# Optimization

(Lectures on Numerical Analysis for Economists III)

Jesús Fernández-Villaverde[1] and Pablo Guerrón[2]

February 24, 2022

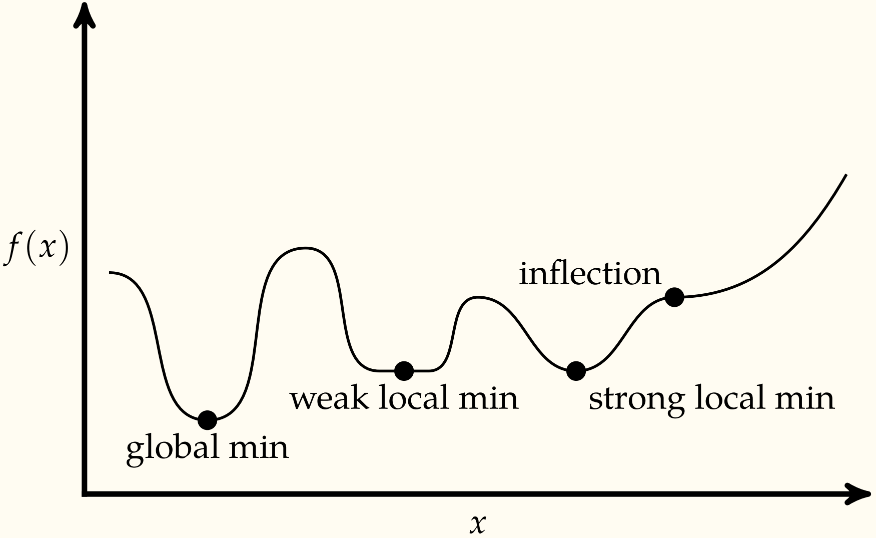
[1]University of Pennsylvania

[2]Boston College

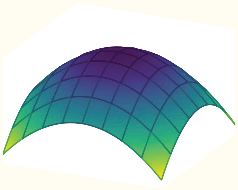
# Optimization

- Optimization of functions is at the core of most economic models: fundamental behavioral assumption of agents (even when we consider cognitive biases).
- Also, key for most methods is classical econometrics.
- Nowadays: machine learning $\rightarrow$ large optimization problems that require efficient computation. Think about OLS with thousands of regressors.
- We rarely have closed-form solutions.
- Minimization vs. maximization.
- Why minimization in this class?

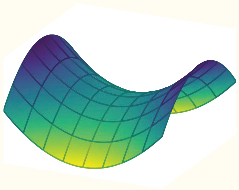
# The challenge, I



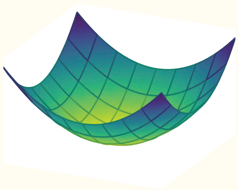
# The challenge, II



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

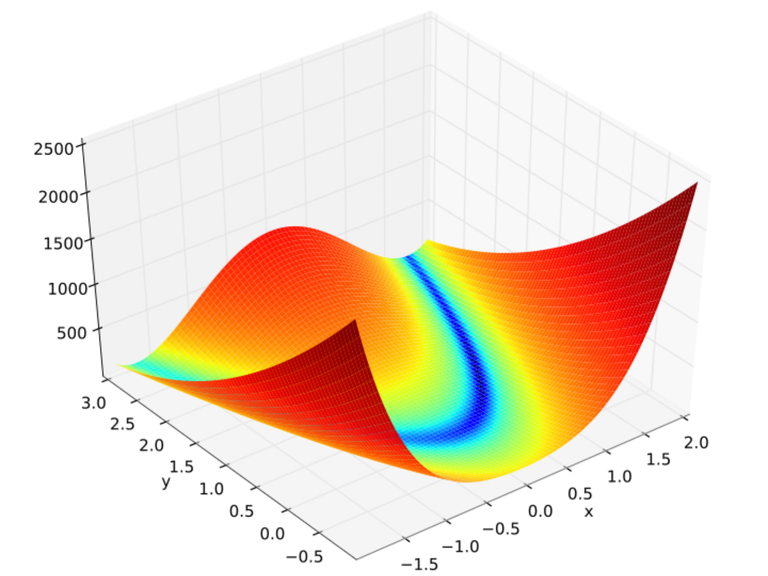


A *saddle*. The gradient at the center is zero, but it is not a local minimum.



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

# The Rosenbrock function: $(a - x)^2 + b(y - x^2)^2$



# Some preliminaries I

- Particularly important to implement it well.
- Optimization is costly. In fact, we want to avoid it if possible.
- Often, it comes nested inside another loop.
- Always possible to miss the exact solution.
- Errors might accumulate.
- Often, hard to parallelize.

# Some preliminaries II

- Transformations of the objective function.
- Including constraints:
  1. Design algorithm: interior point, SQP, trust-region reflective.
  2. Penalty functions and Lagrangian methods.
- When possible, use state-of-the-art software:
  1. NLopt: `https://nlopt.readthedocs.io`.
  2. IPOPT: `https://coin-or.github.io/Ipopt/`.
  3. GNU Linear Programming Kit (GLPK): `https://www.gnu.org/software/glpk/`.
  4. `Matlab` toolboxes.
- Test, test, and test.

# The landscape I

- Algorithms for optimization go back at least to Euclid (325-265 BCE).
- Easy to fill a year-long sequence *just* talking about optimization algorithms.
- We will focus on four classes of methods:
  1. Basic search methods.
  2. Descent direction methods.
  3. Alternative non-derivative-based methods.
  4. Simulation methods.

# The landscape II

- We will skip:

    1. Linear programming (including simplex, interior point, and active-set).
    2. Linear-quadratic programming.
    3. Integer programming.
    4. Multiobjective optimization (including minmax-type problems).
    5. Global optimization: including multistart solvers, generalized pattern search (GPS), generating set search (GSS), and mesh adaptive search (MADS).

# A warning

- No free lunch theorem by Wolpert and Macready (1997).
- Loosely speaking: there is no reason to prefer one algorithm over another, unless we make we know something regarding the probability distribution over the space of possible objective functions.
- In particular, if one algorithm performs better than another on one class of problems, it will perform worse on another class of problems.

# Some references

- *Algorithms for Optimization* by Mykel J. Kochenderfer and Tim A. Wheeler.
- *Numerical Optimization, 2nd edition* by Jorge Nocedal and Stephen Wright.
- *Linear and Nonlinear Programming (3rd ed.)*, by David G. Luenberger and Yinyu Ye.
- *Derivative-Free and Blackbox Optimization* by Charles Audet and Warren Hare.

# Basic search methods

# Grid search

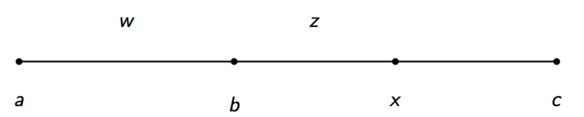
- We define a grid $[x_1, x_2, ..., x_N]$ with $N$ points.
- We check the function $f(\cdot)$ at each point of the grid.
- We keep the lowest (highest) value.
- Slow (with strong curse of dimensionality) and it may fail if grid too coarse.
- But, under certain condition, it can be quite useful:

  1. Discrete choices.
  2. Monotonicities that we can exploit.
  3. Bracket initial choices for other algorithms.
  4. Easy to parallelize.

# Golden section search

- Find minimum $x$ of unimodal continuous $f : X \rightarrow R$ in an interval $[a, c]$.
- By Weierstrass theorem, the minimum exists on $[a, c]$.
- Assume $\exists\ x \in (a, c)$ and $f(x) < min[f(a), f(c)]$.
- Idea:
  1. Select triplet $(a, b, c)$.
  2. Update triplet to $(a', b', c')$ with narrower value range that includes maximum.
  3. Stop when value range is narrow enough.
- Questions:
  1. How do we optimally pick triplet $(a, b, c)$?
  2. How do we optimally update triplet $(a, b, c)$?

# Algorithm




1. Set $b = a + \frac{3-\sqrt{5}}{2} * (c - a)$.
2. Set $x = a + \frac{\sqrt{5}-1}{2} * (c - a)$.
3. If $|x - b| < tol$, then exit the algorithm with return $\frac{x+b}{2}$. If not, go to step 4.
4. If $f(b) < f(x)$, update triplet to $(a, b, x)$ and go to step 1.

   else, update triplet to $(b, x, c)$ and go to step 1.

# Computing the Golden Ratio

- The next $x$ lies either on current $(a, x)$ or on $(b, c)$.
- Minimize the worst by equating the size of the intervals:

$$\frac{b-a}{c-a} = w$$

and

$$\frac{c-b}{c-a} = 1 - w$$

- Scale similarity: choose $w$ to minimize expected length of next interval $\rightarrow$ golden ratio $\approx 0.38197$.
- Then:

$$b = a + \frac{3-\sqrt{5}}{2} * (c-a)$$
$$x = a + \frac{\sqrt{5}-1}{2} * (c-a)$$

# Tolerance

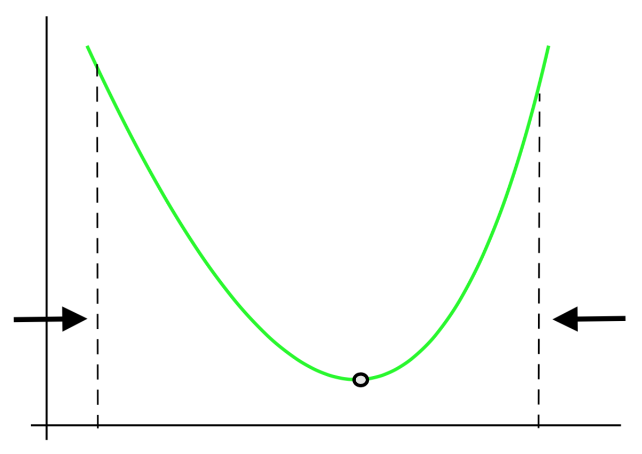
- $\epsilon$ is your computer's floating-point precision.
- Taylor expansion: $f(x) \approx f(b) + \frac{1}{2}f''(b)(x-b)^2$.
- If $f(x)$ and $f(b)$ are indistinguishable for our machine, their difference should be of order $\epsilon$:

$$\frac{1}{2}|f''(b)|(x-b)^2 < \epsilon|f(b)| \iff |x-b| < \sqrt{\frac{2\epsilon|f(b)|}{|f''(b)|}}$$

- $|f(b)|/|f''(b)| \approx 1$ implies $|x-b| < \sqrt{e\epsilon}$ (of order $10^{-4}$ if single precision and of order $10^{-8}$ if double precision).

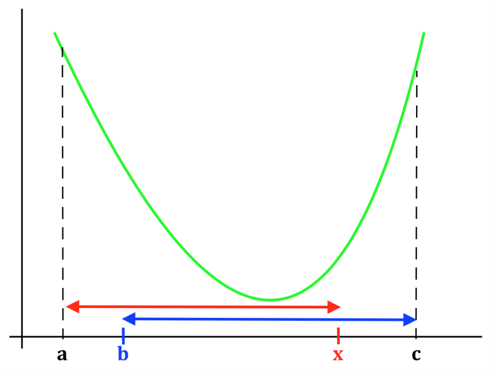
# Graphical explanation I

- Consider interval of function where minimum is located.
- Reduce interval until in converges.



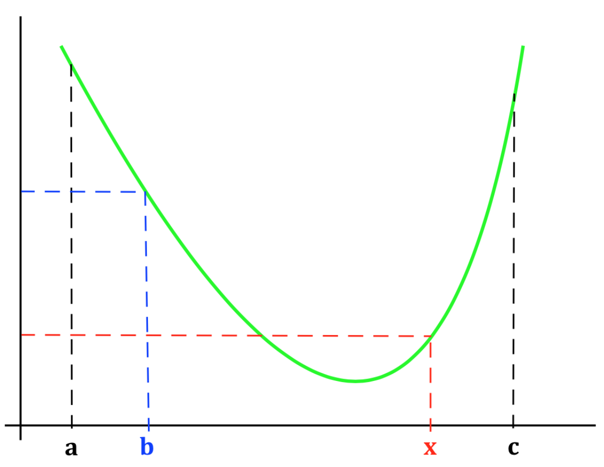
# Graphical explanation II

- Set triplet $(a, b, c)$.
- Choose $x$ such that red and blue lines are equal.
- *Golden section*: Relative size of both lines is a particular number.
- More concretely, $\gamma = \frac{x-a}{c-a} = \frac{c-b}{c-a} = \frac{\sqrt{5}-1}{2} \approx 0.618$



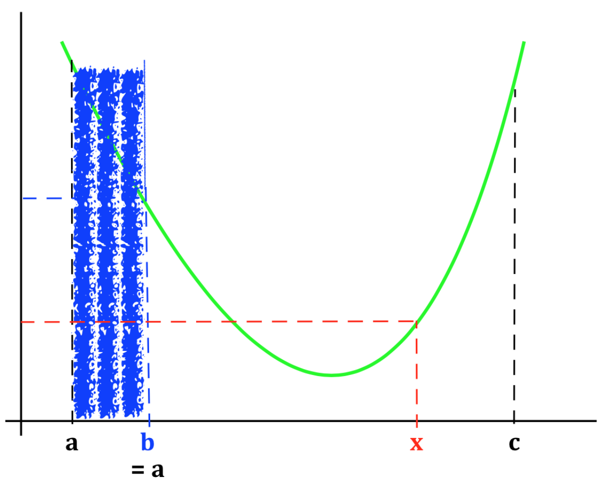
# Graphical explanation III

- Check whether $f(b)$ or $f(x)$ is lower:



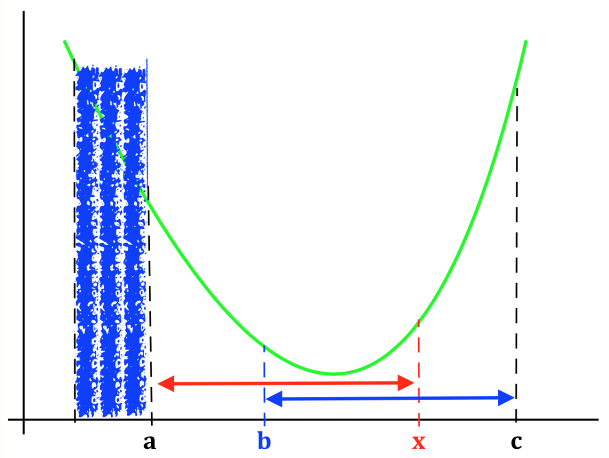
# Graphical explanation IV

- Ignore part of interval to the left of $b$.
- Reset interval $b$ becomes new $a$.



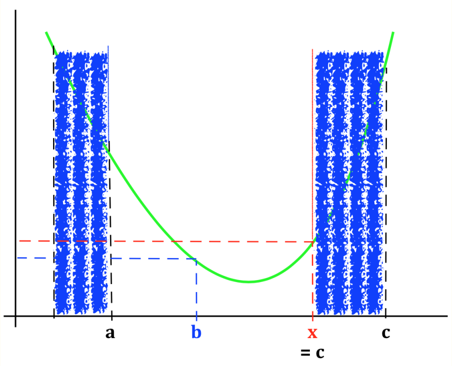
# Graphical explanation V

- Find new $b$.
- Must satisfy same rule as before so: $b = a + \frac{3-\sqrt{5}}{2} * (c - a)$.



# Graphical explanation VI

- Check again whether $f(b)$ or $f(x)$ is lower.
- Ignore part of interval to the right of $x$.
- Reset interval $x$ becomes new $c$.
- Find new $x = a + \gamma(c - a)$.
- Repeat process until $f(b) \approx f(x)$.



# Parabolic interpolation

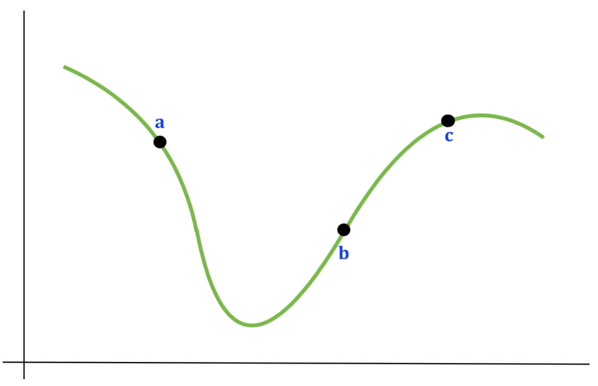
- If the function is parabolic near to the minimum, a parabola fitted through three points will take us to an $\epsilon-$neighborhood of the minimum in a single step.
- Find an abscissa through inverse parabolic interpolation:

$$x = b - \frac{1}{2}\frac{(b-a)^2[f(b)-f(c)]-(b-c)^2[f(b)-f(a)]}{(b-a)[f(b)-f(c)]-(b-c)[f(b)-f(a)]}$$

- This formula fails if the three points are collinear $\Rightarrow$ denominator equals zero

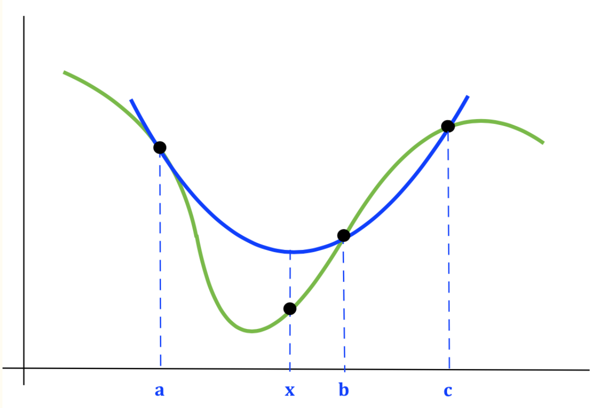
# Graphical explanation I

- Choose three points of the function and draw a parabola through them.



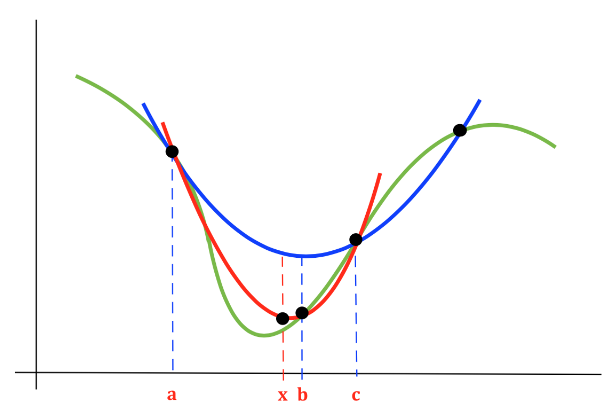
# Graphical explanation II

- Find the minimum of such parabola, evaluate the function at that point, and update points ($c \to b$ and $b \to x$).



# Graphical explanation III

- Draw a second parabola and find its minimum, evaluate, and update points.
- Repeat until convergence.



# Brent's Method

- *Problem:* Formula for $x$ simply finds an extremum, could be a minimum or maximum.
- In practice, no minimization scheme that depends solely on it is likely to succeed.
- *Solution:* Find scheme that relies on a sure-but-slow technique $\Rightarrow$ Combination of golden section search and inverse parabolic interpolation.
- **Brent's method** (a.k.a. Brent-Dekker method): switch between Golden ratio and parabolic interpolation.
- Advantages:
  1. Avoids unnecessary function evaluations in switching between the two methods.
  2. Adequate ending configuration.
  3. Robust scheme to decide when to use either parabolic step or golden sections.

# Brent's method with first derivatives

- Same goal as w/o derivative: Isolate minimum bracketed, but now use information from derivative.
- Not enough to simply search for a zero of the derivative $\rightarrow$ Maximum or minimum?
- Derivatives only useful in choosing new trial points within bracket.
    - If $f'(b) > 0 \rightarrow$ next test point from interval $(a, b)$.
    - If $f'(b) < 0 \rightarrow$ next test point from interval $(b, c)$.

# Descent direction methods

# Descent direction iteration

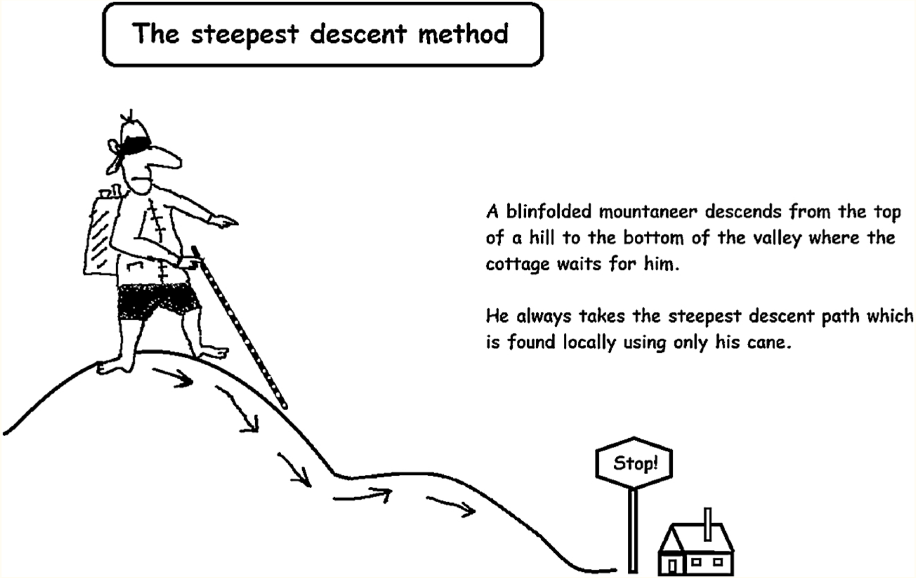
- Most popular optimization method, in practice, is some version of a descent direction iteration method.
- Starting at point $x^{(1)}$ (determined by domain knowledge), a descent direction algorithm generates sequence of steps (called iterates) that converge to a local minimum.
- The descent direction iteration algorithm:
    1. At iteration $k$, check whether $x^{(k)}$ satisfies termination condition. If so stop; otherwise go to step 2.
    2. Determine the descent direction $\mathbf{d}^{(k)}$ using local information such as gradient or Hessian.
    3. Compute step size $\alpha^{(k)}$.
    4. Compute the next candidate point: $x^{(k+1)} \leftarrow x^{(k)} + \alpha^{(k)}\mathbf{d}^{(k)}$.
- Choice of $\alpha$ and $\mathbf{d}$ determines the flavor of the algorithm.

# Gradient descent method, I

- A natural choice for $\mathbf{d}$ is the direction of steepest descent (first proposed by Cauchy in 1847).
- The direction of steepest descent is given by the direction opposite the gradient $\nabla f(x)$. Thus, a.k.a. steepest descent.
- If function is smooth and the step size small, the method leads to improvement (as long as the gradient is not zero).
- The normalized direction of steepest descent is:

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

# Gradient descent method, II



# Gradient descent method, III

- One way to set the step size is to solve a line search:

$$\alpha^k = \arg\min_{\alpha} f(x^{(k)} + \alpha \mathbf{d}^{(k)})$$

  for example with the Brent's method.

- Under this step size choice, it can be shown $\mathbf{d}^{(k+1)}$ and $\mathbf{d}^{(k)}$ are orthogonal.

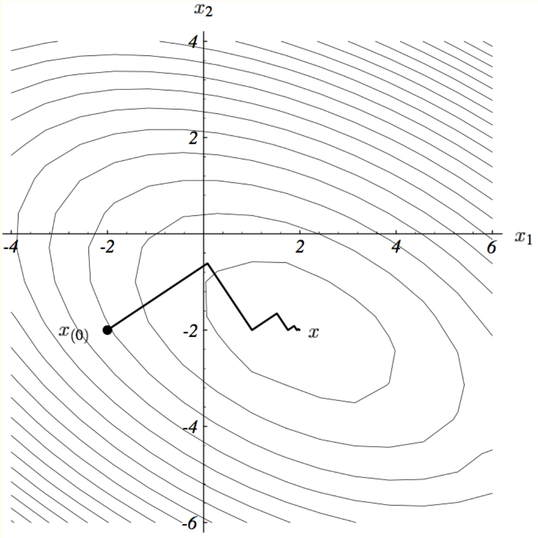
- In practice, line search can be costly and we settle for a fix $\alpha$, a $\alpha^k$ that geometrically decays, or an approximated line search.

- Trade off between speed of convergence and robustness.

**Heard in Minnesota Econ grad student lab**

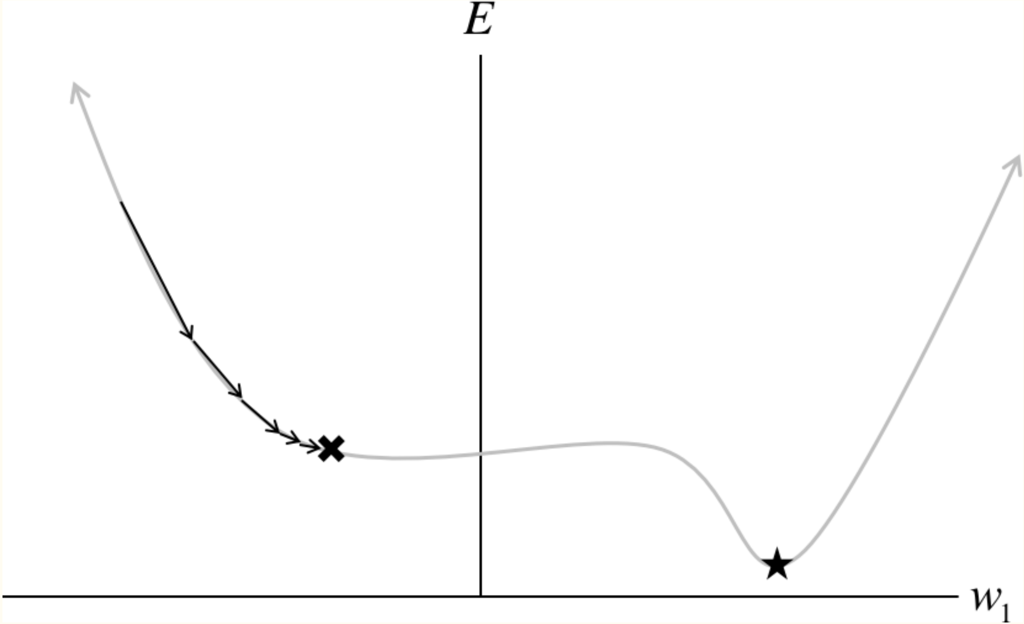
If you do not know where you are going, at least go slowly.

# Gradient descent method, IV

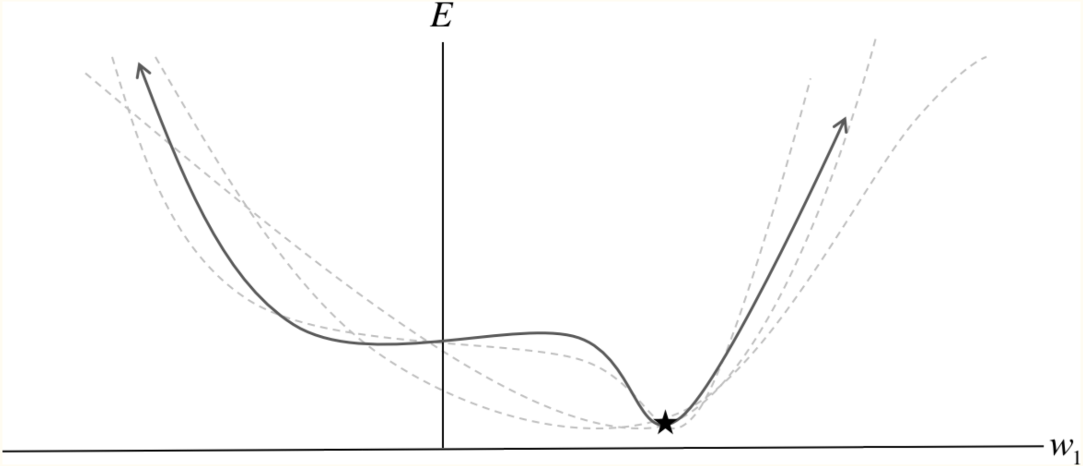


# Stochastic gradient descent

- Even with back propagation, evaluating the gradient when you have many data points can be costly: thousands of points to evaluate!
- Stochastic gradient descent (SDG): We use only one data point to evaluate (an approximation to) the gradient.
- We trade off slower convergence rate for faster computation.
- Intuition from other random algorithms.
- An additional advantage.
- SGD converges almost surely to a global minimum when the objective function is convex (and to a local minimum otherwise).



$E$
$w_1$


E
$w_1$

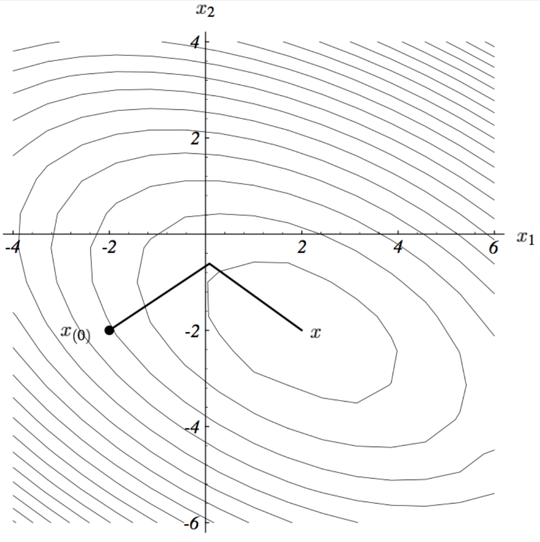
# Minibatch

- A compromise between using the whole training set and pure stochastic gradient descent: minibatch gradient descent.
- This is the most popular algorithm to train neural networks.
- Intuition: the standard error of the mean converges slowly ($\sqrt{n}$).
- Notice also resilience to scaling.
- You can flush the algorithm to a graphics processing unit (GPU) or a tensor processing unit (TPU) instead of a standard CPU.

# Improving gradient descent

- Gradient descent can perform poorly in narrow valleys (it may require many steps to make progress).
- Famous example: Rosenbrock function $\rightarrow (a-x)^2 + b(y-x^2)^2$.
- The *conjugate gradient* method overcomes this problem by constructing a direction conjugate to the old gradient, and to all previous directions traversed.
- Define $g(x) = \nabla f(x)$.
- In first iteration, set: $d^{(1)} = -g(x^{(1)})$ and $x^{(2)} = x^{(1)} + \alpha^{(1)}\mathbf{d}^{(1)}$. Here, $\alpha^{(1)}$ is arbitrary.
- Subsequent iterations set $\mathbf{d}^{(\mathbf{k+1})} = -g^{(k+1)} + \beta^{(k)}\mathbf{d}^{(\mathbf{k})}$.

# Conjugate descent method



# Approaches in traditional optimization

- There are two approaches to set $\beta$:

  1. Fletcher-Reeves:

  $$\beta^{(k)} = \frac{g^{(k)T} g^{(k)}}{g^{(k-1)T} g^{(k-1)}}$$

  2. Olak-Ribiere:

  $$\beta^{(k)} = \frac{g^{(k)T} (g^{(k)} - g^{(k-1)})}{g^{(k-1)T} g^{(k-1)}}$$

- The Olak-Ribiere requires an automatic reset at every iteration: $\beta \leftarrow \max(\beta, 0)$.
- If the function to minimize has flat areas, one can introduce a *momentum* update equation:

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

- The modification reverts to the gradient descent version if $\beta = 0$.
- Intuitively, the momentum update is like a ball rolling down an almost horizontal surface.

# Adam

- Application to neural network training: *Adam* (Adaptive Moment Estimation), Kingma and Ba (2014).
- It uses running averages of both the gradients and the second moments of the gradients.
- Equations

$$
\begin{aligned}
m^{(k+1)} &= \gamma_1 m^{(k)} + (1-\gamma_1)\nabla f(x^{(k)}) \\
v^{(k+1)} &= \gamma_2 v^{(k)} + (1-\gamma_2)\left(\nabla f(x^{(k)})\right)^2 \\
\widehat{m} &= \frac{m^{(k+1)}}{1-\gamma_1} \\
\widehat{v} &= \sqrt{\frac{v^{(k+1)}}{1-\gamma_2}} \\
x^{(k+1)} &= x^{(k)} - \eta\frac{\widehat{m}}{\widehat{v}+\epsilon}
\end{aligned}
$$

# Newton-Raphson method

- Most common optimization method in economics (either basic implementation or, more likely, with modifications).
- Works with univariate and multivariate optimization problems, but requires twice-differentiability of function.
- Named after Isaac Newton and Joseph Raphson.
- Intimately related with the Newton method designed to solve for root to equation $f(x) = 0$.
- Optimizes $f(x)$ by using successive quadratic approximations to it.
- Thus, you can think about the method as a second-order descent method where Hessian gives us size of the step.

## Idea: univariate case

- Given an initial guess $x_0$, compute the second-order Taylor approximation of $f(x)$ around $x_0$:

$$f(x) \approx f(x_0) + f'(x_0)(x - x_0) + \frac{f''(x_0)}{2}(x - x_0)^2$$

- The minimization of this approximation with respect to $x$ has first-order conditions

$$f'(x_0) + f''(x_0)(x^* - x_0) = 0$$

which gives:

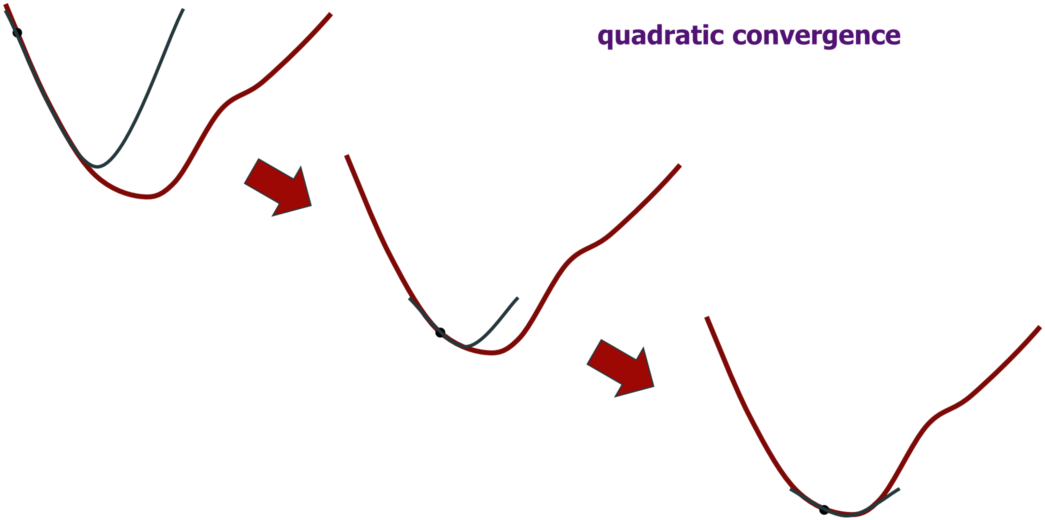
$$x^* = x_0 - \frac{f'(x_0)}{f''(x_0)}$$

- This suggests the iteration

$$x_{n+1} = x_n - \frac{f'(x_n)}{f''(x_n)}$$

that ensures quadratic convergence.

# Graphical view



# Idea: multivariate case

- For a $N$-dimensional vector function $f(x)$, $x \in \mathbb{R}^N$, we can follow the same steps.
- We get:

$$x_{n+1} = x_n - H_n^{-1} \nabla f(x_n)$$

where $\nabla f(x_n)$ is the gradient of $f(x) = 0$ and $H(\cdot)$ its Hessian.

- Problems:
  1. Numerical evaluation of Hessian: curse of dimensionality.
  2. Local vs. global optima.
  3. Very sensitive with respect to initial guess. You can "cool down" the update (manually or with algorithms).

# Quasi-Newton methods

- Evaluating the Hessian is numerically costly: scale $O(n^3)$.
- The Hessian captures the local variation in $\nabla f(x)$.
- First-order Taylor approximation of gradient, from $x_n$ yields:

$$\nabla f(x) \approx \nabla f(x_n) + H_n(n - x_n)$$

- We want to find a $H_n$ such that:
  1. $H_n$ is symmetric. (Strict concavity can guarantee positive-definiteness).
  2. $\nabla f(x_n)$ evaluated through the approximation should equal to the actual one (secant condition).
  3. $H_{n+1}$ should be as "close" to $H_n$ as possible.
- Different proposals to approximate $H_n$ generate different quasi-Newtons.
- For example, we can make $H_n = I$.

# BFGS

- Broyden-Fletcher-Goldfarb-Shanno (BFGS) developed an efficient algorithm to approximate the Hessian:

$$H_{n+1} = H_n + \frac{yy^T}{y^T s} - \frac{H_n ss^T H_n^T}{s^T H_n s}$$

$$s = x_{n+1} - x_n$$

$$y = \nabla f(x_{n+1}) - \nabla f(x_n)$$

- If we take into consideration taking inverse of the Hessian, the scale for computation now is $O(n^2)$.
- Furthermore:

$$H_{n+1}^{-1} = \left(I - \frac{sy^T}{y^T s}\right) H_n^{-1} \left(I - \frac{ys^T}{y^T s}\right) + \frac{ss^T}{y^T s}$$

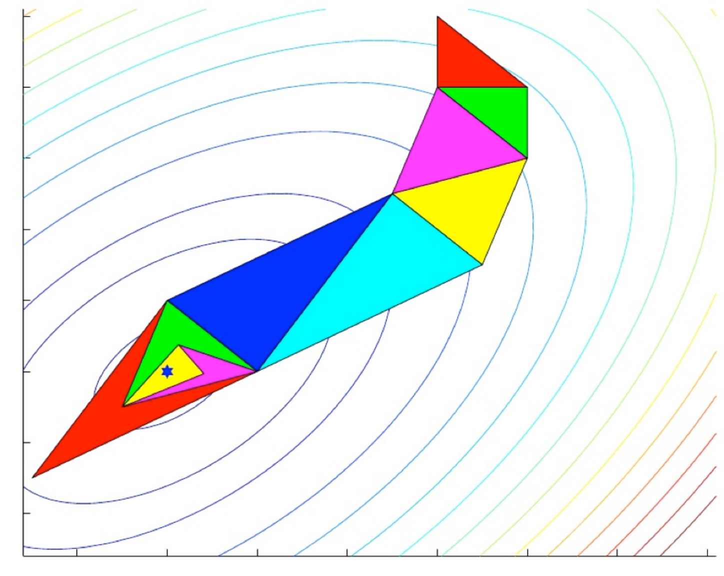
- This is computationally efficient since taking inverse of matrices is very slow.

# Alternative non-derivative-based methods

# Downhill simplex method

- In one-dimensional minimization, possible to bracket a minimum.
- No analogous procedure in multidimensional space.
- **Downhill Simplex Method** by Nelder and Mead (1965):
  - Pros: Requires only function evaluations, not derivatives.
  - Cons: Not very efficient.
- *Simplex:* Geometrical figure consisting, in $N$ dimensions, of $N+1$ points (or vertices) and all their interconnecting line segments, polygonal faces, etc. ($N=2 \rightarrow$ triangle, $N=3 \rightarrow$ tetrahedron)

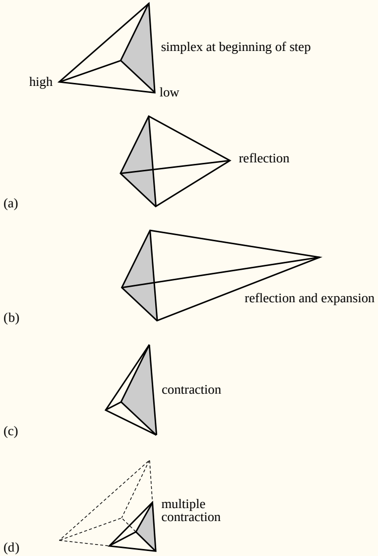
# Graphical explanation



# Algorithm

1. Start with $N + 1$ points $\rightarrow$ Initial simplex.
2. Take one of those points to be initial starting point $\mathbf{P}_0$.
3. Take other $N$ points to be $\mathbf{P}_i = \mathbf{P}_0 + \Delta \mathbf{e}_i$:
    - $\Delta$: Guess of problem's characteristic length scale (possibly $\Delta_i's$ for each vector direction).
    - $\mathbf{e}_i's$: $N$ unit vectors, give direction of where to move.
4. *Reflection* step: Move point of simplex where function is largest through opposite face of simplex to a lower point.
5. Terminate when decrease in value function (or vector distance moved) in last step is fractionally smaller in magnitude than some tolerance.
6. Restart algorithm: Reinitialize $N$ of the $N + 1$ vertices of the simplex again w/ previous equation, w/ $\mathbf{P}_0$ being one of the vertices of the claimed minimum.

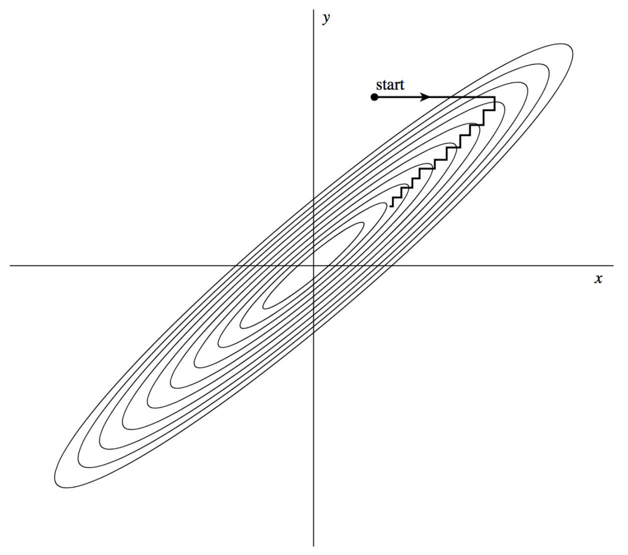
# Different transformations



# Powell's method

- If start at point $\mathbf{P}$ in $N$-dimensional space, and proceed in vector direction $\mathbf{n}$, then any function of $N$ variables $f(\mathbf{P})$ can be minimized along the line $\mathbf{n}$ by one-dimensional methods.
- Simplest case: cyclic coordinate search.
- But efficiency depends on how the next direction $\mathbf{n}$ is chosen.
- **Powell's Method** provides set of $N$ *mutually conjugate directions*.
- Two vectors $\mathbf{u}$ and $\mathbf{v}$ are *conjugate* with respect to $Q$ (or $Q$-orthogonal) if $\mathbf{u}^T\mathbf{Q}\mathbf{v} = 0$.
- Use this set to efficiently perform line minimization (reach minimum after $N$ line minimizations if $f$ quadratic).

# Graphical explanation



# Original algorithm

Initialize the set of directions $\mathbf{u}_i$ to the basis vectors: $\mathbf{u}_i = \mathbf{e}_i$, $i = 0, ..., N-1$.

Repeat following sequence of steps until function stops decreasing:

1. Save your starting position as $\mathbf{P}_0$.
2. For $i = 0, ..., N-1$, move $\mathbf{P}_i$ to the minimum along direction $\mathbf{u}_i$ and call this point $\mathbf{P}_{i+1}$.
3. For $i = 0, ..., N-2$, set $\mathbf{u}_i \leftarrow \mathbf{u}_{i+1}$.
4. Set $\mathbf{u}_{N-1} \leftarrow \mathbf{P}_N - \mathbf{P}_0$.
5. Move $\mathbf{P}_N$ to the minimum along direction $\mathbf{u}_{N-1}$ and call this point $\mathbf{P}_0$.

# Corrected algorithm

**Problem:** throwing away, at each stage, $\mathbf{u}_0$ in favor of $\mathbf{P}_N - \mathbf{P}_0$ tends to produce sets of directions that "fold up on each other" and become linearly dependent.

**Solutions:**

1. Reinitialize the set of directions $\mathbf{u}_i$ to the basis vectors $\mathbf{e}_i$ after every $N$ or $N+1$ iterations of the basic procedure.
2. Reset the set of directions to the columns of any orthogonal matrix.
3. Still take $\mathbf{P}_N - \mathbf{P}_0$ as new direction discarding the old direction along which the function $f(\cdot)$ made its *largest decrease*.

# Simulation methods

# Random walk Metropolis-Hastings I

- We explore a function $f(\cdot)$ by randomly drawing from it.
- Algorithm:
  1. Given a state of the chain $x_{n-1}$, we generate a proposal:
  $$x^* = x_{n-1} + \lambda\varepsilon, \;\; \varepsilon \sim \mathcal{N}(0,1)$$
  2. We compute:
  $$\alpha = \min\left\{1, \frac{f(x^*)}{f(x_{n-1})}\right\}$$
  3. We set:
  $$\begin{aligned} x_n &= x^* \; w.p. \; \alpha \\ x_n &= x_{n-1} \; w.p. \; 1-\alpha \end{aligned}$$
  4. Keep $x_n$ which yields the highest $f(\cdot)$.

# Random walk Metropolis-Hastings II

- Why does it work? Harris recurrence.
- Particularly easy to implement.
- Transformations of $f(\cdot)$.
- More sophisticated proposals.
- Also, it is straightforward to incorporate complex constraints.
- Equivalent to simulated annealing: iteration-varying $\lambda$ ("cooling down").

# Genetic algorithms

- Large class of methods.
- Fraser and Burnell (1970) and Holland (1975).
- Build on two basic ideas of evolution:
  1. Random mutation (sexual or asexual reproduction).
  2. Survival-of-the-fittest.
- Not very efficient set of methods...
- ...but it can handle even the most challenging problems.
- They can be mixed with traditional methods.

# Genetic algorithm basic structure