Computing for Economists

Nonlinear Equation Solving & Derivative-Based Optimization

Toren Fronsdal and Ruby Zhang

Harvard University

Introduction

Without analytical solutions, there are generally two types of problems that need to be computationally solved:

- Root-finding: a set of equations to be solved
- Optimization: minimizing or maximizing a given function

It is easier to perform root-finding, and in many cases, optimization can be turned into a root-finding problem.

- Finding the root of first-order derivatives

Resources

Most of this lecture draws from the following sources:

- Ken Judd Course (Lectures 4 and 5)
- Jesús Fernández-Villaverde Course
- Shunsuke Tsuda Lecture
- Jason DeBacker Course

Table of Contents

Nonlinear Equation Solving

Example: Cournot Duopoly

Two firms $j \in \{1, 2\}$ compete à la Cournot given:

- Inverse demand function $P(Q)$ where $Q = Q_1 + Q_2$
- Cost function $C(Q_j)$

Then firms maximize:

$$\max_{Q_j} P(Q)Q_j - C(Q_j)$$

where the equilibrium is given by satisfying the two equations:

$$0 = \frac{\partial P(Q)}{\partial Q_j} Q_j + P(Q) - \frac{\partial C(Q_j)}{Q_j}$$

Other examples satisfying a set of equilibrium conditions:

- Macro/Trade models
- Game theory and Nash equilibrium

Solving Non-Linear Equations

We will first discuss principles of numerical methods to find the root (i.e. the "zero") of an equation:

$$f(x) = 0$$

- Bisection Method
  - Easy, derivative-free but slow
- Newton's Method
  - Requires differentiability but fast
- Quasi-Newton Methods: first-order approximations
  - Secant/Broyden's Method
  - Broyden's

We will also discuss fixed-point problems (convergence to a point):

$$f(x) = x$$

Bisection Method (One-Dimensional)

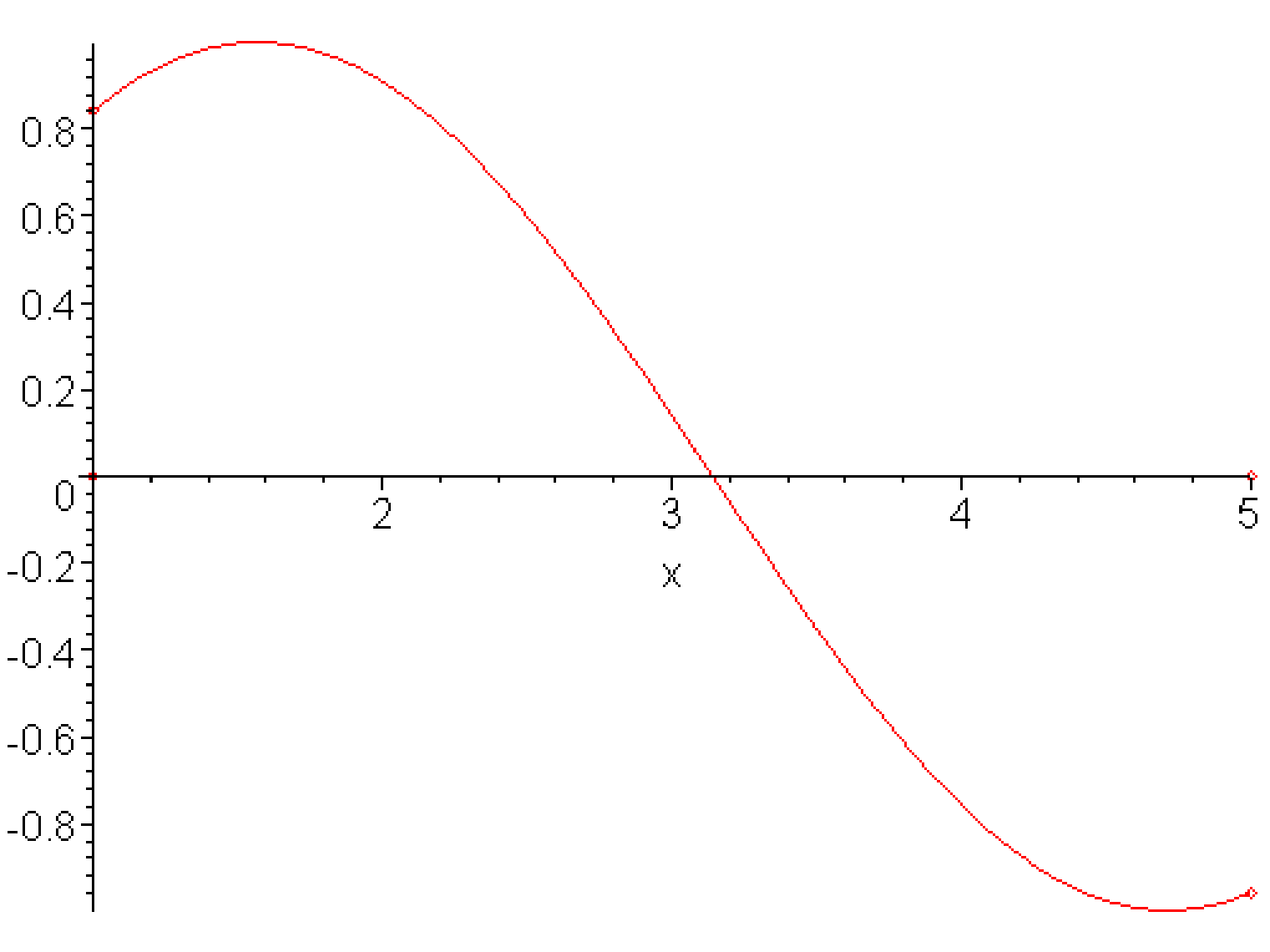
From the Intermediate Value Theorem, given $[a, b]$ s.t. $f(a) < 0 < f(b)$ for a continuous function f, then there is at least one root $x^* \in (a, b)$.

Suppose further that f is monotonic over $[a, b]$, then this guarantees a unique root $x^* \in (a, b)$:

1. Compute midpoint $c = (a + b)/2$
2. Check the following:
   - If $f(c) \approx 0$ for some chosen threshold, then stop
   - If $f(c) > 0$, set $b = c$ and repeat from Step 1
   - If $f(c) < 0$, set $a = c$ and repeat from Step 1

Note: this is reliable and derivative-free, but it can be *very slow*! If there are multiple roots of a function, the range determines which root is found.

Bisection Method (One-Dimensional)



Source: UWaterloo

Newton's Method (One-Dimensional)

Basic Idea: linear functions are easy to solve, and tangent lines are the "best" local linear approximation to a point.

1. Pick an initial guess x_0
2. Our goal is to find the root, x_1, of the tangent line:

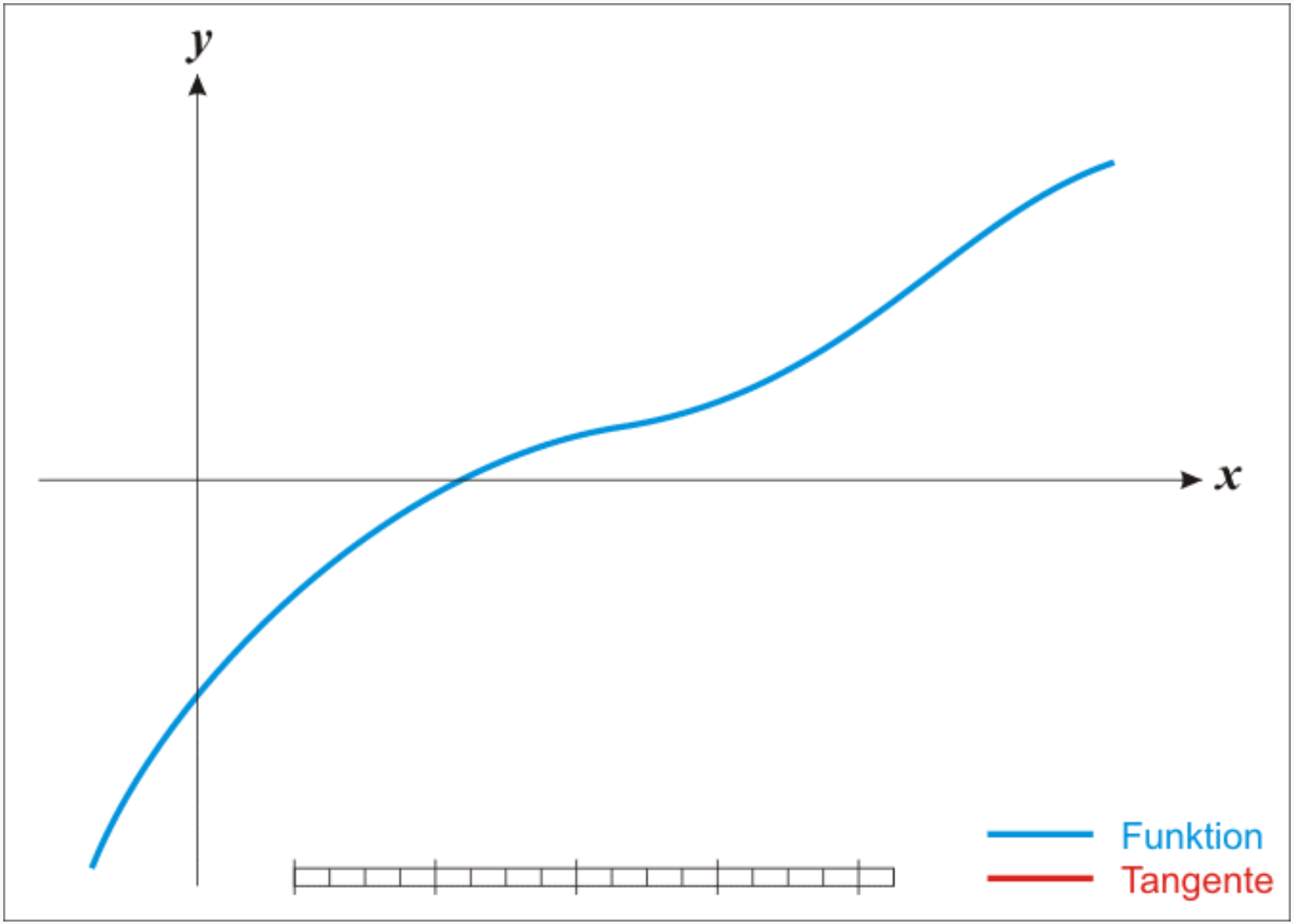
$$0 = f(x_0) + f'(x_0)(x_1 - x_0) \Rightarrow \boxed{x_1 = x_0 - \frac{f(x_0)}{f'(x_0)}}$$

3. If $|x_1 - x_0| > \epsilon$ for some threshold $\epsilon > 0$, repeat step 2 to 3 to get $x_2, x_3, \ldots, x_k$. Otherwise, you found x^* (check that $f(x^*) \approx 0$).

Newton's may not always converge, requires:

- x_0 sufficiently close to x^*
- $f'(x^*) \neq 0$ (i.e. tangent line not horizontal)
- $|f''(x^*)/f'(x^*)| < \infty$ (otherwise divergence)

Newton’s Method (One-Dimensional)



Source: Wikipedia

Newton's Method (Multi-Dimensional)

Similar concept except we're working with a Jacobian (matrix of first derivatives) $\mathbf{J_f}$ and vectors $\mathbf{x}$ instead:

1. Pick an initial guess $\mathbf{x_0}$
2. Find the root, $\mathbf{x_1}$, of:

$$0 = f(\mathbf{x_0}) + \mathbf{J_f}(\mathbf{x_0})(\mathbf{x_1} - \mathbf{x_0}) \Rightarrow \mathbf{x_1} = \mathbf{x_0} - \mathbf{J_f}^{-1}(\mathbf{x_0})f(\mathbf{x_0})$$

3. If $\|\mathbf{x_1} - \mathbf{x_0}\| > \epsilon$ for some threshold $\epsilon > 0$, repeat step 2 to 3. Otherwise, you found $\mathbf{x}^*$ (check that $f(\mathbf{x}^*) \approx \mathbf{0}$)

Note that since we invert the Jacobian, $\mathbf{J_f}(\mathbf{x}^*)$ must exist and $det(\mathbf{J_f}(\mathbf{x}^*)) \neq 0$.

Quasi-Newton Methods

Quasi-Newton methods approximate the first derivative / Jacobian since it is often unavailable and costly to compute.

We can iteratively approximate using finite-differencing (and some initial derivative guess):

- Secant Method (One-Dim):

$$f'(x_k)(x_k - x_{k-1}) \approx f(x_k) - f(x_{k-1})$$

- Broyden's Method (Multi-Dim):

$$\mathbf{J}_{fk}(\mathbf{x}_k)(\mathbf{x}_k - \mathbf{x}_{k-1}) \approx f(\mathbf{x}_k) - f(\mathbf{x}_{k-1})$$

Broyden's Method

- In the multivariate case, the equation is underdetermined (need to solve for a matrix of unknowns from a vector of equations)
- Instead, solve for $\mathbf{J}_{fk}$ which minimizes the modification to $\mathbf{J}_{fk-1}$:

$$\min \|\mathbf{J}_{fk} - \mathbf{J}_{fk-1}\|_F$$

  where F is the Frobenius norm $\|\mathbf{A}\| = \sqrt{\sum_i \sum_j |a_{ij}|^2}$
- Let $\Delta\mathbf{x}_k = \mathbf{x}_k - \mathbf{x}_{k-1}$ and $\Delta\mathbf{f}_k = f(\mathbf{x}_k) - f(\mathbf{x}_{k-1})$
- Then the Jacobian is given by the following updating rule:

$$\mathbf{J}_{fk} = \mathbf{J}_{fk-1} + \frac{\Delta\mathbf{f}_k - \mathbf{J}_{fk-1}\Delta\mathbf{x}_k}{\|\Delta\mathbf{x}_k\|^2}\Delta\mathbf{x}_k{}^T$$

Other Algorithms

We will not cover the following algorithms but will briefly mention here for reference:

- Brent's Method: combine bisection, secant, and interpolation (no derivative)
- Powell's Method: linear search along vectors using Brent's method
- Gaussian methods: Gauss-Jacobi, Gauss-Seidel
  - To find root of multivariate function, successively solve only one component x_i^{k+1} for each of the equations and update

Fixed Point Iteration

- Occurs frequently in economics: Steady-states, value function iteration and Bellman operators
- Note that all fixed point problems are nonlinear equation solving but not vice-versa:

$$f(x) = x \Rightarrow f(x) - x = 0$$

- Fixed point iteration:
  1. Start with some initial value x_0.
  2. Compute the next value iteratively: $x_{k+1} = f(x_k)$.
  3. If $|x_{k+1} - x_k| < \epsilon$ for some small $\epsilon > 0$ then stop. Otherwise repeat the previous step.

Contraction Mapping Theorem

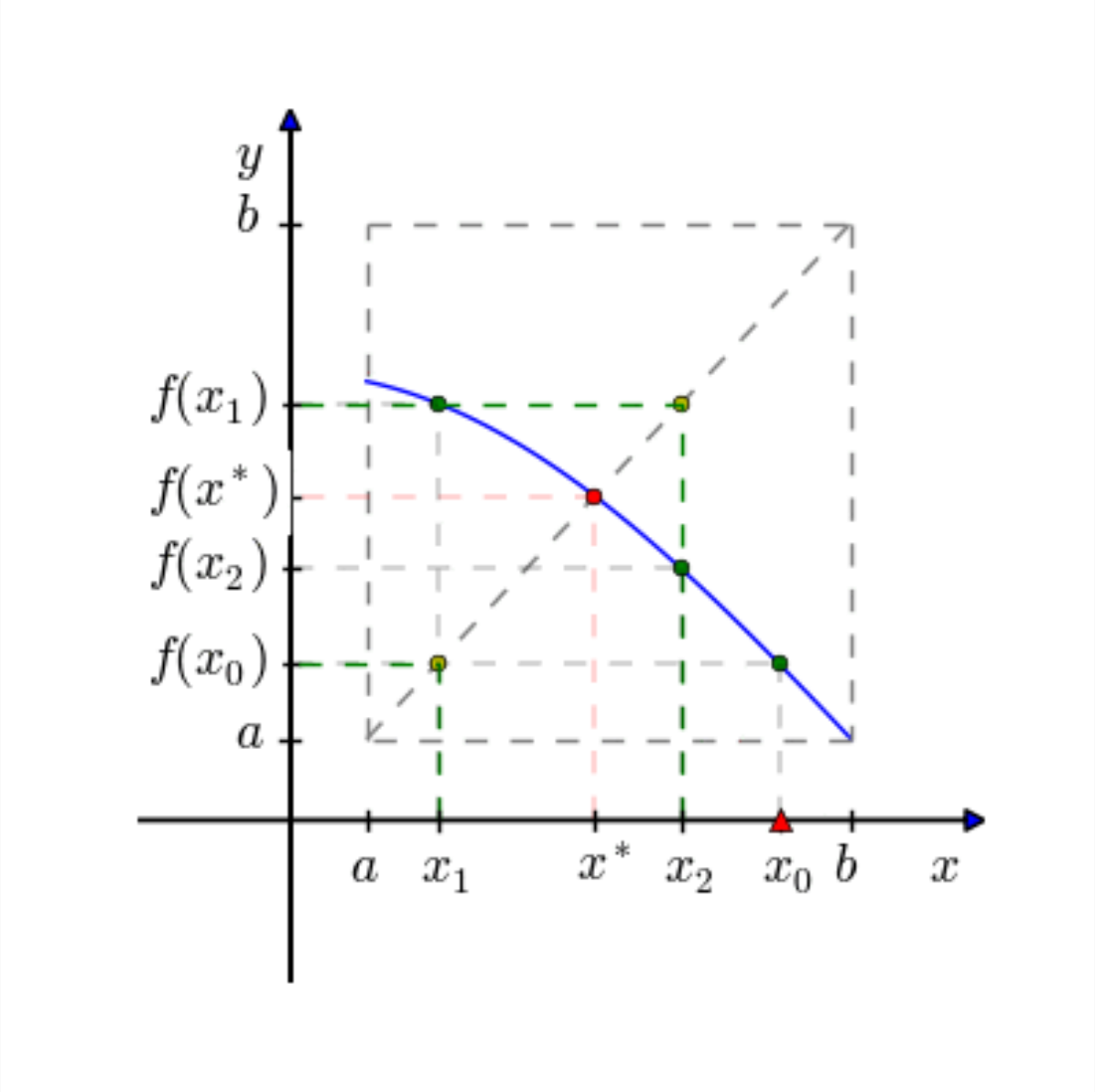
Define a function $g : D \to D$ (where $D \subset \mathbb{R}$) as a *contraction* if there exist constant $k \in [0, 1)$ such that for all $x, y \in D$,

$$|g(x) - g(y)| \leq k|x - y|$$

In other words, $g(.)$ will always bring points in space closer together with each application.

Theorem
If $g : D \to D$ is a contraction on a closed interval D, then g has a unique fixed point $x^* \in D$. Furthermore, for any initial value $x_0 \in D$, iteratively applying $x_{n+1} = g(x_n)$ will converge to x^*.

Fixed Point Iteration in Action



Source: Wikipedia

Summary of Nonlinear Equation Solving Methods

Method	When to Use	Convergence & Suitability
Bisection	- Known continuity - Known sign change	- Guaranteed, slow, linear - Simple, robust, good for preliminary analysis
Newton's	- Differentiable function - Available initial guess	- Fast, quadratic near root - Efficient for well-behaved functions, risky for functions with zero derivatives
Quasi-Newton	- Complex derivatives - High-dimensional problems	- Ideal for complex problems, avoids direct derivative computation
Fixed Point	- Fixed-point form possible - $g(x)$ as a contraction	- Potentially slow, stable for proper $g(x)$s

Introduction to Optimization

What is an optimization problem?

In economics, we commonly see optimization problems in the form of (note all maximization problems are the negative of minimization problems):

$$\min_x \quad f(x) \quad s.t. \quad g(x) = 0, \quad h(x) \leq 0$$

- Objective function $f(x) : \mathbb{R}^n \to \mathbb{R}$
- Equality constraints $g(x) : \mathbb{R}^n \to \mathbb{R}^m$
- Inequality constraints $h(x) : \mathbb{R}^n \to \mathbb{R}^k$

We will first discuss principles of *unconstrained* optimization problems, then go into *constrained* optimization

Example: Maximum Likelihood Estimation (MLE)

Let $(x_1, ..., x_n)$ correspond to realized i.i.d. data drawn from distribution F indexed by F_θ. Let $f(x_i|\theta)$ correspond to the probability of drawing w_i from a population distributed to F_θ.

The goal is to minimize the negative log-likelihood:

$$\hat{\theta}^{MLE} = \arg\min_{\theta} - \sum_{i=1}^{n} \log f(x_i \mid \theta)$$

Probit Example

- Outcome variable y_i is 0 or 1 (e.g. does the treatment group attend college?)
- Given independent variables x_i:

$$\hat{\beta}^{MLE} = \arg\min_{\beta} - \sum_{i=1}^{n} \left[y_i \log \Phi(x_i^T \beta) + (1 - y_i) \log(1 - \Phi(x_i^T \beta)) \right]$$

Example: Generalized Method of Moments (GMM)

MLE is efficient but requires a fully specified distribution.
Instead, given some moment condition,[1]

$$\mathbb{E}[g(X_i, \theta_0)] = 0$$

Then given weighting matrix $\hat{C}$ and data realizations $(x_1, ..., x_n)$, we minimize the GMM objective function:

$$m_n(\theta) = \frac{1}{n} \sum_{i=1}^{n} g(x_i, \theta)$$

$$\hat{\theta}^{GMM} = \arg\min_{\theta} m_n(\theta)' \hat{C} m_n(\theta)$$

GMM is the foundation of structural estimation!

[1] For example, we have a linear regression and instrument Z_i s.t. $Z_i \perp \epsilon_i$, then our moment is $\mathbb{E}[Z_i(Y_i - X_i\theta_0)] = 0$

Categories of Optimization

There are two broad methods of solving numerical optimization problems:

1. **Derivative-Based Methods:**
   - First-order derivative: Gradient descent
   - Second-order derivative: Newtonian methods
2. **Derivative-Free Methods:**
   - Brute force: Grid search
   - Heuristic/Pattern search: Nelder-Mead

Optimization Methods

We will discuss the following numerical optimization methods:

- Derivative-Based Methods
  - Gradient Descent
  - Newton's Method
  - Quasi-Newton Methods
    - Broyden-Fletcher-Goldfarb-Shanno (BFGS)
- Derivative-Free Methods
  - Grid Search
  - Downhill Simplex Method (Nelder-Mead)

Derivative-Based Optimization

Gradient Descent (Steepest Descent)

Idea: If there's a first derivative, then follow the direction of steepest descent, and take a certain step size.

1. Starting at point x_0, find the gradient $\nabla f(x_0)$ and move in the opposite direction $-\nabla f(x_0)$.
2. Let α_0 denote the step size. Then we want to solve the one-dimensional problem:

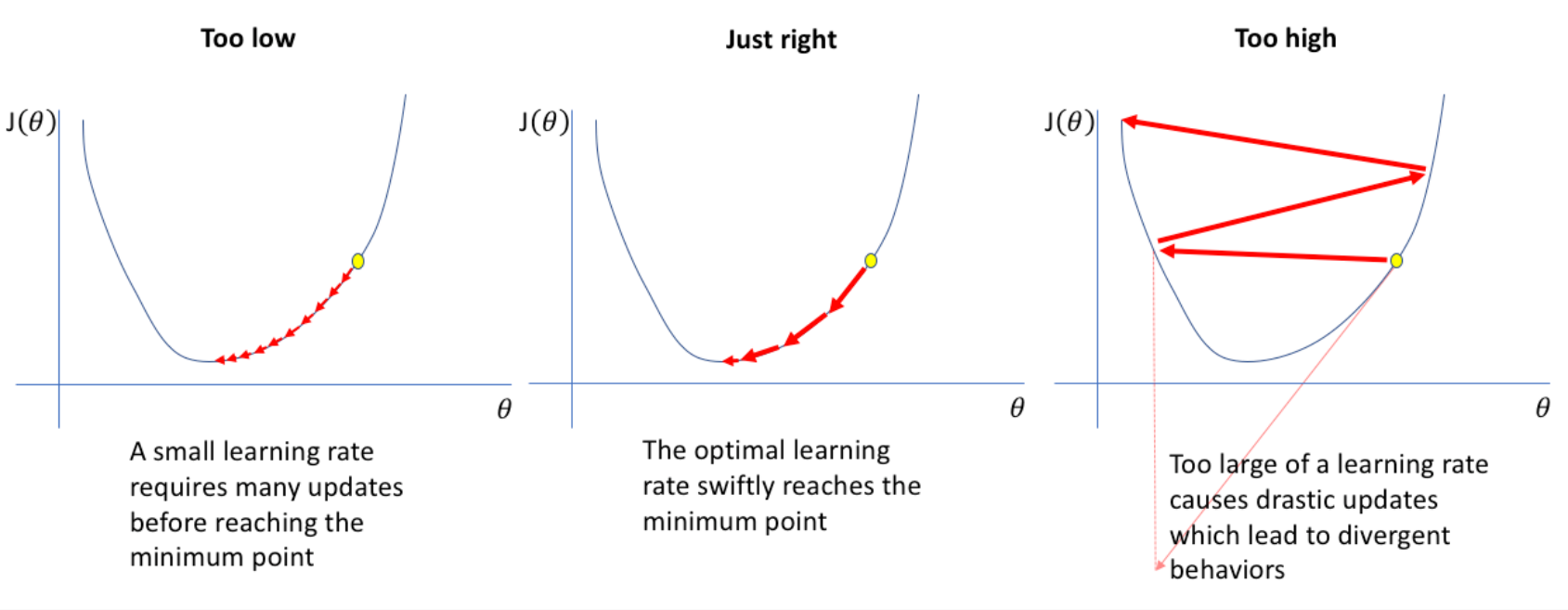
$$\min_{\alpha_0} f(x_0) - \alpha_0 \nabla f(x_0)$$

   Backtracking line search has good theoretical guarantees[2]
3. Repeat until $|x_{k+1} - x_k| < \epsilon$

There's a tradeoff between step size and robustness $\rightarrow$ descending slowly and carefully.

[2]This is an inexact line search satisfying the Armijo rule, part of the Wolfe conditions.

Step-Size Tradeoff



Source: Edouard Duchesnay

Gradient Descent Methods

Gradient descent is commonly used in machine learning where the gradient is numerically calculated. There are a few methods to reduce computational cost:

- **Stochastic Gradient Descent (SGD)**: compute gradient using one training example. Faster but larger, erratic steps (hence stochastic).
- **Mini-batch Gradient Descent**: compute gradient based on subset of training examples. Tradeoff between speed and accuracy.
- **Adaptive Methods**: there are more methods that combine historical gradients or a moving average to continuously adapt the learning rate (such as Adagrad, RMSprop, and Adam).

Newton's Method

This is similar to Newton's method in non-linear equation solving, except now the "root" is the FOC of the original function:

- Given $f(x)$ to minimize, find all x^* s.t. $f'(x^*) = 0$
- Need to check that $f''(x^*) > 0$ for a local minimum
- Requires **twice-differentiability**
- Second-order descent method where the **Hessian gives step size**

Univariate updating:

$$0 = f'(x_0) + f''(x_0)(x_1 - x_0) \Rightarrow \boxed{x_1 = x_0 - \frac{f'(x_0)}{f''(x_0)}}$$

Note: this is also the FOC of a second-order Taylor approximation around x_0:

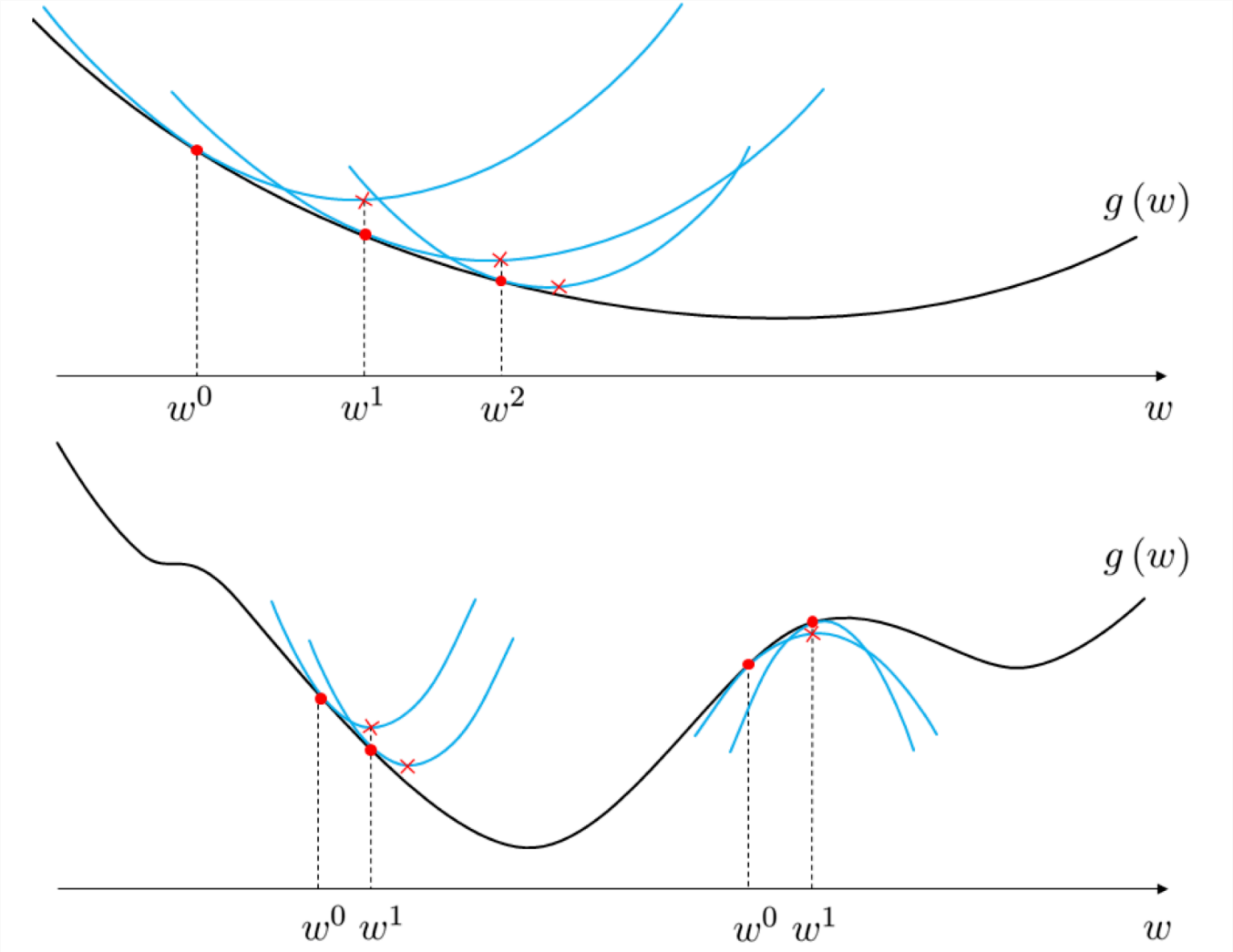
$$f(x_1) \approx f(x_0) + f'(x_0)(x_1 - x_0) + f''(x_0)/2(x_1 - x_0)^2$$

Multivariate updating (Hessian: $H_f(x)$, gradient: $\nabla f(x)$):

$$0 = \nabla f(\mathbf{x_0}) + \mathbf{H_f}(\mathbf{x_0})(\mathbf{x_1} - \mathbf{x_0}) \Rightarrow \boxed{\mathbf{x_1} = \mathbf{x_0} - \mathbf{H_f}^{-1}(\mathbf{x_0})\nabla f(\mathbf{x_0})}$$

Newton's Method (One-Dimensional)

Parabolic (i.e. second-order) Taylor approximation:



Source: Machine Learning Refined

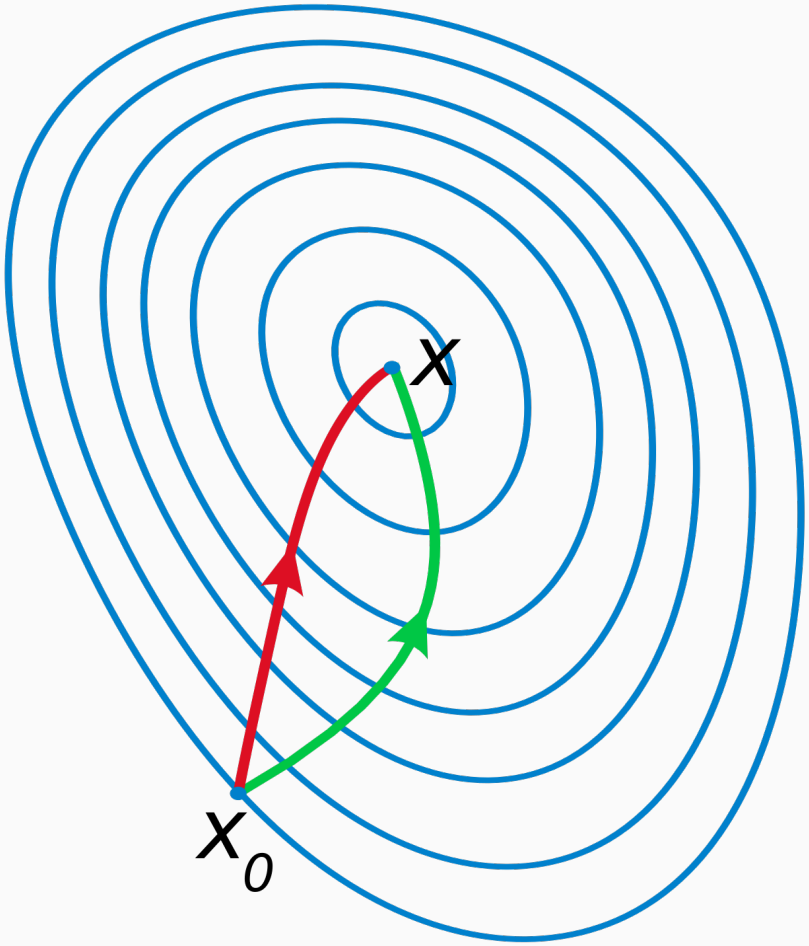
Newton's Method vs. Gradient Descent

Newton's is **faster** but requires the **Hessian**, which can be burdensome for high-dimensional problems:

Newton's Method

Source: Wikipedia

Gradient Descent



Quasi-Newton Methods

However, derivatives and matrix inverses can be very costly to compute, Newton's method of $\mathcal{O}(n^3)$. Quasi-Newton methods refer to using approximate Hessians (or its inverse).

- First-order approximation of Hessian (let $H_k = H_f(x_k)$):

$$\nabla f(x_{k+1}) \approx \nabla f(x_k) + H_k(x_{k+1} - x_k)$$

$$\boxed{H_k(x_{k+1} - x_k) = \nabla f(x_{k+1}) - \nabla f(x_k)}$$

- Need to find H_{k+1} which is close to H_k and symmetric, positive-semidefinite

Broyden-Fletcher-Goldfarb-Shanno (BFGS)

BFGS approximates the Hessians using a generalized secant method. Let:

$$s_k = x_{k+1} - x_k$$

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

BFGS adds two symmetric, positive-definite matrices (scaled by the denominator) to the previous Hessian:

$$H_{k+1} = H_k + \frac{y_k y_k^T}{y_k^T s_k} - \frac{H_k s_k s_k^T H_k^T}{s_k^T H_k s_k}$$

Using the Sherman-Morrison formula, the inverse of the updated Hessian can be directly computed using last step's inverse. This makes BFGS considerably faster at $\mathcal{O}(n^2)$.

Python Implementation

SciPy Library

- *scipy.optimize* contains all of Python's standard root-finding and minimization algorithms.[3]
- All listed functions will begin with *scipy.optimize.[function]*.

General tips:[4]

- Set very small tolerances, double loops appear all the time in structural estimation (10^{-14} & 10^{-8} for inner & outer loops)
- Ideally, check the SOCs for positive definiteness (i.e. Hessian eigenvalues > 0)
- Write code to print objective, errors, gradient norm, etc. for each loop. Configure your optimizer!

[3] Tutorial of each function can be found here.
[4] Courtesy of Jeff Gortmaker.

Root-Finding in Python

Univariate root-finding:

- `bisect()`
- `newton()`
- `brentq()`
- `brenth()`
- `ridder()`

There's also `fixed_point()`

Multivariate root-finding (subset):

- `root(method='hybr')`
- `root(method='lm')`
- `root(method='broyden1')`
- `root(method='broyden2')`
- `root(method='linearmixing')`

Minimization in Python

`scipy.optimize.minimize()` contains Python's minimizers, most come with the option of constraints:

- General:
  - `minimize(method='Newton-CG')`
  - `minimize(method='L-BFGS-B')`
  - `minimize(method='Nelder-Mead')`
- Constrained:
  - `minimize(method='trust-constr')`
  - `minimize(method='SLSQP')`
  - `minimize(method='COBYLA')`