Optimization Without Derivatives

Stefan Wild

Argonne National Laboratory
Mathematics and Computer Science Division

January 2017

The Plan

Motivation

Black-box Optimization

Direct Search Methods

Model-Based Methods

Some Global Optimization

Simulation-Based Optimization and Structure

Nonlinear Least Squares

Least Squares and Partials

Constraints

Nonsmoothness

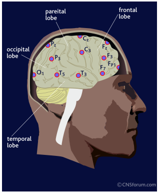
Simulation-Based Optimization

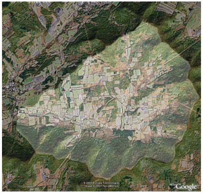
$$\min_{x \in \mathbb{R}^n} \{ f(x) = F[S(x)] : c(S(x)) \leq 0 \}$$

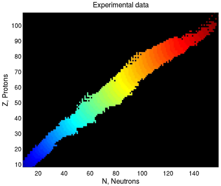
- S (numerical) simulation output, often "noisy" (even when deterministic)
- Derivatives $\nabla_x S$ often unavailable or prohibitively expensive to obtain/approximate directly
- S can contribute to objective and/or constraints
- Single evaluation of S could take seconds/minutes/hours/days

Evaluation is a bottleneck for optimization

Functions of complex numerical simulations arise everywhere







Computing is Responsible for Pervasiveness of Simulations in Sci&Eng

Argonne's Blue Gene/P (2008: 163,840 cores)

Currently 67th fastest in the world

Argonne's Blue Gene/Q (2012: 786,432 cores)

Currently 5th fastest in the world

- Parallel/multi-core environments increasingly common
  - Small clusters/multi-core desktops/multi-core laptops pervasive
  - Leadership class machines increasingly parallel
- Simulations (the "forward problem") become faster/more realistic/more complex

Improvements from Algorithms Can Trump Those From Hardware

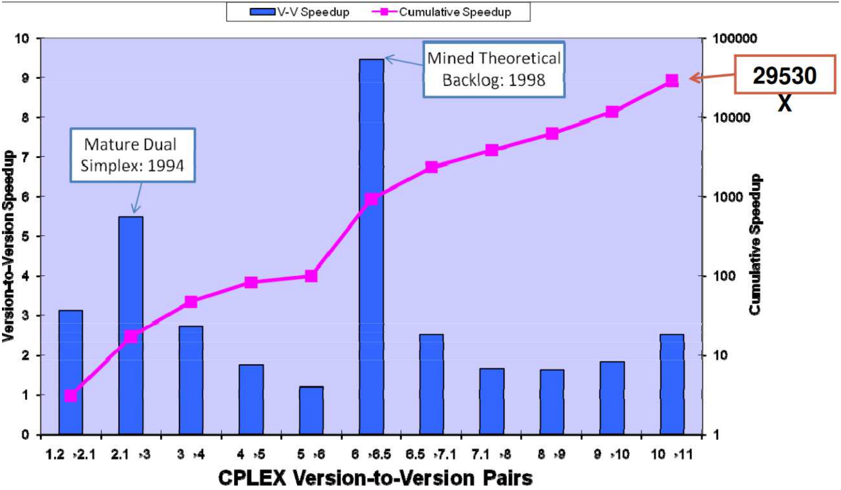
Martin Grötschel's production planning benchmark problem (a MIP):

1988 solve time using current computers and LP algorithms: 82 years

2003 solve time using current computers and LP algorithms: 1 minute

- Speed up of 43,000,000X
  10^3X from processor improvements
  10^4X additional from algorithmic improvements

Improvements from Algorithms Can Trump Those From Hardware



1991 (v1.2) to 2007 (v11.0): Moore's Law transistor speedup: ≈ 256X

[Slide from Bixby (CPLEX/GUROBI)]: Solves 1,852 MIPs

Derivative-Free Optimization

“Some derivatives are unavailable for optimization purposes”

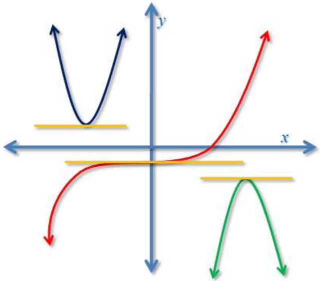
Derivative-Free Optimization

"Some derivatives are unavailable for optimization purposes"

The Challenge: Optimization is tightly coupled with derivatives

Typical optimality (no noise, smooth functions)

$$\nabla_x f(x_*) + \lambda^T \nabla_x c_E(x_*) = 0, c_E(x_*) = 0$$



(sub)gradients $\nabla_x f$, $\nabla_x c$ enable:

- ◇ Faster feasibility
- ◇ Faster convergence
  - ♦ Guaranteed descent
  - ♦ Approximation of nonlinearities
- ◇ Better termination
  - ♦ Measure of criticality $\|\nabla_x f\|$ or $\|\mathcal{P}_\Omega(\nabla_x f)\|$
- ◇ Sensitivity analysis
  - ♦ Correlations, standard errors, UQ, . . .

Ways to Get Derivatives

(assuming they exist)

Handcoding (HC)

"Army of students/programmers"

- ? Prone to errors/conditioning
- ? Intractable as number of ops increases

Algorithmic/Automatic Differentiation (AD)

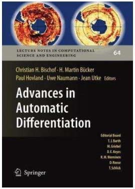
"Exact* derivatives!"

- ? No black boxes allowed
- ? Not always automatic/cheap/well-conditioned

Finite Differences (FD)

"Nonintrusive"

- ? Sensitive to stepsize choice/noise
- ? Expense grows with n



LECTURE NOTES IN COMPUTATIONAL SCIENCE AND ENGINEERING 64

Christian H. Bischof · H. Martin Bücker
Paul Hovland · Uwe Naumann · Jean Utke Editors

Advances in Automatic Differentiation

Editorial Board
T. J. Barth
M. Griebel
D. E. Keyes
R. M. Nieminen
D. Roose
T. Schlick

Springer

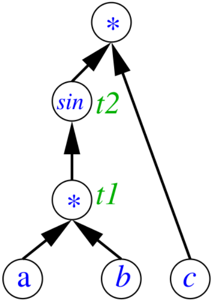
...then apply derivative-based method (that handles inexact derivatives)

Algorithmic Differentiation

Computational Graph

- $y = sin(a * b) * c$
- Forward and reverse modes
- AD tool provides code for your derivatives

You should write codes and formulate problems with AD in mind!



Many tools (see `www.autodiff.org`):

F/C	`Tapenade`, `Rapsodia`
C/C++	`ADOL-C`, `ADIC`
Matlab	`ADiMat`, `INTLAB`
	Also done in `AMPL` and `GAMS`!

Caveats

You will pay a price for not having derivatives

We will focus primarily on:

- **Minimization** (sorry, economists, you'll need to stand on your head)
- **Continuous** domains
- **Unconstrained** problems
  - Simulation-based constraints bring additional challenges
- Underlying **smooth** behavior (whatever that means)
- **Deterministic** objective (or function of psuedorandom number generators)
- Algorithms with convergence guarantees to **local** minima
  - GAs and other heuristics often require far too many evaluations

→ Can address these in detail during lunch, dinner, TA/office hours

Caveats

You will pay a price for not having derivatives

We will focus primarily on:

- **Minimization** (sorry, economists, you'll need to stand on your head)
- **Continuous** domains
- **Unconstrained** problems
  - Simulation-based constraints bring additional challenges
- Underlying **smooth** behavior (whatever that means)
- **Deterministic** objective (or function of psuedorandom number generators)
- Algorithms with convergence guarantees to **local** minima
  - GAs and other heuristics often require far too many evaluations

→ Can address these in detail during lunch, dinner, TA/office hours

Please: Interrupt, Ask Questions

Global Optimization, $\min_{x\in\Omega} f(x)$

Careful:

- Global convergence: Convergence (to a local solution/stationary point) from anywhere in Ω
- Convergence to a global minimizer: Obtain x_* with $f(x_*) \leq f(x) \, \forall x \in \Omega$

Global Optimization, $\min_{x \in \Omega} f(x)$

Careful:

- Global convergence: Convergence (to a local solution/stationary point) from anywhere in Ω
- Convergence to a global minimizer: Obtain x_* with $f(x_*) \leq f(x) \forall x \in \Omega$

Anyone selling you global solutions when derivatives are unavailable:

either assumes more about your problem (e.g., convex f)

or expects you to wait forever

Törn and Žilinskas: An algorithm converges to the global minimum for any continuous f if and only if the sequence of points visited by the algorithm is dense in Ω.

or cannot be trusted

Global Optimization, $\min_{x\in\Omega} f(x)$

Careful:

- Global convergence: Convergence (to a local solution/stationary point) from anywhere in Ω
- Convergence to a global minimizer: Obtain x_* with $f(x_*) \leq f(x) \, \forall x \in \Omega$

Anyone selling you global solutions when derivatives are unavailable:

either assumes more about your problem (e.g., convex f)

or expects you to wait forever

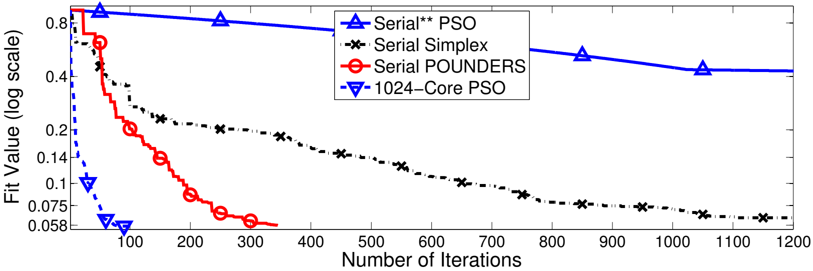
Törn and Žilinskas: An algorithm converges to the global minimum for any continuous f if and only if the sequence of points visited by the algorithm is dense in Ω.

or cannot be trusted

Instead:

- Rapidly find good local solutions and/or be robust to poor solutions
- Consider multistart approaches and/or structure of multimodality

(One Reason) Why We Won't Be Talking About Heuristics

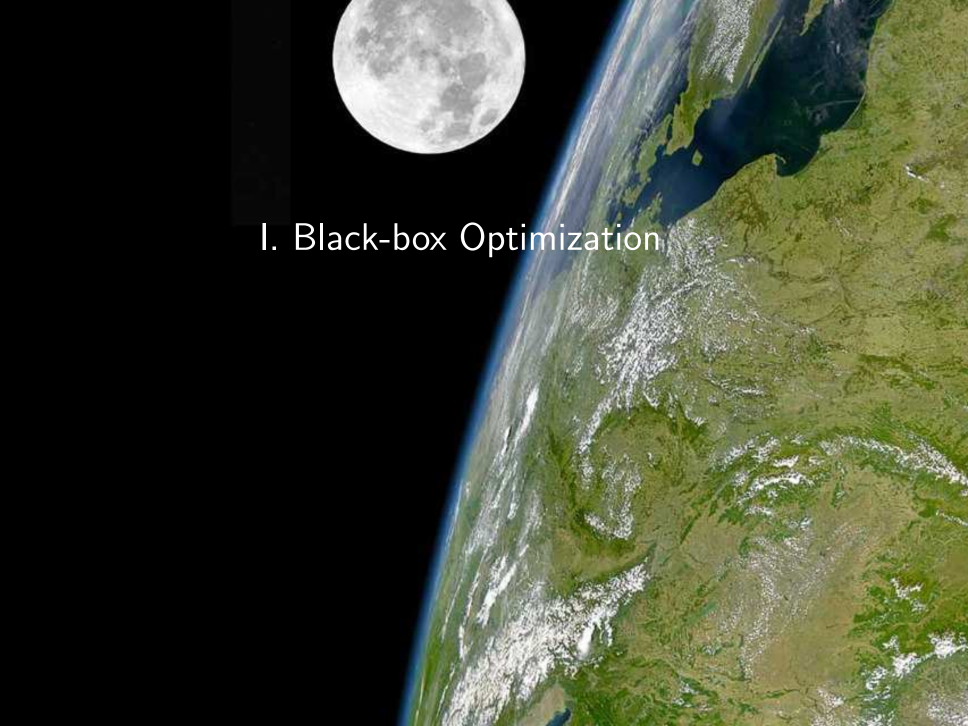


- ◇ Heuristics often "embarrassingly/naturally parallel"; PSO= particle swarm method
  - ♦ Typically through stochastic sampling/evolution
  - ♦ 1024 function evaluations per iteration
- ◇ Simplex is Nelder-Mead; POUNDERS is model-based trust-region algorithm
  - ♦ one function evaluation per iteration

→ Is this an effective use of resources?

→ How many cores would have sufficed?

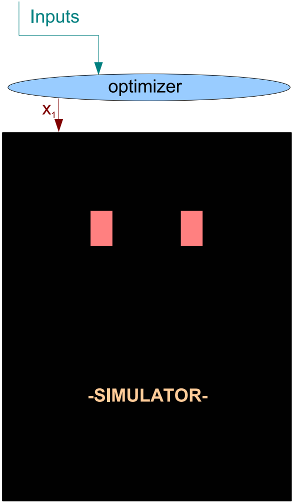
I. Black-box Optimization



Black-box Optimization Problems

NB- My “black box” is different from Ken’s “black box”

Black-box Optimization Problems



NB- My "black box" is different from Ken's "black box"

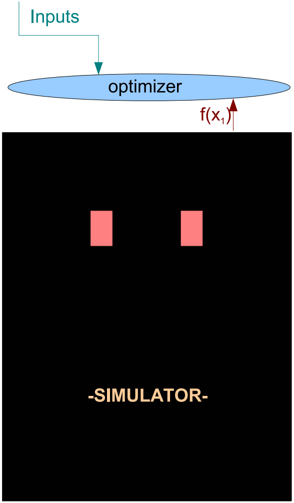
Only knowledge about f is obtained by sampling

- $f = S$ a black box (running some executable-only code or performing an experiment in the lab)
- Only give a single output (no derivatives $\nabla_x S(x)$)

Good solutions guaranteed in the limit, but:

- Usually have computational budget (due to scheduling, finances, deadlines)
- Limited number of evaluations

Black-box Optimization Problems



NB- My "black box" is different from Ken's "black box"

Only knowledge about f is obtained by sampling

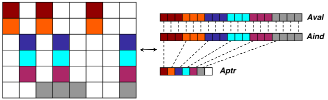
- $f = S$ a black box (running some executable-only code or performing an experiment in the lab)
- Only give a single output (no derivatives $\nabla_x S(x)$)

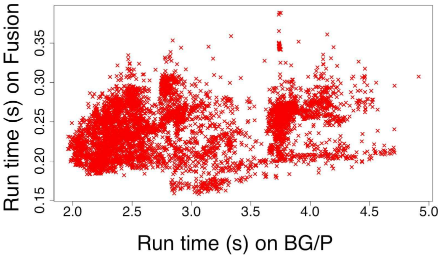
Good solutions guaranteed in the limit, but:

- Usually have computational budget (due to scheduling, finances, deadlines)
- Limited number of evaluations

A Black Box: Automating Empirical Performance Tuning

Given semantically equivalent codes $\mathcal{C}_1, \mathcal{C}_2, \ldots$, minimize run time subject to energy consumption



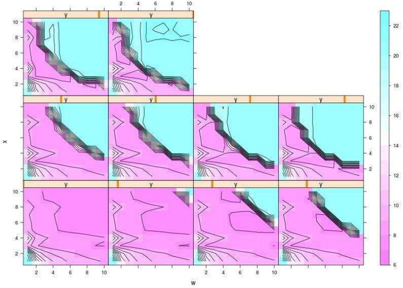


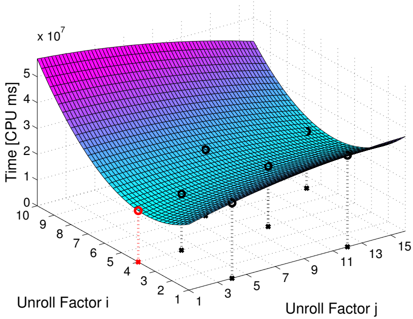
$$\min\left\{f(x) : (x_{\mathcal{C}}, x_{\mathcal{I}}, x_{\mathcal{B}}) \in \Omega_{\mathcal{C}} \times \Omega_{\mathcal{I}} \times \Omega_{\mathcal{B}}\right\}$$

- x multidimensional parameterization (internal tolerances, unroll/tiling factors, compiler flags, compiler type, ...)
- Ω search domain (feasible transformation, no errors)
- f quantifiable performance objective (requires a run)

Optimization for Automatic Tuning of HPC Codes

Evaluation of f requires:
transforming source, compilation, (repeated?) execution, checking for correctness





Challenges:

- Evaluating $f(\Omega)$ prohibitively expensive (e.g., 10^{19} discrete decisions)
- f noisy
- Discrete x unrelaxable
- $\nabla_x f$ unavailable/nonexistent
- Many distinct/local solutions

Black-box Algorithms

Solve general problems $\min\{f(x) : x \in \mathbb{R}^n\}$:

- Only require function values (no $\nabla f(x)$)
- Don't rely on finite-difference approximations to $\nabla f(x)$
- Seek greedy and rapid decrease of function value
- Have asymptotic convergence guarantees
- Assume parallel resources are used within function evaluation

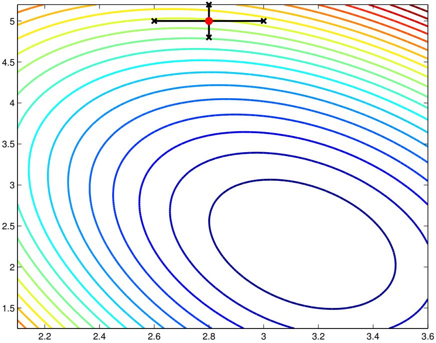
Main styles of DFO algorithms

- Randomized methods (this afternoon!)
- Direct search methods (pattern search, Nelder-Mead, . . .)
- Model-based methods (quadratics, radial basis functions, . . .)

Pattern Search Methods

Choose a set of directions (pattern or mesh) $\mathcal{D}^k$

Ex.- ± coordinate directions ($2n$ directions)



Basic iteration ($k \geq 0$):

- Evaluate $f(x^k + \Delta_k d^j)$, $j = 1, \ldots, |\mathcal{D}^k|$
- If $\left[f(x^k + \Delta_k d^j) < f(x^k)\right]$,
  move to $x^{k+1} = x^k + \Delta_k d^j$

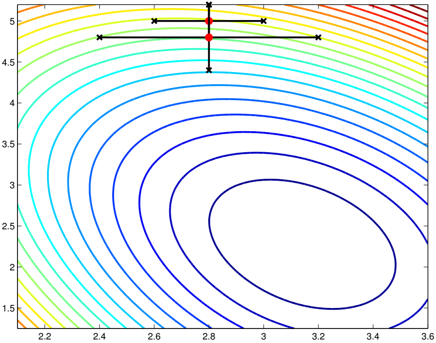
  Otherwise shrink Δ_k
- Update $\mathcal{D}^k$

This is an indicator function, does not say anything about the magnitude of f values, just the ordering

Pattern Search Methods

Choose a set of directions (pattern or mesh) $\mathcal{D}^k$

Ex.- $\pm$ coordinate directions ($2n$ directions)



Basic iteration ($k \geq 0$):

- ◇ Evaluate $f(x^k + \Delta_k d^j)$, $j = 1, \ldots, |\mathcal{D}^k|$
- ◇ If $\left[f(x^k + \Delta_k d^j) < f(x^k)\right]$,
  move to $x^{k+1} = x^k + \Delta_k d^j$

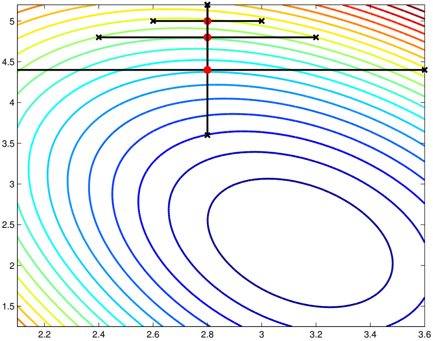
  Otherwise shrink Δ_k
- ◇ Update $\mathcal{D}^k$

This is an indicator function, does not say anything about the magnitude of f values, just the ordering

Pattern Search Methods

Choose a set of directions (pattern or mesh) $\mathcal{D}^k$

Ex.- ± coordinate directions ($2n$ directions)



Basic iteration ($k \geq 0$):

- Evaluate $f(x^k + \Delta_k d^j)$, $j = 1, \ldots, |\mathcal{D}^k|$
- If $\left[f(x^k + \Delta_k d^j) < f(x^k)\right]$,

  move to $x^{k+1} = x^k + \Delta_k d^j$

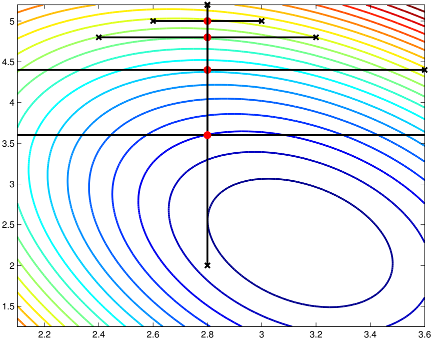
  Otherwise shrink Δ_k
- Update $\mathcal{D}^k$

This is an indicator function, does not say anything about the magnitude of f values, just the ordering

Pattern Search Methods

Choose a set of directions (pattern or mesh) $\mathcal{D}^k$

Ex.- ± coordinate directions ($2n$ directions)



Basic iteration ($k \geq 0$):

- ◇ Evaluate $f(x^k + \Delta_k d^j)$, $j = 1, \ldots, |\mathcal{D}^k|$
- ◇ If $\left[f(x^k + \Delta_k d^j) < f(x^k) \right]$,
  move to $x^{k+1} = x^k + \Delta_k d^j$

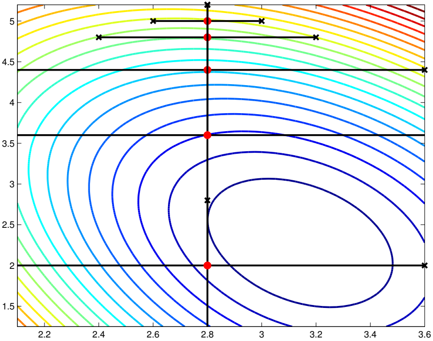
  Otherwise shrink Δ_k
- ◇ Update $\mathcal{D}^k$

This is an indicator function, does not say anything about the magnitude of f values, just the ordering

Pattern Search Methods

Choose a set of directions (pattern or mesh) $\mathcal{D}^k$

Ex.- ± coordinate directions ($2n$ directions)



Basic iteration ($k \geq 0$):

- Evaluate $f(x^k + \Delta_k d^j)$, $j = 1, \ldots, |\mathcal{D}^k|$
- If $\left[f(x^k + \Delta_k d^j) < f(x^k)\right]$,
  move to $x^{k+1} = x^k + \Delta_k d^j$

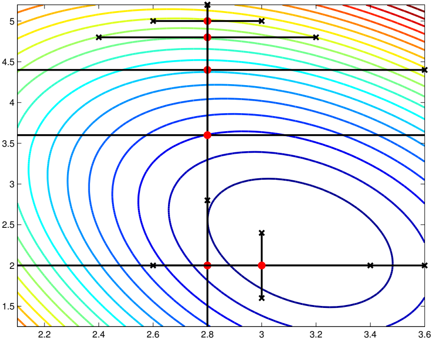
  Otherwise shrink Δ_k
- Update $\mathcal{D}^k$

This is an indicator function, does not say anything about the magnitude of f values, just the ordering

Pattern Search Methods

Choose a set of directions (pattern or mesh) $\mathcal{D}^k$

Ex.- $\pm$ coordinate directions ($2n$ directions)



Basic iteration ($k \geq 0$):

- ◇ Evaluate $f(x^k + \Delta_k d^j)$, $j = 1, \ldots, |\mathcal{D}^k|$
- ◇ If $\left[f(x^k + \Delta_k d^j) < f(x^k)\right]$,

  move to $x^{k+1} = x^k + \Delta_k d^j$

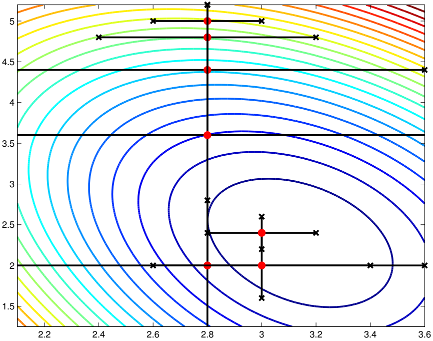
  Otherwise shrink Δ_k
- ◇ Update $\mathcal{D}^k$

This is an indicator function, does not say anything about the magnitude of f values, just the ordering

Pattern Search Methods

Choose a set of directions (pattern or mesh) $\mathcal{D}^k$

Ex.- ± coordinate directions ($2n$ directions)



Basic iteration ($k \geq 0$):

- Evaluate $f(x^k + \Delta_k d^j)$, $j = 1, \ldots, |\mathcal{D}^k|$
- If $\left[f(x^k + \Delta_k d^j) < f(x^k)\right]$,

  move to $x^{k+1} = x^k + \Delta_k d^j$

  Otherwise shrink Δ_k
- Update $\mathcal{D}^k$

This is an indicator function, does not say anything about the magnitude of f values, just the ordering

→ Lends itself well to doing concurrent function evaluations

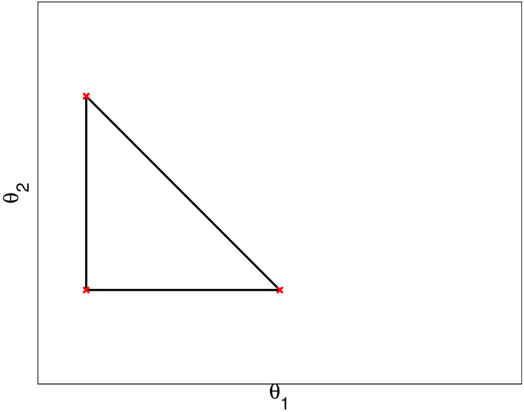
→ See also mesh-adaptive direct search methods (e.g., NOMAD)

→ Can establish convergence for nonsmooth f

The Nelder-Mead Method [1965]

Basic iteration ($k \geq 0$):

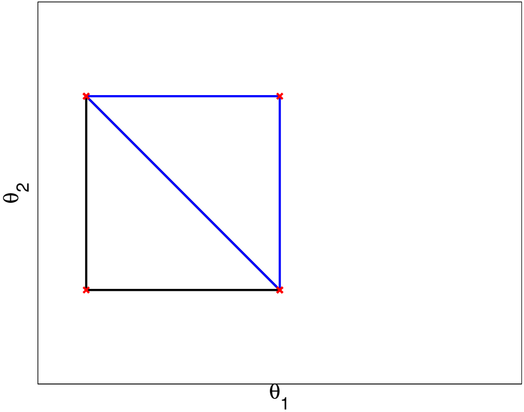
- Evaluate f on the $n+1$ vertices of the simplex $x^k + \Delta_k \mathcal{S}^{(k)}$
- Reflect worst vertex about the best face
- Shrink, contract, or expand $\Delta_k \mathcal{S}^{(k)}$



The Nelder-Mead Method [1965]

Basic iteration ($k \geq 0$):

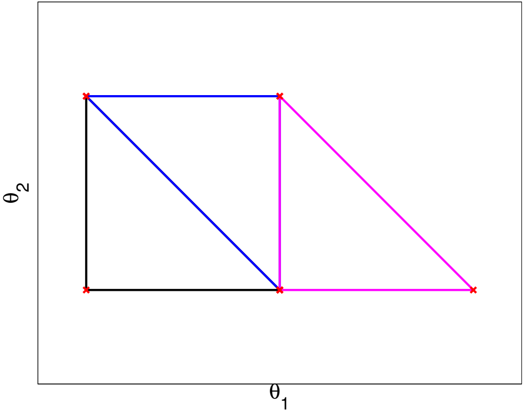
- Evaluate f on the $n+1$ vertices of the simplex $x^k + \Delta_k \mathcal{S}^{(k)}$
- Reflect worst vertex about the best face
- Shrink, contract, or expand $\Delta_k \mathcal{S}^{(k)}$



The Nelder-Mead Method [1965]

Basic iteration ($k \geq 0$):

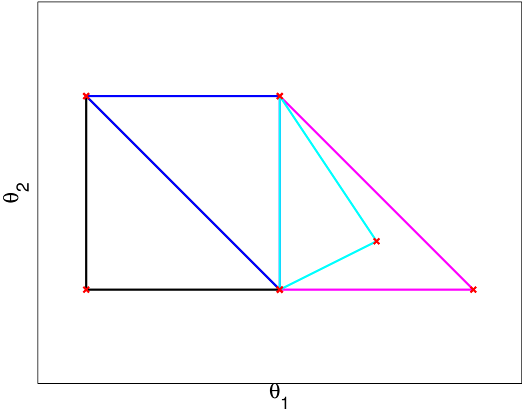
- Evaluate f on the $n+1$ vertices of the simplex $x^k + \Delta_k \mathcal{S}^{(k)}$
- Reflect worst vertex about the best face
- Shrink, contract, or expand $\Delta_k \mathcal{S}^{(k)}$



The Nelder-Mead Method [1965]

Basic iteration ($k \geq 0$):

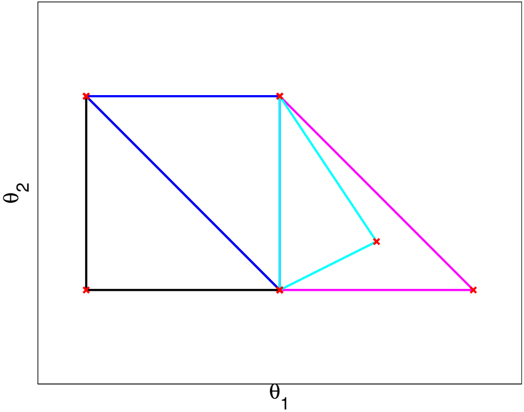
- Evaluate f on the $n+1$ vertices of the simplex $x^k + \Delta_k \mathcal{S}^{(k)}$
- Reflect worst vertex about the best face
- Shrink, contract, or expand $\Delta_k \mathcal{S}^{(k)}$



The Nelder-Mead Method [1965]

Basic iteration ($k \geq 0$):

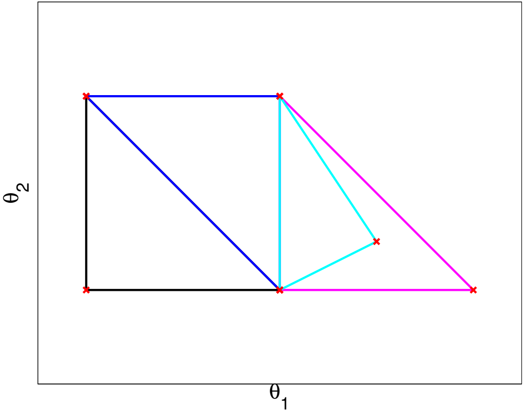
- Evaluate f on the $n+1$ vertices of the simplex $x^k + \Delta_k \mathcal{S}^{(k)}$
- Reflect worst vertex about the best face
- Shrink, contract, or expand $\Delta_k \mathcal{S}^{(k)}$



The Nelder-Mead Method [1965]

Basic iteration ($k \geq 0$):

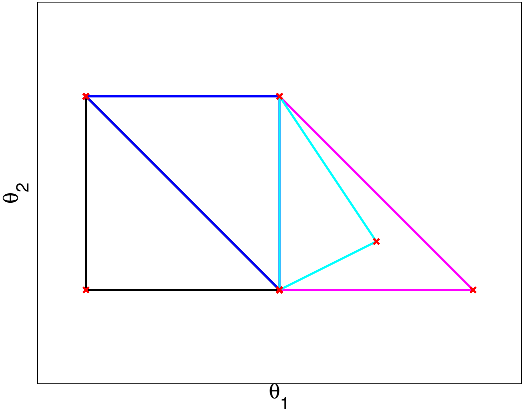
- Evaluate f on the $n+1$ vertices of the simplex $x^k + \Delta_k \mathcal{S}^{(k)}$
- Reflect worst vertex about the best face
- Shrink, contract, or expand $\Delta_k \mathcal{S}^{(k)}$



The Nelder-Mead Method [1965]

Basic iteration ($k \geq 0$):

- Evaluate f on the $n+1$ vertices of the simplex $x^k + \Delta_k \mathcal{S}^{(k)}$
- Reflect worst vertex about the best face
- Shrink, contract, or expand $\Delta_k \mathcal{S}^{(k)}$



Only the order of the function values matter:
$f(\hat{x}) = 1,\ f(\tilde{x}) = 1.0001$ is the same as $f(\hat{x}) = 1,\ f(\tilde{x}) = 10000$.

→ A very popular (due to "Numerical Recipes"), robust first choice

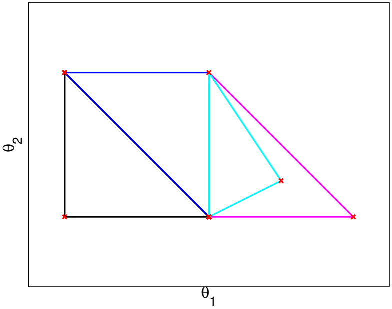
What Are We Missing?

These methods will (eventually) find a local solution
Overview: $\rightarrow$ [Kolda, Lewis, Torczon, SIREV 2003]

Each evaluation of f is expensive (valuable)

N-M:

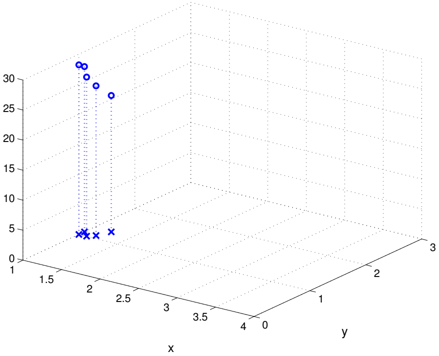
1. Only remembers the last $n+1$ evaluations
2. Neglects the magnitudes of the function values (order only)
3. Doesn't take into account the special (LS) problem structure



$\rightarrow$ This is the reason many direct search methods use a search phase on top of the usual poll phase

Making the Most of Little Information on f

- f is expensive ⇒ can afford to make better use of points
- Overhead of the optimization routine is negligible relative to the cost of evaluating the simulation.



Bank of data, $\{x^i, f(x^i)\}_{i=1}^{k}$:

= Points (& function values) evaluated so far

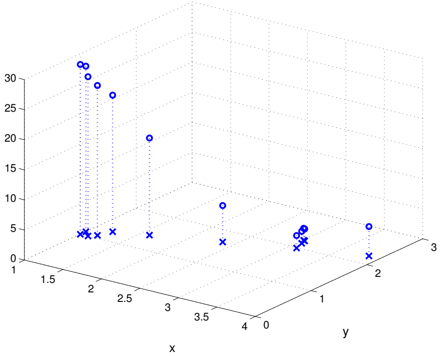
= Everything known about f

Goal:

- Make use of growing Bank as optimization progresses

Making the Most of Little Information on f

- f is expensive ⇒ can afford to make better use of points
- Overhead of the optimization routine is negligible relative to the cost of evaluating the simulation.



Bank of data, $\{x^i, f(x^i)\}_{i=1}^{k}$:

= Points (& function values) evaluated so far

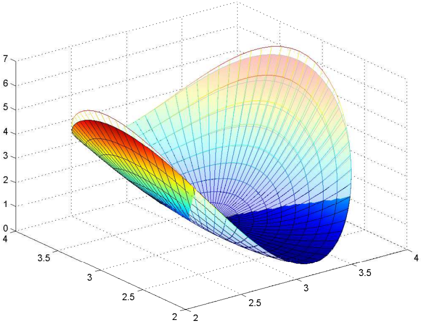
= Everything known about f

Goal:

- Make use of growing Bank as optimization progresses

Trust-Region Methods Use Models Instead of f

To reduce the number of expensive f evaluations
$\rightarrow$ Replace difficult optimization problem $\min f(x)$
with a much simpler one $\min \{m(x) : x \in \mathcal{B}\}$

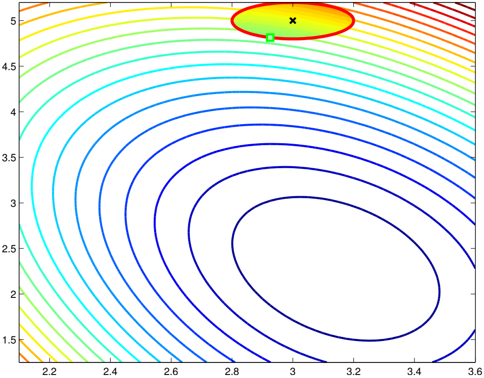


Classic NLP Technique:

- f Original function: computationally expensive, no derivatives
- m Surrogate model: computationally attractive, analytic derivatives

Basic Trust-Region Idea

Use a surrogate $m(x)$ in place of the unwieldy $f(x)$



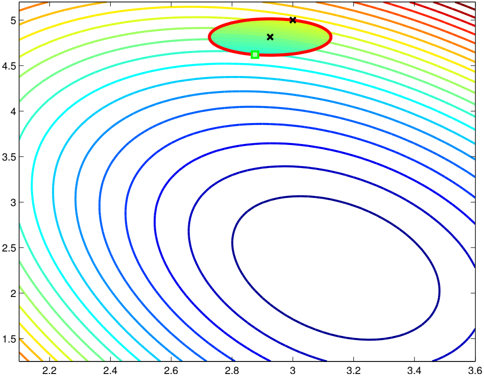
Optimize over m to avoid expense of f:

- Trust m to approximate f within $\mathcal{B} = \{x \in \mathbb{R}^n : \|x - x^k\| \leq \Delta_k\}$,
- Obtain next point from $\min\{m(x) : x \in \mathcal{B}\}$,
- Evaluate function and update (x^k, Δ_k) based on how good the model's prediction was.

[Trust-Region Methods; Conn, Gould, Toint; SIAM, 2000]

Basic Trust-Region Idea

Use a surrogate $m(x)$ in place of the unwieldy $f(x)$



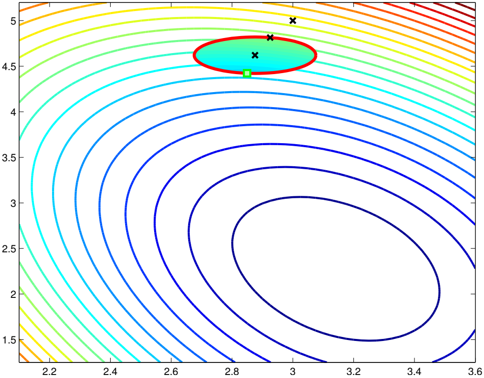
Optimize over m to avoid expense of f:

- Trust m to approximate f within $\mathcal{B} = \{x \in \mathbb{R}^n : \|x - x^k\| \leq \Delta_k\}$,
- Obtain next point from $\min\{m(x) : x \in \mathcal{B}\}$,
- Evaluate function and update (x^k, Δ_k) based on how good the model's prediction was.

[Trust-Region Methods; Conn, Gould, Toint; SIAM, 2000]

Basic Trust-Region Idea

Use a surrogate $m(x)$ in place of the unwieldy $f(x)$



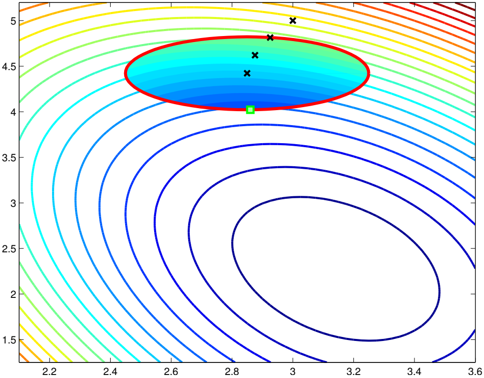
Optimize over m to avoid expense of f:

- Trust m to approximate f within $\mathcal{B} = \{x \in \mathbb{R}^n : \|x - x^k\| \leq \Delta_k\}$,
- Obtain next point from $\min\{m(x) : x \in \mathcal{B}\}$,
- Evaluate function and update (x^k, Δ_k) based on how good the model's prediction was.

[Trust-Region Methods; Conn, Gould, Toint; SIAM, 2000]

Basic Trust-Region Idea

Use a surrogate $m(x)$ in place of the unwieldy $f(x)$



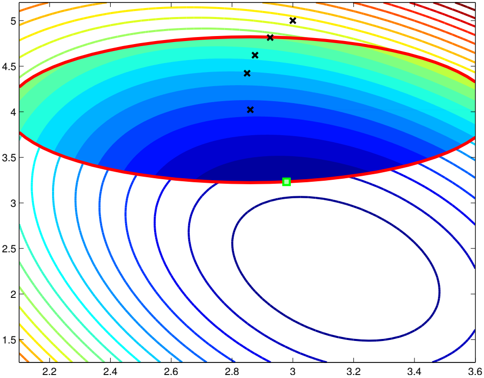
Optimize over m to avoid expense of f:

- Trust m to approximate f within $\mathcal{B} = \{x \in \mathbb{R}^n : \|x - x^k\| \leq \Delta_k\}$,
- Obtain next point from $\min\{m(x) : x \in \mathcal{B}\}$,
- Evaluate function and update (x^k, Δ_k) based on how good the model's prediction was.

[Trust-Region Methods; Conn, Gould, Toint; SIAM, 2000]

Basic Trust-Region Idea

Use a surrogate $m(x)$ in place of the unwieldy $f(x)$



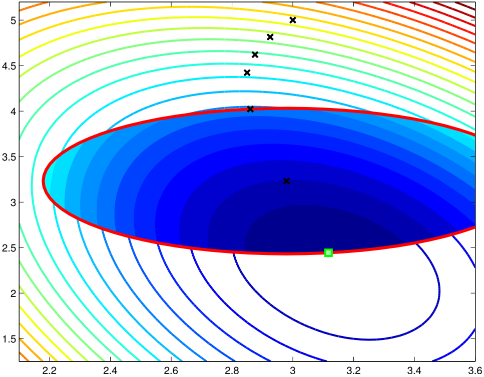
Optimize over m to avoid expense of f:

- Trust m to approximate f within $\mathcal{B} = \{x \in \mathbb{R}^n : \|x - x^k\| \leq \Delta_k\}$,
- Obtain next point from $\min\{m(x) : x \in \mathcal{B}\}$,
- Evaluate function and update (x^k, Δ_k) based on how good the model's prediction was.

[Trust-Region Methods; Conn, Gould, Toint; SIAM, 2000]

Basic Trust-Region Idea

Use a surrogate $m(x)$ in place of the unwieldy $f(x)$



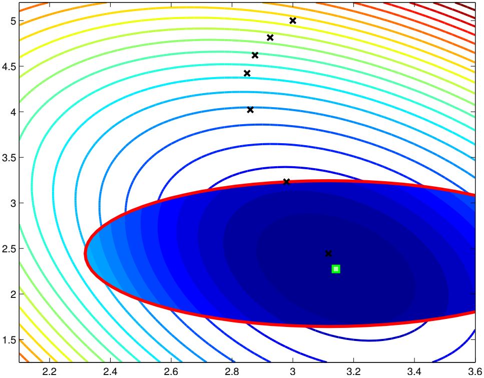
Optimize over m to avoid expense of f:

- Trust m to approximate f within $\mathcal{B} = \{x \in \mathbb{R}^n : \|x - x^k\| \leq \Delta_k\}$,
- Obtain next point from $\min\{m(x) : x \in \mathcal{B}\}$,
- Evaluate function and update (x^k, Δ_k) based on how good the model's prediction was.

[Trust-Region Methods; Conn, Gould, Toint; SIAM, 2000]

Basic Trust-Region Idea

Use a surrogate $m(x)$ in place of the unwieldy $f(x)$



Optimize over m to avoid expense of f:

- Trust m to approximate f within $\mathcal{B} = \{x \in \mathbb{R}^n : \|x - x^k\| \leq \Delta_k\}$,
- Obtain next point from $\min\{m(x) : x \in \mathcal{B}\}$,
- Evaluate function and update (x^k, Δ_k) based on how good the model's prediction was.

[Trust-Region Methods; Conn, Gould, Toint; SIAM, 2000]

Where Does the Model Come From?

When derivatives are available:

Taylor-based model $m(x^k + s) = c + (g^k)^T s + \frac{1}{2} s^T H^k s$

- $g^k = \nabla_x f(x^k)$
- $H^k \approx \nabla^2_{x,x} f(x^k)$

Where Does the Model Come From?

When derivatives are available:

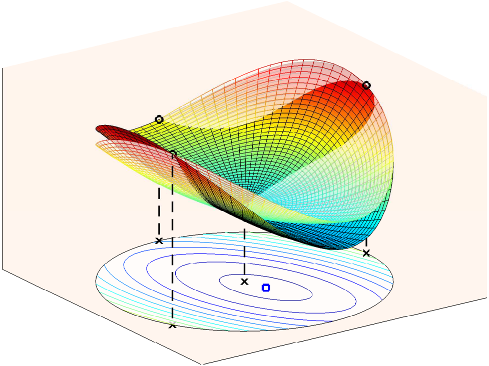
Taylor-based model $m(x^k + s) = c + (g^k)^T s + \frac{1}{2} s^T H^k s$

- $g^k = \nabla_x f(x^k)$
- $H^k \approx \nabla^2_{x,x} f(x^k)$

Without derivatives

- Interpolation-based models
- Regression-based models
- Stochastic/randomized models

Interpolation-Based Quadratic Models



An interpolating quadratic in $\mathbb{R}^2$

$m(x^k + s) = c + g^T s + \frac{1}{2} s^T H s$:

Get the model parameters $c, g, H = H^T$ by demanding interpolation:

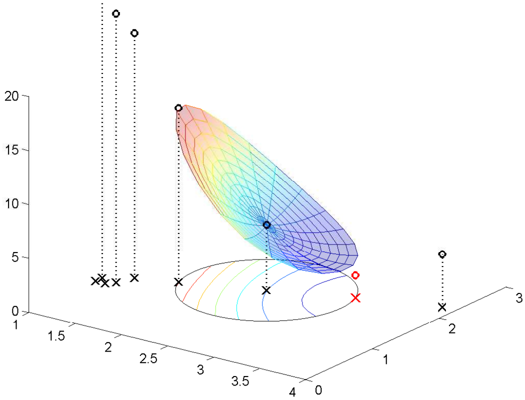
$$m(x^k + y^i) = f(x^k + y^i)$$

for all $y^i \in \mathcal{Y}$ =interpolation set

Main difficulty is $\mathcal{Y}$:

- Use prior function evaluations,
- m well-defined and approximates f locally.

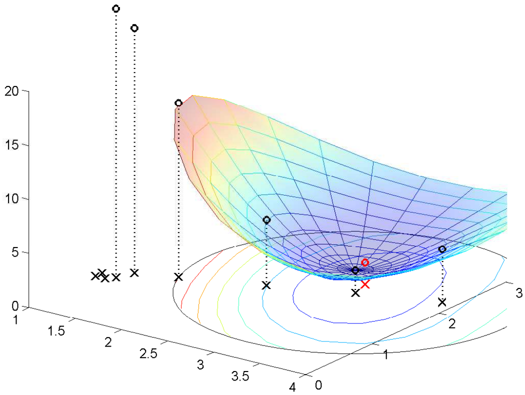
Interpolation-Based Trust-Region Methods



Iteration k:

- Build a model m_k interpolating f on $\mathcal{Y}$
- Trust m_k within region $\mathcal{B}_k$
- Minimize m_k within $\mathcal{B}_k$ to obtain next point for evaluation
- Do expensive evaluation
- Update m_k and $\mathcal{B}_k$ based on how good model prediction was

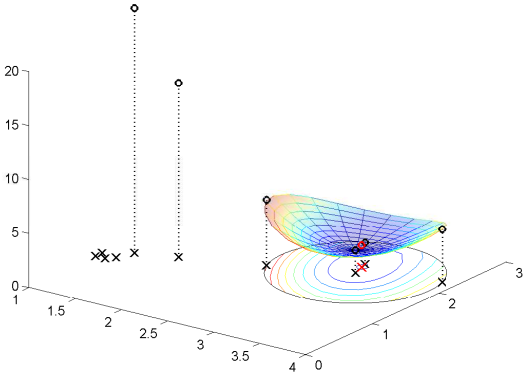
Interpolation-Based Trust-Region Methods



Iteration k:

- Build a model m_k interpolating f on $\mathcal{Y}$
- Trust m_k within region $\mathcal{B}_k$
- Minimize m_k within $\mathcal{B}_k$ to obtain next point for evaluation
- Do expensive evaluation
- Update m_k and $\mathcal{B}_k$ based on how good model prediction was

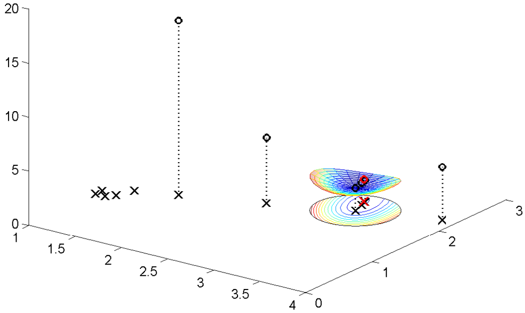
Interpolation-Based Trust-Region Methods



Iteration k:

- Build a model m_k interpolating f on $\mathcal{Y}$
- Trust m_k within region $\mathcal{B}_k$
- Minimize m_k within $\mathcal{B}_k$ to obtain next point for evaluation
- Do expensive evaluation
- Update m_k and $\mathcal{B}_k$ based on how good model prediction was

Interpolation-Based Trust-Region Methods



Iteration k:

- Build a model m_k interpolating f on $\mathcal{Y}$
- Trust m_k within region $\mathcal{B}_k$
- Minimize m_k within $\mathcal{B}_k$ to obtain next point for evaluation
- Do expensive evaluation
- Update m_k and $\mathcal{B}_k$ based on how good model prediction was

Quick Diversion: Polynomial Bases

- Let ϕ denote a basis for some space of polynomials of n variables
  - Linear:

$$\phi(x) = [1,\ x_1,\ \cdots,\ x_n]$$

Quick Diversion: Polynomial Bases

- Let ϕ denote a basis for some space of polynomials of n variables
  - Linear:
$$\phi(x) = [1,\ x_1,\ \cdots,\ x_n]$$
  - Full quadratics:
$$\phi(x) = \left[1,\ x_1,\ \cdots,\ x_n\ x_1^2,\ \cdots,\ x_n^2\ x_1x_2,\ \cdots,\ x_{n-1}x_n\right]$$

Quick Diversion: Polynomial Bases

- Let ϕ denote a basis for some space of polynomials of n variables
  - Linear:
$$\phi(x) = [1,\ x_1,\ \cdots,\ x_n]$$
  - Full quadratics:
$$\phi(x) = \left[1,\ x_1,\ \cdots,\ x_n\ x_1^2,\ \cdots,\ x_n^2\ x_1 x_2,\ \cdots,\ x_{n-1}x_n\right]$$
- Given a collection of $p = |\mathcal{Y}|$ points $\mathcal{Y} = \{y^1, \cdots, y^p\}$:

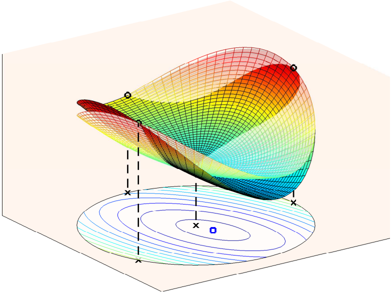
$$\Phi(\mathcal{Y}) = \begin{bmatrix} 1 & y_1^1 & \cdots & y_n^1 & (y_1^1)^2 & \cdots & (y_n^1)^2 & y_1^1 y_2^1 & \cdots & y_{n-1}^1 y_n^1 \\ \vdots & & & & & & & & & \vdots \\ 1 & y_1^p & \cdots & y_n^p & (y_1^p)^2 & \cdots & (y_n^p)^2 & y_1^p y_2^p & \cdots & y_{n-1}^p y_n^p \end{bmatrix}$$

This is a matrix of size $p \times \frac{(n+1)(n+2)}{2}$

Building Models Without Derivatives

Given $\left(\mathcal{Y}^k, f\left(\mathcal{Y}^k\right)\right)$, "solve"

$$\Phi(\mathcal{Y}^k)z = \begin{bmatrix} \Phi_c & \Phi_g & \Phi_H \end{bmatrix} \begin{bmatrix} z_c \\ z_g \\ z_H \end{bmatrix} = \underline{\mathsf{f}} = f\left(\mathcal{Y}^k\right)$$



$n = 2$, $|\mathcal{Y}^k| = 4$

Full quadratics, $|\mathcal{Y}^k| = \frac{(n+1)(n+2)}{2}$

- ◇ Geometric conditions on points in $\mathcal{Y}^k$

Undetermined interpolation, $|\mathcal{Y}^k| < \frac{(n+1)(n+2)}{2}$

- ◇ Use (Powell) Hessian updates

$$\begin{aligned} \min_{g^k, H^k} \quad & \|H^k - H^{k-1}\|_F^2 \\ \text{s.t.} \quad & q_k = \underline{\mathsf{f}} \text{ on } \mathcal{Y}^k \end{aligned}$$

Regression, $|\mathcal{Y}^k| > \frac{(n+1)(n+2)}{2}$

- ◇ Solve $\min_z \|\Phi z - \underline{\mathsf{f}}\|$

Multivariate (Scattered Data) Interpolation is a Different Kind of Animal

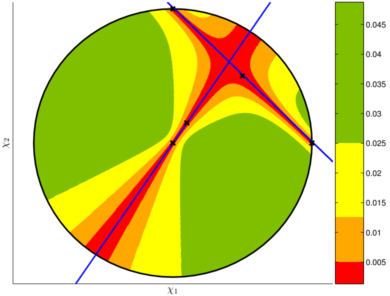
$$m(x^k + y^i) = f(x^k + y^i) \qquad \forall y^i \in \mathcal{Y}$$

$n = 1$ Given p distinct points, can find a unique degree $p-1$ polynomial m

$n > 1$ Not true! (see Mairhuber-Curtis Theorem)

For quadratic models in $\mathbb{R}^n$:

- $\frac{(n+1)(n+2)}{2}$ coefficients
- Unique interpolant may not exist, even when $|\mathcal{Y}| = \frac{(n+1)(n+2)}{2}$
- Locations of the points in $\mathcal{Y}$ must satisfy additional geometric conditions (has nothing to do with f values)



→ [Scattered Data Approximation; Wendland; Cambridge University Press, 2010]

Notions of Nonlinear Model Quality

"Taylor-like" Error Bounds

1. Assuming underlying f is sufficiently smooth
   $=$ derivatives of f exist but are unavailable
2. A model m_k is locally fully linear if:

   For all $x \in \mathcal{B}_k = \{x \in \Omega : \|x - x^k\| \leq \Delta_k\}$

   - $|m_k(x) - f(x)| \leq \kappa_1 \Delta_k^2$
   - $\|\nabla m_k(x) - \nabla f(x)\| \leq \kappa_2 \Delta_k$

   for constants κ_i independent of x and Δ_k.

→[Intro. to DFO; Conn, Scheinberg, Vicente; SIAM 2009]

Notions of Nonlinear Model Quality

"Taylor-like" Error Bounds

1. Assuming underlying f is sufficiently smooth
2. A model m_k is locally fully quadratic if:

   For all $x \in \mathcal{B}_k = \{x \in \Omega : \|x - x^k\| \leq \Delta_k\}$

   - $|m_k(x) - f(x)| \leq \kappa_1 \Delta_k^3$
   - $\|\nabla m_k(x) - \nabla f(x)\| \leq \kappa_2 \Delta_k^2$
   - $\|\nabla^2 m_k(x) - \nabla^2 f(x)\| \leq \kappa_3 \Delta_k$

   for constants κ_i independent of x and Δ_k.

→[Intro. to DFO; Conn, Scheinberg, Vicente; SIAM 2009]

Ingredients for Convergence to Stationary Points

Assuming underlying f is sufficiently smooth and regular (e.g., has bounded level sets)

$\lim_{k\to\infty} \nabla f(x^k) = 0$ provided:

1. Control $\mathcal{B}_k$ based on model quality
2. (Occasional) approximation within $\mathcal{B}_k$
   Our quadratics satisfy
   - $|q_k(x) - f(x)| \le \kappa_1(\gamma_f + \|H^k\|)\Delta_k^2, \qquad x \in \mathcal{B}_k$
   - $\|g^k + H^k(x - x^k) - \nabla f(x)\| \le \kappa_2(\gamma_f + \|H^k\|)\Delta_k, \qquad x \in \mathcal{B}_k$
3. Sufficient decrease

At least model gradient should be good

Radial Basis Function (RBF) models also fit in this framework

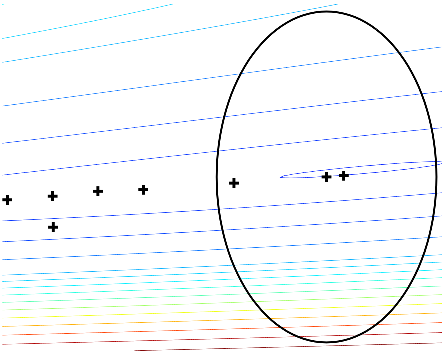
→ [W. & Shoemaker, SIREV 2013]

General introduction → [Intro. to DFO; Conn, Scheinberg, Vicente; SIAM 2009]

Greed. Alone. Can. Hurt.

Model-improvement may be needed when:

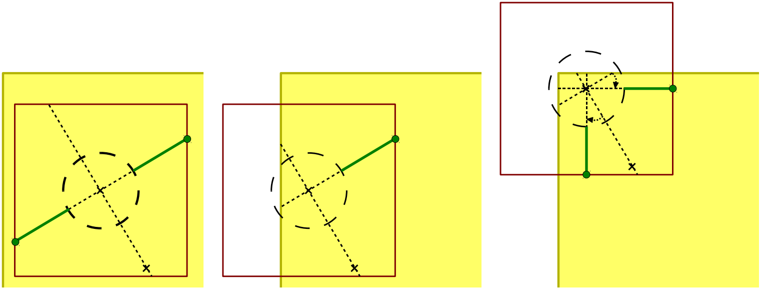
- Nearby points line up
- May not have enough points to ensure model quality in all directions



→ May need n additional evaluations

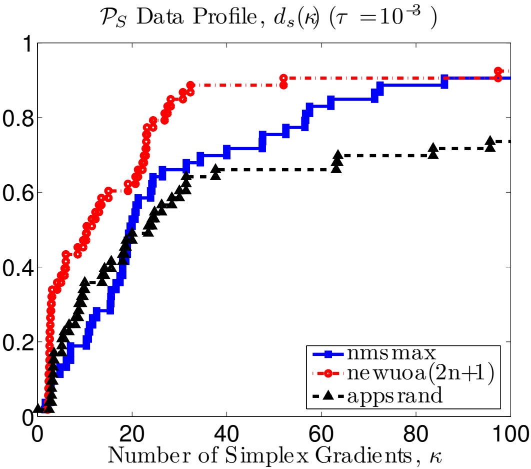
Constraints and Model Quality

Constraints complicate matters
. . . if one does not allow evaluation of infeasible points



→ May need directions normal to nearby constraints

Performance Comparisons on Test Functions

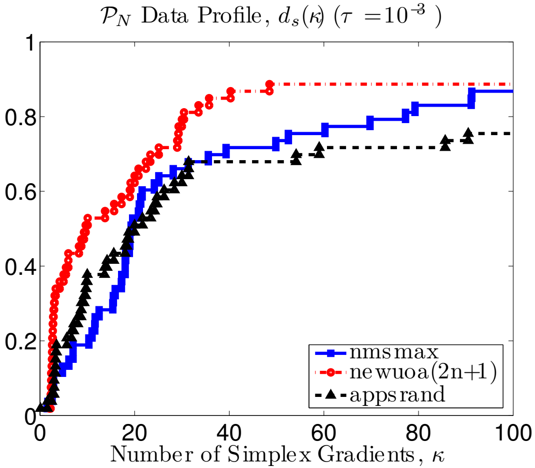


Smooth problems

- When evaluations are sequential, model-based methods (NEWUOA) regularly outperform direct search methods without a search phase (nmsmax, **appsrand**)

→ [Moré & W., SIOPT 2009]

Performance Comparisons on Test Functions



Noisy problems

- ◇ When evaluations are sequential, model-based methods (NEWUOA) regularly outperform direct search methods without a search phase (nmsmax, **appsrand**)

→ [Moré & W., SIOPT 2009]

Many Practical Details In Implementations

- Choice of interpolation points $\mathcal{Y}^k$
- Updating of trust region $\mathcal{B}_k$
- Improvement of models

Many Practical Details In Implementations

- Choice of interpolation points $\mathcal{Y}^k$
- Updating of trust region $\mathcal{B}_k$
- Improvement of models

BOBYQA [Powell], DFO [Scheinberg], POUNDer [W.]

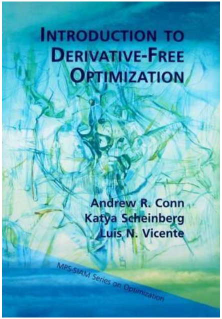
Initialization $p = |\mathcal{Y}^k|$ structured evaluations
Based on input, $\approx 2n+1$
Based on input, no more than $n+1$

Interpolation Set $p = |\mathcal{Y}^k|, \forall k$
Bootstrap to $|\mathcal{Y}^k| = \frac{(n+1)(n+2)}{2}$, then fixed
Varies in $\{n+1, \cdots, \frac{(n+1)(n+2)}{2}\}$ based on available points

Linear Algebra If $p = \mathcal{O}(n)$, model formation costs only $\mathcal{O}(n^2)$
Expensive
Expensive

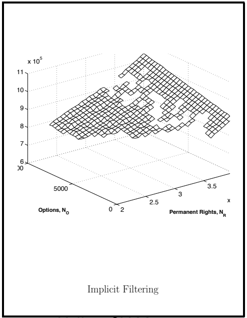
Growing Body of Tools and Resources for Local DFO

? What to use on problems with characteristics X, Y, and Z ?



INTRODUCTION TO DERIVATIVE-FREE OPTIMIZATION
Andrew R. Conn
Katya Scheinberg
Luis N. Vicente
MPS-SIAM Series on Optimization

Conn, Scheinberg, Vicente; SIAM 2009



Kelley; SIAM 2011

Many solvers

Sample considered by Rios & Sahinidis, 2010:

ASA,
CMA-ES,
DAKOTA/*,
FMINSEARCH,
HOPSPACK,
MCS,
NOMAD,
SID-PSM,

BOBYQA,
DFO,
TOMLAB/*,
GLOBAL,
IMFIL,
NEWUOA,
PSWARM,
SNOBFIT

Toward Global Optimization

A quick sketch of a multistart methods and some practical details

- ◇ useful in derivative-based and derivative-free cases
- ◇ obtain a list of distinct minimizers (for post-processing, etc.)
- ◇ simple to get started
- ! simple to abuse/misuse ("I found all minimizers")

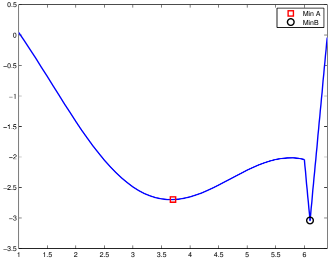
Why Multistart?

Multiple local minima are often of interest in practice:

Design: Multiple objectives (or even constraints) might later be of interest

Simulation Errors: Could have spurious local minima from anomalies in the simulator

Uncertainty: Some minima are more sensitive to perturbations than others (gentle valleys versus steep cliffs)



Global Optimization Multistart Methods

Two phase iterative method

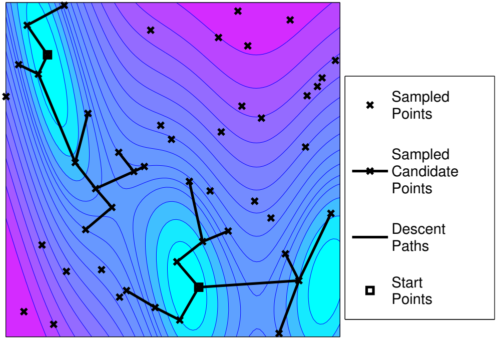
Global Exploration: Sample N points in $\mathcal{D}$. ← Guarantees convergence

Local Refinement: Start a local minimization algorithm $\mathcal{A}$ from some promising subset of the sample points.

Want to find many (good) local minima while avoiding repeatedly finding the same local minima.

Multi Level Single Linkage (MLSL) Clustering Procedure

Where to start $\mathcal{A}$ in kth iteration [Rinnooy Kan & Timmer, Math. Programming 1987]



Ex.: It. 1 Exploration

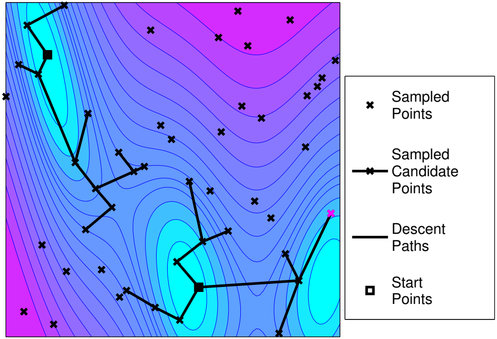
Start $\mathcal{A}$ at each sample point x^i provided:

- $\mathcal{A}$ has not been started from x^i, and
- no other sample point x^j with $f(x^j) < f(x^i)$ is within a distance

$$r_k = \frac{1}{\sqrt{\pi}} \sqrt[n]{\text{vol}(\mathcal{D}) \frac{5\Gamma\left(1+\frac{n}{2}\right)\log(kN)}{kN}},$$

Multi Level Single Linkage (MLSL) Clustering Procedure

Where to start $\mathcal{A}$ in kth iteration [Rinnooy Kan & Timmer, Math. Programming 1987]



Ex.: It. 1 Exploration

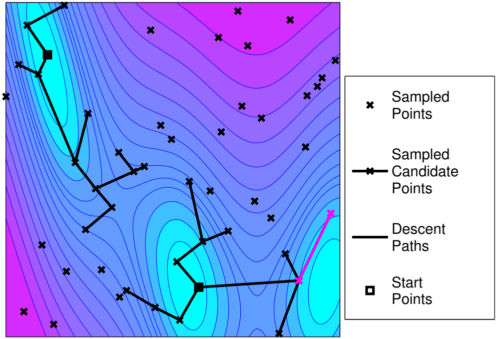
Start $\mathcal{A}$ at each sample point x^i provided:

- $\mathcal{A}$ has not been started from x^i, and
- no other sample point x^j with $f(x^j) < f(x^i)$ is within a distance

$$r_k = \frac{1}{\sqrt{\pi}} \sqrt[n]{\text{vol}(\mathcal{D}) \frac{5\Gamma\left(1+\frac{n}{2}\right)\log(kN)}{kN}},$$

Multi Level Single Linkage (MLSL) Clustering Procedure

Where to start $\mathcal{A}$ in kth iteration [Rinnooy Kan & Timmer, Math. Programming 1987]



Start $\mathcal{A}$ at each sample point x^i provided:

- ◇ $\mathcal{A}$ has not been started from x^i, and
- ◇ no other sample point x^j with $f(x^j) < f(x^i)$ is within a distance

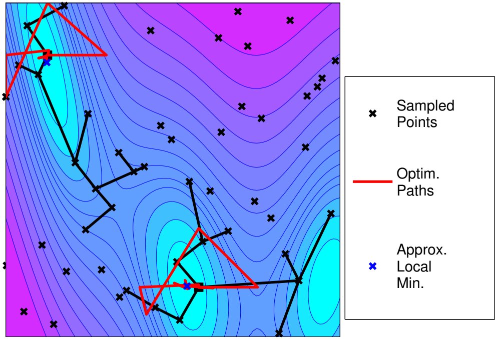
$$r_k = \frac{1}{\sqrt{\pi}} \sqrt[n]{\text{vol}(\mathcal{D}) \frac{5\Gamma\left(1+\frac{n}{2}\right)\log(kN)}{kN}},$$

Ex.: It. 1 Exploration

Thm [RK-T]- Will start finitely many local runs with probability 1.

Multi Level Single Linkage (MLSL) Clustering Procedure

Where to start $\mathcal{A}$ in kth iteration [Rinnooy Kan & Timmer, Math. Programming 1987]



Ex.: It. 1 Refinement

Start $\mathcal{A}$ at each sample point x^i provided:

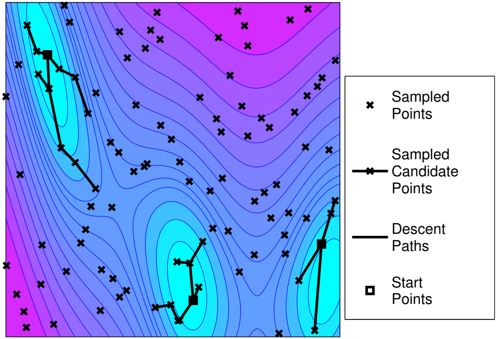
- $\mathcal{A}$ has not been started from x^i, and
- no other sample point x^j with $f(x^j) < f(x^i)$ is within a distance

$$r_k = \frac{1}{\sqrt{\pi}} \sqrt[n]{\text{vol}(\mathcal{D}) \frac{5\Gamma\left(1+\frac{n}{2}\right)\log(kN)}{kN}},$$

Thm [RK-T]- Will start finitely many local runs with probability 1.

Multi Level Single Linkage (MLSL) Clustering Procedure

Where to start $\mathcal{A}$ in kth iteration [Rinnooy Kan & Timmer, Math. Programming 1987]



Start $\mathcal{A}$ at each sample point x^i provided:

- $\mathcal{A}$ has not been started from x^i, and
- no other sample point x^j with $f(x^j) < f(x^i)$ is within a distance

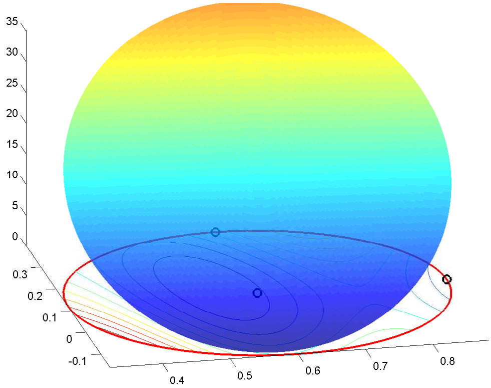
$$r_k = \frac{1}{\sqrt{\pi}} \sqrt[n]{\text{vol}(\mathcal{D}) \frac{5\Gamma\left(1+\frac{n}{2}\right)\log(kN)}{kN}},$$

Ex.: It. 2 Exploration

Thm [RK-T]- Will start finitely many local runs with probability 1.

Using External Points To Form Better Local Models

Initial model interpolating $n+1=3$ points



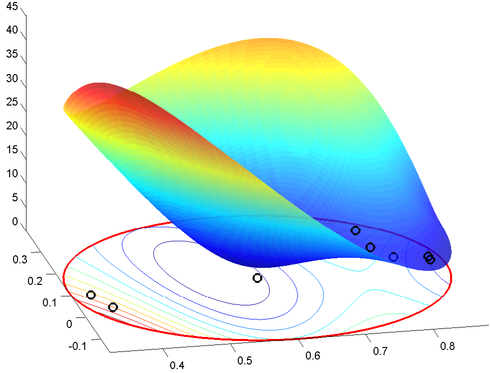
Take advantage of global history:

- from current minimization,
- from previous minimizations, and
- from the global sampling.

More information means rapider progress.

Using External Points To Form Better Local Models

Initial model interpolating 8 sample points

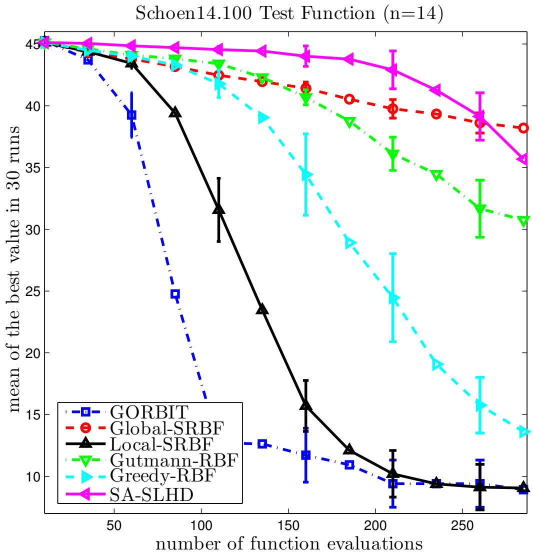


Take advantage of global history:

- from current minimization,
- from previous minimizations, and
- from the global sampling.

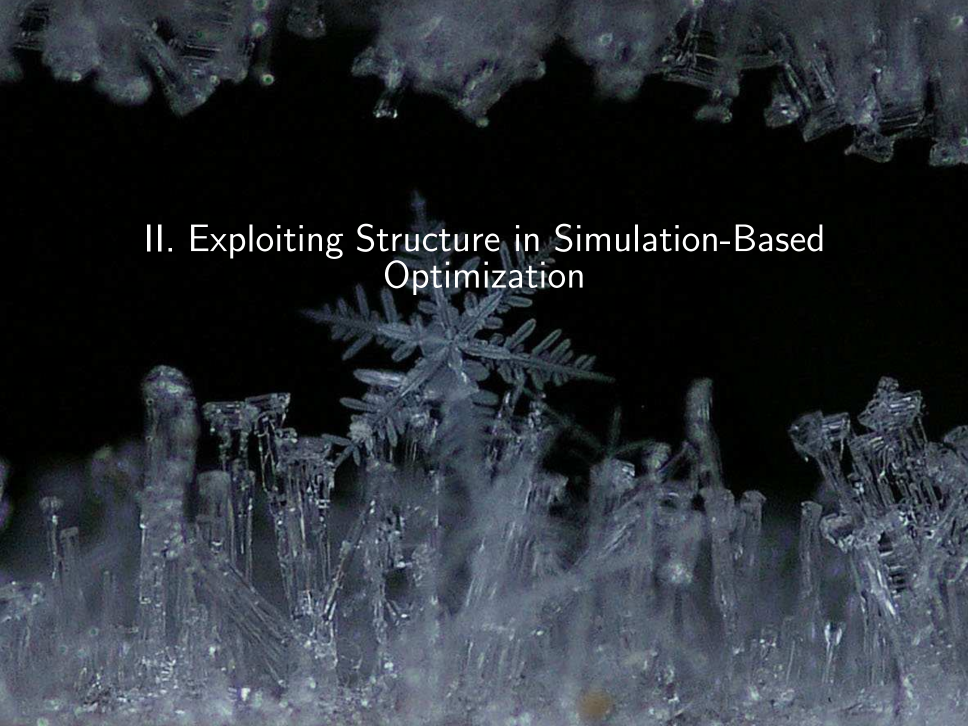
More information means rapider progress.

Performance Comparisons on Test Functions



- ◇ GORBIT is multistart with RBF model-based method
- ◇ SA-SLHD is a heuristic (simulated annealing with a symmetric Latin hypercube design as initialization)

→ [W., Cornell University, 2009]



II. Exploiting Structure in Simulation-Based Optimization

Beyond the Black Box

$$\min f(x) = F[S(x)]$$

So far, $f = S$

Beyond the Black Box

$$\min f(x) = F[S(x)]$$

So far, $f = S$

Your problems are not black-box problems

Beyond the Black Box

$$\min f(x) = F[S(x)]$$

So far, $f = S$

Your problems are not black-box problems

You formulated the problem

$\Rightarrow$ You know more than nothing

Structure in Simulation-Based Optimization, $\min f(x) = F[S(x)]$

f is often not a black box S

- Nonlinear least squares

$$f(x) = \frac{1}{2}\sum_i (S_i(x) - d_i)^2$$

- Not all variables enter simulation

$$f(x) = g(x_I, x_J) + h(S(x_J))$$

- Only some constraints depend on simulation

$$\min\{f(x) : c_1(x) = 0, c_S(x) = 0\}$$

- Slack variables

$$\Omega_S = \{(x_I, x_J) : S(x_J) + x_I = 0, x_I \geq 0\}$$

Model-based methods can be a great way to exploit this structure

Ex. 1- Least Squares $f(x) = \frac{1}{2}\sum_i F_i(x)^2$

Obtain a vector of output $F_1(x), \ldots, F_p(x)$

- Model each F_i

$$F_i(x) \approx q_k^{(i)}(x) = F_i(x^k) + (x - x^k)^\top g^{(i,k)} + \frac{1}{2}(x - x^k)^\top H^{(i,k)}(x - x^k)$$

- Approximate:

$$\begin{aligned}
\nabla f(x) &= \sum_i \nabla \mathbf{F_i}(\mathbf{x}) F_i(x) \qquad \to \sum_i \nabla q_k^{(i)}(x) F_i(x) \\
\nabla^2 f(x) &= \sum_i \nabla \mathbf{F_i}(\mathbf{x}) \nabla \mathbf{F_i}(\mathbf{x})^T + \sum_i F_i(x) \nabla^\mathbf{2} \mathbf{F_i}(\mathbf{x}) \\
&\to \sum_i \nabla q_k^{(i)}(x) \nabla q_k^{(i)}(x)^T + \sum_i F_i(x) \nabla^2 q_k^{(i)}(x)
\end{aligned}$$

- Model f via Gauss-Newton or similar

→ [DFLS; Zhang, Conn, Scheinberg] regularized Hessians

→ [POUNDERS; W., Moré] uses full Newton

Ex. 1- Consequences for $f(x) = \frac{1}{2}\sum_i F_i(x)^2$

- Save linear algebra using interpolation set $\mathcal{Y}^k$ common to all models
  - Single system solve, multiple right hand sides

$$\Phi(\mathcal{Y}^k)\begin{bmatrix} z^{(1)} & \cdots & z^{(p)} \end{bmatrix} = \begin{bmatrix} \underline{\mathsf{F}}_1 & \cdots & \underline{\mathsf{F}}_p \end{bmatrix}$$

  - fully linear $q^{(0)} \Rightarrow$ all $q^{(i)}$ fully linear
- (nearly) exact gradients for F_i (nearly) linear
- No longer interpolate function at data points

$$\begin{aligned} m(x^k+\delta) = \quad & f(x^k) + \delta^T \textstyle\sum_i g^{(i,k)} F_i(x^k) \\ & + \tfrac{1}{2}\delta^T \textstyle\sum_i \left(g^{(i,k)} (g^{(i,k)})^T + F_i(x^k) H^{(i,k)} \right) \delta \\ & + \text{missing h.o. terms} \end{aligned}$$

Ex. 1- Calibrating Energy Density Functionals

$$\min_x \left\{ f(x) = \sum_{i=1}^{p} \left(\frac{s_i(x) - d_i}{w_i} \right)^2 \right\}$$

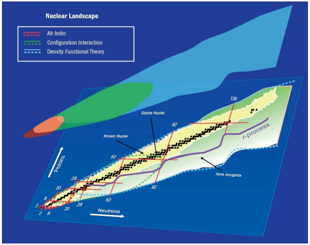
$s_i(x)$ Simulated (DFT) nucleus property

d_i Nucleus i experimental data

w_i Weight for data type i

p Parallel simulations

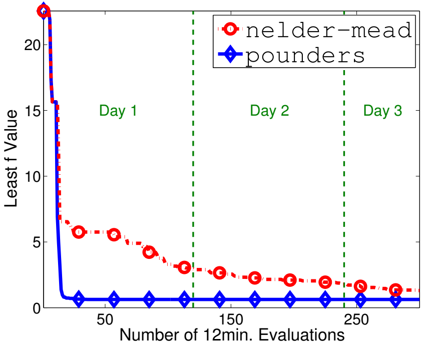
- Engineering starting point x_0
- Bound constraints scaled to unit cube
- 12 CPU minutes per evaluation



SciDAC physics project UNEDF

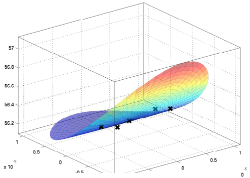
Ex. 1- POUNDERS in Practice

- Models for each residual $\frac{d_i - s_{t,i}(x)}{\sigma_i}$.
  $\{m^i(x)\}_{i=1}^{90}$
- Further reduces # of expensive evaluations

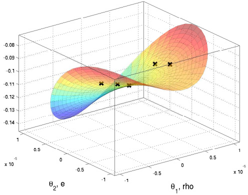


→ [Kortelainen et al., PhysRevC '10]

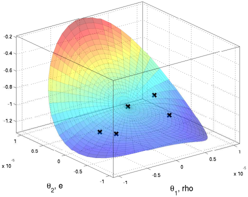
Objective Value



Energy Residual [MeV], Nucleus #10



Energy Residual [MeV], Nucleus #22



Ex. 2- Some Known Partials

$x = (x_I, x_J)$; have $\frac{\partial f}{\partial x_I}$ but not $\frac{\partial f}{\partial x_J}$

"Solve"

$$\Phi z = \underline{\mathsf{f}}$$

with known $z_{g,I}, z_{H,I}$

$$\left[\begin{array}{ccc} \Phi_c & \Phi_{g,J} & \Phi_{H,J} \end{array} \right] \left[\begin{array}{c} z_c \\ z_{g,J} \\ z_{H,J} \end{array} \right] = \underline{\mathsf{f}} - \Phi_{g,I} z_{g,I} - \Phi_{H,I} z_{H,I}$$

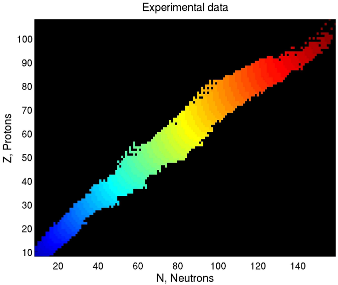
◇ Effectively lowers dimension to $|J| = n - |I|$ for

- approximation
- model-improving evaluations
- linear algebra

◇ Still have interpolation where required

Ex. 2- Multi-level Optimization Structure

$$\min_x \left\{ f(x) = \sum_{i=1}^{p} (s_i(x) - d_i)^2 \right\}$$



[Bertolli, Papenbrock, W., PhysRevC '12]

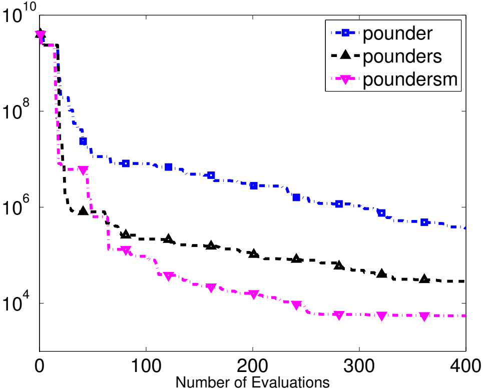
$s_i(x)$ solution to lower level problem

$$\begin{aligned} s_i(x) &= \tilde{g}_i(x) + \min_y \{h_i(x_J; y) : y \in \mathcal{D}_i\} \\ &= \tilde{g}_i(x) + h_i(x_J; y_{i,*}[x_J]) \\ \nabla_{\mathbf{x_J}} \mathbf{s_i(x)} &\approx \nabla_{x_J} \tilde{g}_i(x) + \nabla_{x_J} q^{(i)}(x_J) \end{aligned}$$

For $x = (x_I, x_J)$

- $\nabla_{x_I} s_i(x_I, x_J)$ available
- $s_i(x)$ continuous and smooth in x_I
- $\tilde{g}_i(x)$ cheap to compute!
- No noise/errors introduced in $\tilde{g}_i(x)$

Ex. 2- Numerical Results With Some Partials



s exploit least squares

m use ∇_{x_I} derivatives

◇ $n = 16,\ |I| = 3$

◇ 5-10 seconds/evaluation on 8 cores

"Same algorithmic framework",
performance advantages from exploiting structure

Ex. 2- Convergence With Known Partials ∇_{x_I}

$\lim_{k\to\infty} \nabla f(x^k) = 0$ as before

Approximation bounds

- $|q_k(x) - f(x)| \le \kappa_1(\gamma_f + \|H^k\|)\Delta_k^2, \qquad x \in \mathcal{B}_k$
- $\|g^k + H^k(x - x^k) - \nabla f(x)\| \le \kappa_2(\gamma_f + \|H^k\|)\Delta_k, \qquad x \in \mathcal{B}_k$

with most constants now a function of $n - |I|$.

Guaranteed

- strict descent
- optimality

in some directions.

Ex. 3- General Constraints

$\min\{f(x) : c_1(x) = 0, c_S(x) = 0\}$

- Approximate Lagrangian:

$$\begin{aligned} \nabla L &= \nabla f + \lambda_1^T \nabla c_1 + \lambda_2^T \nabla \mathbf{c_S} \\ &\to \nabla f + \lambda_1^T \nabla c_1 + \lambda_2^T \nabla m \end{aligned}$$

- Use favorite method: filters, augmented Lagrangian, . . .
- Slack variables
  - Do not increase effective dimension
  - Subproblems can treat separately
  - Know derivatives

→[Diniz-Ehrhardt, Martínez, Pedroso, C&A Math. 2011]: modified AL methods

Ex. 3- What Constraint Derivatives Buy You

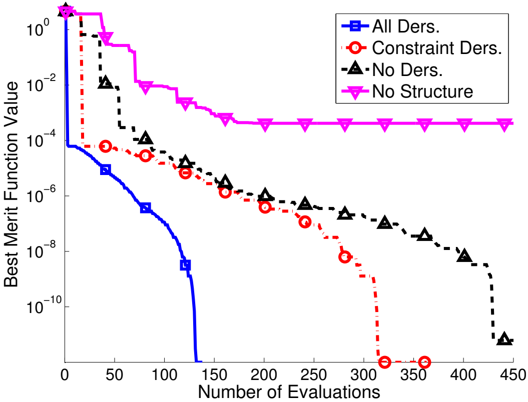
Augmented Lagrangian methods, $L_A(x, \lambda; \mu) = f(x) - \lambda^T c(x) + \frac{1}{\mu}\|c(x)\|^2$

$$\min_x \{f(x) : c(x) = 0\}$$

Four choices:

1. Penalize constraints
2. Treat c and f both as (separate) black boxes
3. Work with f and $\nabla_x c$
4. Have both $\nabla_x f$ and $\nabla_x c$

→ *With Slava Kungurtsev (UCSD)*



$n = 15$, 11 constraints

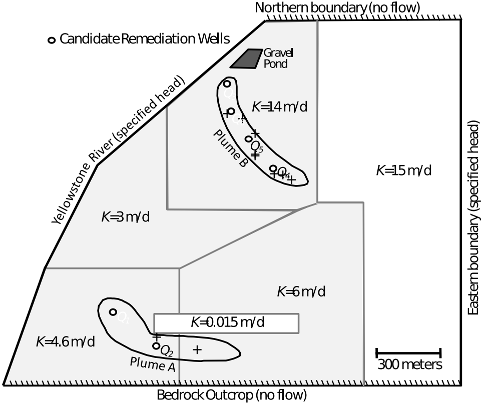
Ex. 4- Remediation of Chlorinated Solvents

Determine extraction rates for 6 installed wells

Structure

- Minimize operating cost (linear)
- Plume flux constraints (expensive simulation)

$$c_P(x) = \sum_{i=1}^{p} |S_{P,i}(x)|$$


Northern boundary (no flow)
Candidate Remediation Wells
Gravel Pond
Yellowstone River (specified head)
Q_3
Q_6
K=14 m/d
Plume B
Q_5
Q_4
K=15 m/d
Eastern boundary (specified head)
K=3 m/d
K=6 m/d
Q_1
K=0.015 m/d
K=4.6 m/d
Q_2
Plume A
300 meters
Bedrock Outcrop (no flow)

Lockwood Solvent Ground Water Plume Site (LSGPS)

Ex. 4- Exploiting Nonsmoothness

What if derivatives of $f(x)$ do not always exist?

$$f(x) = g(x, S(x))$$

- g Nonsmooth, known
- S Smooth, black-box

  Examples:

  - $f(x) = \sum_{i=1}^{p} |F_i[S(x)]|$
  - $f(x) = \max(S_1(x), S_2(x))$

→ *With Aswin Kannan*

Ex. 4- Targeting Nonsmoothness in $f(x) = \sum_{i=1}^{p} |F_i(x)|$

Model-based Approaches:

pounder Ignore structure, model f as usual

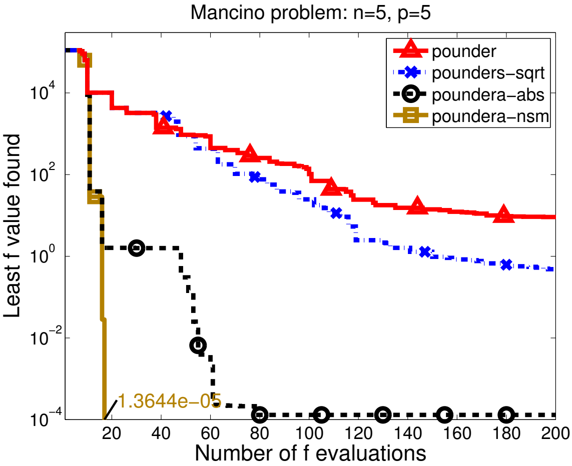
pounders-sqrt $f = \sum\limits_{i=1}^{p} \sqrt{|F_i|}^2$,
model $\sqrt{|F_i|}$ by Q_i
subproblem $\min \sum_{i=1}^{p} \tilde{Q}_i(x)^2$

poundera-abs $f = \sum\limits_{i=1}^{p} |F_i|$,
model $|F_i|$ by Q_i
subproblem $\min \sum_{i=1}^{p} Q_i(x)$

poundera-nsm $f = \sum\limits_{i=1}^{p} |F_i|$,
model F_i by Q_i
subproblem $\min \sum_{i=1}^{p} |Q_i(x)|$

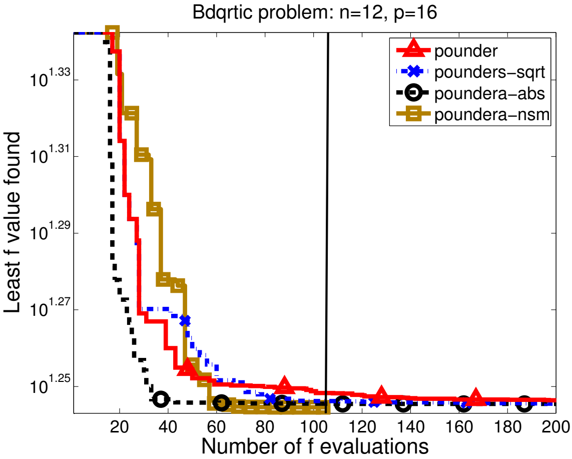
Ex. 4- Preliminary Results, $\min \sum_{i=1}^{p} |F_i(x)|$

pounder	black-box
pounders-sqrt	$\sum_{i=1}^{p} \tilde{Q}_i(x)^2$
poundera-abs	$\sum_{i=1}^{p} Q_i(x)$
poundera-nsm	$\sum_{i=1}^{p} \lvert Q_i(x) \rvert$



Ex. 4- Preliminary Results, $\min \sum_{i=1}^{p} |F_i(x)|$

pounder	black-box
pounders-sqrt	$\sum_{i=1}^{p} \tilde{Q}_i(x)^2$
poundera-abs	$\sum_{i=1}^{p} Q_i(x)$
poundera-nsm	$\sum_{i=1}^{p} \lvert Q_i(x) \rvert$



So You Want To Solve A Hard Optimization Problem?

Mathematically unwrap problems to expose the deepest black boxes!

◇ It is easy to get started with derivative-free methods

◇ You should strive to obtain derivatives & apply methods from Todd's talk

◇ Model-based methods can make use of expensive function values

◇ Structure is everywhere, even in "black-box" / legacy code-driven optimization problems

◇ By exploiting structure, optimization can solve grand-challenge problems in ⟨`insert your field here`⟩:

- Model residuals $\{r_i(x)\}_i$, not $\|r(x)\|$
- Model constraints $\{c_i(x)\}_i$, not a penalty $P(c(x))$
- Explicitly handle nonsmoothness (and noise)