars)
  
  override default mapping of continuous variables only

• const RealVector & response_error_estimates () const
  
  return error estimates associated with each of the finalStatistics

• void initialize_lhs (bool write_message, int num_samples)
  
  increments numLHSRuns, sets random seed, and initializes lhsDriver

• void active_set_mapping ()
  
  in the case of sub-iteration, map from finalStatistics.active_set() requests to activeSet used in evaluate_parameter_sets()

• void mode_counts (const Variables &vars, size_t &cv_start, size_t &num_cv, size_t &div_start, size_t &num_div, size_t &dsv_start, size_t &num_dsv, size_t &drv_start, size_t &num_drv) const
  
  compute sampled subsets (all, active, uncertain) within all variables (acv/adiv/adrv) from samplingVarsMode and model

• void mode_bits (const Variables &vars, BitArray &active_vars, BitArray &active_corr) const
  
  define subset views for sampling modes
Static Protected Member Functions

- static void accumulate_mean (const RealVectorArray &fn_samples, size_t q, size_t &num_samp, Real &mean)
  helper to accumulate sum of finite samples
- static void accumulate_moments (const RealVectorArray &fn_samples, size_t q, short moments_type, Real *moments)
  helper to accumulate higher order sums of finite samples
- static void accumulate_moment_gradients (const RealVectorArray &fn_samples, const RealMatrixArray &grad_samples, size_t q, short moments_type, Real mean, Real mom2, Real *mean_grad, Real *mom2_grad)
  helper to accumulate gradient sums

Protected Attributes

- int seedSpec
  the user seed specification (default is 0)
- int randomSeed
  the current seed
- const int samplesSpec
  initial specification of number of samples
- int samplesRef
  reference number of samples updated for refinement
- int numSamples
  the current number of samples to evaluate
- String rngName
  name of the random number generator
- unsigned short sampleType
  the sample type: default, random, lhs, < incremental random, or incremental lhs
- bool wilksFlag
  flags use of Wilks formula to calculate num samples
- unsigned short wilksOrder
- Real wilksAlpha
- Real wilksBeta
- short wilksSidedness
- RealMatrix momentGrads
  gradients of standardized or central moments of response functions, as determined by finalMomentsType. Calculated in compute_moments() and indexed as (var,moment) when moment id runs from 1:2*numFunctions.
- RealVector finalStatErrors
  standard errors (estimator std deviation) for each of the finalStatistics
- int samplesIncrement
  current increment in a sequence of samples
- Pecos::LHSDriver lhsDriver
  the C++ wrapper for the F90 LHS library
- bool statsFlag
flags computation/output of statistics

- boolean allDataFlag
  flags update of allResponses
  \(<\) (allVariables or allSamples already defined)

- short samplingVarsMode
  the sampling mode: ALEATORY, UNCERTAIN, EPISTEMIC, UNCERTAIN, ACTIVE, or ALL.
  This is a secondary control on top of the variables view that allows sampling over subsets of variables that may differ from the view.

- short sampleRanksMode
  mode for input/output of LHS sample ranks: IGNORE_RANKS, GET_RANKS, SET_RANKS, or SET_GET_RANKS

- boolean varyPattern
  flag for generating a sequence of seed values within multiple \(get\_parameter\_sets()\) calls so that these executions (e.g., for SBO/SBNLS) are not repeated, but are still repeatable

- RealMatrix sampleRanks
  data structure to hold the sample ranks

- SensAnalysisGlobal nonDSampCorr
  initialize statistical post processing

- boolean backfillFlag
  flags whether to use backfill to enforce uniqueness of discrete LHS samples

- RealRealPairArray extremeValues
  Minimum and maximum values of response functions for epistemic calculations (calculated in \(compute\_intervals()\)).

- boolean functionMomentsComputed
  Function moments have been computed; used to determine whether to archive the moments.

### Private Member Functions

- void sample_to_variables (const Real *sample_vars, Variables &vars, Model &model)
  helper function to consolidate update code

- void sample_to_type (const Real *sample_vars, Variables &vars, size_t &cv_index, size_t num_cv, size_t &div_index, size_t num_div, size_t &dsv_index, size_t num_dsv, size_t &drv_index, size_t num_drv, size_t &samp_index, Model &model)
  helper function to copy a range from sample_vars to a variables type

- void sample_to_cv_type (const Real *sample_vars, Variables &vars, size_t &cv_index, size_t num_cv, size_t &div_index, size_t num_div, size_t &dsv_index, size_t num_dsv, size_t &drv_index, size_t num_drv, size_t &samp_index)
  helper function to copy a range from sample_vars to a variables type

- void sample_to_cv (const Real *sample_vars, Variables &vars, size_t &acv_index, size_t num_acv, size_t &samp_index)
  helper function to copy a range from sample_vars to continuous variables

- void sample_to_div (const Real *sample_vars, Variables &vars, size_t &adiv_index, size_t num_adiv, size_t &samp_index)
  helper function to copy a range from sample_vars to discrete int variables

- void sample_to_dsv (const Real *sample_vars, Variables &vars, size_t &adsv_index, size_t num_adsv, size_t &samp_index, const StringSetArray &dss_values)
  helper function to copy a range from sample_vars to discrete string vars
• void sample_to_drv (const Real *sample_vars, Variables &vars, size_t &adrv_index, size_t num_adrv, size_t &samp_index)
  
  helper function to copy a range from sample_vars to discrete real vars

Private Attributes

• size_t numLHSRuns
  
  counter for number of executions of get_parameter_sets() for this object

• RealMatrix momentCIs
  
  Matrix of confidence internals on moments, with rows for mean_lower, mean_upper, sd_lower, sd_upper (calculated in compute_moments())

Additional Inherited Members

14.160.1 Detailed Description

Base class for common code between NonDLHSSampling, NonDAdaptImpSampling, and other specializations.

This base class provides common code for sampling methods which employ the Latin Hypercube Sampling (LHS) package from Sandia Albuquerque’s Risk and Reliability organization. NonDSampling now exclusively utilizes the 1998 Fortran 90 LHS version as documented in SAND98-0210, which was converted to a UNIX link library in

1. The 1970’s vintage LHS (that had been f2c’d and converted to incomplete classes) has been removed.

14.160.2 Constructor & Destructor Documentation

NonDSampling ( Model & model, const RealMatrix & sample_matrix )

alternate constructor for evaluating and computing statistics for the provided set of samples

This alternate constructor defines allSamples from an incoming sample matrix.

References Analyzer::allSamples, Analyzer::compactMode, Iterator::maxEvalConcurrency, NonDSampling::numSamples, NonDSampling::samplesRef, NonDSampling::samplesSpec, and Iterator::subIteratorFlag.

NonDSampling ( ProblemDescDB & problem_db, Model & model ) [protected]

constructor

This constructor is called for a standard letter-envelope iterator instantiation. In this case, set_db_list_nodes has been called and probDescDB can be queried for settings from the method specification.

References Dakota::abort_handler(), NonDSampling::compute_wilks_sample_size(), NonD::epistemicStats, ProblemDescDB::get_real(), ProblemDescDB::get_short(), ProblemDescDB::get_ushort(), NonD::initialize_final_statistics(), Iterator::maxEvalConcurrency, Analyzer::numFunctions, NonDSampling::numSamples, Iterator::probDescDB, NonD::requestedProbLevels, NonDSampling::samplesRef, NonDSampling::sampleType, NonD::totalLevelRequests, and NonDSampling::wilksFlag.

NonDSampling ( unsigned short method_name, Model & model, unsigned short sample_type, int samples, int seed, const String & rng, bool vary_pattern, short sampling_vars_mode ) [protected]

alternate constructor for sample generation and evaluation “on the fly”

This alternate constructor is used for generation and evaluation of on-the-fly sample sets.
References SharedVariablesData::active_components_totals(), Model::current_variables(), NonD::epistemicStats, Iterator::iteratedModel, Iterator::maxEvalConcurrency, NonDSampling::numSamples, NonDSampling::sampleType, NonDSampling::samplingVarsMode, Variables::shared_data(), and Iterator::subIteratorFlag.

NonDSampling ( unsigned short sample_type, int samples, int seed, const String & rng, const RealVector & lower_bnds, const RealVector & upper_bnds ) [protected]

alternate constructor for sample generation "on the fly"

This alternate constructor is used by ConcurrentStrategy for generation of uniform, uncorrelated sample sets.

References Iterator::maxEvalConcurrency, NonDSampling::numSamples, NonDSampling::sampleType, and Iterator::subIteratorFlag.

NonDSampling ( unsigned short sample_type, int samples, int seed, const String & rng, const RealVector & means, const RealVector & std_devs, const RealVector & lower_bnds, const RealVector & upper_bnds, RealSymMatrix & correl ) [protected]

alternate constructor for sample generation of correlated normals "on the fly"

This alternate constructor is used by ConcurrentStrategy for generation of normal, correlated sample sets.

References Iterator::maxEvalConcurrency, NonDSampling::numSamples, NonDSampling::sampleType, and Iterator::subIteratorFlag.

14.160.3 Member Function Documentation

void compute_level_mappings ( const IntResponseMap & samples )
called by compute_statistics() to calculate CDF/CCDF mappings of z to p/beta and of p/beta to z as well as PDFs

Computes CDF/CCDF based on sample binning. A PDF is inferred from a CDF/CCDF within compute_densities() after level computation.

References Dakota::abort_handler(), Response::active_set_derivative_vector(), Response::active_set_request_vector(), NonD::archive_allocate_mappings(), NonD::cdfFlag, NonD::compute_densities(), NonD::computedGenRelLevels, NonD::computedProbLevels, NonD::computedRelLevels, NonD::computedRespLevels, NonD::initialize_level_mappings(), NonD::initialize_level_mappings(), Iterator::iteratedModel, NonDSampling::momentGrads, NonD::momentStats, Analyzer::numFunctions, NonD::pdfOutput, NonD::requestedGenRelLevels, NonD::requestedProbLevels, NonD::requestedRelLevels, NonD::requestedRespLevels, NonD::respLevelTarget, and Model::response_labels().

Referenced by NonDSampling::compute_statistics(), and NonDExpansion::run_sampler().

void transform_samples ( Pecos::ProbabilityTransformation & nataf, bool x_to_u = true ) [inline]
transform allSamples imported by alternate constructor. This is needed since random variable distribution parameters are not updated until run time and an imported sample_matrix is typically in x-space.

transform x_samples to u_samples for use by expansionSampler

References Analyzer::allSamples, and NonDSampling::numSamples.

Referenced by NonDLHSSampling::d_optimal_parameter_set().

void pre_run ( ) [inline], [protected], [virtual]
pre-run portion of run (optional); re-implemented by Iterators which can generate all Variables (parameter sets) a priori
pre-run phase, which a derived iterator may optionally reimplement; when not present, pre-run is likely inte-
gerated into the derived run function. This is a virtual function; when re-implementing, a derived class must call
its nearest parent’s pre_run(), if implemented, typically before performing its own implementation steps.
Reimplemented from Analyzer.
References NonDSampling::active_set_mapping(), Analyzer::pre_run(), and Iterator::subIteratorFlag.
Referenced by NonDMultilevelSampling::pre_run(), and NonDLHSSampling::pre_run().

```cpp
void core_run() [protected], [virtual]
```

Default implementation generates all Responses from either allSamples or allVariables.
Reimplemented from Iterator.
References NonDSampling::allDataFlag, Analyzer::evaluate_parameter_sets(), Iterator::iteratedModel, and Non-
DSampling::statsFlag.

```cpp
int num_samples() const [inline], [protected], [virtual]
```

Return current number of evaluation points. Since the calculation of samples, collocation points, etc. might be
costly, provide a default implementation here that backs out from the maxEvalConcurrency.
Reimplemented from Analyzer.
References NonDSampling::numSamples.
Referenced by NonDLHSSampling::archive_results(), NonDAadaptImpSampling::evaluate_samples(), NonD-
Sampling::get_parameter_sets(), NonDSampling::initialize_lhs(), NonDSampling::print_wilks_statistics(), NonDA-
adaptImpSampling::select_rep_points(), NonDLHSSampling::store_ranks(), and NonDSampling::transform_samples().

```cpp
void sampling_reset(int min_samples, bool all_data_flag, bool stats_flag) [inline], [protected], [virtual]
```

resets number of samples and sampling flags

used by DataFitSurrModel::build_global() to publish the minimum number of samples needed from the sam-
pling routine (to build a particular global approximation) and to set allDataFlag and statsFlag. In this case, all-
DataFlag is set to true (vectors of variable and response sets must be returned to build the global approximation)
and statsFlag is set to false (statistics computations are not needed).
Reimplemented from Iterator.
References NonDSampling::allDataFlag, NonDSampling::numSamples, NonDSampling::samplesIncrement, Non-
DSampling::samplesRef, and NonDSampling::statsFlag.

```cpp
void get_parameter_sets(Model & model) [inline], [protected], [virtual]
```

Uses lhsDriver to generate a set of samples from the distributions/bounds defined in the incoming model.
This version of get_parameter_sets() extracts data from the user-defined model in any of the four sampling
modes and populates class member allSamples.
Reimplemented from Analyzer.
References Analyzer::allSamples, and NonDSampling::numSamples.
Referenced by NonDLHSSampling::compute_pca(), NonDAadaptImpSampling::core_run(), NonDLHSSampling::
d_optimal_parameter_set(), NonDSampling::get_parameter_sets(), NonDLHSSampling::increment_lhs_parameter_-set(), NonDLHSSampling::initialize_lhs_set(), NonDMultilevelSampling::lf_increment(), NonDMultilevel-
Sampling::multilevel_control_variate_mc_Qcorr(), NonDMultilevelSampling::multilevel_control_variate_mc_Ycorr(), Non-
DMultilevelSampling::multilevel_mc_Qsum(), NonDMultilevelSampling::multilevel_mc_Ysum(), NonDLH-
SSampling::NonDLHSSampling(), NonDLHSSampling::pre_run(), and NonDMultilevelSampling::shared_increment().
void get_parameter_sets ( Model & model, const int num_samples, RealMatrix & design_matrix )
[inline], [protected], [virtual]

Uses lhsDriver to generate a set of samples from the distributions/bounds defined in the incoming model and populates the specified design matrix.

This version of get_parameter_sets() extracts data from the user-defined model in any of the four sampling modes and populates the specified design matrix.

Reimplemented from Analyzer.
References NonDSampling::get_parameter_sets().

void get_parameter_sets ( const RealVector & lower_bnds, const RealVector & upper_bnds )
[protected]

Uses lhsDriver to generate a set of uniform samples over lower_bnds/upper_bnds.

This version of get_parameter_sets() does not extract data from the user-defined model, but instead relies on the incoming bounded region definition. It only support a UNIFORM sampling mode, where the distinction of ACTIVE_UNIFORM vs. ALL_UNIFORM is handled elsewhere.

References Analyzer::allSamples, NonDSampling::initialize_lhs(), NonDSampling::lhsDriver, and NonDSampling::numSamples.

void get_parameter_sets ( const RealVector & means, const RealVector & std_devs, const RealVector & lower_bnds, const RealVector & upper_bnds, RealSymMatrix & correl )
[protected]

Uses lhsDriver to generate a set of normal samples.

This version of get_parameter_sets() does not extract data from the user-defined model, but instead relies on the incoming definition. It only support the sampling of normal variables.

References Analyzer::allSamples, NonDSampling::initialize_lhs(), NonDSampling::lhsDriver, and NonDSampling::numSamples.

void active_set_mapping ()
[protected]

in the case of sub-iteration, map from finalStatistics.active_set() requests to activeSet used in evaluate_parameter_sets()

Map ASV/DVV requests in final statistics into activeSet for use in evaluate_parameter_sets()

References Response::active_set_derivative_vector(), Response::active_set_request_vector(), Iterator::activeSet, ActiveSet::derivative_vector(), NonD::finalMomentsType, NonD::finalStatistics, Analyzer::numFunctions, ActiveSet::request_vector(), NonD::requestedGenRelLevels, NonD::requestedProbLevels, NonD::requestedRelLevels, NonD::requestedRespLevels, NonD::respLevelTarget, and NonD::totalLevelRequests.

Referenced by NonDSampling::pre_run().

void mode_counts ( const Variables & vars, size_t & cv_start, size_t & num_cv, size_t & div_start, size_t & num_div, size_t & dsv_start, size_t & num_dsv, size_t & drv_start, size_t & num_drv )
const
[protected]

compute sampled subsets (all, active, uncertain) within all variables (acv/adiv/adrv) from samplingVarsMode and model

This function and its helpers to follow are needed since NonDSampling supports a richer set of sampling modes than just the active variable subset. mode_counts() manages the samplingVarsMode setting, while its helper functions (view {design,aleatory_uncertain,epistemic_uncertain, uncertain,state}.counts) manage the active variables view. Similar to the computation of starts and counts in creating active variable views, the results of this function are starts and counts for use within model.all_() set/get functions.
References Variables::acv(), Variables::adiv(), Variables::adrv(), Variables::adsv(), SharedVariablesData::aleatory_uncertain_counts(), Variables::cv(), Variables::cv_start(), SharedVariablesData::design_counts(), Variables::div(), Variables::div_start(), Variables::drvr(), Variables::dsv(), Variables::dsv_start(), Variables::epistemic_uncertain_counts(), NonDSampling::samplingVarsMode, Variables::shared_data(), SharedVariablesData::state_counts(), Dakota::svd(), and SharedVariablesData::uncertain_counts.

Referenced by NonDLHSSampling::archive_results(), NonDSampling::compute_statistics(), NonDLHSSampling::d_optimal_parameter_set(), NonDSampling::get_parameter_sets(), NonDLHSSampling::post_input(), NonDLHSSampling::pre_run(), NonDSampling::print_statistics(), and NonDSampling::variables_to_sample().

The documentation for this class was generated from the following files:

- NonDSampling.hpp
- NonDSampling.cpp

14.161 NonDSparseGrid Class Reference

Derived nondeterministic class that generates N-dimensional Smolyak sparse grids for numerical evaluation of expectation integrals over independent standard random variables.

Inheritance diagram for NonDSparseGrid:

```
  NonDSparseGrid
  |          |          |
  +--------+--------+
  |        |        |
  |  NonD  |
  +--------+--------+
  |          |          |
  |  NonDIntegration  |
  |          |          |
  +--------+--------+
  |          |          |
  |  Iterator  |
  +----------+----------+
  |          |          |
  |  Analyzer  |
  +----------+----------+

Public Member Functions

- **NonDSparseGrid** (Model &model, unsigned short ssg_level, const RealVector &dim_pref, short exp_coeffs_soln_approach, short driver_mode, short growth_rate=Pecos::MODERATE_RESTRICTED_GROWTH, short refine_control=Pecos::NO_CONTROL, bool track_uniq_prod_wts=true)
- void **sparse_grid_level** (unsigned short ssg_level)
  
  update the sparse grid level (e.g., from a level sequence)
- void **increment_grid** ()
  
  increment ssgDriver::ssgLevel
- void **increment_grid_weights** (const RealVector &aniso_wts)
  
  update ssgDriver::ssgAnisoLevelWts and increment ssgDriver::ssgLevel based on specified anisotropic weighting
- void **increment_grid_weights** ()
  
  increment ssgDriver::ssgLevel based on existing anisotropic weighting
- void **decrement_grid** ()
  
  decrement ssgDriver::ssgLevel
Protected Member Functions

- NonDSparseGrid (ProblemDescDB &problem_db, Model &model)
  constructor
- ~NonDSparseGrid ()
  destructor
- void initialize_grid (const std::vector<Pecos::BasisPolynomial> &poly_basis)
initialize integration grid by drawing from polynomial basis settings

- void getParameterSets (Model &model)
  
  Generate one block of numSamples samples (ndim * num_samples), populating allSamples; ParamStudy is the only class that specializes to use allVariables.

- void samplingReset (int min_samples, bool all_data_flag, bool stats_flag)
- const RealVector & anisotropic_weights () const

Private Attributes

- short ssgDriverType
  type of sparse grid driver: combined, incremental, hierarchical, ...
- Pecos::SparseGridDriver * ssgDriver
  convenience pointer to the numIntDriver representation
- unsigned short ssgLevelSpec
  the user specification for the Smolyak sparse grid level, rendered anisotropic via dimPrefSpec
- unsigned short ssgLevelPrev
  value of ssgDriver->level() prior to increment_grid(), for restoration in decrement_grid() since increment must induce a change in grid size and this adaptive increment is not reversible

Additional Inherited Members

14.161.1 Detailed Description

Derived nondeterministic class that generates N-dimensional Smolyak sparse grids for numerical evaluation of expectation integrals over independent standard random variables.

This class is used by NonDPolynomialChaos and NonDStochCollocation, but could also be used for general numerical integration of moments. It employs 1-D Clenshaw-Curtis and Gaussian quadrature rules within Smolyak sparse grids.

14.161.2 Constructor & Destructor Documentation

NonDSparseGrid ( Model & model, unsigned short ssg_level, const RealVector & dim_pref, short exp_coeffs_soln_approach, short driver_mode, short growth_rate = Pecos::MODERATE_RESTRICTED_GROWTH, short refine_control = Pecos::NO_CONTROL, bool track_uniq_prod_wts = true )

This alternate constructor is used for on-the-fly generation and evaluation of sparse grids within PCE and SC.

References NonDIntegration::numIntDriver, NonDSparseGrid::ssgDriver, and NonDSparseGrid::ssgDriver-Type.

NonDSparseGrid ( ProblemDescDB & problem_db, Model & model ) [protected]

This constructor is called for a standard letter-envelope iterator instantiation. In this case, set Db list_nodes has been called and probDescDB can be queried for settings from the method specification. It is not currently used, as there is not a separate sparse_grid method specification.
14.162. NONDSTOCHCOLLOCATION CLASS REFERENCE

References Iterator::convergenceTol, Model::correction_type(), NonDIntegration::dimPrefSpec, ProblemDescDB::get_bool(), ProblemDescDB::get_int(), ProblemDescDB::get_short(), ProblemDescDB::get_ushort(), NonDSparseGrid::initialize_grid(), Iterator::maxEvalConcurrency, Model::multivariate_distribution(), NonDIntegration::numIntDriver, Iterator::outputLevel, Iterator::probDescDB, NonDSparseGrid::ssgDriver, NonDSparseGrid::ssgDriverType, and NonDSparseGrid::ssgLevelSpec.

14.161.3 Member Function Documentation

**int num_samples ( ) const [inline], [virtual]**

Return current number of evaluation points. Since the calculation of samples, collocation points, etc. might be costly, provide a default implementation here that backs out from the maxEvalConcurrency.

Reimplemented from Analyzer.

References NonDSparseGrid::ssgDriver.

**void sampling_reset ( int min_samples, bool all_data_flag, bool stats_flag ) [protected], [virtual]**

used by DataFitSurrModel::build_global() to publish the minimum number of points needed from the sparse grid routine in order to build a particular global approximation.

Reimplemented from Iterator.

References NonDSparseGrid::ssgDriver.

The documentation for this class was generated from the following files:

- NonDSparseGrid.hpp
- NonDSparseGrid.cpp

14.162 NONDStochCollocation Class Reference

Nonintrusive stochastic collocation approaches to uncertainty quantification.

Inheritance diagram for NonDStochCollocation:

```
   NonDStochCollocation
      |                        |  NonDSparseGrid.cpp
  NonDExpansion
      |                        |
   NonD
      |                        |  NonDSparseGrid.hpp
  NonDStochCollocation
      |                   |
   NonDMultilevelStochCollocation
```

References Iterator::convergenceTol, Model::correction_type(), NonDIntegration::dimPrefSpec, ProblemDescDB::get_bool(), ProblemDescDB::get_int(), ProblemDescDB::get_short(), ProblemDescDB::get_ushort(), NonDSparseGrid::initialize_grid(), Iterator::maxEvalConcurrency, Model::multivariate_distribution(), NonDIntegration::numIntDriver, Iterator::outputLevel, Iterator::probDescDB, NonDSparseGrid::ssgDriver, NonDSparseGrid::ssgDriverType, and NonDSparseGrid::ssgLevelSpec.
Public Member Functions

- **NonDStochCollocation (ProblemDescDB &problem_db, Model &model)**
  standard constructor
- **NonDStochCollocation (Model &model, short exp_coeffs_approach, unsigned short num_int, const RealVector &dim_pref, short u_space_type, short refine_type, short refine_control, short covar_control, short rule_nest, short rule_growth, bool piecewise_basis, bool use_derivs)**
  alternate constructor
- **~NonDStochCollocation ()**
  destructor
- **bool resize ()**
  reinitializes iterator based on new variable size
- **void metric_roll_up ()**
  perform any required expansion roll-ups prior to metric computation

Protected Member Functions

- **NonDStochCollocation (unsigned short method_name, ProblemDescDB &problem_db, Model &model)**
  short-cut ctor allowing derived class to replace logic in base class ctor (method_name is not necessary, rather it is just a convenient overload allowing the derived ML SC class to bypass the standard SC ctor)
- **NonDStochCollocation (unsigned short method_name, Model &model, short exp_coeffs_approach, const RealVector &dim_pref, short refine_type, short refine_control, short covar_control, short ml_alloc_control, short ml_discrep, short rule_nest, short rule_growth, bool piecewise_basis, bool use_derivs)**
  short-cut ctor allowing derived class to replace logic in base class ctor
- **void resolve_inputs (short &u_space_type, short &data_order)**
  perform error checks and mode overrides
- **void initialize_u_space_model ()**
  initialize uSpaceModel polynomial approximations with PCE/SC data
- **Real compute_covariance_metric (bool revert, bool print_metric)**
  compute 2-norm of change in response covariance
- **Real compute_level_mappings_metric (bool revert, bool print_metric)**
  compute 2-norm of change in final statistics
- **void initialize_covariance ()**
  initialize covariance pairings by passing all pointers for approximation j to approximation i
- **void compute_delta_mean (bool update_ref)**
  helper function to compute deltaRespVariance
- **void compute_delta_variance (bool update_ref, bool print_metric)**
  helper function to compute deltaRespVariance
- **void compute_delta_covariance (bool update_ref, bool print_metric)**
  helper function to compute deltaRespCovariance
- **void analytic_delta_level_mappings (const RealVector &level_maps_ref, RealVector &level_maps_new)**
  update analytic level mappings; this uses a lightweight approach for incremental statistics (no derivatives, no finalStatistics update)
- **void config_integration (unsigned short quad_order, unsigned short ssg_level, const RealVector &dim_pref, short u_space_type, Iterator &u_space_sampler, Model &g_u_model)**
configure u_space_sampler based on numerical integration specification

- void config_integration (short exp_coeffs_approach, unsigned short num_int, const RealVector &dim_pref, Iterator &u_space_sampler, Model &g_u_model)

configure u_space_sampler based on expansion coefficients approach

- void config_approximation_type (String &approx_type)

define approx_type based on expansion settings

Private Attributes

- RealVector deltaRespMean
  change in response means induced by a refinement candidate
- RealVector deltaRespVariance
  change in (DIAGONAL) response variance induced by a refinement candidate
- RealSymMatrix deltaRespCovariance
  change in (FULL) response covariance induced by a refinement candidate
- RealVector deltaLevelMaps
  change in response means induced by a refinement candidate

Additional Inherited Members

14.162.1 Detailed Description

Nonintrusive stochastic collocation approaches to uncertainty quantification.

The NonDStochCollocation class uses a stochastic collocation (SC) approach to approximate the effect of parameter uncertainties on response functions of interest. It utilizes the InterpPolyApproximation class to manage multidimensional Lagrange polynomial interpolants.

14.162.2 Constructor & Destructor Documentation

NonDStochCollocation ( ProblemDescDB & problem_db, Model & model )

standard constructor

This constructor is called for a standard letter-envelope iterator instantiation using the ProblemDescDB.

References Response::active_set(), Model::assign_rep(), ParallelLibrary::command_line_check(), NonDStochCollocation::config_approximation_type(), NonDStochCollocation::config_integration(), NonDExpansion::construct_expansion_sampler(), Model::current_response(), ActiveSet::derivative_vector(), ProblemDescDB::get_bool(), ProblemDescDB::get_iv(), ProblemDescDB::get_rv(), ProblemDescDB::get_short(), ProblemDescDB::get_string(), ProblemDescDB::get_ushort(), NonDStochCollocation::initialize_u_space_model(), Iterator::iteratedModel, NonDExpansion::numSamplesOnModel, Iterator::outputLevel, Iterator::parallelLib, Iterator::probDescDB, Model::qoi(), NonDStochCollocation::resolve_inputs(), and NonDExpansion::uSpaceModel.

NonDStochCollocation ( Model & model, short exp_coeffs_approach, unsigned short num_int, const RealVector & dim_pref, short u_space_type, short refine_type, short refine_control, short covar_control, short rule_nest, short rule_growth, bool piecewise_basis, bool use_derivs )

alternate constructor

This constructor is used for helper iterator instantiation on the fly.
References Response::active_set(), Model::assign_reps(), NonDStochCollocation::config_approximation_type(), NonDStochCollocation::config_integration(), Model::current_response(), ActiveSet::derivative_vector(), NonDStochCollocation::initialize_u_space_model(), Iterator::iteratedModel, Iterator::outputLevel, Model::qoi(), NonDStochCollocation::resolve_inputs(), and NonDExpansion::uSpaceModel.

NonDStochCollocation ( unsigned short method_name, ProblemDescDB & problem_db, Model & model )  
[protected]  
short-cut ctor allowing derived class to replace logic in base class ctor (method_name is not necessary, rather it is just a convenient overload allowing the derived ML SC class to bypass the standard SC ctor)  
This constructor is called from derived class constructors that customize the object construction.

NonDStochCollocation ( unsigned short method_name, Model & model, short exp_coeffs_approach, const RealVector & dim_pref, short refine_type, short refine_control, short ml_alloc_control, short ml_discrep, short rule_nest, short rule_growth, bool piecewise_basis, bool use_derivs )  
[protected]  
short-cut ctor allowing derived class to replace logic in base class ctor  
This constructor is called from derived class constructors that customize the object construction.  
References NonDExpansion::multilevAllocControl, and NonDExpansion::multilevDiscrepEmulation.

14.162.3 Member Function Documentation

Real compute_covariance_metric ( bool revert, bool print_metric )  
[protected], [virtual]  
compute 2-norm of change in response covariance  
computes the default refinement metric based on change in respCovariance  
Reimplemented from NonDExpansion.  
References NonDExpansion::compute_covariance_metric(), NonDStochCollocation::compute_delta_covariance(), NonDStochCollocation::compute_delta_mean(), NonDStochCollocation::compute_delta_variance(), NonDExpansion::covarianceControl, NonDStochCollocation::deltaRespCovariance, NonDStochCollocation::deltaRespVariance, NonDExpansion::expansionBasisType, NonDExpansion::relativeMetric, NonDExpansion::respCovariance, and NonDExpansion::respVariance.

Real compute_level_mappings_metric ( bool revert, bool print_metric )  
[protected], [virtual]  
compute 2-norm of change in final statistics  
computes a "goal-oriented" refinement metric employing computed Levels  
Reimplemented from NonDExpansion.  
References NonDExpansion::allVars, NonDStochCollocation::analytic_delta_level_mappings(), Model::approximations(), NonD::cdfFlag, NonDExpansion::compute_level_mappings_metric(), NonDExpansion::compute_numerical_level_mappings(), PecosApproximation::delta_beta(), PecosApproximation::delta_combined_beta(), PecosApproximation::delta_combined_z(), PecosApproximation::delta_z(), NonDStochCollocation::deltaLevelMaps, PecosApproximation::expansion_coefficient_flag(), NonDExpansion::expansionBasisType, NonDExpansion::initialPtU, Analyzer::numFunctions, NonD::print_level_mappings(), NonD::pull_level_mappings(), NonD::push_level_mappings(), NonDExpansion::relativeMetric, NonD::requestedGenRelLevels, NonD::requestedProbLevels, NonD::requestedRelLevels, NonD::requestedRespLevels, NonD::respLevelTarget, NonDExpansion::statsType, NonD::totalLevelRequests, and NonDExpansion::uSpaceModel.
void analytic_delta_level_mappings ( const RealVector & level_maps_ref, RealVector & level_maps_new )
[protected]

update analytic level mappings; this uses a lightweight approach for incremental statistics (no derivatives, no finalStatistics update)

In this function, we leave numerical stats alone, updating analytic level stats either using ref+delta or, if ref is invalid, though recomputation.

References NonDExpansion::allVars, Model::approximations(), PecosApproximation::beta(), NonD::cdfFlag, PecosApproximation::combined_beta(), NonDStochCollocation::deltaLevelMaps, NonDExpansion::initialPtU, Analyzer::numFunctions, NonD::requestedGenRelLevels, NonD::requestedProbLevels, NonD::requestedRelLevels, NonD::requestedRespLevels, NonD::respLevelTarget, NonDExpansion::statsType, NonD::totalLevelRequests, and NonDExpansion::uSpaceModel.

Referenced by NonDStochCollocation::compute_level_mappings_metric().

The documentation for this class was generated from the following files:

- NonDStochCollocation.hpp
- NonDStochCollocation.cpp

14.163  NonDSurrogateExpansion Class Reference

Generic uncertainty quantification with Model-based stochastic expansions.

Inheritance diagram for NonDSurrogateExpansion:

```
NonDSurrogateExpansion
   `-- NonDExpansion
      `-- NonD
         `-- Analyzer
            `-- Iterator

Public Member Functions

- NonDSurrogateExpansion (ProblemDescDB &problem_db, Model &model)
  standard constructor
- ~NonDSurrogateExpansion ()
  destructor
- void core_run ()
  perform a forward uncertainty propagation using PCE/SC methods

Protected Member Functions

- void print_results (std::ostream &)
```
14.163.1 Detailed Description

Generic uncertainty quantification with Model-based stochastic expansions.

The NonDSurrogateExpansion class leverages a Model specification for stochastic expansions (PCE, SC, FT) to build a stochastic emulator and then queries the emulator to generate the set of requested statistics.

14.163.2 Constructor & Destructor Documentation

NonDSurrogateExpansion ( ProblemDescDB & problem_db, Model & model )

standard constructor

This constructor is called for a standard letter-envelope iterator instantiation using the ProblemDescDB.

References Dakota::abort_handler(), NonDExpansion::construct_expansion_sampler(), ProblemDescDB::get_bool(), ProblemDescDB::get_iv(), ProblemDescDB::get_string(), ProblemDescDB::get_ushort(), Iterator::iterated_Model, Model::model_type(), Iterator::probDescDB, Model::surrogate_type(), and NonDExpansion::uSpaceModel.

The documentation for this class was generated from the following files:

- NonDSurrogateExpansion.hpp
- NonDSurrogateExpansion.cpp

14.164 NonDWASABIBayesCalibration Class Reference

WASABI - Weighted Adaptive Surrogate Approximations for Bayesian Inference.

Inheritance diagram for NonDWASABIBayesCalibration:

```
  NonDWASABIBayesCalibration
      NonDBayesCalibration
          NonDCalibration
              NonDAnalyzer
                  Iterator
```

Public Member Functions

- NonDWASABIBayesCalibration (ProblemDescDB &problem_db, Model &model)  
  standard constructor
- ~NonDWASABIBayesCalibration ()  
  destructor
- void compute_responses (RealMatrix &samples, RealMatrix &responses)
14.16. NONDWSABIBAYESCALIBRATION CLASS REFERENCE

Static Public Member Functions

- static void **problem.size** (int &chain_num, int &cr_num, int &gen_num, int &pair_num, int &par_num)
  
  *Initializer for problem size characteristics in WASABI*

- static void **problem.value** (std::string ∗chain_filename, std::string ∗gr_filename, double &gr_threshold, int &jumpstep, double limits[], int par_num, int &printstep, std::string ∗restart_read_filename, std::string ∗restart_write_filename)

  *Filename and data initializer for WASABI.*

Protected Member Functions

- void **calibrate** ()

- void **print_results** (std::ostream &s, short results_state=FINAL_RESULTS)
  
  *Print the final iterator results*

- void **extract_selected_posterior_samples** (const std::vector<int> &points_to_keep, const RealMatrix &samples_for_posterior_eval, const RealVector &posterior_density, RealMatrix &posterior_data) const

  *Extract a subset of samples for posterior eval according to the indices in points_to_keep.*

- void **export_posterior_samples_to_file** (const std::string filename, const RealMatrix &posterior_data) const

  *Export posterior data to file.*

Protected Attributes

- int **numPushforwardSamples**
  
  *Number of samples from the prior that is pushed forward through the model*

- RealVector **dataDistMeans**

  *The mean of the multivariate Gaussian distribution of the obs. data.*

- RealVector **dataDistCovariance**

  *The covariance of the multivariate Gaussian distribution of the obs. data.*

- std::string **dataDistFilename**

  *The filename of the file containing the data that with density estimator defines the distribution of the obs. data.*

- std::string **dataDistCovType**

  *The type of covariance data provided ("diagonal","matrix")*

- std::string **posteriorSamplesImportFile**

  *The filename of the import file containing samples at which the posterior will be evaluated.*

- unsigned short **posteriorSamplesImportFormat**

  *Format of imported posterior samples file.*

- std::string **exportPosteriorDensityFile**

  *The filename of the export file containing an arbitrary set of samples and their corresponding density values.*

- std::string **exportPosteriorSamplesFile**

  *The filename of the export file containing samples from the posterior and their corresponding density values.*

- unsigned short **exportFileFormat**

  *Format of imported posterior samples and values file.*

- bool **generateRandomPosteriorSamples**

  *Flag specifying whether to generate random samples from the posterior.*

- bool **evaluatePosteriorDensity**
Flag specifying whether to evaluate the posterior density at a set of samples.

- RealVector `paramMins`
  lower bounds on calibrated parameters

- RealVector `paramMaxs`
  upper bounds on calibrated parameters

- boost::mt19937 `rnumGenerator`
  random number engine for sampling the prior

- RealMatrix `momentStatistics`
  Matrix for moment statistics. Note that posterior values have density associated with them so we can’t use the compute moments in NonDSampling.

Additional Inherited Members

14.164.1 Detailed Description

WASABI - Weighted Adaptive Surrogate Approximations for Bayesian Inference.

This class performs Bayesian calibration using the WASABI approach

14.164.2 Constructor & Destructor Documentation

`NonDWASABIBayesCalibration ( ProblemDescDB & problem_db, Model & model )`

standard constructor

This constructor is called for a standard letter-envelope iterator instantiation. In this case, set_db_list_nodes has been called and probDescDB can be queried for settings from the method specification.

14.164.3 Member Function Documentation

`void calibrate ( ) [protected], [virtual]`

Perform the uncertainty quantification

Implements NonDBayesCalibration.

References Dakota::abort_handler(), Dakota::copy_data(), NonDWASABIBayesCalibration::dataDistCovariance, NonDWASABIBayesCalibration::dataDistFilename, NonDWASABIBayesCalibration::dataDistMeans, NonDBayesCalibration::emulatorType, NonDWASABIBayesCalibration::evaluatePosteriorDensity, NonDWASABIBayesCalibration::export_posterior_samples_to_file(), NonDWASABIBayesCalibration::exportPosteriorDensityFile, NonDWASABIBayesCalibration::exportPosteriorSamplesFile, NonDWASABIBayesCalibration::extract_selected_posterior_samples(), Dakota::generate_system_seed(), NonDWASABIBayesCalibration::generateRandomPosteriorSamples, NonDBayesCalibration::initialize_model(), NonDBayesCalibration::mcmcModel, NonDWASABIBayesCalibration::momentStatistics, Model::multivariate_distribution(), Analyzer::numContinuousVars, Analyzer::numFunctions, NonDWASABIBayesCalibration::numPushforwardSamples, Iterator::outputLevel, NonDWASABIBayesCalibration::paramMins, NonDWASABIBayesCalibration::posteriorSamplesImportFile, NonDBayesCalibration::prior_density(), NonDBayesCalibration::prior_sample(), NonDBayesCalibration::randomSeed, Dakota::read_unsized_data(), and NonDWASABIBayesCalibration::rnumGenerator.

`void print_results ( std::ostream & s, short results_state = FINAL_RESULTS ) [protected], [virtual]`

print the final iterator results
This virtual function provides additional iterator-specific final results outputs beyond the function evaluation summary printed in finalize_run().
Reimplemented from NonDBayesCalibration.
References Model::current_response(), Response::function_labels(), NonDBayesCalibration::mcmcModel, Non-DWASABIBayesCalibration::momentStatistics, and NonDSampling::print_moments().
The documentation for this class was generated from the following files:
- NonDWASABIBayesCalibration.hpp
- NonDWASABIBayesCalibration.cpp

14.165 NonlinearCGOptimizer Class Reference

Inheritance diagram for NonlinearCGOptimizer:

```
NonlinearCGOptimizer
  ^
  |   Iterator
  |   
  |   Minimizer
  |   
  |   Optimizer
  |   
  v
NonlinearCGOptimizer
```

Public Member Functions

- **NonlinearCGOptimizer** (ProblemDescDB &problem_db, Model &model)
  *standard constructor*
- **~NonlinearCGOptimizer** ()
  *destructor*
- Real linesearch_eval (const Real &trial_step, short req_val=1)
  *evaluate the objective function given a particular step size (public for use in boost_lseval functor; could use friend)*

Protected Member Functions

- void core_run ()
  *core portion of run; implemented by all derived classes and may include pre/post steps in lieu of separate pre/post*

Private Member Functions

- void parse_options ()
  *constructor helper function to parse misc_options from ProblemDescDB*
- void compute_direction ()
  *compute next direction via choice of method*
- bool compute_step ()
  *compute step: fixed, simple decrease, sufficient decrease*
• void bracket_min (Real &xa, Real &xb, Real &xc, Real &fa, Real &fb, Real &fc)
  
  bracket the 1-D minimum in the linesearch
• Real brent_minimize (Real a, Real b, Real tol)
  
  Perform 1-D minimization for the stepLength using Brent’s method.

Private Attributes
• Real initialStep
  
  initial step length
• Real linesearchTolerance
  
  approximate accuracy of abisscas in LS
• unsigned linesearchType
  
  type of line search (if any)
• unsigned maxLinesearchIters
  
  maximum evaluations in line search
• Real relFunctionTol
  
  stopping criterion for rel change in fn
• Real relGradientTol
  
  stopping criterion for rel reduction in g
• bool resetStep
  
  whether to reset step with each linesearch
• unsigned restartIter
  
  iter at which to reset to steepest descent
• unsigned updateType
  
  type of CG direction update
• unsigned iterCurr
  
  current iteration number
• RealVector designVars
  
  current decision variables in the major iteration
• RealVector trialVars
  
  decision variables in the linesearch
• Real functionCurr
  
  current function value
• Real functionPrev
  
  previous function value
• RealVector gradCurr
  
  current gradient
• RealVector gradPrev
  
  previous gradient
• RealVector gradDiff
  
  temporary for gradient difference (gradCurr - gradPrev)
• RealVector searchDirection
  
  current aggregate search direction
14.165. NONLINEARCGOPTIMIZER CLASS REFERENCE

- Real stepLength
  *current step length parameter alpha*
- Real gradDotGrad_init
  *initial gradient norm squared*
- Real gradDotGrad_curr
  *gradCurr dot gradCurr*
- Real gradDotGrad_prev
  *gradPrev dot gradPrev*

Additional Inherited Members

14.165.1 Detailed Description

Experimental implementation of nonlinear CG optimization

14.165.2 Member Function Documentation

`void core_run() [protected], [virtual]`

core portion of run; implemented by all derived classes and may include pre/post steps in lieu of separate pre/post
Virtual run function for the iterator class hierarchy. All derived classes need to redefine it.
Reimplemented from Iterator.
References Iterator::activeSet, Iterator::bestResponseArray, Iterator::bestVariablesArray, NonlinearCGOptimizer::compute_direction(), NonlinearCGOptimizer::compute_step(), Model::continuous_variables(), Iterator::convergenceTol, Dakota::copy_data(), Model::current_response(), NonlinearCGOptimizer::designVars, Model::evaluate(), Response::function_gradient_copy(), Response::function_gradient_view(), Response::function_values(), NonlinearCGOptimizer::functionCurr, NonlinearCGOptimizer::functionPrev, NonlinearCGOptimizer::gradCurr, NonlinearCGOptimizer::gradDotGrad_curr, NonlinearCGOptimizer::gradDotGrad_init, NonlinearCGOptimizer::gradDotGrad_prev, NonlinearCGOptimizer::gradPrev, Iterator::iteratedModel, NonlinearCGOptimizer::iterCurr, NonlinearCGOptimizer::linesearchType, Optimizer::localObjectiveRecast, Iterator::maxIterations, Minimizer::numContinuousVars, Iterator::outputLevel, NonlinearCGOptimizer::relFunctionTol, NonlinearCGOptimizer::relGradientTol, ActiveSet::request_values(), NonlinearCGOptimizer::searchDirection, NonlinearCGOptimizer::stepLength, and NonlinearCGOptimizer::trialVars.

`Real brent_minimize ( Real a, Real b, Real tol ) [private]`

Perform 1-D minimization for the stepLength using Brent’s method.
Perform 1-D minimization for the stepLength using Brent’s method. This is a C translation of fmin.f from Netlib.
References NonlinearCGOptimizer::linesearch_eval(), NonlinearCGOptimizer::maxLinesearchIters, and Iterator::outputLevel.
Referenced by NonlinearCGOptimizer::compute_step().
The documentation for this class was generated from the following files:

- NonlinearCGOptimizer.hpp
- NonlinearCGOptimizer.cpp
14.166 NonlinearCGTraits Class Reference

A version of TraitsBase specialized for NonlinearCG optimizers.

Inheritance diagram for NonlinearCGTraits:

```
  TraitsBase
   |
   v
NonlinearCGTraits
```

**Public Member Functions**

- **NonlinearCGTraits ()**  
  *default constructor*
- **virtual ~NonlinearCGTraits ()**  
  *destructor*
- **virtual bool is_derived ()**  
  *A temporary query used in the refactor.*
- **bool supports_continuous_variables ()**  
  *Return the flag indicating whether method supports continuous variables.*

14.166.1 Detailed Description

A version of TraitsBase specialized for NonlinearCG optimizers.

The documentation for this class was generated from the following file:

- NonlinearCGOptimizer.hpp

14.167 NormalizationScaler Class Reference

Normalizes the data using max and min feature values.

Inheritance diagram for NormalizationScaler:

```
  DataScaler
   |
   v
NormalizationScaler
```

**Public Member Functions**

- **NormalizationScaler (const MatrixXd &features, bool mean_normalization, double norm_factor=1.0)**  
  *Main constructor for NormalizationScaler.*
14.168. **NOSCALER CLASS REFERENCE**

**Additional Inherited Members**

14.167.1 **Detailed Description**

Normalizes the data using max and min feature values.

```cpp
if (mean_normalization): scaler_offsets = mean else: scaler_offsets = min
scale_factors = (max - min)/norm_factor
Setting mean_normalization = false scales each feature to [0,1]
```

14.167.2 **Constructor & Destructor Documentation**

NormalizationScaler (const MatrixXd &features, bool mean_normalization, double norm_factor = 1.0)

Main constructor for NormalizationScaler.

Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>features</th>
<th>Unscaled data matrix - (num_samples by num_features)</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>mean_normalization</td>
<td>Flag for whether to use mean or min value as the offset</td>
</tr>
<tr>
<td>in</td>
<td>norm_factor</td>
<td>Optional scaling factor applied to each feature Has a default value of 1.0</td>
</tr>
</tbody>
</table>

References DataScaler::check_for_zero_scaler_factor(), DataScaler::hasScaling, DataScaler::scaledFeatures, DataScaler::scalerFeaturesOffsets, and DataScaler::scalerFeaturesScaleFactors.

The documentation for this class was generated from the following files:

- DataScaler.hpp
- DataScaler.cpp

14.168 **NoScaler Class Reference**

Leaves the data unscaled.

Inheritance diagram for NoScaler:

```
DataScaler

NoScaler
```

**Public Member Functions**

- **NoScaler** (const MatrixXd &features)
  
  *Main constructor for NoScaler.*

**Additional Inherited Members**

14.168.1 **Detailed Description**

Leaves the data unscaled.

This DataScaler has fixed coefficients that amount to an identity operation. It is useful when the data has already been scaled or scaling is desired.
scaler_offsets = 0.0
scaler_factors = 1.0

14.168.2 Constructor & Destructor Documentation

NoScaler ( const MatrixXd & features )

Main constructor for NoScaler.

Parameters:

- `features`: Unscaled data matrix - (num_samples by num_features)

References DataScaler::hasScaling, DataScaler::scaledFeatures, DataScaler::scalerFeaturesOffsets, and DataScaler::scalerFeaturesScaleFactors.

The documentation for this class was generated from the following files:

- DataScaler.hpp
- DataScaler.cpp

14.169 NOWPACBlackBoxEvaluator Class Reference

Derived class for plugging Dakota evaluations into NOWPAC solver.
Inherits BlackBoxBaseClass.

Public Member Functions:

- **NOWPACBlackBoxEvaluator (Model &model)**
  - `constructor`
- void **evaluate** (std::vector<double> const &x, std::vector<double> &vals, void *param)
- void **evaluate** (std::vector<double> const &x, std::vector<double> &vals, std::vector<double> &noise, void *param)
- double **evaluate_samples** (std::vector<double> const &samples, const unsigned int index, std::vector<double> const &x)
- void **allocate_constraints** ()
- int **num_ineq_constraints** () const
- const SizetList & **nonlinear_inequality_mapping_indices** () const
- const RealList & **nonlinear_inequality_mapping_multipliers** () const
- const RealList & **nonlinear_inequality_mappingOffsets** () const
- void **set_unscaled_bounds** (const RealVector &l_bnds, const RealVector &u_bnds)
  - `set {lower,upper} Bounds`
- void **scale** (const RealVector &unscaled_x, RealArray &scaled_x) const
  - `perform scaling from [lower,upper] to [0,1]`
- void **unscale** (const RealArray &scaled_x, RealVector &unscaled_x) const
  - `invert scaling to return from [0,1] to [lower,upper]`
Private Attributes

- **Model iteratedModel**
  
  cache a local copy of the Model

- **RealVector lowerBounds**
  
  cache the active continuous lower bounds for scaling to [0,1]

- **RealVector upperBounds**
  
  cache the active continuous upper bounds for scaling to [0,1]

- **int numNowpacIneqConstr**
  
  aggregate unsupported constraint types as nonlinear inequalities

- **SizetList nonlinIneqConMappingIndices**
  
  a list of indices for referencing the DAKOTA nonlinear inequality constraints used in computing the corresponding NOWPAC constraints.

- **RealList nonlinIneqConMappingMultipliers**
  
  a list of multipliers for mapping the DAKOTA nonlinear inequality constraints to the corresponding NOWPAC constraints.

- **RealList nonlinIneqConMappingOffsets**
  
  a list of offsets for mapping the DAKOTA nonlinear inequality constraints to the corresponding NOWPAC constraints.

- **SizetList linIneqConMappingIndices**
  
  a list of indices for referencing the DAKOTA linear inequality constraints used in computing the corresponding NOWPAC constraints.

- **RealList linIneqConMappingMultipliers**
  
  a list of multipliers for mapping the DAKOTA linear inequality constraints to the corresponding NOWPAC constraints.

- **RealList linIneqConMappingOffsets**
  
  a list of offsets for mapping the DAKOTA linear inequality constraints to the corresponding NOWPAC constraints.

14.169.1 Detailed Description

Derived class for plugging Dakota evaluations into NOWPAC solver.

The documentation for this class was generated from the following files:

- NOWPACOptimizer.hpp
- NOWPACOptimizer.cpp

14.170 NOWPACOptimizer Class Reference

Wrapper class for the (S)NOWPAC optimization algorithms from Florian Augustin (MIT)

Inheritance diagram for NOWPACOptimizer:
Public Member Functions

- **NOWPACOptimizer** *(ProblemDescDB &problem_db, Model &model)*
  *standard constructor*

- **NOWPACOptimizer** *(Model &model)*
  *alternate constructor*

- **~NOWPACOptimizer** *
destructor*

- void **core_run** ()
  *core portion of run; implemented by all derived classes and may include pre/post steps in lieu of separate pre/post*

Private Member Functions

- void **initialize_options** ()
  *Shared constructor code.*

Private Attributes

- NOWPAC **nowpacSolver**
- NOWPACBlackBoxEvaluator **nowpacEvaluator**

Additional Inherited Members

14.170.1 Detailed Description

Wrapper class for the (S)NOWPAC optimization algorithms from Florian Augustin (MIT)

14.170.2 Member Function Documentation

void **core_run** () [virtual]

core portion of run; implemented by all derived classes and may include pre/post steps in lieu of separate pre/post

Virtual run function for the iterator class hierarchy. All derived classes need to redefine it.

Reimplemented from **Iterator**.
14.171. NOWPACTraits Class Reference

A version of TraitsBase specialized for NOWPAC optimizers.

Inheritance diagram for NOWPACTraits:

```
<table>
<thead>
<tr>
<th>TraitsBase</th>
</tr>
</thead>
<tbody>
<tr>
<td>NOWPACTraits</td>
</tr>
</tbody>
</table>
```

### Public Member Functions

- **NOWPACTraits ()**
  
  *default constructor*

- **virtual ~NOWPACTraits ()**
  
  *destructor*

- **virtual bool is-derived ()**
  
  *A temporary query used in the refactor.*

- **bool supports_continuous_variables ()**
  
  *Return the flag indicating whether method supports continuous variables.*

- **bool supports_linear_inequality ()**
  
  *Return the flag indicating whether method supports linear inequalities.*

- **bool supports_nonlinear_inequality ()**
  
  *Return the flag indicating whether method supports nonlinear inequalities.*

- **NONLINEAR_INEQUALITY_FORMAT nonlinear_inequality_format ()**
  
  *Return the format used for nonlinear inequality constraints.*

### Detailed Description

A version of TraitsBase specialized for NOWPAC optimizers.

The documentation for this class was generated from the following file:

- NOWPACOptimizer.hpp
14.172 NPSOLTraits Class Reference

Wrapper class for the NPSOL optimization library.

Inheritance diagram for NPSOLTraits:

```
  TraitsBase
   `-> NPSOL Traits
```

Public Member Functions

- **NPSOLTraits ()**
  
  *default constructor*

- **virtual ~NPSOLTraits ()**
  
  *destructor*

- **virtual bool is derived ()**
  
  *A temporary query used in the refactor.*

- **bool supports_continuous_variables ()**
  
  *Return the flag indicating whether method supports continuous variables.*

- **bool supports_linear_equality ()**
  
  *Return the flag indicating whether method supports linear equalities.*

- **bool supports_linear_inequality ()**
  
  *Return the flag indicating whether method supports linear inequalities.*

- **bool supports_nonlinear_inequality ()**
  
  *Return the flag indicating whether method supports nonlinear equalities.*

- **bool supports_nonlinear_inequality ()**
  
  *Return the flag indicating whether method supports nonlinear inequalities.*

- **NONLINEAR_INEQUALITY_FORMAT nonlinear_inequality_format ()**
  
  *Return the format used for nonlinear inequality constraints.*

14.172.1 Detailed Description

Wrapper class for the NPSOL optimization library.

The NPSLOptimizer class provides a wrapper for NPSOL, a Fortran 77 sequential quadratic programming library from Stanford University marketed by Stanford Business Associates. It uses a function pointer approach for which passed functions must be either global functions or static member functions. Any attribute used within static member functions must be either local to that function or accessed through a static pointer.

The user input mappings are as follows: **max_function_evaluations** is implemented directly in NPSLOptimizer’s evaluator functions since there is no NPSOL parameter equivalent, and **max_iterations**, **convergence_tolerance**, **output verbosity**, **verify_level**, **function precision**, and **linesearch_tolerance** are mapped into NPSOL’s “Major Iteration Limit”, “Optimality Tolerance”, “Major Print Level” (verbose: Major Print Level = 20; quiet: Major Print Level = 10), “Verify Level”, “Function Precision”, and “Linesearch Tolerance” parameters, respectively, using NPSOL’s npoptn() subroutine (as wrapped by npoptn2())

The documentation for this class was generated from the following file:

- NPSLOptimizer.hpp

14.173 OptDartsOptimizer Class Reference

Wrapper class for OptDarts Optimizer.

Inheritance diagram for OptDartsOptimizer:

```
  Iterator
   |
   v
Minimizer
   |
   v
Optimizer
   |
   v
OptDartsOptimizer
```

**Public Member Functions**

- OptDartsOptimizer (ProblemDescDB &problem_db, Model &model)
  Constructor.
- OptDartsOptimizer (Model &model)
  alternate constructor for Iterator instantiations by name
- ~OptDartsOptimizer ()
  Destructor.
- void core_run ()
  Calls the OptDarts algorithm.

**Private Member Functions**

- void load_parameters (Model &model)
  Convenience function for Parameter loading.
- double opt_darts_f ()
  Function evaluation.
- void opt_darts_execute (size_t num_dim, size_t budget, double *xmin, double *xmax, double TOL, size_t problem_index, double fw_MC, double fb_MC)
  Run the OPT-DARTS method.
- void opt_darts_initiate (double *xmin, double *xmax)
  Initialize OPT-DARTS.
- void opt_darts_reset_convex_hull ()
• size_t opt_darts_pick_candidate (size_t ifunc)
  Choose the next trial iterate.
• void retrieve_extended_neighbors (size_t icandidate)
• void opt_darts_sample_from_candidate_neighborhood (size_t icandidate, size_t ifunc)
• void DIRECT_sample_from_candidate_neighborhood (size_t icandidate)
• void opt_darts_add_dart ()
• void opt_darts_update_K_h_approximate_Voronoi (size_t isample)
• void opt_darts_terminate ()
  Release memory and exit cleanly.
• void opt_darts_plot_discs_2d (size_t icandidate)
  Convenience function for plotting iterates.
• void opt_darts_plot_hull_2d (size_t icandidate, size_t ifunc)
  Convenience function for plotting convex hull.
• void initiate_random_generator (unsigned long x)
• double generate_a_random_number ()
• void sample_uniformly_from_unit_sphere_surface (double *dart, size_t num_dim)
• bool trim_line_using_Hyperplane (size_t num_dim, double *st, double *end, double *qH, double *nH)

Private Attributes
• double * _xmin
• double * _xmax
• double * _dart
• double * _st
• double * _end
• double * _tmp_point
• double * _qH
• double * _nH
• double ** _x
• double ** _xc
• double ** _f
• double ** _K
• double * _h
• double * _r
• size_t **_neighbors
• size_t * _tmp_neighbors
• size_t * _ext_neighbors
• size_t * _num_ext_neighbors
• bool _use_opt_darts
• bool _estimate_K
• size_t _ib
• size_t * _num_samples
• size_t _budget
• size_t _num_dim
• double _diag
• size_t _problem_index
14.174. **OPTDARTSTRAITS CLASS REFERENCE**

- double _fb
- double _fw
- double _fval
- size_t _corner_index
- size_t _num_corners
- size_t * _corners
- double _epsilon
- double _fb_MC
- double _fw_MC
- double ** _xm
- double ** _xp
- double * _alpha_Deceptive
- double _Q [1220]
- int _indx
- double cc
- double c
- double zc
- double zx
- double zy
- size_t qlen
- bool use_DIRECT
- int numTotalVars
- int randomSeed
- int maxBlackBoxEvals
- int maxIterations

**Additional Inherited Members**

14.173.1 **Detailed Description**

Wrapper class for OptDarts Optimizer.

The documentation for this class was generated from the following files:

- OptDartsOptimizer.hpp
- OptDartsOptimizer.cpp

14.174 **OptDartsTraits Class Reference**

A version of **TraitsBase** specialized for OptDarts.

Inheritance diagram for OptDartsTraits:

```
  TraitsBase
     |        |
     v        v
  OptDartsTraits
```
Public Member Functions

- **OptDartsTraits ()**
  
  *default constructor*

- **virtual ~OptDartsTraits ()**
  
  *destructor*

- **virtual bool is_derived ()**
  
  *A temporary query used in the refactor.*

- **bool supports_continuous_variables ()**
  
  *Return the flag indicating whether method supports continuous variables.*

14.174.1 Detailed Description

A version of TraitsBase specialized for OptDarts.

The documentation for this class was generated from the following file:

- **OptDartsOptimizer.hpp**

14.175 Optimizer Class Reference

Base class for the optimizer branch of the iterator hierarchy.

Inheritance diagram for Optimizer:
Public Member Functions

- void get_common_stopping_criteria (int &max_fn_evals, int &max_iters, double &conv_tol, double &min_var_chg, double &obj_target)

- int num_nonlin_ineq_constraints_found () const

- template<typename AdapterT>
  bool get_variable_bounds_from_dakota (typename AdapterT::VecT &lower, typename AdapterT::VecT &upper)

- template<typename VecT>
  void get_responses_from_dakota (const RealVector &dak_fn_vals, VecT &funs, VecT &cEqs, VecT &cIneqs)

Static Public Member Functions

- static void not_available (const std::string &package_name)

  Static helper function: third-party opt packages which are not available.
Protected Member Functions

- **Optimizer**(std::shared_ptr<TraitsBase> traits)
  - default constructor
- **Optimizer**(ProblemDescDB &problem_db, Model &model, std::shared_ptr<TraitsBase> traits)
  - alternate constructor; accepts a model
- **Optimizer**(unsigned short method_name, Model &model, std::shared_ptr<TraitsBase> traits)
  - alternate constructor for "on the fly" instantiations
- **Optimizer**(unsigned short method_name, size_t num_cv, size_t num_div, size_t num_dsv, size_t num_drv, size_t num_lin_ineq, size_t num_lin_eq, size_t num_nln_ineq, size_t num_nln_eq, std::shared_ptr<TraitsBase> traits)
  - alternate constructor for "on the fly" instantiations
- **~Optimizer**();
  - destructor
- void **initialize_run**();
- void **post_run**(std::ostream &s);
- void **finalize_run**();
  - utility function to perform common operations following post_run(); deallocation and resetting of instance pointers
- void **print_results**(std::ostream &s, short results_state=FINAL_RESULTS)
- void **configure_constraint_maps**();
- int **configure_inequality_constraints**(CONSTRAINT_TYPE ctype)
- void **configure_equality_constraints**(CONSTRAINT_TYPE ctype, size_t index_offset)
- template<typename AdapterT>
  - void **get_linear_constraints_and_bounds**(typename AdapterT::VecT &lin_ineq_lower_bnds, typename AdapterT::VecT &lin_ineq_upper_bnds, typename AdapterT::VecT &lin_eq_targets, typename AdapterT::MatT &lin_ineq_coeffs, typename AdapterT::MatT &lin_eq_coeffs)

Protected Attributes

- **size_t numObjectiveFns**
  - number of objective functions (iterator view)
- **bool localObjectiveRecast**
  - flag indicating whether local recasting to a single objective is used
- **Optimizer * prevOptInstance**
  - pointer containing previous value of optimizerInstance
- **int numNonlinearIneqConstraintsFound**
  - number of nonlinear ineq constraints actually used (based on conditional and bigRealBoundSize
- std::vector<int> **constraintMapIndices**
  - map from Dakota constraint number to APPS constraint number
- std::vector<double> **constraintMapMultipliers**
  - multipliers for constraint transformations
- std::vector<double> **constraintMapOffsets**
  - offsets for constraint transformations
Static Protected Attributes

- static Optimizer * optimizerInstance
  
  pointer to Optimizer instance used in static member functions

Private Member Functions

- void reduce_model (bool local_nls_recast, bool require_hessians)
  
  Wrap iteratedModel in a RecastModel that performs (weighted) multi-objective or sum-of-squared residuals transformation.

- void objective_reduction (const Response &full_response, const BoolDeque &sense, const RealVector &full_wts, Response &reduced_response) const
  
  forward mapping: maps multiple primary response functions to a single weighted objective for single-objective optimizers

Static Private Member Functions

- static void primary_resp_reducer (const Variables &full_vars, const Variables &reduced_vars, const Response &full_response, Response &reduced_response)
  
  Recast callback to reduce multiple objectives or residuals to a single objective, with gradients and Hessians as needed.

Additional Inherited Members

14.175.1 Detailed Description

Base class for the optimizer branch of the iterator hierarchy.

The Optimizer class provides common data and functionality for DOTOptimizer, CONMINOptimizer, NPSOLOptimizer, SNLOptimizer, NLPQLPOptimizer, COLINOptimizer, OptDartsOptimizer, NCSUOptimizer, NonlinearCGOptimizer, NomadOptimizer, and JEGAOptimizer.

14.175.2 Member Function Documentation

void get_common_stopping_criteria ( int & max_fn_evals, int & max_iters, double & conv_tol, double & min_var_chg, double & obj_target ) [inline]

Convenience method for common optimizer stopping criteria vectors

  References Iterator::convergenceTol, ProblemDescDB::get_real(), Iterator::maxFunctionEvals, Iterator::maxIterations, and Iterator::probDescDB.

bool get_variable_bounds_from_dakota ( typename AdapterT::VecT & lower, typename AdapterT::VecT & upper ) [inline]

Method for transferring variable bounds from Dakota data to TPL data

  References Minimizer::bigIntBoundSize, Minimizer::bigRealBoundSize, and Iterator::iteratedModel.
void get_responses_from_dakota( const RealVector & dak_fn_vals, VecT & funs, VecT & cEqs, VecT & cIneqs ) [inline]

Method for transferring responses from Dakota data to TPL data

References Optimizer::constraintMapIndices, Optimizer::constraintMapMultipliers, Optimizer::constraintMapOffsets, Dakota::get_responses(), and Iterator::iteratedModel.

Referenced by APPSEvalMgr::recv().

void initialize_run( ) [protected], [virtual]

Implements portions of initialize_run specific to Optimizers. This function should be invoked (or reimplemented) by any derived implementations of initialize_run() (which would otherwise hide it).

Reimplemented from Minimizer.

Reimplemented in SNLLOptimizer, and ROLOptimizer.

References Optimizer::configure_constraint_maps(), Minimizer::initialize_run(), Model::is_null(), Iterator::iteratedModel, Iterator::myModelLayers, Optimizer::optimizerInstance, Optimizer::prevOptInstance, and Model::update_from_subordinate_model().

Referenced by ROLOptimizer::initialize_run(), CONMINOptimizer::initialize_run(), APPSOptimizer::initialize_run(), and SNLLOptimizer::initialize_run().

void post_run( std::ostream & s ) [protected], [virtual]

Implements portions of post_run specific to Optimizers. This function should be invoked (or reimplemented) by any derived implementations of post_run() (which would otherwise hide it).

Reimplemented from Minimizer.

Reimplemented in SNLLOptimizer.

References Dakota::abort_handler(), Iterator::bestResponseArray, Iterator::bestVariablesArray, Variables::continuous_variables(), ScalingModel::cv_scaled2native(), Minimizer::local_recast_retrieve(), Optimizer::localObjectiveRecast, Model::model_rep(), Minimizer::post_run(), ScalingModel::resp_scaled2native(), Minimizer::scaleFlag, and Minimizer::scalingModel.

Referenced by COLINOptimizer::post_run(), and SNLLOptimizer::post_run().

void finalize_run( ) [inline], [protected], [virtual]

utility function to perform common operations following post_run(); deallocation and resetting of instance pointers

Optional: perform finalization phases of run sequence, like deallocating memory and resetting instance pointers. Commonly used in sub-iterator executions. This is a virtual function; when re-implementing, a derived class must call its nearest parent’s finalize_run(), typically after performing its own implementation steps.

Reimplemented from Minimizer.

Reimplemented in SNLLOptimizer.

References Minimizer::finalize_run(), Optimizer::optimizerInstance, and Optimizer::prevOptInstance.

Referenced by SNLLOptimizer::finalize_run().

void print_results( std::ostream & s, short results_state = FINAL_RESULTS ) [protected], [virtual]

Redefines default iterator results printing to include optimization results (objective functions and constraints).

Reimplemented from Iterator.
References Dakota::abort_handler(), Iterator::bestResponseArray, Iterator::bestVariablesArray, Minimizer::calibrationDataFlag, ExperimentData::config_vars(), Dakota::data_pairs, Minimizer::dataTransformModel, Minimizer::expData, Model::interface_id(), Dakota::lookup_by_val(), Model::model_rep(), Minimizer::numContinuousVars, Minimizer::numNonlinearConstraints, Minimizer::numUserPrimaryFns, Minimizer::optimizationFlag, Minimizer::original_model(), Model::primary_response_fn_weights(), DataTransformModel::print_best_responses(), Minimizer::print_residuals(), Model::response_size(), Iterator::run_identifier(), and Variables::write().

`void configure_constraint_maps( ) [protected]`

Implements configuration of constraint maps, etc...

References Dakota::abort_handler(), Minimizer::bigRealBoundSize, Dakota::configure_inequality_constraint_maps(), Optimizer::constraintMapIndices, Optimizer::constraintMapMultipliers, Optimizer::constraintMapOffsets, Iterator::iteratedModel, Optimizer::numNonlinearIneqConstraintsFound, and Iterator::traits().

Referenced by Optimizer::initialize_run().

`void reduce_model ( bool local_nls_recast, bool require_hessians ) [private]`

Wrap iteratedModel in a RecastModel that performs (weighted) multi-objective or sum-of-squared residuals transformation.

Reduce model for least-squares or multi-objective transformation. Doesn’t map variables, or secondary responses. Maps active set for Gauss-Newton. Maps primary responses to single objective so user vs. iterated matters.

References Iterator::activeSet, Model::assign_rep(), Minimizer::calibrationDataFlag, Model::current_response(), Response::function_gradients(), Iterator::gnewton_set_recast(), Model::hessian_type(), Iterator::iteratedModel, Iterator::myModelLayers, Minimizer::numContinuousVars, Minimizer::numFunctions, Minimizer::numIterPrimaryFns, Minimizer::numNonlinearConstraints, Minimizer::numNonlinearIneqConstraints, Optimizer::numObjectiveFns, Minimizer::numTotalCalibTerms, Minimizer::numUserPrimaryFns, Iterator::outputLevel, Model::primary_fn_type(), Optimizer::primary_resp_reducer(), Model::primary_response_fn_sense(), Model::primary_response_fn_weights(), ActiveSet::request_vector(), and Response::reshape().

Referenced by Optimizer::Optimizer().

`void primary_resp.reducer ( const Variables & full_vars, const Variables & reduced_vars, const Response & full_response, Response & reduced_response ) [static], [private]`

Recast callback to reduce multiple objectives or residuals to a single objective, with gradients and Hessians as needed.

Objective function map from multiple primary response functions (objective or residuals) to a single objective. Currently supports weighted sum; may later want more general transformations, e.g., goal-oriented

References Iterator::iteratedModel, Optimizer::objective_reduction(), Optimizer::optimizerInstance, Iterator::outputLevel, Model::primary_response_fn_sense(), Model::primary_response_fn_weights(), and Model::subordinate_model().

Referenced by Optimizer::reduce_model().

`void objective_reduction ( const Response & full_response, const BoolDeque & sense, const RealVector & full_wts, Response & reduced_response ) const [private]`

forward mapping: maps multiple primary response functions to a single weighted objective for single-objective optimizers
CHAPTER 14. CLASS DOCUMENTATION

This function is responsible for the mapping of multiple objective functions into a single objective for publishing to single-objective optimizers. Used in DOTOptimizer, NPSOLOptimizer, SNLLOptimizer, and SGOPTApplication on every function evaluation. The simple weighting approach (using primaryRespFnWts) is the only technique supported currently. The weightings are used to scale function values, gradients, and Hessians as needed.

References Response::active_set, request_vector(), Response::function_gradient_view(), Response::function_gradients(), Response::function_hessian_view(), Response::function_hessians(), Response::function_values(), Response::num_functions(), Minimizer::numConstraints, Minimizer::objective(), Minimizer::objective_gradient(), Minimizer::objective_hessian(), Iterator::outputLevel, and Dakota::write_precision.

Referenced by Optimizer::primary_resp_reducer().

The documentation for this class was generated from the following files:

- DakotaOptimizer.hpp
- DakotaOptimizer.cpp

14.176 OutputManager Class Reference

Class to manage redirection of stdout/stderr, keep track of current redir state, and manage rank 0 output. Also manage tabular data output for post-processing with Matlab, Tecplot, etc. and delegate to Graphics for X Windows Graphics.

Public Member Functions

- **OutputManager ()**
  Default constructor (needed for default environment ctors)
- **OutputManager (const ProgramOptions &prog_opts, int dakota_world_rank=0, bool dakota_mpirun_flag=false)**
  Standard constructor, taking user-specified program options and optionally taking the rank of this process in Dakota’s MPI_Comm.
- **~OutputManager ()**
  Destructor that closes streams and other outputs.
- **void close_streams ()**
  Helper to close streams during destructor or abnormal abort
- **Graphics & graphics ()**
  Retrieve the graphics handler object
- **void parse (const ProgramOptions &prog_opts, const ProblemDescDB &problem_db)**
  Extract environment options from ProblemDescDB and update from late updates to ProgramOptions.
- **void startup_message (const String &start_msg)***
  Set the Dakota startup message ("Running on...")
- **void push_output_tag (const String &iterator_tag, const ProgramOptions &prog_opts, bool force_cout_redirect, bool force_rst_redirect)**
  Update the tag to use on files and rebind any streams as needed.
- **String build_output_tag () const**
  Return the full output tag
- **void pop_output_tag ()**
  (Potentially) remove an output context and rebind streams
- **void output_version (std::ostream &os=Cout) const**
Output the current Dakota version.

- void output_startup_message (std::ostream &os=Cout) const
  Output the startup header and time.

- void output_helper (const String &message, std::ostream &os) const
  Output only on Dakota world rank 0 (for version, help, etc.)

- void append_restart (const ParamResponsePair &prp)
  Append a parameter/response set to the restart file

- void add_datapoint (const Variables &vars, const String &iface, const Response &response)
  Adds data to each window in the 2d graphics and adds a row to the tabular data file based on the results of a model evaluation

- void create_tabular_datastream (const Variables &vars, const Response &resp)
  Initialize the tabular datastream on iterator leaders

- void close_tabular ()
  Close tabular datastream

- void graphics_counter (int cntr)
  Set graphicsCntr equal to cntr

- int graphics_counter () const
  Return graphicsCntr

- void tabular_counter_label (const std::string &label)
  Set tabularCntrLabel equal to label

- void init_results_db ()
  At runtime, initialize the global ResultsManager, tagging filename with MPI worldRank + 1 if needed.

- void archive_input (const ProgramOptions &prog_opts) const
  Archive the input file to the results database.

Public Attributes

- bool graph2DFlag
  Whether user requested 2D graphics plots

- bool tabularDataFlag
  Whether user requested tabular data file

- bool resultsOutputFlag
  Whether user requested results data output

- String tabularDataFile
  Filename for tabulation of graphics data

- String resultsOutputFile
  Filename for results data

- unsigned short modelEvalsSelection
  Models selected to store their evaluations.

- unsigned short interfEvalsSelection
  Interfaces selected to store their evaluations.
Private Member Functions

- void initial_redirects (const ProgramOptions &prog_opts)
  Perform initial output/error redirects from user requests.
- void read_write_restart (bool restart_requested, bool read_restart_flag, const String &read_restart_filename, size_t stop_restart_eval, const String &write_restart_filename)
  conditionally import evaluations from restart file, then always create or overwrite restart file

Private Attributes

- int worldRank
  output manager handles rank 0 only output when needed
- bool mpirunFlag
  some output is only for MPI runs
- StringArray fileTags
  set of tags for various input/output files (default none)
- ConsoleRedirector coutRedirector
  set of redirections for Dakota::Cout; stores any tagged filename when there are concurrent Iterators
- ConsoleRedirector cerrRedirector
  set of redirections for Dakota::Cerr; stores any tagged filename when there are concurrent Iterators and error redirection is requested
- std::vector< std::shared_ptr< RestartWriter > > restartDestinations
  Stack of active restart destinations; end is the last (active) redirection. All remain open until popped or destroyed.
- String startupMessage
  message to print at startup when proceeding to instantiate objects
- Graphics dakotaGraphics
  graphics and tabular data output handler used by meta-iterators, models, and approximations; encapsulated here so destroyed with the OutputManager
- unsigned short tabularFormat
  tabular format options; see enum
- int graphicsCntr
  used for x axis values in 2D graphics and for 1st column in tabular data
- std::ofstream tabularDataFStream
  file stream for tabulation of graphics data within compute_response
- std::string tabularCntrLabel
  label for counter used in first line comment w/i the tabular data file
- short outputLevel
  output level (for debugging only; not passed in)
- unsigned short resultsOutputFormat
  Output results format.

14.176.1 Detailed Description

Class to manage redirection of stdout/stderr, keep track of current redir state, and manage rank 0 output. Also manage tabular data output for post-processing with Matlab, Tecplot, etc. and delegate to Graphics for X Windows Graphics.
14.176.2 Constructor & Destructor Documentation

OutputManager ( const ProgramOptions & prog_opts, int dakota_world_rank = 0, bool dakota_mpirun_flag = false )

Standard constructor, taking user-specified program options and optionally taking the rank of this process in Dakota's MPI Comm.

Only get minimal information off ProgramOptions as may be updated later by broadcast.

References OutputManager::initial_redirects(), OutputManager::mpirunFlag, and Dakota::start_dakota_heartbeat().

14.176.3 Member Function Documentation

void pop_output_tag ( )

(Potentially) remove an output context and rebind streams

For now this assumes the tag is .<int>

References OutputManager::build_output_tag(), OutputManager::cerrRedirector, OutputManager::coutRedirector, OutputManager::fileTags, OutputManager::outputLevel, ConsoleRedirector::pop_back(), OutputManager::restart-Destinations, and OutputManager::worldRank.

Referenced by ParallelLibrary::pop_output_tag().

void add_datapoint ( const Variables & vars, const String & iface, const Response & response )

adds data to each window in the 2d graphics and adds a row to the tabular data file based on the results of a model evaluation

Adds data to each 2d plot and each tabular data column (one for each active variable and for each response function). graphicsCntr is used for the x axis in the graphics and the first column in the tabular data.

References Response::active_set_request_vector(), Graphics::add_datapoint(), OutputManager::dakotaGraphics, OutputManager::graphicsCntr, OutputManager::tabularDataFStream, and OutputManager::tabularFormat.

Referenced by Model::evaluate(), Model::synchronize(), and Model::synchronize_nowait().

void create_tabular_datastream ( const Variables & vars, const Response & response )

initialize the tabular datastream on iterator leaders

Opens the tabular data file stream and prints headings, one for each active continuous and discrete variable and one for each response function, using the variable and response function labels. This tabular data is used for post-processing of DAKOTA results in Matlab, Tecplot, etc.

References OutputManager::build_output_tag(), OutputManager::tabularDataFile, OutputManager::tabularDataFStream, and OutputManager::tabularFormat.

Referenced by SurrBasedMinimizer::initialize_graphics(), and Iterator::initialize_graphics().

The documentation for this class was generated from the following files:

- OutputManager.hpp
- OutputManager.cpp

14.177 OutputWriter Class Reference

Public Member Functions

- outputWriter (std::ostream *output_stream)
  
  ostream constructor; used to construct a writer to existing stream, e.g., std::cout
CHAPTER 14. CLASS DOCUMENTATION

- **OutputWriter** (const String &output_filename)
  
  file redirect constructor; opens an overwriting file stream to given name

- const String & filename () const
  
  the (possibly empty) file name for this stream

- std::ostream * output_stream ()
  
  a pointer to the stream, either cout/cerr or a file

**Protected Attributes**

- String outputFilename
  
  the name of the output file (empty when constructed from pointer)

- std::ofstream outputFS
  
  file output stream for console text; only open if string non-empty

- std::ostream * outputStream
  
  pointer to the stream for this writer

14.177.1 Detailed Description

Component to manage a redirected output or error stream

The documentation for this class was generated from the following files:

- OutputManager.hpp
- OutputManager.cpp

14.178 ParallelConfiguration Class Reference

Container class for a set of ParallelLevel list iterators that collectively identify a particular multilevel parallel configuration.

**Public Member Functions**

- ParallelConfiguration ()
  
  default constructor

- ParallelConfiguration (const ParallelConfiguration &pl)
  
  copy constructor

- ~ParallelConfiguration ()
  
  destructor

- ParallelConfiguration & operator= (const ParallelConfiguration &pl)
  
  assignment operator

- const ParallelLevel & w_parallel_level () const
  
  return the ParallelLevel corresponding to miPLIter.front()

- const ParallelLevel & mi_parallel_level (size_t index=NPOS) const
  
  return the ParallelLevel corresponding to miPLIter[index]

- const ParallelLevel & ie_parallel_level () const
  
  return the ParallelLevel corresponding to iePLIter
**const ParallelLevel & ea_parallel_level () const**

  return the ParallelLevel corresponding to eaPLIter

**bool w_parallel_level_defined () const**

  test for definition of world parallel level

**bool mi_parallel_level_defined (size_t index=_NPOS) const**

  test for definition of meta-iterator-iterator parallel level

**bool ie_parallel_level_defined () const**

  test for definition of iterator-evaluation parallel level

**bool ea_parallel_level_defined () const**

  test for definition of evaluation-analysis parallel level

**ParLevLIter w_parallel_level_iterator () const**

  return miPLIters.front()

**ParLevLIter mi_parallel_level_iterator (size_t index=_NPOS) const**

  return miPLIters[index]

**ParLevLIter ie_parallel_level_iterator () const**

  return iePLIter

**ParLevLIter ea_parallel_level_iterator () const**

  return eaPLIter

**size_t mi_parallel_level_index (ParLevLIter pl_iter) const**

  return the index within miPLIters corresponding to pl_iter

**size_t mi_parallel_level_last_index () const**

  return the index of the last entry in miPLIters

---

**Private Member Functions**

- **void assign (const ParallelConfiguration &pl)**

  assign the attributes of the incoming pl to this object

---

**Private Attributes**

- **short numParallelLevels**

  number of parallel levels

- **std::vector< ParLevLIter > miPLIters**

  list iterator for world level followed by any concurrent iterator partitions (there may be multiple per parallel configuration instance)

- **ParLevLIter iePLIter**

  list iterator identifying the iterator-evaluation parallelLevel (there can only be one)

- **ParLevLIter eaPLIter**

  list iterator identifying the evaluation-analysis parallelLevel (there can only be one)

- **ParLevLIter endPLIter**

  snapshot of the end of ParallelLibrary::parallelLevels; used for detecting when a component of the parallel configuration has been initialized
Friends

- class ParallelLibrary

the ParallelLibrary class has special access privileges in order to streamline implementation

14.178.1 Detailed Description

Container class for a set of ParallelLevel list iterators that collectively identify a particular multilevel parallel configuration.

Rather than containing the multilevel parallel configuration directly, ParallelConfiguration instead provides a set of list iterators which point into a combined list of ParallelLevels. This approach allows different configurations to reuse ParallelLevels without copying them. A list of ParallelConfigurations is contained in ParallelLibrary (ParallelLibrary::parallelConfigurations).

14.178.2 Member Function Documentation

const ParallelLevel & mi_parallel_level ( size_t index = _NPOS ) const [inline]
return the ParallelLevel corresponding to miPLIters[index]
If a meaningful index is not provided, return the last mi parallel level. This is useful within the Model context, for which we need the lowest level partition after any meta-iterator recursions.
References Dakota::NPOS, and ParallelConfiguration::miPLIters.
Referenced by ApplicationInterface::set_evaluation_communicators().

ParLevLIter mi_parallel_level_iterator ( size_t index = _NPOS ) const [inline]
return miPLIters[index]
If a meaningful index is not provided, return the last mi parallel level. This is useful within the Model context, for which we need the lowest level partition after any meta-iterator recursions.
References Dakota::NPOS, and ParallelConfiguration::miPLIters.
The documentation for this class was generated from the following file:

- ParallelLibrary.hpp

14.179 ParallelDirectApplicInterface Class Reference

Sample derived interface class for testing parallel simulator plug-ins using assign_rep().
Inheritance diagram for ParallelDirectApplicInterface:

```
   Interface
      |
      v
ApplicationInterface
      |
      v
DirectApplicInterface
      |
      v
ParallelDirectApplicInterface
```
Public Member Functions

- **ParallelDirectApplicInterface** (const Dakota::ProblemDescDB &problem_db, const MPI_Comm &analysis_comm)
  
  _constructor_

- **~ParallelDirectApplicInterface** ()
  
  _destructor_

Protected Member Functions

- **int derived_map_ac** (const Dakota::String &ac_name)
  
  _execute an analysis code portion of a direct evaluation invocation_

- **void derived_map_async** (const Dakota::ParamResponsePair &pair)
  
  _no-op hides base error; job batching occurs within wait_local_evaluations()_

- **void wait_local_evaluations** (Dakota::PRPQueue &prp_queue)
  
  _evaluate the batch of jobs contained in prp_queue_

- **void test_local_evaluations** (Dakota::PRPQueue &prp_queue)
  
  _invokes wait_local_evaluations() (no special nowait support)_

- **void set_communicators_checks** (int max_eval_concurrency)
  
  _no-op hides default run-time error checks at DirectApplicInterface level_

Private Member Functions

- **int text_book** (const Dakota::RealVector &c_vars, const Dakota::ShortArray &asv, Dakota::RealVector &fn_vals, Dakota::RealMatrix &fn_grads, Dakota::RealSymMatrixArray &fn_hessians)
  
  _demo evaluator function for parallel plug-ins_

Additional Inherited Members

14.179.1 Detailed Description

Sample derived interface class for testing parallel simulator plug-ins using assign_rep().

The plug-in ParallelDirectApplicInterface resides in namespace SIM and uses a copy of textbook() to perform parallel parameter to response mappings. It is used to demonstrate plugging in a parallel direct analysis driver into Dakota in library mode. Test input files can then use an analysis driver of “plugin_textbook”.

14.179.2 Member Function Documentation

**void test_local_evaluations ( Dakota::PRPQueue & prp_queue )** [inline], [protected]

_invokes wait_local_evaluations() (no special nowait support)_

For use by ApplicationInterface::serve_evaluations_async(), which can provide a batch processing capability within message passing schedulers (called using chain ApplicationInterface::serve_evaluations() from Model::serve() from IteratorScheduler::run_iterator()).

References ParallelDirectApplicInterface::wait_local_evaluations().

The documentation for this class was generated from the following files:

- PluginParallelDirectApplicInterface.hpp
- PluginParallelDirectApplicInterface.cpp
14.180 ParallelLevel Class Reference

Container class for the data associated with a single level of communicator partitioning.

Public Member Functions

- **ParallelLevel ()**
  default constructor
- **ParallelLevel (const ParallelLevel &pl)**
  copy constructor
- **~ParallelLevel ()**
  destructor
- **ParallelLevel & operator= (const ParallelLevel &pl)**
  assignment operator
- **bool dedicated_master () const**
  return dedicatedMasterFlag
- **bool server_master () const**
  return serverMasterFlag
- **bool message_pass () const**
  return messagePass
- **bool idle_partition () const**
  return idlePartition
- **int num_servers () const**
  return numServers
- **int processors_per_server () const**
  return procsPerServer
- **int processor_remainder () const**
  return procRemainder
- **const MPI_Comm & server_intra_communicator () const**
  return serverIntraComm
- **int server_communicator_rank () const**
  return serverCommRank
- **int server_communicator_size () const**
  return serverCommSize
- **const MPI_Comm & hub_server_intra_communicator () const**
  return hubServerIntraComm
- **int hub_server_communicator_rank () const**
  return hubServerCommRank
- **int hub_server_communicator_size () const**
  return hubServerCommSize
- **const MPI_Comm & hub_server_inter_communicator () const**
  return hubServerInterComm
- **MPI_Comm * hub_server_inter_communicators () const**
  return hubServerInterComms
• int server_id () const
  return serverId
• void read (MPIUnpackBuffer &s)
  read a ParallelLevel object from a packed MPI buffer
• void write (MPIPackBuffer &s) const
  write a ParallelLevel object to a packed MPI buffer
• bool null (const MPI_Comm &comm)
  test comm for MPI_COMM_NULL
• bool special (const MPI_Comm &comm)
  test comm for special identity that cannot be deallocated
• void clear ()
  deallocate the communicators in this ParallelLevel
• void alias (const ParallelLevel &pl)
  assign the attributes of the incoming pl to this object. For communicators, this is a lightweight copy which assigns
  the same pointer values as the incoming pl, resulting in the same context.
• void copy (const ParallelLevel &pl)
  deep copy the attributes of the incoming pl to this object using MPI_Comm_dup to create equivalent communicators
  with a unique context.
• void copy_config (const ParallelLevel &pl)
  copy the scalar attributes of the incoming pl to this object, omitting communicators

Private Attributes

• bool ownCommFlag
  signals Comm ownership for deallocation
• bool dedicatedMasterFlag
  signals dedicated master partitioning
• bool commSplitFlag
  signals a communicator split was used
• bool serverMasterFlag
  identifies master server processors
• bool messagePass
  flag for message passing at this level, < indicating work assignment among servers
• bool idlePartition
  identifies presence of an idle processor < partition at this level
• int numServers
  number of servers
• int procsPerServer
  processors per server
• int procRemainder
  proc remainder after equal distribution
• int serverId
server identifier

- *MPI_Comm* `serverIntraComm`
  intracommm. for each server partition
- *int* `serverCommRank`
  rank in `serverIntraComm`
- *int* `serverCommSize`
  size of `serverIntraComm`
- *MPI_Comm* `hubServerIntraComm`
  intracomm for all `serverCommRank==0`
  < w/i next higher level `serverIntraComm`
- *int* `hubServerCommRank`
  rank in `hubServerIntraComm`
- *int* `hubServerCommSize`
  size of `hubServerIntraComm`
- *MPI_Comm* `hubServerInterComm`
  intercomm. between a server & the hub
  < (on server partitions only)
- *MPI_Comm* `*hubServerInterComms`
  intercomm. array on hub processor

Friends

- class `ParallelLibrary`
  the ParallelLibrary class has special access privileges in order to streamline implementation

14.180.1 Detailed Description

Container class for the data associated with a single level of communicator partitioning.
A list of these levels is contained in `ParallelLibrary` (`ParallelLibrary::parallelLevels`), which defines all of the parallelism levels across one or more multilevel parallelism configurations.

14.180.2 Member Function Documentation

`void clear ( ) [inline]`

deallocate the communicators in this `ParallelLevel`

This appears to be more robust outside of the destructor due to interactions among managed deallocation and default deallocation (e.g., explicitly freeing a communicator and then default deallocating its handle).

References `ParallelLevel::dedicatedMasterFlag`, `ParallelLevel::hubServerInterComm`, `ParallelLevel::hubServerInterComms`, `ParallelLevel::hubServerIntraComm`, `ParallelLevel::idlePartition`, `ParallelLevel::numServers`, `ParallelLevel::ownCommFlag`, `ParallelLevel::serverId`, `ParallelLevel::serverIntraComm`, and `ParallelLevel::special()`.

The documentation for this class was generated from the following file:

- `ParallelLibrary.hpp`

14.181 ParallelLibrary Class Reference

Class for partitioning multiple levels of parallelism and managing message passing within these levels.
Public Member Functions

- **ParallelLibrary()**
  default constructor (used for dummy_lib)

- **ParallelLibrary(const MPIManager &mpi_mng, ProgramOptions &prog_opts, OutputManager &output_mng)**
  stand-alone and default library mode constructor; don’t require options

- **~ParallelLibrary()**
  destructor

- **const ParallelLevel & init_iterator_communicators(int iterator_servers, int procs_per_iterator, int min_procs_per_iterator, int max_procs_per_iterator, int max_iterator_concurrency, short default_config, short iterator_scheduling, bool peer_dynamic_avail)**
  split MPI_COMM_WORLD into iterator communicators

- **const ParallelLevel & init_evaluation_communicators(int evaluation_servers, int procs_per_evaluation, int min_procs_per_eval, int max_procs_per_eval, int max_evaluation_concurrency, int asynch_local_evaluation_concurrency, short default_config, short evaluation_scheduling, bool peer_dynamic_avail)**
  split an iterator communicator into evaluation communicators

- **const ParallelLevel & init_analysis_communicators(int analysis_servers, int procs_per_analysis, int min_procs_per_analysis, int max_procs_per_analysis, int max_analysis_concurrency, int asynch_local_analysis_concurrency, short default_config, short analysis_scheduling, bool peer_dynamic_avail)**
  split an evaluation communicator into analysis communicators

- **void print_configuration()**
  print the parallel level settings for a particular parallel configuration

- **void push_output_tag(const ParallelLevel &pl)**
  conditionally append an iterator server id tag to the hierarchical output tag, manage restart, and rebind cout/cerr

- **void pop_output_tag(const ParallelLevel &pl)**
  pop the last output tag and rebind streams as needed; pl isn’t yet used, but may be in the future when we generalize to arbitrary output context switching

- **void write_restart(const ParamResponsePair &prp)**
  write a parameter/response set to the restart file

- **ProgramOptions & program_options()**
  return programOptions reference

- **OutputManager & output_manager()**
  return outputManager reference

- **void terminate_modelcenter()**
  terminate ModelCenter if running

- **void abort_helper(int code)**
  finalize MPI with correct communicator for abort

- **bool command_line_check() const**
  return checkFlag

- **bool command_line_pre_run() const**
  return preRunFlag

- **bool command_line_run() const**
  return runFlag

- **bool command_line_post_run() const**
return postRunFlag

• bool command_line_user_modes () const
  return userModesFlag

• const String & command_line_pre_run_input () const
  preRunInput filename

• const String & command_line_pre_run_output () const
  preRunOutput filename

• const String & command_line_run_input () const
  runInput filename

• const String & command_line_run_output () const
  runOutput filename

• const String & command_line_post_run_input () const
  postRunInput filename

• const String & command_line_post_run_output () const
  postRunOutput filename

• void send (MPIPackBuffer &send_buff, int dest, int tag, const ParallelLevel &parent_pl, const ParallelLevel &child_pl)
  blocking buffer send at the current communication level

• void send (int &send_int, int dest, int tag, const ParallelLevel &parent_pl, const ParallelLevel &child_pl)
  blocking integer send at the current communication level

• void isend (MPIPackBuffer &send_buff, int dest, int tag, MPI_Request &send_req, const ParallelLevel &parent_pl, const ParallelLevel &child_pl)
  nonblocking buffer send at the current communication level

• void isend (int &send_int, int dest, int tag, MPI_Request &send_req, const ParallelLevel &parent_pl, const ParallelLevel &child_pl)
  nonblocking integer send at the current communication level

• void recv (MPIUnpackBuffer &recv_buff, int source, int tag, MPI_Status &status, const ParallelLevel &parent_pl, const ParallelLevel &child_pl)
  blocking buffer receive at the current communication level

• void recv (int &recv_int, int source, int tag, MPI_Status &status, const ParallelLevel &parent_pl, const ParallelLevel &child_pl)
  blocking integer receive at the current communication level

• void irecv (MPIUnpackBuffer &recv_buff, int source, int tag, MPI_Request &recv_req, const ParallelLevel &parent_pl, const ParallelLevel &child_pl)
  nonblocking buffer receive at the current communication level

• void irecv (int &recv_int, int source, int tag, MPI_Request &recv_req, const ParallelLevel &parent_pl, const ParallelLevel &child_pl)
  nonblocking integer receive at the current communication level

• void check_mi_index (size_t &index) const
  process_NPOS default and perform error checks

• void send_mi (int &send_int, int dest, int tag, size_t index=_NPOS)
  blocking send at the metaiterator-iterator communication level

• void isend_mi (int &send_int, int dest, int tag, MPI_Request &send_req, size_t index=_NPOS)
  nonblocking send at the metaiterator-iterator communication level
• void recv_mi (int &recv_int, int source, int tag, MPI_Status &status, size_t index=NPOS)
  blocking receive at the metaiterator-iterator communication level
• void irecv_mi (int &recv_int, int source, int tag, MPI_Request &recv_req, size_t index=NPOS)
  nonblocking receive at the metaiterator-iterator communication level
• void send_mi (MPIPackBuffer &send_buff, int dest, int tag, size_t index=NPOS)
  blocking send at the metaiterator-iterator communication level
• void isend_mi (MPIPackBuffer &send_buff, int dest, int tag, MPI_Request &send_req, size_t index=NPOS)
  nonblocking send at the metaiterator-iterator communication level
• void recv_mi (MPIUnpackBuffer &recv_buff, int source, int tag, MPI_Status &status, size_t index=NPOS)
  blocking receive at the metaiterator-iterator communication level
• void irecv_mi (MPIUnpackBuffer &recv_buff, int source, int tag, MPI_Request &recv_req, size_t index=NPOS)
  nonblocking receive at the metaiterator-iterator communication level
• void send_ie (int &send_int, int dest, int tag)
  blocking send at the iterator-evaluation communication level
• void isend_ie (int &send_int, int dest, int tag, MPI_Request &send_req)
  nonblocking send at the iterator-evaluation communication level
• void recv_ie (int &recv_int, int source, int tag, MPI_Status &status)
  blocking receive at the iterator-evaluation communication level
• void irecv_ie (int &recv_int, int source, int tag, MPI_Request &recv_req)
  nonblocking receive at the iterator-evaluation communication level
• void send_ie (MPIPackBuffer &send_buff, int dest, int tag)
  blocking send at the iterator-evaluation communication level
• void isend_ie (MPIPackBuffer &send_buff, int dest, int tag, MPI_Request &send_req)
  nonblocking send at the iterator-evaluation communication level
• void recv_ie (MPIUnpackBuffer &recv_buff, int source, int tag, MPI_Status &status)
  blocking receive at the iterator-evaluation communication level
• void irecv_ie (MPIUnpackBuffer &recv_buff, int source, int tag, MPI_Request &recv_req)
  nonblocking receive at the iterator-evaluation communication level
• void send_ea (int &send_int, int dest, int tag)
  blocking send at the evaluation-analysis communication level
• void isend_ea (int &send_int, int dest, int tag, MPI_Request &send_req)
  nonblocking send at the evaluation-analysis communication level
• void recv_ea (int &recv_int, int source, int tag, MPI_Status &status)
  blocking receive at the evaluation-analysis communication level
• void irecv_ea (int &recv_int, int source, int tag, MPI_Request &recv_req)
  nonblocking receive at the evaluation-analysis communication level
• void bcast (int &data, const ParallelLevel &pl)
  broadcast an integer across the serverIntraComm of a ParallelLevel
• void bcast (short &data, const ParallelLevel &pl)
  broadcast an integer across the serverIntraComm of a ParallelLevel
• void bcast (MPIPackBuffer &send_buff, const ParallelLevel &pl)
  broadcast a MPIPackBuffer across the serverIntraComm of a ParallelLevel
• void bcast (MPIUnpackBuffer &recv_buff, const ParallelLevel &pl)
  broadcast a MPIUnpackBuffer across the serverIntraComm of a ParallelLevel
• void bcast_hs (int &data, const ParallelLevel &pl)
  broadcast an integer across the hubServerIntraComm of a ParallelLevel
• void bcast_hs (MPIPackBuffer &send_buff, const ParallelLevel &pl)
  broadcast a MPIPackBuffer across the hubServerIntraComm of a ParallelLevel
• void bcast_hs (MPIUnpackBuffer &recv_buff, const ParallelLevel &pl)
  broadcast a MPIUnpackBuffer across the hubServerIntraComm of a ParallelLevel
• void bcast_w (int &data)
  broadcast an integer across MPI_COMM_WORLD
• void bcast_i (int &data, size_t index=NPOS)
  broadcast an integer across an iterator communicator
• void bcast_s (short &data, size_t index=NPOS)
  broadcast a short integer across an iterator communicator
• void bcast_e (int &data)
  broadcast an integer across an evaluation communicator
• void bcast_a (int &data)
  broadcast an integer across an analysis communicator
• void bcast_mi (int &data, size_t index=NPOS)
  broadcast an integer across a metaiterator-iterator intra communicator
• void bcast_w (MPIPackBuffer &send_buff)
  broadcast a packed buffer across MPI_COMM_WORLD
• void bcast_i (MPIPackBuffer &send_buff, size_t index=NPOS)
  broadcast a packed buffer across an iterator communicator
• void bcast_e (MPIPackBuffer &send_buff)
  broadcast a packed buffer across an evaluation communicator
• void bcast_a (MPIPackBuffer &send_buff)
  broadcast a packed buffer across an analysis communicator
• void bcast_mi (MPIPackBuffer &send_buff, size_t index=NPOS)
  broadcast a packed buffer across a metaiterator-iterator intra communicator
• void bcast_w (MPIUnpackBuffer &recv_buff)
  matching receive for packed buffer broadcast across MPI_COMM_WORLD
• void bcast_i (MPIUnpackBuffer &recv_buff, size_t index=NPOS)
  matching receive for packed buffer bcast across an iterator communicator
• void bcast_e (MPIUnpackBuffer &recv_buff)
  matching receive for packed buffer bcast across an evaluation communicator
• void bcast_a (MPIUnpackBuffer &recv_buff)
  matching receive for packed buffer bcast across an analysis communicator
• void bcast_mi (MPIUnpackBuffer &recv_buff, size_t index=NPOS)
  matching recv for packed buffer bcast across a metaiterator-iterator intra comm
• void barrier_w ()
  enforce MPI_Barrier on MPI_COMM_WORLD
• void barrier_i (size_t index=NPOS)
enforce MPI Barrier on an iterator communicator
- `void barrier_e()`
enforce MPI Barrier on an evaluation communicator
- `void barrier_a()`
enforce MPI Barrier on an analysis communicator
- `void reduce_sum_ea(double *local_vals, double *sum_vals, int num_vals)`
  compute a sum over an eval-analysis intra-communicator using MPI_Reduce
- `void reduce_sum_a(double *local_vals, double *sum_vals, int num_vals)`
  compute a sum over an analysis communicator using MPI_Reduce
- `void test(MPI_Request &request, int &test_flag, MPI_Status &status)`
  test a nonblocking send/receive request for completion
- `void wait(MPI_Request &request, MPI_Status &status)`
  wait for a nonblocking send/receive request to complete
- `void waitall(int num_recvs, MPI_Request *recv_reqs)`
  wait for all messages from a series of nonblocking receives
- `void waitsome(int num_sends, MPI_Request *recv_requests, int *num_recvvs, int *index_array, MPI_Status *&status_array)`
  wait for at least one message from a series of nonblocking receives but complete all that are available
- `void free(MPI_Request &request)`
  free an MPI Request
- `int world_size() const`
  return MPIManager::worldSize
- `int world_rank() const`
  return MPIManager::worldRank
- `bool mpirun_flag() const`
  return MPIManager::mpirunFlag
- `bool is_null() const`
  return dummyFlag
- `Real parallel_time() const`
  returns current MPI wall clock time
- `void parallel_configuration_iterator(ParConfigLIter pc_iter)`
  set the current ParallelConfiguration node
- `ParConfigLIter parallel_configuration_iterator() const`
  return the current ParallelConfiguration node
- `const ParallelConfiguration & parallel_configuration() const`
  return the current ParallelConfiguration instance
- `size_t num_parallel_configurations() const`
  returns the number of entries in parallelConfigurations
- `bool parallel_configuration_is_complete()`
  identifies if the current ParallelConfiguration has been fully populated
- `void increment_parallel_configuration(ParLevLIter mi_pl_iter)`
  add a new node to parallelConfigurations and increment currPCIter; limit miPLIters within new configuration to mi_pl_iter level
void increment_parallel_configuration ()

add a new node to parallelConfigurations and increment currPCIter; copy all of miPLIters into new configuration

bool parallel_level_defined () const

test current parallel configuration for definition of world parallel level

bool mi_parallel_level_defined (size_t index=NPOS) const

test current parallel configuration for definition of meta-iterator-iterator parallel level

bool ie_parallel_level_defined () const

test current parallel configuration for definition of iterator-evaluation parallel level

bool ea_parallel_level_defined () const

test current parallel configuration for definition of evaluation-analysis parallel level

ParLevLIter parallel_level_iterator ()

for this level, access through ParallelConfiguration is not necessary

size_t parallel_level_index (ParLevLIter pl_iter)

return the index within parallelLevels corresponding to pl_iter

std::vector<MPI_Comm> analysis_intra_communicators ()

return the set of analysis intra communicators for all parallel configurations (used for setting up direct simulation interfaces prior to execution time).

Private Member Functions

void init_mpi_comm ()

convenience function for initializing DAKOTA’s top-level MPI communicators, based on dakotaMPIComm

void initialize_timers ()

initialize DAKOTA and UTILIB timers

void output_timers ()

conditionally output timers in destructor

void init_communicators (const ParallelLevel &parent_pl, int num_servers, int procs_per_server, int min_procs_per_server, int max_procs_per_server, int max_concurrency, int asynch_local_concurrency, short default_config, short scheduling_override, bool peer_dynamic_avail)

split a parent communicator into child server communicators

void split_communicator_dedicated_master (const ParallelLevel &parent_pl, ParallelLevel &child_pl)

split a parent communicator into a dedicated master processor and num_servers child communicators

void split_communicator_peer_partition (const ParallelLevel &parent_pl, ParallelLevel &child_pl)

split a parent communicator into num_servers peer child communicators (no dedicated master processor)

void resolve_inputs (ParallelLevel &child_pl, int avail_procs, int min_procs_per_server, int max_procs_per_server, int max_concurrency, int capacity_multiplier, short default_config, short scheduling_override, bool peer_dynamic_avail, short print_rank)

resolve user inputs into a sensible partitioning scheme

void bcast (int &data, const MPI_Comm &comm)

broadcast an integer across a communicator

void bcast (short &data, const MPI_Comm &comm)

broadcast a short integer across a communicator

void bcast (MPIPackBuffer &send_buff, const MPI_Comm &comm)

send a packed buffer across a communicator using a broadcast
• void bcast (MPIUnpackBuffer &recv_buff, const MPI_Comm &comm)
  matching receive for a packed buffer broadcast
• void barrier (const MPI_Comm &comm)
  enforce MPI_Barrier on comm
• void reduce_sum (double *local_vals, double *sum_vals, int num_vals, const MPI_Comm &comm)
  compute a sum over comm using MPI_Reduce
• void check_error (const String &err_source, int err_code)
  check the MPI return code and abort if error
• void alias_as_server_comm (const ParallelLevel &parent_pl, ParallelLevel &child_pl)
  convenience function for updating child serverIntraComm from parent serverIntraComm (shallow Comm copy)
• void copy_as_server_comm (const ParallelLevel &parent_pl, ParallelLevel &child_pl)
  convenience function for updating child serverIntraComm from parent serverIntraComm (deep Comm copy)
• void alias_as_hub_server_comm (const ParallelLevel &parent_pl, ParallelLevel &child_pl)
  convenience function for updating child hubServerIntraComm from parent serverIntraComm (shallow Comm copy)
• void copy_as_hub_server_comm (const ParallelLevel &parent_pl, ParallelLevel &child_pl)
  convenience function for updating child hubServerIntraComm from parent serverIntraComm (deep Comm copy)

Private Attributes
• const MPIManager & mpiManager
  reference to the MPI manager with Dakota’s MPI options
• ProgramOptions & programOptions
  programOptions is non-const due to updates from broadcast
• OutputManager & outputManager
  Non-const output handler to help with file redirection.
• bool dummyFlag
  prevents multiple MPI_Finalize calls due to dummy_lib
• bool outputTimings
  timing info only beyond help/version/check
• Real startCPUtime
  start reference for UTILIB CPU timer
• Real startWCTime
  start reference for UTILIB wall clock timer
• Real startMPTime
  start reference for MPI wall clock timer
• long startClock
  start reference for local clock() timer measuring < parent+child CPU
• std::list< ParallelLevel > parallelLevels
  the complete set of parallelism levels for managing multilevel parallelism among one or more configurations
• std::list< ParallelConfiguration > parallelConfigurations
  the set of parallel configurations which manage list iterators for indexing into parallelLevels
• ParConfigLIter currPCIter
  list iterator identifying the current node in parallelConfigurations
14.181.1 Detailed Description

Class for partitioning multiple levels of parallelism and managing message passing within these levels.

The ParallelLibrary class encapsulates all of the details of performing message passing within multiple levels of parallelism. It provides functions for partitioning of levels according to user configuration input and functions for passing messages within and across MPI communicators for each of the parallelism levels. If support for other message-passing libraries beyond MPI becomes needed (PVM, ...), then ParallelLibrary would be promoted to a base class with virtual functions to encapsulate the library-specific syntax.

14.181.2 Constructor & Destructor Documentation

ParallelLibrary ( )

default constructor (used for dummy_lib)

This constructor is used for creation of the global dummy_lib object, which is used to satisfy initialization requirements when the real ParallelLibrary object is not available.

ParallelLibrary ( const MPIManager & mpi_mngr, ProgramOptions & prog_opts, OutputManager & output_mgr )

stand-alone and default library mode constructor; don’t require options

library mode constructor accepting communicator

TODO: Update comment.

Same constructor is used for executable and library environments and sequencing of object construction is ordered, so no need to separately get updates off command line (programOptions)

References ParallelLibrary::init_mpi_comm(), and ParallelLibrary::initialize_timers().

14.181.3 Member Function Documentation

void push_output_tag ( const ParallelLevel & pl )

conditionally append an iterator server id tag to the hierarchical output tag, manage restart, and rebind cout/cerr

If the user has specified the use of files for DAKOTA standard output and/or standard error, then bind these filenames to the Cout/Cerr macros. In addition, if concurrent iterators are to be used, create and tag multiple output streams in order to prevent jumbled output. Manage restart file(s) by processing any incoming evaluations from an old restart file and by setting up the binary output stream for new evaluations. Only master iterator processor(s) read & write restart information. This function must follow init_iterator_communicators so that restart can be managed properly for concurrent iterator strategies. In the case of concurrent iterators, each iterator has its own restart file tagged with iterator number.

References ParallelLibrary::bcast(), ParallelLevel::dedicatedMasterFlag, OutputManager::graph2DFlag, ParallelLevel::hubServerCommRank, ParallelLevel::hubServerCommSize, ParallelLevel::hubServerIntraComm, ParallelLevel::numServers, ParallelLibrary::outputManager, ParallelLibrary::programOptions, OutputManager::push_output_tag(), OutputManager::resultsOutputFile, OutputManager::resultsOutputFlag, ParallelLevel::serverCommRank, ParallelLevel::serverId, MPIPackBuffer::size(), OutputManager::tabularDataFile, and OutputManager::tabularDataFlag.

Referenced by Environment::construct(), and IteratorScheduler::partition().

void terminate_modelcenter ( )

terminate ModelCenter if running
Close streams associated with manage_outputs and manage_restart and terminate any additional services that may be active.

References Dakota::abort_handler(), Dakota::dc_ptr_int, and Dakota::mc_ptr_int.

Referenced by ParallelLibrary::~ParallelLibrary().

**void increment_parallel_configuration** ( ParLevLIter mi_pl_iter ) [inline]

add a new node to parallelConfigurations and increment currPCIter; limit miPLIters within new configuration to mi_pl_iter level

Called from the ParallelLibrary ctor and from Model::init_communicators(). An increment is performed for each Model initialization except the first (which inherits the world level from the first partial configuration).

References ParallelLibrary::currPCIter, ParallelConfiguration::eaPLIter, ParallelConfiguration::endPLIter, ParallelConfiguration::miPLIters, ParallelConfiguration::numParallelLevels, ParallelLibrary::parallelConfigurations, and ParallelLibrary::parallelLevels.

Referenced by Iterator::init_communicators(), and Model::init_communicators().

**void init_mpi_comm** ( ) [private]

convenience function for initializing DAKOTA's top-level MPI communicators, based on dakotaMPIComm

shared function for initializing based on passed MPI_Comm

References Dakota::abort_handler(), MPIManager::dakota_mpi_comm(), ParallelLibrary::increment_parallel_configuration(), ParallelLibrary::mpiManager, MPIManager::mpirun_flag(), ParallelLibrary::outputManager, ParallelLibrary::parallelLevels, ParallelLevel::procsPerServer, ParallelLevel::serverCommRank, ParallelLevel::serverCommSize, ParallelLevel::serverId, ParallelLevel::serverIntraComm, ParallelLevel::serverMasterFlag, ParallelLibrary::startMPITime, OutputManager::startup_message(), MPIManager::world_rank(), and MPIManager::world_size().

Referenced by ParallelLibrary::ParallelLibrary().

**void init_communicators** ( const ParallelLevel & parent_pl, int num_servers, int procs_per_server, int min_procs_per_server, int max_procs_per_server, int max_concurrency, int async_local_concurrency, short default_config, short scheduling_override, bool peer_dynamic_avail ) [private]

split a parent communicator into child server communicators

Split parent communicator into concurrent child server partitions as specified by the passed parameters. This constructs new child intra-communicators and parent-child inter-communicators. This fn is called from Meta-Iterators and NestedModel for the concurrent iterator level and from ApplicationInterface::init_communicators() for the concurrent evaluation and concurrent analysis levels.

References ParallelLibrary::currPCIter, ParallelLevel::dedicatedMasterFlag, ParallelLevel::messagePass, ParallelLevel::numServers, ParallelLibrary::parallelLevels, ParallelLevel::procsPerServer, ParallelLibrary::resolve_inputs(), ParallelLevel::serverCommRank, ParallelLevel::serverCommSize, ParallelLibrary::split_communicator_dedicated_master(), and ParallelLibrary::split_communicator_peer_partition().

Referenced by ParallelLibrary::init_analysis_communicators(), ParallelLibrary::init_evaluation_communicators(), and ParallelLibrary::init_iterator_communicators().

**void resolve_inputs** ( ParallelLevel & child_pl, int avail_procs, int min_procs_per_server, int max_procs_per_server, int max_concurrency, int capacity_multiplier, short default_config, short scheduling_override, bool peer_dynamic_avail, bool print_rank ) [private]

resolve user inputs into a sensible partitioning scheme
This function is responsible for the "auto-configure" intelligence of DAKOTA. It resolves a variety of inputs and overrides into a sensible partitioning configuration for a particular parallelism level. It also handles the general case in which a user's specification request does not divide out evenly with the number of available processors for the level. If num_servers & procs_per_server are both nondefault, then the former takes precedence.

References Dakota::abort_handler(), ParallelLevel::dedicatedMasterFlag, ParallelLevel::numServers, ParallelLevel::procRemainder, and ParallelLevel::procsPerServer.

Referenced by ParallelLibrary::init_communicators().

The documentation for this class was generated from the following files:

- ParallelLibrary.hpp
- ParallelLibrary.cpp

14.182 ParamResponsePair Class Reference

Container class for a variables object, a response object, and an evaluation id.

Public Member Functions

- **ParamResponsePair ()**
  default constructor
- **ParamResponsePair (const Variables &vars, const String &interface_id, const Response &response, bool deep_copy=false)**
  alternate constructor for temporaries
- **ParamResponsePair (const Variables &vars, const String &interface_id, const Response &response, const int eval_id, bool deep_copy=true)**
  standard constructor for history uses
- **ParamResponsePair (const ParamResponsePair &pair)**
  copy constructor
- **~ParamResponsePair ()**
  destructor
- **ParamResponsePair & operator= (const ParamResponsePair &pair)**
  assignment operator
- **void read (std::istream &s)**
  read a ParamResponsePair object from an std::istream
- **void write (std::ostream &s) const**
  write a ParamResponsePair object to an std::ostream
- **void read_annotated (std::istream &s)**
  read a ParamResponsePair object in annotated format from an std::istream
- **void write_annotated (std::ostream &s) const**
  write a ParamResponsePair object in annotated format to an std::ostream
- **void write_tabular (std::ostream &s, unsigned short tabular_format) const**
  write a ParamResponsePair object in tabular format (all variables active/inactive) to an std::ostream
- **void write_tabular_labels (std::ostream &s, unsigned short tabular_format) const**
  write PRP labels in tabular format to an std::ostream
- **void read (MPIUnpackBuffer &s)**
read a ParamResponsePair object from a packed MPI buffer

- void write (MPIPackBuffer &s) const
  write a ParamResponsePair object to a packed MPI buffer

- int eval_id () const
  return the evaluation identifier

- void eval_id (int id)
  set the evaluation identifier

- const String & interface_id () const
  return the interface identifier from evalInterfaceIds

- void interface_id (const String &id)
  set the interface identifier within evalInterfaceIds

- const IntStringPair & evalInterfaceIds () const
  return the aggregate eval/interface identifier from the response object

- const Variables & variables () const
  return the parameters object

- Variables & variables ()
  return the parameters object

- void variables (const Variables &vars)
  set the parameters object

- const Response & response () const
  return the response object

- Response & response ()
  return the response object

- void response (const Response &resp)
  set the response object

- const ActiveSet & active_set () const
  return the active set object from the response object

- void active_set (const ActiveSet &set)
  set the active set object within the response object

Private Member Functions

- template<class Archive >
  void serialize (Archive &ar, const unsigned int version)
  serialize the PRP: write and read are symmetric for this class

Private Attributes

- Variables prpVariables
  the set of parameters for the function evaluation

- Response prpResponse
  the response set for the function evaluation

- IntStringPair evalInterfaceIds
  the evalInterfaceIds aggregate
CHAPTER 14. CLASS DOCUMENTATION

Friends

- class boost::serialization::access
  allow boost access to serialize this class
- bool operator==(const ParamResponsePair &pair1, const ParamResponsePair &pair2)
  equality operator
- bool operator!=(const ParamResponsePair &pair1, const ParamResponsePair &pair2)
  inequality operator

14.182.1 Detailed Description

Container class for a variables object, a response object, and an evaluation id. 

ParamResponsePair provides a container class for association of the input for a particular function evaluation (a variables object) with the output from this function evaluation (a response object), along with an evaluation identifier. This container defines the basic unit used in the data_pairs cache, in restart file operations, and in a variety of scheduling algorithm queues. With the advent of STL, replacement of arrays of this class with map<> and pair<>() template constructs may be possible (using map<pair<int,String>,pair<Variables,Response> >, for example), assuming that deep copies, I/O, alternate constructors, etc., can be adequately addressed. Boost tuple<> may also be a candidate.

14.182.2 Constructor & Destructor Documentation

ParamResponsePair ( const Variables & vars, const String & interface_id, const Response & response, bool deep_copy = false ) [inline]
alternate constructor for temporaries

Uses of this constructor often employ the standard Variables and Response copy constructors to share representations since this constructor is commonly used for search_pairs (which are local instantiations that go out of scope prior to any changes to values; i.e., they are not used for history).

ParamResponsePair ( const Variables & vars, const String & interface_id, const Response & response, const int eval_id, bool deep_copy = true ) [inline]
standard constructor for history uses

Uses of this constructor often do not share representations since deep copies are used when history mechanisms (e.g., data_pairs and beforeSynchCorePRPQueue) are involved.

14.182.3 Member Function Documentation

void read ( MPIUnpackBuffer & s ) [inline]
read a ParamResponsePair object from a packed MPI buffer

interfaceId is omitted since master processor retains interface ids and communicates asv and response data only with slaves.

References ParamResponsePair::evalInterfaceIds, ParamResponsePair::prpResponse, and ParamResponsePair::prpVariables.
void write ( MPIPackBuffer & s ) const  [inline]
write a ParamResponsePair object to a packed MPI buffer
    interfaceId is omitted since master processor retains interface ids and communicates asv and response data
only with slaves.
    References ParamResponsePair::evalInterfaceIds, ParamResponsePair::prpResponse, and ParamResponsePair-
::prpVariables.

14.182.4  Member Data Documentation

IntStringPair evalInterfaceIds  [private]
the evalInterfaceIds aggregate
    the function evaluation identifier (assigned from Interface::evalIdCntr) is paired with the interface used to
generate the response object. Used in PRPCache id_vars_set_compare to prevent duplicate detection on results
from different interfaces.  evalInterfaceIds belongs here rather than in Response since some Response objects
involve consolidation of several fn evals (e.g., Model::synchronize_derivatives()) that are not, in total, generated
by a single interface. The prPair, on the other hand, is used for storage of all low level fn evals that get evaluated
in ApplicationInterface::map().
    Referenced by ParamResponsePair::eval_id(), ParamResponsePair::eval_interface_ids(), ParamResponsePair-
::interface_id(), ParamResponsePair::operator=(), Dakota::operator==(), ParamResponsePair::read(), and Param-
ResponsePair::write().
    The documentation for this class was generated from the following file:
    • ParamResponsePair.hpp

14.183  ParamStudy Class Reference

Class for vector, list, centered, and multidimensional parameter studies.
    Inheritance diagram for ParamStudy:

```
    Iterator
    
    Analyzer
    
    PStudyDACE
    
    ParamStudy
```

Public Member Functions

- ParamStudy (ProblemDescDB &problem_db, Model &model)
  constructor
- ~ParamStudy ()
  destructor
- bool resize ()
reinitializes iterator based on new variable size

- **void pre_run()**
  pre-run portion of run (optional); re-implemented by Iterators which can generate all Variables (parameter sets) a priori

- **void core_run()**
  core portion of run; implemented by all derived classes and may include pre/post steps in lieu of separate pre/post

- **void post_input()**
  read tabular data for post-run mode

- **void post_run(std::ostream &s)**
  post-run portion of run (optional); verbose to print results; re-implemented by Iterators that can read all Variables/Responses and perform final analysis phase in a standalone way

- **void archive_model_variables(const Model &, size_t idx) const override**
  Archive variables for parameter set idx.

- **void archive_model_response(const Response &, size_t idx) const override**
  Archive responses for parameter set idx.

### Protected Member Functions

- **void archive_allocate_sets() const**
  Allocate space to archive parameters and responses.

### Private Member Functions

- **void sample()**
  performs the parameter study by sampling from a list of points

- **void vector_loop()**
  performs the parameter study by sampling along a vector, starting from an initial point followed by numSteps increments along continuous/discrete step vectors

- **void centered_loop()**
  performs a number of plus and minus offsets for each parameter centered about an initial point

- **void multidim_loop()**
  performs a full factorial combination for all intersections defined by a set of multidimensional partitions

- **bool load_distribute_points(const String &points_filename, unsigned short tabular_format, bool active_only)**
  load list of points from data file and distribute among listCVPoints, listDIVPoints, listDSVPoints, and listDRVPoints

- **template<typename OrdinalType, typename ScalarTypeA, typename ScalarTypeC, typename ScalarTypeDI, typename ScalarTypeDS, typename ScalarTypeDR> bool distribute(const Teuchos::SerialDenseVector<OrdinalType, ScalarTypeA>& all_data, Teuchos::SerialDenseVector<OrdinalType, ScalarTypeC>& c_data, Teuchos::SerialDenseVector<OrdinalType, ScalarTypeDI>& di_data, Teuchos::SerialDenseVector<OrdinalType, ScalarTypeDS>& ds_data, Teuchos::SerialDenseVector<OrdinalType, ScalarTypeDR>& dr_data)**
  distributes incoming all vector in standard variable ordering among continuous, discrete int, discrete string, and discrete real vectors

- **template<typename ScalarType> bool distribute(const std::vector<ScalarType>& all_data, std::vector<ScalarType>& c_data, std::vector<ScalarType>& di_data, std::vector<ScalarType>& ds_data, std::vector<ScalarType>& dr_data)**
distributes incoming all array in standard variable ordering among continuous, discrete int, discrete string, and
discrete real arrays

- **bool distribute_list_of_points** (const RealVector &list_of_pts)
  distributes list_of_pts coming from user spec among listCVPoints, listDIVPoints, listDSVPoints, and listDRVPoints

- **void final_point_to_step_vector** ()
  compute step vectors from finalPoint, initial points, and numSteps

- **void distribute_partitions** ()
  compute step vectors from \{cont,discInt,discString,discReal\}VarPartitions and global bounds

- **bool check_num_steps** (int num_steps)
  perform error checks on numSteps

- **bool check_step_vector** (const RealVector &step_vector)
  perform error checks on numSteps

- **bool check_final_point** (const RealVector &final_pt)
  perform error checks on finalPoint

- **bool check_steps_per_variable** (const IntVector &steps_per_var)
  perform error checks on stepsPerVariable

- **bool check_variable_partitions** (const UShortArray &partitions)
  perform error checks on variable partitions

- **bool check_finite_bounds** ()
  check for finite variable bounds within iteratedModel, as required for computing partitions of finite ranges

- **bool check_ranges_sets** (int num_steps)
  sanity check for vector parameter study

- **bool check_ranges_sets** (const IntVector &c_steps, const IntVector &di_steps, const IntVector &ds_steps, const IntVector &dr_steps)
  sanity check for centered parameter study

- **bool check_sets** (const IntVector &c_steps, const IntVector &di_steps, const IntVector &ds_steps, const IntVector &dr_steps)
  sanity check for increments along int/real set dimensions

- **int integer_step** (int range, int num_steps) const
  check for integer remainder and return step

- **int index_step** (size_t start, size_t end, int num_steps) const
  check for out of bounds and index remainder and return step

- **void c_step** (size_t c_index, int increment, Variables &vars)
  helper function for performing a continuous step in one variable

- **void dri_step** (size_t di_index, int increment, Variables &vars)
  helper function for performing a discrete step in an integer range variable

- **void dsi_step** (size_t di_index, int increment, const IntSet &values, Variables &vars)
  helper function for performing a discrete step in an integer set variable

- **void dss_step** (size_t ds_index, int increment, const StringSet &values, Variables &vars)
  helper function for performing a discrete step in a string set variable

- **void dsr_step** (size_t dr_index, int increment, const RealSet &values, Variables &vars)
  helper function for performing a discrete step in a real set variable

- **void reset** (Variables &vars)
reset vars to initial point (center)
• void centered_header (const String &type, size_t var_index, int step, size_t hdr_index)
  store a centered parameter study header within allHeaders
• void archive_allocate_cps () const
  specialized per-variable slice output for centered param study
• void archive_cps_vars (const Model &model, size_t idx) const
  specialized per-variable slice output for centered param study
• void archive_cps_resp (const Response &response, size_t idx) const
  specialized per-variable slice output for centered param study
• void index_to_var_step (const size_t study_idx, size_t &var_idx, size_t &step_idx) const
  map an overall parameter study (zero-based) evaluation index to the (zero-based) variable index (among all vari-
  ables) and the (zero-based) step index within that variable

Private Attributes
• size_t numEvals  
total number of parameter study evaluations computed from specification
• RealVectorArray listCVPoints  
array of continuous evaluation points for the list_parameter_study
• IntVectorArray listDIVPoints  
array of discrete int evaluation points for the list_parameter_study
• StringMulti2DArray listDSVPoints  
array of discrete string evaluation points for the list_parameter_study
• RealVectorArray listDRVPoints  
array of discrete real evaluation points for the list_parameter_study
• RealVector initialCVPoint  
the continuous start point for vector and centered parameter studies
• IntVector initialDIVPoint  
the discrete int start point for vector and centered parameter studies
• StringMultiArray initialDSVPoint  
the discrete string start point for vector and centered parameter studies
• RealVector initialDRVPoint  
the discrete real start point for vector and centered parameter studies
• RealVector finalCVPoint  
the continuous ending point for vector_parameter_study
• IntVector finalDIVPoint  
the discrete int range value or set index ending point for vector_parameter_study
• IntVector finalDSVPoint  
the discrete string set index ending point for vector_parameter_study
• IntVector finalDRVPoint  
the discrete real set index ending point for vector_parameter_study
• RealVector contStepVector  
the n-dimensional continuous increment
• IntVector discIntStepVector
The n-dimensional discrete integer range value or set index increment

- IntVector `discStringStepVector`
  The n-dimensional discrete string set index increment

- IntVector `discRealStepVector`
  The n-dimensional discrete real set index increment

- int `numSteps`
  The number of times continuous/discrete step vectors are applied for vector parameter study (a specification option)

- IntVector `stepsPerVariable`
  Number of offsets in the plus and the minus direction for each variable in a centered parameter study

- IntVector `contStepsPerVariable`
  Number of offsets in the plus and the minus direction for each continuous variable in a centered parameter study

- IntVector `discIntStepsPerVariable`
  Number of offsets in the plus and the minus direction for each discrete integer variable in a centered parameter study

- IntVector `discStringStepsPerVariable`
  Number of offsets in the plus and the minus direction for each discrete string variable in a centered parameter study

- IntVector `discRealStepsPerVariable`
  Number of offsets in the plus and the minus direction for each discrete real variable in a centered parameter study

- UShortArray `contVarPartitions`
  Number of partitions for each continuous variable in a multidim parameter study

- UShortArray `discIntVarPartitions`
  Number of partitions for each discrete integer variable in a multidim parameter study

- UShortArray `discStringVarPartitions`
  Number of partitions for each discrete string variable in a multidim parameter study

- UShortArray `discRealVarPartitions`
  Number of partitions for each discrete real variable in a multidim parameter study

Additional Inherited Members

14.183.1 Detailed Description

Class for vector, list, centered, and multidimensional parameter studies.

The ParamStudy class contains several algorithms for performing parameter studies of different types. The vector parameter study steps along an n-dimensional vector from an arbitrary initial point to an arbitrary final point in a specified number of steps. The centered parameter study performs a number of plus and minus offsets in each coordinate direction around a center point. A multidimensional parameter study fills an n-dimensional hypercube based on bounds and a specified number of partitions for each dimension. And the list parameter study provides for a user specification of a list of points to evaluate, which allows general parameter investigations not fitting the structure of vector, centered, or multidim parameter studies.

14.183.2 Member Function Documentation

`void pre_run ( ) [virtual]`

Pre-run portion of run (optional); re-implemented by Iterators which can generate all Variables (parameter sets) a priori
pre-run phase, which a derived iterator may optionally reimplement; when not present, pre-run is likely integrated into the derived run function. This is a virtual function; when re-implementing, a derived class must call its nearest parent’s \texttt{pre\_run()}, if implemented, typically before performing its own implementation steps.

Reimplemented from \texttt{Analyzer}.

References Dakota::abort\_handler(), SharedVariablesData::active\_components\_totals(), Analyzer::allHeaders, Analyzer::allVariables, ParamStudy::centered\_loop(), Variables::continuous\_variables(), ParamStudy::contSteps\_Per\_Variable, ParamStudy::contStepVector, ParamStudy::contVarPartitions, Dakota::copy\_data(), Model::current\_variables(), ParamStudy::discIntSteps\_Per\_Variable, ParamStudy::discIntStepVector, ParamStudy::discIntVarPartitions, ParamStudy::discRealSteps\_Per\_Variable, ParamStudy::discRealStepVector, ParamStudy::discRealVarPartitions, Variables::discrete\_int\_variables(), Variables::discrete\_real\_variables(), Variables::discrete\_string\_variables(), ParamStudy::discStringSteps\_Per\_Variable, ParamStudy::discStringStepVector, ParamStudy::discStringVarPartitions, ParamStudy::dist\_steps\_Per\_Variable, ParamStudy::distVar\_Per\_Variable, ParamStudy::distStep\_Vector, ParamStudy::discReal\_Var\_Partitions, ParamStudy::dist\_steps\_Per\_Variable, ParamStudy::discInt\_steps\_Per\_Variable, ParamStudy::discInt\_steps\_Vector, ParamStudy::discInt\_Var\_Partitions, ParamStudy::disc\_steps\_Per\_Variable, ParamStudy::disc\_steps\_Vector, ParamStudy::disc\_Var\_Partitions, Variables::discrete\_int\_variables(), Variables::discrete\_real\_variables(), Variables::discrete\_string\_variables(), ParamStudy::disc\_string\_steps\_Per\_Variable, ParamStudy::disc\_string\_steps\_Vector, ParamStudy::disc\_string\_Var\_Partitions, ParamStudy::disc\_string\_Partitions(), ParamStudy::final\_point\_to\_step\_vector(), ParamStudy::final\_CVPoint, ParamStudy::final\_DIVPoint, ParamStudy::final\_DRVPoint, ParamStudy::final\_DSVPoint, ParamStudy::initial\_CVPoint, ParamStudy::initial\_DIVPoint, ParamStudy::initial\_DRVPoint, ParamStudy::initial\_DSVPoint, Iterator::iteratedModel, Iterator::method\_enum\_to\_string(), Iterator::methodName, ParamStudy::multidim\_loop(), Analyzer::num\_Discrete\_String\_Vars, ParamStudy::num\_Evals, ParamStudy::num\_Steps, Iterator::output\_Level, Analyzer::pre\_run(), ParamStudy::sample(), Variables::shared\_data(), Dakota::svd(), ParamStudy::vector\_loop(), and Dakota::write\_ordered().

\begin{verbatim}

\textbf{void core\_run( )} [virtual]

core portion of run; implemented by all derived classes and may include pre/post steps in lieu of separate pre/post

Virtual run function for the iterator class hierarchy. All derived classes need to redefine it.

Reimplemented from \texttt{Iterator}.

References ParamStudy::archive\_allocate\_sets(), Analyzer::evaluate\_parameter\_sets(), Iterator::iteratedModel, Iterator::methodName, Analyzer::num\_LSq\_Terms, Analyzer::num\_Obj\_Fns, and Iterator::subIterator\_Flag.

\textbf{void post\_run ( std::ostream & s )} [virtual]

post-run portion of run (optional); verbose to print results; re-implemented by Iterators that can read all Variables-/Responses and perform final analysis phase in a standalone way

Post-run phase, which a derived iterator may optionally reimplement; when not present, post-run is likely integrated into run. This is a virtual function; when re-implementing, a derived class must call its nearest parent’s \texttt{post\_run()}, typically after performing its own implementation steps.

Reimplemented from \texttt{Analyzer}.

References ResultsManager::active(), Analyzer::all\_Responses, Analyzer::all\_Variables, SensAnalysisGlobal::archive\_correlations(), SensAnalysisGlobal::compute\_correlations(), Model::continuous\_variable\_labels(), Model::discrete\_int\_variable\_labels(), Model::discrete\_real\_variable\_labels(), Model::discrete\_set\_string\_values(), Model::discrete\_string\_variable\_labels(), Iterator::iteratedModel, Iterator::methodName, Analyzer::post\_run(), PStudy\_DACE::p\_Study\_DACESens\_Global, Model::response\_labels(), Iterator::results\_DB, Iterator::run\_identifier(), and Iterator::subIterator\_Flag.

\textbf{bool load\_distribute\_points ( const String & points\_filename, unsigned short tabular\_format, bool active\_only )} [private]

load list of points from data file and distribute among list\_CVPoints, list\_DIVPoints, list\_DSVPoints, and list\_DRVPoints

Load from file and distribute points; using this function to manage construction of the temporary arrays. Historically all data was read as a real (mixture of values and indices), but now points\_file is valued-based (reals, integers, strings) so file input matches tabular data output. Return false on success.
\end{verbatim}
14.184. **PARTIAL_PRP_EQUALITY STRUCT REFERENCE**
819

References Dakota::NPOS, Model::continuous_lower_bounds(), Model::continuous_upper_bounds(), Variables::copy(), Model::current_variables(), Model::discrete_int_lower_bounds(), Model::discrete_int_sets(), Model::discrete_int_upper_bounds(), Model::discrete_set_int_values(), Model::discrete_set_real_values(), Model::discrete_set_string_values(), Iterator::iteratedModel, ParamStudy::listCVPoints, ParamStudy::listDIVPoints, ParamStudy::listDRVPoints, ParamStudy::listDSVPoints, Analyzer::numContinuousVars, Analyzer::numDiscreteIntVars, Analyzer::numDiscreteRealVars, Analyzer::numDiscreteStringVars, ParamStudy::numEvals, and Dakota::set_value_to_index().

Referenced by ParamStudy::ParamStudy().

```cpp
bool distribute_list_of_points ( const RealVector &list_of_pts ) [private]
```
distributes list_of_pts coming from user spec among listCVPoints, listDIVPoints, listDSVPoints, and listDRVPoints

Parse list of points into typed data containers; list_of_pts will contain values for continuous and discrete integer range, but indices for all discrete set types (int, string, real)

References Dakota::abort_handler(), Model::discrete_int_sets(), Model::discrete_int_variable_labels(), Model::discrete_real_variable_labels(), Model::discrete_set_int_values(), Model::discrete_set_real_values(), Model::discrete_set_string_values(), Model::discrete_string_variable_labels(), ParamStudy::distribute(), Iterator::iteratedModel, ParamStudy::listCVPoints, ParamStudy::listDIVPoints, ParamStudy::listDRVPoints, ParamStudy::listDSVPoints, Iterator::method_id(), Analyzer::numContinuousVars, Analyzer::numDiscreteIntVars, Analyzer::numDiscreteRealVars, Analyzer::numDiscreteStringVars, ParamStudy::numEvals, and Dakota::set_index_to_value().

Referenced by ParamStudy::ParamStudy().

14.183.3 **Member Data Documentation**

```cpp
IntVector stepsPerVariable [private]
```
number of offsets in the plus and the minus direction for each variable in a centered parameter study

The per-type step arrays below could be made views into this, instead of duplicating, but if so, distribute() will not be allowed to resize the individual vectors.

Referenced by ParamStudy::check_steps_per_variable(), and ParamStudy::index_to_var_step().

The documentation for this class was generated from the following files:

- ParamStudy.hpp
- ParamStudy.cpp

14.184 **partial_prp_equality Struct Reference**
predicate for comparing ONLY the interfaceId and Vars attributes of PRPair

**Public Member Functions**

- bool operator() (const ParamResponsePair &database_pr, const ParamResponsePair &search_pr) const

14.184.1 **Detailed Description**
predicate for comparing ONLY the interfaceId and Vars attributes of PRPair

The documentation for this struct was generated from the following file:

- PRPMultiIndex.hpp
14.185 partial_prp_hash Struct Reference

wrapper to delegate to the ParamResponsePair hash_value function

Public Member Functions

- std::size_t operator() (const ParamResponsePair &prp) const

14.185.1 Detailed Description

wrapper to delegate to the ParamResponsePair hash_value function

The documentation for this struct was generated from the following file:

- PRPMultiIndex.hpp

14.186 PebbldBranching Class Reference

Main Branching class for the PEBBL-based Minimizer.

Inherits branching.

Public Member Functions

- PebbldBranching ()
  Default Constructor.
- ~PebbldBranching ()
  Destructor.
- pebbl::branchSub ∗ blankSub ()
  Method that returns an empty Sub-Branch.
- void setModel (Model &model)
- void setIterator (Iterator &iterator)

Protected Attributes

- Model parentModel
  Original model, before branching.
- Iterator nlpSolver
  Solver to be used at root node.
- RealVector cont_vars
  Initial variable values for root node.
- RealVector lower_bounds
  Lower bounds for root node.
- RealVector upper_bounds
  Upper bounds for root node.
Friends
• class PebbldBranchSub

14.186.1 Detailed Description
Main Branching class for the PEBBL-based Minimizer.
The documentation for this class was generated from the following files:
• PEBBLBranching.hpp
• PEBBLBranching.cpp

14.187 PebbldBranchSub Class Reference
Sub Branch class for the PEBBL-based Minimizer.
Inherits branchSub.

Public Member Functions
• PebbldBranchSub ()
  Constructor.
• ~PebbldBranchSub ()
  Destructor.
• PebbldBranching ∗ global () const
  Returns a reference to the corresponding main Branching object.
• pebbl::branching ∗ bGlobal () const
  Returns a reference to the corresponding main Branching object.
• void setGlobalInfo (PebbldBranching ∗ global_)
  Method that sets up the main Branching object.
• void setRootComputation ()
  Method that is called when declaring the current node as a root node.
• void boundComputation (double ∗ controlParam)
  Method that does the Bounding Operation. In other words, it calls the optimization algorithm on the relaxed domain.
• bool candidateSolution ()
  Method called after the bounding operation, returns true if the bounding resulted in a possible solution to the original non-relaxed problem.
• pebbl::solution ∗ extractSolution ()
  Method that returns a PEBBL-based solution.
• int splitComputation ()
  Method that determines how many branches are created and how. Returns the number of branches created from this object.
• pebbl::branchSub ∗ makeChild (int whichChild)
  Method that returns a new PebbldBranchSub object based on Objective Function improvements and the number of branches.
• void pebbldSubAsChildOf (PebbldBranchSub ∗ parent, int splitVar, int whichChild, std::vector< double > ∗ _candidateX, RealVector ∗ lower_bounds, RealVector ∗ upper_bounds)
  Method that creates a new PebbldBranching object.
Protected Attributes

- `PebbldBranching * globalPtr
  Pointer referring to all info passed to subproblem.
- `Model subModel
  Model used for sub-problem.
- `Iterator subNLPSolver
  Solver used for sub-problems.
- `std::vector<double> candidate
  Candidate solution after solving sub-problem (also the bound)
- `double candidate_objFn
  Objective value at the candidate solution.
- `int splitVar
  Variable to branch on.
- `RealVector cont_vars
  Initial variable values for sub-problem.
- `RealVector lower_bounds
  Lower bounds for sub-problem.
- `RealVector upper_bounds
  Upper bounds for sub-problem.

Friends

- class `PebbldBranching

14.187.1 Detailed Description

Sub Branch class for the PEBBL-based Minimizer.
The documentation for this class was generated from the following files:

- `PEBBLBranching.hpp
- `PEBBLBranching.cpp

14.188 PebbldTraits Class Reference

Wrapper class for experimental PebbldMinimizer.
Inheritance diagram for PebbldTraits:

```
TraitsBase
     |
     V
PebbldTraits
```
Public Member Functions

- **PebbldTraits ()**
  
  *default constructor*

- **virtual ~PebbldTraits ()**
  
  *destructor*

- **virtual bool is_derived ()**
  
  *A temporary query used in the refactor.*

- **bool supports_continuous_variables ()**
  
  *Return the flag indicating whether method supports continuous variables.*

- **bool supports_discrete_variables ()**
  
  *Return the flag indicating whether method supports discrete variables.*

- **bool supports_nonlinear_equality ()**
  
  *Return the flag indicating whether method supports nonlinear equalities.*

- **bool supports_nonlinear_inequality ()**
  
  *Return the flag indicating whether method supports nonlinear inequalities.*

### 14.188.1 Detailed Description

Wrapper class for experimental PebbldMinimizer.

A version of TraitsBase specialized for Pebbld mptimizers

The documentation for this class was generated from the following file:

- PEBBLMinimizer.hpp

### 14.189 PecosApproximation Class Reference

Derived approximation class for global basis polynomials.

Inheritance diagram for PecosApproximation:

```
Approximation

PecosApproximation
```

Public Member Functions

- **PecosApproximation ()**
  
  *default constructor*

- **PecosApproximation (ProblemDescDB &problem_db, const SharedApproxData &shared_data, const String &approx_label)**
  
  *standard ProblemDescDB-driven constructor*

- **PecosApproximation (const SharedApproxData &shared_data)**
  
  *alternate constructor*

- **~PecosApproximation ()**
• void destructor

• void expansion_coefficient_flag (bool coeff_flag)
  
  set pecosBasisApprox.configOptions.expansionCoeffFlag

• bool expansion_coefficient_flag () const
  
  get pecosBasisApprox.configOptions.expansionCoeffFlag

• void expansion_gradient_flag (bool grad_flag)
  
  set pecosBasisApprox.configOptions.expansionGradFlag

• bool expansion_gradient_flag () const
  
  get pecosBasisApprox.configOptions.expansionGradFlag

• void clear_component_effects ()
  
  clear unused Sobol' indices

• void compute_component_effects ()
  
  Performs global sensitivity analysis using Sobol' indices by computing component (main and interaction) effects.

• void compute_total_effects ()
  
  Performs global sensitivity analysis using Sobol' indices by computing total effects.

• const Pecos::RealVector & sobol_indices () const
  
  return polyApproxRep->sobolIndices

• const Pecos::RealVector & total_sobol_indices () const
  
  return polyApproxRep->totalSobolIndices

• size_t sparsity () const
  
  return the number of non-zero coefficients for this QoI

• Pecos::ULongULongMap sparse_sobol_index_map () const
  
  return RegressOrthogPolyApproximation::sparseSobolIndexMap

• const Pecos::RealVector & dimension_decay_rates () const
  
  return OrthogPolyApproximation::decayRates

• void allocate_arrays ()
  
  invoke Pecos::PolynomialApproximation::allocate_arrays()

• void initialize_covariance (Approximation &approx_2)
  
  initialize covariance accumulators with pointers to other QoI

• void clear_covariance_pointers ()
  
  clear covariance pointers to other QoI

• void initialize_products ()
  
  initialize covariance accumulators (also reinitialize after change in stats type)

• bool product_interpolants ()
  
  query whether product interpolants are defined (non-empty)

• Real mean ()
  
  return the mean of the expansion, where all active variables are random

• Real mean (const Pecos::RealVector &x)
  
  return the mean of the expansion for a given parameter vector, where a subset of the active variables are random

• Real combined_mean ()
  
  return the mean of the combined expansion, treating all variables as random

• Real combined_mean (const Pecos::RealVector &x)
return the mean of the combined expansion for a given parameter vector, where a subset of the active variables are treated as random

- const Pecos::RealVector & mean_gradient ()
  return the gradient of the expansion mean for a given parameter vector, where all active variables are random

- const Pecos::RealVector & mean_gradient (const Pecos::RealVector &x, const Pecos::SizetArray &dvv)
  return the gradient of the expansion mean for a given parameter vector and given DVV, where a subset of the active variables are random

- Real variance ()
  return the variance of the expansion, where all active vars are random

- Real variance (const Pecos::RealVector &x)
  return the variance of the expansion for a given parameter vector, where a subset of the active variables are random

- const Pecos::RealVector & variance_gradient ()
  return the gradient of the expansion variance for a given parameter vector, where all active variables are random

- const Pecos::RealVector & variance_gradient (const Pecos::RealVector &x, const Pecos::SizetArray &dvv)
  return the gradient of the expansion variance for a given parameter vector and given DVV, where a subset of the active variables are random

- Real covariance (Approximation &approx)
  return the covariance between two response expansions, treating all variables as random

- Real covariance (const Pecos::RealVector &x, Approximation &approx)
  return the covariance between two response expansions, treating a subset of the variables as random

- Real combined_covariance (Approximation &approx)
  return the covariance between two combined response expansions, where all active variables are random

- Real combined_covariance (const Pecos::RealVector &x, Approximation &approx)
  return the covariance between two combined response expansions, where a subset of the active variables are random

- Real beta (bool cdf_flag, Real z_bar)
  return the reliability index (mapped from z_bar), where all active variables are random

- Real beta (const RealVector &x, bool cdf_flag, Real z_bar)
  return the reliability index (mapped from z_bar), treating a subset of variables as random

- Real combined_beta (bool cdf_flag, Real z_bar)
  return the reliability index (mapped from z_bar), where all active variables are random

- Real combined_beta (const RealVector &x, bool cdf_flag, Real z_bar)
  return the reliability index (mapped from z_bar), treating a subset of variables as random

- Real delta_mean ()
  return the change in mean resulting from expansion refinement, where all active variables are random

- Real delta_mean (const RealVector &x)
  return the change in mean resulting from expansion refinement, treating a subset of variables as random

- Real delta_combined_mean ()
  return the change in mean resulting from combined expansion refinement, where all active variables are random

- Real delta_combined_mean (const RealVector &x)
  return the change in mean resulting from combined expansion refinement, treating a subset of variables as random

- Real delta_std_deviation ()
  return the change in standard deviation resulting from expansion refinement, where all active variables are random

- Real delta_std_deviation (const RealVector &x)
return the change in standard deviation resulting from expansion refinement, treating a subset of variables as random

- Real \texttt{delta\_combined\_std\_deviation} ()
  return the change in standard deviation resulting from combined expansion refinement, where all active variables are random

- Real \texttt{delta\_combined\_std\_deviation} (const RealVector &x)
  return the change in standard deviation resulting from combined expansion refinement, treating a subset of variables as random

- Real \texttt{delta\_variance} ()
  return the change in variance resulting from expansion refinement, where all active variables are random

- Real \texttt{delta\_variance} (const RealVector &x)
  return the change in variance resulting from expansion refinement, treating a subset of variables as random

- Real \texttt{delta\_combined\_variance} ()
  return the change in variance resulting from combined expansion refinement, where all active variables are random

- Real \texttt{delta\_combined\_variance} (const RealVector &x)
  return the change in variance resulting from combined expansion refinement, treating a subset of variables as random

- Real \texttt{delta\_covariance} (Approximation &approx\_2)
  return the change in covariance resulting from expansion refinement, where all active variables are random

- Real \texttt{delta\_covariance} (const Pecos::RealVector &x, Approximation &approx\_2)
  return the change in covariance resulting from expansion refinement, where a subset of the active variables are random

- Real \texttt{delta\_combined\_covariance} (Approximation &approx\_2)
  return the change in covariance resulting from expansion refinement, where all active variables are random

- Real \texttt{delta\_combined\_covariance} (const Pecos::RealVector &x, Approximation &approx\_2)
  return the change in covariance resulting from expansion refinement, where a subset of the active variables are random

- Real \texttt{delta\_beta} (bool cdf\_flag, Real z\_bar)
  return the change in reliability index (mapped from z\_bar) resulting from expansion refinement, where all active variables are random

- Real \texttt{delta\_beta} (const RealVector &x, bool cdf\_flag, Real z\_bar)
  return the change in reliability index (mapped from z\_bar) resulting from expansion refinement, treating a subset of variables as random

- Real \texttt{delta\_combined\_beta} (bool cdf\_flag, Real z\_bar)
  return the change in reliability index (mapped from z\_bar) resulting from expansion refinement, where all active variables are random

- Real \texttt{delta\_combined\_beta} (const RealVector &x, bool cdf\_flag, Real z\_bar)
  return the change in reliability index (mapped from z\_bar) resulting from expansion refinement, treating a subset of variables as random

- Real \texttt{delta\_z} (bool cdf\_flag, Real beta\_bar)
  return the change in response level (mapped from beta\_bar) resulting from expansion refinement, where all active variables are random

- Real \texttt{delta\_z} (const RealVector &x, bool cdf\_flag, Real beta\_bar)
  return the change in response level (mapped from beta\_bar) resulting from expansion refinement, where a subset of the active variables are random
• Real \texttt{delta\_combined\_z} (bool cdf\_flag, Real beta\_bar)
  return the change in response level (mapped from beta\_bar) resulting from expansion refinement, where all active variables are random

• Real \texttt{delta\_combined\_z} (const RealVector &x, bool cdf\_flag, Real beta\_bar)
  return the change in response level (mapped from beta\_bar) resulting from expansion refinement, where a subset of the active variables are random

• void \texttt{compute\_moments} (bool full\_stats=true, bool combined\_stats=false)
  compute moments up to the order supported by the Pecos polynomial approximation

• void \texttt{compute\_moments} (const Pecos::RealVector &x, bool full\_stats=true, bool combined\_stats=false)
  compute moments in all-variables mode up to the order supported by the Pecos polynomial approximation

• const RealVector & \texttt{moments} () const
  return primary moments using Pecos::PolynomialApproximation::moments()

• const RealVector & \texttt{expansion\_moments} () const
  return expansion moments from Pecos::PolynomialApproximation

• const RealVector & \texttt{numerical\_integration\_moments} () const
  return numerical moments from Pecos::PolynomialApproximation

• const RealVector & \texttt{combined\_moments} () const
  return combined moments from multilevel-multifidelity expansion roll-up

• Real \texttt{moment} (size\_t i) const
  return primary moment using Pecos::PolynomialApproximation::moment(i)

• void \texttt{moment} (Real mom, size\_t i)
  set primary moment using Pecos::PolynomialApproximation::moment(i)

• Real \texttt{combined\_moment} (size\_t i) const
  return Pecos::PolynomialApproximation::combinedMoments[i]

• void \texttt{combined\_moment} (Real mom, size\_t i)
  set Pecos::PolynomialApproximation::combinedMoments[i]

• void \texttt{clear\_computed\_bits} ()
  clear tracking of computed moments, due to a change that invalidates previous results

• void \texttt{build\_linear\_system} (RealMatrix &A, const UShort2DArray &multi\_index)
  construct the Vandermonde matrix "A" for PCE regression for Ax = b

• void \texttt{augment\_linear\_system} (const RealVectorArray &samples, RealMatrix &A, const UShort2DArray &multi\_index)

• Pecos::BasisApproximation & \texttt{pecos\_basis\_approximation} ()
  return pecos\_BasisApprox

Protected Member Functions

• Real \texttt{value} (const Variables &\texttt{vars})
  retrieve the approximate function value for a given parameter vector

• const Pecos::RealVector & \texttt{gradient} (const Variables &\texttt{vars})
  retrieve the approximate function gradient for a given parameter vector

• const Pecos::RealSymMatrix & \texttt{hessian} (const Variables &\texttt{vars})
  retrieve the approximate function Hessian for a given parameter vector

• int \texttt{min\_coefficients} () const
return the minimum number of samples (unknowns) required to build the derived class approximation type in num-
Vars dimensions

- void build ()
  builds the approximation from scratch
- void rebuild ()
  rebuilds the approximation incrementally
- void pop_coefficients (bool save_data)
  removes entries from end of SurrogateData::\{vars,resp\}Data (last points appended, or as specified in args)
- void push_coefficients ()
  restores state prior to previous pop()
- void finalize_coefficients ()
  finalize approximation by applying all remaining trial sets
- void combine_coefficients ()
  combine all level approximations into a single aggregate approximation
- void combined_to_active_coefficients (bool clear_combined=true)
  promote combined approximation into active approximation
- void clear_inactive_coefficients ()
  prune inactive coefficients following combination and promotion to active
- void print_coefficients (std::ostream &s, bool normalized)
  print the coefficient array computed in build()/rebuild()

- RealVector approximation_coefficients (bool normalized) const
  return expansion coefficients in a form consistent with the shared multi-index
- void approximation_coefficients (const RealVector &approx_coeffs, bool normalized)
  set expansion coefficients in a form consistent with the shared multi-index
- void coefficient_labels (std::vector< std::string > &coeff_labels) const
  print the coefficient array computed in build()/rebuild()

Private Member Functions

- void approx_type_to_basis_type (const String &approx_type, short &basis_type)
  utility to convert Dakota type string to Pecos type enumeration

Private Attributes

- Pecos::BasisApproximation pecosBasisApprox
  the Pecos basis approximation, encompassing orthogonal and interpolation polynomial approximations
- Pecos::PolynomialApproximation * polyApproxRep
  convenience pointer to representation of Pecos polynomial approximation

Additional Inherited Members

14.189.1 Detailed Description

Derived approximation class for global basis polynomials.

The PecosApproximation class provides a global approximation based on basis polynomials. This includes
orthogonal polynomials used for polynomial chaos expansions and interpolation polynomials used for stochastic
collocation.
14.190. **POLYNOMIALREGRESSION CLASS REFERENCE**

14.189.2 Member Function Documentation

```cpp
void build() [inline], [protected], [virtual]
```
builds the approximation from scratch

This is the common base class portion of the virtual fn and is insufficient on its own; derived implementations should explicitly invoke (or reimplement) this base class contribution.

Reimplemented from `Approximation`.

References `Approximation::build()`, and `PecosApproximation::pecosBasisApprox`.

The documentation for this class was generated from the following files:

- PecosApproximation.hpp
- PecosApproximation.cpp

14.190 PolynomialRegression Class Reference

The `PolynomialRegression` class constructs a polynomial regressor using ordinary least squares.

Inheritance diagram for `PolynomialRegression`:

```
Surrogate

PolynomialRegression
```

Public Member Functions

- **PolynomialRegression ()**
  
  *Constructor that uses defaultConfigOptions and does not build.*

- **PolynomialRegression (const ParameterList &options)**
  
  *Constructor that sets configOptions and does not build.*

- **PolynomialRegression (const MatrixXd &samples, const MatrixXd &response, const ParameterList &options)**
  
  *Constructor sets configOptions and builds the Polynomial Regression surrogate.*

- **~PolynomialRegression ()**
  
  *Default destructor.*

- **void compute_basis_matrix (const MatrixXd &samples, MatrixXd &basis_matrix) const**
  
  Constructs a basis matrix for a set of samples according to the member variable basisIndices.

- **void build (const MatrixXd &samples, const MatrixXd &response) override**
  
  *Build the polynomial surrogate using specified build data.*

- **void value (const MatrixXd &eval_points, MatrixXd &approx_values) override**
  
  *Evaluate the polynomial surrogate at a set of prediction points.*

- **void gradient (const MatrixXd &samples, MatrixXd &gradient, const int qoi=0) override**
  
  *Evaluate the gradient of the polynomial surrogate at a set of prediction points.*

- **void hessian (const MatrixXd &sample, MatrixXd &hessian, const int qoi=0) override**
  
  *Evaluate the Hessian of the polynomial surrogate at a single point.*
CHAPTER 14. CLASS DOCUMENTATION

- const MatrixXd & get_polynomial_coeffs() const
  Get the polynomial surrogate’s coefficients.
- double get_polynomial_intercept() const
  Get the polynomial surrogate’s intercept/offset.
- int get_num_terms() const
  Get the number of terms in the polynomial surrogate.
- void set_polynomial_coeffs(const MatrixXd &coeffs)
  Set the polynomial surrogate’s coefficients.

Private Member Functions

- void default_options() override
  Construct and populate the defaultConfigOptions.

Private Attributes

- MatrixXi basisIndices
  Matrix that specifies the powers of each variable for each term in the polynomial - (numVariables by numTerms).
- std::shared_ptr<util::LinearSolverBase> linearSolver
  Linear solver for the ordinary least squares problem.
- int numTerms
  Number of terms in the polynomial basis.
- MatrixXd polynomialCoeffs
  Vector of coefficients for the polynomial surrogate.
- double polynomialIntercept
  Offset/intercept term for the polynomial surrogate.

Additional Inherited Members

14.190.1 Detailed Description

The PolynomialRegression class constructs a polynomial regressor using ordinary least squares. Users may specify the max degree and p-norm for a hyperbolic cross scheme to specify the terms in the polynomial basis. A p-norm = 1 results in a total order specification of max degree. The DataScaler class provides the option of scaling the basis matrix.

14.190.2 Constructor & Destructor Documentation

PolynomialRegression(const ParameterList &options)

Constructor that sets configOptions and does not build.
Parameters
### 14.190. POLYNOMIALREGRESSION CLASS REFERENCE

<table>
<thead>
<tr>
<th>in</th>
<th>options</th>
<th>List that overrides entries in defaultConfigOptions.</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>References Surrogate::configOptions, and PolynomialRegression::default_options().</td>
</tr>
</tbody>
</table>

**PolynomialRegression ( const MatrixXd & samples, const MatrixXd & response, const ParameterList & options )**

Constructor sets configOptions and builds the Polynomial Regression surrogate.

**Parameters**

<table>
<thead>
<tr>
<th>in</th>
<th>samples</th>
<th>Matrix of data for surrogate construction - (num_samples by num_features).</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>response</td>
<td>Vector of targets for surrogate construction - (num_samples by num_qoi = 1; only 1 response is supported currently).</td>
</tr>
<tr>
<td>in</td>
<td>options</td>
<td>List that overrides entries in defaultConfigOptions</td>
</tr>
</tbody>
</table>

References PolynomialRegression::build(), Surrogate::configOptions, and PolynomialRegression::default_options().

#### 14.190.3 Member Function Documentation

**void compute_basis_matrix ( const MatrixXd & samples, MatrixXd & basis_matrix ) const**

Constructs a basis matrix for a set of samples according to the member variable basisIndices.

**Parameters**

<table>
<thead>
<tr>
<th>in</th>
<th>samples</th>
<th>Matrix of sample points - (num_points by num_features).</th>
</tr>
</thead>
<tbody>
<tr>
<td>out</td>
<td>basis_matrix</td>
<td>Matrix that contains polynomial basis function evaluations in its rows for each sample point - (num_points by numTerms), numTerms being the number of terms in the polynomial basis.</td>
</tr>
</tbody>
</table>

References PolynomialRegression::basisIndices, PolynomialRegression::numTerms, and Surrogate::numVariables.
Referenced by PolynomialRegression::build(), PolynomialRegression::gradient(), PolynomialRegression::hessian(), and PolynomialRegression::value().

**void build ( const MatrixXd & samples, const MatrixXd & response ) [override], [virtual]**

Build the polynomial surrogate using specified build data.

**Parameters**

<table>
<thead>
<tr>
<th>in</th>
<th>samples</th>
<th>Matrix of data for surrogate construction - (num_samples by num_features).</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>response</td>
<td>Vector of targets for surrogate construction - (num_samples by num_qoi = 1; only 1 response is supported currently).</td>
</tr>
</tbody>
</table>

Implements Surrogate.

References PolynomialRegression::basisIndices, PolynomialRegression::compute_basis_matrix(), dakota::surrogates::compute_hyperbolic_indices(), dakota::surrogates::compute_reduced_indices(), Surrogate::configOptions, Surrogates::dataScaler, Surrogate::defaultConfigOptions, PolynomialRegression::linearSolver, Surrogate::numQOI, Surrogate::numSamples, PolynomialRegression::numTerms, Surrogate::numVariables, PolynomialRegression::polynomialCoeffs, PolynomialRegression::polynomialIntercept, dakota::util::scaler_factory(), DataScaler::scaler_type(), dakota::util::solver_factory(), and LinearSolverBase::solver_type().

Referenced by PolynomialRegression::PolynomialRegression().
void value ( const MatrixXd & eval_points, MatrixXd & approx_values ) [override], [virtual]

Evaluate the polynomial surrogate at a set of prediction points.
### Parameters

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>eval_points</td>
</tr>
<tr>
<td>out</td>
<td>approx_values</td>
</tr>
</tbody>
</table>

Implements Surrogate.

References PolynomialRegression::compute_basis_matrix(), Surrogate::dataScaler, PolynomialRegression::polynomialCoeffs, and PolynomialRegression::polynomialIntercept.

```cpp
void gradient ( const MatrixXd &samples, MatrixXd &gradient, const int qoi = 0 ) [override], [virtual]
```

Evaluate the gradient of the polynomial surrogate at a set of prediction points.

Parameters |
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>samples</td>
</tr>
<tr>
<td>out</td>
<td>gradient</td>
</tr>
<tr>
<td>out</td>
<td>qoi</td>
</tr>
</tbody>
</table>

Reimplemented from Surrogate.

References PolynomialRegression::basisIndices, PolynomialRegression::compute_basis_matrix(), Surrogate::dataScaler, PolynomialRegression::numTerms, Surrogate::numVariables, PolynomialRegression::polynomialCoeffs, and dakota::silence_unused_args().

```cpp
void hessian ( const MatrixXd &sample, MatrixXd &hessian, const int qoi = 0 ) [override], [virtual]
```

Evaluate the Hessian of the polynomial surrogate at a single point.

Parameters |
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>sample</td>
</tr>
<tr>
<td>out</td>
<td>hessian</td>
</tr>
<tr>
<td>out</td>
<td>qoi</td>
</tr>
</tbody>
</table>

Reimplemented from Surrogate.

References PolynomialRegression::basisIndices, PolynomialRegression::compute_basis_matrix(), Surrogate::dataScaler, PolynomialRegression::numTerms, Surrogate::numVariables, PolynomialRegression::polynomialCoeffs, and dakota::silence_unused_args().

The documentation for this class was generated from the following files:

- PolynomialRegression.hpp
- PolynomialRegression.cpp

---

### 14.191 PrefixingLineFilter Class Reference

Inherits line_filter.

#### Public Member Functions

- **PrefixingLineFilter** (const std::string &prefix_in)

  Constructor.
Private Member Functions

- `std::string do_filter (const std::string &line)`
  "Filter" the line by adding the prefix

Private Attributes

- `std::string linePrefix`
  Prefix for each line.

14.191.1 Detailed Description

`PrefixingLineFilter` is derived from a Boost stream filter class in order to preface output with specified text. In this case, the intent is to distinguish ROL output.

The documentation for this class was generated from the following file:

- ROLOptimizer.hpp

14.192 ProbabilityTransformModel Class Reference

Probability transformation specialization of `RecastModel`.

Inheritance diagram for `ProbabilityTransformModel`:

```
Model

RecastModel

ProbabilityTransformModel
```

Public Member Functions

- `ProbabilityTransformModel (const Model &sub_model, short u_space_type, bool truncate_bnds=false, Real bnd=10.)`
  *standard constructor*
- `~ProbabilityTransformModel ()`
  *destructor*

Static Public Member Functions

- `static void initialize_distribution_types (short u_space_type, const Pecos::MultivariateDistribution &x_dist, Pecos::MultivariateDistribution &u_dist)`
  *initialize transformed distribution types and instantiate mvDist*
Protected Member Functions

- Pecos::ProbabilityTransformation & probability_transformation ()
  
  return probability transformation employed by the Model (forwarded along to ProbabilityTransformModel recasting)

- bool resize_pending () const
  
  return true if a potential resize is still pending, such that sizing-based initialization should be deferred

- void update_from_subordinate_model (size_t depth=std::numeric_limits<size_t>::max())
  
  propagate vars/labels/bounds/targets from the bottom up

- void nested_variable_mappings (const SizetArray &c_index1, const SizetArray &di_index1, const SizetArray &ds_index1, const SizetArray &dr_index1, const ShortArray &c_target2, const ShortArray &di_target2, const ShortArray &ds_target2, const ShortArray &dr_target2)
  
  set primaryACVarMapIndices and secondaryACVarMapTargets (only, for now)

- const SizetArray & nested_acv1_indices () const
  
  return primaryACVarMapIndices

- const ShortArray & nested_acv2_targets () const
  
  return secondaryACVarMapTargets

- short query_distribution_parameter_derivatives () const
  
  calculate and return potential state of distribution parameter derivatives, but do not cache value in distParamDerivs

- void activate_distribution_parameter_derivatives ()
  
  activate distParamDerivs to {NO,MIXED,ALL,DERIVS}

- void deactivate_distribution_parameter_derivatives ()
  
  reset distParamDerivs to NO_DERIVS

- void assign_instance ()
  
  assign static pointer instance to this for use in static transformation functions

- void trans_grad_X_to_U (const RealVector &fn_grad_x, RealVector &fn_grad_u, const RealVector &x_vars)
  
  transform x-space gradient vector to u-space

- void trans_grad_U_to_X (const RealVector &fn_grad_u, RealVector &fn_grad_x, const RealVector &x_vars)
  
  transform u-space gradient vector to x-space

- void trans_grad_X_to_S (const RealVector &fn_grad_x, RealVector &fn_grad_s, const RealVector &x_vars)
  
  transform x-space gradient vector to gradient with respect to inserted distribution parameters

- void trans_hess_X_to_U (const RealSymMatrix &fn_hess_x, RealSymMatrix &fn_hess_u, const RealVector &x_vars, const RealVector &fn_grad_x)
  
  transform x-space Hessian matrix to u-space

- void initialize_transformation (short u_space_type)
  
  initialize transformed distribution types and natafTransform (construct time)

- void update_transformation ()
  
  update with latest distribution data (run time)

- void initialize_nataf ()
  
  instantiate and initialize natafTransform

- void verify_correlation_support (short u_space_type)
  
  verify that correlation warping is supported by Nataf for given variable types

- void initialize_dakota_variable_types ()
  
  initialize the continuous/discrete variable types using u-space types (converted from Pecos to Dakota)
• **void update_model_bounds** (bool truncate_bnds, Real bnd)
  
  update model bounds using u-space (truncated) distribution bounds

• **bool nonlinear_variables_mapping** (const Pecos::MultivariateDistribution &x_dist, const Pecos::MultivariateDistribution &u_dist) const
  
  detect when the variables transformation is nonlinear

• **size_t rv_index_to_corr_index** (size_t rv_index)
  
  convert vector<RandomVariable> index to active correlation index

• **size_t acv_index_to_corr_index** (size_t acv_index)
  
  convert allContinuousVars index to active correlation index

• **unsigned short pecos_to_dakota_variable_type** (unsigned short pecos_var_type, size_t rv_index)
  
  convert from Pecos To Dakota variable enumeration type for continuous aleatory uncertain variables used in variable transformations

### Static Protected Member Functions

• **static void vars_u_to_x_mapping** (const Variables &u_vars, Variables &x_vars)
  
  static function for RecastModels used for forward mapping of u-space variables from NonD Iterators to x-space variables for Model evaluations

• **static void vars_x_to_u_mapping** (const Variables &x_vars, Variables &u_vars)
  
  static function for RecastModels used for inverse mapping of x-space variables from data import to u-space variables for NonD Iterators

• **static void set_u_to_x_mapping** (const Variables &u_vars, const ActiveSet &u_set, ActiveSet &x_set)
  
  static function for RecastModels used to map u-space ActiveSets from NonD Iterators to x-space ActiveSets for Model evaluations

• **static void resp_x_to_u_mapping** (const Variables &x_vars, const Variables &u_vars, const Response &x_response, Response &u_response)
  
  static function for RecastModels used to map x-space responses from Model evaluations to u-space responses for return to NonD Iterator.

### Private Attributes

• Pecos::ProbabilityTransformation natafTransform
  
  Nonlinear variable transformation that encapsulates the required data for performing transformations from X -> Z -> U and back.

• short distParamDerivs
  
  indicates state of derivatives of final results with respect to distribution parameters s within resp_x_to_u_mapping() using the chain rule df/dx dx/ds. The default is to calculate derivatives with respect to standard random variables u using the chain rule df/dx dx/du.

• **bool truncatedBounds**
  
  boolean flag indicating use of distribution truncation for defining global model bounds

• Real boundVal
  
  number of +/- standard deviations used for defining bounds truncation

• SizetArray primaryACVarMapIndices
  
  "primary" all continuous variable mapping indices flowed down from higher level iteration

• ShortArray secondaryACVarMapTargets
  
  "secondary" all continuous variable mapping targets flowed down from higher level iteration
14.192. PROBABILITYTRANSFORMMODEL CLASS REFERENCE

Static Private Attributes

- static ProbabilityTransformModel * ptmInstance
  
  static pointer to this class for use in static callbacks

Additional Inherited Members

14.192.1 Detailed Description

Probability transformation specialization of RecastModel.

Specialization of RecastModel to transform a sub-model to u-space.

14.192.2 Member Function Documentation

void initialize_distribution_types ( short u_space_type, const Pecos::MultivariateDistribution & x_dist, Pecos::MultivariateDistribution & u_dist ) [static]

initialize transformed distribution types and instantiate mvDist

Build ProbabilityTransformation::ranVar arrays containing the uncertain variable distribution types and their corresponding means/standard deviations. This function is used when the Model variables are in x-space.

References Dakota::abort_handler().

Referenced by NonDLHSSampling::d_optimal_parameter_set(), and ProbabilityTransformModel::initialize_transformation().

void update_from_subordinate_model ( size_t depth = std::numeric_limits<size_t>::max() ) [inline], [protected], [virtual]

propagate vars/labels/bounds/targets from the bottom up

used only for instantiate-on-the-fly model recursions (all RecastModel instantiations and alternate DataFit-SurrModel instantiations). Simulation, Hierarchical, and Nested Models do not redefine the function since they do not support instantiate-on-the-fly. This means that the recursion will stop as soon as it encounters a Model that was instantiated normally, which is appropriate since ProblemDescDB-constructed Models use top-down information flow and do not require bottom-up updating.

Reimplemented from Model.

References RecastModel::subModel, RecastModel::update_from_model(), Model::update_from_subordinate_model(), and ProbabilityTransformModel::update_transformation().

void vars_u_to_x_mapping ( const Variables & u_vars, Variables & x_vars ) [inline], [static], [protected]

static function for RecastModels used for forward mapping of u-space variables from NonD Iterators to x-space variables for Model evaluations

Map the variables from iterator space (u) to simulation space (x).

References Variables::continuous_variables(), Variables::continuous_variables_view(), ProbabilityTransformModel::natafTransform, and ProbabilityTransformModel::ptmInstance.

Referenced by ProbabilityTransformModel::ProbabilityTransformModel().
void vars_x_to_u_mapping ( const Variables & x_vars, Variables & u_vars ) [inline], [static], [protected]

static function for RecastModels used for inverse mapping of x-space variables from data import to u-space variables for NonD Iterators

Map the variables from simulation space (x) to iterator space (u).
References Variables::continuous_variables(), Variables::continuous_variables_view(), ProbabilityTransformModel::natafTransform, and ProbabilityTransformModel::ptmInstance.
Referenced by ProbabilityTransformModel::ProbabilityTransformModel().

void set_u_to_x_mapping ( const Variables & u_vars, const ActiveSet & u_set, ActiveSet & x_set ) [static], [protected]

static function for RecastModels used to map u-space ActiveSets from NonD Iterators to x-space ActiveSets for Model evaluations

Define the DVV for x-space derivative evaluations by augmenting the iterator requests to account for correlations.
References Dakota::NPOS, ProbabilityTransformModel::acv_index_to_corr_index(), Variables::all_continuous_variable_ids(), Dakota::contains(), Variables::continuous_variable_ids(), ActiveSet::derivative_vector(), Variables::inactive_continuous_variable_ids(), Model::multivariate_distribution(), ProbabilityTransformModel::ptmInstance, and RecastModel::subModel.
Referenced by ProbabilityTransformModel::ProbabilityTransformModel().

14.192.3 Member Data Documentation

ProbabilityTransformModel * ptmInstance [static], [private]

static pointer to this class for use in static callbacks
initialization of static needed by RecastModel
Referenced by ProbabilityTransformModel::assign_instance(), ProbabilityTransformModel::resp_x_to_u_mapping(), ProbabilityTransformModel::set_u_to_x_mapping(), ProbabilityTransformModel::vars_u_to_x_mapping(), and ProbabilityTransformModel::vars_x_to_u_mapping().
The documentation for this class was generated from the following files:

- ProbabilityTransformModel.hpp
- ProbabilityTransformModel.cpp

14.193 ProblemDescDB Class Reference

The database containing information parsed from the DAKOTA input file.
Inheritance diagram for ProblemDescDB:

ProblemDescDB
  └── NIDRProblemDescDB
Public Member Functions

- ProblemDescDB ()
  default constructor
- ProblemDescDB (ParallelLibrary &parallel_lib)
  standard constructor
- ProblemDescDB (const ProblemDescDB &db)
  copy constructor
- ~ProblemDescDB ()
  destructor
- ProblemDescDB operator= (const ProblemDescDB &db)
  assignment operator
- void parse_inputs (ProgramOptions &prog_opts, DbCallbackFunctionPtr callback=NULL, void *callback_data=NULL)
  Parses the input file or input string if present and executes callbacks. Does not perform any validation.
- void check_and_broadcast (const ProgramOptions &prog_opts)
  performs check_input, broadcast, and post_process, but for now, allowing separate invocation through the public API as well
- void check_input ()
  verifies that there is at least one of each of the required keywords in the dakota input file
- void broadcast ()
  invokes send_db_buffer() and receive_db_buffer() to broadcast DB data across the processor allocation. Used by manage_inputs().
- void post_process ()
  post-processes the (minimal) input specification to assign default variables/responses specification arrays. Used by manage_inputs().
- void lock ()
  Locks the database in order to prevent data access when the list nodes may not be set properly. Unlocked by a set nodes operation.
- void unlock ()
  Explicitly unlocks the database. Use with care.
- void set_db_list_nodes (const String &method_tag)
  set dataMethodIter based on a method identifier string to activate a particular method specification in dataMethodList and use pointers from this method specification to set all other list iterators.
- void set_db_list_nodes (size_t method_index)
  set dataMethodIter based on an index within dataMethodList to activate a particular method specification and use pointers from this method specification to set all other list iterators.
- void resolve_top_method (bool set_model_nodes=true)
  For a (default) environment lacking a top method pointer, this function is used to determine which of several potential method specifications corresponds to the top method and then sets the list nodes accordingly.
- void set_db_method_node (const String &method_tag)
  set dataMethodIter based on a method identifier string to activate a particular method specification (only).
- void set_db_method_node (size_t method_index)
  set dataMethodIter based on an index within dataMethodList to activate a particular method specification (only).
- size_t get_db_method_node ()
CHAPTER 14. CLASS DOCUMENTATION

- return the index of the active node in dataMethodList
- void set_db_model_nodes (const String &model_tag)
  set the model list iterators (dataModelIter, dataVariablesIter, dataInterfaceIter, and dataResponsesIter) based on the model identifier string
- void set_db_model_nodes (size_t model_index)
  set the model list iterators (dataModelIter, dataVariablesIter, dataInterfaceIter, and dataResponsesIter) based on an index within dataModelList
- size_t get_db_model_node ()
  return the index of the active node in dataModelList
- void set_db_variables_node (const String &variables_tag)
  set dataVariablesIter based on the variables identifier string
- void set_db_interface_node (const String &interface_tag)
  set dataInterfaceIter based on the interface identifier string
- void set_db_responses_node (const String &responses_tag)
  set dataResponsesIter based on the responses identifier string
- ParallelLibrary & parallel_library () const
  return the parallelLib reference
- IteratorList & iterator_list ()
  return a list of all Iterator objects that have been instantiated
- ModelList & model_list ()
  return a list of all Model objects that have been instantiated
- VariablesList & variables_list ()
  return a list of all Variables objects that have been instantiated
- InterfaceList & interface_list ()
  return a list of all Interface objects that have been instantiated
- ResponseList & response_list ()
  return a list of all Response objects that have been instantiated
- const RealMatrixArray & get_rma (const String &entry_name) const
  get a RealMatrixArray out of the database based on an identifier string
- const RealVector & get rv (const String &entry_name) const
  get a RealVector out of the database based on an identifier string
- const IntVector & get_iv (const String &entry_name) const
  get an IntVector out of the database based on an identifier string
- const BitArray & get.ba (const String &entry_name) const
  get a BitArray out of the database based on an identifier string
- const SizetArray & get_sza (const String &entry_name) const
  get an SizetArray out of the database based on an identifier string
- const UShortArray & get_usa (const String &entry_name) const
  get an UShortArray out of the database based on an identifier string
- const RealSymMatrix & get_rsm (const String &entry_name) const
  get a RealSymMatrix out of the database based on an identifier string
- const RealVectorArray & get_rva (const String &entry_name) const
  get a RealVectorArray out of the database based on an identifier string
- const IntVectorArray & get_iva (const String &entry_name) const
  get an IntVectorArray out of the database based on an identifier string
- const IntSet & get_is (const String &entry_name) const
  get an IntSet out of the database based on an identifier string
- const IntSetArray & get_isa (const String &entry_name) const
  get an IntSetArray out of the database based on an identifier string
- const StringSetArray & get_ssa (const String &entry_name) const
  get an StringSetArray out of the database based on an identifier string
- const RealSetArray & get_rsa (const String &entry_name) const
  get a RealSetArray out of the database based on an identifier string
- const IntRealMapArray & get_irma (const String &entry_name) const
  get an IntRealMapArray out of the database based on an identifier string
- const StringRealMapArray & get_srma (const String &entry_name) const
  get an StringRealMapArray out of the database based on an identifier string
- const RealRealMapArray & get_rrma (const String &entry_name) const
  get a RealRealMapArray out of the database based on an identifier string
- const RealRealPairRealMapArray & get_rrrma (const String &entry_name) const
  get a RealRealPairRealMapArray out of the database based on an identifier string
- const IntIntPairRealMapArray & get_iirma (const String &entry_name) const
  get an IntIntPairRealMapArray out of the database based on an identifier string
- const StringArray & get_sa (const String &entry_name) const
  get a StringArray out of the database based on an identifier string
- const String2DArray & get_s2a (const String &entry_name) const
  get a String2DArray out of the database based on an identifier string
- const String & get_string (const String &entry_name) const
  get a String out of the database based on an identifier string
- int get_int (const String &entry_name) const
  get an int out of the database based on an identifier string
- short get_short (const String &entry_name) const
  get a short out of the database based on an identifier string
- unsigned short get_ushort (const String &entry_name) const
  get an unsigned short out of the database based on an identifier string
- size_t get_sizet (const String &entry_name) const
  get a size_t out of the database based on an identifier string
- bool get_bool (const String &entry_name) const
  get a bool out of the database based on an identifier string
- void ** get_voidss (const String &entry_name) const
  for getting a void**, e.g., &dlLib
- void insert_node (const DataEnvironment &data_env)
  set the DataEnvironment object
- void insert_node (const DataMethod &data_method)
add a DataMethod object to the dataMethodList
• void insert_node (const DataModel &data_model)
  add a DataModel object to the dataModelList
• void insert_node (DataVariables &data_variables)
  add a DataVariables object to the dataVariablesList
• void insert_node (const DataInterface &data_interface)
  add a DataInterface object to the dataInterfaceList
• void insert_node (const DataResponses &data_responses)
  add a DataResponses object to the dataResponsesList
• void set (const String &entry_name, const RealVector &rv)
  set a RealVector within the database based on an identifier string
• void set (const String &entry_name, const IntVector &iv)
  set an IntVector within the database based on an identifier string
• void set (const String &entry_name, const BitArray &ba)
  set a BitArray within the database based on an identifier string
• void set (const String &entry_name, const RealSymMatrix &rsm)
  set a RealSymMatrix within the database based on an identifier string
• void set (const String &entry_name, const RealVectorArray &rva)
  set a RealVectorArray within the database based on an identifier string
• void set (const String &entry_name, const IntVectorArray &iva)
  set an IntVectorArray within the database based on an identifier string
• void set (const String &entry_name, const IntSetArray &isa)
  set an IntSetArray within the database based on an identifier string
• void set (const String &entry_name, const RealSetArray &rsa)
  set a RealSetArray within the database based on an identifier string
• void set (const String &entry_name, const IntRealMapArray &irma)
  set an IntRealMapArray within the database based on an identifier string
• void set (const String &entry_name, const StringRealMapArray &srma)
  set a StringRealMapArray within the database based on an identifier string
• void set (const String &entry_name, const RealRealMapArray &rrma)
  set a RealRealMapArray within the database based on an identifier string
• void set (const String &entry_name, const RealRealPairRealMapArray &iirma)
  set a RealRealPairRealMapArray in the db based on an identifier string
• void set (const String &entry_name, const IntIntPairRealMapArray &iirma)
  set an IntIntPairRealMapArray in the db based on an identifier string
• void set (const String &entry_name, const StringArray &sa)
  set a StringArray within the database based on an identifier string
• int min_procs_per_ea ()
  compute minimum evaluation partition size based on lower level overrides
• int max_procs_per_ea ()
  compute maximum evaluation partition size based on lower level overrides and concurrency levels
• int min_procs_per_ie ()
  compute minimum iterator partition size based on lower level overrides
• int max_procs_per_ie (int max_eval_concurrency)
  
  compute maximum iterator partition size based on lower level overrides and concurrency levels

• bool method_locked () const
  
  return methodDBLocked

• bool model_locked () const
  
  return modelDBLocked

• bool variables_locked () const
  
  return variablesDBLocked

• bool interface_locked () const
  
  return interfaceDBLocked

• bool responses_locked () const
  
  return responsesDBLocked

• bool is_null () const
  
  function to check dbRep (does this envelope contain a letter)

**Static Public Member Functions**

• static int min_procs_per_level (int min_procs_per_server, int pps_spec, int num_serv_spec)
  
  compute minimum partition size for a parallel level based on lower level overrides

• static int max_procs_per_level (int max_procs_per_server, int pps_spec, int num_serv_spec, short sched_spec, int asynch_local_cone, bool peer_dynamic_avail, int max_concurrency)
  
  compute maximum partition size for a parallel level based on lower level overrides

**Protected Member Functions**

• ProblemDescDB (BaseConstructor, ParallelLibrary &parallel_lib)
  
  constructor initializes the base class part of letter classes (BaseConstructor overloading avoids infinite recursion in the derived class constructors - Coplien, p. 139)

• virtual void derived_parse_inputs (const std::string &dakota_input_file, const std::string &dakota_input_string, const std::string &parser_options)
  
  derived class specifics within parse_inputs()

• virtual void derived_broadcast ()
  
  derived class specifics within broadcast()

• virtual void derived_post_process ()
  
  derived class specifics within post_process()

**Protected Attributes**

• DataEnvironment environmentSpec
  
  the environment specification (only one allowed) resulting from a call to environment_kwhandler() or insert_node()
list of variables specifications, one for each call to variables{kwhandler()} or insert_node()

- std::list<DataInterface> dataInterfaceList
  list of interface specifications, one for each call to interface{kwhandler()} or insert_node()

- std::list<DataResponses> dataResponsesList
  list of responses specifications, one for each call to responses{kwhandler()} or insert_node()

- size_t environmentCntr
  counter for environment specifications used in check_input

Private Member Functions

- const Iterator & get_iterator ()
  retrieve an existing Iterator, if it exists in iteratorList, or instantiate a new one

- const Iterator & get_iterator (Model &model)
  retrieve an existing Iterator, if it exists in iteratorList, or instantiate a new one

- const Iterator & get_iterator (const String &method_name, Model &model)
  retrieve an existing Iterator, if it exists in iteratorByNameList, or instantiate a new one

- const Model & get_model ()
  retrieve an existing Model, if it exists, or instantiate a new one

- const Variables & get_variables ()
  retrieve an existing Variables, if it exists, or instantiate a new one

- const Interface & get_interface ()
  retrieve an existing Interface, if it exists, or instantiate a new one

- const Response & get_response (short type, const Variables &vars)
  retrieve an existing Response, if it exists, or instantiate a new one

- ProblemDescDB * get_db (ParallelLibrary &parallel_lib)
  Used by the envelope constructor to instantiate the correct letter class.

- void send_db_buffer ()
  MPI send of a large buffer containing environmentSpec and all objects in dataMethodList, dataModelList, dataVariablesList, dataInterfaceList, and dataResponsesList. Used by manage_inputs().

- void receive_db_buffer ()
  MPI receive of a large buffer containing environmentSpec and all objects in dataMethodList, dataModelList, dataVariablesList, dataInterfaceList, and dataResponsesList. Used by manage_inputs().

- bool model_has_interface (DataModelRep *model_rep) const
  helper function for determining whether an interface specification should be active, based on model type

- void echo_input_file (const std::string &dakota_input_file, const std::string &dakota_input_string, const std::string &tmpl_qualifier="")
  echo the (potentially) specified input file or string to stdout

- void enforce_unique_ids ()
  require user-specified block identifiers to be unique
Private Attributes

- **ParallelLibrary & parallelLib**
  - reference to the parallel_lib object passed from main
- **std::list< DataMethod >::iterator dataMethodIter**
  - iterator identifying the active list node in dataMethodList
- **std::list< DataModel >::iterator dataModelIter**
  - iterator identifying the active list node in dataModelList
- **std::list< DataVariables >::iterator dataVariablesIter**
  - iterator identifying the active list node in dataVariablesList
- **std::list< DataInterface >::iterator dataInterfaceIter**
  - iterator identifying the active list node in dataInterfaceList
- **std::list< DataResponses >::iterator dataResponsesIter**
  - iterator identifying the active list node in dataResponsesList
- **IteratorList iteratorList**
  - list of iterator objects, one for each method specification
- **IteratorList iteratorByNameList**
  - list of iterator objects, one for each lightweight instantiation by name
- **ModelList modelList**
  - list of model objects, one for each model specification
- **VariablesList variablesList**
  - list of variables objects, one for each variables specification
- **InterfaceList interfaceList**
  - list of interface objects, one for each interface specification
- **ResponseList responseList**
  - list of response objects, one for each responses specification
- **bool methodDBLocked**
  - prevents use of get_<type> retrieval and set_<type> update functions prior to setting the list node for the active method specification
- **bool modelDBLocked**
  - prevents use of get_<type> retrieval and set_<type> update functions prior to setting the list node for the active model specification
- **bool variablesDBLocked**
  - prevents use of get_<type> retrieval and set_<type> update functions prior to setting the list node for the active variables specification
- **bool interfaceDBLocked**
  - prevents use of get_<type> retrieval and set_<type> update functions prior to setting the list node for the active interface specification
- **bool responsesDBLocked**
  - prevents use of get_<type> retrieval and set_<type> update functions prior to setting the list node for the active responses specification
- **ProblemDescDB * dbRep**
pointer to the letter (initialized only for the envelope)

- int referenceCount
  number of objects sharing dbRep

Friends

- class Model
  Model requires access to get_variables() and get_response()
- class SimulationModel
  SimulationModel requires access to get_interface()
- class HierarchSurrModel
  HierarchSurrModel requires access to get_model()
- class DataFitSurrModel
  DataFitSurrModel requires access to get_iterator() and get_model()
- class NestedModel
  NestedModel requires access to get_interface(), get_response(), get_iterator(), and get_model()
- class ActiveSubspaceModel
- class AdaptedBasisModel
- class RandomFieldModel
- class Environment
  Environment requires access to get_iterator()
- class IteratorScheduler
  Environment requires access to get_iterator()
- class Iterator
  Iterator requires access to get_model()
- class MetaIterator
  Iterator requires access to get_model()
- class SeqHybridMetaIterator
  SeqHybridMetaIterator requires access to get_model()
- class CollabHybridMetaIterator
  CollabHybridMetaIterator requires access to get_model()
- class ConcurrentMetaIterator
  ConcurrentMetaIterator requires access to get_model()
- class SurrBasedLocalMinimizer
  HierarchSurrBasedLocalMinimizer requires access to get_iterator()
- class SurrBasedGlobalMinimizer
  SurrBasedGlobalMinimizer requires access to get_iterator()
- class PebbldMinimizer
  PEBBLMinimizer requires access to get_iterator()
14.193. **Detailed Description**

The database containing information parsed from the DAKOTA input file.

The `ProblemDescDB` class is a database for DAKOTA input file data that is populated by a parser defined in a derived class. When the parser reads a complete keyword, it populates a data class object (`DataEnvironment`, `DataMethod`, `DataVariables`, `DataInterface`, or `DataResponses`) and, for all cases except environment, appends the object to a linked list (`dataMethodList`, `dataVariablesList`, `dataInterfaceList`, or `dataResponsesList`). No environment linked list is used since only one environment specification is allowed.

14.193.2 **Constructor & Destructor Documentation**

`ProblemDescDB ( )`

_default constructor_

The default constructor: `dbRep` is NULL in this case. This makes it necessary to check for NULL in the copy constructor, assignment operator, and destructor.

`ProblemDescDB ( ParallelLibrary & parallel_lib )`

_standard constructor_

This is the envelope constructor which uses `problem_db` to build a fully populated `db` object. It only needs to extract enough data to properly execute `get_db(problem_db)`, since the constructor overloaded with `BaseConstructor` builds the actual base class data inherited by the derived classes.

References Dakota::abort_handler, `ProblemDescDB::dbRep`, and `ProblemDescDB::get_db()`.

`ProblemDescDB ( const ProblemDescDB & db )`

_copy constructor_

Copy constructor manages sharing of `dbRep` and incrementing of `referenceCount`.

References `ProblemDescDB::dbRep`, and `ProblemDescDB::referenceCount`.

`~ProblemDescDB ( )`

_destructor_

Destructor decrements `referenceCount` and only deletes `dbRep` when `referenceCount` reaches zero.

References Dakota::Dak_pddb, `ProblemDescDB::dbRep`, and `ProblemDescDB::referenceCount`.

`ProblemDescDB ( BaseConstructor , ParallelLibrary & parallel_lib ) [protected]`

_constructor initializes the base class part of letter classes (BaseConstructor overloading avoids infinite recursion in the derived class constructors - Coplien, p. 139)_

This constructor is the one which must build the base class data for all derived classes. `get_db()` instantiates a derived class letter and the derived constructor selects this base class constructor in its initialization list (to avoid the recursion of the base class constructor calling `get_db()` again). Since the letter IS the representation, its representation pointer is set to NULL (an uninitialized pointer causes problems in `~ProblemDescDB`).

14.193.3 **Member Function Documentation**

`ProblemDescDB operator= ( const ProblemDescDB & db )`

_assignment operator_
References ProblemDescDB::dbRep, and ProblemDescDB::referenceCount.

void parse_inputs ( ProgramOptions & prog_opts, DbCallbackFunctionPtr callback = NULL, void * callback_data = NULL )

Parses the input file or input string if present and executes callbacks. Does not perform any validation.
DB setup phase 1: parse the input file and execute callback functions if present. Rank 0 only.
DB setup phase 2: optionally insert additional data via late sets. Rank 0 only.
References Dakota::abort_handler(), ProblemDescDB::dbRep, ProblemDescDB::derived_parseInputs(), ProgramOptions::echo_input(), ProblemDescDB::echo_input_file(), ProgramOptions::input_file(), ProgramOptions::input_string(), ProblemDescDB::parallelLib, ProblemDescDB::parse_inputs(), ProgramOptions::parser_options(), ProgramOptions::preproc_cmd(), ProgramOptions::preproc_input(), Dakota::pprepro_input(), Dakota::string_to_tmpfile(), and ParallelLibrary::world_rank().
Referenced by Environment::parse(), and ProblemDescDB::parse_inputs().

void check_and_broadcast ( const ProgramOptions & prog_opts )
performs check_input, broadcast, and post_process, but for now, allowing separate invocation through the public API as well
DB setup phase 3: perform basic checks on keywords counts in current DB state, then sync to all processors.
References ProblemDescDB::broadcast(), ProblemDescDB::check_and_broadcast(), ProblemDescDB::dbRep, ProblemDescDB::parallelLib, ProblemDescDB::post_process(), and ParallelLibrary::world_rank().
Referenced by ProblemDescDB::check_and_broadcast(), LibraryEnvironment::done_modifying_db(), and Environment::parse().

void check_input ( )
verifies that there is at least one of each of the required keywords in the dakota input file
NOTE: when using library mode in a parallel application, check_input() should either be called only on world-Rank 0, or it should follow a matched send_db_buffer()/receive_db_buffer() pair.
References Dakota::abort_handler(), ProblemDescDB::check_input(), ParallelLibrary::command_line_post_run_input(), ParallelLibrary::command_line_post_run_output(), ParallelLibrary::command_line_pre_run_input(), ParallelLibrary::command_line_pre_run_output(), ParallelLibrary::command_line_run_input(), ParallelLibrary::command_line_run_output(), ParallelLibrary::command_line_user_modes(), ProblemDescDB::dataInterfaceList, ProblemDescDB::dataMethodList, ProblemDescDB::dataModelList, ProblemDescDB::dataResponsesList, ProblemDescDB::dataVariablesList, ProblemDescDB::dbRep, ProblemDescDB::environmentCntr, ProblemDescDB::parallelLib, and Dakota::strbegins().
Referenced by ProblemDescDB::check_and_broadcast(), and ProblemDescDB::check_input().

void post_process ( )
post-processes the (minimal) input specification to assign default variables/responses specification arrays. Used by manage_inputs().
When using library mode in a parallel application, post_process() should be called on all processors following broadcast() of a minimal problem specification.
References ProblemDescDB::dbRep, and ProblemDescDB::derived_post_process().
Referenced by ProblemDescDB::check_and_broadcast().
ProblemDescDB * get_db ( ParallelLibrary & parallel_lib ) [private]

Used by the envelope constructor to instantiate the correct letter class.
Initializes dbRep to the appropriate derived type. The standard derived class constructors are invoked.
References Dakota::Dak_pddb.
Referenced by ProblemDescDB::ProblemDescDB().

void enforce_unique_ids ( ) [private]

require user-specified block identifiers to be unique
Require string identifiers id_* to be unique across all blocks of each type (method, model, variables, interface, responses
For now, this allows duplicate empty ID strings. Would be better to require unique IDs when more than one block of a given type appears in the input file (instead of use-the-last-parsed)
References Dakota::abort_handler(), ProblemDescDB::dataInterfaceList, ProblemDescDB::dataMethodList, ProblemDescDB::dataModelList, ProblemDescDB::dataResponsesList, and ProblemDescDB::dataVariablesList.
Referenced by ProblemDescDB::broadcast().
The documentation for this class was generated from the following files:
• ProblemDescDB.hpp
• ProblemDescDB.cpp

14.194 ProcessApplicInterface Class Reference

Derived application interface class that spawns a simulation code using a separate process and communicates with it through files.

Inheritance diagram for ProcessApplicInterface:

Public Member Functions

• ProcessApplicInterface ( const ProblemDescDB &problem_db )
  constructor

• ~ProcessApplicInterface ()
  destructor
Protected Member Functions

- void derived_map (const Variables &vars, const ActiveSet &set, Response &response, int fn_eval_id)
  Called by map() and other functions to execute the simulation in synchronous mode. The portion of performing an evaluation that is specific to a derived class.

- void derived_map_asynch (const ParamResponsePair &pair)
  Called by map() and other functions to execute the simulation in asynchronous mode. The portion of performing an asynchronous evaluation that is specific to a derived class.

- void wait_local_evaluations (PRPQueue &prp_queue)
  For asynchronous function evaluations, this method is used to detect completion of jobs and process their results. It provides the processing code that is specific to derived classes. This version waits for at least one completion.

- void test_local_evaluations (PRPQueue &prp_queue)
  For asynchronous function evaluations, this method is used to detect completion of jobs and process their results. It provides the processing code that is specific to derived classes. This version is nonblocking and will return without any completions if none are immediately available.

- const StringArray & analysis_drivers () const
  retrieve the analysis drivers specification for application interfaces

- void file_cleanup () const

- void file_and_workdir_cleanup (const bfs::path &params_path, const bfs::path &results_path, const bfs::path &workdir_path, const String &tag) const

- void remove_params_results_files (const bfs::path &params_path, const bfs::path &results_path) const
  Remove (potentially autotagged for multiple programs) parameters and results files with passed root names.

- void autotag_files (const bfs::path &params_path, const bfs::path &results_path, const String &eval_id_tag) const
  Utility to automatically tag parameters and results files with passed root names (the files may already need per-program tagging)

- virtual void wait_local_evaluation_sequence (PRPQueue &prp_queue)=0
  version of wait_local_evaluations() managing of set of individual asynchronous evaluations

- virtual void test_local_evaluation_sequence (PRPQueue &prp_queue)=0
  version of test_local_evaluations() managing of set of individual asynchronous evaluations

- virtual void map_bookkeeping (pid_t pid, int fn_eval_id)=0
  bookkeeping of process and evaluation ids for asynchronous maps

- virtual pid_t create_evaluation_process (bool block_flag)=0
  Spawn the evaluation by managing the input filter, analysis drivers, and output filter. Called from derived_map() & derived_map_asynch().

- void wait_local_evaluation_batch (PRPQueue &prp_queue)
  batch version of wait_local_evaluations()

- void test_local_evaluation_batch (PRPQueue &prp_queue)
  batch version of test_local_evaluations()

- void synchronous_local_analyses (int start, int end, int step)
  execute analyses synchronously on the local processor

- void define_filenames (const String &eval_id_tag)
  define modified filenames from user input by handling Unix temp file and optionally tagging with given eval_id_tag

- void write_parameters_files (const Variables &vars, const ActiveSet &set, const Response &response, const int id)
write the parameters data and response request data to one or more parameters files (using one or more invocations of write parameters file()) in either standard or aprepro format

- void read results files (Response &response, const int id, const String &eval id_tag)
  read the response object from one or more results files using full eval id_tag passed

- bfs::path get workdir name ()
  construct a work directory name (tmp or named), with optional tag

- void prepare process environment ()
  set PATH, environment variables, and change directory prior to fork/system/spawn

- void reset process environment ()
  reset PATH and current directory after system/spawn (workdir case)

Protected Attributes

- bool fileTagFlag
  flags tagging of parameter/results files

- bool fileSaveFlag
  flags retention of parameter/results files

- bool commandLineArgs
  flag indicating use of passing of filenames as command line arguments to the analysis drivers and input/output filters

- bool apreproFlag
  flag indicating use of the APREPRO (the Sandia "A PRE PROcessor" utility) format for parameter files

- unsigned short resultsFileFormat
  results file format

- bool multipleParamsFiles
  flag indicating the need for separate parameters files for multiple analysis drivers

- std::string iFilterName
  the name of the input filter (input filter user specification)

- std::string oFilterName
  the name of the output filter (output filter user specification)

- std::vector<String> programNames
  the names of the analysis code programs (analysis drivers user specification)

- std::string specifiedParamsFileName
  the name of the parameters file from user specification

- std::string paramsFileName
  the parameters file name actually used (modified with tagging or temp files); only valid from define filenames to write parameters files

- std::string paramsFileWritten
  actual, qualified name of the params file written, possibly with workdir

- std::string specifiedResultsFileName
  the name of the results file from user specification

- std::string resultsFileName
  the results file name actually used (modified with tagging or temp files); only valid from define filenames to write parameters files
• std::string resultsFileWritten
  actual, qualified name of the results file written, possibly with workdir
• std::string fullEvalId
  complete evalIdTag, possibly including hierarchical tagging and final eval id, but not program numbers, for passing to write_parameters_files
• bool allowExistingResults
  by default analysis code interfaces delete results files if they exist; user may override with this flag and we’ll try to gather and only fork if needed
• std::map<int, PathTriple> fileNameMap
  Maps function evaluation ID to triples (parameters, results, and workdir) paths used in spawning function evaluations. Workdir will be empty if not created specifically for this eval.
• bool useWorkdir
  whether to use a work directory
• std::string workDirName
  work directory name, if specified...
• bool dirTag
  whether to tag the working directory
• bool dirSave
  whether dir_save was specified
• bfs::path curWorkdir
  active working directory for this evaluation; valid only from define_filenames to create_evaluation_process
• bfs::path createdDir
  non-empty if created for this eval; valid only from define_filenames to write_parameters_files
• StringArray linkFiles
  template directory (if specified)
• StringArray copyFiles
  template files (if specified)
• bool templateReplace
  whether to replace existing files

Private Member Functions
• void write_parameters_file (const Variables &vars, const ActiveSet &set, const Response &response, const std::string &prog, const std::vector<String> &an_comps, const std::string &params_fname, const bool file_mode_out=true)
  write the variables, active set vector, derivative variables vector, and analysis components to the specified parameters file in either standard or aprepro format
• void read_results_file (Response &response, const bfs::path &path, const int id)
  Open and read the results file at path, properly handling errors.

14.194.1 Detailed Description
Derived application interface class that spawns a simulation code using a separate process and communicates with it through files.
ProcessApplicInterface is subclassed for process handles or file completion testing.
## 14.194.2 Member Function Documentation

### void file_cleanup() const [protected], [virtual]

Remove any files and directories still referenced in the fileNameMap

- Reimplemented from Interface.
- References WorkdirHelper::concat_path(), ProcessApplicInterface::dirSave, ProcessApplicInterface::fileNameMap, ProcessApplicInterface::fileSaveFlag, ProcessApplicInterface::iFilterName, ProcessApplicInterface::multipleParamsFiles, ProcessApplicInterface::programNames, and WorkdirHelper::recursive_remove().

### void autotag_files(const bfs::path &params_path, const bfs::path &results_path, const String &eval_id_tag) const [protected]

Utility to automatically tag parameters and results files with passed root names (the files may already need per-program tagging)

- Move specified params and results files to unique tagged versions when needed
- References WorkdirHelper::concat_path(), ProcessApplicInterface::iFilterName, ProcessApplicInterface::multipleParamsFiles, ProcessApplicInterface::oFilterName, Interface::outputLevel, ProcessApplicInterface::programNames, WorkdirHelper::rename(), ProcessApplicInterface::specifiedParamsFileName, ProcessApplicInterface::specifiedResultsFileName, and ApplicationInterface::suppressOutput.

### void synchronous_local_analyses (int start, int end, int step) [inline], [protected]

execute analyses synchronously on the local processor

- Execute analyses synchronously in succession on the local processor (start to end in step increments). Modeled after ApplicationInterface::synchronous_local_evaluations().
- References ApplicationInterface::synchronous_local_analysis().
- Referenced by ProcessHandleApplicInterface::create_evaluation_process().

### void prepare_process_environment() [protected]

set PATH, environment variables, and change directory prior to fork/system/spawn

- Guidance: environment (PATH, current directory) should be set immediately before Dakota spawns a process and reset immediately afterwards (except fork which never returns)
- References WorkdirHelper::change_directory(), ProcessApplicInterface::curWorkdir, Interface::outputLevel, ProcessApplicInterface::paramsFileName, ProcessApplicInterface::resultsFileName, WorkdirHelper::set_environment(), WorkdirHelper::set_preferred_path(), and ProcessApplicInterface::useWorkdir.
- Referenced by SpawnApplicInterface::create_analysis_process(), ForkApplicInterface::create_analysis_process(), SysCallApplicInterface::spawn_analysis_to_shell(), SysCallApplicInterface::spawn_evaluation_to_shell(), SysCallApplicInterface::spawn_input_filter_to_shell(), and SysCallApplicInterface::spawn_output_filter_to_shell().

### void reset_process_environment() [protected]

reset PATH and current directory after system/spawn (workdir case)

- Undo anything done prior to spawn
- References Interface::outputLevel, WorkdirHelper::reset(), WorkdirHelper::startup_pwd(), and ProcessApplicInterface::useWorkdir.
- Referenced by SpawnApplicInterface::create_analysis_process(), ForkApplicInterface::create_analysis_process(), SysCallApplicInterface::spawn_analysis_to_shell(), SysCallApplicInterface::spawn_evaluation_to_shell(), SysCallApplicInterface::spawn_input_filter_to_shell(), and SysCallApplicInterface::spawn_output_filter_to_shell().
void read_results_file (Response & response, const bfs::path & path, const int id) [private]

Open and read the results file at path, properly handling errors.
Helper for read_results_files that opens the results file at results_path and reads it, handling various errors/exceptions.
References Dakota::abort_handler(), Response::read(), and ProcessApplicInterface::resultsFileFormat.
Referenced by ProcessApplicInterface::read_results_files().
The documentation for this class was generated from the following files:

- ProcessApplicInterface.hpp
- ProcessApplicInterface.cpp

### 14.195 ProcessHandleApplicInterface Class Reference

Derived application interface class that spawns a simulation code using a separate process, receives a process identifier, and communicates with the spawned process through files.

Inheritance diagram for ProcessHandleApplicInterface:

```
14.195 ProcessHandleApplicInterface Class Reference
```

#### Public Member Functions

- ProcessHandleApplicInterface (const ProblemDescDB & problem_db)  
  constructor
- ~ProcessHandleApplicInterface ()
  destructor

#### Protected Member Functions

- int synchronous_local_analysis (int analysis_id)
- void init_communicators_checks (int max_eval_concurrency)
- void set_communicators_checks (int max_eval_concurrency)
- void map_bookkeeping (pid_t pid, int fn_eval_id)
  bookkeeping of process and evaluation ids for asynchronous maps
- pid_t create_evaluation_process (bool block_flag)
- virtual pid_t create_analysis_process (bool block_flag, bool new_group)=0
  spawn a child process for an analysis component within an evaluation
• virtual size_t wait_local_analyses ()=0
  wait for asynchronous analyses on the local processor, completing at least one job
• virtual size_t test_local_analyses_send (int analysis_id)=0
  test for asynchronous analysis completions on the local processor and return results for any completions by sending messages
• virtual void join_evaluation_process_group (bool new_group)
  create (if new_group) and join the process group for async evaluations
• virtual void join_analysis_process_group (bool new_group)
  create (if new_group) and join the process group for async analyses
• virtual void evaluation_process_group_id (pid_t pgid)
  set evalProcGroupId
• virtual pid_t evaluation_process_group_id () const
  return evalProcGroupId
• virtual void analysis_process_group_id (pid_t pgid)
  set analysisProcGroupId
• virtual pid_t analysis_process_group_id () const
  return analysisProcGroupId
• void process_local_evaluation (PRPQueue &prp_queue, const pid_t pid)
  Common processing code used by \{wait,test\} local evaluations.
• void check_wait (pid_t pid, int status)
  check the exit status of a forked process and abort if an error code was returned
• void asynchronous_local_analyses (int start, int end, int step)
  execute analyses asynchronously on the local processor
• void serve_analyses_asynch ()
  serve the analysis scheduler and execute analysis jobs asynchronously
• void ifilter_argument_list ()
  set argList for execution of the input filter
• void ofilter_argument_list ()
  set argList for execution of the output filter
• void driver_argument_list (int analysis_id)
  set argList for execution of the specified analysis driver
• void create_command_arguments (boost::shared_array< const char * > &av, StringArray &driver_and_args)
  parse argList into argument array av suitable for passing to execvp, appending parameters and results filenames if requested by commandLineArgs

Protected Attributes

• std::map< pid_t, int > evalProcessIdMap
  map of fork process id’s to function evaluation id’s for asynchronous evaluations
• std::map< pid_t, int > analysisProcessIdMap
  map of fork process id’s to analysis job id’s for asynchronous analyses
• std::vector< std::string > argList
  an array of strings for use with execvp(const char *, char * const *). These are converted to an array of const char*’s in fork_program().
14.195.1 Detailed Description

Derived application interface class that spawns a simulation code using a separate process, receives a process identifier, and communicates with the spawned process through files.

ProcessHandleApplicInterface is subclassed for fork/execvp/waitpid (Unix) and spawnvp (Windows).

14.195.2 Constructor & Destructor Documentation

ProcessHandleApplicInterface ( const ProblemDescDB & problem_db ) [inline]
constructor
argList sized 3 for [driver name, input file, output file]

14.195.3 Member Function Documentation

int synchronous_local_analysis ( int analysis_id ) [inline], [protected], [virtual]
This code provides the derived function used by ApplicationInterface::serve_analyses_synch() as well as a convenience function for ProcessHandleApplicInterface::synchronous_local_analyses() below.
Reimplemented from ApplicationInterface.
References ProcessHandleApplicInterface::create_analysis_process(), and ProcessHandleApplicInterface::driver_argument_list().

void init_communicators_checks ( int max_eval_concurrency ) [inline], [protected], [virtual]
No derived interface plug-ins, so perform construct-time checks. However, process init issues as warnings since some contexts (e.g., HierarchSurrModel) initialize more configurations than will be used.
Reimplemented from ApplicationInterface.
References ApplicationInterface::check_multiprocessor_analysis(), and ApplicationInterface::check_multiprocessor_asynchronous().

void set_communicators_checks ( int max_eval_concurrency ) [inline], [protected], [virtual]
Process run-time issues as hard errors.
Reimplemented from ApplicationInterface.
References Dakota::abort_handler(), ApplicationInterface::check_multiprocessor_analysis(), and ApplicationInterface::check_multiprocessor_asynchronous().

pid_t create_evaluation_process ( bool block_flag ) [protected], [virtual]
Manage the input filter, 1 or more analysis programs, and the output filter in blocking or nonblocking mode as governed by block_flag. In the case of a single analysis and no filters, a single fork is performed, while in other cases, an initial fork is reforked multiple times. Called from derived_map() with block_flag == BLOCK and from derived_map_asynch() with block_flag == FALL_THROUGH. Uses create_analysis_process() to spawn individual program components within the function evaluation.
Implements ProcessApplicInterface.
References Dakota::abort_handler(), ProcessHandleApplicInterface::analysis_process_group_id(), ApplicationInterface::analysisServerId, ApplicationInterface::asynchLocalAnalysisConcurrency, ApplicationInterface::asynchLocalAnalysisFlag, ProcessHandleApplicInterface::asynchronous_local_analyses(), ParallelLibrary::barrier_e(), ProcessApplicInterface::commandLineArgs, ProcessHandleApplicInterface::create_analysis_process(), ProcessHandleApplicInterface::driver_argument_list(), ApplicationInterface::eaDedMasterFlag, ApplicationInterface::evalCommRank,
ApplicationInterface::evalCommSize, ProcessHandleApplicInterface::evalProcessIdMap, ProcessHandleApplicInterface::evaluation_process_group_id(), ProcessHandleApplicInterface::ifilter_argument_list(), ProcessApplicInterface::iFilterName, ProcessHandleApplicInterface::join_evaluation_process_group(), ApplicationInterface::master_dynamic_schedule_analyses(), ProcessApplicInterface::multipleParamsFiles, ApplicationInterface::numAnalysisDrivers, ProcessHandleApplicInterface::ofilter_argument_list(), ProcessApplicInterface::oFilterName, ApplicationInterface::parallelLib, ProcessApplicInterface::paramsFileName, ProcessApplicInterface::programNames, ProcessApplicInterface::resultsFileName, ProcessHandleApplicInterface::serve_analyses_asynch(), ApplicationInterface::serve_analyses_synch(), Dakota::substitute_params_and_results(), ApplicationInterface::suppress_Output, and ProcessApplicInterface::synchronous_local_analyses().

void check_wait ( pid_t pid, int status ) [protected]
check the exit status of a forked process and abort if an error code was returned
Check to see if the process terminated abnormally (WIFEXITED(status)==0) or if either execvp or the application returned a status code of -1 (WIFEXITED(status)!=0 && (signed char)WEXITSTATUS(status)==-1). If one of these conditions is detected, output a failure message and abort. Note: the application code should not return a status code of -1 unless an immediate abort of dakota is wanted. If for instance, failure capturing is to be used, the application code should write the word "FAIL" to the appropriate results file and return a status code of 0 through exit(). References Dakota::abort_handler().
Referenced by ForkApplicInterface::create_analysis_process(), SpawnApplicInterface::test_local_analyses_send(), SpawnApplicInterface::test_local_evaluation_sequence(), ForkApplicInterface::wait(), SpawnApplicInterface::wait_local_analyses(), and SpawnApplicInterface::wait_local_evaluation_sequence().

void asynchronous_local_analyses ( int start, int end, int step ) [protected]
execute analyses asynchronously on the local processor Schedule analyses asynchronously on the local processor using a dynamic scheduling approach (start to end in step increments). Concurrency is limited by asynchLocalAnalysisConcurrency. Modeled after ApplicationInterface::asynchronous_local_evaluations(). NOTE: This function should be elevated to ApplicationInterface if and when another derived interface class supports asynchronous local analyses. References Dakota::abort_handler(), ProcessHandleApplicInterface::analysisProcessIdMap, ApplicationInterface::asynchLocalAnalysisConcurrency, ProcessHandleApplicInterface::create_analysis_process(), ProcessHandleApplicInterface::driver_argument_list(), ApplicationInterface::numAnalysisDrivers, and ProcessHandleApplicInterface::wait_local_analyses().
Referenced by ProcessHandleApplicInterface::create_evaluation_process().

void serve_analyses_asynch ( ) [protected]
serve the analysis scheduler and execute analysis jobs asynchronously
This code runs multiple asynch analyses on each server. It is modeled after ApplicationInterface::serve_evaluations_asynch(). NOTE: This fn should be elevated to ApplicationInterface if and when another derived interface class supports hybrid analysis parallelism.
References Dakota::abort_handler(), ProcessHandleApplicInterface::analysisProcessIdMap, ApplicationInterface::asynchLocalAnalysisConcurrency, ProcessHandleApplicInterface::create_analysis_process(), ProcessHandleApplicInterface::driver_argument_list(), ParallelLibrary::irecv_ea(), ApplicationInterface::numAnalysisDrivers, ApplicationInterface::parallelLib, ParallelLibrary::recv_ea(), ParallelLibrary::test(), and ProcessHandleApplicInterface::test_local_analyses_send().
Referenced by ProcessHandleApplicInterface::create_evaluation_process().
void create_command_arguments ( boost::shared_array<const char *> & av, StringArray & driver_and_args ) [protected]

parse argList into argument array av suitable for passing to execvp, appending parameters and results filenames if requested by commandLineArgs.

This function will split the analysis command in argList[0] based on whitespace, but preserve spaces within quoted strings, such that quoted strings can be passed as single command arguments. NOTE: This function allocates memory in av that might be implicitly freed when the child exits (control never returns to caller). driver_and_args needs to be a return argument because av will contain pointers into its c_str()'s when done.

References ProcessHandleApplicInterface::argList, ProcessApplicInterface::commandLineArgs, Dakota::substitute_params_and_results(), and WorkdirHelper::tokenize_driver().

Referenced by SpawnApplicInterface::create_analysis_process(), and ForkApplicInterface::create_analysis_process().

The documentation for this class was generated from the following files:

• ProcessHandleApplicInterface.hpp
• ProcessHandleApplicInterface.cpp

14.196 ProgramOptions Class Reference

ProgramOptions stores options whether from the CLH or from library user; initially valid only on worldRank = 0, but then broadcast in ParallelLibrary::push_output_tag()

Public Member Functions

• ProgramOptions ()
  default constructor (needed for default environment ctors and could be used by library clients to late update data)
• ProgramOptions (int world_rank)
  constructor that accepts world rank to help with I/O control; allows default constructed ProgramOptions to get rank in library mode
• ProgramOptions (int argc, char *argv[], int world_rank)
  standard constructor that uses a CommandLineHandler to parse user options
• const String & input_file () const
  Dakota input file base name (no tag)
• const String & input_string () const
  alternate Dakota input string literal; also set when input is read from stdin
• bool echo_input () const
  is input echo specified?
• bool preproc_input () const
  pre-process input file
• const String & preproc_cmd () const
  pre-processing command, possibly specifying another tool
• const String & parser_options () const
  (deprecated) NIDR parser options
• String output_file () const
  output (user-provided or default) file base name (no tag)
• const String & error_file () const
error file base name (no tag)

- const String & exit_mode () const
  behavior of abort_handler (throw or exit)

- const String & read_restart_file () const
  restart file base name (no tag)

- size_t stop_restart_evals () const
  eval ID at which to stop reading restart

- String write_restart_file () const
  write restart (user-provided or default) file base name (no tag)

- bool help () const
  is help mode active?

- bool version () const
  is version mode active?

- bool check () const
  is check mode active?

- bool pre_run () const
  is pre-run mode active?

- bool run () const
  is run mode active?

- bool post_run () const
  is post-run mode active?

- bool user_modes () const
  are any non-default, user-specified run modes active?

- const String & pre_run_input () const
  filename for pre-run input

- const String & pre_run_output () const
  filename for pre-run output

- const String & run_input () const
  filename for run input

- const String & run_output () const
  filename for run output

- const String & post_run_input () const
  filename for post-run input

- const String & post_run_output () const
  filename for post-run output

- unsigned int pre_run_output_format () const
  tabular format for pre-run output

- unsigned int post_run_input_format () const
  tabular format for post-run input

- bool proceed_to_instantiate () const
  whether steps beyond help/version are requested (instantiation required)

- bool proceed_to_run () const
  Whether steps beyond check are requested.
• bool user_stdout_redirect () const
  whether the user/client code requested a redirect of stdout
• bool user_stderr_redirect () const
  whether the user/client code requested a redirect of stderr
• void world_rank (int world_rank)
  set the world rank to govern early conditional output
• void input_file (const String &in_file)
  set Dakota input file base name (no tag)
• void input_string (const String &in_string)
  set alternate Dakota input string literal
• void echo_input (bool echo_flag)
  set whether to echo input to output
• void preproc_input (bool pp_flag)
  set whether to pre-process input file
• void preproc_cmd (const String &pp_cmd)
  set alternate pre-processing command
• void exit_mode (const String &mode)
  set behavior for abort_handler
• void output_file (const String &out_file)
  set base file name for Dakota output
• void error_file (const String &err_file)
  set base file name for Dakota errors
• void read_restart_file (const String &read_rst)
  set base file name for restart file from which to read
• void stop_restart_evals (size_t stop_rst)
  set eval ID at which to stop reading restart
• void write_restart_file (const String &write_rst)
  set base file name for restart file to write
• void help (bool help_flag)
  set true to print help information and exit
• void version (bool version_flag)
  set true to print version information and exit
• void check (bool check_flag)
  set true to check input and instantiate objects, then exit
• void pre_run (bool pre_run_flag)
  set to enable/disable pre-run phase
• void run (bool run_flag)
  set to enable/disable run phase
• void post_run (bool post_run_flag)
  set to enable/disable post-run phase
• void pre_run_input (const String &pre_run_in)
  Specify the pre-run phase input filename.
• void pre_run_output (const String &pre_run_out)
Specify the pre-run phase output filename.

- void run_input (const String &run_in)
  Specify the run phase input filename.

- void run_output (const String &run_out)
  Specify the run phase output filename.

- void post_run_input (const String &post_run_in)
  Specify the post-run phase input filename.

- void post_run_output (const String &post_run_out)
  Specify the post-run phase output filename.

- void parse (const ProblemDescDB &problem_db)
  Extract environment options from ProblemDescDB.

- void read (MPIUnpackBuffer &s)
  Helper function for reading some class data from MPI buffer

- void write (MPIPackBuffer &s) const
  Helper function for writing some class data to MPI buffer

Private Member Functions

- void parse_environment_options ()
  Any environment variables affecting global behavior get read here

- void manage_run_modes (const CommandLineHandler &clh)
  Retrieve run mode options from the CLH

- void split_filenames (const char *filenames, std::string &input_filename, std::string &output_filename)
  Manage pre/run/post filenames

- void validate ()
  Verify consistency of user settings (helpful for library mode especially)

- void validate_run_modes ()
  Validate user run modes and set userModesFlag

- void set_option (const ProblemDescDB &problem_db, const String &db_name, String &data_member)
  Retrieve environment.db_name from the problem db and update data_member, warning if needed

Private Attributes

- int worldRank
  Cache the world rank to help with conditional output

- String inputFilename
  Dakota input file name, e.g., “dakota.in”.

- String inputString
  Alternate input means for library clients: input string (mutually exclusive with input file)

- bool echoInput
  Whether to echo client’s input file at parse

- bool preprocInput
  Whether to pre-process input with pyprepro/etc.

- String preprocCmd
pre-processing command (default pyprepro.py)

- **String parserOptions**
  
  Deprecated option for NIDR parser options.

- **String exitMode**
  
  Abort or throw on error.

- **String outputFile**
  
  Dakota output base file name, e.g., "dakota.out".

- **String errorFile**
  
  Dakota error base file name, e.g., "dakota.err".

- **String readRestartFile**
  
  e.g., "dakota.old.rst"

- **int stopRestartEvals**
  
  eval number at which to stop restart read

- **String writeRestartFile**
  
  e.g., "dakota.new.rst"

- **bool helpFlag**
  
  whether to print help message and exit

- **bool versionFlag**
  
  whether to print version message and exit

- **bool checkFlag**
  
  flags invocation with command line option -check

- **bool preRunFlag**
  
  flags invocation with command line option -pre_run

- **bool runFlag**
  
  flags invocation with command line option -run

- **bool postRunFlag**
  
  flags invocation with command line option -post_run

- **bool userModesFlag**
  
  whether any user run modes are active

- **String preRunInput**
  
  filename for pre_run input

- **String preRunOutput**
  
  filename for pre_run output

- **String runInput**
  
  filename for run input

- **String runOutput**
  
  filename for run output

- **String postRunInput**
  
  filename for post_run input

- **String postRunOutput**
  
  filename for post_run output

- **unsigned short preRunOutputFormat**
  
  tabular format for pre_run output

- **unsigned short postRunInputFormat**
  
  tabular format for post_run input
14.196.1 Detailed Description

ProgramOptions stores options whether from the CLH or from library user; initially valid only on worldRank = 0, but then broadcast in ParallelLibrary::push_output_tag().

14.196.2 Member Function Documentation

void split_filenames ( const char *filenames, std::string &input_filename, std::string &output_filename )
[private]
manage pre/run/post filenames
Tokenize colon-delimited input and output filenames, returns unchanged strings if tokens not found.
Referenced by ProgramOptions::manage_run_modes().
The documentation for this class was generated from the following files:

- ProgramOptions.hpp
- ProgramOptions.cpp

14.197 PStudyDACE Class Reference

Base class for managing common aspects of parameter studies and design of experiments methods.

Inheritance diagram for PStudyDACE:

```
  Iterator
   |     |
   |     |
  Analyzer
   |     |
   |     |
PStudyDACE

```

Public Member Functions

- bool resize ()
  reinitializes iterator based on new variable size

Protected Member Functions

- PStudyDACE (ProblemDescDB &problem_db, Model &model)
  constructor
- PStudyDACE (unsigned short method_name, Model &model)
  alternate constructor for instantiations "on the fly"
- ~PStudyDACE ()
  destructor
- void print_results (std::ostream &s, short results_state=FINAL_RESULTS)
  print the final iterator results
• void volumetric_quality (int ndim, int num_samples, double *sample_points)
  
  Calculation of volumetric quality measures.

Protected Attributes

• SensAnalysisGlobal pStudyDACESensGlobal
  
  initialize statistical post processing
• bool volQualityFlag
  
  flag which specifies evaluation of volumetric quality measures
• bool varBasedDecompFlag
  
  flag which specifies calculating variance based decomposition sensitivity analysis metrics

Private Attributes

• double chiMeas
  
  quality measure
• double dMeas
  
  quality measure
• double hMeas
  
  quality measure
• double tauMeas
  
  quality measure

Additional Inherited Members

14.197.1 Detailed Description

Base class for managing common aspects of parameter studies and design of experiments methods.

The PStudyDACE base class manages common data and functions, such as those involving the best solutions located during the parameter set evaluations or the printing of final results.

14.197.2 Member Function Documentation

void print_results ( std::ostream & s, short results_state = FINAL_RESULTS ) [protected], [virtual]

print the final iterator results

This virtual function provides additional iterator-specific final results outputs beyond the function evaluation summary printed in finalize_run().

Reimplemented from Analyzer.

References PStudyDACE::chiMeas, Analyzer::compactMode, Model::continuous_variable_labels(), SensAnalysisGlobal::correlations_computed(), Model::discrete_int_variable_labels(), Model::discrete_real_variable_labels(), Model::discrete_string_variable_labels(), PStudyDACE::dMeas, PStudyDACE::hMeas, Iterator::iteratedModel, Analyzer::numLSqTerms, Analyzer::numObjFns, SensAnalysisGlobal::print_correlations(), Analyzer::print_results(), Analyzer::print_sobol_indices(), PStudyDACE::pStudyDACESensGlobal, Model::response_labels(), PStudyDACE::tauMeas, PStudyDACE::varBasedDecompFlag, and PStudyDACE::volQualityFlag.
void volumetric_quality ( int ndim, int num_samples, double * sample_points ) [protected]

Calculation of volumetric quality measures.
Calculation of volumetric quality measures developed by FSU.
References PStudyDACE::chiMeas, PStudyDACE::dMeas, PStudyDACE::hMeas, and PStudyDACE::tauMeas.
Referenced by FSUDesignCompExp::get_parameter_sets(), and DDACEDesignCompExp::get_parameter_sets().
The documentation for this class was generated from the following files:

- DakotaPStudyDACE.hpp
- DakotaPStudyDACE.cpp

14.198 PSUADEDesignCompExp Class Reference

Wrapper class for the PSUADE library.

Inheritance diagram for PSUADEDesignCompExp:

```
  PSUADEDesignCompExp
  |     |
  v     v
Iterator

Analyzer

PStudyDACE

PSUADEDesignCompExp
```

**Public Member Functions**

- PSUADEDesignCompExp (ProblemDescDB &problem_db, Model &model)
  primary constructor for building a standard DACE iterator
- ~PSUADEDesignCompExp ()
  destructor
- bool resize ()
  reinitializes iterator based on new variable size

**Protected Member Functions**

- void pre_run ()
  pre-run portion of run (optional); re-implemented by Iterators which can generate all Variables (parameter sets) a priori
- void post_input ()
  read tabular data for post-run mode
- void core_run ()
  core portion of run; implemented by all derived classes and may include pre/post steps in lieu of separate pre/post
- void post_run (std::ostream &s)
CHAPTER 14. CLASS DOCUMENTATION

post-run portion of run (optional); verbose to print results; re-implemented by Iterators that can read all Variables/Responses and perform final analysis phase in a standalone way

- int num_samples () const
- void sampling_reset (int min_samples, bool all_data_flag, bool stats_flag)
  reset sampling iterator to use at least min_samples
- unsigned short sampling_scheme () const
  return sampling name
- void vary_pattern (bool pattern_flag)
  sets varyPattern in derived classes that support it
- void get_parameter_sets (Model &model)
  Generate one block of numSamples samples (ndim * num_samples), populating allSamples; ParamStudy is the only class that specializes to use allVariables.

Private Member Functions

- void enforce_input_rules ()
  enforce sanity checks/modifications for the user input specification

Private Attributes

- int samplesSpec
  initial specification of number of samples
- int numSamples
  current number of samples to be evaluated
- const UShortArray & varPartitionsSpec
  number of partitions in each variable direction
- int numPartitions
  number of partitions to pass to PSUADE (levels = partitions + 1)
- bool allDataFlag
  flag which triggers the update of allVars/allResponses for use by Iterator::all_variables() and Iterator::all_responses()
- size_t numDACERuns
  counter for number of executions for this object
- bool varyPattern
  flag for generating a sequence of seed values within multiple get_parameter_sets() calls so that the sample sets are not repeated, but are still repeatable
- const int seedSpec
  the user seed specification for the random number generator (allows repeatable results)
- int randomSeed
  current seed for the random number generator
Additional Inherited Members

14.198.1 Detailed Description

Wrapper class for the PSUADE library.

The PSUADEDesignCompExp class provides a wrapper for PSUADE, a C++ design of experiments library from Lawrence Livermore National Laboratory. Currently this class only includes the PSUADE Morris One-at-a-time (MOAT) method to uniformly sample the parameter space spanned by the active bounds of the current Model. It returns all generated samples and their corresponding responses as well as the best sample found.

14.198.2 Constructor & Destructor Documentation

PSUADEDesignCompExp ( ProblemDescDB & problem, Model & model )

primary constructor for building a standard DACE iterator

This constructor is called for a standard iterator built with data from probDescDB.

References Dakota::abort_handler(), Iterator::maxEvalConcurrency, Iterator::method_string(), Iterator::method-Name, Analyzer::numDiscreteIntVars, Analyzer::numDiscreteRealVars, Analyzer::numDiscreteStringVars, and PSUADEDesignCompExp::numSamples.

14.198.3 Member Function Documentation

void pre_run ( ) [protected], [virtual]

pre-run portion of run (optional); re-implemented by Iterators which can generate all Variables (parameter sets) a priori

pre-run phase, which a derived iterator may optionally reimplement; when not present, pre-run is likely integrated into the derived run function. This is a virtual function; when re-implementing, a derived class must call its nearest parent's pre_run(), if implemented, typically before performing its own implementation steps.

Reimplemented from Analyzer.

References PSUADEDesignCompExp::enforce_input_rules(), PSUADEDesignCompExp::get_parameter_sets(), Iterator::iteratedModel, and Analyzer::pre_run().

void core_run ( ) [protected], [virtual]

core portion of run; implemented by all derived classes and may include pre/post steps in lieu of separate pre/post Virtual run function for the iterator class hierarchy. All derived classes need to redefine it.

Reimplemented from Iterator.

References Analyzer::evaluate_parameter_sets(), Iterator::iteratedModel, Analyzer::numLSqTerms, and Analyzer-::numObjFns.

void post_run ( std::ostream & s ) [protected], [virtual]

post-run portion of run (optional); verbose to print results; re-implemented by Iterators that can read all Variables/-Responses and perform final analysis phase in a standalone way

Post-run phase, which a derived iterator may optionally reimplement; when not present, post-run is likely integrated into run. This is a virtual function; when re-implementing, a derived class must call its nearest parent's post_run(), typically after performing its own implementation steps.

Reimplemented from Analyzer.
References Dakota::abort_handler(), Analyzer::allResponses, Analyzer::allSamples, Model::continuous_lower_bounds(), Model::continuous_upper_bounds(), PSUADEDesignCompExp::enforce_input_rules(), Iterator::iterated_Model, Analyzer::numContinuousVars, Analyzer::numFunctions, PSUADEDesignCompExp::numSamples, and Analyzer::post_run().

```cpp
int num_samples() const [inline], [protected], [virtual]
```

Return current number of evaluation points. Since the calculation of samples, collocation points, etc. might be costly, provide a default implementation here that backs out from the maxEvalConcurrency.

Reimplemented from Analyzer.
References PSUADEDesignCompExp::numSamples.

```cpp
void enforce_input_rules() [private]
```

even sanity checks/modifications for the user input specification

Users may input a variety of quantities, but this function must enforce any restrictions imposed by the sampling algorithms.

References Analyzer::numContinuousVars, PSUADEDesignCompExp::numPartitions, PSUADEDesignComp-Exp::numSamples, and PSUADEDesignCompExp::varPartitionsSpec.

Referenced by PSUADEDesignCompExp::post_input(), PSUADEDesignCompExp::post_run(), and PSUADEDesignCompExp::pre_run().

The documentation for this class was generated from the following files:

- PSUADEDesignCompExp.hpp
- PSUADEDesignCompExp.cpp

### 14.199 PythonInterface Class Reference

Inheritance diagram for PythonInterface:

```
  Interface
  ApplicationInterface
    DirectApplicInterface
        PythonInterface
```

**Public Member Functions**

- **PythonInterface** (const ProblemDescDB &problem_db)
  * constructor

- **~PythonInterface** ()
  * destructor
14.199. PYTHONINTERFACE CLASS REFERENCE

Protected Member Functions

- virtual int derived_map_ac (const String &ac_name)
  
  *execute an analysis code portion of a direct evaluation invocation*

- int python_run (const String &ac_name)
  
  *direct interface to Python via API. BMA 07/02/07*

- template<class ArrayT, class Size>
  
  bool python_convert_int (const ArrayT &src, Size size, PyObject **dst)
  
  *convert arrays of integer types to Python list or numpy array*

- bool python_convert (const RealVector &src, PyObject **dst)
  
  *convert RealVector to Python list or numpy array*

- bool python_convert (const RealVector &c_src, const IntVector &di_src, const RealVector &dr_src, PyObject **dst)
  
  *convert RealVector + IntVector + RealVector to Python mixed list or numpy double array*

- template<class StringArrayT>
  
  bool python_convert_strlist (const StringArrayT &src, PyObject **dst)
  
  *convert labels*

- bool python_convert (const StringMultiArray &c_src, const StringMultiArray &di_src, const StringMultiArray &dr_src, PyObject **dst)
  
  *convert all labels to single list*

- bool python_convert (PyObject *pyv, RealVector &rv, const int &dim)
  
  *convert python [list of int or float] or [numpy array of double] to RealVector (for fns)*

- bool python_convert (PyObject *pyv, double *rv, const int &dim)
  
  *convert python [list of int or float] or [numpy array of double] to double[], for use as helper in converting gradients*

- bool python_convert (PyObject *pym, RealMatrix &rm)
  
  *convert python [list of lists of int or float] or [numpy array of dbl] to RealMatrix (for gradients)*

- bool python_convert (PyObject *pym, RealSymMatrix &rm)
  
  *convert python [list of lists of int or float] or [numpy array of dbl] to RealMatrix (used as helper in Hessian conversion)*

- bool python_convert (PyObject *pyma, RealSymMatrixArray &rma)
  
  *convert python [list of lists of lists of int or float] or [numpy array of double] to RealSymMatrixArray (for Hessians)*

Protected Attributes

- bool userNumpyFlag
  
  *whether the user requested numpy data structures in the input file*

- bool ownPython
  
  *true if this class created the interpreter instance*

14.199.1 Detailed Description

Specialization of DirectApplicInterface to link to Python analysis drivers. Includes convenience functions to map data to/from Python.
### 14.199.2 Member Function Documentation

```cpp
int derived_map_ac ( const String & ac_name ) [protected], [virtual]
```

execute an analysis code portion of a direct evaluation invocation

- Python specialization of derived analysis components.
- References ApplicationInterface::analysisServerId, and PythonInterface::python_run().

```cpp
bool python_convert_int ( const ArrayT & src, Size sz, PyObject ** dst ) [protected]
```

convert arrays of integer types to Python list or numpy array
- convert all integer array types including IntVector, ShortArray, and SizetArray to Python list of ints or numpy array of ints
- References PythonInterface::userNumpyFlag.
- Referenced by PythonInterface::python_run().

The documentation for this class was generated from the following files:

- PythonInterface.hpp
- PythonInterface.cpp

### 14.200 QMEApproximation Class Reference

Derived approximation class for QMEA Quadratic Multipoint Exponential Approximation (a multipoint approximation).

Inheritance diagram for QMEApproximation:

```
Approximation

QMEApproximation
```

#### Public Member Functions

- **QMEApproximation ()**
  
  *default constructor*

- **QMEApproximation (ProblemDescDB &problem_db, const SharedApproxData &shared_data, const String &approx_label)**
  
  *standard constructor*

- **QMEApproximation (const SharedApproxData &shared_data)**
  
  *alternate constructor*

- **~QMEApproximation ()**
  
  *destructor*
Protected Member Functions

- `int min_coefficients() const`
  
  return the minimum number of samples (unknowns) required to build the derived class approximation type in num- Vars dimensions

- `void build()`
  
  builds the approximation from scratch

- `Real value(const Variables &vars)`
  
  retrieve the approximate function value for a given parameter vector

- `const RealVector & gradient(const Variables &vars)`
  
  retrieve the approximate function gradient for a given parameter vector

- `void clear_current_active_data()`

Private Member Functions

- `void find_scaled_coefficients()`
  
  compute TANA coefficients based on scaled inputs

- `void offset(const RealVector &x, RealVector &s)`
  
  based on minX, apply offset scaling to x to define s

- `Real apxfn_value(const RealVector &)`

Private Attributes

- `RealVector pExp`
  
  vector of exponent values

- `RealVector minX`
  
  vector of minimum parameter values used in scaling

- `RealVector scX1`
  
  vector of scaled x1 values

- `RealVector scX2`
  
  vector of scaled x2 values

- `Real H`
  
  the scalar Hessian value in the TANA-3 approximation

- `RealVector beta`
  
  vector of QMEA reduced space diagonal Hessian coefficients

- `RealMatrix G_reduced_xfm`
  
  Grahm-Schmidt orthonormal reduced subspace transformation.

- `size_t numUsed`
  
  number of previous data points used (size of reduced subspace)

- `size_t currGradIndex`
  
  index of current expansion point with gradients

- `size_t prevGradIndex`
  
  index of most recent previous point with gradients
Additional Inherited Members

14.200.1 Detailed Description

Derived approximation class for QMEA Quadratic Multipoint Exponential Approximation (a multipoint approximation).

The QMEApproximation class provides a multipoint approximation based on matching value and gradient data from multiple points (typically the current and previous iterates) in parameter space. It forms an exponential approximation in terms of intervening variables.

14.200.2 Member Function Documentation

void build ( ) [protected], [virtual]

builds the approximation from scratch

This is the common base class portion of the virtual fn and is insufficient on its own; derived implementations should explicitly invoke (or reimplement) this base class contribution.

Reimplemented from Approximation.

References Dakota::NPOS, Dakota::abort_handler(), Approximation::approxData, Approximation::build(), QMEApproximation::currGradIndex, QMEApproximation::find_scaled_coefficients(), Dakota::length(), QMEApproximation::minX, SharedApproxData::numVars, QMEApproximation::pExp, QMEApproximation::prevGradIndex, and Approximation::sharedDataRep.

void clear_current_active_data ( ) [inline], [protected], [virtual]

Redefine default implementation to support history mechanism.

Reimplemented from Approximation.

References Dakota::NPOS, Approximation::approxData, QMEApproximation::currGradIndex, SharedApproxData::numVars, QMEApproximation::prevGradIndex, and Approximation::sharedDataRep.

The documentation for this class was generated from the following files:

- QMEApproximation.hpp
- QMEApproximation.cpp

14.201 QuesoJointPdf< V, M > Class Template Reference

Dakota specialization of QUESO generic joint PDF.

Inherits BaseJointPdf< V, M >.

Public Member Functions

- QuesoJointPdf (const char *prefix, const QUESO::VectorSet< V, M >&domainSet, NonDQUESOBayesCalibration *nond_queso_ptr)

  Default constructor.

- virtual ~QuesoJointPdf ()

  Destructor.

- double actualValue (const V &domainVector, const V *domainDirection, V *gradVector, M *hessianMatrix, V *hessianEffect) const

  Actual value of the PDF (scalar function).
• double lnValue (const V &domainVector, const V *domainDirection, V *gradVector, M *hessianMatrix, V *hessianEffect) const
  Logarithm of the value of the function.
• double computeLogOfNormalizationFactor (unsigned int numSamples, bool m_logOfNormalizationFactor) const
  Computes the logarithm of the normalization factor.
• void distributionMean (V &meanVector) const
  Mean value of underlying random variable.
• void distributionVariance (M &covMatrix) const
  Covariance of the underlying random variable.

Private Attributes
• NonDQUESOBayesCalibration * nonDQUESOInstance
  pointer to QUESO instance for PDF evaluation callbacks

14.201.1 Detailed Description
template<class V, class M> class Dakota::QuesoJointPdf< V, M >
Dakota specialization of QUESO generic joint PDF.

14.201.2 Constructor & Destructor Documentation
QuesoJointPdf ( const char * prefix, const QUESO::VectorSet< V, M > & domainSet, NonDQUESOBayesCalibration * nond_queso_ptr )
Default constructor.
  Instantiates an object of the class, i.e. a scalar function, given a prefix and its domain.

14.201.3 Member Function Documentation
void distributionMean ( V & meanVector ) const
Mean value of underlying random variable.
  Assumes meanVector is sized

void distributionVariance ( M & covMatrix ) const
Covariance of the underlying random variable.
  Assumes covMatrix is sized
  The documentation for this class was generated from the following files:
  • QUESOImpl.hpp
  • QUESOImpl.cpp

14.202 QuesoVectorRV< V, M > Class Template Reference
Dakota specialization of QUESO vector-valued random variable.
  Inherits BaseVectorRV< V, M >.
Public Member Functions

- **QuesoVectorRV** (const char *prefix, const QUESO::VectorSet< V, M > &imageSet, NonDQUESOBayesCalibration *nond_queso_ptr)
  
  Default constructor.

- virtual ~QuesoVectorRV ()
  
  Virtual destructor.

- void print (std::ostream &os) const
  
  TODO: Prints the vector RV (required pure virtual).

### 14.202.1 Detailed Description

```cpp
template<class V, class M>
class Dakota::QuesoVectorRV< V, M >
```

Dakota specialization of QUESO vector-valued random variable.

### 14.202.2 Constructor & Destructor Documentation

**QuesoVectorRV** ( const char * prefix, const QUESO::VectorSet< V, M > & imageSet, NonDQUESOBayesCalibration * nond_queso_ptr )

Default constructor.

Constructs a generic queso vector RV, given a prefix and the image set of the vector RV.

The documentation for this class was generated from the following files:

- QUESOImpl.hpp
- QUESOImpl.cpp

### 14.203 RandomFieldModel Class Reference

Random field model, capable of generating and then forward propagating.

Inheritance diagram for RandomFieldModel:

```
Model

RecastModel

RandomFieldModel
```

**Public Member Functions**

- **RandomFieldModel** (ProblemDescDB &problem_db)
  
  Problem database constructor.

- ~RandomFieldModel ()
  
  Destructor

- bool initialize_mapping (ParLevLIter pl_iter)
for KL models, the model is augmented with the random coeffs of the KL

- bool resize_pending() const
  - return true if a potential resize is still pending, such that sizing-based initialization should be deferred

### Protected Member Functions

- void assign_instance()
  - assign static pointer instance to this for use in static transformation functions
- Model get_sub_model(ProblemDescDB &problem_db)
  - retrieve the sub-Model from the DB to pass up the constructor chain
- void init_dace_iterator(ProblemDescDB &problem_db)
  - initialize the RF-generating sampler
- void validate_inputs()
  - validate the build controls and set defaults
- void get_field_data()
  - Source data generation: get the field data either from file or simulation by running the DACE Iterator. Populates rfBuildData.
- void identify_field_model()
  - Generate field representation: generate a KL or PCA/GP.
- void rf_suite_identify_field_model()
  - Generate field representation: utilize RF Suite.
- void initialize_recast()
  - Initialize the base class RecastModel with reduced space variable sizes.
- SizetArray variables_resize()
  - Create a variables components totals array with the reduced space size for continuous variables.
- void initialize_rf_coeffs()
  - For KL models, augment the subModel's uncertain variables with additional N(0,1) variables; set up mvDist for the N(0,1)'s.
- void derived_evaluate(const ActiveSet &set)
  - generate a random field realization, then evaluate the submodel
- void derived_evaluate_nowait(const ActiveSet &set)
  - generate a random field realization, then evaluate the submodel (asynch)
- void generate_kl_realization()
  - generate a KL realization and write to file
- void generate_pca_gp_realization()
  - generate a PCA/GP realization and write to file
- void write_field(const RealVector &field_prediction)
  - write a field realization to console and file

### Static Protected Member Functions

- static void vars_mapping(const Variables &recast_xi_vars, Variables &sub_model_x_vars)
  - map the active continuous recast variables to the active submodel variables (linear transformation)
- static void set_mapping(const Variables &recast_vars, const ActiveSet &recast_set, ActiveSet &sub_model_set)
  - map the inbound ActiveSet to the sub-model (map derivative variables)
Protected Attributes

- String rfDataFilename
  name of the data file with RF build data
- size_t numObservations
  rows of data matrix
- IntVector fieldLengths
  column partitions of data matrix
- RealMatrix rfBuildData
  data matrix with realizations of the random field to approximate
- RealMatrix rfBuildVars
  matrix of samples used to build the RF data
- Iterator daceIterator
  String dataDirectoryBasename;
- unsigned short expansionForm
  unsigned short analyticCovForm;
- unsigned short covarianceForm
  form of the analytic covariance function
- int requestedReducedRank
  current approximation of system rank
- Real percentVariance
  fraction of energy to capture
- int actualReducedRank
  command to run RF Suite
- ReducedBasis rfBasis
  reduced basis representation (for KL or PCA case)
- int fieldRealizationId
  counter for RF Suite
- std::vector<Approximation> gpApproximations
  approximate models used to map the uncertain vars through the PCA approx

Static Protected Attributes

- static RandomFieldModel * rfmInstance
  static pointer to this class for use in static callbacks

Additional Inherited Members

14.203.1 Detailed Description

Random field model, capable of generating and then forward propagating.

Specialization of a RecastModel that optionally identifies an approximate random field model during build phase and creates a RecastModel capable of performing forward UQ including the field and auxialliary uncertain variables reduced space. This RandomFieldModel wraps the random field propagation model (not the RF-generating model)
14.203. **RANDOMFIELDMODEL CLASS REFERENCE**

14.203.2 Member Function Documentation

bool initialize_mapping ( ParLevLIter pl_iter ) [virtual]

for KL models, the model is augmented with the random coeffs of the KL.

May eventually take on init_comms and related operations. Also may want ide of build/update like DataFit-SurrModel, eventually.

Reimplemented from Model.

References RandomFieldModel::covarianceForm, Model::estimate_message_lengths(), RandomFieldModel::expansionForm, RandomFieldModel::fieldRealizationId, RandomFieldModel::get_field_data(), RandomFieldModel::identify_field_model(), RecastModel::initialize_mapping(), RandomFieldModel::initialize_recast(), RandomFieldModel::initialize_rf_coeffs(), and RandomFieldModel::rf_suite_identify_field_model().

void get_field_data ( ) [protected]

Source data generation: get the field data either from file or simulation by running the DACE Iterator. Populates rfBuildData.

Populate rfBuildData

References Iterator::all_responses(), Iterator::all_samples(), Dakota::copy_data(), Model::cv(), RandomFieldModel::daceIterator, RandomFieldModel::expansionForm, Iterator::is_null(), Iterator::num_samples(), Model::num_Fns, Dakota::read_sized_data(), RandomFieldModel::rfBuildData, RandomFieldModel::rfBuildVars, Iterator::run(), and RecastModel::subModel.

Referenced by RandomFieldModel::initialize_mapping().

void rf_suite_identify_field_model ( ) [protected]

Generate field representation: utilize RF Suite.

Alternative to below function when using RFSuite.

References RandomFieldModel::actualReducedRank, and RandomFieldModel::requestedReducedRank.

Referenced by RandomFieldModel::initialize_mapping().

void initialize_recast ( ) [protected]

Initialize the base class RecastModel with reduced space variable sizes.

Initialize the recast model to augment the uncertain variables with actualReducedRank additional N(0,1) variables, with no response function mapping (for now).

References RandomFieldModel::actualReducedRank, Model::continuous_variable_types(), Model::current_response(), Model::cv(), Model::div(), Model::drv(), Model::dsv(), Response::function_gradients(), Response::function_hessians(), RecastModel::init_maps(), RecastModel::init_sizes(), Model::num_nonlinear_ineq_constraints(), Model::num_primary_fns(), Model::num_secondary_fns(), Model::numFns, RandomFieldModel::set_mapping(), RecastModel::subModel, RandomFieldModel::variables_resize(), and RandomFieldModel::vars_mapping().

Referenced by RandomFieldModel::initialize_mapping().

SizetArray variables_resize ( ) [protected]

Create a variables components totals array with the reduced space size for continuous variables.

Create a variables components totals array with the reduced space size for continuous variables TODO: augment normal uncVars for KL case.

References RandomFieldModel::actualReducedRank, SharedVariablesData::components_totals(), Model::current_variables(), RandomFieldModel::expansionForm, Variables::shared_data(), RecastModel::subModel, and Dakota::svd().
void initialize_rf_coeffs() [protected]
For KL models, augment the subModel’s uncertain variables with additional N(0,1) variables; set up mvDist for the N(0,1)’s.
Initialize the aleatory dist params for the KL coeffs
References RandomFieldModel::actualReducedRank, Variables::continuous_variable_label(), Model::continuous_variable_labels(), Model::currentVariables, RandomFieldModel::expansionForm, Model::multivariate_distribution(), Model::mvDist, and RecastModel::subModel.
Referenced by RandomFieldModel::initialize_mapping().

void vars_mapping(const Variables & recast_xi_vars, Variables & sub_model_x_vars) [static], [protected]
map the active continuous recast variables to the active submodel variables (linear transformation)
map the active continuous recast variables to the active submodel variables
References Variables::active_variables(), RandomFieldModel::actualReducedRank, Model::continuous_variable_types(), Variables::continuous_variables(), Model::cv(), Variables::discrete_int_variables(), Model::discrete_int_variables(), Variables::discrete_real_variables(), Model::discrete_real_variables(), Variables::discrete_string_variables(), Model::discrete_string_variables(), RandomFieldModel::expansionForm, RandomFieldModel::rfmInstance, and RecastModel::subModel.
Referenced by RandomFieldModel::initialize_recast().

14.203.3 Member Data Documentation

Iterator daceIterator [protected]
String dataDirectoryBasename;
DACE Iterator to evaluate the RF generating model
Referenced by RandomFieldModel::get_field_data(), RandomFieldModel::init_dace_iterator(), and RandomFieldModel::validate_inputs().

unsigned short expansionForm [protected]
unsigned short analyticCovForm;
form of the RF representation (KL, PCA, ICA)
Referenced by RandomFieldModel::derived_evaluate(), RandomFieldModel::derived_evaluate_nowait(), RandomFieldModel::get_field_data(), RandomFieldModel::identify_field_model(), RandomFieldModel::initialize_mapping(), RandomFieldModel::initialize_rf_coeffs(), RandomFieldModel::resize_pending(), RandomFieldModel::variables_resize(), and RandomFieldModel::vars_mapping().

int actualReducedRank [protected]
command to run RF Suite
number of bases retained in decomposition
Referenced by RandomFieldModel::generate_kl_realization(), RandomFieldModel::generate_pca_gp_realization(), RandomFieldModel::identify_field_model(), RandomFieldModel::initialize_recast(), RandomFieldModel::initialize_rf_coeffs(), RandomFieldModel::rf_suite_identify_field_model(), RandomFieldModel::variables_resize(), and RandomFieldModel::vars_mapping().
RandomFieldModel * rfInstance [static],[protected]  
static pointer to this class for use in static callbacks  
    initialization of static needed by RecastModel  
Referenced by RandomFieldModel::assign_instance(), and RandomFieldModel::vars_mapping().  
The documentation for this class was generated from the following files:  
    • RandomFieldModel.hpp  
    • RandomFieldModel.cpp  

14.204 RealScale Struct Reference  
Data structure for storing real-valued dimension scale.  

Public Member Functions  
- RealScale (const std::string & label, const RealVector & in_items, ScaleScope scope=ScaleScope::UNSARED)  
  Constructor that takes a RealVector.  
- RealScale (const std::string & label, const RealArray & in_items, ScaleScope scope=ScaleScope::UNSARED)  
  Constructor that takes a RealArray.  
- RealScale (const std::string & label, const Real * in_items, const int len, ScaleScope scope=ScaleScope::UNSHARED)  
  Constructor that takes a pointer to Real and length.  
- RealScale (const std::string & in_label, std::initializer_list<Real> in_items, ScaleScope in_scope=ScaleScope::UNSARED)  
  Constructor that takes an initializer list.  
- RealScale (const std::string &in_label, const RealVectorArray &in_items, ScaleScope in_scope=ScaleScope::::UNSARED)  
  Constructor that takes a RealVectorArray.  

Public Attributes  
- std::string label  
- ScaleScope scope  
- RealVector items  
- int numCols  
  Number of columns; equals length of scale when 1D.  
- bool isMatrix  
  2d or 1d?  

14.204.1 Detailed Description  
Data structure for storing real-valued dimension scale.  
The documentation for this struct was generated from the following file:  
- dakota_results_types.hpp
14.205 RecastModel Class Reference

Derived model class which provides a thin wrapper around a sub-model in order to recast the form of its inputs and/or outputs.

Inheritance diagram for RecastModel:

```
  Model
    RecastModel
      ActiveSubspaceModel
      AdaptedBasisModel
      DataTransformModel
      ProbabilityTransformModel
      RandomFieldModel
      ScalingModel
      WeightingModel
```

Public Member Functions


  standard (full) constructor; assumes provided sizes and map functions are final and constructs all member data

- **RecastModel** (const Model &sub_model, const SizetArray &vars_comps_totals, const BitArray &all_relax_di, const BitArray &all_relax_dr, size_t num_recast_primary_fns, size_t num_recast_secondary_fns, size_t recast secondary offset, short recast Resp_order)

  alternate constructor; uses provided sizes to construct Variables, Response and Constraints so Model can be passed to an Iterator; requires subsequent init_maps() call.

- **RecastModel** (ProblemDescDB &problem_db, const Model &sub_model)

  Problem DB-based ctor, e.g., for use in subspace model; assumes mappings to be initialized later; only initializes based on sub-model.

- **RecastModel** (const Model &sub_model)

  lightest constructor used when transform sizes aren’t known at construct time; doesn’t initialize variables and responses, so this Model can’t be used to construct an Iterator; requires subsequent init_sizes() and init_maps() calls.

- **~RecastModel** ()

  destructor

- void init_sizes (const SizetArray &vars_comps_totals, const BitArray &all_relax_di, const BitArray &all_relax_dr, size_t num_recast_primary_fns, size_t num_recast_secondary_fns, size_t recast secondary offset, short recast Resp_order)

  update recast sizes and size Variables and Response members after alternate construction

initialize recast indices and map callbacks after alternate construction

- void inverse_mappings (void(*inv_vars_map)(const Variables &sub_model_vars, Variables &recast_vars), void(*inv_set_map)(const Variables &sub_model_vars, const ActiveSet &sub_model_set, ActiveSet &recast_set), void(*inv_pri_resp_map)(const Variables &recast_vars, const Variables &sub_model_vars, const Response &recast_resp, Response &sub_model_resp), void(*inv_sec_resp_map)(const Variables &recast_vars, const Variables &sub_model_vars, const Response &recast_resp, Response &sub_model_resp))

  provide optional inverse mappings

- void transform_variables (const Variables &recast_vars, Variables &sub_model_vars)

  perform transformation of Variables (recast -> sub-model)

- void transform_set (const Variables &recast_vars, const ActiveSet &recast_set, ActiveSet &sub_model_set)

  into sub_model_set for use with subModel.

- void transform_response (const Variables &recast_vars, const Variables &sub_model_vars, const Response &sub_model_resp, Response &recast_resp)

  perform transformation of Response (sub-model -> recast)

- void transform_response_map (const IntResponseMap &old_resp_map, IntResponseMap &new_resp_map)

  invoke transform_response() on each response within old_resp_map to create new_resp_map

- void inverse_transform_variables (const Variables &sub_model_vars, Variables &recast_vars)

  perform inverse transformation of Variables (sub-model -> recast)

- void inverse_transform_set (const Variables &sub_model_vars, const ActiveSet &sub_model_set, ActiveSet &recast_set)

  into sub_model_set for use with subModel.

- void inverse_transform_response (const Variables &sub_model_vars, const Variables &recast_vars, const Response &recast_resp, Response &sub_model_resp)

  perform inverse transformation of Response (recast -> sub-model)

- void submodel_supports_derivative_estimation (bool sed_flag)

  override the submodel’s derivative estimation behavior

- String root_model_id ()

  Return the model ID of the “innermost” model. For all derived Models except RecastModels, return modelId. The RecastModel override returns the root_model_id() of the subModel.

- ActiveSet default_active_set ()

  Declare a model’s sources to the evaluationsDB.

- bool nonlinear_variables_mapping () const

  return nonlinearVarsMapping


Protected Member Functions

- Pecos::ProbabilityTransformation & probability_transformation ()

  return probability transformation employed by the Model (forwarded along to ProbabilityTransformModel recasting)

- bool initialize_mapping (ParLevLitIter pl_iter)

  Perform any global updates prior to individual evaluate() calls; returns true if the variables size has changed.

- bool finalize_mapping ()

  restore state in preparation for next initialization; returns true if the variables size has changed
CHAPTER 14. CLASS DOCUMENTATION

- `void nested_variable_mappings` (const SizetArray &c_index1, const SizetArray &di_index1, const SizetArray &ds_index1, const SizetArray &dr_index1, const ShortArray &c_target2, const ShortArray &di_target2, const ShortArray &ds_target2, const ShortArray &dr_target2)
  
  `set primaryA{C,DI,DS,DR} VarMapIndices, secondaryA{C,DI,DS,DR} VarMapTargets (coming from a higher-level NestedModel context to inform derivative est.)`

- `const SizetArray & nested_acv1_indices()` const
  
  `return primaryACVarMapIndices`

- `const ShortArray & nested_acv2_targets()` const
  
  `return secondaryACVarMapTargets`

- `short query_distribution_parameter_derivatives()` const
  
  `calculate and return derivative composition of final results w.r.t. distribution parameters (none, all, or mixed)`

- `void activate_distribution_parameter_derivatives()`
  
  `activate derivative setting w.r.t. distribution parameters`

- `void deactivate_distribution_parameter_derivatives()`
  
  `deactivate derivative setting w.r.t. distribution parameters`

- `void trans_grad_X_to_U` (const RealVector &fn_grad_x, RealVector &fn_grad_u, const RealVector &x_vars)
  
  `transform x-space gradient vector to u-space`

- `void trans_grad_U_to_X` (const RealVector &fn_grad_u, RealVector &fn_grad_x, const RealVector &x_vars)
  
  `transform u-space gradient vector to x-space`

- `void trans_grad_X_to_S` (const RealVector &fn_grad_x, RealVector &fn_grad_s, const RealVector &x_vars)
  
  `transform x-space gradient vector to gradient with respect to inserted distribution parameters`

- `void trans_hess_X_to_U` (const RealSymMatrix &fn_hess_x, RealSymMatrix &fn_hess_u, const RealVector &x_vars, const RealVector &fn_grad_x)
  
  `transform x-space Hessian matrix to u-space`

- `size_t qoi()` const
  
  `return number of unique response functions (managing any aggregations)`

- `void derived_evaluate` (const ActiveSet &set)
  
  `portion of evaluate() specific to RecastModel (forward to subModel.evaluate())`

- `void derived_evaluate_nowait` (const ActiveSet &set)
  
  `portion of evaluate_nowait() specific to RecastModel (forward to subModel.evaluate_nowait())`

- `const IntResponseMap & derived_synchronize()`
  
  `portion of synchronize() specific to RecastModel (forward to subModel.synchronize())`

- `const IntResponseMap & derived_synchronize_nowait()`
  
  `portion of synchronize_nowait() specific to RecastModel (forward to subModel.synchronize_nowait())`

- `Iterator & subordinate_iterator()`
  
  `return sub-iterator, if present, within subModel`

- `Model & subordinate_model()`
  
  `return subModel`

- `void active_model_key` (const UShortArray &mi_key)
  
  `set key in subModel`

- `void clear_model_keys()`
  
  `remove keys in subModel`

- `Model & surrogate_model()`
return surrogate model, if present, within subModel

- const Model & surrogate_model () const
  return surrogate model, if present, within subModel

- Model & truth_model ()
  return truth model, if present, within subModel

- const Model & truth_model () const
  return truth model, if present, within subModel

- void derived_subordinate_models (ModelList &ml, bool recurse_flag)
  add subModel to list and recurse into subModel

- void resize_from_subordinate_model (size_t depth=std::numeric_limits< size_t >::max())
  pass request to subModel if recursing and then resize from its results

- void update_from_subordinate_model (size_t depth=std::numeric_limits< size_t >::max())
  pass request to subModel if recursing and then update from its results

- Interface & derived_interface ()
  return subModel interface

- size_t solution_levels () const
  return size of subModel::solnControlCostMap

- void solution_level_index (unsigned short lev_index)
  activate entry in subModel::solnControlCostMap

- unsigned short solution_level_index () const
  return active entry in subModel::solnControlCostMap

- RealVector solution_level_costs () const
  return cost estimates from subModel::solnControlCostMap

- Real solution_level_cost () const
  return active cost estimate from subModel::solnControlCostMap

- void primary_response_fn_weights (const RealVector &wts, bool recurse_flag=true)
  set the relative weightings for multiple objective functions or least squares terms and optionally recurses into sub-Model

- void surrogate_function_indices (const IntSet &surr_fn_indices)
  update the subModel’s surrogate response function indices (DataFitSurrModel::surrogateFnIndices)

- void surrogate_response_mode (short mode)
  update the subModel’s surrogate response mode (SurrogateModel::responseMode)

- short correction_type ()
  retrieve subModel’s correction type

- void correction_type (short corr_type)
  update subModel’s correction type

- const RealVector & error_estimates ()
  retrieve error estimates corresponding to the subModel

- void build_approximation ()
  builds the subModel approximation

- bool build_approximation (const Variables &vars, const IntResponsePair &response_pr)
  builds the subModel approximation

- void rebuild_approximation ()
updates a subModel approximation

- **void update_approximation** (bool rebuild_flag)
  replaces data in the subModel approximation

- **void update_approximation** (const Variables &vars, const IntResponsePair &response_pr, bool rebuild_flag)
  replaces data in the subModel approximation

- **void update_approximation** (const VariablesArray &vars_array, const IntResponseMap &resp_map, bool rebuild_flag)
  replaces data in the subModel approximation

- **void append_approximation** (bool rebuild_flag)
  appends data to the subModel approximation

- **void append_approximation** (const Variables &vars, const IntResponsePair &response_pr, bool rebuild_flag)
  appends data to the subModel approximation

- **void append_approximation** (const VariablesArray &vars_array, const IntResponseMap &resp_map, bool rebuild_flag)
  appends data to the subModel approximation

- **void pop_approximation** (bool save_surr_data, bool rebuild_flag=false)
  remove the previous data set addition to a surrogate (e.g., due to a previous append_approximation() call); flag manages storing of surrogate data for use in a subsequent push_approximation()

- **void push_approximation** ()
  push a previous approximation data state; reverse of pop_approximation

- **bool push_available** ()
  query for whether a trial increment is restorable within a surrogate

- **void finalize_approximation** ()
  finalize an approximation by applying all previous trial increments

- **void combine_approximation** ()
  combine the current approximation with previously stored data sets

- **void combined_to_active** (bool clear_combined=true)
  promote the combined approximation into the active approximation

- **void clear_inactive** ()
  clear inactive approximations (finalization + combination completed)

- **std::vector< Approximation > & approximations** ()
  retrieve the set of Approximations from the subModel

- **const RealVectorArray & approximation_coefficients** (bool normalized=false)
  retrieve the approximation coefficients from the subModel

- **void approximation_coefficients** (const RealVectorArray &approx_coeffs, bool normalized=false)
  set the approximation coefficients within the subModel

- **const RealVector & approximation_variances** (const Variables &vars)
  retrieve the approximation variances from the subModel

- **const Pecos::SurrogateData & approximation_data** (size_t fn_index)
  retrieve the approximation data from the subModel

- **void component_parallel_mode** (short mode)
  *RecastModel* only supports parallelism in subModel, so this virtual function redefinition is simply a sanity check.

- **size_t mi_parallel_level_index** () const
return subModel’s MI parallel level index

- short local\_eval\_synchronization ()
  return subModel local synchronization setting

- int local\_eval\_concurrency ()
  return subModel local evaluation concurrency

- bool derived\_master\_overload () const
  flag which prevents overloading the master with a multiprocessor evaluation (request forwarded to subModel)

- IntIntPair estimate\_partition\_bounds (int max\_eval\_concurrency)
  estimate the minimum and maximum partition sizes that can be utilized by this Model

- void derived\_init\_communicators (ParLev\_Iter pl\_iter, int max\_eval\_concurrency, bool recurse\_flag=true)
  set up RecastModel for parallel operations (request forwarded to subModel)

- void derived\_init\_serial ()
  set up RecastModel for serial operations (request forwarded to subModel).

- void derived\_set\_communicators (ParLev\_Iter pl\_iter, int max\_eval\_concurrency, bool recurse\_flag=true)
  set active parallel configuration within subModel

- void derived\_free\_communicators (ParLev\_Iter pl\_iter, int max\_eval\_concurrency, bool recurse\_flag=true)
  deallocate communicator partitions for the RecastModel (request forwarded to subModel)

- void serve\_run (ParLev\_Iter pl\_iter, int max\_eval\_concurrency)
  Service subModel job requests received from the master. Completes when a termination message is received from stop\_servers().

- void stop\_servers ()
  executed by the master to terminate subModel server operations when RecastModel iteration is complete.

- void inactive\_view (short view, bool recurse\_flag=true)
  update the Model’s inactive view based on higher level (nested) context and optionally recurse into subModel

- const String & interface\_id () const
  return the subModel interface identifier

- bool evaluation\_cache (bool recurse\_flag=true) const
  if recurse\_flag, return the subModel evaluation cache usage

- bool restart\_file (bool recurse\_flag=true) const
  if recurse\_flag, return the subModel restart file usage

- int derived\_evaluation\_id () const
  return the current evaluation id for the RecastModel

- void set\_evaluation\_reference ()
  set the evaluation counter reference points for the RecastModel (request forwarded to subModel)

- void fine\_grained\_evaluation\_counters ()
  request fine-grained evaluation reporting within subModel

- void print\_evaluation\_summary (std::ostream &s, bool minimal\_header=false, bool relative\_count=true) const
  print the evaluation summary for the RecastModel (request forwarded to subModel)

- void warm\_start\_flag (const bool flag)
  set the warm start flag, including the orderedModels

- void eval\_tag\_prefix (const String &eval\_id\_str)
  set the hierarchical eval ID tag prefix
CHAPTER 14. CLASS DOCUMENTATION

- bool db_lookup (const Variables &search_vars, const ActiveSet &search_set, Response &found_resp)
  - RecastModel may need to map variables, asv before DB lookup, or responses after lookup.
- virtual void assign_instance ()
  - assign static pointer instance to this for use in static transformation functions
- bool init_variables (const SizetArray &vars_comps_totals, const BitArray &all_relax_di, const BitArray &all_relax_dr)
  - initialize currentVariables and related info from the passed size/type info
- void init_response (size_t num_recast_primary_fns, size_t num_recast_secondary_fns, short recast_resp_order, bool reshape_vars)
  - initialize currentResponse from the passed size info
- void reshape_response (size_t num_recast_primary_fns, size_t num_recast_secondary_fns)
  - Reshape the RecastModel Response, assuming no change in variables or derivative information.
- void init_constraints (size_t num_recast_secondary_fns, size_t recast_secondary_offset, bool reshape_vars)
  - initialize userDefinedConstraints from the passed size info
- void update_from_model (Model &model)
  - update current variables/bounds/labels/constraints from subModel
- bool update_variables_from_model (Model &model)
  - update active variables/bounds/labels from subModel
- void update_variables_active_complement_from_model (Model &model)
  - update complement of active variables/bounds/labels from subModel
- void update_response_from_model (Model &model)
  - update nonlinear constraint bounds/targets from subModel

Static Protected Member Functions

- static short response_order (const Model &sub_model)
  - helper to compute the recast response order during member initialization
- static String recast_model_id (const String &root_id, const String &type)
  - Generate a model id for recast models.

Protected Attributes

- Model subModel
  - the sub-model underlying the transformations
- int recastModelEvalCntr
  - local evaluation id counter used for id mapping
- IntActiveSetMap recastSetMap
  - map of recast active set passed to derived_evaluate_nowait(). Needed for currentResponse update in synchronization routines.
- IntVariablesMap recastVarsMap
  - map of recast variables used by derived_evaluate_nowait(). Needed for primaryRespMapping() and secondaryRespMapping() in synchronization routines.
- IntVariablesMap subModelVarsMap
  - map of subModel variables used by derived_evaluate_nowait(). Needed for primaryRespMapping() and secondaryRespMapping() in synchronization routines.
### Static Protected Attributes

- **IntResponseMap recastResponseMap**
  - map of recast responses used by `RecastModel::derived_synchronize()` and `RecastModel::derived_synchronize_nowait()`

- **IntIntMap recastIdMap**
  - mapping from subModel evaluation ids to `RecastModel` evaluation ids

- **bool nonlinearVarsMapping**
  - boolean set to true if the variables mapping involves a nonlinear transformation. Used in `transform_set()` to manage the requirement for gradients within the Hessian transformations. This does not require a `BoolDeque` for each individual variable, since response gradients and Hessians are managed per function, not per variable.

### Private Member Functions

- **void initialize_data_from_submodel()**
  - code shared among constructors to initialize base class data from submodel

- **void resize_response_mapping()**
  - resize `{primary,secondary}`MapIndices and nonlinearRespMapping to synchronize with subModel sizes

### Private Attributes

- **Sizet2DArray varsMapIndices**
  - For each subModel variable, identifies the indices of the recast variables used to define it (maps `RecastModel` variables to subModel variables; data is packed with only the variable indices employed rather than a sparsely filled $N_{sm} \times N_{r}$ matrix).

- **Sizet2DArray primaryRespMapIndices**
  - For each recast primary function, identifies the indices of the subModel functions used to define it (maps subModel response to `RecastModel` Response).

- **Sizet2DArray secondaryRespMapIndices**
  - For each recast secondary function, identifies the indices of the subModel functions used to define it (maps sub-Model response to `RecastModel` response).

- **BoolDequeArray nonlinearRespMapping**
  - array of `BoolDeques`, one for each recast response function. Each `BoolDeque` defines which subModel response functions contribute to the recast function using a nonlinear mapping. Used in `transform_set()` to augment the subModel function value/gradient requirements.

- **RealVector mappedErrorEstimates**
  - mapping of subModel.error_estimates() through response mappings

- **void(* variablesMapping)(const Variables &recast_vars, Variables &sub_model_vars)**
  - holds pointer for variables mapping function passed in ctor/initialize

- **void(* setMapping)(const Variables &recast_vars, const ActiveSet &recast_set, ActiveSet &sub_model_set)**
  - holds pointer for set mapping function passed in ctor/initialize

- **void(* primaryRespMapping)(const Variables &sub_model_vars, const Variables &recast_vars, const Response &sub_model_response, Response &recast_response)**
  -
CHAPTER 14. CLASS DOCUMENTATION

holds pointer for primary response mapping function passed in ctor/initialize

- void(* secondaryRespMapping ) (const Variables &sub_model_vars, const Variables &recast_vars, const Response &sub_model_response, Response &recast_response)
  holds pointer for secondary response mapping function passed in ctor/initialize

- void(* invVarsMapping ) (const Variables &sub_model_vars, Variables &recast_vars)
  holds pointer for optional inverse variables mapping function passed in inverse_mappings()

- void(* invSetMapping ) (const Variables &sub_model_vars, const ActiveSet &sub_model_set, ActiveSet &recast_set)
  holds pointer for optional inverse set mapping function passed in inverse_mappings()

- void(* invPriRespMapping ) (const Variables &recast_vars, const Variables &sub_model_vars, const Response &recast_response, Response &sub_model_response)
  holds pointer for optional inverse primary response mapping function passed in inverse_mappings()

- void(* invSecRespMapping ) (const Variables &recast_vars, const Variables &sub_model_vars, const Response &recast_response, Response &sub_model_response)
  holds pointer for optional inverse secondary response mapping function passed in inverse_mappings()

Additional Inherited Members

14.205.1 Detailed Description

Derived model class which provides a thin wrapper around a sub-model in order to recast the form of its inputs and/or outputs.

The RecastModel class uses function pointers to allow recasting of the subModel input/output into new problem forms. For example, this is used to recast SBO approximate subproblems, multiobjective and least-squares reductions, and variable/response.

For now, making the assumption that variables mappings are ordered by submodel active continuous, discrete int, discrete string, discrete real variables, even though all current use cases are continuous only.

When not using the standard (full) constructor, client code must make sure to complete initialization before using the RecastModel’s mapping functions. Initialization steps:

1. sub model (all ctors do this)
2. init_sizes: once known, size Variables, Response, Constraints (full and intermediate ctor do this)
3. init_maps: set indices and callback pointers (only full ctor does this)

14.205.2 Constructor & Destructor Documentation

RecastModel ( const Model & sub_model, const Sizet2DArray & vars_map_indices, const SizetArray & vars_comps_totals, const BitArray & all_relax_di, const BitArray & all_relax_dr, bool nonlinear_vars_mapping, void(*)(const Variables &recast_vars, Variables &sub_model_vars) variables_map, void(*)(const Variables &recast_vars, const ActiveSet &recast_set, ActiveSet &sub_model_set) set_map, const Sizet2DArray & primary_resp_map_indices, const Sizet2DArray & secondary_resp_map_indices, size_t recast_secondary_offset, short recast_resp_order, const BoolDequeArray & nonlinear_resp_mapping, void(*)(const Variables &sub_model_vars, const Variables &recast_vars, const Response &sub_model_response, Response &recast_response) primary_resp_map, void(*)(const Variables &sub_model_vars, const Variables &recast_vars, const Response &sub_model_response, Response &recast_response) secondary_resp_map )

standard (full) constructor; assumes provided sizes and map functions are final and constructs all member data
Default recast model constructor. Requires full definition of the transformation; if any mappings are NULL, they are assumed to remain so in later initialization or updates. Parameter vars_comps_totals indicates the number of each type of variable \(4\text{ types} \times 3\text{ domains}\) in the recast variable space. Note: recast_secondary_offset is the start index for equality constraints, typically num nonlinear ineq constraints.

References Dakota::abort_handler(), Response::copy(), Variables::copy(), Model::current_response(), Model::current_variables(), Model::currentResponse, Model::currentVariables, Variables::cv(), RecastModel::init_constraints(), RecastModel::init_response(), RecastModel::init_variables(), RecastModel::initialize_data_from_submodel(), Model::modelId, Model::modelType, RecastModel::nonlinearRespMapping, Response::num_functions(), Model::num_DerivVars, Model::numFns, RecastModel::primaryRespMapIndices, RecastModel::primaryRespMapping, RecastModel::recast_model_id(), RecastModel::root_model_id(), RecastModel::secondaryRespMapIndices, RecastModel::secondaryRespMapping, RecastModel::subModel, Model::supportsEstimDerivs, and RecastModel::variables-Mapping.

\[
\text{RecastModel ( const Model & sub_model, const SizetArray & vars_comps_totals, const BitArray & all_relax_di, const BitArray & all_relax_dr, size_t num_recast_primary_fns, size_t num_recast_secondary_fns, size_t recast_secondary_offset, short recast_resp_order )}
\]

alternate constructor; uses provided sizes to construct Variables, Response and Constraints so Model can be passed to an Iterator; requires subsequent init_maps() call.

This alternate constructor defers initialization of the function pointers until a separate call to initialize(), and accepts the minimum information needed to construct currentVariables, currentResponse, and userDefined-Constraints. The resulting model is sufficiently complete for passing to an Iterator. Parameter vars_comps_totals indicates the number of each type of variable \(4\text{ types} \times 3\text{ domains}\) in the recast variable space. Note: recast_secondary_offset is the start index for equality constraints, typically num nonlinear ineq constraints.

References RecastModel::init_sizes(), RecastModel::initialize_data_from_submodel(), Model::modelId, Model::modelType, RecastModel::recast_model_id(), RecastModel::root_model_id(), and Model::supportsEstimDerivs.

### 14.205.3 Member Function Documentation

\[
\text{void init_maps ( const Sizet2DArray & vars_map_indices, bool nonlinear_vars_mapping, void(*)(const Variables & recast_vars, Variables & sub_model_vars) variables_map, void(*)(const Variables & recast_vars, const ActiveSet & recast_set, ActiveSet & sub_model_set) set_map, const Sizet2DArray & primary_resp_map_indices, const Sizet2DArray & secondary_resp_map_indices, const BoolDequeArray & nonlinear_resp_mapping, void(*)(const Variables & sub_model_vars, const Variables & recast_vars, const Response & sub_model_response, Response & recast_response) primary_resp_map, void(*)(const Variables & sub_model_vars, const Variables & recast_vars, const Response & sub_model_response, Response & recast_response) secondary_resp_map )}
\]

initialize recast indices and map callbacks after alternate construction

This function is used for late initialization of the recasting functions. It is used in concert with the alternate constructor.


Referenced by NonDGlobalInterval::core_run(), NonDLocalInterval::core_run(), DataTransformModel::DataTransformModel(), RandomFieldModel::initialize_recast(), AdaptedBasisModel::initialize_recast(), ActiveSubspaceModel::initialize_recast(), EffGlobalMinimizer::minimize_surrogates_on_model(), NonDLocalReliability::mpp_search(), NonDGlobalReliability::optimize_gaussian_process(), ProbabilityTransformModel::ProbabilityTransformModel(), ScoringModel::ScoringModel(), and WeightingModel::WeightingModel().
void derived_evaluate ( const ActiveSet & set ) [protected], [virtual]

portion of evaluate() specific to RecastModel (forward to subModel.evaluate())

The RecastModel is evaluated by an Iterator for a recast problem formulation. Therefore, the currentVariables, incoming active set, and output currentResponse all correspond to the recast inputs/outputs.

Reimplemented from Model.

References Response::active_set(), Model::current_response(), Model::current_variables(), Model::currentResponse, Model::currentVariables, Model::evaluate(), RecastModel::primaryRespMapping, RecastModel::recastModelEvalCntr, RecastModel::secondaryRespMapping, RecastModel::subModel, RecastModel::transform_response(), RecastModel::transform_set(), RecastModel::transform_variables(), and Response::update().

Referenced by AdaptedBasisModel::derived_evaluate(), ActiveSubspaceModel::derived_evaluate(), DataTransformModel::derived_evaluate(), and RandomFieldModel::derived_evaluate().

void eval_tag_prefix ( const String & eval_id_str ) [inline], [protected], [virtual]

set the hierarchical eval ID tag prefix

RecastModel just forwards any tags to its subModel

Reimplemented from Model.

References Model::eval_tag_prefix(), and RecastModel::subModel.

void update_from_model ( Model & model ) [protected]

update current variables/bounds/labels/constraints from subModel

Update inactive values and labels in currentVariables and inactive bound constraints in userDefinedConstraints from variables and constraints data within subModel.

References RecastModel::update_response_from_model(), RecastModel::update_variables_active_complement_from_model(), and RecastModel::update_variables_from_model().

Referenced by ProbabilityTransformModel::update_from_subordinate_model(), and RecastModel::update_from_subordinate_model().

The documentation for this class was generated from the following files:

- RecastModel.hpp
- RecastModel.cpp

14.206 ReducedBasis Class Reference

Public Member Functions

- ReducedBasis ()
  
  default constructor
- void set_matrix ( const RealMatrix &)
- const RealMatrix & get_matrix ()
- void center_matrix ()
  
  center the matrix by scaling each column by its means
- void update_svd ( bool center_matrix_by_col_means=true)
  
  ensure that the factorization is current, centering if requested
- bool is_valid () const
- const Real & get_singular_values_sum () const
- const Real & get_eigen_values_sum () const
14.207. **RELAXEDVARCONSTRAINTS CLASS REFERENCE**

- const RealVector & `get_column_means`()
- const RealVector & `get_singular_values`() const
- RealVector `get_singular_values` (const TruncationCondition & ) const
- const RealMatrix & `get_left_singular_vector` () const
  
  *the num_observations n x num_observations n orthogonal matrix U; the left singular vectors are the first min(n,p) columns*

- const RealMatrix & `get_right_singular_vector_transpose` () const
  
  *the num_responses p x num_responses p orthogonal matrix V'; the right singular vectors are the first min(n,p) rows of V (columns of V)*

**Private Attributes**

- RealMatrix `matrix`
- RealMatrix `workingMatrix`
- RealMatrix `U_matrix`
- RealVector `S_values`
- RealMatrix `VT_matrix`
- RealVector `column_means`
- bool `col_means_computed`
- bool `is_centered`
- bool `is_valid_svd`
- Real `singular_values_sum`
- Real `eigen_values_sum`
- TruncationCondition * `truncation`

### 14.206.1 Detailed Description

The **ReducedBasis** class is used to ...

(TODO - RWH)

Class to manage data-driven dimension reduction. The passed matrix with num_observations n rows and num_responses p columns contains realizations of a set of responses. The class optionally centers the matrix by the column means. Stores a singular value decomposition of the passed data matrix $X = U*S*V'$, which can also be used for PCA, where we seek an eigendecomposition of the covariance: $X'*X = V*D*V'^{-1} = V*S'^2*V'$

The documentation for this class was generated from the following files:

- ReducedBasis.hpp
- ReducedBasis.cpp

### 14.207 RelaxedVarConstraints Class Reference

Derived class within the **Constraints** hierarchy which employs relaxation of discrete variables.

Inheritance diagram for RelaxedVarConstraints:

```
          Constraints
             |
             v
   RelaxedVarConstraints
```
Public Member Functions

- **RelaxedVarConstraints (const SharedVariablesData &svd)**
  
  *lightweight constructor*

- **RelaxedVarConstraints (const ProblemDescDB &problem_db, const SharedVariablesData &svd)**

  *standard constructor*

- **∼RelaxedVarConstraints ()**

  *destructor*

- **void write (std::ostream &s) const**

  *write a variable constraints object to an std::ostream*

- **void read (std::istream &s)**

  *read a variable constraints object from an std::istream*

Additional Inherited Members

14.207.1 Detailed Description

Derived class within the Constraints hierarchy which employs relaxation of discrete variables.

Derived variable constraints classes take different views of the design, uncertain, and state variable types and the continuous and discrete domain types. The RelaxedVarConstraints derived class combines continuous and discrete domain types through integer relaxation. The branch and bound method uses this approach (see Variables::get_variables(problem_db) for variables type selection; variables type is passed to the Constraints constructor in Model).

14.207.2 Constructor & Destructor Documentation

**RelaxedVarConstraints ( const ProblemDescDB & problem_db, const SharedVariablesData & svd )**

*standard constructor*

In this class, a relaxed data approach is used in which continuous and discrete arrays are combined into a single continuous array (integrality is relaxed; the converse of truncating reals is not currently supported but could be in the future if needed). Iterators which use this class include: BranchBndOptimizer.

References SharedVariablesData::all_relaxed_discrete_int(), SharedVariablesData::all_relaxed_discrete_real(), Constraints::allContinuousLowerBnds, Constraints::allContinuousUpperBnds, Constraints::allDiscreteIntLowerBnds, Constraints::allDiscreteIntUpperBnds, Constraints::allDiscreteRealLowerBnds, Constraints::allDiscreteRealUpperBnds, Dakota::copy_data_partial(), ProblemDescDB::get_iv(), ProblemDescDB::get_rv(), Constraints::sharedVarsData, and SharedVariablesData::view().

The documentation for this class was generated from the following files:

- RelaxedVarConstraints.hpp
- RelaxedVarConstraints.cpp

14.208 RelaxedVariables Class Reference

Derived class within the Variables hierarchy which employs the relaxation of discrete variables.

Inheritance diagram for RelaxedVariables:
Public Member Functions

- **RelaxedVariables** (const ProblemDescDB &problem_db, const std::pair< short, short > &view)
  standard constructor
- **RelaxedVariables** (const SharedVariablesData &svd)
  lightweight constructor
- **~RelaxedVariables** ()
  destructor

Protected Member Functions

- void **read** (std::istream &s)
  read a variables object from an std::istream
- void **write** (std::ostream &s, unsigned short vars_part=ALL_VARS) const
  write a variables object to an std::ostream, e.g., the console, optionally specifying which partition (all/active/inactive)
- void **write_aprepro** (std::ostream &s) const
  write a variables object to an std::ostream in aprepro format, e.g., a parameters file
- void **read_tabular** (std::istream &s, unsigned short vars_part=ALL_VARS)
- void **write_tabular** (std::ostream &s, unsigned short vars_part=ALL_VARS) const
  write a variables object in tabular format to an std::ostream, optionally specifying which partition (all/active/inactive)
- void **write_tabular_labels** (std::ostream &s, unsigned short vars_part=ALL_VARS) const
  write the labels in input spec order to a std::ostream, optionally specifying which partition (all/active/inactive)
- template<typename Reader>
  void **read_core** (std::istream &s, Reader read_handler, unsigned short vars_part)
    Implementation of reading various formats using the specified read handler, accounting for reordering due to relaxation.
- template<typename Writer>
  void **write_core** (std::ostream &s, Writer write_handler, unsigned short vars_part) const
    Implementation of writing various formats using the specified write handler, accounting for reordering due to relaxation.

Additional Inherited Members

14.208.1 Detailed Description

Derived class within the Variables hierarchy which employs the relaxation of discrete variables.

Derived variables classes take different views of the design, uncertain, and state variable types and the continuous and discrete domain types. The RelaxedVariables derived class combines continuous and discrete domain types but separates design, uncertain, and state variable types. The branch and bound method uses this approach (see Variables::get_variables(problem_db)).
14.208.2 Constructor & Destructor Documentation

RelaxedVariables ( const ProblemDescDB & problem_db, const std::pair< short, short > & view )

standard constructor

In this class, a relaxed data approach is used in which continuous and discrete arrays are combined into a single continuous array (integrality is relaxed; the converse of truncating reals is not currently supported but could be in the future if needed). Iterators/strategies which use this class include: BranchBndOptimizer. Extract fundamental variable types and labels and merge continuous and discrete domains to create aggregate arrays and views.

References SharedVariablesData::all_relaxed_discrete_int(), SharedVariablesData::all_relaxed_discrete_real(), Variables::allContinuousVars, Variables::allDiscreteIntVars, Variables::allDiscreteRealVars, Variables::allDiscreteStringVars, Dakota::copy_data_partial(), ProblemDescDB::get_iv(), ProblemDescDB::get_rv(), ProblemDescDB::get_sa(), Variables::sharedVarsData, and SharedVariablesData::view().

14.208.3 Member Function Documentation

void read_tabular ( std::istream & s, unsigned short vars_part = ALL_VARS ) [protected], [virtual]

Presumes variables object is appropriately sized to receive data

Reimplemented from Variables.

References RelaxedVariables::read_core().

void read_core ( std::istream & s, Reader read_handler, unsigned short vars_part ) [protected]

Implementation of reading various formats using the specified read handler, accounting for reordering due to relaxation.

Reordering is required in all read/write cases that will be visible to the user since all derived vars classes should use the same ordering for clarity. Neutral file I/O, binary streams, and packed buffers do not need to reorder (so long as read/write are consistent) since this data is not intended for public consumption.

References SharedVariablesData::active_components_totals(), Variables::all_continuous_variable_labels(), Variables::all_discrete_int_variable_labels(), Variables::all_discrete_real_variable_labels(), Variables::all_discrete_string_variable_labels(), SharedVariablesData::all_relaxed_discrete_int(), SharedVariablesData::all_relaxed_discrete_real(), Variables::allContinuousVars, Variables::allDiscreteIntVars, Variables::allDiscreteRealVars, Variables::allDiscreteStringVars, SharedVariablesData::components_totals(), SharedVariablesData::cv_start(), SharedVariablesData::div_start(), SharedVariablesData::drv_start(), SharedVariablesData::dsv_start(), SharedVariablesData::icv_start(), SharedVariablesData::idiv_start(), SharedVariablesData::idrv_start(), SharedVariablesData::idsv_start(), SharedVariablesData::inactive_components_totals(), and Variables::sharedVarsData.

Referenced by RelaxedVariables::read(), and RelaxedVariables::read_tabular().

The documentation for this class was generated from the following files:

- RelaxedVariables.hpp
- RelaxedVariables.cpp

14.209 Response Class Reference

Container class for response functions and their derivatives. Response provides the enveloper base class.

Inheritance diagram for Response:
Public Member Functions

- **Response ()**
  default constructor

- **Response (short type, const Variables &vars, const ProblemDescDB &problem_db)**
  standard constructor built from problem description database

- **Response (const SharedResponseData &srd, const ActiveSet &set)**
  alternate constructor that shares response data

- **Response (short type, const ActiveSet &set)**
  alternate constructor using limited data without sharing

- **Response (const SharedResponseData &srd)**
  alternate constructor using limited data (explicit disallows implicit type conversion)

- **Response (const Response &response)**
  copy constructor

- **virtual ~Response ()**
  destructor

- **Response operator= (const Response &response)**
  assignment operator

- **const SharedResponseData & shared_data () const**
  return sharedRespData

- **SharedResponseData & shared_data ()**
  return sharedRespData

- **size_t num_functions () const**
  return the number of response functions

- **const ActiveSet & active_set () const**
  return the active set

- **void active_set (const ActiveSet &set)**
  set the active set

- **const ShortArray & active_set_request_vector () const**
  return the active set request vector

- **void active_set_request_vector (const ShortArray &asrv)**
  set the active set request vector and verify consistent number of response functions

- **const SizetArray & active_set_derivative_vector () const**
  return the active set derivative vector

- **void active_set_derivative_vector (const SizetArray &asdv)**
  set the active set derivative vector and reshape functionGradients/functionHessians if needed

- **const Real & function_value (size_t i) const**
return a function value

- Real & function_value_view (size_t i)
  return a "view" of a function value for updating in place
- const RealVector & function_values () const
  return all function values
- RealVector function_values_view ()
  return all function values as a view for updating in place
- RealVector function_values_view () const
  return all function values as a view for accessing the function values vector from a const response
- void function_value (const Real &fn_val, size_t i)
  set a function value
- void function_values (const RealVector &fn_vals)
  set all function values
- const Real * function_gradient (int i) const
  return the i-th function gradient as a const Real*
- RealVector function_gradient_view (int i)
  return the i-th function gradient as a SerialDenseVector view (shallow copy) for updating in place
- RealVector function_gradient_view (int i) const
  return the i-th function gradient as a SerialDenseVector view (shallow copy) for accessing a column vector from a const matrix
- RealVector function_gradient_copy (int i) const
  return the i-th function gradient as a SerialDenseVector Teuchos::Copy (deep copy)
- const RealMatrix & function_gradients () const
  return all function gradients
- RealMatrix function_gradients_view ()
  return all function gradients as a view for updating in place
- RealMatrix function_gradients_view () const
  return all function gradients as a view for updating in place
- void function_gradient (const RealVector &fn_grad, int i)
  set a function gradient
- void function_gradients (const RealMatrix &fn_grads)
  set all function gradients
- const RealSymMatrix & function_hessian (size_t i) const
  return the i-th function Hessian
- RealSymMatrix function_hessian_view (size_t i)
  return the i-th function Hessian as a Teuchos::View (shallow copy) for updating in place
- RealSymMatrix function_hessian_view (size_t i) const
  return the i-th function Hessian as a Teuchos::View (shallow copy) for accessing the i-th matrix within a const matrix array
- const RealSymMatrixArray & function_hessians () const
  return all function Hessians
- RealSymMatrixArray function_hessians_view ()
  return all function Hessians as Teuchos::Views (shallow copies) for updating in place
• RealSymMatrixArray function_hessians_view () const
  return all function Hessians as Teuchos::Views (shallow copies) for updating in place
• void function_hessian (const RealSymMatrix &fn_hessian, size_t i)
  set a function Hessian
• void function_hessians (const RealSymMatrixArray &fn_hessians)
  set all function Hessians
• const IntVector & field_lengths () const
  return the field lengths (from SharedResponseData)
• void field_lengths (const IntVector &field_lens)
  set the field lengths (within SharedResponseData)
• RealVector field_values_view (size_t i) const
  return const field values for the i-th field
• RealVector field_values_view (size_t i)
  return a "view" of the i-th field values for updating in place
• void field_values (const RealVector &field_val, size_t i)
  set the values for the i-th field
• RealMatrix field_gradients_view (size_t i) const
  return a view of the gradients of each element of the i-th field
• RealSymMatrixArray field_hessians_view (size_t i) const
  return a view of the hessians of each element of the i-th field
• RealMatrix field_coords_view (size_t i)
  return a "view" of the i-th field’s coordinates
• const RealMatrix field_coords_view (size_t i) const
  return a const "view" of the i-th field’s coordinates
• void field_coords (const RealMatrix &field.coords, size_t i)
  set the i-th field’s coordinates
• const IntVector & num_coords_per_field () const
  return the number of coordinates each field has (from SharedResponseData)
• const StringArray & function_labels () const
  return the fine-grained (unrolled) response function identifier strings from sharedRespData
• void function_labels (const StringArray &labels)
  set the fine-grained (unrolled) response function identifier strings within sharedRespData
• const StringArray & field_group_labels ()
  return the user-provided field group labels instead of the unrolled labels available through function_labels()
• void read (std::istream &s, const unsigned short format=FLEXIBLE_RESULTS)
  read a response object of specified format from an std::istream
• void write (std::ostream &s) const
  write a response object to an std::ostream
• void read.annotated (std::istream &s)
  read a response object in annotated format from an std::istream
• void write.annotated (std::ostream &s) const
  write a response object in annotated format to an std::ostream
• void read.tabular (std::istream &s)
read responseRep::functionValues in tabular format from an std::istream

- void write_tabular (std::ostream &s) const
  write responseRep::functionValues in tabular format to an std::ostream
- void write_tabular_labels (std::ostream &s) const
  write the labels to a tabular data stream
- void read (MPIUnpackBuffer &s)
  read a response object from a packed MPI buffer
- void write (MPIPackBuffer &s) const
  write a response object to a packed MPI buffer
- Response copy (bool deep_srd=false) const
  return a deep response copy of the contained responseRep for use in history mechanisms (SharedResponseData uses a shallow copy by default)
- int data_size ()
  return the number of doubles active in response. Used for sizing double* response_data arrays passed into read_data and write_data.
- void read_data (double *response_data)
  read from an incoming double* array
- void write_data (double *response_data)
  write to an incoming double* array
- void overlay (const Response &response)
  add incoming response to functionValues/Gradients/Hessians
- void update (const Response &response)
  Used in place of operator= when only results data updates are desired (functionValues/functionGradients/functionHessians are updated, ASV/labels/id’s/etc. are not). Care is taken to allow different derivative array sizing between the two response objects.
- void update (const RealVector &source_fn_vals, const RealMatrix &source_fn_grads, const RealSymMatrixArray &source_fn_hessians, const ActiveSet &source_set)
  Overloaded form which allows update from components of a response object. Care is taken to allow different derivative array sizing.
- void update_partial (size_t start_index_target, size_t num_items, const Response &response, size_t start_index_source)
  partial update of this response object from another response object. The response objects may have different numbers of response functions.
- void update_partial (size_t start_index_target, size_t num_items, const RealVector &source_fn_vals, const RealMatrix &source_fn_grads, const RealSymMatrixArray &source_fn_hessians, const ActiveSet &source_set, size_t start_index_source)
  Overloaded form which allows partial update from components of a response object. The response objects may have different numbers of response functions.
- void reshape (size_t num_fns, size_t num_params, bool grad_flag, bool hess_flag)
  reshapes response data arrays
- void reset ()
  resets all response data to zero
- void reset_inactive ()
  resets all inactive response data to zero
- bool is_null () const
function to check responseRep (does this handle contain a body)

- virtual void `set_scalar_covariance` (RealVector &scalars)
  
  method to set the covariance matrix defined for `ExperimentResponse`

- virtual const
  ExperimentCovariance & `experiment_covariance` () const
  
  retrieve the `ExperimentCovariance` structure

- virtual void `set_full_covariance` (std::vector<RealMatrix> &matrices, std::vector<RealVector> &diagonals, RealVector &scalars, IntVector matrix_map_indices, IntVector diagonal_map_indices, IntVector scalar_map_indices)
  
  method to set the full covariance matrices for `ExperimentResponse`

- virtual Real `apply_covariance` (const RealVector &residuals) const
  
  method to compute the triple product $v^T \cdot \text{inv}(C) \cdot v$.

- virtual void `apply_covariance_inv_sqrt` (const RealVector &residuals, RealVector &weighted_residuals) const
  
  method to compute $(v^T \cdot \text{inv}(C) \cdot v)^{1/2}$, to compute weighted residual

- virtual void `apply_covariance_inv_sqrt` (const RealMatrix &gradients, RealMatrix &weighted_gradients) const

- virtual void `apply_covariance_inv_sqrt` (const RealSymMatrixArray &hessians, RealSymMatrixArray &weighted_hessians) const

- virtual void `get_covariance_diagonal` (RealVector &diagonal) const

- virtual Real `covariance_determinant` () const
  
  covariance determinant for one experiment (default 1.0)

- virtual Real `log_covariance_determinant` () const
  
  log of covariance determinant for one experiment (default 0.0)

Protected Member Functions

- `Response` (BaseConstructor, const Variables &vars, const ProblemDescDB &problem_db)
  
  constructor initializes the base class part of letter classes (BaseConstructor overloading avoids infinite recursion in the derived class constructors - Coplien, p. 139)

- `Response` (BaseConstructor, const SharedResponseData &srd, const ActiveSet &set)
  
  constructor initializes the base class part of letter classes (BaseConstructor overloading avoids infinite recursion in the derived class constructors - Coplien, p. 139)

- `Response` (BaseConstructor, const ActiveSet &set)
  
  constructor initializes the base class part of letter classes (BaseConstructor overloading avoids infinite recursion in the derived class constructors - Coplien, p. 139)

- `Response` (BaseConstructor, const SharedResponseData &srd)
  
  constructor initializes the base class part of letter classes (BaseConstructor overloading avoids infinite recursion in the derived class constructors - Coplien, p. 139)

- virtual void `copy_rep` (Response *source Resp_rep)
  
  Implementation of data copy for `Response` letters (specialized by some derived letter types); pulls base class data from `source Resp_rep` into this object.
CHAPTER 14. CLASS DOCUMENTATION

Protected Attributes

- **SharedResponseData sharedRespData**
  - reference-counted instance of shared response data: id’s, labels
- **RealVector functionValues**
  - Abstract set of response functions. Ordered: [primary Scalar, primary Field, nonlinear inequality, nonlinear equality].
- **RealMatrix functionGradients**
  - first derivatives of the response functions
- **RealSymMatrixArray functionHessians**
  - second derivatives of the response functions
- **IntRealMatrixMap fieldCoords**
  - coordinates (independent vars like x,t) on which field values depend
- **ActiveSet responseActiveSet**
  - copy of the ActiveSet used by the Model to generate a Response instance

Private Member Functions

- **template<class Archive , typename OrdinalType , typename ScalarType >
  - void write_sdm_col (Archive &ar, int col, const Teuchos::SerialDenseMatrix< OrdinalType, ScalarType > &sdm) const
    - write a column of a SerialDenseMatrix
  - void read_sdm_col (Archive &ar, int col, Teuchos::SerialDenseMatrix< OrdinalType, ScalarType > &sdm)
    - read a column of a SerialDenseMatrix
  - template<class Archive >
    - void load (Archive &ar, const unsigned int version)
      - read a Response from an archive
  - template<class Archive >
    - void load_rep (Archive &ar, const unsigned int version)
      - read a Response letter object from an archive
  - void save (Archive &ar, const unsigned int version) const
    - write a Response to an archive
  - template<class Archive >
    - void save_rep (Archive &ar, const unsigned int version) const
      - write a Response letter object to an archive
- **BOOST_SERIALIZATION_SPLIT_MEMBER () Response *get_response(short type**
  - Used by standard envelope constructor to instantiate a new letter class.
- **Response * get_response (const SharedResponseData &srd, const ActiveSet &set) const**
  - Used by alternate envelope constructor to instantiate a new letter class.
- **Response * get_response (short type, const ActiveSet &set) const**
  - Used by alternate envelope constructor to instantiate a new letter class.
- **Response * get_response (const SharedResponseData &srd) const**
  - Used by copy() to instantiate a new letter class.
- **Response** *get_response* (short type) const
  
  Used by read functions to instantiate a new letter class.

- void **read_annotated_rep** (std::istream &s)
  
  read a letter object in annotated format from an std::istream

- void **write_annotated_rep** (std::ostream &s) const
  
  write a letter object in annotated format to an std::ostream

- void **read_rep** (MPIUnpackBuffer &s)
  
  read a letter object from a packed MPI buffer

- void **write_rep** (MPIPackBuffer &s) const
  
  write a letter object to a packed MPI buffer

- void **shape_rep** (const ActiveSet &set, bool initialize=true)
  
  resizes the representation’s containers

- void **reshape_rep** (size_t num_fns, size_t num_params, bool grad_flag, bool hess_flag)
  
  resizes the representation’s containers

- void **read_gradients** (std::istream &s, const ShortArray &asv, std::ostringstream &error)
  
  Read gradients from a freeform stream. Insert error messages.

- void **read_hessians** (std::istream &s, const ShortArray &asv, std::ostringstream &error)
  
  Read Hessians from a freeform stream. Insert error messages.

- void **read_labeled_fn_vals** (std::istream &s, const ShortArray &asv, std::ostringstream &errors)
  
  Read function values from an annotated stream. Insert error messages.

- void **read_flexible_fn_vals** (std::istream &s, const ShortArray &asv, std::ostringstream &errors)
  
  Read function values from a stream in a "flexible" way – ignoring any labels. Insert error messages into errors stream.

- bool **failure_reported** (std::istream &s)
  
  Check for FAIL in stream.

### Private Attributes

- const Variables & **vars**
- const Variables const **ProblemDescDB &problem_db**
- **Response** *responseRep*
  
  pointer to the body (handle-body idiom)

- int **referenceCount**
  
  number of handle objects sharing responseRep

### Friends

- class **boost::serialization::access**
- bool **operator==** (const Response &resp1, const Response &resp2)
  
  equality operator

- bool **operator!=** (const Response &resp1, const Response &resp2)
  
  inequality operator
14.209.1 Detailed Description

Container class for response functions and their derivatives. Response provides the enveloper base class.

The Response class is a container class for an abstract set of functions (functionValues) and their first (function-Gradients) and second (functionHessians) derivatives. The functions may involve objective and constraint functions (optimization data set), least squares terms (parameter estimation data set), or generic response functions (uncertainty quantification data set). When field responses are present, the stored response elements are ordered: [primary_scalar, primary_field, nonlinear_inequality, nonlinear_equality].

For memory efficiency, it employs the "letter-envelope idiom" approach to reference counting and representation sharing (see Coplien "Advanced C++"), for which the base Response class serves as the envelope and one of its derived classes serves as the letter.

14.209.2 Member Data Documentation

RealMatrix functionGradients  [protected]

first derivatives of the response functions

the gradient vectors (plural) are column vectors in the matrix (singular) with (row, col) = (variable index, response fn index).

Referenced by Response::field_gradients_view(), Response::function_gradient(), Response::function_gradient_copy(), Response::function_gradient_view(), Response::function_gradients(), and Response::function_gradients_view().

The documentation for this class was generated from the following file:

• DakotaResponse.hpp

14.210 RestartWriter Class Reference

Public Member Functions

• RestartWriter ()
  optional default ctor allowing a non-outputting RestartWriter

• RestartWriter (const String &write_restart_filename)
  typical ctor taking a filename; this class encapsulates the output stream

• RestartWriter (std::ostream &write_restart_stream)
  alternate ctor taking a stream, helpful for testing; assumes client manages the output stream

• const String & filename ()
  output filename for this writer

• void append_prp (const ParamResponsePair &prp_in)
  add the passed pair to the restart file

• void flush ()
  flush the restart stream so we have a complete restart record should Dakota abort

Private Member Functions

• RestartWriter (const RestartWriter &)
  copy constructor is disallowed due to file stream

• const RestartWriter & operator= (const RestartWriter &)
  assignment is disallowed due to file stream
Private Attributes

- String restartOutputFilename
  
  *the name of the restart output file*

- std::ofstream restartOutputFS
  
  *Binary stream to which restart data is written.*

- std::unique_ptr<
  boost::archive::binary_oarchive>
restartOutputArchive
  
  *Binary output archive to which data is written (pointer since no default ctor for oarchive and may not be initialized)*.

14.210.1 Detailed Description

Component for writing restart files. Creation and destruction of archive and associated stream are managed here.

The documentation for this class was generated from the following files:

- OutputManager.hpp
- OutputManager.cpp

14.211 ResultAttribute< T > Struct Template Reference

Data structure for a single Real, String, or int valued attribute.

Public Member Functions

- **ResultAttribute** (const String &label, const T &value)
  
  *Construct an attribute.*

Public Attributes

- String label
  
  *Key for the attribute.*

- T value
  
  *Value for the attribute.*

14.211.1 Detailed Description

`template<typename T> struct Dakota::ResultAttribute< T >`

Data structure for a single Real, String, or int valued attribute.

The documentation for this struct was generated from the following file:

- dakota_results_types.hpp
## 14.212 ResultsDBAny Class Reference

Inheritance diagram for ResultsDBAny:

```
ResultsDBBase

ResultsDBAny
```

### Public Member Functions

- **ResultsDBAny** (const String &filename)
- **void insert** (const StrStrSizet &iterator_id, const std::string &data, const boost::any &result, const MetaDataType &metadata) override
  
  *record addition with metadata map*

- **void flush** () const
  
  *Write data to file.*

- **void insert** (const StrStrSizet &iterator_id, const StringArray &location, const boost::any &data, const DimScaleMap &scales=DimScaleMap(), const AttributeArray &attrs=AttributeArray(), const bool &transpose=false) override
  
  *insert an arbitrary type (RealMatrix) with metadata*

- **void allocate_vector** (const StrStrSizet &iterator_id, const StringArray &location, ResultsOutputType stored_type, const int &len, const DimScaleMap &scales=DimScaleMap(), const AttributeArray &attrs=AttributeArray())
  
  *Pre-allocate a vector and (optionally) attach dimension scales and attributes. Insert elements using insert_into(*...*)*

- **void allocate_matrix** (const StrStrSizet &iterator_id, const StringArray &location, ResultsOutputType stored_type, const int &num_rows, const int &num_cols, const DimScaleMap &scales=DimScaleMap(), const AttributeArray &attrs=AttributeArray())
  
  *Pre-allocate a matrix and (optionally) attach dimension scales and attributes. Insert rows or columns using insert_into(*...*)*

- **void insert_into** (const StrStrSizet &iterator_id, const StringArray &location, const boost::any &data, const int &index, const bool &row)
  
  *Insert a row or column into a pre-allocated matrix.*

- **void add_metadata_to_method** (const StrStrSizet &iterator_id, const AttributeArray &attrs) override
  
  *Add key:value metadata to method.*

- **void add_metadata_to_execution** (const StrStrSizet &iterator_id, const AttributeArray &attrs) override
  
  *Add key:value metadata to execution.*

- **void add_metadata_to_object** (const StrStrSizet &iterator_id, const StringArray &location, const AttributeArray &attrs) override
  
  *Associate key:value metadata with the object at the location.*

- **void add_metadata_to_study** (const AttributeArray &attrs) override
  
  *Associate key:value metadata with the study.*
Private Member Functions

- void `print_metadata` (std::ostream &os, const MetaDataType &md) const
  print metadata to ostream
- void `extract_data` (const boost::any &dataholder, std::ostream &os) const
determine the type of contained data and output it to ostream
- void `output_data` (const std::vector<double> &data, std::ostream &os) const
  output data to ostream
- void `output_data` (const std::vector<RealVector> &data, std::ostream &os) const
  output data to ostream
- void `output_data` (const std::vector<std::string> &data, std::ostream &os) const
  output data to ostream
- void `output_data` (const std::vector<std::vector<std::string>> &data, std::ostream &os) const
  output data to ostream
- void `output_data` (const RealMatrix &data, std::ostream &os) const
  output data to ostream

Private Attributes

- String `fileName`
  name of database file

Additional Inherited Members

14.212.1 Detailed Description

Class: ResultsDBAny

Description: A map-based container to store DAKOTA Iterator results in underlying boost::any, with optional metadata

14.212.2 Member Function Documentation

void `insert` ( const StrStrSizet &iterator_id, const std::string &data_name, const boost::any &result, const MetaDataType &metadata ) [override], [virtual]

record addition with metadata map

Add or update existing entry

Implements ResultsDBBase.

References ResultsDBBase::iteratorData, and Dakota::make_key().
void extract_data ( const boost::any & dataholder, std::ostream & os ) const  [private]

determine the type of contained data and output it to ostream
    Extract the data from the held any and map to supported concrete types int double RealVector (Teuchos::SerialDenseVector<int,double>) RealMatrix (Teuchos::SerialDenseMatrix<int,double>)
    References ResultsDBAny::output_data().
    Referenced by ResultsDBAny::flush().
    The documentation for this class was generated from the following files:
    - ResultsDBAny.hpp
    - ResultsDBAny.cpp

14.213 ResultsDBBase Class Reference

Inheritance diagram for ResultsDBBase:

```
ResultsDBBase
       |__________ResultsDBAny
               |__________ResultsDBHDF5
```

Public Member Functions

- virtual void flush () const
  If supported, flush data to the database or disk.
- virtual void allocate_vector (const StrStrSize &iterator_id, const StringArray &location, ResultsOutputType stored_type, const int &len, const DimScaleMap &scales=DimScaleMap(), const AttributeArray &attrs=AttributeArray())=0
  Pre-allocate a vector and (optionally) attach dimension scales and attributes. Insert elements using insert_into(...)
- virtual void allocate_matrix (const StrStrSize &iterator_id, const StringArray &location, ResultsOutputType stored_type, const int &num_rows, const int &num_cols, const DimScaleMap &scales=DimScaleMap(), const AttributeArray &attrs=AttributeArray())=0
  Pre-allocate a matrix and (optionally) attach dimension scales and attributes. Insert rows or columns using insert_into(...)
- virtual void insert (const StrStrSize &iterator_id, const StringArray &location, const boost::any &data, const DimScaleMap &scales=DimScaleMap(), const AttributeArray &attrs=AttributeArray(), const bool &transpose=false)=0
  addition with dimension scales and attributes
- virtual void insert_into (const StrStrSize &iterator_id, const StringArray &location, const boost::any &data, const int &index, const bool &row)=0
  Insert a row or column into a pre-allocated matrix.
- virtual void add_metadata_to_method (const StrStrSize &iterator_id, const AttributeArray &attrs)=0
  Add key:value metadata to a method.
- virtual void add_metadata_to_execution (const StrStrSize &iterator_id, const AttributeArray &attrs)=0
  Add key:value metadata to an execution.
virtual void add_metadata_to_object (const StrStrSizet &iterator_id, const StringArray &location, const AttributeArray &attrs)=0

Associate key:value metadata with the object at the location.

virtual void add_metadata_to_study (const AttributeArray &attrs)=0

Associate key:value metadata to the study.

template<typename StoredType>
void array_allocate (const StrStrSizet &iterator_id, const std::string &data_name, size_t array_size, const MetaDataType &metadata)

allocate an entry with sized array of the StoredType, e.g., array across response functions or optimization result sets

template<typename StoredType>
void array_insert (const StrStrSizet &iterator_id, const std::string &data_name, size_t index, const StoredType &sent_data)

insert sent_data in specified position in previously allocated array

virtual void insert (const StrStrSizet &iterator_id, const std::string &data_name, const boost::any &result, const MetaDataType &metadata)=0

record addition with metadata map

Protected Attributes

std::map< ResultsKeyType, ResultsValueType > iteratorData

core data storage (map from key to value type)

14.213.1 Detailed Description

Class: ResultsDBBase

Description: A map-based container to store DAKOTA Iterator results in underlying boost::any, with optional metadata

14.213.2 Member Function Documentation

void array_insert ( const StrStrSizet & iterator_id, const std::string & data_name, size_t index, const StoredType & sent_data )

insert sent_data in specified position in previously allocated array

insert requires previous allocation, and does not allow metadata update

References Dakota::abort_handler(), ResultsDBBase::iteratorData, and Dakota::make_key().

The documentation for this class was generated from the following file:

• ResultsDBBase.hpp

14.214 ResultsDBHDF5 Class Reference

Manage interactions between ResultsManager and the low-level HDFIOHelper class.

Inheritance diagram for ResultsDBHDF5:
Public Member Functions

- **ResultsDBHDF5** (bool in_core, std::shared_ptr<HDF5IOHelper> hdf5_helper_ptr)
  - void **flush** () const
    - Flush HDF5 cache to disk.
  - void **allocate_vector** (const StrStrSizet &iterator_id, const StringArray &location, ResultsOutputType stored_type, const int &len, const DimScaleMap &scales=DimScaleMap(), const AttributeArray &attrs=AttributeArray())
    - Pre-allocate a vector and (optionally) attach dimension scales and attributes. Insert elements using insert_into(...)
  - void **allocate_matrix** (const StrStrSizet &iterator_id, const StringArray &location, ResultsOutputType stored_type, const int &num_rows, const int &num_cols, const DimScaleMap &scales=DimScaleMap(), const AttributeArray &attrs=AttributeArray())
    - Pre-allocate a matrix and (optionally) attach dimension scales and attributes. Insert rows or columns using insert_into(...)
  - void **insert_into** (const StrStrSizet &iterator_id, const StringArray &location, const boost::any &data, const int &index, const bool &row)
    - Insert a row or column into a pre-allocated matrix.
  - void **insert** (const StrStrSizet &iterator_id, const StringArray &location, const boost::any &data, const DimScaleMap &scales=DimScaleMap(), const AttributeArray &attrs=AttributeArray(), const bool &transpose=false) override
    - Insert an arbitrary type (e.g., RealMatrix) with scales
  - void **add_metadata_to_method** (const StrStrSizet &iterator_id, const AttributeArray &attrs) override
    - Add attributes to the HDF5 method group.
  - void **add_metadata_to_execution** (const StrStrSizet &iterator_id, const AttributeArray &attrs) override
    - Add attributes to the HDF5 execution group.
  - void **add_metadata_to_object** (const StrStrSizet &iterator_id, const StringArray &location, const AttributeArray &attrs) override
    - Associate key:value metadata with the object at the location.
  - void **add_metadata_to_study** (const AttributeArray &attrs) override
    - Associate key:value metadata with the study.
  - void **insert** (const StrStrSizet &iterator_id, const std::string &data_name, const boost::any &result, const MetaDataType &metadata) override
    - Record addition with metadata map
  - template<typename StoredType>
    - void **array_allocate** (const StrStrSizet &iterator_id, const std::string &data_name, size_t array_size, const MetaDataType &metadata)
      - Allocate an entry with sized array of the StoredType, e.g., array across response functions or optimization results sets
  - template<typename StoredType>
    - void **array_insert** (const StrStrSizet &iterator_id, const std::string &data_name, size_t index, const StoredType &stored_data)
Private Member Functions

- void attach_scales (const String &dset_name, const StrStrSizet &iterator_id, const StringArray &location, const DimScaleMap &scales)
  Attach a scale to a dataset.
- void add_attributes (const String &linkname, const AttributeArray &attrs)
  Add attributes to the object with linkname.
- void add_name_to_method (const StrStrSizet &iterator_id)
  Add the name (Dakota keyword) as metadata to a method group.
- bool method_in_cache (const StrStrSizet &iterator_id) const
  Check whether the name has already been added to a method group.

Private Attributes

- std::set<String> methodIdCache
  Cached method Ids; used to know which methods have already had their method_name attribute set. Hopefully faster than querying the HDF5 file.
- std::shared_ptr<HDF5IOHelper> hdf5Stream
  Instance of HDF5IOHelper (must be a pointer because it’s shared with the global evaluation store instance).

Static Private Attributes

- static const std::string outputVersion = "2.1.0"
  Version of the output file. See comments near the definition in ResultsDBHDF5.cpp.

Additional Inherited Members

14.214.1 Detailed Description

Manage interactions between ResultsManager and the low-level HDFIOHelper class.

The documentation for this class was generated from the following files:

- ResultsDBHDF5.hpp
- ResultsDBHDF5.cpp

14.215 ResultsEntry< StoredType > Class Template Reference

Class to manage in-core vs. file database lookups.

Public Member Functions

- ResultsEntry (const ResultsManager &results_mgr, const StrStrSizet &iterator_id, const std::string &data_name)
  Construct ResultsEntry containing retrieved item of StoredType.
- ResultsEntry (const ResultsManager &results_mgr, const StrStrSizet &iterator_id, const std::string &data_name, size_t array_index)
  Construct ResultsEntry to retrieve item array_index from array of StoredType.
Private Member Functions

- ResultsEntry ()
  return a reference to the stored data, whether from core or file

Private Attributes

- bool coreActive
  whether the ResultsManager has an active in-core database
- StoredType dbData
  data retrieved from file data base
- const StoredType * dbDataPtr
  non-const pointer to const data we don’t own in the core case

14.215.1 Detailed Description

template<typename StoredType> class Dakota::ResultsEntry<StoredType>

Class to manage in-core vs. file database lookups. ResultsEntry manages database lookups. If a core database is available, will return a reference directly to the stored data; if disk, will return reference to a local copy contained in this class. Allows disk-stored data to persist for minimum time during lookup to support true out-of-core use cases.

14.215.2 Constructor & Destructor Documentation

ResultsEntry () [private]

return a reference to the stored data, whether from core or file

default construction disallowed: data must be initialized from DB lookup if needed

The documentation for this class was generated from the following file:

- ResultsManager.hpp

14.216 ResultsFileError Class Reference

exception throw for other results file read error

Inheritance diagram for ResultsFileError:

```
  FileReadException
    RESULTSFILEERROR
```

Public Member Functions

- ResultsFileError (const std::string &msg)
### 14.217. RESULTSMANAGER CLASS REFERENCE

#### 14.216.1 Detailed Description

Exception throw for other results file read error

The documentation for this class was generated from the following file:

- dakota_global_defs.hpp

### 14.217 ResultsManager Class Reference

Results manager for iterator final data.

#### Public Member Functions

- **ResultsManager ()**
  
  *default constructor: no databases active until they are added*

- **void clear_databases ()**
  
  *Delete all databases.*

- **void add_database (std::unique_ptr< ResultsDBase >)***

  *Add a database.*

- **bool active () const**

  *whether any databases are active*

- **void flush () const**

  *Flush data to the database or disk, if supported.*

- **void close ()**

  *Close the database, if supported. This removes it from the active list of databases.*

- **template<typename StoredType> void insert (const StrStrSizet &iterator_id, const StringArray &location, const StoredType &sent_data, const DimScaleMap &scales=DimScaleMap(), const AttributeArray &attrs=AttributeArray(), const bool &transpose=false) const***

  *Insert using dimension scales and attributes (DimScaleMap and AttributeArray in dakota_results_types.hpp)*

- **void allocate_matrix (const StrStrSizet &iterator_id, const StringArray &location, ResultsOutputType stored_type, const int &num_rows, const int &num_cols, const DimScaleMap &scales=DimScaleMap(), const AttributeArray &attrs=AttributeArray())***

  *Pre-allocate a matrix and (optionally) attach dimension scales and attributes. Insert rows or columns using insert_into(…)*

- **void allocate_vector (const StrStrSizet &iterator_id, const StringArray &location, ResultsOutputType stored_type, const int &len, const DimScaleMap &scales=DimScaleMap(), const AttributeArray &attrs=AttributeArray())***

  *Pre-allocate a vector and (optionally) attach dimension scales and attributes. Insert elements insert_into(…)*

- **template<typename StoredType> void insert_into (const StrStrSizet &iterator_id, const StringArray &location, const StoredType &data, const int &index, const bool &row=true) const**

  *Insert a row or column into a matrix that was pre-allocated using allocate_matrix.*

- **void add_metadata_to_method (const StrStrSizet &iterator_id, const AttributeArray &attrs)***

  *Associate key:value metadata with all the results and executions of a method.*

- **void add_metadata_to_execution (const StrStrSizet &iterator_id, const AttributeArray &attrs)***
Associate key:value metadata with all the results for this execution of a method.

- void `add_metadata_to_object` (const `StrStrSizet &iterator_id`, const `StringArray &location`, const `AttributeArray &attrs`)
  
  Associate key:value metadata with the object at the location.

- void `add_metadata_to_study` (const `AttributeArray &attrs`)
  
  Associate key:value metadata with the object at the location.

- template<typename StoredType>
  void `array_allocate` (const `StrStrSizet &iterator_id`, const `std::string &data_name`, size_t `array_size`, const `MetaDataType metadata=MetaDataType()`)

  allocate an entry with array of StoredType of array_size for future insertion; likely move to non-templated accessors for these

- template<typename StoredType>
  void `array_insert` (const `StrStrSizet &iterator_id`, const `std::string &data_name`, size_t `index`, const `StoredType &sent_data`)

  insert into a previously allocated array of StoredType at index specified; metadata must be specified at allocation

- template<typename StoredType>
  void `array_insert` (const `StrStrSizet &iterator_id`, const `std::string &data_name`, size_t `index`, `StringMultiArrayConstView sent_data`)

  specialization: insert a SMACV into a previously allocated array of StringArrayStoredType at index specified; metadata must be specified at allocation

- template<typename StoredType>
  void `insert` (const `StrStrSizet &iterator_id`, const `std::string &data_name`, const `StoredType &sent_data`, const `MetaDataType metadata=MetaDataType()`)

  insert data

- void `insert` (const `StrStrSizet &iterator_id`, const `std::string &data_name`, `StringMultiArrayConstView sma_labels`, const `MetaDataType metadata=MetaDataType()`)

Public Attributes

- `ResultsNames results_names`

  Copy of valid results names for when manager is passed around.

Private Member Functions

- `ResultsManager` (const `ResultsManager &`)
14.218. **RESULTS NAMES CLASS REFERENCE**

### 14.217.1 Detailed Description

Results manager for iterator final data.

The results manager provides the API for posting and retrieving iterator results data (and eventually run config/statistics). It can manage a set of underlying results databases, in or out of core, depending on configuration.

The key for a results entry is documented in results_types.hpp, e.g., `tuple<std::string, std::string, size_t, std::string>`

For now, using concrete types for most insertion, since underlying databases like HDF5 might need concrete types; though template parameter for array allocation and retrieval.

All insertions overwrite any previous data.

The documentation for this class was generated from the following files:

- ResultsManager.hpp
- ResultsManager.cpp

### 14.218 ResultsNames Class Reference

List of valid names for iterator results.

#### Public Member Functions

- **ResultsNames ()**
  
  *Default constructor.*

#### Public Attributes

- `size_t namesVersion = 0`
- `std::string best_cv = "Best Continuous Variables"`
- `std::string best_div = "Best Discrete Integer Variables"`
- `std::string best_dsv = "Best Discrete std::string Variables"`
- `std::string best_drv = "Best Discrete Real Variables"`
- `std::string best_fns = "Best Functions"`
- `std::string moments_std = "Moments: Standard"`
- `std::string moments_central = "Moments: Central"`
- `std::string moments_std_num = "Moments: Standard: Numerical"`
- `std::string moments_central_num = "Moments: Central: Numerical"`
- `std::string moments_std_exp = "Moments: Standard: Expansion"`
- `std::string moments_central_exp = "Moments: Central: Expansion"`
- `std::string moment_cis = "Moment Confidence Intervals"`
- `std::string extreme_values = "Extreme Values"`
- `std::string mapResp_prob = "Response to Probability Mapping"`
- `std::string mapResp_rel = "Response to Reliability Mapping"`
- `std::string mapResp_genrel = "Response to Generalized Reliability Mapping"`
- `std::string map_prob_resp = "Probability to Response Mapping"`
- `std::string map_rel_resp = "Reliability to Response Mapping"`
- `std::string map_genrel_resp = "Generalized Reliability to Response Mapping"
- `std::string pdf_histograms = "PDF Histograms"`
- `std::string correl_simple_all = "Simple Correlations (All)"`
• std::string correl_simple_io = "Simple Correlations (I/O)"
• std::string correl_partial_io = "Partial Correlations (I/O)"
• std::string correl_simple_rank_all = "Simple Rank Correlations (All)"
• std::string correl_simple_rank_io = "Simple Rank Correlations (I/O)"
• std::string correl_partial_rank_io = "Partial Rank Correlations (I/O)"
• std::string pce_coeffs = "PCE Coefficients: Standardized"
• std::string pce_coeff_labels = "PCE Coefficient Labels"
• std::string cv_labels = "Continuous Variable Labels"
• std::string div_labels = "Discrete Integer Variable Labels"
• std::string dsv_labels = "Discrete std::string Variable Labels"
• std::string drv_labels = "Discrete Real Variable Labels"
• std::string fn_labels = "Function Labels"

14.218.1 Detailed Description

List of valid names for iterator results.

All data in the ResultsNames class is public, basically just a struct
The documentation for this class was generated from the following file:

• ResultsManager.hpp

14.219 RichExtrapVerification Class Reference

Class for Richardson extrapolation for code and solution verification.

Inheritance diagram for RichExtrapVerification:

```
Iterator
     |
     v
Analyzer
     |
     v
Verification
     v
RichExtrapVerification
```

Public Member Functions

• RichExtrapVerification (ProblemDescDB &problem_db, Model &model)
  constructor
• ~RichExtrapVerification ()
  destructor
• void core_run ()
  core portion of run; implemented by all derived classes and may include pre/post steps in lieu of separate pre/post
• void print_results (std::ostream &s, short results_state=FINAL_RESULTS)
  print the final iterator results
Private Member Functions

- void `estimate_order()`
  perform a single estimation of `convOrder` using `extrapolation()`

- void `converge_order()`
  iterate using `extrapolation()` until `convOrder` stabilizes

- void `converge_qoi()`
  iterate using `extrapolation()` until QOIs stabilize

- void `extrapolation` (const `RealVector &refine_triple`, `RealMatrix &qoi_triples`)
  estimate `convOrder` from refinement and quantity of interest (QOI) triples

- void `extrapolate_result` (const `RealVector &refine_triple`, const `RealMatrix &qoi_triples`)
  predict the converged value based on the convergence rate and the value of Phi

Private Attributes

- unsigned short `studyType`
  internal code for extrapolation study type: `SUBMETHOD.{CONVERGE_ORDER,CONVERGE_QOI,ESTIMATE_ORDER}`

- size_t `numFactors`
  number of refinement factors defined from active state variables

- `RealVector initialCVars`
  initial reference values for refinement factors

- size_t `factorIndex`
  the index of the active factor

- `Real refinementRate`
  rate of mesh refinement (default = 2.)

- `RealMatrix convOrder`
  the orders of convergence of the QOIs (numFunctions by numFactors)

- `RealMatrix extrapQOI`
  the extrapolated value of the QOI (numFunctions by numFactors)

- `RealMatrix numErrorQOI`
  the numerical uncertainty associated with level of refinement (numFunctions by numFactors)

- `RealVector refinementRefPt`
  This is a reference point reported for the converged extrapQOI and numErrorQOI. It currently corresponds to the coarsest mesh in the final refinement triple.

Additional Inherited Members

14.219.1 Detailed Description

Class for Richardson extrapolation for code and solution verification.

The `RichExtrapVerification` class contains several algorithms for performing Richardson extrapolation.
14.219.2 Member Function Documentation

void core_run ( ) [virtual]

core portion of run; implemented by all derived classes and may include pre/post steps in lieu of separate pre/post
virtual run function for the iterator class hierarchy. All derived classes need to redefine it.
Reimplemented from Iterator.
References Dakota::abort_handler(), Model::continuous_variables(), RichExtrapVerification::converge_order(),
RichExtrapVerification::converge_qoi(), RichExtrapVerification::convOrder, RichExtrapVerification::estimate_order(),
RichExtrap Verification::extrapQOI, RichExtrapVerification::initialCVars, Iterator::iteratedModel, RichExtrapVerification::
numErrorQOI, RichExtrapVerification::numFactors, Analyzer::numFunctions, Iterator::outputLevel, RichExtrap-
Verification::refinementRefPt, and RichExtrapVerification::studyType.

void print_results ( std::ostream & s, short results_state = FINAL_RESULTS ) [virtual]

print the final iterator results
This virtual function provides additional iterator-specific final results outputs beyond the function evaluation
summary printed in finalize_run().
Reimplemented from Verification.
References Model::continuous variable_labels(), RichExtrapVerification::convOrder, Dakota::copy_data(), Rich-
ExtrapVerification::extrapQOI, Iterator::iteratedModel, RichExtrapVerification::numErrorQOI, Verification::print-
results(), RichExtrapVerification::refinementRate, RichExtrapVerification::refinementRefPt, and Model::response-
_labels().

void estimate_order ( ) [private]

perform a single estimation of convOrder using extrapolation()
This algorithm executes a single refinement triple and returns convergence order estimates.
References RichExtrapVerification::extrapolate_result(), RichExtrapVerification::extrapolation(), RichExtrap-
Verification::extrapQOI, RichExtrapVerification::factorIndex, RichExtrapVerification::initialCVars, RichExtrap-
Verification::numErrorQOI, RichExtrapVerification::numFactors, Analyzer::numFunctions, RichExtrapVerification-
::refinementRate, and RichExtrapVerification::refinementRefPt.
Referenced by RichExtrapVerification::core_run().

void converge_order ( ) [private]

iterate using extrapolation() until convOrder stabilizes
This algorithm continues to refine until the convergence order estimate converges.
References Iterator::convergenceTol, RichExtrapVerification::convOrder, Dakota::copy_data(), RichExtrap-
Verification::extrapolate_result(), RichExtrapVerification::extrapolation(), RichExtrapVerification::extrapQOI, Rich-
ExtrapVerification::factorIndex, RichExtrapVerification::initialCVars, Iterator::maxIterations, RichExtrapVerification-
::numErrorQOI, RichExtrapVerification::numFactors, Analyzer::numFunctions, Iterator::outputLevel, RichExtrap-
Verification::refinementRate, and RichExtrapVerification::refinementRefPt.
Referenced by RichExtrapVerification::core_run().

void converge_qoi ( ) [private]

iterate using extrapolation() until QOIs stabilize
This algorithm continues to refine until the discretization error lies within a prescribed tolerance.
14.220  ROLOOptimizer Class Reference

Inheritance diagram for ROLOOptimizer:

```
ROLOptimizer
   ↓
Minimizer
   ↓
Optimizer
   ↓
Iterator
```

Public Member Functions

- **ROLOptimizer (ProblemDescDB &problem_db, Model &model)**
  *Standard constructor.*
- **ROLOptimizer (const String &method_name, Model &model)**
  *Alternate constructor for Iterator instantiations by name.*
- **~ROLOptimizer ()**
  *Destructor.*
- **void initialize_run () override**
  *Initializes the ROLOptimizer with values available after the chain of constructors has finished.*
- **void core_run () override**
  *Iterates the ROL solver to determine the optimal solution.*
- **void reset_solver_options (const Teuchos::ParameterList &)**
  *Support resetting ROL solver options.*
- **ROL::OptimizationProblem< Real > & get_rol_problem ()**
  *Accessor for the underlying ROL Problem.*

Protected Member Functions

- **void set_problem ()**
  *Helper function called during construction to extract problem information from the Model and set it for ROL.*
- **void set_rol_parameters ()**
  *Convenience function to map Dakota input and power-user parameters to ROL.*
Protected Attributes

- Teuchos::ParameterList optSolverParams
  Parameters for the ROL::OptimizationSolver.
- unsigned short problemType
  ROL problem type.
- Teuchos::RCP< std::vector< Real > > rolX
  Handle to ROL’s solution vector.
- Teuchos::RCP< ROL::StdVector< Real > > lowerBounds
  Handle to ROL’s lower bounds vector.
- Teuchos::RCP< ROL::StdVector< Real > > upperBounds
  Handle to ROL’s upper bounds vector.
- ROL::OptimizationProblem< Real > optProblem
  Handle to ROL::OptimizationProblem, part of ROL’s simplified interface.

Additional Inherited Members

14.220.1 Detailed Description

ROLOptimizer specializes DakotaOptimizer to construct and run a ROL solver appropriate for the type of problem specified by the user.

14.220.2 Constructor & Destructor Documentation

ROLOptimizer ( ProblemDescDB & problem_db, Model & model )

Standard constructor.

Implementation of ROLOptimizer class.
References ROLOptimizer::set_problem(), and ROLOptimizer::set_rol_parameters().
The documentation for this class was generated from the following files:

- ROLOptimizer.hpp
- ROLOptimizer.cpp

14.221 ROLTraits Class Reference

Inheritance diagram for ROLTraits:

```
  TraitsBase
  ├── ROLTraits
```
14.222. SCALINGMODEL CLASS REFERENCE

Public Types

- typedef std::vector<Real> VecT
  
  ROL default data type to be used by Dakota data adapters.

Public Member Functions

- ROLTraits ()
  
  Default constructor.
- virtual ~ROLTraits ()
  
  Destructor.
- bool supports_continuous_variables ()
  
  Return flag indicating ROL supports continuous variables.
- bool supports_linear_equality ()
  
  Return flag indicating ROL supports linear equalities.
- bool supports_linear_inequality ()
  
  Return flag indicating ROL supports linear inequalities.
- bool supports_nonlinear_equality ()
  
  Return flag indicating ROL supports nonlinear equalities.
- NONLINEAR_EQUALITY_FORMAT nonlinear_equality_format ()
  
  Return ROL format for nonlinear equality constraints.
- bool supports_nonlinear_inequality ()
  
  Return flag indicating ROL supports nonlinear inequalities.
- NONLINEAR_INEQUALITY_FORMAT nonlinear_inequality_format ()
  
  Return ROL format for nonlinear inequality constraints.

14.221.1 Detailed Description

ROLTraits defines the types of problems and data formats ROL supports by overriding the default traits accessors in TraitsBase.

The documentation for this class was generated from the following file:

- ROLOptimizer.hpp

14.222 ScalingModel Class Reference

Scaling specialization of RecastModel.

Inheritance diagram for ScalingModel:

```
Model
  
RecastModel
  
ScalingModel
```
Public Member Functions

- **ScalingModel (Model &sub_model)**
  standard constructor
- **~ScalingModel ()**
  destructor
- **RealVector cv_scaled2native (const RealVector &scaled_cv)**

Public members for help in final results recovery

- **void resp_scaled2native (const Variables &native_vars, Response &updated_resp)**
  map responses from scaled to native space, updating provided Response in-place (on entry it’s scaled response, on exit it’s native)
- **void secondary_resp_scaled2native (const RealVector &scaled_nln, const ShortArray &asv, RealVector &native_fns)**
  Use scaled nonlinear constraints (sized total functions) to update the nonlinear constraints portion of the passed native_fns array.
- **void response_modify_s2n (const Variables &native_vars, const Response &scaled_response, Response &native_response, int start_offset, int num_responses, bool response_unscale=true)**
  map responses from scaled to native space
- **ActiveSet default_active_set ()**

Protected Member Functions

- **void assign_instance ()**
  assign static pointer instance to this for use in static transformation functions
- **void initialize_scaling (Model &sub_model)**
  initialize scaling types, multipliers, and offsets; perform error checking
- **void compute_scaling (int object_type, int auto_type, int num_vars, RealVector &lbs, RealVector &ubs, RealVector &targets, const UShortArray &spec_types, const RealVector &scales, UShortArray &scale_types, RealVector &scale_mults, RealVector &scale_offsets)**
  general helper function for initializing scaling types and factors on a vector of variables, functions, constraints, etc.
- **RealMatrix lin_coeffs_modify_n2s (const RealMatrix &native_coeffs, const RealVector &cv_multipliers, const RealVector &lin_multipliers)**
  general linear coefficients mapping from native to scaled space
- **bool compute_scale_factor (const Real lower_bound, const Real upper_bound, Real *multiplier, Real *offset)**
  automatically compute a single scaling factor – bounds case
- **bool compute_scale_factor (const Real target, Real *multiplier)**
  automatically compute a single scaling factor – target case
- **void print_scaling (const String &info, const UShortArray &scale_types, const RealVector &scale_mults, const RealVector &scale_offsets, const StringArray &labels)**
  print scaling information for a particular response type in tabular form
- **bool need_resp_trans_byvars (const ShortArray &asv, int start_index, int num_resp)**
  determine if response transformation is needed due to variable transformations
- **RealVector modify_n2s (const RealVector &native_vars, const UShortArray &scale_types, const RealVector &multipliers, const RealVector &offsets)**
general `RealVector` mapping from native to scaled variables vectors:

- `RealVector modify_s2n` (const `RealVector` &`scaled_vars`, const `UShortArray` &`scale_types`, const `RealVector` &`multipliers`, const `RealVector` &`offsets`) const

  general `RealVector` mapping from scaled to native variables (and values)

- `void response_modify_n2s` (const `Variables` &`scaled_vars`, const `Response` &`native_response`, `Response` &`scaled_response`, int start_offset, int num_responses) const

  map responses from native to scaled variable space

**Static Protected Member Functions**

- static bool `scaling_active` (const `UShortArray` &`scale_types`)

  check whether the passed scale types include any active (!= none) scale types

- static void `variables_scaler` (const `Variables` &`scaled_vars`, `Variables` &`native_vars`)

  `RecastModel` callback for variables scaling: transform variables from scaled to native (user) space.

- static void `variables_unscaler` (const `Variables` &`native_vars`, `Variables` &`scaled_vars`)

  `RecastModel` callback for inverse variables scaling: transform variables from native (user) to scaled space.

- static void `primary_resp_scaler` (const `Variables` &`native_vars`, const `Variables` &`scaled_vars`, const `Response` &`native_response`, `Response` &`iterator_response`)

  `RecastModel` callback for primary response scaling: transform responses (grads, Hessians) from native (user) to scaled space.

- static void `secondary_resp_scaler` (const `Variables` &`native_vars`, const `Variables` &`scaled_vars`, const `Response` &`native_response`, `Response` &`scaled_response`)

  `RecastModel` callback for secondary response scaling: transform constraints (grads, Hessians) from native (user) to scaled space.

**Protected Attributes**

- bool `varsScaleFlag`

  flag for variables scaling

- bool `primaryRespScaleFlag`

  flag for primary response scaling

- bool `secondaryRespScaleFlag`

  flag for secondary response scaling

- `UShortArray` `cvScaleTypes`

  scale flags for continuous vars.

- `RealVector` `cvScaleMultipliers`

  scales for continuous variables

- `RealVector` `cvScaleOffsets`

  offsets for continuous variables

- `UShortArray` `responseScaleTypes`

  scale flags for all responses

- `RealVector` `responseScaleMultipliers`

  scales for all responses

- `RealVector` `responseScaleOffsets`
CHAPTER 14. CLASS DOCUMENTATION

offsets for all responses (zero < for functions, not for nonlin con)

- UShortArray linearIneqScaleTypes
  scale flags for linear ineq
- RealVector linearIneqScaleMultipliers
  scales for linear ineq constrs.
- RealVector linearIneqScaleOffsets
  offsets for linear ineq constrs.
- UShortArray linearEqScaleTypes
  scale flags for linear eq.
- RealVector linearEqScaleMultipliers
  scales for linear constraints
- RealVector linearEqScaleOffsets
  offsets for linear constraints

Static Protected Attributes

- static ScalingModel ∗ scaleModelInstance
  static pointer to this class for use in static callbacks

Additional Inherited Members

14.222.1 Detailed Description

Scaling specialization of RecastModel.

Specialization of RecastModel to scale Variables and/or Responses This class provides a simple constructor that forwards to the more complicated RecastModel API

14.222.2 Constructor & Destructor Documentation

ScalingModel ( Model & sub_model )

standard constructor

This constructor computes various indices and mappings, then updates the properties of the RecastModel. References Model::cv(), ScalingModel::vScaleTypes, Model::div(), Model::drv(), RecastModel::init_maps(), ScalingModel::initialize_scaling(), RecastModel::inverse_mappings(), Model::modelId, Model::num_primary_fns(), Model::num_secondary_fns(), Model::outputLevel, ScalingModel::primary_resp_scaler(), Model::primary_response_fn_sense(), Model::primary_response_fn_weights(), ScalingModel::primaryRespScaleFlag, RecastModel::recast_model_id(), ScalingModel::responseScaleTypes, RecastModel::root_model_id(), ScalingModel::secondary_resp_scaler(), ScalingModel::secondaryRespScaleFlag, ScalingModel::variables_scaler(), ScalingModel::variables_unscaler(), and ScalingModel::varsScaleFlag.

14.222.3 Member Function Documentation

RealVector cv_scaled2native ( const RealVector & scaled_cv ) const

Public members for help in final results recovery

recover native variable values from the scaled space
Since this convenience function is public, it must have a fall-through to return a copy for when this scaling type isn’t active.

References ScalingModel::cvScaleMultipliers, ScalingModel::cvScaleOffsets, ScalingModel::cvScaleTypes, ScalingModel::modify_s2n(), and ScalingModel::varsScaleFlag.

Referenced by LeastSq::post_run(), and Optimizer::post_run().

```cpp
void resp_scaled2native ( const Variables & native_vars, Response & updated.resp ) const
```

map responses from scaled to native space, updating provided Response in-place (on entry it’s scaled response, on exit it’s native)

Since this convenience function is public, it must behave correctly when this scale type isn’t active. It does, because it modifies in-place

References Response::active_set(), Request::vector(), Dakota::copy(), ScalingModel::need_resp_trans_byvars(), ScalingModel::modify_s2n(), ScalingModel::need_resp_trans_byvars(), ScalingModel::cvScaleMultipliers, ScalingModel::cvScaleOffsets, ScalingModel::cvScaleTypes, and ScalingModel::responseScaleMultipliers.

Referenced by Optimizer::post_run().

```cpp
void secondary_resp_scaled2native ( const RealVector & scaled.nln_cons, const ShortArray & asv, RealVector & native.fns ) const
```

Use scaled nonlinear constraints (sized total functions) to update the nonlinear constraints portion of the passed native.fns array.

Since this convenience function is public, it must have a fall-through to return a copy for when this scaling type isn’t active.

References Dakota::copy_data_partial(), ScalingModel::modify_s2n(), ScalingModel::need_resp_trans_byvars(), Model::num_nonlinear_eq_constraints(), Model::num_nonlinear_ineq_constraints(), Model::num_primary_fns(), ScalingModel::primaryRespScaleFlag, ScalingModel::response_modify_s2n(), ScalingModel::secondaryRespScaleFlag, and Response::update_partial().

Referenced by Optimizer::post_run().

```cpp
void response_modify_s2n ( const Variables & native_vars, const Response & scaled_response, Response & native_response, int start_offset, int num_responses, bool unscale_resp = true ) const
```

map responses from scaled to native space

Unscaling response mapping: modifies response from scaled (iterator) to native (user) space. Maps num_responses starting at response_offset. If response_unscale = false, only variables will be unscaled, and responses left in scaled space.

References Response::active_set(), Variables::acv(), Variables::all_continuous_variable_ids(), Variables::all_continuous_variables(), Variables::continuous_variable_ids(), Variables::continuous_variables(), Dakota::copy_data(), Variables::cv(), ScalingModel::cvScaleMultipliers, ScalingModel::cvScaleOffsets, ScalingModel::cvScaleTypes, ActiveSet::derivative_vector(), Dakota::find_index(), Response::function_gradient_view(), Response::function_gradients(), Response::function_hessian_view(), Response::function_hessians(), Response::function_labels(), Response::function_value(), Response::function_values(), Variables::icv(), Variables::inactive_continuous_variable_ids(), Variables::inactive_continuous_variables(), Model::num_primary_fns(), Model::outputLevel, ActiveSet::request_vector(), ScalingModel::responseScaleMultipliers, ScalingModel::responseScaleOffsets, ScalingModel::responseScaleTypes, Dakota::SCALINGLNLOGBASE, Dakota::SCALINGLOGBASE, and Dakota::write_precision.

Referenced by LeastSq::get_confidence_intervals(), and ScalingModel::resp_scaled2native().
CHAPTER 14. CLASS DOCUMENTATION

```cpp
void initialize_scaling ( Model & sub_model ) [protected]
initialize scaling types, multipliers, and offsets; perform error checking
  Initialize scaling types, multipliers, and offsets. Update the iteratedModel appropriately
  References Dakota::abort_handler(), ScalingModel::compute_scaling(), Model::continuous_lower_bounds(),
  Model::continuous_upper_bounds(), Model::continuous_variable_labels(), Model::continuous_variables(), Dakota-
  ::copy_data(), Model::cv(), ScalingModel::cvScaleMultipliers, ScalingModel::cvScaleOffsets, ScalingOptions-
  ::cvScales, ScalingOptions::cvScaleTypes, ScalingModel::cvScaleTypes, ScalingModel::lin_coeffs_modify_n2s(),
  Model::linear_eq_constraint_coeffs(), Model::linear_eq_constraint_targets(), Model::linear_ineq_constraint_coeffs(),
  Model::linear_ineq_constraint_lower_bounds(), Model::linear_ineq_constraint_upper_bounds(), ScalingModel::linear-
  EqScaleMultipliers, ScalingModel::linearEqScaleOffsets, ScalingModel::linearEqScaleTypes, ScalingModel::linear-
  IneqScaleMultipliers, ScalingModel::linearIneqScaleOffsets, ScalingModel::linearIneqScaleTypes, ScalingOptions-
  ::linEqScales, ScalingOptions::linEqScaleTypes, ScalingOptions::linIneqScales, ScalingOptions::linIneqScaleTypes,
  ScalingModel::modify_n2s(), ScalingOptions::nlnEqScales, ScalingOptions::nlnEqScaleTypes, ScalingOptions-
  ::nlnIneqScales, ScalingOptions::nlnIneqScaleTypes, Model::nonlinear_eq_constraint_targets(), Model::nonlinear-
  ineq_constraint_lower_bounds(), Model::nonlinear_ineq_constraint_upper_bounds(), Model::num_linear_eq_constraints(),
  Model::num_linear_ineq_constraints(), Model::num_nonlinear_eq_constraints(), Model::num_nonlinear_ineq_constraints(),
  Model::num_primary_fns(), Model::numFns, Model::outputLevel, ScalingModel::primaryRespScaleFlag, Scaling-
  Model::print_scaling(), ScalingOptions::priScales, ScalingOptions::priScaleTypes, Model::response_labels(), Scaling-
  Model::responseScaleMultipliers, ScalingModel::responseScaleOffsets, ScalingModel::responseScaleTypes, Scaling-
  Model::scaling_active(), Model::scalingOptS, ScalingModel::secondaryRespScaleFlag, Model::supports_derivative-
  estimation(), Model::supportsEstimDerivs, and ScalingModel::varsScaleFlag.
  Referenced by ScalingModel::ScalingModel().

RealMatrix lin_coeffs_modify_n2s ( const RealMatrix & src_coeffs, const RealVector & cv_multipliers,
const RealVector & lin_multipliers ) const [protected]
general linear coefficients mapping from native to scaled space
  compute scaled linear constraint matrix given design variable multipliers and linear scaling multipliers. Only
  scales components corresponding to continuous variables so for src_coeffs of size MxN, lin_multipliers.size() <=
  M, cv_multipliers.size() <= N
  Referenced by ScalingModel::initialize_scaling().

void variables_scaler ( const Variables & scaled_vars, Variables & native_vars ) [static],
[protected]
RecastModel callback for variables scaling: transform variables from scaled to native (user) space.
  Variables map from iterator/scaled space to user/native space using a RecastModel.
  References Variables::continuous_variable_labels(), Variables::continuous_variables(), ScalingModel::cvScale-
  Multipliers, ScalingModel::cvScaleOffsets, ScalingModel::cvScaleTypes, ScalingModel::modify_s2n(), Model-
  ::outputLevel, and ScalingModel::scaleModelInstance.
  Referenced by ScalingModel::ScalingModel().

void secondary_resp_scaler ( const Variables & native_vars, const Variables & scaled_vars, const Response &
native_response, Response & iterator_response ) [static], [protected]
RecastModel callback for secondary response scaling: transform constraints (grads, Hessians) from native (user)
to scaled space.
  Constraint function map from user/native space to iterator/scaled/combined space using a RecastModel.
```
References Response::active_set_request_vector(), ScalingModel::need_resp_trans_byvars(), Model::num_nonlinear_eq_constraints(), Model::num_nonlinear_ineq_constraints(), Model::num_primary_fns(), Model::output_level, ScalingModel::response_modify_n2s(), ScalingModel::scaleModelInstance, ScalingModel::secondaryRespScaleFlag, and Response::update_partial().

Referenced by ScalingModel::ScalingModel().

```cpp
bool need_resp_trans_byvars ( const ShortArray & avs, int start_index, int num_res ) const
[protected]
```
determine if response transformation is needed due to variable transformations

- Determine if variable transformations present and derivatives requested, which implies a response transformation is necessary

  References ScalingModel::varsScaleFlag.

Referenced by ScalingModel::primary_resp_scaler(), ScalingModel::resp_scaled2native(), ScalingModel::secondary_resp_scaled2native(), and ScalingModel::secondary_resp_scaler().

```cpp
RealVector modify_n2s ( const RealVector & native_vars, const UShortArray & scale_types, const RealVector & multipliers, const RealVector & offsets ) const [protected]
```
general RealVector mapping from native to scaled variables vectors:

- general RealVector mapping from native to scaled variables; loosely, in greatest generality: scaled_var = log((native_var - offset) / multiplier)

  References Dakota::SCALING_LN_LOGBASE.

Referenced by ScalingModel::initialize_scaling(), and ScalingModel::variables_unscaler().

```cpp
RealVector modify_s2n ( const RealVector & scaled_vars, const UShortArray & scale_types, const RealVector & multipliers, const RealVector & offsets ) const [protected]
```
general RealVector mapping from scaled to native variables (and values)

- general RealVector mapping from scaled to native variables and/or vals; loosely, in greatest generality: scaled_var = (LOG_BASE^scaled_var) * multiplier + offset

  References Dakota::SCALING_LOGBASE.

Referenced by ScalingModel::cv_scaled2native(), ScalingModel::secondary_resp_scaled2native(), and ScalingModel::variables_scaler().

```cpp
void response_modify_n2s ( const Variables & native_vars, const Response & native_response, Response & recast_response, int start_offset, int num_responses ) const [protected]
```
map responses from native to scaled variable space

- Scaling response mapping: modifies response from a model (user/native) for use in iterators (scaled). Maps num_responses starting at response_offset

  References Response::active_set(), Variables::acv(), Variables::all_continuous_variable_ids(), Variables::all_continuous_variables(), Variables::continuous_variable_ids(), Variables::continuous_variables(), Dakota::copy_data(), Variables::cv(), ScalingModel::cvScaleMultipliers, ScalingModel::cvScaleOffsets, ScalingModel::cvScaleTypes, ActiveSet::derivative_vector(), Dakota::find_index(), Response::function_gradient_view(), Response::function_gradients(), Response::function_hessian_view(), Response::function_hessians(), Response::function_labels(), Response::function_value(), Response::function_values(), Variables::icv(), Variables::inactive_continuous_variable_ids(), Variables::inactive_continuous_variables(), Model::num_primary_fns(), Model::output_level, ActiveSet::request_vector(), ScalingModel::responseScaleMultipliers, ScalingModel::responseScaleOffsets, ScalingModel::responseScaleTypes, Dakota::SCALING_LN_LOGBASE, and Dakota::write_precision.

Referenced by ScalingModel::primary_resp_scaler(), and ScalingModel::secondary_resp_scaler().
### 14.222.4 Member Data Documentation

ScalingModel * scaleModelInstance  [static], [protected]

static pointer to this class for use in static callbacks
initialization of static needed by RecastModel
Referenced by ScalingModel::assign_instance(), ScalingModel::primary_resp_scaler(), ScalingModel::secondary_resp_scaler(), ScalingModel::variables_scaler(), and ScalingModel::variables_unscaler().

The documentation for this class was generated from the following files:

- ScalingModel.hpp
- ScalingModel.cpp

### 14.223 ScalingOptions Class Reference

Simple container for user-provided scaling data, possibly expanded by replicates through the models.

**Public Member Functions**

- **ScalingOptions ()**
  *default ctor: no scaling specified*

- **ScalingOptions (const ProblemDescDB &problem_db, const SharedResponseData &srd)**
  *standard ctor: scaling from problem DB*

**Public Attributes**

- **UShortArray cvScaleTypes**
  *continuous variables scale types*

- **RealVector cvScales**
  *continuous variables scale values*

- **UShortArray priScaleTypes**
  *primary response scale types*

- **RealVector priScales**
  *primary response scale values*

- **UShortArray nlnIneqScaleTypes**
  *nonlinear inequality constraint scale types*

- **RealVector nlnIneqScales**
  *nonlinear inequality constraint scale values*

- **UShortArray nlnEqScaleTypes**
  *nonlinear equality constraint scale types*

- **RealVector nlnEqScales**
  *nonlinear equality constraint scale values*

- **UShortArray linIneqScaleTypes**
  *linear inequality constraint scale types*

- **RealVector linIneqScales**
  *linear inequality constraint scale values*
• UShortArray linEqScaleTypes
  linear equality constraint scale types
• RealVector linEqScales
  linear equality constraint scale values

Static Private Member Functions
• static void default_scale_types (const RealVector &scale_values, UShortArray &scale_types)
  when values are given, but not types, initialize type to value
• static UShortArray scale_str2enum (const StringArray &scale_strs)
  convert problem DB strings to unsigned shorts

14.223.1 Detailed Description
Simple container for user-provided scaling data, possibly expanded by replicates through the models.
The documentation for this class was generated from the following files:
• ScalingOptions.hpp
• ScalingOptions.cpp

14.224 ScilabInterface Class Reference
Inheritance diagram for ScilabInterface:

```
+---------------------
| Interface           |
+---------------------
    |                   |
    | ApplicationInterface|
    |                   |
    | DirectApplicInterface|
+---------------------
    |                   |
    | ScilabInterface    |
+---------------------
```

Public Member Functions
• ScilabInterface (const ProblemDescDB &problem_db)
  Constructor: start Matlab engine.
• ~ScilabInterface ()
  Destructor: close Matlab engine.

Protected Member Functions
• virtual int derived_map_ac (const String &ac_name)
  execute an analysis code portion of a direct evaluation invocation
• int scilab_engine_run (const String &ac_name)
  principal Scilab execute function
Protected Attributes

- int scilabEngine
  identifier for the running Scilab engine

14.224.1 Detailed Description

Specialization of DirectApplicInterface to link to Scilab analysis drivers. Includes convenience functions to map data to/from Scilab.

The documentation for this class was generated from the following files:

- ScilabInterface.hpp
- ScilabInterface.cpp

14.225 SensAnalysisGlobal Class Reference

Class for a utility class containing correlation calculations and variance-based decomposition.

Public Member Functions

- SensAnalysisGlobal ()
  constructor

- ~SensAnalysisGlobal ()
  destructor

- void compute_correlations (const VariablesArray &vars_samples, const IntResponseMap &resp_samples, const StringSetArray &dss_vals)
  computes four correlation matrices for input and output data simple, partial, simple rank, and partial rank

- void compute_correlations (const RealMatrix &vars_samples, const IntResponseMap &resp_samples)
  computes four correlation matrices for input and output data simple, partial, simple rank, and partial rank

- void archive_correlations (const StrStrSizet &run_identifier, ResultsManager &iterator_results, StringMultiArrayConstView cv_labels, StringMultiArrayConstView div_labels, StringMultiArrayConstView dsv_labels, StringMultiArrayConstView drv_labels, const StringArray &resp_labels, const size_t &inc_id=0) const
  save correlations to database

- bool correlations_computed () const
  returns corrComputed to indicate whether compute_correlations() has been invoked

- void print_correlations (std::ostream &s, StringMultiArrayConstView cv_labels, StringMultiArrayConstView div_labels, StringMultiArrayConstView dsv_labels, StringMultiArrayConstView drv_labels, const StringArray &resp_labels) const
  prints the correlations computed in compute_correlations()

Private Member Functions

- size_t find_valid_samples (const IntResponseMap &resp_samples, BoolDeque &valid_sample)
  find samples with finite response (any sample with any Nan or +/-Inf observation will be dropped)

- void valid_sample_matrix (const VariablesArray &vars_samples, const IntResponseMap &resp_samples, const StringSetArray &dss_vals, const BoolDeque is_valid_sample, RealMatrix &valid_data)
  extract a compact valid sample (vars resp) matrix from the passed data
• **void valid_sample_matrix** (const RealMatrix &vars_samples, const IntResponseMap &resp_samples, const BoolDeque is_valid_sample, RealMatrix &valid_samples)
  *extract a compact valid sample (vars/resp) matrix from the passed data*

• **void values_to_ranks** (RealMatrix &valid_data)
  *replace sample values with their ranks, in-place*

• **void center_rows** (RealMatrix &data_matrix)
  *center the passed matrix by its mean, in-place*

• **void correl_adjust** (Real &corr_value)
  *if result was NaN/Inf, preserve it, otherwise truncate to [-1.0, 1.0]*

• **void simple_corr** (RealMatrix &total_data, const int &num_in, RealMatrix &corr_matrix)
  *computes simple correlations, populating corr_matrix*

• **void partial_corr** (RealMatrix &total_data, const int num_in, const RealMatrix &simple_corr_mat, RealMatrix &corr_matrix, bool &numerical_issues)
  *computes partial correlations, populating corr_matrix and numerical_issues*

• **bool has_nan_or_inf** (const RealMatrix &corr) const
  *Return true if there are any NaN or Inf entries in the matrix.*

**Static Private Member Functions**

• **static bool rank_sort** (const int &x, const int &y)
  *sort algorithm to compute ranks for rank correlations*

**Private Attributes**

• RealMatrix **simpleCorr**
  *matrix to hold simple raw correlations*

• RealMatrix **simpleRankCorr**
  *matrix to hold simple rank correlations*

• RealMatrix **partialCorr**
  *matrix to hold partial raw correlations*

• RealMatrix **partialRankCorr**
  *matrix to hold partial rank correlations*

• size_t **numFns**
  *number of responses*

• size_t **numVars**
  *number of inputs*

• bool **numericalIssuesRaw**
  *flag indicating numerical issues in partial raw correlation calculations*

• bool **numericalIssuesRank**
  *flag indicating numerical issues in partial rank correlation calculations*

• bool **corrComputed**
  *flag indicating whether correlations have been computed*
Static Private Attributes

- static RealArray rawData = RealArray()
  
  array to hold temporary data before sort

14.225.1 Detailed Description

Class for a utility class containing correlation calculations and variance-based decomposition.

This class provides code for several of the sampling methods both in the NonD branch and in the PStudyD-ACE branch. Currently, the utility functions provide global sensitivity analysis through correlation calculations (e.g. simple, partial, rank, raw) as well as variance-based decomposition.

14.225.2 Member Function Documentation

void compute_correlations ( const VariablesArray & vars_samples, const IntResponseMap & resp_samples, const StringSetArray & dss_vals )

computes four correlation matrices for input and output data simple, partial, simple rank, and partial rank

This version is used when full variables objects are being processed. Calculates simple correlation, partial correlation, simple rank correlation, and partial rank correlation coefficients.

References Dakota::abort_handler(), SensAnalysisGlobal::corrComputed, SensAnalysisGlobal::find_valid_samples(), SensAnalysisGlobal::numericalIssuesRank, SensAnalysisGlobal::numericalIssuesRaw, SensAnalysisGlobal::numFns, SensAnalysisGlobal::numVars, SensAnalysisGlobal::partial_corr(), SensAnalysisGlobal::partialCorr, SensAnalysisGlobal::partialRankCorr, SensAnalysisGlobal::simple_corr(), SensAnalysisGlobal::simpleCorr, SensAnalysisGlobal::simpleRankCorr, SensAnalysisGlobal::valid_sample_matrix(), and SensAnalysisGlobal::values_to_ranks().

Referenced by NonDSampling::compute_statistics(), ParamStudy::post_run(), FSUDesignCompExp::post_run(), and DDACEDesignCompExp::post_run().

void compute_correlations ( const RealMatrix & vars_samples, const IntResponseMap & resp_samples )

computes four correlation matrices for input and output data simple, partial, simple rank, and partial rank

This version is used when compact samples matrix is being processed. Calculates simple correlation, partial correlation, simple rank correlation, and partial rank correlation coefficients.

References Dakota::abort_handler(), SensAnalysisGlobal::corrComputed, SensAnalysisGlobal::find_valid_samples(), SensAnalysisGlobal::numericalIssuesRank, SensAnalysisGlobal::numericalIssuesRaw, SensAnalysisGlobal::numFns, SensAnalysisGlobal::numVars, SensAnalysisGlobal::partial_corr(), SensAnalysisGlobal::partialCorr, SensAnalysisGlobal::partialRankCorr, SensAnalysisGlobal::simple_corr(), SensAnalysisGlobal::simpleCorr, SensAnalysisGlobal::simpleRankCorr, SensAnalysisGlobal::valid_sample_matrix(), and SensAnalysisGlobal::values_to_ranks().

Referenced by NonDSampling::compute_statistics(), ParamStudy::post_run(), FSUDesignCompExp::post_run(), and DDACEDesignCompExp::post_run().

void values_to_ranks ( RealMatrix & valid_data ) [private]

replace sample values with their ranks, in-place

When converting values to ranks, uses the average ranks of any tied values

Referenced by SensAnalysisGlobal::compute_correlations().

void simple_corr ( RealMatrix & total_data, const int & num_in, RealMatrix & corr_matrix ) [private]

computes simple correlations, populating corr_matrix

Calculates simple correlation coefficients from a matrix of data (oriented factors x observations):
• num_corr is number of rows of total data

• num_in indicates whether only pairs of correlations should be calculated between pairs of columns (num_in vs. num_corr-num_in); if num_in = num_corr, correlations are calculated between all columns

References SensAnalysisGlobal::center_rows(), and SensAnalysisGlobal::correl_adjust(). Referenced by SensAnalysisGlobal::compute_correlations().

void partial_corr ( RealMatrix & total_data, const int num_in, const RealMatrix & simple_corr_mat, RealMatrix & corr_matrix, bool & numerical_issues ) [private]

computes partial correlations, populating corr_matrix and numerical_issues

Calculates partial correlation coefficients between num_in inputs and numRows() - num_in outputs.

References Dakota::abort_handler(), SensAnalysisGlobal::center_rows(), SensAnalysisGlobal::correl_adjust(), Dakota::qr(), Dakota::qr_rsolve(), and Dakota::svd().

Referenced by SensAnalysisGlobal::compute_correlations().

The documentation for this class was generated from the following files:

• SensAnalysisGlobal.hpp
• SensAnalysisGlobal.cpp

14.226 SeqHybridMetaIterator Class Reference

Method for sequential hybrid iteration using multiple optimization and nonlinear least squares methods on multiple models of varying fidelity.

Inheritance diagram for SeqHybridMetaIterator:

```
  Iterator
  ↓
MetaIterator
  ↓
SeqHybridMetaIterator
```

Public Member Functions

- SeqHybridMetaIterator (ProblemDescDB &problem_db)
  standard constructor
- SeqHybridMetaIterator (ProblemDescDB &problem_db, Model &model)
  alternate constructor
- ~SeqHybridMetaIterator ()
  destructor
CHAPTER 14. CLASS DOCUMENTATION

Protected Member Functions

- void core_run ()
  Performs the hybrid iteration by executing a sequence of iterators, using a similar sequence of models that may vary in fidelity.
- void print_results (std::ostream &s, short results_state=FINAL_RESULTS)
  print the final iterator results
- void derived_init_communicators (ParLevLIter pl_iter)
  derived class contributions to initializing the communicators associated with this Iterator instance
- void derived_set_communicators (ParLevLIter pl_iter)
  derived class contributions to setting the communicators associated with this Iterator instance
- void derived_free_communicators (ParLevLIter pl_iter)
  derived class contributions to freeing the communicators associated with this Iterator instance
- IntIntPair estimate_partition_bounds ()
  estimate the minimum and maximum partition sizes that can be utilized by this Iterator
- const Variables & variables_results () const
  return the final solution from selectedIterators (variables)
- const Response & response_results () const
  return the final solution from selectedIterators (response)
- void initialize_iterator (int job_index)
  used by IteratorScheduler to set the starting data for a run
- void pack_parameters_buffer (MPIPackBuffer &send_buffer, int job_index)
  used by IteratorScheduler to pack starting data for an iterator run
- void unpack_parameters_initialize (MPIUnpackBuffer &recv_buffer, int job_index)
  used by IteratorScheduler to unpack starting data and initialize an iterator run
- void pack_results_buffer (MPIPackBuffer &send_buffer, int job_index)
  used by IteratorScheduler to pack results data from an iterator run
- void unpack_results_buffer (MPIUnpackBuffer &recv_buffer, int job_index)
  used by IteratorScheduler to unpack results data from an iterator run
- void update_local_results (int job_index)
  used by IteratorScheduler to update local results arrays
- void declare_sources ()
  Declare sources to the evaluations database.

Private Member Functions

- void run_sequential ()
  run a sequential hybrid
- void run_sequential_adaptive ()
  run a sequential adaptive hybrid
- void partition_sets (size_t num_sets, int job_index, size_t &start_index, size_t &job_size)
  convert num_sets and job_index into a start_index and job_size for extraction from parameterSets
- void extract_parameter_sets (int job_index, VariablesArray &partial_param_sets)
  extract partial_param_sets from parameterSets based on job_index
void update_local_results (PRPArray &prp_results, int job_id)
  update the partial set of final results from the local iterator execution

void initialize_iterator (const VariablesArray &param_sets)
  called by unpack_parameters_initialize(MPIUnpackBuffer) and initialize_iterator(int) to update the active Model and Iterator

Private Attributes

- String seqHybridType
  empty (default) or "adaptive"
- StringArray methodStrings
  the list of method pointer or method name identifiers
- StringArray modelStrings
  the list of model pointer identifiers for method identification by name
- bool lightwtMethodCtor
  use of lightweight Iterator construction by name
- bool singlePassedModel
  use of constructor that enforces use of a single passed Model
- IteratorArray selectedIterators
  the set of iterators, one for each entry in methodStrings
- ModelArray selectedModels
  the set of models, one for each iterator (if not lightweight construction)
- size_t seqCount
  hybrid sequence counter: 0 to numIterators-1
- Real progressThreshold
  when the progress metric falls below this threshold, the sequential adaptive hybrid switches to the next method
- PRP2DArray prpResults
  2-D array of results corresponding to numIteratorJobs, one set of results per job (iterators may return multiple final solutions)
- VariablesArray parameterSets
  1-D array of variable starting points for the iterator jobs

Friends

- class IteratorScheduler
  protect scheduler callback functions from general access

Additional Inherited Members

14.226.1 Detailed Description

Method for sequential hybrid iteration using multiple optimization and nonlinear least squares methods on multiple models of varying fidelity.

Sequential hybrid meta-iteration supports two approaches: (1) the non-adaptive sequential hybrid runs one method to completion, passes its best results as the starting point for a subsequent method, and continues this succession until all methods have been executed (the stopping rules are controlled internally by each iterator), and
the adaptive sequential hybrid uses adaptive stopping rules for the iterators that are controlled externally by
this method. Any iterator may be used so long as it defines the notion of a final solution which can be passed as
starting data for subsequent iterators.

14.226.2 Member Function Documentation

```cpp
void print_results ( std::ostream & s, short results_state = FINAL_RESULTS ) [protected], [virtual]
print the final iterator results
This virtual function provides additional iterator-specific final results outputs beyond the function evaluation
summary printed in finalize_run().
Reimplemented from Iterator.
References Response::function_values(), Response::is_null(), Variables::is_null(), MetaIterator::iterSched, Iterator-
Scheduler::messagePass, and SeqHybridMetaIterator::prpResults.
void run_sequential ( ) [private]
run a sequential hybrid
In the sequential nonadaptive case, there is no interference with the iterators. Each runs until its own conver-
gence criteria is satisfied. Status: fully operational.
References Iterator::accepts_multiple_points(), ParallelLibrary::bcast(), ParallelLibrary::bcast_hs(), Iterator-
::initialize_graphics(), Model::interface_id(), Iterator::iteratedModel, IteratorScheduler::iterator_message_lengths(),
IteratorScheduler::iteratorCommRank, IteratorScheduler::iteratorCommSize, IteratorScheduler::iteratorScheduling,
IteratorScheduler::iteratorServerId, MetaIterator::iterSched, Iterator::methodP-CIter, SeqHybridMetaIterator::methodStrings, IteratorScheduler::miPLIndex, Iterator::num_final_solutions(), Iterator-
Scheduler::numIteratorJobs, IteratorScheduler::numIteratorServers, SeqHybridMetaIterator::pack_parameters_buffer(),
Iterator::parallelLib, SeqHybridMetaIterator::parameterSets, SeqHybridMetaIterator::prpResults, ParallelLibrary-
::recv(), Iterator::response_results(), IteratorScheduler::schedule_iterators(), SeqHybridMetaIterator::selectedIterators,
SeqHybridMetaIterator::selectedModels, ParallelLibrary::send(), SeqHybridMetaIterator::seqCount, SeqHybridMeta-
Iterator::singlePassedModel, MPIPackBuffer::size(), Iterator::summaryOutputFlag, and Iterator::variables_results().
Referenced by SeqHybridMetaIterator::core_run().
void run_sequential_adaptive ( ) [private]
run a sequential adaptive hybrid
In the sequential adaptive case, there is interference with the iterators through the use of the ++ overloaded op-
erator. iterator++ runs the iterator for one cycle, after which a progress_metric is computed. This progress metric
is used to dictate method switching instead of each iterator’s internal convergence criteria. Status: incomplete.
References Iterator::finalize_run(), Iterator::initialize_graphics(), Iterator::initialize_run(), IteratorScheduler-
::iteratorCommRank, IteratorScheduler::iteratorServerId, MetaIterator::iterSched, SeqHybridMetaIterator::method-
Strings, IteratorScheduler::numIteratorServers, SeqHybridMetaIterator::progressThreshold, Iterator::response-
results(), SeqHybridMetaIterator::selectedIterators, SeqHybridMetaIterator::selectedModels, SeqHybridMetaIterator-
::seqCount, Iterator::summaryOutputFlag, and Iterator::variables_results().
Referenced by SeqHybridMetaIterator::core_run().
void extract_parameter_sets ( int job_index, VariablesArray & partial_param_sets ) [inline], [private]
extract partial_param_sets from parameterSets based on job_index
This convenience function is executed on an iterator master (static scheduling) or a meta-iterator master (self scheduling) at run initialization time and has access to the full parameterSets array (this is All-Reduced for all peers at the completion of each cycle in run_sequential()).
References SeqHybridMetaIterator::parameterSets, and SeqHybridMetaIterator::partition_sets().
Referenced by SeqHybridMetaIterator::initialize_iterator(), and SeqHybridMetaIterator::pack_parameters_buffer().
The documentation for this class was generated from the following files:

- SeqHybridMetaIterator.hpp
- SeqHybridMetaIterator.cpp

14.227 SerialDirectApplicInterface Class Reference

Sample derived interface class for testing serial simulator plug-ins using assign_rep().
Inheritance diagram for SerialDirectApplicInterface:

```
    Interface
     |    |
     |    |
     |    v
ApplicationInterface
     |    |
     |    |
     |    v
DirectApplicInterface
     |    |
     |    |
     |    v
SerialDirectApplicInterface
```

Public Member Functions

- **SerialDirectApplicInterface** (const Dakota::ProblemDescDB &problem_db)
  
  constructor

- **~SerialDirectApplicInterface** ()
  
  destructor

Protected Member Functions

- int **derived_map_ac** (const Dakota::String &ac_name)
  
  execute an analysis code portion of a direct evaluation invocation

- void **derived_map_asynch** (const Dakota::ParamResponsePair &pair)
  
  no-op hides base error; job batching occurs within wait_local_evaluations()

- void **wait_local_evaluations** (Dakota::PRPQueue &prp_queue)
  
  evaluate the batch of jobs contained in prp_queue

- void **test_local_evaluations** (Dakota::PRPQueue &prp_queue)
  
  invokes wait_local_evaluations() (no special nowait support)

- void **set_communicators_checks** (int max_eval_concurrency)
  
  no-op hides default run-time error checks at DirectApplicInterface level
Private Member Functions

- int rosenbrock (const Dakota::RealVector &c_vars, short asv, Dakota::Real &fn_val, Dakota::RealVector &fn_grad, Dakota::RealSymMatrix &fn_hess)

  Rosenbrock plug-in test function.

Additional Inherited Members

14.227.1 Detailed Description

Sample derived interface class for testing serial simulator plug-ins using assign_rep().

The plug-in SerialDirectApplicInterface resides in namespace SIM and uses a copy of rosenbrock() to perform serial parameter to response mappings. It is used to demonstrate plugging in a serial direct analysis driver into Dakota in library mode. Test input files can then use an analysis_driver of "plugin_rosenbrock".

14.227.2 Member Function Documentation

void test_local_evaluations ( Dakota::PRPQueue & prp_queue ) [inline], [protected]

invokes wait_local_evaluations() (no special nowait support)

For use by ApplicationInterface::serve_evaluations_asynch(), which can provide a batch processing capability within message passing schedulers (called using chain IteratorScheduler::run_iterator() – Model::serve() – ApplicationInterface::serve_evaluations() – ApplicationInterface::serve_evaluations_asynch()).

References SerialDirectApplicInterface::wait_local_evaluations().

The documentation for this class was generated from the following files:

- PluginSerialDirectApplicInterface.hpp
- PluginSerialDirectApplicInterface.cpp

14.228 TrackerHTTP::Server Struct Reference

struct to hold tracker/proxy pairs

Public Member Functions

- Server (std::string t, std::string p)

Public Attributes

- std::string tracker
- std::string proxy

14.228.1 Detailed Description

struct to hold tracker/proxy pairs

The documentation for this struct was generated from the following file:

- TrackerHTTP.hpp
14.229 SharedApproxData Class Reference

Base class for the shared approximation data class hierarchy.

Inheritance diagram for SharedApproxData:

```
SharedApproxData
  SharedC3ApproxData
  SharedPecosApproxData
  SharedSurfpackApproxData
```

Public Member Functions

- **SharedApproxData ()**
  *default constructor*

- **SharedApproxData (ProblemDescDB &problem_db, size_t num_vars)**
  *standard constructor for envelope*

- **SharedApproxData (const String &approx_type, const UShortArray &approx_order, size_t num_vars, short data_order, short output_level)**
  *alternate constructor for envelope*

- **SharedApproxData (const SharedApproxData &approx)**
  *copy constructor*

- virtual **~SharedApproxData ()**
  *destructor*

- **SharedApproxData operator= (const SharedApproxData &approx)**
  *assignment operator*

- virtual void **active_model_key (const UShortArray &key)**
  *activate an approximation state based on its multi-index key*

- virtual void **clear_model_keys ()**
  *reset initial state by clearing all model keys for an approximation*

- virtual void **integration_iterator (const Iterator &iterator)**
  *set integration driver for structured grid approximations*

- virtual short **discrepancy_type () const**
  *return the discrepancy type for approximations that support MLMF*

- virtual void **build ()**
  *builds the shared approximation data from scratch*

- virtual void **rebuild ()**
  *rebuilds the shared approximation data incrementally*

- virtual void **pop (bool save_surr_data)**
  *back out the previous increment to the shared approximation data*

- virtual bool **push_available ()**
  *queries availability of pushing data associated with a trial set*

- virtual size_t **push_index (const UShortArray &key)**
  *return index for restoring trial set within stored data sets*
virtual void pre_push ()
push a previous state of the shared approximation data
virtual void post_push ()
clean up popped bookkeeping following push
virtual size_t finalize_index (size_t i, const UShortArray &key)
return index of i-th trial set within restorable bookkeeping sets
virtual void pre_finalize ()
finalize the shared approximation data following a set of increments
virtual void post_finalize ()
clean up popped bookkeeping following aggregation
virtual void clear_inactive ()
clear inactive approximation data
virtual void pre_combine ()
aggregate the shared approximation data from current and stored states
virtual void post_combine ()
clean up stored data sets after aggregation
virtual void combined_to_active (bool clear_combined=true)
promote aggregated data sets to active state
virtual void increment_order ()
increments polynomial expansion order (PCE, FT)
virtual void decrement_order ()
decrements polynomial expansion order (PCE, FT)
virtual void construct_basis (const Pecos::MultivariateDistribution &mv_dist)
construct the shared basis for an expansion-based approximation
virtual void update_basis_distribution_parameters (const Pecos::MultivariateDistribution &mvd)
propagate updates to random variable distribution parameters to a polynomial basis
virtual void configuration_options (const Pecos::ExpansionConfigOptions &ec_options)
set ExpansionConfigOptions instance as a group specification
virtual void configuration_options (const Pecos::BasisConfigOptions &bc_options)
set BasisConfigOptions instance as a group specification
virtual void configuration_options (const Pecos::RegressionConfigOptions &rc_options)
set BasisConfigOptions instance as a group specification
virtual void random_variables_key (const BitArray &random_vars_key)
assign key identifying a subset of variables that are to be treated as random for statistical purposes (e.g. expectation)
virtual void refinement_statistics_type (short stats_type)
assign statistics mode: {ACTIVE, COMBINED}, EXPANSION_STATS
virtual const Pecos::BitArrayULongMap & sobol_index_map () const
return set of Sobol indices that have been requested (e.g., as constrained by throttling) and are computable by a (sparse) expansion of limited order
virtual const UShortArray & active_model_key () const
return active multi-index key
• bool formulation_updated () const
  query whether the form of an approximation has been updated

• void set_bounds (const RealVector &c_l_bnds, const RealVector &c_u_bnds, const IntVector &di_l_bnds,
  const IntVector &di_u_bnds, const RealVector &dr_l_bnds, const RealVector &dr_u_bnds)
  set approximation lower and upper bounds (currently only used by graphics)

• SharedApproxData * data_rep () const
  returns dataRep for access to derived class member functions that are not mapped to the top SharedApproxData
  level

Protected Member Functions

• SharedApproxData (BaseConstructor, ProblemDescDB &problem_db, size_t num_vars)
  constructor initializes the base class part of letter classes (BaseConstructor overloading avoids infinite recursion
  in the derived class constructors - Coplien, p. 139)

• SharedApproxData (NoDBBaseConstructor, const String &approx_type, size_t num_vars, short data_order,
  short output_level)
  constructor initializes the base class part of letter classes (BaseConstructor overloading avoids infinite recursion
  in the derived class constructors - Coplien, p. 139)

Protected Attributes

• size_t numVars
  number of variables in the approximation

• String approxType
  approximation type identifier

• short buildDataOrder
  order of the data used for surrogate construction, in ActiveSet request vector 3-bit format.

• short outputLevel
  output verbosity level: {SILENT,QUIET,NORMAL,VERBOSE,DEBUG}.OUTPUT

• UShortArray activeKey
  multi-index key indicating the active model or model-pair used for approximation data

• UShort2DArray approxDataKeys
  set of multi-index model keys to enumerate when updating the SurrogateData for each Approximation

• String modelExportPrefix
  Prefix for model export files.

• unsigned short modelExportFormat
  Bitmapped format request for exported models.

• RealVector approxCLowerBnds
  approximation continuous lower bounds (used by 3D graphics and Surfpack KrigingModel)

• RealVector approxCUpperBnds
  approximation continuous upper bounds (used by 3D graphics and Surfpack KrigingModel)

• IntVector approxDILowerBnds
  approximation continuous lower bounds

• IntVector approxDIUpperBnds
  approximation continuous upper bounds
• RealVector approxDRLowerBnds
  approximation continuous lower bounds
• RealVector approxDRUpperBnds
  approximation continuous upper bounds
• std::map< UShortArray, bool > formUpdated
  tracker for changes in order,rank configuration since last build (used by DataFitSurrModel::rebuild_approximation())

Private Member Functions

• SharedApproxData * get_shared_data (ProblemDescDB &problem_db, size_t num_vars)
  Used only by the standard envelope constructor to initialize dataRep to the appropriate derived type.
• SharedApproxData * get_shared_data (const String &approx_type, const UShortArray &approx_order, size_t num_vars, short data_order, short output_level)
  Used only by the alternate envelope constructor to initialize dataRep to the appropriate derived type.

Private Attributes

• SharedApproxData * dataRep
  pointer to the letter (initialized only for the envelope)
• int referenceCount
  number of objects sharing dataRep

Friends

• class Approximation
• class TaylorApproximation
• class TANA3Approximation
• class QMEApproximation
• class GaussProcApproximation
• class VPSApproximation
• class PecosApproximation
• class C3Approximation
• class SurfpackApproximation
• class SurrogatesGPApprox
• class SurrogatesPolyApprox

14.229.1 Detailed Description

Base class for the shared approximation data class hierarchy.

The SharedApproxData class is the base class for the shared approximation data class hierarchy in DAKOTA. For memory efficiency and enhanced polymorphism, the approximation hierarchy employs the "letter/envelope idiom" (see Coplien "Advanced C++", p. 133), for which the base class (SharedApproxData) serves as the envelope and one of the derived classes (selected in SharedApproxData::get_shared_data()) serves as the letter.
14.229.2 Constructor & Destructor Documentation

SharedApproxData ( )
default constructor

For the default constructor, dataRep is NULL. This makes it necessary to check for NULL in the copy constructor, assignment operator, and destructor.

Referenced by SharedApproxData::get_shared_data().

SharedApproxData ( ProblemDescDB & problem_db, size_t num_vars )

standard constructor for envelope

Envelope constructor only needs to extract enough data to properly execute get_shared_data, since SharedApproxData(BaseConstructor, problem_db) builds the actual base class data for the derived approximations.

References Dakota::abort_handler(), SharedApproxData::dataRep, and SharedApproxData::get_shared_data().

SharedApproxData ( const String & approx_type, const UShortArray & approx_order, size_t num_vars, short data_order, short output_level )

alternate constructor for envelope

This is the alternate envelope constructor for instantiations on the fly. Since it does not have access to problem_db, it utilizes the NoDBBaseConstructor constructor chain.

References Dakota::abort_handler(), SharedApproxData::dataRep, and SharedApproxData::get_shared_data().

SharedApproxData ( const SharedApproxData & shared_data )
copy constructor

Copy constructor manages sharing of dataRep and incrementing of referenceCount.

References SharedApproxData::dataRep, and SharedApproxData::referenceCount.

~SharedApproxData ( ) [virtual]
destructor

Destructor decrements referenceCount and only deletes dataRep when referenceCount reaches zero.

References SharedApproxData::dataRep, and SharedApproxData::referenceCount.

SharedApproxData ( BaseConstructor, ProblemDescDB & problem_db, size_t num_vars ) [protected]

constructor initializes the base class part of letter classes (BaseConstructor overloading avoids infinite recursion in the derived class constructors - Coplien, p. 139)

This constructor is the one which must build the base class data for all derived classes. get_shared_data() instantiates a derived class letter and the derived constructor selects this base class constructor in its initialization list (to avoid recursion in the base class constructor calling get_shared_data() again). Since the letter IS the representation, its rep pointer is set to NULL (an uninitialized pointer causes problems in ~SharedApproxData).

References SharedApproxData::approxDataKeys, SharedApproxData::approxType, SharedApproxData::buildDataOrder, ProblemDescDB::get_bool(), ProblemDescDB::get_db_model_node(), ProblemDescDB::get_string(), ProblemDescDB::set_db_model_nodes(), Dakota::strbegins(), and Dakota::strends().
SharedApproxData ( NoDBBaseConstructor, const String & approx_type, size_t num_vars, short data_order, short output_level ) [protected]

constructor initializes the base class part of letter classes (BaseConstructor overloading avoids infinite recursion in the derived class constructors - Coplien, p. 139)

This constructor is the one which must build the base class data for all derived classes. get_shared_data() instantiates a derived class letter and the derived constructor selects this base class constructor in its initialization list (to avoid recursion in the base class constructor calling get_shared_data() again). Since the letter IS the representation, its rep pointer is set to NULL (an uninitialized pointer causes problems in ~SharedApproxData).

References SharedApproxData::approxDataKeys, SharedApproxData::approxType, SharedApproxData::buildDataOrder, Dakota::strbegins(), and Dakota::strends().

14.229.3 Member Function Documentation

SharedApproxData operator= ( const SharedApproxData & shared_data )

assignment operator


References SharedApproxData::dataRep, and SharedApproxData::referenceCount.

SharedApproxData * get_shared_data ( ProblemDescDB & problem_db, size_t num_vars ) [private]

Used only by the standard envelope constructor to initialize dataRep to the appropriate derived type.

Used only by the envelope constructor to initialize dataRep to the appropriate derived type.

References ProblemDescDB::get_string(), SharedApproxData::SharedApproxData(), and Dakota::strends().

Referenced by SharedApproxData::SharedApproxData().

SharedApproxData * get_shared_data ( const String & approx_type, const UShortArray & approx_order, size_t num_vars, short data_order, short output_level ) [private]

Used only by the alternate envelope constructor to initialize dataRep to the appropriate derived type.

Used only by the envelope constructor to initialize dataRep to the appropriate derived type.

References SharedApproxData::SharedApproxData(), and Dakota::strends().

14.229.4 Member Data Documentation

short buildDataOrder [protected]

order of the data used for surrogate construction, in ActiveSet request vector 3-bit format.

This setting distinguishes derivative data intended for use in construction (includes derivatives w.r.t. the build variables) from derivative data that may be approximated separately (excludes derivatives w.r.t. auxiliary variables). This setting should also not be inferred directly from the responses specification, since we may need gradient support for evaluating gradients at a single point (e.g., the center of a trust region), but not require gradient evaluations at every point.

Referenced by SharedSurfpackApproxData::add_sd_to_surfdata(), TaylorApproximation::build(), TaylorApproximation::gradient(), TaylorApproximation::hessian(), TaylorApproximation::min_coefficients(), Approximation::min_points(), QMEApproximation::QMEApproximation(), Approximation::recommended_points(), SharedApproxData::SharedApproxData(), SharedPecosApproxData::SharedPecosApproxData(), SurfpackApproximation::SurfpackApproximation(), SurfpackApproximation::surrogates_to_surfdata(), TANA3Approximation::TANA3Approximation(), and TaylorApproximation::value().
14.230. SHARED_C3_APPROXDATA CLASS REFERENCE

The documentation for this class was generated from the following files:

- SharedApproxData.hpp
- SharedApproxData.cpp

14.230 SharedC3ApproxData Class Reference

Derived approximation class for global basis polynomials.

Inheritance diagram for SharedC3ApproxData:

```
<table>
<thead>
<tr>
<th>SharedC3ApproxData</th>
</tr>
</thead>
<tbody>
<tr>
<td>SharedApproxData</td>
</tr>
</tbody>
</table>
```

Public Member Functions

- **SharedC3ApproxData ()**
  default constructor
- **SharedC3ApproxData (ProblemDescDB &problem_db, size_t num_vars)**
  standard ProblemDescDB-driven constructor
- **SharedC3ApproxData (const String &approx_type, const UShortArray &approx_order, size_t num_vars, short data_order, short output_level)**
  on-the-fly constructor (no problem DB)
- **~SharedC3ApproxData ()**
  destructor
- **size_t regression_size ()**
  return number of FT unknowns using numVars, start_rank(), start_orders()
- **void set_parameter (String var, const UShortArray &val)**
  set UShortArray attribute value based on identifier string
- **void set_parameter (String var, unsigned short val)**
  set unsigned short attribute value based on identifier string
- **void set_parameter (String var, size_t val)**
  set size_t attribute value based on identifier string
- **void set_parameter (String var, bool val)**
  set bool attribute value based on identifier string
- **void set_parameter (String var, short val)**
  set short attribute value based on identifier string
- **void set_parameter (String var, double val)**
  set double attribute value based on identifier string
- **void set_parameter (String var, int val)**
  set int attribute value based on identifier string
- **const UShortArray & start_orders ()**
  const
return current basis polynomial order (active key in startOrders)

- `size_t start_rank()` const
  return current expansion rank (active key in startRank)

- `void update_basis()`
  update oneApproxOpts with active basis orders after an order change

- `void update_basis(const UShortArray &start_orders, unsigned short max_order)`
  update oneApproxOpts with passed basis orders after an order change

Static Public Member Functions

- `static size_t regression_size(size_t num_v, size_t rank, const UShortArray &orders)`
  return number of FT unknowns given scalars: num vars, rank, order

Protected Member Functions

- `void active_model_key(const UShortArray &key)`
  activate an approximation state based on its multi-index key

- `void construct_basis(const Pecos::MultivariateDistribution &mv_dist)`
  construct the shared basis for an expansion-based approximation

- `short discrepancy_type() const`
  return the discrepancy type for approximations that support MLMF

- `void random_variables_key(const BitArray &random_vars_key)`
  assign key identifying a subset of variables that are to be treated as random for statistical purposes (e.g. expectation)

- `void build()`
  builds the shared approximation data from scratch

- `void increment_order()`
  increments polynomial expansion order (PCE, FT)

- `void decrement_order()`
  decrements polynomial expansion order (PCE, FT)

- `void pop(bool save_surr_data)`
  back out the previous increment to the shared approximation data

- `bool push_available()`
  queries availability of pushing data associated with a trial set

- `size_t push_index(const UShortArray &key)`
  return index for restoring trial set within stored data sets

- `void post_push()`
  clean up popped bookkeeping following push

- `void pre_combine()`
  aggregate the shared approximation data from current and stored states
Protected Attributes

- std::vector< OneApproxOpts * > oneApproxOpts
  1D approximation options (basis type, poly order, etc.)
- MultiApproxOpts * multiApproxOpts
  n-D approximation options, augmenting 1D options
- UShortArray startOrderSpec
  starting user specification for polynomial orders (from start_order scalar plus anisotropic dimension preference)
- std::map< UShortArray, UShortArray > startOrders
  starting values for polynomial order (prior to adaptive refinement); for each model key, there is an array of polynomial orders per variable
- UShortArray combinedOrders
  polynomial basis order for combined expansion for each variable core
- unsigned short maxOrder
  maximum value for polynomial order (if adapted)
- size_t startRankSpec
  starting user specification for rank (not augmented by dimension preference); Note: rank sequence spec is managed externally and becomes reflected in startRank model index mapping
- std::map< UShortArray, size_t > startRank
  starting values for rank (note: adapt_rank currently covers refinement); for each model index key, there is a scalar starting rank (recovered rank in C3FnrainPtrs can vary per core/variable and per QoI)
- size_t kickRank
  user specification for increment in rank used within adapt_rank
- size_t maxRank
  user specification for maximum rank used within adapt_rank
- bool adaptRank
  internal C3 adaptation that identifies the best rank representation for a set of sample data based on cross validation
- short regressType
  type of regression solver for forming FT approximation
- double regressRegParam
  penalty parameter if regularized regression
- double solverTol
  tolerance on regression solver
- double roundingTol
  tolerance when performing a truncation operation on FT expansion
- double arithmeticTol
  a separate rounding tolerance for c3axpy() in combine_coefficients()
- int maxSolverIterations
  maximum number of iterations for regression solver
- int crossMaxIter
  maximum number of iterations for (future) cross iteration solver
- int c3Verbosity
  verbosity setting for C3 TPL, mapped from Dakota verbosity
CHAPTER 14. CLASS DOCUMENTATION

- bool adaptConstruct
  a more general adaptive construction option, distinct from adapt_rank
- bool crossVal
  C3 FT can support CV over polynomial order in addition to adapt_rank.
- short combineType
  type of discrepancy calculation: additive, multiplicative, or both
- short discrepancyType
  type of multilevel discrepancy emulation: distinct or recursive
- short allocControl
  type of multilevel strategy for sample allocation: ESTIMATOR, VARIANCE, RANK_SAMPLING, GREEDY
- SizetArray randomIndices
  indices for random subset when approximating in all-variables mode
- std::map< UShortArray, size_t > poppedCounts
  number of instances within the popped arrays (mostly a placeholder for supporting push_available())

Friends
- class C3Approximation

14.230.1 Detailed Description
Derived approximation class for global basis polynomials.

The SharedC3ApproxData class provides a global approximation based on basis polynomials. This includes orthogonal polynomials used for polynomial chaos expansions and interpolation polynomials used for stochastic collocation.

14.230.2 Member Function Documentation

size_t regression_size ( size_t num_v, size_t rank, const UShortArray & orders ) [inline], [static]
return number of FT unknowns given scalars: num vars, rank, order
simplified estimation for scalar-valued rank and order (e.g., from start rank/order user specification)

inline size_t SharedC3ApproxData:: regression_size(size_t num_v, size_t rank, size_t order) { Each dimension has its own rank within the product of function cores. This fn estimates for the case where rank and order are either constant across dimensions or averaged into a scalar.

the first and last core contribute p*r terms the middle cores contribute r+r+p terms
size_t p = order+1.; switch (num_v) { case 1: return p; break; // collapses to a 1D PCE case 2: return 2.*p*rank; break; // first and last core, no middle default: return p*rank+(2. + (num_v-2)*rank); break; // first,last,middle }
simplified estimation for scalar-valued rank and vector-valued order (e.g., from start rank/start order/dimension pref user specification)

The documentation for this class was generated from the following files:

- SharedC3ApproxData.hpp
- SharedC3ApproxData.cpp
14.231 SharedPecosApproxData Class Reference

Derived approximation class for global basis polynomials.

Inheritance diagram for SharedPecosApproxData:

```
SharedApproxData

SharedPecosApproxData
```

Public Member Functions

- `SharedPecosApproxData()`
  - default constructor
- `SharedPecosApproxData(const String &approx_type, const UShortArray &approx_order, size_t num_vars, short data_order, short output_level)`
  - alternate constructor
- `SharedPecosApproxData(const ProblemDescDB &problem_db, size_t num_vars)`
  - standard ProblemDescDB-driven constructor
- `~SharedPecosApproxData()`
  - destructor
- `void random_variables_key(const Pecos::BitArray &random_vars_key)`
  - set pecosBasisApprox.randomVarsKey
- `void update_basis_distribution_parameters(const Pecos::MultivariateDistribution &u_dist)`
  - invoke Pecos::SharedPolyApproxData::update_basis_distribution_parameters()
- `void polynomial_basis(const std::vector<Pecos::BasisPolynomial> &poly_basis)`
  - set Pecos::SharedOrthogPolyApproxData::polynomialBasis
- `std::vector<Pecos::BasisPolynomial> & polynomial_basis()` const
  - get Pecos::SharedOrthogPolyApproxData::polynomialBasis
- `const std::vector<Pecos::BasisPolynomial> & polynomial_basis()` const
  - get Pecos::SharedOrthogPolyApproxData::polynomialBasis
- `void allocate(const UShort2DArray &mi)`
  - set Pecos::SharedOrthogPolyApproxData::multiIndex and allocate associated arrays
- `const UShort2DArray & multi_index()` const
  - get active Pecos::SharedOrthogPolyApproxData::multiIndex
- `const std::map< UShortArray, UShort2DArray > & multi_index_map()` const
  - get Pecos::SharedOrthogPolyApproxData::multiIndex
- `const Pecos::BitArrayULongMap & sobol_index_map()` const
  - return Pecos::SharedPolyApproxData::sobolIndexMap
- `void coefficients_norms_flag(bool flag)`
  - invoke Pecos::SharedOrthogPolyApproxData::coefficients_norms_flag()
• size_t expansion_terms () const
  
  return Pecos::SharedOrthogPolyApproxData::expansion_terms()

• const UShortArray & expansion_order () const
  
  return Pecos::SharedOrthogPolyApproxData::expansion_order()

• void expansion_order (const UShortArray &order)
  
  invokes Pecos::SharedOrthogPolyApproxData::expansion_order(UShortArray&)

• void configuration_options (const Pecos::ExpansionConfigOptions &ec_options)
  
  set the expansion configuration options within Pecos::SharedPolyApproxData

• void configuration_options (const Pecos::BasisConfigOptions &bc_options)
  
  set the basis configuration options within Pecos::SharedPolyApproxData

• void configuration_options (const Pecos::RegressionConfigOptions &rc_options)
  
  set the regression configuration options within Pecos::SharedRegressOrthogPolyApproxData

• void refinement_statistics_type (short stats_type)
  
  update ExpansionConfigOptions::refineStatsType

Protected Member Functions

• void active_model_key (const UShortArray &key)
  
  activate an approximation state based on its multi-index key

• void clear_model_keys ()
  
  reset initial state by clearing all model keys for an approximation

• void construct_basis (const Pecos::MultivariateDistribution &mv_dist)
  
  construct the shared basis for an expansion-based approximation

• void integration_iterator (const Iterator &iterator)
  
  set integration driver for structured grid approximations

• short discrepancy_type () const
  
  return the discrepancy type for approximations that support MLMF

• void build ()
  
  builds the shared approximation data from scratch

• void rebuild ()
  
  rebuilds the shared approximation data incrementally

• void pop (bool save_surr_data)
  
  back out the previous increment to the shared approximation data

• bool push_available ()
  
  queries availability of pushing data associated with a trial set

• size_t push_index (const UShortArray &key)

• void pre_push ()
  
  push a previous state of the shared approximation data

• void post_push ()
  
  clean up popped bookkeeping following push

• size_t finalize_index (size_t i, const UShortArray &key)
  
  return index of i-th trial set within restorable bookkeeping sets

• void pre_finalize ()
finalize the shared approximation data following a set of increments

- void post_finalize ()
  clean up popped bookkeeping following aggregation

- void pre_combine ()
  aggregate the shared approximation data from current and stored states

- void post_combine ()
  clean up stored data sets after aggregation

- void combined_to_active (bool clear_combined=true)
  promote aggregated data sets to active state

- void clear_inactive ()
  clear inactive approximation data

- void increment_order ()
  increments polynomial expansion order (PCE, FT)

- void decrement_order ()
  decrements polynomial expansion order (PCE, FT)

Private Member Functions

- Pecos::SharedBasisApproxData & pecos_shared_data ()
  return pecosSharedData

- Pecos::SharedPolyApproxData * pecos_shared_data_rep ()
  return pecosSharedDataRep

- void approx_type_to_basis_type (const String &approx_type, short &basis_type)
  utility to convert Dakota type string to Pecos type enumeration

Private Attributes

- Pecos::SharedBasisApproxData pecosSharedData
  the Pecos shared approximation data

- Pecos::SharedPolyApproxData * pecosSharedDataRep
  convenience pointer to derived letter within pecosSharedData

Friends

- class PecosApproximation

Additional Inherited Members

14.231.1 Detailed Description

Derived approximation class for global basis polynomials.

The SharedPecosApproxData class provides a global approximation based on basis polynomials. This includes orthogonal polynomials used for polynomial chaos expansions and interpolation polynomials used for stochastic collocation.
14.231.2  Member Function Documentation

size_t push_index( const UShortArray & key )  [inline], [protected], [virtual]

In Pecos, SharedPolyApproxData::push_index() is for internal use as it can be either a flattened (ISGDriver) or level-specific index (HSGDriver), as convenient for combined or hierarchical data restoration. SharedPolyApproxData::{restore,finalize} index() are intended for external use and will map from push_index() to provide a consistent (flattened) representation. Dakota, however, does not make this distinction and uses {push,finalize} -index() semantics for consistency with {push,finalize}_data().

Reimplemented from SharedApproxData.

References SharedPecosApproxData::pecosSharedDataRep.
The documentation for this class was generated from the following files:

- SharedPecosApproxData.hpp
- SharedPecosApproxData.cpp

14.232  SharedResponseData Class Reference

Container class encapsulating variables data that can be shared among a set of Response instances.

Public Member Functions

- SharedResponseData ()
  default constructor
- SharedResponseData (const ProblemDescDB &problem_db)
  standard constructor
- SharedResponseData (const ActiveSet &set)
  alternate on-the-fly constructor (explicit disallows its use for implicit type conversion so that instantiations of Response(set) are invalid)
- SharedResponseData (const SharedResponseData &srd)
  copy constructor
- ~SharedResponseData ()
  destructor
- SharedResponseData & operator= (const SharedResponseData &srd)
  assignment operator
- bool operator==(const SharedResponseData &other)
  experimental operator== for use in unit testing
- size_t num_scalar_responses () const
  number of scalar responses: primary_scalar + nonlinear constraints; note that these are non-contiguous when primary fields are present.
- size_t num_scalar_primary () const
  number of scalar primary responses (objectives, calibration terms, generic)
- size_t num_field_response_groups () const
  number of primary fields (primary field groups)
- size_t num_response_groups () const
  total number of response groups (number scalars + number pri field groups)
- size_t num_field_functions () const
total number of primary field functions (elements); 1-norm of priFieldLengths

- `size_t num_functions()` const
  total number of response functions (scalars + 1-norm of priFieldLengths)

- `const IntVector & field_lengths()` const
  length of each primary field

- `void field_lengths(const IntVector &field_lens)`
  set field lengths (e.g., if experiment different from simulation)

- `const IntVector & num_coords_per_field()` const
  number of independent coordinates for each primary field

- `const String & function_label(size_t i)` const
  return a response function identifier string

- `const StringArray & function_labels()` const
  return the response function identifier strings

- `StringArray & function_labels()`
  return the response function identifier strings

- `void function_label(const String &label, size_t i)`
  set a response function identifier string

- `void function_labels(const StringArray &labels)`
  set the response function identifier strings

- `const StringArray & field_group_labels()`
  return the coarse (per-group) primary field response labels

- `void field_group_labels(const StringArray &field_labels)`
  set the coarse primary field group labels (must agree with number fields)

- `const String & responses_id()` const
  return the response identifier

- `short response_type()` const
  return the response type: {BASE,SIMULATION,EXPERIMENT} RESPONSE

- `void response_type(short type)`
  set the response type: {BASE,SIMULATION,EXPERIMENT} RESPONSE

- `short primary_fn_type()` const
  get the primary function type (generic, objective, calibration)

- `void primary_fn_type(short type)`
  set the primary function type (generic, objective, calibration)

- `const RealVector & simulation_error()` const
  retrieve simulation variance

- `SharedResponseData copy()` const
  create a deep copy of the current object and return by value

- `void reshape(size_t num_fns)`
  reshape the data, disconnecting a shared rep if necessary

- `void reshape_labels(StringArray &resp_labels, size_t num_fns)`
  reshape the response labels using inflation/deflation if possible

- `bool is_null()` const
  return true if empty handle with null representation
- long reference_count () const
  how many handles (including this) are sharing this representation (body); for debugging/testing only
- template<class Archive >
  void serialize (Archive &ar, const unsigned int version)

Private Member Functions

- template<class Archive >
  void serialize (Archive &ar, const unsigned int version)
  serialize through the pointer, which requires object tracking; write and read are symmetric for this class

Private Attributes

- boost::shared_ptr
  < SharedResponseDataRep > srdRep
  pointer to the body (handle-body idiom)

Friends

- class boost::serialization::access
  allow boost access to serialize this class

14.232.1 Detailed Description

Container class encapsulating variables data that can be shared among a set of Response instances.

An array of Response objects (e.g., Analyzer::allResponse) contains repeated configuration data (id’s, labels, counts). SharedResponseData employs a handle-body idiom to allow this shared data to be managed in a single object with many references to it, one per Response object in the array. This allows scaling to larger sample sets.

14.232.2 Member Function Documentation

SharedResponseData copy ( ) const
create a deep copy of the current object and return by value
Deep copies are used when recasting changes the nature of a Response set.
References SharedResponseData::srdRep.
Referenced by ExperimentData::initialize(), and ExperimentData::load_data().
The documentation for this class was generated from the following files:

- SharedResponseData.hpp
- SharedResponseData.cpp

14.233 SharedResponseDataRep Class Reference

The representation of a SharedResponseData instance. This representation, or body, may be shared by multiple SharedResponseData handle instances.
Public Member Functions

- \(\sim\)SharedResponseDataRep ()
  destructor must be public for shared_ptr
- \(\text{template}<\text{class Archive}>\)
  void serialize (Archive &ar, const unsigned int version)

Private Member Functions

- SharedResponseDataRep ()
  default constructor
- SharedResponseDataRep (const ProblemDescDB &problem_db)
  standard constructor
- SharedResponseDataRep (const ActiveSet &set)
  alternate on-the-fly constructor
- void copy_rep (SharedResponseDataRep *srd_rep)
  copy the data from srd_rep to the current representation
- \(\text{template}<\text{class Archive}>\)
  void serialize (Archive &ar, const unsigned int version)
  serialize the core shared response data: write and read are symmetric for this class
- bool operator== (const SharedResponseDataRep &other)
  experimental operator== for use in unit testing
- void build_field_labels ()
  build/update the unrolled field labels based on fieldLabels and group lengths

Private Attributes

- short responseType
  enumeration of BASE_RESPONSE, SIMULATION_RESPONSE, or EXPERIMENT_RESPONSE
- short primaryFnType
  data set type for primary response: generic, objective, calibration
- String responsesId
  response identifier string from the input file
- StringArray functionLabels
  fine-grained (unrolled) set of response function identifiers used to improve output readability; length Response-::functionValues
- StringArray priFieldLabels
  labels for each primary response field
- RealVector simulationVariance
  simulation variance
- size_t numScalarResponses = 0
  number of scalar responses
- size_t numScalarPrimary = 0
  number of scalar primary responses (secondary computed from difference)
- IntVector priFieldLengths
length of each primary response field

- IntVector coordsPerPriField
  number of independent coordinates, e.g., $x, t$, for each field $f(x,t)$

Friends

- class SharedResponseData
- class boost::serialization::access
  allow boost access to serialize this class

14.233.1 Detailed Description

The representation of a SharedResponseData instance. This representation, or body, may be shared by multiple SharedResponseData handle instances.

The SharedResponseData/SharedResponseDataRep pairs utilize a handle-body idiom (Coplien, Advanced C++).

14.233.2 Member Function Documentation

void copy_rep ( SharedResponseDataRep * srd_rep ) [private]

copy the data from srd_rep to the current representation

Deep copies are used when recasting changes the nature of a Response set.

References SharedResponseDataRep::coordsPerPriField, SharedResponseDataRep::functionLabels, SharedResponseDataRep::numScalarPrimary, SharedResponseDataRep::numScalarResponses, SharedResponseDataRep::priFieldLabels, SharedResponseDataRep::priFieldLengths, SharedResponseDataRep::primaryFnType, SharedResponseDataRep::responsesId, SharedResponseDataRep::responseType, and SharedResponseDataRep::simulation-Variance.

The documentation for this class was generated from the following files:

- SharedResponseData.hpp
- SharedResponseData.cpp

14.234 SharedSurfpackApproxData Class Reference

Derived approximation class for Surfpack approximation classes. Interface between Surfpack and Dakota.

Inheritance diagram for SharedSurfpackApproxData:
Public Member Functions

- `SharedSurfpackApproxData ()`
  default constructor

- `SharedSurfpackApproxData (const String &approx_type, const UShortArray &approx_order, size_t num_vars, short data_order, short output_level)`
  alternate constructor

- `SharedSurfpackApproxData (ProblemDescDB &problem_db, size_t num_vars)`
  standard constructor: Surfpack surface of appropriate type will be created

- `~SharedSurfpackApproxData ()`
  destructor

Private Member Functions

- `void add_sd_to_surfdata (const Pecos::SurrogateDataVars &sdv, const Pecos::SurrogateDataResp &sdr, short fail_code, SurfData &surf_data)`
  add Pecos::SurrogateData::SurrogateData{Vars,Resp} to SurfData, accounting for buildDataOrder available

- `void copy_matrix (const RealSymMatrix &rsm, SurfpackMatrix<Real> &surfpack_matrix)`
  copy RealSymMatrix to SurfpackMatrix (Real type only)

- `void merge_variable_arrays (const RealVector &cv, const IntVector &div, const RealVector &drv, RealArray &ra)`
  merge cv, div, and drv vectors into a single ra array

- `void sdv_to_realarray (const Pecos::SurrogateDataVars &sdv, RealArray &ra)`
  aggregate \{continuous,discrete int,discrete real\} variables from SurrogateDataVars into ra

- `void vars_to_realarray (const Variables &vars, RealArray &ra)`
  aggregate \{active,all\} \{continuous,discrete int,discrete real\} variables into ra

Private Attributes

- `unsigned short approxOrder`
  order of polynomial approximation

- `StringArray diagnosticSet`
  set of diagnostic metrics

- `bool crossValidateFlag`
  whether to perform cross validation

- `unsigned numFolds`
  number of folds for CV

- `Real percentFold`
  percentage of data for CV

- `bool pressFlag`
  whether to perform PRESS

Friends

- `class SurfpackApproximation`
- `class VPSApproximation`
Additional Inherited Members

14.234.1 Detailed Description

Derived approximation class for Surfpack approximation classes. Interface between Surfpack and Dakota.

The SharedSurfpackApproxData class is the interface between Dakota and Surfpack. Based on the information in the ProblemDescDB that is passed in through the constructor, SharedSurfpackApproxData builds a Surfpack Surface object that corresponds to one of the following data-fitting techniques: polynomial regression, kriging, artificial neural networks, radial basis function network, or multivariate adaptive regression splines (MARS).

14.234.2 Constructor & Destructor Documentation

SharedSurfpackApproxData ( const String &approx_type, const UShortArray &approx_order, size_t num_vars, short data_order, short output_level )

alternate constructor
- On-the-fly constructor which uses mostly Surfpack model defaults.
- References Dakota::abort_handler(), SharedSurfpackApproxData::approxOrder, and SharedApproxData::approxType.

SharedSurfpackApproxData ( ProblemDescDB &problem_db, size_t num_vars )

standard constructor: Surfpack surface of appropriate type will be created
- Initialize the embedded Surfpack surface object and configure it using the specifications from the input file.
- Data for the surface is created later.
- References SharedSurfpackApproxData::approxOrder, SharedApproxData::approxType, ProblemDescDB::get_short(), and ProblemDescDB::get_string().
- The documentation for this class was generated from the following files:
  - SharedSurfpackApproxData.hpp
  - SharedSurfpackApproxData.cpp

14.235 SharedVariablesData Class Reference

Container class encapsulating variables data that can be shared among a set of Variables instances.

Public Member Functions

- SharedVariablesData ()
  default constructor
- SharedVariablesData ( const ProblemDescDB &problem_db, const std::pair< short, short > &view )
  standard constructor
- SharedVariablesData ( const std::pair< short, short > &view, const std::map< unsigned short, size_t > &vars_comps, const BitArray &all_relax_di=BitArray(), const BitArray &all_relax_dr=BitArray() )
  medium weight constructor providing detailed variable counts
- SharedVariablesData ( const std::pair< short, short > &view, const SizetArray &vars_comps_totals, const BitArray &all_relax_di=BitArray(), const BitArray &all_relax_dr=BitArray() )
  lightweight constructor providing variable count totals
• **SharedVariablesData** (const SharedVariablesData &svd)
  copy constructor

• **~SharedVariablesData** ()
  destructor

• **SharedVariablesData & operator=** (const SharedVariablesData &svd)
  assignment operator

• **SharedVariablesData copy ()** const
  create a deep copy of the current object and return by value

• **void all_counts** (size_t &num_acv, size_t &num_adiv, size_t &num_adsv, size_t &num_adrv) const
  compute all variables sums from SharedVariablesDataRep::variablesCompsTotals and SharedVariablesDataRep::allRelaxedDiscrete{Int, Real}

• **void design_counts** (size_t &num_cdv, size_t &num_ddiv, size_t &num_ddsv, size_t &num_ddrv) const
  compute design variables sums from SharedVariablesDataRep::variablesCompsTotals and SharedVariablesDataRep::allRelaxedDiscrete{Int, Real}

• **void aleatory_uncertain_counts** (size_t &num_cauv, size_t &num_dauv, size_t &num_dausv, size_t &num_daurv) const
  compute aleatory uncertain variables sums from SharedVariablesDataRep::variablesCompsTotals and SharedVariablesDataRep::allRelaxedDiscrete{Int, Real}

• **void epistemic_uncertain_counts** (size_t &num_ceuv, size_t &num_deuv, size_t &num_deusv, size_t &num_deurv) const
  compute epistemic uncertain variables sums from SharedVariablesDataRep::variablesCompsTotals and SharedVariablesDataRep::allRelaxedDiscrete{Int, Real}

• **void uncertain_counts** (size_t &num_cuv, size_t &num_duv, size_t &num_dusv, size_t &num_durv) const
  compute uncertain variables sums from SharedVariablesDataRep::variablesCompsTotals and SharedVariablesDataRep::allRelaxedDiscrete{Int, Real}

• **void state_counts** (size_t &num_csv, size_t &num_dsv, size_t &num_dssv, size_t &num_dsrv) const
  compute state variables sums from SharedVariablesDataRep::variablesCompsTotals and SharedVariablesDataRep::allRelaxedDiscrete{Int, Real}

• **void active_subsets** (bool &cdv, bool &ddv, bool &cauv, bool &dauv, bool &ceuv, bool &deuv, bool &csv, bool &dsv) const
  define active variable subsets based on active view

• **void inactive_subsets** (bool &cdv, bool &ddv, bool &cauv, bool &dauv, bool &ceuv, bool &deuv, bool &csv, bool &dsv) const
  define active variable subsets based on active view

• **void complement_subsets** (bool &cdv, bool &ddv, bool &cauv, bool &dauv, bool &ceuv, bool &deuv, bool &csv, bool &dsv) const
  define active variable subsets based on active view

• **size_t cv_index_to_all_index** (size_t cv_index) const
  convert index within active continuous variables to index within aggregated variables (all continuous, discrete {int, string, real})

• **size_t icv_index_to_all_index** (size_t icv_index) const
  convert index within inactive continuous variables to index within aggregated variables (all continuous, discrete {int, string, real})

• **size_t ccv_index_to_all_index** (size_t ccv_index) const
  convert index within complement of active continuous variables to index within aggregated variables (all continuous, discrete {int, string, real})
• size_t acv_index_to_all_index (size_t acv_index) const
  convert index within all continuous variables to index within aggregated variables (all continuous, discrete {int,string,real})

• size_t div_index_to_all_index (size_t div_index) const
  convert index within active discrete integer variables to index within aggregated variables (all continuous, discrete {int,string,real})

• size_t idiv_index_to_all_index (size_t div_index) const
  convert index within inactive discrete integer variables to index within aggregated variables (all continuous, discrete {int,string,real})

• size_t cdv_index_to_all_index (size_t div_index) const
  convert index within complement of active discrete integer variables to index within aggregated variables (all continuous, discrete {int,string,real})

• size_t adiv_index_to_all_index (size_t adiv_index) const
  convert index within all discrete integer variables to index within aggregated variables (all continuous, discrete {int,string,real})

• size_t dsv_index_to_all_index (size_t dsv_index) const
  convert index within active discrete string variables to index within aggregated variables (all continuous, discrete {int,string,real})

• size_t idsv_index_to_all_index (size_t dsv_index) const
  convert index within inactive discrete string variables to index within aggregated variables (all continuous, discrete {int,string,real})

• size_t cdsv_index_to_all_index (size_t dsv_index) const
  convert index within complement of active discrete string variables to index within aggregated variables (all continuous, discrete {int,string,real})

• size_t adsv_index_to_all_index (size_t adsv_index) const
  convert index within all discrete string variables to index within aggregated variables (all continuous, discrete {int,string,real})

• size_t drv_index_to_all_index (size_t drv_index) const
  convert index within active discrete real variables to index within aggregated variables (all continuous, discrete {int,string,real})

• size_t idrv_index_to_all_index (size_t drv_index) const
  convert index within inactive discrete real variables to index within aggregated variables (all continuous, discrete {int,string,real})

• size_t cdrv_index_to_all_index (size_t drv_index) const
  convert index within complement of active discrete real variables to index within aggregated variables (all continuous, discrete {int,string,real})

• size_t adrv_index_to_all_index (size_t adrv_index) const
  convert index within all discrete real variables to index within aggregated variables (all continuous, discrete {int,string,real})

• size_t cv_index_to_active_index (size_t cv_index) const
  convert index within active continuous variables to index within aggregated active variables (active continuous, discrete {int,string,real})

• size_t div_index_to_active_index (size_t div_index) const
  convert index within active discrete integer variables to index within aggregated active variables (active continuous, discrete {int,string,real})

• size_t dsv_index_to_active_index (size_t dsv_index) const
  convert index within active discrete string variables to index within aggregated active variables (active continuous, discrete {int,string,real})
• `size_t drv_index_to_active_index` (size_t drv_index) const
  convert index within active discrete real variables to index within aggregated active variables (active continuous, discrete {int,string,real})

• `size_t ccv_index_to_acv_index` (size_t ccv_index) const
  convert index within complement of active continuous variables to index within all continuous variables

• `size_t cdiv_index_to_adiv_index` (size_t div_index) const
  convert index within complement of active discrete integer variables to index within all discrete integer variables

• `size_t cdsv_index_to_adsv_index` (size_t dsv_index) const
  convert index within complement of active discrete string variables to index within all discrete string variables

• `size_t cdrv_index_to_adrv_index` (size_t drv_index) const
  convert index within complement of active discrete real variables to index within all discrete real variables

• `BitArray cv_to_all_mask` () const
  create a BitArray indicating the active continuous subset of all {continuous, discrete {int,string,real}} variables

• `BitArray icv_to_all_mask` () const
  create a BitArray indicating the inactive continuous subset of all {continuous, discrete {int,string,real}} variables

• `BitArray ccv_to_all_mask` () const
  create a BitArray indicating the complement continuous subset of all {continuous, discrete {int,string,real}} variables

• `BitArray acv_to_all_mask` () const
  create a BitArray indicating the all continuous subset of all {continuous, discrete {int,string,real}} variables

• `BitArray div_to_all_mask` () const
  create a BitArray indicating the active discrete int subset of all {continuous, discrete {int,string,real}} variables

• `BitArray idiv_to_all_mask` () const
  create a BitArray indicating the inactive discrete int subset of all {continuous, discrete {int,string,real}} variables

• `BitArray cdiv_to_all_mask` () const
  create a BitArray indicating the complement discrete int subset of all {continuous, discrete {int,string,real}} variables

• `BitArray adiv_to_all_mask` () const
  create a BitArray indicating the all discrete int subset of all {continuous, discrete {int,string,real}} variables

• `BitArray dsv_to_all_mask` () const
  create a BitArray indicating the active discrete string subset of all {continuous, discrete {int,string,real}} variables

• `BitArray idsv_to_all_mask` () const
  create a BitArray indicating the inactive discrete string subset of all {continuous, discrete {int,string,real}} variables

• `BitArray cdsv_to_all_mask` () const
  create a BitArray indicating the complement discrete string subset of all {continuous, discrete {int,string,real}} variables

• `BitArray adsv_to_all_mask` () const
  create a BitArray indicating the all discrete string subset of all {continuous, discrete {int,string,real}} variables

• `BitArray drv_to_all_mask` () const
  create a BitArray indicating the active discrete real subset of all {continuous, discrete {int,string,real}} variables

• `BitArray idrv_to_all_mask` () const
  create a BitArray indicating the inactive discrete real subset of all {continuous, discrete {int,string,real}} variables

• `BitArray cdrv_to_all_mask` () const
create a BitArray indicating the complement discrete real subset of all {continuous,discrete \{int,string,real\}} variables

- BitArray \texttt{adrv\_to\_all\_mask}() const
  create a BitArray indicating the all discrete real subset of all {continuous,discrete \{int,string,real\}} variables

- \textbf{void} \texttt{initialize\_active\_start\_counts}()
  initialize start index and counts for active variables

- \textbf{void} \texttt{initialize\_inactive\_start\_counts}()
  initialize start index and counts for inactive variables

- \textbf{void} \texttt{initialize\_active\_components}()
  initialize the active components totals given active variable counts

- \textbf{void} \texttt{initialize\_inactive\_components}()
  initialize the inactive components totals given inactive variable counts

- \textbf{const} BitArray & \texttt{all\_relaxed\_discrete\_int}() const
  return allRelaxedDiscreteInt

- \textbf{const} BitArray & \texttt{all\_relaxed\_discrete\_real}() const
  return allRelaxedDiscreteReal

- StringMultiArrayView \texttt{all\_continuous\_labels}(size_t start, size_t num\_items) const
  get num\_items continuous labels beginning at index start

- \textbf{void} \texttt{all\_continuous\_labels(StringMultiArrayConstView cv\_labels, size\_t start, size\_t num\_items)}
  set num\_items continuous labels beginning at index start

- \textbf{void} \texttt{all\_continuous\_label(const String \&cv\_label, size\_t index)}
  set continuous label at index start

- StringMultiArrayView \texttt{all\_discrete\_int\_labels}(size_t start, size_t num\_items) const
  get num\_items discrete integer labels beginning at index start

- \textbf{void} \texttt{all\_discrete\_int\_labels(StringMultiArrayConstView div\_labels, size\_t start, size\_t num\_items)}
  set num\_items discrete integer labels beginning at index start

- \textbf{void} \texttt{all\_discrete\_int\_label(const String \&div\_label, size\_t index)}
  set discrete integer label at index start

- StringMultiArrayView \texttt{all\_discrete\_string\_labels}(size_t start, size_t num\_items) const
  get num\_items discrete string labels beginning at index start

- \textbf{void} \texttt{all\_discrete\_string\_labels(StringMultiArrayConstView dsv\_labels, size\_t start, size\_t num\_items)}
  set num\_items discrete string labels beginning at index start

- \textbf{void} \texttt{all\_discrete\_string\_label(const String \&dsv\_label, size\_t index)}
  set discrete string label at index start

- StringMultiArrayView \texttt{all\_discrete\_real\_labels}(size_t start, size_t num\_items) const
  get num\_items discrete real labels beginning at index start

- \textbf{void} \texttt{all\_discrete\_real\_labels(StringMultiArrayConstView drv\_labels, size\_t start, size\_t num\_items)}
  set num\_items discrete real labels beginning at index start

- \textbf{void} \texttt{all\_discrete\_real\_label(const String \&drv\_label, size\_t index)}
  set discrete real label at index start

- \textbf{void} \texttt{assemble\_all\_labels(StringArray \&all\_labels)}
  assemble all variable labels (continuous and discrete \{ int,string,real \}) in standard (input specification-based) order

- UShortMultiArrayConstView \texttt{all\_continuous\_types}(size_t start, size_t num\_items) const
get num_items continuous types beginning at index start
• void all_continuous_types (UShortMultiArrayConstView cv_types, size_t start, size_t num_items)
  set num_items continuous types beginning at index start
• void all_continuous_type (unsigned short cv_type, size_t index)
  set continuous type at index
• UShortMultiArrayConstView all_discrete_int_types (size_t start, size_t num_items) const
  get num_items discrete integer types beginning at index start
• void all_discrete_int_types (UShortMultiArrayConstView div_types, size_t start, size_t num_items)
  set num_items discrete integer types beginning at index start
• void all_discrete_int_type (unsigned short div_type, size_t index)
  set discrete integer type at index
• UShortMultiArrayConstView all_discrete_string_types (size_t start, size_t num_items) const
  get num_items discrete string types beginning at index start
• void all_discrete_string_types (UShortMultiArrayConstView dsv_types, size_t start, size_t num_items)
  set num_items discrete string types beginning at index start
• void all_discrete_string_type (unsigned short dsv_type, size_t index)
  set discrete string type at index
• UShortMultiArrayConstView all_discrete_real_types (size_t start, size_t num_items) const
  get num_items discrete real types beginning at index start
• void all_discrete_real_types (UShortMultiArrayConstView drv_types, size_t start, size_t num_items)
  set num_items discrete real types beginning at index start
• void all_discrete_real_type (unsigned short drv_type, size_t index)
  set discrete real type at index
• SizetMultiArrayConstView all_continuous_ids (size_t start, size_t num_items) const
  get num_items continuous ids beginning at index start
• void all_continuous_ids (SizetMultiArrayConstView cv_ids, size_t start, size_t num_items)
  set num_items continuous ids beginning at index start
• void all_continuous_id (size_t id, size_t index)
  set num_items continuous ids beginning at index start
• SizetMultiArrayConstView all_discrete_int_ids (size_t start, size_t num_items) const
  get num_items discrete int ids beginning at index start
• SizetMultiArrayConstView all_discrete_string_ids (size_t start, size_t num_items) const
  get num_items discrete string ids beginning at index start
• SizetMultiArrayConstView all_discrete_real_ids (size_t start, size_t num_items) const
  get num_items discrete real ids beginning at index start
• const String & id () const
  return the user-provided or default Variables identifier
• const SizetArray & components_totals () const
  return variable type counts for {continuous,discrete integer,discrete real} {design,aleatory uncertain,epistemic uncertain,state}
• const SizetArray & active_components_totals () const
  return active variable type counts for {continuous,discrete integer,discrete real} {design,aleatory uncertain,epistemic uncertain,state}
• const SizetArray & inactive_components_totals () const
  return inactive variable type counts for \{continuous,discrete integer,discrete real\} \{design,aleatory uncertain,epistemic uncertain,state\}

• size_t vc_lookup (unsigned short key) const
  retrieve the variables type count within svdRep->variablesComponents corresponding to (a fine-grain variables type) key

• const std::pair <short, short> & view () const
  retrieve the Variables view

• void inactive_view (short view2)
  set the inactive Variables view

• size_t cv () const
  get number of active continuous vars

• size_t cv_start () const
  get start index of active continuous vars

• size_t div () const
  get number of active discrete int vars

• size_t div_start () const
  get start index of active discrete int vars

• size_t dsv () const
  get number of active discrete string vars

• size_t dsv_start () const
  get start index of active discrete string vars

• size_t drv () const
  get number of active discrete real vars

• size_t drv_start () const
  get start index of active discrete real vars

• size_t icv () const
  get number of inactive continuous vars

• size_t icv_start () const
  get start index of inactive continuous vars

• size_t idiv () const
  get number of inactive discrete int vars

• size_t idiv_start () const
  get start index of inactive discrete int vars

• size_t idsv () const
  get number of inactive discrete string vars

• size_t idsv_start () const
  get start index of inactive discrete string vars

• size_t idrv () const
  get number of inactive discrete real vars

• size_t idrv_start () const
  get start index of inactive discrete real vars

• void cv (size_t ncv)
set number of active continuous vars

- void cv_start (size_t cvs)

set start index of active continuous vars

- void div (size_t ndiv)

set number of active discrete int vars

- void div_start (size_t divs)

set start index of active discrete int vars

- void dsv (size_t ndsv)

set number of active discrete string vars

- void dsv_start (size_t dsvs)

set start index of active discr string vars

- void drv (size_t ndrv)

set number of active discrete real vars

- void drv_start (size_t drvs)

set start index of active discrete real vars

- void icv (size_t nicv)

set number of inactive continuous vars

- void icv_start (size_t icvs)

set start index of inactive continuous vars

- void idiv (size_t nidiv)

set number of inactive discrete int vars

- void idiv_start (size_t idivs)

set start index of inactive discr int vars

- void idsv (size_t nidsv)

set number of inactive discr string vars

- void idsv_start (size_t idsvs)

set start index of inact discr string vars

- void idrv (size_t nidrv)

set number of inactive discrete real vars

- void idrv_start (size_t idrvs)

set start index of inact discr real vars

- template<class Archive>
  void serialize (Archive &ar, const unsigned int version)

Private Member Functions

- template<class Archive>
  void serialize (Archive &ar, const unsigned int version)

  serialize through the pointer, which requires object tracking: write and read are symmetric for this class

Private Attributes

- boost::shared_ptr
  < SharedVariablesDataRep > svdRep

  pointer to the body (handle-body idiom)
Friends

- class boost::serialization::access
  
  allow boost access to serialize this class

14.235.1 Detailed Description

Container class encapsulating variables data that can be shared among a set of Variables instances.

An array of Variables objects (e.g., Analyzer::allVariables) contains repeated configuration data (id’s, labels, counts). SharedVariablesData employs a handle-body idiom to allow this shared data to be managed in a single object with many references to it, one per Variables object in the array. This allows scaling to larger sample sets.

14.235.2 Member Function Documentation

SharedVariablesData copy ( ) const

create a deep copy of the current object and return by value

Deep copies are used when recasting changes the nature of a Variables set.

References Dakota::svd(), and SharedVariablesData::svdRep.

Referenced by Model::Model().

The documentation for this class was generated from the following files:

- SharedVariablesData.hpp
- SharedVariablesData.cpp

14.236 SharedVariablesDataRep Class Reference

The representation of a SharedVariablesData instance. This representation, or body, may be shared by multiple SharedVariablesData handle instances.

Public Member Functions

- ~SharedVariablesDataRep ()
  
  destructor must be public for shared_ptr

- template<class Archive >
  void save (Archive &ar, const unsigned int version) const

- template<class Archive >
  void load (Archive &ar, const unsigned int version)

Private Member Functions

- SharedVariablesDataRep (const ProblemDescDB &problem_db, const std::pair< short, short > &view)
  
  standard constructor

- SharedVariablesDataRep (const std::pair< short, short > &view, const std::map< unsigned short, size_t > &vars_comps, const BitArray &all_relax_di, const BitArray &all_relax_dr)
  
  medium weight constructor providing detailed variable counts

- SharedVariablesDataRep (const std::pair< short, short > &view, const SizetArray &vars_comps_totals, const BitArray &all_relax_di, const BitArray &all_relax_dr)
  
  lightweight constructor providing variable count totals
• **SharedVariablesDataRep ()**
  - default constructor

• **void initialize_components_totals (const ProblemDescDB &problem_db)**
  - populate variables {Components,CompsTotals} from user variable type and count specifications

• **void components_to_totals ()**
  - update variables CompsTotals from variables Components

• **void relax_noncategorical (const ProblemDescDB &problem_db)**
  - populate allRelaxedDiscrete {Int,Real} from user specifications (relax variables that are not declared as categorical)

• **void set_relax (const BitArray &user_cat_spec, size_t ucs_index, size_t ard_cntr, BitArray &ard_container)**
  - Set the ard_cntr entry in the all-relaxed-discrete integer or real container ard_container, based on user-specification of categorical, accounting for empty.

• **void all_counts (size_t &num_acv, size_t &num_adiv, size_t &num_adsv, size_t &num_adrv) const**
  - compute all variables sums from variables CompsTotals

• **void relax_counts (size_t &num_cv, size_t &num_div, size_t &num_drv, size_t offset_di, size_t offset_dr) const**
  - adjust counts based on allRelaxedDiscrete {Int,Real}

• **void design_counts (size_t &num_cdv, size_t &num_ddiv, size_t &num_ddsv, size_t &num_ddrv) const**
  - compute design variables sums from variables CompsTotals

• **void aleatory_uncertain_counts (size_t &num_cauv, size_t &num_dauiv, size_t &num_dauvs, size_t &num_daurv) const**
  - compute aleatory uncertain variables sums from variables CompsTotals

• **void epistemic_uncertain_counts (size_t &num_ceuv, size_t &num_deuiv, size_t &num_deuv, size_t &num_deuv, size_t &num_deusv) const**
  - compute epistemic uncertain variables sums from variables CompsTotals

• **void uncertain_counts (size_t &num_cuv, size_t &num_duv, size_t &num_dusv, size_t &num_durv) const**
  - compute uncertain variables sums from variables CompsTotals

• **void state_counts (size_t &num_csv, size_t &num_dsv, size_t &num_dsuv, size_t &num_dsrv) const**
  - compute state variables sums from variables CompsTotals

• **void view_start_counts (short view, size_t &cv_start, size_t &div_start, size_t &dsv_start, size_t &drv_start, size_t &num_cv, size_t &num_div, size_t &num_dsv, size_t &num_drv) const**
  - define start indices and counts for active variables based on view

• **void view_subsets (short view, bool &cdv, bool &ddv, bool &cauv, bool &dauv, bool &ceuv, bool &deuv, bool &csv, bool &dsv) const**
  - define active variable subsets based on active view

• **void size_all_labels ()**
  - size all {Continuous,DiscreteInt,DiscreteString,DiscreteReal} Labels, with or without discrete relaxation

• **void size_all_types ()**
  - size all {Continuous,DiscreteInt,DiscreteString,DiscreteReal} Types, with or without discrete relaxation

• **size_t ev_index_to_all_index (size_t ev_index, bool cdv, bool cauv, bool ceuv, bool csv) const**
  - convert index within active continuous variables (as identified by bools) to index within aggregated variables (all continuous, discrete {int,string,real})

• **size_t div_index_to_all_index (size_t div_index, bool cdv, bool cauv, bool deuv, bool dsv) const**
  - convert index within active discrete integer variables (as identified by bools) to index within aggregated variables (all continuous, discrete {int,string,real})
CHAPTER 14. CLASS DOCUMENTATION

- size_t dsv_index_to_all_index(size_t dsv_index, bool ddv, bool dauv, bool deuv, bool dsv) const
  
  convert index within active discrete string variables (as identified by bools) to index within aggregated variables (all continuous, discrete {int,string,real})

- size_t drv_index_to_all_index(size_t drv_index, bool ddv, bool dauv, bool deuv, bool dsv) const
  
  convert index within active discrete real variables (as identified by bools) to index within aggregated variables (all continuous, discrete {int,string,real})

- BitArray cv_to_all_mask(bool cdv, bool cauv, bool ceuv, bool csv) const
  
  create a BitArray indicating the active continuous subset of all \{continuous,discrete \{int,string,real\}\} variables

- BitArray div_to_all_mask(bool ddv, bool dauv, bool deuv, bool dsv) const
  
  create a BitArray indicating the active discrete int subset of all \{continuous,discrete \{int,string,real\}\} variables

- BitArray dsv_to_all_mask(bool ddv, bool dauv, bool deuv, bool dsv) const
  
  create a BitArray indicating the active discrete string subset of all \{continuous,discrete \{int,string,real\}\} variables

- BitArray drv_to_all_mask(bool ddv, bool dauv, bool deuv, bool dsv) const
  
  create a BitArray indicating the active discrete real subset of all \{continuous,discrete \{int,string,real\}\} variables

- void initialize_all_labels(const ProblemDescDB &problem_db)
  
  aggregate all\{Continuous,DiscreteInt,DiscreteString,DiscreteReal\}Labels from user specification or defaults

- void initialize_all_types()
  
  initialize all\{Continuous,DiscreteInt,DiscreteString,DiscreteReal\}Types, with or without discrete relaxation

- void initialize_all_ids()
  
  initialize allContinuousIds (discrete not currently needed), with or without discrete relaxation

- void initialize_active_start_counts()
  
  initialize \{c,di,dr\}vStart and num\{D,DI,DR\}V

- void initialize_inactive_start_counts()
  
  initialize i\{c,di,dr\}vStart and numI\{D,DI,DR\}V

- void initialize_active_components()
  
  initialize activeVarsCompsTotals given \{c,di,dr\}vStart and num\{C,DI,DR\}V

- void initialize_inactive_components()
  
  initialize inactiveVarsCompsTotals given i\{c,di,dr\}vStart and numI\{C,DI,DR\}V

- size_t vc_lookup(unsigned short key) const
  
  retrieve the count within variablesComponents corresponding to key

- void copy_rep(SharedVariablesDataRep *svd_rep)
  
  copy the data from svd_rep to the current representation

- template<class Archive>
  void save(Archive &ar, const unsigned int version) const
    
    serialize the core shared variables data

- template<class Archive>
  void load(Archive &ar, const unsigned int version)
    
    load the core shared variables data and restore class state

- BOOST_SERIALIZATION_SPLITT_MEMBER() String variablesId
  
  variables identifier string from the input file
Private Attributes

- `std::map< unsigned short, size_t > variablesComponents`
  map linking variable types to counts
- `SisetArray variablesCompsTotals`
  totals for variable type counts for `{continuous,discrete integer,discrete string,discrete real}` `{design,aleatory uncertain,epistemic uncertain,state}`.
- `SisetArray activeVarsCompsTotals`
  totals for active variable type counts for `{continuous,discrete integer,discrete string,discrete real}` `{design,aleatory uncertain,epistemic uncertain,state}`.
- `SisetArray inactiveVarsCompsTotals`
  totals for inactive variable type counts for `{continuous,discrete integer,discrete string,discrete real}` `{design,aleatory uncertain,epistemic uncertain,state}`.
- `std::pair< short, short > variablesView`
  the variables view pair containing active (first) and inactive (second) view enumerations
- `size_t cvStart`
  start index of active continuous variables within allContinuousVars
- `size_t divStart`
  start index of active discrete integer variables within allDiscreteIntVars
- `size_t dsvStart`
  start index of active discrete string vars within allDiscreteStringVars
- `size_t drvStart`
  start index of active discrete real variables within allDiscreteRealVars
- `size_t icvStart`
  start index of inactive continuous variables within allContinuousVars
- `size_t idivStart`
  start index of inactive discrete integer vars within allDiscreteIntVars
- `size_t idsvStart`
  start index of inactive discrete string vars within allDiscreteStringVars
- `size_t idrvStart`
  start index of inactive discrete real variables within allDiscreteRealVars
- `size_t numCV`
  number of active continuous variables
- `size_t numDIV`
  number of active discrete integer variables
- `size_t numDSV`
  number of active discrete string variables
- `size_t numDRV`
  number of active discrete real variables
- `size_t numICV`
  number of inactive continuous variables
- `size_t numIDIV`
  number of inactive discrete integer variables
- `size_t numIDSV`
number of inactive discrete string variables
- size_t numIDRV

number of inactive discrete real variables
- StringMultiArray allContinuousLabels
  array of variable labels for all of the continuous variables
- StringMultiArray allDiscreteIntLabels
  array of variable labels for all of the discrete integer variables
- StringMultiArray allDiscreteStringLabels
  array of variable labels for all of the discrete string variables
- StringMultiArray allDiscreteRealLabels
  array of variable labels for all of the discrete real variables
- UShortMultiArray allContinuousTypes
  array of variable types for all of the continuous variables
- UShortMultiArray allDiscreteIntTypes
  array of variable types for all of the discrete integer variables
- UShortMultiArray allDiscreteStringTypes
  array of variable types for all of the discrete string variables
- UShortMultiArray allDiscreteRealTypes
  array of variable types for all of the discrete real variables
- SizetMultiArray allContinuousIds
  array of 1-based position identifiers for the all continuous variables array
- SizetMultiArray allDiscreteIntIds
  array of 1-based ids (into total variable set) for discrete int
- SizetMultiArray allDiscreteStringIds
  array of 1-based ids (into total variable set) for discrete string
- SizetMultiArray allDiscreteRealIds
  array of 1-based ids (into total variable set) for discrete real
- BitArray allRelaxedDiscreteInt
  array of booleans to indicate relaxation (promotion from DiscreteInt to Continuous) for all specified discrete int variables Note: container will be empty when not relaxing variables
- BitArray allRelaxedDiscreteReal
  array of booleans to indicate relaxation (promotion from DiscreteReal to Continuous) for all specified discrete real variables Note: container will be empty when not relaxing variables

Friends

- class SharedVariablesData
- class boost::serialization::access
  allow boost access to serialize this class

14.236.1 Detailed Description

The representation of a SharedVariablesData instance. This representation, or body, may be shared by multiple SharedVariablesData handle instances.

The SharedVariablesData/SharedVariablesDataRep pairs utilize a handle-body idiom (Coplien, Advanced C++).
14.236. **Member Function Documentation**

```cpp
void copy_rep ( SharedVariablesDataRep * svd_rep ) [private]
```

copy the data from svd_rep to the current representation

Deep copies are used when recasting changes the nature of a `Variables` set.

References `SharedVariablesDataRep::activeVarsCompsTotals`, `SharedVariablesDataRep::allContinuousIds`, `SharedVariablesDataRep::allContinuousLabels`, `SharedVariablesDataRep::allContinuousTypes`, `SharedVariablesDataRep::allDiscreteIntIds`, `SharedVariablesDataRep::allDiscreteIntLabels`, `SharedVariablesDataRep::allDiscreteIntTypes`, `SharedVariablesDataRep::allDiscreteRealIds`, `SharedVariablesDataRep::allDiscreteRealLabels`, `SharedVariablesDataRep::allDiscreteRealTypes`, `SharedVariablesDataRep::allRelaxedDiscreteInt`, `SharedVariablesDataRep::allRelaxedDiscreteReal`, `SharedVariablesDataRep::cvStart`, `SharedVariablesDataRep::divStart`, `SharedVariablesDataRep::drvStart`, `SharedVariablesDataRep::dsvStart`, `SharedVariablesDataRep::icvStart`, `SharedVariablesDataRep::idivStart`, `SharedVariablesDataRep::idrvStart`, `SharedVariablesDataRep::idsvStart`, `SharedVariablesDataRep::inactiveVarsCompsTotals`, `SharedVariablesDataRep::numCV`, `SharedVariablesDataRep::numDIV`, `SharedVariablesDataRep::numDRV`, `SharedVariablesDataRep::numDSV`, `SharedVariablesDataRep::numICV`, `SharedVariablesDataRep::numIDIV`, `SharedVariablesDataRep::numIDRV`, `SharedVariablesDataRep::numIDSV`, `SharedVariablesDataRep::variablesComponents`, `SharedVariablesDataRep::variablesCompsTotals`, and `SharedVariablesDataRep::variablesView`.

14.236.3 **Member Data Documentation**

`SizetArray variablesCompsTotals [private]`

totals for variable type counts for `{continuous,discrete integer,discrete string,discrete real}` `{design,aleatory uncertain,epistemic uncertain,state}`.

This data reflects the variable counts as originally specified and is not altered by relaxation.

Referenced by `SharedVariablesDataRep::aleatory_uncertain_counts()`, `SharedVariablesDataRep::all_counts()`, `SharedVariablesDataRep::components_to_totals()`, `SharedVariablesDataRep::copy_rep()`, `SharedVariablesDataRep::design_counts()`, `SharedVariablesDataRep::epistemic_uncertain_counts()`, `SharedVariablesDataRep::initialize_active_components()`, `SharedVariablesDataRep::initialize_all_ids()`, `SharedVariablesDataRep::initialize_components_totals()`, `SharedVariablesDataRep::initialize_inactive_components()`, `SharedVariablesDataRep::relax_noncategorical()`, `SharedVariablesDataRep::state_counts()`, `SharedVariablesDataRep::uncertain_counts()`, and `SharedVariablesDataRep::view_start_counts()`.

`SizetArray activeVarsCompsTotals [private]`

totals for active variable type counts for `{continuous,discrete integer,discrete string,discrete real}` `{design,aleatory uncertain,epistemic uncertain,state}`.

This data reflects the variable counts as originally specified and is not altered by relaxation.

Referenced by `SharedVariablesDataRep::copy_rep()`, and `SharedVariablesDataRep::initialize_active_components()`.

`SizetArray inactiveVarsCompsTotals [private]`

totals for inactive variable type counts for `{continuous,discrete integer,discrete string,discrete real}` `{design,aleatory uncertain,epistemic uncertain,state}`.

This data reflects the variable counts as originally specified and is not altered by relaxation.

Referenced by `SharedVariablesDataRep::copy_rep()`, and `SharedVariablesDataRep::initialize_inactive_components()`.
SizetMultiArray allContinuousIds  [private]
array of 1-based position identifiers for the all continuous variables array
These identifiers define positions of the all continuous variables array within the total variable sequence. A primary use case is for defining derivative ids (DVV) based on an active subset.
Referenced by SharedVariablesDataRep::copy_rep(), and SharedVariablesDataRep::initialize_all_ids().
The documentation for this class was generated from the following files:
- SharedVariablesData.hpp
- SharedVariablesData.cpp

14.237  SimulationModel Class Reference
Derived model class which utilizes a simulation-based application interface to map variables into responses.
Inheritance diagram for SimulationModel:

```
Model
SimulationModel
```

Public Member Functions
- **SimulationModel** (ProblemDescDB &problem_db)
  
  *constructor*
- **~SimulationModel** ()
  
  *destructor*

Protected Member Functions
- **Interface & derived_interface** ()
  
  *Return the "default" or maximal ActiveSet for the model.*
- **size_t solution_levels** (bool lwr_bnd=true) const
  
  *return size of solnControlCostMap, optionally enforcing lower bound of 1 solution level*
- **void solution_level_index** (unsigned short lev_index)
  
  *activate entry in solnControlCostMap*
- **unsigned short solution_level_index** () const
  
  *return active entry in solnControlCostMap*
- **RealVector solution_level_costs** () const
  
  *return all cost estimates from solnControlCostMap*
- **Real solution_level_cost** () const
  
  *return active cost estimate from solnControlCostMap*
- **void derived_evaluate** (const ActiveSet &set)
  
  *portion of evaluate() specific to SimulationModel (invokes a synchronous map() on userDefinedInterface)*
- **void derived_evaluate_nowait** (const ActiveSet &set)
portion of evaluate_nowait() specific to SimulationModel (invokes an asynchronous map() on userDefinedInterface)

• const IntResponseMap & derived_synchronize()
  portion of synchronize() specific to SimulationModel (invokes synch() on userDefinedInterface)

• const IntResponseMap & derived_synchronize_nowait()
  portion of synchronize_nowait() specific to SimulationModel (invokes synch_nowait() on userDefinedInterface)

• short local_eval_synchronization()
  return userDefinedInterface synchronization setting

• int local_eval_concurrency()
  return userDefinedInterface asynchronous evaluation concurrency

• bool derived_master_overload() const
  flag which prevents overloading the master with a multiprocessor evaluation (request forwarded to userDefinedInterface)

• IntIntPair estimate_partition_bounds(int max_eval_concurrency)
  estimate the minimum and maximum partition sizes that can be utilized by this Model

• void derived_init_communicators(ParLevLIter pl_iter, int max_eval_concurrency, bool recurse_flag=true)
  set up SimulationModel for parallel operations (request forwarded to userDefinedInterface)

• void derived_init_serial()
  set up SimulationModel for serial operations (request forwarded to userDefinedInterface).

• void derived_set_communicators(ParLevLIter pl_iter, int max_eval_concurrency, bool recurse_flag=true)
  set active parallel configuration for the SimulationModel (request forwarded to userDefinedInterface)

• void serve_run(ParLevLIter pl_iter, int max_eval_concurrency)
  Service userDefinedInterface job requests received from the master. Completes when a termination message is received from stop_servers().

• void stop_servers()
  executed by the master to terminate userDefinedInterface server operations when SimulationModel iteration is complete.

• const String & interface_id() const
  return the userDefinedInterface identifier

• int derived_evaluation_id() const
  return the current evaluation id (simModelEvalCnt)

• bool evaluation_cache(bool recurse_flag=true) const
  return flag indicated usage of an evaluation cache by the SimulationModel (request forwarded to userDefinedInterface)

• bool restart_file(bool recurse_flag=true) const
  return flag indicated usage of a restart file by the SimulationModel (request forwarded to userDefinedInterface)

• void set_evaluation_reference()
  set the evaluation counter reference points for the SimulationModel (request forwarded to userDefinedInterface)

• void fine_grained_evaluation_counters() 
  request fine-grained evaluation reporting within the userDefinedInterface

• void print_evaluation_summary(std::ostream &s, bool minimal_header=false, bool relative_count=true) const
  print the evaluation summary for the SimulationModel (request forwarded to userDefinedInterface)

• void eval_tag_prefix(const String &eval_id_str)
  set the hierarchical eval ID tag prefix
ActiveSet default\_interface\_active\_set ()

Return the "default" or maximal ActiveSet for the userDefinedInterface.

void declare\_sources ()

Declare this model's sources.

Private Member Functions

void initialize\_solution\_control (const String &control, const RealVector &cost)

process the solution level inputs to define solnControlVarIndex, solnControlVarType, and solnControlCostMap

Private Attributes

Interface userDefinedInterface

the interface used for mapping variables to responses

short solnCntlVarType

type of the discrete variable that controls the set/range of solution levels

size\_t solnCntlADVIndex

index of the discrete variable (within all view) that controls the set/range of solution levels

size\_t solnCntlRVIndex

index of the discrete set variable (within aggregated array of RandomVariables) that controls the set/range of solution levels

std::map< Real, size\_t > solnCntlCostMap

sorted array of relative costs associated with a set of solution levels

size\_t simModelEvalCntr

counter for calls to derived\_evaluate()/derived\_evaluate\_nowait()

IntIntMap simIdMap

map from userDefinedInterface evaluation ids to SimulationModel ids (may differ in case where the same interface instance is shared by multiple models)

IntResponseMap simResponseMap

map of simulation-based responses returned by derived\_synchronize() and derived\_synchronize\_nowait()

Additional Inherited Members

14.237.1 Detailed Description

Derived model class which utilizes a simulation-based application interface to map variables into responses.

The SimulationModel class is the simplest of the derived model classes. It provides the capabilities of the original Model class, prior to the development of surrogate and nested model extensions. The derived response computation and synchronization functions utilize an application interface to perform the function evaluations.

14.237.2 Member Function Documentation

Interface & derived\_interface ( ) [inline], [protected], [virtual]

Return the "default" or maximal ActiveSet for the model.

return userDefinedInterface

Reimplemented from Model.

References SimulationModel::userDefinedInterface.
void eval_tag_prefix ( const String & eval_id_str ) [protected], [virtual]

set the hierarchical eval ID tag prefix
- SimulationModel doesn’t need to change the tagging, so just forward to Interface
- Reimplemented from Model.
- References Interface::eval_tag_prefix(), and SimulationModel::userDefinedInterface.
- The documentation for this class was generated from the following files:
  - SimulationModel.hpp
  - SimulationModel.cpp

14.238 SimulationResponse Class Reference

Container class for response functions and their derivatives. SimulationResponse provides the body class.

Inheritance diagram for SimulationResponse:

```
Response
  |
  |
SimulationResponse
```

Public Member Functions

- **SimulationResponse ()
  default constructor**

- **SimulationResponse (const Variables &vars, const ProblemDescDB &problem_db)
  standard constructor built from problem description database**

- **SimulationResponse (const SharedResponseData &srd, const ActiveSet &set)
  alternate constructor that shares a SharedResponseData instance**

- **SimulationResponse (const SharedResponseData &srd)
  alternate constructor that shares a SharedResponseData instance**

- **SimulationResponse (const ActiveSet &set)
  alternate constructor using limited data**

- **~SimulationResponse ()
  destructor**

Additional Inherited Members

14.238.1 Detailed Description

Container class for response functions and their derivatives. SimulationResponse provides the body class.

The SimulationResponse class is the "representation" of the response container class. It is the "body" portion of the "handle-body idiom" (see Coplien "Advanced C++", p. 58). The handle class (Response) provides for memory efficiency in management of multiple response objects through reference counting and representation sharing. The body class (SimulationResponse) actually contains the response data (functionValues, function-Gradients, functionHessians, etc.). The representation is hidden in that an instance of SimulationResponse may
only be created by Response. Therefore, programmers create instances of the Response handle class, and only
need to be aware of the handle/body mechanisms when it comes to managing shallow copies (shared representa-
tion) versus deep copies (separate representation used for history mechanisms).

The documentation for this class was generated from the following files:

- SimulationResponse.hpp
- SimulationResponse.cpp

14.239 SNLLBase Class Reference

Base class for OPT++ optimization and least squares methods.

Inheritance diagram for SNLLBase:

```
SNLLBase
 / | \
|  v  |
SNLLLeastSq  SNLLOptimizer
```

**Public Member Functions**

- **SNLLBase ()**
  
  default constructor

- **SNLLBase (ProblemDescDB &problem_db)**
  
  standard constructor

- **~SNLLBase ()**
  
  destructor

**Protected Member Functions**

- void **copy_con_vals_dak_to_optpp** (const RealVector &local_fn_vals, RealVector &g, size_t offset)
  
  convenience function for copying local_fn_vals to g; used by constraint evaluator functions

- void **copy_con_vals_optpp_to_dak** (const RealVector &g, RealVector &local_fn_vals, size_t offset)
  
  convenience function for copying g to local_fn_vals; used in final solution logging

- void **copy_con_grad** (const RealMatrix &local_fn_grads, RealMatrix &grad_g, size_t offset)
  
  convenience function for copying local_fn_grads to grad_g; used by constraint evaluator functions

- void **copy_con_hess** (const RealSymMatrixArray &local_fn_hessians, OPTPP::OptppArray< RealSymMatrix, OPTPP::OptimizeClass * the_optimizer, OPTPP::FDNLF1 * fn_hess_fcn, OPTPP::FDNLF1 * fd_hess_fcn > &hess_g, size_t offset)
  
  convenience function for copying local_fn_hessians to hess_g; used by constraint evaluator functions

- void **snll_pre_instantiate** (bool bound_constr_flag, int num_constr)
  
  convenience function for setting OPT++ options prior to the method instantiation

- void **snll_post_instantiate** (int num_cv, bool vendor_num_grad_flag, const String &finite_diff_type, const RealVector &fdss, int max_iter, int max_fn_evals, Real conv_tol, Real grad_tol, Real max_step, bool bound_constr_flag, int num_constr, short output_lev, OPTPP::OptimizeClass * the_optimizer, OPTPP::NLP0 * nlf_objective, OPTPP::FDNLF1 * fd_nlf1, OPTPP::FDNLF1 * fd_nlf1_con)
  
  convenience function for setting OPT++ options after the method instantiation
• void snll_initialize_run (OPTPP::NLP0 *nlf_objective, OPTPP::NLP *nlp_constraint, const RealVector &init_pt, bool bound_constr_flag, const RealVector &lower_bnds, const RealVector &upper_bnds, const RealMatrix &lin_ineq_coeffs, const RealVector &lin_ineq_lbnds, const RealVector &lin_ineq_ubnds, const RealMatrix &lin_eq_coeffs, const RealVector &lin_eq_targets, const RealVector &nln_ineq_lbnds, const RealVector &nln_ineq_ubnds, const RealVector &nln_eq_targets)
  convenience function for OPT++ configuration prior to the method invocation
• void snll_post_run (OPTPP::NLP0 *nlf_objective)
  convenience function for setting OPT++ options after the method instantiations
• void reset_base ()
  reset last\{FnEvalLocn,EvalMode,EvalVars\}

Static Protected Member Functions
• static void init_fn (int n, RealVector &x)
  An initialization mechanism provided by OPT++ (not currently used).

Protected Attributes
• String searchMethod
  value_based_line_search, gradient_based_line_search, trust_region, or tr_pds
• OPTPP::SearchStrategy searchStrat
  enum: LineSearch, TrustRegion, or TrustPDS
• OPTPP::MeritFcn meritFn
  enum: NormFmu, ArgaezTapia, or VanShanno
• Real maxStep
  value from max_step specification
• Real stepLenToBndry
  value from steplength_to_boundary specification
• Real centeringParam
  value from centering_parameter specification
• bool constantASVFlag
  flags a user selection of active_set_vector == constant. By mapping this into mode override, reliance on duplicate
detection can be avoided.

Static Protected Attributes
• static Minimizer * optLsqInstance
  pointer to the active base class object instance used within the static evaluator functions in order to avoid the need
  for static data
• static bool modeOverrideFlag
  flags OPT++ mode override (for combining value, gradient, and Hessian requests)
• static EvalType lastFnEvalLocn
  an enum used to track whether an nlf evaluator or a constraint evaluator was the last location of a function eval-
  uation
• static int lastEvalMode
  copy of mode from constraint evaluators
• static RealVector lastEvalVars

  copy of variables from constraint evaluators

### 14.239.1 Detailed Description

Base class for OPT++ optimization and least squares methods.

The `SNLLBase` class provides a common base class for `SNLLOptimizer` and `SNLLLeastSq`, both of which are wrappers for OPT++, a C++ optimization library from the Computational Sciences and Mathematics Research (CSMR) department at Sandia’s Livermore CA site.

The documentation for this class was generated from the following files:

- SNLLBase.hpp
- SNLLBase.cpp

### 14.240 SNLLLeastSq Class Reference

Wrapper class for the OPT++ optimization library.

Inheritance diagram for SNLLLeastSq:

```
  SNLLLeastSq
  |      |      |      
  |      |      |      
  |      |      |      
  SNLLBase
  |      |      |      
  |      |      |      
  LeastSq
  |      |      |
  Minimizer
  |      |
  Iterator
```

#### Public Member Functions

- **SNLLLeastSq** (ProblemDescDB &problem_db, Model &model)
  - standard constructor
- **SNLLLeastSq** (const String &method_name, Model &model)
  - alternate constructor for instantiations without ProblemDescDB support
- **~SNLLLeastSq** ()
  - destructor
- void **core_run** ()
  - compute the least squares solution
- void **reset** ()
  - restore initial state for repeated sub-iterator executions
Protected Member Functions

- void initialize_run()
  *invokes LeastSq::initialize_run, SNLLBase::snll_initialize_run, and performs other set-up*
- void finalize_run()
  *restores instances*

Static Private Member Functions

- static void nlf2_evaluator_gn(int mode, int n, const RealVector &x, double &f, RealVector &grad_f, RealSymMatrix &hess_f, int &result_mode)
  *objective function evaluator function which obtains values and gradients for least square terms and computes objective function value, gradient, and Hessian using the Gauss-Newton approximation.*
- static void constraint1_evaluator_gn(int mode, int n, const RealVector &x, RealVector &g, RealMatrix &grad_g, int &result_mode)
  *constraint evaluator function which provides constraint values and gradients to OPT++ Gauss-Newton methods.*
- static void constraint2_evaluator_gn(int mode, int n, const RealVector &x, RealVector &g, RealMatrix &grad_g, OPTPP::OptppArray<RealSymMatrix> &hess_g, int &result_mode)
  *constraint evaluator function which provides constraint values, gradients, and Hessians to OPT++ Gauss-Newton methods.*

Private Attributes

- SNLLLeastSq * prevSnllLsqInstance
  *pointer to the previously active object instance used for restoration in the case of iterator/model recursion*
- OPTPP::NLP0 * nlf0Objective
  *objective NLF base class pointer*
- OPTPP::NLP0 * nlfConstraint
  *constraint NLF base class pointer*
- OPTPP::NLP * nlpConstraint
  *constraint NLP pointer*
- OPTPP::NLF2 * nlf2
  *pointer to objective NLF for full Newton optimizers*
- OPTPP::NLF2 * nlf2Con
  *pointer to constraint NLF for full Newton optimizers*
- OPTPP::NLF1 * nlf1Con
  *pointer to constraint NLF for Quasi Newton optimizers*
- OPTPP::OptimizeClass * theOptimizer
  *optimizer base class pointer*
- OPTPP::OptNewton * optnewton
  *Newton optimizer pointer.*
- OPTPP::OptBCNewton * optbcnewton
  *Bound constrained Newton optimizer ptr.*
- OPTPP::OptDHNIPS * optdhnips
  *Disaggregated Hessian NIPS optimizer ptr.*
Static Private Attributes

- static SNLLLeastSq * snllLSqInstance
  
  pointer to the active object instance used within the static evaluator functions in order to avoid the need for static data

Additional Inherited Members

14.240.1 Detailed Description

Wrapper class for the OPT++ optimization library.

The SNLLLeastSq class provides a wrapper for OPT++, a C++ optimization library of nonlinear programming and pattern search techniques from the Computational Sciences and Mathematics Research (CSMR) department at Sandia's Livermore CA site. It uses a function pointer approach for which passed functions must be either global functions or static member functions. Any attribute used within static member functions must be either local to that function, a static member, or accessed by static pointer.

The user input mappings are as follows: max_iterations, max_function_evaluations, convergence_tolerance, max_step, gradient_tolerance, search_method, and search_scheme_size are set using OPT++’s setMaxIter(), setMaxFeval(), setFcnTol(), setMaxStep(), setGradTol(), setSearchStrategy(), and setSSS() member functions, respectively; output verbosity is used to toggle OPT++’s debug mode using the setDebug() member function. Internal to OPT++, there are 3 search strategies, while the DAKOTA search_method specification supports 4 (value_based_line_search, gradient_based_line_search, trust_region, or tr_pds). The difference stems from the "is_expensive" flag in OPT++. If the search strategy is LineSearch and "is_expensive" is turned on, then the value_based_line_search is used. Otherwise (the "is_expensive" default is off), the algorithm will use the gradient_based_line_search. Refer to [Meza, J.C., 1994] and to the OPT++ source in the Dakota/packages/OPTPP directory for information on OPT++ class member functions.

14.240.2 Member Function Documentation

void nlf2_evaluator_gn ( int mode, int n, const RealVector & x, double & f, RealVector & grad_f, RealSymMatrix & hess_f, int & result_mode ) [static], [private]

objective function evaluator function which obtains values and gradients for least square terms and computes objective function value, gradient, and Hessian using the Gauss-Newton approximation.

This nlf2 evaluator function is used for the Gauss-Newton method in order to exploit the special structure of the nonlinear least squares problem. Here, \( f(x) = \sum (T_i - \bar{T}_i)^2 \) and Response is made up of residual functions and their gradients along with any nonlinear constraints. The objective function and its gradient vector and Hessian matrix are computed directly from the residual functions and their derivatives (which are returned from the Response object).

References Dakota::abort_handler(), Iterator::activeSet, Model::continuous_variables(), Model::current_response(), Model::evaluate(), Response::function_gradients(), Response::function_values(), Iterator::iteratedModel, SNLLBase::lastEvalMode, SNLLBase::lastEvalVars, SNLLBase::lastFnEvalLocn, Minimizer::numFunctions, Least-Sq::numLeastSqTerms, Minimizer::numNonlinearConstraints, Iterator::outputLevel, ActiveSet::request_vector(), SNLLLeastSq::snllLSqInstance, and Dakota::write_precision.

Referenced by SNLLLeastSq::SNLLLeastSq().

void constraint1_evaluator_gn ( int mode, int n, const RealVector & x, RealVector & g, RealMatrix & grad_g, int & result_mode ) [static], [private]

constraint evaluator function which provides constraint values and gradients to OPT++ Gauss-Newton methods.
While it does not employ the Gauss-Newton approximation, it is distinct from constraint1::evaluator() due to its need to anticipate the required modes for the least squares terms. This constraint evaluator function is used with diaggeregated Hessian NIPS and is currently active.

References Dakota::abort_handler(), Iterator::activeSet, Model::continuous_variables(), SNLLBase::copy_con_grad(), SNLLBase::copy_con_vals_dak_to_optpp(), Model::current_response(), Model::evaluate(), Response::function_gradients(), Response::function_values(), Iterator::iteratedModel, SNLLBase::lastEvalMode, SNLLBase::lastEvalVars, SNLLBase::lastFnEvalLocn, Minimizer::numFunctions, LeastSq::numLeastSqTerms, Iterator::outputLevel, ActiveSet::request_vector(), and SNLLLeastSq::snllLSqInstance.

Referenced by SNLLLeastSq::SNLLLeastSq().

```cpp
void constraint2::evaluator_gn ( int mode, int n, const RealVector & x, RealVector & g, RealMatrix & grad_g, OPTPP::OptppArray<RealSymMatrix> & hess_g, int & result_mode ) [static],
[private]
```

costRAINT evaluator function which provides constraint values, gradients, and Hessians to OPT++ Gauss-Newton methods.

While it does not employ the Gauss-Newton approximation, it is distinct from constraint2::evaluator() due to its need to anticipate the required modes for the least squares terms. This constraint evaluator function is used with full Newton NIPS and is currently inactive.

References Dakota::abort_handler(), Iterator::activeSet, Model::continuous_variables(), SNLLBase::copy_con_grad(), SNLLBase::copy_con_hess(), SNLLBase::copy_con_vals_dak_to_optpp(), Model::current_response(), Model::evaluate(), Response::function_gradients(), Response::function_hessians(), Response::function_values(), Iterator::iteratedModel, SNLLBase::lastEvalMode, SNLLBase::lastEvalVars, SNLLBase::lastFnEvalLocn, SNLLBase::modeOverrideFlag, Minimizer::numFunctions, LeastSq::numLeastSqTerms, Iterator::outputLevel, ActiveSet::request_vector(), and SNLLLeastSq::snllLSqInstance.

The documentation for this class was generated from the following files:

- SNLLLeastSq.hpp
- SNLLLeastSq.cpp

14.241 SNLLLeastSqTraits Class Reference

A version of TraitsBase specialized for SNLLLeastSq.

Inheritance diagram for SNLLLeastSqTraits:

```
TraitsBase
    \|-- SNLLLeastSqTraits
```

**Public Member Functions**

- `SNLLLeastSqTraits ()`
  
  *default constructor*

- `virtual ~SNLLLeastSqTraits ()`
  
  *destructor*

- `virtual bool is_derived ()`
A temporary query used in the refactor:

- **bool supports_continuous_variables()**
  
  *Return the flag indicating whether method supports continuous variables.*

- **bool supports_linear_equality()**
  
  *Return the flag indicating whether method supports linear equalities.*

- **bool supports_linear_inequality()**
  
  *Return the flag indicating whether method supports linear inequalities.*

- **bool supports_nonlinear_equality()**
  
  *Return the flag indicating whether method supports nonlinear equalities.*

- **bool supports_nonlinear_inequality()**
  
  *Return the flag indicating whether method supports nonlinear inequalities.*

### 14.241.1 Detailed Description

A version of TraitsBase specialized for SNLLLeastSq.

The documentation for this class was generated from the following file:

- SNLLLeastSq.hpp

### 14.242 SNLLOptimizer Class Reference

Wrapper class for the OPT++ optimization library.

Inheritance diagram for SNLLOptimizer:

![Inheritance Diagram](image)

#### Public Member Functions

- **SNLLOptimizer (ProblemDescDB &problem_db, Model &model)**
  
  *standard constructor*

- **SNLLOptimizer (const String &method_string, Model &model)**
  
  *alternate constructor for instantiations "on the fly"*

- **SNLLOptimizer (const RealVector &initial_pt, const RealVector &var_l_bnds, const RealVector &var_u_bnds, const RealMatrix &lin_ineq_coeffs, const RealVector &lin_ineq_l_bnds, const RealVector &lin_ineq_u_bnds, const RealMatrix &lin_eq_coeffs, const RealVector &lin_eq_tgts, const RealVector &nln_ineq_l_bnds, const RealVector &nln_ineq_u_bnds, const RealVector &nln_eq_tgts, void(*user_obj_eval)(int mode, int n, const RealVector &x, double &f, RealVector &grad_f, int &result_mode), void(*user_con_eval)(int mode, int n, const RealVector &x, RealVector &g, RealMatrix &grad_g, int &result_mode))**
14.242. SNLLOPTIMIZER CLASS REFERENCE

alternate constructor for instantiations "on the fly"

- **SNLLOPTIMIZER** (const RealVector &initial_pt, const RealVector &var_l_bnds, const RealVector &var_u_bnds, const RealMatrix &lin_ineq_coeffs, const RealVector &lin_ineq_l_bnds, const RealVector &lin_ineq_u_bnds, const RealMatrix &lin_eq_coeffs, const RealVector &lin_eq_tgts, const RealVector &nln_ineq_l_bnds, const RealVector &nln_ineq_u_bnds, const RealVector &nln_eq_tgts, void(*user_obj_eval)(int mode, int n, const RealVector &x, double &f, RealVector &grad_f, int &result_mode), void(*user_con_eval)(int mode, int n, const RealVector &x, RealVector &g, RealMatrix &grad_g, int &result_mode), const int max_iter, const int max_fn_evals, const Real conv_tol, const Real grad_tol, Real max_step)

alternate constructor for instantiations "on the fly", also specifying different optimizer properties

- \texttt{\textasciitilde SNLLOPTIMIZER} ()
  destructor
- void \texttt{core\_run()}  
  Performs the iterations to determine the optimal solution.
- void \texttt{reset()}  
  restore initial state for repeated sub-iterator executions
- void \texttt{declare\_sources()}  
  Declare sources to the evaluations database.

Protected Member Functions

- void \texttt{initialize\_run()}  
  invokes \texttt{Optimizer::initialize\_run()}, \texttt{SNLLBase::snll\_initialize\_run()}, and performs other set-up
- void \texttt{post\_run(std::ostream \&s)}  
  performs data recovery and calls \texttt{Optimizer::post\_run()}
- void \texttt{finalize\_run()}  
  performs cleanup, restores instances and calls parent finalize

Private Member Functions

- void \texttt{default\_instantiate\_q\_newton(void(*obj\_eval)(int mode, int n, const RealVector \&x, double \&f, RealVector \&grad\_f, int \&result\_mode), void(*con\_eval)(int mode, int n, const RealVector \&x, RealVector \&g, RealMatrix \&grad\_g, Real\_symMatrix \&hess\_g, int \&result\_mode))}
  instantiate an OPTPP\_Q\_NEWTON solver using standard settings
- void \texttt{default\_instantiate\_newton(void(*obj\_eval)(int mode, int n, const RealVector \&x, double \&f, RealVector \&grad\_f, Real\_symMatrix \&hess\_f, int \&result\_mode), void(*con\_eval)(int mode, int n, const RealVector \&x, RealVector \&g, RealMatrix \&grad\_g, OPTPP\_\_OptppArray\_Real\_symMatrix \&hess\_g, int \&result\_mode))}
  instantiate an OPTPP\_NEWTON solver using standard settings

Static Private Member Functions

- static void \texttt{nlf0\_evaluator(int n, const RealVector \&x, double \&f, int \&result\_mode)}  
  objective function evaluator function for OPT++ methods which require only function values.
- static void \texttt{nlf1\_evaluator(int mode, int n, const RealVector \&x, double \&f, RealVector \&grad\_f, int \&result\_mode)}  
  objective function evaluator function which provides function values and gradients to OPT++ methods.
CHAPTER 14. CLASS DOCUMENTATION

- static void nlf2_evaluator (int mode, int n, const RealVector &x, double &f, RealVector &grad_f, RealSymMatrix &hess_f, int &result_mode)
  
  objective function evaluator function which provides function values, gradients, and Hessians to OPT++ methods.

- static void constraint0_evaluator (int n, const RealVector &x, RealVector &g, int &result_mode)
  
  constraint evaluator function for OPT++ methods which require only constraint values.

- static void constraint1_evaluator (int mode, int n, const RealVector &x, RealVector &g, RealMatrix &grad_g, int &result_mode)
  
  constraint evaluator function which provides constraint values and gradients to OPT++ methods.

- static void constraint2_evaluator (int mode, int n, const RealVector &x, RealVector &g, RealMatrix &grad_g, OPTPP::OptppArray< RealSymMatrix > &hess_g, int &result_mode)
  
  constraint evaluator function which provides constraint values, gradients, and Hessians to OPT++ methods.

Private Attributes

- SNLLOptimizer * prevSnllOptInstance
  
  pointer to the previously active object instance used for restoration in the case of iterator/model recursion

- OPTPP::NLP0 * nlfObjective
  
  objective NLF base class pointer

- OPTPP::NLP0 * nlfConstraint
  
  constraint NLF base class pointer

- OPTPP::NLP * nlpConstraint
  
  constraint NLP pointer

- OPTPP::NLF0 * nlf0
  
  pointer to objective NLF for nongradient optimizers

- OPTPP::NLF1 * nlf1
  
  pointer to objective NLF for (analytic) gradient-based optimizers

- OPTPP::NLF1 * nlf1Con
  
  pointer to constraint NLF for (analytic) gradient-based optimizers

- OPTPP::FDNLF1 * fdnlf1
  
  pointer to objective NLF for (finite diff) gradient-based optimizers

- OPTPP::FDNLF1 * fdnlf1Con
  
  pointer to constraint NLF for (finite diff) gradient-based optimizers

- OPTPP::NLF2 * nlf2
  
  pointer to objective NLF for full Newton optimizers

- OPTPP::NLF2 * nlf2Con
  
  pointer to constraint NLF for full Newton optimizers

- OPTPP::OptimizeClass * theOptimizer
  
  optimizer base class pointer

- OPTPP::OptPDS * optpds
  
  PDS optimizer pointer.

- OPTPP::OptCG * optcg
  
  CG optimizer pointer.

- OPTPP::OptLBFGS * optlbfgs
  
  L-BFGS optimizer pointer.
• OPTPP::OptNewton * optnewton
  Newton optimizer pointer.
• OPTPP::OptQNewton * optqnewton
  Quasi-Newton optimizer pointer.
• OPTPP::OptFDNewton * optfdnewton
  Finite Difference Newton opt pointer.
• OPTPP::OptBCNewton * optbcnewton
  Bound constrained Newton opt pointer.
• OPTPP::OptBCQNewton * optbcqnewton
  Bnd constrained Quasi-Newton opt ptr.
• OPTPP::OptBCFDNewton * optbcfdnewton
  Bound constrained FD-Newton opt ptr.
• OPTPP::OptNIPS * optnips
  NIPS optimizer pointer.
• OPTPP::OptQNIPS * optqnips
  Quasi-Newton NIPS optimizer pointer.
• OPTPP::OptFDNIPS * optfdnips
  Finite Difference NIPS opt pointer.
• String setUpType
  flag for iteration mode: "model" (normal usage) or "user functions" (user-supplied functions mode for "on the fly" instantiations). NonDReliability currently uses the user functions mode.
• RealVector initialPoint
  holds initial point passed in for "user functions" mode.
• RealVector lowerBounds
  holds variable lower bounds passed in for "user functions" mode.
• RealVector upperBounds
  holds variable upper bounds passed in for "user functions" mode.

Static Private Attributes

• static SNLLOptimizer * snlOptInstance
  pointer to the active object instance used within the static evaluator functions in order to avoid the need for static data

Additional Inherited Members

14.242.1 Detailed Description

Wrapper class for the OPT++ optimization library.

The SNLLOptimizer class provides a wrapper for OPT++, a C++ optimization library of nonlinear programming and pattern search techniques from the Computational Sciences and Mathematics Research (CSMR) department at Sandia’s Livermore CA site. It uses a function pointer approach for which passed functions must be either global functions or static member functions. Any attribute used within static member functions must be either local to that function, a static member, or accessed by static pointer.

The user input mappings are as follows: max_iterations, max_function_evaluations, convergence_tolerance, max_step, gradient_tolerance, search_method, and search_scheme_size are
set using OPT++’s setMaxIter(), setMaxFeval(), setFcnTol(), setMaxStep(), setGradTol(), setSearchStrategy(), and setSSS() member functions, respectively; output verbosity is used to toggle OPT++’s debug mode using the setDebug() member function. Internal to OPT++, there are 3 search strategies, while the DAKOTA search method specification supports 4 (value_based_line_search, gradient_based_line_search, trust_region, or tr.pds). The difference stems from the "is_expensive" flag in OPT++. If the search strategy is LineSearch and "is_expensive" is turned on, then the value_based_line_search is used. Otherwise (the "is_expensive" default is off), the algorithm will use the gradient_based_line_search. Refer to [Meza, J.C., 1994] and to the OPT++ source in the Dakota/packages/OPTPP directory for information on OPT++ class member functions.

14.242.2 Constructor & Destructor Documentation

SNLLOptimizer ( ProblemDescDB & problem_db, Model & model )

standard constructor

This constructor is used for normal instantiations using data from the ProblemDescDB.

References Dakota::abort_handler(), Minimizer::boundConstraintFlag, SNLLBase::centeringParam, SNLLOptimizer::constraint0_evaluator(), SNLLOptimizer::constraint1_evaluator(), SNLLOptimizer::constraint2_evaluator(), SNLLOptimizer::default_instantiate_newton(), SNLLOptimizer::default_instantiate_q_newton(), Model::fd_gradient_step_size(), SNLLOptimizer::fdnlf1, SNLLOptimizer::fdnlf1Con, ProblemDescDB::get_int(), ProblemDescDB::get_real(), SNLLBase::init_fn(), Model::interval_type(), Iterator::iteratedModel, Dakota::LARGE_SCALE, Iterator::maxEvalConcurrency, Iterator::maxFunctionEvals, Iterator::maxIterations, SNLLOptimizer::maxStep, SNLLBase::meritFn, Iterator::method_enum_to_string(), Iterator::MethodName, SNLLOptimizer::nlf0, SNLLOptimizer::nlf0_evaluator(), SNLLOptimizer::nlf1, SNLLOptimizer::nlf1_evaluator(), SNLLOptimizer::nlf1Con, SNLLOptimizer::nlf2_evaluator(), SNLLOptimizer::nlf2_constraint, SNLLOptimizer::nlfObjective, SNLLOptimizer::nlpConstraint, Minimizer::numConstraints, Minimizer::numContinuousVars, Minimizer::numNonlinearConstraints, SNLLOptimizer::optbcfdnewton, SNLLOptimizer::optbcqnewton, SNLLOptimizer::optcg, SNLLOptimizer::optdfnewton, SNLLOptimizer::optdfnips, SNLLOptimizer::optlbfgs, SNLLOptimizer::optpds, SNLLOptimizer::optqnnewton, SNLLOptimizer::optqnips, Iterator::outputLevel, Iterator::probDescDB, SNLLOptimizer::searchStrat, SNLLBase::snll_post_instantiate(), SNLLBase::snll_pre_instantiate(), SNLLBase::stepLenToBndry, SNLLOptimizer::theOptimizer, and Minimizer::vendorNumericalGradFlag.

SNLLOptimizer ( const String & method_string, Model & model )

alternate constructor for instantiations "on the fly"

This is an alternate constructor for instantiations on the fly using a Model but no ProblemDescDB.

References Dakota::abort_handler(), Minimizer::boundConstraintFlag, SNLLOptimizer::constraint1_evaluator(), SNLLOptimizer::constraint2_evaluator(), Iterator::convergenceTol, SNLLOptimizer::default_instantiate_newton(), SNLLOptimizer::default_instantiate_q_newton(), Model::fd_gradient_step_size(), Model::interval_type(), Iterator::iteratedModel, Iterator::maxFunctionEvals, Iterator::maxIterations, Iterator::MethodName, SNLLOptimizer::nlf1_evaluator(), SNLLOptimizer::nlf2_evaluator(), SNLLOptimizer::nlfObjective, Minimizer::numConstraints, Minimizer::numContinuousVars, Iterator::outputLevel, SNLLOptimizer::snll_post_instantiate(), SNLLOptimizer::snll_pre_instantiate(), SNLLOptimizer::theOptimizer, and Minimizer::vendorNumericalGradFlag.
SNLLOptimizer ( const RealVector & initial pt, const RealVector & var lbnds, const RealVector & var ubnds, const RealMatrix & lin ineq coeff, const RealVector & lin ineq lbnds, const RealVector & lin ineq ubnds, const RealMatrix & lin eq coeff, const RealVector & lin eq tgt, const RealVector & nln ineq lbnds, const RealVector & nln ineq ubnds, const RealVector & nln eq tgt, void(*) (int mode, int n, const RealVector & x, double & f, RealVector & grad f, int & result mode) user obj eval, void(*) (int mode, int n, const RealVector & x, RealVector & g, RealMatrix & grad g, int & result mode) user con eval )

alternate constructor for instantiations "on the fly"

This is an alternate constructor for performing an optimization using the passed in objective function and constraint function pointers.

References Minimizer::bigRealBoundSize, Minimizer::boundConstraintFlag, SNLLOptimizer::default instantiate q newton(), SNLLOptimizer::initialPoint, SNLLOptimizer::lowerBounds, SNLLOptimizer::nlfObjective, SNLLOptimizer::nlpcConstraint, Minimizer::numConstraints, Minimizer::numContinuousVars, Iterator::outputLevel, SNLLOptimizer::snll initialize run(), SNLLOptimizer::snll post instantiate(), SNLLOptimizer::snll pre instantiate(), SNLLOptimizer::theOptimizer, and SNLLOptimizer::upperBounds.

SNLLOptimizer ( const RealVector & initial pt, const RealVector & var lbnds, const RealVector & var ubnds, const RealMatrix & lin ineq coeff, const RealVector & lin ineq lbnds, const RealVector & lin ineq ubnds, const RealMatrix & lin eq coeff, const RealVector & lin eq tgt, const RealVector & nln ineq lbnds, const RealVector & nln ineq ubnds, const RealVector & nln eq tgt, void(*)(int mode, int n, const RealVector & x, double & f, RealVector & grad f, int & result mode) user obj eval, void(*)(int mode, int n, const RealVector & x, RealVector & g, RealMatrix & grad g, int & result mode) user con eval, const int max iter, const int max fn evals, const Real conv tol, const Real grad tol, Real max step )

alternate constructor for instantiations "on the fly", also specifying different optimizer properties

This is an alternate constructor for performing an optimization using the passed in objective function and constraint function pointers additionally specifying optimization settings.

References Minimizer::bigRealBoundSize, Minimizer::boundConstraintFlag, SNLLOptimizer::default instantiate q newton(), SNLLOptimizer::initialPoint, SNLLOptimizer::lowerBounds, SNLLOptimizer::nlfObjective, SNLLOptimizer::nlpcConstraint, Minimizer::numConstraints, Minimizer::numContinuousVars, Iterator::outputLevel, SNLLOptimizer::snll initialize run(), SNLLOptimizer::snll post instantiate(), SNLLOptimizer::snll pre instantiate(), SNLLOptimizer::theOptimizer, and SNLLOptimizer::upperBounds.

### 14.242.3 Member Function Documentation

void nlf0 evaluator ( int n, const RealVector & x, double & f, int & result mode ) [static], [private]

objective function evaluator function for OPT++ methods which require only function values.

For use when DAKOTA computes f and gradients are not directly available. This is used by non-gradient-based optimizers such as PDS and by gradient-based optimizers in vendor numerical gradient mode (opt++’s internal finite difference routine is used).

References Model::continuous_variables(), Model::current_response(), Model::evaluate(), Response::function_value(), Iterator::iteratedModel, SNLLOptimizer::lastEvalVars, SNLLOptimizer::lastFnEvalLocn, Minimizer::numNonlinearConstraints, Iterator::outputLevel, Model::primary_response fn sense(), and SNLLOptimizer::snllOptInstance.

Referenced by SNLLOptimizer::SNLLOptimizer().

void nlf1 evaluator ( int mode, int n, const RealVector & x, double & f, RealVector & grad f, int & result mode ) [static], [private]

objective function evaluator function which provides function values and gradients to OPT++ methods.
For use when DAKOTA computes $f$ and $df/dX$ (regardless of gradient type). Vendor numerical gradient case is handled by `nlf0_evaluator`.

References `Iterator::activeSet`, `Model::continuous_variables()`, `Model::current_response()`, `Model::evaluate()`, `Response::function_gradient_copy()`, `Response::function_value()`, `Iterator::iteratedModel`, `SNLLBase::lastEvalMode`, `SNLLBase::lastEvalVars`, `SNLLBase::lastFnEvalLocn`, `Minimizer::numNonlinearConstraints`, `Iterator::outputLevel`, `Model::primary_response_fn_sense()`, `ActiveSet::request_values()`, and `SNLLOptimizer::snllOptInstance`.

Referenced by `SNLLOptimizer::SNLLOptimizer()`.

```cpp
void nlf2_evaluator ( int mode, int n, const RealVector & x, double & f, RealVector & grad_f, RealSymMatrix & hess_f, int & result_mode ) [static], [private]
```

objective function evaluator function which provides function values, gradients, and Hessians to OPT++ methods.

For use when DAKOTA receives $f$, $df/dX$, $d^2f/dx^2$ from the `ApplicationInterface` (analytic only). Finite differencing does not make sense for a full Newton approach, since lack of analytic gradients & Hessian should dictate the use of quasi-newton or fd-newton. Thus, there is no fdnlf2_evaluator for use with full Newton approaches, since it is preferable to use quasi-newton or fd-newton with nlf1. Gauss-Newton does not fit this model; it uses nlf2_evaluator GN instead of nlf2_evaluator.

References `Iterator::activeSet`, `Model::continuous_variables()`, `Model::current_response()`, `Model::evaluate()`, `Response::function_gradient_copy()`, `Response::function_hessian()`, `Response::function_value()`, `Iterator::iteratedModel`, `SNLLBase::lastEvalMode`, `SNLLBase::lastEvalVars`, `SNLLBase::lastFnEvalLocn`, `Minimizer::numNonlinearConstraints`, `Iterator::outputLevel`, `Model::primary_response_fn_sense()`, `ActiveSet::request_values()`, and `SNLLOptimizer::snllOptInstance`.

Referenced by `SNLLOptimizer::SNLLOptimizer()`.

```cpp
void constraint0_evaluator ( int n, const RealVector & x, RealVector & g, int & result_mode ) [static], [private]
```

constraint evaluator function for OPT++ methods which require only constraint values.

For use when DAKOTA computes $g$ and gradients are not directly available. This is used by nongradient-based optimizers and by gradient-based optimizers in vendor numerical gradient mode (opt++'s internal finite difference routine is used).

References `Model::continuous_variables()`, `SNLLBase::copy_con_vals_dak_to_optpp()`, `Model::current_response()`, `Model::evaluate()`, `Response::function_values()`, `Iterator::iteratedModel`, `SNLLBase::lastEvalVars`, `SNLLBase::lastFnEvalLocn`, `Optimizer::numObjectiveFns`, `Iterator::outputLevel`, and `SNLLOptimizer::snllOptInstance`.

Referenced by `SNLLOptimizer::SNLLOptimizer()`.

```cpp
void constraint1_evaluator ( int mode, int n, const RealVector & x, RealVector & g, RealMatrix & grad_g, int & result_mode ) [static], [private]
```

constraint evaluator function which provides constraint values and gradients to OPT++ methods.

For use when DAKOTA computes $g$ and $dg/dX$ (regardless of gradient type). Vendor numerical gradient case is handled by constraint0_evaluator.

References `Iterator::activeSet`, `Model::continuous_variables()`, `SNLLBase::copy_con_grad()`, `SNLLBase::copy_con_vals_dak_to_optpp()`, `Model::current_response()`, `Model::evaluate()`, `Response::function_gradients()`, `Response::function_values()`, `Iterator::iteratedModel`, `SNLLBase::lastEvalMode`, `SNLLBase::lastEvalVars`, `SNLLBase::lastFnEvalLocn`, `Optimizer::numObjectiveFns`, `Iterator::outputLevel`, `ActiveSet::request_values()`, and `SNLLOptimizer::snllOptInstance`.

Referenced by `SNLLOptimizer::SNLLOptimizer()`.
void constraint2.evaluator ( int mode, int n, const RealVector & x, RealVector & g, RealMatrix & grad g, OPTPP::OptppArray<RealSymMatrix> & hess g, int & result mode ) [static], [private]

constraint evaluator function which provides constraint values, gradients, and Hessians to OPT++ methods.

For use when DAKOTA computes g, dg/dX, & d^2g/dx^2 (analytic only).

References Iterator::activeSet, Model::continuous_variables(), SNLLBase::copy_con_grad(), SNLLBase::copy_con_hess(), SNLLBase::copy_vals_dak_to_optpp(), Model::current_response(), Model::evaluate(), Response::function_gradients(), Response::function_hessians(), Response::function_values(), Iterator::iteratedModel, SNLLBase::lastEvalMode, SNLLBase::lastEvalVars, SNLLBase::lastFnEvalLocn, Optimizer::numObjectiveFns, Iterator::outputLevel, ActiveSet::request_values(), and SNLLOptimizer::snllOptInstance.

Referenced by SNLLOptimizer::SNLLOptimizer().

The documentation for this class was generated from the following files:

- SNLLOptimizer.hpp
- SNLLOptimizer.cpp

### 14.243 SNLLTraits Class Reference

A version of TraitsBase specialized for SNLL optimizers.

Inheritance diagram for SNLLTraits:

```
  TraitsBase
     ↓
  SNLLTraits
```

**Public Member Functions**

- **SNLLTraits ()**
  
  *default constructor*

- **virtual ~SNLLTraits ()**
  
  *destructor*

- **virtual bool is_derived ()**
  
  A temporary query used in the refactor.

- **bool supports_continuous_variables ()**
  
  Return the value of supportsContinuousVariables.

- **bool supports_linear_equality ()**
  
  Return the flag indicating whether method supports linear equalities.

- **bool supports_linear_inequality ()**
  
  Return the flag indicating whether method supports linear inequalities.

- **bool supports_nonlinear_equality ()**
  
  Return the flag indicating whether method supports nonlinear equalities.

- **bool supports_nonlinear_inequality ()**
  
  Return the flag indicating whether method supports nonlinear inequalities.

- **NONLINEAR_INEQUALITY_FORMAT nonlinear_inequality_format ()**
  
  Return the format used for nonlinear inequality constraints.
14.243.1 Detailed Description

A version of TraitsBase specialized for SNLL optimizers.

The documentation for this class was generated from the following file:

- SNLLOptimizer.hpp

14.244 SOLBase Class Reference

Base class for Stanford SOL software.

Inheritance diagram for SOLBase:

```
SOLBase
   |   |
   v   v
NLSSOLLLeastSq
```

Public Member Functions

- **SOLBase ()**
  
  *default constructor*

- **SOLBase (Model &model)**

  *standard constructor*

- **~SOLBase ()**

  *destructor*

Protected Member Functions

- **void allocate_arrays (int num_cv, size_t num_nln_con, const RealMatrix &lin_ineq_coeffs, const RealMatrix &lin_eq_coeffs)**

  *Allocates miscellaneous arrays for the SOL algorithms.*

- **void deallocate_arrays ()**

  *Deallocates memory previously allocated by allocate_arrays().*

- **void allocate_workspace (int num_cv, int num_nln_con, int num_lin_con, int num_lsq)**

  *Allocates real and integer workspaces for the SOL algorithms.*

- **void set_options (bool speculative_flag, bool vendor_num_grad_flag, short output_lev, int verify_lev, Real fn_prec, Real linesrch_tol, int max_iter, Real constr_tol, Real conv_tol, const std::string &grad_type, const RealVector &fdss)**

  *Sets SOL method options using calls to npoptn2.*

- **void augment_bounds (RealVector &augmented_lower_bnds, RealVector &augmented_upper_bnds, const Model &model)**

  *Augments variable bounds with linear and nonlinear constraint bounds.*

- **void augment_bounds (RealVector &augmented_l_bnds, RealVector &augmented_u_bnds, const RealVector &lin_ineq_l_bnds, const RealVector &lin_ineq_u_bnds, const RealVector &lin_eq_targets, const RealVector &nln_ineq_l_bnds, const RealVector &nln_ineq_u_bnds, const RealVector &nln_eq_targets)**

  *Augments variable bounds with linear and nonlinear constraint bounds.*
**Static Protected Member Functions**

- static void `constraint_eval` (int &mode, int &ncnln, int &n, int &nrowj, int *needc, double *x, double *c, double *cjac, int &nstate)

  `CONFUN` in `NPSOL` manual: computes the values and first derivatives of the nonlinear constraint functions.

**Protected Attributes**

- int `realWorkSpaceSize`
  
  `size of realWorkSpace`

- int `intWorkSpaceSize`
  
  `size of intWorkSpace`

- RealArray `realWorkSpace`
  
  `real work space for NPSOL/NLSSOL`

- IntArray `intWorkSpace`
  
  `int work space for NPSOL/NLSSOL`

- int `nlnConstraintArraySize`
  
  `used for non-zero array sizing (nonlinear constraints)`

- IntArray `constraintState`
  
  `ISTATE` from `NPSOL` manual: constraint status.

- int `informResult`
  
  `INFORM` from `NPSOL` manual: optimization status on exit.

- int `numberIterations`
  
  `ITER` from `NPSOL` manual: number of (major) iterations performed.

- int `boundsArraySize`
  
  `length of augmented bounds arrays (variable bounds plus linear and nonlinear constraint bounds)`

- double * `linConstraintMatrixF77`
  
  `[A]` matrix from `NPSOL` manual: linear constraint coefficients

- double * `upperFactorHessianF77`
  

- double * `constraintJacMatrixF77`
  
  `[CJAC]` matrix from `NPSOL` manual: nonlinear constraint Jacobian

- int `fnEvalCntr`
  
  `counter for testing against maxFunctionEvals`

- size_t `constrOffset`
  
  `used in constraint_eval()` to bridge `NLSSOLLeastSq::numLeastSqTerms` and `NPSOLOptimizer::numObjectiveFns`
Static Protected Attributes

- static SOLBase * solInstance
  pointer to the active object instance used within the static evaluator functions in order to avoid the need for static data
- static Minimizer * optLSqInstance
  pointer to the active base class object instance used within the static evaluator functions in order to avoid the need for static data

14.244.1 Detailed Description

Base class for Stanford SOL software.

The SOLBase class provides a common base class for NPSOLOptimizer and NLSSOLLeastSq, both of which are Fortran 77 sequential quadratic programming algorithms from Stanford University marketed by Stanford Business Associates.

The documentation for this class was generated from the following files:

- SOLBase.hpp
- SOLBase.cpp

14.245 SoleilDirectApplicInterface Class Reference

Sample derived interface class for testing serial simulator plug-ins using assign_rep().

Inheritance diagram for SoleilDirectApplicInterface:

```
    Interface
       |        
       |        
    ApplicationInterface
       |        
       |        
    DirectApplicInterface
       |        
       |        
SoleilDirectApplicInterface
```

Public Member Functions

- SoleilDirectApplicInterface (const Dakota::ProblemDescDB &problem_db)
  constructor
- ~SoleilDirectApplicInterface ()
  destructor

Protected Member Functions

- int derived_map_ac (const Dakota::String &ac_name)
  execute an analysis code portion of a direct evaluation invocation
- void derived_map_asynch (const Dakota::ParamResponsePair &pair)
no-op hides base error; job batching occurs within `wait_local_evaluations()`

- **void** `wait_local_evaluations` (Dakota::PRPQueue &prp_queue)
  evaluate the batch of jobs contained in `prp_queue`
- **void** `test_local_evaluations` (Dakota::PRPQueue &prp_queue)
  invokes `wait_local_evaluations()` (no special nowait support)
- **void** `set_communicators_checks` (int max_eval_concurrency)
  no-op hides default run-time error checks at `DirectApplicInterface` level

**Private Member Functions**

- **int** `rosenbrock` (const Dakota::RealVector &c_vars, short asv, Dakota::Real &fn_val, Dakota::RealVector &fn_grad, Dakota::RealSymMatrix &fn_hess)
  *Rosenbrock plug-in test function.*

**Additional Inherited Members**

**14.245.1 Detailed Description**

Sample derived interface class for testing serial simulator plug-ins using `assign_rep()`.

The plug-in `SoleilDirectApplicInterface` resides in namespace `SIM` and uses a copy of `rosenbrock()` to perform serial parameter to response mappings. It is used to demonstrate plugging in a serial direct analysis driver into `Dakota` in library mode. Test input files can then use an analysis driver of "plugin_rosenbrock".

**14.245.2 Member Function Documentation**

**int** `derived_map_ac` (const Dakota::String &ac_name) [protected], [virtual]

execute an analysis code portion of a direct evaluation invocation

Redefine this for serial/blocking execution of single Soleil simulations.

Reimplemented from `DirectApplicInterface`.

References Dakota::abort_handler(), ApplicationInterface::analysisServerId, DirectApplicInterface::directFn-ASV, DirectApplicInterface::fnGrads, DirectApplicInterface::fnHessians, DirectApplicInterface::fnVals, Application-Interface::multiProcAnalysisFlag, SoleilDirectApplicInterface::rosenbrock(), and DirectApplicInterface::xC.

**void** `wait_local_evaluations` (Dakota::PRPQueue & prp_queue) [protected]

evaluate the batch of jobs contained in `prp_queue`

Redefine this for (Legion-based) execution of a batch of Soleil simulations. The incoming `prp_queue` is defined from ApplicationInterface::asynchLocalActivePRPQueue which is a local subset of beforeSynchCorePRPQueue. This function must complete at least one job (whereas `test_local_evaluations()` may complete zero). Populating completionSet results in decrementing the active queue and backfilling as indicated by concurrency level. For Soleil, we should not limit the concurrency level and will not combine with MPI scheduling — incoming `prp_queue` is the full beforeSynchCorePRPQueue (no MPI distribution + no throttling). Further, we should complete the full local queue or we may need to distinguish still-running jobs from incoming new ones.

References Dakota::abort_handler(), ApplicationInterface::completionSet, Variables::continuous_variables(), Response::function_gradient_view(), Response::function_hessian_view(), Response::function_value_view(), Application-Interface::multiProcAnalysisFlag, Interface::outputLevel, ActiveSet::request_vector(), and SoleilDirectApplicInterface-::rosenbrock().

Referenced by SoleilDirectApplicInterface::test_local_evaluations().
void test\_local\_evaluations ( Dakota::PRPQueue & prp\_queue ) [inline],[protected]

invokes wait\_local\_evaluations() (no special nowait support)

For use by ApplicationInterface::serve\_evaluations\_asynch(), which can provide a batch processing capability within message passing schedulers (called using chain IteratorScheduler::run\_iterator() \rightarrow Model::serve() \rightarrow ApplicationInterface::serve\_evaluations\_asynch() \rightarrow ApplicationInterface::serve\_evaluations\_asynch()

References SoleilDirectApplicInterface::wait\_local\_evaluations().
The documentation for this class was generated from the following files:

- SoleilDirectApplicInterface.hpp
- SoleilDirectApplicInterface.cpp

### 14.246 SpawnApplicInterface Class Reference

Derived application interface class which spawns simulation codes using spawnvp.

Inheritance diagram for SpawnApplicInterface:

```
  Interface
  \_______________
  |               |
  | ApplicationInterface |
  |                   |
  |                   |
  | ProcessApplicInterface |
  |                   |
  |                   |
  | ProcessHandleApplicInterface |
  \_______________
  |               |
  | SpawnApplicInterface |
```

**Public Member Functions**

- **SpawnApplicInterface** (const ProblemDescDB &problem\_db)
  
  constructor

- **~SpawnApplicInterface** ()
  
  destructor

**Protected Member Functions**

- void **wait\_local\_evaluation\_sequence** (PRPQueue &prp\_queue)
  
  version of wait\_local\_evaluations() managing of set of individual asynchronous evaluations

- void **test\_local\_evaluation\_sequence** (PRPQueue &prp\_queue)
  
  version of test\_local\_evaluations() managing of set of individual asynchronous evaluations

- pid\_t **create\_analysis\_process** (bool block\_flag, bool new\_group)
  
  spawn a child process for an analysis component within an evaluation

- size\_t **wait\_local\_analyses** ()
  
  wait for asynchronous analyses on the local processor, completing at least one job
Additional Inherited Members

14.246.1 Detailed Description

Derived application interface class which spawns simulation codes using spawnvp.

SpawnApplic Interface is used on Windows systems and is a peer to ForkApplic Interface for Unix systems.
The documentation for this class was generated from the following files:

• SpawnApplicInterface.hpp
• SpawnApplicInterface.cpp

14.247 StandardizationScaler Class Reference

Standardizes the data so the each feature has zero mean and unit variance.

Inheritance diagram for StandardizationScaler:

```
  DataScaler
   
  StandardizationScaler
```

Public Member Functions

• StandardizationScaler (const MatrixXd &features, double norm_factor=1.0)
  
  Main constructor for StandardizationScaler.

Additional Inherited Members

14.247.1 Detailed Description

Standardizes the data so the each feature has zero mean and unit variance.

scaler_offsets = mean
scale_factors = standard_deviation/norm_factor

14.247.2 Constructor & Destructor Documentation

StandardizationScaler ( const MatrixXd & features, double norm_factor = 1.0 )

Main constructor for StandardizationScaler.
Parameters

<table>
<thead>
<tr>
<th></th>
<th>features</th>
<th>Unscaled data matrix - (num_samples by num_features)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>norm_factor</td>
<td>Optional scaling factor applied to each feature Has a default value of 1.0</td>
</tr>
</tbody>
</table>

References DataScaler::check_for_zero_scaler_factor(), DataScaler::hasScaling, DataScaler::scaledFeatures, Data-Scaler::scalerFeaturesOffsets, and DataScaler::scalerFeaturesScaleFactors.

The documentation for this class was generated from the following files:

- DataScaler.hpp
- DataScaler.cpp

14.248 StringScale Struct Reference

Data structure for storing string-valued dimension scale.

Public Member Functions

- **StringScale** (const std::string &in_label, const char *const in_items[], const int &len, ScaleScope inScope=ScaleScope::UNSHARED)
  
  Constructor that takes a C-style array of C-strings.

- **StringScale** (const std::string &in_label, std::initializer_list< const char * > in_items, ScaleScope inScope=ScaleScope::UNSHARED)
  
  Constructor that takes and initializer list of string literals.

- **StringScale** (const std::string &in_label, const std::vector<String> &in_items, ScaleScope in_scope=ScaleScope::UNSHARED)
  
  Constructor that takes a vector of strings.

- **StringScale** (const std::string &in_label, const std::vector< String > &in_items, ScaleScope in_scope=ScaleScope::UNSHARED)
  
  Constructor that takes a vector of C-style strings.

- **StringScale** (const std::string &in_label, const StringMultiArrayConstView in_items, ScaleScope in_scope=ScaleScope::UNSHARED)
  
  Constructor that takes a StringMultiArrayConstView.

- **StringScale** (const std::string &in_label, const std::vector< String > &in_items, const size_t first, const size_t num, ScaleScope in_scope=ScaleScope::UNSHARED)
  
  Constructor that takes indexes into a StringArray.

- **StringScale** (const std::string &in_label, std::vector< std::vector< const char * > > in_items, ScaleScope in_scope=ScaleScope::UNSHARED)
  
  Constructor that takes a vector<vector<const char *>> to produce a 2D scale.

Public Attributes

- std::string label
  
  Scale label.

- ScaleScope scope
  
  Scale scope (whether the scaled is shared among responses)

- std::vector< const char * > items
Pointers to the strings that make up the scale.

- **int numCols**
  
  *Number of columns; equals length of scale when 1D.*

- **bool isMatrix**
  
  *2d or 1d?*

### 14.248.1 Detailed Description

Data structure for storing string-valued dimension scale.

The documentation for this struct was generated from the following file:

- dakota_results_types.hpp

### 14.249 SurfpackApproximation Class Reference

Derived approximation class for Surfpack approximation classes. **Interface** between Surfpack and Dakota.

Inheritance diagram for SurfpackApproximation:

```
Approximation

SurfpackApproximation
```

### Public Member Functions

- **SurfpackApproximation ()**
  
  *default constructor*

- **SurfpackApproximation (const ProblemDescDB &problem_db, const SharedApproxData &shared_data, const String &approx_label)**
  
  *standard constructor: Surfpack surface of appropriate type will be created*

- **SurfpackApproximation (const SharedApproxData &shared_data)**
  
  *alternate constructor*

- **~SurfpackApproximation ()**
  
  *destructor*

### Protected Member Functions

- **int min_coefficients () const override**
  
  *return the minimum number of samples (unknowns) required to build the derived class approximation type in num-Vars dimensions*

- **int recommended_coefficients () const override**
  
  *return the recommended number of samples (unknowns) required to build the derived class approximation type in numVars dimensions*

- **void build () override**
SurfData object will be created from Dakota’s SurrogateData, and the appropriate Surfpack build method will be invoked.

- `void export_model` (const String &fn_label, const String &export_prefix="", const unsigned short export_format=NO_MODEL_FORMAT) override
  
  `export the Surpack model to disk or console`

- `Real value` (const Variables &vars) override
  
  `Return the value of the Surfpack surface for a given parameter vector x.`

- `const RealVector & gradient` (const Variables &vars) override
  
  `retrieve the approximate function gradient for a given parameter vector x`

- `const RealSymMatrix & hessian` (const Variables &vars) override
  
  `retrieve the approximate function Hessian for a given parameter vector x`

- `Real prediction_variance` (const Variables &vars) override
  
  `retrieve the variance of the predicted value for a given parameter set x (KrigingModel only)`

- `bool diagnostics_available` () override
  
  `check if the diagnostics are available (true for the Surfpack types)`

- `Real diagnostic` (const String &metric_type) override
  
  `retrieve a single diagnostic metric for the diagnostic type specified on the primary model and data`

- `Real diagnostic` (const String &metric_type, const SurfpackModel &model, const SurfData &data)
  
  `retrieve a single diagnostic metric for the diagnostic type specified on the given model and data - not inherited`

- `void primary_diagnostics` (int fn_index) override
  
  `compute and print all requested diagnostics and cross-validation`

- `void challenge_diagnostics` (int fn_index, const RealMatrix &challenge_points, const RealVector &challenge_responses) override
  
  `compute and print all requested diagnostics for user provided challenge pts`

- `RealArray cv_diagnostic` (const StringArray &metric_types, unsigned num_folds) override
  
  `compute and return cross-validation for metric_type with num_folds`

- `RealArray challenge_diagnostic` (const StringArray &metric_types, const RealMatrix &challenge_points, const RealVector &challenge_responses) override
  
  `compute and print all requested diagnostics for user provided challenge pts`

**Private Member Functions**

- `SurfData * surrogates_to_surf_data` ()
  
  `copy from SurrogateData to SurfPoint/SurfData`

- `void add_constraints_to_surfdata` (const Pecos::SurrogateDataVars &anchor_vars, const Pecos::SurrogateDataResp &anchor_res, short fail_code, SurfData &surf_data)
  
  `set the anchor point (including gradient and hessian if present) into surf_data`
14.249. SURFPACK APPROXIMATION CLASS REFERENCE

Private Attributes

- SurfpackModel * model
  The native Surfpack approximation.
- SurfpackModelFactory * factory
  factory for the SurfpackModel instance
- SurfData * surfData
  The data used to build the approximation, in Surfpack format.

Additional Inherited Members

14.249.1 Detailed Description

Derived approximation class for Surfpack approximation classes. Interface between Surfpack and Dakota.

The SurfpackApproximation class is the interface between Dakota and Surfpack. Based on the information in
the ProblemDescDB that is passed in through the constructor, SurfpackApproximation builds a Surfpack Surface
object that corresponds to one of the following data-fitting techniques: polynomial regression, kriging, artificial
neural networks, radial basis function network, or multivariate adaptive regression splines (MARS).

14.249.2 Constructor & Destructor Documentation

SurfpackApproximation ( const ProblemDescDB & problem_db, const SharedApproxData & shared_data, const String & approx_label )

standard constructor: Surfpack surface of appropriate type will be created

- Initialize the embedded Surfpack surface object and configure it using the specifications from the input file.
- Data for the surface is created later.
- References Dakota::abort_handler(), SharedSurfpackApproxData::approxOrder, SharedApproxData::approxType, SharedApproxData::buildDataOrder, Dakota::copy_data(), SharedSurfpackApproxData::crossValidateFlag, SharedSurfpackApproxData::diagnosticSet, SurfpackApproximation::factory, ProblemDescDB::get_real(), ProblemDescDB::get_rv(), ProblemDescDB::get_short(), ProblemDescDB::get_string(), SharedSurfpackApproxData::numFolds, SharedApproxData::numVars, SharedApproxData::outputLevel, SharedSurfpackApproxData::percentFold, and Approximation::sharedDataRep.

SurfpackApproximation ( const SharedApproxData & shared_data )

alternate constructor

- On-the-fly constructor which uses mostly Surfpack model defaults.
- References Dakota::abort_handler(), SharedApproxData::approxOrder, SharedApproxData::approxType, SharedApproxData::buildDataOrder, SurfpackApproximation::factory, SharedApproxData::numVars, SharedApproxData::outputLevel, and Approximation::sharedDataRep.

14.249.3 Member Function Documentation

void build() [override], [protected], [virtual]

SurfData object will be created from Dakota’s SurrogateData, and the appropriate Surfpack build method will be invoked.

- surfData will be deleted in dtor
Todo Right now, we’re completely deleting the old data and then recopying the current data into a SurfData object. This was just the easiest way to arrive at a solution that would build and run. This function is frequently called from addPoint rebuild, however, and it’s not good to go through this whole process every time one more data point is added.

Reimplemented from Approximation.
References Dakota::abort_handler(), SharedApproxData::approxCLowerBnds, SharedApproxData::approxCUpperBnds, SharedApproxData::approxDLowerBnds, SharedApproxData::approxDUpperBnds, SharedApproxData::approxDRLowerBnds, SharedApproxData::approxDRUpperBnds, Approximation::build(), SurfpackApproximation::factory, SharedSurfpackApproxData::merge_variable_arrays(), SurfpackApproximation::model, Approximation::sharedDataRep, SurfpackApproximation::surfData, and SurfpackApproximation::surrogates_to_surf_data().

const RealSymMatrix & hessian ( const Variables & vars ) [override], [protected], [virtual]
retrieve the approximate function Hessian for a given parameter vector x

Todo Make this acceptably efficient

Reimplemented from Approximation.
References Dakota::abort_handler(), Approximation::approxHessian, SharedApproxData::approxType, Variables::cv(), SurfpackApproximation::model, and Approximation::sharedDataRep.
Referenced by SurfpackApproximation::add_constraints_to_surfdata().

const RealSymMatrix & hessian ( const RealVector & c_vars ) [override], [protected], [virtual]
retrieve the approximate function Hessian for a given parameter vector x

Todo Make this acceptably efficient

Reimplemented from Approximation.
References Dakota::abort_handler(), Approximation::approxHessian, SharedApproxData::approxType, SurfpackApproximation::model, and Approximation::sharedDataRep.

SurfData * surrogates_to_surf_data ( ) [private]
copy from SurrogateData to SurfPoint/SurfData

Copy the data stored in Dakota-style SurrogateData into Surfpack-style SurfPoint and SurfData objects.
References Dakota::NPOS, SurfpackApproximation::add_constraints_to_surfdata(), SharedSurfpackApproxData::add_sd_to_surfdata(), Approximation::approxData, SharedApproxData::buildDataOrder, SurfpackApproximation::factory, SharedApproxData::outputLevel, and Approximation::sharedDataRep.
Referenced by SurfpackApproximation::build().

void add_constraints_to_surfdata ( const Pecos::SurrogateDataVars & anchor_vars, const Pecos::SurrogateDataResp & anchor_resp, short fail_code, SurfData & surf_data ) [private]
set the anchor point (including gradient and hessian if present) into surf_data

If there is an anchor point, add an equality constraint for its response value. Also add constraints for gradient and hessian, if applicable.
References Dakota::abort_handler(), Dakota::copy_data(), SharedSurfpackApproxData::copy_matrix(), SurfpackApproximation::gradient(), SurfpackApproximation::hessian(), SharedApproxData::outputLevel, SharedSurfpackApproxData::sdv_to_realarray(), and Approximation::sharedDataRep.
14.250  SurrBasedGlobalMinimizer Class Reference

Inheritance diagram for SurrBasedGlobalMinimizer:

```
  SurrBasedGlobalMinimizer  
  ^                       
  |                       
  |                       
  |                       
  V                       
Minimizer                
  ^                       
  |                       
  |                       
  |                       
  V                       
SurrBasedMinimizer       
  ^                       
  |                       
  |                       
  |                       
  V                       
Iterator                  
```

Public Member Functions

- **SurrBasedGlobalMinimizer** (ProblemDescDB &problem db, Model &model)
  
  Constructor

- **~SurrBasedGlobalMinimizer**()
  
  Destructor

Protected Member Functions

- **void core_run ()**
  
  Performs global surrogate-based optimization by repeatedly optimizing on and improving surrogates of the response functions.

- **bool returns_multiple_points () const**
  
  Global surrogate-based methods can return multiple points.

Private Attributes

- **bool replacePoints**
  
  Flag for replacing the previous iteration’s point additions, rather than continuing to append, during construction of the next surrogate.

Additional Inherited Members

14.250.1  Detailed Description

This method uses a SurrogateModel to perform minimization (optimization or nonlinear least squares) through a set of iterations. At each iteration, a surrogate is built, the surrogate is minimized, and the optimal points from
the surrogate are then evaluated with the "true" function, to generate new points upon which the surrogate for the next iteration is built.

The documentation for this class was generated from the following files:

- SurrBasedGlobalMinimizer.hpp
- SurrBasedGlobalMinimizer.cpp

### 14.251 SurrBasedGlobalTraits Class Reference

The global surrogate-based minimizer which sequentially minimizes and updates a global surrogate model without trust region controls.

Inheritance diagram for SurrBasedGlobalTraits:

```
TraitsBase
```

```
SurrBasedGlobalTraits
```

#### Public Member Functions

- **SurrBasedGlobalTraits ()**
  - default constructor
- **virtual ~SurrBasedGlobalTraits ()**
  - destructor
- **virtual bool is_derived ()**
  - A temporary query used in the refactor.
- **bool supports_continuous_variables ()**
  - Return the flag indicating whether method supports continuous variables.
- **bool supports_discrete_variables ()**
  - Return the flag indicating whether method supports discrete variables.
- **bool supports_linear_equality ()**
  - Return the flag indicating whether method supports linear equalities.
- **bool supports_linear_inequality ()**
  - Return the flag indicating whether method supports linear inequalities.
- **bool supports_nonlinear_equality ()**
  - Return the flag indicating whether method supports nonlinear equalities.
- **bool supports_nonlinear_inequality ()**
  - Return the flag indicating whether method supports nonlinear inequalities.

#### 14.251.1 Detailed Description

The global surrogate-based minimizer which sequentially minimizes and updates a global surrogate model without trust region controls.

A version of TraitsBase specialized for surrogate-based global minimizer

The documentation for this class was generated from the following file:

- SurrBasedGlobalMinimizer.hpp
14.252 SurrBasedLocalMinimizer Class Reference

Class for provably-convergent local surrogate-based optimization and nonlinear least squares.

Inheritance diagram for SurrBasedLocalMinimizer:

```
SurrBasedLocalMinimizer
  ↓
SurrBasedMinimizer
  ↓
Minimizer
  ↓
Iterator
```

Public Member Functions

- **SurrBasedLocalMinimizer** (ProblemDescDB &problem_db, Model &model, std::shared_ptr<TraitsBase> > traits)
  
  * constructor

- **~SurrBasedLocalMinimizer** ()
  
  * destructor

Protected Member Functions

- **void pre_run ()**
  
  * pre-run portion of run (optional); re-implemented by Iterators which can generate all Variables (parameter sets) a priori

- **void core_run ()**
  
  * Performs local surrogate-based minimization by minimizing local, global, or hierarchical surrogates over a series of trust regions.

- **void post_run (std::ostream &s)**
  
  * post-run portion of run (optional); verbose to print results; re-implemented by Iterators that can read all Variables/Responses and perform final analysis phase in a standalone way

- **void reset ()**
  
  * restore initial state for repeated sub-iterator executions

- **virtual SurrBasedLevelData & trust_region ()=0**
  
  * return the active SurrBasedLevelData instance

- **virtual void update_trust_region ()=0**
  
  * update the trust region bounds, strictly contained within global bounds

- **virtual void build ()=0**
  
  * build the approximation over the current trust region

- **virtual void minimize ()**
  
  * solve the approximate subproblem

- **virtual void verify ()=0**
  
  *
verify the approximate iterate and update the trust region for the next approximate optimization cycle

- virtual unsigned short converged ()=0
  return the convergence code for the truth level of the trust region hierarchy
- void initialize_sub_model ()
  construct and initialize approxSubProbModel
- void initialize_sub_minimizer ()
  construct and initialize approxSubProbMinimizer
- void initialize_multipliers ()
  initialize lagrangeMult and augLagrangeMult
- void reset_penalties ()
  reset all penalty parameters to their initial values
- void reset_multipliers ()
  reset Lagrange multipliers to initial values for cases where they are accumulated instead of computed directly
- void update_trust_region_data (SurrBasedLevelData &tr_data, const RealVector &parent_l_bnds, const RealVector &parent_u_bnds)
  update the trust region bounds, strictly contained within global bounds
- void update_approx_sub_problem (SurrBasedLevelData &tr_data)
  update variables and bounds within approxSubProbModel
- void compute_trust_region_ratio (SurrBasedLevelData &tr_data, bool check_interior=false)
  compute trust region ratio (for SBLM iterate acceptance and trust region resizing) and check for soft convergence (diminishing returns)
- void hard_convergence_check (SurrBasedLevelData &tr_data, const RealVector &lower_bnds, const RealVector &upper_bnds)
  check for hard convergence (norm of projected gradient of merit function < tolerance)
- void print_convergence_code (std::ostream &s, unsigned short code)
  print out the state corresponding to the code returned by converged()
- void update_penalty (const RealVector &fns_center_truth, const RealVector &fns_star_truth)
  initialize and update the penaltyParameter
- bool find_approx_response (const Variables &search_vars, Response &search Resp)
  locate an approximate response with the data_pairs cache
- bool find_truth_response (const Variables &search_vars, Response &search Resp)
  locate a truth response with the data_pairs cache
- bool find_response (const Variables &search_vars, Response &search Resp, const String &search_id, short set_request)
  locate a response with the data_pairs cache
- void relax_constraints (SurrBasedLevelData &tr_data)
  relax constraints by updating bounds when current iterate is infeasible

Static Protected Member Functions

- static void approx_subprob_objective_eval (const Variables &surrogate_vars, const Variables &recast_vars, const Response &surrogate_response, Response &recast_response)
  static function used to define the approximate subproblem objective.
- static void approx_subprob_constraint_eval (const Variables &surrogate_vars, const Variables &recast_vars, const Response &surrogate_response, Response &recast_response)
static function used to define the approximate subproblem constraints.

- static void hom_objective_eval (int &mode, int &n, double *tau_and_x, double &f, double *grad_f, int &)
  static function used by NPSOL as the objective function in the homotopy constraint relaxation formulation.

- static void hom_constraint_eval (int &mode, int &ncnln, int &n, int &nrowj, int *needc, double *tau_and_x, double *c, double *cjac, int &nstate)
  static function used by NPSOL as the constraint function in the homotopy constraint relaxation formulation.

Protected Attributes

- Model approxSubProbModel
  the approximate sub-problem formulation solved on each approximate minimization cycle: may be a shallow copy of iteratedModel, or may involve a RecastModel recursion applied to iteratedModel

- short approxSubProbObj
  type of approximate subproblem objective: ORIGINAL_OBJ, LAGRANGIAN_OBJ, or AUGMENTED_LAGRANGIAN_OBJ

- short approxSubProbCon
  type of approximate subproblem constraints: NO_CON, LINEARIZED_CON, or ORIGINAL_CON

- bool recastSubProb
  flag to indicate when approxSubProbModel involves a RecastModel recursion

- short meritFnType
  type of merit function used in trust region ratio logic: PENALTY_MERIT, ADAPTIVE_PENALTY_MERIT, LAGRANGIAN_MERIT, or AUGMENTED_LAGRANGIAN_MERIT

- short acceptLogic
  type of iterate acceptance test logic: FILTER or TR_RATIO

- short trConstraintRelax
  type of trust region constraint relaxation for infeasible starting points: NO_RELAX or HOMOTOPY

- int minimizeCycles
  counter for number of minimization cycles that have accumulated prior to convergence at the minimizeIndex level (used for ramping penalties)

- int penaltyIterOffset
  iteration offset used to update the scaling of the penalty parameter for adaptive_penalty merit functions

- RealVector origTrustRegionFactor
  original user specification for trust region initial size

- Real minTrustRegionFactor
  a soft convergence control: stop SBLM when the trust region factor is reduced below the value of minTrustRegionFactor

- Real trRatioContractValue
  trust region ratio min value: contract tr if ratio below this value

- Real trRatioExpandValue
  trust region ratio sufficient value: expand tr if ratio above this value

- Real gammaContract
  trust region contraction factor

- Real gammaExpand
  trust region expansion factor

- unsigned short softConvLimit
convergence control limiting the number of consecutive iterations that fail to achieve sufficient decrease. If exceeded by softConvCount, stop.

- short *truthSetRequest*
  derivative order of truth data used within the SBLM process

- short *approxSetRequest*
  derivative order of surrogate data used within the SBLM process

- short *correctionType*
  flags the use of surrogate correction techniques at the center of each trust region

- RealVector *initialPoint*
  starting point prior to sequence of SBLM iterations

- RealVector *globalLowerBnds*
  Global lower bounds.

- RealVector *globalUpperBnds*
  Global Upper bounds.

- RealVector *nonlinIneqLowerBndsSlack*
  individual violations of nonlinear inequality constraint lower bounds

- RealVector *nonlinIneqUpperBndsSlack*
  individual violations of nonlinear inequality constraint upper bounds

- RealVector *nonlinEqTargetsSlack*
  individual violations of nonlinear equality constraint targets

- Real *tau*
  constraint relaxation parameter

- Real *alpha*
  constraint relaxation parameter backoff parameter (multiplier)

**Static Protected Attributes**

- static *SurrBasedLocalMinimizer * sblmInstance*
  pointer to SBLM instance used in static member functions

**Additional Inherited Members**

**14.252.1 Detailed Description**

Class for provably-convergent local surrogate-based optimization and nonlinear least squares.

This minimizer uses a SurrogateModel to perform minimization based on local, global, or hierarchical surrogates. It achieves provable convergence through the use of a sequence of trust regions and the application of surrogate corrections at the trust region centers.

**14.252.2 Member Function Documentation**

```cpp
void pre_run ( ) [protected], [virtual]
```

pre-run portion of run (optional); re-implemented by Iterators which can generate all Variables (parameter sets) a priori
pre-run phase, which a derived iterator may optionally reimplement; when not present, pre-run is likely inte-
gerated into the derived run function. This is a virtual function; when re-implementing, a derived class must call
its nearest parent’s pre_run(), if implemented, typically before performing its own implementation steps.
Reimplemented from Iterator.
References Model::continuous_lower_bounds(), Model::continuous_upper_bounds(), Model::continuous_variables(),
SurrBasedLocalMinimizer::converged(), Dakao::copy_data(), SurrBasedLocalMinimizer::globalLowerBnds, Surr-
BasedLocalMinimizer::globalUpperBnds, SurrBasedLocalMinimizer::initialPoint, Iterator::iteratedModel, and Surr-
BasedLocalMinimizer::reset().

void core_run ( ) [protected], [virtual]
Performs local surrogate-based minimization by minimizing local, global, or hierarchical surrogates over a series
of trust regions.
Trust region-based strategy to perform surrogate-based optimization in subregions (trust regions) of the param-
eter space. The minimizer operates on approximations in lieu of the more expensive simulation-based response
functions. The size of the trust region is adapted according to the agreement between the approximations and the
ture response functions.
Reimplemented from Iterator.
References SurrBasedLocalMinimizer::build(), SurrBasedLocalMinimizer::converged(), SurrBasedLocalMinimizer-
::minimize(), SurrBasedLocalMinimizer::sblmInstance, SurrBasedLocalMinimizer::update_trust_region(), and Surr-
BasedLocalMinimizer::verify().

void post_run ( std::ostream & s ) [protected], [virtual]
pic-run portion of run (optional); verbose to print results; re-implemented by Iterators that can read all Variables/
Responses and perform final analysis phase in a standalone way
Post-run phase, which a derived iterator may optionally reimplement; when not present, post-run is likely
integrated into run. This is a virtual function; when re-implementing, a derived class must call its nearest parent’s
post_run(), typically after performing its own implementation steps.
Reimplemented from Minimizer.
References SurrBasedLocalMinimizer::converged(), SurrBasedMinimizer::globalIterCount, Minimizer::post-
run(), and SurrBasedLocalMinimizer::print_convergence_code().

void compute_trust_region_ratio ( SurrBasedLevelData & tr_data, bool check_interior = false ) [protected]
compute trust region ratio (for SBLM iterate acceptance and trust region resizing) and check for soft convergence
diminishing returns)
Assess acceptance of SBLM iterate (trust region ratio or filter) and compute soft convergence metrics (number
of consecutive failures, min trust region size, etc.) to assess whether the convergence rate has decreased to a point
where the process should be terminated (diminishing returns).
References SurrBasedLocalMinimizer::acceptLogic, SurrBasedLocalMinimizer::approxSubProbObj, SurrBased-
Minimizer::augmented_lagrangian_merit(), SurrBasedMinimizer::constraintViolation(), Minimizer::constraintTol,
Iterator::convergenceTol, SurrBasedMinimizer::etaSequence, Response::function_values(), SurrBasedLocalMinimizer-
::gammaContract, SurrBasedLocalMinimizer::gammaExpand, SurrBasedLocalMinimizer::globalLowerBnds, Surr-
BasedLocalMinimizer::globalUpperBnds, SurrBasedMinimizer::initialize_filter(), Response::is_null(), Iterator::iterated-
Model, SurrBasedMinimizer::lagrangian_merit(), SurrBasedLocalMinimizer::meritFnType, Minimizer::numContinuous-
Vars, Minimizer::numNonlinearConstraints, Minimizer::objective(), SurrBasedMinimizer::origNonlinEqTargets,
void hard_convergence_check ( SurrBasedLevelData & tr_data, const RealVector & lower_bnds, const RealVector & upper_bnds ) [protected]  
check for hard convergence (norm of projected gradient of merit function < tolerance)  
The hard convergence check computes the gradient of the merit function at the trust region center, performs a projection for active bound constraints (removing any gradient component directed into an active bound), and signals convergence if the 2-norm of this projected gradient is less than convergenceTol.  
References SurrBasedLocalMinimizer::approxSubProbObj, SurrBasedMinimizer::constraint_violation(), Minimizer::constraintTol, Iterator::convergenceTol, Response::function_gradients(), Response::function_values(), Iterator::iteratedModel, SurrBasedMinimizer::lagrangian_gradient(), SurrBasedLocaMinimizer::minimizeCycles, Minimizer::numContinuousVars, Minimizer::numNonlinearConstraints, SurrBasedMinimizer::origNonlinEqTargets, SurrBasedMinimizer::origNonlinIneqLowerBnds, SurrBasedMinimizer::origNonlinIneqUpperBnds, Iterator::outputLevel, Model::primary_response_fn_sense(), and Model::primary_response_fn_weights().

void update_penalty ( const RealVector & fns_center_truth, const RealVector & fns_star_truth ) [protected]  
initialize and update the penaltyParameter  
Scaling of the penalty value is important to avoid rejecting SBLM iterates which must increase the objective to achieve a reduction in constraint violation. In the basic penalty case, the penalty is ramped exponentially based on the iteration counter. In the adaptive case, the ratio of relative change between center and star points for the objective and constraint violation values is used to rescale penalty values.

References SurrBasedMinimizer::alphaEta, SurrBasedLocalMinimizer::approxSubProbObj, SurrBasedMinimizer::constraint_violation(), Minimizer::constraintTol, Iterator::constrainedModel, SurrBasedMinimizer::eta, SurrBasedMinimizer::etaSequence, Iterator::iteratedModel, SurrBasedLocalMinimizer::meritFnType, SurrBasedLocalMinimizer::minimizeCycles, Minimizer::objective(), Iterator::outputLevel, SurrBasedLocalMinimizer::penaltyIterOffset, SurrBasedMinimizer::penaltyParameter, Model::primary_response_fn_sense(), and Model::primary_response_fn_weights().

Referenced by SurrBasedLocalMinimizer::compute_trust_region_ratio().

void approx_subprob_objective_eval ( const Variables & surrogate_vars, const Variables & recast_vars, const Response & surrogate_response, Response & recast_response ) [static], [protected]  
static function used to define the approximate subproblem objective.

Objective functions evaluator for solution of approximate subproblem using a RecastModel.

References Response::active_set_request_vector(), SurrBasedLocalMinimizer::approxSubProbCon, SurrBasedLocalMinimizer::approxSubProbModel, SurrBasedLocalMinimizer::approxSubProbObj, SurrBasedMinimizer::augmented_lagrangian_gradient(), SurrBasedMinimizer::augmented_lagrangian_hessian(), SurrBasedMinimizer::augmented_lagrangian_merit(), Response::function_gradient(), Response::function_gradient_view(), Response::function_gradients(), Response::function_hessian(), Response::function_hessians(), Response::function_values(), Response::function_values(), Iterator::iteratedModel, SurrBasedMinimizer::lagrangian_gradient(), SurrBasedMinimizer::lagrangian_hessian(), SurrBasedMinimizer::lagrangian_merit(), Model::nonlinear_eq_constraint_targets(), Model::nonlinear_ineq_constraint_lower_bounds(), Model::nonlinear_ineq_constraint_upper_bounds(), Minimizer::numUserPrimary-
Fns, Minimizer::objective(), Minimizer::objective_gradient(), Minimizer::objective_hessian(), SurrBasedMinimizer::origNonlinEqTargets, SurrBasedMinimizer::origNonlinIneqLowerBnds, SurrBasedMinimizer::origNonlinIneqUpperBnds, Model::primary_response_fn_sense(), Model::primary_response_fn_weights(), and SurrBasedLocalMinimizer::sblmInstance.

Referenced by SurrBasedLocalMinimizer::initialize_sub_model().

void approx_subprob_constraint_eval ( const Variables & surrogate_vars, const Variables & recast_vars, const Response & surrogate_response, Response & recast_response ) [static], [protected]

static function used to define the approximate subproblem constraints.

Constraint functions evaluator for solution of approximate subproblem using a RecastModel.

References Response::active_set_derivative_vector(), Response::active_set_request_vector(), SurrBasedLocalMinimizer::approxSubProbCon, SurrBasedLocalMinimizer::approxSubProbObj, Variables::continuous_variables(), Response::function_gradient(), Response::function_gradient_view(), Response::function_gradients(), Response::function_hessian(), Response::function_hessian_view(), Response::function_values(), Minimizer::numUserPrimaryFns, SurrBasedLocalMinimizer::sblmInstance, and SurrBasedLocalMinimizer::trust_region().

Referenced by SurrBasedLocalMinimizer::initialize_sub_model().

void hom_objective_eval ( int & mode, int & n, double * tau_and_x, double & f, double * grad_f, int & ) [static], [protected]

static function used by NPSOL as the objective function in the homotopy constraint relaxation formulation.

NPSOL objective functions evaluator for solution of homotopy constraint relaxation parameter optimization. This constrained optimization problem performs the update of the tau parameter in the homotopy heuristic approach used to relax the constraints in the original problem.

Referenced by SurrBasedLocalMinimizer::relax_constraints().

void hom_constraint_eval ( int & mode, int & ncnln, int & n, int & nrowj, int & needc, double * tau_and_x, double * c, double * cjac, int & nstate ) [static], [protected]

static function used by NPSOL as the constraint function in the homotopy constraint relaxation formulation.

NPSOL constraint functions evaluator for solution of homotopy constraint relaxation parameter optimization. This constrained optimization problem performs the update of the tau parameter in the homotopy heuristic approach used to relax the constraints in the original problem.

References Response::active_set(), SurrBasedLocalMinimizer::approxSubProbModel, Model::continuous_variables(), Model::current_response(), Model::evaluate(), Response::function_gradients(), Response::function_values(), SurrBasedLocalMinimizer::nonlinEqTargetsSlack, SurrBasedLocalMinimizer::nonlinIneqLowerBndsSlack, SurrBasedLocalMinimizer::nonlinIneqUpperBndsSlack, Minimizer::numNonlinearEqConstraints, ActiveSet::request_vector(), Model::response_size(), SurrBasedLocalMinimizer::sblmInstance, and SurrBasedLocalMinimizer::tau.

Referenced by SurrBasedLocalMinimizer::relax_constraints().

The documentation for this class was generated from the following files:

- SurrBasedLocalMinimizer.hpp
- SurrBasedLocalMinimizer.cpp

14.253  SurrBasedMinimizer Class Reference

Base class for local/global surrogate-based optimization/least squares.

Inheritance diagram for SurrBasedMinimizer:
CHAPTER 14. CLASS DOCUMENTATION

Protected Member Functions

- **SurrBasedMinimizer** (ProblemDescDB &problem.db, Model &model, std::shared_ptr<TraitsBase> traits)
  
  *constructor*

- **~SurrBasedMinimizer** ()
  
  *destructor*

- void derived_init_communicators (ParLevLIter pl_iter)
  
  *derived class contributions to initializing the communicators associated with this Iterator instance*

- void derived_set_communicators (ParLevLIter pl_iter)
  
  *derived class contributions to setting the communicators associated with this Iterator instance*

- void derived_free_communicators (ParLevLIter pl_iter)
  
  *derived class contributions to freeing the communicators associated with this Iterator instance*

- void initialize_graphics (int iterator_server_id=1)
  
  *initialize graphics customized for surrogate-based iteration*

- void print_results (std::ostream &s, short results_state=FINAL_RESULTS)

- void update_lagrange_multipliers (const RealVector &fn_vals, const RealMatrix &fn_grads, SurrBasedLevelData &tr_data)
  
  *initialize and update Lagrange multipliers for basic Lagrangian*

- void update_augmented_lagrange_multipliers (const RealVector &fn_vals)
  
  *initialize and update the Lagrange multipliers for augmented Lagrangian*

- void initialize_filter (SurrBasedLevelData &tr_data, const RealVector &fn_vals)
  
  *(re-)initialize filter from a set of function values*

- bool update_filter (SurrBasedLevelData &tr_data, const RealVector &fn_vals)
  
  *update filter using a new set of function values*

- Real lagrangianMerit (const RealVector &fn_vals, const BoolDeque &sense, const RealVector &primary_wts, const RealVector &nl_nineq_l_bnds, const RealVector &nl_nineq_u_bnds, const RealVector &nl_neq_tgtgs)
  
  *compute a Lagrangian function from a set of function values*

- void lagrangianGradient (const RealVector &fn_vals, const RealMatrix &fn_grads, const BoolDeque &sense, const RealVector &primary_wts, const RealVector &nl_nineq_l_bnds, const RealVector &nl_nineq_u_bnds, const RealVector &nl_neq_tgtgs, const RealVector &nl_eq_target, RealVector &lag_grad)
  
  *compute the gradient of the Lagrangian function*
void lagrangian_hessian (const RealVector &fn_vals, const RealMatrix &fn_grads, const RealSymMatrix-Array &fn_hessians, const BoolDeque &sense, const RealVector &primary_wts, const RealVector &nln_ineq_l_bnds, const RealVector &nln_ineq_u_bnds, const RealVector &nln_eq_tgts, RealSymMatrix &lag_hess)

compute the Hessian of the Lagrangian function

Real augmented_lagrangianMerit (const RealVector &fn_vals, const BoolDeque &sense, const RealVector &primary_wts, const RealVector &nln_ineq_l_bnds, const RealVector &nln_ineq_u_bnds, const RealVector &nln_eq_tgts)

compute an augmented Lagrangian function from a set of function values

void augmented_lagrangian_gradient (const RealVector &fn_vals, const RealMatrix &fn_grads, const BoolDeque &sense, const RealVector &primary_wts, const RealVector &nln_ineq_l_bnds, const RealVector &nln_ineq_u_bnds, const RealVector &nln_eq_tgts, RealVector &alag_grad)

compute the gradient of the augmented Lagrangian function

void augmented_lagrangian_hessian (const RealVector &fn_vals, const RealMatrix &fn_grads, const RealSymMatrixArray &fn_hessians, const BoolDeque &sense, const RealVector &primary_wts, const RealVector &nln_ineq_l_bnds, const RealVector &nln_ineq_u_bnds, const RealVector &nln_eq_tgts, RealSymMatrix &alag_hess)

compute the Hessian of the augmented Lagrangian function

Real penalty_merit (const RealVector &fn_vals, const BoolDeque &sense, const RealVector &primary_wts)

compute a penalty function from a set of function values

void penalty_gradient (const RealVector &fn_vals, const RealMatrix &fn_grads, const BoolDeque &sense, const RealVector &primary_wts, RealVector &pen_grad)

compute the gradient of the penalty function

Real constraintViolation (const RealVector &fn_vals, const Real &constraint_tol)

compute the constraint violation from a set of function values

Protected Attributes

• Iterator approxSubProbMinimizer
  *the minimizer used on the surrogate model to solve the approximate subproblem on each surrogate-based iteration*

• size_t globalIterCount
  *global iteration counter corresponding to number of surrogate-based minimizations*

• RealVector lagrangeMult
  *Lagrange multipliers for basic Lagrangian calculations.*

• RealVector augLagrangeMult
  *Lagrange multipliers for augmented Lagrangian calculations.*

• Real penaltyParameter
  *the penalization factor for violated constraints used in quadratic penalty calculations; increased in update_penalty()*

• RealVector origNonlinIneqLowerBnds
  *original nonlinear inequality constraint lower bounds (no relaxation)*

• RealVector origNonlinIneqUpperBnds
  *original nonlinear inequality constraint upper bounds (no relaxation)*

• RealVector origNonlinEqTargets
  *original nonlinear equality constraint targets (no relaxation)*
• Real \eta
  constant used in etaSequence updates
• Real alphaEta
  power for etaSequence updates when updating penalty
• Real betaEta
  power for etaSequence updates when updating multipliers
• Real etaSequence
  decreasing sequence of allowable constraint violation used in augmented Lagrangian updates (refer to Conn, Gould, and Toint, section 14.4)
• size_t miPLIndex
  index for the active ParallelLevel within ParallelConfiguration::miPLI ters

Additional Inherited Members
14.253.1 Detailed Description
Base class for local/global surrogate-based optimization/least squares.
  These minimizers use a SurrogateModel to perform optimization based either on local trust region methods or
  global updating methods.

14.253.2 Member Function Documentation

void print_results ( std::ostream & s, short results_state = FINAL_RESULTS ) [protected], [virtual]
Redefines default iterator results printing to include optimization results (objective functions and constraints).
  Reimplemented from Iterator.
  References Dakota::abort_handler(), Iterator::activeSet, Iterator::bestResponseArray, Iterator:: bestVariablesArray, Dakota::data_pairs, Model::interface_id(), Iterator::iteratedModel, Dakota:: lookup_by_val(), Iterator::method Name, Minimizer::numFunctions, Minimizer::numUserPrimaryFns, Minimizer::optimizationFlag, Minimizer::print residuals(), ActiveSet::request_values(), and Model::truth_model().

void update_lagrange_multipliers ( const RealVector & fn_vals, const RealMatrix & fn_grads, SurrBasedLevelData & tr_data ) [protected]
initialize and update Lagrange multipliers for basic Lagrangian
  For the Rockafellar augmented Lagrangian, simple Lagrange multiplier updates are available which do not
  require the active constraint gradients. For the basic Lagrangian, Lagrange multipliers are estimated through
  solution of a nonnegative linear least squares problem.
  References Dakota::abort_handler(), Minimizer::bigRealBoundSize, Minimizer::constraintTol, Model:: continuous_lower_bounds(), Model::continuous_upper_bounds(), Iterator::iteratedModel, SurrBasedMinimizer::lagrangeMult, Minimizer::numContinuousVars, Minimizer::numNonlinearEqConstraints, Minimizer::numNonlinearIneqConstraints, Minimizer::numUserPrimaryFns, Minimizer::objectiveGradient(), SurrBasedMinimizer::origNonlinIneqLowerB nds, SurrBasedMinimizer::origNonlinIneqUpperB nds, Model::primary_response_fn_sense(), and Model:: primary_response_fn_weights().
  Referenced by SurrBasedLocalMinimizer::hard_convergence_check().
void update_augmented_lagrange_multipliers ( const RealVector & fn_vals ) [protected]
initialize and update the Lagrange multipliers for augmented Lagrangian

For the Rockafellar augmented Lagrangian, simple Lagrange multiplier updates are available which do not require the active constraint gradients. For the basic Lagrangian, Lagrange multipliers are estimated through solution of a nonnegative linear least squares problem.

References SurrBasedMinimizer::augLagrangeMult, SurrBasedMinimizer::betaEta, Minimizer::bigRealBoundSize, SurrBasedMinimizer::etaSequence, Minimizer::numNonlinearEqConstraints, Minimizer::numNonlinearIneqConstraints, Minimizer::numUserPrimaryFns, SurrBasedMinimizer::origNonlinIneqTargets, SurrBasedMinimizer::origNonlinIneqLowerBnds, SurrBasedMinimizer::origNonlinIneqUpperBnds, and SurrBasedMinimizer::penaltyParameter.

Referenced by SurrBasedLocalMinimizer::compute_trust_region_ratio(), SurrBasedLocalMinimizer::hard_convergence_check(), and EffGlobalMinimizer::minimize_surrogates_on_model().

bool update_filter ( SurrBasedLevelData & tr_data, const RealVector & fn_vals ) [protected]
update filter using a new set of function values

Update the paretoFilter with fn_vals if new iterate is non-dominated.

References SurrBasedMinimizer::constraintViolation(), Iterator::iteratedModel, Minimizer::numNonlinearConstraints, Minimizer::objective(), Model::primary_response_fn_sense(), and Model::primary_response_fn_weights().

   Referenced by SurrBasedLocalMinimizer::compute_trust_region_ratio().

Real lagrangian_merit ( const RealVector & fn_vals, const BoolDeque & sense, const RealVector & primary_wts, const RealVector & nln_ineq_l_bnds, const RealVector & nln_ineq_u_bnds, const RealVector & nln_eq_tgts ) [protected]
compute a Lagrangian function from a set of function values

The Lagrangian function computation sums the objective function and the Lagrange multiplier terms for inequality/equality constraints. This implementation follows the convention in Vanderplaats with g<=0 and h=0. The bounds/targets passed in may reflect the original constraints or the relaxed constraints.

References Minimizer::bigRealBoundSize, Minimizer::constraintTol, SurrBasedMinimizer::lagrangeMult, Minimizer::numNonlinearEqConstraints, Minimizer::numNonlinearIneqConstraints, Minimizer::numUserPrimaryFns, and Minimizer::objective().

   Referenced by SurrBasedLocalMinimizer::approx_subprob_objective_eval(), and SurrBasedLocalMinimizer::compute_trust_region_ratio().

Real augmented_lagrangian_merit ( const RealVector & fn_vals, const BoolDeque & sense, const RealVector & primary_wts, const RealVector & nln_ineq_l_bnds, const RealVector & nln_ineq_u_bnds, const RealVector & nln_eq_tgts ) [protected]
compute an augmented Lagrangian function from a set of function values

The Rockafellar augmented Lagrangian function sums the objective function, Lagrange multiplier terms for inequality/equality constraints, and quadratic penalty terms for inequality/equality constraints. This implementation follows the convention in Vanderplaats with g<=0 and h=0. The bounds/targets passed in may reflect the original constraints or the relaxed constraints.

References SurrBasedMinimizer::augLagrangeMult, Minimizer::bigRealBoundSize, Minimizer::numNonlinearEqConstraints, Minimizer::numNonlinearIneqConstraints, Minimizer::numUserPrimaryFns, Minimizer::objective(), and SurrBasedMinimizer::penaltyParameter.

   Referenced by SurrBasedLocalMinimizer::approx_subprob_objective_eval(), SurrBasedLocalMinimizer::compute_trust_region_ratio(), EffGlobalMinimizer::get_augmented_lagrangian(), EffGlobalMinimizer::get_best_sample(), and EffGlobalMinimizer::minimize_surrogates_on_model().
Real penalty_merit (const RealVector & fn_vals, const BoolDeque & sense, const RealVector & primary_wts) [protected]

compute a penalty function from a set of function values
The penalty function computation applies a quadratic penalty to any constraint violations and adds this to the objective function(s) \( p = f + r_p \, cv \).
References SurrBasedMinimizer::constraint_violation(), Minimizer::constraintTol, Minimizer::objective(), and SurrBasedMinimizer::penaltyParameter.
Referenced by SurrBasedLocalMinimizer::compute_trust_region_ratio().

Real constraint_violation (const RealVector & fn_vals, const Real & constraint_tol) [protected]

compute the constraint violation from a set of function values
Compute the quadratic constraint violation defined as \( cv = g^+ T g^+ \)
- \( h^+ T h^+ \). This implementation supports equality constraints and 2-sided inequalities. The constraint_tol allows for a small constraint infeasibility (used for penalty methods, but not Lagrangian methods).
References Minimizer::bigRealBoundSize, Minimizer::numNonlinearEqConstraints, Minimizer::numNonlinearIneqConstraints, Minimizer::numUserPrimaryFns, SurrBasedMinimizer::origNonlinEqTargets, SurrBasedMinimizer::origNonlinIneqLowerBnds, and SurrBasedMinimizer::origNonlinIneqUpperBnds.
Referenced by SurrBasedLocalMinimizer::compute_trust_region_ratio(), SurrBasedLocalMinimizer::hard_convergence_check(), SurrBasedMinimizer::initialize_filter(), EffGlobalMinimizer::minimize_surrogates_on_model(), SurrBasedMinimizer::penalty_merit(), SurrBasedLocalMinimizer::relax_constraints(), SurrBasedMinimizer::update_filter(), and SurrBasedLocalMinimizer::update_penalty().
The documentation for this class was generated from the following files:
- SurrBasedMinimizer.hpp
- SurrBasedMinimizer.cpp

14.254 Surrogate Class Reference

Parent class for surrogate models.
Inheritance diagram for Surrogate:

```
Surrogate

GaussianProcess
PolynomialRegression
```

Public Member Functions
- **Surrogate ()**
  
  Default constructor.
- **virtual ~Surrogate ()**
  
  Default destructor.
- **virtual void build (const MatrixXd &samples, const MatrixXd &response)=0**
  
  Build the Surrogate using specified build data.
• virtual void value (const MatrixXd &samples, MatrixXd &value)=0
  Evaluate the Surrogate at a set of prediction points.
• virtual void gradient (const MatrixXd &samples, MatrixXd &gradient, int qoi)
  Evaluate the gradient of the Surrogate at a set of prediction points.
• virtual void hessian (const MatrixXd &samples, MatrixXd &hessian, int qoi)
  Evaluate the Hessian of the Surrogate at a single point.
• void set_options (const ParameterList &options)
  Set the Surrogate's configOptions.
• void get_options (ParameterList &options)
  Get the Surrogate's configOptions.
• virtual void default_options ()=0
  Initialize the Surrogate's defaultConfigOptions.

Public Attributes
• std::shared_ptr< util::DataScaler > dataScaler
  DataScaler class for a Surrogate’s build samples.

Protected Attributes
• int numSamples
  Number of samples in the Surrogate’s build samples.
• int numVariables
  Number of features/variables in the Surrogate’s build samples.
• int numQOI
  Number of quantities of interest predicted by the surrogate. For scaler-valued surrogates numQOI = 1.
• ParameterList defaultConfigOptions
  Default Key/value options to configure the surrogate.
• ParameterList configOptions
  Key/value options to configure the surrogate - will override defaultConfigOptions.

14.254.1 Detailed Description

Parent class for surrogate models.

The Surrogate class defines the API for surrogate models contained in the Dakota surrogates module.
Pure virtual functions include build, value, and default_options. Gradient and Hessian methods are optional.
Configuration options for a surrogate are set through the use of a Teuchos ParameterList named configOptions.

14.254.2 Member Function Documentation

virtual void build ( const MatrixXd &samples, const MatrixXd &response ) [pure virtual]

Build the Surrogate using specified build data.
CHAPTER 14. CLASS DOCUMENTATION

Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>samples</th>
<th>Matrix of data for surrogate construction - (num_samples by num_features)</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>response</td>
<td>Vector of responses/targets for surrogate construction - (num_samples by num_qoi)</td>
</tr>
</tbody>
</table>

Implemented in GaussianProcess, and PolynomialRegression.

virtual void value ( const MatrixXd & samples, MatrixXd & value ) [pure virtual]

Evaluate the Surrogate at a set of prediction points.

Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>samples</th>
<th>Matrix of prediction points - (num_pts by num_features).</th>
</tr>
</thead>
<tbody>
<tr>
<td>out</td>
<td>value</td>
<td>Values of the Surrogate at the prediction points - (num_pts by num_qoi)</td>
</tr>
</tbody>
</table>

Implemented in GaussianProcess, and PolynomialRegression.

Referenced by dakota::surrogates::fd_check_gradient(), and dakota::surrogates::fd_check_hessian().

void gradient ( const MatrixXd & samples, MatrixXd & gradient, int qoi ) [virtual]

Evaluate the gradient of the Surrogate at a set of prediction points.

Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>samples</th>
<th>Coordinates of the prediction points - (num_pts by num_features).</th>
</tr>
</thead>
<tbody>
<tr>
<td>out</td>
<td>gradient</td>
<td>Matrix of gradient vectors at the prediction points - (num_pts by num_features).</td>
</tr>
<tr>
<td>in</td>
<td>qoi</td>
<td>Index of the quantity of interest for gradient evaluation - 0 for scalar-valued surrogates</td>
</tr>
</tbody>
</table>

Reimplemented in GaussianProcess, and PolynomialRegression.

Referenced by dakota::surrogates::fd_check_gradient().

void hessian ( const MatrixXd & samples, MatrixXd & hessian, int qoi ) [virtual]

Evaluate the Hessian of the Surrogate at a single point.

Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>samples</th>
<th>Coordinates of the prediction point - (num_samples by num_features).</th>
</tr>
</thead>
<tbody>
<tr>
<td>out</td>
<td>hessian</td>
<td>Hessian matrix at the prediction point - (num_features by num_features).</td>
</tr>
<tr>
<td>in</td>
<td>qoi</td>
<td>Index of the quantity of interest for Hessian evaluation - 0 for scalar-valued surrogates</td>
</tr>
</tbody>
</table>

Reimplemented in GaussianProcess, and PolynomialRegression.

References dakota::silence_unused_args().

Referenced by dakota::surrogates::fd_check_hessian().

void set_options ( const ParameterList & options )

Set the Surrogate’s configOptions.
Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>options</th>
<th>ParameterList of configuration options.</th>
</tr>
</thead>
</table>

References Surrogate::configOptions.

void get_options ( ParameterList & options )

Get the Surrogate’s configOptions.

Parameters

<table>
<thead>
<tr>
<th>out</th>
<th>options</th>
<th>ParameterList of configuration options.</th>
</tr>
</thead>
</table>

References Surrogate::configOptions.

The documentation for this class was generated from the following files:

- Surrogate.hpp
- Surrogate.cpp

14.255 SurrogateModel Class Reference

Base class for surrogate models (DataFitSurrModel and HierarchSurrModel).

Inheritance diagram for SurrogateModel:

```
   Model
     |
     SurrogateModel
       |
       DataFitSurrModel   HierarchSurrModel
```

Protected Member Functions

- **SurrogateModel (ProblemDescDB &problem_db)**
  
  constructor

- **SurrogateModel (ProblemDescDB &problem_db, ParallelLibrary &parallel_lib, const SharedVariablesData &svd, bool share_svd, const SharedResponseData &srd, bool share_srd, const ActiveSet &set, short corr_type, short output_level)**
  
  alternate constructor

- **~SurrogateModel ()**
  
  destructor

- Pecos::ProbabilityTransformation & probability_transformation ()
  
  return probability transformation employed by the Model (forwarded along to ProbabilityTransformModel recasting)

- void activate_distribution_parameter_derivatives ()
  
  activate derivative setting w.r.t. distribution parameters

- void deactivate_distribution_parameter_derivatives ()
  
  deactivate derivative setting w.r.t. distribution parameters
• void trans_grad_X_to_U (const RealVector &fn_grad_x, RealVector &fn_grad_u, const RealVector &x_vars)
  transform x-space gradient vector to u-space
• void trans_grad_U_to_X (const RealVector &fn_grad_u, RealVector &fn_grad_x, const RealVector &x_vars)
  transform u-space gradient vector to x-space
• void trans_grad_X_to_S (const RealVector &fn_grad_x, RealVector &fn_grad_s, const RealVector &x_vars)
  transform x-space gradient vector to gradient with respect to inserted distribution parameters
• void trans_hess_X_to_U (const RealSymMatrix &fn_hess_x, RealSymMatrix &fn_hess_u, const RealVector &x_vars, const RealVector &fn_grad_x)
  transform x-space Hessian matrix to u-space
• Model & subordinate_model ()
  return truth_model()
• void active_model_key (const UShortArray &key)
  set the active multi-index key within surrogate data, grid driver, and approximation classes that support the management of multiple approximation states within surrogate models
• short surrogate_response_mode () const
  return responseMode
• int derived_evaluation_id () const
  return the current evaluation id for this Model
• size_t mi_parallel_level_index () const
  return miPLIndex
• virtual void asv_split (const ShortArray &orig_asv, ShortArray &actual_asv, ShortArray &approx_asv, bool build_flag)
  distributes the incoming orig_asv among actual_asv and approx_asv
• virtual void check_submodel_compatibility (const Model &sub_model)
  verify compatibility between SurrogateModel attributes and attributes of the submodel (DataFitSurrModel::actual-Model or HierarchSurrModel::highFidelityModel)
• void extract_model_keys (const UShortArray &active_key, UShortArray &truth_key, UShortArray &surr_key)
  define truth and surrogate keys from incoming active key
• unsigned short surrogate_level_index () const
  return the level index from active low fidelity model key
• unsigned short truth_level_index () const
  return the level index from active high fidelity model key
• void check_key (int key1, int key2) const
  check for consistency in response map keys
• bool force_rebuild ()
  evaluate whether a rebuild of the approximation should be forced based on changes in the inactive data
• void asv_combine (const ShortArray &actual_asv, const ShortArray &approx_asv, ShortArray &combined_asv)
  reconstitutes a combined_asv from actual_asv and approx_asv
• void response_combine (const Response &actual_response, const Response &approx_response, Response &combined_response)
  overlays actual_response and approx_response to update combined_response
• void aggregate_response (const Response &hf_resp, const Response &lf_resp, Response &agg_resp)
  aggregate \{HF,LF\} response data to create a new response with 2x size
Protected Attributes

- IntSet surrogateFnIndices
  for mixed response sets, this array specifies the response function subset that is approximated
- short responseMode
  an enumeration that controls the response calculation mode in {DataFit,Hierarch}SurrogateModel approximate response computations
- UShortArray activeKey
  array of indices that identify the currently active model key
- UShortArray surrModelKey
  array of indices that identify the surrogate (e.g., low fidelity) model (trailing portion of activeKey, if aggregated models)
- UShortArray truthModelKey
  array of indices that identify the truth (e.g., high fidelity) model (leading portion of activeKey, if aggregated models)
- short corrType
  type of correction: additive, multiplicative, or combined
- IntIntMap truthIdMap
  map from actualModel/highFidelityModel evaluation ids to DataFitSurrModel/HierarchSurrModel ids
- IntIntMap surrIdMap
  map from approxInterface/lowFidelityModel evaluation ids to DataFitSurrModel/HierarchSurrModel ids
- int surrModelEvalCntr
  counter for calls to derived_evaluate()/derived_evaluate_nowait(); used to key response maps from SurrogateModels
- IntResponseMap surrResponseMap
  map of surrogate responses returned by derived_synchronize() and derived_synchronize_nowait()
- IntResponseMap cachedApproxRespMap
  map of approximate responses retrieved in derived_synchronize_nowait() that could not be returned since corresponding truth model response portions were still pending.
- IntVariablesMap rawVarsMap
  map of raw continuous variables used by apply_correction(). Model::varsList cannot be used for this purpose since it does not contain lower level variables sets from finite differencing.
- size_t approxBuilds
  number of calls to build_approximation()
- size_t miPLIndex
  the index of the active metaiterator-iterator parallelism level (corresponding to ParallelConfiguration::miPLIters) used at runtime
- RealVector referenceCLBnds
  stores a reference copy of active continuous lower bounds when the approximation is built; used to detect when a rebuild is required.
- RealVector referenceCUBnds
  stores a reference copy of active continuous upper bounds when the approximation is built; used to detect when a rebuild is required.
- IntVector referenceDILBnds
  stores a reference copy of active discrete int lower bounds when the approximation is built; used to detect when a rebuild is required.
- IntVector referenceDIUBnds
  stores a reference copy of active discrete int upper bounds when the approximation is built; used to detect when a rebuild is required.
stores a reference copy of active discrete int upper bounds when the approximation is built; used to detect when a rebuild is required.

- RealVector referenceDRLBnds
  stores a reference copy of active discrete real lower bounds when the approximation is built; used to detect when a rebuild is required.

- RealVector referenceDRUBnds
  stores a reference copy of active discrete real upper bounds when the approximation is built; used to detect when a rebuild is required.

- RealVector referenceICVars
  stores a reference copy of the inactive continuous variables when the approximation is built using a Distinct view; used to detect when a rebuild is required.

- IntVector referenceIDIVars
  stores a reference copy of the inactive discrete int variables when the approximation is built using a Distinct view; used to detect when a rebuild is required.

- StringMultiArray referenceIDSVars
  stores a reference copy of the inactive discrete string variables when the approximation is built using a Distinct view; used to detect when a rebuild is required.

- RealVector referenceIDRVars
  stores a reference copy of the inactive discrete real variables when the approximation is built using a Distinct view; used to detect when a rebuild is required.

Private Attributes

- Variables truthModelVars
  copy of the truth model variables object used to simplify conversion among differing variable views in force_rebuild()

- Constraints truthModelCons
  copy of the truth model constraints object used to simplify conversion among differing variable views in force_rebuild()

Additional Inherited Members

14.255.1 Detailed Description

Base class for surrogate models (DataFitSurrModel and HierarchSurrModel).

The SurrogateModel class provides common functions to derived classes for computing and applying corrections to approximations.

14.255.2 Member Function Documentation

int derived_evaluation_id ( ) const [inline], [protected], [virtual]

return the current evaluation id for this Model

return the SurrogateModel evaluation id counter. Due to possibly intermittent use of lower level components, this is not the same as approxInterface, actualModel, or orderedModels evaluation counts, which requires a consistent evaluation rekeying process.

Reimplemented from Model.

References SurrogateModel::surrModelEvalCntr.
bool force_rebuild() [protected], [virtual]

evaluate whether a rebuild of the approximation should be forced based on changes in the inactive data

This function forces a rebuild of the approximation according to the sub-model variables view, the approximation type, and whether the active approximation bounds or inactive variable values have changed since the last approximation build.

Reimplemented from Model.

References Constraints::all_continuous_lower_bounds(), Constraints::all_continuous_upper_bounds(), Variables::all_continuous_variables(), Constraints::all_discrete_int_lower_bounds(), Constraints::all_discrete_int_upper_bounds(), Variables::all_discrete_int_variables(), Constraints::all_discrete_real_lower_bounds(), Constraints::all_discrete_real_upper_bounds(), Variables::all_discrete_real_variables(), Variables::all_discrete_string_variables(), Constraints::continuous_lower_bounds(), Model::continuous_lower_bounds(), Constraints::continuous_upper_bounds(), Model::continuous_upper_bounds(), Variables::continuous_variables(), Constraints::copy(), Variables::copy(), Model::current_variables(), Model::currentVariables, Constraints::discrete_int_lower_bounds(), Model::discrete_int_lower_bounds(), Constraints::discrete_int_upper_bounds(), Model::discrete_int_upper_bounds(), Variables::discrete_int_variables(), Constraints::discrete_real_lower_bounds(), Model::discrete_real_lower_bounds(), Constraints::discrete_real_upper_bounds(), Model::discrete_real_upper Bounds(), Variables::discrete_variables(), Variables::discrete_int_variables(), Variables::discrete_real_variables(), Variables::discrete_string_variables(), Constraints::is_null(), Variables::is_null(), Model::is_null(), Model::model_type(), SurrogateModel::referenceCLBnds, SurrogateModel::referenceCU_bnds, SurrogateModel::referenceDILBnds, SurrogateModel::referenceDILBnds, SurrogateModel::referenceDRLBnds, SurrogateModel::referenceDRLBnds, SurrogateModel::referenceDRUBnds, SurrogateModel::referenceICVars, SurrogateModel::referenceIDIVars, SurrogateModel::referenceIDIVars, SurrogateModel::referenceDVars, SurrogateModel::referenceDSVars, Dakota::strbegins(), Model::subordinate_model(), Model::surrogate_Type, Model::truth_model(), SurrogateModel::truthModelCons, SurrogateModel::truthModelVars, Model::user_defined_constraints(), Model::userDefinedConstraints, and Variables::view().

Referenced by HierarchSurrModel::derived_evaluate(), DataFitSurrModel::derived_evaluate(), HierarchSurrModel::derived_evaluate_nowait(), and DataFitSurrModel::derived_evaluate_nowait().

### 14.255.3 Member Data Documentation

**short responseMode [protected]**

an enumeration that controls the response calculation mode in \{DataFit,Hierarch\}SurrogateModel approximate response computations

**SurrBasedLocalMinimizer** toggles this mode since compute_correction() does not back out old corrections.

Referenced by HierarchSurrModel::active_model_key(), SurrogateModel::asv_split(), HierarchSurrModel::component_parallel_mode(), DataFitSurrModel::DataFitSurrModel(), DataFitSurrModel::declare_sources(), HierarchSurrModel::derived_evaluate(), DataFitSurrModel::derived_evaluate(), HierarchSurrModel::derived_evaluate_nowait(), DataFitSurrModel::derived_evaluate_nowait(), HierarchSurrModel::derived_set_communicators(), DataFitSurrModel::derived_synchronize(), DataFitSurrModel::derived_synchronize_approx(), HierarchSurrModel::derived_synchronize_combine(), HierarchSurrModel::derived_synchronize_combine_nowait(), DataFitSurrModel::derived_synchronize_nowait(), HierarchSurrModel::derived_synchronize_sequential(), SurrogateModel::extract_model_keys(), HierarchSurrModel::HierarchSurrModel(), HierarchSurrModel::qoi(), DataFitSurrModel::qoi(), HierarchSurrModel::resize_from_subordinate_model(), HierarchSurrModel::resize_response(), HierarchSurrModel::serve_run(), SurrogateModel::surrogate_response_mode(), HierarchSurrModel::surrogate_response_mode(), DataFitSurrModel::surrogate_response_mode(), SurrogateModel::SurrogateModel(), and HierarchSurrModel::update_from_subordinate_model().

**size_t approxBuilds [protected]**

number of calls to build_approximation()
used as a flag to automatically build the approximation if one of the derived evaluate functions is called prior to build_approximation().

Referenced by DataFitSurrModel::approximation_coefficients(), HierarchSurrModel::build_approximation(), DataFitSurrModel::build_global(), DataFitSurrModel::build_local_multipoint(), HierarchSurrModel::derived_evaluate(), DataFitSurrModel::derived_evaluate(), HierarchSurrModel::derived_evaluate_nowait(), DataFitSurrModel::derived_evaluate_nowait(), HierarchSurrModel::init_model(), DataFitSurrModel::init_model(), DataFitSurrModel::pop_, approximation(), DataFitSurrModel::rebuild_approximation(), DataFitSurrModel::rebuild_global(), and DataFitSurrModel::update_from_model().

The documentation for this class was generated from the following files:

- SurrogateModel.hpp
- SurrogateModel.cpp

### 14.256 SurrogatesGPApprox Class Reference

Derived approximation class for Surrogates approximation classes.

Inheritance diagram for SurrogatesGPApprox:

```
Approximation

SurrogatesGPApprox
```

**Public Member Functions**

- SurrogatesGPApprox ()
  
  *default constructor*

- SurrogatesGPApprox (const ProblemDescDB &problem_db, const SharedApproxData &shared_data, const String &approx_label)
  
  *standard constructor:*

- SurrogatesGPApprox (const SharedApproxData &shared_data)
  
  *alternate constructor*

- ~SurrogatesGPApprox ()
  
  *destructor*

- dakota::ParameterList & getSurrogateOpts ()

**Protected Member Functions**

- int min_coefficients () const override
  
  *return the minimum number of samples (unknowns) required to build the derived class approximation type in numVars dimensions*

- void build () override
  
  *Do the build.*

- Real value (const Variables &vars) override
  
  *Return the value of the surrogate for a given Variables object vars.*
14.257. **SURROGATESPOLYAPPROX CLASS REFERENCE**

- const RealVector & gradient (const Variables &vars)
  retrieve the approximate function gradient for a given parameter vector
- Real value (const RealVector &c_vars) override
  Return the value of the surrogate for a given parameter vector x.
- const RealVector & gradient (const RealVector &c_vars)
  retrieve the approximate function gradient for a given parameter vector

**Private Attributes**

- dakota::ParameterList surrogateOpts
  Key/value config options for underlying surrogate.
- std::shared_ptr< dakota::surrogates::GaussianProcess > model
  The native surrogate model.

**Additional Inherited Members**

14.256.1 **Detailed Description**

Derived approximation class for Surrogates approximation classes.

The GPApproximation class is the interface between Dakota and the Surrogates Gaussian Process Module.

14.256.2 **Constructor & Destructor Documentation**

SurrogatesGPApprox ( const SharedApproxData & shared_data )

alternate constructor

On-the-fly constructor.

References SurrogatesGPApprox::surrogateOpts.

The documentation for this class was generated from the following files:

- DakotaSurrogatesGP.hpp
- DakotaSurrogatesGP.cpp

14.257 **SurrogatesPolyApprox Class Reference**

Derived approximation class for Surrogates Polynomial approximation classes.

Inheritance diagram for SurrogatesPolyApprox:

```
                        Approximation
                           |
                           v
SurrogatesPolyApprox
```
CHAPTER 14. CLASS DOCUMENTATION

Public Member Functions

- **SurrogatesPolyApprox ()**
  
  default constructor

- **SurrogatesPolyApprox (const ProblemDescDB &problem_db, const SharedApproxData &shared_data, const String &approx_label)**
  
  standard constructor:

- **SurrogatesPolyApprox (const SharedApproxData &shared_data)**
  
  alternate constructor

- **~SurrogatesPolyApprox ()**
  
  destructor

Protected Member Functions

- **int min_coefficients () const override**

  return the minimum number of samples (unknowns) required to build the derived class approximation type in num-Vars dimensions

- **void build () override**

  Do the build.

- **Real value (const Variables &vars) override**

  Return the value of the surrogate for a given Variables object vars.

- **const RealVector & gradient (const Variables &vars)**

  retrieve the approximate function gradient for a given parameter vector

- **Real value (const RealVector &c_vars) override**

  Return the value of the surrogate for a given parameter vector x.

- **const RealVector & gradient (const RealVector &c_vars)**

  retrieve the approximate function gradient for a given parameter vector

Private Attributes

- **dakota::ParameterList surrogateOpts**

  Key/value config options for underlying surrogate.

- **std::shared_ptr < dakota::surrogates::PolynomialRegression > model**

  The native surrogate model.

Additional Inherited Members

14.257.1 Detailed Description

Derived approximation class for Surrogates Polynomial approximation classes.

The GPApproximation class is the interface between Dakota and the Surrogates Gaussian Process Module.
14.257.2 Constructor & Destructor Documentation

SurrogatesPolyApprox (const SharedApproxData &shared_data)

alternate constructor
On-the-fly constructor.
The documentation for this class was generated from the following files:

- DakotaSurrogatesPoly.hpp
- DakotaSurrogatesPoly.cpp

14.258 SysCallApplicInterface Class Reference

Derived application interface class which spawns simulation codes using system calls.

Inheritance diagram for SysCallApplicInterface:

```
  Interface
  |__ ApplicationInterface
  |__ ProcessApplicInterface
      |__ SysCallApplicInterface
          |__ GridApplicInterface
```

Public Member Functions

- **SysCallApplicInterface** (const ProblemDescDB &problem_db)  
  constructor
- **~SysCallApplicInterface** ()  
  destructor

Protected Member Functions

- void wait_local_evaluation_sequence (PRPQueue &prp_queue)
- void test_local_evaluation_sequence (PRPQueue &prp_queue)
- int synchronous_local_analysis (int analysis_id)
- void init_communicators_checks (int max_eval_concurrency)
- void set_communicators_checks (int max_eval_concurrency)
- void map_bookkeeping (pid_t pid, int fn_eval_id)  
  bookkeeping of process and evaluation ids for asynchronous maps
- pid_t create_evaluation_process (bool block_flag)  
  Spawn the evaluation by managing the input filter, analysis drivers, and output filter. Called from derived_map() & derived_map_async().
CHAPTER 14. CLASS DOCUMENTATION

Private Member Functions

- bool system_call_file_test (const bfs::path &root_file)
  
  detect completion of a function evaluation through existence of the necessary results file(s)
- void spawn_evaluation_to_shell (bool block_flag)
  
  spawn a complete function evaluation
- void spawn_input_filter_to_shell (bool block_flag)
  
  spawn the input filter portion of a function evaluation
- void spawn_analysis_to_shell (int analysis_id, bool block_flag)
  
  spawn a single analysis as part of a function evaluation
- void spawn_output_filter_to_shell (bool block_flag)
  
  spawn the output filter portion of a function evaluation

Private Attributes

- IntSet sysCallSet
  
  set of function evaluation id’s for active asynchronous system call evaluations
- IntShortMap failCountMap
  
  map linking function evaluation id’s to number of response read failures

Additional Inherited Members

14.258.1 Detailed Description

Derived application interface class which spawns simulation codes using system calls.

system() is part of the C API and can be used on both Windows and Unix systems.

14.258.2 Member Function Documentation

void wait_local_evaluation_sequence ( PRPQueue & prp_queue ) [inline], [protected], [virtual]

Check for completion of active async jobs (tracked with sysCallSet). Wait for at least one completion and complete all jobs that have returned. This satisfies a “fairness” principle, in the sense that a completed job will always be processed (whereas accepting only a single completion could always accept the same completion - the case of very inexpensive fn. evals. - and starve some servers).

Implements ProcessApplicInterface.

References ApplicationInterface::completionSet, and SysCallApplicInterface::test_local_evaluation_sequence().

void test_local_evaluation_sequence ( PRPQueue & prp_queue ) [protected], [virtual]

Check for completion of active async jobs (tracked with sysCallSet). Make one pass through sysCallSet & complete all jobs that have returned.

Implements ProcessApplicInterface.

References Dakota::abort_handler(), Response::active_set(), ApplicationInterface::completionSet, SysCallApplicInterface::failCountMap, ApplicationInterface::fileNameMap, ApplicationInterface::final_eval_id_tag(), Dakota::lookup_by_eval_id(), ApplicationInterface::manage_failure(), ProcessApplicInterface::read_results_files(), SysCallApplicInterface::sysCallSet, and SysCallApplicInterface::system_call_file_test().

Referenced by SysCallApplicInterface::wait_local_evaluation_sequence().
int synchronous_local_analysis ( int analysis_id ) [inline], [protected], [virtual]

This code provides the derived function used by ApplicationInterface::serve_analyses_synch().
Reimplemented from ApplicationInterface.
References SysCallApplicInterface::spawn_analysis_to_shell().

void init_communicators_checks ( int max Eval concurrency ) [inline], [protected], [virtual]

No derived interface plug-ins, so perform construct-time checks. However, process init issues as warnings since some contexts (e.g., HierarchSurrModel) initialize more configurations than will be used.
Reimplemented from ApplicationInterface.
References ApplicationInterface::check_multiprocessor_analysis().

void set_communicators_checks ( int max Eval concurrency ) [inline], [protected], [virtual]

Process run-time issues as hard errors.
Reimplemented from ApplicationInterface.
References Dakota::abort_handler(), and ApplicationInterface::check_multiprocessor_analysis().

void spawn_evaluation_to_shell ( bool block_flag ) [private]

spawn a complete function evaluation
Put the SysCallApplicInterface to the shell. This function is used when all portions of the function evaluation (i.e., all analysis drivers) are executed on the local processor.
References CommandShell::asynch_flag(), ProcessApplicInterface::commandLineArgs, ProcessApplicInterface::curWorkdir, Dakota::flush(), ProcessApplicInterface::iFilterName, ProcessApplicInterface::multipleParamsFiles, ProcessApplicInterface::oFilterName, ProcessApplicInterface::paramsFileName, ProcessApplicInterface::prepare_process_environment(), ProcessApplicInterface::programNames, ProcessApplicInterface::reset_process_environment(), ProcessApplicInterface::resultsFileName, Dakota::substitute_params_and_results(), CommandShell::suppress_output_flag(), ApplicationInterface::suppressOutput, and ProcessApplicInterface::useWorkdir.
Referenced by SysCallApplicInterface::create_evaluation_process().

void spawn_input_filter_to_shell ( bool block_flag ) [private]

spawn the input filter portion of a function evaluation
Put the input filter to the shell. This function is used when multiple analysis drivers are spread between processors. No need to check for a Null input filter, as this is checked externally. Use of nonblocking shells is supported in this fn, although its use is currently prevented externally.
References CommandShell::asynch_flag(), ProcessApplicInterface::commandLineArgs, Dakota::flush(), ProcessApplicInterface::iFilterName, ProcessApplicInterface::paramsFileName, ProcessApplicInterface::prepare_process_environment(), ProcessApplicInterface::reset_process_environment(), ProcessApplicInterface::resultsFileName, Dakota::substitute_params_and_results(), CommandShell::suppress_output_flag(), and ApplicationInterface::suppressOutput.
Referenced by SysCallApplicInterface::create_evaluation_process().

void spawn_analysis_to_shell ( int analysis_id, bool block_flag ) [private]

spawn a single analysis as part of a function evaluation
Put a single analysis to the shell. This function is used when multiple analysis drivers are spread between processors. Use of nonblocking shells is supported in this fn, although its use is currently prevented externally.
References CommandShell::asynch_flag(), ProcessApplicInterface::commandLineArgs, Dakota::flush(), ProcessApplicInterface::multipleParamsFiles, ProcessApplicInterface::paramsFileName, ProcessApplicInterface::prepare_process_environment(), ProcessApplicInterface::programNames, ProcessApplicInterface::reset_process_environment(), ProcessApplicInterface::resultsFileName, Dakota::substitute_params_and_results(), CommandShell::suppress_output_flag(), and ApplicationInterface::suppressOutput.

Referenced by SysCallApplicInterface::create_evaluation_process(), SysCallApplicInterface::synchronous_local_analysis(), and GridApplicInterface::synchronous_local_analysis().

```cpp
void spawn_output_filter_to_shell ( bool block_flag ) [private]
```

spawn the output filter portion of a function evaluation
Put the output filter to the shell. This function is used when multiple analysis drivers are spread between processors. No need to check for a Null output filter, as this is checked externally. Use of nonblocking shells is supported in this fn, although its use is currently prevented externally.

References CommandShell::asynch_flag(), ProcessApplicInterface::commandLineArgs, Dakota::flush(), ProcessApplicInterface::oFilterName, ProcessApplicInterface::paramsFileName, ProcessApplicInterface::prepare_process_environment(), ProcessApplicInterface::reset_process_environment(), ProcessApplicInterface::resultsFileName, Dakota::substitute_params_and_results(), CommandShell::suppress_output_flag(), and ApplicationInterface::suppressOutput.

Referenced by SysCallApplicInterface::create_evaluation_process().

The documentation for this class was generated from the following files:

- SysCallApplicInterface.hpp
- SysCallApplicInterface.cpp

### 14.259 TabularDataTruncated Class Reference

exception thrown when data read truncated

Inheritance diagram for TabularDataTruncated:

```
TabularDataTruncated
   |___________________________
        |_________________________
FileReadException
   |___________________________
        |_________________________
TabularDataTruncated
```

#### Public Member Functions

- `TabularDataTruncated (const std::string &msg)`

### 14.259.1 Detailed Description

exception thrown when data read truncated

The documentation for this class was generated from the following file:

- dakota_global_defs.hpp
14.260 TabularReader Class Reference

Utility used in derived read_core to read values in tabular format.

Public Member Functions

- template<typename ArrayType>
  void operator() (std::istream &s, size_t start_index, size_t num_items, ArrayType &array_data, StringMultiArrayView label_array)

14.260.1 Detailed Description

Utility used in derived read_core to read values in tabular format.

14.260.2 Member Function Documentation

void operator() (std::istream &s, size_t start_index, size_t num_items, ArrayType &array_data, StringMultiArrayView label_array) [inline]

The tabular reader doesn’t forward the label arrays

The documentation for this class was generated from the following file:

- DakotaVariables.hpp

14.261 TabularWriter Class Reference

Utility used in derived write_core to write values in tabular format.

Public Member Functions

- template<typename ArrayType>
  void operator() (std::ostream &s, size_t start_index, size_t num_items, const ArrayType &array_data, StringMultiArrayConstView label_array)

14.261.1 Detailed Description

Utility used in derived write_core to write values in tabular format.

14.261.2 Member Function Documentation

void operator() (std::ostream &s, size_t start_index, size_t num_items, const ArrayType &array_data, StringMultiArrayConstView label_array) [inline]

The tabular writer doesn’t forward the label arrays

The documentation for this class was generated from the following file:

- DakotaVariables.hpp
14.262  TANA3Approximation Class Reference

Derived approximation class for TANA-3 two-point exponential approximation (a multipoint approximation). Inheritance diagram for TANA3Approximation:

```
Approximation
    
TANA3Approximation
```

Public Member Functions

- `TANA3Approximation ()`
  default constructor
- `TANA3Approximation (ProblemDescDB &problem_db, const SharedApproxData &shared_data, const String &approx_label)`
  standard constructor
- `TANA3Approximation (const SharedApproxData &shared_data)`
  alternate constructor
- `~TANA3Approximation ()`
  destructor

Protected Member Functions

- `int min_coefficients () const`
  return the minimum number of samples (unknowns) required to build the derived class approximation type in num-Vars dimensions
- `void build ()`
  builds the approximation from scratch
- `Real value (const Variables &vars)`
  retrieve the approximate function value for a given parameter vector
- `const RealVector & gradient (const Variables &vars)`
  retrieve the approximate function gradient for a given parameter vector
- `void clear_current_active_data ()`

Private Member Functions

- `void find_scaled_coefficients ()`
  compute TANA coefficients based on scaled inputs
- `void offset (const RealVector &x, RealVector &s)`
  based on minX, offset original parameters (x) to define positive parameters (s)
Private Attributes

- RealVector pExp
  vector of exponent values
- RealVector minX
  vector of minimum param values used for offset/scaling
- RealVector scX1
  vector of scaled and/or offset x1 values
- RealVector scX2
  vector of scaled and/or offset x2 values
- Real H
  the scalar Hessian value in the TANA-3 approximation

Additional Inherited Members

14.262.1 Detailed Description

Derived approximation class for TANA-3 two-point exponential approximation (a multipoint approximation).

The TANA3Approximation class provides a multipoint approximation based on matching value and gradient data from two points (typically the current and previous iterates) in parameter space. It forms an exponential approximation in terms of intervening variables.

14.262.2 Member Function Documentation

void build ( ) [protected], [virtual]

builds the approximation from scratch

This is the common base class portion of the virtual fn and is insufficient on its own; derived implementations should explicitly invoke (or reimplement) this base class contribution.

Reimplemented from Approximation.

References Dakota::abort_handler(), Approximation::approxData, Approximation::build(), TANA3Approximation::find_scaled_coefficients(), Dakota::length(), TANA3Approximation::minX, SharedApproxData::numVars, TANA3Approximation::pExp, and Approximation::sharedDataRep.

void clear_current_active_data ( ) [inline], [protected], [virtual]

Redefine default implementation to support history mechanism.

Reimplemented from Approximation.

References Approximation::approxData, SharedApproxData::approxDataKeys, and Approximation::sharedDataRep.

The documentation for this class was generated from the following files:

- TANA3Approximation.hpp
- TANA3Approximation.cpp
14.263 TaylorApproximation Class Reference

Derived approximation class for first- or second-order Taylor series (a local approximation).

Inheritance diagram for TaylorApproximation:

```
Approximation
```
```
TaylorApproximation
```

Public Member Functions

- **TaylorApproximation ()**
  default constructor
- **TaylorApproximation (ProblemDescDB &problem_db, const SharedApproxData &shared_data, const String &approx_label)**
  standard constructor
- **TaylorApproximation (const SharedApproxData &shared_data)**
  alternate constructor
- **~TaylorApproximation ()**
  destructor

Protected Member Functions

- **int min_coefficients () const**
  return the minimum number of samples (unknowns) required to build the derived class approximation type in num-
  Vars dimensions
- **void build ()**
  builds the approximation from scratch
- **Real value (const Variables &vars)**
  retrieve the approximate function value for a given parameter vector
- **const RealVector & gradient (const Variables &vars)**
  retrieve the approximate function gradient for a given parameter vector
- **const RealSymMatrix & hessian (const Variables &vars)**
  retrieve the approximate function Hessian for a given parameter vector

Additional Inherited Members

14.263.1 Detailed Description

Derived approximation class for first- or second-order Taylor series (a local approximation).

The **TaylorApproximation** class provides a local approximation based on data from a single point in parameter space. It uses a zeroth-, first- or second-order Taylor series expansion: \( f(x) = f(x_c) \) for zeroth-order, plus \( \text{grad}(x_c)' (x - x_c) \) for first- and second-order, and plus \( (x - x_c)' \text{Hess}(x_c) (x - x_c) / 2 \) for second-order.
14.263.2 Member Function Documentation

```cpp
void build() [protected], [virtual]
```

builds the approximation from scratch

This is the common base class portion of the virtual fn and is insufficient on its own; derived implementations
should explicitly invoke (or reimplement) this base class contribution.

Reimplemented from `Approximation`.

References Dakota::abort_handler(), Approximation::approxData, Approximation::build(), SharedApproxData-
::buildDataOrder, SharedApproxData::numVars, and Approximation::sharedDataRep.

The documentation for this class was generated from the following files:

- TaylorApproximation.hpp
- TaylorApproximation.cpp

14.264 TestDriverInterface Class Reference

Inheritance diagram for TestDriverInterface:

```
TestDriverInterface
```

<table>
<thead>
<tr>
<th>Interface</th>
</tr>
</thead>
<tbody>
<tr>
<td>ApplicationInterface</td>
</tr>
<tr>
<td>DirectApplicInterface</td>
</tr>
<tr>
<td>TestDriverInterface</td>
</tr>
</tbody>
</table>

Public Member Functions

- `TestDriverInterface (const ProblemDescDB &problem_db)`
  
  constructor

- `~TestDriverInterface ()`
  
  destructor

Protected Member Functions

- virtual int `derived_map_ac (const Dakota::String &ac_name)`
  
  execute an analysis code portion of a direct evaluation invocation

Private Member Functions

- int `cantilever ()`
  
  scaled cantilever test function for optimization

- int `mod_cantilever ()`
  
  unscaled cantilever test function for UQ
• int cantilever_mtl ()
  unscaled cantilever test function for UQ with levels
• int cyl_head ()
  the cylinder head constrained optimization test fn
• int multimodal ()
  multimodal UQ test function
• int log_ratio ()
  the log_ratio UQ test function
• int short_column ()
  the short_column UQ/OUU test function
• int If_short_column ()
  a low fidelity short_column test function
• int mf_short_column ()
  alternate short_column formulations for < multifidelity or model form studies
• int alternate_short_column_forms (int form)
  helper fn for alternate forms
• int side_impact_cost ()
  the side_impact_cost UQ/OUU test function
• int side_impact_perf ()
  the side_impact_perf UQ/OUU test function
• int rosenbrock ()
  the Rosenbrock optimization and least squares test fn
• int modified_rosenbrock ()
  the modified Rosenbrock optimization and least squares test fn. The modification is the addition of an \( \sin^2 \) term so that function can not be exactly approximated by a low degree polynomial
• int generalized_rosenbrock ()
  n-dimensional Rosenbrock (Schittkowski)
• int extended_rosenbrock ()
  n-dimensional Rosenbrock (Nocedal/Wright)
• int lf_rosenbrock ()
  a low fidelity version of the Rosenbrock function
• int extra_lf_rosenbrock ()
  an extra low fidelity version of the Rosenbrock function
• int mf_rosenbrock ()
  alternate Rosenbrock formulations for < multifidelity or model form studies
• int lf_poly_prod ()
  modified low fidelity Rosenbrock to test SBO with < hierarchical approximations
• int poly_prod ()
  modified low fidelity Rosenbrock to test SBO with < hierarchical approximations
• int gerstner ()
  the isotropic/anisotropic Gerstner test function family
• int scalable_gerstner ()
scalable versions of the Gerstner test family

- void **get_genz_coefficients** (int num_dims, Real factor, int c_type, RealVector &c, RealVector &w)
  
  define coefficients needs for genz model

- int **genz** ()
  
  scalable test functions from the Genz test suite

- int **damped_oscillator** ()
  
  1d-6d that returns field values (ode solution)

- int **steady_state_diffusion_1d** ()
  
  solve the 1d steady-state diffusion eqn < with uncertain field diffusivity

- int **ss_diffusion_discrepancy** ()
  
  difference steady_state_diffusion_1d() < across two consecutive resolutions

- int **transient_diffusion_1d** ()
  
  solve the 1d transient diffusion equation < with uncertain scalar diffusivity

- int **predator_prey** ()

- int **steel_column_cost** ()
  
  solve a predator prey population dynamics model

- int **steel_column_perf** ()
  
  the steel_column_perf UQ/OUU test function

- int **sobol_rational** ()
  
  Sobol SA rational test function.

- int **sobol_g_function** ()
  
  Sobol SA discontinuous test function.

- int **sobol_ishigami** ()
  
  Sobol SA transcendental test function.

- int **text_book** ()
  
  the text_book constrained optimization test function

- int **text_book1** ()
  
  portion of text_book() evaluating the objective fn

- int **text_book2** ()
  
  portion of text_book() evaluating constraint 1

- int **text_book3** ()
  
  portion of text_book() evaluating constraint 2

- int **text_book_ouu** ()
  
  the text_book_ouu OUU test function

- int **scalable_text_book** ()
  
  scalable version of the text_book test function

- int **scalable_monomials** ()
  
  simple monomials for UQ exactness testing

- int **mogatest1** ()
  
  MOP2 from Van Veldhuizen, pp. 5-13.

- int **mogatest2** ()
  

- int **mogatest3** ()
Srinivas’ from Van Veldhuizen, pp. B-5.

- `illumination()`
  "illumination example in Boyd as a general minimization problem"

- `barnes()`
  "barnes test for SBO performance from Rodriguez, < Perez, Renaud, et al.

- `barnes_low()`
  "lo-fi barnes test for SBO performance"

- `herbie1D(size_t der_mode, Real xc_loc, std::vector<Real>& w_and_ders)`
  "1D components of herbie function"

- `smooth_herbie1D(size_t der_mode, Real xc_loc, std::vector<Real>& w_and_ders)`
  "1D components of smooth herbie function"

- `shubert1D(size_t der_mode, Real xc_loc, std::vector<Real>& w_and_ders)`
  "1D components of shubert function"

- `herbie()`
  "returns the N-D herbie function"

- `smooth_herbie()`
  "returns the N-D smooth herbie function"

- `shubert()`
  "returns the N-D shubert function"

- `bayes_linear()`
  "Scalable test function for Bayesian methods, to estimate parameters."

- `problem18()`

- `problem18_f(const double &x)`

- `problem18_g(const double &x)`

- `problem18_Ax(const double &A, const double &x)`

- `separable_combine(Real mult_scale_factor, std::vector<Real>& w, std::vector<Real>& d1w, std::vector<Real>& d2w)`
  "utility to combine components of separable fns"

- `levenshtein_distance(const String &v)`
  "Compute Levenshtein distance between v and LEV_REF."

- `salinas()`
  "direct interface to the SALINAS structural dynamics code"

- `mc_api_run()`
  "direct interface to ModelCenter via API, HKIM 4/3/03"

- `aniso_quad_form()`
  "1-D function using a anisotropic quadratic form"

- `steady_state_diffusion_core( SpectralDiffusionModel &model, RealVector &domain_limits)`
  "shared helper function between steady_state_diffusion_1d() and ss_diffusion_discrepancy()"

---

**Static Private Attributes**

- `static StringRealMap levenshteinDistanceCache`
  "Cache results of Levenshtein distance calc for efficiency."
Additional Inherited Members

14.264.1 Detailed Description

Specialization of DirectApplicInterface to embed algebraic test function drivers directly in Dakota

14.264.2 Member Function Documentation

```cpp
int derived_map_ac ( const Dakota::String & ac_name ) [protected], [virtual]
```

execute an analysis code portion of a direct evaluation invocation
Derived map to evaluate a particular built-in test analysis function
Reimplemented from DirectApplicInterface.
References Dakota::abort_handler(), ApplicationInterface::analysisServerId, TestDriverInterface::aniso_quad_form(), TestDriverInterface::barnes(), TestDriverInterface::barnes_Lf(), TestDriverInterface::bayes_linear(), TestDriverInterface::cantilever(), TestDriverInterface::cantilever_ml(), TestDriverInterface::cyl_head(), TestDriverInterface::damped_oscillator(), DirectApplicInterface::driverTypeMap, TestDriverInterface::extended_rosenbrock(), TestDriverInterface::extra_Lf_rosenbrock(), TestDriverInterface::generalized_rosenbrock(), TestDriverInterface::genz(), TestDriverInterface::gerstner(), TestDriverInterface::herbie(), TestDriverInterface::illumination(), TestDriverInterface::lf_poly_prod(), TestDriverInterface::lf_rosenbrock(), TestDriverInterface::lf_short_column(), TestDriverInterface::log_ratio(), TestDriverInterface::mc_api_run(), TestDriverInterface::mf_rosenbrock(), TestDriverInterface::multimodal(), TestDriverInterface::poly_prod(), TestDriverInterface::rosenbrock(), TestDriverInterface::salinas(), TestDriverInterface::scalable_gerstner(), TestDriverInterface::scalable_monomials(), TestDriverInterface::scalable_text_book(), TestDriverInterface::short_column(), TestDriverInterface::shubert(), TestDriverInterface::side_impact_cost(), TestDriverInterface::side_impact_perf(), TestDriverInterface::smooth_herbie(), TestDriverInterface::sobol_g_function(), TestDriverInterface::sobol_ishigami(), TestDriverInterface::sobol_rational(), TestDriverInterface::ss_diffusion_discrepancy(), TestDriverInterface::steady_state_diffusion_1d(), TestDriverInterface::steel_column_cost(), TestDriverInterface::steel_column_perf(), TestDriverInterface::text_book(), TestDriverInterface::text_book1(), TestDriverInterface::text_book2(), TestDriverInterface::text_book3(), TestDriverInterface::text_book_ouu(), and TestDriverInterface::transient_diffusion_1d().

```cpp
int lf_poly_prod ( ) [private]
```

modified low fidelity Rosenbrock to test SBO with < hierarchical approximations
modified lo-fi Rosenbrock to test SBO with hierarchical approximations
References Dakota::abort_handler(), DirectApplicInterface::directFnASV, DirectApplicInterface::fnGrads, DirectApplicInterface::fnHessians, DirectApplicInterface::fnVals, DirectApplicInterface::gradFlag, DirectApplicInterface::hessFlag, ApplicationInterface::multiProcAnalysisFlag, DirectApplicInterface::numACV, DirectApplicInterface::numADIV, DirectApplicInterface::numADRV, DirectApplicInterface::numFns, and DirectApplicInterface::xC.
Referenced by TestDriverInterface::derived_map_ac().

```cpp
int poly_prod ( ) [private]
```

modified low fidelity Rosenbrock to test SBO with < hierarchical approximations
modified lo-fi Rosenbrock to test SBO with hierarchical approximations
References Dakota::abort_handler(), DirectApplicInterface::directFnASV, DirectApplicInterface::fnGrads, DirectApplicInterface::fnHessians, DirectApplicInterface::fnVals, DirectApplicInterface::gradFlag, DirectApplicInterface::hessFlag, ApplicationInterface::multiProcAnalysisFlag, DirectApplicInterface::numACV, DirectApplicInterface::numADIV, DirectApplicInterface::numADRV, DirectApplicInterface::numFns, and DirectApplicInterface::xC.
CHAPTER 14. CLASS DOCUMENTATION

Referenced by TestDriverInterface::derived_map_ac().

int steady_state_diffusion_1d() [private]
solve the 1d steady-state diffusion eqn < with uncertain field diffusivity

Solve the 1D diffusion equation with an uncertain variable coefficient using the spectral Chebyshev collocation method.

del(k del(u)) = f on [0,1] subject to u(0) = 0 u(1) = 0
Here we set f = -1 and k = 1.4.*sum_d [cos(2*pi*x)/(pi*d)^2]*z[d] d=1,...,num_dims where z_d are random variables, typically i.i.d uniform[-1,1]
References Dakota::NPOS, Dakota::abort_handler(), Dakota::find_index(), DirectApplicInterface::fnVals, TestDriverInterface::steady_state_diffusion_core(), DirectApplicInterface::xC, DirectApplicInterface::xDI, DirectApplicInterface::xDILabels, DirectApplicInterface::xDS, and DirectApplicInterface::xDSLLabels.
Referenced by TestDriverInterface::derived_map_ac().

int steel_column_cost() [private]
solve a predator prey population dynamics model

the steel_column_cost UQ/UOU test function

References Dakota::abort_handler(), DirectApplicInterface::directFnASV, DirectApplicInterface::fnGrads, DirectApplicInterface::inHessians, DirectApplicInterface::fnVals, DirectApplicInterface::numDerivVars, DirectApplicInterface::numFns, DirectApplicInterface::numVars, DirectApplicInterface::varTypeDVV, and DirectApplicInterface::xC.
Referenced by TestDriverInterface::derived_map_ac().

int barnes() [private]
barnes test for SBO performance from Rodriguez, < Perez, Renaud, et al.

barnes test for SBO performance from Rodriguez, Perez, Renaud, et al. Modified 3/7/18 to incorporate random a[]

References Dakota::abort_handler(), DirectApplicInterface::directFnASV, DirectApplicInterface::directFnD-VV, DirectApplicInterface::fnGrads, DirectApplicInterface::fnVals, DirectApplicInterface::gradFlag, DirectApplicInterface::hessFlag, ApplicationInterface::multiProcAnalysisFlag, DirectApplicInterface::numACV, DirectApplicInterface::numADIV, DirectApplicInterface::numADRV, DirectApplicInterface::numDerivVars, DirectApplicInterface::numFns, and DirectApplicInterface::xC.
Referenced by TestDriverInterface::derived_map_ac().

int barnes lf() [private]
l0-fi barnes test for SBO performance

lo-fi barnes test for SBO performance from Rodriguez, Perez, Renaud, et al.

References Dakota::abort_handler(), DirectApplicInterface::directFnASV, DirectApplicInterface::directFnD-VV, DirectApplicInterface::fnGrads, DirectApplicInterface::fnVals, DirectApplicInterface::gradFlag, DirectApplicInterface::hessFlag, ApplicationInterface::multiProcAnalysisFlag, DirectApplicInterface::numACV, DirectApplicInterface::numADIV, DirectApplicInterface::numADRV, DirectApplicInterface::numDerivVars, DirectApplicInterface::numFns, and DirectApplicInterface::xC.
Referenced by TestDriverInterface::derived_map_ac().
void herbie1D ( size_t der_mode, Real xc_loc, std::vector<Real> & w_and_ders ) [private]
1D components of herbie function
1D Herbie function and its derivatives (apart from a multiplicative factor)
Referenced by TestDriverInterface::herbie().

void smooth_herbie1D ( size_t der_mode, Real xc_loc, std::vector<Real> & w_and_ders ) [private]
1D components of smooth_herbie function
1D Smoothed Herbie= 1DHerbie minus the high frequency sine term, and its derivatives (apart from a multi-
plicative factor)
Referenced by TestDriverInterface::smooth_herbie().

void shubert1D ( size_t der_mode, Real xc_loc, std::vector<Real> & w_and_ders ) [private]
1D components of shubert function
1D Shubert function and its derivatives (apart from a multiplicative factor)
Referenced by TestDriverInterface::shubert().

int herbie ( ) [private]
returns the N-D herbie function
N-D Herbie function and its derivatives.
References DirectApplicInterface::directFnASV, DirectApplicInterface::directFnDVV, TestDriverInterface::herbie1-
D(), DirectApplicInterface::numDerivVars, DirectApplicInterface::numVars, TestDriverInterface::separable_combine(),
and DirectApplicInterface::xC.
Referenced by TestDriverInterface::derived_map_ac().

int smooth_herbie ( ) [private]
returns the N-D smooth herbie function
N-D Smoothed Herbie function and its derivatives.
References DirectApplicInterface::directFnASV, DirectApplicInterface::directFnDVV, DirectApplicInterface-
::numDerivVars, DirectApplicInterface::numVars, TestDriverInterface::separable_combine(), TestDriverInterface-
::smooth_herbie1D(), and DirectApplicInterface::xC.
Referenced by TestDriverInterface::derived_map_ac().

void separable_combine ( Real mult_scale_factor, std::vector<Real> & w, std::vector<Real> & d1w,
std::vector<Real> & d2w ) [private]
utility to combine components of separable fns
this function combines N 1D functions and their derivatives to compute a N-D separable function and its
derivatives, logic is general enough to support different 1D functions in different dimensions (can mix and match)
References DirectApplicInterface::directFnASV, DirectApplicInterface::directFnDVV, DirectApplicInterface-
::fnGrads, DirectApplicInterface::fnHessians, DirectApplicInterface::fnVals, DirectApplicInterface::numDeriv-
Vars, and DirectApplicInterface::numVars.
Referenced by TestDriverInterface::herbie(), TestDriverInterface::shubert(), and TestDriverInterface::smooth-
_herbie().
Real levenshtein_distance (const String &v) [private]

Compute Levenshtein distance between v and LEV_REF.

Levenshtein distance is the number of changes (single character
References Dakota::LEV_REF, and TestDriverInterface::levenshteinDistanceCache.
Referenced by TestDriverInterface::text_book1(), TestDriverInterface::text_book2(), and TestDriverInterface::text_book3().

int mc_api_run() [private]

direct interface to ModelCenter via API, HKIM 4/3/03
The ModelCenter interface doesn’t have any specific construct vs. run time functions. For now, we manage it along with the integrated test drivers
References Dakota::abort_handler(), Interface::analysisComponents. DirectApplicInterface::analysisDriverIndex, Dakota::dc_ptr_int, DirectApplicInterface::directFnASV, Interface::fnLabels, DirectApplicInterface::fnVals, Dakota::mc_ptr_int, ApplicationInterface::multiProcAnalysisFlag, DirectApplicInterface::numACV, DirectApplicInterface::numADV, DirectApplicInterface::numADIR, DirectApplicInterface::numFns, DirectApplicInterface::xC, DirectApplicInterface::xCLabels, DirectApplicInterface::xDI, DirectApplicInterface::xDIRLabels, DirectApplicInterface::xDR, and DirectApplicInterface::xDRLabels.
Referenced by TestDriverInterface::derived_map_ac().
The documentation for this class was generated from the following files:

- TestDriverInterface.hpp
- TestDriverInterface.cpp

14.265 TKFactoryDIPC Class Reference

Custom RW TKFactory: passes Dakota QUESO instance pointer to the TK at build.
Inherits TransitionKernelFactory.

Public Member Functions

- TKFactoryDIPC (const std::string &name)
  Constructor for Dakota RW transition kernel factory.
- virtual ~TKFactoryDIPC ()
  Destructor for Dakota RW transition kernel factory.
- void set_callback (NonDQUESOBayesCalibration *queso_instance)
  Update the factory’s QUESO callback pointer.

Protected Member Functions

- virtual QUESO::SharedPtr < QUESO::BaseTKGroup
  < QUESO::GslVector,
  QUESO::GslMatrix > Type build_tk () override
  build and return the custom TK
Private Attributes

- NonDQUESOBayesCalibration * nonDQUESOInstance
  pointer for callbacks to Dakota QUESO class

14.265.1 Detailed Description

Custom RW TKfactory: passes Dakota QUESO instance pointer to the TK at build.
Can’t share this factory between random walk and logit as their constructor arguments differ.
The documentation for this class was generated from the following file:

- QUESOImpl.hpp

14.266 TKFactoryDIPCLogit Class Reference

Custom Logit RW TKfactory: passed Dakota QUESO instance pointer to the TK at build.
Inherits TransitionKernelFactory.

Public Member Functions

- TKFactoryDIPCLogit (const std::string &name)
  Constructor for Dakota Logit RW transition kernel factory.
- virtual ~TKFactoryDIPCLogit ()
  Destructor for Dakota Logit RW transition kernel factory.
- void set_callback (NonDQUESOBayesCalibration *queso_instance)
  Update the factory’s QUESO callback pointer.

Protected Member Functions

- virtual QUESO::SharedPtr< QUESO::BaseTKGroup< QUESO::GslVector, QUESO::GslMatrix > > build_tk () override
  build and return the custom TK

Private Attributes

- NonDQUESOBayesCalibration * nonDQUESOInstance
  pointer for callbacks to Dakota QUESO class

14.266.1 Detailed Description

Custom Logit RW TKfactory: passed Dakota QUESO instance pointer to the TK at build.
Can’t share this factory between random walk and logit as their constructor arguments differ.
The documentation for this class was generated from the following file:

- QUESOImpl.hpp
14.267  TPLDataTransfer Class Reference

Public Member Functions

- **TPLDataTransfer ()**
  default constructor
- **~TPLDataTransfer ()**
  destructor
- void **configure_data_adapters** (std::shared_ptr< TraitsBase >, const Model &)
  Construct maps, etc. needed to exchange data to/from Dakota and the TPL.
- int **num_dakota_nonlin_eq_constraints** () const
  Number of nonlinear equality constraints from Dakota perspective.
- int **num_tpl_nonlin_eq_constraints** () const
  Number of nonlinear equality constraints from TPL perspective.
- int **num_dakota_nonlin_ineq_constraints** () const
  Number of nonlinear inequality constraints from Dakota perspective.
- int **num_tpl_nonlin_ineq_constraints** () const
  Number of nonlinear inequality constraints from TPL perspective.
- Real **get_response_value_from_dakota** (const Response &resp) const
- template<typename VecT >
  void **get_nonlinear_ineq_constraints_from_dakota** (const Response &resp, VecT &values)
- template<typename VecT >
  void **get_best_nonlinear_ineq_constraints_from_tpl** (const VecT &values, RealVector &target)
- template<typename VecT >
  void **get_nonlinear_eq_constraints_from_dakota** (const Response &resp, VecT &values)
- template<typename VecT >
  void **get_best_nonlinear_eq_constraints_from_tpl** (const VecT &values, RealVector &target)

Protected Member Functions

- void **configure_nonlinear_eq_adapters** (NONLINEAR_EQUALITY_FORMAT, const Constraints &)
  Construct nonlinear equality maps needed to exchange data to/from Dakota and the TPL.
- void **configure_nonlinear_ineq_adapters** (NONLINEAR_INEQUALITY_FORMAT, const Constraints &, bool split_eqs)
  Construct nonlinear inequality maps needed to exchange data to/from Dakota and the TPL.

Protected Attributes

- int **numDakotaObjectiveFns**
  number of objective functions from Dakota perspective
- bool **maxSense**
  Single boolean (could be extended to multiple) indicating min/max sense of optimal value.
- int **numDakotaNonlinearEqConstraints**
  number of nonlinear equality constraints from Dakota perspective
- int **numTPLNonlinearEqConstraints**
  number of nonlinear equality constraints from TPL perspective
• std::vector<int> nonlinearEqConstraintMapIndices
  map from Dakota constraint number to TPL constraint number
• std::vector<double> nonlinearEqConstraintMapMultipliers
  multipliers for constraint transformations - may not be needed? - RWH
• std::vector<double> nonlinearEqConstraintMapTargets
  offsets for constraint transformations
• int numDakotaNonlinearIneqConstraints
  number of nonlinear inequality constraints from Dakota perspective
• int numTPLNonlinearIneqConstraints
  number of nonlinear inequality constraints actually used ... based conditionally on lower bounds using bigRealBoundSize and on whether TPL splits equalities into two inequalities
• std::vector<int> nonlinearIneqConstraintMapIndices
  map from Dakota constraint number to TPL constraint number
• std::vector<double> nonlinearIneqConstraintMapMultipliers
  multipliers for constraint transformations
• std::vector<double> nonlinearIneqConstraintMapShifts
  offsets for constraint transformations

14.267.1 Detailed Description

The TPLDataTransfer class provides ...

The documentation for this class was generated from the following files:

• DakotaTPLDataTransfer.hpp
• DakotaTPLDataTransfer.cpp

14.268 TrackerHTTP Class Reference

TrackerHTTP: a usage tracking module that uses HTTP/HTTPS via the curl library.

Classes

• struct Server
  struct to hold tracker/proxy pairs

Public Member Functions

• TrackerHTTP()
  default constructor is allowed, but doesn’t generate output
• TrackerHTTP(int world_rank=0)
  standard constructor with ProblemDescDB, rank
• ~TrackerHTTP()
  destructor to free handles
• void post_start(ProblemDescDB &problem_db)
  post the start of an analysis and archive start time
• void post_finish(unsigned runtime=0)
  post the completion of an analysis including elapsed time
Private Member Functions

- void initialize (int world_rank=0)
  shared initialization functions across constructors
- void url_add_field (std::string &url, const char *keyword, const std::string &value, bool delimit=true) const
  append keyword/value pair to url in GET style (with &keyword=value); set delimit = false to omit the &
- void build_default_data (std::string &url, std::time_t &rawtime, const std::string &mode) const
  construct URL with shared information for start/finish
- void send_data_using_get (const std::string &urltopost) const
  transmit data to the web server using GET
- void send_data_using_post (const std::string &datatopost)
  POST separate location and query; datatopost=\"name=daniel&project=curl\"
- void split_string (const std::string &s, const char &delim, std::vector<std::string> &elems)
  Split a string on a delimiter and place tokens in elems.
- void parse_tracking_string (const std::string &dt)
  Populate serverList with tracker and proxy URLs from dt.
- void populate_method_list (ProblemDescDB &problem_db)
  extract list of methods from problem database
- std::string get_uid () const
  get the real user ID
- std::string get_username () const
  get the username as reported by the environment
- std::string getHostname () const
  get the system hostname
- std::string get_os () const
  get the operating system
- std::string get_datetime (const std::time_t &rawtime) const
  get the date and time as a string YYYYMMDDHHMMSS

Private Attributes

- CURL * curlPtr
  pointer to the curl handler instance
- FILE * devNull
  pointer to /dev/null
- std::list<Server> serverList
  List of servers to try (tracker and proxy)
- long timeoutSeconds
  seconds until the request will timeout (may have issues with signals)
- std::string methodList
  list of active methods
- std::string dakotaVersion
  DAKOTA version.
14.268.1 Detailed Description

TrackerHTTP: a usage tracking module that uses HTTP/HTTPS via the curl library.

14.268.2 Member Function Documentation

```cpp
void send_data_using_get ( const std::string & urltopost ) const [private]
```
transmit data to the web server using GET
whole url including location&fields
References TrackerHTTP::curlPtr, and TrackerHTTP::outputLevel.

```cpp
void send_data_using_post ( const std::string & datatopost ) [private]
```
POST separate location and query; datatopost="name=daniel&project=curl".
separate location and query; datatopost="name=daniel&project=curl"
References TrackerHTTP::curlPtr, TrackerHTTP::outputLevel, and TrackerHTTP::serverList.
Referenced by TrackerHTTP::post_finish(), and TrackerHTTP::post_start().
The documentation for this class was generated from the following files:

- TrackerHTTP.hpp
- TrackerHTTP.cpp

14.269 TraitsBase Class Reference

Base class for traits.

Inheritance diagram for TraitsBase:
Public Member Functions

- **TraitsBase ()**
  *default constructor*
- **virtual ~TraitsBase ()**
  *destructor*
- **virtual bool is_derived ()**
  *A temporary query used in the refactor.*
- **virtual bool requires_bounds ()**
  *Return the flag indicating whether method requires bounds.*
- **virtual bool supports_linear_equality ()**
  *Return the flag indicating whether method supports linear equalities.*
- **virtual bool supports_linear_inequality ()**
  *Return the flag indicating whether method supports linear inequalities.*
- **virtual bool supports_linear_inequality_FORMAT linear_inequality_format ()**
  *Return the format used for linear inequality constraints.*
- **virtual bool supports_nonlinear_equality ()**
Return the flag indicating whether method supports nonlinear equalities.
• virtual NONLINEAR_EQUALITY_FORMAT nonlinear_equation_format ()
  Return the format used for nonlinear equality constraints.
• virtual bool supports_nonlinear_inequality ()
  Return the flag indicating whether method supports nonlinear inequalities.
• virtual NONLINEAR_INEQUALITY_FORMAT nonlinear_inequality_format ()
  Return the format used for nonlinear inequality constraints.
• virtual bool expects_nonlinear_inequalities_first ()
  Return the flag indicating whether method expects nonlinear inequality constraints followed by nonlinear equality constraints.
• virtual bool supports_scaling ()
  Return the flag indicating whether method supports parameter scaling.
• virtual bool supports_least_squares ()
  Return the flag indicating whether method supports least squares.
• virtual bool supports_multiobjectives ()
  Return the flag indicating whether method supports multiobjective optimization.
• virtual bool supports_continuous_variables ()
  Return the flag indicating whether method supports continuous variables.
• virtual bool supports_discrete_variables ()
  Return the flag indicating whether method supports discrete variables.
• virtual bool provides_best_objective ()
  Return the flag indicating whether method provides best objective result.
• virtual bool provides_best_parameters ()
  Return the flag indicating whether method provides best parameters result.
• virtual bool provides_best_constraint ()
  Return the flag indicating whether method provides best constraint result.
• virtual bool provides_final_gradient ()
  Return the flag indicating whether method provides final gradient result.
• virtual bool provides_final_hessian ()
  Return the flag indicating whether method provides final hessian result.

14.269.1 Detailed Description
Base class for traits.
TraitsBase provides default traits through various accessors.
The documentation for this class was generated from the following files:
• DakotaTraitsBase.hpp
• DakotaTraitsBase.cpp

14.270 UsageTracker Class Reference
Lightweight class to manage conditionally active Curl-based HTTP tracker via PIMPL.
Public Member Functions

- **UsageTracker ()**
  
  *default construction: no output*

- **UsageTracker (int world_rank)**
  
  *standard constructor; will output on rank 0*

- **void post_start (ProblemDescDB &problem_db)**
  
  *post the start of an analysis and archive start time*

- **void post_finish (unsigned runtime=0)**
  
  *post the completion of an analysis including elapsed time*

Private Member Functions

- **UsageTracker (const UsageTracker &)**
  
  *copy construction is disallowed*

Private Attributes

- **std::shared_ptr< TrackerHTTP > pTrackerHTTP**
  
  *posts usage data to Web server; using shared_ptr due to potentially incomplete type and requirements for checked-delete in debug builds (scoped_ptr would suffice)*

14.270.1 Detailed Description

Lightweight class to manage conditionally active Curl-based HTTP tracker via PIMPL.

All conditional compilation in managed in the cpp file; all operations are no-op in this wrapper if not enabling tracking...

14.270.2 Constructor & Destructor Documentation

**UsageTracker ( int world_rank )**

*standard constructor; will output on rank 0*

*standard constructor; will output on rank 0 and only initializes if tracking compiled in and not disable by environment*

References UsageTracker::pTrackerHTTP.

The documentation for this class was generated from the following files:

- UsageTracker.hpp
- UsageTracker.cpp

14.271 Var_icheck Struct Reference

structure for verifying bounds and initial point for string-valued vars
Public Attributes

- const char * name
- size_t DataVariablesRep::* n
- void(* vgen )(DataVariablesRep *, size_t)
- IntVector DataVariablesRep::* L
- IntVector DataVariablesRep::* U
- IntVector DataVariablesRep::* V
- StringArray DataVariablesRep::* Lbl

14.271.1 Detailed Description

structure for verifying bounds and initial point for integer-valued vars

The documentation for this struct was generated from the following file:

- NIDRProblemDescDB.cpp

14.272 Var_rcheck Struct Reference

structure for verifying bounds and initial point for real-valued vars

Public Attributes

- const char * name
- size_t DataVariablesRep::* n
- void(* vgen )(DataVariablesRep *, size_t)
- RealVector DataVariablesRep::* L
- RealVector DataVariablesRep::* U
- RealVector DataVariablesRep::* V
- StringArray DataVariablesRep::* Lbl

14.272.1 Detailed Description

structure for verifying bounds and initial point for real-valued vars

The documentation for this struct was generated from the following file:

- NIDRProblemDescDB.cpp

14.273 Variables Class Reference

Base class for the variables class hierarchy.

Inheritance diagram for Variables:

```
Variables
  MixedVariables
  RelaxedVariables
```
Public Member Functions

- **Variables ()**
  - default constructor
- **Variables (const ProblemDescDB &problem_db)**
  - standard constructor (explicit disallows its use for implicit type conversion)
- **Variables (const SharedVariablesData &svd)**
  - alternate constructor for instantiations on the fly (explicit disallows its use for implicit type conversion)
- **Variables (const Variables &vars)**
  - copy constructor
- **virtual ~Variables ()**
  - destructor
- **Variables operator= (const Variables &vars)**
  - assignment operator
- **virtual void read (std::istream &s)**
  - read a variables object from an std::istream
- **virtual void write (std::ostream &s, unsigned short vars_part=ALL_VARS) const**
  - write a variables object to an std::ostream, e.g., the console, optionally specifying which partition (all/active/inactive)
- **virtual void write_aprepro (std::ostream &s) const**
  - write a variables object to an std::ostream in aprepro format, e.g., a parameters file
- **virtual void read.annotated (std::istream &s)**
  - read a variables object in annotated format from an istream
- **virtual void write.annotated (std::ostream &s) const**
  - write a variables object in annotated format to an std::ostream
- **virtual void read_tabular (std::istream &s, unsigned short vars_part=ALL_VARS)**
  - read a variables object in tabular format from an istream, optionally specifying which partition (all/active/inactive)
- **virtual void write_tabular (std::ostream &s, unsigned short vars_part=ALL_VARS) const**
  - write a variables object in tabular format to an std::ostream, optionally specifying which partition (all/active/inactive)
- **virtual void write_tabular_labels (std::ostream &s, unsigned short vars_part=ALL_VARS) const**
  - write the labels in input spec order to a std::ostream, optionally specifying which partition (all/active/inactive)
- **virtual void read (MPIUnpackBuffer &s)**
  - read a variables object from a packed MPI buffer
- **virtual void write (MPIPackBuffer &s) const**
  - write a variables object to a packed MPI buffer
- **size_t tv () const**
  - total number of vars
- **size_t total_active () const**
  - total number of active vars
- **size_t cv () const**
  - number of active continuous vars
- **size_t cv_start () const**
  - start index of active continuous vars
- **size_t div () const**
number of active discrete int vars

- \( \text{size}_t \) \text{div}_\text{start}() \text{ const}
  - start index of active discrete int vars

- \( \text{size}_t \) \text{dsv}() \text{ const}
  - number of active discrete string vars

- \( \text{size}_t \) \text{dsv}_\text{start}() \text{ const}
  - start index of active discrete string vars

- \( \text{size}_t \) \text{drv}() \text{ const}
  - number of active discrete real vars

- \( \text{size}_t \) \text{drv}_\text{start}() \text{ const}
  - start index of active discrete real vars

- \( \text{size}_t \) \text{idiv}() \text{ const}
  - number of inactive discrete int vars

- \( \text{size}_t \) \text{idiv}_\text{start}() \text{ const}
  - start index of inactive discrete int vars

- \( \text{size}_t \) \text{idsv}() \text{ const}
  - number of inactive discrete string vars

- \( \text{size}_t \) \text{idsv}_\text{start}() \text{ const}
  - start index of inactive discrete string vars

- \( \text{size}_t \) \text{idrv}() \text{ const}
  - number of inactive discrete real vars

- \( \text{size}_t \) \text{idrv}_\text{start}() \text{ const}
  - start index of inactive discrete real vars

- \( \text{size}_t \) \text{acv}() \text{ const}
  - total number of continuous vars

- \( \text{size}_t \) \text{adiv}() \text{ const}
  - total number of discrete integer vars

- \( \text{size}_t \) \text{adsv}() \text{ const}
  - total number of discrete string vars

- \( \text{size}_t \) \text{adr}() \text{ const}
  - total number of discrete real vars

- \text{const \ SharedVariablesData & shared\_data}() \text{ const}
  - return \text{sharedVarsData}

- \text{SharedVariablesData & shared\_data}()
  - return \text{sharedVarsData}

- \text{void \ shape}()
  - shape a \text{Variables} object based on \text{sharedVarsData}

- \text{void \ reshape}()
  - reshape an existing \text{Variables} object based on updated \text{sharedVarsData}
• Real continuous_variable (size_t index) const
  
  return an active continuous variable

• const RealVector & continuous_variables () const

  return the active continuous variables (Note: returns a view by const reference, but initializing a RealVector from
  this reference invokes the Teuchos matrix copy constructor to create a Teuchos::Copy instance; to obtain a mutable
  view, use continuous_variables_view())

• void continuous_variable (Real c_var, size_t index)

  set an active continuous variable

• void continuous_variables (const RealVector &c_vars)

  set the active continuous variables

• int discrete_int_variable (size_t index) const

  return an active discrete integer variable

• const IntVector & discrete_int_variables () const

  return the active discrete integer variables (Note: returns a view by const reference, but initializing an IntVector
  from this reference invokes the Teuchos matrix copy constructor to create a Teuchos::Copy instance; to obtain a
  mutable view, use discrete_int_variables_view())

• void discrete_int_variable (int di_var, size_t index)

  set an active discrete integer variable

• void discrete_int_variables (const IntVector &di_vars)

  set the active discrete integer variables

• const String & discrete_string_variable (size_t index) const

  return an active discrete string variable

• StringMultiArrayConstView discrete_string_variables () const

  return the active discrete string variables (Note: returns a view by const reference, but initializing a StringArray
  from this reference invokes the Teuchos matrix copy constructor to create a Teuchos::Copy instance; to obtain a
  mutable view, use discrete_string_variables_view())

• void discrete_string_variable (const String &ds_var, size_t index)

  set an active discrete string variable

• void discrete_string_variables (StringMultiArrayConstView ds_vars)

  set the active discrete string variables

• Real discrete_real_variable (size_t index) const

  return an active discrete real variable

• const RealVector & discrete_real_variables () const

  return the active discrete real variables (Note: returns a view by const reference, but initializing a RealVector from
  this reference invokes the Teuchos matrix copy constructor to create a Teuchos::Copy instance; to obtain a mutable
  view, use discrete_real_variables_view())

• void discrete_real_variable (Real dr_var, size_t index)

  set an active discrete real variable

• void discrete_real_variables (const RealVector &dr_vars)

  set the active discrete real variables

• void active_variables (const Variables &vars)

  copy the active cv/div/dsv/drv variables from vars

• void all_variables (const Variables &vars)

  copy all cv/div/dsv/drv variables from vars

• RealVector & continuous_variables_view ()
return a mutable view of the active continuous variables

- IntVector & _discrete_int_variables_view_ ()
  return a mutable view of the active integer variables

- StringMultiArrayView _discrete_string_variables_view_ ()
  return a mutable view of the active string variables

- RealVector & _discrete_real_variables_view_ ()
  return a mutable view of the active real variables

- StringMultiArrayConstView _continuous_variable_labels_ () const
  return the active continuous variable labels

  - void _continuous_variable_labels_ (StringMultiArrayConstView cv_labels)
    set the active continuous variable labels

  - void _continuous_variable_label_ (const String &cv_label, size_t index)
    set an active continuous variable label

- StringMultiArrayConstView _discrete_int_variable_labels_ () const
  return the active discrete integer variable labels

  - void _discrete_int_variable_labels_ (StringMultiArrayConstView div_labels)
    set the active discrete integer variable labels

  - void _discrete_int_variable_label_ (const String &div_label, size_t index)
    set an active discrete integer variable label

- StringMultiArrayConstView _discrete_string_variable_labels_ () const
  return the active discrete string variable labels

  - void _discrete_string_variable_labels_ (StringMultiArrayConstView dsv_labels)
    set the active discrete string variable labels

  - void _discrete_string_variable_label_ (const String &dsv_label, size_t index)
    set an active discrete string variable label

- StringMultiArrayConstView _discrete_real_variable_labels_ () const
  return the active discrete real variable labels

  - void _discrete_real_variable_labels_ (StringMultiArrayConstView drv_labels)
    set the active discrete real variable labels

  - void _discrete_real_variable_label_ (const String &drv_label, size_t index)
    set an active discrete real variable label

- UShortMultiArrayConstView _continuous_variable_types_ () const
  return the active continuous variable types

  - void _continuous_variable_types_ (UShortMultiArrayConstView cv_types)
    set the active continuous variable types

  - void _continuous_variable_type_ (unsigned short cv_type, size_t index)
    set an active continuous variable type

- UShortMultiArrayConstView _discrete_int_variable_types_ () const
  return the active discrete integer variable types

  - void _discrete_int_variable_types_ (UShortMultiArrayConstView div_types)
    set the active discrete integer variable types

  - void _discrete_int_variable_type_ (unsigned short div_type, size_t index)
    set an active discrete integer variable type
• `UShortMultiArrayConstView discrete_string_variable_types () const`  
  return the active discrete string variable types

• `void discrete_string_variable_types (UShortMultiArrayConstView dsv_types)`  
  set the active discrete string variable types

• `void discrete_string_variable_type (unsigned short dsv_type, size_t index)`  
  set an active discrete string variable type

• `UShortMultiArrayConstView discrete_real_variable_types () const`  
  return the active discrete real variable types

• `void discrete_real_variable_types (UShortMultiArrayConstView drv_types)`  
  set the active discrete real variable types

• `void discrete_real_variable_type (unsigned short drv_type, size_t index)`  
  set an active discrete real variable type

• `SizetMultiArrayConstView continuous_variable_ids () const`  
  return the active continuous variable position identifiers

• `void continuous_variable_ids (SizetMultiArrayConstView cv_ids)`  
  set the active continuous variable position identifiers

• `void continuous_variable_id (size_t cv_id, size_t index)`  
  set an active continuous variable position identifier

• `const RealVector & inactive_continuous_variables () const`  
  return the inactive continuous variables

• `void inactive_continuous_variables (const RealVector &ic_vars)`  
  set the inactive continuous variables

• `void inactive_continuous_variable (Real ic_var, size_t index)`  
  set an inactive continuous variable

• `const IntVector & inactive_discrete_int_variables () const`  
  return the inactive discrete int variables

• `void inactive_discrete_int_variables (const IntVector &idi_vars)`  
  set the inactive discrete int variables

• `void inactive_discrete_int_variable (int idi_var, size_t index)`  
  set an inactive discrete int variable

• `StringMultiArrayConstView inactive_discrete_string_variables () const`  
  return the inactive discrete string variables

• `void inactive_discrete_string_variables (StringMultiArrayConstView ids_vars)`  
  set the inactive discrete string variables

• `void inactive_discrete_string_variable (const String &ids_var, size_t index)`  
  set an inactive discrete string variable

• `const RealVector & inactive_discrete_real_variables () const`  
  return the inactive discrete real variables

• `void inactive_discrete_real_variables (const RealVector &idr_vars)`  
  set the inactive discrete real variables

• `void inactive_discrete_real_variable (Real idr_var, size_t index)`  
  set an inactive discrete real variable

• `StringMultiArrayConstView inactive_continuous_variable_labels () const`
return the inactive continuous variable labels
- void inactive_continuous_variable_labels (StringMultiArrayConstView ic_vars)

set the inactive continuous variable labels
- StringMultiArrayConstView inactive_discrete_int_variable_labels () const

return the inactive discrete variable labels
- void inactive_discrete_int_variable_labels (StringMultiArrayConstView idi_vars)

set the inactive discrete variable labels
- StringMultiArrayConstView inactive_discrete_string_variable_labels () const

return the inactive discrete variable labels
- void inactive_discrete_string_variable_labels (StringMultiArrayConstView ids_vars)

set the inactive discrete variable labels
- StringMultiArrayConstView inactive_discrete_real_variable_labels () const

return the inactive discrete variable labels
- void inactive_discrete_real_variable_labels (StringMultiArrayConstView idr_vars)

set the inactive discrete variable labels
- UShortMultiArrayConstView inactive_continuous_variable_types () const

return the inactive continuous variable types
- UShortMultiArrayConstView inactive_discrete_int_variable_types () const

return the inactive discrete integer variable types
- UShortMultiArrayConstView inactive_discrete_string_variable_types () const

return the inactive discrete string variable types
- UShortMultiArrayConstView inactive_discrete_real_variable_types () const

return the inactive discrete real variable types
- SizetMultiArrayConstView inactive_continuous_variable_ids () const

return the inactive continuous variable position identifiers
- const RealVector & all_continuous_variables () const

returns a single array with all continuous variables
- void all_continuous_variables (const RealVector &ac_vars)

sets all continuous variables using a single array
- void all_continuous_variable (Real ac_var, size_t index)

set a variable within the all continuous array
- const IntVector & all_discrete_int_variables () const

returns a single array with all discrete variables
- void all_discrete_int_variables (const IntVector &adi_vars)

sets all discrete variables using a single array
- void all_discrete_int_variable (int adi_var, size_t index)

set a variable within the all discrete array
- StringMultiArrayConstView all_discrete_string_variables () const

returns a single array with all discrete variables
- void all_discrete_string_variables (StringMultiArrayConstView ads_vars)

sets all discrete variables using a single array
- void all_discrete_string_variable (const String &ads_var, size_t index)

set a variable within the all discrete array
• const RealVector & **all_discrete_real_variables**() const
  returns a single array with all discrete variables

• void **all_discrete_real_variables**(const RealVector &adr_vars)
  sets all discrete variables using a single array

• void **all_discrete_real_variable** (Real adr_var, size_t index)
  set a variable within the all discrete array

• void **as_vector** (const StringSetArray &dss_vals, RealVector &var_values) const
  get the active variables as a vector of reals, converting string values to zero-based set indices

• StringMultiArrayView **all_continuous_variable_labels**() const
  returns a single array with all continuous variable labels

• void **all_continuous_variable_labels** (StringMultiArrayConstView acv_labels)
  sets all continuous variable labels using a single array

• void **all_continuous_variable_label** (const String &acv_label, size_t index)
  set a label within the all continuous label array

• StringMultiArrayView **all_discrete_int_variable_labels** () const
  returns a single array with all discrete variable labels

• void **all_discrete_int_variable_labels** (StringMultiArrayConstView adiv_labels)
  sets all discrete variable labels using a single array

• void **all_discrete_int_variable_label** (const String &adiv_label, size_t index)
  set a label within the all discrete label array

• StringMultiArrayView **all_discrete_string_variable_labels** () const
  returns a single array with all discrete variable labels

• void **all_discrete_string_variable_labels** (StringMultiArrayConstView adsv_labels)
  sets all discrete variable labels using a single array

• void **all_discrete_string_variable_label** (const String &adsv_label, size_t index)
  set a label within the all discrete label array

• StringMultiArrayView **all_discrete_real_variable_labels** () const
  returns a single array with all discrete variable labels

• void **all_discrete_real_variable_labels** (StringMultiArrayConstView adrv_labels)
  sets all discrete variable labels using a single array

• void **all_discrete_real_variable_label** (const String &adrv_label, size_t index)
  set a label within the all discrete label array

• UShortMultiArrayConstView **all_continuous_variable_types** () const
  return all continuous variable types

• UShortMultiArrayConstView **all_discrete_int_variable_types** () const
  return all discrete variable types

• UShortMultiArrayConstView **all_discrete_string_variable_types** () const
  return all discrete variable types

• UShortMultiArrayConstView **all_discrete_real_variable_types** () const
  return all discrete variable types

• SizetMultiArrayConstView **all_continuous_variable_ids** () const
  return all continuous variable position identifiers

• SizetMultiArrayConstView **all_discrete_int_variable_ids**() const
return all discrete integer variable position identifiers
• SizedMultiArrayConstView all_discrete_string_variable_ids () const
  return all discrete string variable position identifiers
• SizedMultiArrayConstView all_discrete_real_variable_ids () const
  return all discrete real variable position identifiers
• StringArray ordered_labels (unsigned short vars_part=ALL_VARS) const
  get all or active labels in input spec order
• Variables copy (bool deep_svd=false) const
  a deep variables copy for use in history mechanisms (SharedVariablesData uses a shallow copy by default)
• const std::pair< short, short > & view () const
  returns variablesView
• std::pair< short, short > get_view (const ProblemDescDB &problem_db) const
  defines variablesView from problem_db attributes
• void inactive_view (short view2)
  sets the inactive view based on higher level (nested) context
• const String & variables_id () const
  returns the variables identifier string
• const SizedArray & variables_components_totals () const
  returns the number of variables for each of the constitutive components
• bool is_null () const
  function to check variablesRep (does this envelope contain a letter)

Protected Member Functions
• Variables (BaseConstructor, const ProblemDescDB &problem_db, const std::pair< short, short > &view)
  constructor initializes the base class part of letter classes (BaseConstructor overloading avoids infinite recursion
  in the derived class constructors - Coplien, p. 139)
• Variables (BaseConstructor, const SharedVariablesData &svd)
  constructor initializes the base class part of letter classes (BaseConstructor overloading avoids infinite recursion
  in the derived class constructors - Coplien, p. 139)
• void build_views ()
  construct active/inactive views of all variables arrays
• void build_active_views ()
  construct active views of all variables arrays
• void build_inactive_views ()
  construct inactive views of all variables arrays

Protected Attributes
• SharedVariablesData sharedVarsData
  reference-counted instance of shared variables data: id’s, labels, counts
• RealVector allContinuousVars
  array combining all of the continuous variables
• IntVector allDiscreteIntVars
array combining all of the discrete integer variables

- StringMultiArray allDiscreteStringVars
  array combining all of the discrete string variables

- RealVector allDiscreteRealVars
  array combining all of the discrete real variables

- RealVector continuousVars
  the active continuous variables array view

- IntVector discreteIntVars
  the active discrete integer variables array view

- RealVector discreteRealVars
  the active discrete real variables array view

- RealVector inactiveContinuousVars
  the inactive continuous variables array view

- IntVector inactiveDiscreteIntVars
  the inactive discrete integer variables array view

- RealVector inactiveDiscreteRealVars
  the inactive discrete real variables array view

Private Member Functions

- Variables * get_variables (const ProblemDescDB &problem_db)
  Used by the standard envelope constructor to instantiate the correct letter class.

- Variables * get_variables (const SharedVariablesData &svd) const
  Used by the alternate envelope constructors, by read functions, and by copy() to instantiate a new letter class.

- short method_map (short view_spec, bool relaxed) const
  infer domain from method selection

- short method_domain (const ProblemDescDB &problem_db) const
  infer domain from method selection

- short method_view (const ProblemDescDB &problem_db) const
  infer view from method selection

- short response_view (const ProblemDescDB &problem_db) const
  infer view from type of response data set

- void check_view_compatibility ()
  perform sanity checks on view.first and view.second after update

- template<class Archive>
  void load (Archive &ar, const unsigned int version)
    read a Variables object from an archive

- template<class Archive>
  void save (Archive &ar, const unsigned int version) const
    write a Variables object to an archive

- BOOST_SERIALIZATION_SPLIT_MEMBER () Variables *variablesRep
  pointer to the letter (initialized only for the envelope)
14.274. VERIFICATION CLASS REFERENCE

Private Attributes

- int referenceCount
  
  number of objects sharing variables

Friends

- class boost::serialization::access
  
  for serializing private data members
- bool operator== (const Variables &vars1, const Variables &vars2)
  strict equality operator (for boost hash-based lookups)
- bool operator!= (const Variables &vars1, const Variables &vars2)
  strict inequality operator
- bool nearby (const Variables &vars1, const Variables &vars2, Real rel_tol)
  tolerance-based equality operator
- std::size_t hash_value (const Variables &vars)
  hash_value

14.273.1 Detailed Description

Base class for the variables class hierarchy.

The Variables class is the base class for the class hierarchy providing design, uncertain, and state variables for continuous and discrete domains within a Model. Using the fundamental arrays from the input specification, different derived classes define different views of the data. For memory efficiency and enhanced polymorphism, the variables hierarchy employs the "letter/envelope idiom" (see Coplien "Advanced C++", p. 133), for which the base class (Variables) serves as the envelope and one of the derived classes (selected in Variables::get_variables()) serves as the letter.

14.273.2 Member Function Documentation

StringMultiArrayView discrete_string_variables_view ( ) [inline]

return a mutable view of the active discrete string variables
  
  same as discrete_string_variables(), except mutable view
  
  References Variables::allDiscreteStringVars, SharedVariablesData::dsv(), SharedVariablesData::dsv_start(), and Variables::sharedVarsData.

  The documentation for this class was generated from the following file:

  - DakotaVariables.hpp

14.274 Verification Class Reference

Base class for managing common aspects of verification studies.

Inheritance diagram for Verification:
Public Member Functions

- `bool resize()` 
  reinitializes iterator based on new variable size

Protected Member Functions

- `Verification (ProblemDescDB &problem_db, Model &model) constructor`
- `Verification (unsigned short method_name, Model &model) alternate constructor for instantiations "on the fly"
- `~Verification () destructor`
- `void print_results (std::ostream &s, short results_state=FINAL_RESULTS) print the final iterator results`

Additional Inherited Members

14.274.1 Detailed Description

Base class for managing common aspects of verification studies.
The `Verification` base class manages common data and functions, such as those involving ...

14.274.2 Member Function Documentation

```cpp
void print_results ( std::ostream & s, short results_state = FINAL_RESULTS ) [protected], [virtual]
```

print the final iterator results

This virtual function provides additional iterator-specific final results outputs beyond the function evaluation summary printed in `finalize_run()`.
Reimplemented from `Analyzer`.
Reimplemented in `RichExtrapVerification`.
References `Analyzer::print_results()`.
Referenced by `RichExtrapVerification::print_results()`.
The documentation for this class was generated from the following files:

- DakotaVerification.hpp
- DakotaVerification.cpp
14.275  **VLint Struct Reference**

structure for validating integer uncertain variable labels, bounds, values

**Public Attributes**

- int n
- VarLabel Var_Info::* VL
- Var_uinfo * vui
- StringArray DataVariablesRep::* Labels
- IntVector DataVariablesRep::* LowerBnds
- IntVector DataVariablesRep::* UpperBnds
- IntVector DataVariablesRep::* UncVars

### 14.275.1 Detailed Description

structure for validating integer uncertain variable labels, bounds, values

The documentation for this struct was generated from the following file:

- NIDRProblemDescDB.cpp

14.276  **VLreal Struct Reference**

structure for validating real uncertain variable labels, bounds, values

**Public Attributes**

- int n
- VarLabel Var_Info::* VL
- Var_uinfo * vui
- StringArray DataVariablesRep::* Labels
- RealVector DataVariablesRep::* LowerBnds
- RealVector DataVariablesRep::* UpperBnds
- RealVector DataVariablesRep::* UncVars

### 14.276.1 Detailed Description

structure for validating real uncertain variable labels, bounds, values

The documentation for this struct was generated from the following file:

- NIDRProblemDescDB.cpp

14.277  **VLstr Struct Reference**

structure for validating string uncertain variable labels, bounds, values
Public Attributes

- int n
- VarLabel Var_Info::* VL
- Var_uinfo * vui
- StringArray DataVariablesRep::* Labels
- StringArray DataVariablesRep::* LowerBnds
- StringArray DataVariablesRep::* UpperBnds
- StringArray DataVariablesRep::* UncVars

14.277.1 Detailed Description

Structure for validating string uncertain variable labels, bounds, values

The documentation for this struct was generated from the following file:

- NIDRProblemDescDB.cpp

14.278 VPSApproximation Class Reference

Derived approximation class for VPS implementation.

Inheritance diagram for VPSApproximation:

```
Approximation
    VPSApproximation
```

Public Member Functions

- VPSApproximation ()
  
  Default constructor

- VPSApproximation (const ProblemDescDB &problem_db, const SharedApproxData &shared_data, const String &approx_label)
  
  Standard constructor (to call VPS from an input deck)

- VPSApproximation (const SharedApproxData &shared_data)
  
  Alternate constructor (to call VPS from another method like POF-darts)

- ~VPSApproximation ()
  
  Destructor

- bool VPS_execute ()

- void VPS_create_containers ()

- void VPS_retrieve_neighbors (size_t ipoint, bool update_point_neighbors)

- void VPS_adjust_extend_neighbors_of_all_points ()

- void VPS_extend_neighbors (size_t ipoint)

- void VPS_build_local_surrogate (size_t cell_index)

- double VPS_evaluate_surrogate (double *x)

- void VPS_destroy_global_containers ()
• void ** retrieve_permutations (size_t &m, size_t **&perm, size_t num_dim, size_t upper_bound, bool force_sum_constraint, size_t sum_constraint)
• void build_radial_basis_function (size_t icell)
• void VPS_LS_retrieve_weights (size_t cell_index)
• double evaluate_basis_function (double *x, size_t icell, size_t ibasis)
• int constrained_leastSquare (size_t n, size_t m, double **H, double *w, double *f)
• double vec_dot_vec (size_t n, double *vec_a, double *vec_b)
• double vec_pow_vec (size_t num_dim, double *vec_a, size_t *vec_b)
• bool Cholesky (int n, double **A, double **LD)
• void Cholesky_solver (int n, double **LD, double *b, double *x)
• void GMRES (size_t n, double **A, double *b, double *x, double eps)
• void printMatrix (size_t m, size_t n, double **M)
• void initiate_random_number_generator (unsigned long x)
• double generate_a_random_number ()
• size_t ** retrieve_closest_cell (double **x)
• bool trim_line_using_Hyperplane (size_t num_dim, double *st, double *end, double *qH, double *nH)
• double f_test (double *x)
• double ** grad_f_test (double *x)
• double ** hessian_f_test (double *x)
• void generate_poisson_disk_sample (double r)
• void generate_MC_sample ()
• void isocontouring (std::string file_name, bool plot_test_function, bool plot_surrogate, std::vector<double> contours)
• void isocontouring_solid (std::string file_name, bool plot_test_function, bool plot_surrogate, std::vector<double> contours)
• void plot_neighbors ()

Protected Member Functions

• int min_coefficients () const
  
  return the minimum number of samples (unknowns) required to build the derived class approximation type in num-Vars dimensions
• void build ()
  
  builds the approximation from scratch
• Real value (const Variables &vars)
  
  retrieve the predicted function value for a given parameter set
• const RealVector & gradient (const Variables &vars)
  
  retrieve the function gradient at the predicted value for a given parameter set
• Real prediction_variance (const Variables &vars)
  
  retrieve the variance of the predicted value for a given parameter set

Private Types

• enum subsurrogate { LS, GP }
• enum subsurrogate_basis { polynomial, radial }
• enum testfunction {
  SmoothHerbie, Herbie, Cone, Cross, UnitSphere, Linear34 }

14.278. VPSAPPROXIMATION CLASS REFERENCE 1061
Private Member Functions

- void VPSmodel_build()
  
  Function to compute coefficients governing the VPS surrogates.

- void VPSmodel_apply(const RealVector &new_x, bool variance_flag, bool gradients_flag)
  
  Function returns a response value using the VPS surface.

Private Attributes

- Real approxValue
  
  Value of the approximation returned by value().

- Real approxVariance
  
  Value of the approximation returned by prediction_variance().

- RealMatrix trainPoints
  
  A 2-D array (num sample sites = rows, num vars = columns) used to create the Gaussian process.

- RealMatrix trainValues
  
  An array of response values; one response value per sample site.

- size_t numObs
  
  The number of observations on which the GP surface is built.

- int surrogateOrder
  
  The order of the polynomial in each Voronoi cell.

- subsurrogate_vps_subsurrogate

- subsurrogate_basis_vps_subsurrogate_basis

- testfunction_vps_test_function

- double Q[1220]

- int indx

- double cc

- double c

- double zc

- double zx

- double zy

- size_t qlen

- size_t n_dim

- double ** xmin

- double ** xmax

- double ** diag

- size_t ** num_inserted_points

- double ** sample_points

- double ** _fval

- double ** _fgrad

- double ** _fhess

- size_t ** sample_neighbors

- size_t ** vps_ext_neighbors

- size_t vps_order

- size_t ** num_GMRES

- size_t * num_cell_basis_functions
**Static Private Attributes**

- static `VPSApproximation * VPSinstance`
  
  *pointer to the active object instance used within the static evaluator*

**Additional Inherited Members**

**14.278.1 Detailed Description**

Derived approximation class for VPS implementation.

The `VPSApproximation` class provides a set of piecewise surrogate approximations each of which is valid within a Voronoi cell.

**14.278.2 Member Function Documentation**

```cpp
void VPSmodel::apply ( const RealVector & new_x, bool variance_flag, bool gradients_flag ) [private]
```

Function returns a response value using the VPS surface.

The response value is computed at the design point specified by the `RealVector` function argument.

References `VPSApproximation::approxValue`.

Referenced by `VPSApproximation::gradient`, `VPSApproximation::prediction_variance`, and `VPSApproximation::value`.

**14.278.3 Member Data Documentation**

```cpp
VPSApproximation * VPSinstance [static],[private]
```

*pointer to the active object instance used within the static evaluator*

*default constructor*

The documentation for this class was generated from the following files:

- `VPSApproximation.hpp`
- `VPSApproximation.cpp`
Weighting specialization of RecastModel.

Inheritance diagram for WeightingModel:

```
     Model
    /   \
RecastModel
    |    
WeightingModel
```

Public Member Functions

- **WeightingModel** (Model &sub_model)
  
  *standard constructor*

- **~WeightingModel**()
  
  *destructor*

Protected Member Functions

- void assign_instance()
  
  *assign static pointer instance to this for use in static transformation functions*

Static Protected Member Functions

- static void primary_resp_weighter(const Variables &sub_model_vars, const Variables &recast_vars, const Response &sub_model_response, Response &weighted_response)

- static void primary_resp_unweighter(const Variables &recast_vars, const Variables &sub_model_vars, const Response &weighted_res, Response &unweighted_res)=delete

Static Private Attributes

- static WeightingModel *weightModelInstance
  
  *static pointer to this class for use in static callbacks*

Additional Inherited Members

14.279.1 Detailed Description

Weighting specialization of RecastModel.

Specialization of a RecastModel that manages Response weighting (could be implemented as special case of ScalingModel, but kept separate for simplicity for now). This class provides a simple constructor that forwards to the more complicated RecastModel API.
14.279.2 Member Data Documentation

WeightingModel * weightModelInstance  [static], [private]

static pointer to this class for use in static callbacks
initialization of static needed by RecastModel
Referenced by WeightingModel::assign_instance().
The documentation for this class was generated from the following files:

- WeightingModel.hpp
- WeightingModel.cpp

14.280 WorkdirHelper Class Reference

Static Public Member Functions

- static void initialize ()
  initialize (at runtime) cached values for paths and environment
- static const std::string & startup_pwd ()
  Query for dakota’s startup $PWD.
- static void change_directory (const bfs::path &new_dir)
  change current directory
- static void prepend_preferred_env_path (const std::string &extra_path)
  Prepend cached preferredEnvPath with extra_path and update $PATH environment variable.
- static void set_environment (const std::string &env_name, const std::string &env_val, bool overwrite_flag=true)
  Set an environment variable.
- static bfs::path which (const std::string &driver_name)
  Returns the bfs::path for the analysis driver, supporting typical windows filename extensions, or empty if not found.
- static bfs::path rel_to_abs (const bfs::path &subdir_path)
  get a valid absolute bfs::path to a subdirectory relative to rundir
- static StringArray tokenize_driver (const String &user_driver)
  tokenize a white-space separated analysis driver, respecting escapes and nested quotes
- static bool resolve_driver_path (String &an_driver)
  parse off the first whitespace-separated entry in the user’s analysis_driver, and convert it to an absolute path if it begins with ./ or ../, replacing the passed string if needed. Returns true if the first token was modified.
- static void splitWildcard (const std::string &path with_wildcard, bfs::path &search_dir, bfs::path &wildcard)
  given a string with an optional path and a wildcard, e.g., /tmp/D*.?pp, parse it into the search path /tmp (default) and the wildcard D*.?pp. Return wildcard as path to reduce wstring conversions
- static bfs::path concat_path (const bfs::path &p_in, const String &tag)
  concatenate a string onto the end of a path
- static bfs::path system_tmp_file (const std::string &prefix)
  generate a valid temporary file name prefix,%%%%%%
- static bfs::path system_tmp_path ()
  get the system tmp path, e.g., /tmp or C:\\temp
- static bool create_directory (const bfs::path &dir_path, short mkdir_option)
  Create a directory, with options for remove or error.
• static void `recursive_remove` (const bfs::path &rm_path, short fileop_option)
  Remove a path (file, directory, or symlink) without regard to its type. Only error if existed and there's an error in
  the remove.
• static void `rename` (const bfs::path &old_path, const bfs::path &new_path, short fileop_option)
  Rename a file, catching any errors and optionally warning/erroring.
• static void `link_items` (const StringArray &source_items, const bfs::path &dest_dir, bool overwrite)
  top-level link a list of source_paths (files, directories, symlinks), potentially including wildcards, from destination
  dir, which must exist
• static void `copy_items` (const StringArray &source_items, const bfs::path &dest_dir, bool overwrite)
  copy a list of source_paths (files, directories, symlinks), potentially including wildcards into destination dir, which
  must exist
• static void `prepend_path_items` (const StringArray &source_items)
  prepend any directories (including wildcards) found in source_items to the preferred environment path; this will
  update cached preferred path and PATH
• static bool `check_equivalent_dest` (const StringArray &source_items, const bfs::path &dest_dir)
  check whether any of the passed source items are filesystem equivalent to the destination path, return true if any
  one is equivalent to dest
• static bool `find_driver` (const StringArray &source_items, const bfs::path &search_driver)
  check whether the any of the passed source items (possibly including wildcards to be expanded) matches the passed
  search driver
• static bool `link` (const bfs::path &src_path, const bfs::path &dest_dir, bool overwrite)
  create link from dest_dir/src_path.filename() to a single path (file, dir, link) in source directory
• static bool `recursive_copy` (const bfs::path &src_path, const bfs::path &dest_dir, bool overwrite)
  Recursive copy of src_path into dest_dir, with optional top-level overwrite (remove/recreate) of dest_dir/src_path.-
  filename()
• static bool `prepend_path_item` (const bfs::path &src_path, const bfs::path &dest_dir, bool overwrite)
  prepend the preferred env path with source path if it's a directory; this will update cached preferred path and
  manipulate PATH
• static bool `check_equivalent` (const bfs::path &src_path, const bfs::path &dest_dir, bool overwrite)
  return true if the src and dest are filesystem equivalent
• static bool `find_file` (const bfs::path &src_path, const bfs::path &search_file, bool overwrite)
  return true if the src_path is a regular file and has same filename as search_file
• static bool `file_op_items` (const file_op_function &file_op, const StringArray &source_paths, const bfs::path
  &dest_dir, bool overwrite)
  recursively perform file_op (copy, path adjust, etc.) on a list of source_paths (files, directories, symlinks), which
  potentially include wildcards, w.r.t. destination_dir
• static void `set_preferred_path` ()
  set/reset PATH to dakPreferredEnvPath
• static void `set_preferred_path` (const boost::filesystem::path &extra_path)
  set PATH to absolute(extra_path):dakPreferredEnvPath, without changing cached preferred PATH
• static void `reset` ()
  Resets the working directory "state" to its initial state when DAKOTA was launched.
Private Member Functions

- **WorkdirHelper ()**  
  default constructor
- **WorkdirHelper (const WorkdirHelper &)**  
  copy constructor
- **~WorkdirHelper ()**  
  destructor
- **WorkdirHelper & operator= (const WorkdirHelper &)**  
  assignment operator

Static Private Member Functions

- **static bfs::path po which (const std::string &driver_name)**  
  Returns the bfs::path for the analysis driver - POSIX-style implementation, returns empty if not found.
- **static std::string init_startup_path ()**  
  Initializes class member, startupPATH.
- **static std::string initPreferred_env_path ()**  
  Initializes class member, dakPreferredEnvPath.
- **static std::vector<std::string> tokenize_env_path (const std::string &path)**  
  Tokenizes $PATH$ environment variable into a "list" of directories.

Static Private Attributes

- **static std::string startupPWD = "."**  
  Value of $PWD$ var upon entry to dakota main()
- **static std::string startupPATH = "."**  
  Value of $PATH$ (PATH% on windows) var upon entry to dakota main(), omitting any leading PATH= or Path=.
- **static std::string dakPreferredEnvPath = "."**  
  Dakota preferred search PATH/Path = ".:startupPWD:startupPATH", omitting any leading PATH= or Path=.

14.280.1 Detailed Description

Utility class for cross-platform management of environment and paths. Including directory and file operations. On initialization, this class does not manipulate the present working directory, nor the PATH environment variable, but stores context to manipulate them later.

14.280.2 Member Function Documentation

**void initialize () [static]**

initialize (at runtime) cached values for paths and environment

Initialize defers calls to Boost filesystem utilities until runtime (required on some operating systems.

References WorkdirHelper::dakPreferredEnvPath, WorkdirHelper::initPreferred_env_path(), WorkdirHelper::init_startup_path(), WorkdirHelper::startupPATH, and WorkdirHelper::startupPWD.

Referenced by Environment::Environment().
void prepend_preferred_env_path ( const std::string & extra_path ) [static]

Prepend cached preferredEnvPath with extra_path and update $PATH environment variable.

Overwrites $PATH with an additional directory prepended, typically for the purpose of ensuring templatedir is in the $PATH; updates cached preferred PATH and environment PATH, so exercise caution with repeated calls.

References WorkdirHelper::dakPreferredEnvPath, WorkdirHelper::set_environment(), and WorkdirHelper::startup-PWD.

Referenced by WorkdirHelper::prepend_path_item().

bfs::path which ( const std::string & driver_name ) [static]

Returns the bfs::path for the analysis driver, supporting typical windows filename extensions, or empty if not found.

Uses string representing $PATH to locate an analysis driver on the host computer. Returns the path to the driver (as a string)

This version is a wrapper over the "plain ol' which" implementation, allowing an array of windows, 3-letter extensions to be checked.

References Dakota::get_pathext(), and WorkdirHelper::po_which().

Referenced by NIDRProblemDescDB::check_driver().

void split_wildcard ( const std::string & path_with_wc, bfs::path & search_dir, bfs::path & wildcard ) [static]

given a string with an optional path and a wildcard, e.g., /tmp/D*.?pp, parse it into the search path /tmp (default .) and the wildcard D*.?pp. Return wildcard as path to reduce wstring conversions

Input: path_with_wc; Output: search_dir, wildcard

Referenced by WorkdirHelper::file_op_items().

bool create_directory ( const bfs::path & dir_path, short mkdir_option ) [static]

Create a directory, with options for remove or error.

mkdir_option is DIR_CLEAN (remove and recreate), DIR_PERSIST (leave existing), or DIR_ERROR (don't allow existing) returns whether a new directory was created.

References Dakota::abort_handler(), and WorkdirHelper::recursive_remove().

Referenced by NonDMUQBayesCalibration::calibrate(), and ProcessApplicInterface::define_filenames().

void link_items ( const StringArray & source_items, const bfs::path & dest_dir, bool overwrite ) [static]

top-level link a list of source_paths (files, directories, symlinks), potentially including wildcards, from destination_dir, which must exist

Iterate source items (paths or wildcards), linking each of them from the destination. If overwrite, remove and replace any existing destination target, otherwise, allow to persist

References WorkdirHelper::file_op_items(), and WorkdirHelper::link().

Referenced by ProcessApplicInterface::define_filenames().

void copy_items ( const StringArray & source_items, const bfs::path & dest_dir, bool overwrite ) [static]

copy a list of source_paths (files, directories, symlinks), potentially including wildcards into destination_dir, which must exist
Iterate source items (paths or wildcards), copying each of them into the destination. If overwrite, remove and replace any existing destination target, otherwise, allow to persist.

References WorkdirHelper::file_op_items(), and WorkdirHelper::recursive_copy().

Referenced by ProcessApplicInterface::define_filenames().

**bool link ( const bfs::path & src_path, const bfs::path & dest_dir, bool overwrite ) [static]**

create link from dest_dir/src_path.filename() to a single path (file, dir, link) in source directory

Assumes source file exists since it was iterated in the calling context. If overwrite, any existing file in dest_dir will be removed prior to creating the new link.

References Dakota::abort_handler().

Referenced by WorkdirHelper::link_items().

**bool recursive_copy ( const bfs::path & src_path, const bfs::path & dest_dir, bool overwrite ) [static]**

Recursive copy of src_path into dest_dir, with optional top-level overwrite (remove/recreate) of dest_dir/src_path.-filename()

note dest_dir is the containing folder for the src_path contents to be placed in for consistency with other convenience functions (may need to reconsider)

References Dakota::abort_handler().

Referenced by WorkdirHelper::copy_items().

**bool prepend_path_item ( const bfs::path & src_path, const bfs::path & dest_dir, bool overwrite ) [static]**

prepend the preferred env path with source path if it’s a directory; this will update cached preferred path and manipulate PATH

prepend the env path with source path if it’s a directory or directory symlink

References Dakota::abort_handler(), and WorkdirHelper::prepend_preferred_env_path().

Referenced by WorkdirHelper::prepend_path_items().

**bool file_op_items ( const file_op_function & file_op, const StringArray & source_items, const bfs::path & dest_dir, bool overwrite ) [static]**

recursively perform file_op (copy, path adjust, etc.) on a list of source_paths (files, directories, symlinks), which potentially include wildcards, w.r.t. destination_dir

Iterator implementation for copy, link, etc file operation. Iterate source items (paths or wildcards), performing file_op on each w.r.t. destination. If overwrite, remove and replace any existing destination target (at top-level), otherwise, allow to persist. Return code true indicates abnormal behavior.

References WorkdirHelper::splitWildcard(), and Dakota::strContains().

Referenced by WorkdirHelper::checkEquivalent_dest(), WorkdirHelper::copy_items(), WorkdirHelper::find_driver(), WorkdirHelper::link_items(), and WorkdirHelper::prepend_path_items().

**void set_preferred_path ( const boost::filesystem::path & extra_path ) [static]**

set PATH to absolute(extra_path):dakPreferredEnvPath, without changing cached preferred PATH

If needed, convert the passed item to an absolute path (while could make sense to prepend a relative path, no current use cases) and prepend when setting environment. Does not update cached preferred path.

References WorkdirHelper::dakPreferredEnvPath, WorkdirHelper::rel_to_abs(), and WorkdirHelper::set_environment().
bfs::path po_which ( const std::string & driver_name ) [static], [private]

Returns the bfs::path for the analysis driver - POSIX-style implementation, returns empty if not found.
For absolute driver_name, validates that is regular file. For relative, uses string representing $PATH (preferred path) to locate an analysis driver on the host computer. Returns the path to the driver, or empty if not found.
This is the "plain ol' which" impl that worked well, historically, on POSIX.
References Dakota::contains(), WorkdirHelper::dakPreferredEnvPath, and WorkdirHelper::tokenize_env_path().
Referenced by WorkdirHelper::which().

std::string init_startup_path () [static], [private]

Initializes class member, startupPATH.
Gets the $PATH (PATH% on windows) and returns the std::string value
Referenced by WorkdirHelper::initialize().

std::string init_preferred_env_path () [static], [private]

Initializes class member, dakPreferredEnvPath.
Prepends `.` and the startupPWD to the initial startup $PATH string so that analysis driver detection is more robust
References WorkdirHelper::startupPATH, and WorkdirHelper::startupPWD.
Referenced by WorkdirHelper::initialize().

std::vector< std::string > tokenize_env_path ( const std::string & env_path ) [static], [private]

Tokenizes $PATH environment variable into a "list" of directories.
Creates a a vector of directories (as an aid to search) by breaking up the $PATH environment variable (passed in as a string argument)
Referenced by WorkdirHelper::po_which().
The documentation for this class was generated from the following files:

- WorkdirHelper.hpp
- WorkdirHelper.cpp
Chapter 15

File Documentation

15.1 dakota_dll_api.cpp File Reference

This file contains a DakotaRunner class, which launches DAKOTA.

Namespaces

- Dakota
  
The primary namespace for DAKOTA.

Constant Groups

- Dakota
  
The primary namespace for DAKOTA.

Functions

- void DAKOTA_DLL_FN dakota_create (int *dakota_ptr_int, const char *logname)
  
  create and configure a new DakotaRunner, adding it to list of instances
- int DAKOTA_DLL_FN dakota_readInput (int id, const char *dakotaInput)
  
  command DakotaRunner instance id to read from file dakotaInput
- void DAKOTA_DLL_FN dakota_get_variable_info (int id, char ***pVarNames, int *pNumVarNames, char ***pRespNames, int *pNumRespNames)
  
  return the variable and response names
- int DAKOTA_DLL_FN dakota_start (int id)
  
  command DakotaRunner instance id to start (plugin interface and run strategy)
- void DAKOTA_DLL_FN dakota_destroy (int id)
  
  delete Dakota runner instance id and remove from active list
- void DAKOTA_DLL_FN dakota_stop (int *id)
  
  command DakotaRunner instance id to stop execution
- const char *DAKOTA_DLL_FN dakota_getStatus (int id)
  
  return current results output as a string
- int get_mc_ptr_int ()

1071
get the DAKOTA pointer to ModelCenter

• void set_mc_ptr_int (int ptr_int)
  set the DAKOTA pointer to ModelCenter

• int get_dc_ptr_int ()
  get the DAKOTA pointer to ModelCenter current design point

• void set_dc_ptr_int (int ptr_int)
  set the DAKOTA pointer to ModelCenter current design point

15.1.1 Detailed Description
This file contains a DakotaRunner class, which launches DAKOTA.

15.1.2 Function Documentation
void DAKOTA_DLL_FN dakota_stop ( int * id )
command DakotaRunner instance id to stop execution
  TODO: trick application to quit through the syscall interface or throw exception.

15.2 dakota_dll_api.h File Reference
API for DLL interactions.

Functions
• void DAKOTA_DLL_FN dakota_create (int *dakota_ptr_int, const char *logname)
  create and configure a new DakotaRunner, adding it to list of instances

• int DAKOTA_DLL_FN dakota_readInput (int id, const char *dakotaInput)
  command DakotaRunner instance id to read from file dakotaInput

• int DAKOTA_DLL_FN dakota_start (int id)
  command DakotaRunner instance id to start (plugin interface and run strategy)

• void DAKOTA_DLL_FN dakota_destroy (int id)
  delete Dakota runner instance id and remove from active list

• void DAKOTA_DLL_FN dakota_stop (int *id)
  command DakotaRunner instance id to stop execution

• const char *DAKOTA_DLL_FN dakota_getStatus (int id)
  return current results output as a string

• int DAKOTA_DLL_FN get_mc_ptr_int ()
  get the DAKOTA pointer to ModelCenter

• void DAKOTA_DLL_FN set_mc_ptr_int (int ptr_int)
  set the DAKOTA pointer to ModelCenter

• int DAKOTA_DLL_FN get_dc_ptr_int ()
  get the DAKOTA pointer to ModelCenter current design point

• void DAKOTA_DLL_FN set_dc_ptr_int (int ptr_int)
  set the DAKOTA pointer to ModelCenter current design point
15.3. DAKOTA_LINEAR_ALGEBRA.HPP FILE REFERENCE

- void DAKOTA_DLL_FN dakota_get_variable_info (int id, char ***pVarNames, int *pNumVarNames, char ***pRespNames, int *pNumRespNames)
  return the variable and response names

15.2.1 Detailed Description

API for DLL interactions.

15.2.2 Function Documentation

void DAKOTA_DLL_FN dakota_stop ( int * id )
command DakotaRunner instance id to stop execution
  TODO: trick application to quit through the syscall interface or throw exception.

15.3 dakota_linear_algebra.hpp File Reference

Dakota linear algebra utilities.

Namespaces

- Dakota
  The primary namespace for DAKOTA.

Constant Groups

- Dakota
  The primary namespace for DAKOTA.

Functions

- void svd (RealMatrix &matrix, RealVector &singular_vals, RealMatrix &v_trans, bool compute_vectors=true)
  Compute the SVD of an arbitrary matrix A = USVᵀ.
- void singular_values (RealMatrix &matrix, RealVector &singular_values)
  compute the singular values without storing any singular vectors (A will be destroyed)
- int qr (RealMatrix &A)
  Compute an in-place QR factorization A = QR.
- int qr_rsolve (const RealMatrix &q_r, bool transpose, RealMatrix &rhs)
  Perform a multiple right-hand sides Rinv * rhs solve using the R from a qr factorization.
- double det_AtransA (RealMatrix &A)
  Use SVD to compute det(AᵀA), destroying A with the SVD.

15.3.1 Detailed Description

Dakota linear algebra utilities. Convenience functions to perform Teuchos::LAPACK operations on Dakota RealMatrix/RealVector.
15.4 dakota_tabular_io.hpp File Reference

Utility functions for reading and writing tabular data files. Emerging utilities for tabular file I/O. For now, just extraction of capability from separate contexts to facilitate rework. These augment (and leverage) those in data_util.h.

Namespaces

- **Dakota**

  The primary namespace for DAKOTA.

Constant Groups

- **Dakota**

  The primary namespace for DAKOTA.

Functions

- **String format name** (unsigned short tabular_format)

  Translate tabular format into a user-friendly name.

- **void print_expected_format** (std::ostream &s, unsigned short tabular_format, size_t num_rows, size_t num_cols)

  Describe the expected data file format based on passed parameters.

- **void print_unexpected_data** (std::ostream &s, const String &filename, const String &context_message, unsigned short tabular_format)

  Print a warning if there's extra data in the file.

- **void open_file** (std::ifstream &data_file, const std::string &input_filename, const std::string &context_message)

  Open the file specified by name for reading, using passed input stream, presenting context-specific error on failure.

- **void open_file** (std::ofstream &data_file, const std::string &output_filename, const std::string &context_message)

  Open the file specified by name for writing, using passed output stream, presenting context-specific error on failure.

- **void close_file** (std::ifstream &data_file, const std::string &input_filename, const std::string &context_message)

  Close the file specified by name after reading, using passed input stream, presenting context-specific error on failure.

- **void close_file** (std::ofstream &data_file, const std::string &output_filename, const std::string &context_message)

  Close the file specified by name after writing, using passed output stream, presenting context-specific error on failure.

- **void write_header_tabular** (std::ostream &tabular_ostream, const Variables &vars, const Response &response, const std::string &counter_label, unsigned short tabular_format)

  Output the header row (labels) for a tabular data file for variables and responses, with variables in input spec order. Conditionally include interface ID. Primary uses: environment tabular data, pre-run output, surrogate approx evals.

- **void write_header_tabular** (std::ostream &tabular_ostream, const Variables &vars, const StringArray &addtnl_labels, const std::string &counter_label, unsigned short tabular_format)
Output the header row (labels) for a tabular data file for variables and additional labels not tied to a response. Variables are in input spec order. Conditionally include interface ID. Primary uses: MCMC chain export, including calibration sigmas.

- **void `write_leading_columns`** (std::ostream &tabular_ostream, size_t eval_id, const String &iface_id, unsigned short tabular_format)
  
  Write the leading column with eval ID and conditionally, the interface ID.

- **void `write_data_tabular`** (std::ostream &tabular_ostream, const Variables &vars, const String &iface, size_t counter, unsigned short tabular_format)
  
  Output a row of tabular data from a variables object. All active/inactive variables written in input spec order. Conditionally include interface ID. Primary uses: output of sampling sets.

- **void `write_data_tabular`** (std::ostream &tabular_ostream, const Variables &vars, const String &iface, const Response &response, size_t counter, unsigned short tabular_format)
  
  Output a row of tabular data from variables and response objects. All active/inactive variables written in input spec order. Conditionally include interface ID. Primary uses: environment tabular data, pre-run output, surrogate approx evals.

- **void `write_data_tabular`** (const std::string &output_filename, const std::string &context_message, const RealVectorArray &output_coeffs, const UShort2DArray &output_indices)
  
  PCE export: write freeform format file with whitespace-separated data where each row has `num_fns` reals from `coeffs`, followed by `num_vars` unsigned shorts from `indices`.

- **bool `exists_extra_data`** (std::istream &tabular_file)
  
  Check if an input stream contains unexpected additional data.

- **StringArray `read_header_tabular`** (std::istream &input_stream, unsigned short tabular_format)
  
  Read and discard header line from the stream.

- **void `read_leading_columns`** (std::istream &input_stream, unsigned short tabular_format)
  
  Read leading columns [ int eval_id [ String iface_id ] ]

- **void `read_leading_columns`** (std::istream &input_stream, unsigned short tabular_format, int &eval_id, String &iface_id)
  
  Read leading columns [ int eval_id [ String iface_id ] ]

- **void `read_data_tabular`** (const std::string &input_filename, const std::string &context_message, RealVector &input_data, size_t num_entries, unsigned short tabular_format)
  
  Read possibly header-annotated whitespace-separated data into a vector of length `num_entries`; if annotated then it’s a column vector for now.

- **void `read_data_tabular`** (const std::string &input_filename, const std::string &context_message, Variables vars, size_t num_fns, RealMatrix &vars_matrix, RealMatrix &resp_matrix, unsigned short tabular_format, bool verbose=false, bool use_var_labels=false, bool active_only=false)
  
  Tabular read for ApproximationInterface challenge data: read possibly header-annotated whitespace-separated data of possible mixed Variables, followed by `num_fns`, each into RealMatrix with minimal error checking.

- **void `read_data_tabular`** (const std::string &input_filename, const std::string &context_message, RealVectorArray &input_coeffs, UShort2DArray &input_indices, unsigned short tabular_format, size_t num_vars, size_t num_fns)
  
  Tabular read for PCE import: read possibly header-annotated whitespace-separated data of unknown length where each row has `num_fns` reals followed by `num_vars` unsigned shorts; append data to arrays passed by reference.

- **void `read_data_tabular`** (const std::string &input_filename, const std::string &context_message, Variables vars, Response resp, PRPList &input_prp, unsigned short tabular_format, bool verbose=false, bool use_var_labels=false, bool active_only=false)
  
  Tabular read for DataFitSurrModel (build points): read whitespace-separated data with optional row and column headers into lists of Variables and Responses until out of data.
• void \texttt{read\_data\_tabular}(\texttt{const std::string &input\_filename, const std::string &context\_message, RealMatrix &input\_matrix, size\_t record\_len, unsigned short tabular\_format, bool verbose=false})

  \textit{Tabular read for import\_approx\_points\_file: read whitespace-separated data with optional row and column headers into a single matrix, with length of record as specified and number of records to be determined by file content. The matrix is stored as record\_len rows by num\_records columns.}

• void \texttt{read\_data\_tabular}(\texttt{const std::string &input\_filename, const std::string &context\_message, RealMatrix &input\_matrix, size\_t num\_rows, size\_t num\_cols, unsigned short tabular\_format, bool verbose=false})

  \textit{Tabular read for GPMSA data: read whitespace-separated data with optional row and column headers into a single matrix, with size as specified (one experiment per row).}

• size\_t \texttt{read\_data\_tabular}(\texttt{const std::string &input\_filename, const std::string &context\_message, RealVectorArray &cva, IntVectorArray &diva, StringMulti2DArray &dsva, RealVectorArray &drva, unsigned short tabular\_format, bool active\_only, Variables vars})

  \textit{Tabular read for ParamStudy: read specified input data file into arrays with sizes specified by the passed vc\_totals array.}

15.4.1 Detailed Description

Utility functions for reading and writing tabular data files. Emerging utilities for tabular file I/O. For now, just extraction of capability from separate contexts to facilitate rework. These augment (and leverage) those in data\_util.h. Design/capability goals: Ability to read / write data with row/col headers or in free-form Detect premature end of file, report if extra data More consistent and reliable checks for file open errors Require right number of cols in header mode; only total data checking in free-form (likely) Allow comment character for header rows or even in data? variables vs. variables/responses for both read and write Should we support CSV? delimiter = ','; other? Verify treatment of trailing newline without reading a zero Allow reading into the transpose of the data structure

15.5 dll\_tester.cpp File Reference

Test the DLL with a DAKOTA input file.

Functions

• int \texttt{main}(\texttt{int argc, char *argv[]})

  \textit{The main program for exercising the DLL API with a simple command-line.}

15.5.1 Detailed Description

Test the DLL with a DAKOTA input file.

15.6 JEGAOptimizer.cpp File Reference

Contains the implementation of the JEGAOptimizer class.

Classes

• class \texttt{JEGAOptimizer::Evaluator}

  \textit{An evaluator specialization that knows how to interact with Dakota.}
15.7. JEGAOPTIMIZER.HPP FILE REFERENCE

- class JEGAOptimizer::EvaluatorCreator
  A specialization of the JEGA::FrontEnd::EvaluatorCreator that creates a new instance of a Evaluator.
- class JEGAOptimizer::Driver
  A subclass of the JEGA front end driver that exposes the individual protected methods to execute the algorithm.

Namespaces
- Dakota
  The primary namespace for DAKOTA.

Constant Groups
- Dakota
  The primary namespace for DAKOTA.

Functions
- template<typename T>
  string asstring (const T &val)
  Creates a string from the argument val using an ostringstream.

15.6.1 Detailed Description
Contains the implementation of the JEGAOptimizer class.

15.7 JEGAOptimizer.hpp File Reference
Contains the definition of the JEGAOptimizer class.

Classes
- class JEGAOptimizer
  A version of Dakota::Optimizer for instantiation of John Eddy’s Genetic Algorithms (JEGA).
- class JEGATraits
  A version of TraitsBase specialized for John Eddy’s Genetic Algorithms (JEGA).

Namespaces
- Dakota
  The primary namespace for DAKOTA.

Constant Groups
- Dakota
  The primary namespace for DAKOTA.
15.7.1 Detailed Description

Contains the definition of the JEGAOptimizer class.

15.8 library_mode.cpp File Reference

file containing a mock simulator main for testing Dakota in library mode

Classes

• struct callback_data

Functions

• void fpinit_AS ()

• void run_dakota_parse (const char *dakota_input_file)

  Run a Dakota LibraryEnvironment, mode 1: parsing an input file.

• void run_dakota_data ()

  Run a Dakota LibraryEnvironment, mode 2: from C++ API inserted data.

• void run_dakota_mixed (const char *dakota_input_file, bool mpirun_flag)

  Run a Dakota LibraryEnvironment, from string or input file input, supplemented with additional C++ API adjustments.

• void serial_interface_plugin (Dakota::LibraryEnvironment &env)

  Convenience function with simplest example of interface plugin: plugin a serial DirectApplicInterface that can be constructed independent of Dakota’s configuration details.

• void parallel_interface_plugin (Dakota::LibraryEnvironment &env)

  Convenience function to plug a library client’s interface into the appropriate model, demonstrating use of Dakota parallel configuration in constructing the plugin Interface on the right MPI_Comm.

• static void callback_function (Dakota::ProblemDescDB *db, void *ptr)

  Example: user-provided post-parse callback (Dakota::DbCallbackFunction)

• int main (int argc, char *argv[ ])

  A mock simulator main for testing Dakota in library mode.

Variables

• static const char serial_input []

  Default Dakota input string for serial case (rosenbrock):

• static const char parallel_input []

  Default Dakota input string for parallel case (text_book)

15.8.1 Detailed Description

file containing a mock simulator main for testing Dakota in library mode
15.8.2 Function Documentation

void fpinit_ASL ( )

Floating-point initialization from AMPL: switch to 53-bit rounding if appropriate, to eliminate some cross-
platform differences.
   Referenced by main().

void run_dakota_parse ( const char * dakota_input_file )

Run a Dakota LibraryEnvironment, mode 1: parsing an input file.
   Simplest library case: this function parses from an input file to define the ProblemDescDB data.
   References Environment::execute(), ProgramOptions::input_file(), Environment::mpi_managet(), MPIManager-:
   ::mpirun_flag(), parallel_interface_plugin(), serial_interface_plugin(), and MPIManager::world_rank().
   Referenced by main().

void run_dakota_data ( )

Run a Dakota LibraryEnvironment, mode 2: from C++ API inserted data.
   Rather than parsing from an input file, this function populates Data class objects directly using a minimal
   specification and relies on constructor defaults and post-processing in post_process() to fill in the rest.
   References DataInterface::data_rep(), DataResponses::data_rep(), DataVariables::data_rep(), DataMethod::data-
   rep(), LibraryEnvironment::done_modifying_db(), Environment::execute(), Environment::exit_mode(), DataResponses-
   Rep::gradientType, DataResponsesRep::hessianType, LibraryEnvironment::insert_nodes(), DataMethodRep::method-
   Name, Environment::mpi_manager(), MPIManager::mpirun_flag(), ParallelLibrary::mpirun_flag(), DataVariables-
   Rep::numContinuousDesVars, DataResponsesRep::numNonlinearIneqConstraints, DataResponsesRep::numObjective-
   Functions, parallel_interface_plugin(), Environment::parallel_library(), serial_interface_plugin(), and ParallelLibrary-
   ::world_rank().
   Referenced by main().

void run_dakota_mixed ( const char * dakota_input_file, bool mpirun_flag )

Run a Dakota LibraryEnvironment, from string or input file input, supplemented with additional C++ API adjust-
ments.
   Function to encapsulate the Dakota object instantiations for mode 3: mixed parsing and direct updating.
   This function showcases multiple features. For parsing, either an input file (dakota_input_file != NULL) or a
   default input string (dakota_input_file == NULL) are shown. This parsed input is then mixed with input from three
   sources: (1) input from a user-supplied callback function, (2) updates to the DB prior to Environment instantiation,
   (3) updates directly to Iterators/Models following Environment instantiation.
   References callback_function(), LibraryEnvironment::done_modifying_db(), ProgramOptions::echo_input(), Environment-
   ::execute(), LibraryEnvironment::filtered_model_list(), ProblemDescDB::get_sa(), ProgramOptions::input_file(),
   ProgramOptions::input_string(), Environment::mpi_manager(), MPIManager::mpirun_flag(), parallel_input, parallel-
   _interface_plugin(), Environment::parallel_library(), Environment::problem_description_db(), ProblemDescDB::resolve-
   _top_method(), callback_data::rosen_cdv_upper_bd, serial_input, serial_interface_plugin(), ProblemDescDB::set(),
   and ParallelLibrary::world_rank().
   Referenced by main().

void serial_interface_plugin ( Dakota::LibraryEnvironment & env )

Convenience function with simplest example of interface plugin: plugin a serial DirectApplicInterface that can be
constructed independent of Dakota’s configuration details.
Demonstration of simple plugin where client code doesn’t require access to detailed Dakota data (such as Model-based parallel configuration information) to construct the DirectApplicInterface. This example plugs-in a derived serial direct application interface instance (“plugin_rosenbrock”).

References Dakota::abort_handler(), LibraryEnvironment::plugin_interface(), and Environment::problem_description_db().

Referenced by run_dakota_data(), run_dakota_mixed(), and run_dakota_parse().

```c
void parallel_interface_plugin ( Dakota::LibraryEnvironment & env )
```

Convenience function to plug a library client’s interface into the appropriate model, demonstrating use of Dakota parallel configuration in constructing the plugin Interface on the right MPI Comm.

From a filtered list of Model candidates, plug-in a derived direct application interface instance (“plugin_textbook” for parallel). This approach provides more complete access to the Model, e.g., for access to analysis communicators.

References Dakota::abort_handler(), Interface::assign_rep(), LibraryEnvironment::filtered_model_list(), ProblemDescDB::get_db_model_node(), Environment::problem_description_db(), and ProblemDescDB::set_db_model_nodes().

Referenced by run_dakota_data(), run_dakota_mixed(), and run_dakota_parse().

```c
static void callback_function ( Dakota::ProblemDescDB * db, void * ptr ) [static]
```

Example: user-provided post-parse callback (Dakota::DbCallbackFunction)

Example of user-provided callback function (an instance of Dakota::DbCallbackFunction) to override input provided by parsed Dakota input file or input string data.

References Dakota::contains(), ProblemDescDB::get_sa(), ProblemDescDB::get_ushort(), ProblemDescDB::resolve_top_method(), callback_data::rosen_cdv_upper_bd, and ProblemDescDB::set().

Referenced by run_dakota_mixed().

```c
int main ( int argc, char * argv[] )
```

A mock simulator main for testing Dakota in library mode.

Overall Usage: dakota_library_mode [-mixed] [dakota.in]

Uses alternative instantiation syntax as described in the library mode documentation within the Developers Manual. Tests several problem specification modes:

1. run_dakota_parse: reads all problem specification data from a Dakota input file. Usage: dakota_library_mode dakota.in
2. run_dakota_data: creates all problem specification from direct Data instance instantiations in the C++ code. Usage: dakota_library_mode
3. run_dakota_mixed: a mixture of input parsing and direct data updates, where the data updates occur: (a) via the DB during Environment instantiation, and (b) via Iterators/Models following Environment instantiation. Usage: dakota_library_mode -mixed (input from default string) dakota_library_mode -mixed dakota.in (input from specified file)

Serial cases use a plugin rosenbrock model, while parallel cases use textbook.

References MPIManager::detect_parallel_launch(), fpinit_ASL(), Dakota::mpi_debug_hold(), run_dakota_data(), run_dakota_mixed(), and run_dakota_parse().

### 15.8.3 Variable Documentation

```c
const char serial_input[ ] [static]
```

Initial value:
15.9. **LIBRARY_SPLIT.CPP FILE REFERENCE**

```cpp
= "method," optpp_q_newton" 
= max_iterations = 50* 
= convergence_tolerance = 1e-4* 
= variables,* 
= continuous_design = 2* 
= descriptors 'x1' 'x2'* 
= interface,* 
= direct* 
= analysis_driver = 'plugin_robenbrock'" 
= responses,* 
= num_objective_functions = 1* 
= analytic_gradients" 
= no_hessians"
```

Default Dakota input string for serial case (rosenbrock):
Referenced by run_dakota_mixed().

```cpp
const char parallel_input[] [static]
```

Initial value:

```cpp
= "method," optpp_q_newton" 
= max_iterations = 50* 
= convergence_tolerance = 1e-4* 
= variables,* 
= continuous_design = 2* 
= descriptors 'x1' 'x2'* 
= interface,* 
= direct* 
= analysis_driver = 'plugin_text_book'" 
= responses,* 
= num_objective_functions = 1* 
= analytic_gradients" 
= no_hessians"
```

Default Dakota input string for parallel case (text_book)
Referenced by run_dakota_mixed().

---

15.9 **library_split.cpp File Reference**

file containing a mock simulator main for testing DAKOTA in library mode on a split communicator

**Functions**

- void manage_mpi (MPI_Comm &my_comm, int &color)
  
  Split MPI_COMM_WORLD, returning the comm and color.

- void gen_dakota_input (const int &color, std::string &input)
  
  Return the appropriate DAKOTA input based on color (1 or 2)

- void run_dakota (const MPI_Comm &comm, const std::string &input, const int &color)
  
  Launch DAKOTA on passed communicator, tagging output/error with color.

- void collect_results ()
  
  Wait for and collect results from DAKOTA runs.

- int main (int argc, char *argv[])
  
  Driver routine for testing library mode with partitioned MPI_Comm. This test fixture requires MPI and can be run on 3–8 processors.
15.9.1 Detailed Description

file containing a mock simulator main for testing DAKOTA in library mode on a split communicator

15.10 main.cpp File Reference

file containing the main program for DAKOTA

Functions

- void fpinit_ASLS()
- int main (int argc, char *argv[])

The main DAKOTA program.

15.10.1 Detailed Description

file containing the main program for DAKOTA

15.10.2 Function Documentation

void fpinit_ASLS()

Floating-point initialization from AMPL: switch to 53-bit rounding if appropriate, to eliminate some cross-platform differences.

int main ( int argc, char *argv[])

The main DAKOTA program.

  Manage command line inputs, input files, restart file(s), output streams, and top level parallel iterator communicators. Instantiate the ExecutableEnvironment and invoke its execute() virtual function.

  References Environment::check(), ExecutableEnvironment::execute(), fpinit_ASLS(), Dakota::mpi_debug_hold(), and Dakota::register_signal_handlers().

15.11 QUESOImpl.hpp File Reference

Classes

- class QuesoJointPdf< V, M >
  
  Dakota specialization of QUESO generic joint PDF.

- class QuesoVectorRV< V, M >
  
  Dakota specialization of QUESO vector-valued random variable.

- class DerivInformedPropCovTK< V, M >
  
  Dakota transition kernel that updates proposal covariance based on derivatives (for random walk case)

- class DerivInformedPropCovLogitTK< V, M >
  
  Dakota transition kernel that updates proposal covariance based on derivatives (for logit random walk case)

- class TKFactoryDIPC
  
  Custom RW TKfactory: passes Dakota QUESO instance pointer to the TK at build.

- class TKFactoryDIPCLogit
  
  Custom Logit RW TKfactory: passed Dakota QUESO instance pointer to the TK at build.
Namespaces

- Dakota
  
  *The primary namespace for DAKOTA.*

Constant Groups

- Dakota
  
  *The primary namespace for DAKOTA.*

15.11.1 Detailed Description

QUESO specializations for evaluations and utilities

15.12 restart_util.cpp File Reference

file containing the DAKOTA restart utility main program

Namespaces

- Dakota
  
  *The primary namespace for DAKOTA.*

Constant Groups

- Dakota
  
  *The primary namespace for DAKOTA.*

Functions

- void print_usage (std::ostream &s)
  
  *print restart utility help message*
- void print_restart (StringArray pos_args, String print_dest)
  
  *print a restart file*
- void print_restart_pdb (StringArray pos_args, String print_dest)
  
  *print a restart file (PDB format)*
- void print_restart_tabular (StringArray pos_args, String print_dest, unsigned short tabular_format, int tabular_precision)
  
  *print a restart file (tabular format)*
- void read_neutral (StringArray pos_args)
  
  *read a restart file (neutral file format)*
- void repair_restart (StringArray pos_args, String identifier_type)
  
  *repair a restart file by removing corrupted evaluations*
- void concatenate_restart (StringArray pos_args)
  
  *concatenate multiple restart files*
- int main (int argc, char *argv[])
  
  *The main program for the DAKOTA restart utility.*
15.12.1 Detailed Description

file containing the DAKOTA restart utility main program

15.12.2 Function Documentation

int main ( int argc, char * argv[])

The main program for the DAKOTA restart utility.

- Parse command line inputs and invoke the appropriate utility function (print_restart(), print_restart_tabular(), read_neutral(), repair_restart(), or concatenate_restart()).

- References Dakota::concatenate_restart(), Dakota::print_restart(), Dakota::print_restart_pdb(), Dakota::print_restart_tabular(), Dakota::print_usage(), Dakota::read_neutral(), Dakota::repair_restart(), and Dakota::write_precision.
Bibliography

Index

~Approximation
  Dakota::Approximation, 237
~Constraints
  Dakota::Constraints, 297
~DataFitSurrModel
  Dakota::DataFitSurrModel, 319
~Environment
  Dakota::Environment, 411
~Interface
  Dakota::Interface, 481
~Iterator
  Dakota::Iterator, 492
~Model
  Dakota::Model, 560
~NonDAdaptiveSampling
  Dakota::NonDAdaptiveSampling, 621
~NonDMultilevelFunctionTrain
  Dakota::NonDMultilevelFunctionTrain, 704
~ProblemDescDB
  Dakota::ProblemDescDB, 847
~SharedApproxData
  Dakota::SharedApproxData, 941
_initPts
  Dakota::JEGAOptimizer, 510
_model
  Dakota::JEGAOptimizer::Evaluator, 417

A
  Dakota::CONMINOptimizer, 287
APPSEvalMgr, 246
  Dakota::APPSEvalMgr, 247
APPSOptimizer, 248
abmInstance
  Dakota::AdaptedBasisModel, 206
abort_handler_t
  Dakota, 164
abort_mode
  Dakota, 174
abort_throw_or_exit
  Dakota, 164
acceptanceChain
  Dakota::NonDBayesCalibration, 633
  accepts_multiple_points
    Dakota::JEGAOptimizer, 509
active_set_mapping
  Dakota::NonDSampling, 759
active_variables
  Dakota::Model, 567
ActiveSet, 193
ActiveSubspaceModel, 195
activeVarsCompsTotals
  Dakota::SharedVariablesDataRep, 969
actualModel
  Dakota::DataFitSurrModel, 326
actualReducedRank
  Dakota::RandomFieldModel, 878
AdaptedBasisModel, 202
add_array
  Dakota::Approximation, 238
add_constraints_to_surfdata
  Dakota::SurfpackApproximation, 998
add_datapoint
  Dakota::Graphics, 458
  Dakota::OutputManager, 793
AddAttributeVisitor, 206
algorithm_space_model
  Dakota::Analyzer, 213
  Dakota::EffGlobalMinimizer, 405
  Dakota::Minimizer, 529
  Dakota::NonDBayesCalibration, 632
  Dakota::NonDExpansion, 653
  Dakota::NonDGlobalInterval, 660
  Dakota::NonDReliability, 747
allContinuousIds
  Dakota::SharedVariablesDataRep, 969
allocationTarget
  Dakota::NonDMultilevelSampling, 720
analytic_delta_level_mappings
  Dakota::NonDStochCollocation, 766
Analyzer, 207

1086
anisotropic_order_to_dimension_preference
  Dakota::NonDIntegration, 675
append_approximation
  Dakota::ApproximationInterface, 243, 244
  Dakota::DataFitSurrModel, 322
append_columns
  dakota::util, 190
  ApplicationInterface, 215
apply_linear_constraints
  Dakota, 167
apply_matrix_partial
  Dakota, 163
apply_matrix_transpose_partial
  Dakota, 163
apply_nonlinear_constraints
  Dakota, 167
approx_subprob_constraint_eval
  Dakota::SurrBasedLocalMinimizer, 1007
approx_subprob_objective_eval
  Dakota::SurrBasedLocalMinimizer, 1006
approxBuilds
  Dakota::SurrogateModel, 1019
Approximation, 231
  Dakota::Approximation, 237, 238
ApproximationInterface, 239
AppsTraits, 251
ApreproWriter, 252
archive_best_config_variables
  Dakota::DataTransformModel, 369
archive_sobol_indices
  Dakota::Analyzer, 214
array_insert
  Dakota::ResultsDBBase, 907
asmInstance
  Dakota::ActiveSubspaceModel, 202
assign_rep
  Dakota::Interface, 481
  Dakota::Iterator, 496
  Dakota::Model, 566
asstring
  Dakota, 171
asynchronous_local_analyses
  Dakota::ProcessHandleApplicInterface, 857
asynchronous_local_evaluations
  Dakota::ApplicationInterface, 227
asynchronous_local_evaluations_nowait
  Dakota::ApplicationInterface, 229
AttachScaleVisitor, 253
augmented_lagrangian_merit
  Dakota::SurrBasedMinimizer, 1011
autotag_files
  Dakota::ProcessApplicInterface, 853
B
  Dakota::CONMINOptimizer, 287
  barnes
    Dakota::TestDriverInterface, 1036
  barnes_lf
    Dakota::TestDriverInterface, 1036
  BaseConstructor, 253
bootstrapRNG
  Dakota::BootstrapSamplerBase, 257
  BootstrapSampler< Data >, 254
  BootstrapSampler< Teuchos::SerialDenseMatrix< OrdinalType, ScalarType > >, 255
  BootstrapSamplerBase< Data >, 256
  BootstrapSamplerWithGS< Data, Getter, Setter >, 257
brent_minimize
  Dakota::NonlinearCGOptimizer, 773
build
  Dakota::Approximation, 238
  Dakota::C3Approximation, 262
  Dakota::PecosApproximation, 829
  Dakota::QMEApproximation, 872
  Dakota::SurfpackApproximation, 997
dakota::surrogates::GaussianProcess, 441
dakota::surrogates::PolynomialRegression, 831
dakota::surrogates::Surrogate, 1013
dakota::surrogates::TANA3Approximation, 1029
dakota::surrogates::TaylorApproximation, 1031
build_approximation
  Dakota::ApproximationInterface, 244
  Dakota::DataFitSurrModel, 320
build_cv_surrogate
  Dakota::ActiveSubspaceModel, 200
build_global
  Dakota::DataFitSurrModel, 324
build_gradient_of_sum_square_residuals_from_function_data
  Dakota::ExperimentData, 427
build_hessian_of_sum_square_residuals_from_function_data
  Dakota::ExperimentData, 427
build_local_multipoint
  Dakota::DataFitSurrModel, 324
build_surrogate
  Dakota::ActiveSubspaceModel, 201
buildDataOrder
  Dakota::SharedApproxData, 942
INDEX

C
  Dakota::CONMINOptimizer, 287
C3Approximation, 258
C3FnTrainPtrs, 263
CAUVLbl
  Dakota, 175
CEUVLbl
  Dakota, 176
COLINApplication, 266
COLINOptimizer, 269
  Dakota::COLINOptimizer, 271
COLINTraits, 273
CONMINOptimizer, 281
CONMINTraits, 288
CT
  Dakota::CONMINOptimizer, 286
cache
  acceptance_chain
    Dakota::NonDGPMSABayesCalibration, 672
cache_chain
  Dakota::NonDDREAMBayesCalibration, 643
  Dakota::NonDMUQBayesCalibration, 726
  Dakota::NonDQUESTOBayesCalibration, 743
calibrate
  Dakota::NonDDREAMBayesCalibration, 643
  Dakota::NonDMUQBayesCalibration, 725
  Dakota::NonDQUESTOBayesCalibration, 743
  Dakota::NonDWASABIBayesCalibration, 770
callback_data, 264
callback_function
  library_mode.cpp, 1080
check_and_broadcast
  Dakota::ProblemDescDB, 848
check_driver
  Dakota::NIDRProblemDescDB, 595
check_for_zero_scaler_factor
  dakota::util::DataScaler, 362
check_input
  Dakota::ProblemDescDB, 848
check_wait
  Dakota::ProcessHandleApplicInterface, 857
CholeskySolver, 264
clear
  Dakota::ParallelLevel, 800
clear_current_active_data
  Dakota::Approximation, 238
  Dakota::QMEApproximation, 872
  Dakota::TANA3Approximation, 1029
clear_data
  Dakota::Approximation, 239
Clone
  Dakota::JEGAOptimizer::Evaluator, 416
colin_cache_lookup
  Dakota::COLINOptimizer, 273
colin_request_to_dakota_request
  Dakota::COLINApplication, 268
CollabHybridMetaIterator, 274
collect_evaluation_impl
  Dakota::COLINApplication, 268
CommandLineHandler, 276
CommandShell, 277
calculate
  Dakota::DiscrepancyCorrection, 398
calculate_basis_matrix
  dakota::surrogates::PolynomialRegression, 831
calculate_correlations
  Dakota::SensAnalysisGlobal, 930
calculate_covariance_metric
  Dakota::NonDExpansion, 654
  Dakota::NonDStochCollocation, 766
calculate_densities
  Dakota::NonD, 611
calculate_first_deriv_pred_gram
  dakota::surrogates::GaussianProcess, 444
calculate_gram
  dakota::surrogates::GaussianProcess, 444
calculate_hyperbolic_indices
  dakota::surrogates, 182
calculate_hyperbolic_level_indices
  dakota::surrogates, 183
calculate_hyperbolic_subdim_level_indices
  dakota::surrogates, 183
calculate_level_mappings
  Dakota::NonDSampling, 757
calculate_level_mappings_metric
  Dakota::NonDExpansion, 654
  Dakota::NonDStochCollocation, 766
calculate_next_combination
  dakota::surrogates, 182
calculate_pred_dists
  dakota::surrogates::GaussianProcess, 444
calculate_reduced_indices
  dakota::surrogates, 184
calculate_second_deriv_pred_gram
  dakota::surrogates::GaussianProcess, 444
calculate_statistics
  Dakota::NonDExpansion, 654
calculate_trust_region_ratio
Dakota::SurrBasedLocalMinimizer, 1005
compute_vbd_stats
Dakota::Analyzer, 214
concatenate_restart
Dakota, 173
ConcurrentMetaIterator, 278
configure
Dakota::IteratorScheduler, 503
configure_constraint_maps
Dakota::Optimizer, 789
configure_equality_constraint_maps
Dakota, 167
configure_inequality_constraint_maps
Dakota, 167
conminInfo
Dakota::CONMINOptimizer, 285
ConsoleRedirector, 289
constraint0_evaluator
Dakota::SNLLOptimizer, 986
constraint1_evaluator
Dakota::SNLLOptimizer, 986
constraint1_evaluator_forward
Dakota::SNLLLeastSq, 978
constraint2_evaluator
Dakota::SNLLOptimizer, 986
constraint2_evaluator_forward
Dakota::SNLLLeastSq, 979
constraint_violation
Dakota::COLINOptimizer, 273
Dakota::SurrBasedMinimizer, 1012
constraintValues
Dakota::CONMINOptimizer, 285
Constraints, 290
Dakota::Constraints, 297
contains
Dakota, 174
covariance_variance
Dakota::NonDMultilevelSampling, 718
converge_order
Dakota::RichExtrapVerification, 916
converge_qoi
Dakota::RichExtrapVerification, 916
copy
Dakota::Constraints, 298
Dakota::SharedResponseData, 952
Dakota::SharedVariablesData, 964
copy_field_data
Dakota, 170
copy_items
Dakota::WorkdirHelper, 1068
copy_rep
Dakota::SharedResponseDataRep, 954
Dakota::SharedVariablesDataRep, 969
core_run
Dakota::APPSOptimizer, 250
Dakota::COLINOptimizer, 271
Dakota::CONMINOptimizer, 285
Dakota::DDACEDesignCompExp, 388
Dakota::FSUDesignCompExp, 436
Dakota::Iterator, 494
Dakota::JEGAOptimizer, 509
Dakota::NCSUOptimizer, 580
Dakota::NL2SOLLeastSq, 598
Dakota::NLSSOLLeastSq, 602
Dakota::NonDAAdaptiveSampling, 621
Dakota::NonDBayesCalibration, 631
Dakota::NonDGGlobalReliability, 663
Dakota::NonDGPlmpSampling, 667
Dakota::NonDIIntegration, 675
Dakota::NonDLHSSampling, 684
Dakota::NonDLHSSamplingSampling, 695
Dakota::NonDMultilevelSampling, 717
Dakota::NonDKDDarts, 749
Dakota::NonDSampling, 758
Dakota::NonlinearCGOptimizer, 773
Dakota::NOWPACOptimizer, 778
Dakota::ParamStudy, 818
Dakota::PSUADEDesignCompExp, 867
Dakota::RichExtrapVerification, 916
Dakota::SurrBasedLocalMinimizer, 1005
cov_determinant
Dakota::ExperimentData, 427
create_command_arguments
Dakota::ProcessHandleApplicInterface, 857
create_directory
Dakota::WorkdirHelper, 1068
create_evaluation_process
Dakota::ProcessHandleApplicInterface, 856
create_plots_2d
Dakota::Graphics, 458
create_tabular_datastream
Dakota::OutputManager, 793
CreateEvaluator
Dakota::JEGAOptimizer::EvaluatorCreator, 420
cv_scaled2native
Dakota::ScalingModel, 922
d_optimal_parameter_set
Dakota::NonDLHSSampling, 684
generate_system_seed, 165
get_bounds, 166
get_initial_values, 166
get_linear_constraints, 167
get_mixed_bounds, 166
get_nonlinear_bounds, 169
get_nonlinear_eq_constraints, 168, 169
get_nonlinear_ineq_constraints, 169
get_pathext, 174
get_responses, 168
get_variable_bounds, 166
get_variables, 168
getRmax, 170
id_vars_exact_compare, 172
lookup_by_val, 172
mindist, 170
mindistindx, 170
mpi_debug_hold, 164
NUMBER_OF_FIELDS, 175
operator==, 166, 171
PRPMultiIndexCache, 163
PRPMultiIndexQueue, 163
perform_analysis, 171
print_restart, 172
print_restart_pdb, 173
print_restart_tabular, 173
qr, 165
qr_rsolve, 165
read_neutral, 173
register_signal_handlers, 164
repair_restart, 173
SCL_FIELD_NAMES, 181
SCL_NUMBER_OF_FIELDS, 181
SUBMETHOD_COLLABORATIVE, 163
set_best_responses, 168
set_compare, 171
set_variables, 168
start_dakota_heartbeat, 171
start_grid_computing, 171
stop_grid_computing, 171
submethod_map, 174
svd, 164
symmetric_eigenvalue_decomposition, 170
VLUncertainInt, 178
VLUncertainReal, 177
VLUncertainStr, 178
var_mp_cbound, 180
var_mp_check_cau, 179
INDEX

var_mp_check_ceu, 180
var_mp_check_cv, 178
var_mp_check_daui, 179
var_mp_check_daur, 179
var_mp_check_daus, 179
var_mp_check_deui, 180
var_mp_check_deur, 180
var_mp_check_deus, 180
var_mp_check_dset, 179
var_mp_drange, 180
write_ordered, 169
dakota, 87
Dakota::APPSEvalMgr
  APPSEvalMgr, 247
  isReadyForWork, 248
  recv, 248
  submit, 248
Dakota::APPSOptimizer
  core_run, 250
  initialize_run, 250
  initialize_variables_and_constraints, 251
  set_apps_parameters, 250
Dakota::ActiveSet
  derivVarsVector, 195
  requestVector, 195
Dakota::ActiveSubspaceModel
  ActiveSubspaceModel, 200
  asmInstance, 202
  build_cv_surrogate, 200
  build_surrogate, 201
  derived_init_communicators, 200
  initialize_mapping, 200
  initialize_recast, 201
  response_mapping, 202
  set_mapping, 201
  uncertain_vvars_to_subspace, 201
  vars_mapping, 201
Dakota::AdaptedBasisModel
  abmInstance, 206
  derived_init_communicators, 205
  initialize_recast, 205
  response_mapping, 206
  set_mapping, 206
  uncertain_vvars_to_subspace, 205
  vars_mapping, 205
Dakota::Analyzer
  algorithm_space_model, 213
  archive_sobol_indices, 214
  compute_vbd_stats, 214
  evaluate_parameter_sets, 213
  finalize_run, 212
  get_vbd_parameter_sets, 213
  initialize_run, 211
  num_samples, 211
  post_run, 212
  pre_output, 212
  pre_run, 211
  print_results, 213
  print_sobol_indices, 214
  read_variables_responses, 214
  sample_to_variables, 211
  variables_to_sample, 215
Dakota::ApplicationInterface
  asynchronous_local_evaluations, 227
  asynchronous_local_evaluations_nowait, 229
  duplication_detect, 225
  init_communicators_checks, 225
  init_default_asv, 226
  init_serial, 223
  map, 223
  master_dynamic_schedule_analyses, 225
  master_dynamic_schedule_evaluations, 226
  master_dynamic_schedule_evaluations_nowait, 228
  peer_dynamic_schedule_evaluations, 227
  peer_dynamic_schedule_evaluations_nowait, 229
  peer_static_schedule_evaluations, 226
  peer_static_schedule_evaluations_nowait, 228
  serve_analyses_synch, 225
  serve_evaluations, 224
  serve_evaluations_asynch, 230
  serve_evaluations_asynch_peer, 230
  serve_evaluations_synch, 229
  serve_evaluations_synch_peer, 230
  set_communicators_checks, 225
  stop_evaluation_servers, 224
  synchronize, 223
  synchronize_nowait, 224
  synchronous_local_evaluations, 228
Dakota::Approximation
  ~Approximation, 237
  add_array, 238
  Approximation, 237, 238
  build, 238
  clear_current_active_data, 238
  clear_data, 239
  get_approx, 239
  operator=, 238
Dakota::ApproximationInterface
append_approximation, 243, 244
build_approximation, 244
export_approximation, 244
functionSurfaces, 245
pop_approximation, 245
push_approximation, 245
read_challenge_points, 245
rebuild_approximation, 244
restore_data_key, 245
update_approximation, 243
Dakota::BootstrapSamplerBase bootstrapRNG, 257
Dakota::C3Approximation build, 262
regression_size, 262
Dakota::COLINApplication colin_request_to_dakota_request, 268
collect_evaluation_impl, 268
dakota_response_to_colin_response, 268
evaluation_available, 268
map_domain, 269
perform_evaluation_impl, 268
set_problem, 267
spawn_evaluation_impl, 267
Dakota::COLINOptimizer COLINOptimizer, 271
colin_cache_lookup, 273
constraintViolation, 273
core_run, 271
post_run, 272
returns_multiple_points, 271
set_rng, 272
set_solver_parameters, 272
solver_setup, 272
Dakota::CONMINOptimizer A, 287
B, 287
C, 287
CT, 286
conminInfo, 285
constraintValues, 285
core_run, 285
DF, 287
G1, 286
G2, 286
IC, 288
ISC, 287
MS1, 287
N1, 285
N2, 286
N3, 286
N4, 286
N5, 286
printControl, 285
S, 286
SCAL, 287
Dakota::CommandLineHandler output_helper, 277
Dakota::CommandShell flush, 278
operator<<, 278
Dakota::ConcurrentMetaIterator pre_run, 281
print_results, 281
Dakota::Constraints ~Constraints, 297
Constraints, 297
copy, 298
get_constraints, 299
manage_linear_constraints, 298
operator=, 298
reshape, 298
shape, 298
Dakota::DDACEDesignCompExp core_run, 388
DDACEDesignCompExp, 387
num_samples, 388
post_run, 388
pre_run, 387
resolve_samples_symbols, 388
Dakota::DakotaROLEqConstraints DakotaROLEqConstraints, 300
Dakota::DakotaROLEqConstraintsGrad DakotaROLEqConstraintsGrad, 301
Dakota::DakotaROLEqConstraintsHess DakotaROLEqConstraintsHess, 302
Dakota::DakotaROLIneqConstraints DakotaROLIneqConstraints, 303
Dakota::DakotaROLIneqConstraintsGrad DakotaROLIneqConstraintsGrad, 304
Dakota::DakotaROLIneqConstraintsHess DakotaROLIneqConstraintsHess, 304
Dakota::DakotaROLObjective DakotaROLObjective, 305
Dakota::DakotaROLObjectiveGrad DakotaROLObjectiveGrad, 306
Dakota::DakotaROLObjectiveHess DakotaROLObjectiveHess, 307
Dakota::DataFitSurfModel
- DataFitSurfModel, 319
- actualModel, 326
- append_approximation, 322
- build_approximation, 320
- build_global, 324
- build_local_multipoint, 324
derived_evaluate, 319
derived_evaluate_nowait, 319
derived_synthesize, 319
derived_synthesize_nowait, 320
extport_point, 324
finalize_export, 323
finalize_mapping, 319
import_points, 323
init_model, 325
initialize_export, 323
rebuild_approximation, 320
rebuild_global, 324
update_approximation, 321
update_from_model, 326
update_model, 325
Dakota::DataTransformModel
- archive_best_config_variables, 369
- DataTransformModel, 366
derived_evaluate, 367
derived_evaluate_nowait, 367
derived_synthesize, 367
derived_synthesize_nowait, 367
dtModelInstance, 369
expand_primary_array, 368
filter_submodel_responses, 368
init_continuous_vars, 368
set_mapping, 368
transform_response_map, 368
update_from_subordinate_model, 366
variables_expand, 366
Dakota::DirectApplicInterface
- derived_map_ac, 394
- init_communicators_checks, 394
- set_communicators_checks, 394
- synchronous_local_analysis, 394
Dakota::DiscrepancyCorrection
- compute, 398
Dakota::EffGlobalMinimizer
- algorithm_space_model, 405
- EIF_objective_eval, 405
- get_best_sample, 405
Dakota::Environment
- Environment, 411
- exit_mode, 411
- get_environment, 412
- operator=, 411
- parse, 412
Dakota::ExperimentData
- build_gradient_of_sum_square_residuals_from_function_data, 427
- build_hessian_of_sum_square_residuals_from_function_data, 427
cov_determinant, 427
form_residuals, 426
half_log_cov_det_gradient, 428
half_log_cov_det_hessian, 428
half_log_cov_determinant, 427
load_experiment, 428
parse_sigma_types, 428
recover_model, 426
residuals_per_multipliers, 428
scale_residuals, 427
Dakota::FSUDesignCompExp
- core_run, 436
- enforce_input_rules, 436
- FSUDesignCompExp, 435
- num_samples, 436
- post_run, 436
- pre_run, 436
Dakota::GaussProcApproximation
- GPmodel_apply, 451
- GaussProcApproximation, 451
trendOrder, 451
Dakota::GetLongOpt
- enroll, 454
- GetLongOpt, 454
- MandatoryValue, 453
- OptType, 453
- OptionalValue, 453
- parse, 454
- retrieve, 454
- usage, 455
- Valueless, 453
Dakota::Graphics
- add_datapoint, 458
- create_plots_2d, 458
- new_dataset, 459
Dakota::GridApplicInterface
- synchronous_local_analysis, 460
Dakota::HDF5IOHelper
   read_vector, 466
   set_scalar, 465
   set_vector_scalar_field, 465
   set_vector_vector_field, 465, 466

Dakota::HierarchSurrModel
   derived_evaluate, 471
   derived_evaluate_nowait, 472
   derived_synchronize, 472
   derived_synchronize_nowait, 472
   finalize_mapping, 471
   initialize_mapping, 471

Dakota::Interface
   ~Interface, 481
   assign_rep, 481
   eval_tag_prefix, 482
   get_interface, 482
   Interface, 481
   no_spec_id, 483
   operator=, 481
   rawResponseMap, 483
   response_mapping, 482
   user_auto_id, 483

Dakota::Iterator
   ~Iterator, 492
   assign_rep, 496
   core_run, 494
   eval_tag_prefix, 496
   finalize_run, 494
   get_iterator, 497
   gnewton_set_recast, 496
   initialize_graphics, 495
   initialize_run, 493
   Iterator, 491–493
   maxEvalConcurrency, 498
   no_spec_id, 497
   operator=, 493
   post_run, 494
   pre_run, 493
   print_results, 495
   probDescDB, 498
   run, 495
   user_auto_id, 497
   uses_method, 495

Dakota::IteratorScheduler
   configure, 503
   free_iterator, 503
   init_iterator, 502
   IteratorScheduler, 502
   master_dynamic_schedule_iterators, 504
   partition, 504
   run_iterator, 503
   schedule_iterators, 504
   serve_iterators, 504
   set_iterator, 502

Dakota::JEGAOptimizer
   _initPts, 510
   accepts_multiple_points, 509
   core_run, 509
   GetBestMOSolutions, 509
   GetBestSOSolutions, 509
   GetBestSolutions, 508
   initial_points, 510
   JEGAOptimizer, 507
   LoadAlgorithmConfig, 507
   LoadDakotaResponses, 507
   LoadProblemConfig, 508
   LoadTheConstraints, 508
   LoadTheDesignVariables, 508
   LoadTheObjectiveFunctions, 508
   LoadTheParameterDatabase, 507
   returns_multiple_points, 510
   ToDoubleMatrix, 509

Dakota::JEGAOptimizer::Driver
   DestroyAlgorithm, 402
   Driver, 401
   ExtractAllData, 401
   PerformIterations, 402

Dakota::JEGAOptimizer::Evaluator
   _model, 417
   Clone, 416
   Description, 414
   Evaluate, 415, 416
   Evaluator, 413, 414
   GetDescription, 416
   GetName, 416
   GetNumberLinearConstraints, 415
   GetNumberNonLinearConstraints, 415
   Name, 414
   RecordResponses, 415
   SeparateVariables, 415

Dakota::JEGAOptimizer::EvaluatorCreator
   CreateEvaluator, 420
   EvaluatorCreator, 419

Dakota::LabelsWriter
   operator(), 512

Dakota::LeastSq
   finalize_run, 514
get_confidence_intervals, 515
initialize_run, 514
LeastSq, 514
post_run, 514
print_results, 515
weight_model, 515
Dakota::LibraryEnvironment
    filtered_interface_list, 517
    filtered_model_list, 517
    LibraryEnvironment, 517
Dakota::MatlabInterface
derived_map_ac, 521
matlab_engine_run, 521
Dakota::MetaIterator
    post_run, 523
Dakota::Minimizer
    algorithm_space_model, 529
data_transform_model, 529
finalize_run, 529
initialize_run, 528
local_recast_retrieve, 531
Minimizer, 528
    objective, 530
    objective_gradient, 530
    objective_hessian, 530
    post_run, 529
    resize_bestResp_array, 531
    resize_bestVars_array, 531
    scale_model, 530
Dakota::MixedVarConstraints
    MixedVarConstraints, 532
Dakota::MixedVariables
    MixedVariables, 534
    read_core, 534
    read_tabular, 534
Dakota::Model
    ~Model, 560
    active_variables, 567
    assign_rep, 566
derivative_concurrency, 567
derived_interface, 563
    estimate_derivatives, 569
    estimate_message_lengths, 566
eval_tag_prefix, 565
evaluation_cache, 564
    FDstep1, 570
    FDstep2, 571
    fdGradStepSize, 571
    fdHessByGradStepSize, 571
get_model, 569
inactive_variables, 568
init_communicators, 565
init_serial, 566
initialize_distribution, 568
initialize_h, 570
interface_id, 564
local_eval_concurrency, 564
local_eval_synchronization, 564
manage_asv, 570
manage_data_recastings, 566
Model, 560, 561
no_spec_id, 568
operator=, 561
probDescDB, 571
restart_file, 565
solution_level_index, 563
solution_levels, 563
subordinate_iterator, 561
subordinate_model, 562
subordinate_models, 565
surrogate_model, 562
synchronize_derivatives, 569
truth_model, 562
update_from_subordinate_model, 563
update_quasi_hessians, 570
update_response, 569
user_auto_id, 568
Dakota::NCSUOptimizer
    core_run, 580
    NCSUOptimizer, 579
    objective_eval, 580
Dakota::NIDRProblemDescDB
check_driver, 595
derived_parse_inputs, 595
make_variable_defaults, 595
Dakota::NL2SOLLeastSq
core_run, 598
Dakota::NLSSOLLeastSq
core_run, 602
NLSSOLLeastSq, 602
Dakota::NOWPACOptimizer
core_run, 778
Dakota::NestedModel
derived_evaluate, 587
derived_evaluate_nowait, 587
derived_evaluation_id, 589
derived_init_communicators, 588
derived_master_overload, 588
derived_synchronize, 588
local_eval_concurrency, 588
local_eval_synchronization, 588
response_mapping, 589
subModel, 590
Dakota::NomadOptimizer::Evaluator
eval_x, 418
Evaluator, 418
Dakota::NonD
compute_densities, 611
finalize_run, 611
initialize_final_statistics, 611
initialize_run, 611
level_mappings_file, 612
print_level_map, 612
print_level_mappings, 610
Dakota::NonDAdaptImpSampling
initialize, 616
NonDAdaptImpSampling, 615
Dakota::NonDAdaptiveSampling
~NonDAdaptiveSampling, 621
core_run, 621
NonDAdaptiveSampling, 621
print_results, 622
Dakota::NonDBayesCalibration
acceptanceChain, 633
algorithm_space_model, 632
core_run, 631
export_chain, 633
init_map_optimizer, 632
log_likelihood, 632
neg_log_post_resp_mapping, 632
NonDBayesCalibration, 630
pre_run, 631
print_results, 631
prior_mean, 631
prior_variance, 631
scale_model, 633
weight_model, 633
Dakota::NonDC3FunctionTrain
NonDC3FunctionTrain, 636
sample_allocation_metric, 636
Dakota::NonDCalibration
NonDCalibration, 637
Dakota::NonDCubature
increment_grid_preference, 639, 640
NonDCubature, 639
num_samples, 640
sampling_reset, 639
Dakota::NonDDREAMBayesCalibration
cache_chain, 643
calibrate, 643
NonDDREAMBayesCalibration, 642
prior_density, 642
prior_sample, 643
problem_size, 642
problem_value, 642
sample_likelihood, 643
Dakota::NonDExpansion
algorithm_space_model, 653
compute_covariance_metric, 654
compute_level_mappings_metric, 654
compute_statistics, 654
decrement_order_and_grid, 655
increment_order_and Grid, 655
increment_specification_sequence, 653
infer_pilot_sample, 653
random_seed, 655
update_expansion, 654
update_samples_from_order_decrement, 655
useDerivs, 656
Dakota::NonDGPImpSampling
core_run, 667
NonDGPImpSampling, 667
Dakota::NonDGPMASBayesCalibration
cache_acceptance_chain, 672
calibrate, 671
fill_simulation_data, 671
NonDGPMASBayesCalibration, 671
print_results, 672
Dakota::NonDGlobalInterval
algorithm_space_model, 660
Dakota::NonDGlobalReliability
core_run, 663
print_results, 663
Dakota::NonDIntegration
anisotropic_order_to_dimension_preference, 675
core_run, 675
dimension_preference_to_anisotropic_order, 675
NonDIntegration, 675
print_points_weights, 675
Dakota::NonDLHSSampling
core_run, 684
d_optimal_parameter_set, 684
NonDLHSSampling, 683, 684
Dakota::NonDLocalReliability
<table>
<thead>
<tr>
<th>Function/Class</th>
<th>Arguments/Options</th>
</tr>
</thead>
<tbody>
<tr>
<td>core_run</td>
<td>695</td>
</tr>
<tr>
<td>dg_ds_eval</td>
<td>699</td>
</tr>
<tr>
<td>dp2_dbeta_factor</td>
<td>699</td>
</tr>
<tr>
<td>initial_taylor_series</td>
<td>697</td>
</tr>
<tr>
<td>initialize_class_data</td>
<td>697</td>
</tr>
<tr>
<td>initialize_level_data</td>
<td>698</td>
</tr>
<tr>
<td>initialize_mpp_search_data</td>
<td>698</td>
</tr>
<tr>
<td>PMA2_constraint_eval</td>
<td>697</td>
</tr>
<tr>
<td>PMA_constraint_eval</td>
<td>696</td>
</tr>
<tr>
<td>PMA_objective_eval</td>
<td>696</td>
</tr>
<tr>
<td>pre_run</td>
<td>695</td>
</tr>
<tr>
<td>print_results</td>
<td>695</td>
</tr>
<tr>
<td>probability</td>
<td>700</td>
</tr>
<tr>
<td>RIA_constraint_eval</td>
<td>696</td>
</tr>
<tr>
<td>RIA_objective_eval</td>
<td>696</td>
</tr>
<tr>
<td>update_level_data</td>
<td>699</td>
</tr>
<tr>
<td>update_mpp_search_data</td>
<td>698</td>
</tr>
<tr>
<td>Dakota::NonDMUQBayesCalibration</td>
<td>cache_chain, 726</td>
</tr>
<tr>
<td></td>
<td>calibrate, 725</td>
</tr>
<tr>
<td></td>
<td>NonDMUQBayesCalibration, 725</td>
</tr>
<tr>
<td></td>
<td>print_results, 725</td>
</tr>
<tr>
<td>Dakota::NonDMultilevelFunctionTrain</td>
<td>~NonDMultilevelFunctionTrain, 704</td>
</tr>
<tr>
<td></td>
<td>increment_specification_sequence, 705</td>
</tr>
<tr>
<td></td>
<td>infer_pilot_sample, 705</td>
</tr>
<tr>
<td></td>
<td>NonDMultilevelFunctionTrain, 703</td>
</tr>
<tr>
<td>Dakota::NonDMultilevelPolynomialChaos</td>
<td>increment_specification_sequence, 708</td>
</tr>
<tr>
<td></td>
<td>infer_pilot_sample, 709</td>
</tr>
<tr>
<td></td>
<td>NonDMultilevelPolynomialChaos, 707, 708</td>
</tr>
<tr>
<td>Dakota::NonDMultilevelSampling</td>
<td>allocationTarget, 720</td>
</tr>
<tr>
<td></td>
<td>control_variate_mc, 718</td>
</tr>
<tr>
<td></td>
<td>core_run, 717</td>
</tr>
<tr>
<td></td>
<td>multilevel_control_variate_mc_Qcorr, 719</td>
</tr>
<tr>
<td></td>
<td>multilevel_control_variate_mc_Ycorr, 719</td>
</tr>
<tr>
<td></td>
<td>multilevel_mc_Qsum, 718</td>
</tr>
<tr>
<td></td>
<td>multilevel_mc_Ysum, 718</td>
</tr>
<tr>
<td></td>
<td>NonDMultilevelSampling, 717</td>
</tr>
<tr>
<td></td>
<td>post_run, 717</td>
</tr>
<tr>
<td></td>
<td>pre_run, 717</td>
</tr>
<tr>
<td></td>
<td>print_results, 717</td>
</tr>
<tr>
<td></td>
<td>qoiAggregation, 720</td>
</tr>
<tr>
<td></td>
<td>random_seed, 720</td>
</tr>
<tr>
<td></td>
<td>uncentered_to_centered, 720</td>
</tr>
<tr>
<td>Dakota::NonDMultilevelStochCollocation</td>
<td>increment_specification_sequence, 723</td>
</tr>
<tr>
<td></td>
<td>NonDMultilevelStochCollocation, 722</td>
</tr>
<tr>
<td></td>
<td>Dakota::NonDPolynomialChaos</td>
</tr>
<tr>
<td></td>
<td>increment_order_from_grid, 734</td>
</tr>
<tr>
<td></td>
<td>NonDPolynomialChaos, 732–734</td>
</tr>
<tr>
<td>Dakota::NonDQUESOBayesCalibration</td>
<td>cache_chain, 743</td>
</tr>
<tr>
<td></td>
<td>calibrate, 743</td>
</tr>
<tr>
<td></td>
<td>init_parameter_domain, 744</td>
</tr>
<tr>
<td></td>
<td>logitTransform, 745</td>
</tr>
<tr>
<td></td>
<td>NonDQUESOBayesCalibration, 743</td>
</tr>
<tr>
<td></td>
<td>print_results, 743</td>
</tr>
<tr>
<td></td>
<td>prior_proposal_covariance, 744</td>
</tr>
<tr>
<td></td>
<td>set_ip_options, 744</td>
</tr>
<tr>
<td></td>
<td>user_proposal_covariance, 744</td>
</tr>
<tr>
<td>Dakota::NonDQuadature</td>
<td>initialize_grid, 738</td>
</tr>
<tr>
<td></td>
<td>NonDQuadature, 737, 738</td>
</tr>
<tr>
<td></td>
<td>num_samples, 738</td>
</tr>
<tr>
<td></td>
<td>sampling_reset, 738</td>
</tr>
<tr>
<td>Dakota::NonDRKDDarts</td>
<td>core_run, 749</td>
</tr>
<tr>
<td></td>
<td>post_run, 749</td>
</tr>
<tr>
<td></td>
<td>pre_run, 749</td>
</tr>
<tr>
<td>Dakota::NonDReliability</td>
<td>algorithm_space_model, 747</td>
</tr>
<tr>
<td></td>
<td>post_run, 747</td>
</tr>
<tr>
<td>Dakota::NonDSampling</td>
<td>active_set_mapping, 759</td>
</tr>
<tr>
<td></td>
<td>compute_level_mappings, 757</td>
</tr>
<tr>
<td></td>
<td>core_run, 758</td>
</tr>
<tr>
<td></td>
<td>get_parameter_sets, 758, 759</td>
</tr>
<tr>
<td></td>
<td>mode_counts, 759</td>
</tr>
<tr>
<td></td>
<td>NonDSampling, 756, 757</td>
</tr>
<tr>
<td></td>
<td>num_samples, 758</td>
</tr>
<tr>
<td></td>
<td>pre_run, 757</td>
</tr>
<tr>
<td></td>
<td>sampling_reset, 758</td>
</tr>
<tr>
<td></td>
<td>transform_samples, 757</td>
</tr>
<tr>
<td>Dakota::NonDSparseGrid</td>
<td>NonDSparseGrid, 762</td>
</tr>
<tr>
<td></td>
<td>num_samples, 763</td>
</tr>
<tr>
<td></td>
<td>sampling_reset, 763</td>
</tr>
<tr>
<td>Dakota::NonDStochCollocation</td>
<td>analytic_delta_level_mappings, 766</td>
</tr>
<tr>
<td></td>
<td>compute_covariance_metric, 766</td>
</tr>
<tr>
<td></td>
<td>compute_level_mappings_metric, 766</td>
</tr>
<tr>
<td></td>
<td>NonDStochCollocation, 765, 766</td>
</tr>
<tr>
<td>Dakota::NonDSurrogateExpansion</td>
<td>NonDSurrogateExpansion, 768</td>
</tr>
<tr>
<td>Dakota::NonDWasABIBayesCalibration</td>
<td>calibrate, 770</td>
</tr>
</tbody>
</table>
NonDWASABIBayesCalibration, 770
print_results, 770
Dakota::NonlinearCGOptimizer
brent_minimize, 773
core_run, 773
Dakota::Optimizer
configure_constraint_maps, 789
finalize_run, 788
get_common_stopping_criteria, 787
get_responses_from_dakota, 787
get_variable_bounds_from_dakota, 787
initialize_run, 788
objective_reduction, 789
post_run, 788
primary_resp_reducer, 789
print_results, 788
reduce_model, 789
Dakota::OutputManager
add_datapoint, 793
create_tabular_datastream, 793
OutputManager, 793
pop_output_tag, 793
Dakota::PSUADEDesignCompExp
core_run, 867
enforce_input_rules, 868
num_samples, 868
PSUADEDesignCompExp, 867
post_run, 867
pre_run, 867
Dakota::PStudyDACE
print_results, 864
volumetric_quality, 864
Dakota::ParallelConfiguration
mi_parallel_level, 796
mi_parallel_level_iterator, 796
Dakota::ParallelLevel
clear, 800
Dakota::ParallelLibrary
increment_parallel_configuration, 809
init_communicators, 809
init_mpi_comm, 809
ParallelLibrary, 808
push_output_tag, 808
resolve_inputs, 809
terminate_modelcenter, 808
Dakota::ParamResponsePair
evalInterfaceIds, 813
ParamResponsePair, 812
read, 812
write, 812
Dakota::ParamStudy
core_run, 818
distribute_list_of_points, 819
load_distribute_points, 818
post_run, 818
pre_run, 817
stepsPerVariable, 819
Dakota::PecosApproximation
build, 829
Dakota::ProbabilityTransformModel
initialize_distribution_types, 837
ptmInstance, 838
set_u_to_x_mapping, 838
update_from_subordinate_model, 837
vars_u_to_x_mapping, 837
vars_x_to_u_mapping, 837
Dakota::ProblemDescDB
∼ProblemDescDB, 847
check_and_broadcast, 848
check_input, 848
enforce_unique_ids, 849
get_db, 848
operator=, 847
parse_inputs, 848
post_process, 848
ProblemDescDB, 847
Dakota::ProcessApplicInterface
autotag_files, 853
file_cleanup, 853
prepare_process_environment, 853
read_results_file, 853
reset_process_environment, 853
synchronous_local_analyses, 853
Dakota::ProcessHandleApplicInterface
asynchronous_local_analyses, 857
check_wait, 857
create_command_arguments, 857
create_evaluation_process, 856
init_communicators_checks, 856
ProcessHandleApplicInterface, 856
serve_analyses_asynch, 857
set_communicators_checks, 856
synchronous_local_analysis, 856
Dakota::ProgramOptions
split_filenames, 863
Dakota::PythonInterface
derived_map_ac, 870
python_convert_int, 870
Dakota::QMEApproximation
  build, 872
clear_current_active_data, 872
Dakota::QuesoJointPdf
distributionMean, 873
distributionVariance, 873
QuesoJointPdf, 873
Dakota::QuesoVectorRV
QuesoVectorRV, 874
Dakota::RLOptimizer
RLOptimizer, 918
Dakota::RandomFieldModel
  actualReducedRank, 878
daceIterator, 878
expansionForm, 878
get_field_data, 877
initialize_mapping, 877
initialize_recast, 877
initialize_rf_coeffs, 878
rf_suite_identify_field_model, 877
rfmInstance, 878
variables_resize, 877
vars_mapping, 878
Dakota::RecastModel
derived_evaluate, 889
eval_tag_prefix, 890
init_maps, 889
RecastModel, 888, 889
update_from_model, 890
Dakota::RelaxedVarConstraints
  RelaxedVarConstraints, 892
Dakota::RelaxedVariables
read_core, 894
read_tabular, 894
RelaxedVariables, 894
Dakota::Response
  functionGradients, 902
Dakota::ResultsDBAny
  extract_data, 905
  insert, 905
Dakota::ResultsDBBase
array_insert, 907
Dakota::ResultsEntry
ResultsEntry, 910
Dakota::RichExtrapVerification
  converge_order, 916
  converge_qoi, 916
  core_run, 916
  estimate_order, 916
  print_results, 916
Dakota::SNLLLeastSq
  constraint1_evaluator_gn, 978
  constraint2_evaluator_gn, 979
  nlf2_evaluator_gn, 978
Dakota::SNLLOptimizer
  constraint0_evaluator, 986
  constraint1_evaluator, 986
  constraint2_evaluator, 986
  nlf0_evaluator, 985
  nlf1_evaluator, 985
  nlf2_evaluator, 986
  SNLLOptimizer, 984, 985
Dakota::ScalingModel
cv_scaled2native, 922
initialize_scaling, 923
lin_coeffs_modify_n2s, 924
modify_n2s, 925
modify_s2n, 925
need_resp_trans_byvars, 925
resp_scaled2native, 923
response_modify_n2s, 925
response_modify_s2n, 923
scaleModelInstance, 926
ScalingModel, 922
secondary_resp_scaled2native, 923
secondary_resp_scaler, 924
variables_scaler, 924
Dakota::SensAnalysisGlobal
  compute_correlations, 930
  partial_corr, 931
  simple_corr, 930
  values_to_ranks, 930
Dakota::SeqHybridMetaIterator
  extract_parameter_sets, 934
  print_results, 934
  run_sequential, 934
  run_sequential_adaptive, 934
Dakota::SharedApproxData
~SharedApproxData, 941
buildDataOrder, 942
get_shared_data, 942
operator=, 942
SharedApproxData, 941
Dakota::SharedC3ApproxData
  regression_size, 946
Dakota::SharedPecosApproxData
  push_index, 950
Dakota::SharedResponseData
Dakota::SharedResponseDataRep
  copy, 952
Dakota::SharedSurfpackApproxData
  SharedSurfpackApproxData, 956
Dakota::SharedVariablesData
  copy, 964
Dakota::SharedVariablesDataRep
  activeVarsCompsTotals, 969
  allContinuousIds, 969
  copy_rep, 969
  inactiveVarsCompsTotals, 969
  variablesCompsTotals, 969
Dakota::SimulationModel
  derived_interface, 972
  eval_tag_prefix, 972
Dakota::SurfpackApproximation
  add_constraints_to_surfdata, 998
  build, 997
  hessian, 998
  SurfpackApproximation, 997
  surrogates_to_surf_data, 998
Dakota::SurrBasedLocalMinimizer
  approx_subprob_constraint_eval, 1007
  approx_subprob_objective_eval, 1006
  compute_trust_region_ratio, 1005
  core_run, 1005
  hard_convergence_check, 1006
  hom_constraint_eval, 1007
  hom_objective_eval, 1007
  post_run, 1005
  pre_run, 1004
  update_penalty, 1006
Dakota::SurrBasedMinimizer
  augmented_lagrangian_merit, 1011
  constraintViolation, 1012
  lagrangian_merit, 1011
  penalty_merit, 1011
  print_results, 1010
  update_augmented_lagrange_multipliers, 1010
  update_filter, 1011
  update_lagrange_multipliers, 1010
Dakota::SurrogateModel
  approxBuilds, 1019
  derived_evaluation_id, 1018
  force_rebuild, 1018
  responseMode, 1019
Dakota::SurrogatesGPApprox
  SurrogatesGPApprox, 1021
Dakota::SurrogatesPolyApprox
  SurrogatesPolyApprox, 1023
Dakota::SysCallApplicInterface
  init_communicators_checks, 1025
  set_communicators_checks, 1025
  spawn_analysis_to_shell, 1025
  spawn_evaluation_to_shell, 1025
  spawn_input_filter_to_shell, 1025
  spawn_output_filter_to_shell, 1026
  synchronous_local_analysis, 1024
  test_local_evaluation_sequence, 1024
  wait_local_evaluation_sequence, 1024
Dakota::TANA3Approximation
  build, 1029
  clear_current_active_data, 1029
Dakota::TabularReader
  operator(), 1027
Dakota::TabularWriter
  operator(), 1027
Dakota::TaylorApproximation
  build, 1031
Dakota::TestDriverInterface
  barnes, 1036
  barnes_lf, 1036
  derived_map_ac, 1035
  herbie, 1037
  herbie1D, 1036
  levenshtein_distance, 1037
  lf_poly_prod, 1035
  mc_api_run, 1038
  poly_prod, 1035
  separable_combine, 1037
  shubert1D, 1037
  smooth_herbie, 1037
  smooth_herbie1D, 1037
  steady_state_diffusion_1d, 1036
  steel_column_cost, 1036
Dakota::TrackerHTTP
  send_data_using_get, 1043
  send_data_using_post, 1043
Dakota::UsageTracker
  UsageTracker, 1046
Dakota::VPSApproximation
  VPSinstance, 1063
  VPSmodel_apply, 1063
Dakota::Variables
  discrete_string_variables_view, 1057
Dakota::Verification
  print_results, 1058
Dakota::WeightingModel
weightModelInstance, 1065
Dakota::WorkdirHelper
    copy_items, 1068
    create_directory, 1068
    file_opItems, 1069
    init_preferred_env_path, 1070
    init_startup_path, 1070
    initialize, 1067
    link, 1069
    link_items, 1068
    po_which, 1069
    prepend_path_item, 1069
    prepend_preferred_env_path, 1067
    recursive_copy, 1069
    set_preferred_path, 1069
    split_wildcard, 1068
    tokenize_env_path, 1070
    which, 1068

dakota::surrogates, 181
    compute_hyperbolic_indices, 184
    compute_hyperbolic_level_indices, 183
    compute_hyperbolic_subdim_level_indices, 183
    compute_next_combination, 182
    compute_reduced_indices, 184
    fd_check_gradient, 184
    fd_check_hessian, 184
    size_level_index_vector, 183

dakota::surrogates::GP_Ojective
    GP_Objective, 456
    getVector, 457
    gradient, 456
    pdiff, 456
    value, 456

dakota::surrogates::GaussianProcess
    build, 441
    compute_first_deriv_pred_gram, 444
    compute_gram, 444
    compute_pred_dists, 444
    compute_second_deriv_pred_gram, 444
    GaussianProcess, 440, 441
    generate_initial_guesses, 446
    get_num_opt_variables, 443
    get_num_variables, 443
    get_posterior_covariance, 443
    get_posterior_std_dev, 443
    gradient, 442
    hessian, 442
    negative_marginal_log_likelihood, 442
    set_opt_params, 443
    setup_default_optimization_params, 446
    value, 441

dakota::surrogates::PolynomialRegression
    build, 831
    compute_basis_matrix, 831
    gradient, 833
    hessian, 833
    PolynomialRegression, 830, 831
    value, 831

dakota::surrogates::Surrogate
    build, 1013
    get_options, 1015
    gradient, 1014
    hessian, 1014
    set_options, 1014
    value, 1014

dakota::util, 185
    append_columns, 190
    error, 187
    matrix_equals, 187
    n_choose_k, 189
    nonzero, 189
    num_nonzeros, 189
    p_norm, 190
    populateMatricesFromFile, 188
    populateVectorsFromFile, 188
    scaler_factory, 188
    solver_factory, 189
    type_name_bimap, 190
    variance, 188

dakota::util::CholeskySolver
    factorize, 265
    solve, 265, 266

dakota::util::DataScaler
    check_for_zero_scaler_factor, 362
    get_scaled_features, 362
    get_scaler_features_offsets, 362
    get_scaler_features_scale_factors, 362
    scale_samples, 361
    scaler_type, 362

dakota::util::LinearSolverBase
    factorize, 519
    solve, 519, 520
    solver_type, 519

dakota::util::NoScaler
    NoScaler, 776

dakota::util::NormalizationScaler
    NormalizationScaler, 775
dakota::util::StandardizationScaler
  StandardizationScaler, 993
dakota_dll_api.cpp, 1071
dakota_stop, 1072
dakota_dll_api.h, 1072
dakota_stop, 1073
dakota_linear_algebra.hpp, 1073
dakota_response_io::couljn_response
  Dakota::COLINAApplication, 268
dakota_stop
  dakota_dll_api.cpp, 1072
dakota_dll_api.h, 1073
dakota_tabular_io.hpp, 1074
DakotaROLEqConstraints, 299
  Dakota::DakotaROLEqConstraints, 300
DakotaROLEqConstraintsGrad, 300
  Dakota::DakotaROLEqConstraintsGrad, 301
DakotaROLEqConstraintsHess, 301
  Dakota::DakotaROLEqConstraintsHess, 302
DakotaROLIneqConstraints, 302
  Dakota::DakotaROLIneqConstraints, 303
DakotaROLIneqConstraintsGrad, 303
  Dakota::DakotaROLIneqConstraintsGrad, 304
DakotaROLIneqConstraintsHess, 304
  Dakota::DakotaROLIneqConstraintsHess, 304
DakotaROLObjective, 305
  Dakota::DakotaROLObjective, 305
DakotaROLObjectiveGrad, 306
  Dakota::DakotaROLObjectiveGrad, 306
DakotaROLObjectiveHess, 306
  Dakota::DakotaROLObjectiveHess, 307
data_transform_model
  Dakota::Minimizer, 529
DataEnvironment, 307
DataEnvironmentRep, 308
DataFitSurrBasedLocalTraits, 311
DataFitSurrModel, 312
DataInterface, 327
DataMethod, 328
DataMethodRep, 329
DataModel, 346
DataModelRep, 347
DataResponses, 354
DataResponsesRep, 356
DataScaler, 360
DataTransformModel, 363
  Dakota::DataTransformModel, 366
DataVariables, 369
DataVariablesRep, 371
decrement_order_and_grid
  Dakota::NonDExpansion, 655
DerivInformedPropCovLogitTK< V, M >, 389
DerivInformedPropCovTK< V, M >, 389
derivVarsVector
  Dakota::ActiveSet, 195
derivative_concurrency
  Dakota::Model, 567
derived_evaluate
  Dakota::DataFitSurrModel, 319
  Dakota::DataTransformModel, 367
  Dakota::HierarchSurrModel, 471
  Dakota::NestedModel, 587
  Dakota::RecastModel, 889
derived_evaluate_nowait
  Dakota::DataFitSurrModel, 319
  Dakota::DataTransformModel, 367
  Dakota::HierarchSurrModel, 472
  Dakota::NestedModel, 587
derived_evaluation_id
  Dakota::NestedModel, 589
  Dakota::SurrogateModel, 1018
derived_init_communicators
  Dakota::ActiveSubspaceModel, 200
  Dakota::AdaptedBasisModel, 205
  Dakota::DataFitSurrModel, 323
  Dakota::NestedModel, 588
derived_interface
  Dakota::Model, 563
  Dakota::SimulationModel, 972
derived_map_ac
  Dakota::DirectApplicInterface, 394
  Dakota::MatlabInterface, 521
  Dakota::PythonInterface, 870
  Dakota::TestDriverInterface, 1035
  StanfordPSAAP::SoleilDirectApplicInterface, 991
derived_master_overload
  Dakota::NestedModel, 588
derived_parse_input
  Dakota::NIDRProblemDescDB, 595
derived_synchronize
  Dakota::DataFitSurrModel, 319
  Dakota::DataTransformModel, 367
  Dakota::HierarchSurrModel, 472
  Dakota::NestedModel, 588
derived_synchronize_nowait
  Dakota::DataFitSurrModel, 320
  Dakota::DataTransformModel, 367
  Dakota::HierarchSurrModel, 472
INDEX

Description
Dakota::JEGAOptimizer::Evaluator, 414
DesignAndStateLabelsCheck
Dakota, 177
DestroyAlgorithm
Dakota::JEGAOptimizer::Driver, 402
dg_ds_eval
Dakota::NonDLocalReliability, 699
dimension_preference_to_anisotropic_order
Dakota::NonDIntegration, 675
DirectApplicInterface, 390
DiscSetLbl
Dakota, 177
DiscrepancyCorrection, 395
discrete_string_variables_view
Dakota::Variables, 1057
distribute_list_of_points
Dakota::ParamStudy, 819
distributionMean
Dakota::QuesoJointPdf, 873
distributionVariance
Dakota::QuesoJointPdf, 873
dll_tester.cpp, 1076
dp2_dbeta_factor
Dakota::NonDLocalReliability, 699
Driver
Dakota::JEGAOptimizer::Driver, 401
dtModelInstance
Dakota::DataTransformModel, 369
duplication_detect
Dakota::ApplicationInterface, 225

EIF_objective_eval
Dakota::EffGlobalMinimizer, 405
EffGlobalMinimizer, 402
EffGlobalTraits, 405
EmbedHybridMetaIterator, 406
enforce_input_rules
Dakota::FSUDesignCompExp, 436
Dakota::PSUADEDesignCompExp, 868
enforce_unique_ids
Dakota::ProblemDescDB, 849
enroll
Dakota::GetLongOpt, 454
Environment, 408
Dakota::Environment, 410, 411
error
dakota::util, 187
estimate_derivatives
Dakota::Model, 569
estimate_message_lengths
Dakota::Model, 566
estimate_order
Dakota::RichExtrapVerification, 916
eval_tag_prefix
Dakota::Interface, 482
Dakota::Iterator, 496
Dakota::Model, 565
Dakota::RecastModel, 890
Dakota::SimulationModel, 972
eval_x
Dakota::NomadOptimizer::Evaluator, 418
evalInterfaceIds
Dakota::ParamResponsePair, 813
Evalue
Dakota::JEGAOptimizer::Evaluator, 415, 416
evaluate_parameter_sets
Dakota::Analyzer, 213
evaluation_available
Dakota::COLINApplication, 268
evaluation_cache
Dakota::Model, 564
Evaluator
Dakota::JEGAOptimizer::Evaluator, 413, 414
Dakota::NomadOptimizer::Evaluator, 418
EvaluatorCreator
Dakota::JEGAOptimizer::EvaluatorCreator, 419
ExecutableEnvironment, 420
exit_mode
Dakota::Environment, 411
expand_primary_array
Dakota::DataTransformModel, 368
expansionForm
Dakota::RandomFieldModel, 878
ExperimentData, 421
ExperimentResponse, 429
export_approximation
Dakota::ApproximationInterface, 244
export_chain
Dakota::NonDBayesCalibration, 633
export_point
Dakota::DataFitSurrModel, 324
extract_data
Dakota::ResultsDBAny, 905
extract_parameter_sets
Dakota::SeqHybridMetaIterator, 934
ExtractAllData
Dakota::JEGAOptimizer::Driver, 401
FDstep1
INDEX

Dakota::RandomFieldModel, 877
get_initial_values
Dakota, 166
get_interface
Dakota::Interface, 482
get_iterator
Dakota::Iterator, 497
get_linear_constraints
Dakota, 167
get_mixed_bounds
Dakota, 166
get_model
Dakota::Model, 569
get_nonlinear_bounds
Dakota, 169
get_nonlinear_eq_constraints
Dakota, 168, 169
get_nonlinear_ineq_constraints
Dakota, 169
get_num_opt_variables
Dakota::surrogates::GaussianProcess, 443
get_num_variables
Dakota::surrogates::GaussianProcess, 443
get_options
Dakota::surrogates::Surrogate, 1015
get_parameter_sets
Dakota::NonDSampling, 758, 759
get_pathext
Dakota, 174
get_posterior_covariance
Dakota::surrogates::GaussianProcess, 443
get_posterior_std_dev
Dakota::surrogates::GaussianProcess, 443
get_responses
Dakota, 168
get_responses_from_dakota
Dakota::Optimizer, 787
get_scaled_features
Dakota::util::DataScaler, 362
get_scaler_features_offsets
Dakota::util::DataScaler, 362
get_scaler_features_scale_factors
Dakota::util::DataScaler, 362
get_shared_data
Dakota::SharedApproxData, 942
get_variable_bounds
Dakota, 166
get_variable_bounds_from_dakota
Dakota::Optimizer, 787
get_variables
Dakota, 168
get_vbd_parameter_sets
Dakota::Analyzer, 213
GetBestMOSolutions
Dakota::JEGAOptimizer, 509
GetBestSOSolutions
Dakota::JEGAOptimizer, 509
GetBestSolutions
Dakota::JEGAOptimizer, 508
GetDescription
Dakota::JEGAOptimizer::Evaluator, 416
GetLongOpt
Dakota::Optimizer, 452
Dakota::GetLongOpt, 454
GetName
Dakota::JEGAOptimizer::Evaluator, 416
GetNumberOfLinearConstraints
Dakota::JEGAOptimizer::Evaluator, 415
GetNumberOfNonLinearConstraints
Dakota::JEGAOptimizer::Evaluator, 415
getRmax
Dakota, 170
getVector
Dakota::surrogates::GP_Objective, 457
getdist
Dakota, 170
gnewton_set_recast
Dakota::Iterator, 496
gradient
Dakota::surrogates::GaussianProcess, 442
Dakota::surrogates::GP_Objective, 456
Dakota::surrogates::PolynomialRegression, 833
Dakota::surrogates::Surrogate, 1014
Graphics, 457
GridApplicInterface, 459
HDF5IOHelper, 461
half_log_cov_det_gradient
Dakota::ExperimentData, 428
half_log_cov_det_hessian
Dakota::ExperimentData, 428
half_log_cov_det_hessian
Dakota::ExperimentData, 427
hard_convergence_check
Dakota::SurrogateLocalMinimizer, 1006
herbie
Dakota::TestDriverInterface, 1037
herbie1D
Dakota::TestDriverInterface, 1036
hessian
Dakota::SurfpackApproximation, 998
dakota::surrogates::GaussianProcess, 442
dakota::surrogates::PolynomialRegression, 833
dakota::surrogates::Surrogate, 1014
HierarchSurrBasedLocalTraits, 466
HierarchSurrModel, 467
hom_constr_eval
  Dakota::SurrBasedLocalMinimizer, 1007
hom_objective_eval
  Dakota::SurrBasedLocalMinimizer, 1007
IC
  Dakota::CONMINOptimizer, 288
ISC
  Dakota::CONMINOptimizer, 287
id_vars_exact_compare
  Dakota, 172
import_points
  Dakota::DataFitSurrModel, 323
inactive_variables
  Dakota::Model, 568
inactiveVarsCompsTotals
  Dakota::SharedVariablesDataRep, 969
increment_grid_preference
  Dakota::NonDCubature, 639, 640
increment_order_and_grid
  Dakota::NonDExpansion, 655
increment_order_from_grid
  Dakota::NonDExpansion, 653
increment_parallel_configuration
  Dakota::ParallelLibrary, 809
increment_specification_sequence
  Dakota::NonDExpansion, 653
  Dakota::NonDMultilevelFunctionTrain, 705
  Dakota::NonDMultilevelPolynomialChaos, 708
  Dakota::NonDMultilevelStochCollocation, 723
infer_pilot_sample
  Dakota::NonDExpansion, 653
  Dakota::NonDMultilevelFunctionTrain, 705
  Dakota::NonDMultilevelPolynomialChaos, 709
init_communicators
  Dakota::Model, 565
  Dakota::ParallelLibrary, 809
init_communicators_checks
  Dakota::ApplicationInterface, 225
  Dakota::DirectApplicInterface, 394
  Dakota::ProcessHandleApplicInterface, 856
  Dakota::SysCallApplicInterface, 1025
init_continuous_vars
  Dakota::DataTransformModel, 368
init_default_asv
  Dakota::ApplicationInterface, 226
init_iterator
  Dakota::IteratorScheduler, 502
init_map_optimizer
  Dakota::NonDBayesCalibration, 632
init_maps
  Dakota::RecastModel, 889
init_model
  Dakota::DataFitSurrModel, 325
init_mpi_comm
  Dakota::ParallelLibrary, 809
init_parameter_domain
  Dakota::NonDQUESOBayesCalibration, 744
init Preferred_env_path
  Dakota::WorkdirHelper, 1070
init_serial
  Dakota::ApplicationInterface, 223
  Dakota::Model, 566
init_startup_path
  Dakota::WorkdirHelper, 1070
initial_points
  Dakota::JEGAOptimizer, 510
initial_taylor_series
  Dakota::NonDQUESOBayesCalibration, 744
initialize
  Dakota::NonDAdaptImpSampling, 616
  Dakota::WorkdirHelper, 1067
initialize_class_data
  Dakota::NonDLocalReliability, 697
initialize_distribution
  Dakota::Model, 568
initialize_distribution_types
  Dakota::ProbabilityTransformModel, 837
initialize_export
  Dakota::DataFitSurrModel, 323
initialize_final_statistics
  Dakota::NonD, 611
initialize_graphics
  Dakota::Iterator, 495
initialize_grid
  Dakota::NonDQuadrature, 738
initialize_h
  Dakota::Model, 570
initialize_level_data
  Dakota::NonDLocalReliability, 698
initialize_mapping
  Dakota::ActiveSubspaceModel, 200
  Dakota::HierarchSurrSpaceModel, 471
Dakota::Model, 564
Dakota::NestedModel, 588
local_recast_retrieve
Dakota::Minimizer, 531
log_likelihood
Dakota::NonDBayesCalibration, 632
logitTransform
Dakota::NonDQUESOBayesCalibration, 745
lookup_by_val
Dakota, 172
MPIManager, 572
MPIPackBuffer, 573
MPIUnpackBuffer, 575
MS1
   Dakota::CONMINOptimizer, 287
main
      library_mode.cpp, 1080
      main.cpp, 1082
      restart_util.cpp, 1084
main.cpp, 1082
   fpinit_ASC, 1082
   main, 1082
make_variable_defaults
Dakota::NIDRProblemDescDB, 595
manage_asv
Dakota::Model, 570
manage_data_recastings
Dakota::Model, 566
manage_linear_constraints
Dakota::Constraints, 298
MandatoryValue
Dakota::GetLongOpt, 453
map
Dakota::ApplicationInterface, 223
map_domain
Dakota::COLINApplication, 269
master_dynamic_schedule_analyses
Dakota::ApplicationInterface, 225
master_dynamic_schedule_evaluations
Dakota::ApplicationInterface, 226
master_dynamic_schedule_evaluations_nowait
Dakota::ApplicationInterface, 228
master_dynamic_schedule_iterators
Dakota::IteratorScheduler, 504
MatchesWC, 520
matlab_engine_run
Dakota::MatlabInterface, 521
MatlabInterface, 521
matrix_equal
Dakota::util, 187
maxEvalConcurrent
Dakota::Iterator, 498
mc_api_run
Dakota::TestDriverInterface, 1038
MetaIterator, 522
mi_parallel_level
Dakota::ParallelConfiguration, 796
mi_parallel_level_iterator
Dakota::ParallelConfiguration, 796
mindist
Dakota, 170
mindistindx
Dakota, 170
Minimizer, 524
Dakota::Minimizer, 528
MixedVarConstraints, 532
Dakota::MixedVarConstraints, 532
MixedVariables, 533
Dakota::MixedVariables, 534
mode_counts
Dakota::NonDSampling, 759
Model, 535
Dakota::Model, 560, 561
modify_n2s
Dakota::ScalingModel, 925
modify_s2n
Dakota::ScalingModel, 925
mpi_debug_hold
Dakota, 164
multilevel_control_variate_me_Qcorr
Dakota::NonDMultilevelSampling, 719
multilevel_control_variate_me_Ycorr
Dakota::NonDMultilevelSampling, 719
multilevel_me_Qsum
Dakota::NonDMultilevelSampling, 718
multilevel_me_Ysum
Dakota::NonDMultilevelSampling, 718
N1
   Dakota::CONMINOptimizer, 285
N2
   Dakota::CONMINOptimizer, 286
N3
   Dakota::CONMINOptimizer, 286
N4
   Dakota::CONMINOptimizer, 286
N5
   Dakota::CONMINOptimizer, 286
n_choose_k
dakota::util, 189
NCSUOptimizer, 577
  Dakota::NCSUOptimizer, 579
NCSUTraits, 580
NIDRProblemDescDB, 590
NL2Res, 596
NL2SOLLeastSq, 596
NL2SOLLeastSqTraits, 599
NLPLPTraits, 599
NLSSOLLeastSq, 601
  Dakota::NLSSOLLeastSq, 602
NLSSOLLeastSqTraits, 603
NOWPACBlackBoxEvaluator, 776
NOWPACOptimizer, 777
NOWPACTraits, 779
NPSOLTraits, 780
NUMBER_OF_FIELDS
  Dakota, 175
Name
  Dakota::JEGAOptimizer::Evaluator, 414
need_responsibility_trans, byvars
  Dakota::ScalingModel, 925
neg_log_post responsibility_mapping
  Dakota::NonDBayesCalibration, 632
negative_marginal_log_likelihood
  dakota::surrogates::GaussianProcess, 442
NestedModel, 581
new_dataset
  Dakota::Graphics, 459
nlf0_evaluator
  Dakota::SNLLOptimizer, 985
nlf1_evaluator
  Dakota::SNLLOptimizer, 985
nlf2_evaluator
  Dakota::SNLLOptimizer, 986
nlf2_evaluator_gn
  Dakota::NLSSOLastSq, 978
no_spec_id
  Dakota::Interface, 483
  Dakota::Iterator, 497
  Dakota::Model, 568
NoDBBaseConstructor, 603
NoScaler, 775
  dakota::util::NoScaler, 776
NomadOptimizer::Evaluator, 417
NomadTraits, 604
NonD, 605
NonDAdaptiveSampling, 612
  Dakota::NonDAdaptiveSampling, 617
  Dakota::NonDAdaptiveSampling, 621
NonDBayesCalibration, 622
  Dakota::NonDBayesCalibration, 630
NonDC3FunctionTrain, 634
  Dakota::NonDC3FunctionTrain, 636
NonDCalibration, 636
  Dakota::NonDCalibration, 637
NonDCubature, 638
  Dakota::NonDCubature, 639
NonDDREAMBayesCalibration, 640
  Dakota::NonDDREAMBayesCalibration, 642
NonDExpansion, 644
NonDGPImpSampling, 665
  Dakota::NonDGPImpSampling, 667
NonDGPMPSABayesCalibration, 668
  Dakota::NonDGPMPSABayesCalibration, 671
NonDGlobalEvidence, 656
NonDGlobalInterval, 657
NonDGlocalReliability, 661
NonDGlocalSingleInterval, 664
NonDIntegration, 672
  Dakota::NonDIntegration, 675
NonDInterval, 676
NonDLHSEvidence, 678
NonDLHSInterval, 679
NonDLHSSampling, 681
  Dakota::NonDLHSSampling, 683, 684
NonDLHSSingleInterval, 685
NonDLLocalEvidence, 686
NonDLLocalInterval, 687
NonDLLocalReliability, 690
NonDLLocalSingleInterval, 700
NonDMUQBayesCalibration, 723
  Dakota::NonDMUQBayesCalibration, 725
NonDMultilevelFunctionTrain, 702
  Dakota::NonDMultilevelFunctionTrain, 703
NonDMultilevelPolynomialChaos, 705
  Dakota::NonDMultilevelPolynomialChaos, 707, 708
NonDMultilevelSampling, 709
  Dakota::NonDMultilevelSampling, 717
NonDMultilevelStochCollocation, 721
  Dakota::NonDMultilevelStochCollocation, 722
NonDPOFDarts, 726
NonDPolynomialChaos, 729
  Dakota::NonDPolynomialChaos, 732–734
NonDQUESOBayesCalibration, 739
  Dakota::NonDQUESOBayesCalibration, 743
NonDQuadrature, 734
Dakota::NonDQuadrature, 737, 738
NonDRKDDarts, 747
NonDReliability, 745
NonDSampling, 750
  Dakota::NonDSampling, 756, 757
NonDSparseGrid, 760
  Dakota::NonDSparseGrid, 762
NonDSurrogateCollocation, 763
  Dakota::NonDSurrogateCollocation, 765, 766
NonDSurrogateExpansion, 767
  Dakota::NonDSurrogateExpansion, 768
NonDWASABIbaysCalibration, 768
  Dakota::NonDWASABIbaysCalibration, 770
NonlinearCGOptimizer, 771
NonlinearCGTraits, 774
nonzero
dakota::util, 189
NormalizationScaler, 774
dakota::util::NormalizationScaler, 775
num_nonzeros
dakota::util, 189
num_samples
Dakota::Analyzer, 211
Dakota::DACEDesignCompExp, 388
Dakota::FSUDesignCompExp, 436
Dakota::NonDCubature, 640
Dakota::NonDQuadrature, 738
Dakota::NonDSampling, 758
Dakota::NonDSparseGrid, 763
Dakota::PSUADesignCompExp, 868
objective
Dakota::Minimizer, 530
objective_eval
Dakota::NCSUOptimizer, 580
objective_gradient
Dakota::Minimizer, 530
objective_hessian
Dakota::Minimizer, 530
objective_reduction
Dakota::Optimizer, 789
operator<<
Dakota::CommandShell, 278
operator()
Dakota::LabelsWriter, 512
Dakota::TabularReader, 1027
Dakota::TabularWriter, 1027
operator=
Dakota::Approximation, 238
Dakota::Constraints, 298
Dakota::Environment, 411
Dakota::Interface, 481
Dakota::Iterator, 493
Dakota::Model, 561
Dakota::ProblemDescDB, 847
Dakota::SharedApproxData, 942
operator==
Dakota, 166, 171
OptDartsOptimizer, 781
OptDartsTraits, 783
OptType
Dakota::GetLongOpt, 453
Optimizer, 784
OptionalValue
Dakota::GetLongOpt, 453
output_helper
Dakota::CommandLineHandler, 277
OutputManager, 790
Dakota::OutputManager, 793
OutputWriter, 793
p_norm
dakota::util, 190
PMA2_constraint_eval
Dakota::NonDLocalReliability, 697
PMA_constraint_eval
Dakota::NonDLocalReliability, 696
PMA_objective_eval
Dakota::NonDLocalReliability, 696
PRPMultiIndexCache
Dakota, 163
PRPMultiIndexQueue
Dakota, 163
PSUADesignCompExp, 865
Dakota::PSUADesignCompExp, 867
PStudyDACE, 863
parallel_input
library_mode.cpp, 1081
parallel_interface_plugin
library_mode.cpp, 1080
ParallelConfiguration, 794
ParallelDirectApplicInterface, 796
ParallelLevel, 798
ParallelLibrary, 800
Dakota::ParallelLibrary, 808
ParamResponsePair, 810
Dakota::ParamResponsePair, 812
ParamStudy, 813
parse
Dakota::Environment, 412
Dakota, 172
print_restart_pdb
Dakota, 173
print_restart_tabular
Dakota, 173
print_results
Dakota::Analyzer, 213
Dakota::ConcurrentMetaIterator, 281
Dakota::Iterator, 495
Dakota::LeastSq, 515
Dakota::NonDAdaptiveSampling, 622
Dakota::NonDBayesCalibration, 631
Dakota::NonDGGlobalReliability, 663
Dakota::NonDGPMSABayesCalibration, 672
Dakota::NonDLGlobalReliability, 695
Dakota::NonDMultilevelSampling, 717
Dakota::NonDMUQBayesCalibration, 725
Dakota::NonDQUESOBayesCalibration, 743
Dakota::NonDWASABIBayesCalibration, 770
Dakota::Optimizer, 788
Dakota::PStudyDACE, 864
Dakota::RichExtrapVerification, 916
Dakota::SeqHybridMetaIterator, 934
Dakota::SurrBasedMinimizer, 1010
Dakota::Verification, 1058
print_sobol_indices
Dakota::Analyzer, 214
printControl
Dakota::CONMINOptimizer, 285
prior_density
Dakota::NonDDREAMBayesCalibration, 642
prior_mean
Dakota::NonDBayesCalibration, 631
prior_proposal_covariance
Dakota::NonDQUESOBayesCalibration, 744
prior_sample
Dakota::NonDDREAMBayesCalibration, 643
prior_variance
Dakota::NonDBayesCalibration, 631
probDescDB
Dakota::Iterator, 498
Dakota::Model, 571
probability
Dakota::NonDLGlobalReliability, 700
ProbabilityTransformModel, 834
problem_size
Dakota::NonDDREAMBayesCalibration, 642
problem_value
Dakota::NonDDREAMBayesCalibration, 642
ProblemDescDB, 838
Dakota::ProblemDescDB, 847
ProcessApplicInterface, 849
ProcessHandleApplicInterface, 854
Dakota::ProcessHandleApplicInterface, 856
ProgramOptions, 858
ptmInstance
Dakota::ProbabilityTransformModel, 838
push_approximation
Dakota::ApproximationInterface, 245
push_index
Dakota::SharedPecosApproxData, 950
push_output_tag
Dakota::ParallelLibrary, 808
python_convert_int
Dakota::PythonInterface, 870
PythonInterface, 868
QMEApproximation, 870
QUESOImpl.hpp, 1082
qoiAggregation
Dakota::NonDMultilevelSampling, 720
qr
Dakota, 165
qr_rsolve
Dakota, 165
QuesoJointPdf
Dakota::QuesoJointPdf, 873
QuesoJointPdf< V, M >, 872
QuesoVectorRV
Dakota::QuesoVectorRV, 874
QuesoVectorRV< V, M >, 873
RIA_constraint_eval
Dakota::NonDLGlobalReliability, 696
RIA_objective_eval
Dakota::NonDLGlobalReliability, 696
ROLOptimizer, 917
Dakota::ROLOptimizer, 918
ROLTraits, 918
random_seed
Dakota::NonDExpansion, 655
Dakota::NonDMultilevelSampling, 720
RandomFieldModel, 874
rawResponseMap
Dakota::Interface, 483
read
Dakota::ParamResponsePair, 812
read_challenge_points
Dakota::ApproximationInterface, 245
INDEX

read_core
  Dakota::MixedVariables, 534
  Dakota::RelaxedVariables, 894
read_neutral
  Dakota, 173
read_results_file
  Dakota::ProcessApplicInterface, 853
read_tabular
  Dakota::MixedVariables, 534
  Dakota::RelaxedVariables, 894
read_variables_responses
  Dakota::Analyzer, 214
read_vector
  Dakota::HDF5IOHelper, 466
RealScale, 879
rebuild_approximation
  Dakota::ApproximationInterface, 244
  Dakota::DataFitSurrModel, 320
rebuild_global
  Dakota::DataFitSurrModel, 324
RecastModel, 880
  Dakota::RecastModel, 888, 889
RecordResponses
  Dakota::JEGAOptimizer::Evaluator, 415
recover_model
  Dakota::ExperimentData, 426
recursive_copy
  Dakota::WorkdirHelper, 1069
recv
  Dakota::APPSEvalMgr, 248
reduce_model
  Dakota::Optimizer, 789
ReducedBasis, 890
register_signal_handlers
  Dakota, 164
regression_size
  Dakota::C3Approximation, 262
  Dakota::SharedC3ApproxData, 946
RelaxedVarConstraints, 891
  Dakota::RelaxedVarConstraints, 892
RelaxedVariables, 892
  Dakota::RelaxedVariables, 894
repair_restart
  Dakota, 173
requestVector
  Dakota::ActiveSet, 195
reset_process_environment
  Dakota::ProcessApplicInterface, 853
reshape
  Dakota::Constraints, 298
  residuals_per_multiplier
  Dakota::ExperimentData, 428
  resize_best_resp_array
  Dakota::Minimizer, 531
  resize_best_vars_array
  Dakota::Minimizer, 531
resolve_inputs
  Dakota::ParallelLibrary, 809
  resolve_samples_symbols
  Dakota::DDACEDesignCompExp, 388
  resp_scaled2native
  Dakota::ScalingModel, 923
Response, 894
response_mapping
  Dakota::ActiveSubspaceModel, 202
  Dakota::AdaptedBasisModel, 206
  Dakota::Interface, 482
  Dakota::NestedModel, 589
response_modify_n2s
  Dakota::ScalingModel, 925
response_modify_s2n
  Dakota::ScalingModel, 923
responseMode
  Dakota::SurrogateModel, 1019
restart_file
  Dakota::Model, 565
  restart_util.cpp, 1083
  main, 1084
RestartWriter, 902
restore_data_key
  Dakota::ApproximationInterface, 245
ResultAttribute< T >, 903
ResultsDBAny, 904
ResultsDBBase, 906
ResultsDBHDF5, 907
ResultsEntry
  Dakota::ResultsEntry, 910
  ResultsEntry< StoredType >, 909
  ResultsFileError, 910
  ResultsManager, 911
  ResultsNames, 913
retrieve
  Dakota::GetLongOpt, 454
returns_multiple_points
  Dakota::COLINOptimizer, 271
  Dakota::JEGAOptimizer, 510
rf_suite_identify_field_model
  Dakota::RandomFieldModel, 877
Dakota::RandomFieldModel, 878

run
Dakota::Iterator, 495
run_dakota.data
library_mode.cpp, 1079
run_dakota.mixed
library_mode.cpp, 1079
run_dakota_parse
library_mode.cpp, 1079
run_iterator
Dakota::IteratorScheduler, 503
run_sequential
Dakota::SeqHybridMetaIterator, 934
run_sequential_adaptive
Dakota::SeqHybridMetaIterator, 934

S
Dakota::CONMINOptimizer, 286
SCAL
Dakota::CONMINOptimizer, 287
SCI_FIELD_NAMES
Dakota, 181
SCI_NUMBER_OF_FIELDS
Dakota, 181
SIM, 191
SIM::ParallelDirectApplicInterface
test_local_evaluations, 797
SIM::SerialDirectApplicInterface
test_local_evaluations, 936
SNLLBase, 974
SNLLLeastSq, 976
SNLLLeastSqTraits, 979
SNLLOptimizer, 980
Dakota::SNLLOptimizer, 984, 985
SNLLTraits, 987
SOLBase, 988
SUBMETHOD_COLLABORATIVE
Dakota, 163
sample_allocation_metric
Dakota::NonDC3FunctionTrain, 636
sample_likelihood
Dakota::NonDDREAMBayesCalibration, 643
sample_to_variables
Dakota::Analyzer, 211
sampling_reset
Dakota::NonDCubature, 639
Dakota::NonDQuadrature, 738
Dakota::NonDSampling, 758

Dakota::NonDSparseGrid, 763
scale_model
Dakota::Minimizer, 530
Dakota::NonDBayesCalibration, 633
scale_residuals
Dakota::ExperimentData, 427
scale_samples
dakota::util::DataScaler, 361
scaleModelInstance
Dakota::ScalingModel, 926
scaler_factory
dakota::util, 188
scaler_type
dakota::util::DataScaler, 362
ScalingModel, 919
Dakota::ScalingModel, 922
ScalingOptions, 926
schedule_iterators
Dakota::IteratorScheduler, 504
ScilabInterface, 927
secondary_respond_scaled2native
Dakota::ScalingModel, 923
secondary_respond_scaler
Dakota::ScalingModel, 924
send_data_using_get
Dakota::TrackerHTTP, 1043
send_data_using_post
Dakota::TrackerHTTP, 1043
SensAnalysisGlobal, 928
separable_combine
Dakota::TestDriverInterface, 1037
SeparateVariables
Dakota::JEGAOptimizer::Evaluator, 415
SeqHybridMetaIterator, 931
serial_input
library_mode.cpp, 1080
serial_interface_plugin
library_mode.cpp, 1079
SerialDirectApplicInterface, 935
serve_analyses_asynch
Dakota::ProcessHandleApplicInterface, 857
serve_analyses_synch
Dakota::ApplicationInterface, 225
serve_evaluations
Dakota::ApplicationInterface, 224
serve_evaluations_asynch
Dakota::ApplicationInterface, 230
serve_evaluations_asynch_peer
Dakota::ApplicationInterface, 230
serve_evaluations_synch
   Dakota::ApplicationInterface, 229
serve_evaluations_synch_peer
   Dakota::ApplicationInterface, 230
serve_iterators
   Dakota::IteratorScheduler, 504
set_apps_parameters
   Dakota::APPSOptimizer, 250
set_best_responses
   Dakota, 168
set_communicators_checks
   Dakota::ApplicationInterface, 225
   Dakota::ProcessHandleApplicInterface, 856
   Dakota::SysCallApplicInterface, 1025
set_compare
   Dakota, 171
set_ip_options
   Dakota::NonDQUESOBayesCalibration, 744
set_iterator
   Dakota::IteratorScheduler, 502
set_mapping
   Dakota::ActiveSubspaceModel, 201
   Dakota::AdaptedBasisModel, 206
   Dakota::DataTransformModel, 368
set_opt_params
   dakota::surrogates::GaussianProcess, 443
set_options
   dakota::surrogates::Surrogate, 1014
set_preferred_path
   Dakota::WorkdirHelper, 1069
set_problem
   Dakota::COLINApplication, 267
set_rng
   Dakota::COLINOptimizer, 272
set_scalar
   Dakota::HDF5IOHelper, 465
set_solver_parameters
   Dakota::COLINOptimizer, 272
set_u_to_x_mapping
   Dakota::ProbabilityTransformModel, 838
set_variables
   Dakota, 168
set_vector_scalar_field
   Dakota::HDF5IOHelper, 465
set_vector_vector_field
   Dakota::HDF5IOHelper, 465, 466
setup_default_optimization_params
   dakota::surrogates::GaussianProcess, 446
shape
   Dakota::Constraints, 298
SharedApproxData, 937
SharedApproxData, 941
SharedC3ApproxData, 943
SharedPecosApproxData, 947
SharedResponseData, 950
SharedResponseDataRep, 952
SharedSurfpackApproxData, 954
   Dakota::SharedSurfpackApproxData, 956
SharedVariablesData, 956
SharedVariablesDataRep, 964
shubert1D
   Dakota::TestDriverInterface, 1037
simple_corr
   Dakota::SensAnalysisGlobal, 930
SimulationModel, 970
SimulationResponse, 973
size_level_index_vector
   dakota::surrogates, 183
smooth_herbie
   Dakota::TestDriverInterface, 1037
smooth_herbie1D
   Dakota::TestDriverInterface, 1037
SoleilDirectApplicInterface, 990
solution_level_index
   Dakota::Model, 563
solution_levels
   Dakota::Model, 563
solve
   dakota::util::CholeskySolver, 265, 266
dakota::util::LinearSolverBase, 519, 520
solver_factory
   dakota::util, 189
solver_setup
   Dakota::COLINOptimizer, 272
solver_type
   dakota::util::LinearSolverBase, 519
spawn_analysis_to_shell
   Dakota::SysCallApplicInterface, 1025
spawn_evaluation_impl
   Dakota::COLINApplication, 267
spawn_evaluation_to_shell
   Dakota::SysCallApplicInterface, 1025
spawn_input_filter_to_shell
   Dakota::SysCallApplicInterface, 1025
spawn_output_filter_to_shell
   Dakota::SysCallApplicInterface, 1026
SpawnApplicInterface, 992
split_filenames
  Dakota::ProgramOptions, 863
split_wildcard
  Dakota::WorkdirHelper, 1068
StandardizationScaler, 993
dakota::util::StandardizationScaler, 993
StanfordPSAAP, 191
StanfordPSAAP::SoleilDirectApplicInterface
derived_map_ac, 991
test_local_evaluations, 991
wait_local_evaluations, 991
start_dakota_heartbeat
  Dakota, 171
start_grid_computing
  Dakota, 171
steady_state_diffusion_1d
  Dakota::TestDriverInterface, 1036
steel_column_cost
  Dakota::TestDriverInterface, 1036
stepsPerVariable
  Dakota::ParamStudy, 819
stop_evaluation_servers
  Dakota::ApplicationInterface, 224
stop_grid_computing
  Dakota, 171
StringScale, 994
subModel
  Dakota::NestedModel, 590
submethod_map
  Dakota, 174
submit
  Dakota::APPSEvalMgr, 248
subordinate_iterator
  Dakota::Model, 561
subordinate_model
  Dakota::Model, 562
subordinate_models
  Dakota, 565
SurfpackApproximation, 995
  Dakota::SurfpackApproximation, 997
SurrBasedGlobalMinimizer, 999
SurrBasedGlobalTraits, 1000
SurrBasedLocalMinimizer, 1001
SurrBasedMinimizer, 1007
Surrogate, 1012
surrogate_model
  Dakota::Model, 562
SurrogateModel, 1015
surrogates_to_surf_data
  Dakota::SurfpackApproximation, 998
SurrogatesGPApprox, 1020
Dakota::SurrogatesGPApprox, 1021
SurrogatesPolyApprox, 1021
Dakota::SurrogatesPolyApprox, 1023
svd
  Dakota, 164
symmetric_eigenvalue_decomposition
  Dakota, 170
synchronize
  Dakota::ApplicationInterface, 223
synchronize_derivatives
  Dakota::Model, 569
synchronize_nowait
  Dakota::ApplicationInterface, 224
synchronous_local_analyses
  Dakota::ProcessApplicInterface, 853
synchronous_local_analysis
  Dakota::DirectApplicInterface, 394
Dakota::GridColumnApplicInterface, 460
Dakota::ProcessHandleApplicInterface, 856
Dakota::SysCallApplicInterface, 1024
synchronous_local_evaluations
  Dakota::ApplicationInterface, 228
SysCallApplicInterface, 1023
TANA3Approximation, 1028
TKFactoryDIPC, 1038
TKFactoryDIPCLogit, 1039
TPLDataTransfer, 1040
TabularDataTruncated, 1026
TabularReader, 1027
TabularWriter, 1027
TaylorApproximation, 1030
terminate_modelcenter
  Dakota::ParallelLibrary, 808
test_local_evaluation_sequence
  Dakota::SysCallApplicInterface, 1024
test_local_evaluations
  SIM::ParallelDirectApplicInterface, 797
  SIM::SerialDirectApplicInterface, 936
  StanfordPSAAP::SoleilDirectApplicInterface, 991
TestDriverInterface, 1031
ToDoubleMatrix
  Dakota::JEGAOptimizer, 509
tokenize_env_path
  Dakota::WorkdirHelper, 1070
TrackerHTTP, 1041
TrackerHTTP::Server, 936
TraitsBase, 1043
transform_response_map
  Dakota::DataTransformModel, 368
transform_samples
  Dakota::NonDSampling, 757
trendOrder
  Dakota::GaussProcApproximation, 451
truth_model
  Dakota::Model, 562
type_name_bimap
  dakota::util, 190

uncentered_to_centered
  Dakota::NonDMultilevelSampling, 720
uncertain_vars_to_subspace
  Dakota::ActiveSubspaceModel, 201
  Dakota::AdaptedBasisModel, 205
update_approximation
  Dakota::ApproximationInterface, 243
  Dakota::DataFitSurrModel, 321
update_augmented_lagrange_multipliers
  Dakota::SurrBasedMinimizer, 1010
update_expansion
  Dakota::NonDExpansion, 654
update_filter
  Dakota::SurrBasedMinimizer, 1011
update_from_model
  Dakota::DataFitSurrModel, 326
  Dakota::RecastModel, 890
update_from_subordinate_model
  Dakota::DataTransformModel, 366
  Dakota::Model, 563
  Dakota::ProbabilityTransformModel, 837
update_lagrange_multipliers
  Dakota::SurrBasedMinimizer, 1010
update_level_data
  Dakota::NonDLocalReliability, 699
update_model
  Dakota::DataFitSurrModel, 325
update_mpp_search_data
  Dakota::NonDLocalReliability, 698
update_penalty
  Dakota::SurrBasedLocalMinimizer, 1006
update_quasi_hessians
  Dakota::Model, 570
update_response
  Dakota::Model, 569
update_samples_from_order_decrement
  Dakota::NonDExpansion, 655
usage
  Dakota::GetLongOpt, 455

UsageTracker, 1045
  Dakota::UsageTracker, 1046
useDerivs
  Dakota::NonDExpansion, 656
user_auto_id
  Dakota::Iterator, 497
  Dakota::Model, 568
user_proposal_covariance
  Dakota::NonDQUESOBayesCalibration, 744
uses_method
  Dakota::Iterator, 495

VLUncertainInt
  Dakota, 178
VLUncertainReal
  Dakota, 177
VLUncertainStr
  Dakota, 178
VLInt, 1059
VLreal, 1059
VLstr, 1059
VPSApproximation, 1060
VPSInstance
  Dakota::VPSApproximation, 1063
VPSmodel_apply
  Dakota::VPSApproximation, 1063
value
  dakota::surrogates::GaussianProcess, 441
dakota::surrogates::GP_Objective, 456
dakota::surrogates::PolynomialRegression, 831
dakota::surrogates::Surrogate, 1014
Valueless
  Dakota::GetLongOpt, 453
values_to_ranks
  Dakota::SensAnalysisGlobal, 930
Var_icheck, 1046
var_mp_cbound
  Dakota, 180
var_mp_check_cau
  Dakota, 179
var_mp_check_ceu
  Dakota, 180
var_mp_check_cv
  Dakota, 178
var_mp_check_dai
  Dakota, 179
var_mp_check_daur
  Dakota, 179
var_mp_check_daus
Dakota, 179  
var_mp_check_deui  
   Dakota, 180  
var_mp_check_deur  
   Dakota, 180  
var_mp_check_deus  
   Dakota, 180  
var_mp_check_dset  
   Dakota, 179  
var_mp_drange  
   Dakota, 180  
Var_rcheck, 1047  
Variables, 1047  
variables_expand  
   Dakota::DataTransformModel, 366  
variables_resize  
   Dakota::RandomFieldModel, 877  
variables_scaler  
   Dakota::ScalingModel, 924  
variables_to_sample  
   Dakota::Analyzer, 215  
variablesCompsTotals  
   Dakota::SharedVariablesDataRep, 969  
variance  
   dakota::util, 188  
vars_mapping  
   Dakota::ActiveSubspaceModel, 201  
   Dakota::AdaptedBasisModel, 205  
   Dakota::RandomFieldModel, 878  
vars_u_to_x_mapping  
   Dakota::ProbabilityTransformModel, 837  
vars_x_to_u_mapping  
   Dakota::ProbabilityTransformModel, 837  
Verification, 1057  
volumetric_quality  
   Dakota::PStudyDACE, 864  
wait_local_evaluation_sequence  
   Dakota::SysCallApplicInterface, 1024  
wait_local_evaluations  
   StanfordPSAAP::SoleilDirectApplicInterface, 991  
weight_model  
   Dakota::LeastSq, 515  
   Dakota::NonDBayesCalibration, 633  
weightModelInstance  
   Dakota::WeightingModel, 1065  
WeightingModel, 1064  
which  
   Dakota::WorkdirHelper, 1068  
WorkdirHelper, 1065