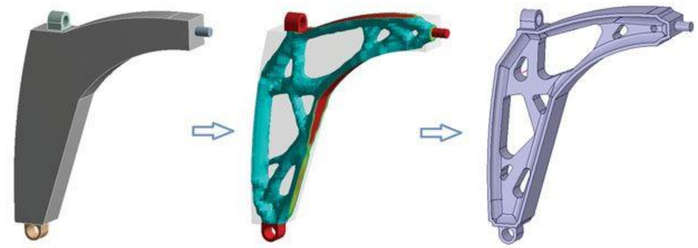
香港中文大學
The Chinese University of Hong Kong

MAEG5160: Design for Additive Manufacturing

Lecture 0: Introduction to optimization



Prof SONG Xu

Department of Mechanical and Automation Engineering,
The Chinese University of Hong Kong.

Ingredients

- Objective function
- Variables
- Constraints

Find values of the variables
that minimize or maximize the objective function
while satisfying the constraints

Different Kinds of Optimization

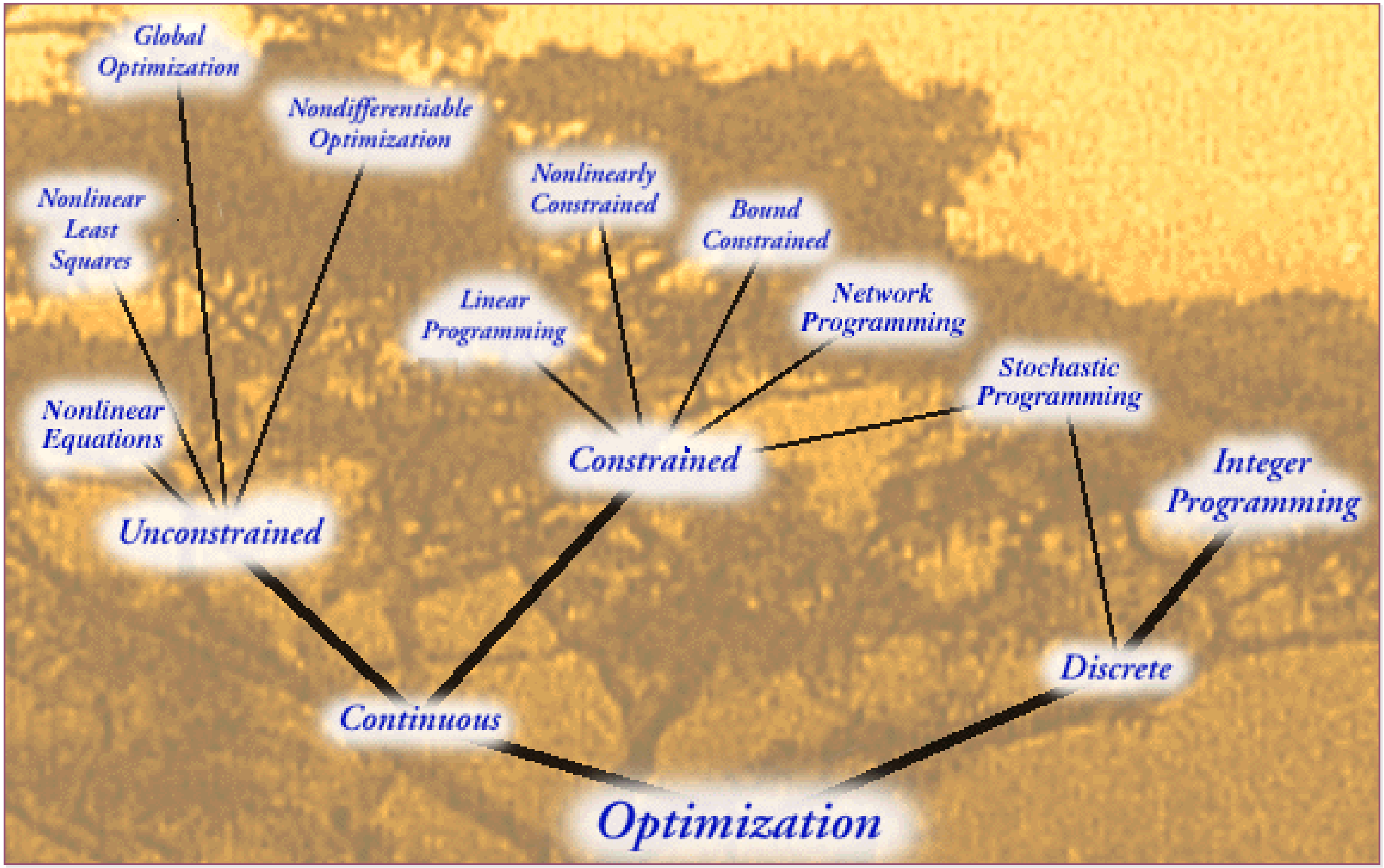
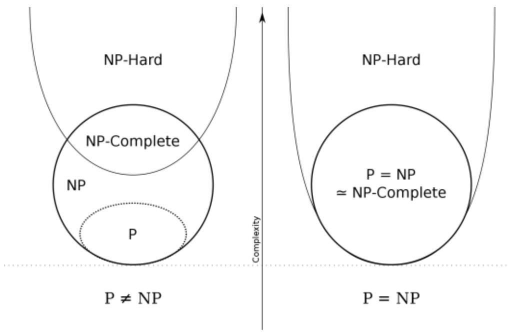


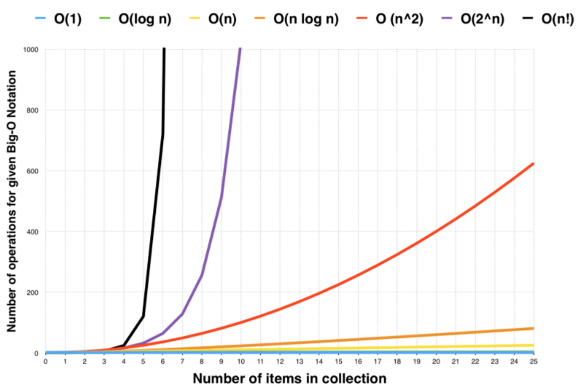
Figure from: Optimization Technology Center
http://www-fp.mcs.anl.gov/otc/Guide/OptWeb/

Different Optimization Techniques

- Algorithms have very different flavor depending on specific problem
  - Closed form vs. numerical vs. discrete
  - Local vs. global minima
  - Running times ranging from O(1) to NP-hard

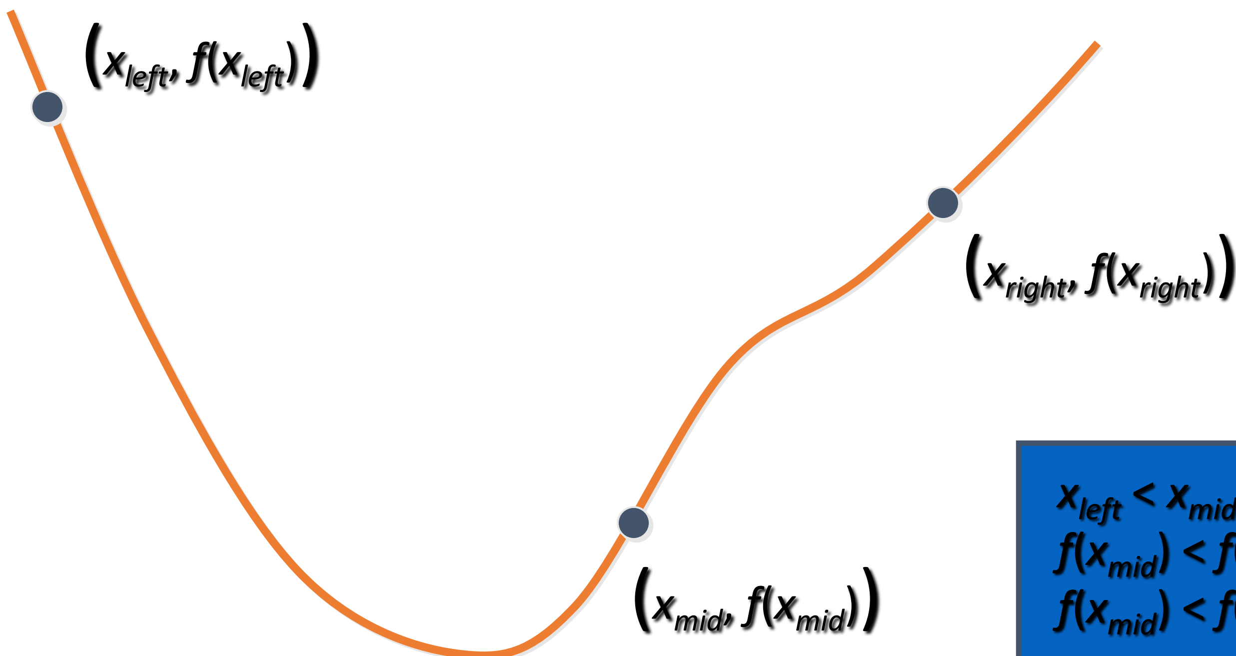


- Today:
  - Focus on continuous numerical methods



Optimization in 1-D

- Look for analogies to bracketing in root-finding
- What does it mean to *bracket* a minimum?



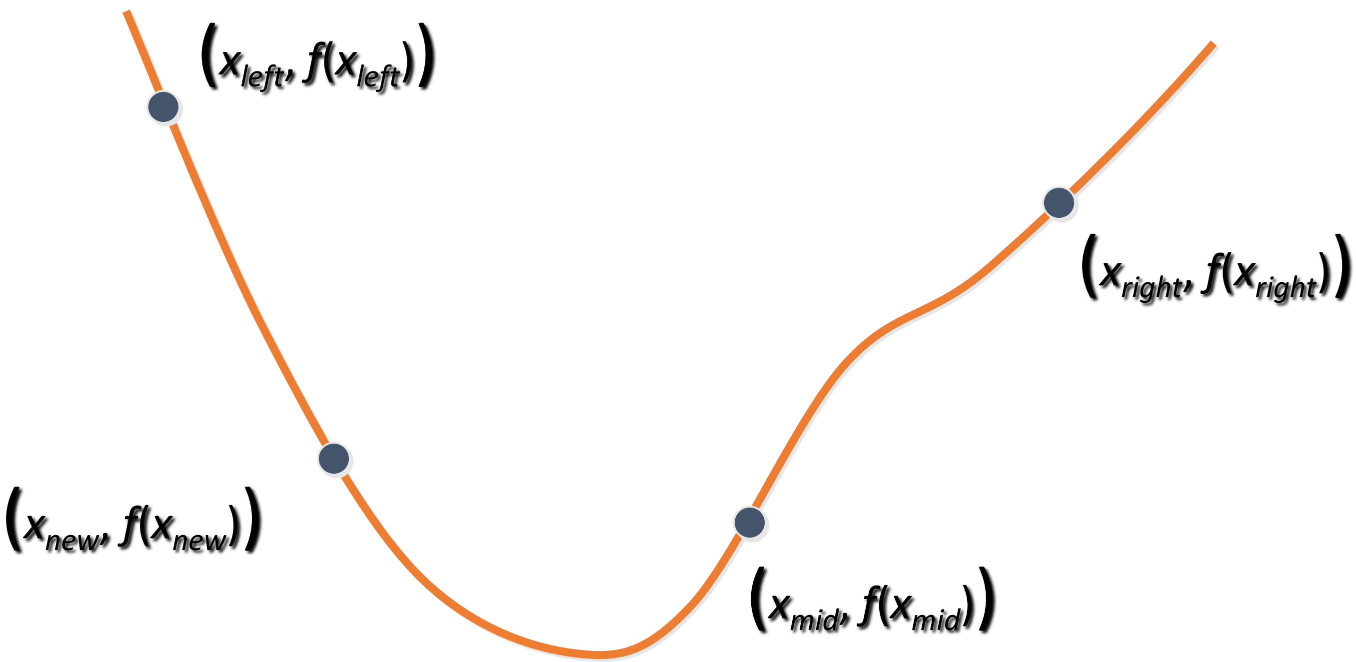
$x_{left} < x_{mid} < x_{right}$
$f(x_{mid}) < f(x_{left})$
$f(x_{mid}) < f(x_{right})$

Optimization in 1-D

- Once we have these properties, there is at least one local minimum between x_{left} and x_{right}
- Establishing bracket initially:
  - Given $x_{initial}$, *increment*
  - Evaluate $f(x_{initial})$, $f(x_{initial}+increment)$
  - If decreasing, step until find an increase
  - Else, step in opposite direction until find an increase
  - Grow increment at each step
- For maximization: substitute $-f$ for f

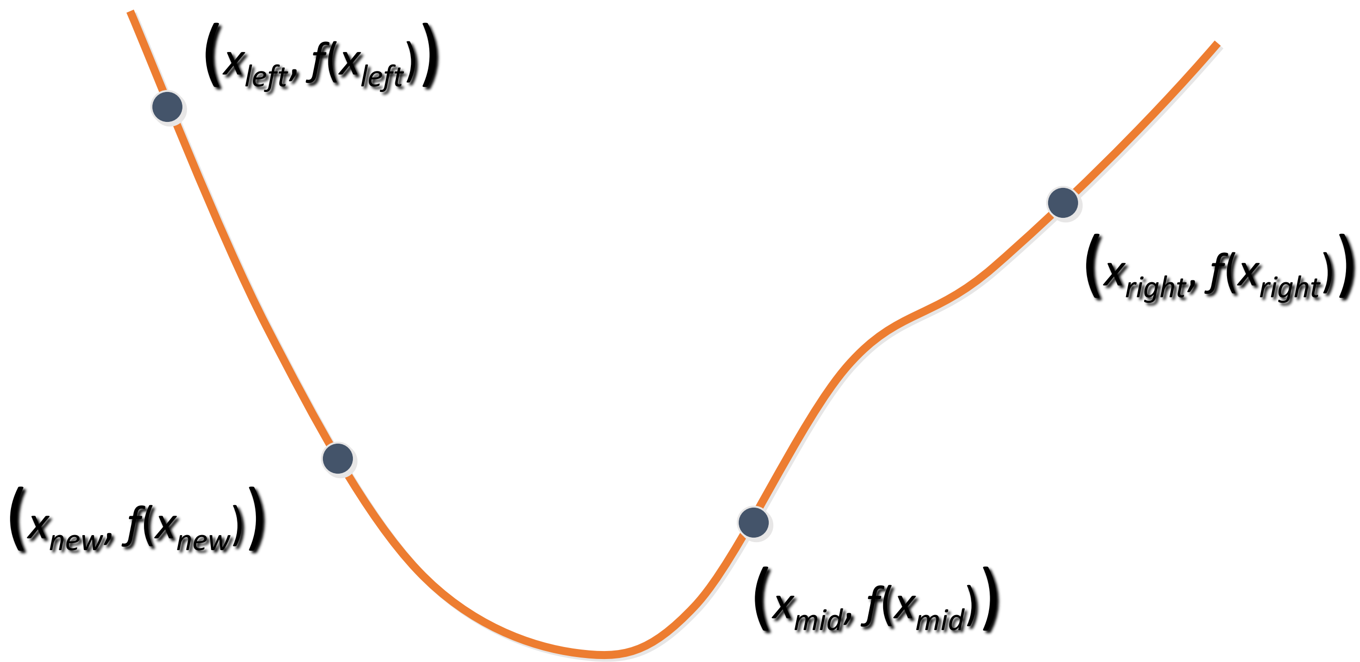
Optimization in 1-D

- Strategy: evaluate function at some x_{new}



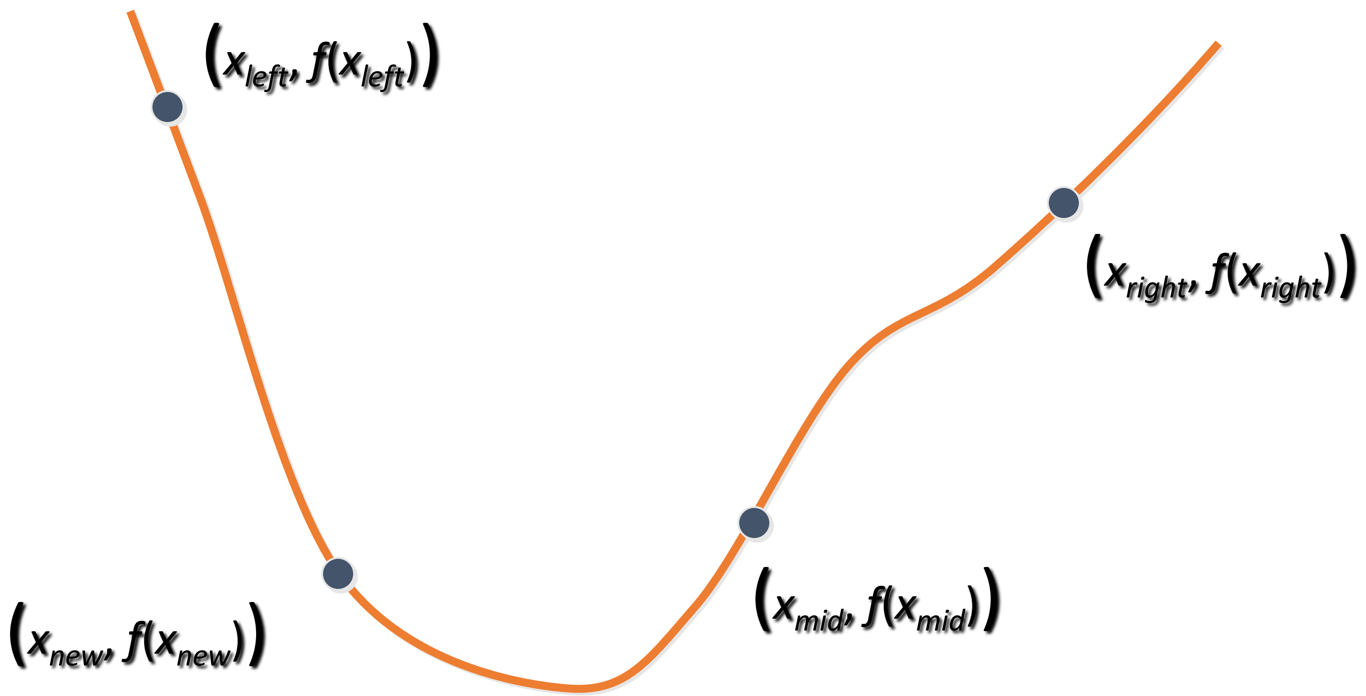
Optimization in 1-D

- Strategy: evaluate function at some x_{new}
  - Here, new “bracket” points are x_{new}, x_{mid}, x_{right}



Optimization in 1-D

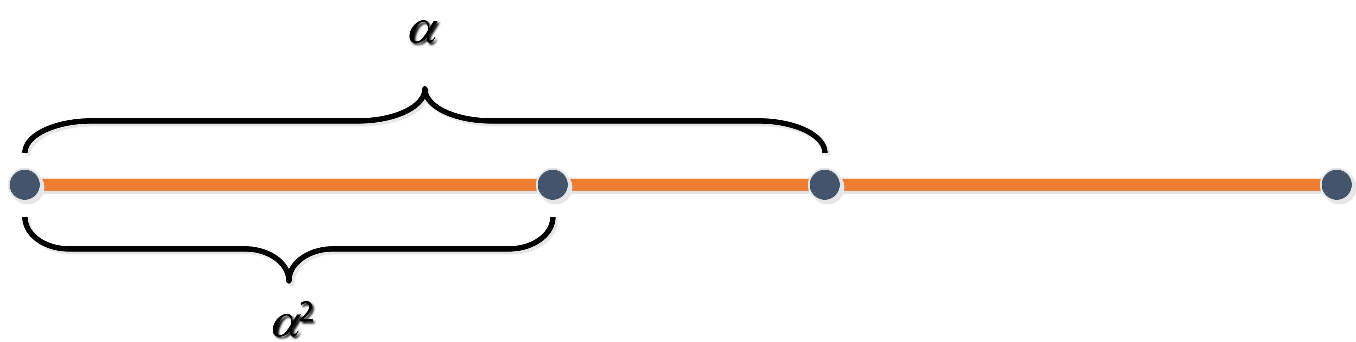
- Strategy: evaluate function at some x_{new}
  - Here, new "bracket" points are x_{left}, x_{new}, x_{mid}



Optimization in 1-D

- Unlike with root-finding, can't always guarantee that interval will be reduced by a factor of 2
- Let's find the optimal place for x_{mid}, relative to left and right, that will guarantee same factor of reduction regardless of outcome

Optimization in 1-D



if $f(x_{new}) < f(x_{mid})$

new interval = α

else

new interval = $1-\alpha^2$

Golden Section Search

- To assure same interval, want $\alpha = 1-\alpha^2$
- So,

$$\alpha = \frac{\sqrt{5}-1}{2} = \overline{\varphi}$$

- This is the “golden ratio” = 0.618…
- So, interval decreases by 30% per iteration
  - *Linear convergence*

Error Tolerance

- Around minimum, derivative = 0, so

$$f(x+\Delta x) = f(x) + \tfrac{1}{2} f''(x)\Delta x^2 + \ldots$$

$$f(x+\Delta x) - f(x) = \tfrac{1}{2} f''(x)\Delta x^2 = \text{machine } \varepsilon$$

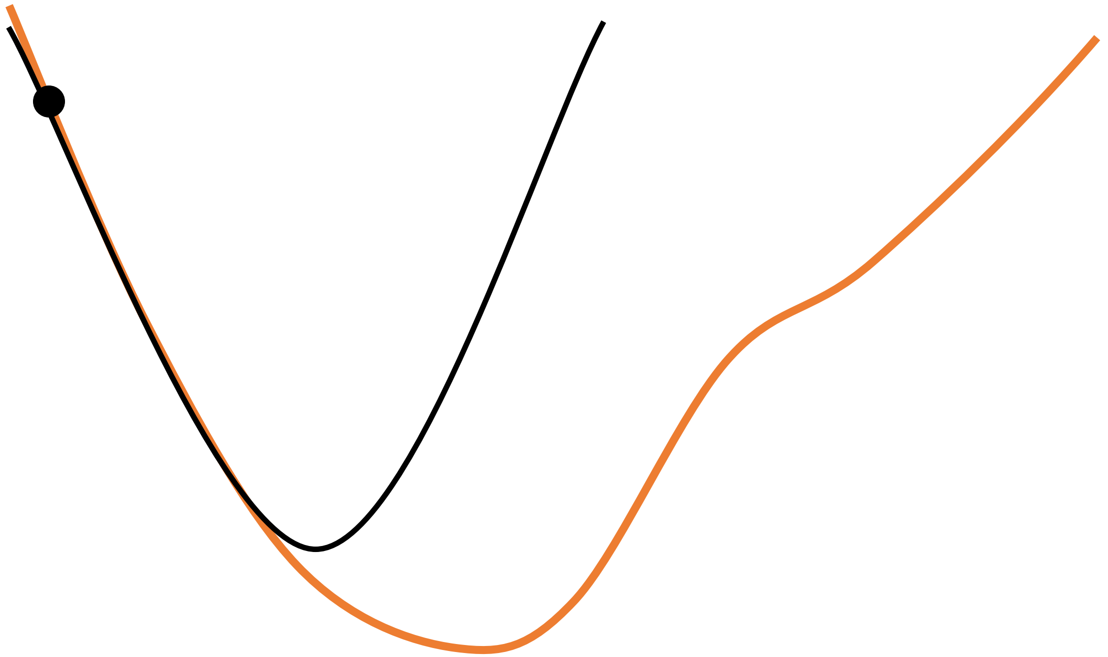
$$\Rightarrow \Delta x \sim \sqrt{\varepsilon}$$

- Rule of thumb: pointless to ask for more accuracy than sqrt(ε)
  - Can use double precision if you want a single-precision result (and/or have single-precision data)

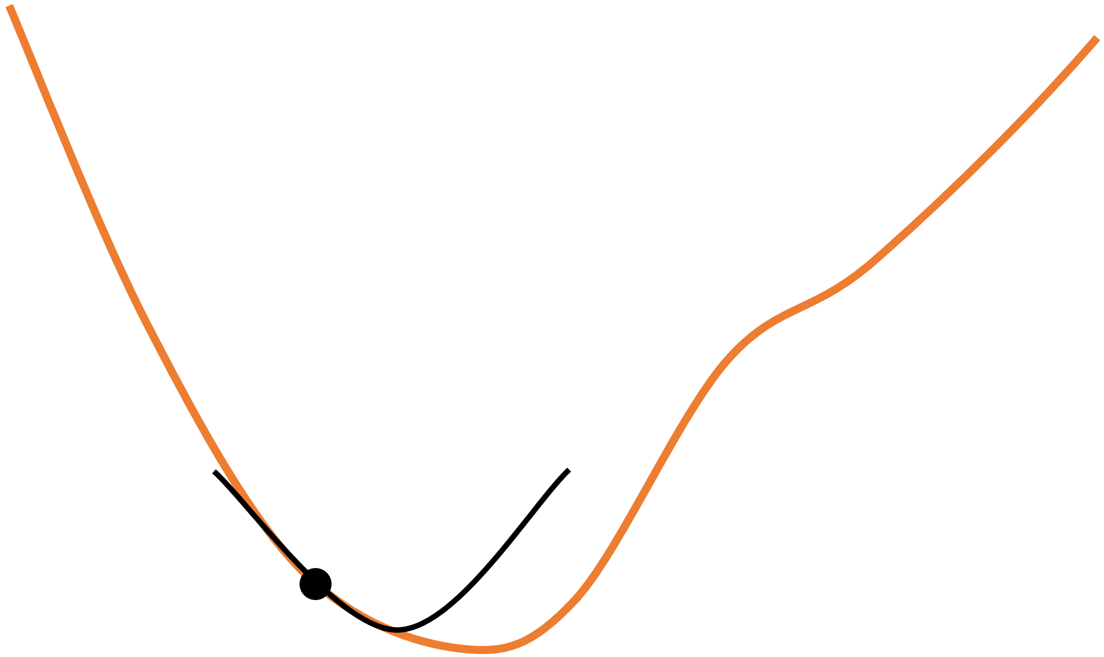
Faster 1-D Optimization

- Trade off super-linear convergence for worse robustness
  - Combine with Golden Section search for safety
- Usual bag of tricks:
  - Fit parabola through 3 points, find minimum
  - Compute derivatives as well as positions, fit cubic
  - Use *second* derivatives: Newton

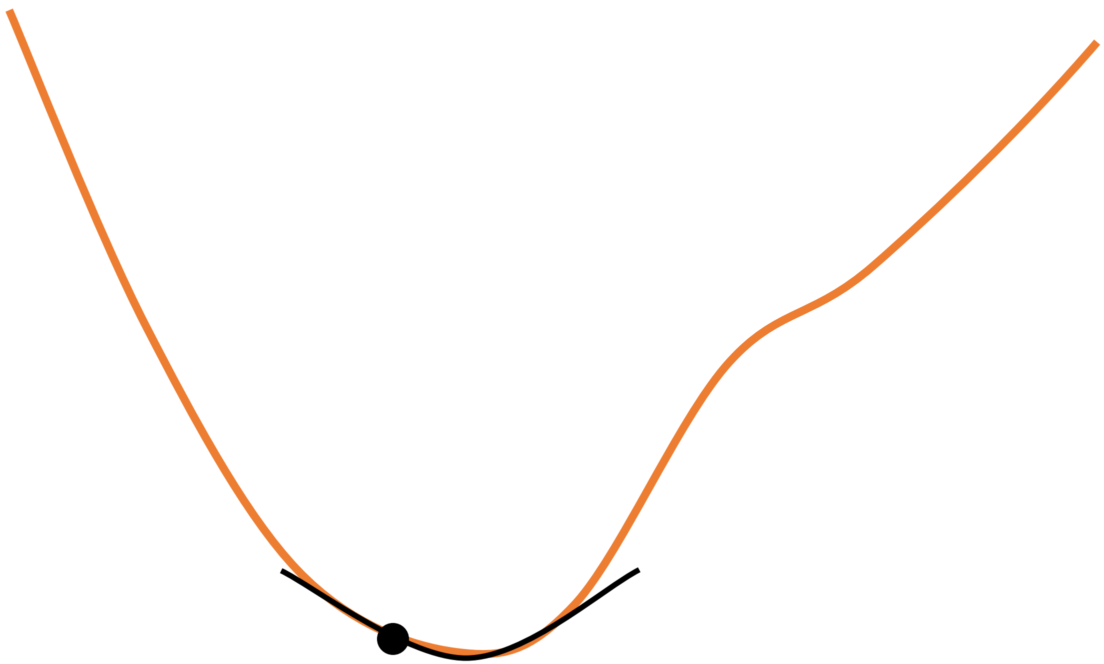
Newton’s Method



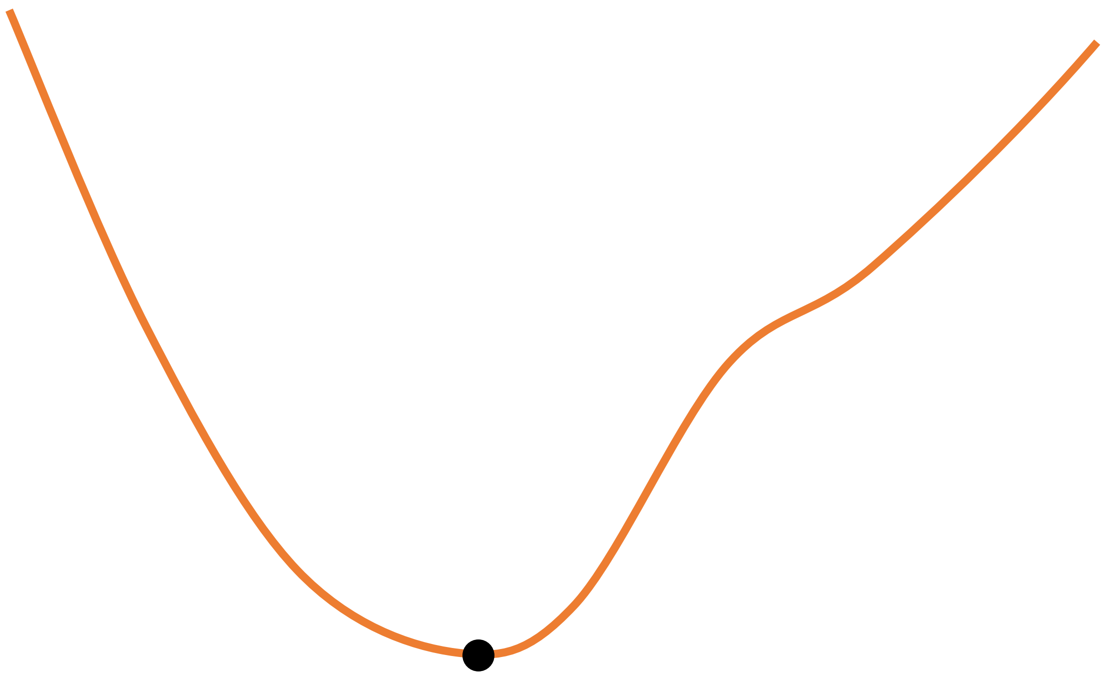
Newton's Method



Newton's Method



Newton’s Method



Newton's Method

- At each step:

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

- Requires 1st and 2nd derivatives
- Quadratic convergence

The second-order Taylor expansion of f around x_k is

$$f(x_k + t) \approx f(x_k) + f'(x_k)t + \frac{1}{2}f''(x_k)t^2.$$

The next iterate x_{k+1} is defined so as to minimize this quadratic approximation in t, and setting $x_{k+1} = x_k + t$. If the second derivative is positive, the quadratic approximation is a convex function of t, and its minimum can be found by setting the derivative to zero. Since

$$0 = \frac{\mathrm{d}}{\mathrm{d}t}\left(f(x_k) + f'(x_k)t + \frac{1}{2}f''(x_k)t^2\right) = f'(x_k) + f''(x_k)t,$$

the minimum is achieved for

$$t = -\frac{f'(x_k)}{f''(x_k)}.$$

Putting everything together, Newton's method performs the iteration

$$x_{k+1} = x_k + t = x_k - \frac{f'(x_k)}{f''(x_k)}.$$

Multi-Dimensional Optimization

- Important in many areas
  - Fitting a model to measured data
  - Finding best design in some parameter space
- Hard in general
  - Weird shapes: multiple extrema, saddles, curved or elongated valleys, etc.
  - Can't bracket
- In general, easier than rootfinding
  - Can always walk "downhill"

Newton's Method in Multiple Dimensions

- Replace 1st derivative with gradient, 2nd derivative with Hessian

$$f(x, y)$$

$$\nabla f = \begin{pmatrix} \frac{\partial f}{\partial x} \\ \frac{\partial f}{\partial y} \end{pmatrix}$$

$$H = \begin{pmatrix} \frac{\partial^2 f}{\partial x^2} & \frac{\partial^2 f}{\partial x \partial y} \\ \frac{\partial^2 f}{\partial x \partial y} & \frac{\partial^2 f}{\partial y^2} \end{pmatrix}$$

Newton's Method in Multiple Dimensions

- Replace 1st derivative with gradient, 2nd derivative with Hessian
- So,

$$\vec{x}_{k+1} = \vec{x}_k - H^{-1}(\vec{x}_k)\nabla f(\vec{x}_k)$$

- Tends to be extremely fragile unless function very smooth and starting close to minimum

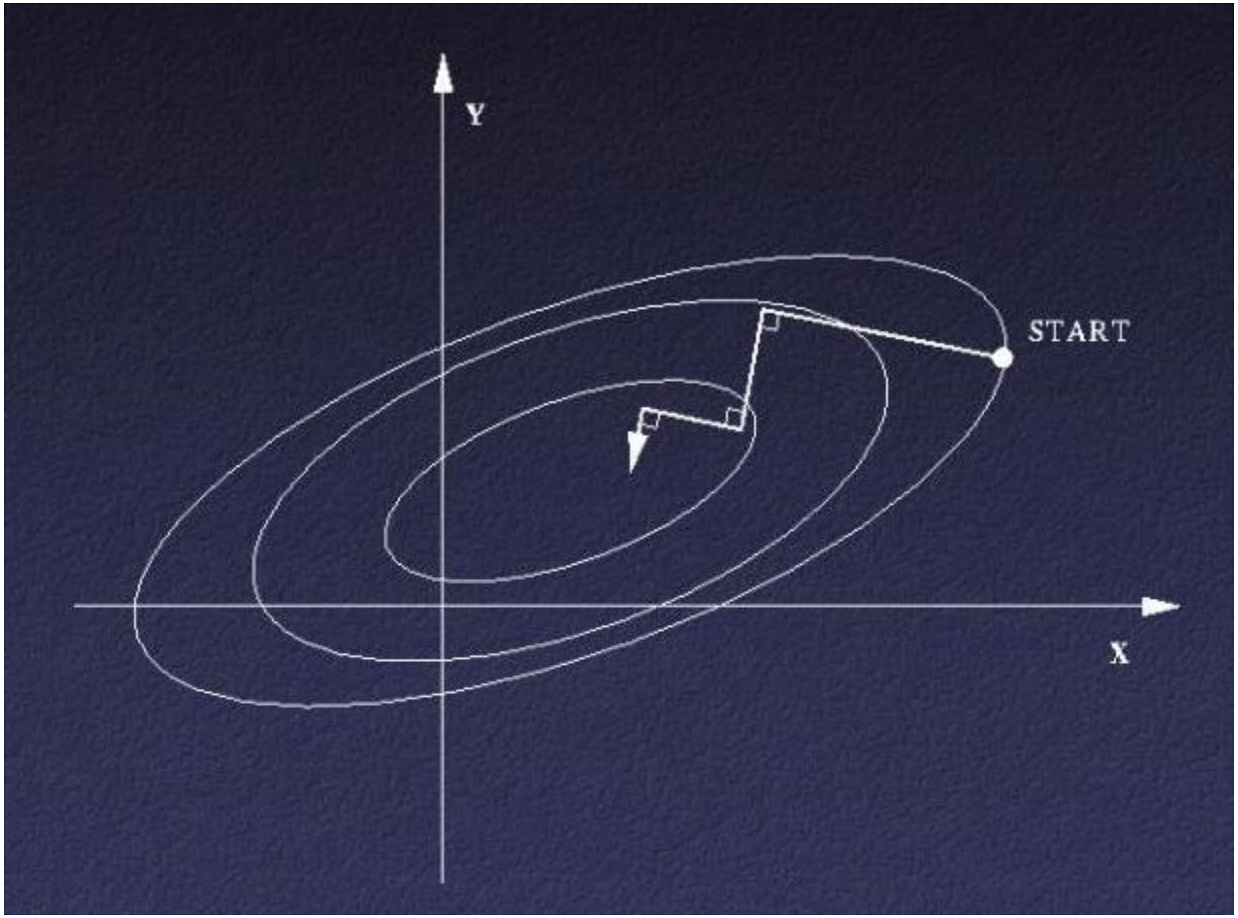
Important classification of methods

- Use function + gradient + Hessian (Newton)
- Use function + gradient (most/steepest descent methods)
- Use function values only (Nelder-Mead, called also “simplex”, or “amoeba” method)

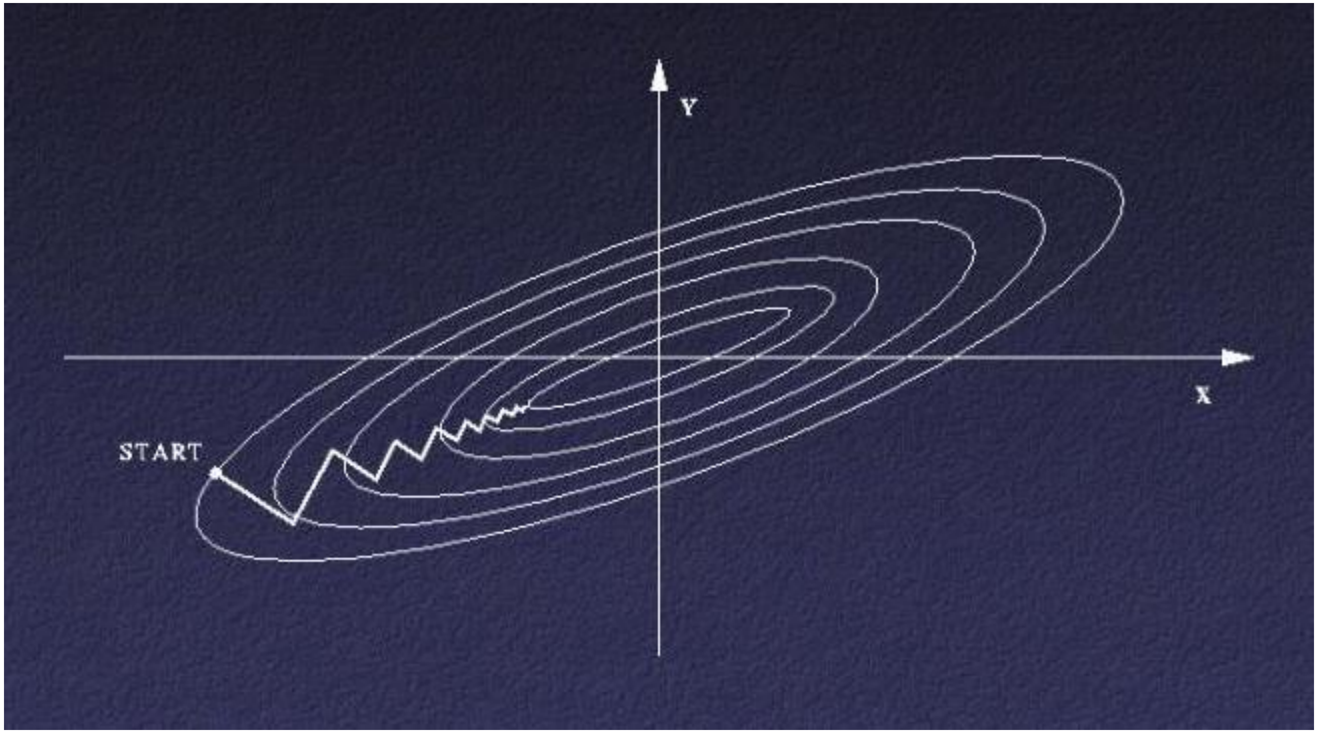
Steepest Descent Methods

- What if you can't / don't want to use 2nd derivative?
- "Quasi-Newton" methods estimate Hessian
- Alternative: walk along (negative of) gradient…
    - Perform 1-D minimization along line passing through current point in the direction of the gradient
    - Once done, re-compute gradient, iterate

Problem With Steepest Descent



Problem With Steepest Descent



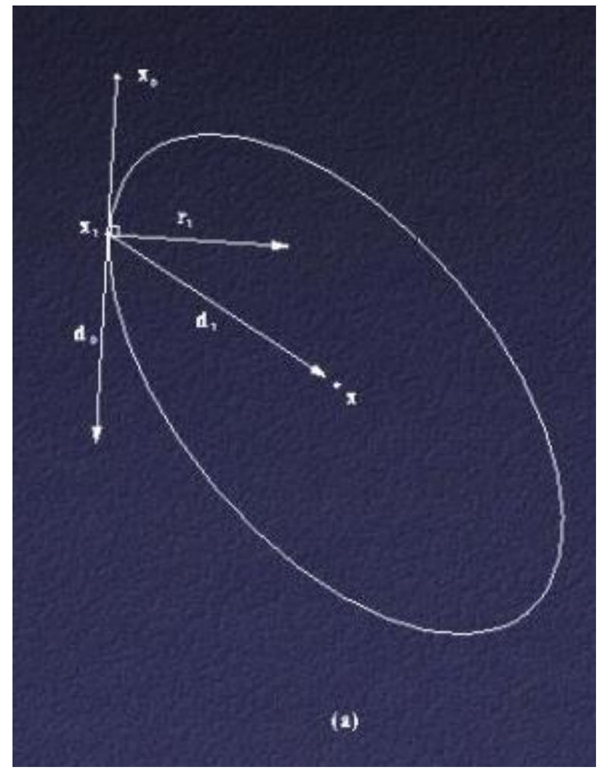
Conjugate Gradient Methods

- Idea: avoid “undoing” minimization that’s already been done
- Walk along direction

$$d_{k+1} = -g_{k+1} + \beta_k d_k$$

- Polak and Ribiere formula:

$$\beta_k = \frac{g_{k+1}^{\mathrm{T}} (g_{k+1} - g_k)}{g_k^{\mathrm{T}} g_k}$$



Conjugate Gradient Methods

- Conjugate gradient implicitly obtains information about Hessian
- For quadratic function in n dimensions, gets *exact* solution in n steps (ignoring roundoff error)
- Works well in practice…

Value-Only Methods in Multi-Dimensions

- If can't evaluate gradients, life is hard
- Can use approximate (numerically evaluated) gradients:

$$\nabla f(x) = \begin{pmatrix} \frac{\partial f}{\partial e_1} \\ \frac{\partial f}{\partial e_2} \\ \frac{\partial f}{\partial e_3} \\ \vdots \end{pmatrix} \approx \begin{pmatrix} \frac{f(x+\delta \cdot e_1) - f(x)}{\delta} \\ \frac{f(x+\delta \cdot e_2) - f(x)}{\delta} \\ \frac{f(x+\delta \cdot e_3) - f(x)}{\delta} \\ \vdots \end{pmatrix}$$

Generic Optimization Strategies

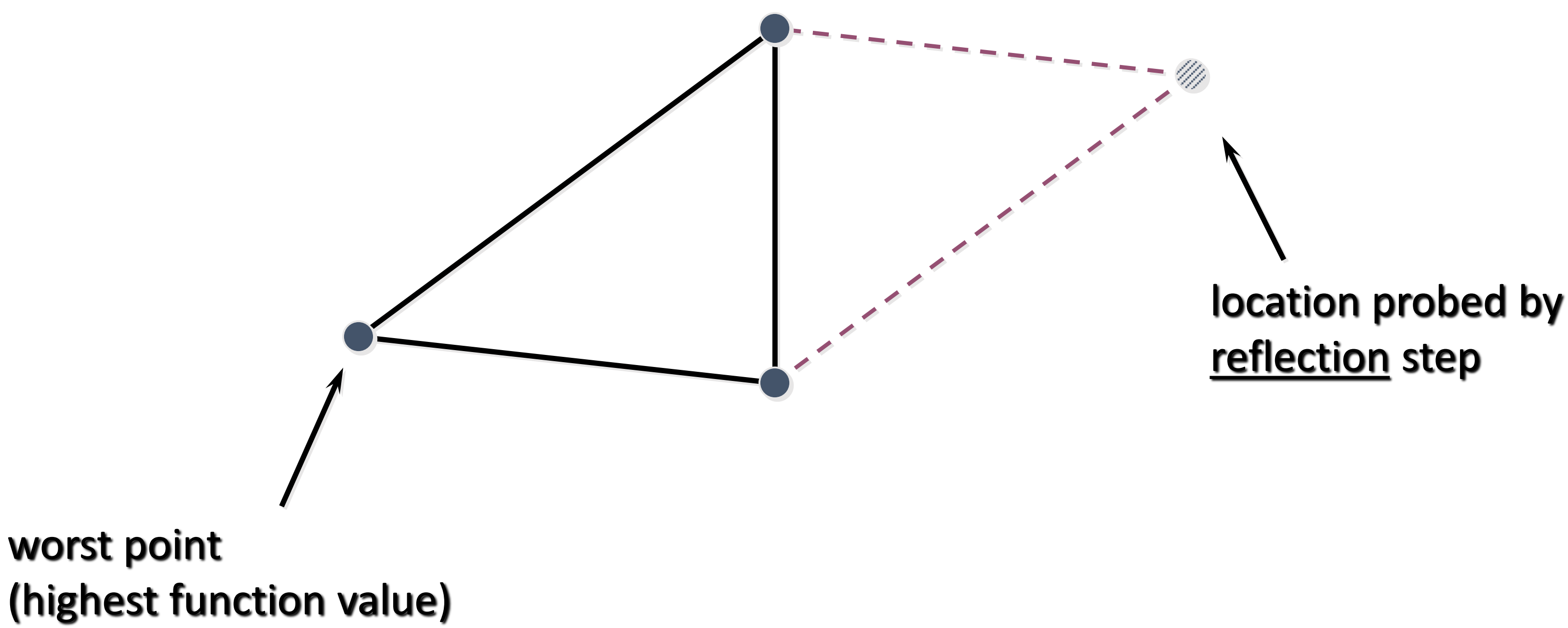
- Uniform sampling:
  - Cost rises exponentially with # of dimensions
- Simulated annealing:
  - Search in random directions
  - Start with large steps, gradually decrease
  - “Annealing schedule” – how fast to cool?

Downhill Simplex Method (Nelder-Mead)

- Keep track of *n*+1 points in *n* dimensions
  - Vertices of a *simplex* (triangle in 2D tetrahedron in 3D, etc.)
- At each iteration: simplex can move, expand, or contract
  - Sometimes known as *amoeba method*: simplex "oozes" along the function

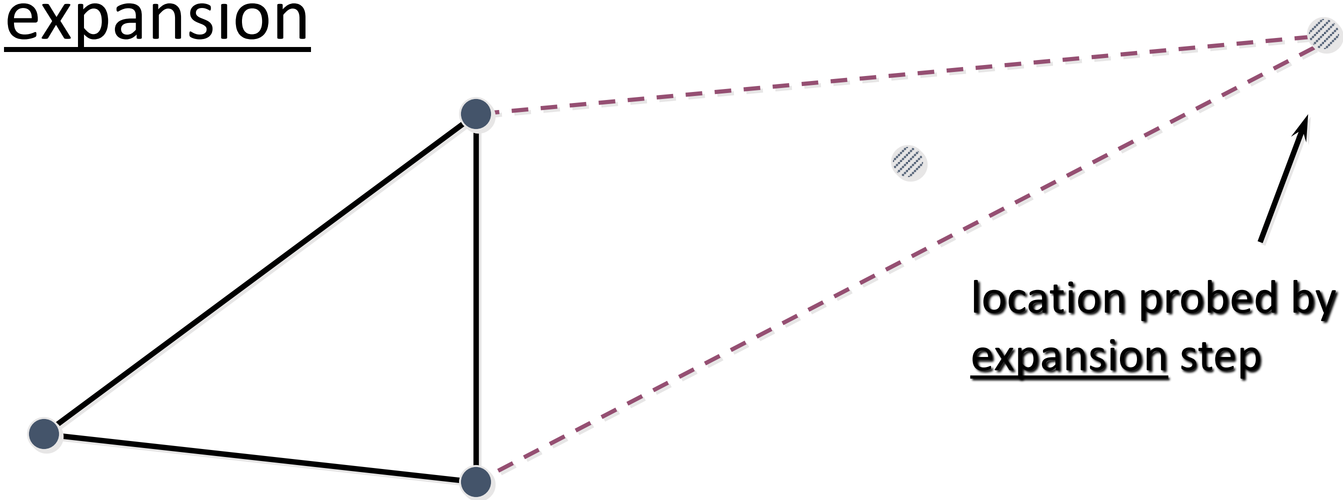
Downhill Simplex Method (Nelder-Mead)

- Basic operation: reflection



Downhill Simplex Method (Nelder-Mead)

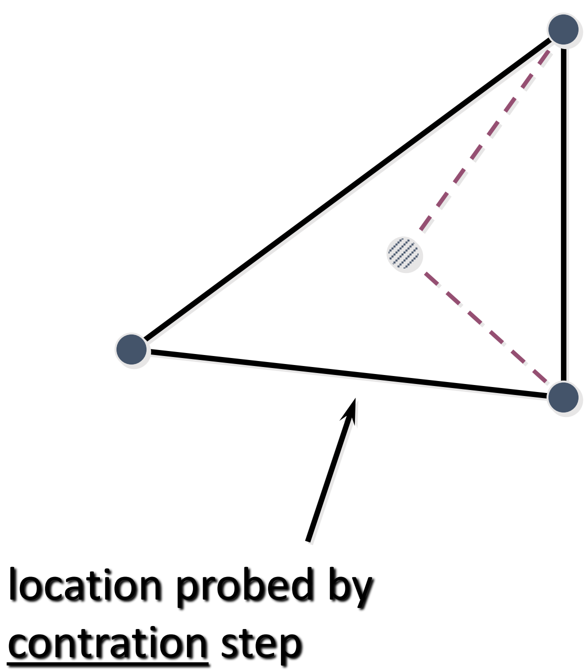
- If reflection resulted in best (lowest) value so far, try an <u>expansion</u>



- Else, if reflection helped at all, keep it

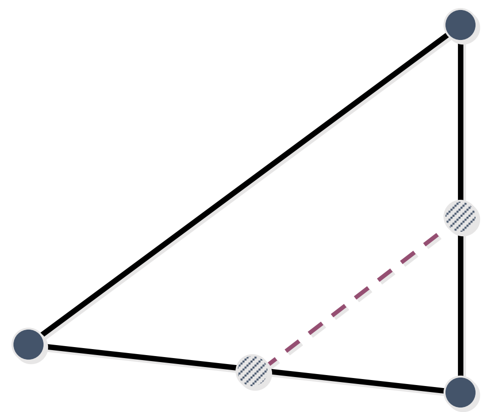
Downhill Simplex Method (Nelder-Mead)

- If reflection didn't help (reflected point still worst) try a <u>contraction</u>



Downhill Simplex Method (Nelder-Mead)

- If all else fails <u>shrink</u> the simplex around the *best* point



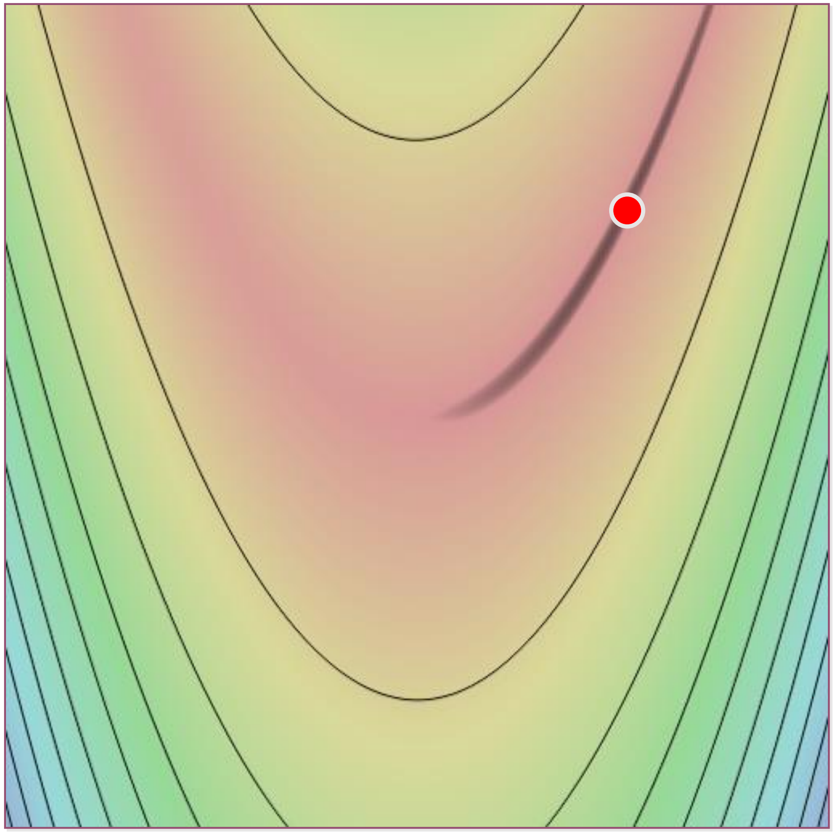
Downhill Simplex Method (Nelder-Mead)

- Method fairly efficient at each iteration (typically 1-2 function evaluations)
- Can take *lots* of iterations
- Somewhat flakey – sometimes needs *restart* after simplex collapses on itself, etc.
- Benefits: simple to implement, doesn’t need derivative, doesn’t care about function smoothness, etc.

Rosenbrock’s Function

- Designed specifically for testing optimization techniques
- Curved, narrow valley

$$f(x,y) = 100(y - x^2)^2 + (1 - x)^2$$



Constrained Optimization

- Equality constraints: optimize *f(x)* subject to $g_i(x)=0$
- Method of Lagrange multipliers: convert to a higher-dimensional problem
- Minimize $f(x)+\sum \lambda_i g_i(x)$ w.r.t. $(x_1 \ldots x_n; \lambda_1 \ldots \lambda_k)$

Constrained Optimization

- Inequality constraints are harder…
- If objective function and constraints all linear, this is "linear programming"
- Observation: minimum must lie at corner of region formed by constraints
- Simplex method: move from vertex to vertex, minimizing objective function

Constrained Optimization

- General “nonlinear programming” hard
- Algorithms for special cases (e.g. quadratic)

Global Optimization

- In general, can’t guarantee that you’ve found global (rather than local) minimum
- Some heuristics:
  - Multi-start: try local optimization from several starting positions
  - Very slow simulated annealing
  - Use analytical methods (or graphing) to determine behavior, guide methods to correct neighborhoods

Thank you for your attention