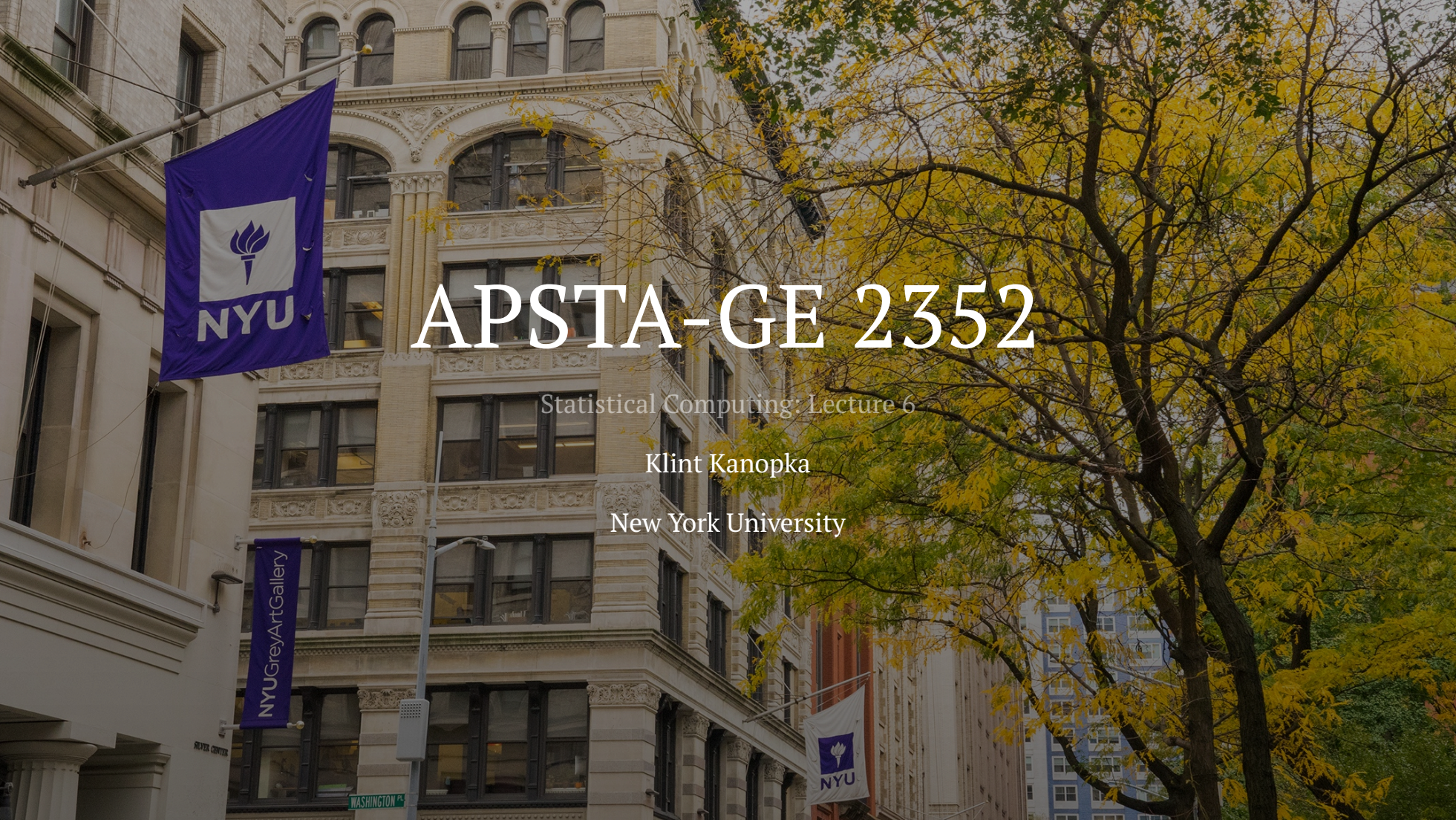


APSTA-GE 2352

Statistical Computing: Lecture 6

Klint Kanopka

New York University

Table of Contents

Announcements

- PS3 is due next week
- PS2 grades and solutions coming soon
- Lecture this week gets pretty math-dense. Please ask questions as we go!

Check-In

- PollEv.com/klintkanopka

Rejection Sampling

Rejection Sampling

- Check the .qmd and .pdf I pinned in our Slack channel
- **Core Idea:** Monte Carlo simulations are a really powerful problem solving tool

Rejection Sampling

- Check the .qmd and .pdf I pinned in our Slack channel
- **Core Idea:** Monte Carlo simulations are a really powerful problem solving tool, *assuming you can generate random samples from an appropriate distribution*

Rejection Sampling

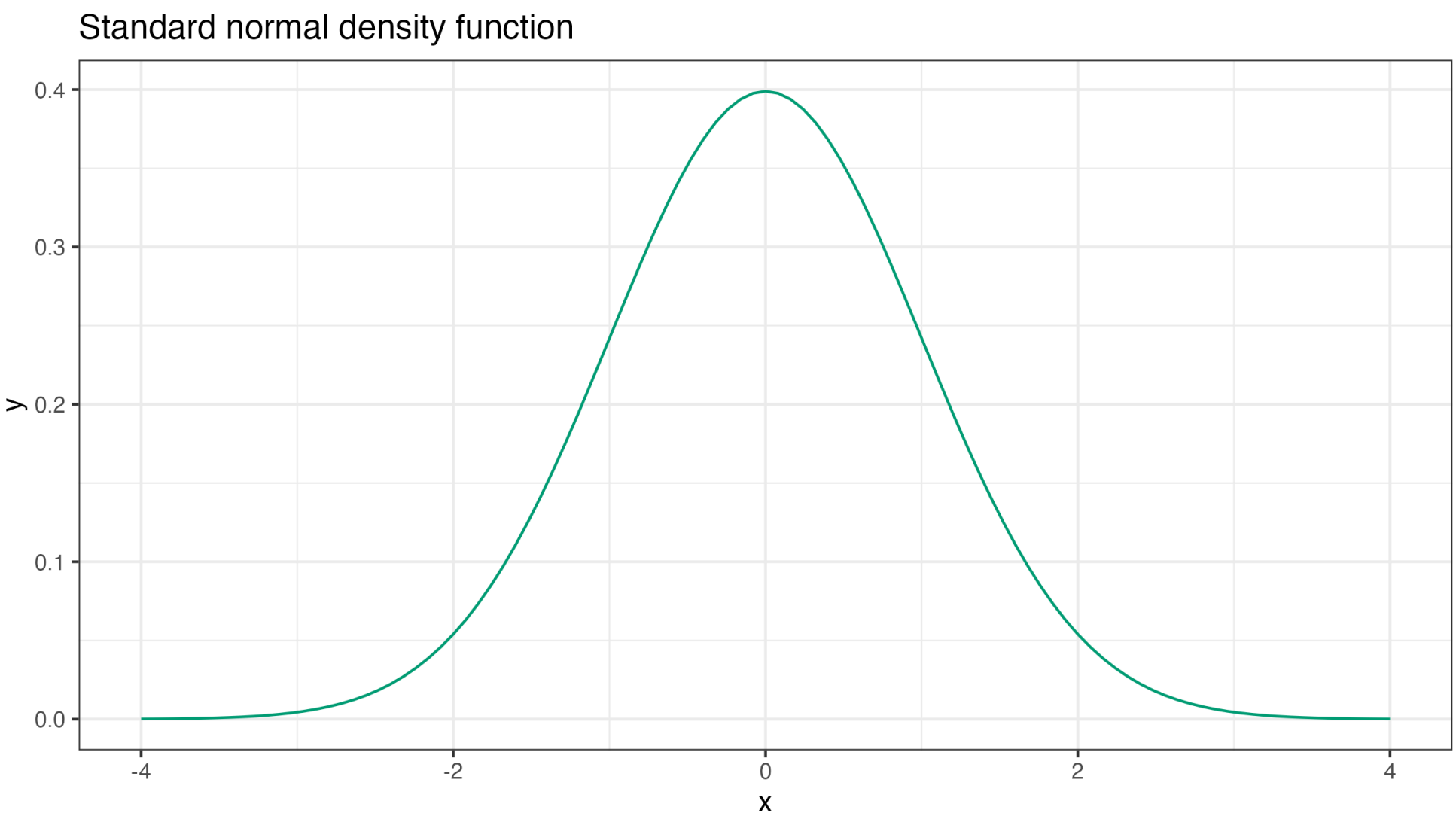
- Check the .qmd and .pdf I pinned in our Slack channel
- **Core Idea:** Monte Carlo simulations are a really powerful problem solving tool, *assuming you can generate random samples from an appropriate distribution*
- What if you need to draw samples from a complicated distribution? One that doesn't have a handy built-in function in `R`?

Rejection Sampling

- Check the .qmd and .pdf I pinned in our Slack channel
- **Core Idea:** Monte Carlo simulations are a really powerful problem solving tool, *assuming you can generate random samples from an appropriate distribution*
- What if you need to draw samples from a complicated distribution? One that doesn't have a handy built-in function in `R`?
- **Rejection sampling** can be a useful tool for sampling from *any* density function

Probability Density Functions

- First, what is a **density function**?
- A *probability density function* (PDF) describes the relative likelihood of observing different values of a continuous random variable (RV)
- We call the values a RV can take on the *support*
- While the probability of each unique value is zero, the integral of the PDF between two points can tell you the probability of observing a value between those points


Standard normal density function
0.4
0.3
0.2
0.1
0.0
y
-4
-2
0
2
4
x

Rejection Sampling

Generic approach to draw N samples from a random variable X with PDF f_X:

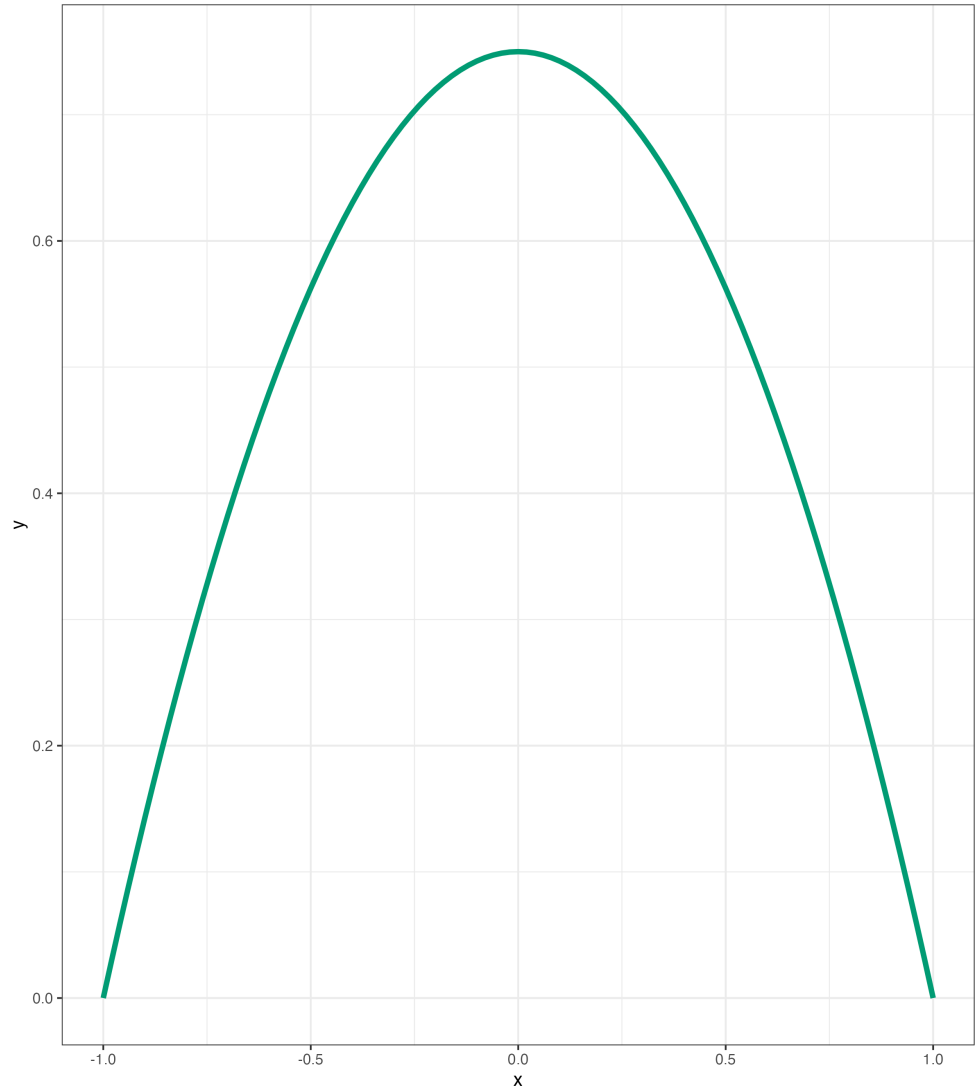
1. **Propose a sample**
    1. Draw a proposed sample, x, uniformly along the support of X
    2. Draw a value, $p_x \sim \text{Uniform}(0, 1)$
2. **Evaluate the proposal**
    1. If $p_x < f_X(x)$, *accept* the sample x
    2. Otherwise, *reject* the sample x
3. **Repeat**
    1. Continue repeating 1&2 until you have accepted N samples
    2. These samples are guaranteed to be distributed according to your PDF

Parabolic Density Function

```r
parabolic <- function(x){
  k <- 0.75 * (1 - x^2)
  return(k)
}

d <- data.frame(
    x = seq(
        from = -1,
        to = 1,
        length.out = 1e3
    )
)

ggplot(d, aes(x = x)) +
  geom_function(
    fun = parabolic,
    color = okabeito_colors(3),
    linewidth = 1.5
  ) +
  theme_bw()
```



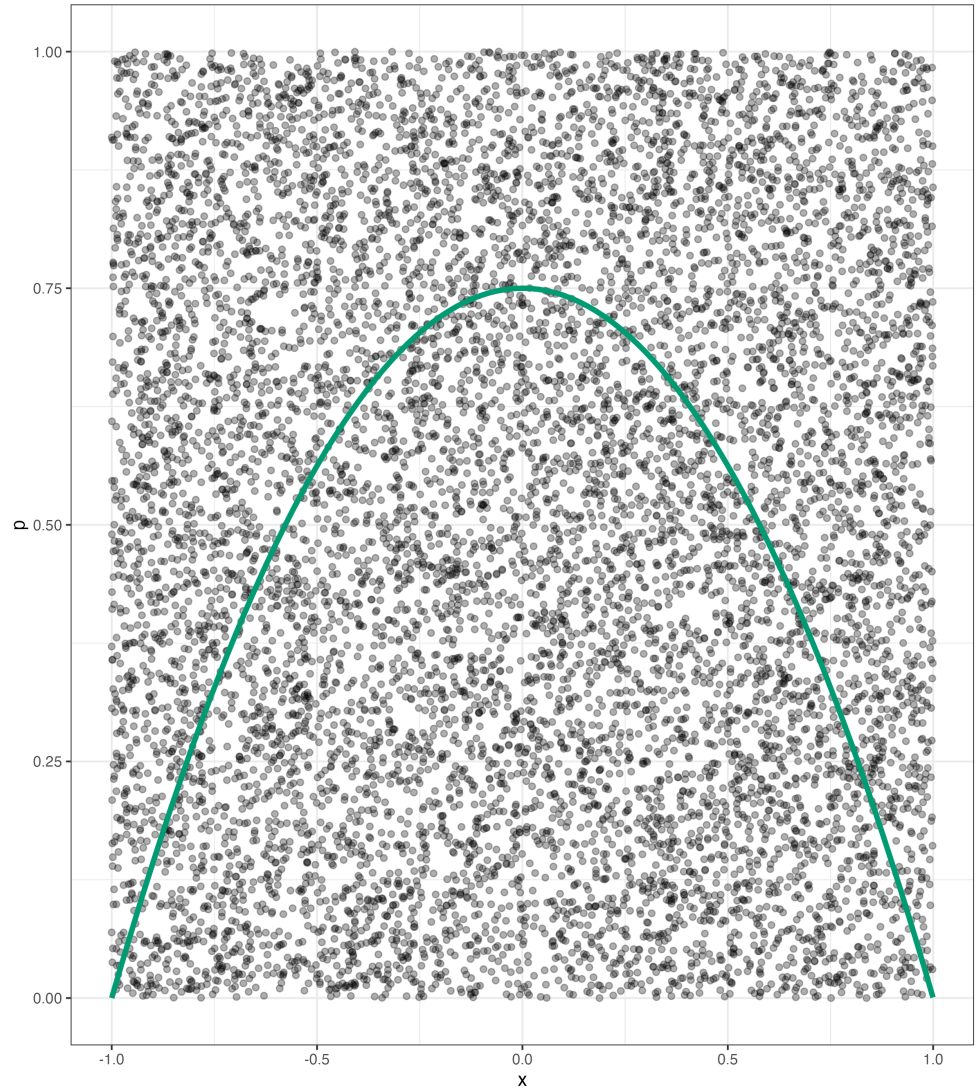
Parabolic Density Function

Step 1: Propose samples

```
N <- 1e4

d <- data.frame(
    x = runif(N, -1, 1),
    p = runif(N, 0, 1)
)

ggplot(d, aes(x = x, y = p)) +
  geom_point(alpha = 0.3) +
  geom_function(
    fun = parabolic,
    color = okabeito_colors(3),
    linewidth = 1.5
  ) +
  theme_bw()
```

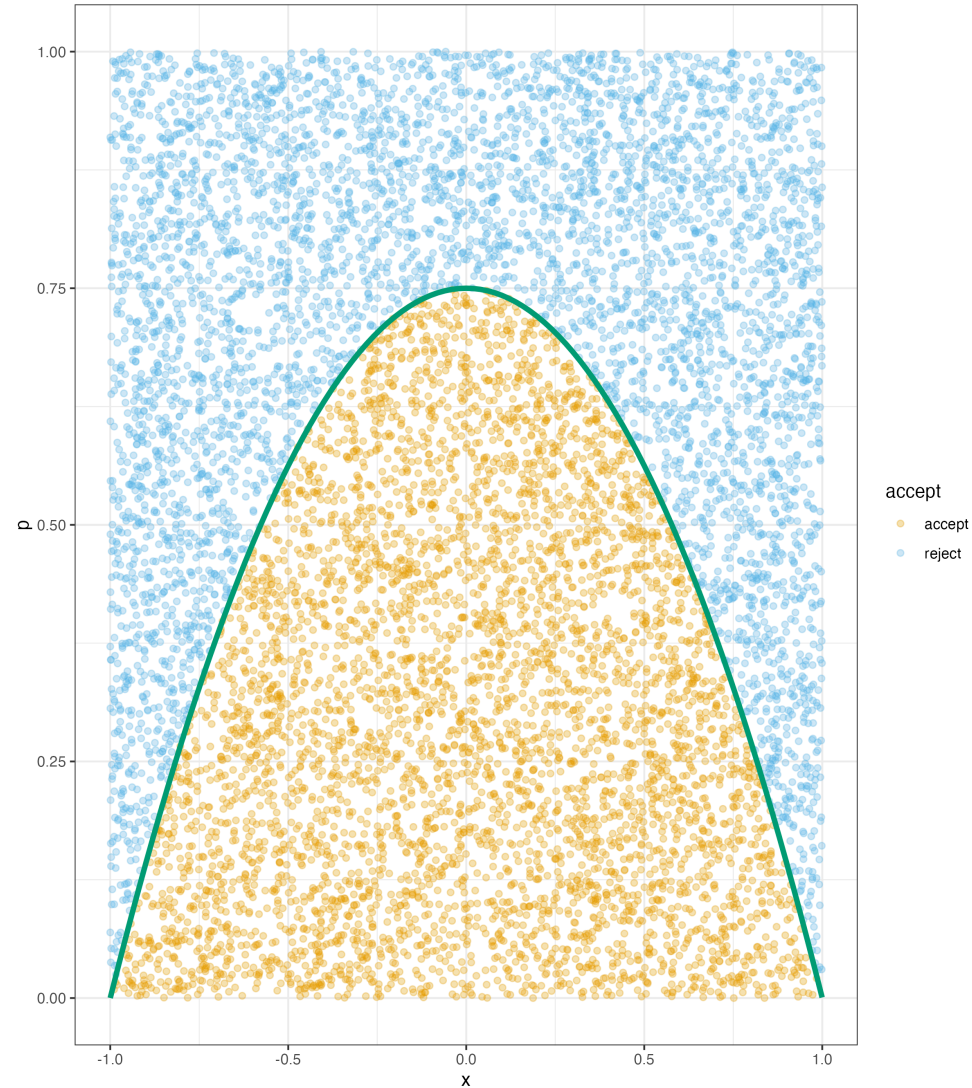


Parabolic Density Function

Step 2: Evaluate Proposals

```
d <- d |>
  mutate(accept = if_else(
    p < parabolic(x),
    'accept',
    'reject'
  )
)

ggplot(d, aes(x = x, y = p)) +
  geom_point(alpha = 0.3) +
  geom_function(
    fun = parabolic,
    color = okabeito_colors(3),
    linewidth = 1.5
  ) +
  theme_bw()
```



Parabolic Density Function

Did it work?

```
d |>
  filter(accept == 'accept') |>
  ggplot(aes(x = x)) +
  geom_density(
    color = okabeito_colors(1),
    linewidth = 1.5
  ) +
  geom_function(
    fun = parabolic,
    color = okabeito_colors(3),
    linewidth = 1.5
  ) +
  theme_bw()

sum(d$accept == 'accept')
```

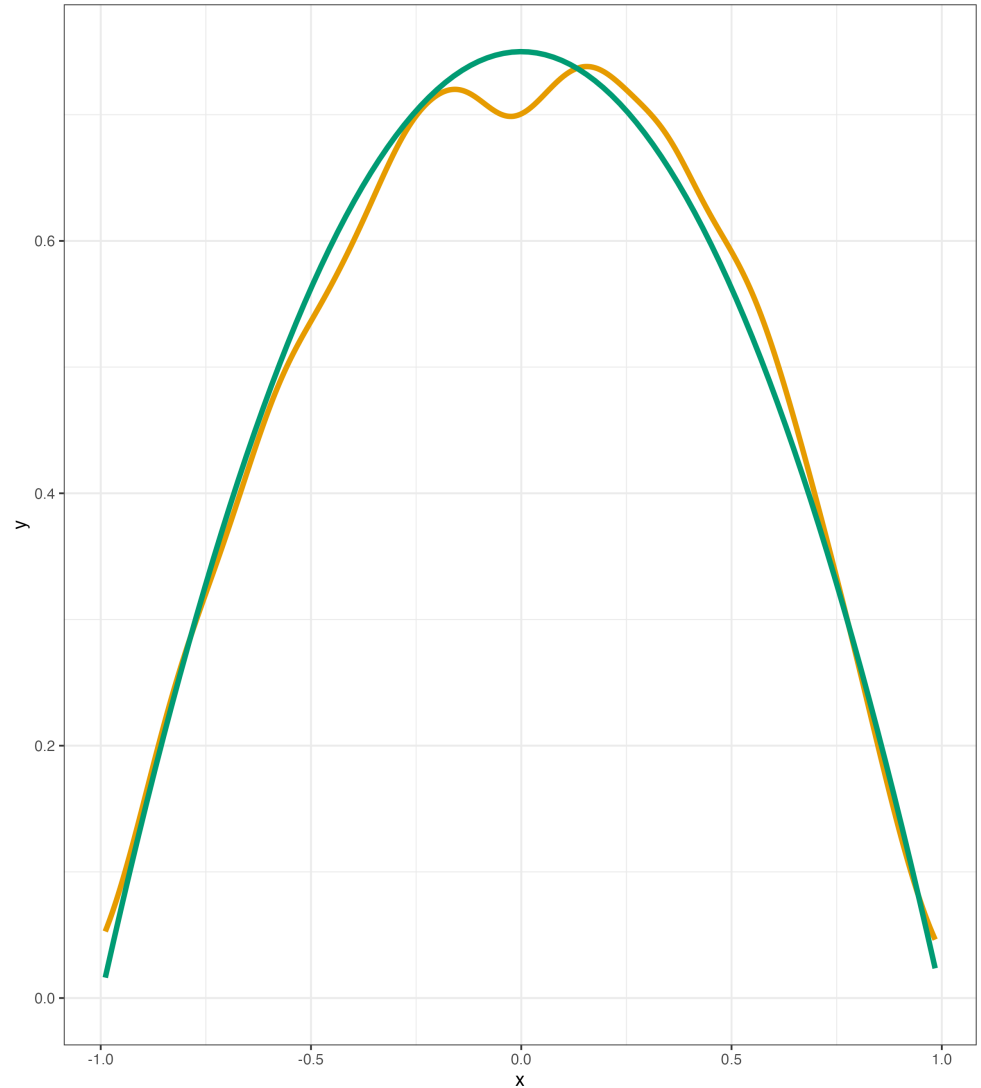
Parabolic Density Function

Did it work?

```
d |>
  filter(accept == 'accept') |>
  ggplot(aes(x = x)) +
  geom_density(
    color = okabeito_colors(1),
    linewidth = 1.5
  ) +
  geom_function(
    fun = parabolic,
    color = okabeito_colors(3),
    linewidth = 1.5
  ) +
  theme_bw()

sum(d$accept == 'accept')

# [1] 4979
```



Motivating Problem

Motivating Problem: High Dimensional Data

Someone gives you a dataset with over 500 variables in it. They ask you to build a predictive model and some visuals to communicate what's going on with the data and the model you fit to some stakeholders of unknown statistical experience. What do you do?

Tools

Mathematical Objects in `R`

Scalars and Vectors

- **Scalars** are single real numbers
  - Can be added and multiplied with each other
  - These are the numbers you're used to working with basically all the time
  - Examples: 2, π, -0.7
- **Vectors** represent quantities that can't just be summarized in a single number
  - These are quantities that have a *magnitude* ("length") and *direction* (direction)
  - Either represented as a scalar with a direction *or* a single row/column matrix with each element representing a distance projected along each dimension
  - Can be added or multiplied with each other *if they have the same number of elements*
  - Can also be multiplied by scalars
  - Examples: $9.8 \frac{\mathrm{m}}{\mathrm{s}^2}$ down, $\begin{bmatrix} 3 \\ 4 \end{bmatrix}$, $\begin{bmatrix} 1 & 1 & 1 \end{bmatrix}$

Matrices and Tensors

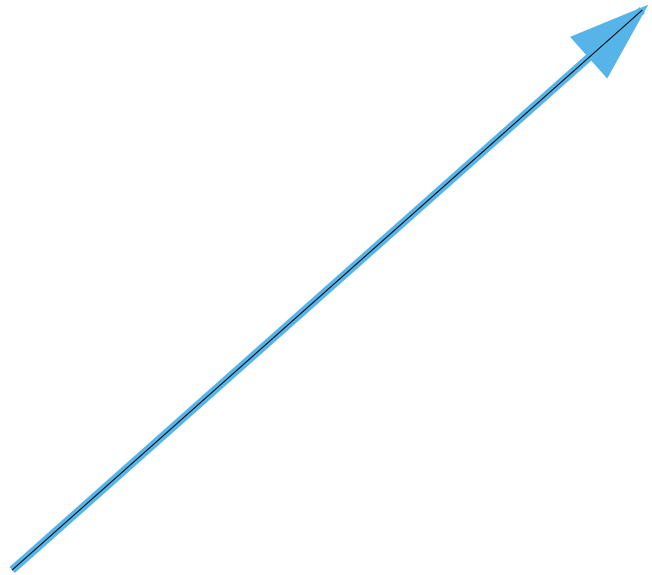
- **Matrices**
  - A rectangular generalization of the vector that is an array of numbers arranged in rows and columns
  - Can be added to each other (if they have the same shape)
  - Can be multiplied with each other (under some conditions)
  - Can be multiplied with vectors (under some conditions)
  - Can be multiplied with scalars (always)
  - Examples: $\begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{bmatrix}, \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$
- **Tensors**
  - These are generalizations of all these quantities!
  - Scalars are rank 0 tensors (A), vectors are rank 1 tensors (A_i), matrices are rank 2 tensors (A_{ij})
  - The rank is the number of indices required to refer to the elements
  - A rank 3 tensor is a *cube* of numbers (A_{ijk})
  - We won't use these today, but they show up a *lot* in applied deep learning

Rethinking Addition: Graphically

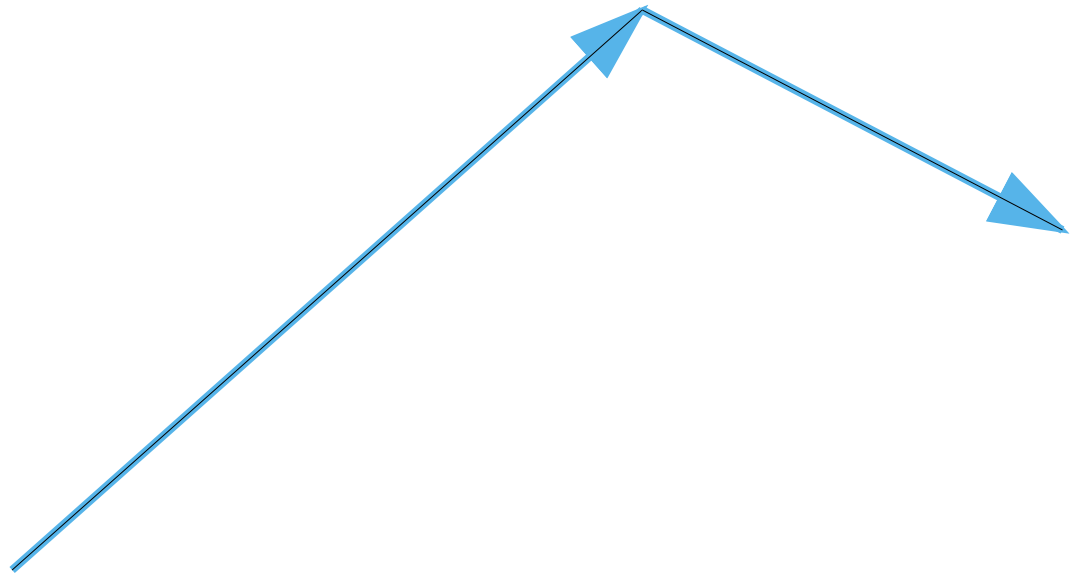
- What do addition and multiplication *do*?
- This isn't really easy to think about in terms of scalars alone, so let's include vectors
- For two vectors, $\vec{u}$, $\vec{v}$, what does $\vec{u} + \vec{v}$ do?
- Addition is *translation*, it moves things around
- Addition will move one vector to the end of the other and then draw a new vector from the tail of the first to the tip of the second
- Mechanically, if $\vec{w} = \vec{u} + \vec{v}$:
  - $w_i = u_i + v_i$
  - $\begin{bmatrix} 1 \\ 2 \end{bmatrix} + \begin{bmatrix} 3 \\ 4 \end{bmatrix} = \begin{bmatrix} 4 \\ 6 \end{bmatrix}$
  - This is how `R` adds vectors—but note that in a linear algebra sense, **recycling is not allowed** and you can only add vectors with the same number of elements

Rethinking Addition: Graphically

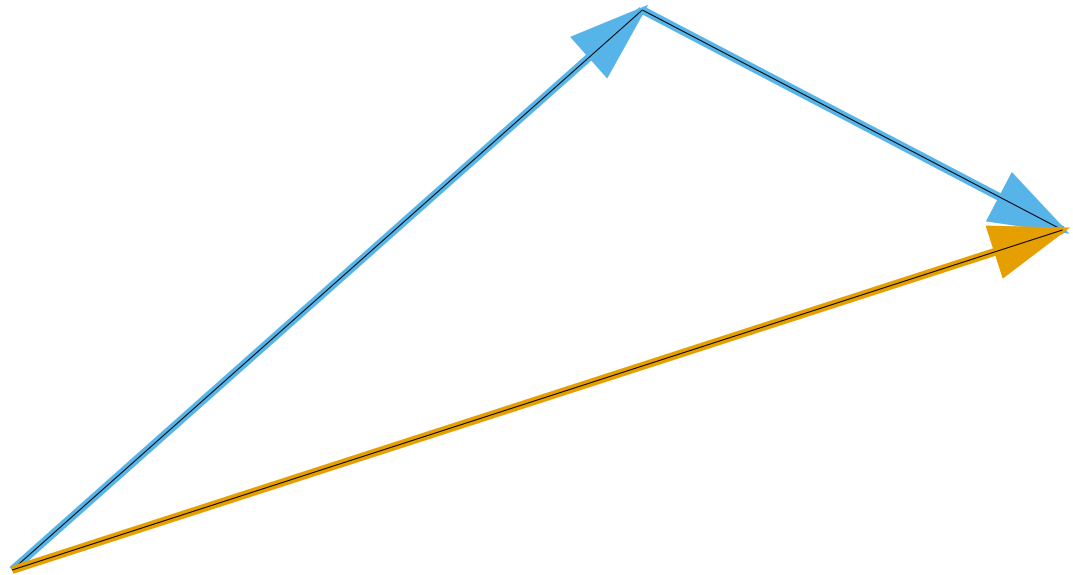
Rethinking Addition: Graphically



Rethinking Addition: Graphically



Rethinking Addition: Graphically

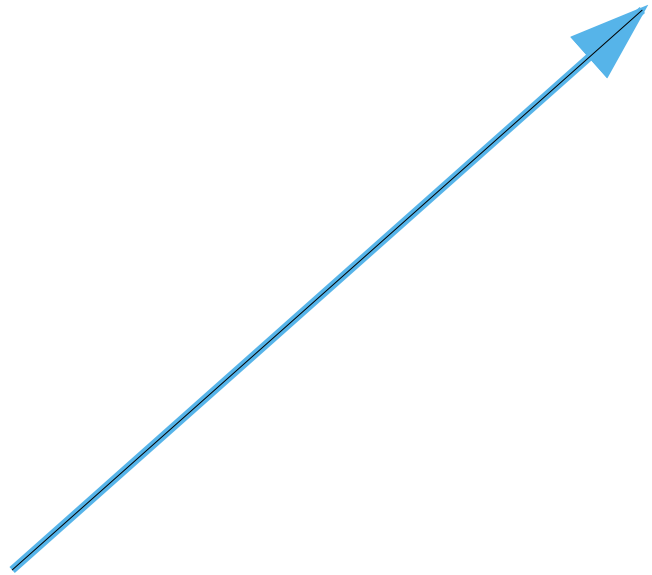


Rethinking Multiplication

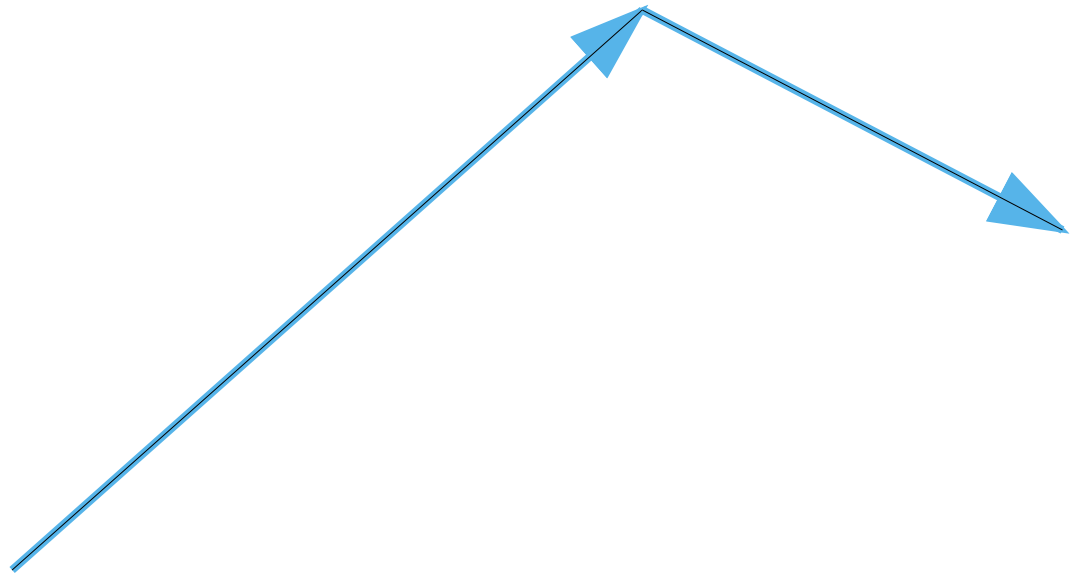
- Scalar multiplication is always a rescaling and never changes the direction of a vector
  - $2\vec{u}$ is a vector in the same direction as $\vec{u}$ with twice the length
  - $\frac{1}{2}\begin{bmatrix}1\\2\end{bmatrix} = \begin{bmatrix}0.5\\1\end{bmatrix}$
- Vector multiplication can go two different ways:
  - The *dot* product (or scalar product) $\vec{u} \cdot \vec{v}$ is a projection of $\vec{u}$ onto $\vec{v}$
    - The result is the length of $\vec{u}$ in the direction of $\vec{v}$
    - Computed as $\vec{u} \cdot \vec{v} = \sum_i u_i v_i$
    - Computed in `R` using `%*%`
  - The *cross* product (or vector product) $\vec{u} \times \vec{v}$ constructs a vector orthogonal to $\vec{u}, \vec{v}$
    - The magnitude of the result is the *area* of the parallelogram with sides $\vec{u}, \vec{v}$
    - Computed in `R` using `crossprod()`

Rethinking Multiplication: Graphically

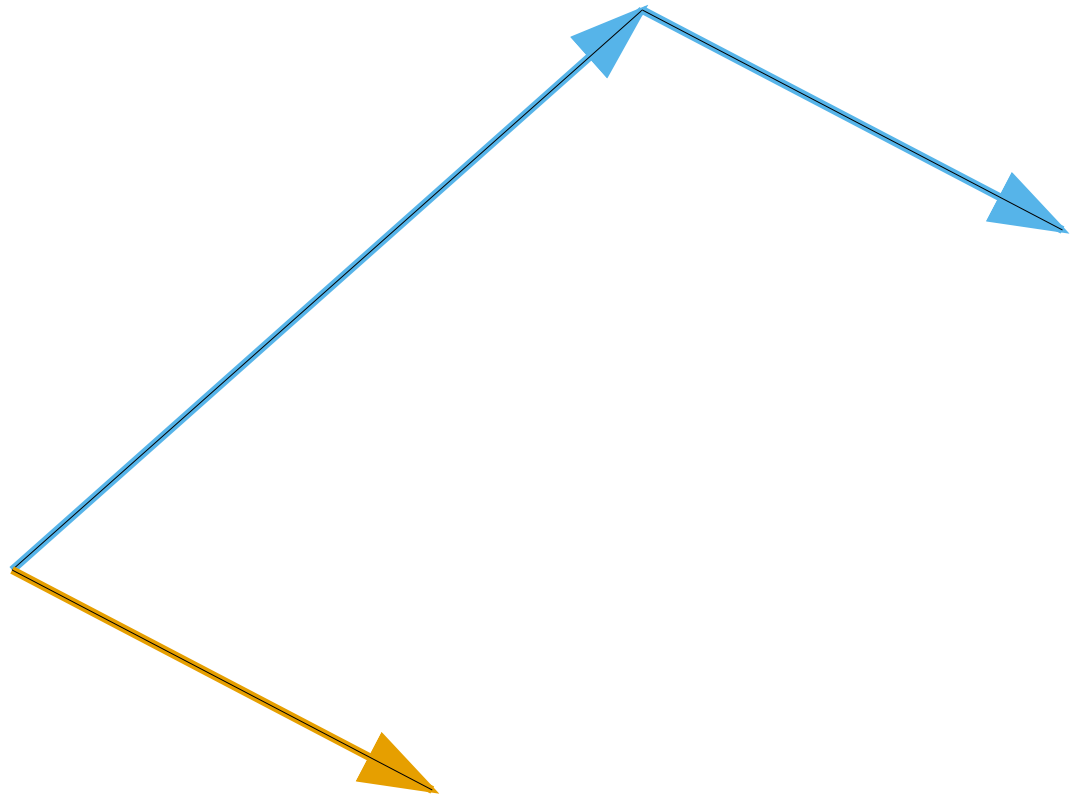
Rethinking Multiplication: Graphically



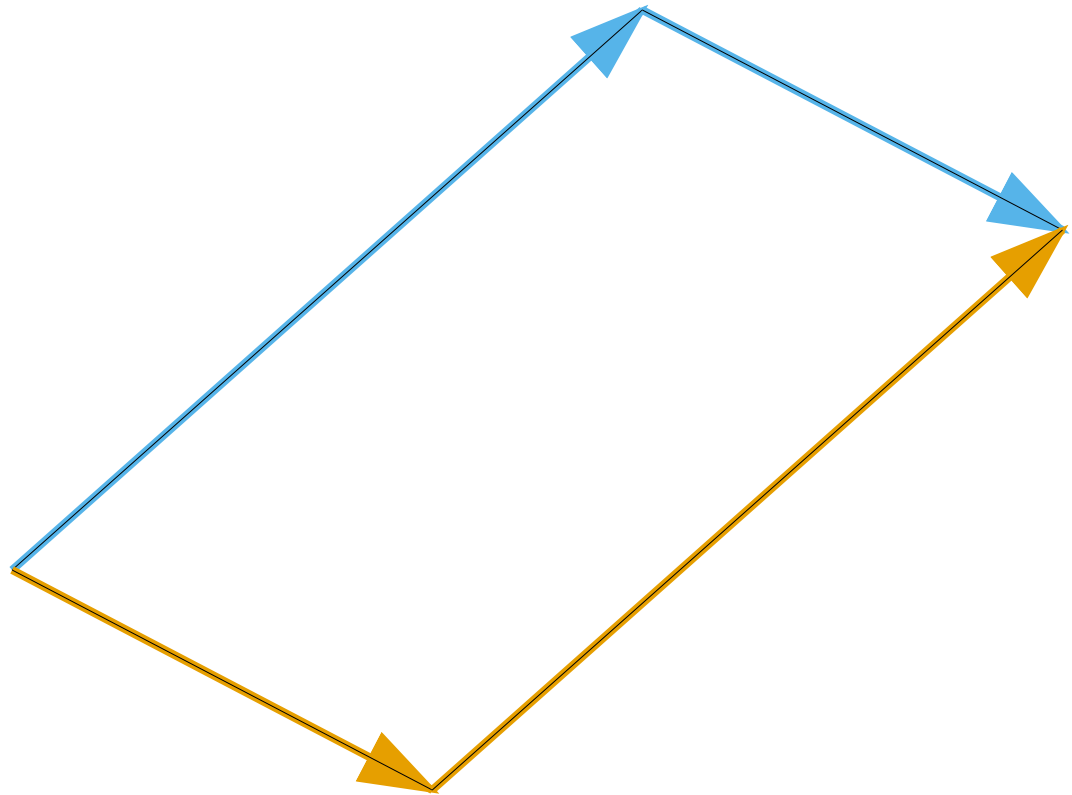
Rethinking Multiplication: Graphically



Rethinking Multiplication: Graphically

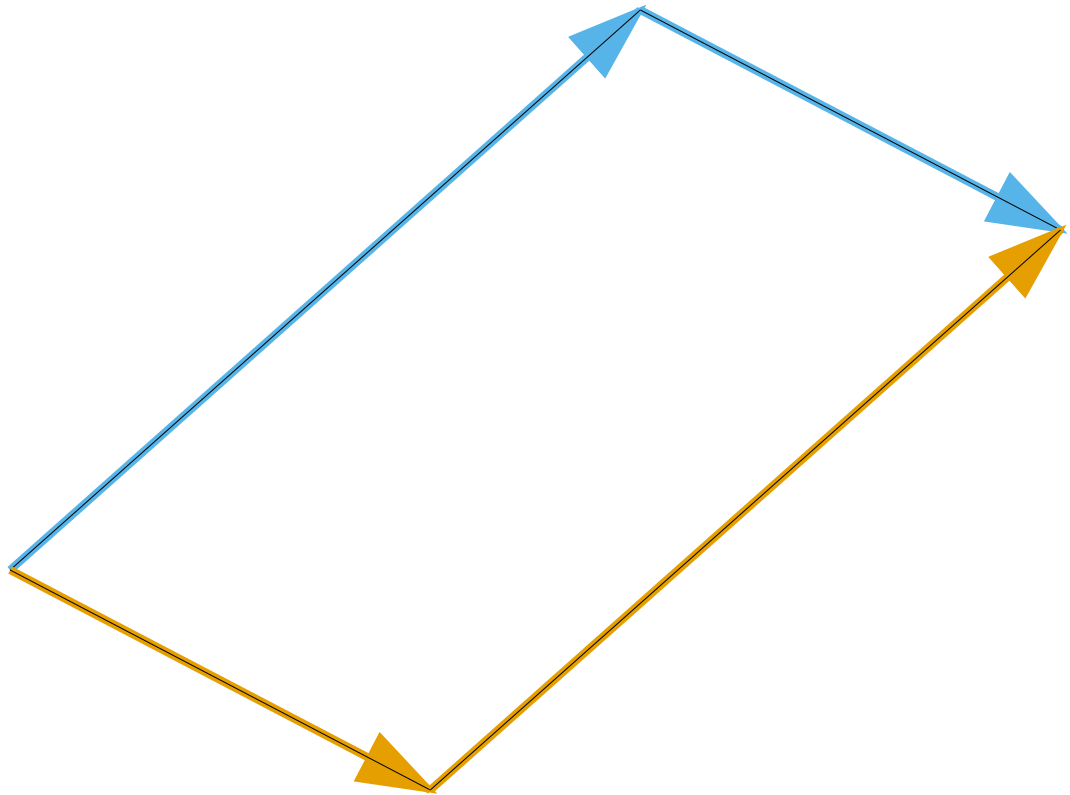


Rethinking Multiplication: Graphically



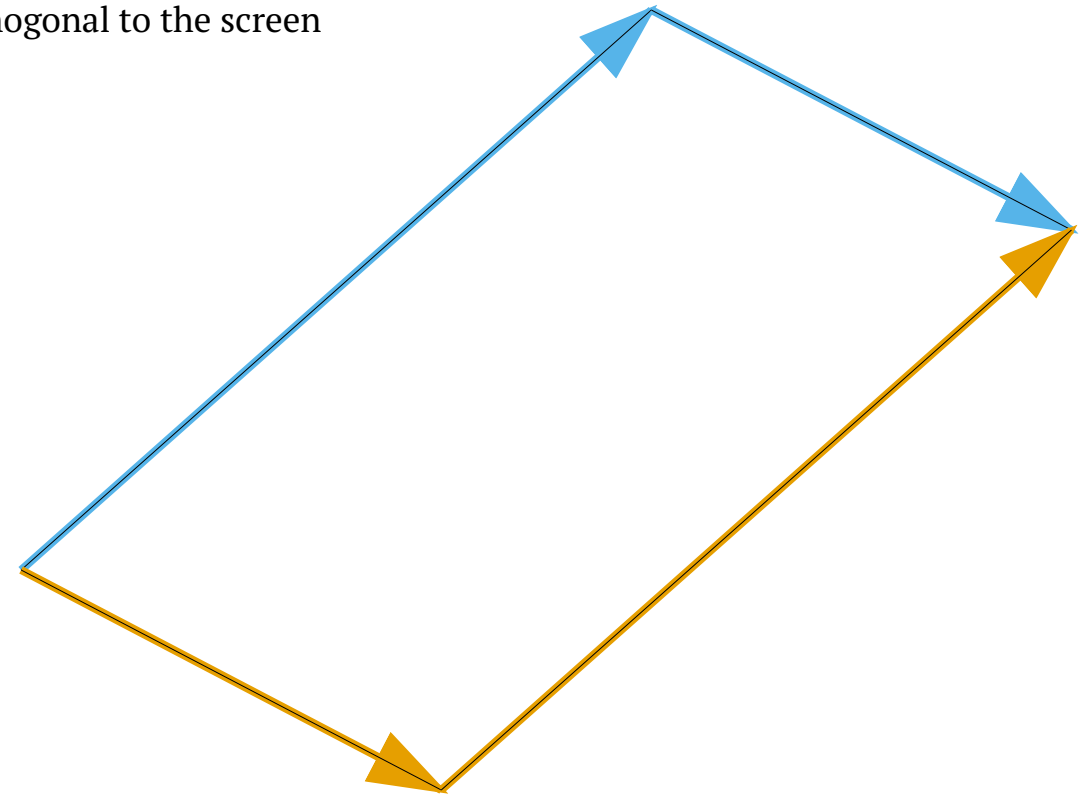
Rethinking Multiplication: Graphically

Magnitude: Area of the paralellogram



Rethinking Multiplication: Graphically

Magnitude: Area of the paralellogram
Direction: Orthogonal to the screen



What About Matrices?

- We add a new operation - *transposition*
  - Switches the rows and columns
  - Done in `R` using `t()`
- Matrix addition is still translation and done element-wise
  - `R` does this correctly
- Matrix multiplication is weird
  - Requires that the number of columns on the left matrix is equal to the number of rows on the right matrix
  - Result produces a matrix with the number of rows from the left matrix and the number of columns from the right matrix
  - For the matrix multiplication $\mathbf{AB} = \mathbf{C}$:
    - $\mathbf{C}_{ij} = a_i \cdot b_j$, where a_i is the ith row vector in $\mathbf{A}$ and b_j is the jth column vector in $\mathbf{B}$
    - Computed in `R` using `%*%`

Multiplying Matrices

What are the results?

1. $\begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix} \begin{bmatrix} 1 & 1 \\ 1 & 1 \end{bmatrix} =$

2. $\begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix} \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} =$

3. $\begin{bmatrix} 1 & 2 \\ 3 & 4 \\ 5 & 6 \end{bmatrix} \begin{bmatrix} 1 & 1 \\ 2 & 2 \\ 3 & 3 \end{bmatrix} =$

4. $\begin{bmatrix} 1 & 2 \\ 3 & 4 \\ 5 & 6 \end{bmatrix} \begin{bmatrix} 1 & 2 & 3 \\ 1 & 2 & 3 \end{bmatrix} =$

Multiplying Matrices

What are the results?

1. $\begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix} \begin{bmatrix} 1 & 1 \\ 1 & 1 \end{bmatrix} = \begin{bmatrix} 3 & 3 \\ 7 & 7 \end{bmatrix}$
2. $\begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix} \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} =$
3. $\begin{bmatrix} 1 & 2 \\ 3 & 4 \\ 5 & 6 \end{bmatrix} \begin{bmatrix} 1 & 1 \\ 2 & 2 \\ 3 & 3 \end{bmatrix} =$
4. $\begin{bmatrix} 1 & 2 \\ 3 & 4 \\ 5 & 6 \end{bmatrix} \begin{bmatrix} 1 & 2 & 3 \\ 1 & 2 & 3 \end{bmatrix} =$

Multiplying Matrices

What are the results?

1. $\begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix} \begin{bmatrix} 1 & 1 \\ 1 & 1 \end{bmatrix} = \begin{bmatrix} 3 & 3 \\ 7 & 7 \end{bmatrix}$
2. $\begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix} \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} = \begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix}$
3. $\begin{bmatrix} 1 & 2 \\ 3 & 4 \\ 5 & 6 \end{bmatrix} \begin{bmatrix} 1 & 1 \\ 2 & 2 \\ 3 & 3 \end{bmatrix} =$
4. $\begin{bmatrix} 1 & 2 \\ 3 & 4 \\ 5 & 6 \end{bmatrix} \begin{bmatrix} 1 & 2 & 3 \\ 1 & 2 & 3 \end{bmatrix} =$

Multiplying Matrices

What are the results?

1. $\begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix} \begin{bmatrix} 1 & 1 \\ 1 & 1 \end{bmatrix} = \begin{bmatrix} 3 & 3 \\ 7 & 7 \end{bmatrix}$
2. $\begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix} \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} = \begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix}$
3. $\begin{bmatrix} 1 & 2 \\ 3 & 4 \\ 5 & 6 \end{bmatrix} \begin{bmatrix} 1 & 1 \\ 2 & 2 \\ 3 & 3 \end{bmatrix} = \text{NOPE}$
4. $\begin{bmatrix} 1 & 2 \\ 3 & 4 \\ 5 & 6 \end{bmatrix} \begin{bmatrix} 1 & 2 & 3 \\ 1 & 2 & 3 \end{bmatrix} =$

Multiplying Matrices

What are the results?

1. $\begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix} \begin{bmatrix} 1 & 1 \\ 1 & 1 \end{bmatrix} = \begin{bmatrix} 3 & 3 \\ 7 & 7 \end{bmatrix}$
2. $\begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix} \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} = \begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix}$
3. $\begin{bmatrix} 1 & 2 \\ 3 & 4 \\ 5 & 6 \end{bmatrix} \begin{bmatrix} 1 & 1 \\ 2 & 2 \\ 3 & 3 \end{bmatrix} = \text{NOPE}$
4. $\begin{bmatrix} 1 & 2 \\ 3 & 4 \\ 5 & 6 \end{bmatrix} \begin{bmatrix} 1 & 2 & 3 \\ 1 & 2 & 3 \end{bmatrix} = \begin{bmatrix} 3 & 6 & 9 \\ 7 & 14 & 21 \\ 11 & 22 & 33 \end{bmatrix}$

But what is Matrix Multiplication?

- Multiplying a vector $\vec{u}$ by the $n \times m$ matrix $\mathbf{A}$ does something *really* weird
  - It provides a linear transformation from $\mathbb{R}^n \rightarrow \mathbb{R}^m$
  - So: $\vec{u}\mathbf{A} = \vec{u}'$, where $\vec{u}$ is n-dimensional and $\vec{u}'$ is m-dimensional
  - The transformations encoded in A can scale, squeeze, shear, reflect, and rotate the vectors they are applied to!
- If I construct a matrix $\mathbf{U}$ where the columns are a bunch of vectors, the multiplication $\mathbf{UA}$ transforms all of the column vectors of U simultaneously
  - This is how internally three-dimensional representations of spaces in video games are quickly projected to two dimensions for display on a screen
  - The camera's position and orientation describes a linear transformation encoded in a matrix, and this multiplication decides what is shown
  - It's why GPUs/graphics cards are just optimized to do lots of matrix multiplication really really fast
  - Deep learning is also just a ton of matrix multiplication, which is why it's done on GPUs not CPUs

Principal Component Analysis

Finding Structure in Matrices with PCA

Finding Structure in Matrices with PCA

- **Key Idea:** We can represent our data in a matrix, $\mathbf{X}$, and then construct another matrix that projects our data into a lower dimensional space ($m < n$) that has nice properties derived from the structure of $\mathbf{X}$ so that it preserves a bunch of really important information in $\mathbf{X}$ and makes it easier to look at!
- What kind of structure can we find? How do we find it in a smart way?
- Today we'll introduce *Principal Component Analysis* (PCA) as one way to find structure and then look at a way to do it
- The overarching goal of PCA:
  - We have data, comprised of n m-dimensional vectors, $\mathbf{x}_1, \ldots, \mathbf{x}_n$
    - Here, n is our number of observations and m is our number of variables
  - We want to express this data matrix as linear combinations of k m-dimensional vectors, $\mathbf{v}_1, \ldots, \mathbf{v}_k$, so that $\mathbf{x}_i \approx \sum_{j=1}^{k} a_{ij}\mathbf{v}_j$
  - We want $k < m$, so the collection of k vectors is a good approximation of the information in our original m variables

Principal Component Analysis

- The first *principal component* (when $k = 1$) is a special type of "best-fit line" that minimizes the average squared Euclidean distance (or *reconstruction error*) between each data point and the line:

$$\underset{\mathbf{v}:||\mathbf{v}||=1}{\operatorname{argmin}} \frac{1}{n} \sum_{i=1}^{n} (\text{distance between } \mathbf{x}_i \text{ and the line defined by } \mathbf{v})^2$$

- Note that $||\mathbf{v}|| = \sqrt{\mathbf{v} \cdot \mathbf{v}}$ and is the length of $\mathbf{v}$

Principal Component Analysis

- Finding that distance is kind of annoying, but we can wiggle this a bit using the Pythagorean theorem

$$\big(\text{dist}(\mathbf{x}_i \rightarrow \text{line})\big)^2 + \big(\mathbf{x_i} \cdot \mathbf{v}\big)^2 = ||\mathbf{x}_i||^2$$

- Because the RHS is a constant, minimizing the first term is *maximizing* the second term!

$$\underset{\mathbf{v}:||\mathbf{v}||=1}{\text{argmax}} \frac{1}{n} \sum_{i=1}^{n} \big(\mathbf{x_i} \cdot \mathbf{v}\big)^2$$

- This means PCA is maximizing the average variance in $\mathbf{x_i}$ explained by the line defined by $\mathbf{v}$

More Components = More Good

- What about when $k > 1$?
- The Top-k Principal Components are the k orthonormal vectors $\mathbf{v}_1, \ldots, \mathbf{v}_k$ that maximize:

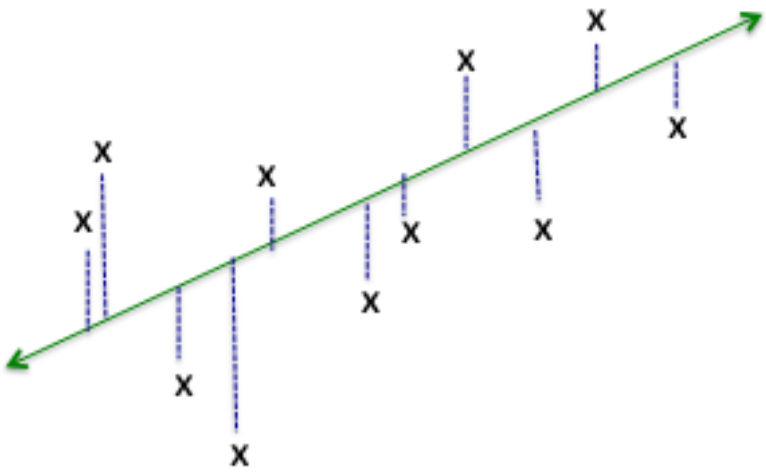
$$\frac{1}{n} \sum_{i=1}^{n} \sum_{j=1}^{k} \left(\mathbf{x_i} \cdot \mathbf{v}_j \right)^2$$

- Orthonormal means the set of vectors we find are orthogonal and normalized so $||\mathbf{v}_j|| = 1$
  - If $\mathbf{u}, \mathbf{v}$ are orthogonal, $\mathbf{u} \cdot \mathbf{v} = 0$

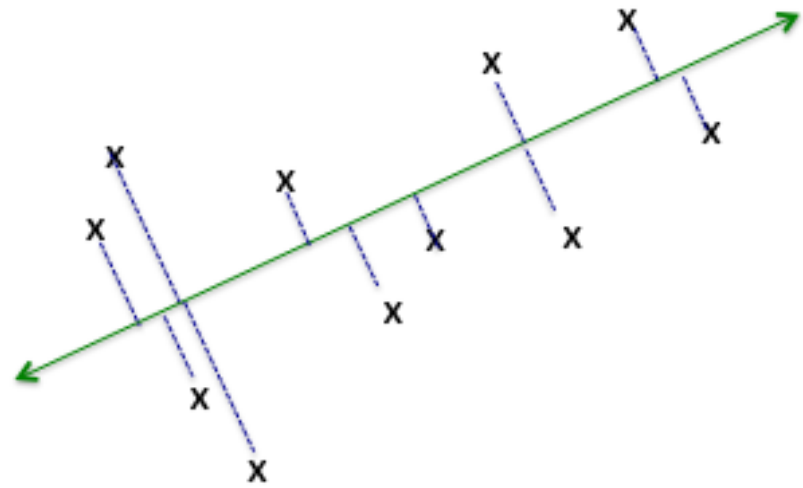
A Big Warning

- You may have seen some things that make you think PCA is a lot like OLS regression—**do not be fooled!**
  - They have different uses
  - They minimize different objective functions
- What are the core differences?
  - OLS minimizes the *prediction error*, that is $\sum_i (y_i - \hat{y}_i)^2$
    - Always relative to some favored outcome, y_i, and minimizes the vertical distance to the line
  - PCA has no outcome variable, and minimizes the *reconstruction error*
    - No favored outcome, so minimizes the orthogonal distance to the line

Contrasting OLS and PCA



(a) Linear regression



(b) PCA

Algorithm: Power Iteration

Rewriting the Optimization Problem of PCA

Let's rewrite our objective function in terms of matrix operations, starting from the $k = 1$ case:

$$\underset{\mathbf{v}:||\mathbf{v}||=1}{\operatorname{argmax}} \frac{1}{n} \sum_{i=1}^{n} \left(\mathbf{x_i} \cdot \mathbf{v}\right)^2$$

1. We construct a data matrix, $\mathbf{X}$, that has our observations $\mathbf{x}_1, \ldots, \mathbf{x}_n$ along the rows
2. For a unit vector, $\mathbf{v}$, we can write:

$$\mathbf{X}\mathbf{v} = \begin{bmatrix} \mathbf{x}_1 \cdot \mathbf{v} \\ \vdots \\ \mathbf{x}_n \cdot \mathbf{v} \end{bmatrix}$$

Rewriting the Optimization Problem of PCA

3. We want the sum of the squares of these, so we take the dot product of $\mathbf{Xv}$ with itself:

$$(\mathbf{Xv})^\top(\mathbf{Xv}) = \mathbf{v}^\top\mathbf{X}^\top\mathbf{Xv} = \sum_{i=1}^{n}\left(\mathbf{x_i}\cdot\mathbf{v}\right)^2$$

4. Now we define a matrix $\mathbf{A} = \mathbf{X}^\top\mathbf{X}$, which we call the covariance, correlation, or co-occurrence matrix (depending on if the columns of $\mathbf{X}$ were normalized and the specific application)
5. We use a result from linear algebra: Any symmetric square matrix like $\mathbf{A}$ can be rewritten as the product of an orthonormal matrix, $\mathbf{Q}$ and a diagonal matrix, $\mathbf{D}$ like so:

$$\mathbf{A} = \mathbf{Q}^\top\mathbf{D}\mathbf{Q}$$

Pause: Why do all this?

- Remember that matrices define linear transformations—projections into spaces of different dimensions.
- What happens when you multiply a diagonal matrix? Well, it just scales each individual column or row (depending on how the multiplication is set up) by the diagonal elements of the matrix
- Think of this as a diagonal matrix, $\mathbf{D}$, stretching the dimensions of a matrix that you multiply it by
- For PCA, we are trying to find the dimensions that are stretched the most by the covariance matrix, $\mathbf{A} = \mathbf{X}^\top \mathbf{X}$
- As such, the top-k PCs of $\mathbf{A}$ are the columns of $\mathbf{Q}$ that correspond with the largest k diagonal elements of $\mathbf{D}$
- The columns of the orthonormal matrix $\mathbf{Q}$ are the *eigenvectors* of $\mathbf{A}$ and the associated diagonal elements of $\mathbf{D}$ are their *eigenvalues*
- We will return to eigenvalues and eigenvectors in a few weeks when we learn the Singular Value Decomposition—another way to solve this problem!

Power Iteration for Computing Eigenvectors

- This is an iterative method for computing eigenvectors
- The idea is that if multiplication by $\mathbf{A}$ is going to stretch a vector the most in the direction of the largest variance, if we apply multiplication by $\mathbf{A}$ repeatedly, we will eventually end up with a vector that points in the direction of the largest variance
  - This finds the eigenvector associated with the largest eigenvalue of matrix $\mathbf{A}$
  - If $\mathbf{A}$ is a covariance matrix, this means we also get the first PC!
- Algorithm:
  1. Select a random unit vector, $\mathbf{u}_0$
  2. for $i = 1, 2, \ldots$, set $\mathbf{u}_i = \mathbf{A}^i \mathbf{u}_0$. If $\mathbf{u}_i / ||\mathbf{u}_i|| \approx \mathbf{u}_{i-1} / ||\mathbf{u}_{i-1}||$, stop
  3. Return $\mathbf{u}_i / ||\mathbf{u}_i||$
- $\mathbf{v}_1 = \mathbf{u}_i / ||\mathbf{u}_i||$ is the largest eigenvector (and first PC) of A

Finding Additional PCs

- After the first instance of power iteration, we have the top-1 PC
- To find the next PC:
  1. Project the data matrix orthogonally onto $\mathbf{v}_1$: $\mathbf{x}_i \rightarrow \mathbf{x}_i - (\mathbf{x}_i \cdot \mathbf{v}_1)\mathbf{v}_1$
  2. Carry out power iteration on the projected data to find the next PC
  3. Repeat steps 1&2 until you want to stop finding PCs
- This makes power iteration a *greedy* algorithm (we'll come back to this idea, too)
- Typically, you can plot the eigenvalues and treat it like the "elbow plots" from k-Means. Sometimes people stop when the eigenvalues get below one. Sometimes people only keep the first 2-3 if they're just making plots.
  - Freak what you feel
- PS4 has you implementing Power Iteration to do PCA!

A Neat Power Iteration Application: PageRank

- The original Google PageRank Algorithm is just power iteration to find the first eigenvector of the web transition matrix!
- The idea was that "important pages link to important pages"
- They simulated web browsing as a random walk around the web graph
- They began by putting together a web-adjacency matrix, $\mathbf{A}$, where $\mathbf{A}_{ij} = 1$ if page i is linked to by j and $\mathbf{A}_{ij} = 0$ otherwise
- Then they simulated random web browsing behavior by defining a transition probability, $\beta = 0.05$, that you move to a random webpage that isn't linked from your current spot
- $\mathbf{A}$ got column normalized, so the entries summed to 1, so $\mathbf{A}_{ij}$ represents the probability of navigating to page i from page j

A Neat Power Iteration Application: PageRank

- How do you solve this problem, then?
- You find the first eigenvector of $\mathbf{A}$ through power iteration!
- This eigenvector represents the probability that a random web browser will be on any individual website, providing a measure of importance!
- Power iteration will show up again when we discuss Markov Chain Monte Carlo
- You'll apply it to measuring individual importance in a political network in a later pset
- But, generally, power iteration can be a powerful way to understand importance in directed and undirected networks through a modified application of PageRank

Back to PCA

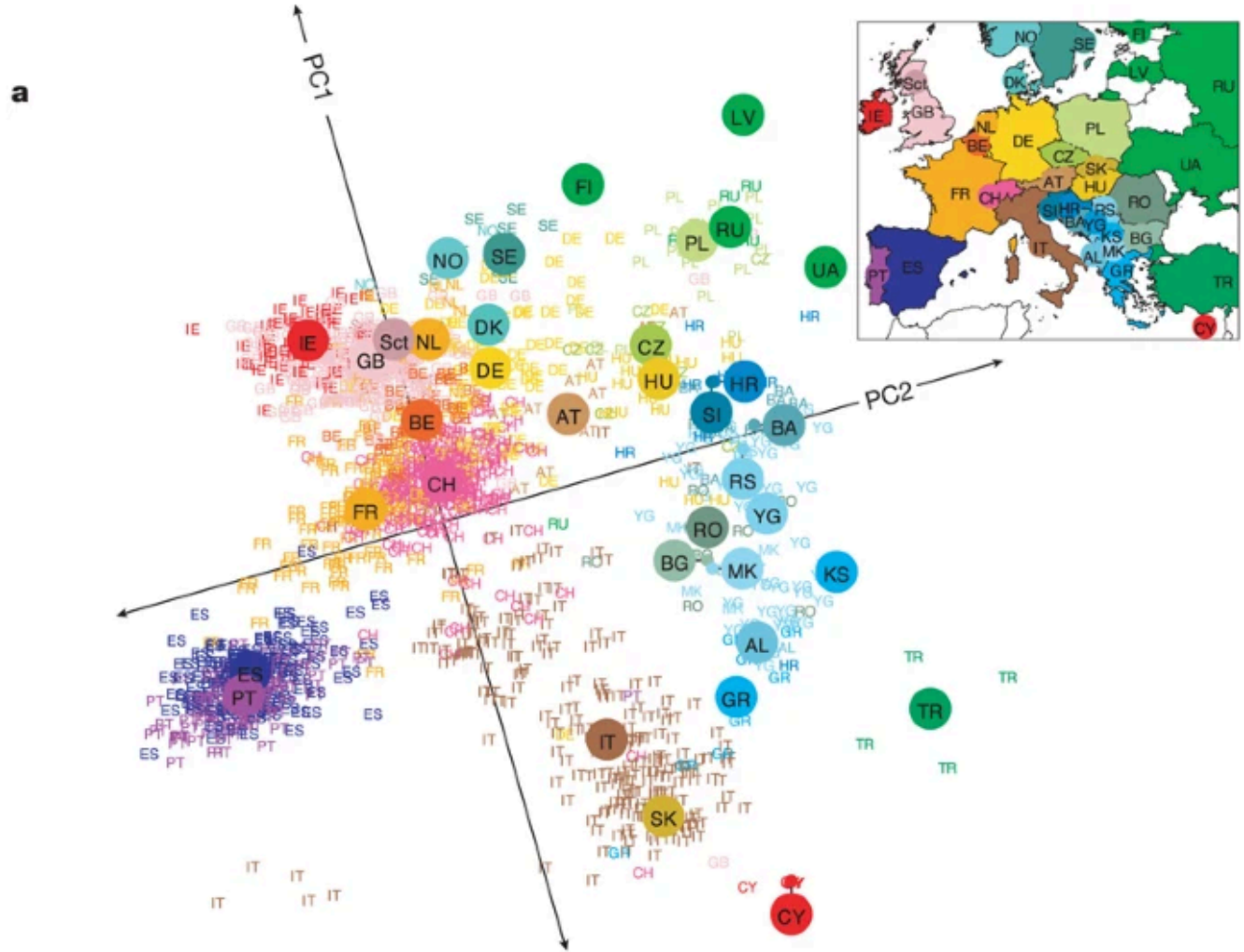
Using PCA

- PCs are often really interpretable
  - Look at what variables are the largest positive and largest negative weights—do they have things in common?
  - When you project your original data into the lower dimensional PC space, do observations that score high/low on some dimensions have things in common?
- Really commonly used to visualize data
- Also commonly used to "cluster" variables that may measure similar things
- Also can be used to "cluster" observations along the PCs
- PS4 will have you not only implementing PCA from scratch, but also exploring some of its properties!

A Neat PCA Application: Genes

- A genetics paper in 2008 published in Nature did something *very* cool with PCA!
- They took 3000 genotyped Europeans and performed PCA on their genomes. Then they plotted the individuals' projections onto the top 2 components
  - What do you think the top two PCs captured?
  - What do you think they saw?
- John Novembre, Toby Johnson, Katarzyna Bryc, Zolt´an Kutalik, Adam R. Boyko, Adam Auton, Amit Indap, Karen S. King, Sven Bergmann, Matthew R. Nelson, Matthew Stephens, and Carlos D. Bustamante. (2008) Genes mirror geography within Europe. *Nature*, 456:98–101.

Figure 1: Population structure within Europe.



Wrap Up

Recap

- As it turns out, the structure of matrix math can make a lot of potentially complex calculations much simpler
- Power iteration is an algorithm we'll return to; if you can reframe problems in terms of repeated matrix multiplication, it's a very easy way to solve them!
- PCA is a really strong visualization tool and good way to summarize your data. You'll implement and interpret it in lab and PS4!

Final Thoughts

- PollEv.com/klintkanopka