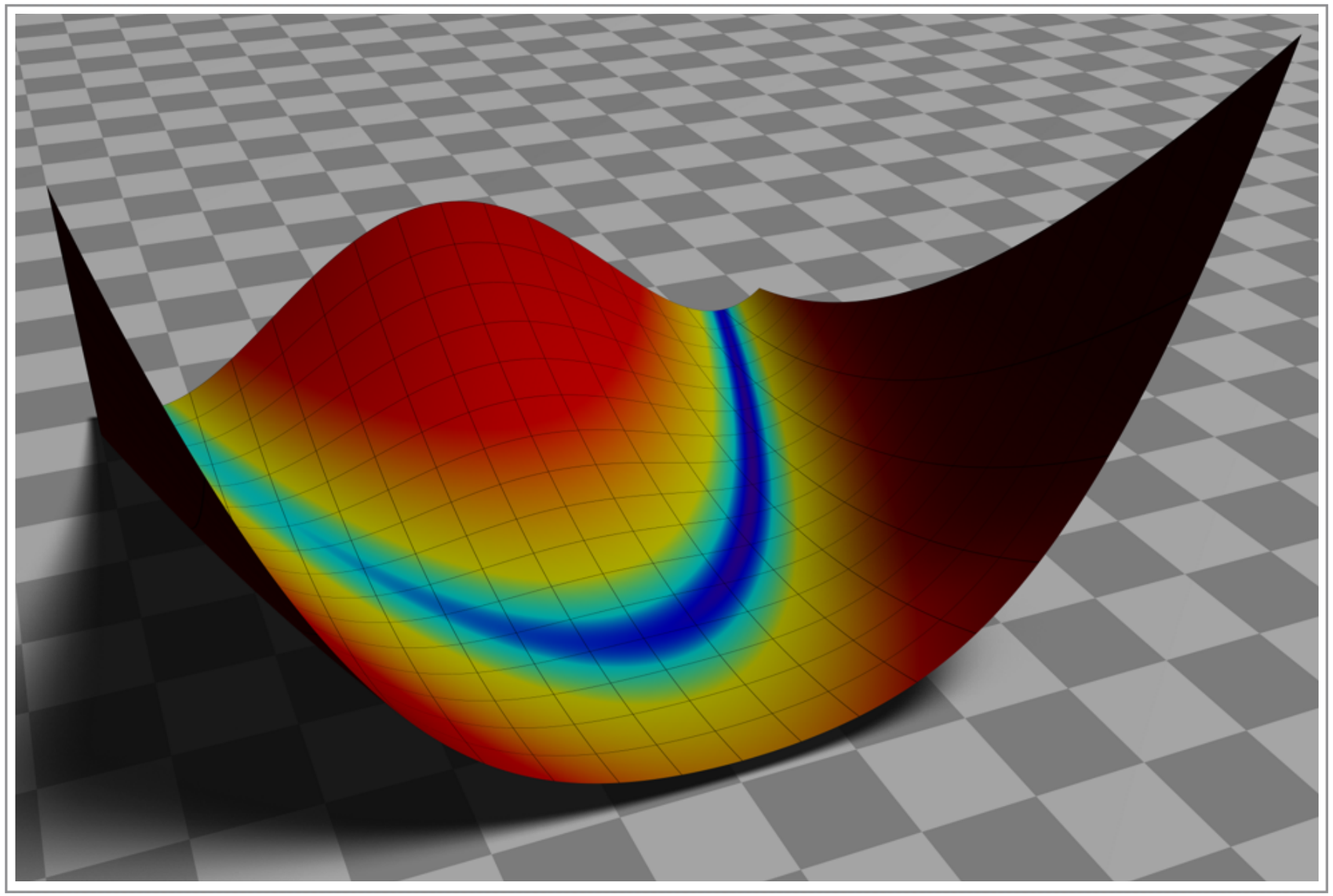
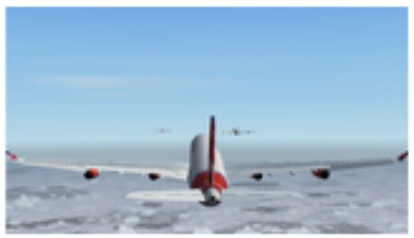


Nonlinear Optimization for Engineers: An Overview of the Fundamentals

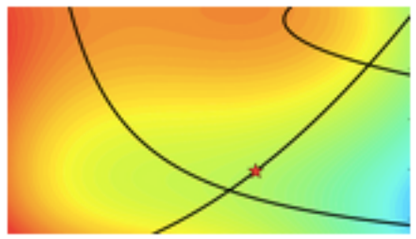
S. Andrew Ning
National Renewable Energy Laboratory
September 2013

About Me

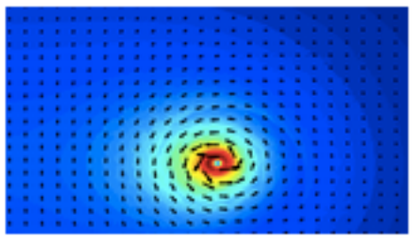
Wind Turbines



Aircraft



Optimization



Aerodynamics

NREL: Wind Turbine Design, Systems Engineering, Aero/Structural Modeling

Stanford: Aircraft Design, Formation Flight, Multifidelity Optimization

Introduction

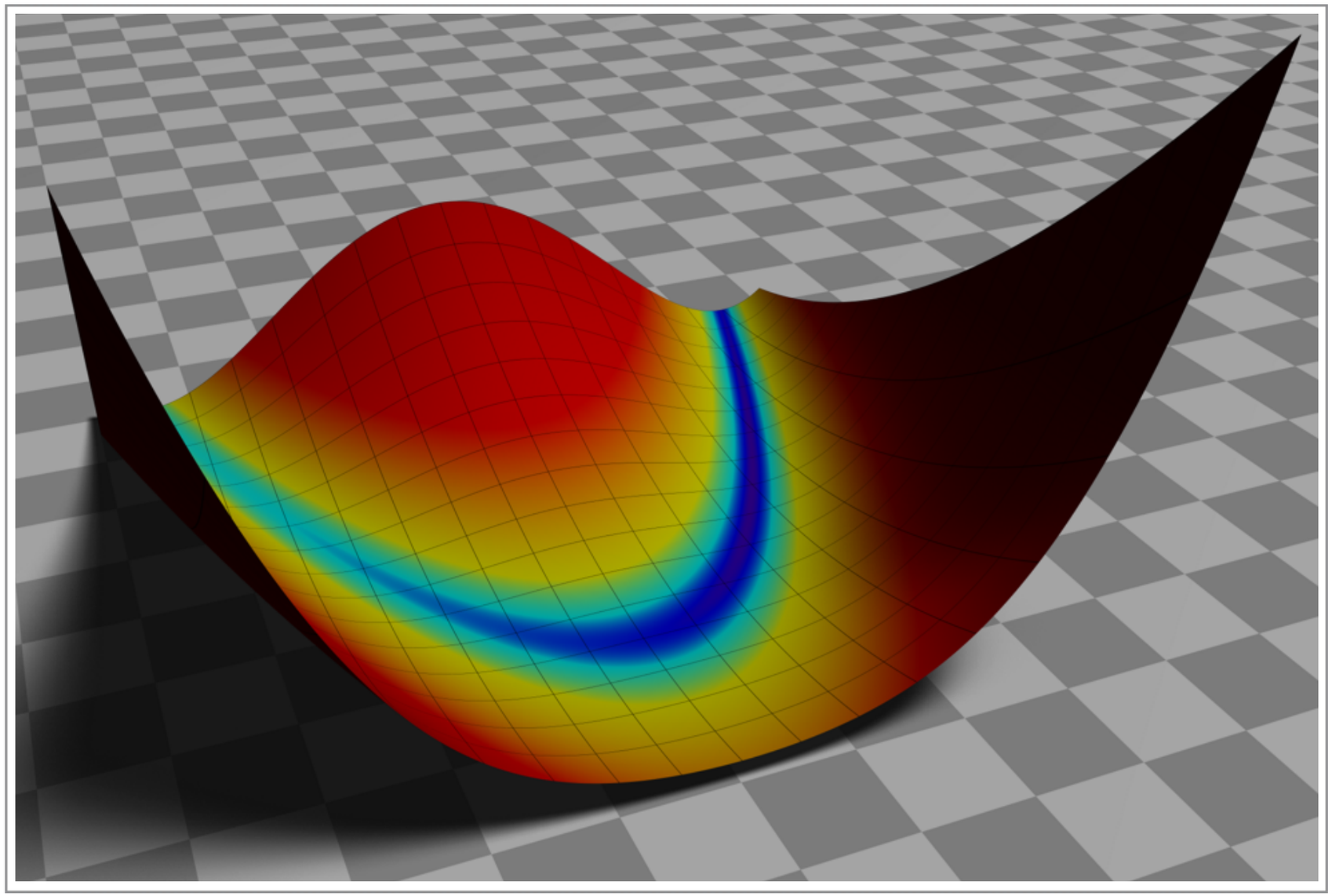
Gradient-based Algorithms

Sensitivity Analysis

Gradient-free Algorithms

Problem Formulation Strategies

Truss Optimization Example

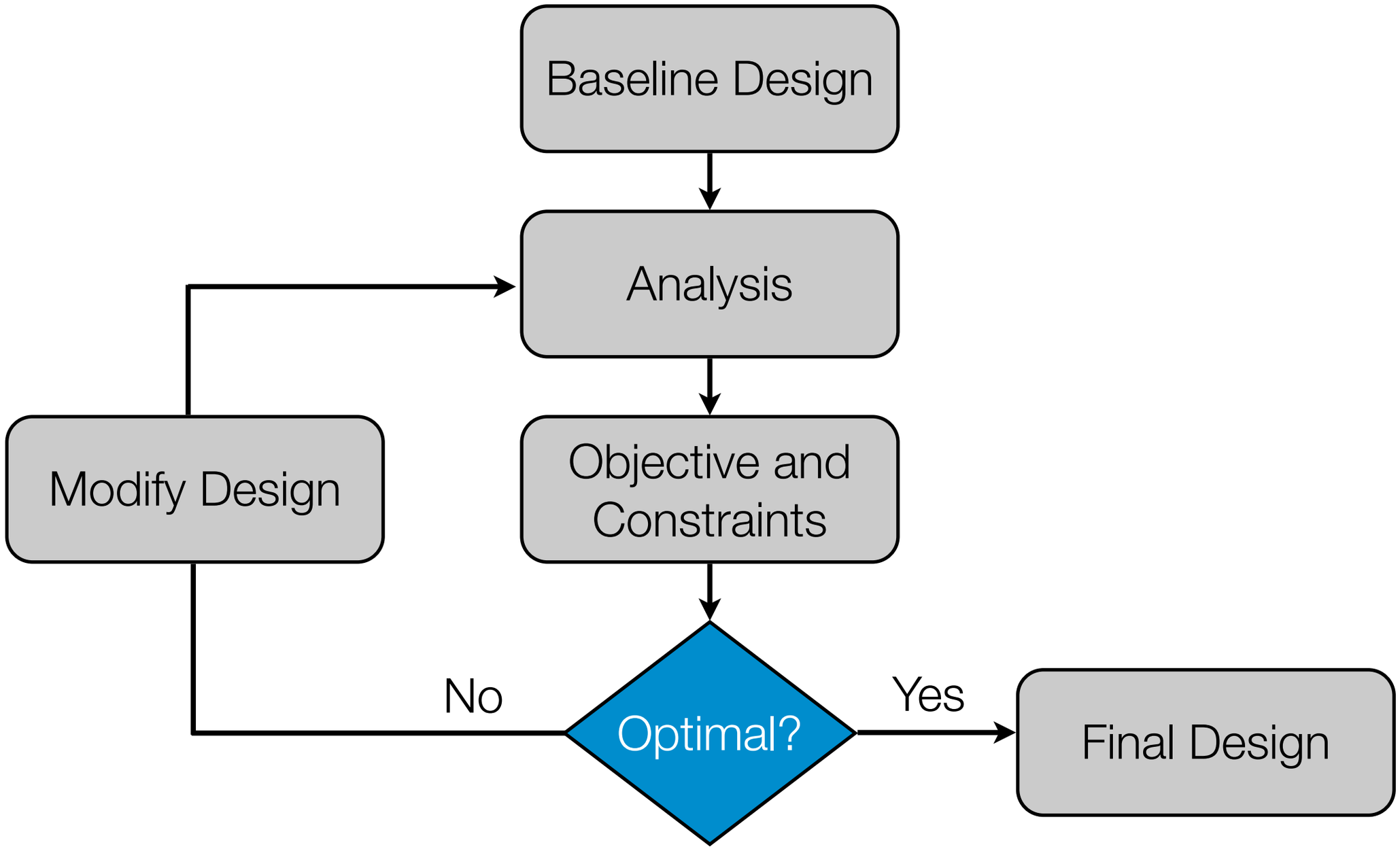


Introduction

Why Optimization?

- Design: What turbine design and/or placement minimizes the wind plant's cost of energy?
- Data Fitting: What model parameters minimize the predicted error?
- Decision Making: What control inputs will minimize deviation from the desired trajectory?

Optimization Process



Objective Function

- allows comparison between designs, this is the function we want to minimize
- selecting a good objective function is critical, and not always easy
- the role of multiobjective optimization is debatable

Design Variables

- set of variables that are allowed to vary in order to optimize the design
- design variables should be independent
- design variables can be continuous or discrete (often called integer variables)

Gradient and Hessian

$$\nabla f(x) \equiv g(x) \equiv \begin{bmatrix} \frac{\partial f}{\partial x_1} \\ \frac{\partial f}{\partial x_2} \\ \vdots \\ \frac{\partial f}{\partial x_n} \end{bmatrix}$$

local minimum:

$$||g(x^*)|| = 0$$

$$H(x^*) \text{ is positive definite}$$

$$\nabla^2 f(x) \equiv H(x) \equiv \begin{bmatrix} \frac{\partial^2 f}{\partial x_1^2} & \cdots & \frac{\partial^2 f}{\partial x_1 \partial x_n} \\ \vdots & & \vdots \\ \frac{\partial^2 f}{\partial x_n x_1} & \cdots & \frac{\partial^2 f}{\partial x_n^2} \end{bmatrix}$$

Constraints

- bound $x \leq 0$
- linear $Ax \leq 0$
- nonlinear $c(x) \leq 0$

- equality constraints are also sometimes used
- constraints are either active or inactive

Optimization Problem

$$\begin{aligned} \text{minimize} \quad & f(x) \\ \text{with respect to} \quad & x_i \in \mathcal{R}^n \\ \text{subject to} \quad & c_j(x) < 0, \quad j = 1 \ldots m \end{aligned}$$

Optimization Problem

$$\begin{aligned} \text{min.} \quad & f(x) \\ \text{w.r.t.} \quad & x \\ \text{s.t.} \quad & c(x) < 0 \end{aligned}$$

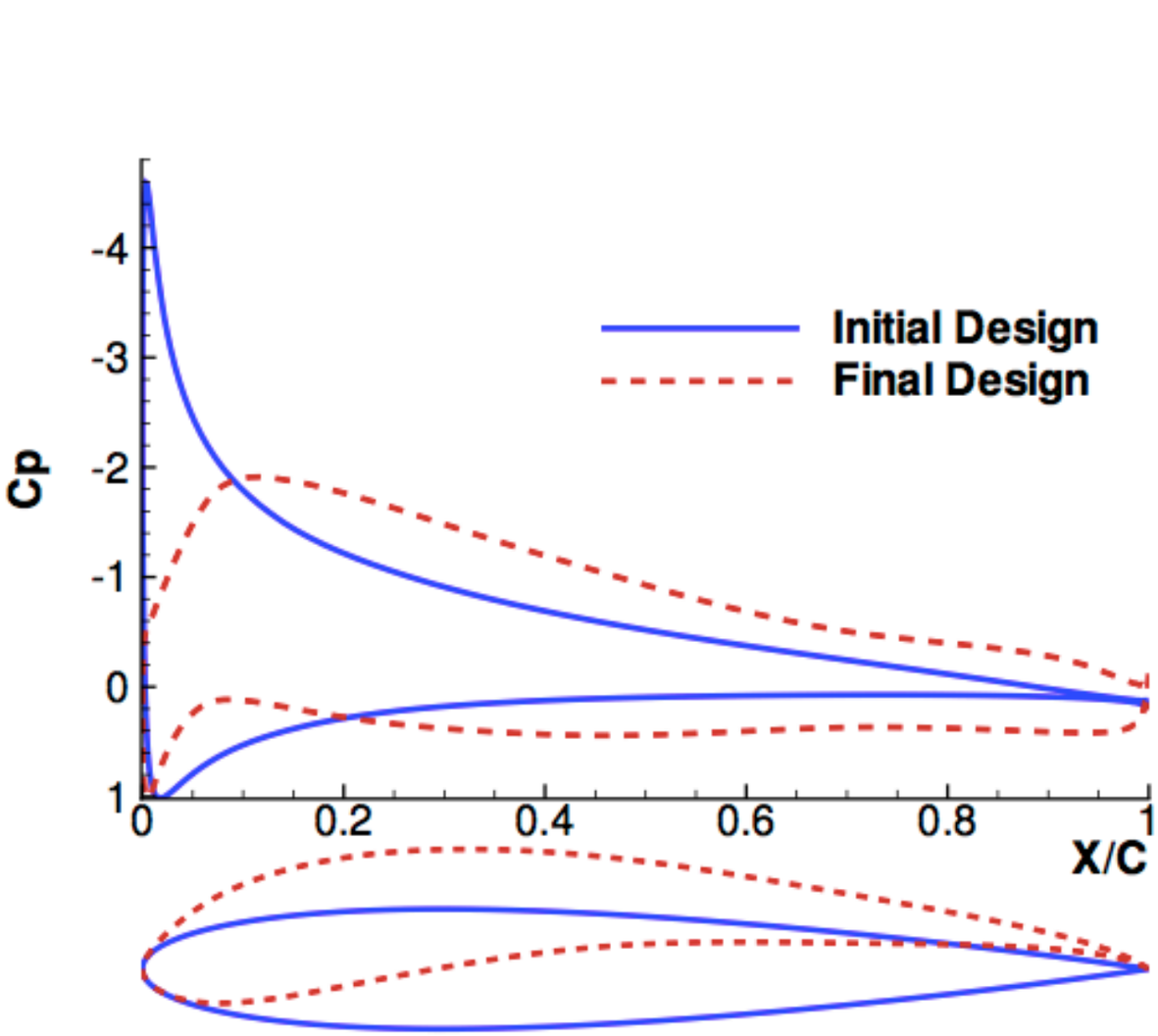
Classification

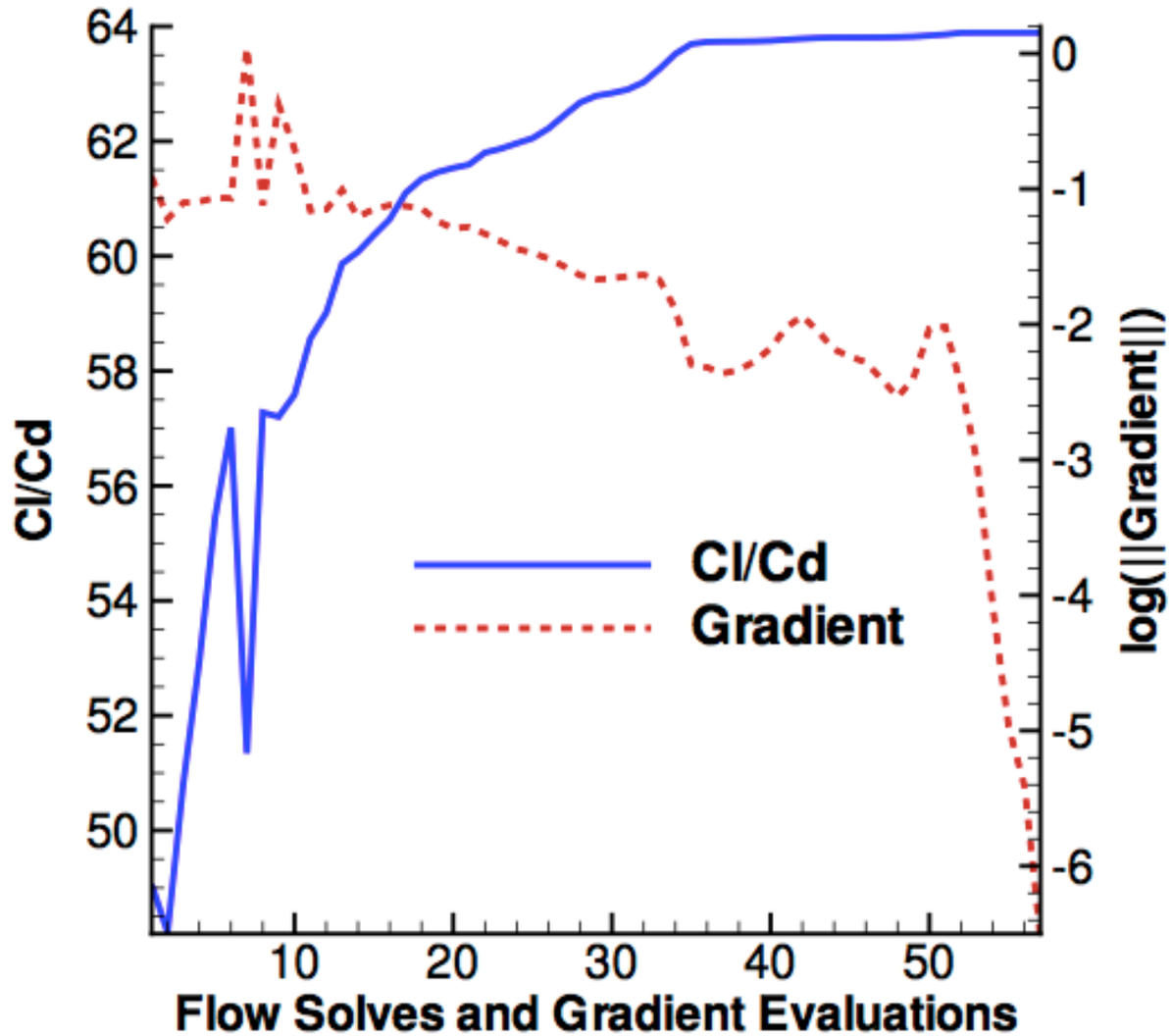
- Continuous or discontinuous?
- Continuous design variables or discrete?
- Convex or non-convex?
- Linear or nonlinear?
- Constrained or unconstrained?

Algorithm Selection

- Convex
- Gradient-based
- Gradient-free
- Integer or Mixed-integer

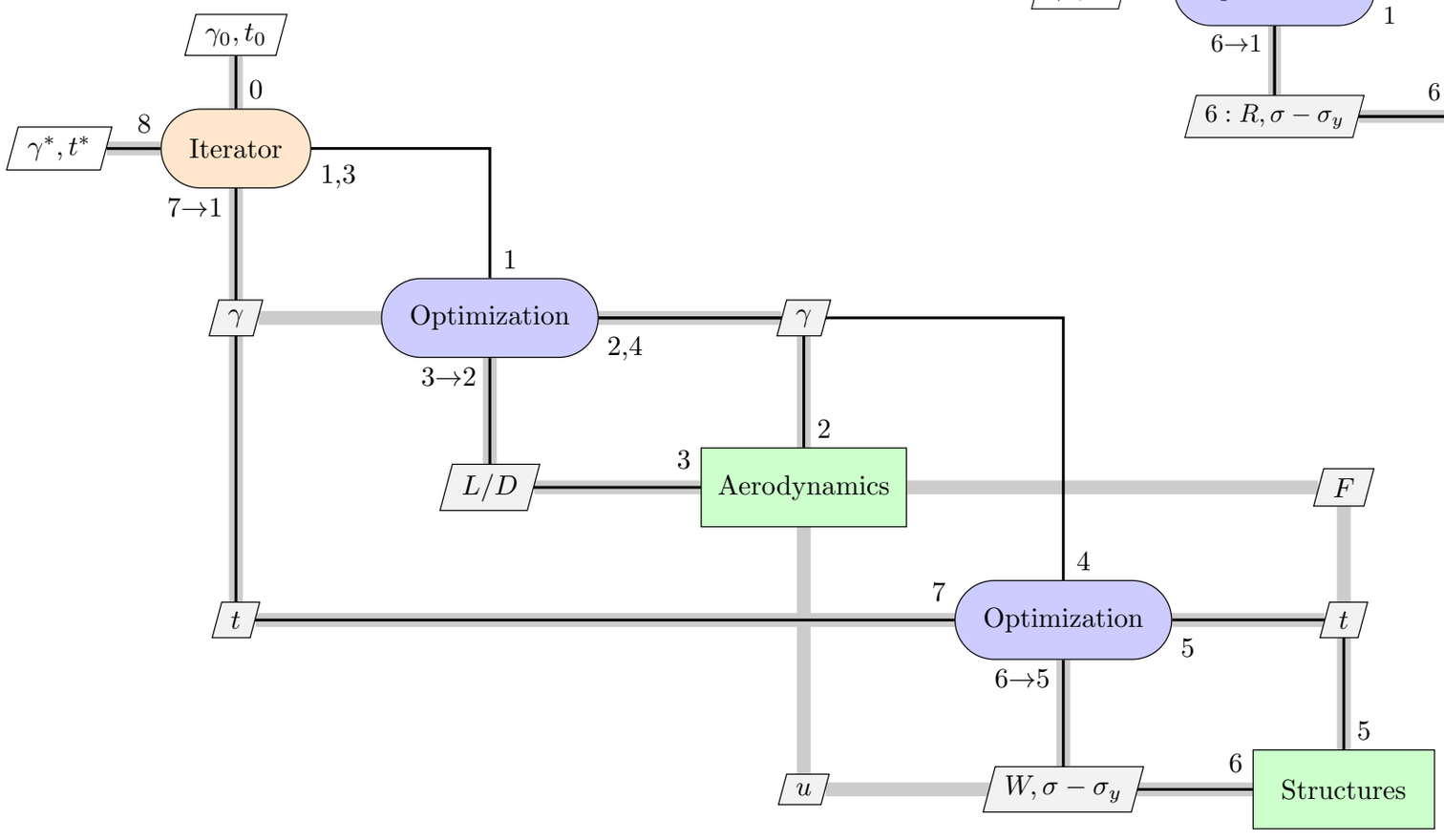
Iteration History



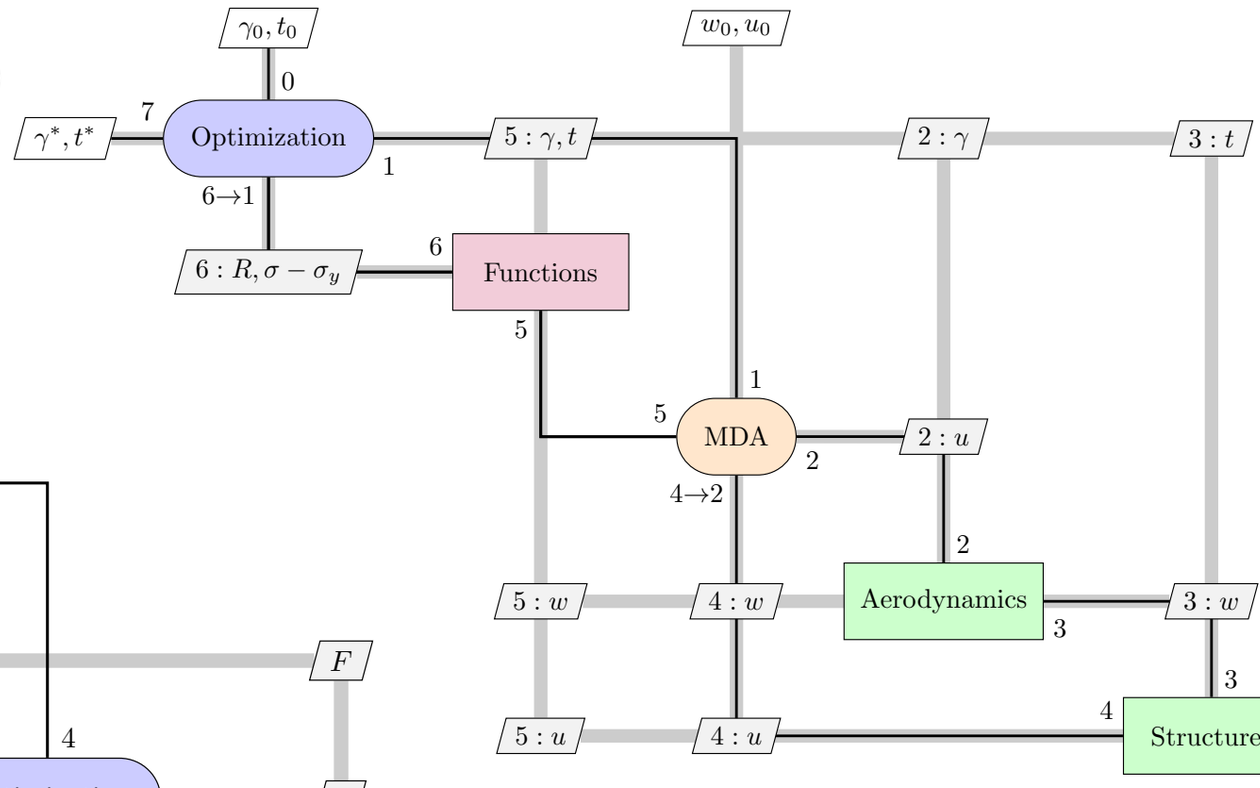


Sequential vs MDO

Sequential



MDO



Simple Example

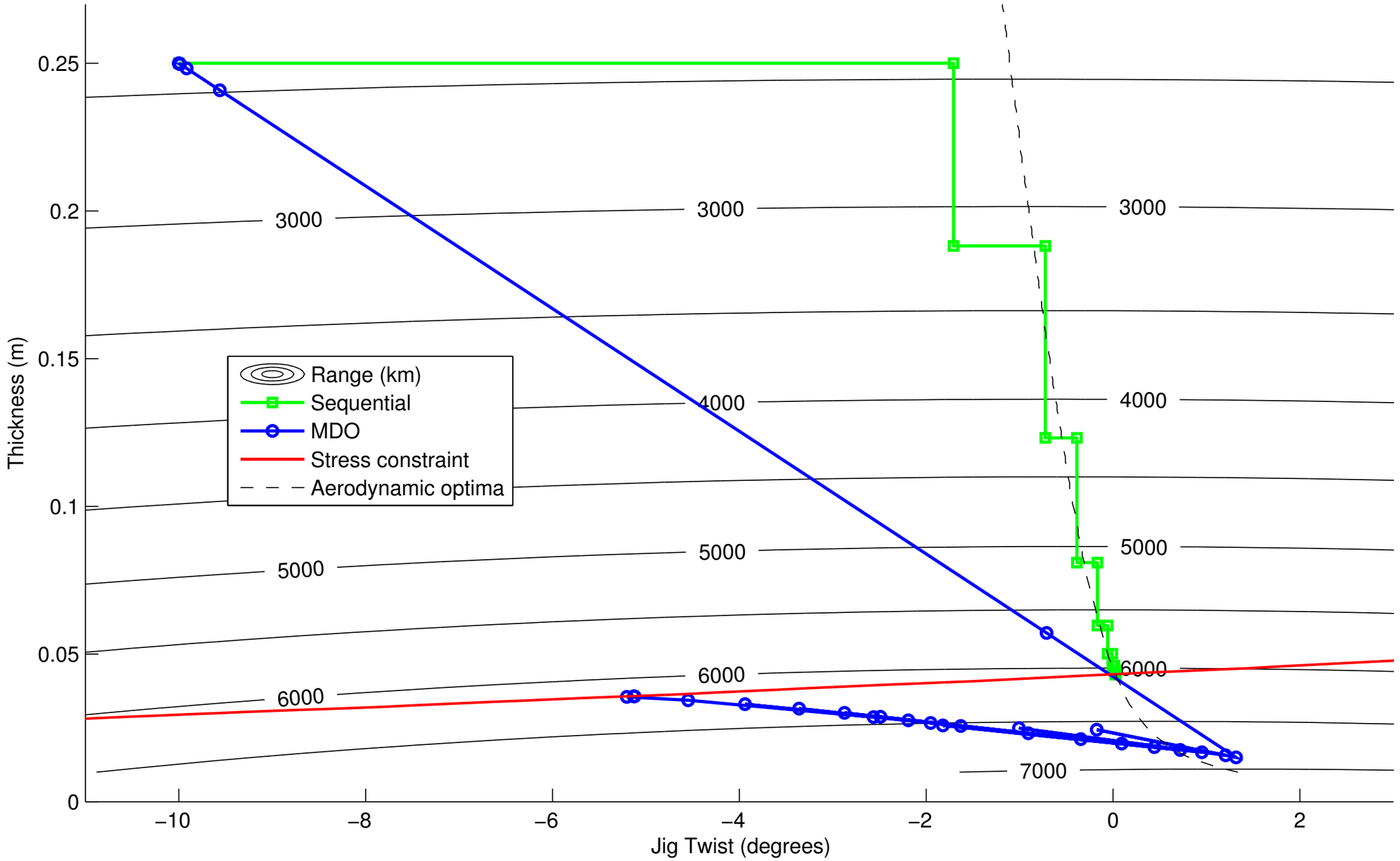
- Aerodynamics: panel code computes induced drag. Design variables: wing twist

- Structures: beam finite element model of spar that computes displacements and stresses. Design variables: spar thickness.

- Objective: maximize range.

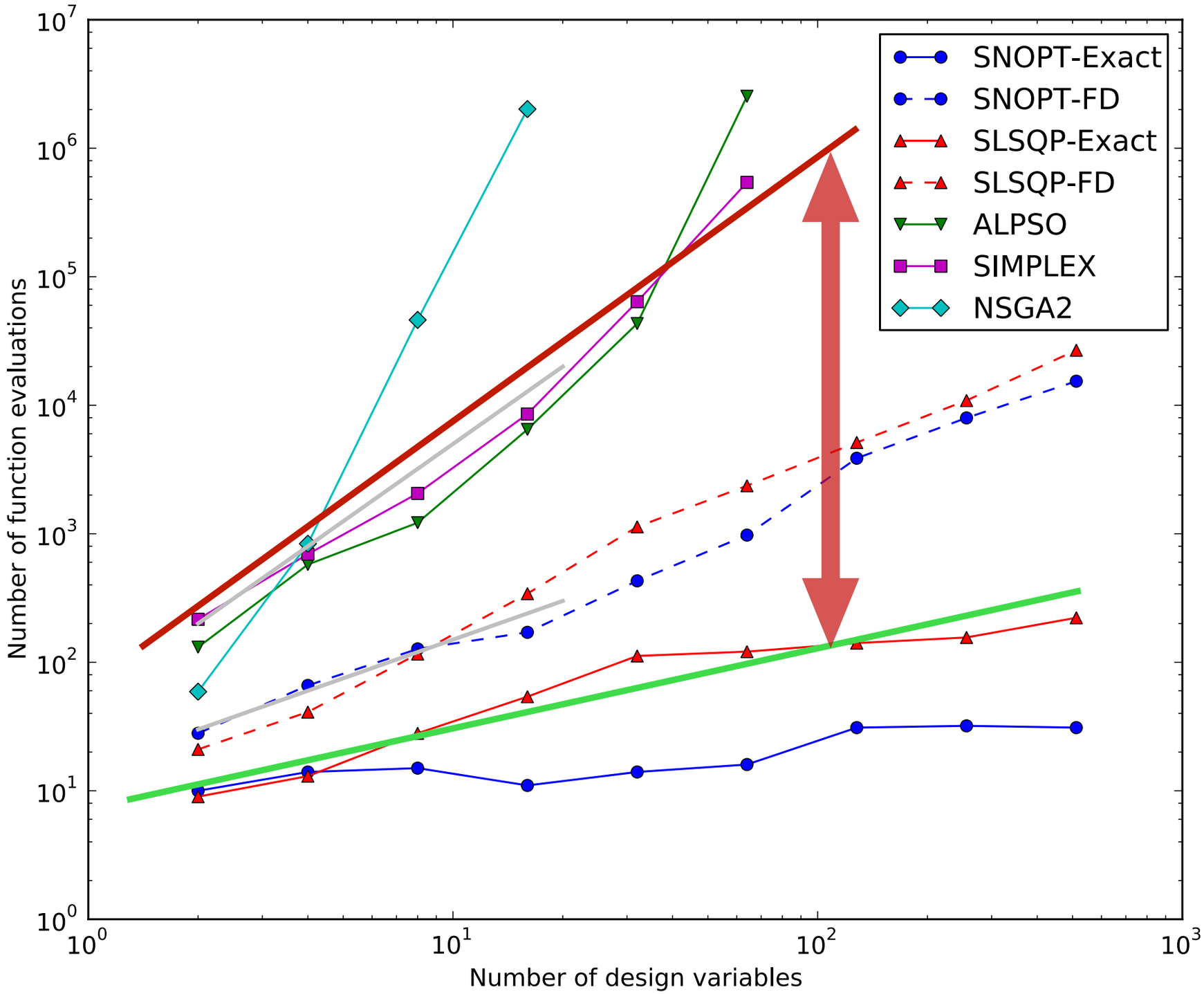
$$\text{Range} = \frac{V}{c}\frac{L}{D}\ln\frac{W_i}{W_f}$$

Sequential vs MDO



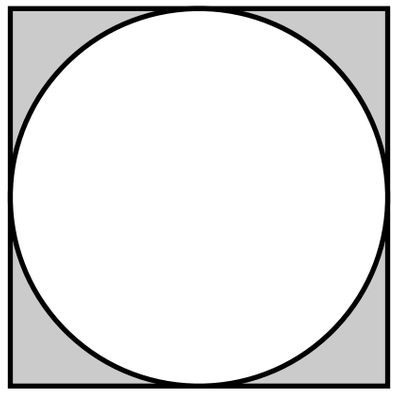
Chittick and Martins, Structural and Multidisciplinary Optimization, 2008

Gradient vs Gradient-Free



Intuition in Higher Dimensions

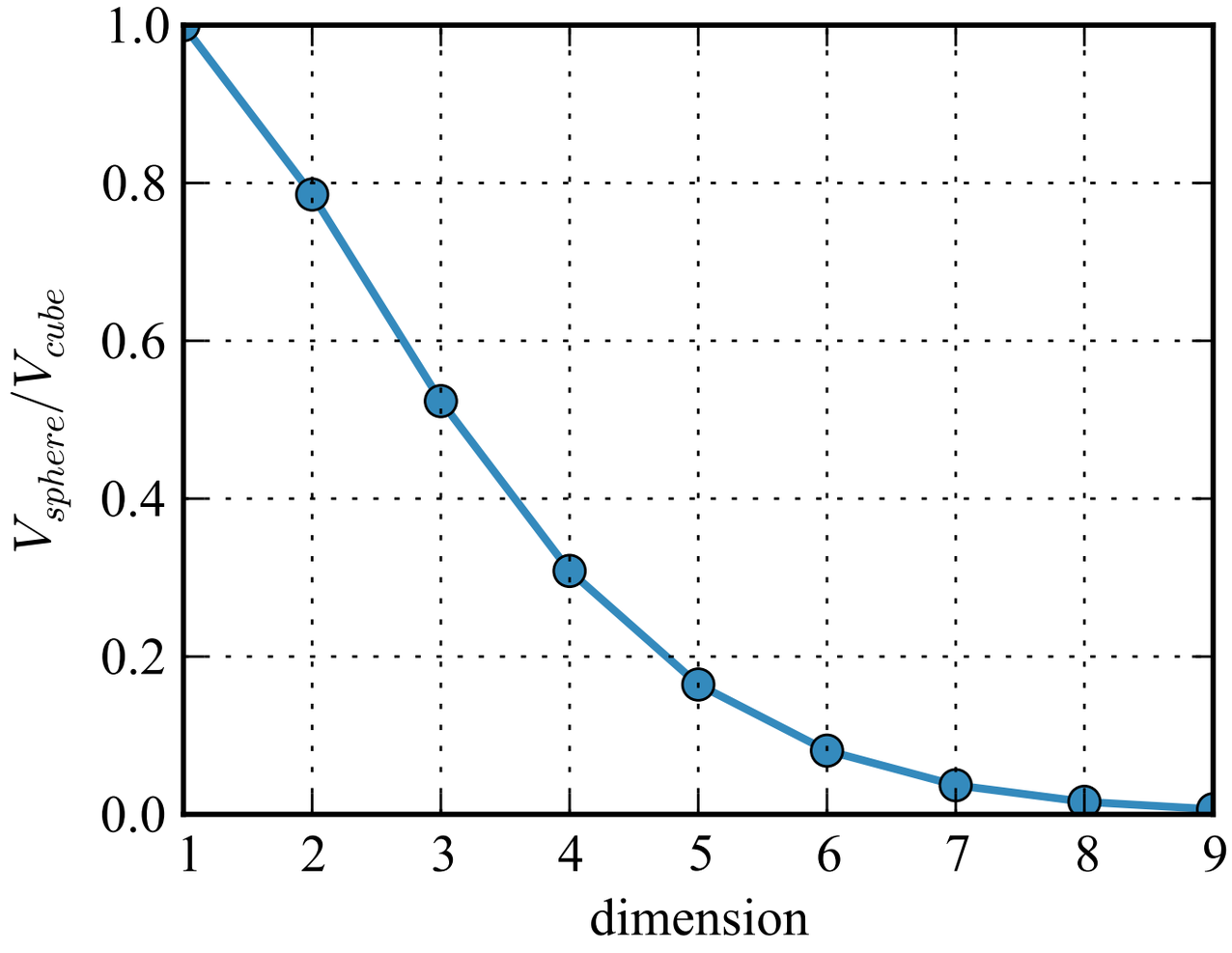
hypersphere circumscribed in hypercube

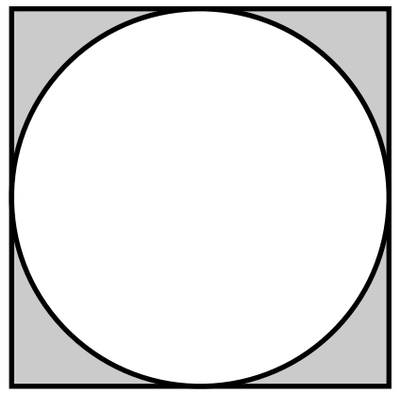


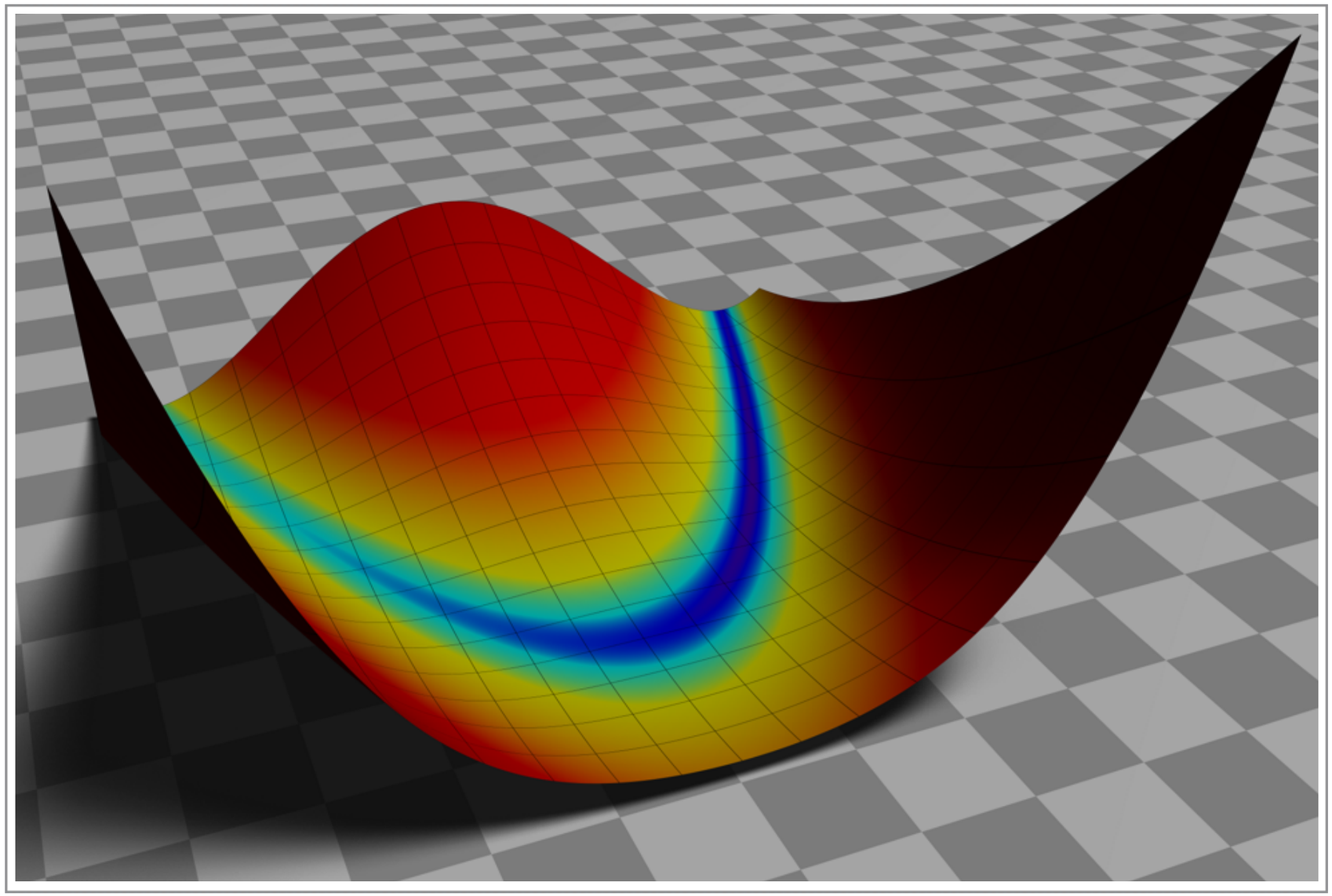
volume of sphere ? volume of cube

Intuition in Higher Dimensions

hypersphere circumscribed in hypercube







Gradient-Based Methods

Convex Optimization

- convex problems are the best type of optimization problems to have. very efficient algorithms, extremely robust, and have guarantees of global optimality

- not common in mechanical engineering problems, but can be useful in idealized versions of the problem or as a subproblem. more common in electrical engineering and machine learning.

- often difficult to recognize---usually requires reformulation of the problem

- methods: Linear Programming, Quadratic Programming, Second Order Cone Programming, etc.

Lift Distribution Optimization

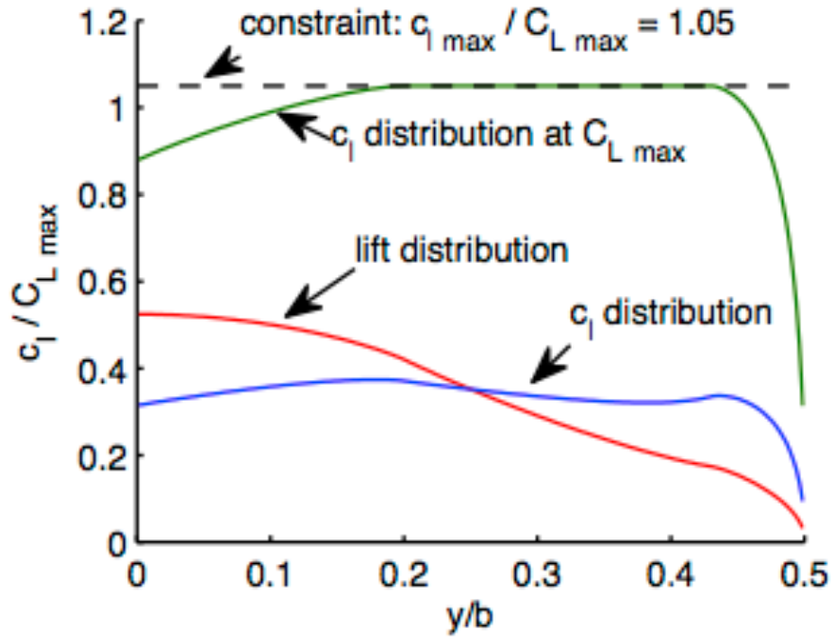
$$\min_{\gamma} \quad \gamma^T [D2] \gamma + D1^T \gamma$$

$$\text{w.r.t.} \quad \gamma$$

$$\text{s.t} \quad LIC^T \gamma \geq L_{ref}$$

$$WIC^T \gamma \leq W_{ref}$$

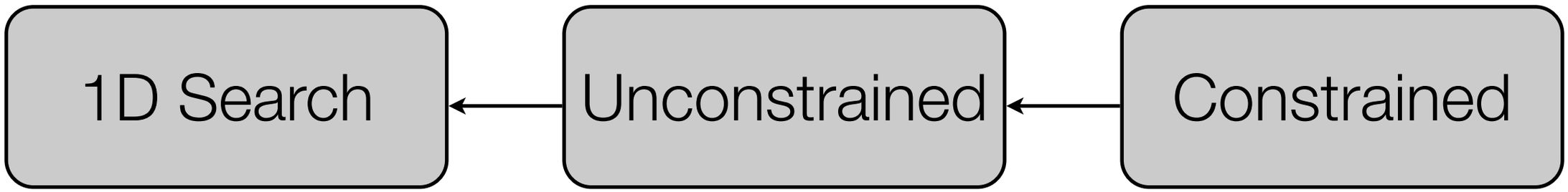
$$CL1^T \gamma + CL0 \leq c_{l\,max}$$



Gradient-Based

- Large body of theory and methods based on gradient information.
- efficient *if* you have reliable gradients
- can even be used for multi-modal problems, or in hybrid methods
- methods: Sequential Quadratic Programming, Interior Point Methods, Trust Region Methods.

Use What We Know



1D Search

- Bisection
- Newton’s method
- Secant method
- Golden Section
- Polynomial Interpolation

Descent Methods

choose starting point: x_0

repeat until convergence criteria met

compute search direction: Δx

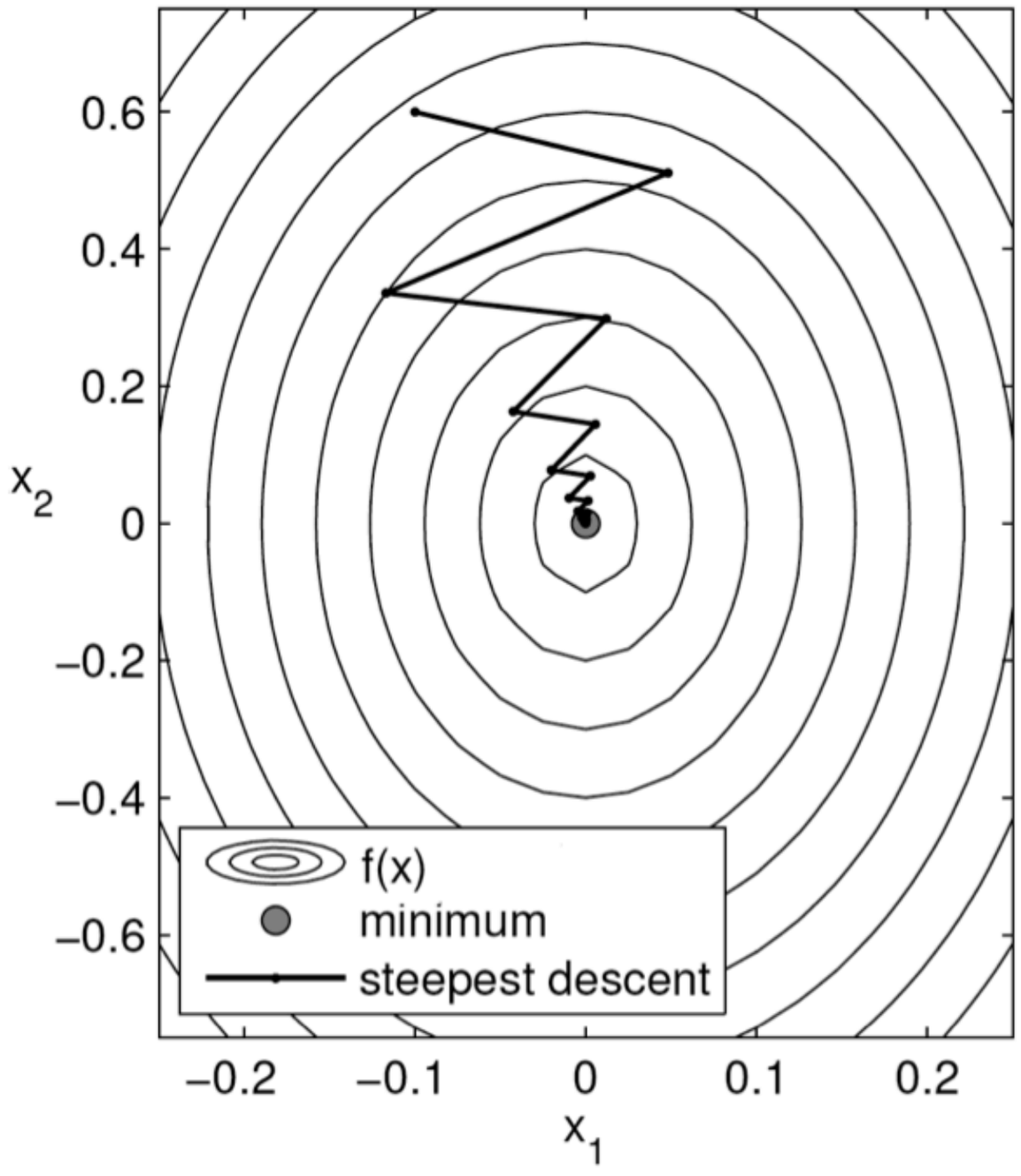
pick step size α such that: $f(x_i + \alpha \Delta x) < f(x_i)$

update design variables: $x_{i+1} = x_i + \alpha \Delta x$

Gradient Descent

$$\Delta x = -\nabla f(x_i)$$

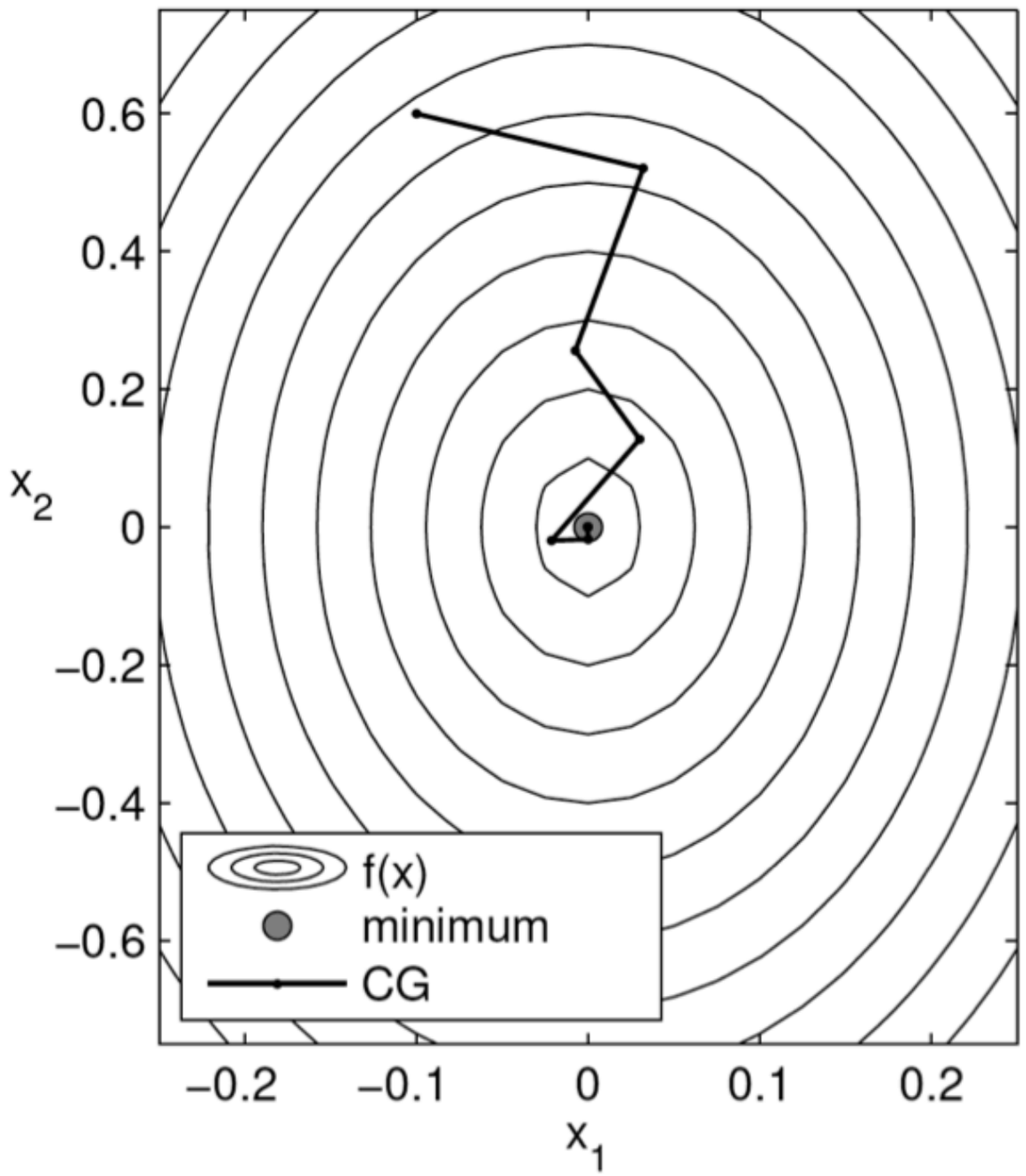
simple and intuitive, but inefficient (depending on condition number of Hessian)



Conjugate Gradient

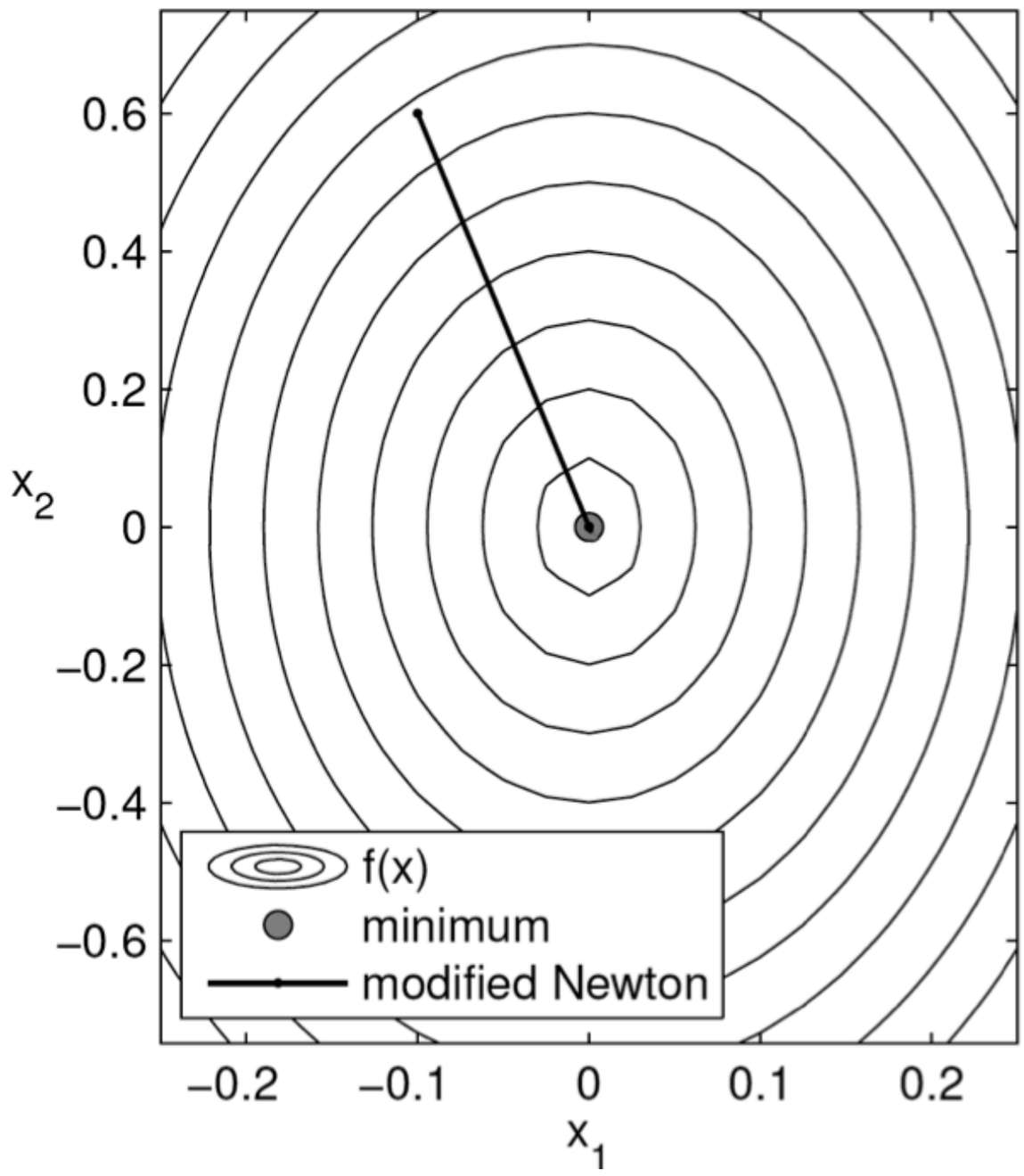
- Similar to steepest descent but uses information from last iteration

- Conjugate directions lead to minimum in at most n iterations (for a quadratic)



Modified Newton

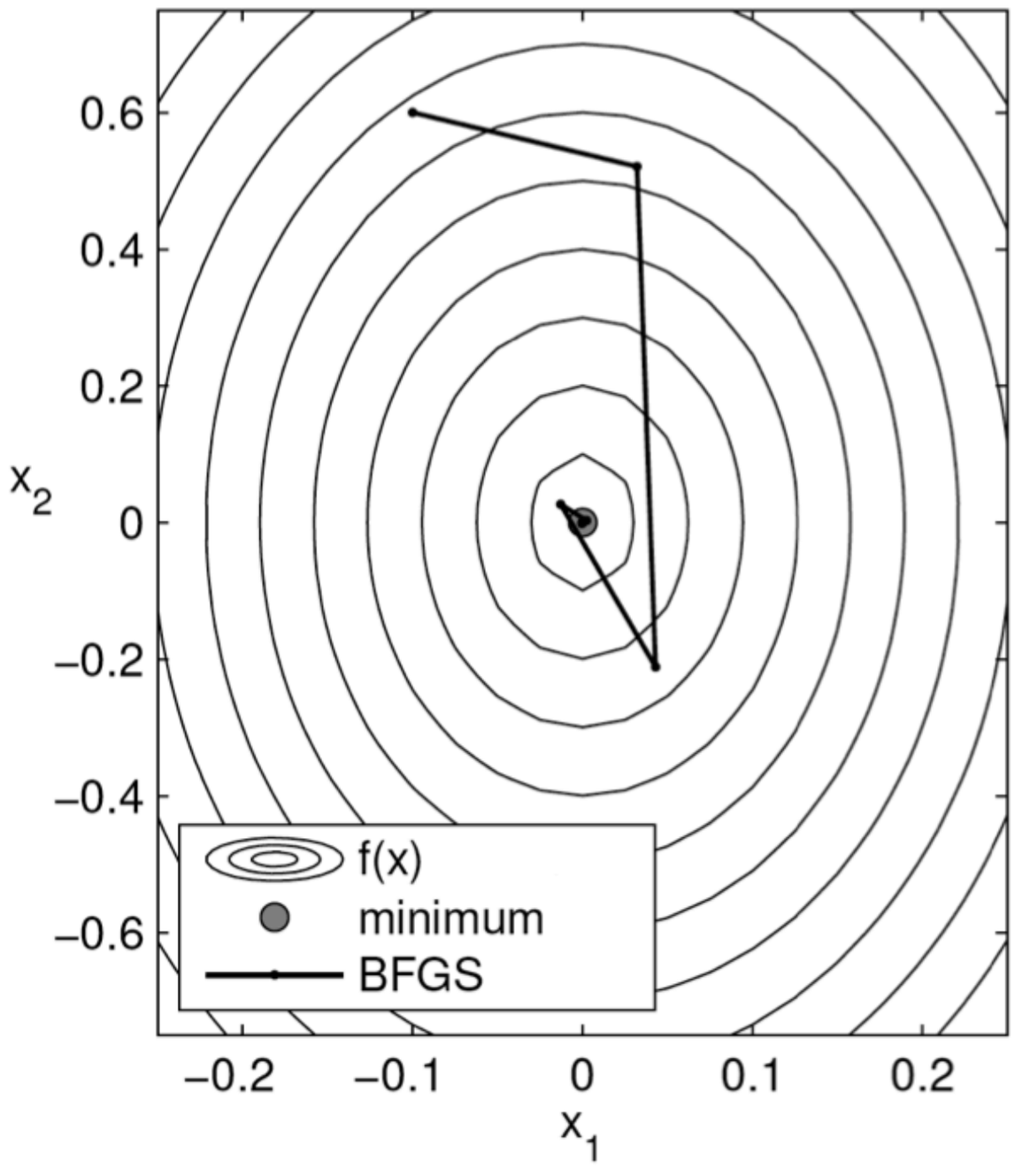
- Requires Hessian
- affine invariant
- Uses a line search (not a pure Newton method)
- Care must be taken to ensure Hessian is positive definite and not ill-conditioned. Has all the typical problems of Newton's method



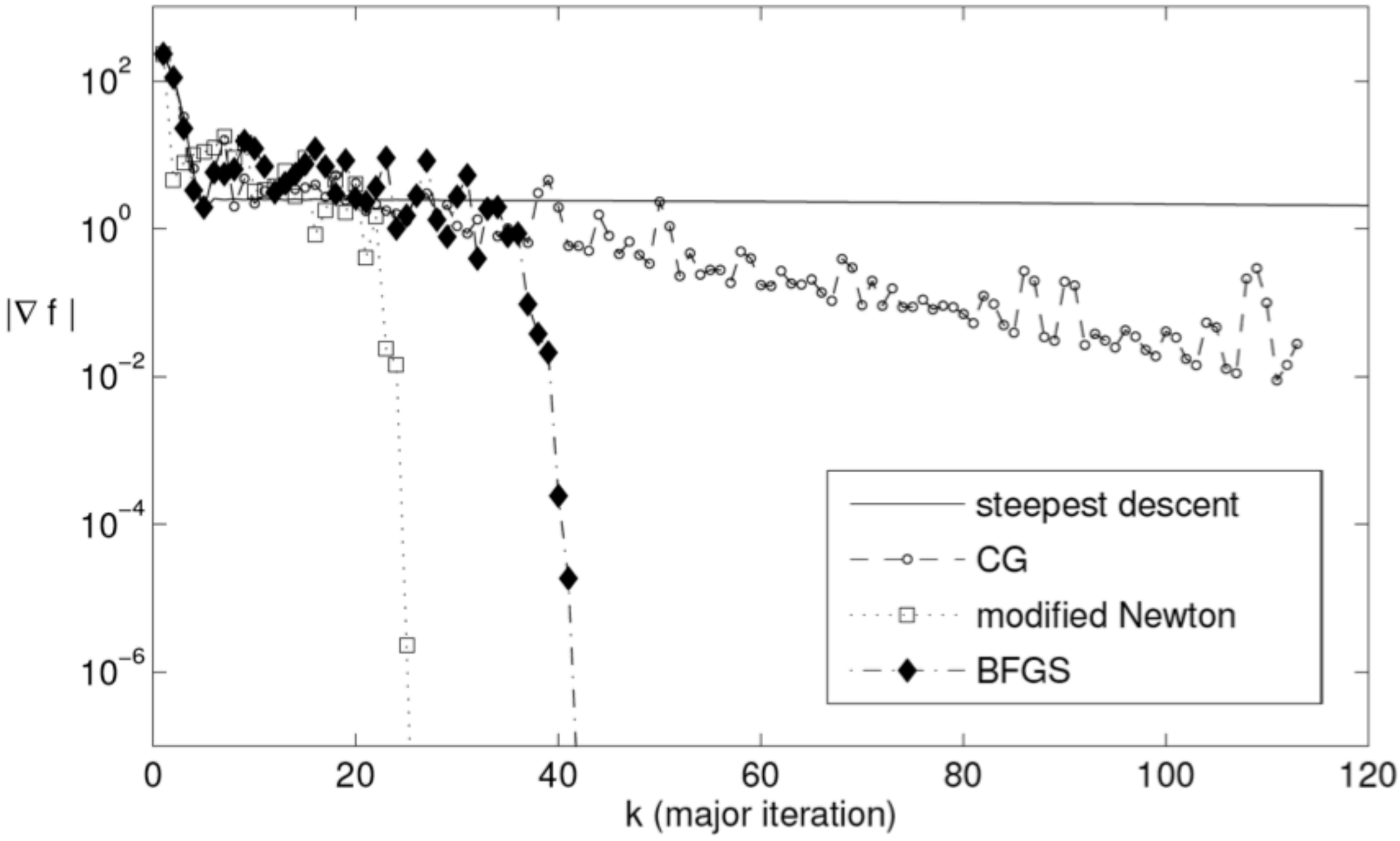
Quasi-Newton Methods

- Only needs first-order information but builds up an approximation to the Hessian based on previous iterations

- BFGS is one popular quasi-Newton update approach



Convergence: Rosenbrock



Line Search

$$x_{i+1} = x_i + \alpha \Delta x$$

- goal is to pick a step length with a substantial decrease without using many iterations

- at a basic level the line search is typically a backtracking line search

- must meet sufficient decrease and curvature conditions, often uses bracketing, and may abandon the search and use the iterates to rebuild the local quadratic model

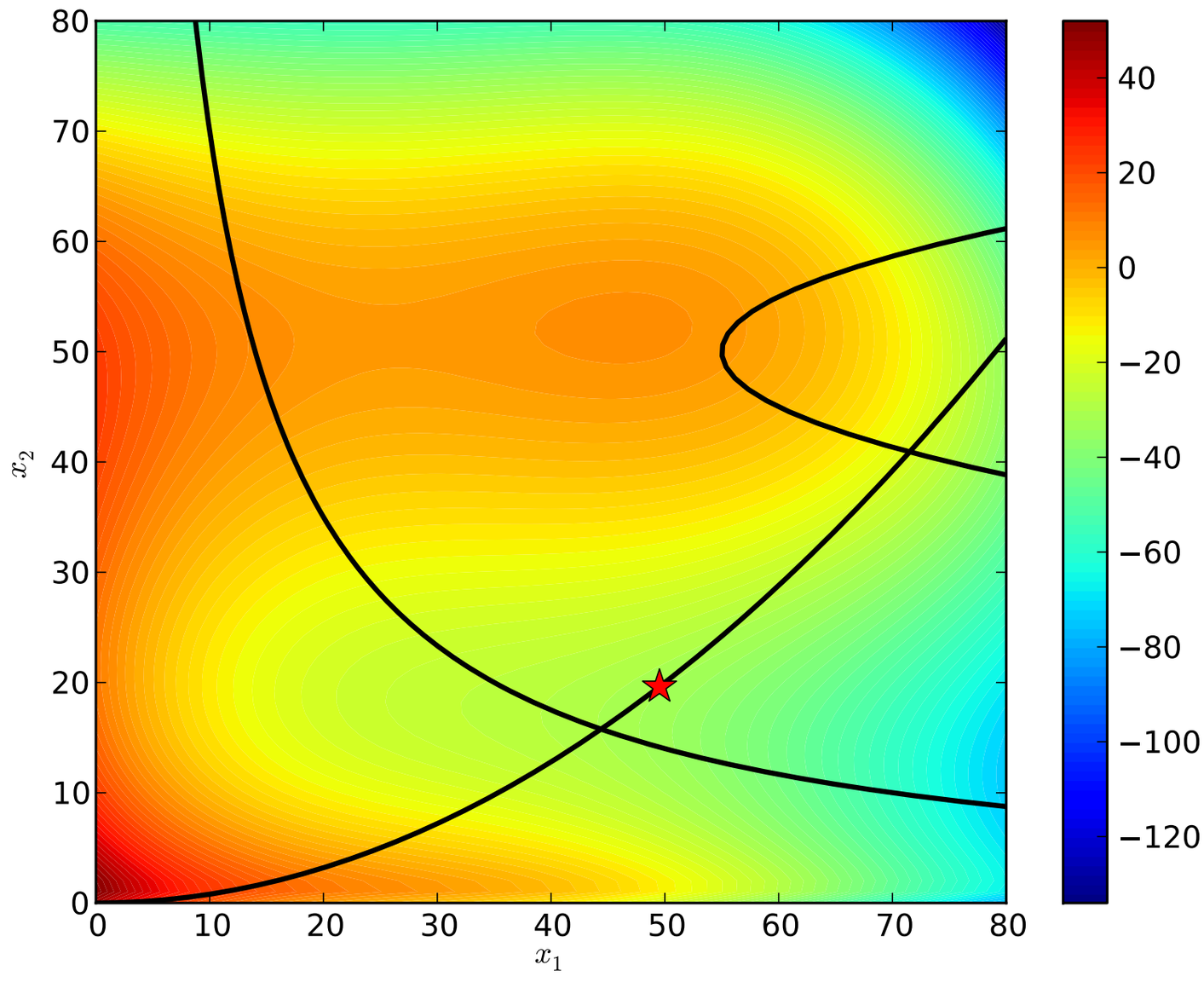
Trust-Region Methods

- An alternative to line search methods
- Creates a region in which we “trust” a surrogate model (typically a quadratic)
- The region size grows or shrinks using heuristics based on how well the decrease matches predictions

Constrained Optimization

$$\begin{aligned} \text{min.} \quad & f(x) \\ \text{w.r.t.} \quad & x \\ \text{s.t.} \quad & c(x) < 0 \end{aligned}$$

Simple 2D Example



Lagrangian

$$\mathcal{L}(x, \lambda, s) = f(x) + \lambda^T (c(x) + s^2)$$

$\mathcal{L}$: Lagrangian

λ : Lagrange multipliers

s : slack variables

Karush–Kuhn–Tucker (KKT)

$$\mathcal{L}(x, \lambda, s) = f(x) + \lambda^T (c(x) + s^2)$$

$$\frac{\partial \mathcal{L}}{\partial x_i} = \frac{\partial f}{\partial x_i} + \sum_j \lambda_j \frac{\partial c_j}{\partial x_i} = 0, \quad i = 1 \dots n$$

$$\frac{\partial \mathcal{L}}{\partial \lambda_j} = c_j + s_j^2 = 0, \quad j = 1 \dots m$$

$$\frac{\partial \mathcal{L}}{\partial s_j} = \lambda_j s_j = 0, \quad j = 1 \dots m$$

$$\lambda_j > 0, \quad j = 1 \dots m$$

Sequential Quadratic Programming (SQP)

- quadratic subproblem at each point (quadratic objective, equality constraints), for inequality constraints a common approach is to only consider the active-set

- can be considered application of Newton's method to KKT conditions

- requires estimate of Hessian of the Lagrangian (typically quasi-Newton approximations)

- uses merit functions in line search step (penalty, estimate of Lagrangian, filter methods)

Penalty Functions

$$\hat{f}(x, \rho) = f(x) + \rho \sum_i (\max[0, c_i(x)])^2$$

exterior quadratic penalty

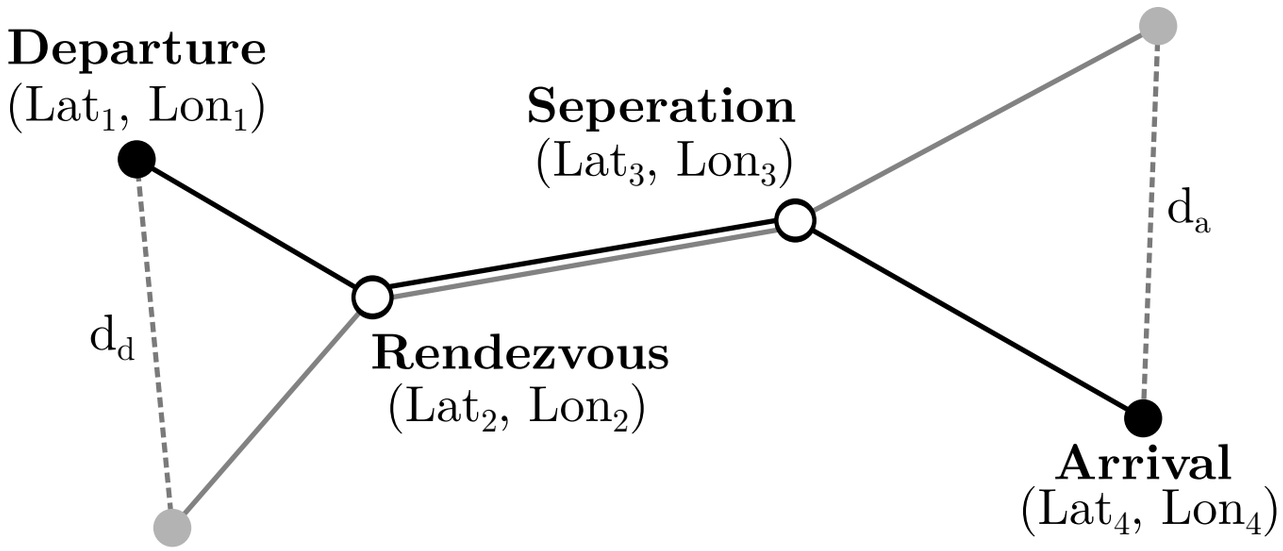
$$x^* = \lim_{\rho \to \infty} x^*(\rho)$$

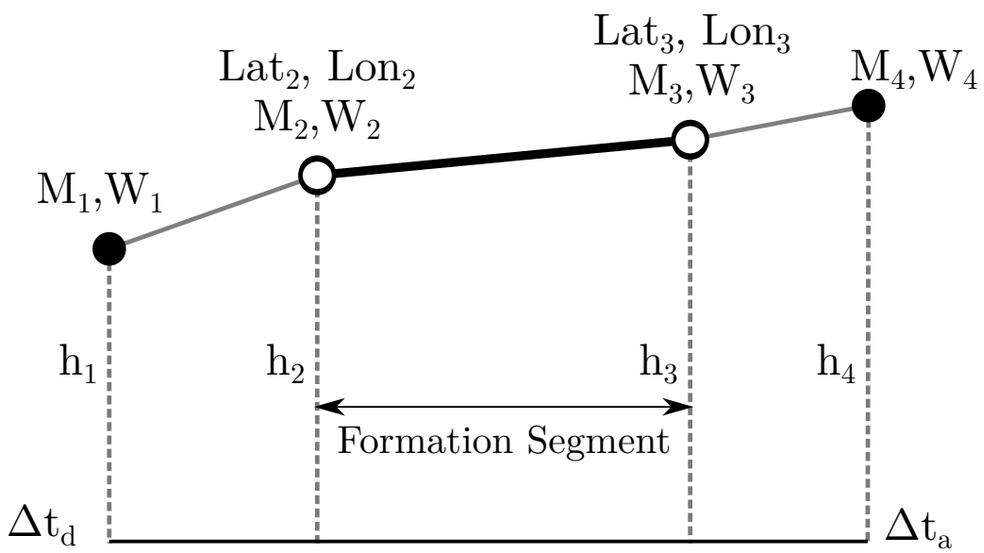
$$\hat{f}(x, \rho) = f(x) - \rho \sum_i \log(-c_i(x))$$

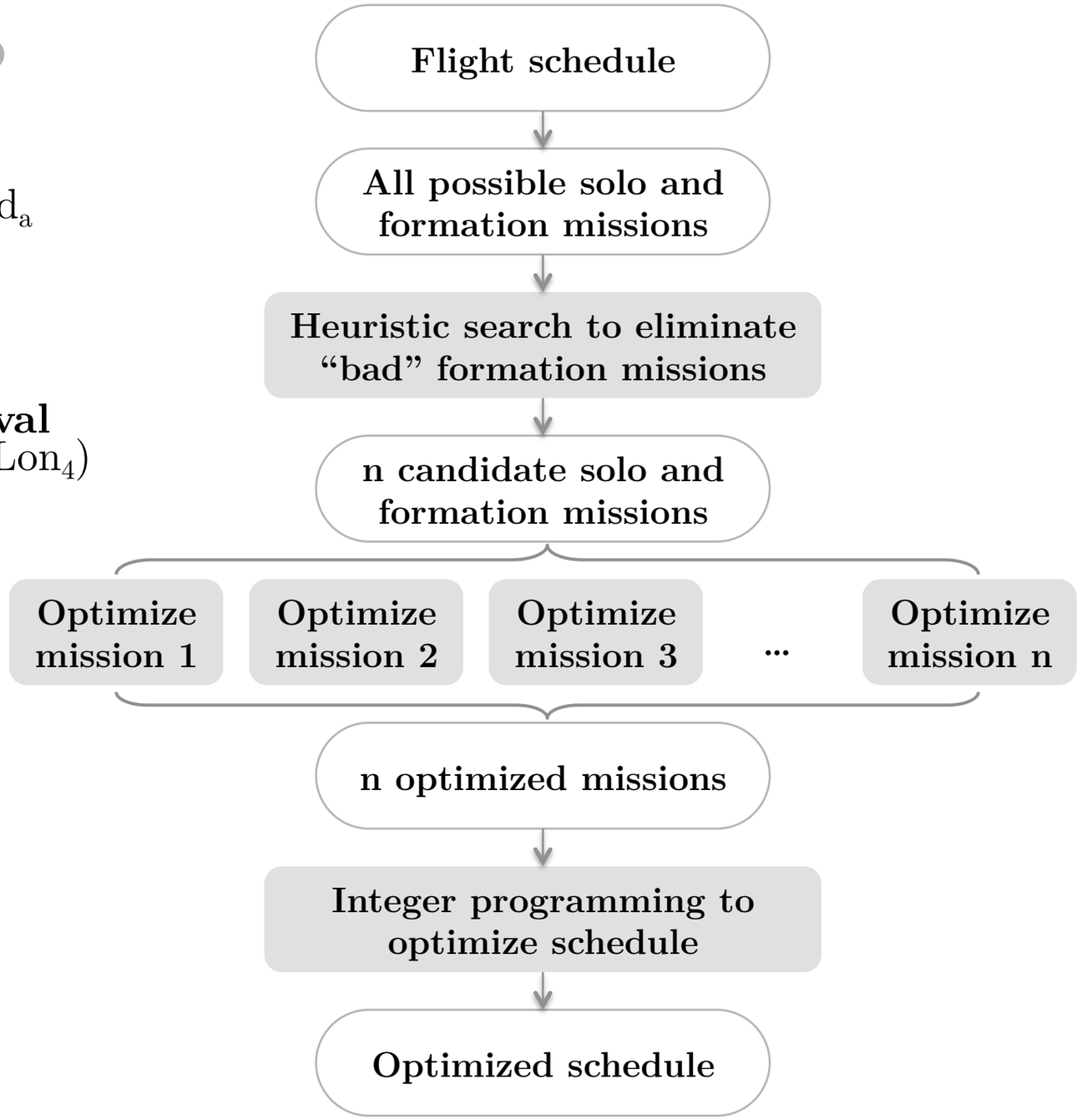
interior log barrier

$$x^* = \lim_{\rho \to 0} x^*(\rho)$$

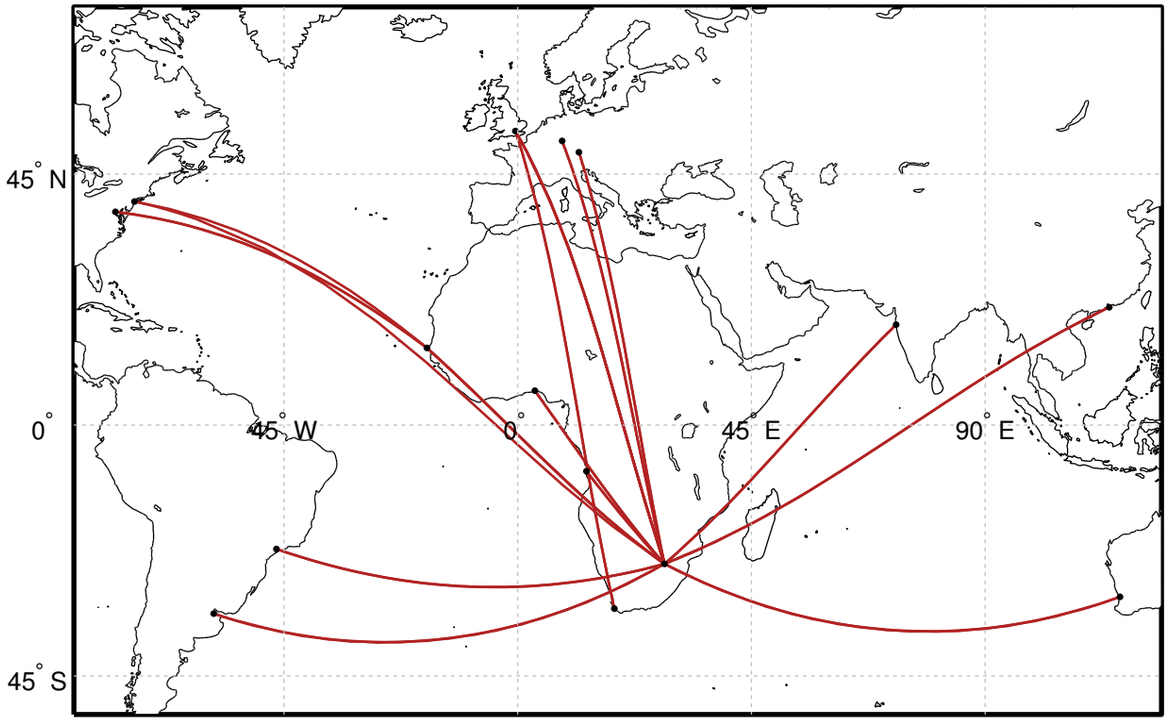
Airline Route Optimization

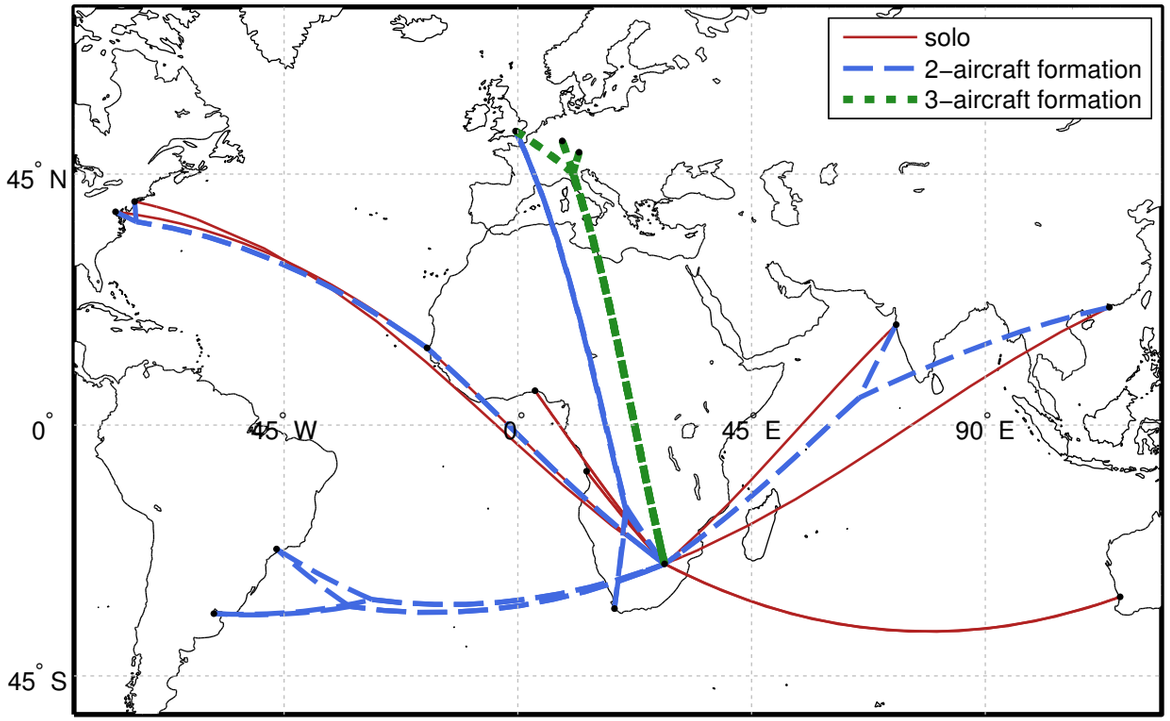


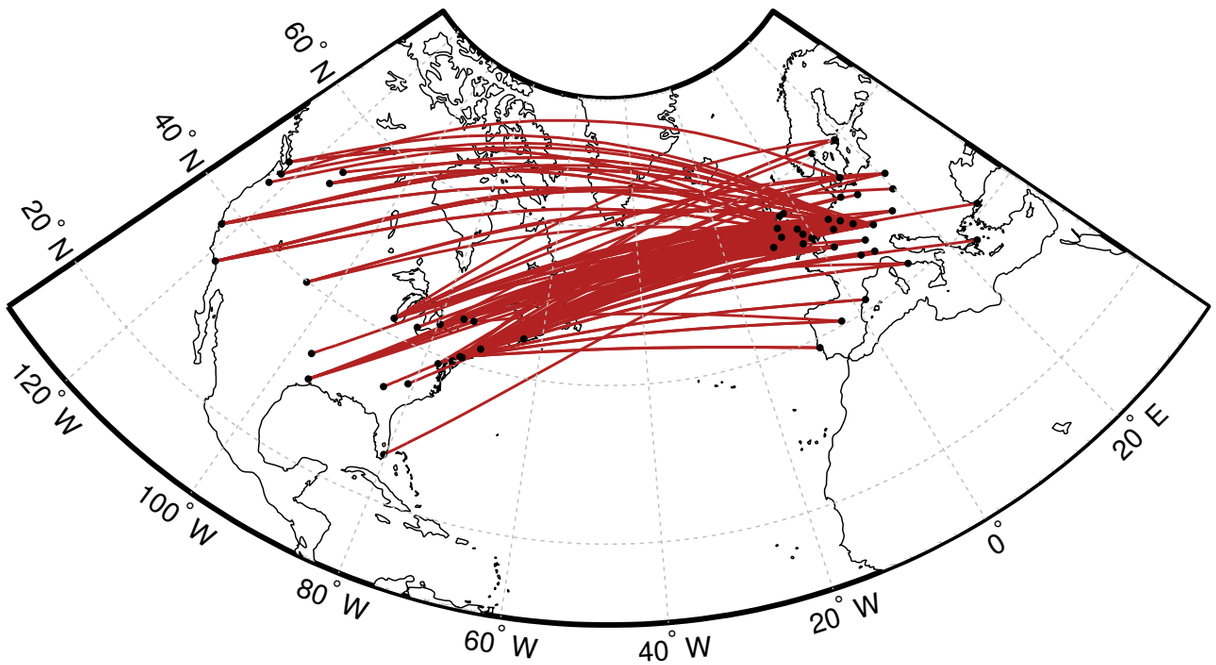


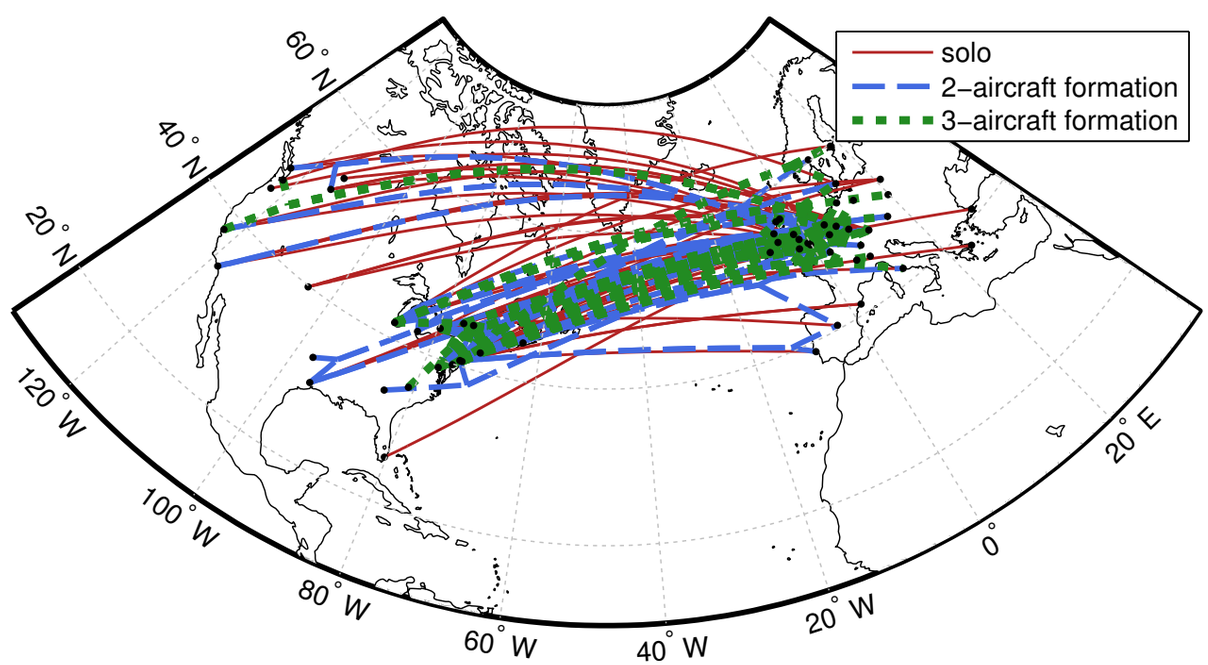


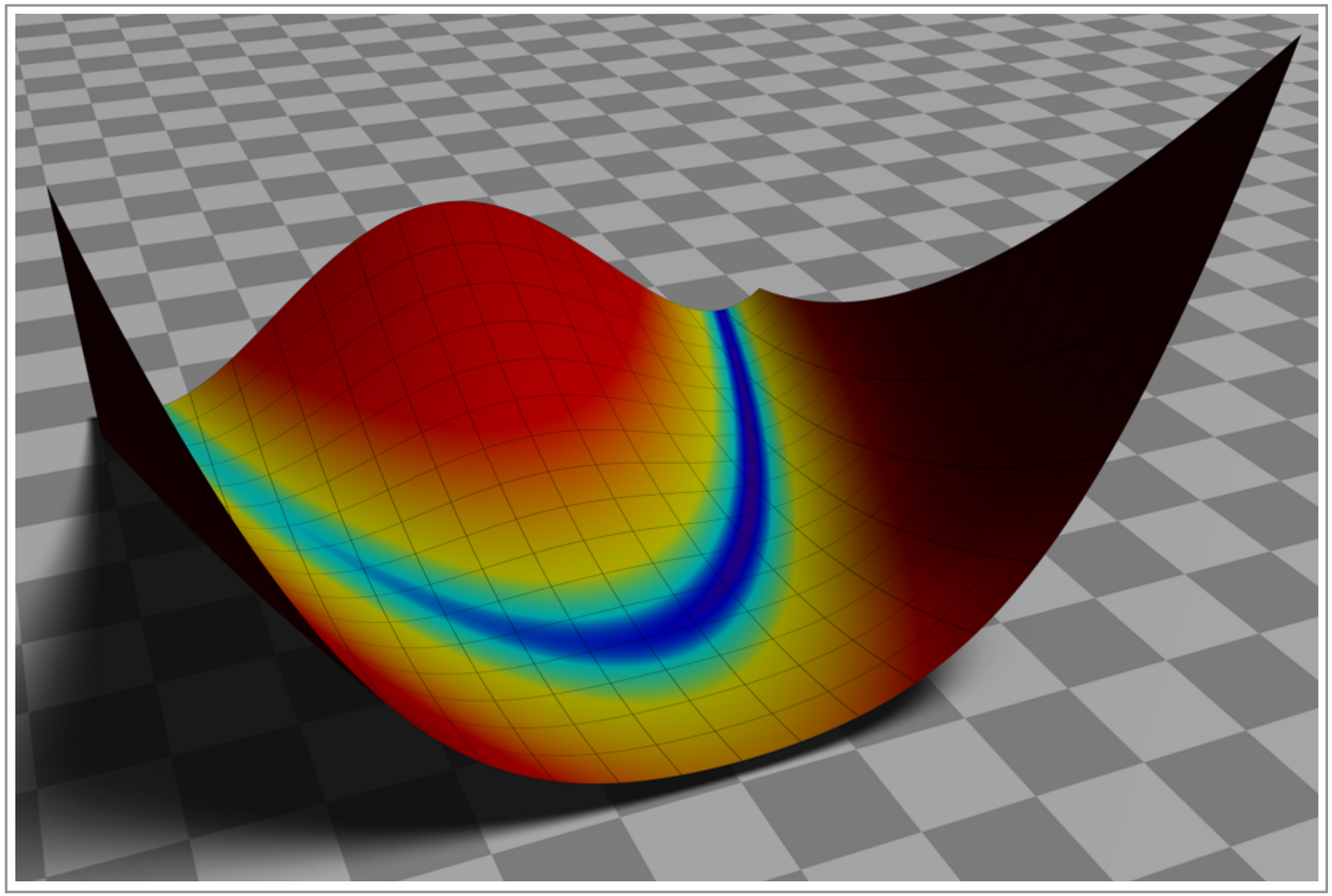
Optimized Formation Routes











Sensitivity Analysis

Sensitivity Analysis

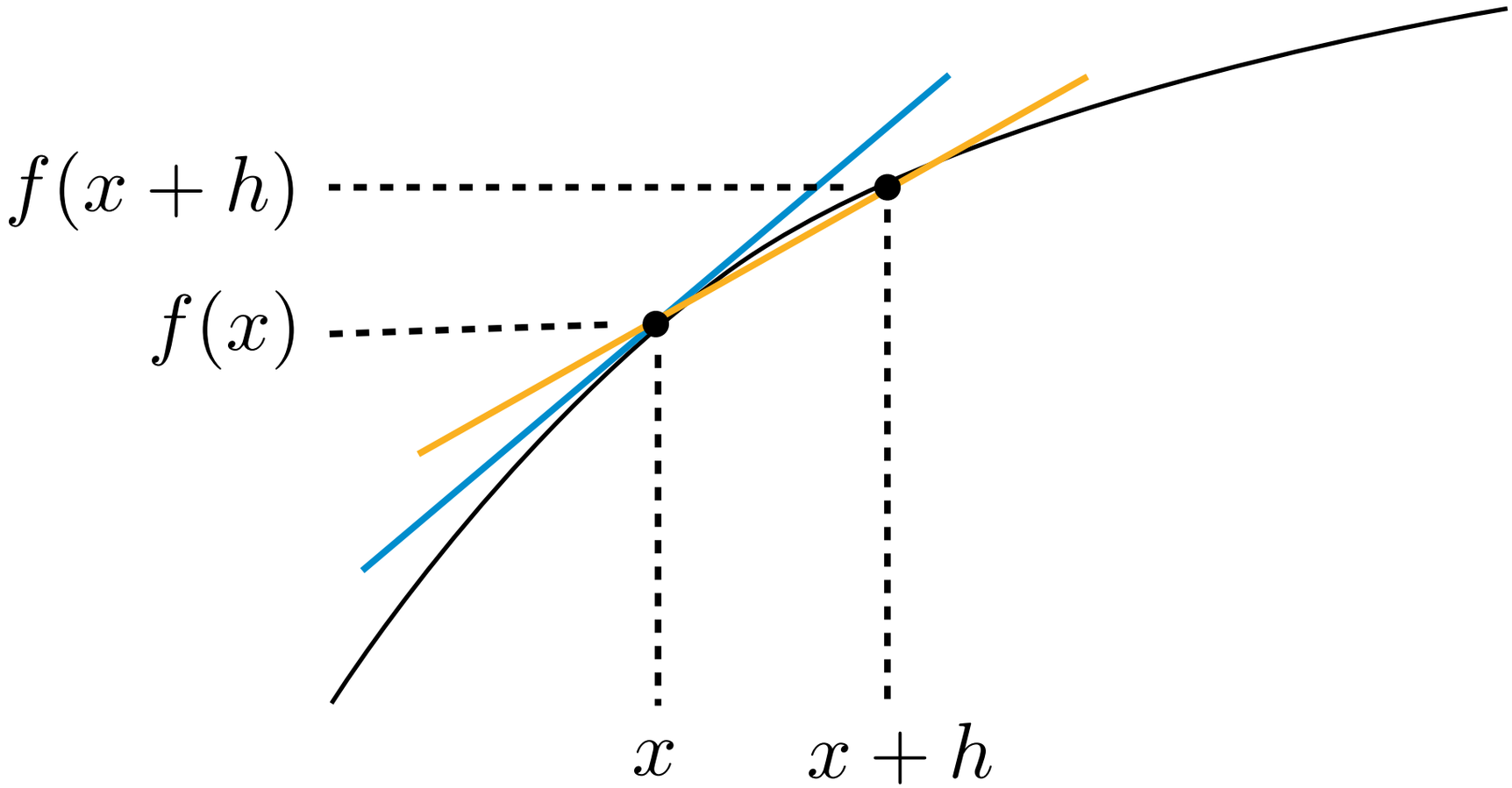
$$\begin{aligned} \text{minimize} \quad & f(x_i) \\ \text{with respect to} \quad & x_i, \quad i = 1 \ldots m \\ \text{subject to} \quad & c_j(x_i) < 0, \quad j = 1 \ldots n \end{aligned}$$

$$\frac{\partial f}{\partial x_i} \qquad \frac{\partial c_j}{\partial x_i}$$

Sensitivity Analysis

- Finite Difference
- Complex Step
- Automatic Differentiation
- Direct/Adjoint
- Coupled Derivatives

Finite Difference



$$f'(x) \approx \frac{f(x+h) - f(x)}{h} + \mathcal{O}(h)$$

Finite Difference

$$f'(x) \approx \frac{f(x+h) - f(x)}{h} + \mathcal{O}(h)$$ forward

$$f'(x) \approx \frac{f(x+h) - f(x-h)}{2h} + \mathcal{O}(h^2)$$ central

How to choose h?

Step Size

small enough to minimize truncation error

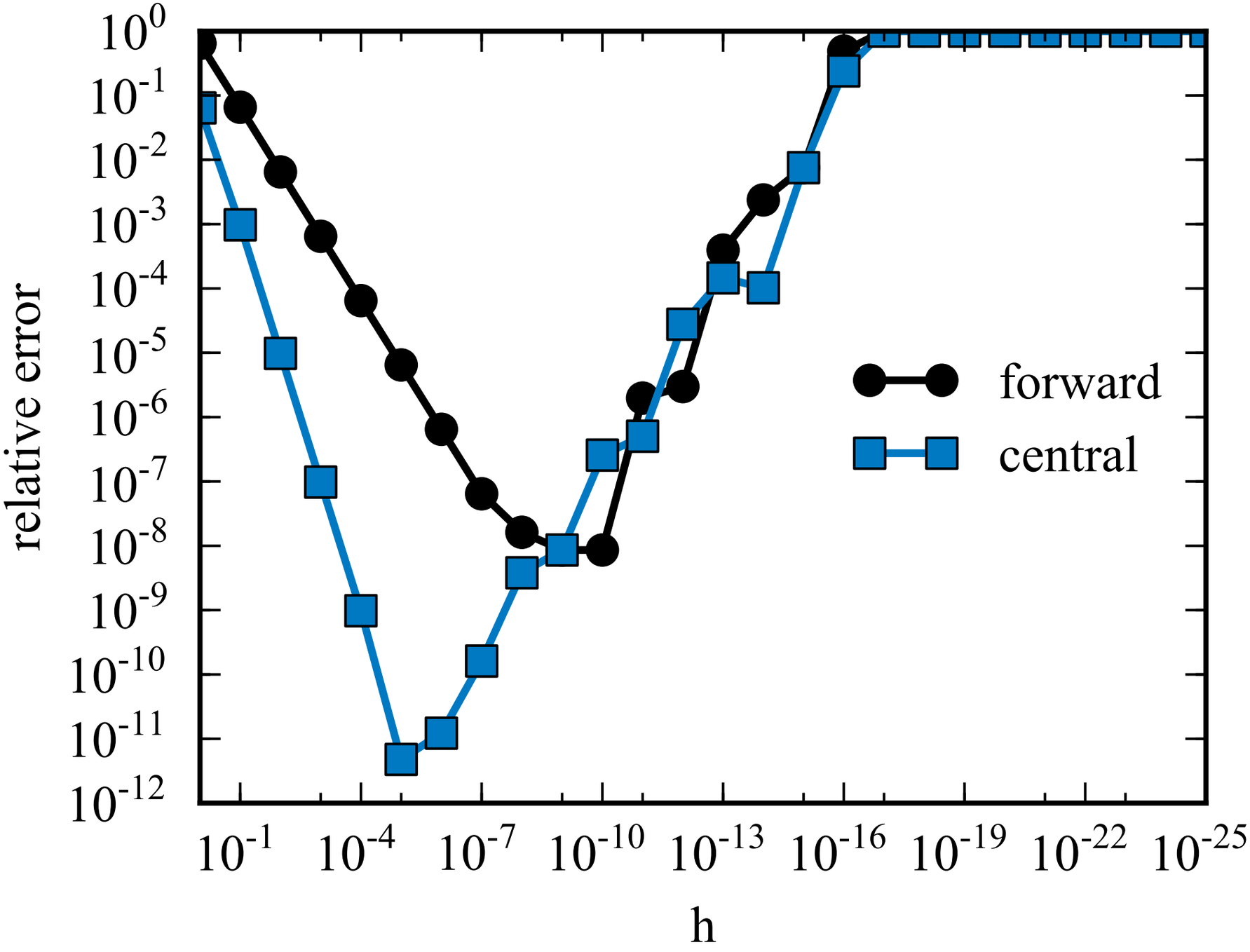
$$f'(x) \approx \frac{f(x+h) - f(x)}{h} + \mathcal{O}(h)$$

large enough to avoid subtractive cancellation

$$\begin{aligned} f(x) &= 0.790439083213615 \\ f(x+h) &= 0.790439083213637 \end{aligned}$$

How to choose h?

Step Size


relative error
10^{0}
10^{-1}
10^{-2}
10^{-3}
10^{-4}
10^{-5}
10^{-6}
10^{-7}
10^{-8}
10^{-9}
10^{-10}
10^{-11}
10^{-12}
10^{-1} 10^{-4} 10^{-7} 10^{-10} 10^{-13} 10^{-16} 10^{-19} 10^{-22} 10^{-25}
h
forward
central

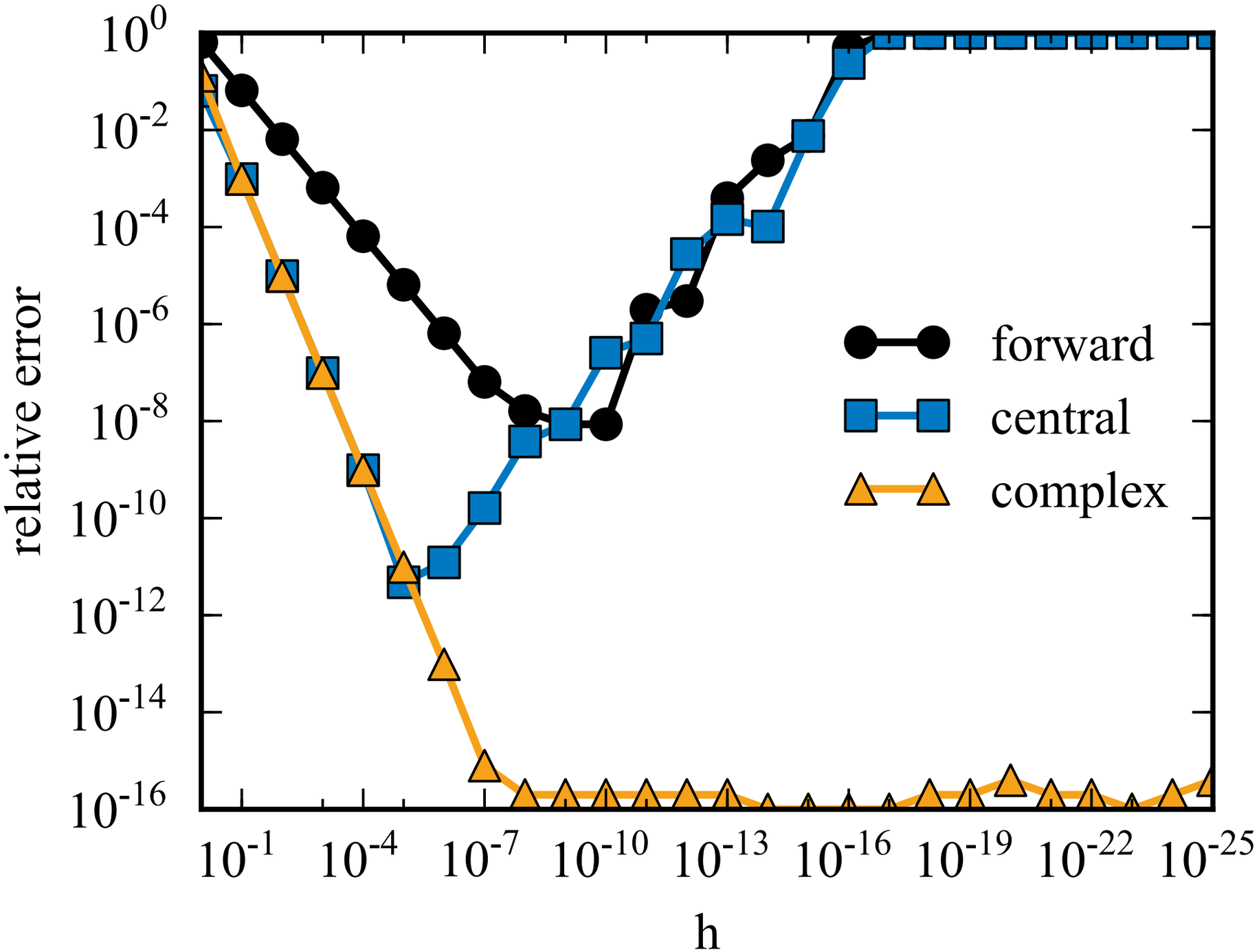
Complex Step

$$f(x+ih) = f(x) + ihf'(x) - h^2\frac{f''(x)}{2!} - ih^3\frac{f'''(x)}{3!} + \dots$$

$$f'(x) = \frac{\mathbf{I}[f(x+ih)]}{h} + \mathcal{O}(h^2)$$

no subtractive cancellation error

Step Size


relative error
10^{0}
10^{-2}
10^{-4}
10^{-6}
10^{-8}
10^{-10}
10^{-12}
10^{-14}
10^{-16}
10^{-1} 10^{-4} 10^{-7} 10^{-10} 10^{-13} 10^{-16} 10^{-19} 10^{-22} 10^{-25}
h
forward
central
complex

Implementation

- some operations should only operate on the real part (relational logic, max, min)
- absolute value must be redefined because it is not analytic
- other functions must be redefined because of the way some compilers implement them (sin, arcsin, etc.)
- for some languages (Fortran, C, Matlab, Python) there are modules, scripts, header files to make this easy

http://mdolab.engin.umich.edu/content/guide-complex-step-derivative-approximation-0

Automatic Differentiation

- chain rule applied to every variable and operation in the computer program
- accuracy of analytic derivatives
- can be automated for several languages though generally requires more changes to source code than complex step
- research has addressed many of the issues like iterative solutions, conditional statements, etc.
- in some cases it can be computationally or memory expensive

Automatic Differentiation

$$v_i = V_i(v_1, \ldots, v_{i-1})$$

$$\frac{dv_i}{dv_j}$$

forward mode: choose one v_j, move forward in i to total derivative

reverse mode: choose one v_i, move backward in j to independent variables

Implementation

- Source Code Transformation
  - source code processed with parser to produce new function
  - need to rerun script every time code is changed
  - generally the resulting code is faster
- Derived datatypes and operator overloading
  - new type where variable stores value and derivative
- Many tools are available for a variety of languages

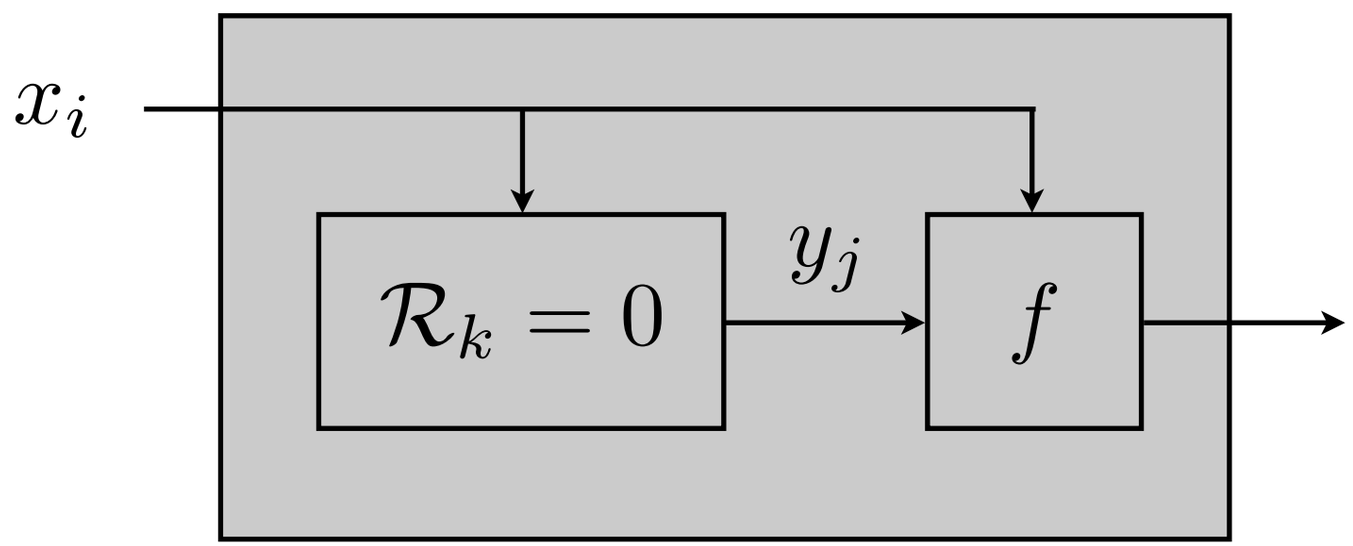
Analytic Methods

x_i design variables

y_j state variables

$f = f(x_i, y_j)$ function(s) of interest

$\mathcal{R}_k(x_i, y_j) = 0$ residual



Analytic Methods

$$f = f(x_i, y_j) \qquad \mathcal{R}_k(x_i, y_j) = 0$$

$$\frac{df}{dx_i} = \frac{\partial f}{\partial x_i} + \frac{\partial f}{\partial y_j}\frac{dy_j}{dx_i} \qquad \frac{d\mathcal{R}_k}{dx_i} = \frac{\partial \mathcal{R}_k}{\partial x_i} + \frac{\partial \mathcal{R}_k}{\partial y_j}\frac{dy_j}{dx_i} = 0$$

direct method

$$\frac{df}{dx_i} = \frac{\partial f}{\partial x_i} - \underbrace{\frac{\partial f}{\partial y_j}\overbrace{\left[\frac{\partial \mathcal{R}_k}{\partial y_j}\right]^{-1}}}_{\Psi_k}\frac{\partial \mathcal{R}_k}{\partial x_i}$$

adjoint method

Direct/Adjoint

direct method

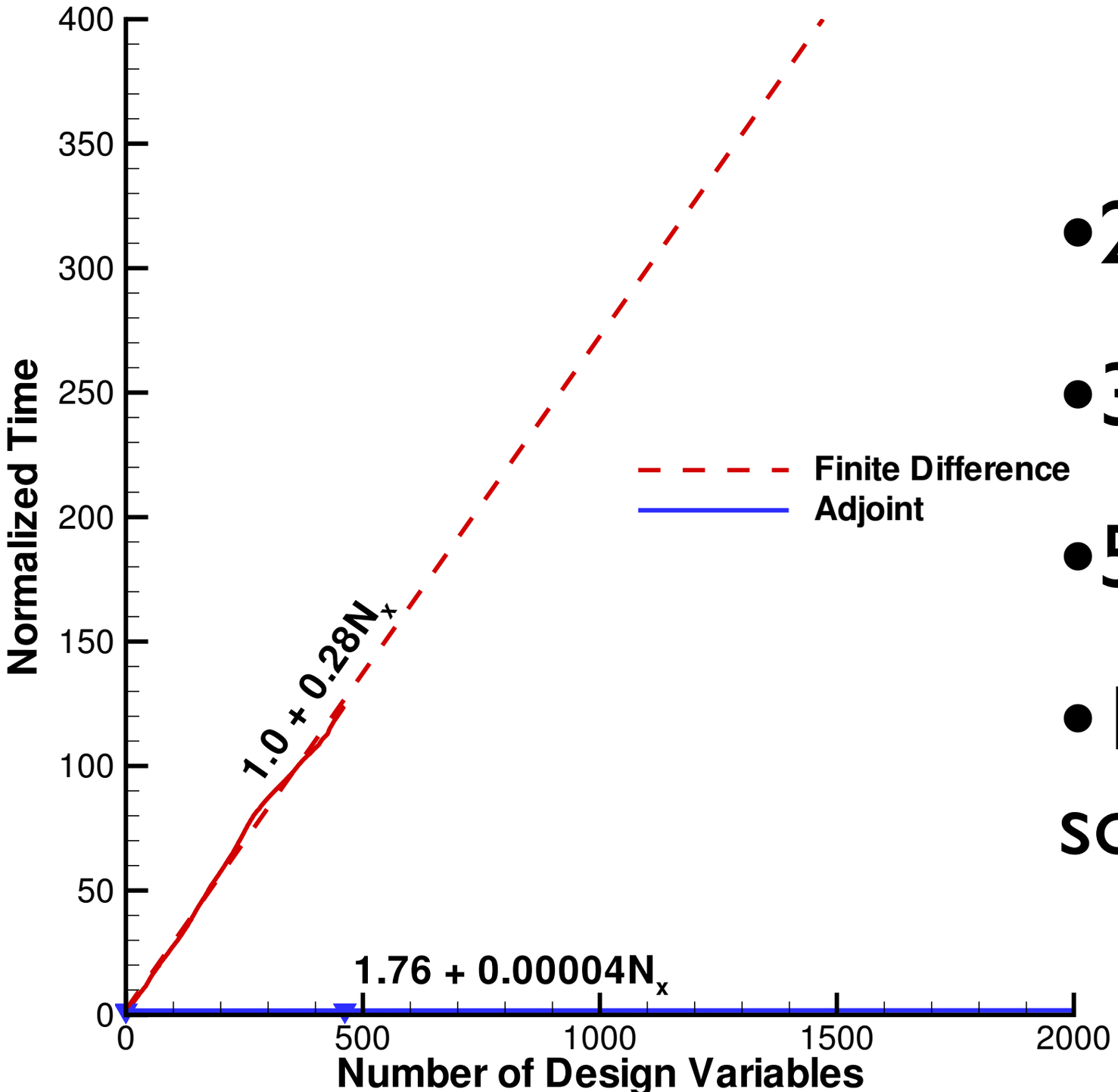
$$\frac{df}{dx_i} = \frac{\partial f}{\partial x_i} - \underbrace{\frac{\partial f}{\partial y_j}\left[\overbrace{\frac{\partial \mathcal{R}_k}{\partial y_j}}\right]^{-1}}_{\Psi_k} \frac{\partial \mathcal{R}_k}{\partial x_i}$$

adjoint method

direct: $\dfrac{\partial \mathcal{R}_k}{\partial y_j} z = \dfrac{\partial \mathcal{R}_k}{\partial x_i}$ solve for each *design variable*

adjoint: $\dfrac{\partial \mathcal{R}_k}{\partial y_j} \Psi_k = \dfrac{\partial f}{\partial y_j}$ solve for each *function*

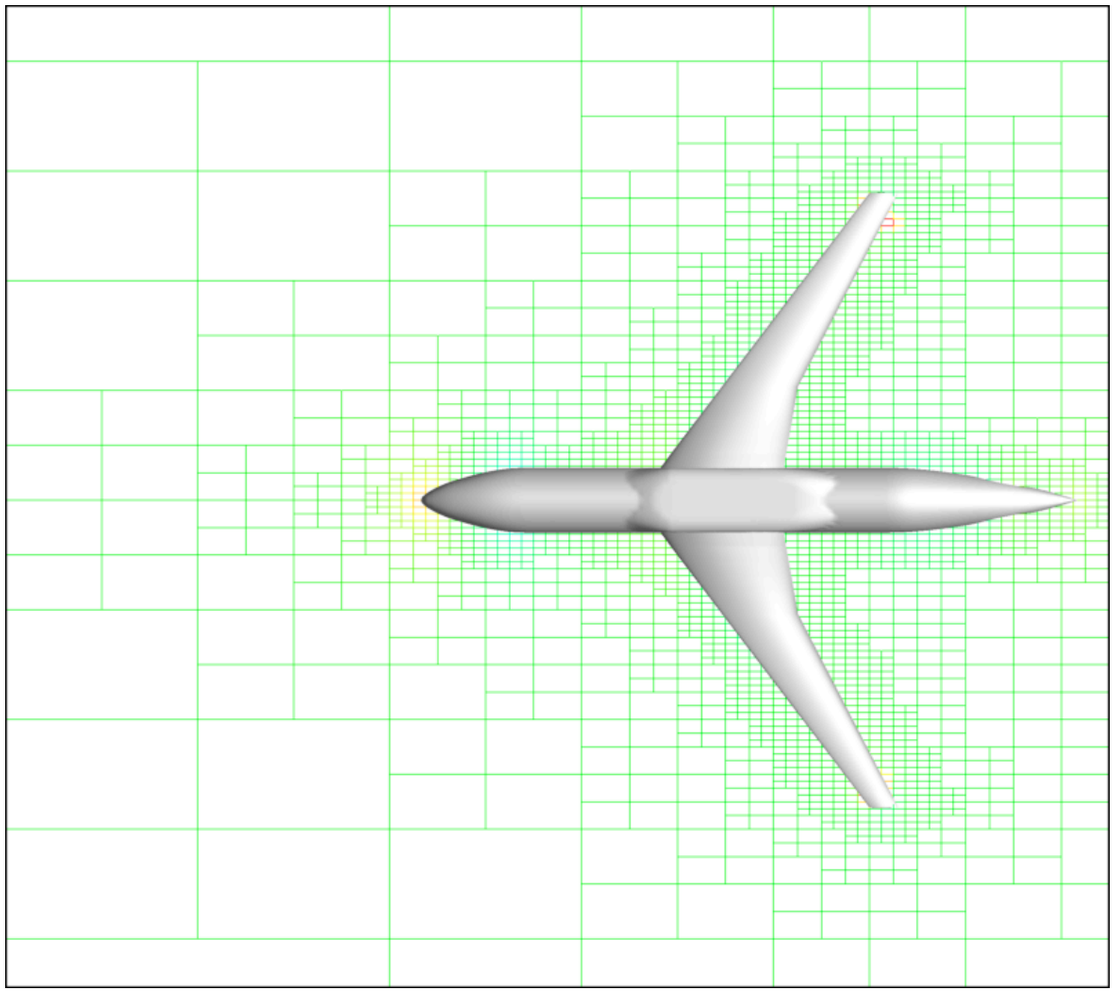
CFD/FEA Example

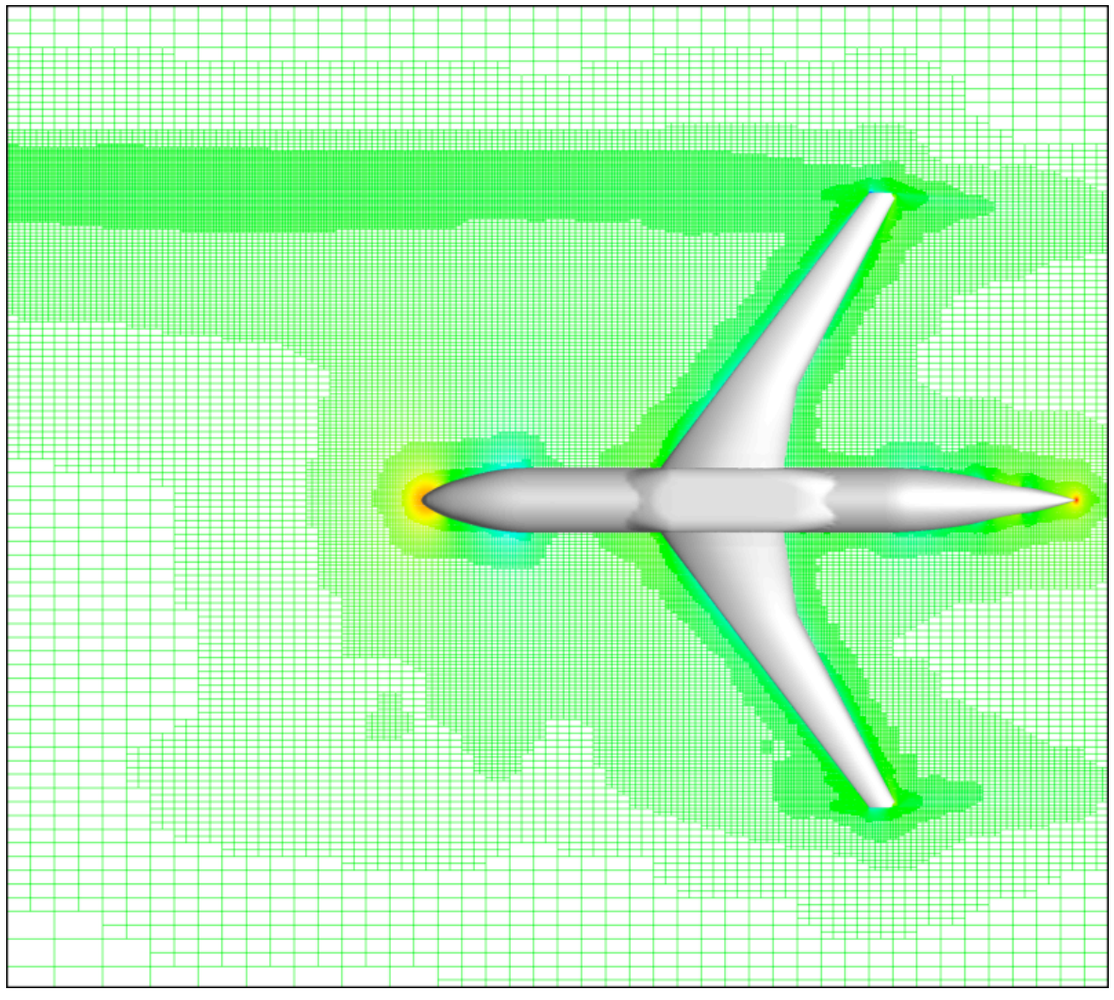


- 2M CFD cells
- 300k CSM DOFs
- 56 processors
- 1 aerostructural solution = 5.5 min

Kenway, Kennedy and Martins, AIAA Journal, 2012

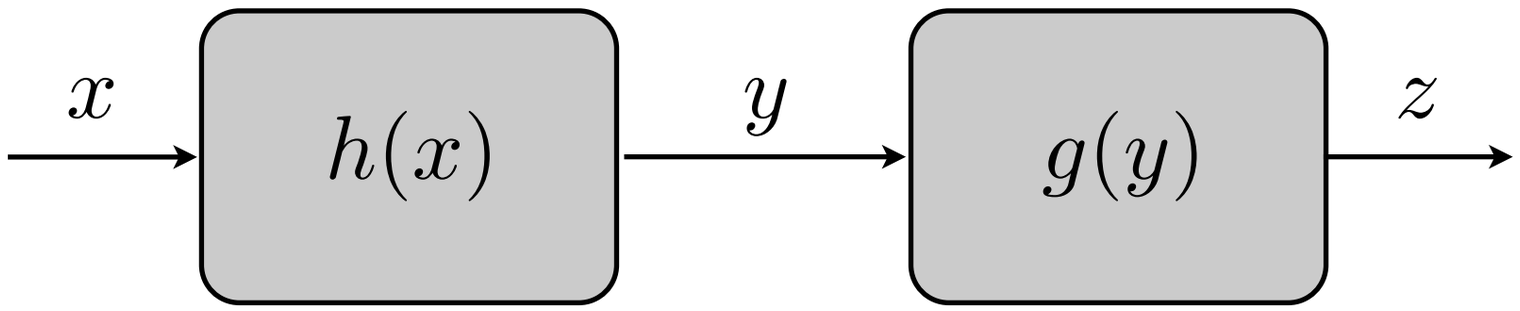
Cart3D





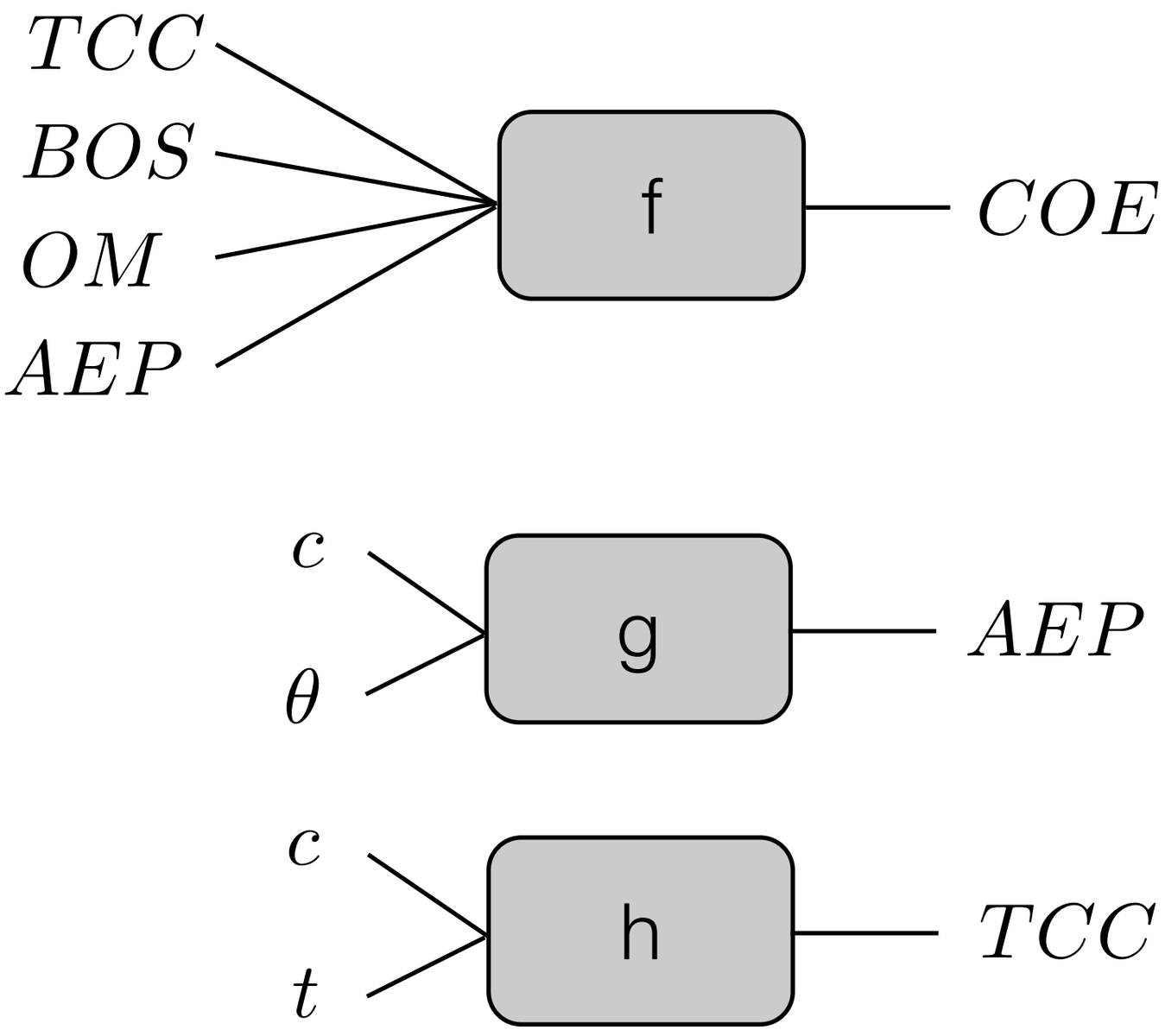
Modularize

$$f(x) = g(h(x))$$



$$\frac{dz}{dx} = \frac{dz}{dy}\frac{dy}{dx}$$

Simple Cost Example



Simple Cost Example

$$COE = \frac{(TCC + BOS)(1 + cr)fcr + OM}{AEP}$$

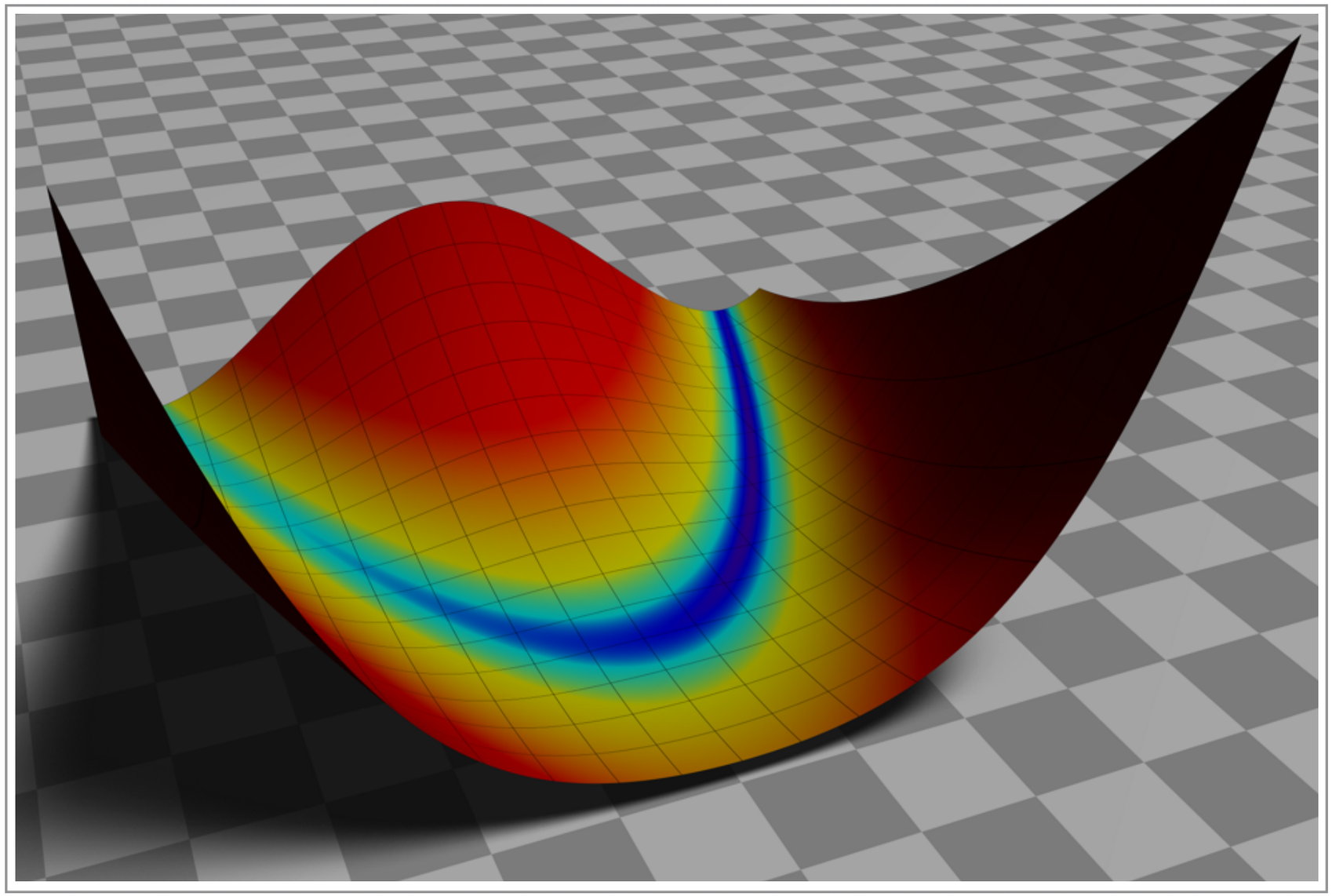
$$\frac{\partial COE}{\partial TCC} = \frac{(1 + cr)fcr}{AEP} \qquad \frac{\partial COE}{\partial AEP} = -\frac{COE}{AEP}$$

$$\frac{\partial TCC}{\partial c} = \text{AD or FD} \qquad \frac{\partial AEP}{\partial c} = \text{direct/adjoint and AD}$$

$$\frac{dCOE}{dc} = \frac{\partial COE}{\partial AEP}\frac{\partial AEP}{\partial c} + \frac{\partial COE}{\partial TCC}\frac{\partial TCC}{\partial c} + \dots$$

Sensitivity Analysis

- Use “analytic” gradients if...
  - at all possible (the required effort is almost always worth it)
- Use finite-difference if...
  - you don’t have access to the source code
  - the dimensionality of your problem is not high, and you have been careful in your analysis
  - you’ve modularized your code and some submodules are difficult to get gradients for



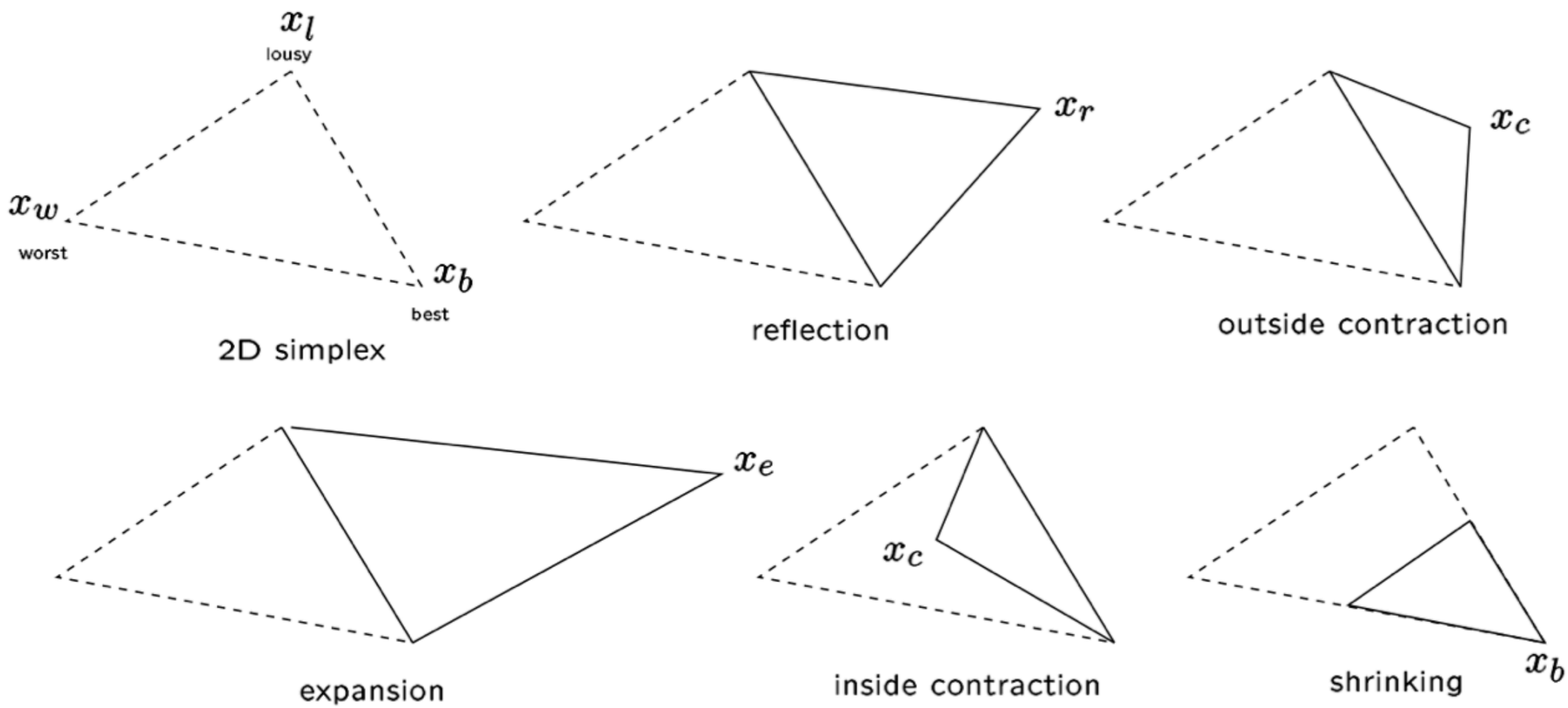
Gradient-Free Methods and Response Surfaces

Gradient-Free Methods

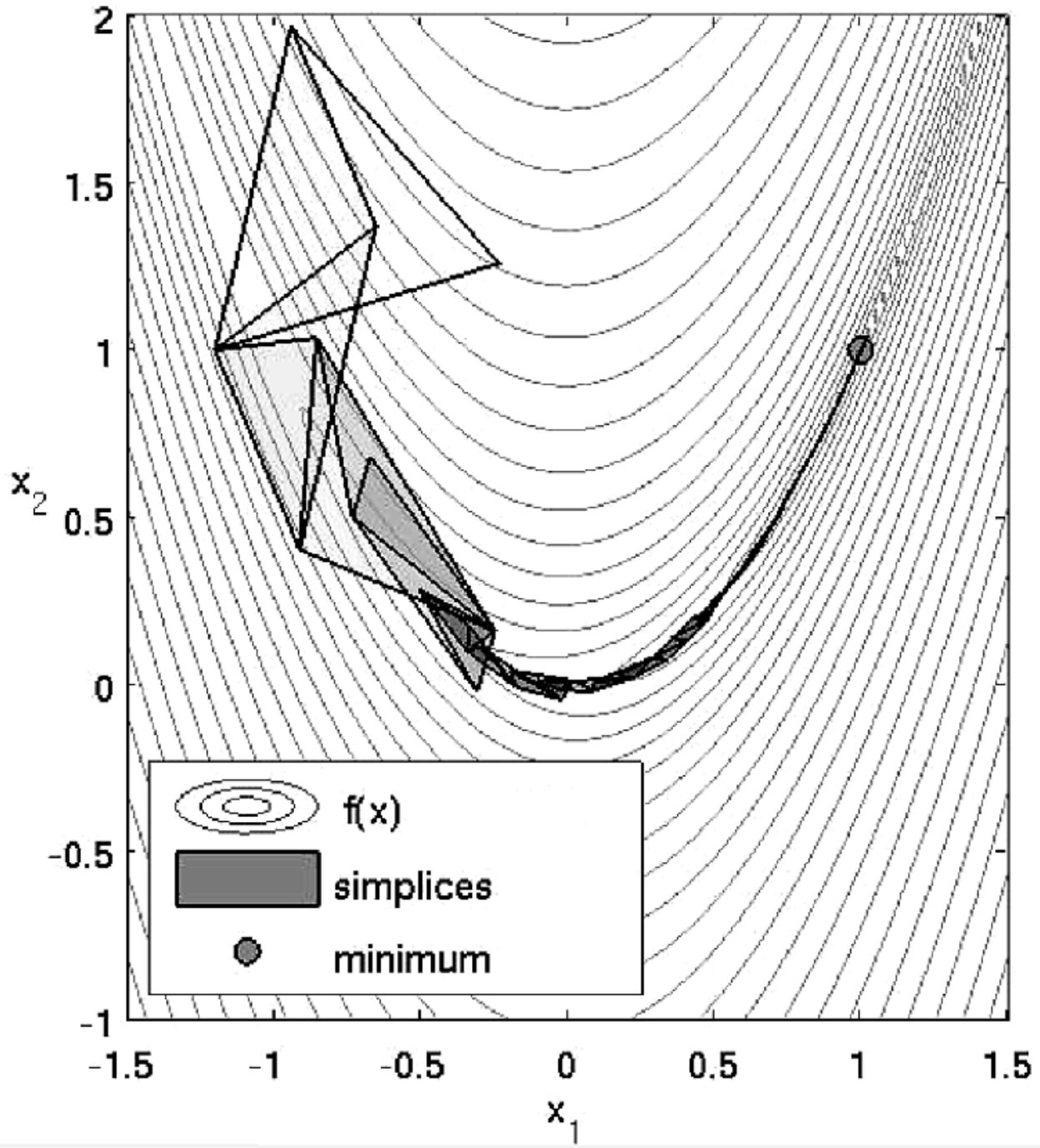
- useful if analysis is non-differentiable, discontinuous, or highly multi-modal
- generally easy to setup and more forgiving of deficiencies in the analysis
- these methods primarily rely on heuristics
- generally they will not find a true optimum, but can find many *good* solutions
- methods: Nelder-Mead Simplex, Genetic Algorithm, Particle Swarm

Nelder-Mead Simplex

- simplex is a hypertetrahedron (triangle in 2D, tetrahedron in 3D)
- generally only works for < 10 dimensions



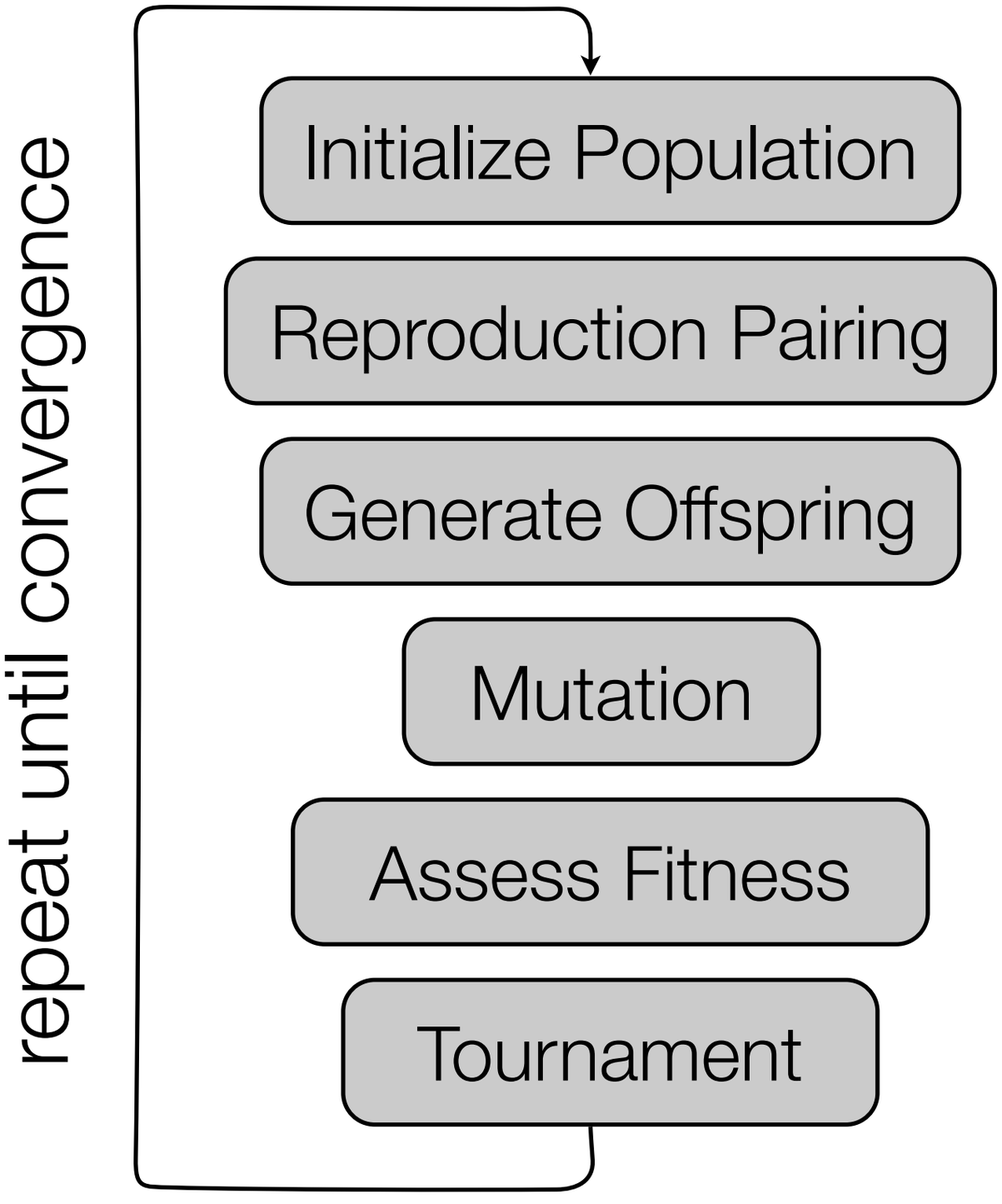
Nelder-Mead Simplex



Genetic Algorithm

- inspired by evolution
- three key aspects: selection (survival of the fittest), crossover (reproduction), mutation (variation)

Genetic Algorithm



GA Plus/Minus

- Advantages:
  - Whole populations are evaluated at the same time (embarrassingly parallel)
  - Easily used with multiobjective optimization
  - Can utilize discrete design variables
  - Easy to implement
- Disadvantages:
  - Very expensive compared to gradient-based methods.

Particle Swarm Optimization

- applies concept of swarm intelligence
- each agent (representing a design point) moves in n-dimensional space searching for best solution
- updates position based on inertia, knowledge of best solution agent has found, and knowledge of best solution swarm has found

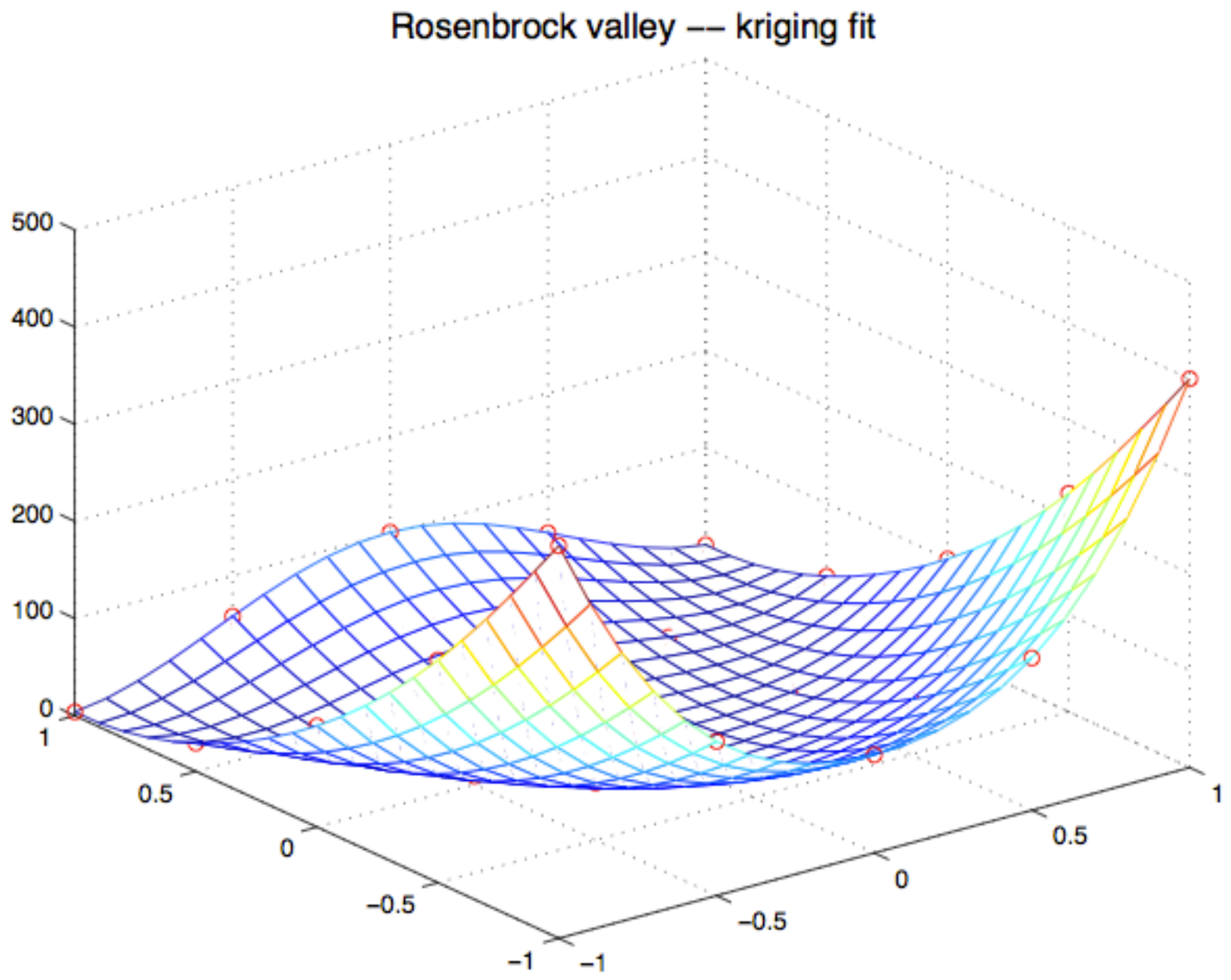
Algorithm Selection

- Use a gradient method if...
  - at all possible (the required effort is almost always worth it)
- Use a gradient-free method if...
  - your problem (not your code) is fundamentally discontinuous, non-smooth or highly multi-modal
  - or you don’t have access to the source code and the gradients are unreliable
  - or getting a quick decent answer is more important than finding “the optimum”
  - or the problem is well-suited for hybrid-optimization
  - and the dimensionality of your problem is low

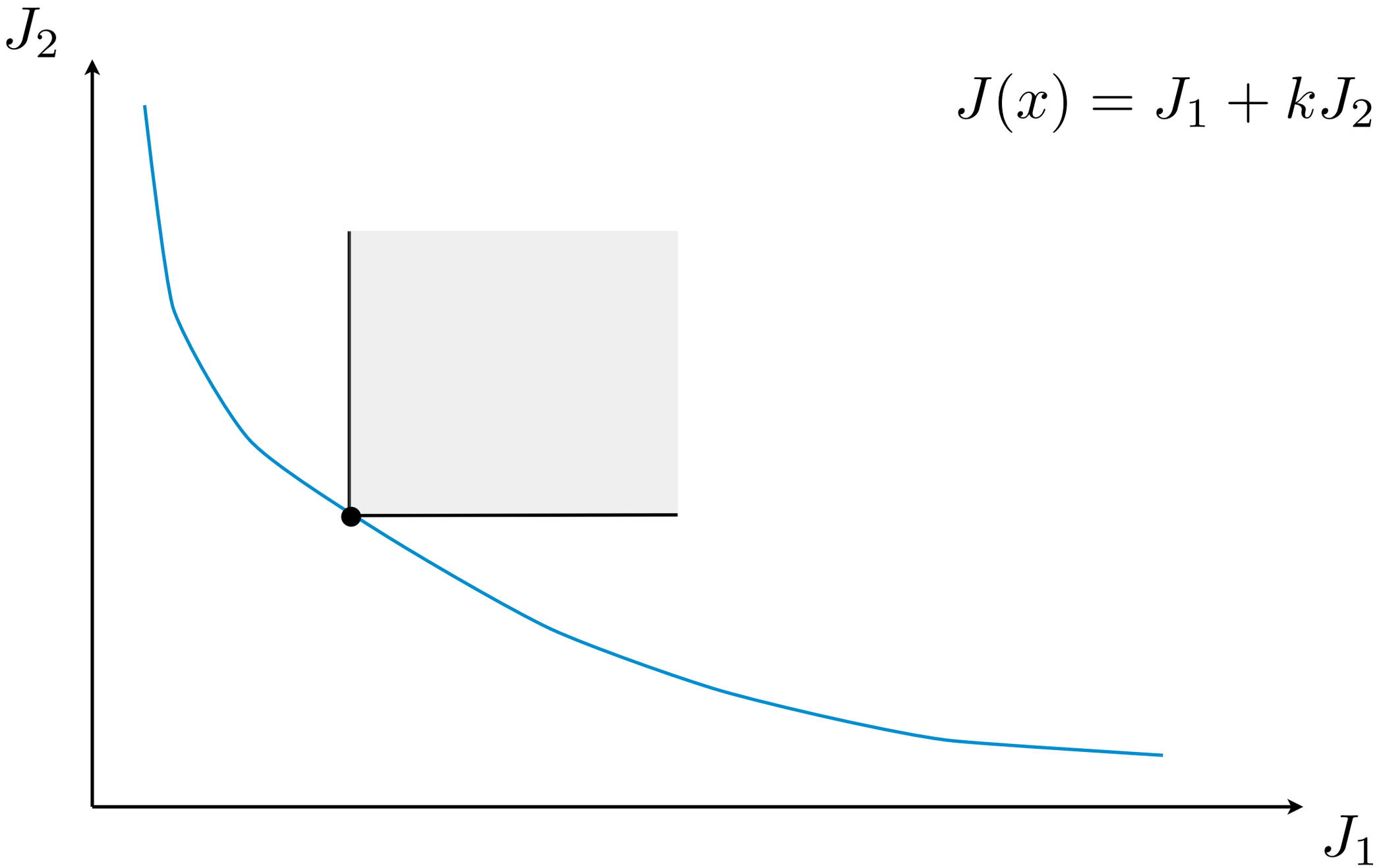
Surrogate Modeling

- Trust Region Surrogate Management
- Kriging
- many others ...

[Akima (1D)]

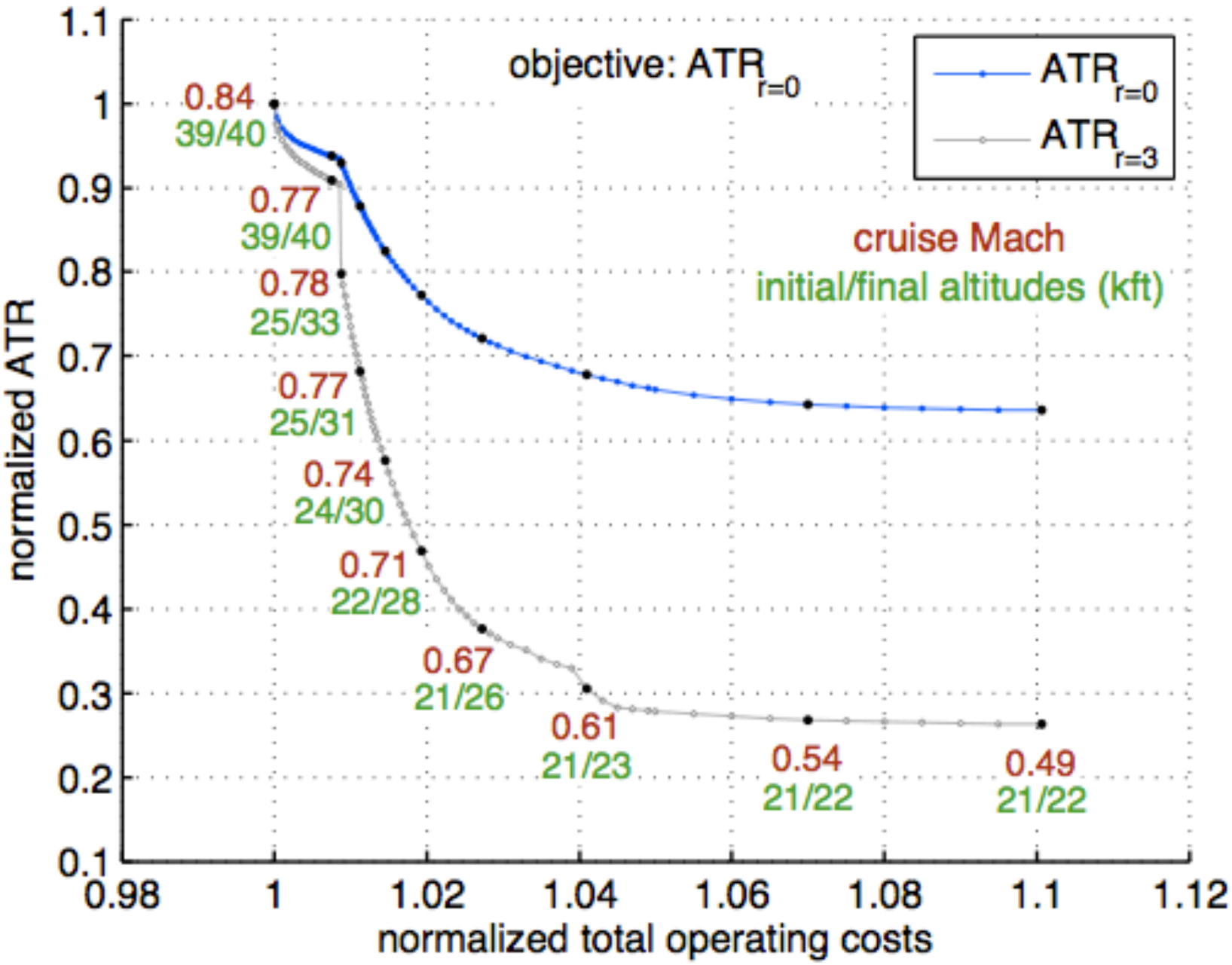


Pareto Front

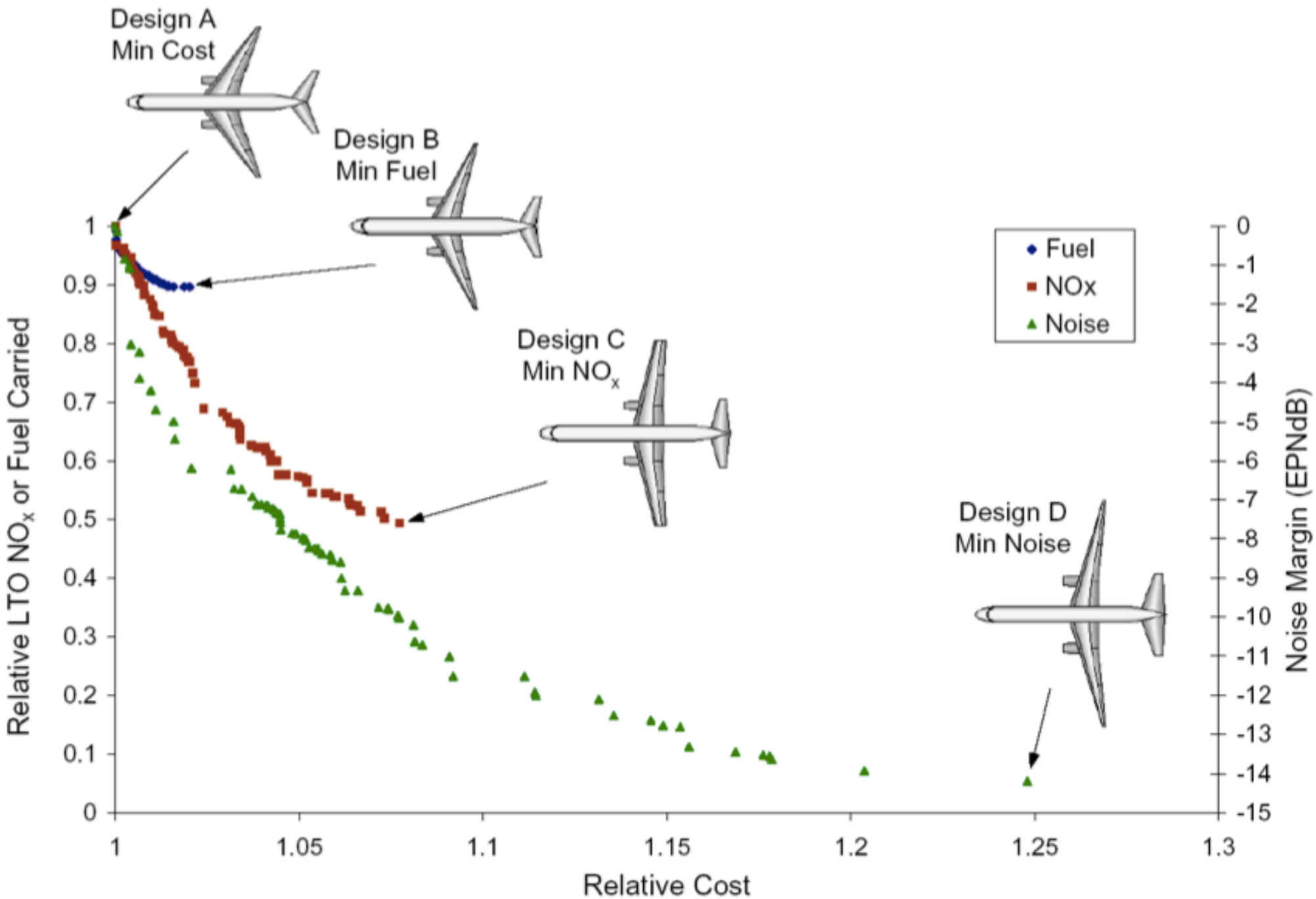


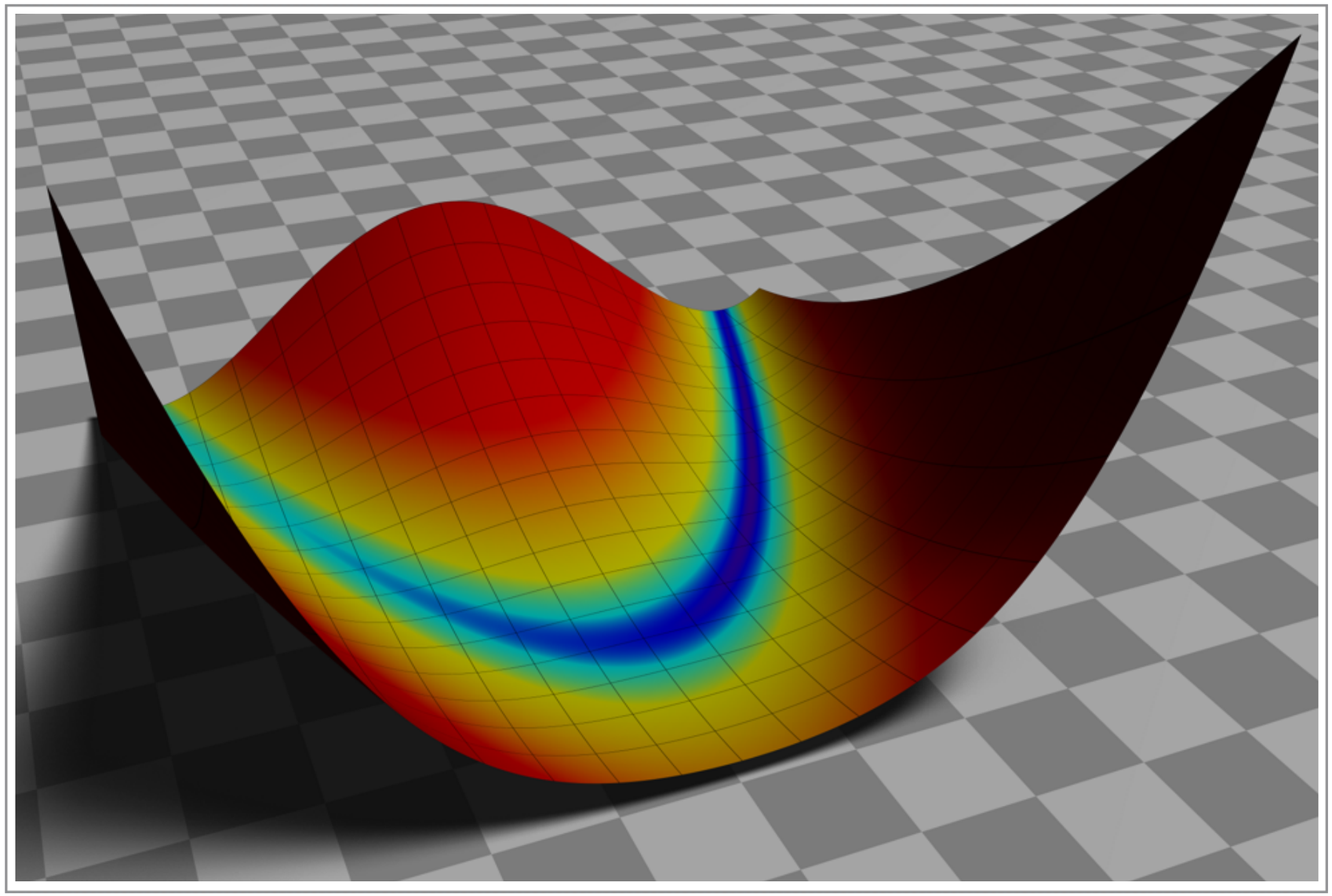
$$J(x) = J_1 + kJ_2$$

Multi-Objective Optimization



Multi-Objective Optimization





Problem Formulation Strategies

Source Code

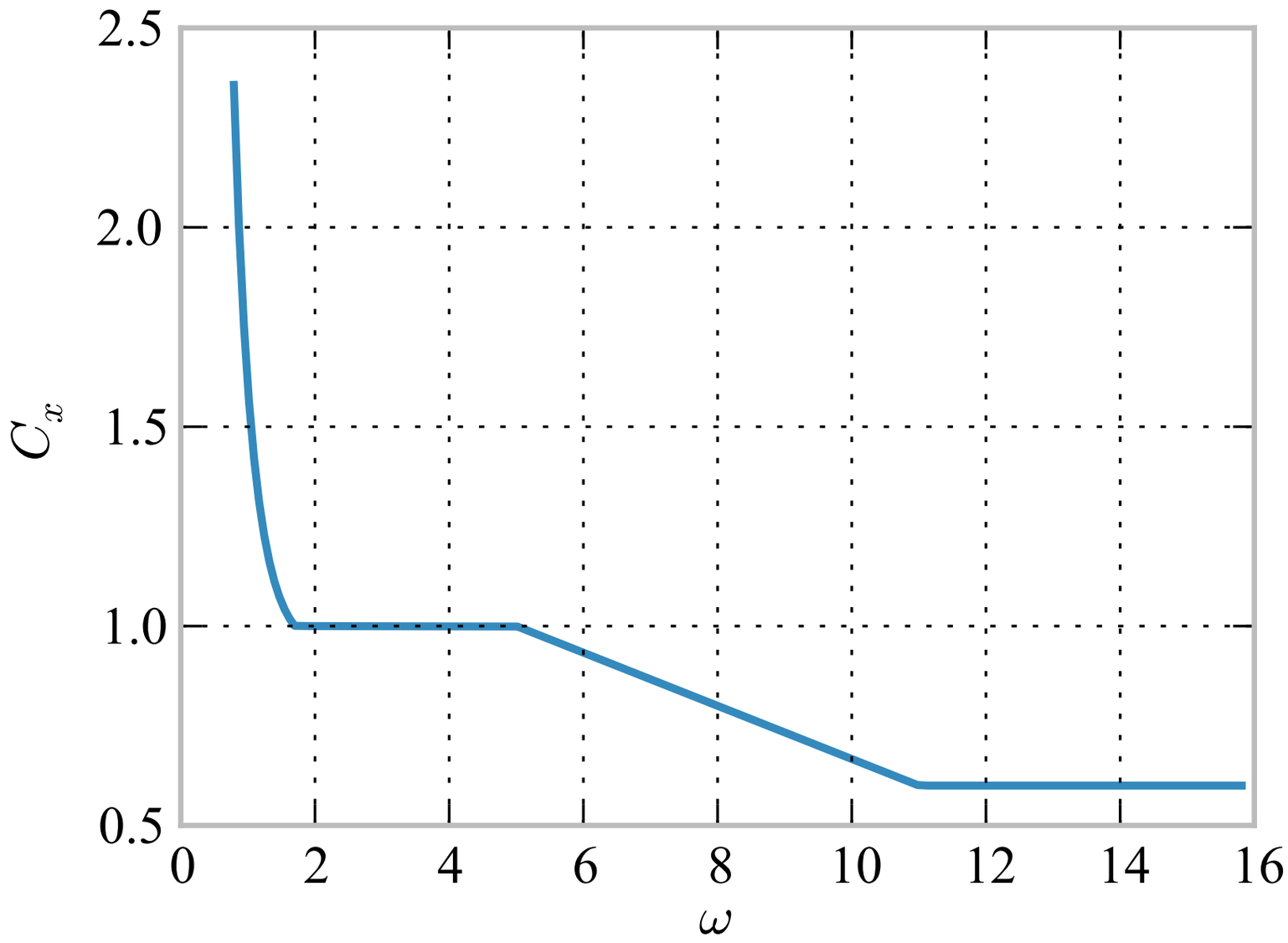
- Always keep C^0 and C^1 continuity "smoothness" in mind when writing your analysis
- Input files are excellent, but provide a way to access your code without the input file
- If possible, supply gradients (optionally) along with function values

“Smoothing” the Analysis

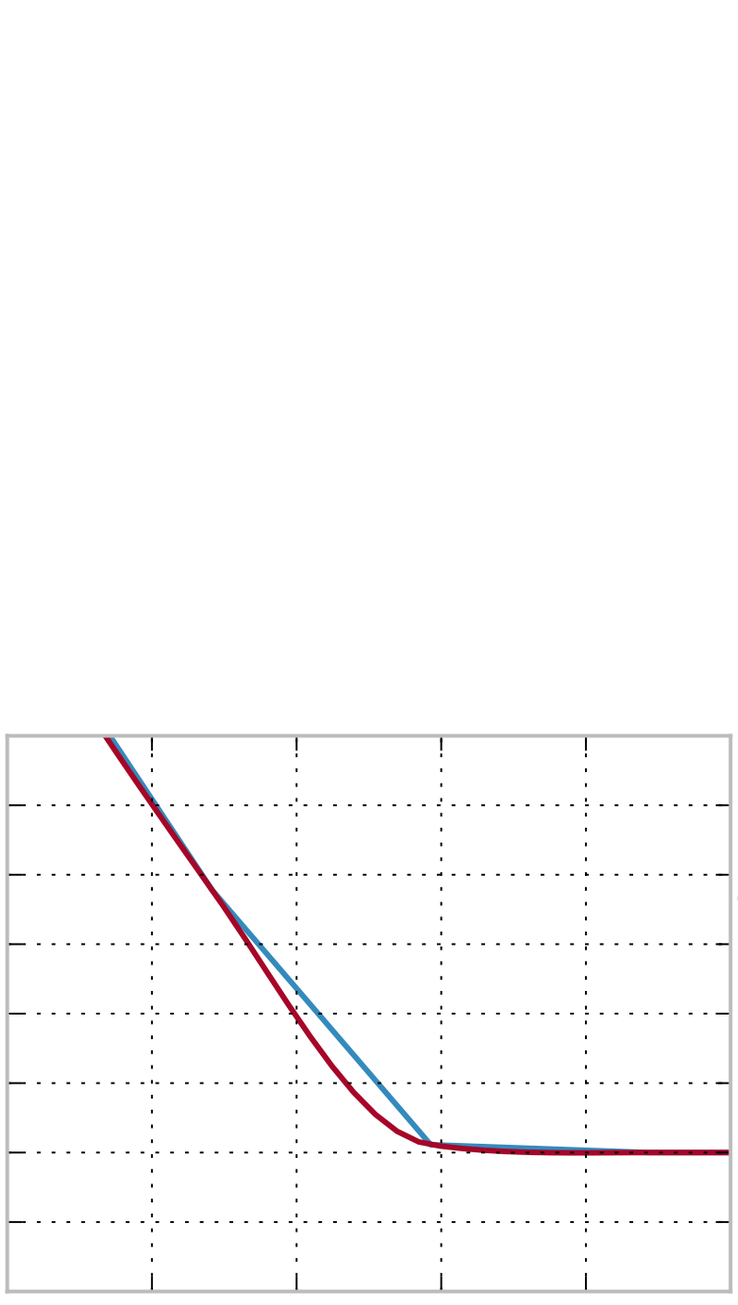
- closed-source: surrogate modeling
- open-source: modify code
- writing code: plan analysis development with optimization in mind

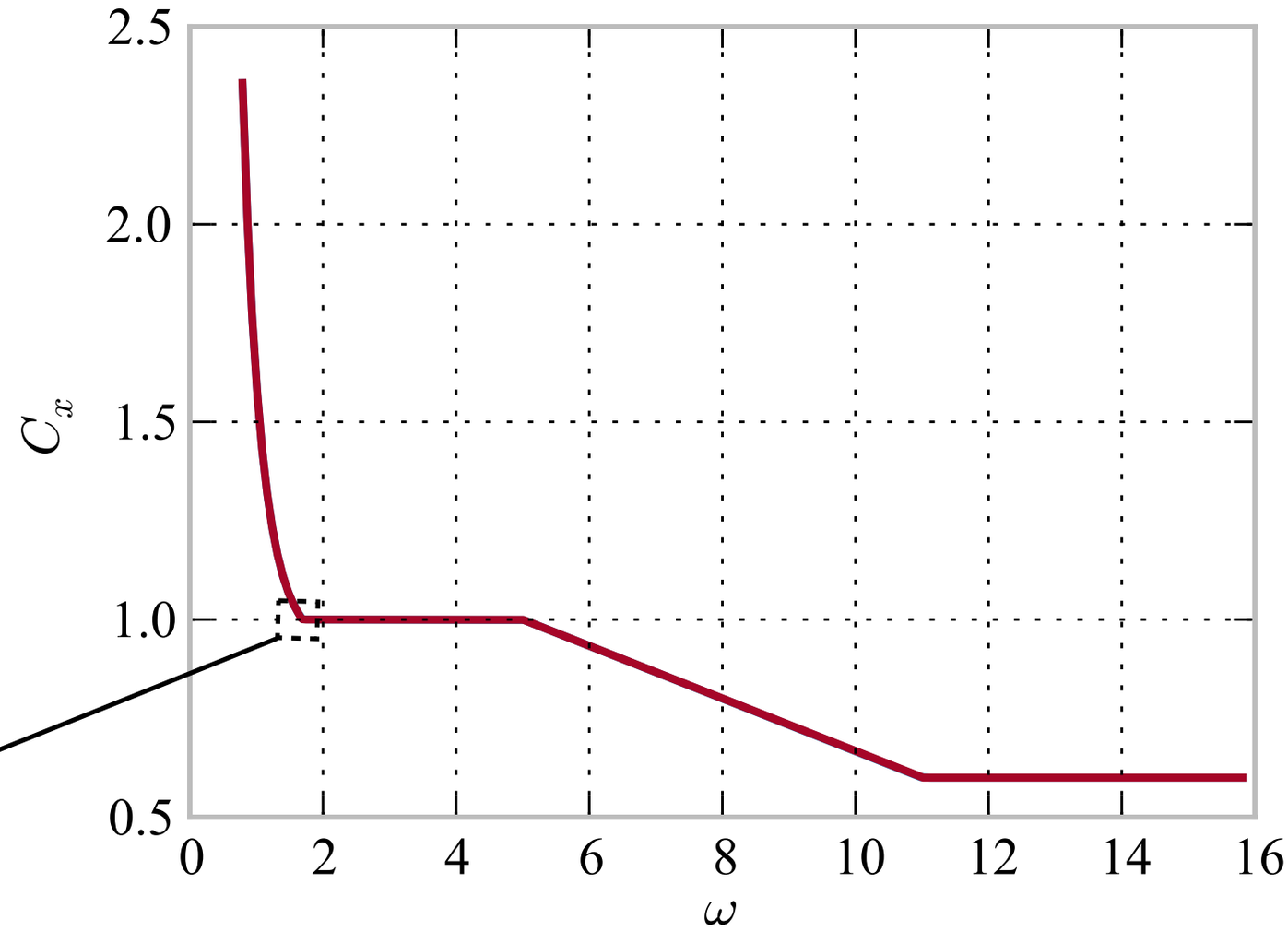
max, min, abs, piecewise functions, convergence loops, noisy output, empirical models, discretization

“Smoothing” the Analysis



“Smoothing” the Analysis



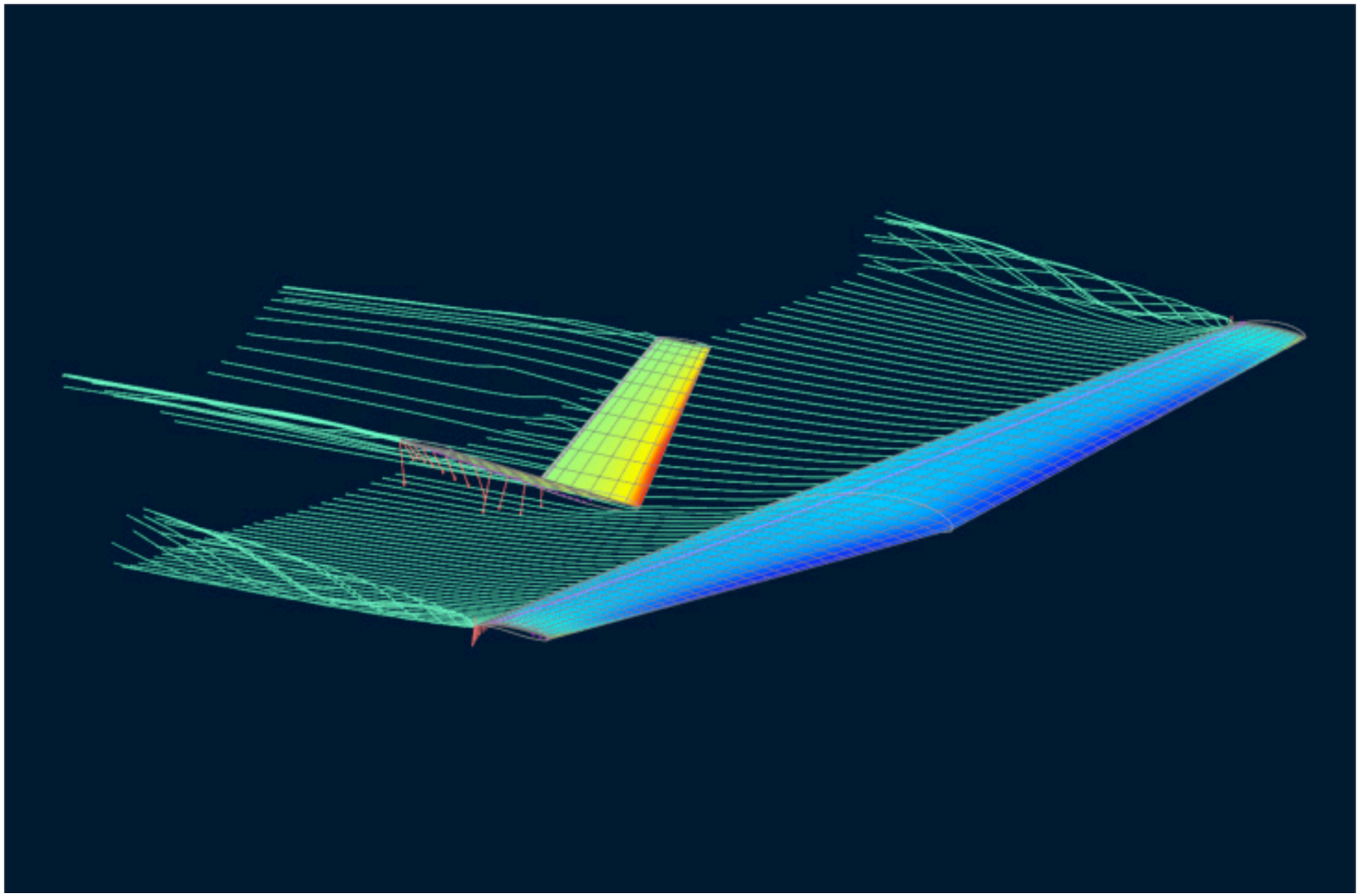


Reformulate Constraints

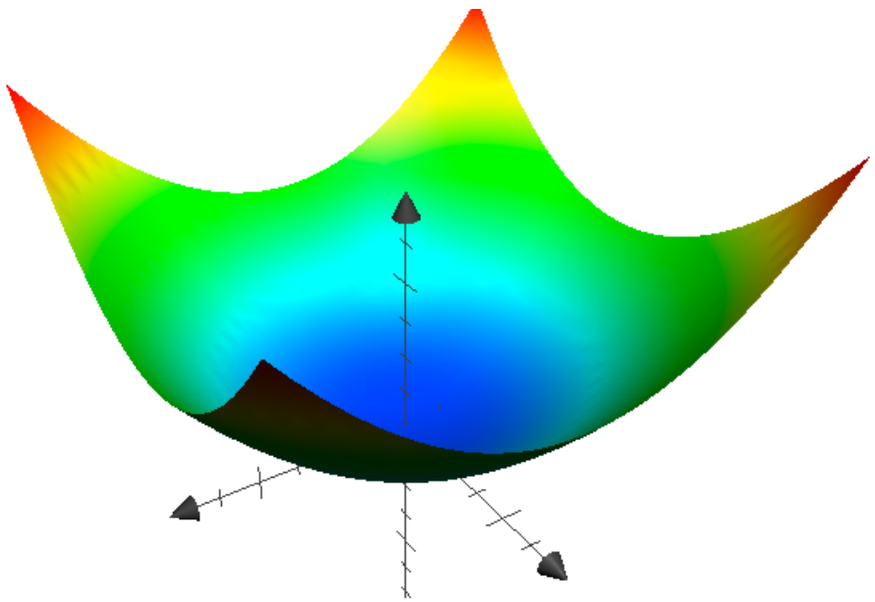
$$\max_i \sigma_i < \sigma_y \quad \Rightarrow \quad \sigma_i < \sigma_y$$

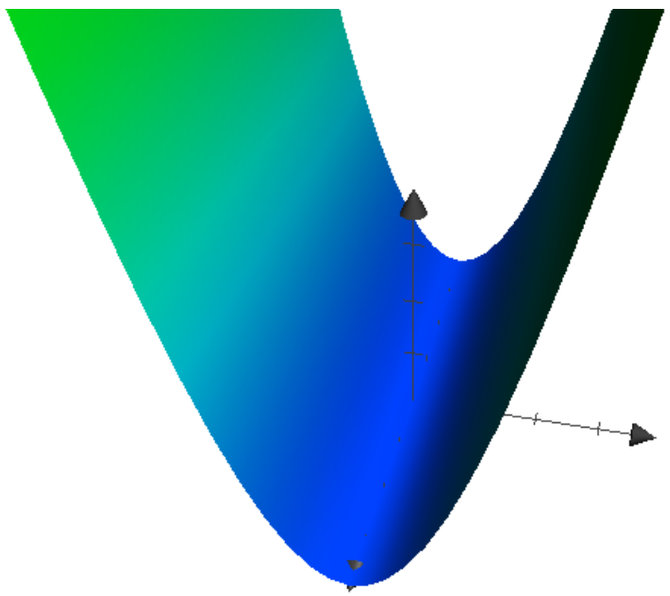
$$|\sigma| < \sigma_y \quad \Rightarrow \quad \begin{array}{l} \sigma < \sigma_y \\ \sigma > -\sigma_y \end{array}$$

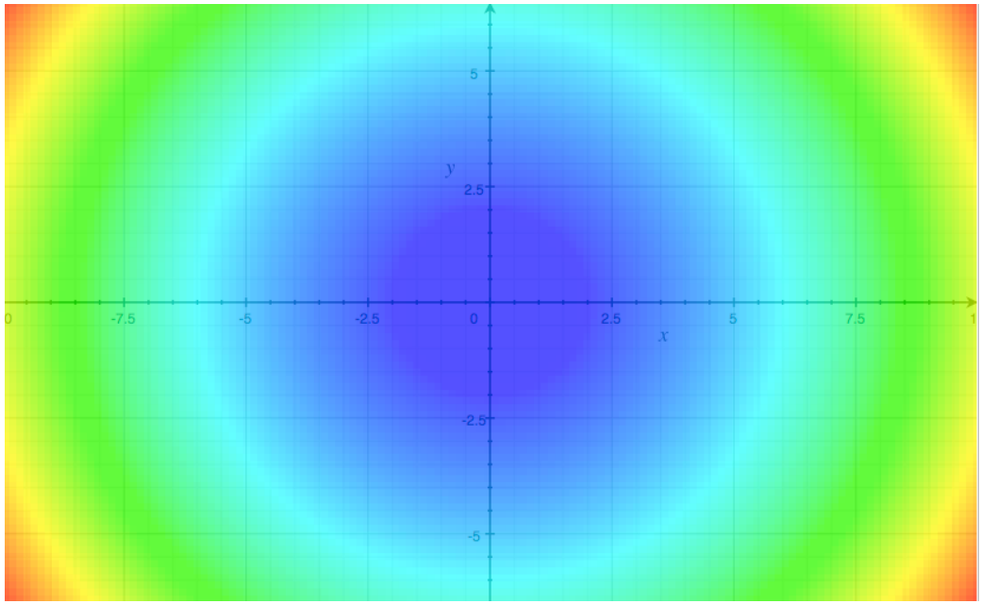
Outer iterations

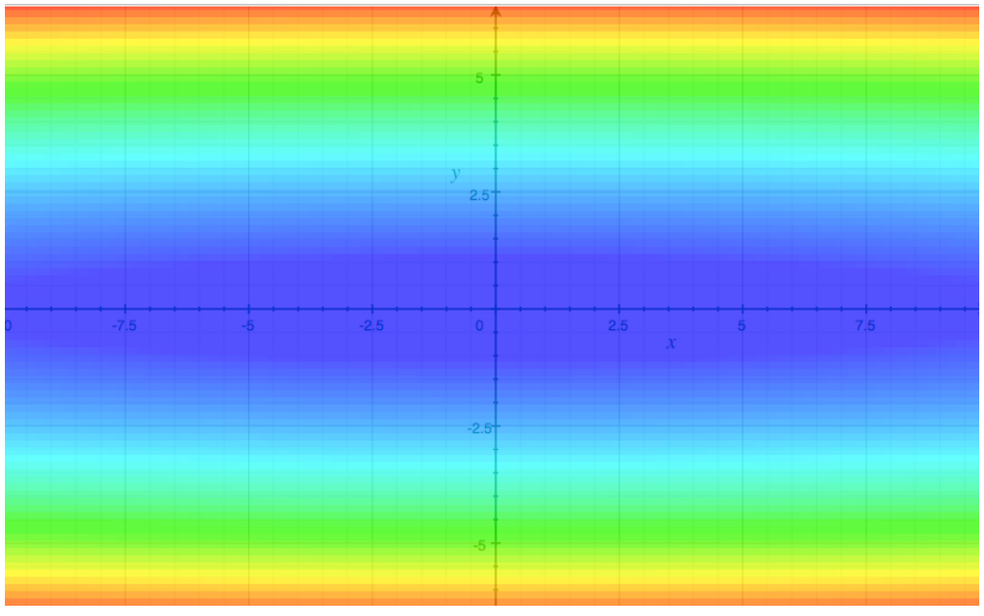


Scaling









Scaling

- affects step size, stopping criteria, condition number

- As a first cut, scale all design variables, objectives, and constraints to be of order one.

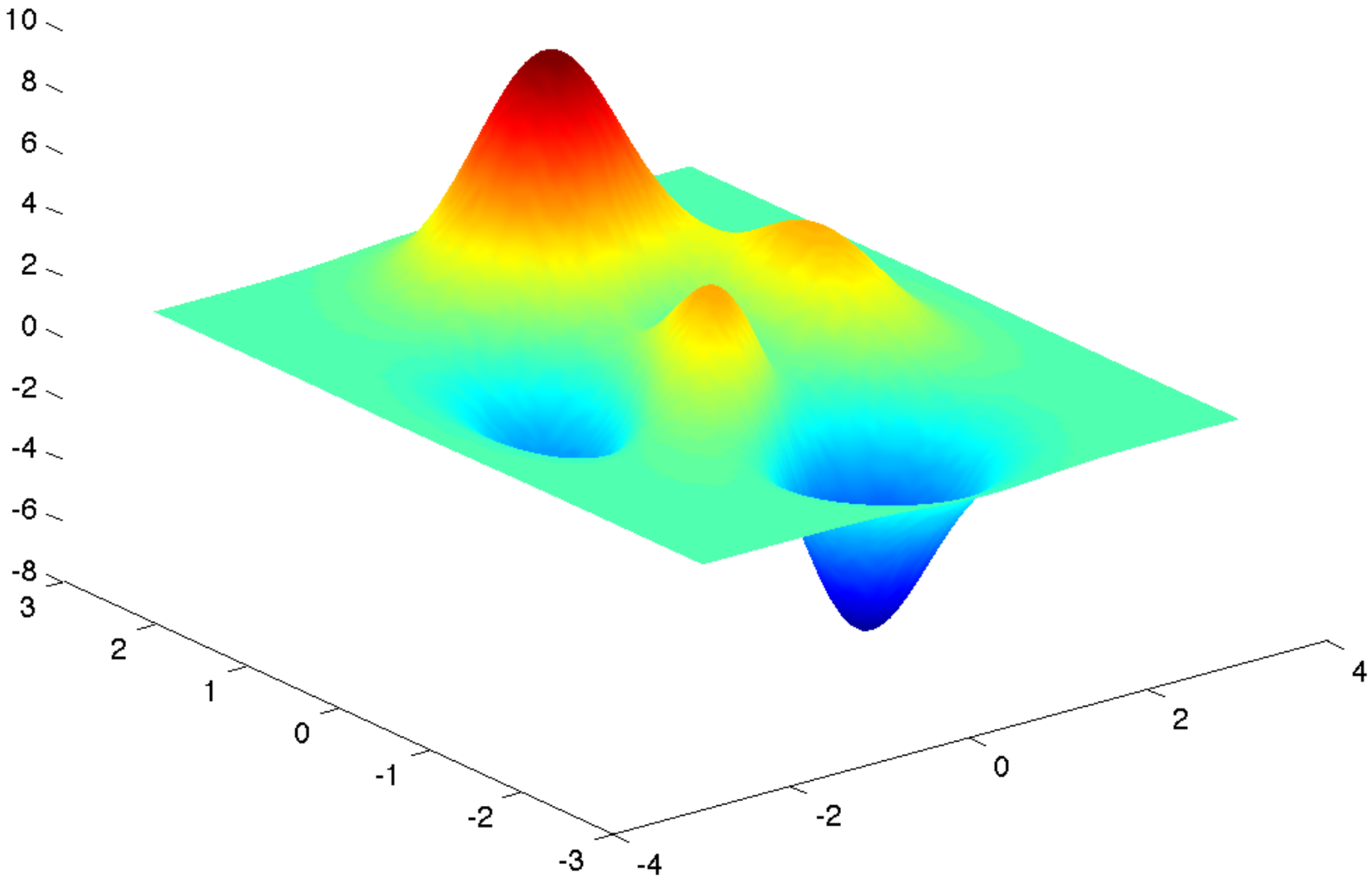
- Ideally the Hessian should be close to the identity matrix (esp. near the solution)

- Sometimes skewed scaling is desirable

Skewed Scaling

$$0 < \frac{\phi}{1000} < \frac{120^\circ}{1000}$$

Multi-Start

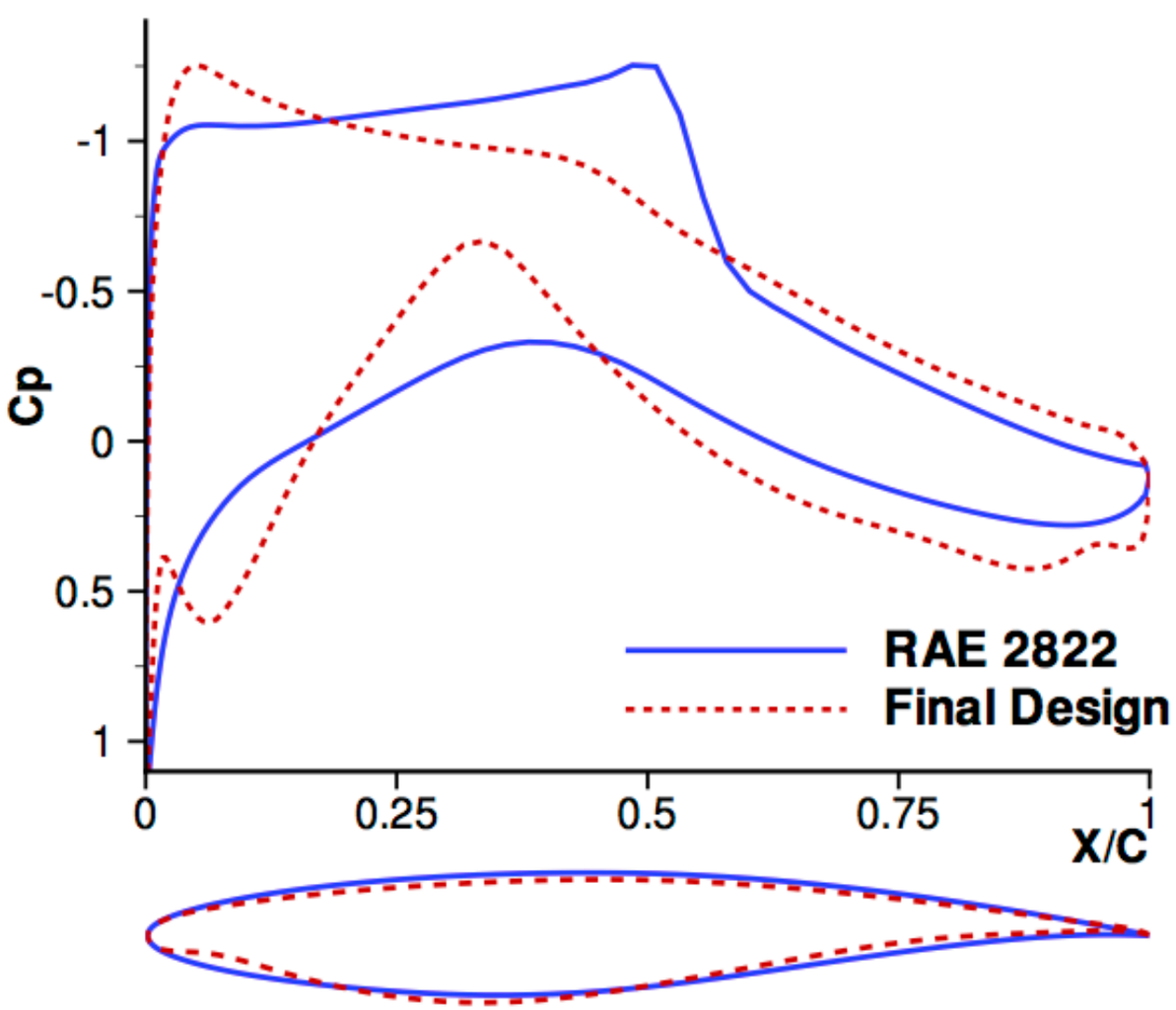


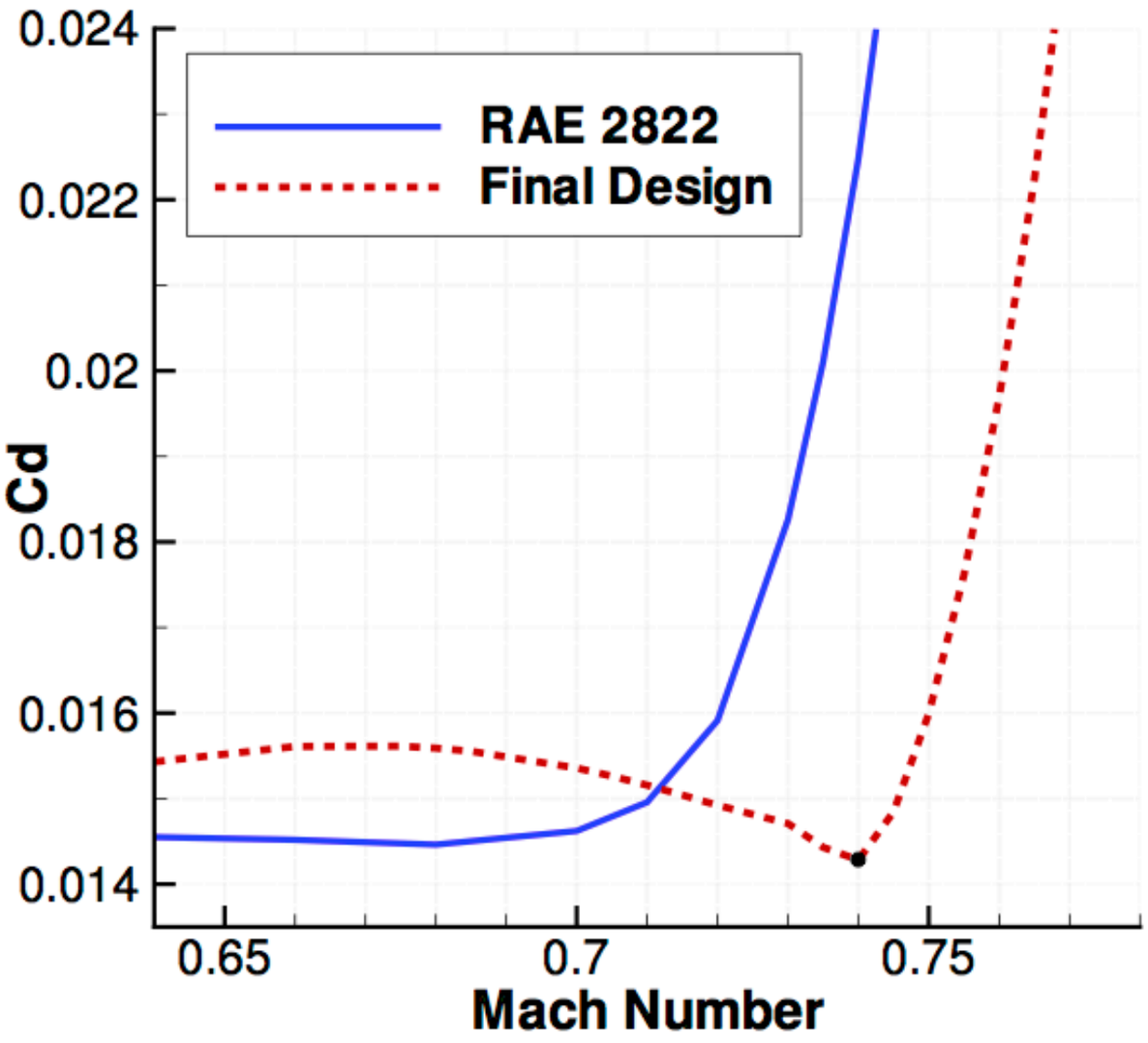
$$J_1^* = f(x_1^*)$$

$$J_2^* = f(x_2^*)$$

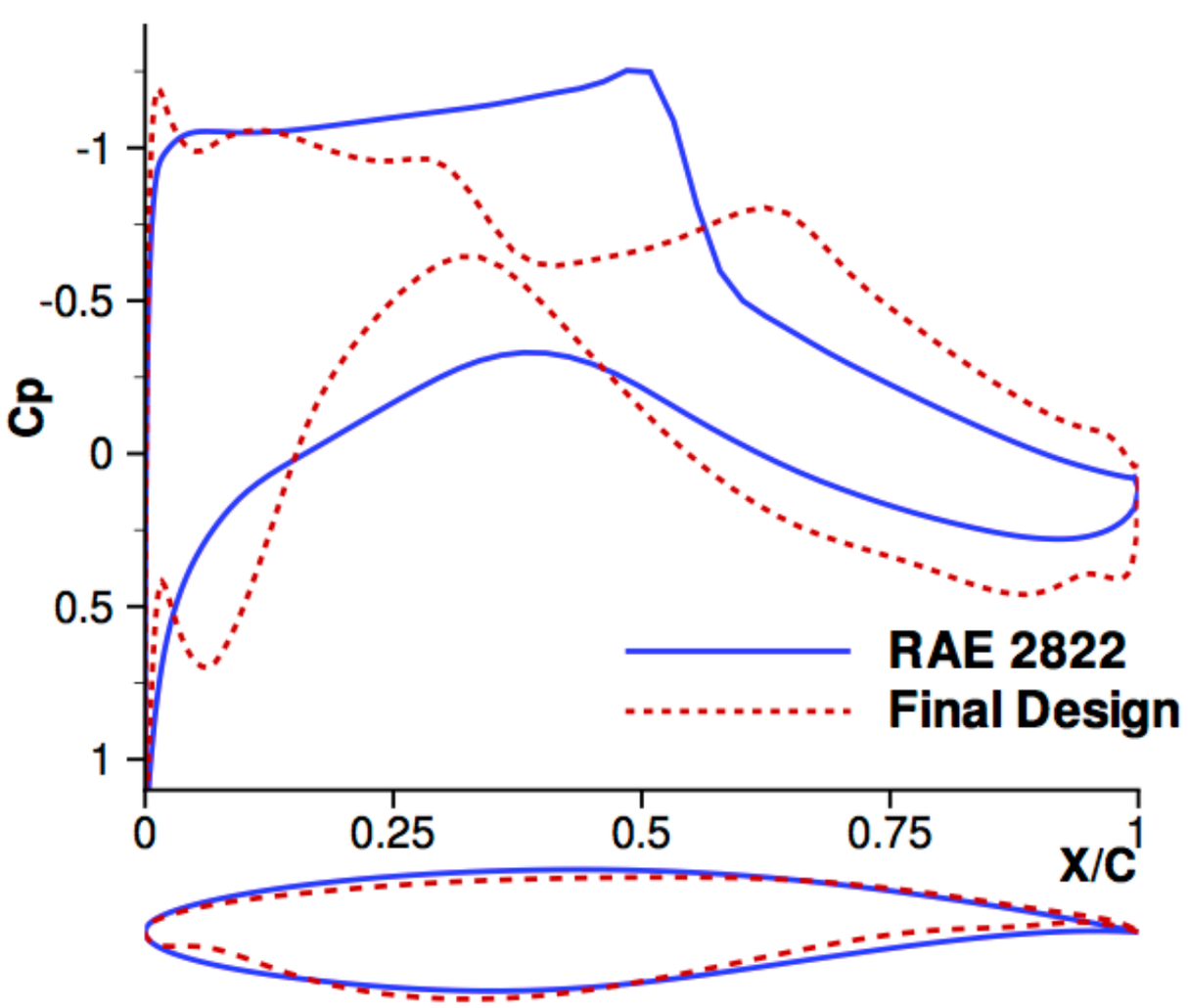
$$J_3^* = f(x_3^*)$$

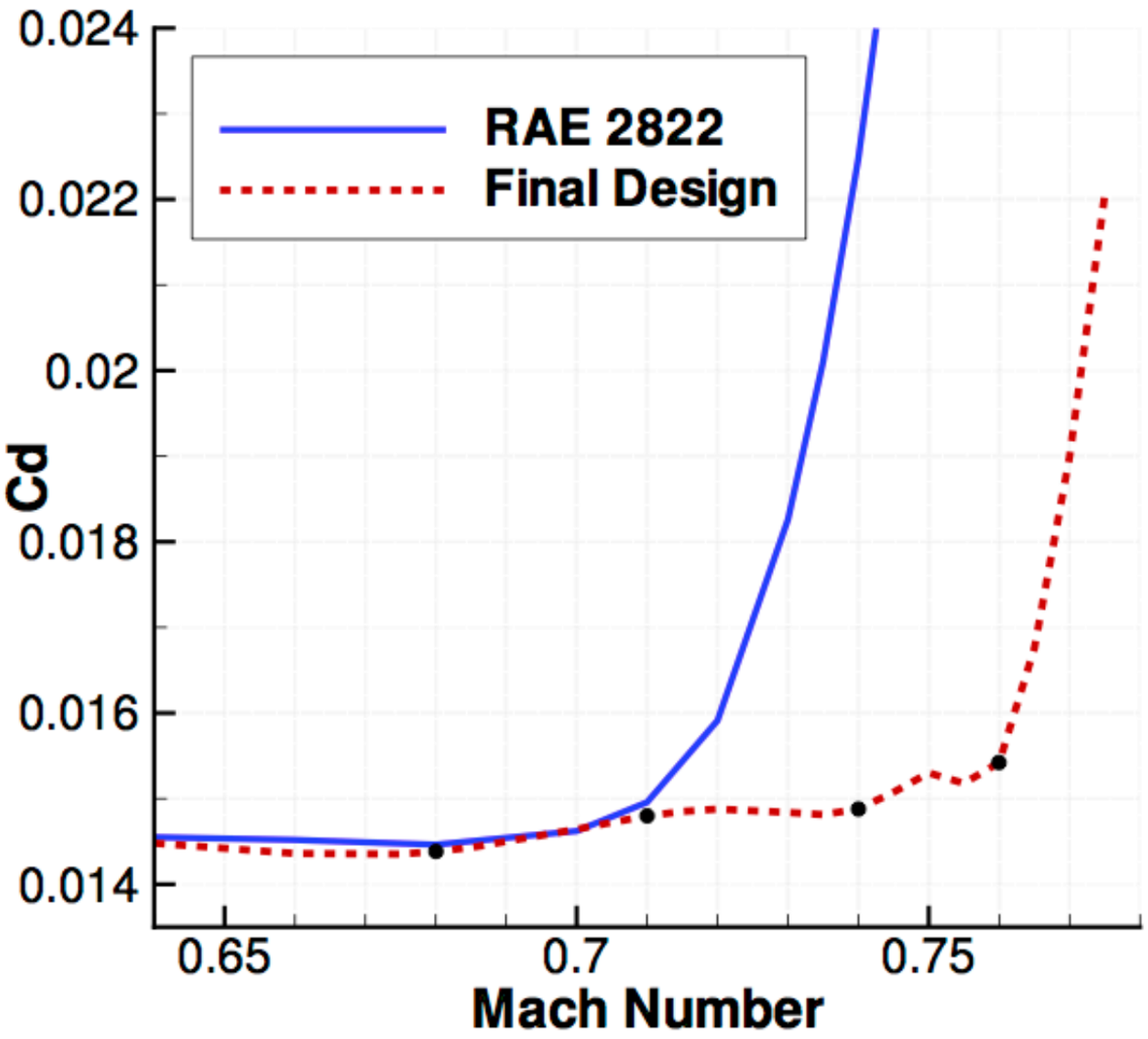
Multi-Point


Cp
-1
-0.5
0
0.5
1
0
0.25
0.5
0.75
1
X/C
RAE 2822
Final Design


Cd
0.024
0.022
0.02
0.018
0.016
0.014
0.65
0.7
0.75
Mach Number
RAE 2822
Final Design

Multi-Point

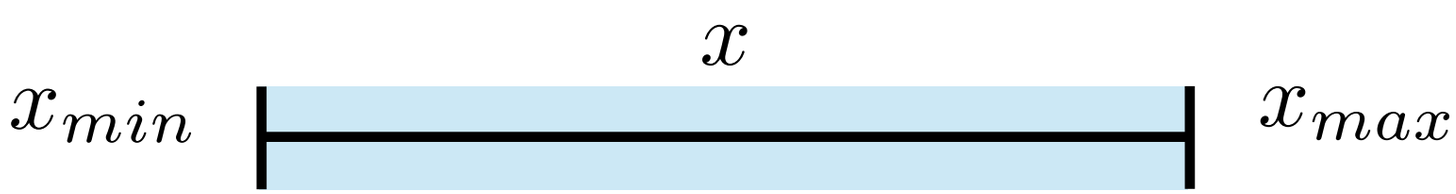

-1
-0.5
0
0.5
1
Cp
0
0.25
0.5
0.75
1
X/C
RAE 2822
Final Design

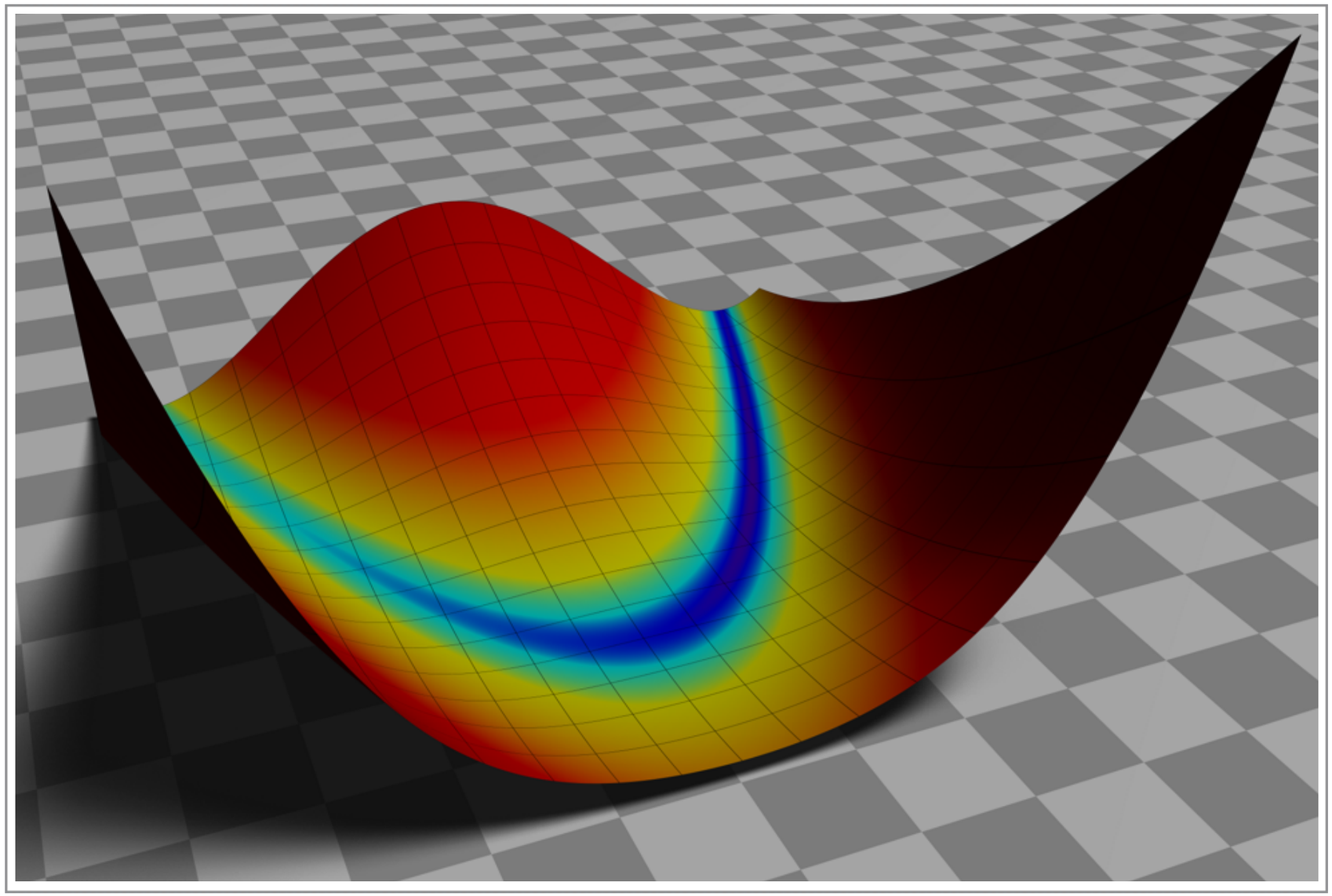

0.024
0.022
0.02
0.018
0.016
0.014
Cd
0.65
0.7
0.75
Mach Number
RAE 2822
Final Design

Bound Constraints

- unless the bound represents an actual constraint, keep bounds large enough so that they are not active at the solution

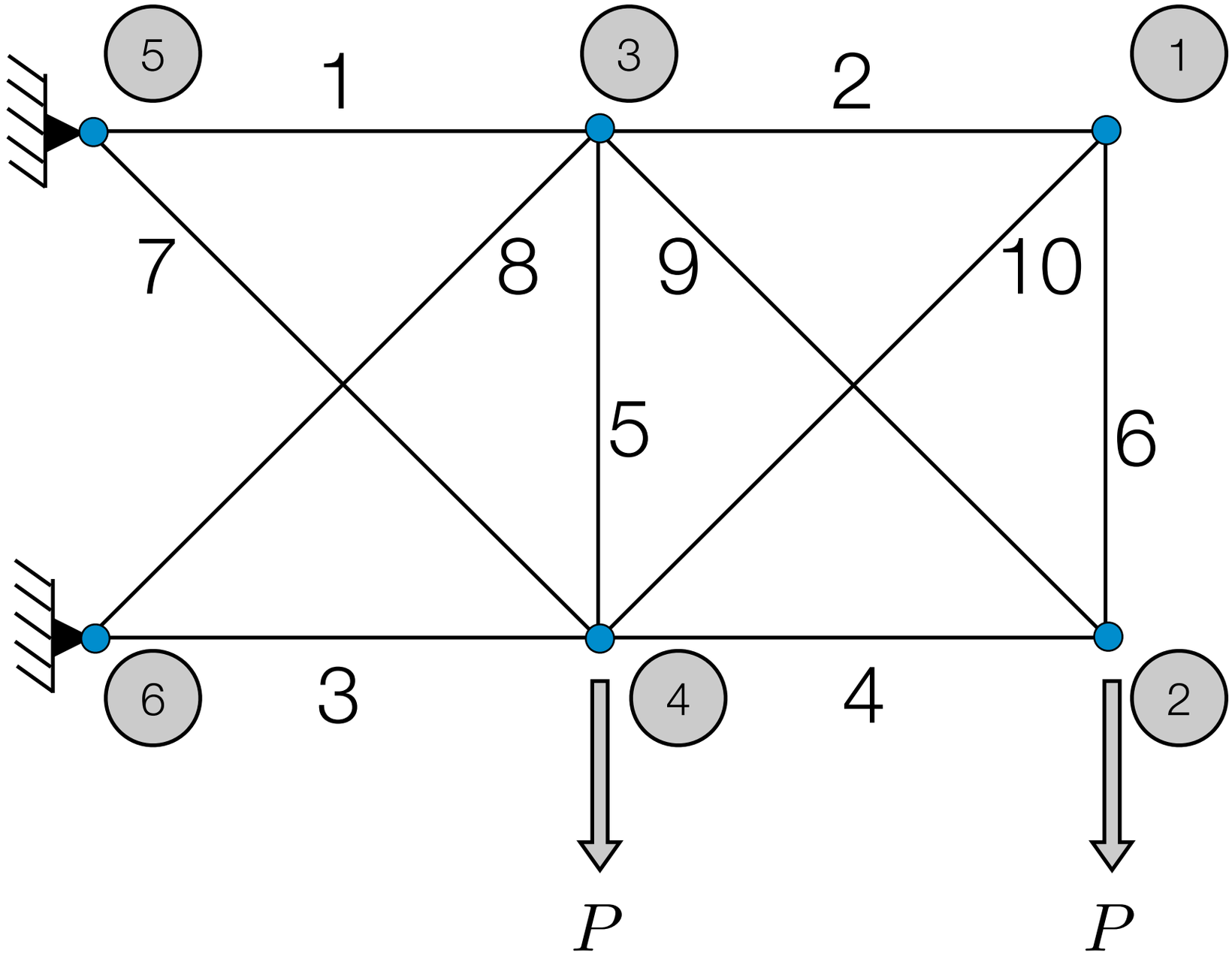
- don’t make them ridiculously large though -- depending on the algorithm the bounds may be used in scaling or choosing appropriate step sizes (if using a direct search method it’s better to keep the domain as small as you can)





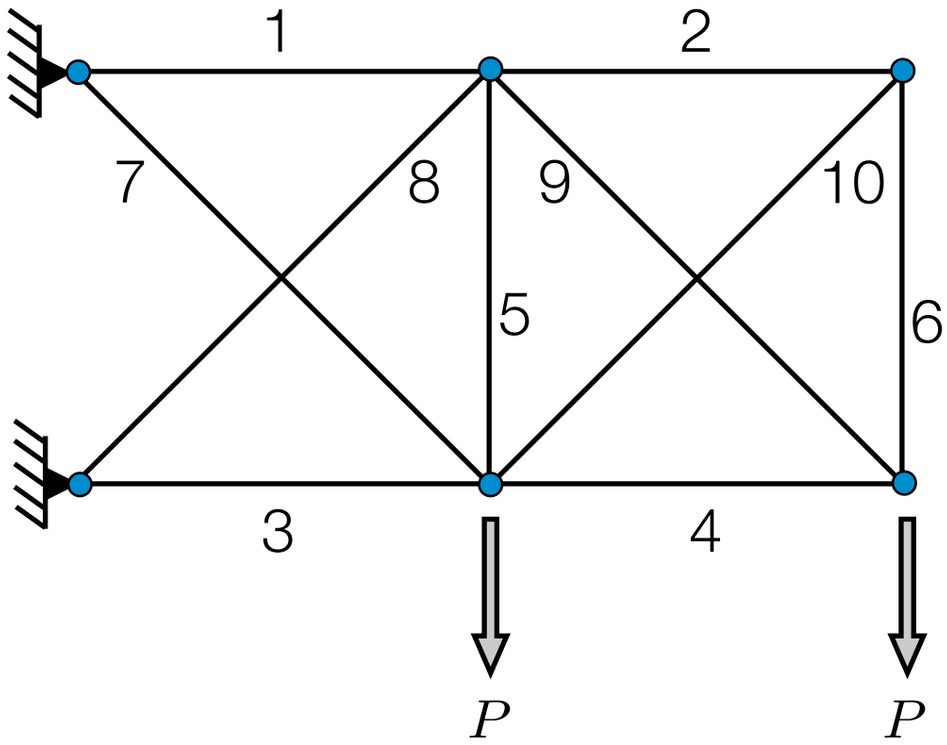
Truss Optimization Example

10-bar Truss



Optimization

$$
\begin{aligned}
min \quad & mass(x) \\
w.r.t \quad & x = [A_1, \ldots A_{10}] \\
s.t. \quad & \sigma(x)_j < 25,000 \text{ for } j \neq 9 \\
& \sigma(x)_j > -25,000 \text{ for } j \neq 9 \\
& \sigma(x)_9 < 75,000 \\
& \sigma(x)_9 > -75,000 \\
& A_i > 0.1
\end{aligned}
$$



```
% problem variables
nbar = 10;
syield = 25e3*ones(nbar, 1);
syield(9) = 75e3;

% -------- starting point and bounds --------------
x0 = 5*ones(1, nbar);
ub = 100*ones(1, nbar);
lb = 0.1*ones(1, nbar);
% -------------------------------------------------
```

```
% problem variables
nbar = 10;
syield = 25e3*ones(nbar, 1);
syield(9) = 75e3;

% -------- starting point and bounds -------------
x0 = 5*ones(1, nbar);
ub = 100*ones(1, nbar);
lb = 0.1*ones(1, nbar);
% ------------------------------------------------

% ------ linear constraint -----------------
A = [];
b = [];
Aeq = [];
beq = [];
% ----------------------------------------
```

```
% problem variables
nbar = 10;
syield = 25e3*ones(nbar, 1);
syield(9) = 75e3;

% -------- starting point and bounds --------------
x0 = 5*ones(1, nbar);
ub = 100*ones(1, nbar);
lb = 0.1*ones(1, nbar);
% -------------------------------------------------

% ------ linear constraint -----------------
A = [];
b = [];
Aeq = [];
beq = [];
% ------------------------------------
```

```
% ---------- Objective Function -----------------
function [J] = obj(x)
    [mass, stress] = truss(x);

    J = mass / 1e3;
end
% -------------------------------------------------
```

```
% problem variables
nbar = 10;
syield = 25e3*ones(nbar, 1);
syield(9) = 75e3;

% -------- starting point and bounds -------------
x0 = 5*ones(1, nbar);
ub = 100*ones(1, nbar);
lb = 0.1*ones(1, nbar);
% ------------------------------------------------

% ------ linear constraint -----------------
A = [];
b = [];
Aeq = [];
beq = [];
% ------------------------------------
```

```
% ---------- Objective Function -----------------
function [J] = obj(x)
    [mass, stress] = truss(x);

    J = mass / 1e3;
end
% ------------------------------------------------

% ------------- Constraints ------------------------
function [c, ceq] = con(x)
    [~, stress] = truss(x);
    stress = stress./syield;

    c = [stress-1, -1-stress];
    ceq = [];
end
% ------------------------------------------------
```

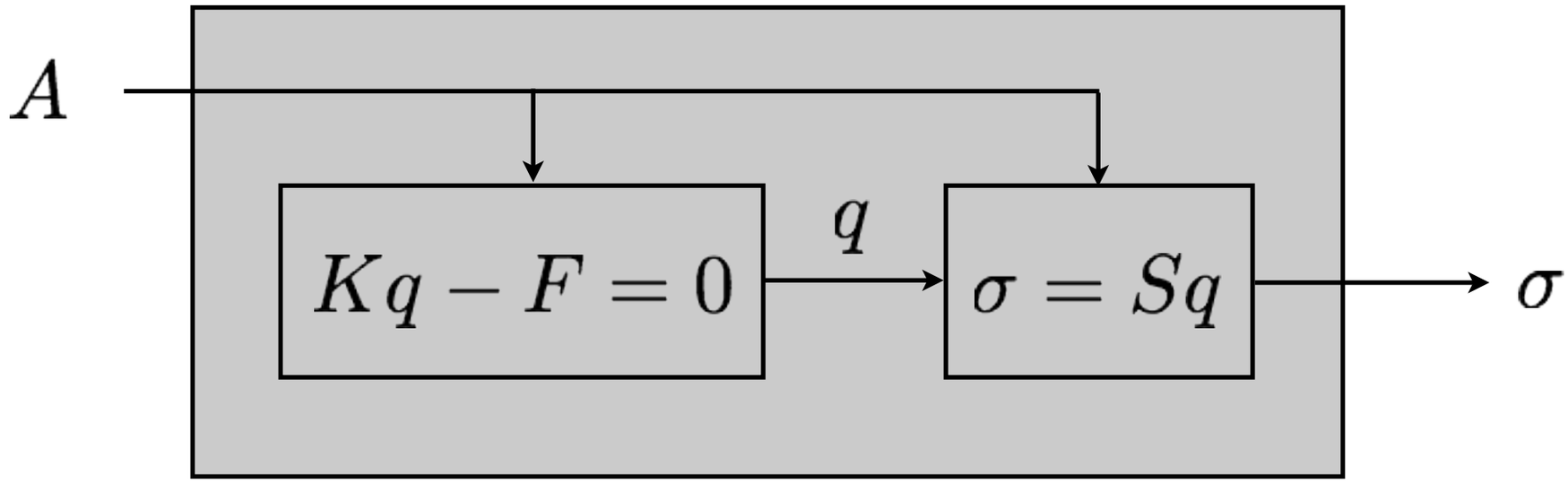
```
% ------------------ optimize --------------------
% 'interior-point', 'sqp', 'active-set', 'trust-region-reflective'
options = optimset(...
    'Algorithm', 'interior-point', ...
    'AlwaysHonorConstraints', 'bounds', ...
    'display', 'iter-detailed', ...
    'MaxIter', 500, ...
    'MaxFunEvals', 10000, ...
    'TolCon', 1e-6, ...
    'TolFun', 1e-6, ...
    'FinDiffType', 'forward', ...
    'GradObj', 'off', ...
    'GradConstr', 'off', ...
    'Diagnostics', 'on');
[xopt, fopt, exitflag] = fmincon(@obj, x0, A, b, Aeq, beq, lb, ub, @con, options);
% ------------------------------------------------
```

Gradients

$$m = \sum_i \rho_i A_i L_i$$

$$\frac{\partial m}{\partial A_i} = \rho_i L_i$$

Gradients



$$\frac{\partial R}{\partial y} = \frac{\partial (Kq - F)}{\partial q} = K \qquad \frac{\partial f}{\partial y} = \frac{\partial \sigma}{\partial q} = S$$

$$\frac{\partial R}{\partial x} = \frac{\partial (Kq - F)}{\partial A} = \frac{\partial K}{\partial A} q \qquad \frac{\partial f}{\partial x} = \frac{\partial \sigma}{\partial A} = 0$$

Gradients

direct method

$$\frac{df}{dx_i} = \frac{\partial f}{\partial x_i} - \frac{\partial f}{\partial y_j}\overbrace{\left[\frac{\partial \mathcal{R}_k}{\partial y_j}\right]^{-1}\frac{\partial \mathcal{R}_k}{\partial x_i}}^{\Psi_k}$$

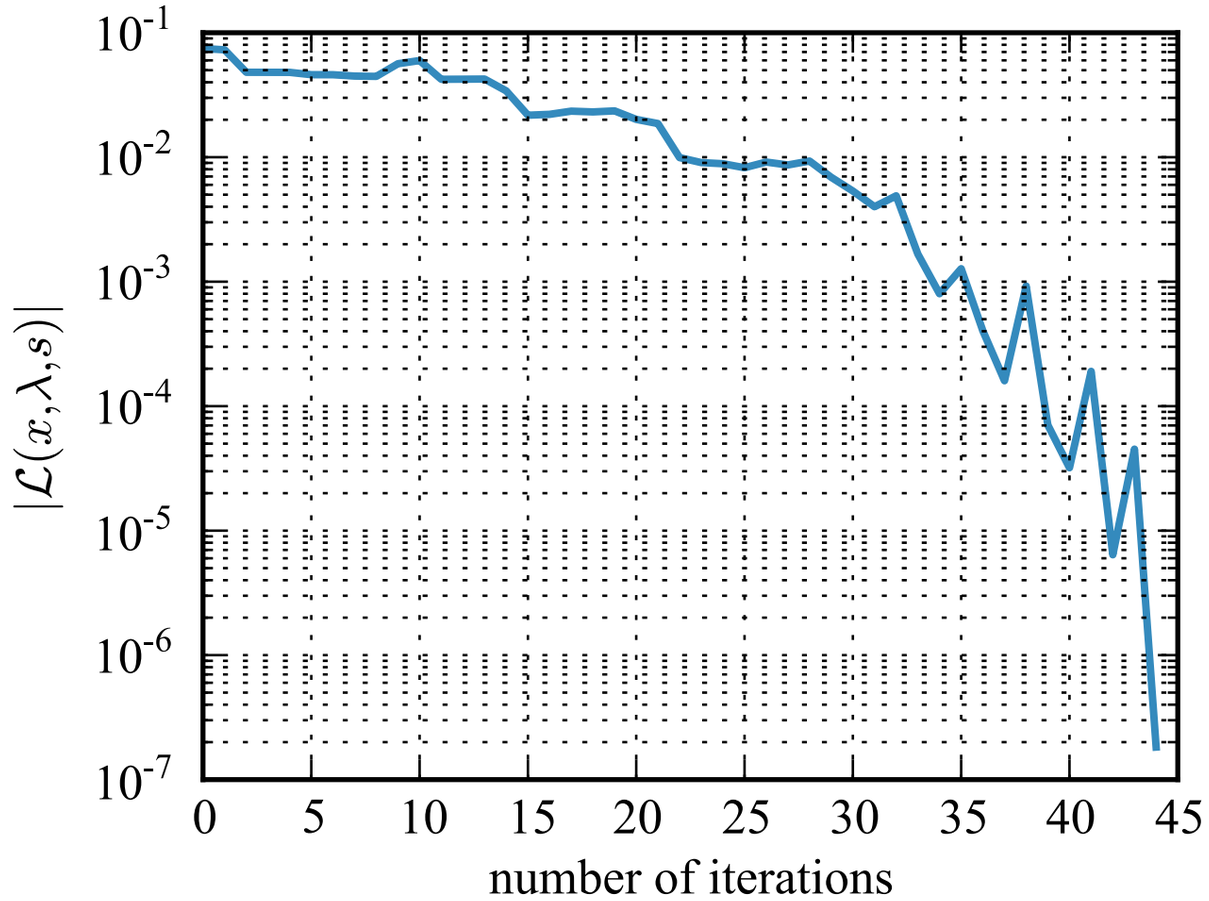
adjoint method

$$\frac{d\sigma}{dA_i} = -SK^{-1}\frac{\partial K}{\partial A_i}q$$

$$k = \frac{EA}{L}\begin{bmatrix} k_0 & -k_0 \\ -k_0 & k_0 \end{bmatrix}$$

```
gradient = True

% ---------- Objective Function -----------------
function [J, g] = obj(x)
    [mass, stress, dmass_dx, dstress_dx] = truss(x, gradient);

    J = mass / 1e3;
    g = dmass_dx / 1e3;
end
% -----------------------------------------------

% ------------- Constraints ---------------------
function [c, ceq, gc, geq] = con(x)
    [~, stress, ~, dstress_dx] = truss(x, gradient);

    stress = stress./syield;
    dstress_dx = dstress_dx./(syield*ones(1, nbar));

    c = [stress-1, -1-stress];
    ceq = [];
    gc = [dstress_dx', -dstress_dx'];
    geq = [];

end
% -----------------------------------------------
```

Convergence

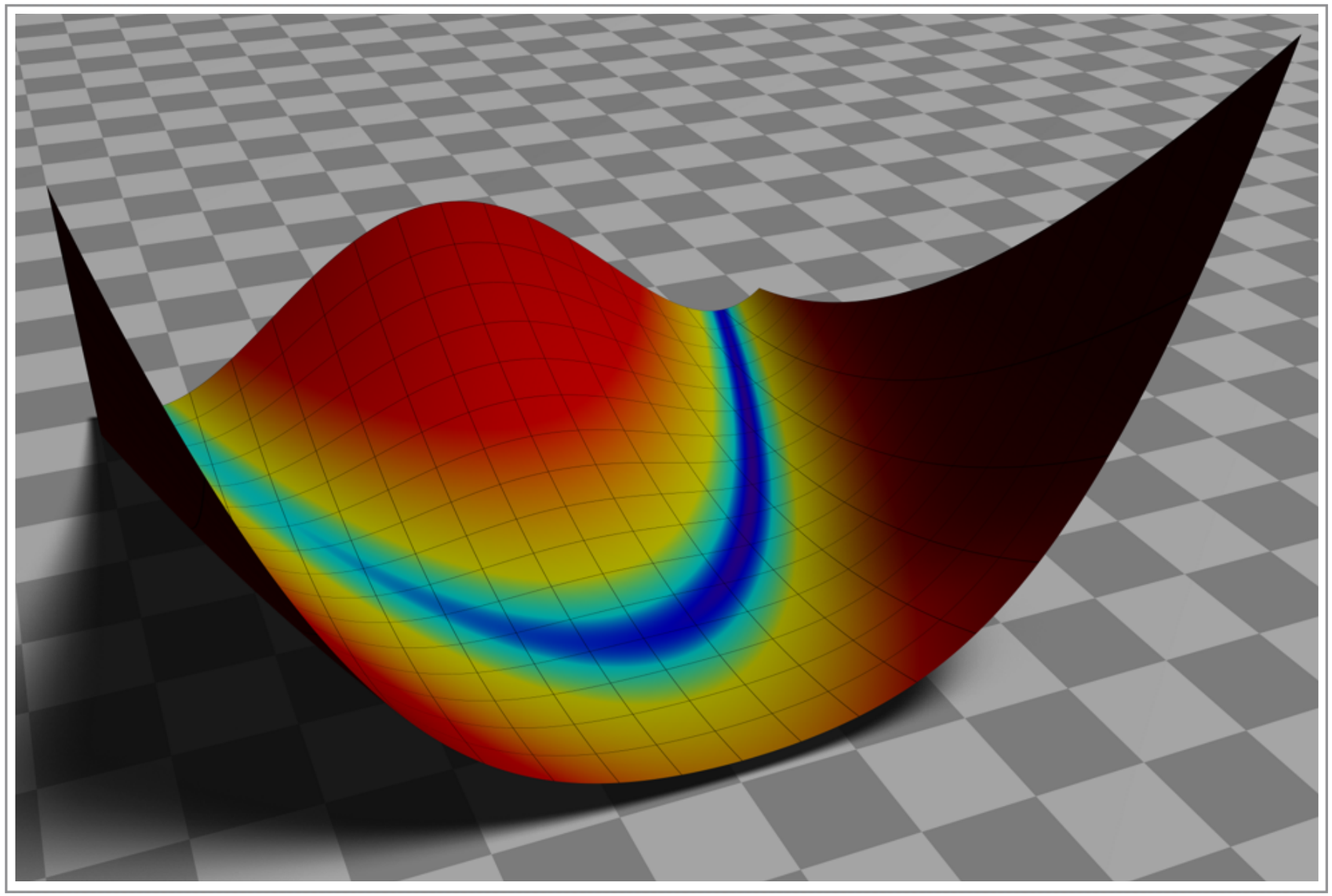


$$m^* = 1,497.6 \text{ lb}$$

$$A^* = [7.90, 0.10, 8.10, 3.90, 0.10, 0.10, 5.80, 5.52, 3.68, 0.14]$$

Comparison

	function calls	mass*
analytic gradients	52	1,497.6
FD gradient	502	1,497.6
GA	75,700	1,501.5



Wrap-up

Optimization Software

- MATLAB -- variety of algorithms, robust, easy to use
- KNITRO -- designed for large problems, can handle MIPs, interfaces to *many* programming languages
- SNOPT -- large-scale problems, good performance, particularly advantageous for sparse constraints
- CVX -- convex optimization (Matlab based)
- CPLX -- linear and integer programming
- IPOPT, SLSQP, CONMIN -- open source, generally only work well for simple problems

Optimization Frameworks

- DAKOTA -- only has open-source optimizers, but allows easy coupling to UQ algorithms
- pyOpt -- easy to use, interface to a wide variety of open-source and commercial optimizers
- AMPL -- A Mathematical Programming Language, supports many optimizers
- OpenMDAO -- not a “black box” approach, coupled derivatives, MDO architectures, HPC support

Optimization Textbooks

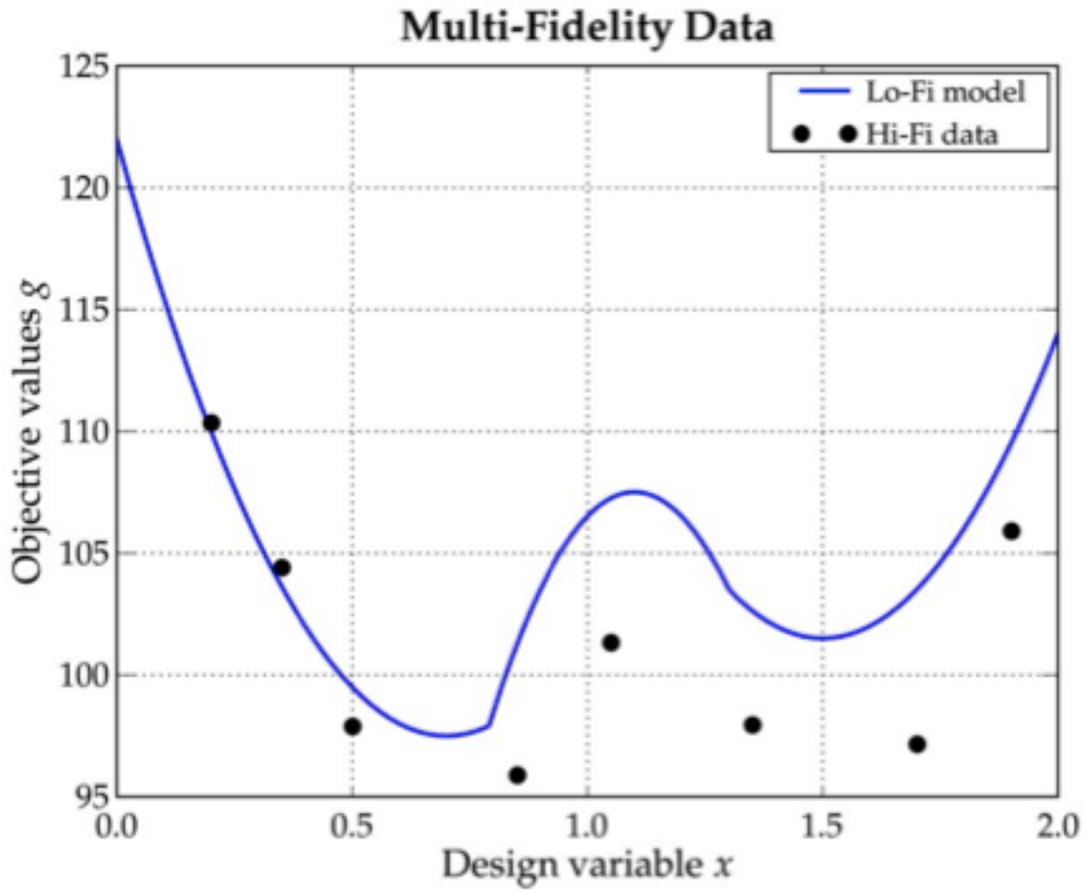
- MDO Notes -- Joaquim Martins. Currently in progress, search “AA222 Stanford” for an older version.

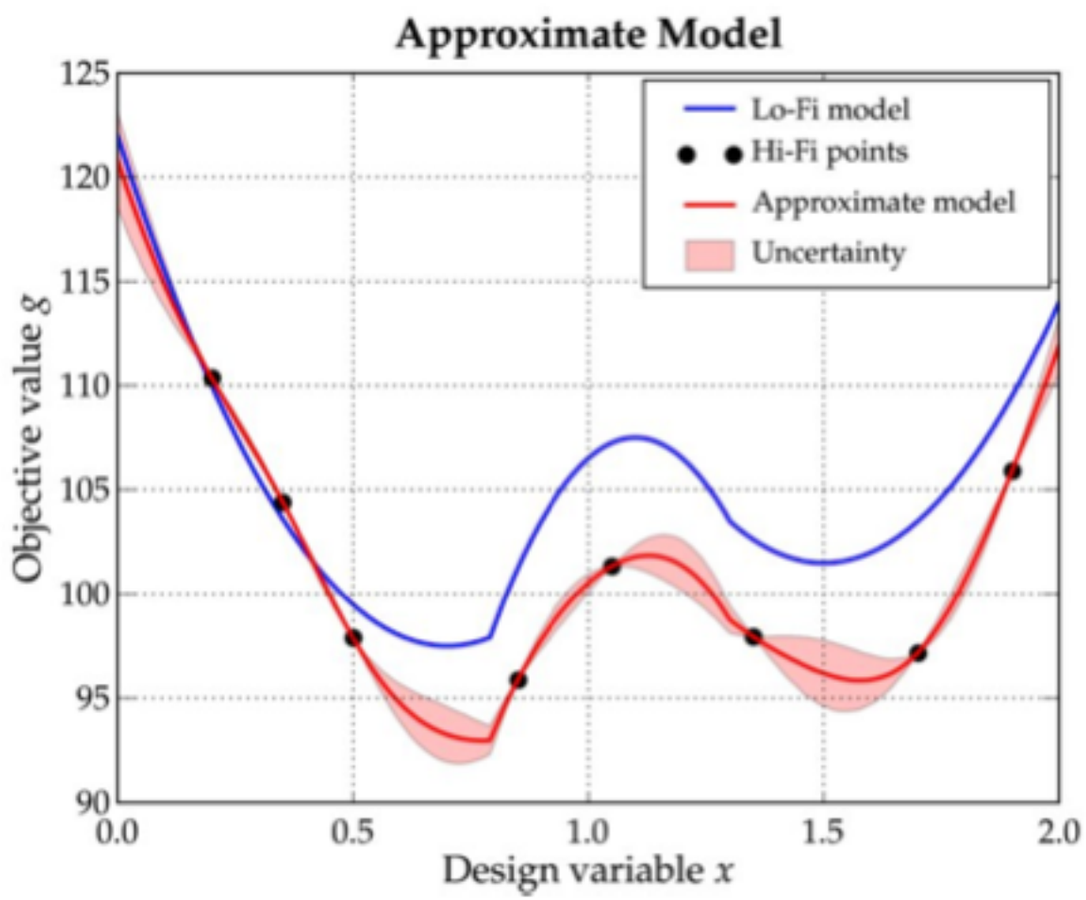
- Optimization Concepts and Applications in Engineering -- Ashok Belegundu.

- Convex Optimization -- Stephen Boyd. PDF version freely available.

- Practical Optimization -- Gill and Murray.

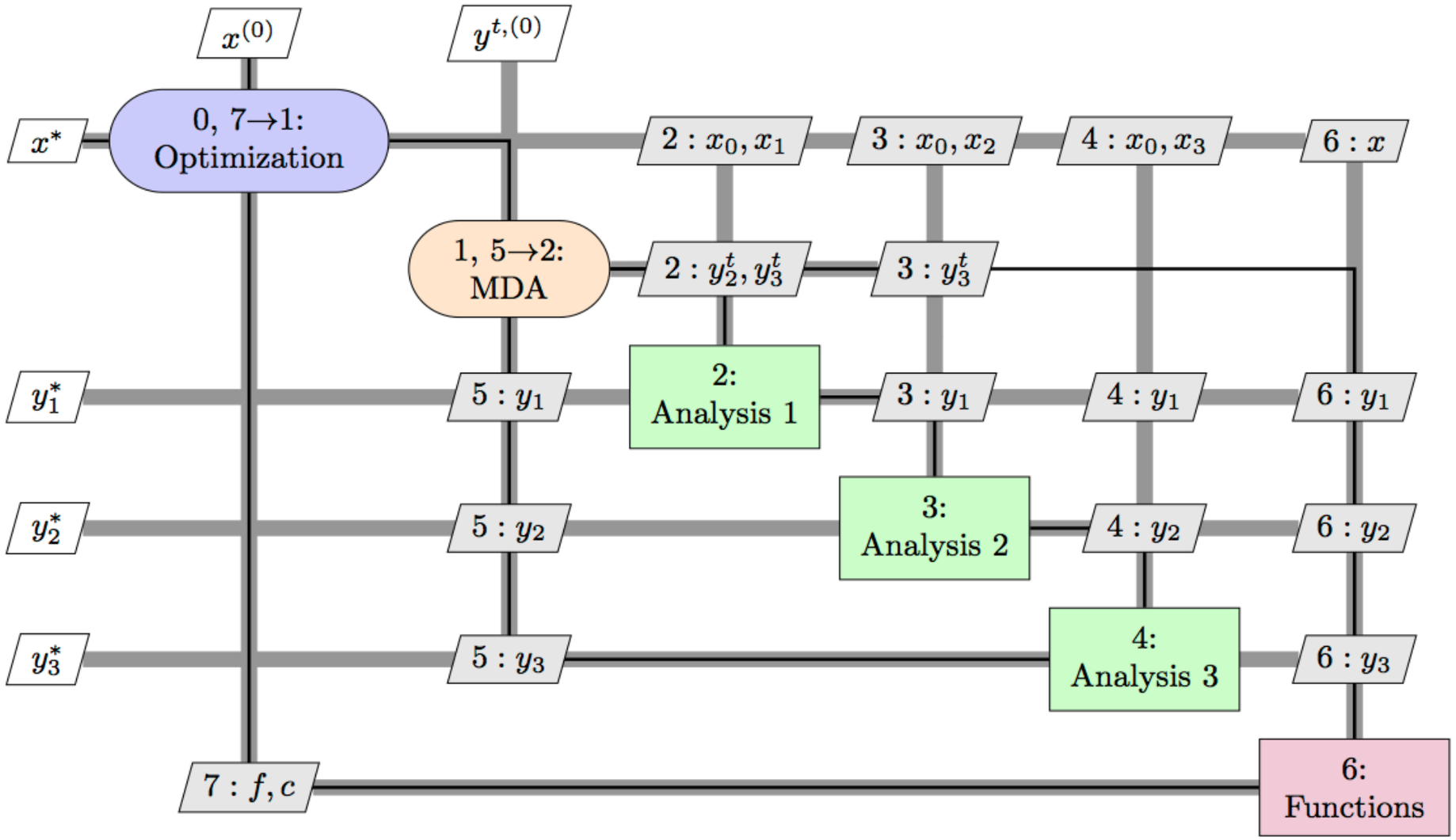
Advanced Topics





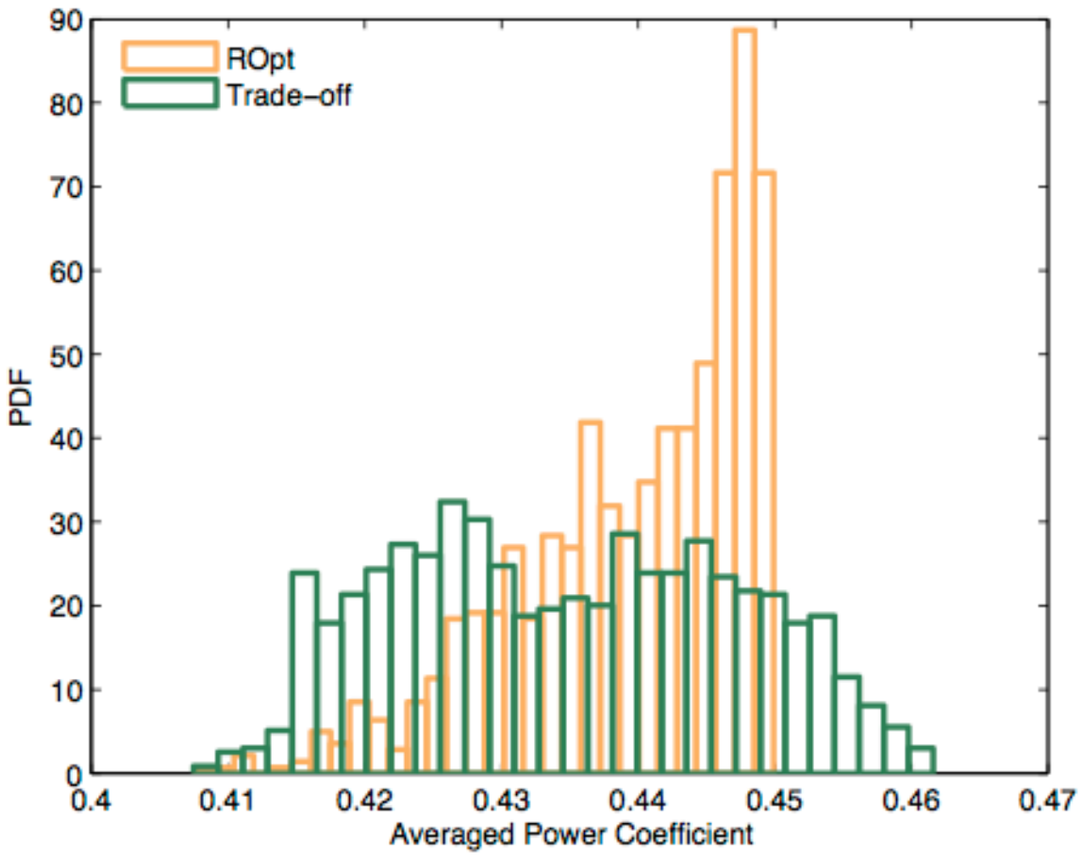
Multifidelity Optimization

Advanced Topics

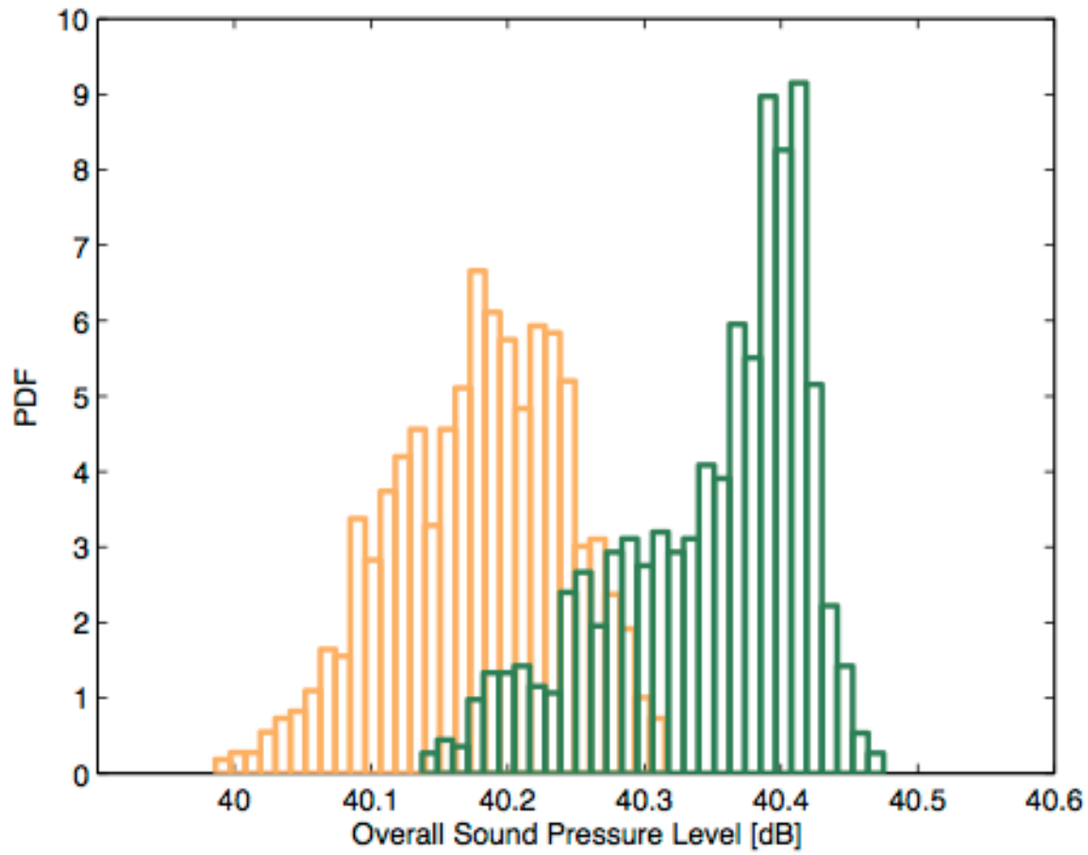


Multidisciplinary Optimization Architectures

Advanced Topics



(a) PDF of Averaged Power Coefficient



(b) PDF of Overall Sound Pressure Level

Optimization Under Uncertainty

Optimization Tips

- Put in time upfront before even beginning optimization to thoroughly explore the analysis. Ensure that the functions and constraints are well-behaved and C^1 continuous. Work on producing accurate gradients. 9 times out of 10, problems with the optimization arise from unexpected problems with the analysis.

- As a first cut, normalize all objectives, constraints, and design variables

- Start simple. Ideally start with a known optimal solution, then add complexity.

- Don't be afraid to experiment. Try different optimization algorithms, sensitivity analysis methods, scaling, starting points, etc.

- Iterate. Seek feedback both from an numerical perspective as well as a design perspective.

Questions