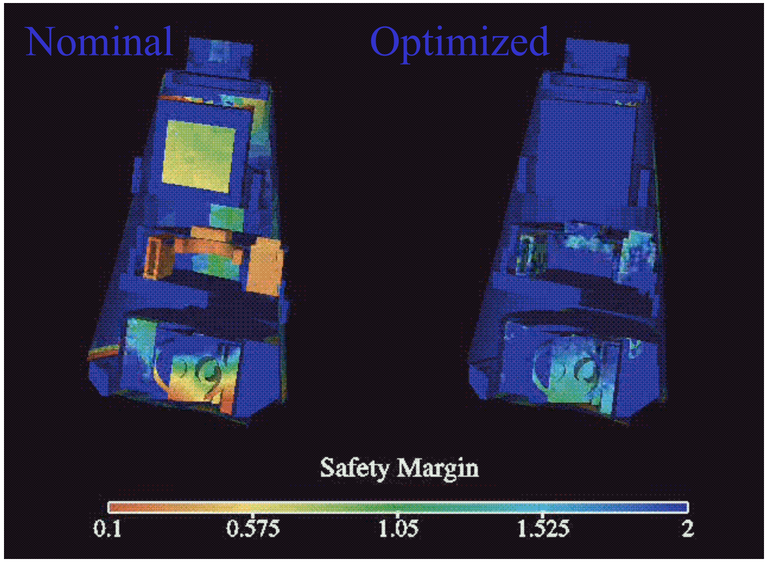
SAND2009-0463P

DAKOTA Training

Optimization

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



*Sandia is a multiprogram laboratory operated by Sandia Corporation, a Lockheed Martin Company, for the United States Department of Energy's National Nuclear Security Administration under contract DE-AC04-94AL85000.

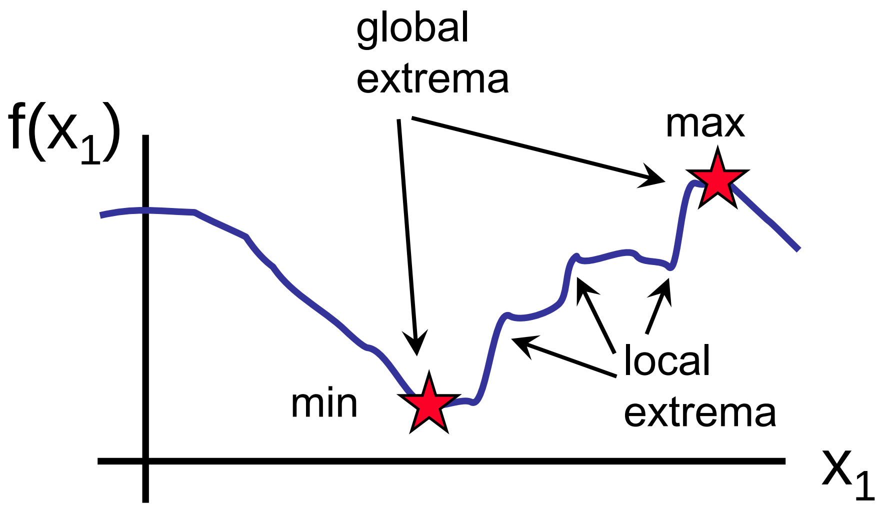
Optimization Learning Goals

Use optimization methods to find parameters yielding the best performing or minimum cost design. Or maximize agreement between simulation and experimental results (calibration).

- **Survey optimization terminology, problem formulations, and sample problems**
- **Understand considerations for selecting an optimization method**
- **Run DAKOTA examples of optimization methods**
  - **Gradient-based methods**
  - **Non-gradient pattern search and genetic algorithms**
  - **Constrained optimization**
- **Using least-squares solvers for model calibration (parameter estimation)**

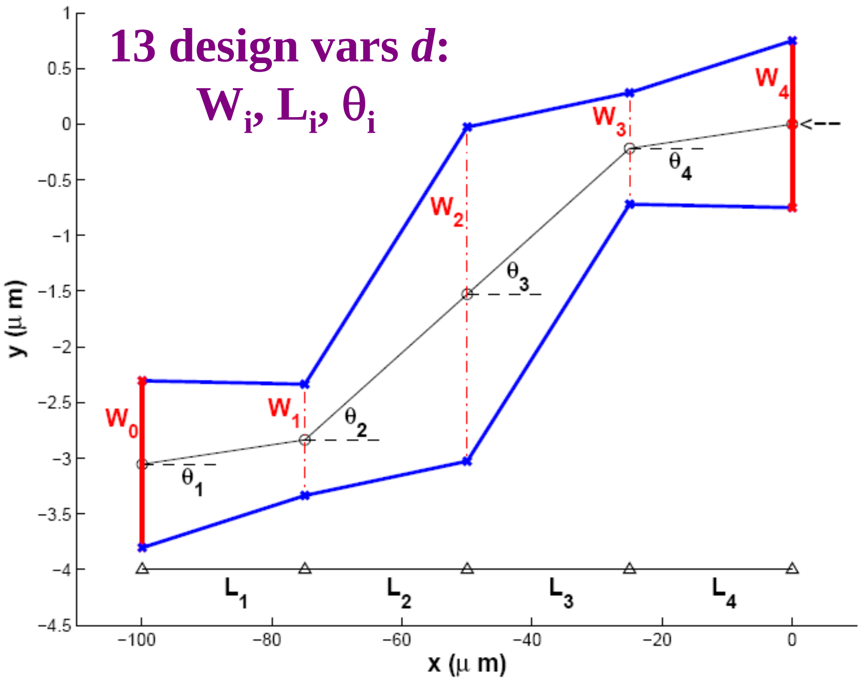
Optimization

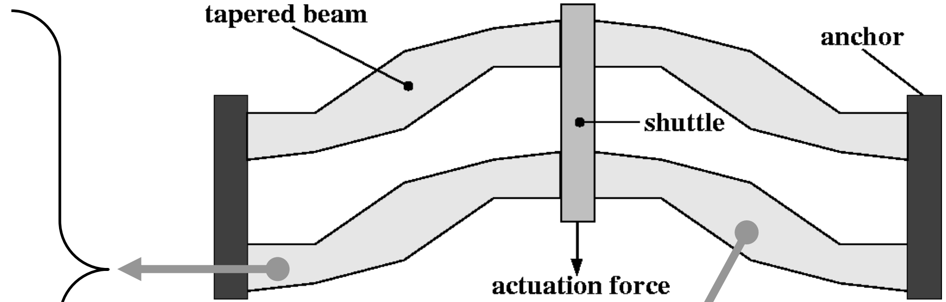
- **GOAL: Vary parameters to extremize objectives, while satisfying constraints to find (or tune) the best design, estimate best parameters, analyze worst-case surety, e.g., determine:**
  - **delivery network maximizing profit / minimizing environ. impact**
  - **case geometry that minimizes drag and weight, yet is sufficiently strong and safe**
  - **AF&F with maximum design margin (title slide)**
  - **material atomic configuration of minimum energy**

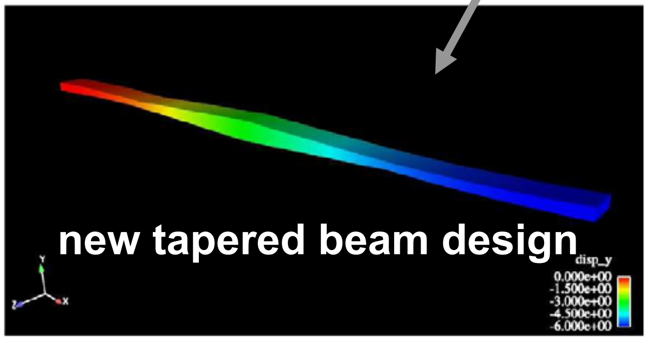


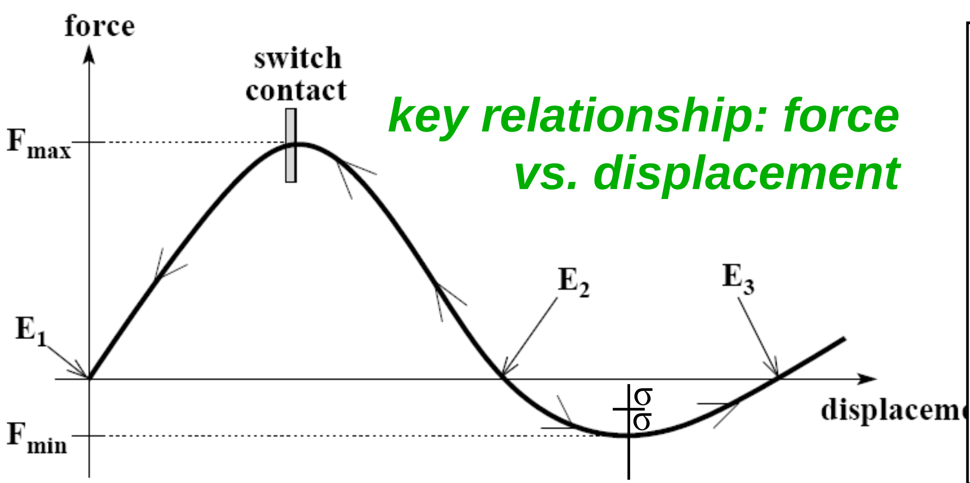
Some applications: local improvement suffices; others: must find global minimum at any cost

MEMS Switch Design: Geometry Optimization









key relationship: force vs. displacement

Typical design specifications:

- **actuation force F_{min} reliably 5 μN**
- **bistable ($F_{max} > 0$, $F_{min} < 0$)**
- **maximum force: $50 < F_{max} < 150$**
- **equilibrium E2 < 8 μm**
- **maximum stress < 1200 MPa**

Optimization for Lockheed-Martin F-35 External Fuel Tank Design

This wind tunnel model of F-35 features an optimized external fuel tank.

F-35: stealth and supersonic cruise

~ $20 billion cost

~ 2600 aircraft (USN, USAF, USMC, UK & other foreign buyers)

LM CFD code:

- ***Expensive*: 8 hrs/job on 16 processors**
- **Fluid flow around tank *highly sensitive* to shape changes**

“Lockheed Martin Aeronautics conducted a trade study for the F-35 Joint Strike Fighter (JSF) aircraft to design the external fuel tank for improved performance, store separation, and flutter. **CFD was used in conjunction with Sandia National Laboratories’ Dakota optimization code to determine the optimal shape of the tank that minimizes drag for maximum range and minimizes yawing moment for separation of adjacent stores.** Data obtained at several wind tunnel facilities verified the predicted performance of the new aeroshaped, compartmented tank for separation and flutter, as well as acceptable characteristics for loads, stability, and control.” -- Dec. 2004 *Aerospace America*, p. 22

Optimization Problem Formulation

Minimize:	$f(x_1, \ldots, x_N)$	*Objective function(s)**
Subject to:	$g_{LB} \le g(x) \le g_{UB}$	*Nonlinear inequality constraints*
	$h(x) = h_E$	*Nonlinear equality constraints*
		(Metrics above are typically computed by or extracted from a simulation code)
		(Analytic metrics below are typically specified directly in a DAKOTA input deck)
	$A_I x \le b_I$	*Linear inequality constraints*
	$A_E x = b_E$	*Linear equality constraints*
	$x_{LB} \le x \le x_{UB}$	*Bound constraints*

** In practice, we can have multiple f-values in the objective function (aka “multiobjective optimization”), and multiple constraints of each type.*

Basic Constraint Lingo

Unconstrained problem: neither bound constraints nor linear/ nonlinear constraints

Bound-constrained problem: bound (variable space x) constraints only (no linear/nonlinear constraints)

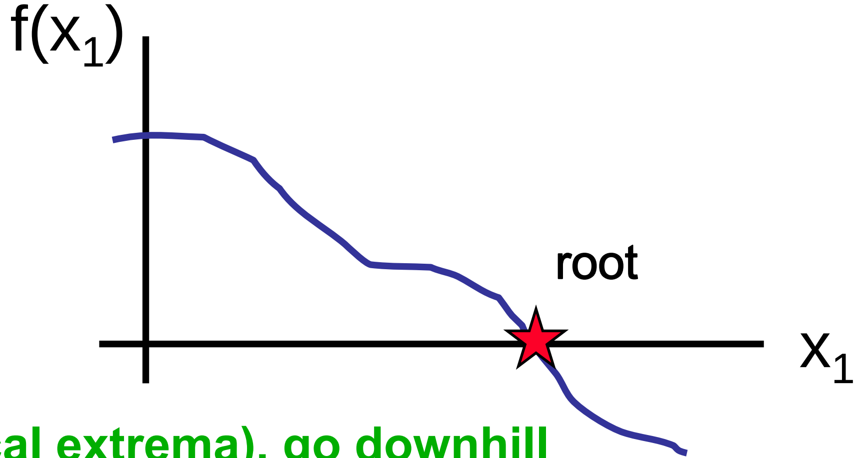
Linearly-constrained problem: the constraints are linear with respect to the x-variables (may also have bound constraints)

Nonlinearly-constrained problem: the g(x) and h(x) constraints, nonlinear w.r.t. the x variables, are present (may also have bound constraints); *perhaps most typical in engineering applications*

Gradient-based Optimization

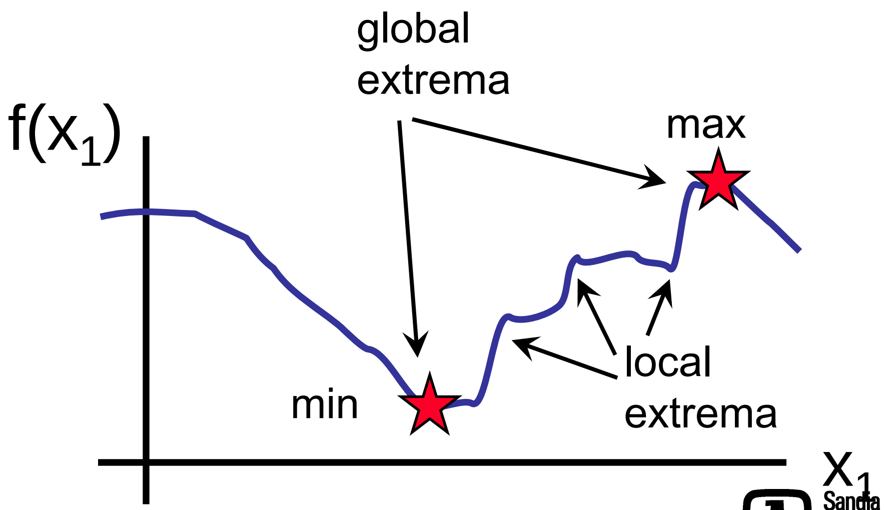
Modify Newton's root-finding method for solving f(x) = 0.

$$x_{k+1} = x_k - \frac{f(x_k)}{f'(x_k)}$$



For optimization: find zeros of f'(x) = 0 (local extrema), go downhill

$$x_{k+1} = x_k - \frac{f'(x_k)}{f''(x_k)}$$



These derivatives extend to gradients and Hessians in the multivariate case:

$$\nabla_x f(x), \quad \nabla_x^2 f(x)$$

DAKOTA Optimization Methods

Gradient-based methods

(DAKOTA will compute finite difference gradients and FD/quasi-Hessians if necessary)

- *DOT (various constrained)*
- CONMIN (FRCG, MFD)
- *NPSOL (SQP)*
- *NLPQL (SQP)*
- OPT++ (CG, Newton)

Calibration (least-squares)

- NL2SOL (GN + QH)
- *NLSSOL (SQP)*
- OPT++ (Gauss-Newton)

Derivative-free methods

- COLINY (PS, APPS, Solis-Wets, COBYLA2, EAs, DIRECT)
- JEGA (single/multi-obj GAs)
- EGO (efficient global opt via Gaussian Process models)
- DIRECT (Gablonsky)
- OPT++ (parallel direct search)
- *TMF (templated meta-heuristics framework)*

Surrogate-based optimization

Sandia National Laboratories

Getting Ready for an Optimization Study with DAKOTA

Key decision criteria:

- Local and global sensitivity study data; trend and smoothness
- Simulation expense
- Constraint types present
- Goal: local optimization (improvement) or global optimization (best possible)

Unconstrained or bound-constrained problems:

- Smooth and cheap: nearly any method; gradient-based methods will be fastest
- Smooth and expensive: gradient-based methods
- Nonsmooth and cheap: non-gradient methods such as pattern search (local opt), genetic algorithms (global opt), DIRECT (global opt), or surrogate-based optimization (quasi local/global opt)
- Nonsmooth and expensive: surrogate-based optimization (SBO)*

Nonlinearly-constrained problems:

- Smooth and cheap: gradient-based methods
- Smooth and expensive: gradient-based methods
- Nonsmooth and cheap: non-gradient methods w/ penalty functions, SBO
- Nonsmooth and expensive: SBO

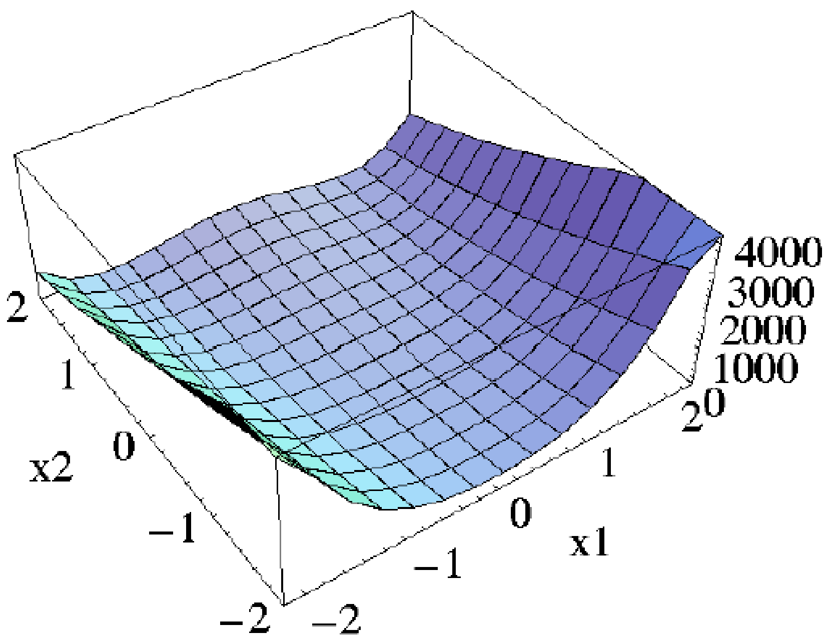
See guidance in User's Manual, Chapter 19

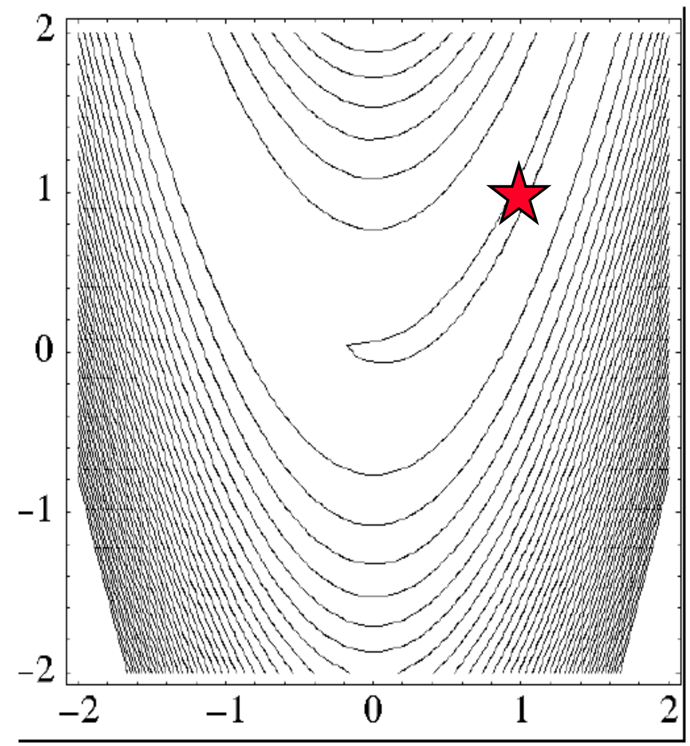
Optimization Learning Goals

Use optimization methods to find parameters yielding the best performing or minimum cost design. Or maximize agreement between simulation and experimental results (calibration).

- **Survey optimization terminology, problem formulations, and sample problems**
- **Understand considerations for selecting an optimization method**
- **Run DAKOTA examples of optimization methods**
  - **Gradient-based methods**
  - **Non-gradient pattern search and genetic algorithms**
  - **Constrained optimization**
- **Using least-squares solvers for model calibration (parameter estimation)**

Recall: Rosenbrock Function





minimize $f(x1,x2) = 100*(x2-x1*x1)^2 + (1-x1)^2$

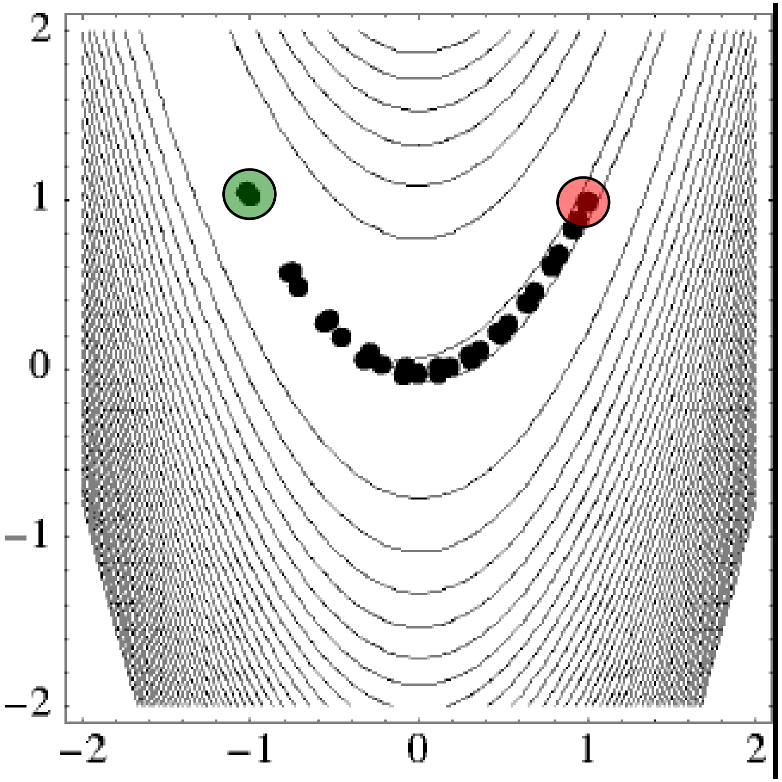
s.t. $-2 \leq x1 \leq 2$

$-2 \leq x2 \leq 2$

Optimum point: (x1,x2) = (1,1); f(1,1) = 0.0

Optimize Rosenbrock (Gradient-based Method)

- View and run examples/tutorial/ dakota_rosenbrock_grad_opt.in (see User's Manual 2.4.3)
- Started at (x1,x2) = (-1.0, 1.2) (try other starting points)
- Search algorithm follows the general descent direction "around the bend" of the Rosenbrock function.
- Gradient-based optimization is very efficient: ~30-100 evaluations of the function values needed to find the minimum here.
- Next: dakota_rosenbrock_ps_opt.in, dakota_rosenbrock_ea_opt.in



Compare Gradient-based to Derivative-free Inputs

dakota_rosenbrock_grad_opt.in

```
strategy,
  single_method
  graphics
  tabular_graphics_data

method,
  conmin_frcg
    max_iterations = 100
    convergence_tolerance = 1e-4

variables,
  continuous_design = 2
    cdv_initial_point -1.2 1.0
    cdv_lower_bounds -2.0 -2.0
    cdv_upper_bounds 2.0 2.0
    cdv_descriptors 'x1' 'x2'

interface,
  direct
    analysis_driver = 'rosenbrock'

responses,
  num_objective_functions = 1
  numerical_gradients
    method_source dakota
    interval_type forward
    fd_gradient_step_size = 1.e-5
  no_hessians
```

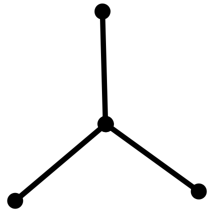
dakota_rosenbrock_ps_opt.in

```
strategy,
  single_method
  graphics
  tabular_graphics_data

method,
  coliny_pattern_search
    max_iterations = 1000
    max_function_evaluations = 2000
    solution_accuracy = 1e-4
    initial_delta = 0.5
    threshold_delta = 1e-4
    exploratory_moves basic_pattern
    contraction_factor = 0.75

variables,
  continuous_design = 2
  cdv_initial_point 0.0 0.0
  cdv_lower_bounds -2.0 -2.0
  cdv_upper_bounds 2.0 2.0
  cdv_descriptors 'x1' 'x2'

interface,
  direct
    analysis_driver = 'rosenbrock'
responses,
  num_objective_functions = 1
  no_gradients
  no_hessians
```

Rosenbrock: Pattern Search

- **Copy and run examples/tutorial/ dakota_rosenbrock_ps_opt.in**
  (pattern search: non-gradient method)

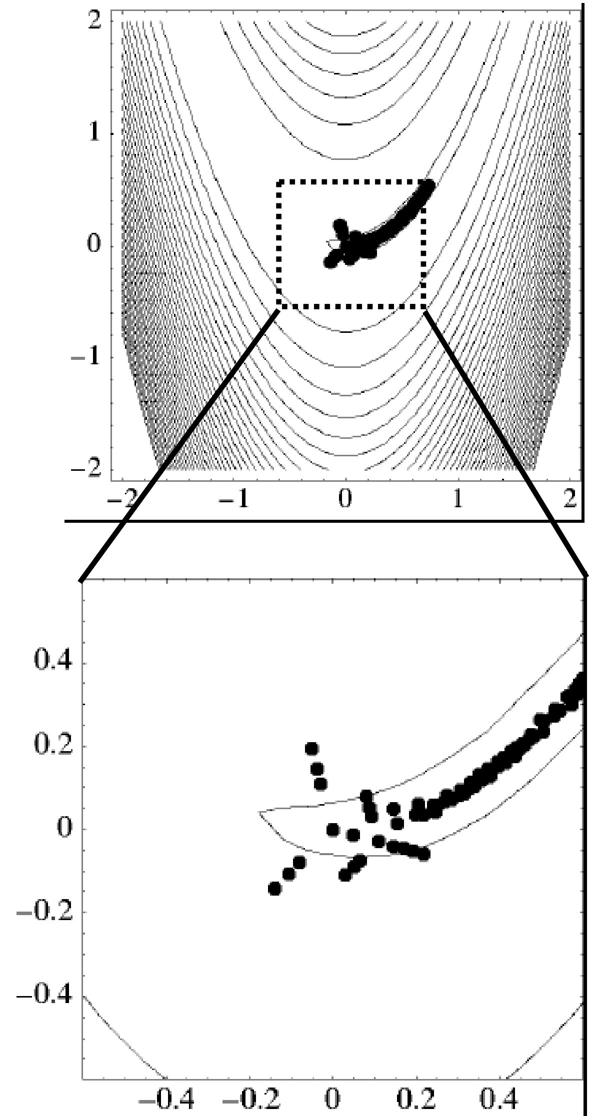
- **Stencil-based with expansion/contraction (reliable local convergence)**



- **Started at (x1,x2) = (0,0)**

- **Search algorithm has made some progress toward the minimum after generating ~2000 function values, but still not converged to the minimum.**

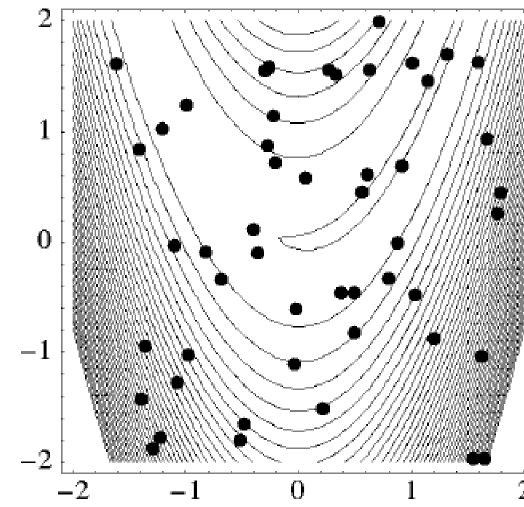
- **Next: dakota_rosenbrock_ea_opt.in**



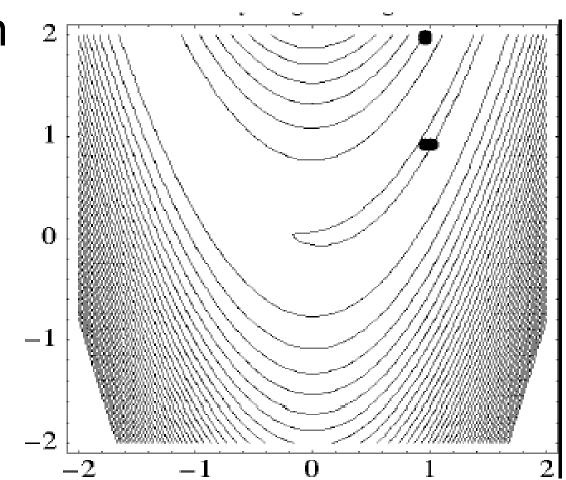
Rosenbrock: Evolutionary Algorithm

- **See examples/tutorial/ dakota_rosenbrock_ea_opt.in** **(Genetic algorithm (GA): non-gradient method, more global search than pattern search)**
- **Started with 50 random points in the parameter space; fitness, selection, reproduction**
- **GA search algorithm run to generate 10,000 f-values. 46 of the 50 samples have settled close to the true optimum**
- **GAs and other global optimizers are great for problems with many local minima in which a gradient-based optimizer might get trapped.**

Initial population (50 random samples)

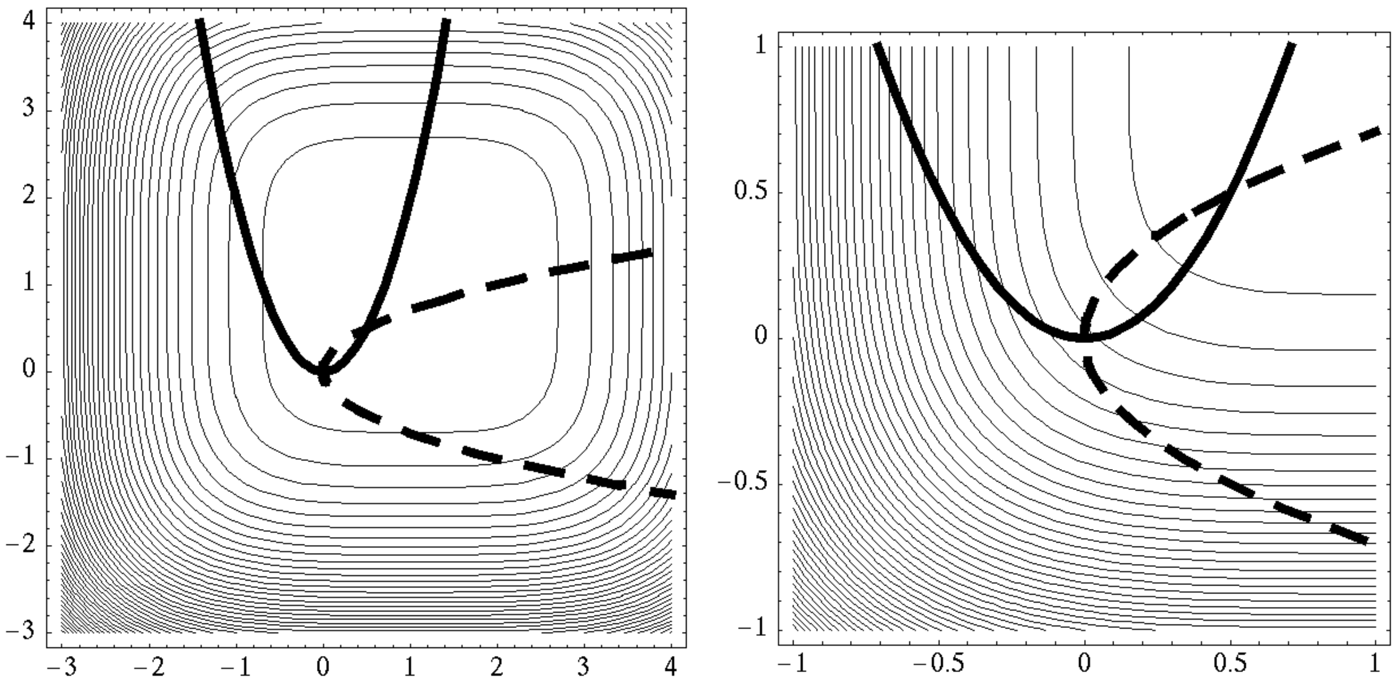


Final population (46 of 50 near minimum)



Gradient-based Constrained Optimization

- GOAL: Minimize, subject to nonlinearly-constrained feasible region (see textbook example, page 27 of User's Manual; 2.2).
- See examples/tutorial/dakota_textbook.in (see 2.4.4); notice constraints in input deck responses
- Modify to use fork interface with parameters_file and results_file, file_tag, file_save
- Inspect a results.out.x file to see the derivatives and constraints being returned to DAKOTA



Summary on Optimization Methods

- Selecting the right optimization method that matches the particular attributes of your problem is critical, especially if your simulation code is expensive!

- You won't have a good idea of the best optimization method UNLESS you perform some local and global sensitivity studies BEFORE you start optimizing.

DAKOTA Training

Calibration

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

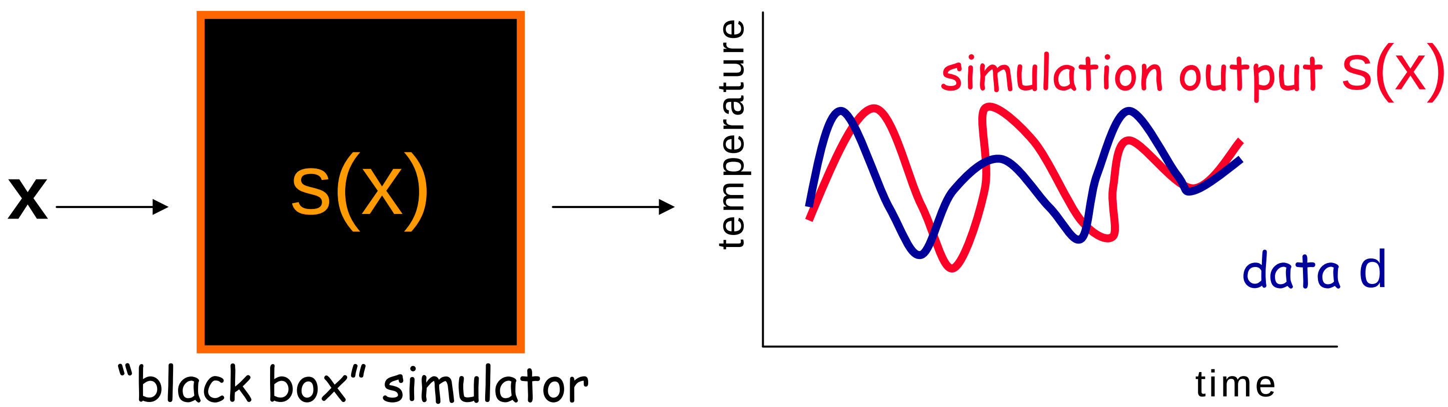
Learning goals:

- **Understand what calibration *is and is not* and why it is important**
- **Differentiate between optimization and least-squares calibration mathematical formulations**
- **Use various DAKOTA methods to perform model calibration to data**
- **How to run DAKOTA: specify (input deck) and run an analysis**

*Sandia is a multiprogram laboratory operated by Sandia Corporation, a Lockheed Martin Company, for the United States Department of Energy's National Nuclear Security Administration under contract DE-AC04-94AL85000.

What is calibration?

$$f(x) = \sum_{i=1}^{n} (\underbrace{s_i(x)}_{\text{simulation output that depends on x}} - \underbrace{d_i}_{\text{given data}})^2$$



- **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.

Sandia National Laboratories

Nonlinear Least Squares

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

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

- **Example: `osborne1` analytic test problem, with *i = 1,…,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)**

DAKOTA Input: osborne1

```
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

Run Dakota on osborne1

```
> cd nlls
> 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
```

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_y'
  analytic_gradients    # For finite differences, comment this
  # numerical_gradients # and uncomment this line.
  no_hessians
```

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
`nl2sol`	trust region	X	X	
`nlssol`	line search	X	X	X
`optpp_g_newton`	trust region or line search	X	X	X

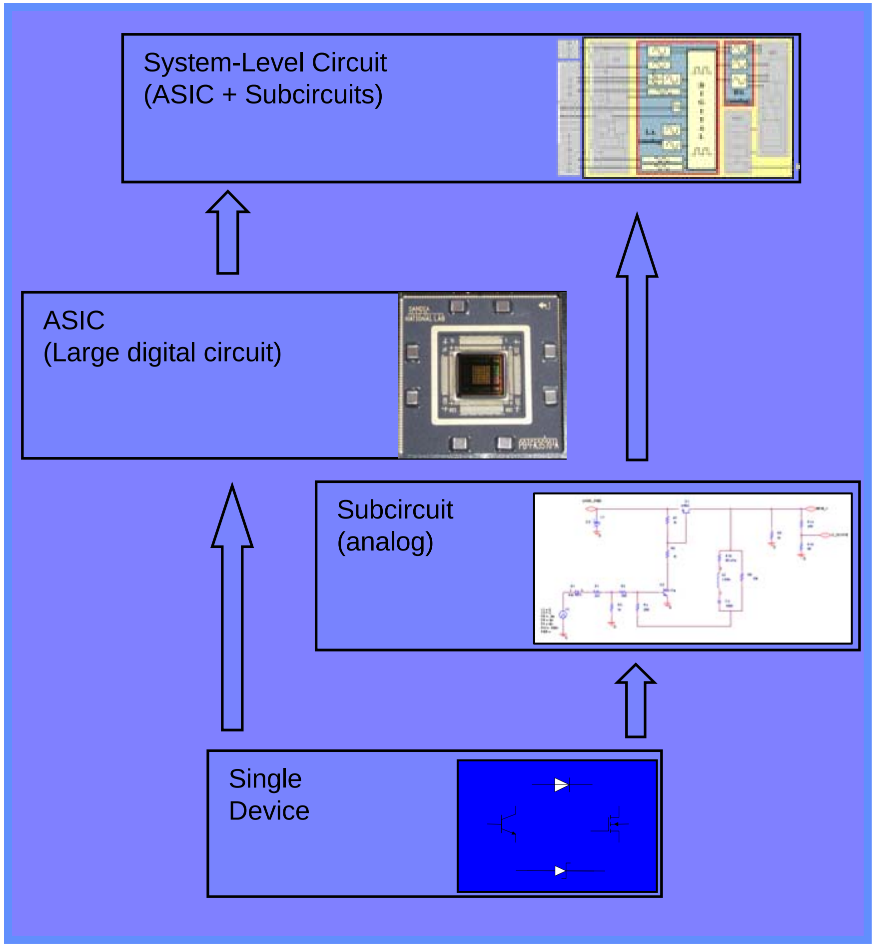
NL2SOL can handle highly nonlinear problems.

Confidence Intervals on Params

```
dakota lls.in

...
<<<<< Best parameters =
3.9975104529e-01 cdv_1
7.8306751279e-01 cdv_2
-1.1317783545e-01 cdv_3
...

Confidence Interval for cdv_1
is [ -5.2378467908e-01, 1.3232867697e+00 ]

Confidence Interval for cdv_2
is [ -9.4840422538e-01, 2.5145392510e+00 ]

Confidence Interval for cdv_3
is [ -1.5865346409e+00, 1.3601789700e+00 ]
```

Example: Electrical Application

- Radiation-aware electrical models
- Predict responses of electrical devices in hostile environments
- Building blocks of a large electrical system being examined hierarchically
- Access to code and model developers



Hierarchical Electrical Model

Least Squares Objective Function

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

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

Parameters

- Device model has ~30 parameters
- Parameters ranked by modeler
  - How much does the model rely on the parameter being chosen correctly?
  - How uncertain are we about the current value being used?
- Selected
  - 8 parameters for calibration
  - Either physical parameters or covering “missing” physics
- Modeler provided bounds and starting points by “hand tuning” process

Typical Xyce input file
dakota_xyce.in

```
method,
        nl2sol
        max_iterations = 50                    Method independent
        convergence_tolerance = 1.0e-4              options
model,
        single
variables,
        continuous_design = 8
        cdv_initial_point     5e-3  1.4e-3  1e-8  2e-8  4e-3 1.6e-3  1e-9 2e-9
        cdv_lower_bounds   2.5e-4  1e-4  1e-9  1e-9  2.5e-4  1e-4  1e-9  1e-9
        cdv_upper_bounds   3.55e-3  1.3e-3  1e-5  3.55e-3  1e-3  1e-3  1e-5  1e-5
        cdv_descriptor      'cdn'  'cdp'  'ctau0'  'ctauinf'  'cdnhi'  'cdphi'  'ctau0hi'  'ctauinfhi'
interface,
        system
          analysis_driver = './xyce.csh'
responses,
        num_least_squares_terms = 50
        analytic_gradients
        no_hessians
```

Essential components of xyce.csh

- **Preprocessing**

```
./dprepro $argv[1] bft92_tmplt.net bft92_new.net
```

- **Execution of simulation**

```
./xyce bft92_new.net
```

- **Post-processing**

```
./compute_residuals
```

EXTRA SLIDES

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**

nl2sol method dependent options

```
{nl2sol}                                    \
[function_precision = <REAL>]               \
[absolute_conv_tol = <REAL>]                \
[x_conv_tol = <REAL>]                       \
[singular_conv_tol = <REAL>]                \
[singular_radius = <REAL>]                  \
[false_conv_tol = <REAL>]                   \
[initial_trust_radius = <REAL>]             \
[covariance = <INTEGER>]                    \
[regression_diagnostics]                    \
```

Reference Manual Chapter 2

Circuit Simulator

- Written at Sandia to support electrical (circuit) design simulation
- Started with Berkeley SPICE 3f5
  - Mostly Algebraic/Differential Equations with Behavioral Model Options
  - Physics based models (instead of empirical based models)
  - Improvement over industry standard PSPICE capability
- Massively parallel code that allows simulation of large-scale complex system circuit model
  - Investigate circuit interactions
  - Simulate large digital components

Hutchinson, Keiter, Hoekstra, Rankin, Waters, Russo, Wix, Ballard, …