



DAKOTA 101

Calibration

<http://www.cs.sandia.gov/dakota>

Learning goals:

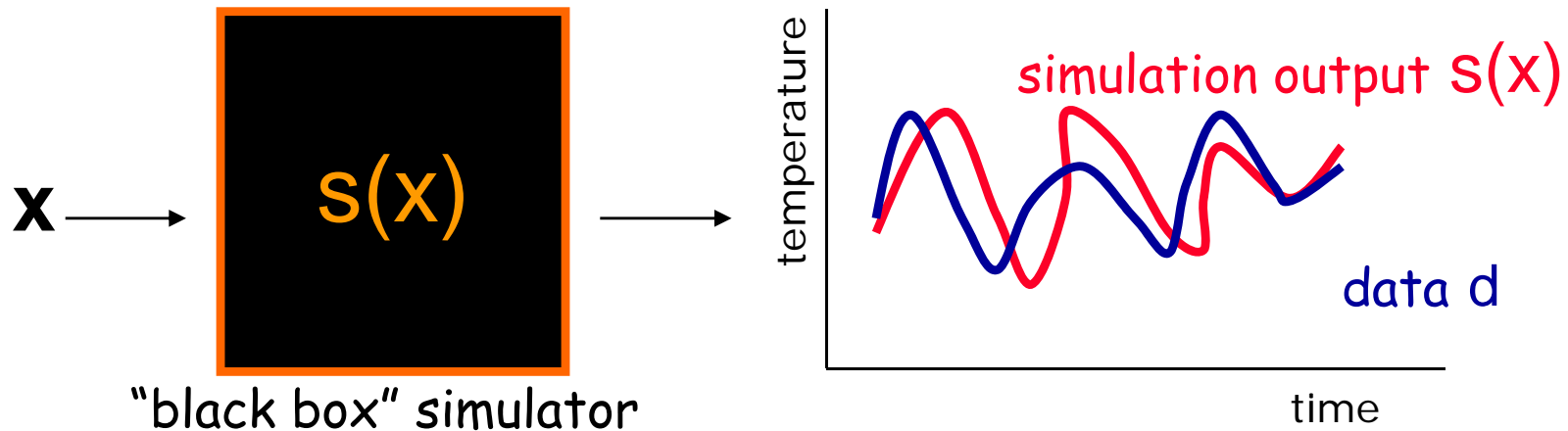
- Define calibration and know when and why to apply it
- Run a nonlinear least squares problem and understand how to specify residuals or model/data to DAKOTA
- Understand the problem formulation in DAKOTA

What is calibration?

$$f(x) = \sum_{i=1}^n (s_i(x) - d_i)^2$$

simulation output that depends on x

given data



- **Calibration:** Adjust model parameters x to maximize agreement with a set of experimental data.
- A.K.A. parameter estimation, parameter identification, systems identification, nonlinear least-squares, inverse problem.



Why use calibration?

- Ensure sufficient simulation code predictive capability
- Decrease the amount of info lost due to using a model instead of the “truth” (minimize discrepancy)
- Increased understanding of design space
- Find parameters yielding improved model robustness
- **Calibration is not validation!** Separate data should be used for calibration vs. validation.



Nonlinear Least Squares (NLLS)

- Calibration problems are often formulated to minimize the two norm of the error between the model and data: *minimize*

$$f(x) = \frac{1}{2} r(x)^T r(x) = \frac{1}{2} [s(x) - d]^T [s(x) - d] = \frac{1}{2} \sum_{i=1}^n (s_i(x) - d_i)^2$$

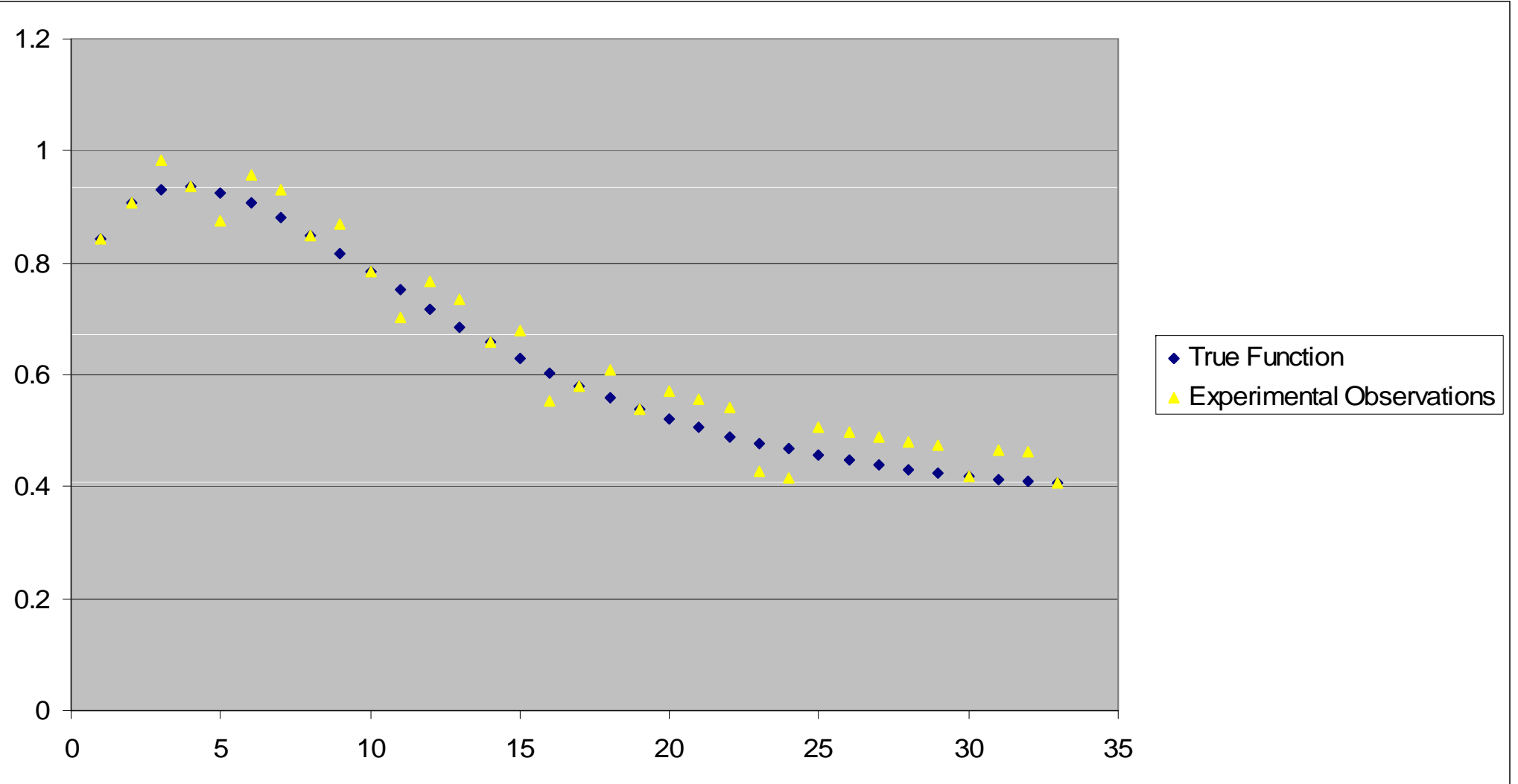
- Example: `osborne1` analytic test problem, with $i = 1, \dots, 33$:

$$r_i(x) = \underbrace{\left(x_1 + x_2 e^{t_i x_4} + x_3 e^{t_i x_5} \right)}_{\text{model/simulation}} - \underbrace{d_i}_{\text{data}}; \quad t_i = -10(i-1)$$

- A specialized class of optimization algorithms exploit this structure for efficient solution without second derivative information (more coming soon)



Example Data Set (osborne1)



Exercise: Open with an editor and terminal run `osborne1.in`



```
method,
  nl2sol
  max_iterations =
  convergence_tolerance =
model,
  single
variables,
  continuous_design = 5
  cdv_initial_point      .5      1.5      -1      .01      .02
  cdv_lower_bounds .3      0.7      -2      .001     .001
  cdv_upper_bounds .6      1.8      0       .2       .23
  cdv_descriptor         'x1'    'x2'    'x3'    'x4'    'x5'
interface,
  system
  analysis_driver = './osborne1'
responses,
  num_least_square_terms = 33
  analytic_gradients
  no_hessians
```

Method independent options

Note: application must return 33 residuals $r_i = s_i - d_i$ to DAKOTA



Output for osborne1

```
> cd osborne
```

```
> dakota -i osborne1.in
```

```
<<<<< Function evaluation summary: 27 total (26 new, 1 duplicate)
```

```
<<<<< Best parameters =
```

```
3.7541004764e-01 cdv_1
```

```
1.9358463401e+00 cdv_2
```

```
-1.4646865611e+00 cdv_3
```

```
1.2867533504e-02 cdv_4
```

```
2.2122702031e-02 cdv_5
```

```
<<<<< Best residual norm = 7.3924926090e-03; 0.5 * norm
```

```
<<<<< Best residual terms =
```

```
2.5698266188e-03
```

```
-4.4759880011e-03
```



NLLS Exercise 1

- **Goal: Be able to formulate and run a NLLS calibration; understand f vs. residuals**
 - Identify the input parameters, data set, and least squares terms
- **Run the osborne1 example**
- **Revise the input deck to use an application that returns separate model (osborne1b) and data (osborne1_y.dat)**

Using a Separate Data Source (d_i): osborne1b(b)



```
method nl2sol
  output silent
  convergence_tolerance = -1.

variables,
  continuous_design = 5
  initial_point .5 1.5 .01 -1 .02
  lower_bounds .2 1.0 .005 -1.5 .01
  upper_bounds .6 2.0 .012 1.5 .05

interface,
  system
  analysis_driver = './osborne1b'

responses,
  num_least_squares_terms = 33
  least_squares_data_file 'osborne1_y2.dat'
  analytic_gradients      # For finite differences, comment this
  # numerical_gradients # and uncomment this line.
  no_hessians
```



Extra Slides

Least Squares Objective Function



$$\min \sum_{i=1}^N w_i(T_i) \sum_{t=1}^{T_i} (s_i(t; x) - e_i(t))^2$$

N = number of tests

T_i = (relevant) number of experimental values for test **i**

w_i(T_i) = weighting factor (depends on number of experimental points)

S_i(t;x) = simulated value, calculated with parameters **x**, corresponding to experimental point **t** for experiment **i**

e_i(t) = test value of point **t** in test **i**



Least-squares Structure

- When minimizing $f(x)$ with gradient-based methods, can take advantage of the form of its derivatives:

$$f(x) = \frac{1}{2} r(x)^T r(x) = \frac{1}{2} [s(x) - d]^T [s(x) - d]$$

$$\nabla f(x) = J(x)^T r(x); \quad J_{ij} = \frac{\partial r_i}{\partial x_j}$$

$$\nabla^2 f(x) = J^T J + \sum_{i=1}^n r_i(x) \nabla^2 r_i(x)$$

Algorithms vary in how they approximate this Hessian.

Hessian Approximations



$$\nabla^2 f(x) = J^T J + \sum_{i=1}^n r_i(x) \nabla^2 r_i(x)$$

Gauss-Newton: $J(x)^T J(x)$

Levenberg-Marquardt: $J(x)^T J(x) + \mu I$, with $\mu \geq 0$

NL2SOL: $J(x)^T J(x) + S$,

with $S = 0$ or $S =$ Quasi-Newton approximation to $\sum_{i=1}^n f_i(x) \nabla^2 f_i(x)$

DAKOTA Method Selection



Calibration Method	Step Control	Unconstrained	Bounds	Linear/ Nonlinear
<code>nl2sol</code>	trust region	X	X	
<code>nlssol</code>	line search	X	X	X
<code>optpp_g_newton</code>	trust region or line search	X	X	X

NL2SOL can handle highly nonlinear problems.



Examples in nlls.tgz

```
gzip -dc nlls.tgz | tar xf -
```

gives directory nlls containing:

lls	analysis driver compiled from lls.c
lls.c	source for lls
lls.in	DAKOTA input file using lls
osborne1	python script as analysis driver
osborne1[ab]	variations on osborne1 script
osborne1*.in	input files using osborne1*
osborne1	y right-hand side file (data) for osborne1b and osborne1bb