Data Science for Economists

Lecture 9: Intro to Numerical Methods

Alex Marsh
University of North Carolina | ECON 390

Table of contents

Introduction

Agenda

Today we will cover numerical methods and optimization.

This lecture might be a bit more theory heavy than programming heavy.

What is important here is to understand the concepts rather than the details.

- Unless you specialize in computational methods for economics later on, you will likely not need to understand the details.

Motivation

- There are many times there is an equation we would like to solve or know the value of.
- While ideally, we would solve these analytically to get an exact solution, for any non-trivial problem, this is either incredibly tedious or not possible.
- Today we will cover a handful of methods that you might need at some point.
- While some topics will relate within this lecture, most likely, they will feel disjoint and spattered.
- Understand that I am teaching a handful of tools that you will likely need at some point.

Introduction

Review

(Some important math and stat concepts)

Math Review: Derivative

- The derivative of a function tells you about it's rate of change.
  - "Instantaneous slope" (this is actually nonsensical the more one thinks about it)
- We denote the derivative of f with respect to x a few different ways:
  - $f'(x)$
  - $\frac{df}{dx}$
- The derivative tells us more than just the slope:
  - $f'(x) > 0 \rightarrow$ function is increasing
  - $f'(x) < 0 \rightarrow$ function is decreasing
  - $f'(x) = 0 \rightarrow$ function is not changing i.e. "critical point"
- The formal definition of a derivative is as follows:

$$f'(x) = \lim_{h \to 0} \frac{f(x+h) - f(x)}{h}$$

- We will actually be using this definition!

Math Review: Integrals

- Integrals calculate the area "under the curve"
- More broadly, they are used when you want to "sum up" a lot of very small things or take the average value of some function.
- The formal definition of the integral is

$$\int_a^b f(x)dx = \lim_{n\to\infty} \sum_{i=1}^{n} f(x_i)\Delta x$$

  where $a \leq x_1 < x_2 < \ldots < x_n \leq b$ and $x_{i+1} - x_i = \Delta x = (b-a)/n$
- We will be using this definition!

Intro to Numerical Methods

Approximating Derivatives

- While getting the analytical expression for $f'(x)$ is ideal due to accuracy and computational demand, sometimes it is not feasible to do so.
- Instead, we can approximate the derivative!
- Remember that

$$f'(x) = \lim_{h \to 0} \frac{f(x+h) - f(x)}{h}$$

.

- So for a "small" $h > 0$,

$$f'(x) \approx \frac{f(x+h) - f(x)}{h}$$

- This is the "forward differencing" approach; we can also "backward difference"

$$f'(x) \approx \frac{f(x) - f(x-h)}{h}$$

- These works, but we can get a more accurate approximation for a fixed h.

Approximating Derivatives (Cont.)

- Instead of forward or backward differencing, we can center difference:

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

- Instead of favoring one side, we are approximating the derivative equally around a "neighborhood" of x.
- It is possible to show that the error generator by center differencing is smaller than right or left differencing.
- There are even more accurate methods that a bit more computationally demanding.
  - For those curious, look up Richardson's Extrapolation Method.

Example

Let's keep things simple and look at the derivative of $f(x) = x^2$. The analytical expression of the derivative is $f'(x) = 2x$.

```
f            = function(x){x^2}           #make function f(x)
fp           = function(x){2*x}           #make f'(x)
xs           = seq(0,2,0.5)               #store x's
h            = 0.01                       #store step size
der          = fp(xs)                     #calc actual derivatives
center_der   = (f(xs+h)-f(xs-h))/(2*h)    #calc center differenced
forward_der  = (f(xs+h)-f(xs))/(h)        #calc forward differenced
backward_der = (f(xs)-f(xs-h))/(h)        #calc backwards differenced
deriv_data   = data.table("f'(x)"=der,"Center"=center_der,
                        "Foward"=forward_der,"Backward"=backward_der)
deriv_data
```

```
##     f'(x) Center Foward Backward
## 1:      0      0   0.01    -0.01
## 2:      1      1   1.01     0.99
## 3:      2      2   2.01     1.99
## 4:      3      3   3.01     2.99
## 5:      4      4   4.01     3.99
```

Derivatives in Multiple Dimensions

- If instead of $f(x)$, we have something like $f(x_1, x_2, \ldots, x_n)$, not much changes.
- The derivative will be a vector called the gradient denoted

$$\nabla_f = \Big(\frac{\partial f}{\partial x_1}, \ldots, \frac{\partial f}{\partial x_n}\Big)$$

- To approximate each (partial) derivative, follow the same approach.
- The partial derivative of f in the x_i dimension at $x = (x_1, \ldots, x_n)$ is,

$$\frac{\partial f}{\partial x_i} \approx \frac{f(x_1, \ldots, x_i + h, \ldots, x_n) - f(x_1, \ldots, x_i - h, \ldots, x_n)}{2h}$$

- Do this for all inputs and then collect them in a vector.
- This requires evaluating the function $2n$ times.

Example

Suppose that $f(x,y) = x^2 + \sin(y)$. $\frac{\partial f}{\partial x} = 2x$ and $\frac{\partial f}{\partial y} = \cos(y)$

```
f              = function(x){x[1]^2+sin(x[2])}        #make function f(x)
fp             = function(x){c(2*x[1],cos(x[2]))}     #make gradient of f(x)
h              = 0.01                                 #store step size
fp(c(1,0.5))
```

```
## [1] 2.0000000 0.8775826
```

```
c((f(c(1+h,0.5))-f(c(1-h,0.5)))/(2*h),(f(c(1,0.5+h))-f(c(1,0.5-h)))/(2*h))
```

```
## [1] 2.0000000 0.8775679
```

```
c((f(c(1+h,0.5))-f(c(1,0.5)))/h,(f(c(1,0.5+h))-f(c(1,0.5)))/h)
```

```
## [1] 2.0100000 0.8751708
```

```
c((f(c(1,0.5))-f(c(1-h,0.5)))/h,(f(c(1,0.5))-f(c(1,0.5-h)))/h)
```

```
## [1] 1.990000 0.879965
```

Approximating Integrals

- Just like derivatives, sometimes we might need to approximate an integral.
- Unlike derivatives, sometimes integrals don't have a closed-form expression at all!
- Therefore, all we can get for integrals is a numerical approximation.
- Remember that

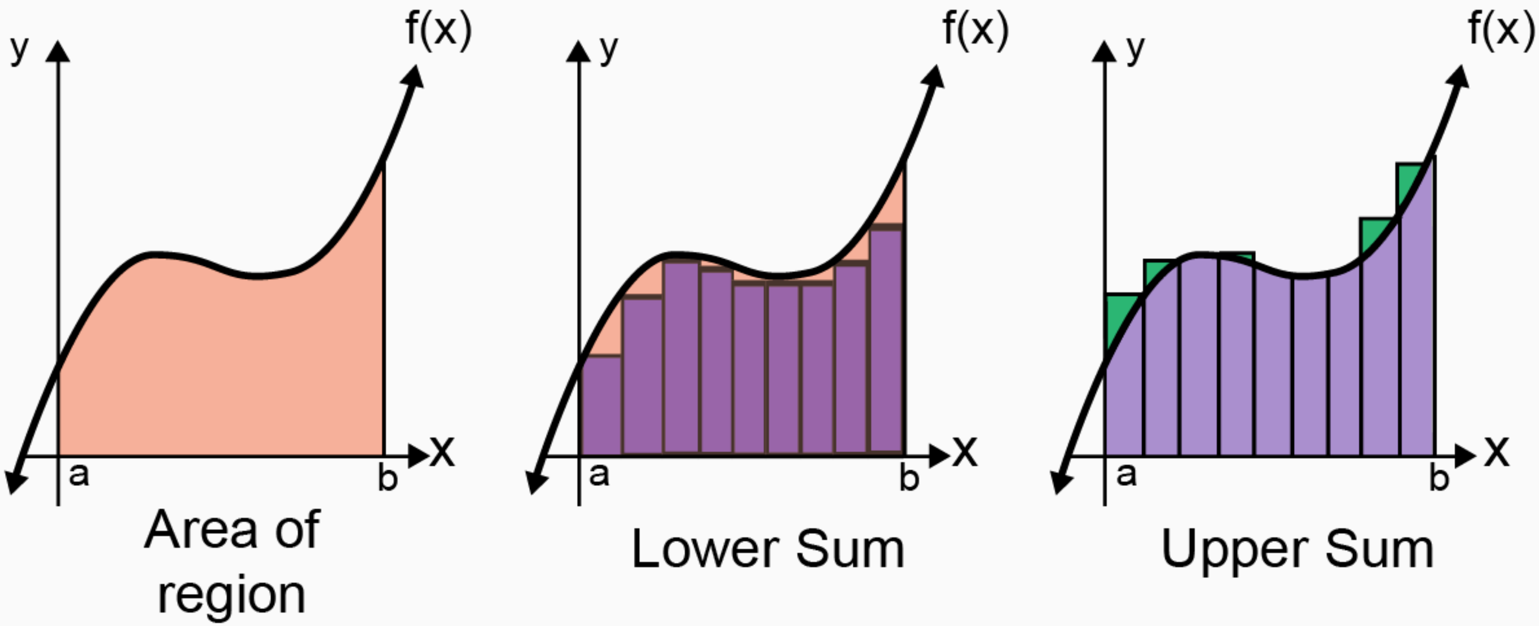
$$\int_a^b f(x)dx = \lim_{n\to\infty} \sum_{i=1}^{n} f(x_i)\Delta x$$

- So for certain "partition" of points, $a \leq x_1 < x_2 < \ldots < x_n \leq b$ and $x_{i+1} - x_i = \Delta x = (b-a)/n$,

$$\int_a^b f(x)dx \approx \sum_{i=1}^{n} f(x_i)\Delta x$$

- Note that a larger n makes this more accurate but means f must be evaluated n-times.
- Also note that depending on if we include a or b as a point makes this an upper sum or a lower sum.
  - Upper sums overestimate the integral.
  - Lower sums underestimate the integral.
  - Which to choose?

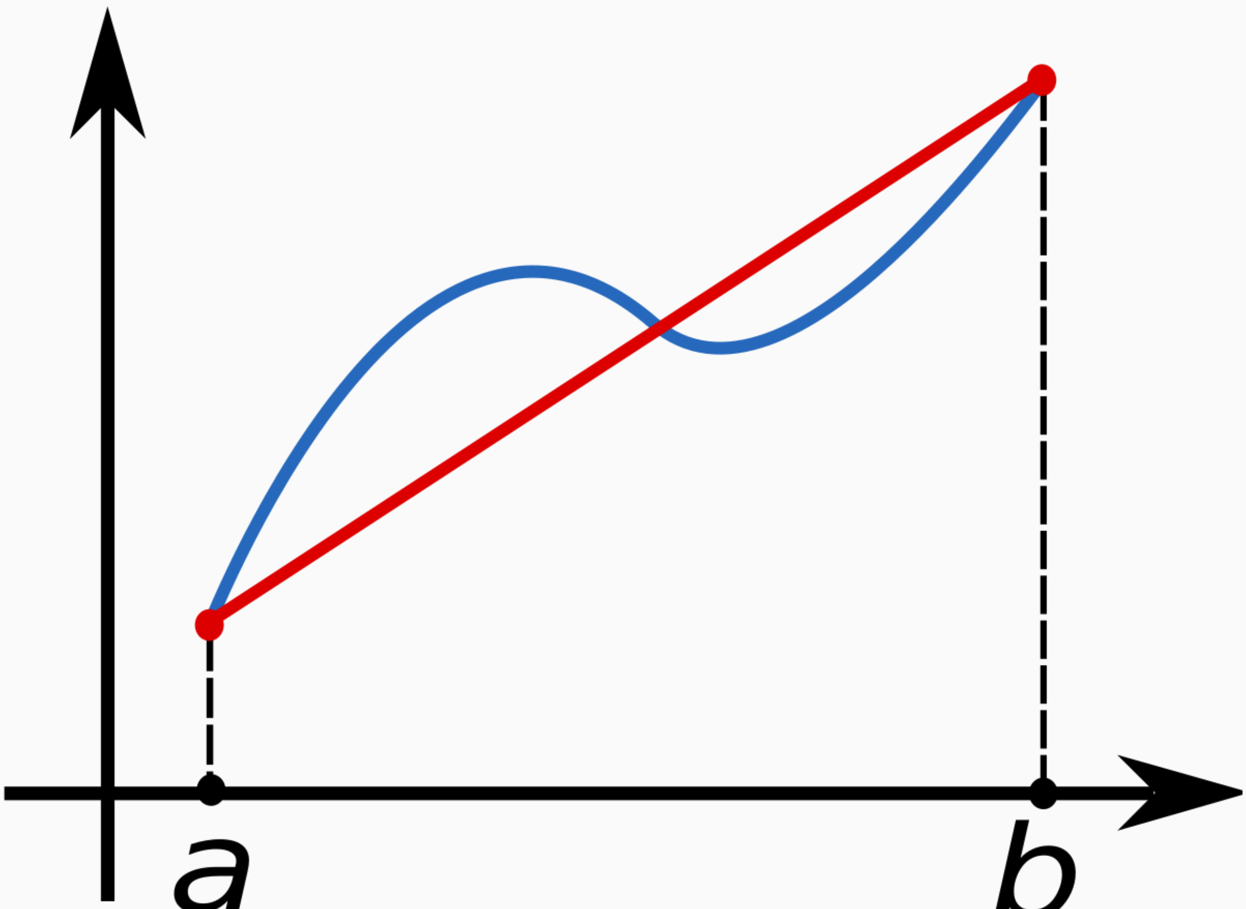
Upper sums and lower sums



Calcworkshop.com

The Trapezoidal Rule

- We don't need to pick upper or lower sums, we can use both! Sorta.
- Instead of rectangles, we use trapezoids.



The Trapezoidal Rule

- Each of the trapezoids is $\frac{a+b}{2}h$
- Here $h = \Delta x_i$, $a = f(x_{i-1})$, $b = f(x_i)$, $\Delta x_i = x_i - x_{i-1}$.
- So the area of each trapezoid is $\frac{f(x_{i-1})+f(x_{i1})}{2}\Delta x_i$
- Therefore, we know

$$\int_a^b f(x)dx \approx \sum_{i=1}^{n} \frac{f(x_{i-1}) + f(x_i)}{2}\Delta x_i$$

- The above expression works for an arbitrary grid i.e. Δx_i need not equal Δx_{i+k}
- If the grid is uniform, the above becomes a simple expression:

$$\sum_{i=1}^{n} \frac{f(x_{i-1}) + f(x_i)}{2}\Delta x_i = \frac{\Delta x}{2}(f(x_0) + 2f(x_1) + \ldots + 2f(x_{n-1}) + f(x_n))$$

- This approximation is more accurate than the upper or lower sums.

Example

Suppose $f(x) = x^2$ and we want to evaluate $\int_0^{1.5} f(x)dx$.

```
f              = function(x){x^2}          #make function f(x)
Fx             = function(x){x^3/3}
trap_rule = function(func,a,b,n=20){
  dx = (b-a)/n
  xs = seq(a,b,dx)
  val = sum(2*func(xs[-c(1,n)]))
  val = val + func(xs[1])+func(xs[n])
  val*dx/2
}
true_val = Fx(1.5)-Fx(0)
my_approx_val20 = trap_rule(f,0,1.5,n=20)
my_approx_val100 = trap_rule(f,0,1.5,n=100)
approx_val = integrate(f,0,1.5)[[1]]
c(true_val,my_approx_val20,my_approx_val100,approx_val)
```

```
## [1] 1.125000 1.134633 1.125392 1.125000
```

Multivariate Integration

- Unlike derivatives, multidimensional integration becomes difficult quickly.
- One approach to multidimensional integration is actually Monte Carlo simulation, which we covered last lecture.
- A large class of multidimensional integration needed for probability distributions can be evaluated using Markov Chain Monte Carlo (MCMC) techniques.
  - Some examples include the Metropolis-Hastings algorithm and Gibbs Sampling.
  - These are needed for Bayesian inference.
  - Beyond the scope of this class, but worth looking into on your own.

Solving for Roots

- Many times you'll want to solve an equation numerically.
  - This may because the closed-form solution is impossible or just tedious.
- When this is, we say we want to find the roots (i.e. x values) s.t.

$$f(x) = 0$$

- While we will cover the simplest methods to do so, the important thing to remember is the motivation and when you might want to use such techniques.
  - Better programmers than you have programmed faster and more robust methods.
- Dates back to the Babylonians and calculating square roots:

$$\sqrt{a} = x$$

or

$$x^2 - a = 0$$

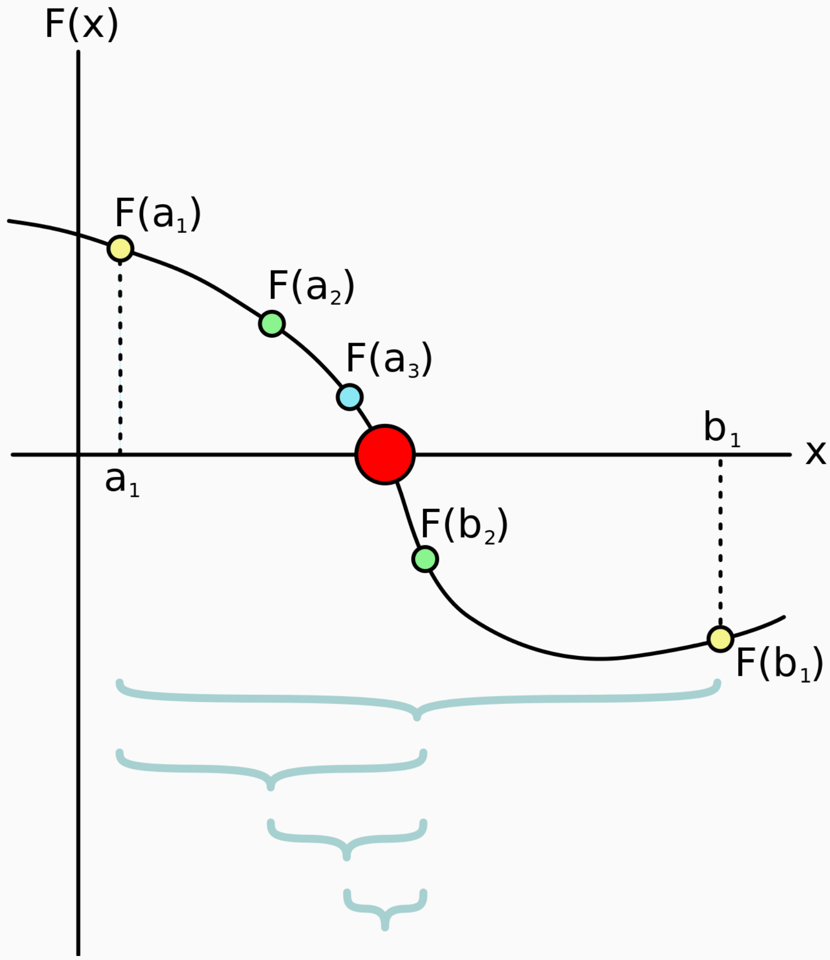
Bisection Method

- One of the two methods we will cover here is the bisection method.
- While this method will always work, it can be slow.
- In order to use the method, you must have starting values a and b where

$$f(a) < 0 < f(b) \text{ or } f(a) > 0 > f(b)$$

- The idea is we take the midpoint between a and b, which we cal c.
- If $\mathrm{sign}(f(c)) = \mathrm{sign}(f(a))$, replace a with c, otherwise b with c.
- Continue until $|f(c)| < \varepsilon$ where ε is a tolerance value.

Bisection Method



Reflection

- The bisection method works, but it is slow and sorta fancy "guess and check."
- We are only using the sign of the function to update the guesses.
- If we could incorporate more information of the function into the search process, it might be faster.
- Idea: Can we use the derivative of the function to tell us where to go next?
- This is known as Newton's method and it was the first improvement

Newton's Method

- Equation of tangent line in point-slope form at x:

$$f(x) - f(x_n) = f'(x_n)(x - x_n)$$

- Set $f(x) = 0$, replace x with x_{n+1}, and solve for x:

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

- So make an initial guess x_0, form $x_1 = x_0 - \frac{f(x_0)}{f'(x_0)}$.
- Keep updating x_{n+1} until $\left|f(x_{n+1})\right| < \varepsilon$
- Question: What technique that we learned earlier will most likely be needed for this method?

Comparing Methods

```
f = function(x){x^2-2}
bisect_results = biscetion(f,0,2)
newton_results = Newtons(f,2)
bisect_results
```

```
## [1]  1.414214 47.000000
```

```
newton_results
```

```
## [1] 1.414214 5.000000
```

Fixed Points

- Many times in economics, we are interested in finding a fixed point, which is defined as

$$f(x) = x$$

  or

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

- Example: Supply and Demand equlibirum can be written as

$$P^S(Q^d(p))) = p$$

- At it's core, there is nothing different for solving fixed points.
  - Define $g(x) = f(x) - x$ and solve $g(x) = 0$.
- However, for some special functions, you can solve $f(x) = x$ via fixed point iteration.
- If the function f is a contraction, then we can simply iterate
  - $x_1 = f(x_0)$
  - $x_2 = f(x_1)$
  - $x_{n+1} = f(x_n)$
- What is a contraction?

Contractions

- Example: $f(x) = \cos(x)$

```
x0 = seq(0,2*pi,0.25)
xn = x0
for(n in 1:100){
  xnp1 = cos(xn)
  xn   = xnp1
}
xnp1
```

```
##  [1] 0.7390851 0.7390851 0.7390851 0.7390851 0.7390851 0.7390851 0.7390851
##  [8] 0.7390851 0.7390851 0.7390851 0.7390851 0.7390851 0.7390851 0.7390851
## [15] 0.7390851 0.7390851 0.7390851 0.7390851 0.7390851 0.7390851 0.7390851
## [22] 0.7390851 0.7390851 0.7390851 0.7390851 0.7390851
```

- Unfortunately, most functions are not contractions.
- However, there are some mathematical objects that are universally contractions and if you go onto graduate studies, you will experience them.
  - Dynamic programming and value functions.

Multivariate Root Finding

- You might need to find

$$f_i(x_1, \ldots, x_n) = 0 \text{ for } i = 1, \ldots, n$$

- There are methods similar to the ones we have already discussed for solving these problems.
- However, they are much outside the scope of this class.
- For now, it is sufficient to know that there exists a package in `R` for solving these problems: `nleqslv`.
- While the algorithms are more complex, the ideas remain the same.
- You will need `nleqslv` for the problem set.

Optimization

- The last numerical methods topic that we will cover is optimization.
- An optimization problem is one that can be written as follows:

$$\min_x f(x)$$

- Note, if we want to maximize, that's the same as minimizing $-f(x)$.
- We want to find the x value that minimizes $f(x)$.
- Sometimes, these will result in closed form solutions (e.g. OLS), but usually will be oto complicated.
- We write the solution to this optimization problem as

$$x^* = \arg\min_x f(x)$$

- For functions that are differentiable, they are optimized at $f'(x) = 0$.
  - Look familiar?
- This is just root finding of $g(x) = 0$ where $f'(x) = g(x)$.

General Issues That Araise

- The first big issue that arises is that maxs, mins, and saddle points all have $f'(x) = 0$.
- Also, there can be local minimums if the function is not globally convex.
- This makes numerical optimization difficult!
- Using the second derivative/Hessian matrix can help solve some of these issues, but it is very computationally expensive to compute.
- As such, no one technique will always work.
- Lots of time and research goes into figuring out the best way to optimize one specific optimization problem.
    - E.g. check out this best practices paper for one model used in my subfield.
- Sometimes an entire project can live or die depending upon if you can optimize the objective function correctly.
    - E.g. one of my current projects...

Grid Search

- If the dimension of x is small enough, you can make a grid of points to search on and see which set of values minimizes f.
- While this seems simple, in practice, you rarely want to do it.
- If x has more than two dimensions, must search many points.
  - Particularly bad of f takes awhile to run.
- Must make grid fine enough so that you're not skipping over too many points, but too fine runs into the same problem as before.

```
x1 = seq(0,5,0.05)
x2 = seq(0,5,0.05)
x3 = seq(0,5,0.05)
nrow(expand.grid(x1,x2))
```

```
## [1] 10201
```

```
nrow(expand.grid(x1,x2,x3))
```

```
## [1] 1030301
```

Newton's Method... Again!

- We can actually use Newton's Method to solve optimization problems.
- Since we want to solve $f'(x) = 0$ and using $g(x) = f'(x)$, apply the formula from before:

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

or

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

- Note that we must compute a derivative and a second derivative for each iteration.
- Computing the derivative requires evaluating the function twice.
- Computing the second derivative requires evaluating the function four times.
- Generally, if you have n inputs, you have to evaluate the function $4n^2/2$ times for the Hessian.
- So while this works, if your objective function takes awhile to run or has a lot of inputs, it might not be ideal.
  - Usually not ideal...

Gradient Descent

- All the second derivative does is scale the updating process so we don't "learn" too fast or too slow.
- Idea: Instead of calculating the scaling amount via the second derivative, we just set some paramter $\alpha > 0$ to update

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

- This is the idea behind gradient descent.
- Choosing α:
    - Sometimes α is just fixed at some small value.
    - However, you can also pick an optimal α:
    1. Calculate $f'(x_n)$ and save it.
    2. Then choose α to minimize $f(x_n - \alpha f'(x_n))$.
    3. Repeat this step each time you update x_{n+1}.
    - Whether this is beneficial is problem specific.
    - You have to solve a smaller optimization problem during your larger optimization problem.

Other Methods

- There are other methods.
    - Quasi-Newton: Calculate a function $B(x_n)$ that approximates the Hessian but is easier to calculate.
    - Derivative free methods:
    1. Nelder-Mead
    2. Simulated annealing
    3. BOBYQA,COBYLA
- Which method to choose?
    - Problem specific!
    - Depends on the properties of f, how long it takes f to run, how many inputs f has, etc.
    - Unfortunately, there is no one answer.
    - Optimization can make or break a project.

Other Methods

- Robustness: Performs well for various problems and starting values.
- Efficiency: Achieves the solution relatively quickly.
- Accuracy: Identify a solution with precision, not sensitive to starting values.

Up Next: Data Wrangling