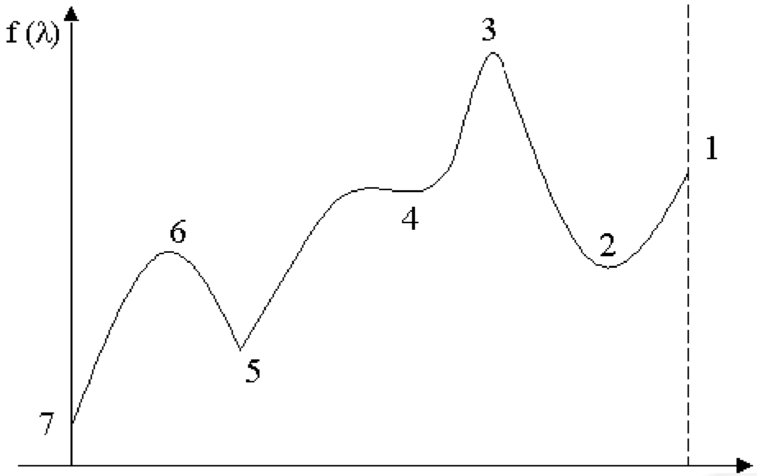
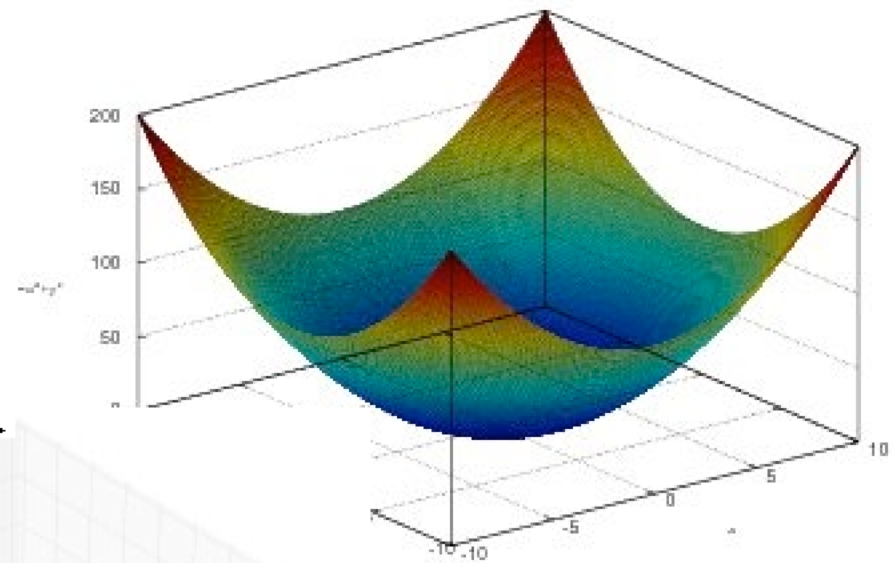
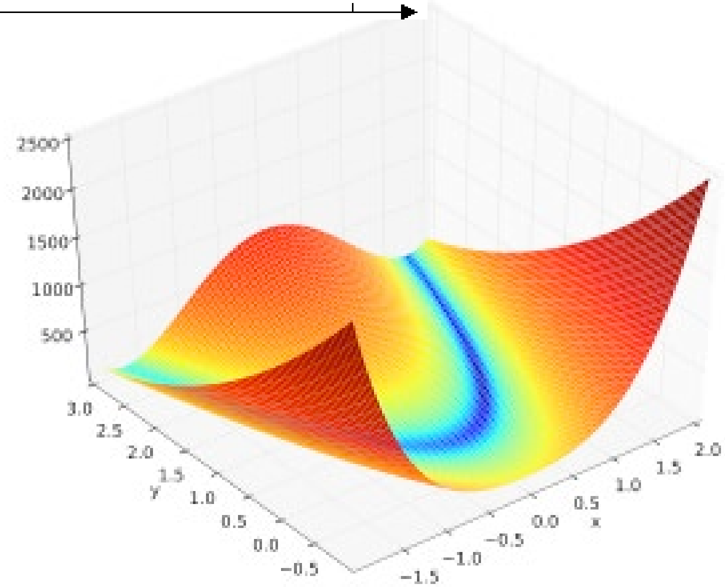
Optimization







Last time

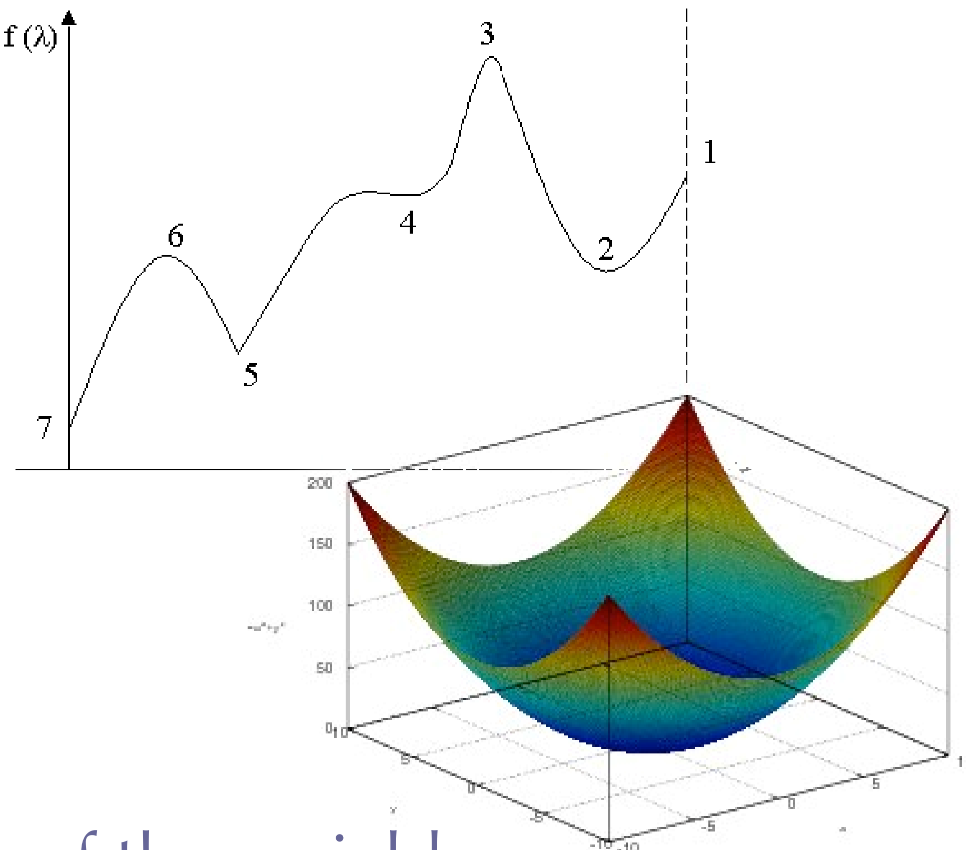
- Root finding: definition, motivation
- Algorithms: Bisection, false position, secant, Newton-Raphson
- Convergence & tradeoffs
- Example applications of Newton's method
- Root finding in > 1 dimension

Today

- Introduction to optimization
- Definition and motivation
- 1-dimensional methods
  - Golden section, discussion of error
  - Newton's method
- Multi-dimensional methods
  - Newton's method, steepest descent, conjugate gradient
- General strategies, value-only methods

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

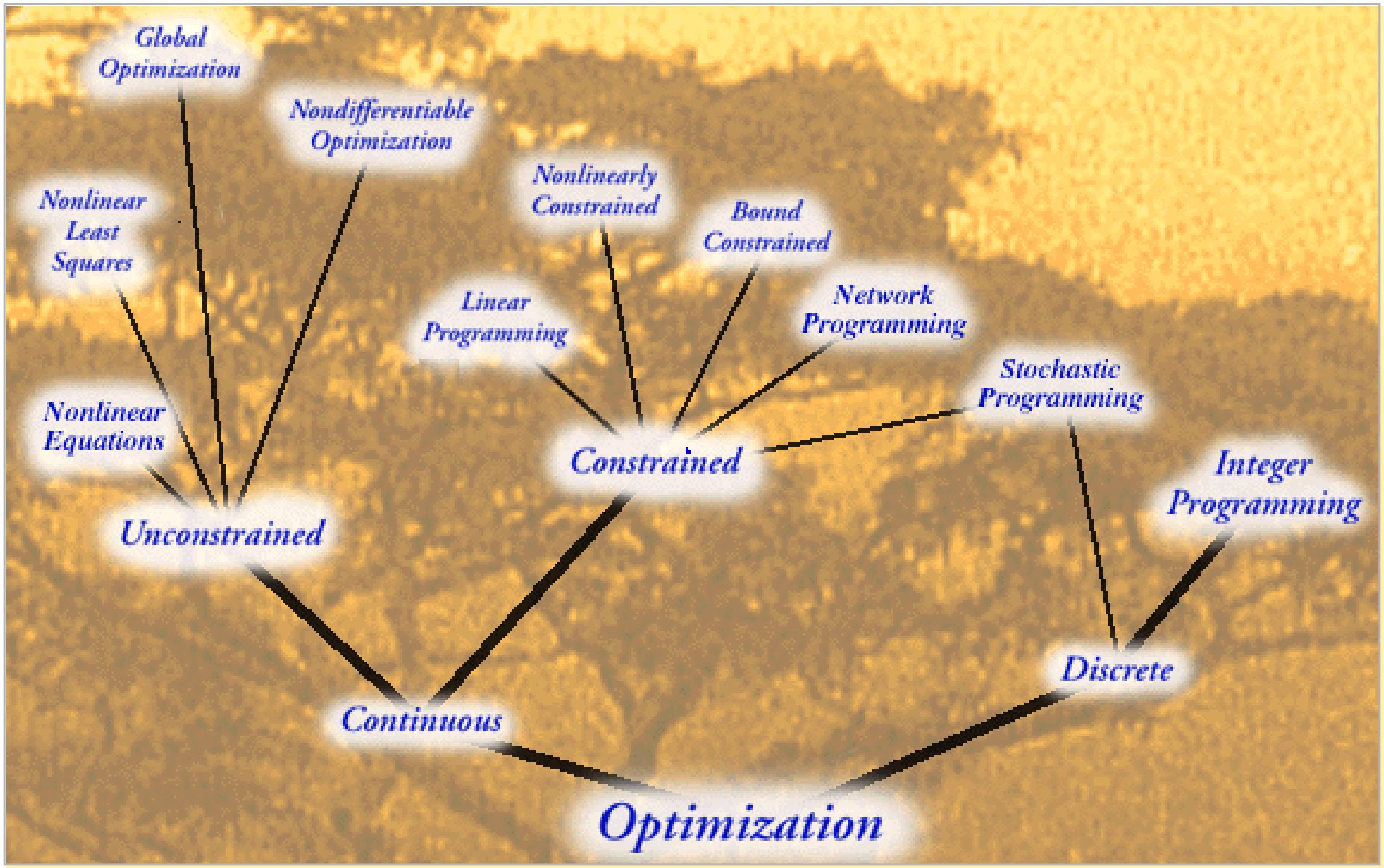


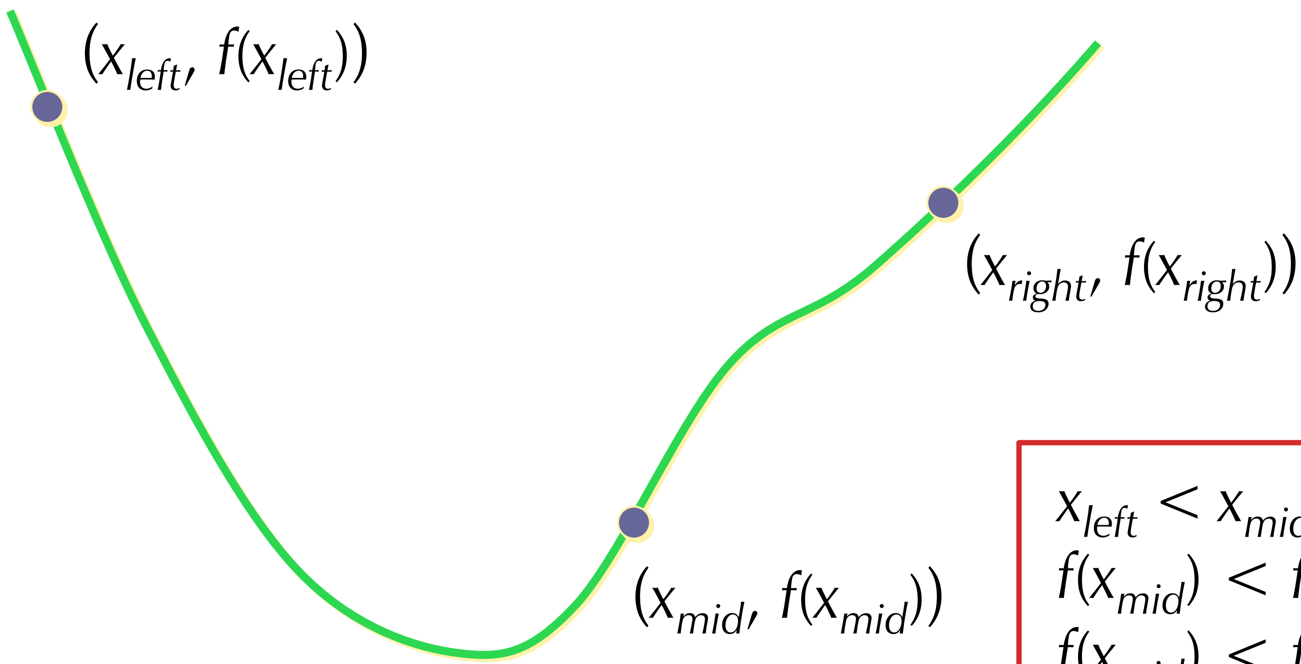
Figure from: Optimization Technology Center
http://www-fp.mcs.anl.gov/otc/Guide/OptWeb/

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?



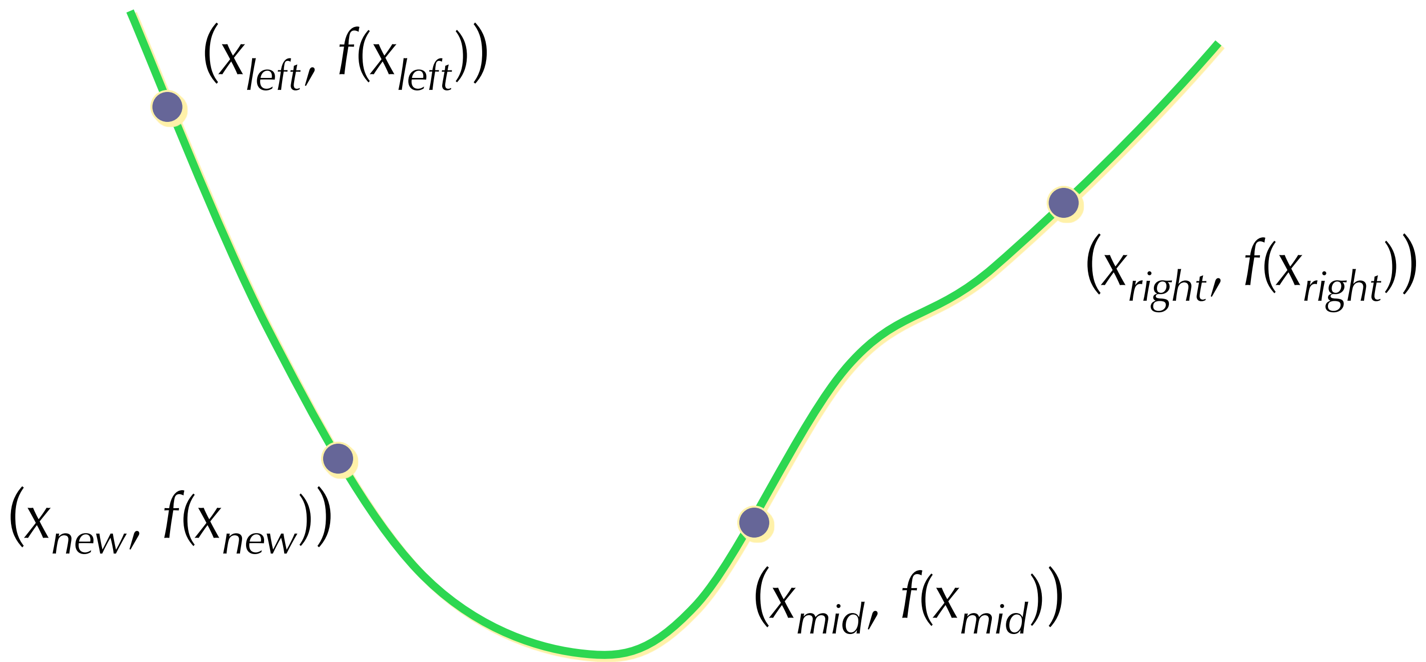
$$x_{left} < x_{mid} < x_{right}$$
$$f(x_{mid}) < f(x_{left})$$
$$f(x_{mid}) < f(x_{right})$$

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 (by a constant factor) at each step
- For maximization: substitute $-f$ for f

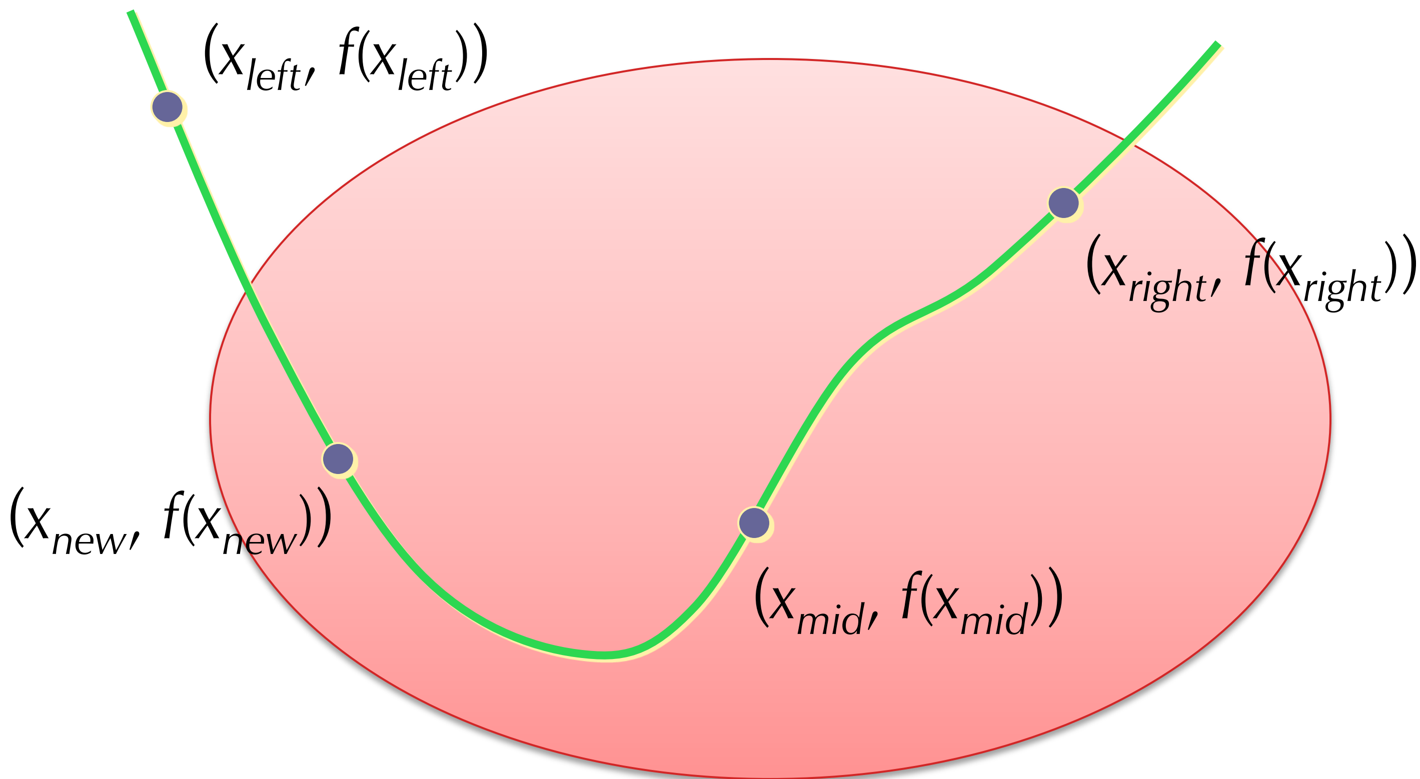
Optimization in 1-D

- Strategy: evaluate function at some x_{new}



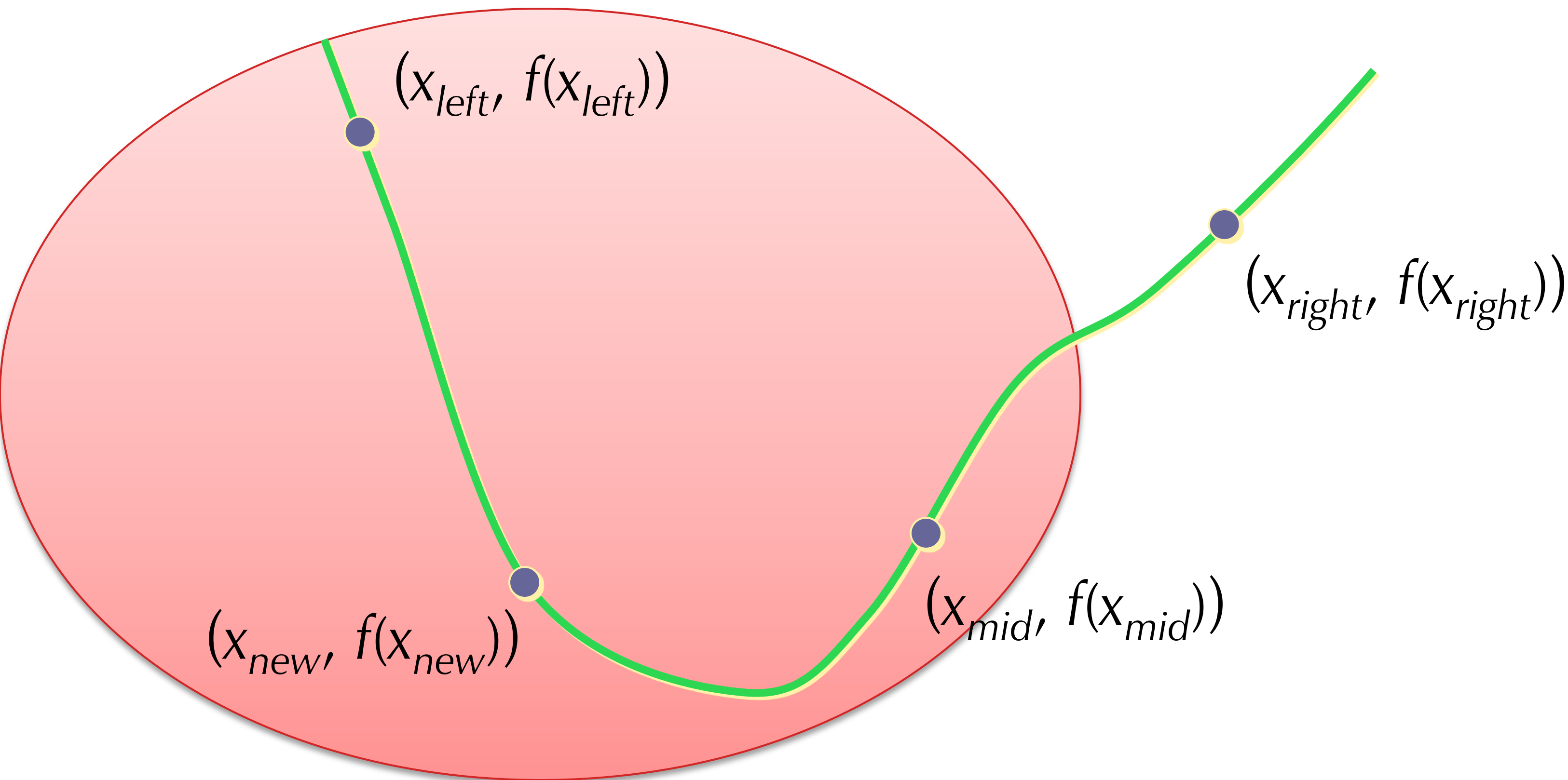
Optimization in 1-D

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



Optimization in 1-D

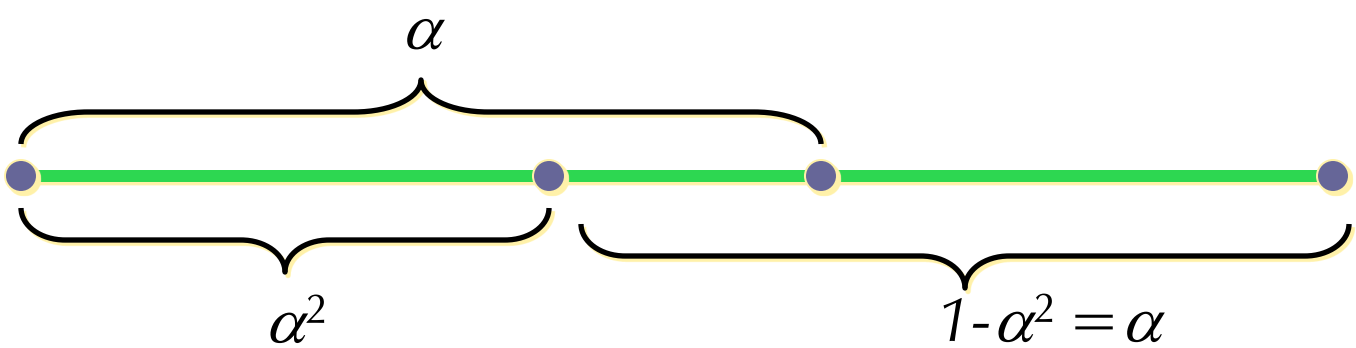
- Strategy: evaluate function at some x_{new}
  - Here, new "bracket" points are x_{left}, x_{new}, x_{mid}



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 $= \alpha$

else

new interval $= 1-\alpha^2$

Golden Section Search

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

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

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

Sources of Error

- When we "find" a minimum value, x, why is it different from true minimum?
  1. Obvious: width of bracket

$$\left|\frac{x - x_{\min}}{x_{\min}}\right| \le b - a$$

  2. Less obvious: floating point representation

$$\left|\frac{F(x_{\min}) - f(x_{\min})}{f(x_{\min})}\right| \le \varepsilon_{mach}$$

- Q: When is (b – a) **small enough** that discrepancy between x and x_{min} limited by rounding error in $f(x_{min})$?

Stopping criterion for Golden Section

- When is (b – a) **small enough** that discrepancy between x and x_{min} attributed to rounding error in $f(x_{min})$?

$$|b-a| \leq \sqrt{\frac{\epsilon}{L}} \qquad \text{where} \qquad L = \left|\frac{f''(x_m)}{2f(x_m)}\right|$$

Why? Use Taylor series, knowing $f'(x_m)$ is around 0 :

$$f(x) \approx f(x_m) + \frac{f''(x_m)}{2}(x-x_m)^2 = f(x_m)\,(1+\psi)$$

$$\text{where} \qquad \psi = \frac{f''(x_m)}{2f(x_m)}(x-x_m)^2$$

Implications

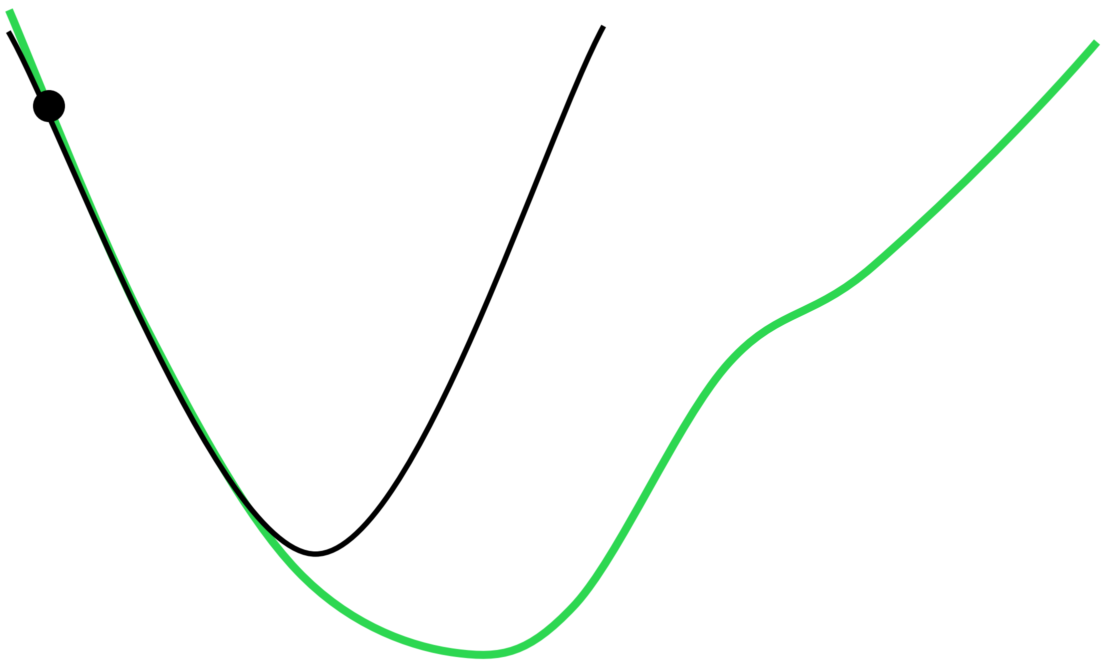
- Rule of thumb: pointless to ask for more accuracy than sqrt(ε)

- Q:, what happens to # of accurate digits in results when you switch from single precision (~7 digits) to double (~16 digits) for x, f(x)?
  - A: Gain only ~4 more accurate digits.

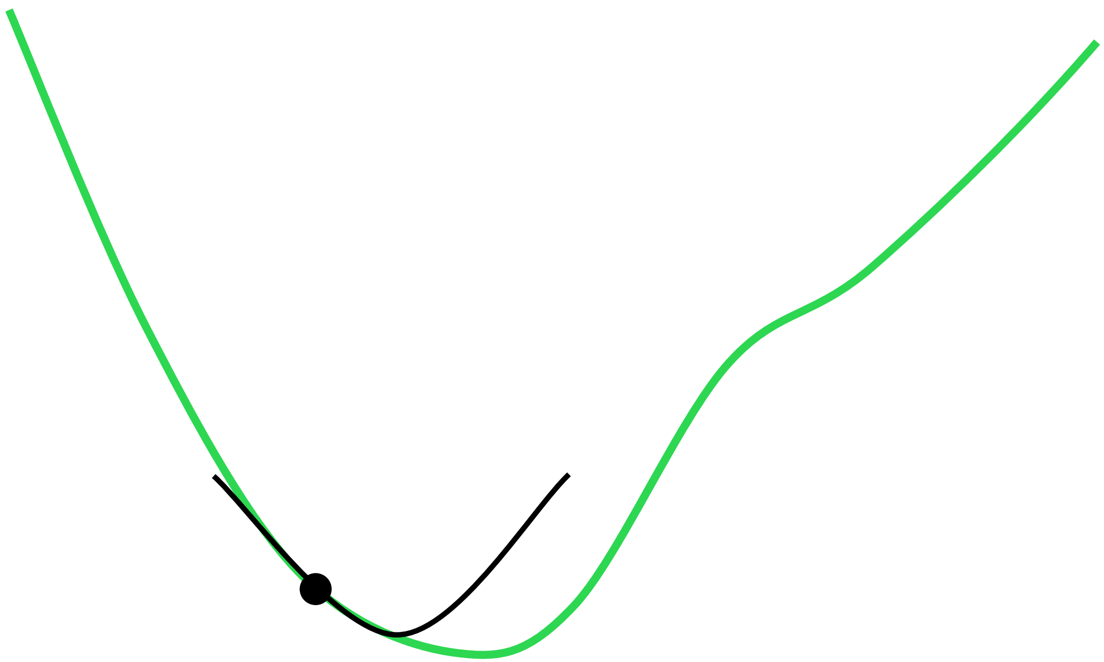
Faster 1-D Optimization

- 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

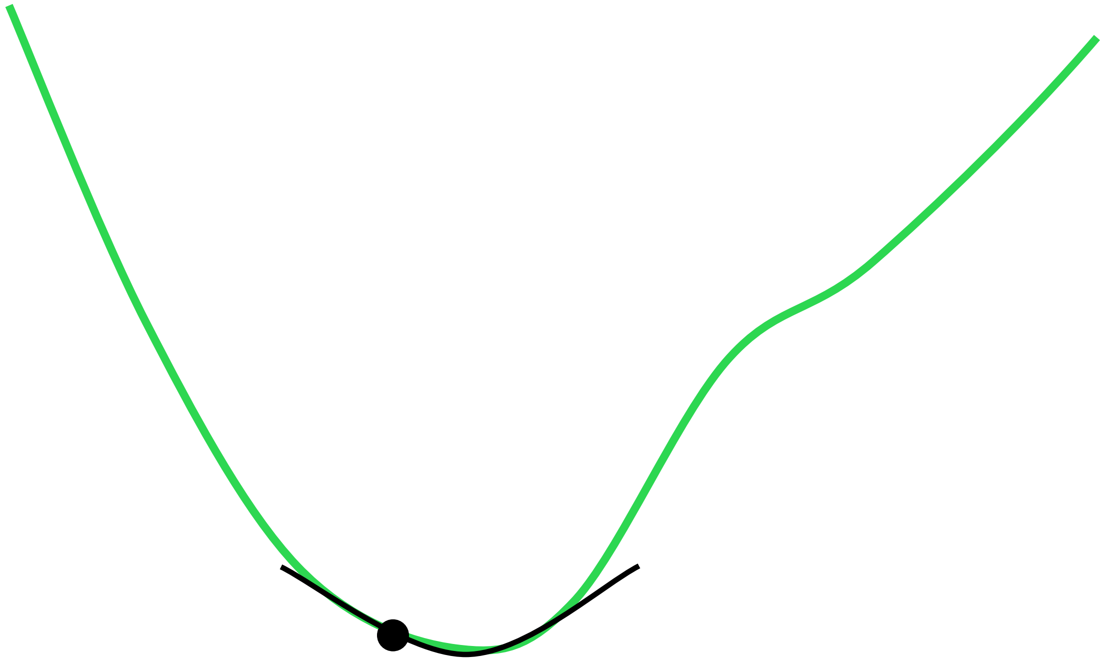
Newton's Method



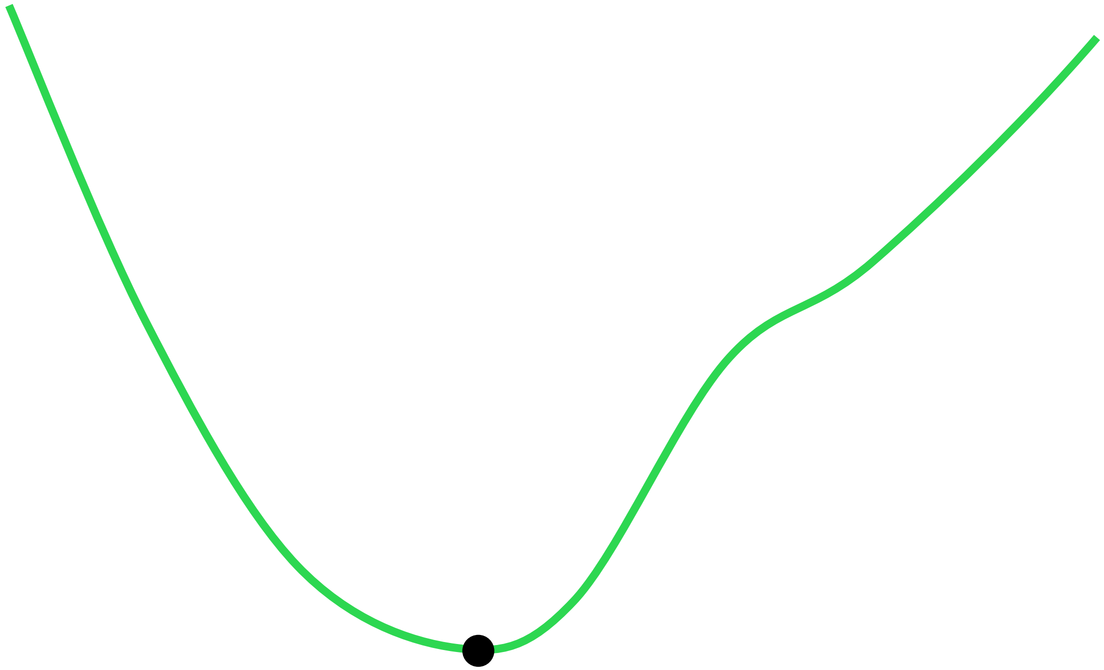
Newton's Method



Newton's Method



Newton's Method



Newton's Method

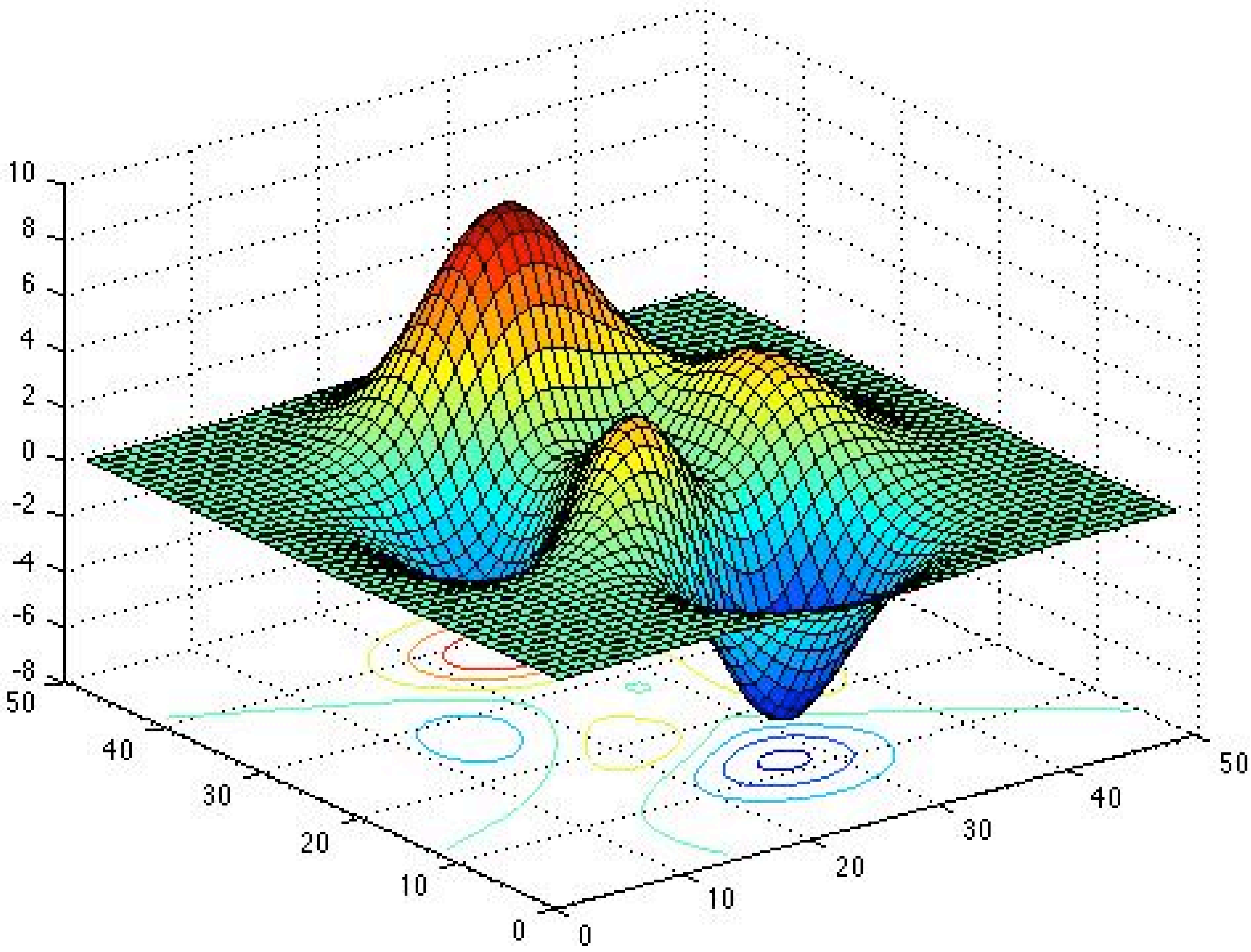
- At each step:

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

- Requires 1st and 2nd derivatives
- Quadratic convergence

Questions?

Multidimensional Optimization

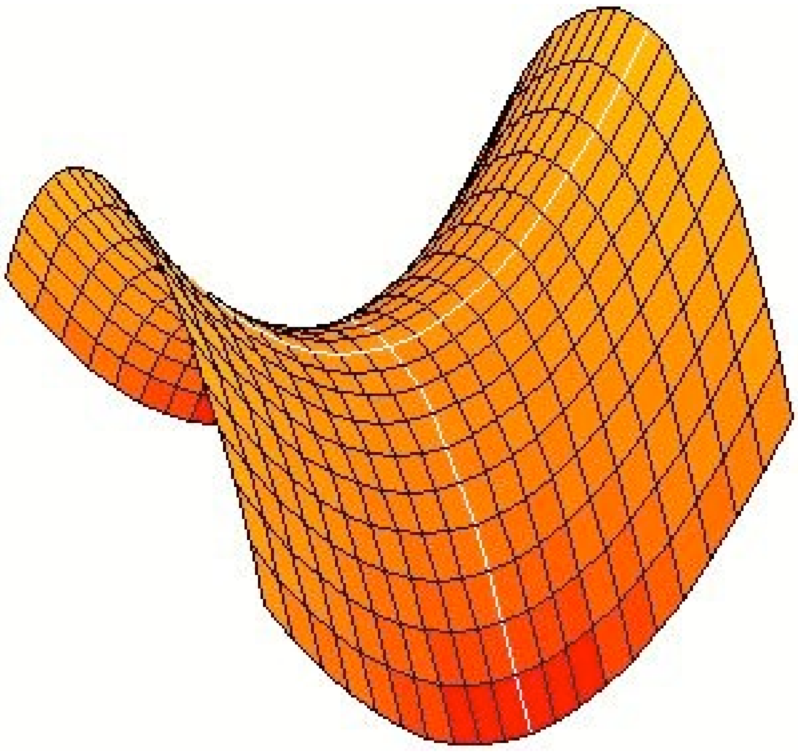


10
8
6
4
2
0
-2
-4
-6
-8
50
40
30
20
10
0
0
10
20
30
40
50

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 (but there are "trust region" methods)
- In general, easier than rootfinding
  - Can always walk "downhill"

Problem with Saddle



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

- in 1 dimension: $x_{k+1} = x_k - \frac{f'(x_k)}{f''(x_k)}$
- Replace 1st derivative with gradient, 2nd derivative with Hessian
- So,

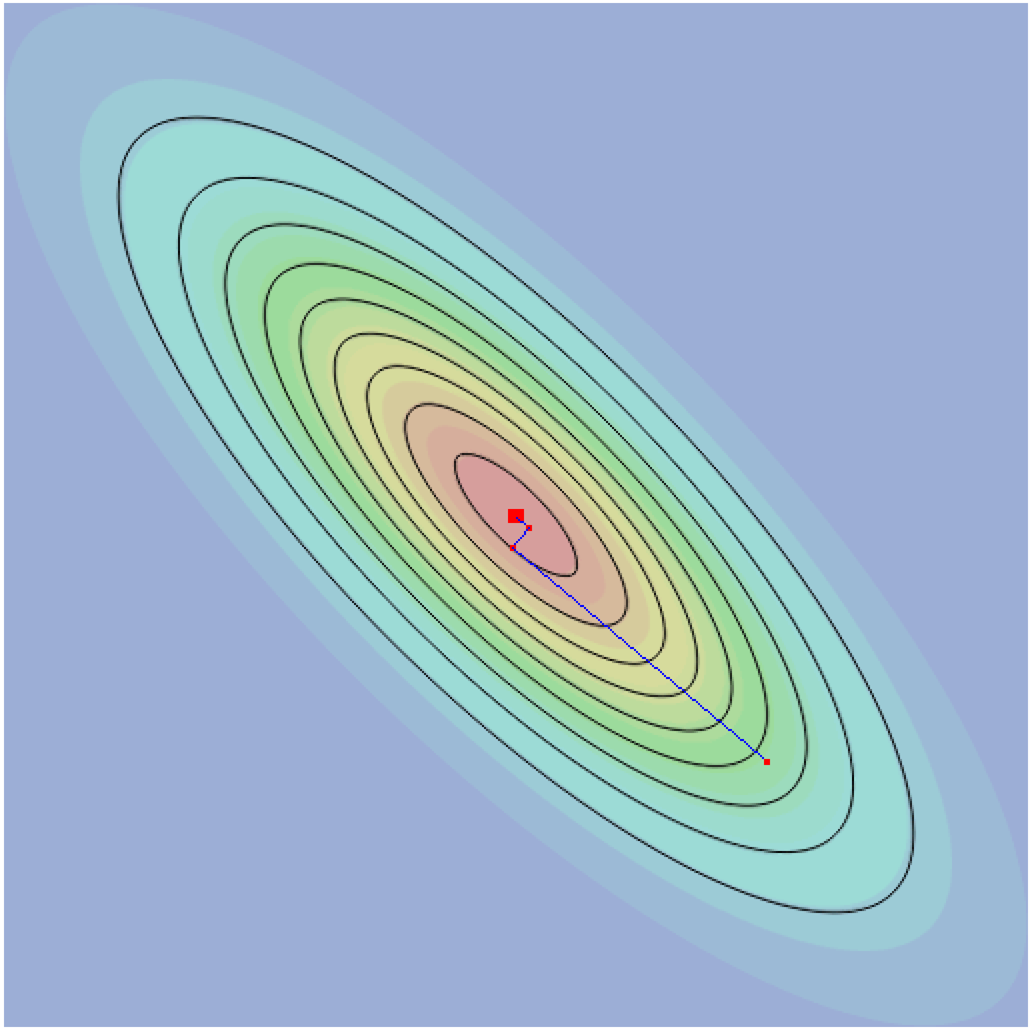
$$\vec{x}_{k+1} = \vec{x}_k - H^{-1}(\vec{x}_k)\nabla f(\vec{x}_k)$$

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

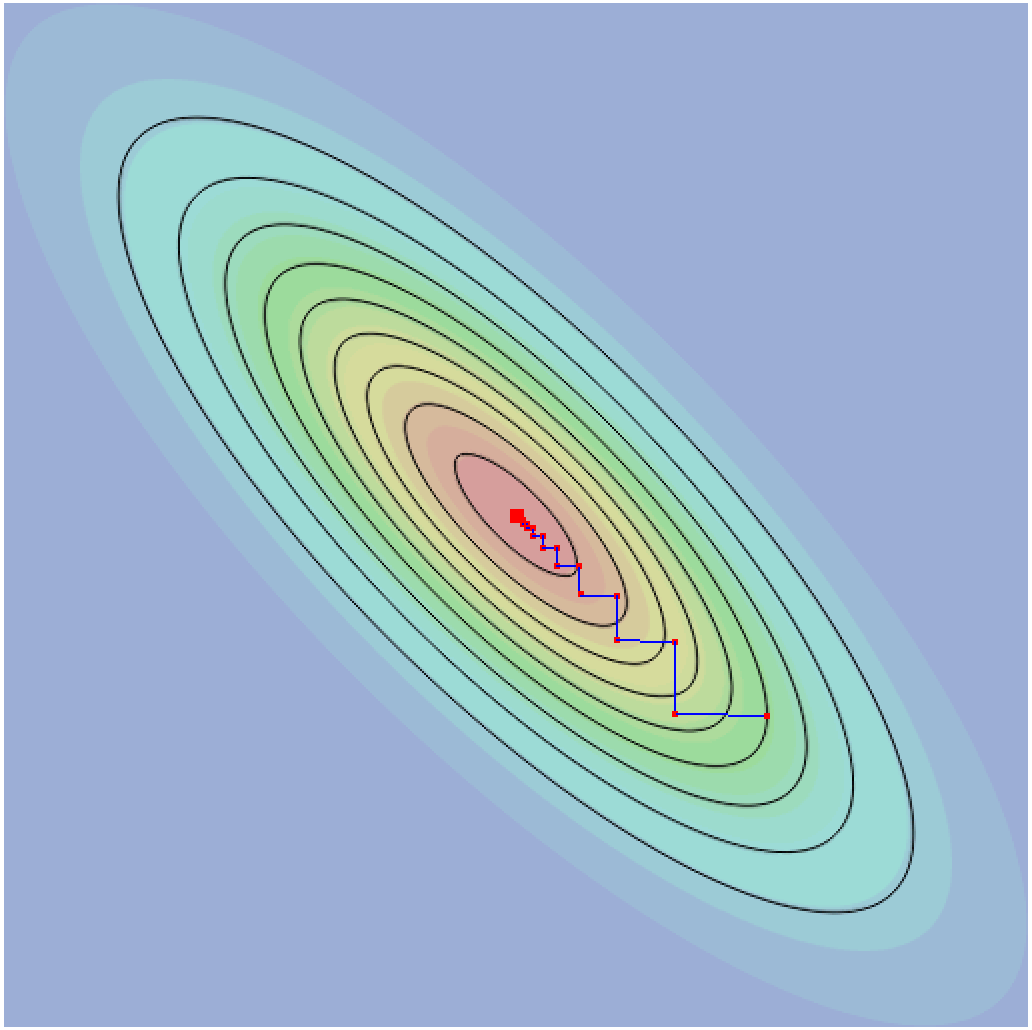
Other Methods

- What if you can't / don't want to use 2^{nd} 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

Steepest Descent



Problem With Steepest Descent



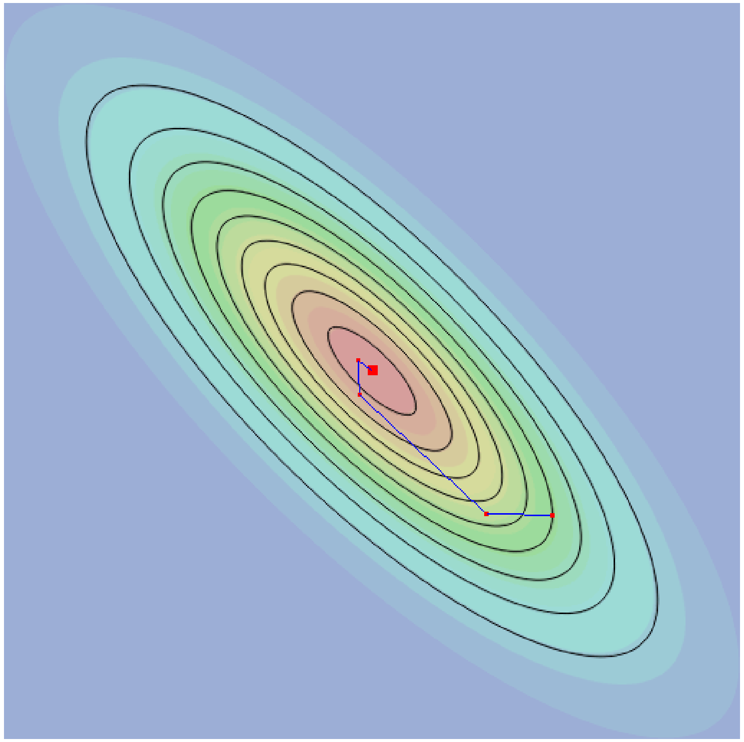
Conjugate Gradient Methods

- 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…

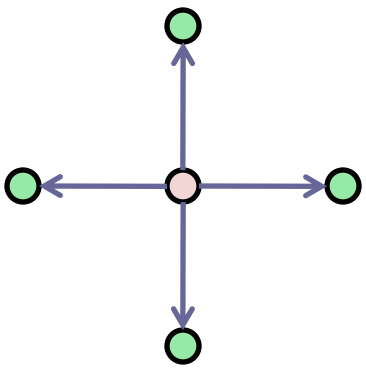
Value-Only Methods in Multi-Dimensions

- 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
- Compass search:
  - Try a step along each coordinate in turn
  - If can't find a lower value, halve step size



Generic Optimization Strategies

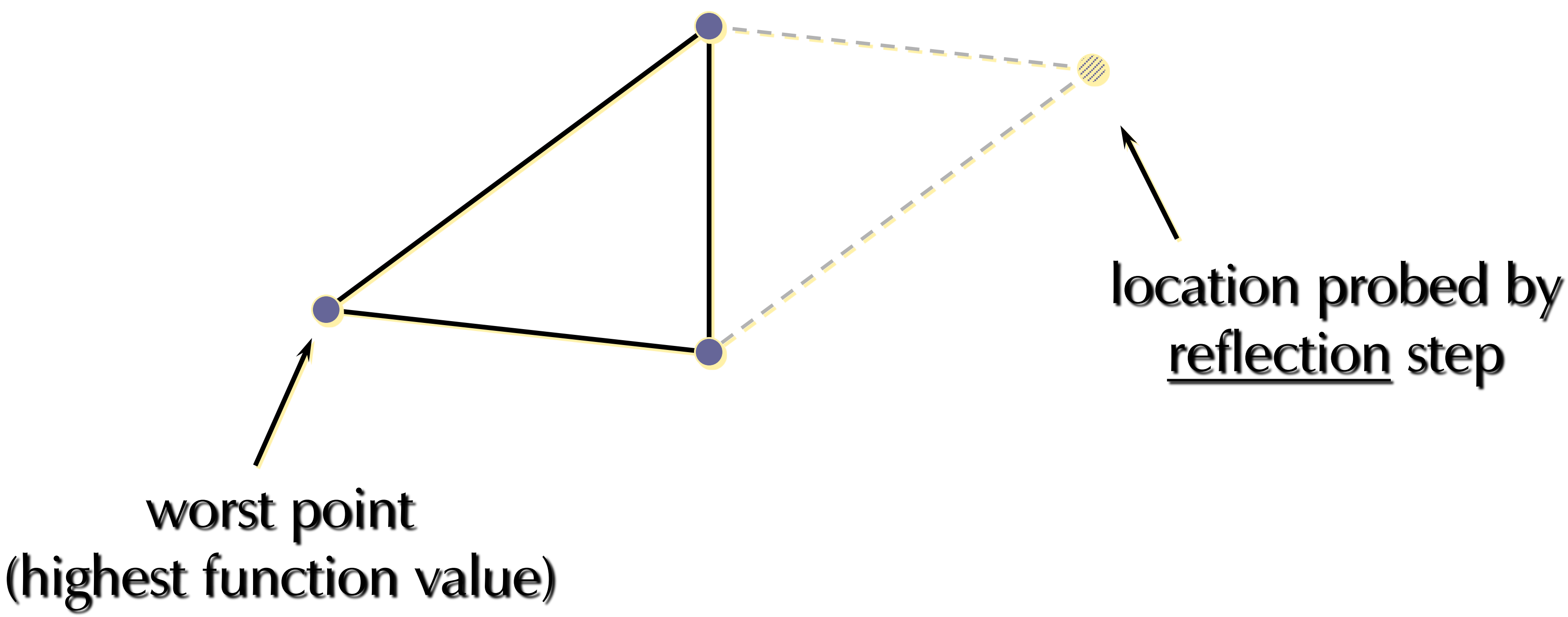
- Simulated annealing:
  - Maintain a "temperature" T
  - Pick random direction *d*, and try a step of size dependent on T
  - If value lower than current, accept
  - If value higher than current, accept with probability $\sim \exp((f(x) - f(x'))/T)$
  - "Annealing schedule" – how fast does T decrease?
- Slow but robust: can avoid non-global minima

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

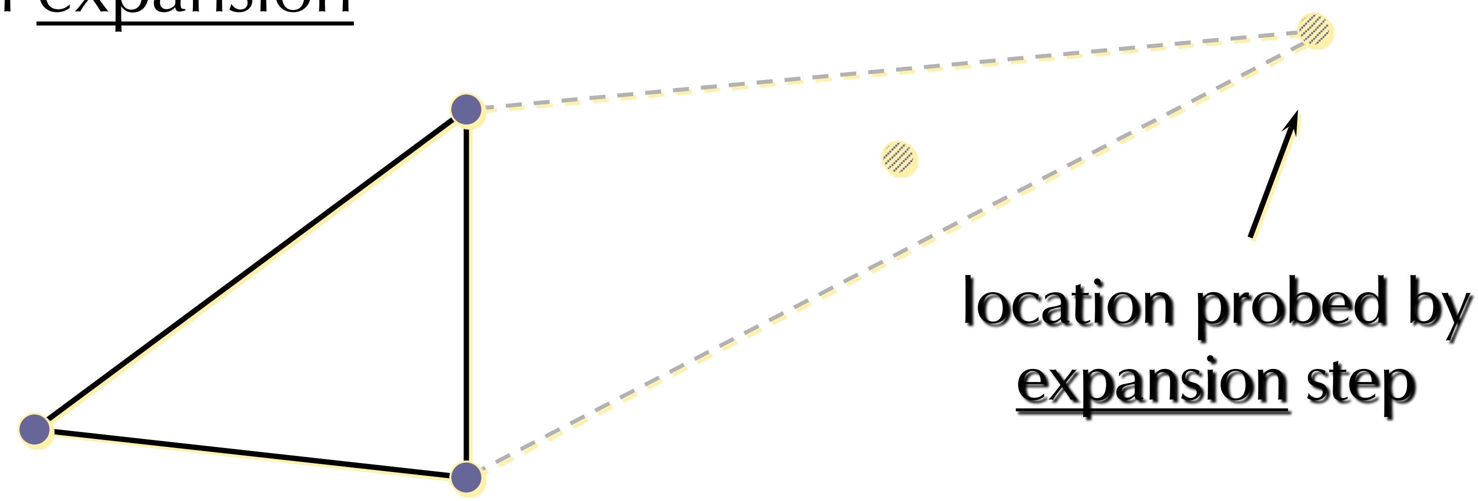
Downhill Simplex Method (Nelder-Mead)

- Basic operation: reflection



Downhill Simplex Method (Nelder-Mead)

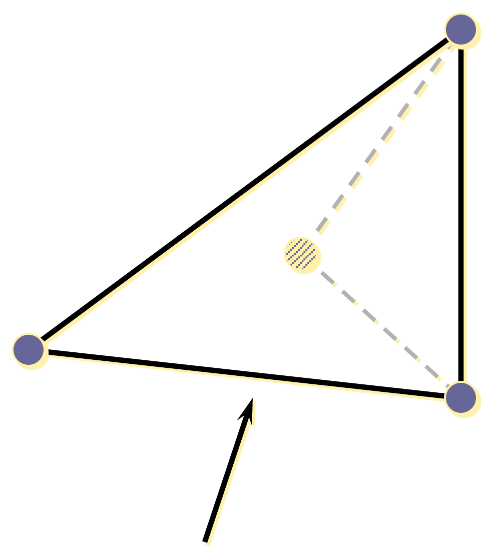
- If reflection resulted in best (lowest) value so far, try an expansion



- Else, if reflection helped at all, keep it

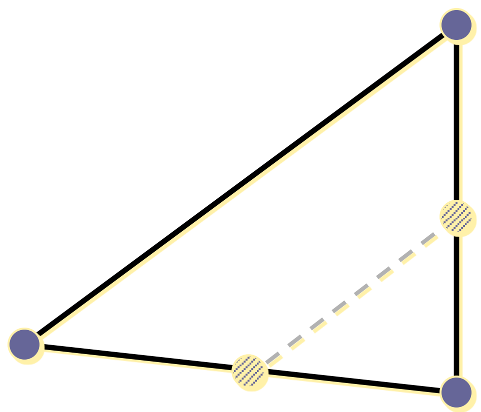
Downhill Simplex Method (Nelder-Mead)

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



Downhill Simplex Method (Nelder-Mead)

- If all else fails shrink the simplex around the *best* point



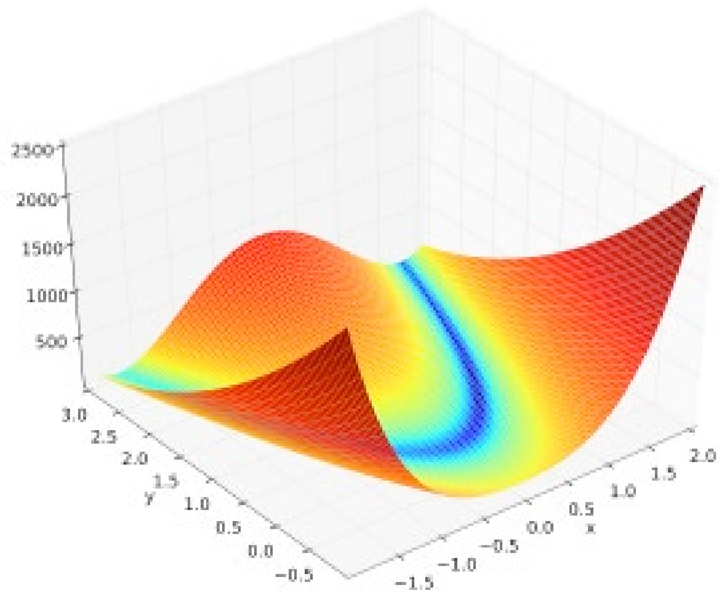
Downhill Simplex Method (Nelder-Mead)

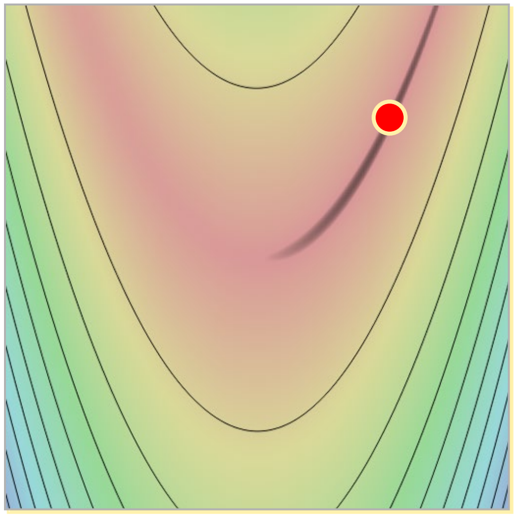
- Method fairly efficient at each iteration (typically 1-2 function evaluations)
- Can take *lots* of iterations
- Somewhat flakey – sometimes needs *restart* after simplex collapses on itself, etc.
- Benefits: simple to implement, doesn't need derivative, doesn't care about function smoothness, etc.

Rosenbrock's Function

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

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





Demo

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

Software notes

Software

- Matlab:
  - fminbnd
    - For function of 1 variable with bound constraints
    - Based on golden section & parabolic interpolation
    - f(x) doesn't need to be defined at endpoints
  - fminsearch
    - Simplex method (i.e., no derivative needed)
  - Optimization Toolbox (available free @ Princeton)
  - meshgrid
  - surf
- Excel: Solver