Numerical Optimization

Physics 113

02/10/21

Algorithms for Optimization M J Kochenderfer, T A Wheeler
https://algorithmsbook.com/optimization/ Stanford AA222

What is it?

- Finding the extremal value of a function.
- Finding the ’best’ (optimal) value for a set of design parameters.

$$\begin{aligned} \underset{\mathbf{x}}{\text{minimize}} \quad & f(\mathbf{x}) \\ \text{subject to} \quad & \mathbf{x} \in \mathcal{X} \end{aligned}$$

$$\underset{\mathbf{x}}{\text{minimize}} -f(\mathbf{x}) \text{ subject to } \mathbf{x} \in \mathcal{X}$$

$$\underset{\mathbf{x}}{\text{maximize}}\, f(\mathbf{x}) \text{ subject to } \mathbf{x} \in \mathcal{X}$$

Why is it useful?

Ubiquitous in the physical sciences:

- Model Fitting
- Experiment Design
- Minimum energy states (Euler-Lagrange)

$$\frac{\partial L}{\partial q_i}(t, \boldsymbol{q}(t), \dot{\boldsymbol{q}}(t)) \qquad = 0 \quad \text{for } i = 1, \dots, n.$$

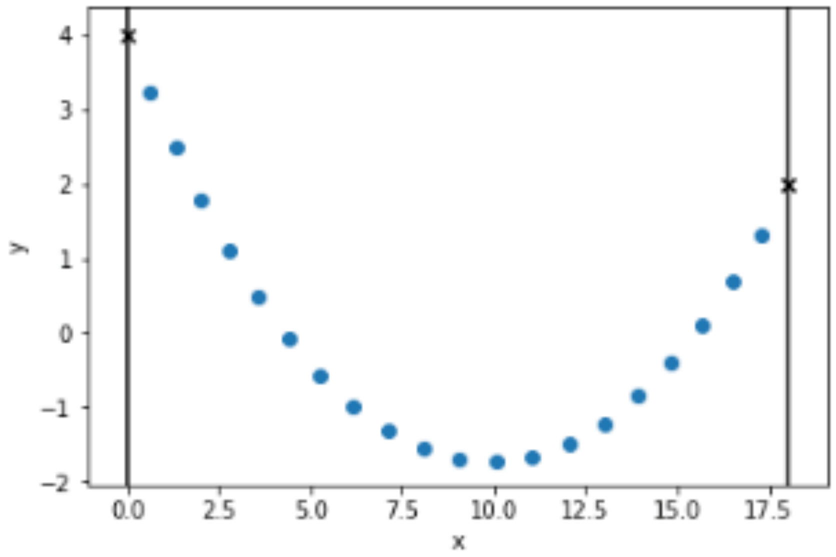
- Machine Learning!
- Engineering, finance, economics….

Some concrete examples

- Classical Mechanics -- Shape of chain of hanging masses:

$$V(\mathbf{x}) = \sum_{i=1}^{N} m_i g y_i + \sum_{i=1}^{N} \frac{1}{2} k_i (\sqrt{(x_i - x_{i+1})^2 + (y_i - y_{i+1})^2} - l_i)^2$$

$$\mathbf{x} \in \mathbb{R}^{2N}$$



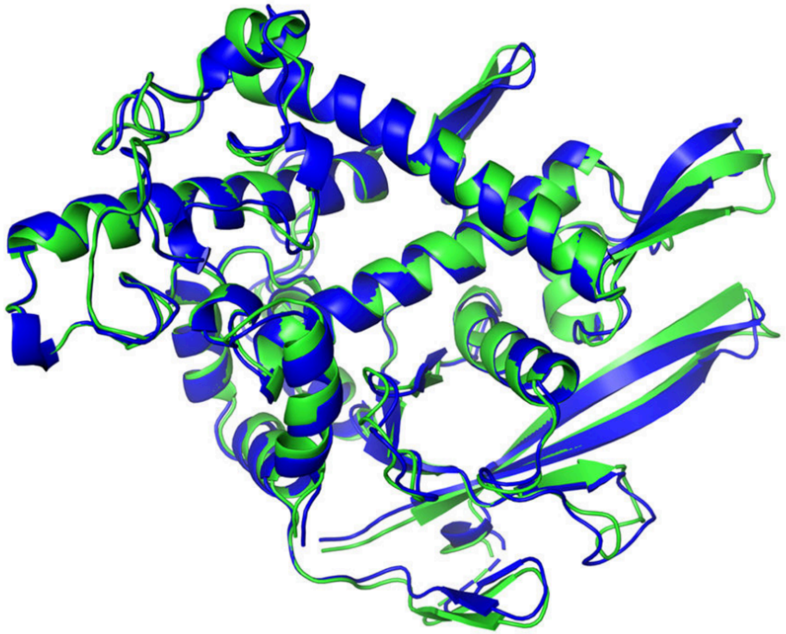
Biophysics

- Biophysics -- Ground state equilibrium geometry for molecules and proteins. (AMBER)

$$V(r^N) = \sum_{i\in\text{bonds}} k_{b\,i}(l_i - l_i^0)^2 + \sum_{i\in\text{angles}} k_{a\,i}(\theta_i - \theta_i^0)^2$$

$$+ \sum_{i\in\text{torsions}} \sum_{n} \frac{1}{2} V_i^n [1 + \cos(n\omega_i - \gamma_i)]$$

$$+ \sum_{j=1}^{N-1} \sum_{i=j+1}^{N} f_{ij} \left\{ \epsilon_{ij} \left[\left(\frac{r_{ij}^0}{r_{ij}} \right)^{12} - 2 \left(\frac{r_{ij}^0}{r_{ij}} \right)^{6} \right] + \frac{q_i q_j}{4\pi\epsilon_0 r} \right.$$

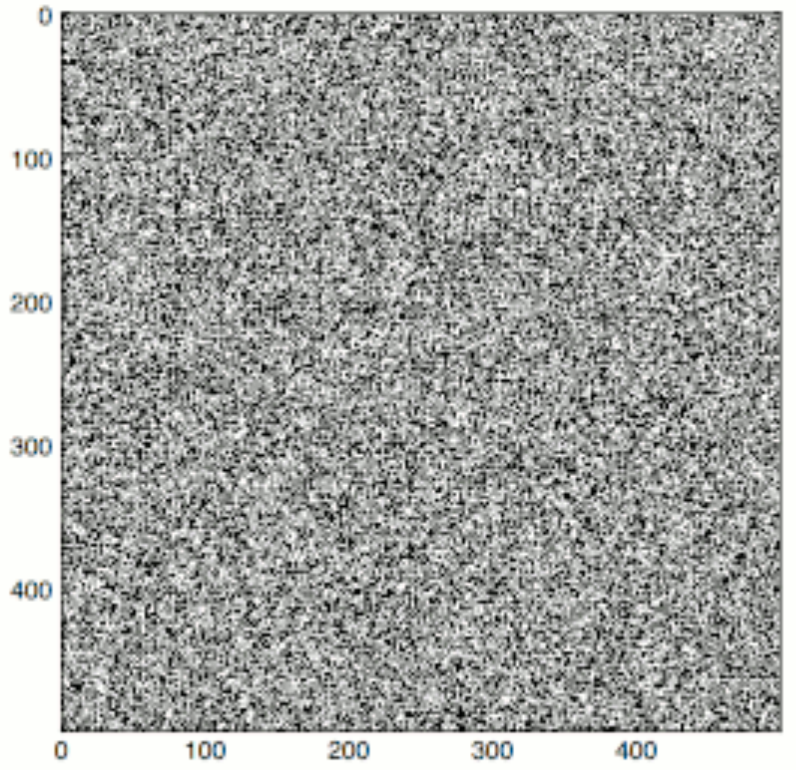


Ising model

- Ferromagnets, spin glasses, neuroscience

$$H(\sigma) = -\sum_{\langle i\ j\rangle} J_{ij}\sigma_i\sigma_j - \mu\sum_j h_j\sigma_j$$

$$P_\beta(\sigma) = \frac{e^{-\beta H(\sigma)}}{Z_\beta}$$



- Best approached with stochastic optimization methods like MCMC

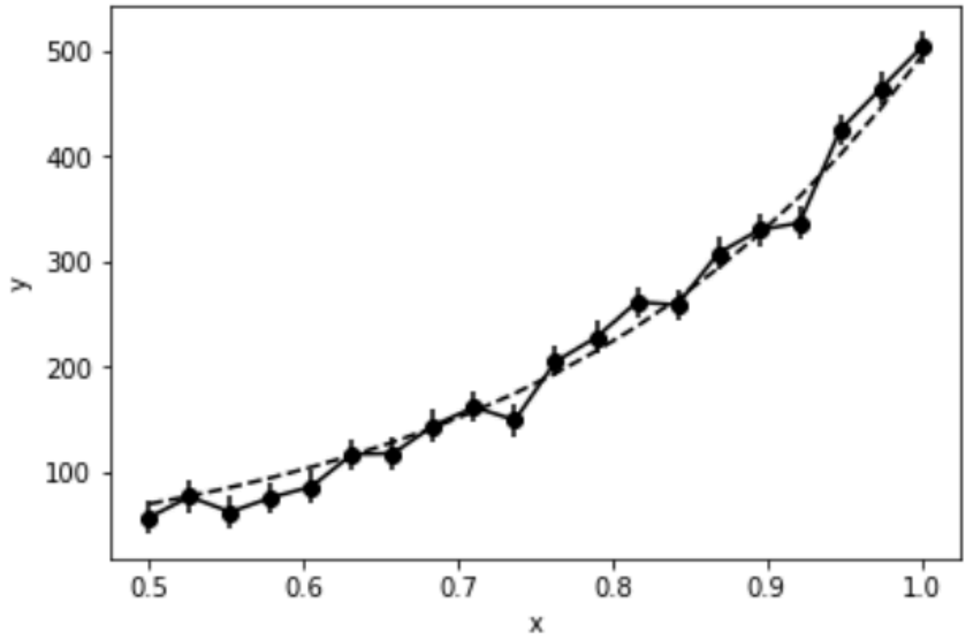
Model fitting

- Dataset and any model f. $\{x_i, y_i, \sigma_i\}_{i=1}^{N}$
- Minimize sum of model residuals:

$$\sum_{i=1}^{N} \frac{\|y_i - f(x_i)\|^2}{\sigma_i^2}$$

- Only analytical solution for linear models.
- Neural Networks famously non-linear.

$$f(x) = a + be^{\frac{x}{c}}$$



Partial Differential Equations

- Relaxation methods for boundary value problems (Gravity, electrodynamics, fluid dynamics)
- Solving systems of linear equations

$$\nabla^2 \phi = 4\pi G\rho.$$

$$(\nabla^2 u)_{ij} = \frac{1}{\Delta x^2}(u_{i+1,j} + u_{i-1,j} + u_{i,j+1} + u_{i,j-1} - 4u_{ij}) = g_{ij}$$

$$A\vec{u} = \vec{b}$$

Outline

- Conditions for optimality
- Derivatives and Numerical differentiation
- 1D Optimization and root finding
- Multi-dimensional Optimization
- First order (gradient) Based methods
- Second order methods
- Next lecture: Applications to model fitting, MCMC

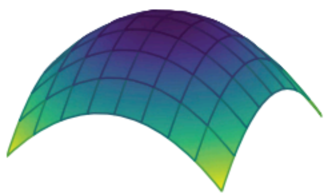
Conditions for Optimality

$\mathbb{R}^1$

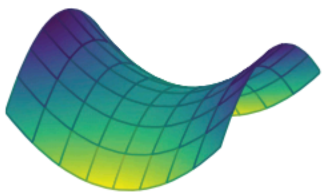
1. $f'(x^*) = 0$, the *first-order necessary condition* (FONC)
2. $f''(x^*) \geq 0$, the *second-order necessary condition* (SONC)

$\mathbb{R}^N$

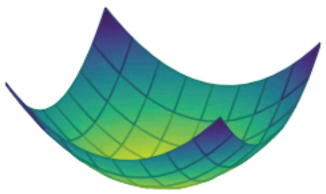
1. $\nabla f(\mathbf{x}) = 0$, the first-order necessary condition (FONC)
2. $\nabla^2 f(\mathbf{x})$ is positive semidefinite (SONC)



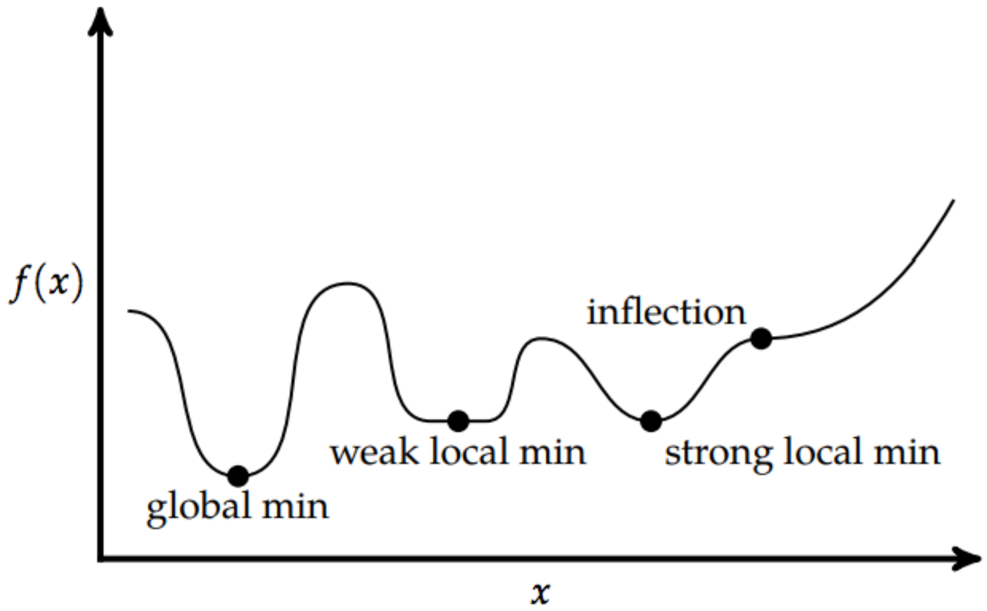
A *local maximum*. The gradient at the center is zero, but the Hessian is negative definite.



A *saddle*. The gradient at the center is zero, but it is not a local minimum.

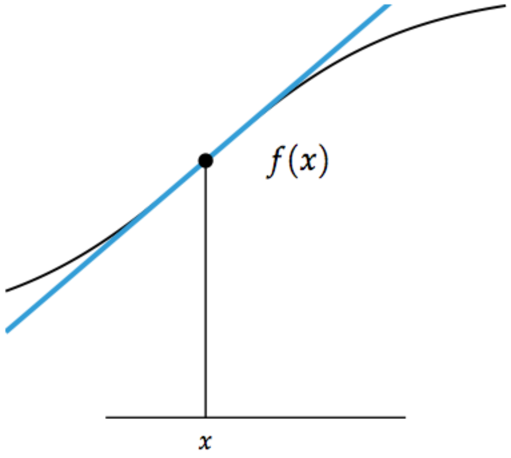


A *bowl*. The gradient at the center is zero and the Hessian is positive definite. It is a local minimum.



Important Aside: Derivatives

$\mathbb{R}^1$



$$f(x) \approx f(x_0) + f'(x_0)(x - x_0)$$

First Order Approximation.

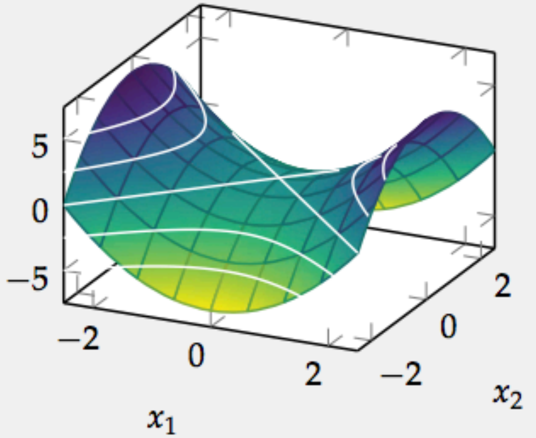
$$f(\mathbf{x}) \approx f(\mathbf{x_0}) + \nabla f(\mathbf{x_0})^T (\mathbf{x} - \mathbf{x_0})$$

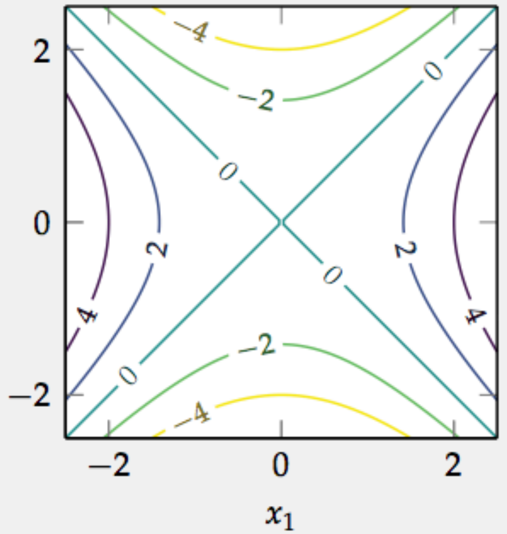
$$\nabla f(\mathbf{x}) = \left[\frac{\partial f(\mathbf{x})}{\partial x_1}, \quad \frac{\partial f(\mathbf{x})}{\partial x_2}, \quad \ldots, \quad \frac{\partial f(\mathbf{x})}{\partial x_n}\right]$$

Gradient

$\mathbb{R}^2$

$f(x_1, x_2) = x_1^2 - x_2^2$





$$df = \nabla f(\mathbf{x})^T d\mathbf{x}$$

$$\nabla_{\mathbf{s}} f(\mathbf{x}) = \nabla f(\mathbf{x})^\top \mathbf{s}$$

Directional Derivative

$$\nabla^2 f(\mathbf{x}) = \begin{bmatrix} \frac{\partial^2 f(\mathbf{x})}{\partial x_1 \partial x_1} & \frac{\partial^2 f(\mathbf{x})}{\partial x_1 \partial x_2} & \cdots & \frac{\partial^2 f(\mathbf{x})}{\partial x_1 \partial x_n} \\ & \vdots & & \\ \frac{\partial^2 f(\mathbf{x})}{\partial x_n \partial x_1} & \frac{\partial^2 f(\mathbf{x})}{\partial x_n \partial x_2} & \cdots & \frac{\partial^2 f(\mathbf{x})}{\partial x_n \partial x_n} \end{bmatrix}$$

Hessian

$\nabla f(x_1, x_2) = [\, 2x_1, 2x_2]$

$\nabla f(\,-1, 0\,) = [\,-2, 0\,]$

Numerical Differentiation

- When the gradient isn't available directly, we can often approximate it quite well using function evaluations.

$$f'(x) \approx \underbrace{\frac{f(x+h)-f(x)}{h}}_{\text{forward difference}} \approx \underbrace{\frac{f(x+h/2)-f(x-h/2)}{h}}_{\text{central difference}} \approx \underbrace{\frac{f(x)-f(x-h)}{h}}_{\text{backward difference}}$$

$O(h)$ $\qquad$ $O(h^2)$ $\qquad$ $O(h)$

For multivariate functions:

$$\frac{\partial f}{\partial x}(a,b) = \lim_{h\to 0} \frac{f(a+h,b)-f(a,b)}{h},$$
$$\frac{\partial f}{\partial y}(a,b) = \lim_{h\to 0} \frac{f(a,b+h)-f(a,b)}{h}.$$

$$f'(x) = \frac{\mathrm{Im}(f(x+ih))}{h}$$

Complex Step method

$O(h^2)$

Connection: Optimization and Root finding

Root Finding

$$f(\mathbf{x}) = 0$$

$$\begin{array}{ll} \underset{\mathbf{x}}{\text{minimize}} & \|f(\mathbf{x})\|^2 \\ \text{subject to} & \mathbf{x} \in \mathcal{X} \end{array}$$

Optimization

$$f'(\mathbf{x}) = 0$$

$$\begin{array}{ll} \underset{\mathbf{x}}{\text{minimize}} & f(\mathbf{x}) \\ \text{subject to} & \mathbf{x} \in \mathcal{X} \end{array}$$

Single Variable (1D) Optimization

- Scalar function of scalar variable.

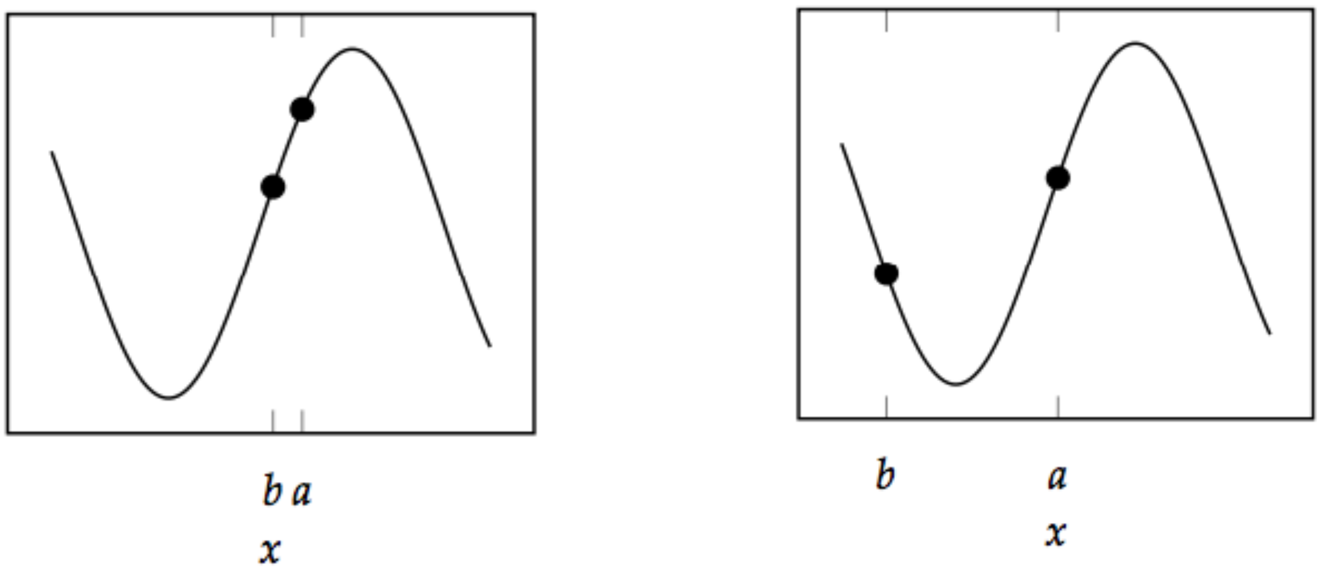
$$\underset{\mathbf{x}}{\text{minimize}} \qquad f(\mathbf{x})$$

- Minimizing non-analytic functions
- Solving transcendental equations

$$e^{-x} - x = 0$$

- Sub-problem for multivariate optimization

Bracketing



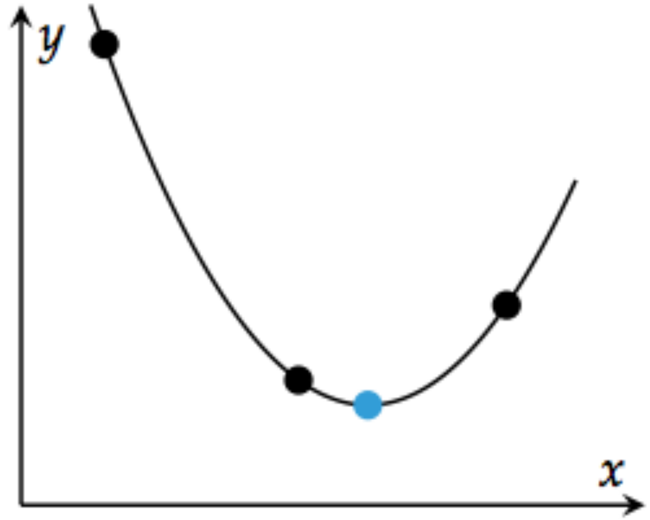
- Bracketing is the process of identifying an interval in which a local minimum lies and then successively shrinking the interval.

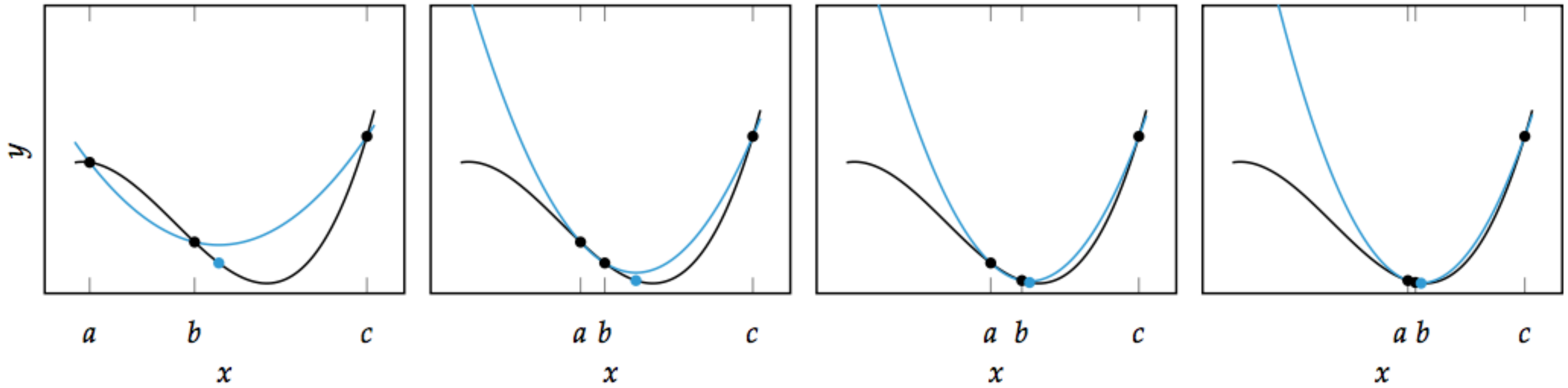
Method 1: Quadratic Fit Search

$$(a, y_a),\ (b, y_b),\ \text{and}\ (c, y_c)$$

$$q(x) = p_1 + p_2 x + p_3 x^2$$

$$\begin{bmatrix} p_1 \\ p_2 \\ p_3 \end{bmatrix} = \begin{bmatrix} 1 & a & a^2 \\ 1 & b & b^2 \\ 1 & c & c^2 \end{bmatrix}^{-1} \begin{bmatrix} y_a \\ y_b \\ y_c \end{bmatrix}$$

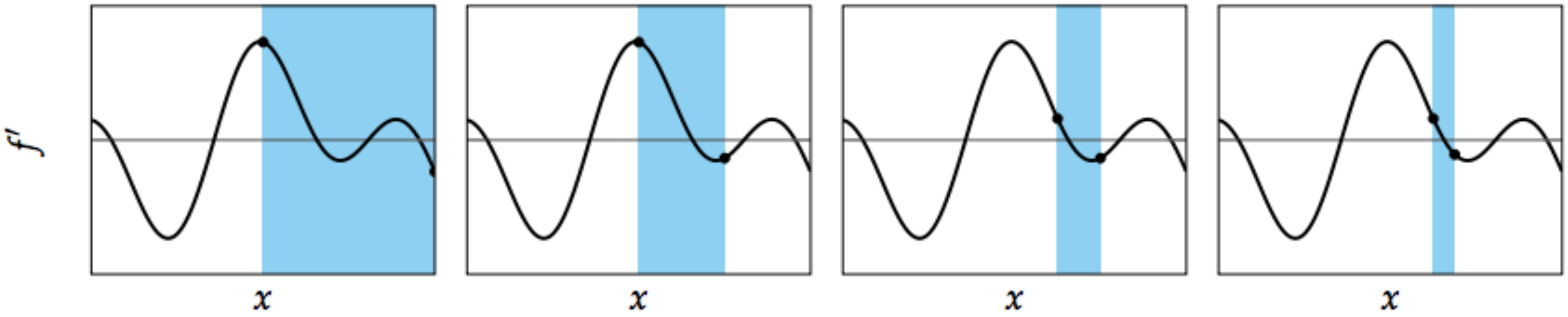




Method 2: Bisection Method

- Works for both root finding and optimization.

1. Identify interval [a,b] that contains minimum.
   (i.e. identify interval with f’(a) < 0 and f’(b) > 0).
2. Take midpoint (a+b)/2.
3. Identify new interval that contains minimum,
   e.g. [(a+b)/2, b].
4. Repeat until convergence.



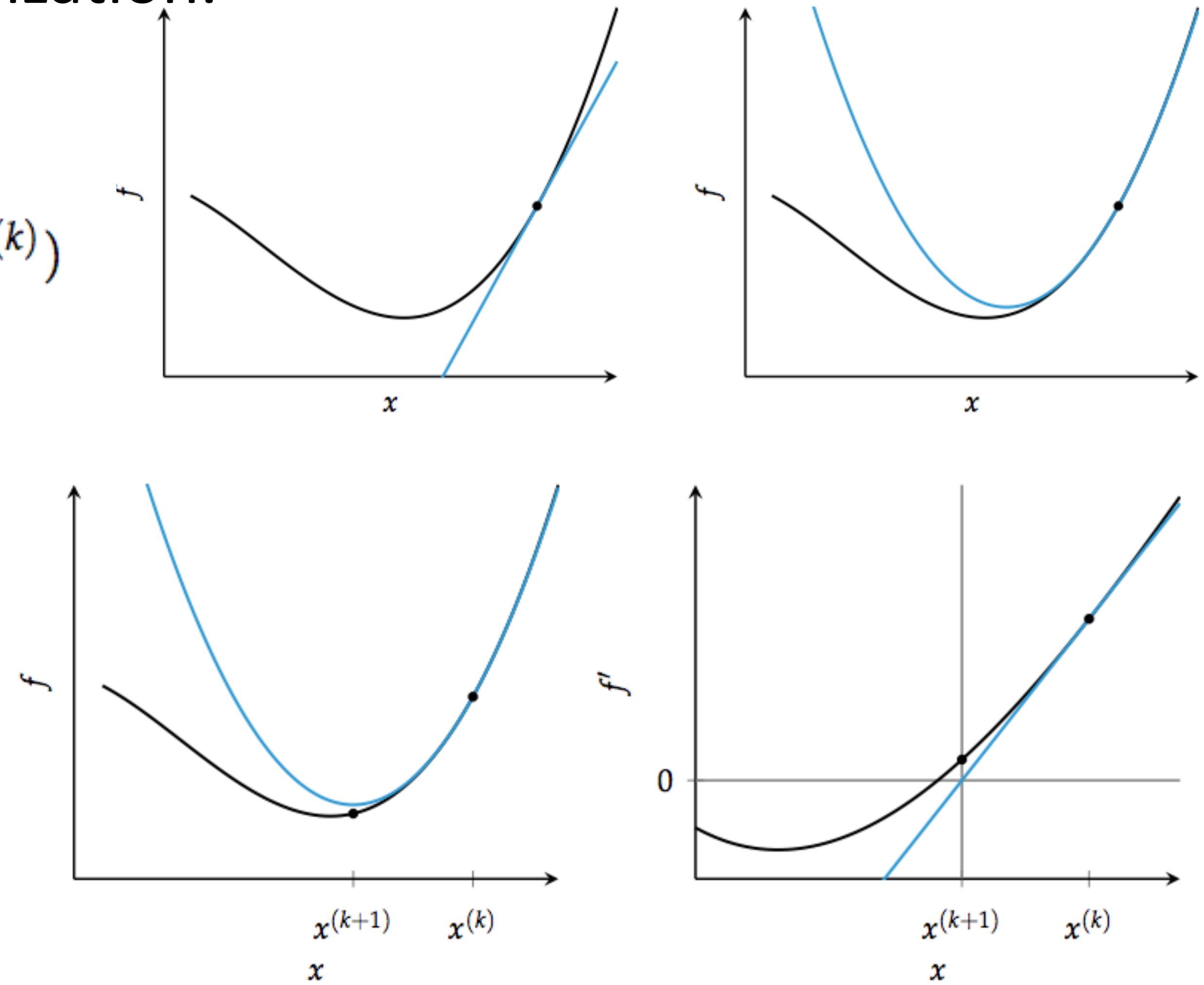
Method 3: Newton’s Method

- Works for both root finding and optimization.

$$q(x) = f(x^{(k)}) + (x - x^{(k)})f'(x^{(k)}) + \frac{(x - x^{(k)})^2}{2} f''(x^{(k)})$$

$$\frac{\partial}{\partial x} q(x) = f'(x^{(k)}) + (x - x^{(k)})f''(x^{(k)}) = 0$$

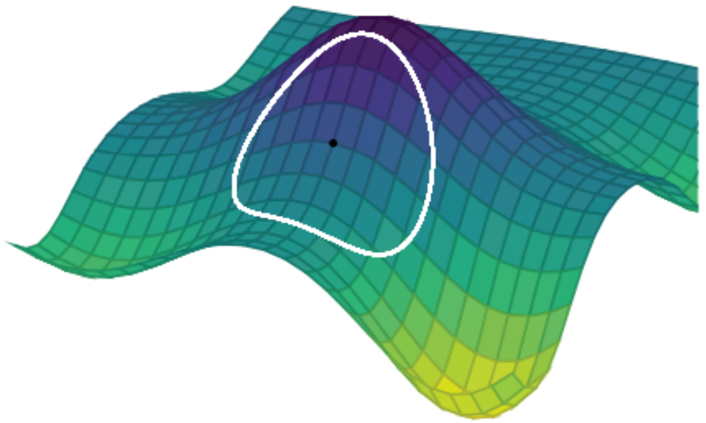
$$x^{(k+1)} = x^{(k)} - \frac{f'(x^{(k)})}{f''(x^{(k)})}$$



Multivariate Optimization: Local Descent

$$\underset{\mathbf{x}}{\text{minimize}} \quad f(\mathbf{x})$$

- Scalar function of a vector variable
- In multivariate problems, we incrementally improve our design point x by taking a step that minimizes an approximation of f(x) based on local information.



Iterative descent procedure

1. Check whether $\mathbf{x}^{(k)}$ satisfies the termination conditions. If it does, terminate; otherwise proceed to the next step.
2. Determine the *descent direction* $\mathbf{d}^{(k)}$ using local information such as the gradient or Hessian. Some algorithms assume $\|\mathbf{d}^{(k)}\| = 1$, but others do not.
3. Determine the step size or learning rate $\alpha^{(k)}$. Some algorithms attempt to optimize the step size so that the step maximally decreases f.[2]
4. Compute the next design point according to:

$$\mathbf{x}^{(k+1)} \leftarrow \mathbf{x}^{(k)} + \alpha^{(k)} \mathbf{d}^{(k)} \tag{4.1}$$

Checking for convergence/stopping

1. Check whether $\mathbf{x}^{(k)}$ satisfies the termination conditions. If it does, terminate; otherwise proceed to the next step.

- Terminate after fixed number of steps.
- If function change small, then terminate. $f(\mathbf{x}^{(k)}) - f(\mathbf{x}^{(k+1)}) < \epsilon_a$
- If norm of the gradient small, terminate. $\|\nabla f(\mathbf{x}^{(k+1)})\| < \epsilon_g$

Which direction to take?

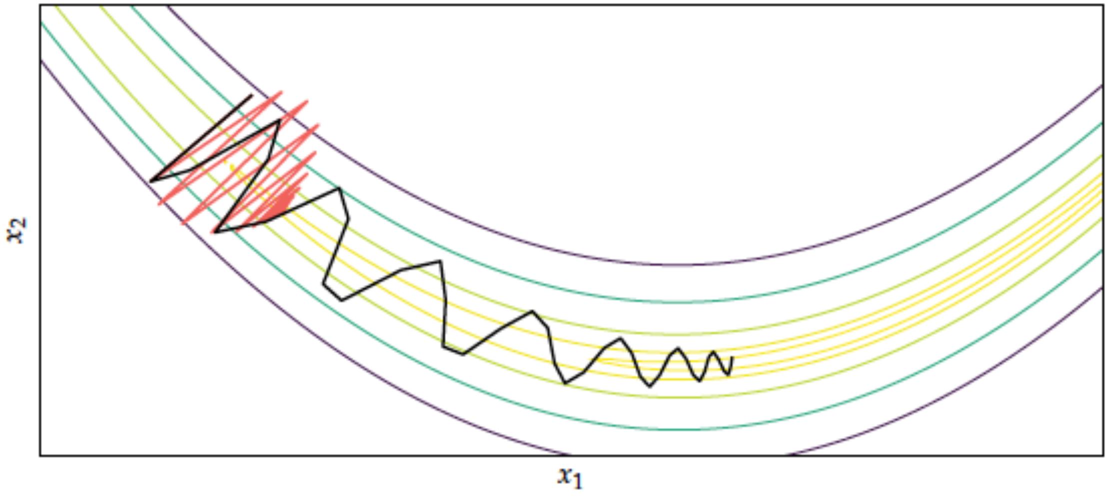
2. Determine the *descent direction* $\mathbf{d}^{(k)}$ using local information such as the gradient or Hessian. Some algorithms assume $\|\mathbf{d}^{(k)}\| = 1$, but others do not.

- Negative gradient -> Direction of maximum decrease of your function.

Negative Gradient not always the best direction:

Some alternatives:

- Conjugate gradient
- Noisy gradient
- Gradient with Momentum



How big a Step?

3. Determine the step size or learning rate $\alpha^{(k)}$. Some algorithms attempt to optimize the step size so that the step maximally decreases f.[2]

4. Compute the next design point according to:

$$\mathbf{x}^{(k+1)} \leftarrow \mathbf{x}^{(k)} + \alpha^{(k)}\mathbf{d}^{(k)} \tag{4.1}$$

- Decaying step size $\alpha^{(k)} = \alpha^{(1)}\gamma^{k-1} \quad \text{for } \gamma \in (0,1]$
- Line Search $\underset{\alpha}{\text{minimize}}\, f(\mathbf{x} + \alpha\mathbf{d})$

Line Search

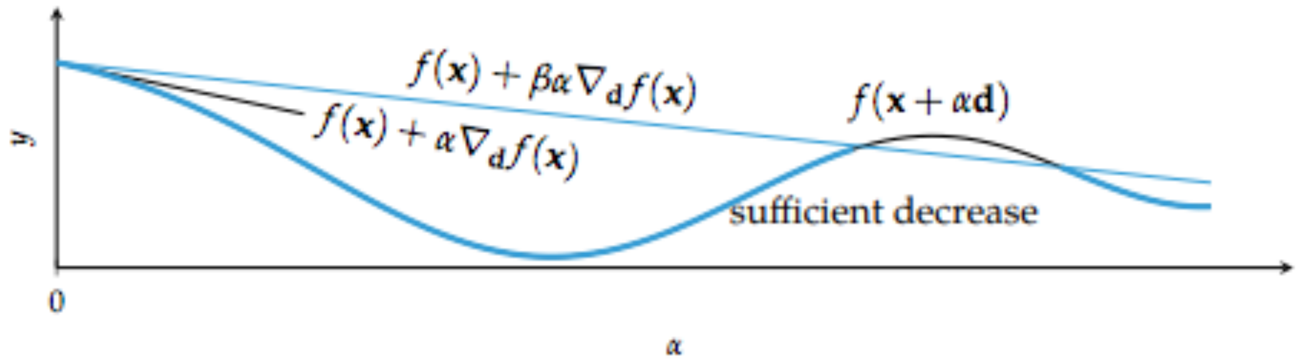
$$\underset{\alpha}{\text{minimize}}\, f(\mathbf{x} + \alpha\mathbf{d})$$

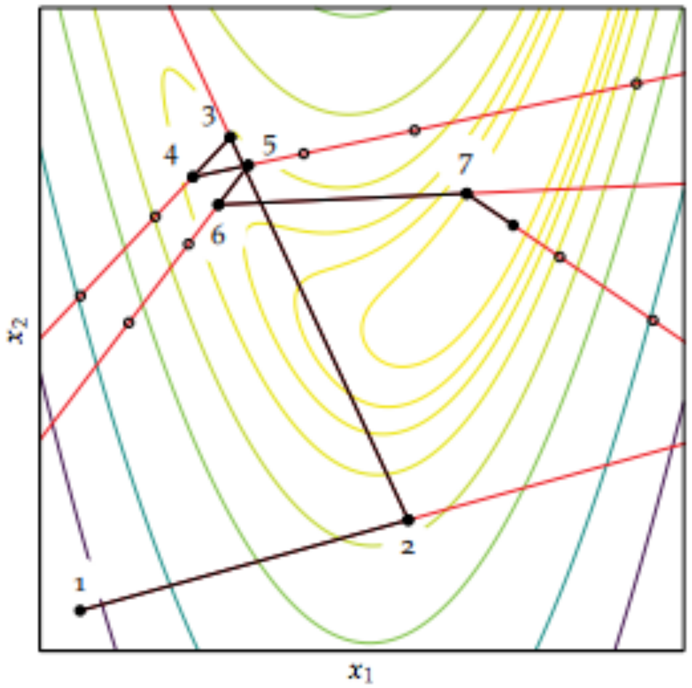
- Generally expensive to solve fully -> Use approximate

Line search.

$$f(\mathbf{x}^{(k+1)}) \le f(\mathbf{x}^{(k)}) + \beta\alpha\nabla_{\mathbf{d}^{(k)}} f(\mathbf{x}^{(k)})$$

$$\beta \in [0, 1]$$





First Order methods 1: Gradient Descent

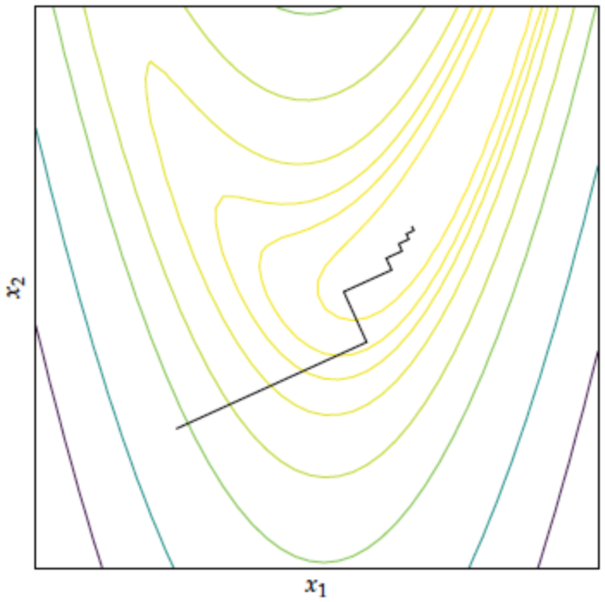
- First order methods use gradient information at each step.

- Gradient descent
- Jagged steps -> gets stuck in valleys.

$$\mathbf{g}^{(k)} = \nabla f(\mathbf{x}^{(k)}) \qquad \mathbf{d}^{(k)} = -\frac{\mathbf{g}^{(k)}}{\|\mathbf{g}^{(k)}\|}$$

- Neural networks use stochastic gradient descent

-> less likely to get stuck

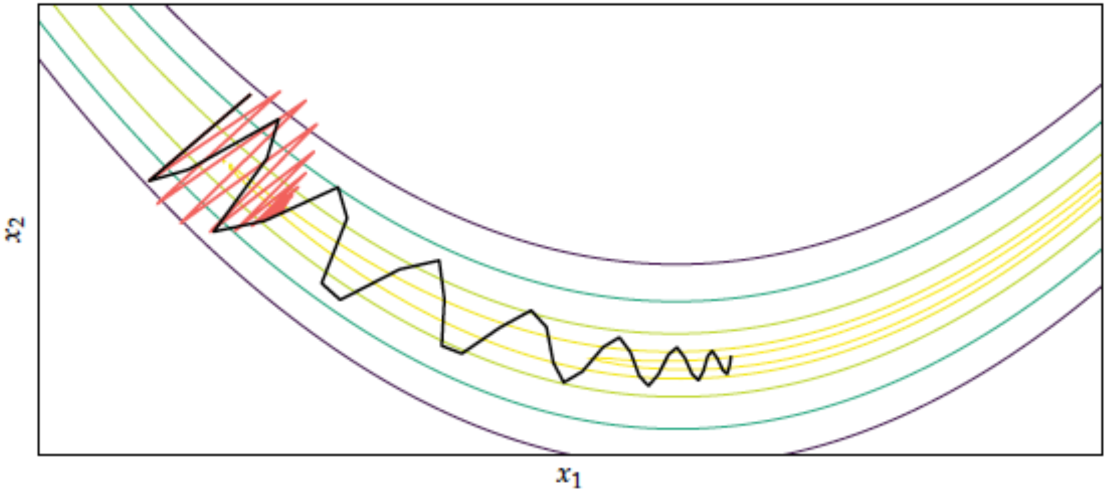


Method 2: Gradient Descent with Momentum

- Keeps momentum along previous direction steps:

$$\mathbf{v}^{(k+1)} = \beta \mathbf{v}^{(k)} - \alpha \mathbf{g}^{(k)}$$
$$\mathbf{x}^{(k+1)} = \mathbf{x}^{(k)} + \mathbf{v}^{(k+1)}$$

- Less likely to get stuck in valleys



Method 3: Conjugate Gradient Method

- Originally for minimizing quadratic functions / solving systems of linear equations

$$\underset{\mathbf{x}}{\text{minimize}}\, f(\mathbf{x}) = \frac{1}{2}\mathbf{x}^\top \mathbf{A}\mathbf{x} + \mathbf{b}^\top \mathbf{x} + c$$

$$\nabla f(\mathbf{x}) = \mathbf{A}\mathbf{x} - \mathbf{b} = 0$$

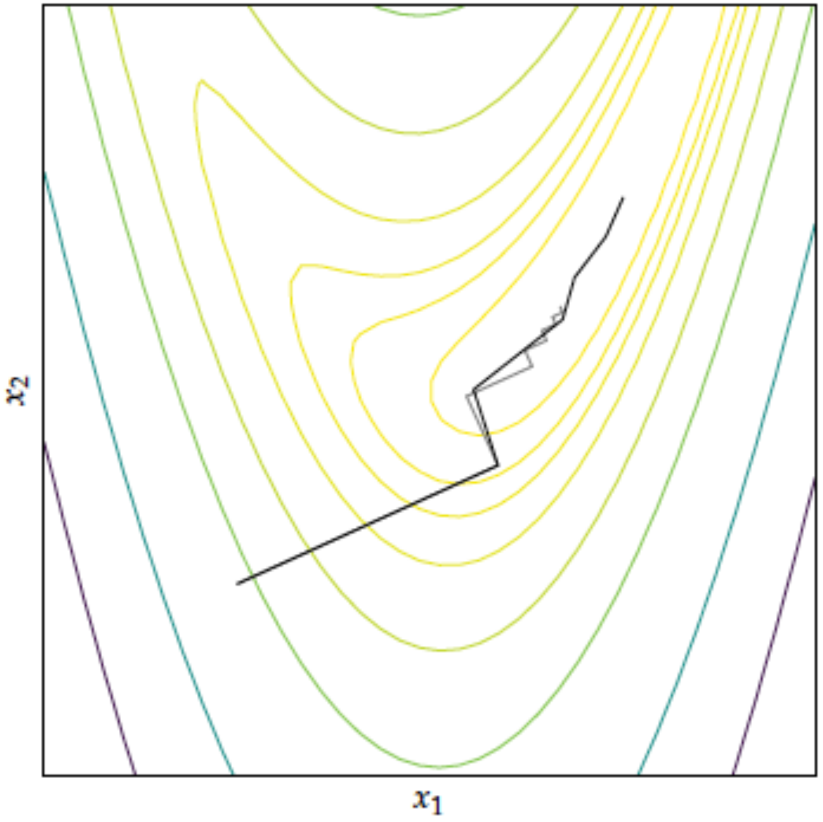
- Modified for general optimization:

$$\mathbf{d}^{(i)\top} \mathbf{A}\, \mathbf{d}^{(j)} = 0 \quad \text{for all} \quad i \neq j$$

$$\mathbf{d}^{(1)} = -\mathbf{g}^{(1)}$$

$$\mathbf{d}^{(k+1)} = -\mathbf{g}^{(k+1)} + \beta^{(k)} \mathbf{d}^{(k)}$$

$$\beta^{(k)} = \frac{\mathbf{g}^{(k)\top}\left(\mathbf{g}^{(k)} - \mathbf{g}^{(k-1)}\right)}{\mathbf{g}^{(k-1)\top}\mathbf{g}^{(k-1)}}$$



Second Order Methods 1: Newton's Method

- Second order methods use gradient and hessian (first and second derivative information at each point).

- Newton's method: Extension of 1D method

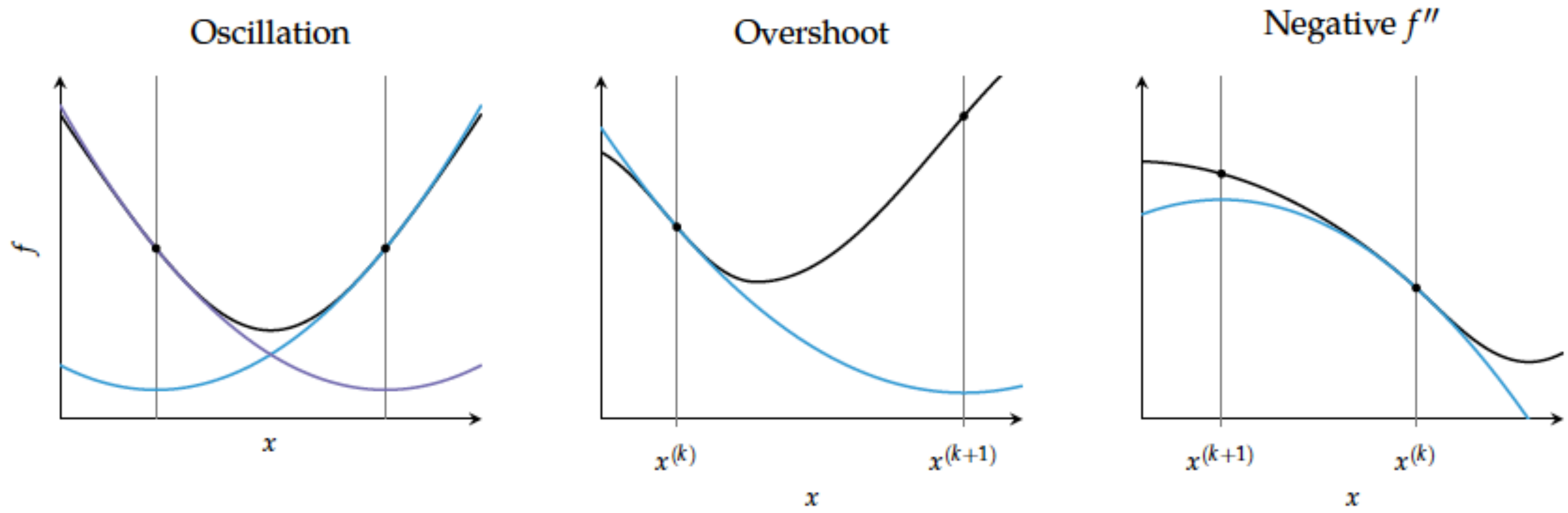
$$f(\mathbf{x}) \approx q(\mathbf{x}) = f(\mathbf{x}^{(k)}) + (\mathbf{g}^{(k)})^\top (\mathbf{x} - \mathbf{x}^{(k)}) + \frac{1}{2}(\mathbf{x} - \mathbf{x}^{(k)})^\top \mathbf{H}^{(k)} (\mathbf{x} - \mathbf{x}^{(k)})$$

$$\nabla q(\mathbf{x}^{(k)}) = \mathbf{g}^{(k)} + \mathbf{H}^{(k)}(\mathbf{x} - \mathbf{x}^{(k)}) = \mathbf{0}$$

$$\mathbf{x}^{(k+1)} = \mathbf{x}^{(k)} - (\mathbf{H}^{(k)})^{-1}\mathbf{g}^{(k)}$$

- Quadratic Convergence: Converges very fast if function bowl shaped.
- Inverting Hessian can be expensive.

Newton Failures



Can be adjusted to include line search and/or step size to avoid these failures

Method 2: Quasi Newton Methods (DFP, BFGS)

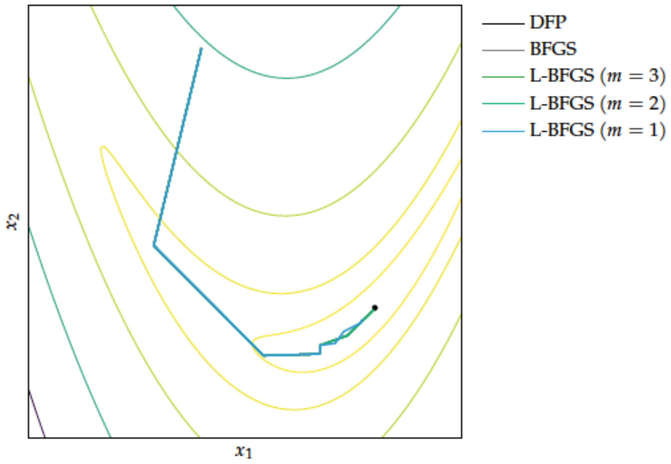
- If we don't know Hessian or it is too expensive to compute, we can approximate it numerically:

$$\mathbf{x}^{(k+1)} \leftarrow \mathbf{x}^{(k)} - \alpha^{(k)}\mathbf{Q}^{(k)}\mathbf{g}^{(k)}$$

$$\gamma^{(k+1)} \equiv \mathbf{g}^{(k+1)} - \mathbf{g}^{(k)}$$
$$\delta^{(k+1)} \equiv \mathbf{x}^{(k+1)} - \mathbf{x}^{(k)}$$

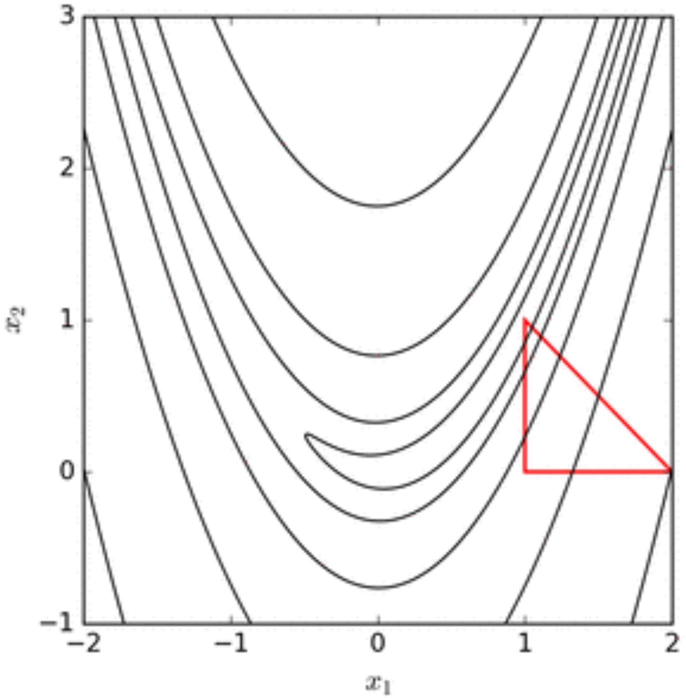
$$\mathbf{Q} \leftarrow \mathbf{Q} - \frac{\mathbf{Q}\gamma\gamma^\top\mathbf{Q}}{\gamma^\top\mathbf{Q}\gamma} + \frac{\delta\delta^\top}{\delta^\top\gamma}$$

$$\text{1D}\begin{cases} f''(x^{(k)}) \approx \dfrac{f'(x^{(k)}) - f'(x^{(k-1)})}{x^{(k)} - x^{(k-1)}} \\ x^{(k+1)} \leftarrow x^{(k)} - \dfrac{x^{(k)} - x^{(k-1)}}{f'(x^{(k)}) - f'(x^{(k-1)})} f'(x^{(k)}) \end{cases}$$



Direct Methods (0th order)

- Can use methods mentioned so far and just approximate gradient numerically.
- There do exist specialized methods that only use function information (no gradient).
- Nelder-Mead Simplex:



Scipy.optimize.minimize()

method : *str or callable, optional*

Type of solver. Should be one o

- 'Nelder-Mead' (see here)
- 'Powell' (see here)
- 'CG' (see here)
- 'BFGS' (see here)
- 'Newton-CG' (see here)
- 'L-BFGS-B' (see here)
- 'TNC' (see here)
- 'COBYLA' (see here)
- 'SLSQP' (see here)
- 'trust-constr'(see here)
- 'dogleg' (see here)
- 'trust-ncg' (see here)
- 'trust-exact' (see here)
- 'trust-krylov' (see here)