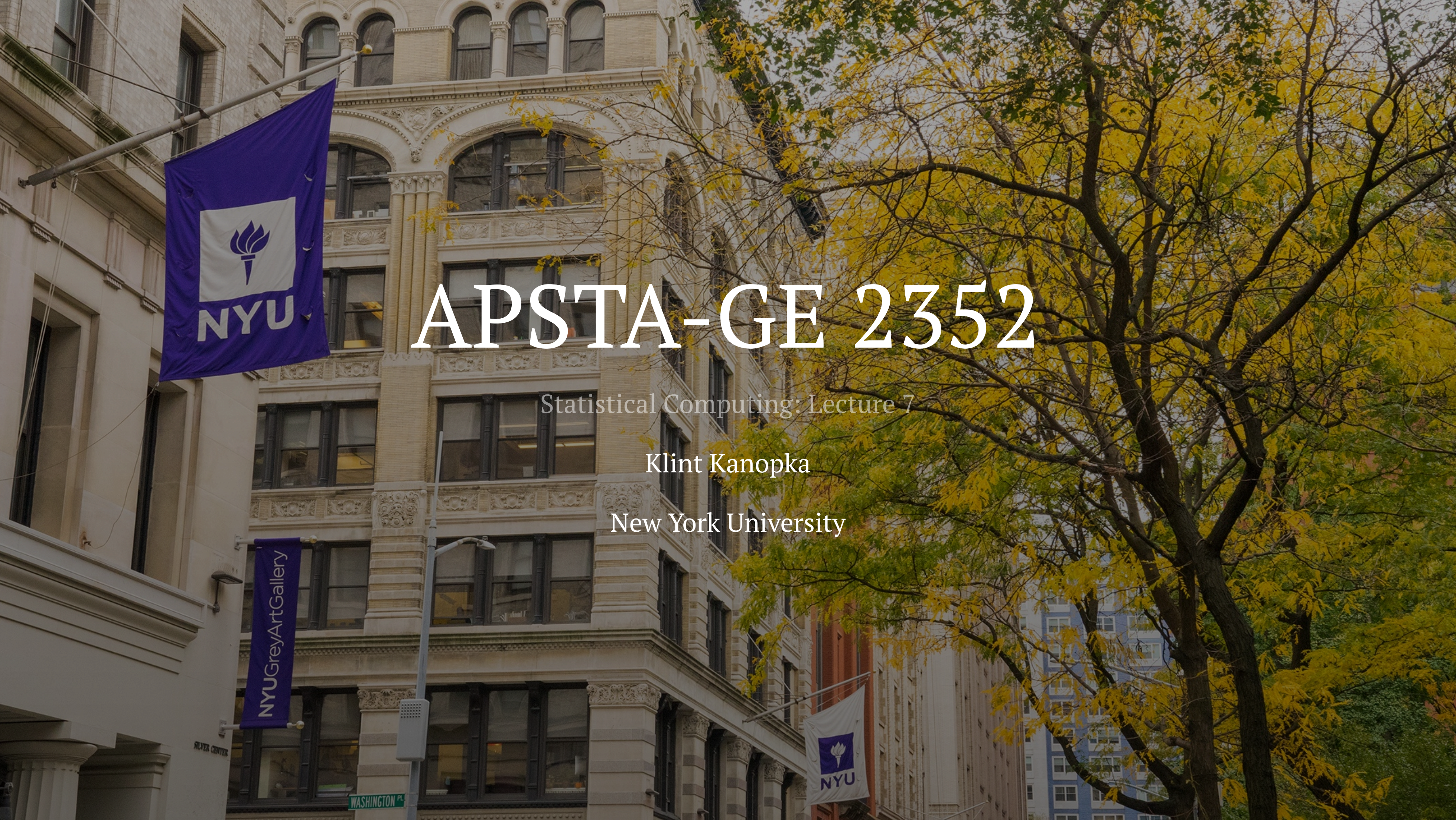


APSTA-GE 2352

Statistical Computing: Lecture 7

Klint Kanopka

New York University

Table of Contents

Announcements

- PS3 is now late
- PS4 is posted, due in two weeks
  - This is all numerical optimization
  - The PCA activity, while cool, got punted to PS5
  - Happened because the optimization activities are plenty long on their own

Check-In

- PollEv.com/klintkanopka

Motivating Problem

Numerical Optimization

- We have a function, $f\left(\mathbf{X}, \theta\right)$, where $\mathbf{X}$ is data and θ are parameters
- Numerical optimization answers the question:

$$\underset{\theta}{\operatorname{argmin}}\, f\left(\mathbf{X}, \theta\right)$$

- From last week, remember argmin (or argmax) says: what value of θ minimizes (or maximizes) the output of $f\left(\mathbf{X}, \theta\right)$?
- Note that now $\mathbf{X}$ is a matrix argument (that also contains our outcome, y) and θ is (often) a vector argument
- If we are using argmin, $f\left(\mathbf{X}, \theta\right)$ is often referred to as a *loss function*
  - Our goal is to write problems such that when we minimize the loss function, we've found the answer!

Just thinking about argmin

- How could we use a computer to minimize this?

$$\operatorname*{argmin}_x x^2 - 2x - 3$$

- Take the derivative with respect to x, set $\frac{df}{dx} = 0$, and solve
- Make a list of possible x values, plug them in, and check which one gives the lowest value
- Plot it and eyeball it
- Some secret fourth thing???

How do we optimize?

- Remember that analytic solutions can be hard/impossible and grid search can be inefficient/slow/imprecise
- Numerical optimization leverages an algorithm to find a solution faster, more efficiently, and more precisely than grid search
- There are LOTS of algorithms with different tradeoffs
- We will develop my favorite one today
  - It might actually be my all-time favorite algorithm?

Tools

Single Precision Floating Point Numbers

- In computers, numbers are typically stored using 32 bits of memory
  - Each bit can have a value of either 0 or 1
  - This means that there are only 4,294,967,296 possible values
  - This has to cover every possible positive number, negative number, decimal, or super huge value
- The way computers handle this is using floating point numbers
  - Think of this as scientific notation
  - $2,395,423 \rightarrow 2.395423 \times 10^6$
- How is this stored?
  - One bit stores the sign (+/-)
  - Eight bits store the exponent (256 values, ranging from -126 to 127 with all 0s or 1s held back)
  - 23 bits stores the mantissa (this is about 6-8 decimal places of precision)
  - Importantly, the more extreme the value of the exponent, the less precise the entire number
  - This means nearly all computations contain some amount of rounding error

Monotonic Functions and Transformations

- A function is monotonic if it preserves order
- For a monotonic function, f:
  - $a > b \implies f(a) > f(b)$
- We will use the *log transformation* all the time in numerical optimization
- Because $\log(x)$ is a monotonic function:

$$\underset{x}{\operatorname{argmin}}\, f(x) = \underset{x}{\operatorname{argmin}} \log f(x)$$

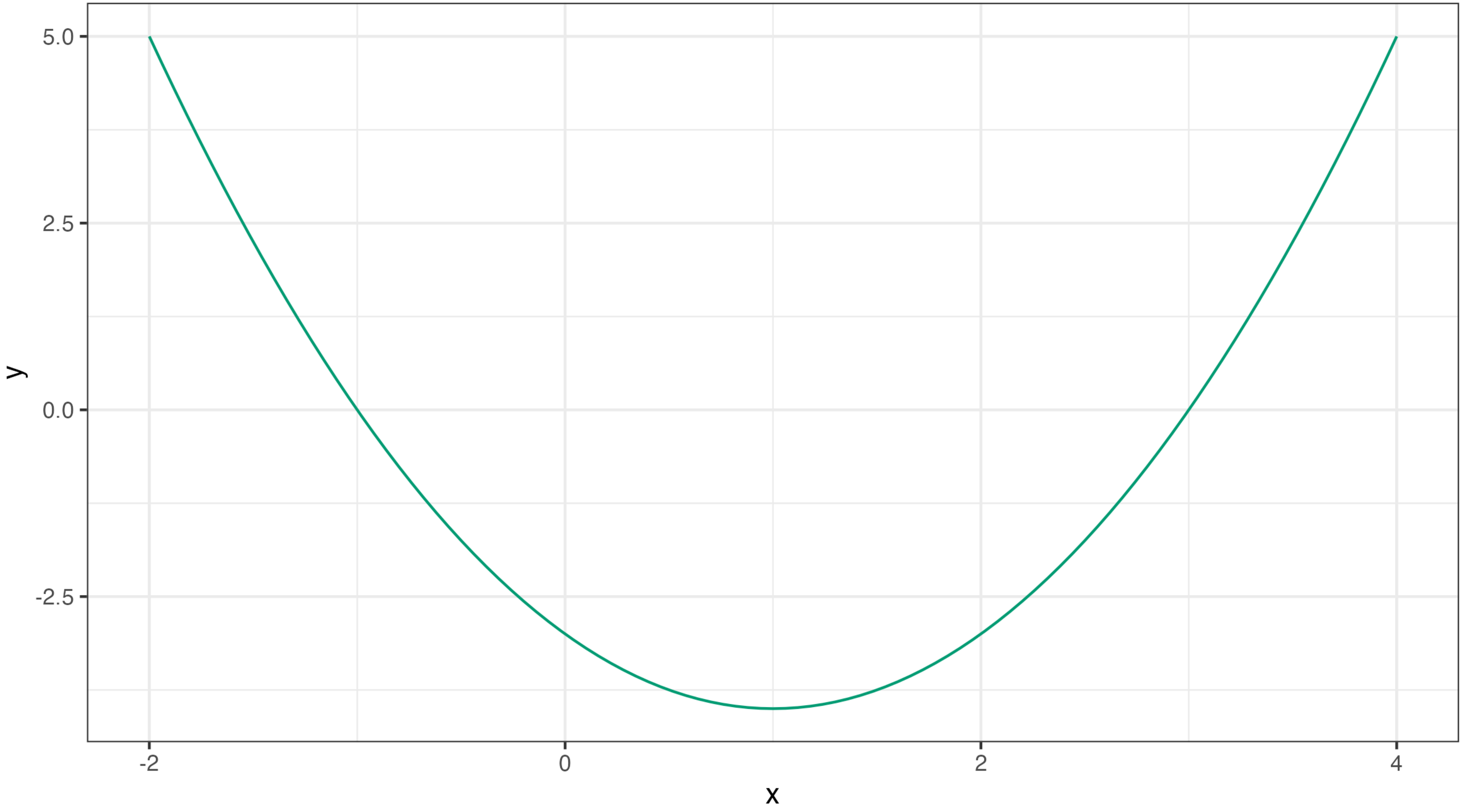
- You can turn multiplication into addition, leading to less extreme exponent values
  - $\log(ab) = \log(a) + \log(b)$
- The log transform maps positive numbers with negative exponents to negative numbers (with smaller exponents) and positive numbers with positive exponents into positive numbers (with smaller exponents)

Derivatives and Gradients

- Recall the derivative of a function with respect to a variable x is the slope of a line tangent to the function at a specific value of x
- The gradient is the multivariate (read: *vector*) generalization of the derivative
  - The gradient constructs a column vector of partial derivatives
- If you think of a multivariate function as a surface, the gradient is a vector that points "uphill"
- We write the derivative of f with respect to x as: $\frac{df}{dx}$
- If f takes a vector argument, $\mathbf{x}$, of dimension k, we write the gradient of f as: ∇f

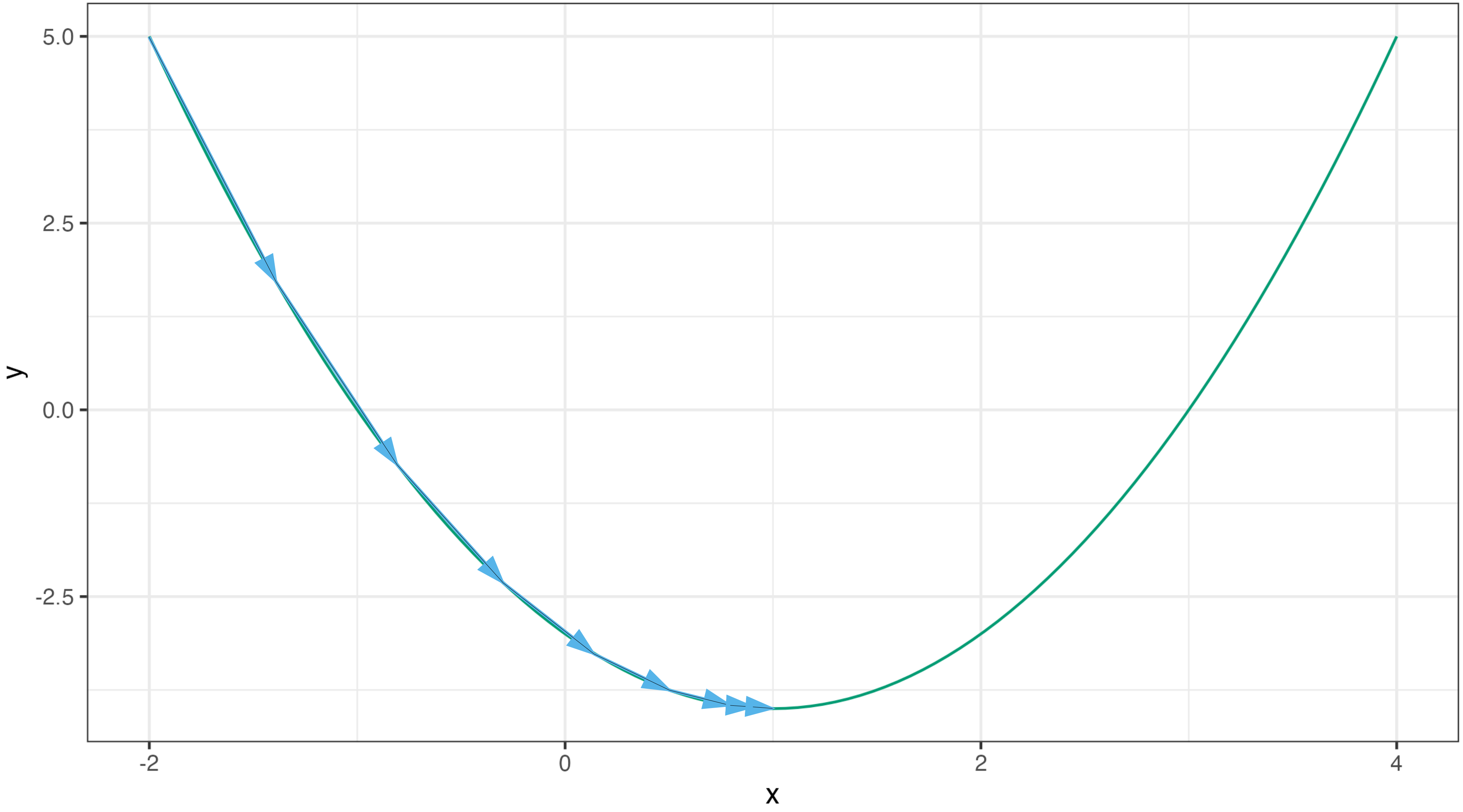
$$\nabla f = \frac{\partial f}{\partial \mathbf{x}} = \left[\frac{\partial f}{\partial x_1} \frac{\partial f}{\partial x_2} \cdots \frac{\partial f}{\partial x_k} \right]^\top$$

Algorithm: Gradient Descent


5.0
2.5
0.0
-2.5
y
-2
0
2
4
x

Gradient Descent

- Big idea: To find the minimum value of a function, we move downhill until we get to the bottom
- Algorithm: To find the argmin of a function f given parameter values θ_n at iteration n, and step size λ:
  1. Compute the gradient of f with respect to θ_n
  2. Find the next parameter values, θ_{n+1}, according to the update rule:
     - $\theta_{n+1} \leftarrow \theta_n - \lambda \nabla f$
  3. Repeat 1&2 until convergence
- What actually happens: Starting at some guess, use the gradient to repeatedly take steps "downhill" until you hit the bottom


5.0
2.5
0.0
-2.5
y
-2
0
2
4
x

The OLS Loss Function

- So how do we write a loss function?
- Frame your solution as a minimization problem
  - Note that if you want to maximize something, it's the same as minimizing the negative of that thing
- Recall that OLS attempts to minimize the sum of squared residuals
  - For OLS with a single covariate x, outcome y, and coefficients, β_0, β_1, write down the sum of squared residuals
  - Then how do we express that we want to minimize? What do we minimize with respect to?
  - What does this give us?

The OLS Loss Function

Set up the problem:

$$\underbrace{\sum_{x_i, y_i \in \mathbf{X}} \big(y_i - (\beta_0 + \beta_1 x_i)\big)^2}_{\text{The sum of squared residuals}}$$

The OLS Loss Function

Set up the problem:

$$\underbrace{\operatorname*{argmin}_{\beta}}_{\text{Find } \beta \text{ that minimizes}} \underbrace{\sum_{x_i, y_i \in \mathbf{X}} \big(y_i - (\beta_0 + \beta_1 x_i)\big)^2}_{\text{The sum of squared residuals}}$$

The OLS Loss Function

Set up the problem:

$$\underbrace{\hat{\beta}}_{\text{Estimated coefficients}} = \underbrace{\underset{\beta}{\operatorname{argmin}}}_{\text{Find } \beta \text{ that minimizes}} \underbrace{\sum_{x_i, y_i \in \mathbf{X}} \left(y_i - (\beta_0 + \beta_1 x_i)\right)^2}_{\text{The sum of squared residuals}}$$

Gradient Descent with OLS

- We start with our loss function:

$$\ell(\mathbf{X}, \beta) = \sum_{x_i, y_i \in \mathbf{X}} \big(y_i - (\beta_0 + \beta_1 x_i)\big)^2$$

- Now we need to take the gradient wrt the parameters we optimize over:

$$\nabla \ell(\mathbf{X}, \beta) = \begin{bmatrix} \frac{\partial \ell}{\partial \beta_0} \\ \frac{\partial \ell}{\partial \beta_1} \end{bmatrix}$$

- The gradient:

$$\nabla \ell(\mathbf{X}, \beta) = \begin{bmatrix} \frac{\partial \ell}{\partial \beta_0} \\ \frac{\partial \ell}{\partial \beta_1} \end{bmatrix} = \begin{bmatrix} \sum_{x_i, y_i \in \mathbf{X}} -2\big(y_i - (\beta_0 + \beta_1 x_i)\big) \\ \sum_{x_i, y_i \in \mathbf{X}} -2x_i\big(y_i - (\beta_0 + \beta_1 x_i)\big) \end{bmatrix}$$

Gradient Descent with OLS

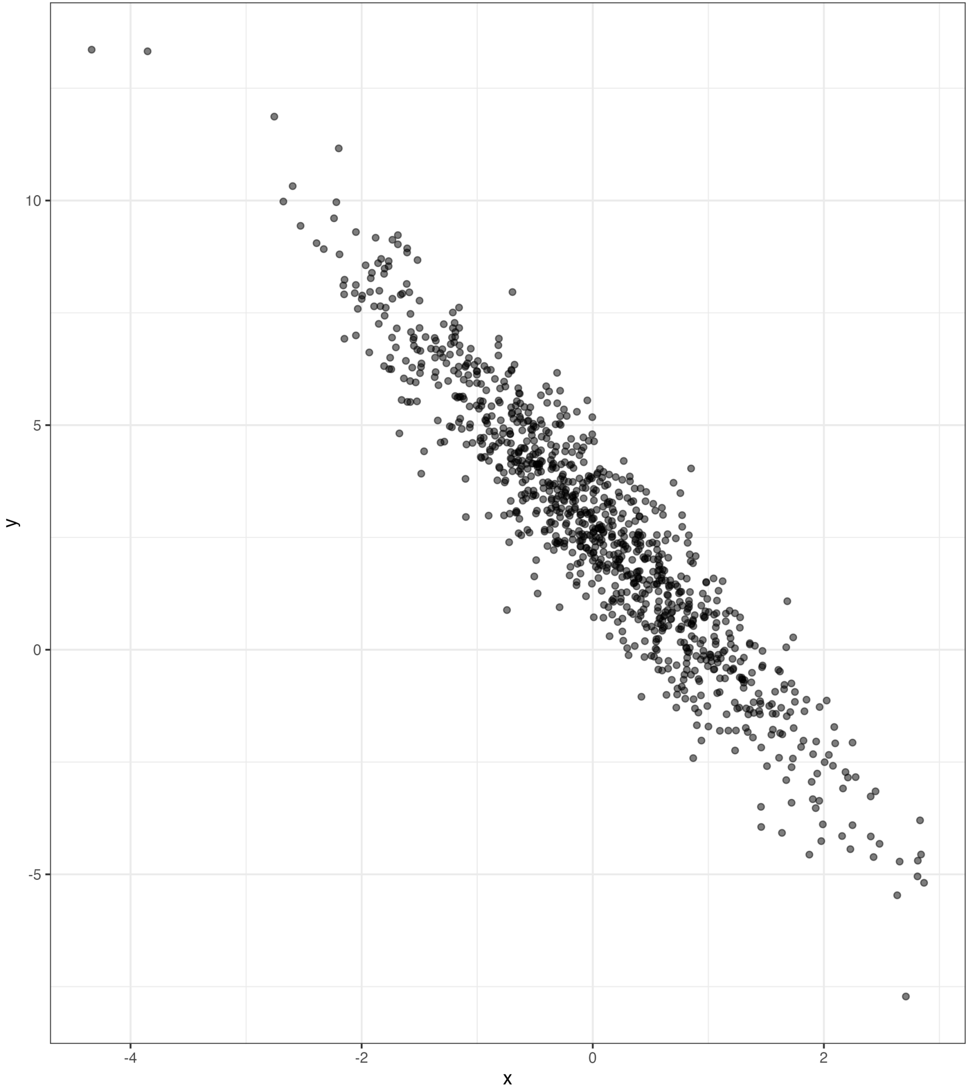
- Next we pick a step size, λ
  - Smaller values of λ result in a more precise solution, but slower convergence time
  - Larger values of λ are less precise but converge faster
  - If you make λ too large, the optimization may become unstable and never converge!
- Initialize starting values for your parameters
  - Lots of choices! Start at zero? Pick randomly?
- Update your parameters:
  - Recall $\theta_{n+1} \leftarrow \theta_n - \lambda \nabla f$
  - $\hat{\beta}_{0,n+1} \leftarrow \hat{\beta}_{0,n} + 2\lambda \sum_{x_i, y_i \in \mathbf{X}} \left(y_i - (\hat{\beta}_{0,n} + \hat{\beta}_{1,n} x_i)\right)$
  - $\hat{\beta}_{1,n+1} \leftarrow \hat{\beta}_{1,n} + 2\lambda \sum_{x_i, y_i \in \mathbf{X}} x_i \left(y_i - (\hat{\beta}_{0,n} + \hat{\beta}_{1,n} x_i)\right)$

Simulating some OLS data

```r
set.seed(242424)

N <- 1e3
true_beta <- c(2.718, -2.718)

d <- data.frame(x = rnorm(N))
d$y <- true_beta[1] + true_beta[2] * d$x + rnorm(N)

ols <- lm(y ~ x, d)
ols_beta <- coef(ols)
ols_beta

# (Intercept)           x
#    2.783908   -2.686621
```

Simulating some OLS data

```r
ggplot(d, aes(x = x, y = y)) +
  geom_point(alpha = 0.5) +
  theme_bw()
```



Implementing the Gradient Descent Update

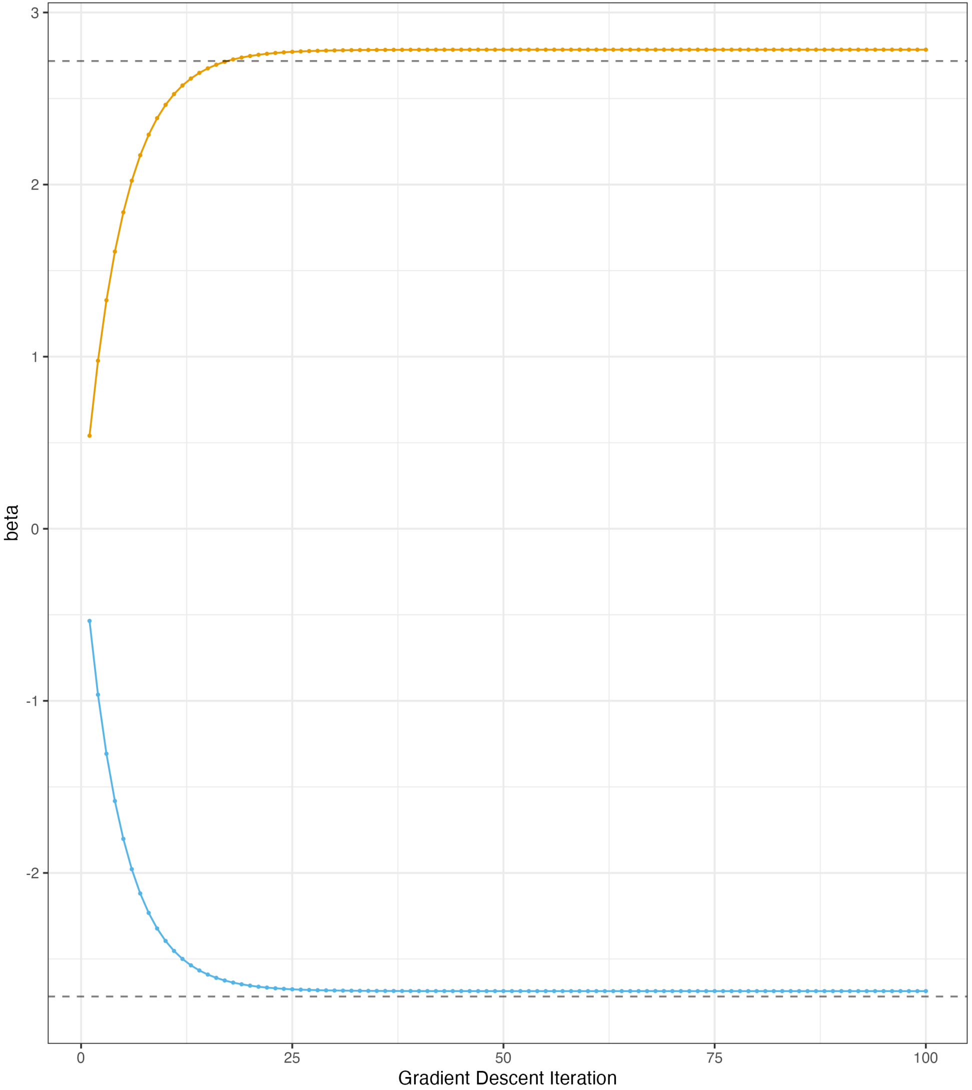
```r
lr <- 1e-4
beta <- c(0, 0)

beta[1] <- beta[1] + 2 * lr * sum(d$y - (beta[1] + beta[2] * d$x))
beta[2] <- beta[2] + 2 * lr * sum(d$x * (d$y - (beta[1] + beta[2] * d$x)))

beta

# [1]  0.5407162 -0.5354659

beta[1] <- beta[1] + 2 * lr * sum(d$y - (beta[1] + beta[2] * d$x))
beta[2] <- beta[2] + 2 * lr * sum(d$x * (d$y - (beta[1] + beta[2] * d$x)))

beta

# [1]  0.9764911 -0.9641414
```

How long until convergence? $\lambda = 10^{-4}$

```r
M <- 1e2
lr <- 1e-4
beta <- c(0, 0)
betas <- data.frame(i = 1:M, beta_0 = numeric(M), beta_1 = numeric(M))

for (i in 1:M) {
  beta[1] <- beta[1] + 2 * lr * sum(d$y - (beta[1] + beta[2] * d$x))
  beta[2] <- beta[2] + 2 * lr * sum(d$x * (d$y - (beta[1] + beta[2] * d$x)))

  betas$beta_0[i] <- beta[1]
  betas$beta_1[i] <- beta[2]
}

beta

# [1]  2.783908 -2.686621
```

$\lambda = 10^{-4}$

```r
ggplot(betas, aes(x = i)) +
  geom_line(aes(y = beta_0),
            color = okabeito_colors(1)) +
  geom_line(aes(y = beta_1),
            color = okabeito_colors(2)) +
  geom_point(aes(y = beta_0),
             size = 0.5,
             color = okabeito_colors(1)) +
  geom_point(aes(y = beta_1),
             size = 0.5,
             color = okabeito_colors(2)) +
  geom_hline(aes(yintercept = true_beta[1]),
             lty = 2,
             alpha = 0.5) +
  geom_hline(aes(yintercept = true_beta[2]),
                 lty = 2,
                 alpha = 0.5) +
  labs(x='Gradient Descent Iteration',
       y = 'beta') +
  theme_bw()
```

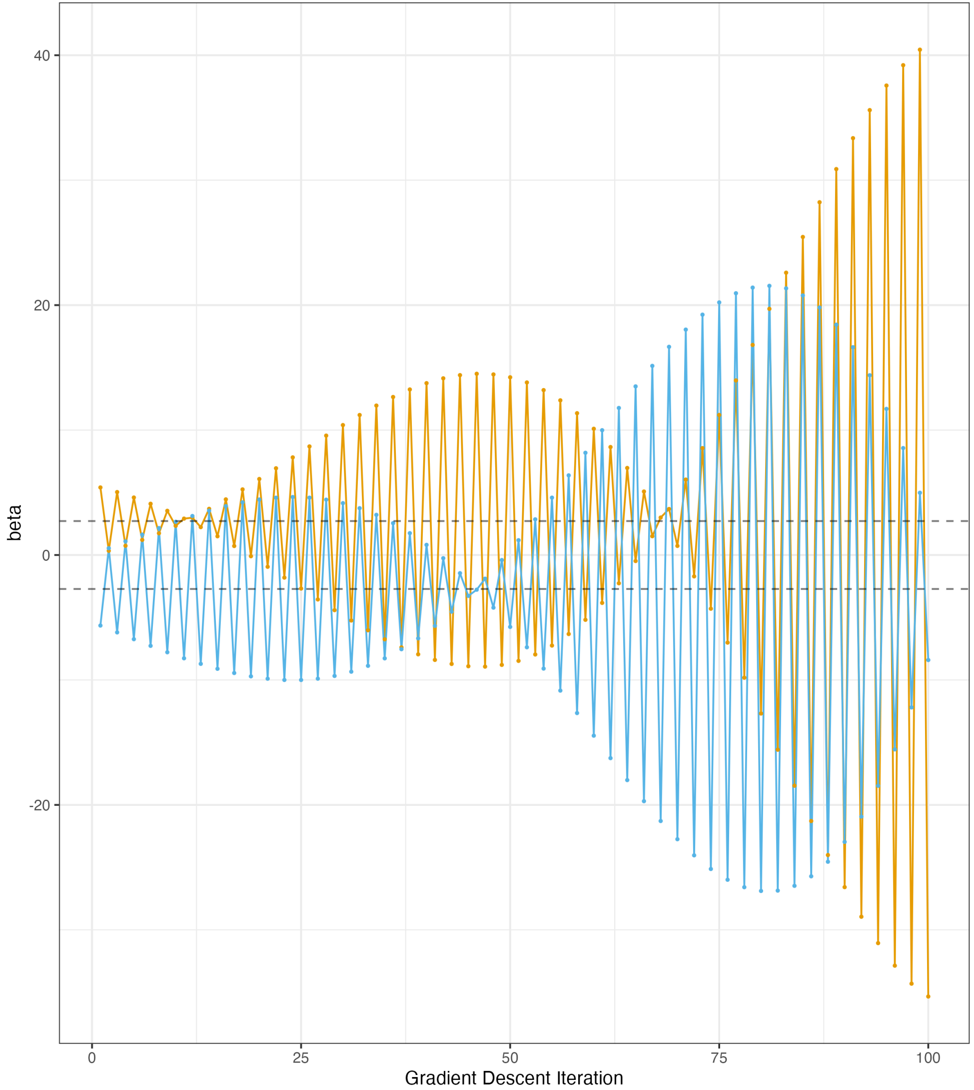


What about a faster learning rate? $\lambda = 10^{-3}$

```r
M <- 1e2
lr <- 1e-3
beta <- c(0, 0)
betas <- data.frame(i = 1:M, beta_0 = numeric(M), beta_1 = numeric(M))

for (i in 1:M) {
  beta[1] <- beta[1] + 2 * lr * sum(d$y - (beta[1] + beta[2] * d$x))
  beta[2] <- beta[2] + 2 * lr * sum(d$x * (d$y - (beta[1] + beta[2] * d$x)))

  betas$beta_0[i] <- beta[1]
  betas$beta_1[i] <- beta[2]
}

beta

# [1] -35.335523  -8.404527
```

$$\lambda = 10^{-3}$$

```r
ggplot(betas, aes(x = i)) +
  geom_line(aes(y = beta_0),
            color = okabeito_colors(1)) +
  geom_line(aes(y = beta_1),
            color = okabeito_colors(2)) +
  geom_point(aes(y = beta_0),
             size = 0.5,
             color = okabeito_colors(1)) +
  geom_point(aes(y = beta_1),
             size = 0.5,
             color = okabeito_colors(2)) +
  geom_hline(aes(yintercept = true_beta[1]),
             lty = 2,
             alpha = 0.5) +
  geom_hline(aes(yintercept = true_beta[2]),
                 lty = 2,
                 alpha = 0.5) +
  labs(x='Gradient Descent Iteration',
       y = 'beta') +
  theme_bw()
```

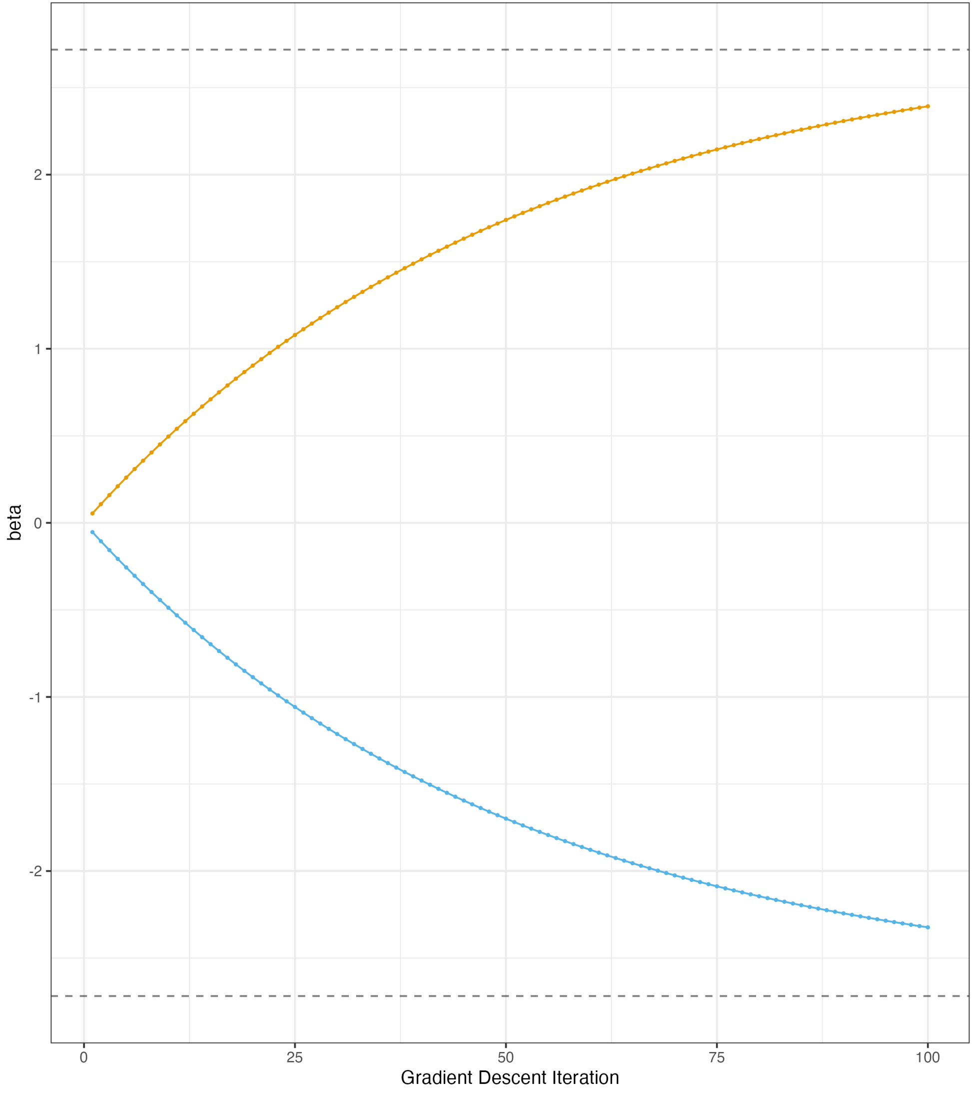


What about a slower learning rate? $\lambda = 10^{-5}$

```r
M <- 1e2
lr <- 1e-5
beta <- c(0, 0)
betas <- data.frame(i = 1:M, beta_0 = numeric(M), beta_1 = numeric(M))

for (i in 1:M) {
  beta[1] <- beta[1] + 2 * lr * sum(d$y - (beta[1] + beta[2] * d$x))
  beta[2] <- beta[2] + 2 * lr * sum(d$x * (d$y - (beta[1] + beta[2] * d$x)))

  betas$beta_0[i] <- beta[1]
  betas$beta_1[i] <- beta[2]
}

beta

# [1]  2.392762 -2.323370
```

$\lambda = 10^{-5}$

```
ggplot(betas, aes(x = i)) +
  geom_line(aes(y = beta_0),
            color = okabeito_colors(1)) +
  geom_line(aes(y = beta_1),
            color = okabeito_colors(2)) +
  geom_point(aes(y = beta_0),
             size = 0.5,
             color = okabeito_colors(1)) +
  geom_point(aes(y = beta_1),
             size = 0.5,
             color = okabeito_colors(2)) +
  geom_hline(aes(yintercept = true_beta[1]),
             lty = 2,
             alpha = 0.5) +
  geom_hline(aes(yintercept = true_beta[2]),
                 lty = 2,
                 alpha = 0.5) +
  labs(x='Gradient Descent Iteration',
       y = 'beta') +
  theme_bw()
```



When do you stop?

- The *easiest* way to do it is just pick a number of iterations and then stop
  - You may not reach the solution
  - You may take too many iterations that you didn't need to
- Typically we stop when the solution *converges*
  - Often this looks like picking a threshold value, ϵ
  - After every iteration, check to see how much the estimate has changed by
  - Stop when the change in parameter estimates is smaller than the threshold
  - Lets you set the level of precision you want in your answer
  - If your threshold is too small relative to the step size, the optimization routine may never converge

More Tools

Maximum Likelihood Estimation

Maximum Likelihood Estimation

- A linear regression assumes normally distributed error with constant error variance, or:

$$y_i \sim \mathcal{N}\left(\beta_0 + \beta_1 x_i, \sigma^2\right)$$

- We can write down the *likelihood* (think Bayes' theorem) of observing a set of parameters conditioned on our observed data by multiplying together the normal density functions for each observation:

$$L(\beta|\mathbf{X}) = \prod_{y_i, x_i \in \mathbf{X}} \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{1}{2}\left(\frac{y_i - (\beta_0 + \beta_1 x_i)}{\sigma}\right)^2}$$

Log Transformin'

- We can make our lives way easier with a log transform! Why?
- Multiplying lots of probabilities results in numerical instability by ending up with tiny numbers!
- The log transform can turn these products into a sum!

$$\log \prod_{y_i, x_i \in \mathbf{X}} \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{1}{2}\left(\frac{y_i-(\beta_0+\beta_1 x_i)}{\sigma}\right)^2} = \sum_{y_i, x_i \in \mathbf{X}} \log \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{1}{2}\left(\frac{y_i-(\beta_0+\beta_1 x_i)}{\sigma}\right)^2}$$

$$= \sum_{y_i, x_i \in \mathbf{X}} \log \frac{1}{\sigma\sqrt{2\pi}} + \log e^{-\frac{1}{2}\left(\frac{y_i-(\beta_0+\beta_1 x_i)}{\sigma}\right)^2}$$

$$= \sum_{y_i, x_i \in \mathbf{X}} \log \frac{1}{\sigma\sqrt{2\pi}} - \frac{1}{2}\left(\frac{y_i-(\beta_0+\beta_1 x_i)}{\sigma}\right)^2$$

Maximum (Log) Likelihood Estimation

- Maximum Likelihood Estimation (MLE) is just finding the values of your parameters that maximize the likelihood
- Because the log transform is monotonic, we can just maximize the log likelihood instead

$$\ell(\theta|\mathbf{X}) = \log L(\theta|\mathbf{X})$$

$$\hat{\theta} = \underset{\theta}{\operatorname{argmax}}\, L(\theta|\mathbf{X}) = \underset{\theta}{\operatorname{argmax}}\, \ell(\theta|\mathbf{X})$$

- We have a few options on how this works:

1. Take the gradient of the log likelihood and set it equal to zero to find the estimates $\hat{\theta}$ analytically
2. Use numerical optimization
3. Some secret third thing that we haven't really discussed yet

MLE for OLS

$$\hat{\beta} = \underset{\beta}{\operatorname{argmax}} \left[\sum_{y_i, x_i \in \mathbf{X}} \log \frac{1}{\sigma\sqrt{2\pi}} - \frac{1}{2}\left(\frac{y_i - (\beta_0 + \beta_1 x_i)}{\sigma} \right)^2 \right]$$

$$\hat{\beta} = \underset{\beta}{\operatorname{argmax}} \left[N \log \frac{1}{\sigma\sqrt{2\pi}} - \frac{1}{2\sigma^2} \sum_{y_i, x_i \in \mathbf{X}} \big(y_i - (\beta_0 + \beta_1 x_i)\big)^2 \right]$$

$$\hat{\beta} = \underset{\beta}{\operatorname{argmax}} \left[\underbrace{N \log \frac{1}{\sigma\sqrt{2\pi}}}_{\text{Constant}} - \underbrace{\frac{1}{2\sigma^2}}_{\text{Constant}} \sum_{y_i, x_i \in \mathbf{X}} \big(y_i - (\beta_0 + \beta_1 x_i)\big)^2 \right]$$

$$\hat{\beta} = \underset{\beta}{\operatorname{argmax}} \left[- \sum_{y_i, x_i \in \mathbf{X}} \big(y_i - (\beta_0 + \beta_1 x_i)\big)^2 \right]$$

- Does this last line look familiar?

MLE for OLS

$$\hat{\beta} = \underset{\beta}{\operatorname{argmin}} \sum_{y_i, x_i \in \mathbf{X}} \big(y_i - (\beta_0 + \beta_1 x_i)\big)^2$$

- The MLE for OLS is identical to minimizing the sum of squared residuals
- This is not always the case for other estimators, though!
- Now we have to actually minimize it—for this, we'll use `optim()`

`optim()`

All hail the greatest built-in function in base `R`, `optim()`

- `optim()` is a general purpose optimization function in `R`
- It has a bunch of different optimization methods and is used by just about every single modeling function under the hood
- `optim()` specifically does **argmin**
  - You write a function that takes parameters as inputs
  - You give `optim()` starting values for those parameters (as a vector) and the function to be argmin'd
  - `optim()` returns the values of the parameters that minimize the function
- **If you can write something as a maximization or minimization problem,** `optim()` **can solve it for you**
  - Caveats about convexity and identifiability and some other stuff

MLE with optim() for OLS

```r
SumSquaredResid <- function(beta, d){
  # TODO: Compute sum of squared residuals
  resid <- d$y - (beta[1] + beta[2]*d$x)
  ssr <- sum(resid^2)

  return(ssr)
}

SumSquaredResid(c(0,0), d)

# [1] 15666.95

out <- optim(c(0,0),          # starting vals for parameters
             SumSquaredResid, # fn to minimize
             d=d)             # args to pass to fn

# pass control=list(scale=-1) to make optim argmax

mle_beta <- out$par

mle_beta

# [1]  2.784030 -2.686716
```

Checking results

```
true_beta

# [1]  2.718 -2.718

ols_beta

# (Intercept)           x
#    2.783908   -2.686621

gd_beta

# [1]  2.783908 -2.686621

mle_beta

# [1]  2.784030 -2.686716
```

Some notes on `optim()`

- The function you minimize needs to output only a single scalar value
- The first argument of the function you minimize needs to be all the parameters you want to optimize, passed as a vector
- There are a whole bunch of options you can fiddle with in order to change convergence conditions, speed, or optimization algorithm
- If you make a function that also returns the gradient, you can gain a lot of speed and precision!
- Works best with *convex* problems
  - Otherwise you can get stuck in local minima!
  - If you can't be convex, just start from a bunch of random spots and take the best solution
  - If your problem involves finding where a function is equal to zero, squaring it and finding the minimum will do the job!

Wrap Up

Recap

- Numerical optimization is the most useful skill we learn in this course
- If you can frame something as an optimization problem, `optim()` can generally solve it
- Working in log space has tons of advantages, especially when you're dealing with probabilities

Final Thoughts

- PollEv.com/klintkanopka