EECS 245

EECS 245 Course Notes

University of Michigan

Compiled on Tuesday 19th May, 2026
Last repo commit: May 19, 2026 at 10:29 PM UTC (a8a8d04)

Note: This PDF export is experimental and is bound to have bugs, and any interactive visualizations and diagrams won't be present. If something doesn't look right, let us know at rampure@umich.edu.

About These Notes

Mathematics for Machine Learning: Course Notes for EECS 245 at the University of Michigan

by **Suraj Rampure** (rampure@umich.edu; rampure.org)

Google Chrome Recommended

For the best experience, please view these notes using **Google Chrome** (or another Chromium-based browser, like Arc). You may experience slower scrolling and performance issues if using Safari.

Overview

Linear algebra, calculus, and probability form the basis of modern machine learning and artificial intelligence. **This course introduces linear algebra from scratch by focusing on methods and examples from machine learning.** It will give students strong intuition for how linear algebra, calculus and probability are used in machine learning. While the course is primarily theoretical, we'll look at practical applications involving real data in Python each week, so that students are able to apply what they've learned.

These notes are organized into 10 core chapters, plus an appendix that covers some mathematical foundations that are used throughout the course.

1. Introduction to Supervised Learning
2. Simple Linear Regression
3. Vectors
4. Linear Independence
5. Matrices
6. Linear Transformations and Projections
7. Regression using Linear Algebra
8. Gradients
9. Eigenvalues and Eigenvectors
10. Singular Value Decomposition

Each chapter consists of several pages, called "sections", so Chapter 3.2 (for example) is a section.

Prerequisites

The prerequisites for this course at the University of Michigan are Calculus 2 *or* discrete mathematics. That is, we **don't** assume students have already seen linear algebra or multivariable calculus, but assume some mathematical maturity beyond a first course in calculus. We will also assume some basic programming experience; we will show examples of Python code throughout.

Why Write These Notes?

There are a plethora of great resources for learning these ideas already, and I've read and taken inspiration from *many* of them (linked below) in writing these notes and preparing for EECS 245.

So, why write these notes? I – like other teachers – have a "story" in my mind that I want to tell, and that story has a particular order and flair that I haven't seen in other books or courses on linear algebra. Most linear algebra courses start with analyzing systems of equations and unknowns. These are extremely important to machine learning, as they are in other fields, but not in an immediately obvious way (at least not to me). This course will start by introducing the foundations of supervised learning – a branch of machine learning that deals with predicting a target variable, y, given a set of input variables, x – and dive deep into the relevant ideas from linear algebra as they become necessary to advance this story.

This is, admittedly, an experiment in how to teach linear algebra. We may change directions in future iterations of the course if it turns out that the best practice is to cover content in a more conventional order. But, I'm excited and optimistic about this plan, and you, the student, are sure to learn a great deal regardless. The first offering in Fall 2025 went well, so I'm optimistic you'll have a great experience.

While I could refer students to different books for different topics, I want to provide a single source of truth that I can refer students to, so that the course narrative is consistent. This is especially important since EECS 245 will be taught *without* slides. Instead, in each lecture, I will start with a blank whiteboard (on my iPad) and provide high-level overviews of the content. These notes will follow the same story, but with added details, examples, and exercises.

Additionally, I'm using these digital notes as an opportunity to develop interactive visualizations of various concepts in linear algebra, to bring these ideas to life.

Acknowledgements, Feedback, and License

Here are some other resources that I've found helpful in writing these notes. Some of these are also linked in particular pages of the notes.

- Gilbert Strang's lecture videos and textbook on linear algebra. The intuition Strang provides in his lecture videos is legendary, and I've studied them closely in preparing for EECS 245. He also has a new book titled Linear Algebra and Learning from Data which is similar in spirit to these notes. We've borrowed many exercises from his book.
- 3blue1brown's Essence of Linear Algebra series on YouTube is a great resource for developing a visual intuition for linear algebra.

If you have any feedback or suggestions, please don't hesitate to reach out to me at `rampure@umich.edu`. (If you're a stranger on the internet who has found these notes, I'd love to hear from you too!) Alternatively, feel free to use the feedback form linked in the top-right corner of the page.

In addition to the resources above, I'd like to thank **Vincent Shen**, a former student of mine, who helped write and proofread various sections of the notes in Summer 2025.

Accessibility

It's important to us that all students are able to fully engage with our course content. We're working to ensure these notes meet WCAG 2.1 AA accessibility standards, as mandated by the federal government of all public

universities in the United States. If you find any accessibility issues (hard-to-read colors, missing alt text, videos with missing captions, or anything else that makes the content difficult to access), please let Suraj know as soon as possible at rampure@umich.edu.

Advice to the Student

EECS 245 was first offered in Fall 2025 at the University of Michigan. At the end of that semester, we asked students for advice to relay to future students. Here are some (unedited) responses we received.

> The only time where reading the notes is a lot more helpful than you think - read the notes.
>
> READ THE COURSE NOTES. This is more important than going and watching lectures
>
> Reading course notes and mastering course exercises is really necessary for doing well in the course.
>
> Treat this as similar to 203, and read the lecture notes.
>
> Make sure you tell people that the course notes are MANDATORY (I got cooked on exam one because I didn't know something existed in the course notes and not in lecture).
>
> This class showed me the way I wish all classes were taught. Writing the notes as the class was being taught really let me see the thinking behind everything rather than just reading off of a slideshow, and then the course notes allowed me to go back and make sure I knew everything correctly.
>
> I loved the course notes!!! Whenever I missed lecture it felt really easy to get up to speed because of how clear they were!!!! 10000/10, wish more courses had lecture notes like this
>
> I would just make it clear at the beginning of the course that course notes and exercises are your best friend

This is all to say, **make sure to get in the habit of reading the course notes AND doing the exercises and activities in them**. They are your friend!

Introduction to Supervised Learning

1.1. What is Machine Learning?

If you're reading this, you've chosen to take a course that explores the mathematical foundations of machine learning, with a particular focus on linear algebra. I'm not going to assume you know anything about machine learning or linear algebra. But, before I dive into any details, I do want to give you a sense of what machine learning actually is, and how recent advances like ChatGPT – which you almost certainly know something about – connect to what you'll learn.

Motivation

Example: Handwritten Digit Recognition. Using your mouse, draw a **handwritten digit in the box below**. As you draw, a *model* (source) will try and guess the digit you're drawing, and describe how confident it is in its guess.

Draw a few different digits, and hit "clear" after each one. You'll notice that the model isn't perfect (for instance, it often confuses my 0s with 2s, 5s, 7s, and 9s, depending on where in the box I draw the 0), but it works reasonably well.

How did the model figure out how to identify digits, and why is it sometimes wrong?

If you were to implement such a model, one technique you *could* try is to hardcode a bunch of `if`-statements that check for certain patterns in the digit you drew. For example, you could check if the digit has a straight line in the middle, as that would make it likely to be a 1, 7, or 9; or if it has a closed loop, as that would make it likely to be a 0, 2, 4, 6, 8, or 9. But, this is very tedious, and requires you to have a deep understanding of the patterns that make up each digit. And if there were 100 possible digits instead of 10, this would be much more difficult.

The Machine Learning Way. Machine learning uses a different approach. **Machine learning is about automatically learning patterns from data.**

In the case of the digit recognition model, the creators of the model showed the model thousands of pictures of handwritten digits, **along with the correct answer for each digit**. These correct answers are called **labels**. The model then used this data to learn the patterns that make up each digit. This process, of having a model learn the patterns of the data so that we can use it to make predictions, is called **training** the model.

The specific dataset this model was trained on is called the MNIST database, a common dataset used to test and benchmark machine learning models.

I'm using the term "model" loosely here. One, I haven't formally defined the term, and I also haven't actually specified *how* the model makes its guesses. There are plenty of types of models that we could train for the purposes of digit recognition – neural networks are the most popular choice these days, but decision trees, support vector machines, logistic regression, and so on are all reasonable choices. More flexible models can learn more complex patterns, but come with the risk of overfitting to the training data; this is an idea we'll discuss more in Chapter 1.2.

The MNIST database contains 70,000 images of handwritten digits, each of which is grayscale (no color) and 28x28 pixels. The dataset contains the true labels for each image – that is, the model is told that the first row above contains all 0s, the second row contains all 1s, and so on. If we wanted to instead build a model that could guess whether an image is of a cat, dog, or hamster, we'd need to show the model many labeled images of cats, dogs, and hamsters.

A Brief History. Machine learning – and more recently, *artificial intelligence* – are hot topics these days, and indeed, new breakthroughs appear every few months. But, the origins of the field are much, much older than you and me.

Linear regression, which I think of as the foundation of machine learning, was first used in the early 1800s by

Figure 1.1: *
A few of the images in the MNIST dataset.

physicists Legendre and Gauss to predict the motions of bodies in the solar system. That's the same Gauss behind the Gaussian distribution, i.e. normal distribution or bell curve, that is ubiquitous in statistics and that we'll study in a later chapter on probability. (In particular, we'll learn about how the normal distribution plays a role in machine learning.)

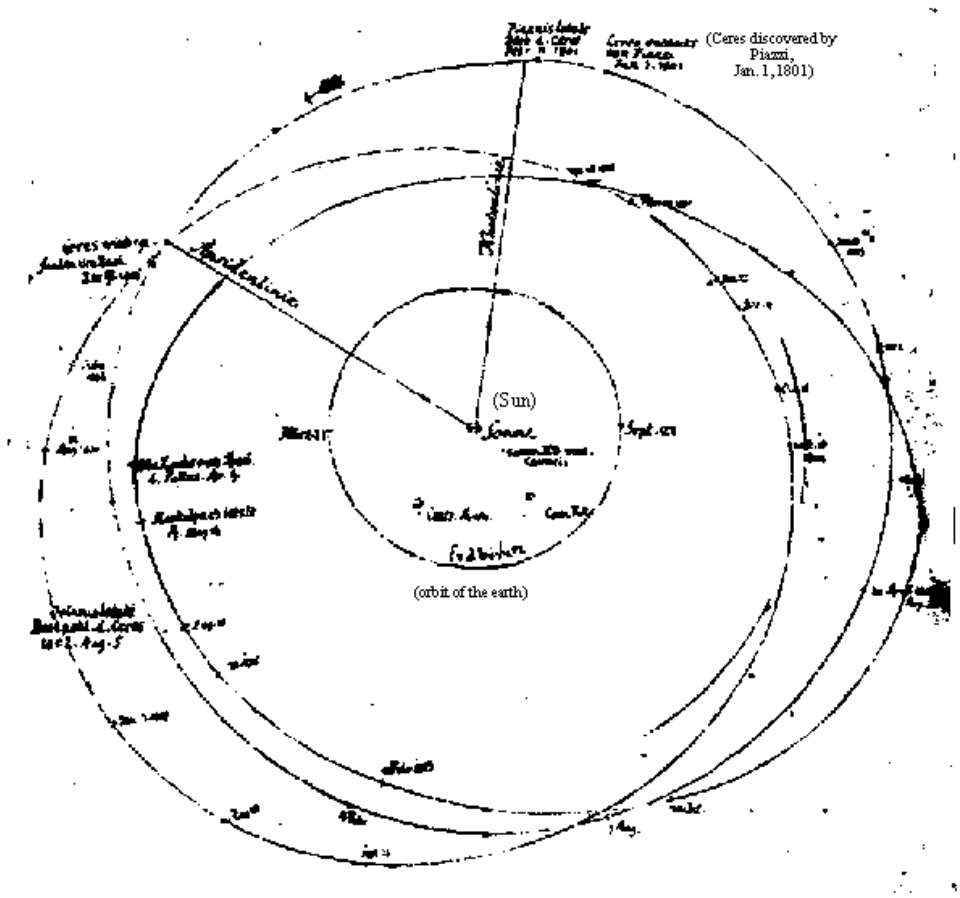
Regression was then given the name "regression" and further developed by Galton and Pearson in the late 1800s to predict the heights of children given the heights of their parents. Galton was the cousin of Charles Darwin and was the originator of the field of eugenics, which led to a dark period in history.

Neural networks, which power most of the advances you've heard of recently, were first developed by neuroscientists in the 1940s to model the behavior of neurons in the brain. For now, think of a neural network as a sophisticated type of model that can be trained on data to make predictions. The popularity of neural networks skyrocketed in the early 2010s, after they were shown to outperform traditional AI-but-not-neural-network-based models on tasks like image recognition. This improvement in performance was made possible by several factors, including new mathematical techniques but also the advent of GPUs that made it possible to train models on large datasets efficiently. (I'll touch more on what a GPU is and why they're relevant in a later chapter.)

The physicists and statisticians of the 1800s did not think they were "doing" machine learning, as that term only came into use in the 1960s. To hear more about how machine learning and statistics are related fields, but with slightly different goals, read the famous Two Cultures paper by Leo Breiman. And if you want to know more about the history of statistics, machine learning, and computer science, this seminar course I taught at UC San Diego might be of interest to you.

Focus on the Ideas!

The rest of this section will move quickly and introduce a lot of new ideas. Focus on just the big-picture ideas. Any relevant mathematical details will be re-introduced in Chapter 1.2 onward.



Sketch of the orbits of Ceres and Pallas (nachlaß Gauß, Handb. 4). Courtesy of Universitätsbibliothek Göttingen.

Figure 1.2: *
An early sketch by Gauss, when trying to determine the motion of the planet Ceres, for which he developed the method of least squares.

Supervised Learning

There are several types of problems that we can use machine learning to solve. The taxonomy below shows *some* of them.

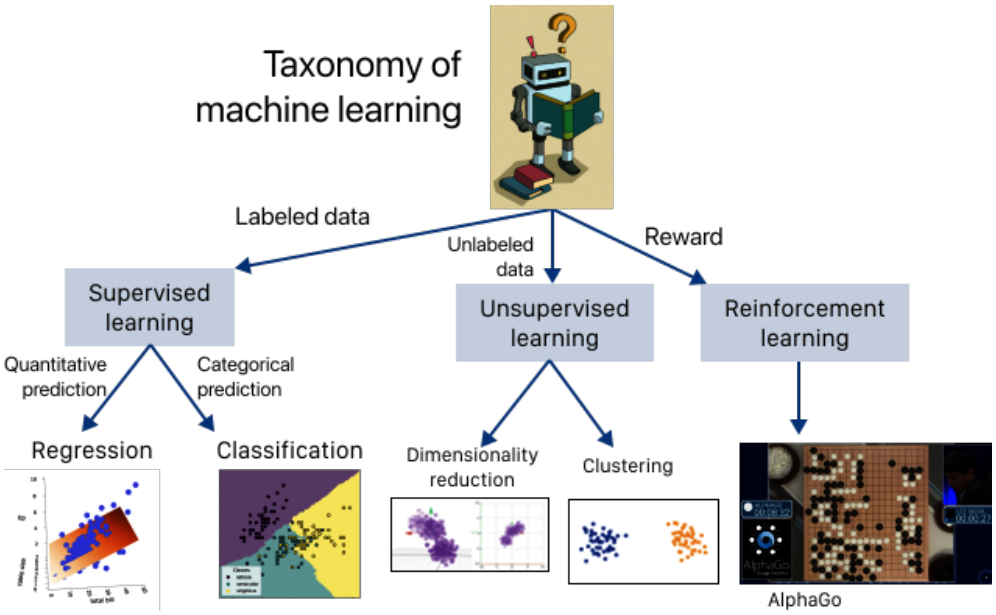


Figure 1.3: *
Source: Sam Lau

Supervised learning is the type of machine learning that we'll spend most of our time on, and the digit recognition problem from above is an example of a supervised learning problem.

Definition: Supervised Learning

A supervised learning problem can be thought of as "given data, **predict** a label", or "given x, predict y". To train a model for a supervised learning problem, we need to have a dataset that contains information about each individual, along with the correct label for each individual.

The "supervising" comes from having a correct answer (label) for each situation in the dataset; you can think of the computer as being "supervised" by the correct answers.

There are two types of supervised learning problems: classification and regression.

Classification. In a classification problem, the label is a category, i.e. a categorical variable, with a finite number of possible values.

Example	Labels
Digit recognition	0, 1, 2, 3, 4, 5, 6, 7, 8, or 9
Email spam detection	"spam" or "not spam"
Image classification	"dog", "cat", "hamster", or "zebra"
Medical diagnosis	"diabetes" or "no diabetes"
Risk assessment	"low risk", "medium risk", or "high risk"

The email spam detection and medical diagnosis problems are **binary** classification problems, as there are only two possible labels. In binary classification problems, we typically represent the labels as 1 (spam, diabetes)

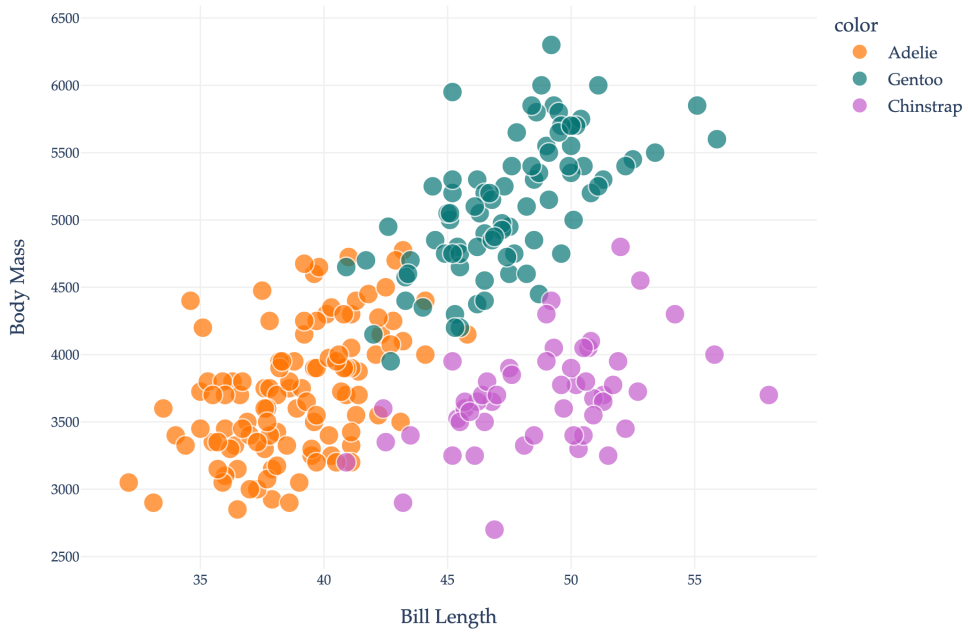
and 0 (not spam, no diabetes).

As another example, suppose we have a dataset of penguins. For each penguin in the dataset, we know its bill length in millimeters, its body mass in grams, and its species, which is either Adelie, Chinstrap, or Gentoo. The dataset shown below reflects real data about real penguins in Antarctica; read more about the dataset here.

	Bill Length	Body Mass	Species (Label)
0	43.200000	4100.000000	Adelie
1	43.200000	4775.000000	Adelie
2	46.200000	5300.000000	Gentoo
3	50.200000	3775.000000	Chinstrap
4	51.300000	3700.000000	Chinstrap
5	48.200000	5100.000000	Gentoo
6	40.100000	4300.000000	Adelie
7	36.000000	3450.000000	Adelie
8	43.500000	4700.000000	Gentoo
9	50.900000	3675.000000	Chinstrap

The **boxed column** contains the labels for each penguin. Think of this as the "y" in "given x, predict y", or as the answer key.

We can use a scatter plot to visualize the relationship between body mass and bill length, with colors indicating the species.



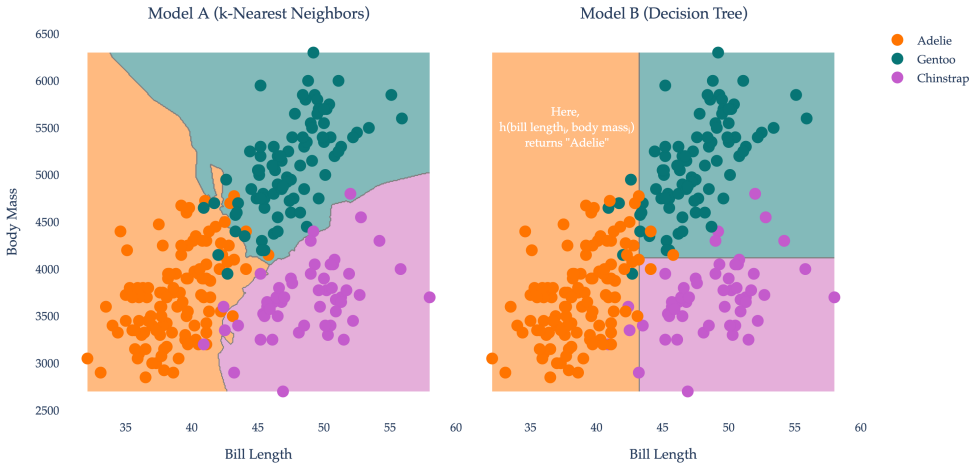
Fun fact: our local Detroit Zoo is home to Chinstrap and Gentoo penguins.

Once we train a model to predict the species of penguins, we can use it to predict the species of **new** penguins who weren't in our original dataset. All models for this task take the form of a function h:

$$\text{predicted species}_i = h(\text{bill length}_i, \text{body mass}_i)$$

where h can only output one of three things: Adelie, Chinstrap, or Gentoo.

The **decision boundaries** of two different models – that is, two different functions h – are shown below. The regions are colored based on what the model would predict for a new penguin.



For instance, if some new penguin comes along with a bill length of 38 mm and a body mass of 6000 g, Model A would predict that it is a Gentoo penguin, while Model B would predict that it is an Adelie penguin. You can think of a model's job as being to best separate the different species of penguins, based on the characteristics of the penguins it has access to. In order for a model to be useful, it needs to be able to make good predictions for new penguins, not just the penguins it already saw.

I've provided comprehensive examples of classification problems above, mostly because most of our semester will involve regression, **not** classification, and I want to make sure you're familiar with the broad landscape of machine learning.

Regression. In a regression problem, the label is a continuous variable, i.e. a numerical variable that can take on any real number.

Example	Labels
Commute time prediction	Times in minutes (82, 63.1256, -4, etc.)
House price prediction	Prices in dollars
Temperature prediction	Degrees in Fahrenheit

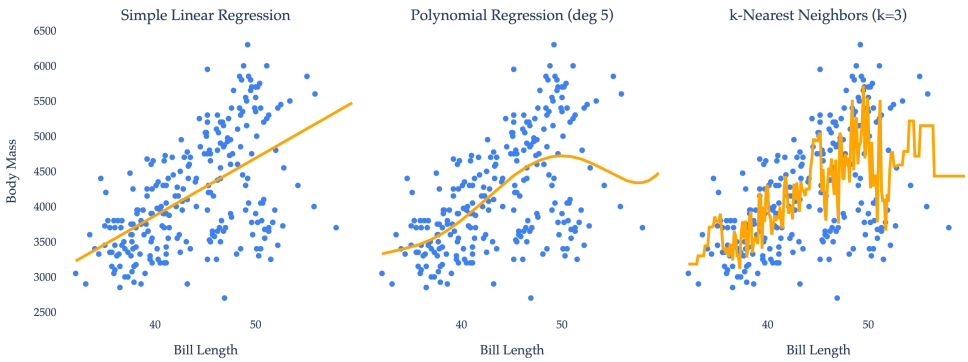
The first problem mentioned, commute time prediction, is the one we'll begin studying in Chapter 1.2; we'll revisit it throughout the semester. Training a model will involve finding a useful function h such that:

$$\text{predicted commute time}_i = h(\text{time I leave home}_i, \text{day of week}_i)$$

In the context of the penguin example, what might a regression problem look like? One possibility is to predict the body mass of a penguin given its bill length, i.e. to find a function h such that:

$$\text{predicted body mass}_i = h(\text{bill length}_i)$$

Three possible functions h are shown below. Each function's predictions are shown in orange.



In Chapter 1.2, we'll learn *how* to find models that make good predictions, starting with (a simpler version of) the simple linear regression model from above. As we'll soon see, this linear regression model is called "simple" because it only takes in one input variable, or **feature** (bill length). "Multiple" linear regression models take in multiple features, e.g. if we used bill length and bill depth to predict body mass.

Activity 1

Activity 1

Identify whether each of the following problems is a classification or regression problem.

1. Given customer purchase history and demographics, predict whether they will buy a premium membership next month.
2. Using sensor data from manufacturing equipment, predict the remaining useful life of a machine component in hours.
3. Based on patient symptoms and vital signs, determine if treatment should be aggressive, moderate, or conservative.
4. Using weather data and soil conditions, predict crop yield per acre for the upcoming harvest season.
5. Given user behavior patterns on a website, predict which content category they will click on next.

The distinction between classification and regression isn't always straightforward. For instance, predicting the exact number of times a customer will visit a website before making a purchase is considered a regression problem, even though the possible outcomes are limited to whole numbers like 0, 1, 2, 3, 4, 5, and so on, rather than any real value as in the other regression examples given above.

Unsupervised Learning

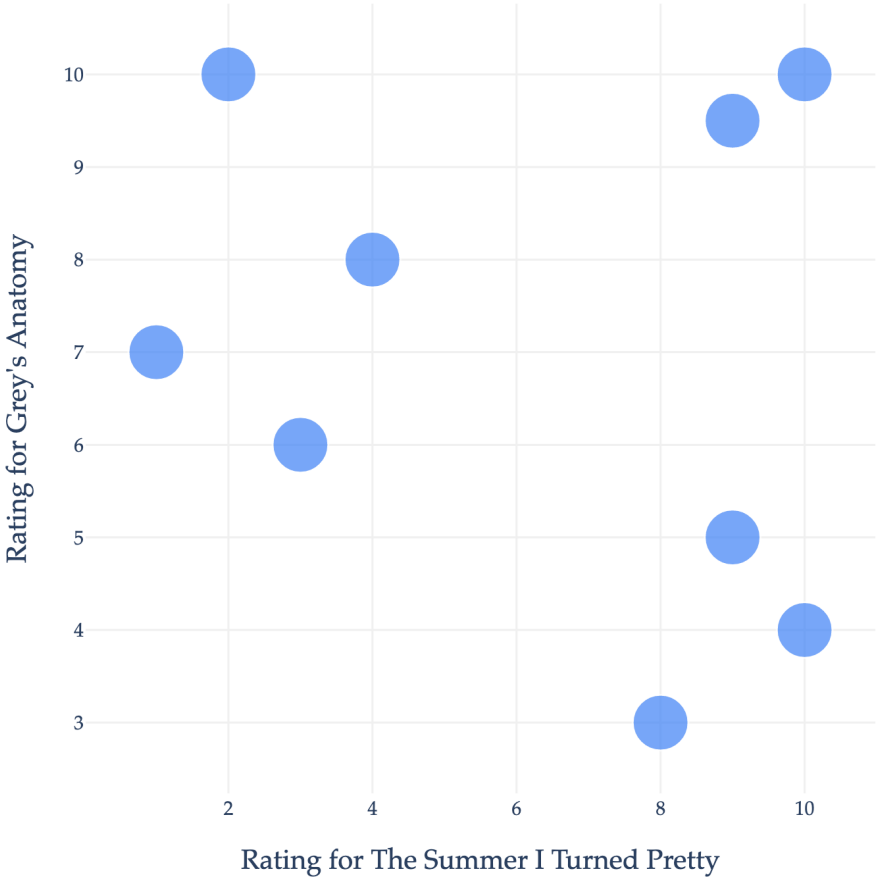
In supervised learning, the computer is "supervised" by the correct answers in its goal to find a function that makes good predictions. But, what if we don't have any answers, and aren't trying to *predict* anything?

Definition: Unsupervised Learning

An unsupervised learning problem can be thought of as "given data, **find structure** in the data".
To train a model for an unsupervised learning problem, we do not need (and will not use) any labels.

Our focus in this course will mostly be on supervised learning, but I'll briefly discuss two common forms of unsupervised learning: clustering and dimensionality reduction.

Clustering. Imagine you work at Netflix, and keep track of the ratings each user gives to each TV show. You might want to find groups of users who have similar tastes, so that you can recommend shows to them that they're likely to enjoy.



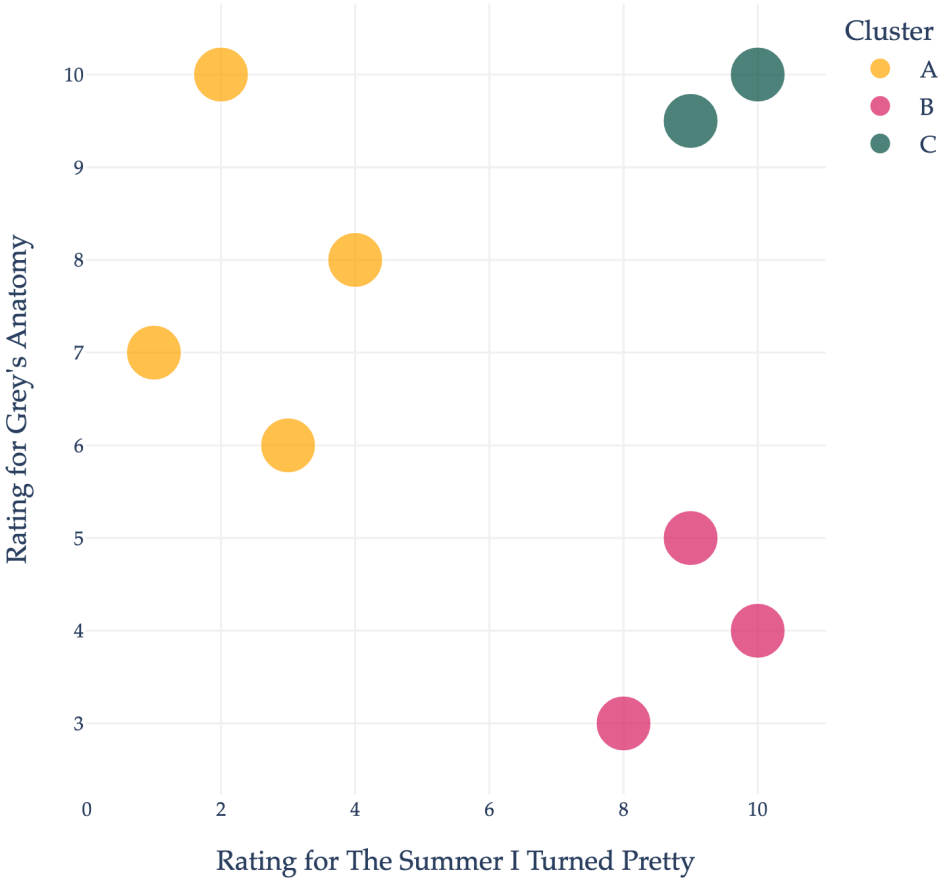
Neither of these shows are on Netflix, but they're what I spent the summer watching, so they'll have to work for now.

Each point in the plot above represents a single user. We *could* try and predict a user's rating for Grey's Anatomy (GA) given their rating for The Summer I Turned Pretty (TSITP), which would be a supervised learning (regression) problem, but that's not what we're investigating now.

Instead, let's treat the data as unlabeled. You might notice that the data naturally falls into three groups, or **clusters**:

- The top-left group doesn't like TSITP (a romantic drama) but does like GA (a medical drama).
- The bottom-right group likes TSITP, but not GA.
- The top-right group likes both TSITP and GA.

If I could mathematically describe these clusters, I could use their boundaries to recommend shows to other users. For instance, if most users in the bottom-right cluster like, say, The Office, and some new user who hasn't watched The Office but likes TSITP and GA comes along, I could recommend The Office to them.



Definition: Clustering

The goal of clustering, an unsupervised learning problem, is to place data into groups of similar data points.

Placing points into clusters seems easy in the example above, but in practice, Netflix has millions of users and thousands of shows, meaning that the data is too large to visualize, much less cluster by hand. There are a variety of clustering algorithms that exist, each with their own strengths and weaknesses. We will not discuss clustering in this course – you'll learn more in a future course dedicated to machine learning – but you can read more about how different clustering algorithms work here.

Activity 2

Activity 2

How would you explain the difference between clustering and classification to someone with no background in computer science or machine learning?

Dimensionality Reduction. As alluded to above, we'll soon work with datasets with features (pieces of information) for millions of individuals. We typically store this information in a table or matrix, where each row corresponds to an individual and each column corresponds to a feature.

To illustrate, let's return to the digit recognition example from the start of the section. I mentioned that each image in the dataset is a 28x28 pixel image, which means that to train the model, we're using $28 \cdot 28 = 784$ features per image. A table containing this information would have 70,000 rows (one per image) and 784 columns (one per pixel).

	pixel1	pixel2	pixel3	pixel4	pixel5	pixel6	pixel7	pixel8	pixel9	pixel10	...	pixel775	pixel776	pixel777	pixel778	pixel779	pixel780	pixel781	pixel782	pixel783	pixel784
0	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0
2	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0
3	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0
4	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
69995	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0
69996	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0
69997	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0
69998	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0
69999	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0

70000 rows × 784 columns

Activity 3

Activity 3

Why do you think most of the values above are 0?

This dataset is far too high-dimensional to visualize all at once – to draw a scatter plot as we did in earlier examples above, we'd need to be able to see 784 dimensions at once, while we can only see 3. We can look at a few images at a time, but we have no tool for visualizing the entire dataset.

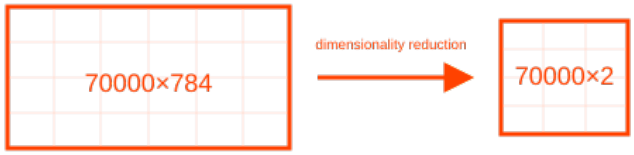
That's where dimensionality reduction comes in.

Definition: Dimensionality Reduction

Dimensionality reduction is an unsupervised learning technique for reducing the number of features in a dataset while preserving the information in the dataset.

A naïve approach to dimensionality reduction would be to simply drop some of the features. For instance, if we wanted to draw a scatter plot containing all 70,000 images, we could pick just 2 of the 784 features (pixels) to plot against each other. But, this would lose a lot of information, as just two pixels don't tell us much about an image.

More sophisticated approaches, like principal component analysis (PCA), can "invent" new features that capture the most important information in the dataset.



Below, you'll find the result of applying PCA to the MNIST dataset. **Each point corresponds to an image.** We've actually only included a random sample of 10,000 of the 70,000 images, as the full dataset would crowd the plot too much.

The PCA process **did not** use the labels in the dataset, i.e. it did not factor in whether the image was actually of a 0, 1, 2, 3, etc. when inventing these two new features. But, you'll notice that in the plot of new feature 2 vs. new feature 1, the images are roughly clustered by true digit! For example, the 0s (middle right) all look similar to one another in 2D space. The plot above is interactive, so you can hover over a point to see its true digit.

We will cover PCA in depth in Chapter 10 of these notes – it is surprisingly involved, and requires a deep understanding of linear algebra to fully grasp. All in due time!

Reinforcement Learning

Reinforcement learning is less about finding a function that makes good predictions or finding structure in data, and more about finding a function (formally called a **policy**) that **makes good decisions**. Reinforcement learning is likened to the way humans pick up new skills, like cooking or riding a bike – by trying something, failing, and trying again, now equipped with the knowledge of what worked and what didn't.

Perhaps the most famous example of reinforcement learning is AlphaGo, a model trained by Google DeepMind to play the game Go. Go is a board game similar to chess or checkers, but with a 19x19 grid and more sophisticated rules. To train AlphaGo, first, researchers showed the model 30 million games of Go played by human players. Then, they had the model play many games of Go **against itself**, giving it a **reward** for winning and a **penalty** for losing. Eventually, through trial-and-error, the model learned to play Go at a higher level than the human players. In 2016, AlphaGo defeated the reigning world champion at the time.

We won't discuss reinforcement learning any further in this course. If you'd like to learn more, start by reading this blog post. It was written by Demis Hassabis, the CEO of Google DeepMind – who also recently won the Nobel Prize in Chemistry – and explains the inner workings of AlphaGo in 2016 *before* it beat the world champion. I like this post not only because it's written in a way that's accessible to non-technical readers, but also because it shows you a snapshot of the growing excitement surrounding machine learning a decade ago.

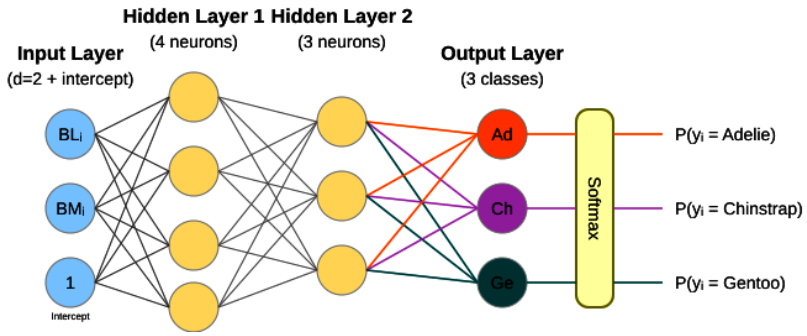
Deep Learning

"Deep learning" is a popular term that doesn't appear in the taxonomy I presented earlier, but it's a term you've probably heard of.

All three types of machine learning – supervised learning, unsupervised learning, and reinforcement learning – could involve "deep learning", which really just means machine learning using neural networks. Neural networks are called "deep" because they can have many "layers" of computation, i.e. many functions that are applied to the input data before returning a prediction, representation, or action.

Figure 1.4: *
Watch the AlphaGo documentary for free on YouTube.

For example, here's a neural network with several layers, that could be used to predict the species of a penguin given its bill length and body mass, by first predicting the probability of each species.



While we won't spend much time on neural networks in this course – you'll need to wait for your next machine learning course to see them in depth – we **will** spend the entire semester understanding all of the operations they use and how they work together. Every line, circle, and box you see in the diagram above represents some sort of operation in linear algebra.

Large Language Models

I've given you an overview of three big branches of machine learning: supervised learning, unsupervised learning, and reinforcement learning. What's not clear is how large language models, like ChatGPT, fit into this taxonomy. It turns out that large language models are created using **all three of these types of machine learning** (and also deep learning).

Definition: Language Model

A language model is a model that predicts the next word in a sequence of words.

A language model is trained on a collection of text documents, called a **corpus**. To illustrate how they work,

let's look at a small example corpus.

is the umich the best? my friend said ucsd **is the best**, **is the umich** better?

Think of predicting the next word in a sequence of words as implementing randomized, intelligent autocomplete, using patterns from data. For instance, if we start with the phrase "**is the**", a language model trained on this corpus might predict "**umich**" or "**best**" as the next word, since both "**is the umich**" and "**is the best**" appear in the corpus. It probably wouldn't predict "friend" as the next word, as "is the friend" doesn't appear in the corpus.

"**is the umich**" appears twice as often as "**is the best**". Language models estimate **probabilities**, so a language model trained on this specific corpus would learn:

$$\mathbb{P}(\text{“umich”} \mid \text{“is the”}) = \frac{2}{3} \mathbb{P}(\text{“best”} \mid \text{“is the”}) = \frac{1}{3}$$

In order for language models to produce natural-sounding text, researchers have found that they need to use some randomness in how they generate outputs. So, instead of always autocompleting "**is the**" with the most likely word (i.e., "**umich**"), they sample from a distribution of words, weighted by the probabilities learned above. So, $\frac{2}{3}$ of the time, the model will autocomplete "**is the**" with "**umich**", and $\frac{1}{3}$ of the time, it will autocomplete "**is the**" with "**best**". Then, it'll continue to predict the next word in the sequence, using the last few words as context. This explanation is a bit simplistic, but captures the gist of how language models work at a high level.

A **large** language model (LLM) is a language model that is trained on a **large** (massive!) dataset of text. The recently released GPT-5, and its predecessors, GPT-4 and GPT-3.5, are all LLMs created by OpenAI; ChatGPT is a chatbot that uses GPT for autocompletion, with some additional logic to make the interactions seem more natural. Claude, Gemini, and Grok are also LLMs, created by Anthropic, Google, and X (formerly Twitter), respectively.

The corpuses for these LLMs are **trillions** of words long, containing books, most publicly available websites, and more; since they were trained on such massive datasets, they are able to answer questions about a broad range of topics. With the above explanation in mind, it's not hard to see why LLMs sometimes output realistic-sounding text that is not actually true – they randomly generate sequences of words that are statistically likely to appear in their corpuses, but not necessarily true.

All three of the branches of machine learning – supervised learning, unsupervised learning, and reinforcement learning – play a role in training LLMs.

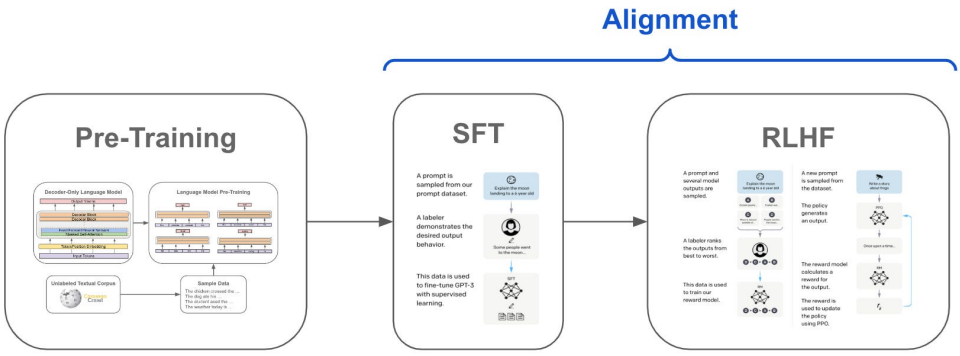


Figure 1.5: *
The three stages of LLM training (image source; more reading from OpenAI).

1. First, an LLM is taught how to predict the next word in a sequence of words, by estimating probabilities like $\mathbb{P}(\text{“umich”} \mid \text{“is the”})$. In this task, there are no labels – the "labels" come from the corpus of text

itself. So, this first stage – called **pre-training** – is somewhere in between supervised learning (because it's predicting) and unsupervised learning (because there are no labels), and is often thought of as **self-supervised learning**.

2. Then, more supervised learning – specifically, **supervised fine-tuning (SFT)** – is used to train the LLM to get better at answering questions for any one particular task, using a dataset of questions x and ideal answers y created by humans.
3. Finally, to ensure the LLM gives answers that humans find useful, it is repeatedly asked a question, and humans vote on their favorite answers. These human preferences are used to create a **reward function** that the LLM can use to improve its answers using reinforcement learning (sometimes called **reinforcement learning with human feedback (RLHF)**).

Neural networks – and more specifically, the transformer architecture – are the computational backbone that makes all these steps possible. Transformers, first introduced in 2017 by researchers at Google in a now-famous paper, are excellent at extracting meaning from text data out-of-order. For instance, in the sentence "The bank was steep," a transformer can distinguish that "bank" refers to a riverbank rather than a financial institution by considering the context provided by "steep", even though the word "bank" appears before "steep" in the sentence. For a detailed technical introduction to how transformers work, see the transformers course by Hugging Face.

Training state-of-the-art LLMs is a resource-intensive process – some insiders estimate that training OpenAI's new GPT-5 cost a few **billion** dollars and several months. You and I (probably) don't have billions of dollars, but we do have a few months together, and in that time, we will learn about the mathematical underpinnings that make this all possible.

1.2. Squared Loss and the Constant Model

Motivation

Suppose that you're looking for an off campus apartment for next year. Unfortunately, none of them are in your price range, so you decide to live with your parents in Detroit and commute. To see if you can save some time on the road each day, you keep track of how long it takes for you to get to school.

	date	day	departure_hour	minutes
0	5/15/2023	Mon	10.816667	68.0
1	5/16/2023	Tue	7.750000	94.0
2	5/22/2023	Mon	8.450000	63.0
3	5/23/2023	Tue	7.133333	100.0
4	5/30/2023	Tue	9.150000	69.0

This is a real dataset, collected by Joseph Hearn, except he lived in Seattle, not Metro Detroit. The full dataset contains more columns than are shown above, but we'll focus on these few for now.

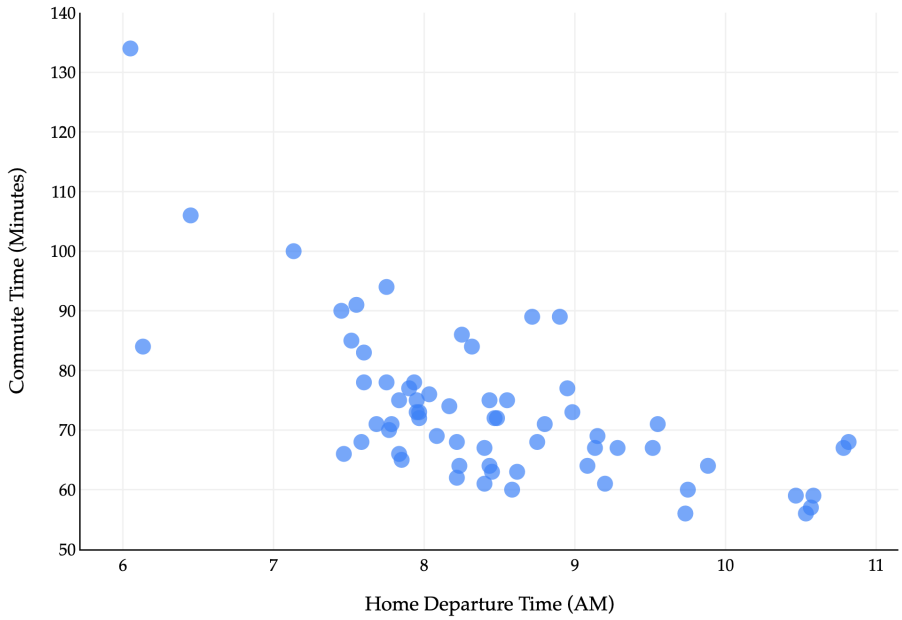
Our goal is to predict commute times, stored in the `minutes` column. This is our y variable. The natural first input variable, or **feature**, to consider, is `departure_hour`. This is our x variable.

We'll use the subscript i to index the ith data point, for $i = 1, 2, \ldots, n$. Using the dataset above, $x_1 = 10.816667$ and $y_1 = 68$, for instance.

Departure hours are stored as decimals, but correspond to times of the day. For example, 7.75 corresponds to 7:45 AM, and 10.816667 corresponds to 10:49 AM.

$$\begin{aligned} 10.816667 &= 10 + 0.816667 \text{ hours} \\ &= 10 \text{ hours} + 0.816667 \cdot 60 \text{ minutes} \\ &= 10 \text{ hours} + 49 \text{ minutes} \end{aligned}$$

Before we get any further, we should **look** at our data. Since we're working with two quantitative variables, we should draw a scatter plot.



There's a general downward trend: the later the departure time, the lower the commute time tends to be.

Again, our goal is to predict commute time given departure hour. That is, we'd like to build a useful function h such that:

$$\text{predicted commute time}_i = h(\text{departure hour}_i)$$

This is a **regression** problem, because the variable we are predicting – commute time – is quantitative.

To build this function, the approach we'll take is the machine learning approach – that is, to **learn a pattern from the dataset** that we've collected. (This is not the only approach one could take – we could build the function h however we want.)

However, in order to learn a pattern from the dataset that we've collected, we need to make an important assumption.

The Key Assumption in Machine Learning

In order for the patterns we learn from a dataset to be useful, **we must assume that future data will look like past data!**

We don't really need our function h to make good predictions on the dataset that we've already collected. We know the actual commute times on day 1, day 2, ..., day n. In other words, we're working with a **labeled** dataset, in which we're given the values of $y_1, y_2, \ldots, y_n$.

What we **do** need is for our function h to make good predictions on **unseen** data from the future, i.e. for future commutes, the ones we don't know about yet. This is the only way our function h will be practically useful.

But, if the future doesn't resemble the past, the patterns we learn from the past will not be **generalizable** to the future. For example, if a new highway between Detroit and Ann Arbor gets built, the patterns previously learned won't necessarily still exist. This idea of generalizability is key, so keep it in mind even if we're not explicitly talking about it.

Models

I've used the word "model" loosely, but let's give it a formal definition.

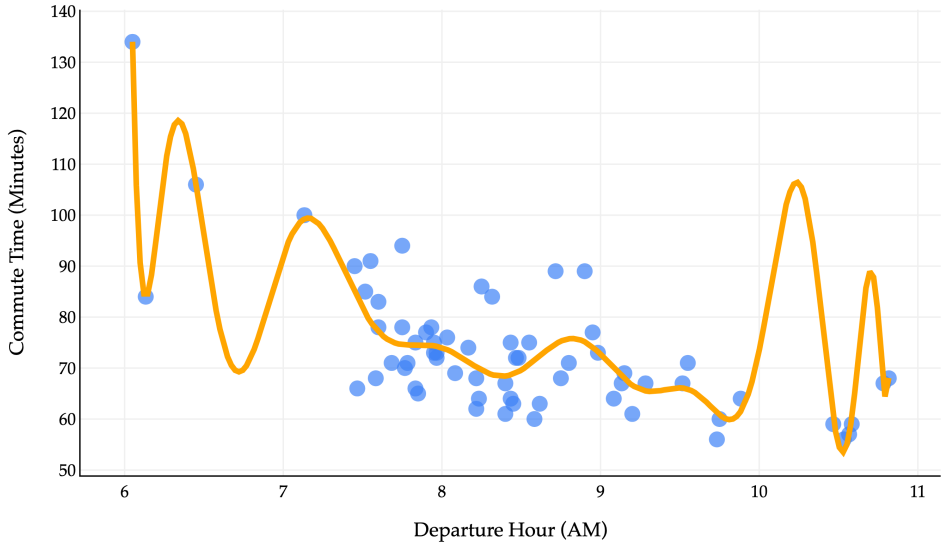
Definition: Model

A **model** is a set of assumptions about how data were generated.

> "All models are wrong, but some are useful." - George Box

My interpretation of George Box's famous quote is that no matter how complex a model is, it will never be 100% correct, so sometimes – especially when we're starting our machine learning journey – it's better to use a simpler model that is also wrong but **interpretable**.

We gain value from simple models all the time. In a physics class, you may have learned that acceleration due to gravity is $9.81\ \text{m/s}^2$ towards the center of the Earth. This is not fully accurate – think about how parachutes work, for example – but it's still a useful approximation, and one that is relatively easy to understand. A related idea is Occam's razor, which states that the simplest explanation of a phenomenon is often the best.



Above, you'll see a degree-40 polynomial fit to our dataset. We'll learn how to build such polynomials throughout the semester.

At first glance, it looks to be quite accurate, albeit complex. In fact, it's a little too complex, and the phenomenon we see above is called **overfitting**. For x_i's in the dataset that we collected, sure, the polynomial will make accurate predictions, but for any x_i's that don't match the exact pattern in the dataset, the predictions will be off. (For example, it's unlikely that commutes will take 110 minutes around 10:15AM, but that's what the model predicts.) This polynomial model wouldn't generalize well to unseen data.

If we look at the scatter plot closely, it seems reasonable to start with a line of best fit, much like you may have seen in a statistics class. In fact, we'll start with something even more simple than that. But first, some notation.

Hypothesis Functions

Definition: Hypothesis Function

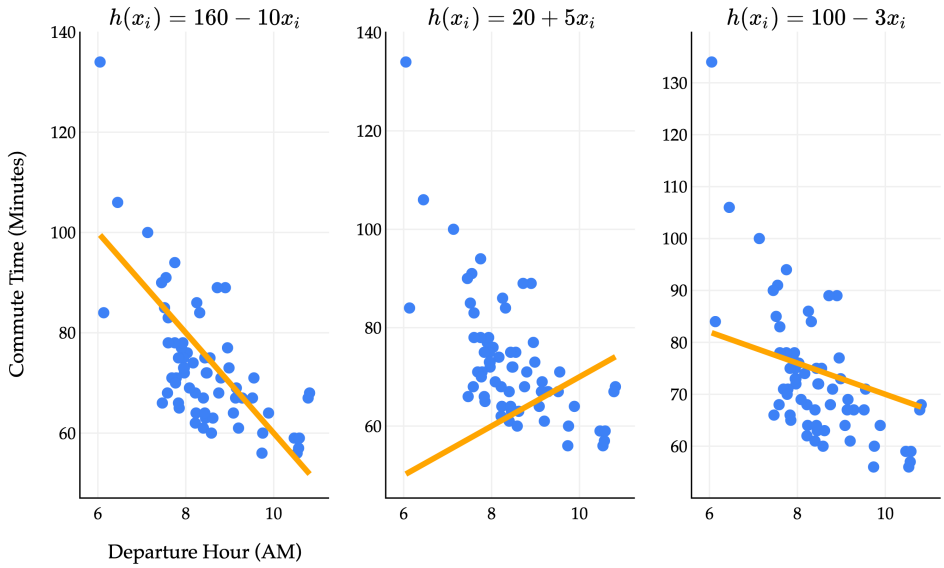
A **hypothesis function** h takes in an x_i as an input and returns a predicted y_i.

$$\text{predicted } y_i = h(x_i)$$

The hypothesis functions we'll study have **parameters**, usually denoted by w, which describe the relationship between the input and output. The two hypothesis functions we'll study to start with are:

1. **Constant model**: $h(x_i) = w$
2. **Simple linear regression model**: $h(x_i) = w_0 + w_1 x_i$

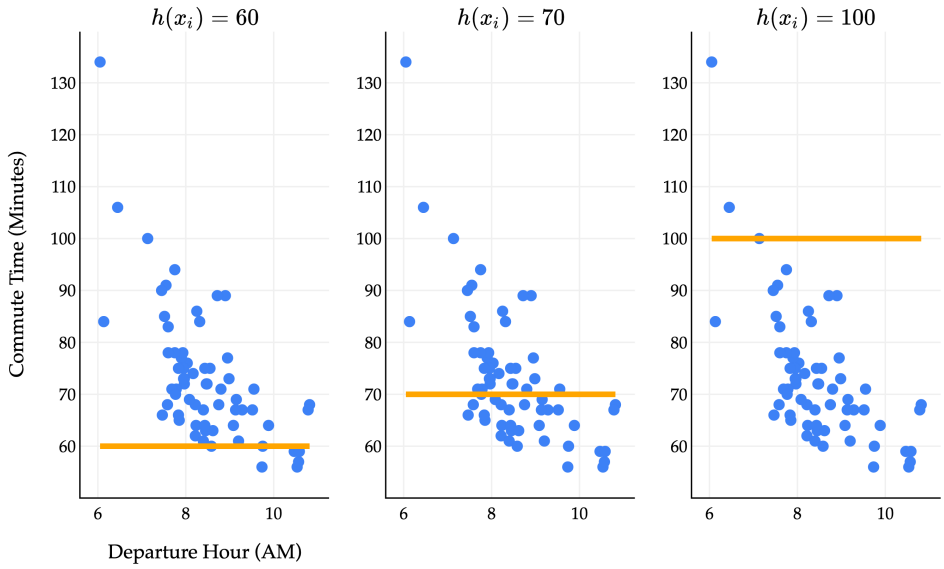
We'll study the constant model first, but it's easier to understand the role of parameters in the simple linear regression model. As we'll soon see, this linear regression model is called "simple" because it only takes in one input variable, x_i. "Multiple" linear regression models take in multiple features.



$h(x_i) = w_0 + w_1 x_i$ represents the equation of a line, where w_0 is the intercept and w_1 is the slope. Above, we see that different choices of parameters w_0 and w_1 result in different lines. The million dollar question is: **among all of the infinitely many choices of w_0 and w_1, which one is the best?**

To fully answer that, we'll have to wait until Chapter 2.2. Surprisingly, that answer involves multivariable calculus.

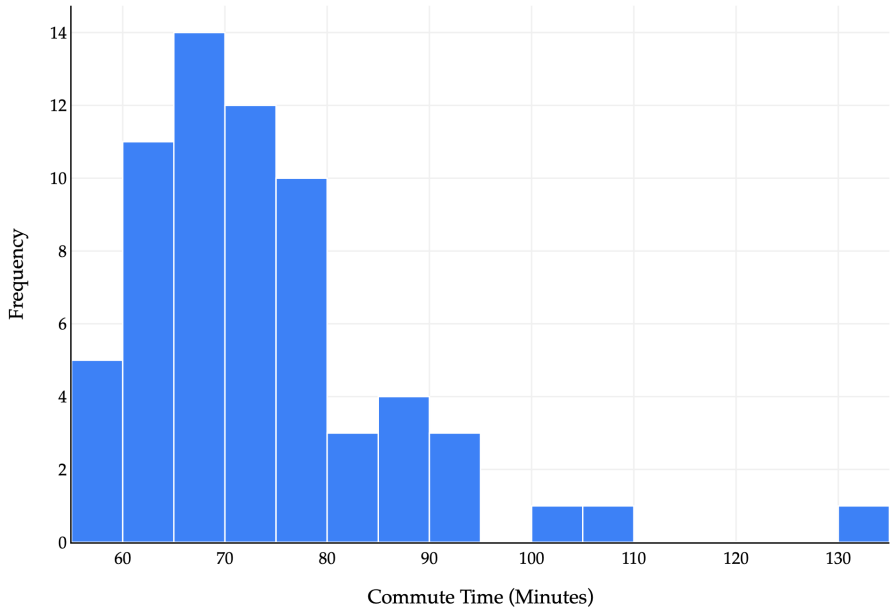
For now, let's return to the constant model, $h(x_i) = w$. The constant model predicts the same value for all x_i's, and looks like a flat line.



We'll use the constant model for the rest of this section to illustrate core ideas in machine learning, and will move to more sophisticated models in Chapter 2.1.

If we're forced to use a constant model, it's clear that some choices of w (the height of the line) are better than others. $w = 100$ yields a flat line that is far from most of the data. $w = 60$ and $w = 70$ seem like much more reasonable predictions, but how we can quantify which one is better, and which w would be the best?

Since the constant model doesn't depend on departure hours x_i, we can instead draw a histogram of just the true commute times.



An equivalent way of phrasing the problem is, which constant w best summarizes the histogram above? Most commute times seem to be in the 60 to 80 range, so somewhere there makes sense. How can we be more precise?

Loss Functions

To illustrate, let's consider a small dataset of only 5 commute times.

$$y_1 = 72, \quad y_2 = 90, \quad y_3 = 61, \quad y_4 = 85, \quad y_5 = 92$$

If asked to find the constant that best summarizes these 5 numbers, you might think of the mean or median, which are common **summary statistics**. There are other valid choices too, like the mode, or halfway between the minimum and maximum, or the most recent. What we need is a way to compare these choices.

A **loss function** quantifies how bad a prediction is for a single data point.

- If our prediction is **close** True to the actual value, we should have **low** loss.
- If our prediction is **far** False from the actual value, we should have **high** loss.

We'll start by computing the error for a single data point, defined as the difference between an actual y-value and its corresponding predicted y-value.

$$e_i = y_i - h(x_i)$$

where y_i is the actual value and $h(x_i)$ is the predicted value.

Could this be a loss function? Let's think this through. Suppose we have the true commute time $y_i = 80$.

- If I predict 75, $e_i = 80 - 75 = 5$.
- If I predict 72, $e_i = 80 - 72 = 8$.
- If I predict 100, $e_i = 80 - 100 = -20$.

A lower error is better, so 75 (error of 5) is a better prediction than 72 (error of 8). 100 seems to be the worst of the three predictions, but technically has the smallest error (-20). The issue is that some errors are positive and some are negative, and so it's hard to compare them directly.

So ideally, a loss function shouldn't have negative outputs. How can we take these errors, in which some are positive and some are negative, and enforce that they're all positive?

Squared Loss. The most common solution to the problem of negative errors is to square each error. This gives rise to the first loss function we'll explore, and arguably the most common loss function in machine learning: **squared loss**.

The squared loss function, L_{sq}, computes $(\text{actual} - \text{predicted})^2$. That is:

$$L_{\text{sq}}(y_i, h(x_i)) = (y_i - h(x_i))^2$$

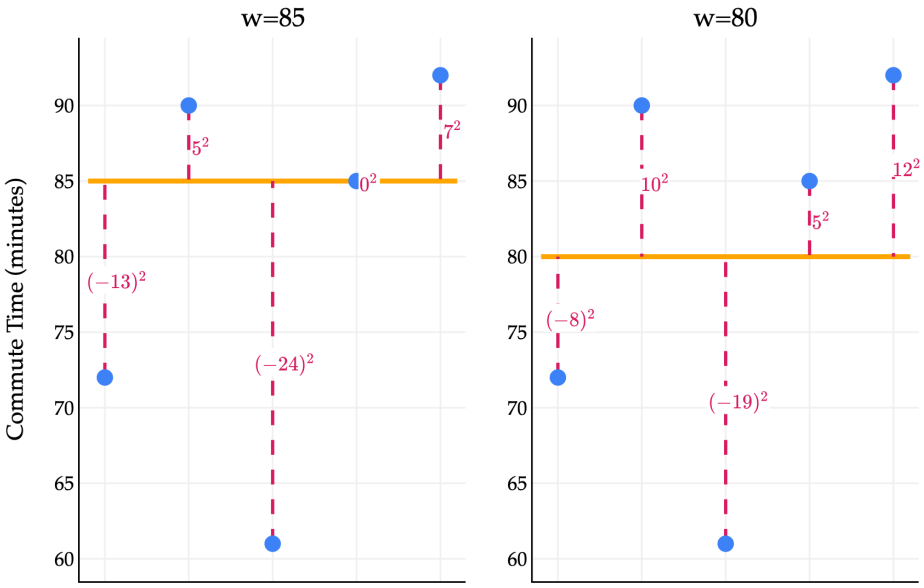
Why did we square instead of take the absolute value? Absolute loss is a perfectly valid loss function – in fact, we'll study it in Chapter 1.3 – and different loss functions have different pros and cons. That said, squared loss is a good first choice because:

- The resulting optimization problem is differentiable, as we'll see in just a few moments.
- It has a nice relationship to the normal distribution in statistics, as we'll see in Chapter 6, at the end of the course.

Let's return to our small example dataset of 5 commute times.

$$y_1 = 72, \quad y_2 = 90, \quad y_3 = 61, \quad y_4 = 85, \quad y_5 = 92$$

How can we use squared loss to compare two choices of w, say $w = 85$ (the median) and $w = 80$ (the mean)? Let's draw a picture (in which the x-axis positions of each point are irrelevant, since we're not using departure hours).



Each output of L_{sq}, shown in pink, describes the quality of a prediction for a single data point. For example, in the left plot above, the annotated $(-13)^2$ came from an actual value of 72 and a predicted value of 85:

$$L_{\text{sq}}(72, 85) = (72 - 85)^2 = (-13)^2 = 169$$

What we'd like is a single number which describes the quality of our predictions across the whole dataset, almost like a "score" for each choice of w. Then, we can compare scores to choose the best possible w. One way to construct such a score is to take the **average** of the squared losses.

- For the median, $w = 85$:

$$\frac{1}{5}\left((72 - 85)^2 + (90 - 85)^2 + (61 - 85)^2 + (85 - 85)^2 + (92 - 85)^2\right)$$
$$= 163.8$$

- For the mean, $w = 80$:

$$\frac{1}{5}\left((72 - 80)^2 + (90 - 80)^2 + (61 - 80)^2 + (85 - 80)^2 + (92 - 80)^2\right)$$
$$= 138.8$$

Losses are bad, so the better choice of w has a **lower** average squared loss. Since $138.8 < 163.8$, the mean is a better prediction than the median.

Another term for average squared loss is **mean squared error** (MSE); this is the more common name for the technique we just defined.

Minimizing Mean Squared Error

Let's start by generalizing mean squared error to any prediction w for our small commute times dataset.

$$R_{\text{sq}}(w) = \frac{1}{5}\left((72-w)^2 + (90-w)^2 + (61-w)^2 + (85-w)^2 + (92-w)^2\right)$$

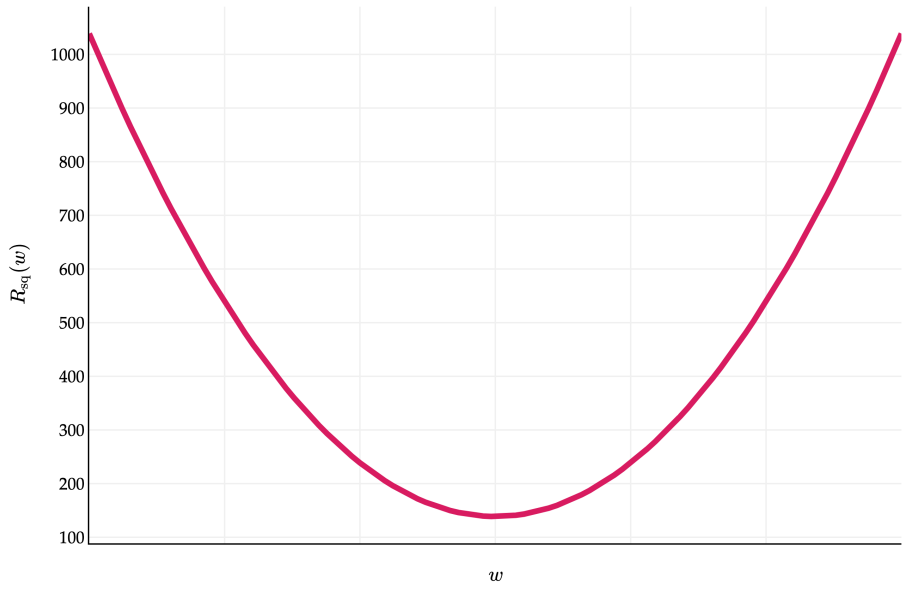
The function R_{sq} takes in any prediction w and outputs the mean squared error of that w. We're searching for the value of w that makes $R_{\text{sq}}(w)$ as small as possible, as that would correspond to the w that makes the best possible predictions, for our humble constant model.

Where did the letter R come from? It stands for risk, as in "empirical risk". I'll speak more on this soon. For now, remember that:

- L always refers to **loss** for a single data point.
- R always refers to **average loss** across an entire dataset.

What does $R_{\text{sq}}(w)$ actually look like, if we were to plot it? It is the sum of 5 quadratic functions – namely, $\frac{1}{5}(72-w)^2$, $\frac{1}{5}(90-w)^2$, and so on – and so it's a quadratic function too, and looks like a parabola.

$$R_{\text{sq}}(w) = \frac{1}{5}\left((72-w)^2 + (90-w)^2 + (61-w)^2 + (85-w)^2 + (92-w)^2\right)$$



The question is, though, **what is the w-value of the vertex of this parabola**? That is, which w minimizes $R_{\text{sq}}(w)$?

Before we find the answer, let's cast our problem in more general terms, so that the answer is applicable to any dataset. Suppose we have a dataset of n actual commute times, $y_1, y_2, \ldots, y_n$. Our goal is to find the w that minimizes:

$$\begin{aligned} R_{\text{sq}}(w) &= \frac{1}{n}\left((y_1 - w)^2 + (y_2 - w)^2 ... + (y_n - w)^2\right) \\ &= \frac{1}{n}\sum_{i=1}^{n}(y_i - w)^2 \end{aligned}$$

While it looks like there are many variables in this equation, we know the actual values in the dataset, so we can treat $y_1, y_2, \ldots, y_n$ as constants. The only true variable is w.

How do we minimize $R_{\text{sq}}(w)$? There are a few approaches. We'll use a calculus-based approach here, though in Homework 1 you'll look at an alternative approach. For a refresher on the relevant calculus ideas, see Appendix 2.

$R_{\text{sq}}(w)$ is a function of a single variable, w. To minimize a function of a single variable, we should:

1. Take the derivative of $R_{\text{sq}}(w)$ with respect to w.
2. Set the derivative equal to 0 and solve for w.
3. Verify that the second derivative at the critical point is positive.

Let's go through these steps one by one. The video below walks through these steps as well, but was recorded for a different class, so the notation is slightly different (it uses h instead of w).

Step 1: Take the derivative of $R_{\text{sq}}(w)$ with respect to w

$$R_{\text{sq}}(w) = \frac{1}{n}\sum_{i=1}^{n}(y_i - w)^2$$

$$\frac{\text{d}}{\text{d}w}R_{\text{sq}}(w) = \frac{\text{d}}{\text{d}w}\left(\frac{1}{n}\sum_{i=1}^{n}(y_i - w)^2\right)$$

Remember that constants can be pulled out of derivatives, e.g. the derivative of $2f(x)$ is 2 times the derivative of $f(x)$.

$$\frac{\text{d}}{\text{d}w}R_{\text{sq}}(w) = \frac{1}{n}\left(\frac{\text{d}}{\text{d}w}\sum_{i=1}^{n}(y_i - w)^2\right)$$

From here, we'll use the fact that the derivative of a sum is the sum of derivatives, to "push" the derivative operator inside the sum.

$$\frac{\text{d}}{\text{d}w}R_{\text{sq}}(w) = \frac{1}{n}\sum_{i=1}^{n}\frac{\text{d}}{\text{d}w}(y_i - w)^2$$

What is $\frac{\text{d}}{\text{d}w}(y_i - w)^2$? Try and work it out on your own, then check the solution below.

Solution

The chain rule and power rule are our friends here:

$$\begin{aligned}\frac{\mathrm{d}}{\mathrm{d}w}(y_i - w)^2 &= 2(y_i - w) \cdot \frac{\mathrm{d}}{\mathrm{d}w}(y_i - w) \\ &= 2(y_i - w) \cdot (-1) \\ &= -2(y_i - w)\end{aligned}$$

Using that result, we have:

$$\frac{\mathrm{d}}{\mathrm{d}w} R_{\text{sq}}(w) = \frac{1}{n}\sum_{i=1}^{n}(-2(y_i - w))$$

Finally, we'll pull the constant of -2 out of the sum.

$$\boxed{\frac{\mathrm{d}}{\mathrm{d}w} R_{\text{sq}}(w) = -\frac{2}{n}\sum_{i=1}^{n}(y_i - w)}$$

We *could* simplify this further, but this form will do just fine. Don't forget, though, that the expression on the right side is a function of w.

Step 2: Set the derivative equal to 0 and solve for w

$$-\frac{2}{n}\sum_{i=1}^{n}(y_i - w) = 0$$

First, we'll multiply both sides by $-\frac{n}{2}$ to get rid of the fraction.

$$\sum_{i=1}^{n}(y_i - w) = 0$$

Separating the sum into two parts gives us:

$$\sum_{i=1}^{n} y_i - \sum_{i=1}^{n} w = 0$$

$\sum_{i=1}^{n} y_i$ can't be broken down much further. But, $\sum_{i=1}^{n} w$ is the sum of n copies of w, i.e. $w + w + \ldots + w$. This is just nw!

$$\sum_{i=1}^{n} y_i - nw = 0$$

Adding nw to both sides, then dividing both sides by n, gives us:

$$\boxed{w^* = \frac{1}{n}\sum_{i=1}^{n} y_i}$$

The value of w that minimizes $R_{\text{sq}}(w)$ is $w^* = \frac{1}{n}\sum_{i=1}^{n} y_i$. Notice that I've called it w^*; think of "star" as meaning "best" or "optimal".

The formula for w^* should look very familar. It's the mean of $y_1, y_2, \ldots, y_n$!

Step 3: Verify that the second derivative at the critical point is positive

We already know that $R_{\text{sq}}(w)$ is a parabola, which means that its only critical point is a global minimum. But, we'll be thorough just to set a good example.

Here, we'll need to find the second derivative of $R_{\text{sq}}(w)$ with respect to w.

$$\begin{aligned}
\frac{\text{d}^2}{\text{d}w^2} R_{\text{sq}}(w) &= \frac{\text{d}}{\text{d}w}\left(-\frac{2}{n}\sum_{i=1}^{n}(y_i - w)\right) \\
&= -\frac{2}{n}\sum_{i=1}^{n}\frac{\text{d}}{\text{d}w}(y_i - w) \\
&= -\frac{2}{n}\sum_{i=1}^{n}(-1) \\
&= -\frac{2}{n}(-n) \\
&= 2
\end{aligned}$$

The second derivative is 2 for all values of w, including at the w^* we found. This tells us that $R_{\text{sq}}(w)$ is concave opening upwards across its entire domain, so the critical point we've found corresponds to a global minimum.

Conclusion

What was the point of all of that algebra? To recap:

- We decided to use the constant model, $h(x_i) = w$, to make predictions.
- To find the best value of w – a model parameter · we decided to minimize mean squared error:

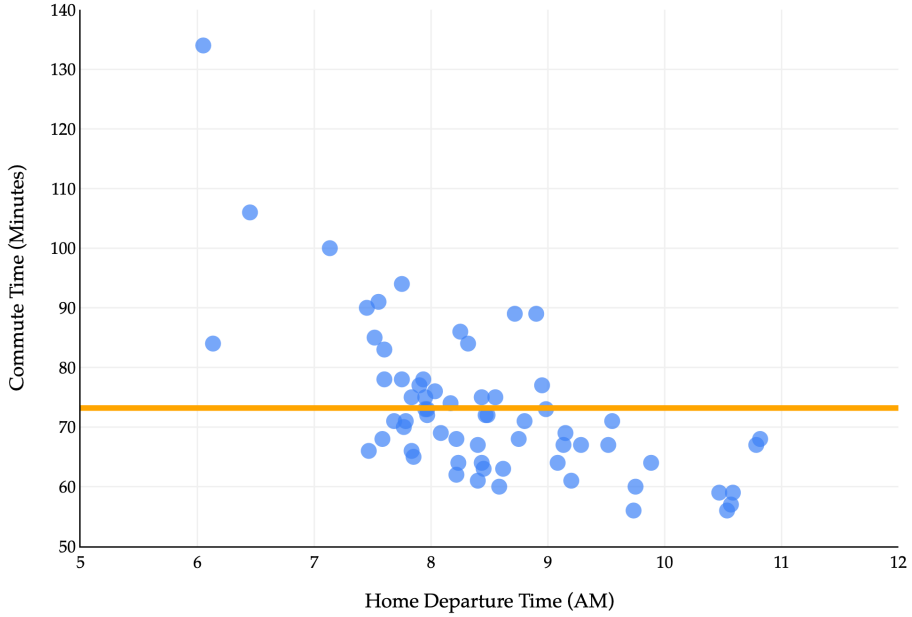
$$R_{\text{sq}}(w) = \frac{1}{n}\sum_{i=1}^{n}(y_i - w)^2$$

- Using calculus, we found that the value of w that minimizes $R_{\text{sq}}(w)$ is

$$w^* = \frac{1}{n}\sum_{i=1}^{n} y_i = \text{Mean}(y_1, y_2, \ldots, y_n)$$

In other words, **the mean minimizes mean squared error**. This is a remarkable result. We use the mean all of the time in daily life, and now we've proven that it is **optimal** in some sense. It is the constant with the smallest mean squared error, no matter the dataset we're working with.

Another name for w^* is an **optimal model parameter**. In the context of our full commute times dataset, the optimal model parameter is the mean commute time. Visually, the value of $w^* \approx 73$ tells us the optimal "height" at which we should draw the constant model, $h(x_i) = w$.



Is this the best possible model? No, of course not – we're not capturing the fact that later departure times are associated with shorter commute times. But as a first attempt at building a model, the constant model is valuable. If someone asked you how long your commutes are, saying something like "about 73 minutes" is reasonable.

What's next?

- In Chapter 1.3, we'll investigate other loss functions, like absolute loss.
- In Chapter 2.1, we'll reintroduce the simple linear regression model, $h(x_i) = w_0 + w_1 x_i$, and see how to find the best values of w_0 and w_1.

1.3. Absolute Loss

The Modeling Recipe

In Chapter 1.2, we implicitly introduced a three-step process for building a machine learning model.

The Modeling Recipe

1. **Choose a model.**

   We chose the constant model, whose predictions lie on a flat line, and don't factor in any features (like commute time).

   $$h(x_i) = w$$

   Here, w is the model parameter.

2. **Choose a loss function.**

   We chose squared loss:

   $$L_{\text{sq}}(y_i, h(x_i)) = (y_i - h(x_i))^2$$

   This is the loss for a single data point.

   For the constant model, since $h(x_i) = w$, we have:

   $$L_{\text{sq}}(y_i, w) = (y_i - w)^2$$

3. **Minimize average loss to find optimal model parameters.**

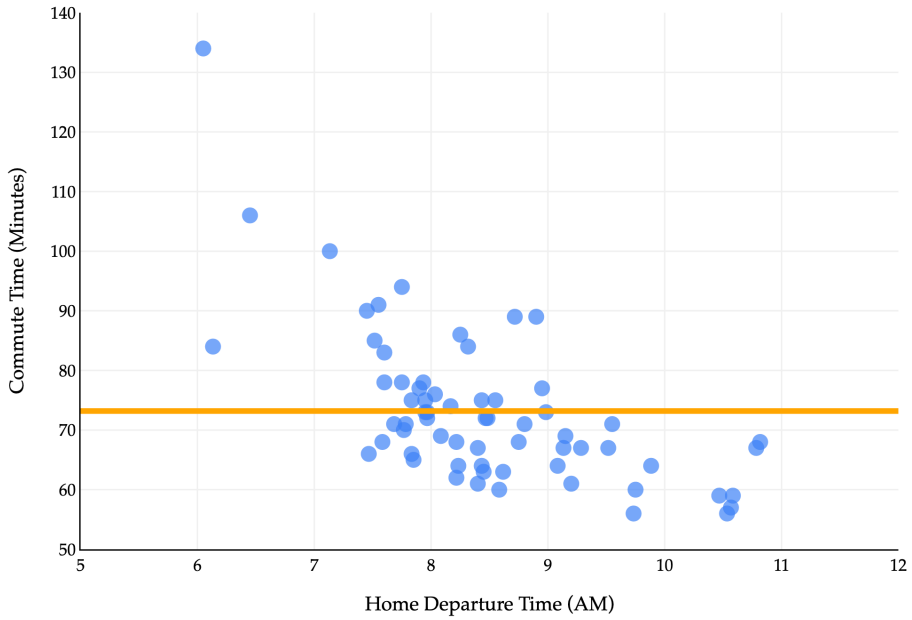
   The average loss – also known as mean squared error here – is:

   $$R_{\text{sq}}(w) = \frac{1}{n}\sum_{i=1}^{n}(y_i - w)^2$$

   Using calculus, we found that the value of w that minimizes $R_{\text{sq}}(w)$ is the mean of $y_1, y_2, \ldots, y_n$:

   $$w^* = \bar{y} = \frac{1}{n}\sum_{i=1}^{n} y_i$$

   This gave us the location of the optimal "flat line" to use in making predictions for y_i given x_i.



Most modern supervised learning algorithms follow these same three steps, just with different models, loss functions, and techniques for optimization.

Another name given to this process is **empirical risk minimization**.

Definition: Empirical Risk

Empirical risk is another term for "average loss across an entire dataset", and empirical risk minimization is the process of finding optimal model parameters by minimizing empirical risk.

When using squared loss, all three of these mean the same thing:

- Average squared loss.
- Mean squared error.
- Empirical risk.

Risk is an idea from theoretical statistics that we'll visit in a later chapter on probability. It refers to the expected error of a model, when considering the probability distribution of the data. "Empirical" risk refers to risk calculated using an actual, concrete dataset, rather than a theoretical distribution. The reason we call the average loss R is precisely because it is empirical **r**isk.

The first half of the course – and in some ways, the entire course – is focused on empirical risk minimization, and so we will make many passes through the three-step modeling recipe ourselves, with differing models and loss functions.

A common question you'll see in labs, homeworks, and exams will involve finding the optimal model parameters for a given model and loss function – in particular, for a combination of model and loss function that you've never seen before. For practice with this sort of exercise, work through the following activity. If you feel stuck, try reading through the rest of this section for context, then come back.

Activity 1

Activity 1

Suppose we'd like to find the optimal parameter, w^*, for the constant model $h(x_i) = w$. To do so, we use the following loss function:

$$L(y_i, h(x_i)) = (4y_i - 3h(x_i))^2$$

What value of w minimizes average loss for this new loss function?

Solution

Since we're using the constant model, $h(x_i) = w$, the loss function simplifies to:

$$L(y_i, w) = (4y_i - 3w)^2$$

Then, **average loss** is:

$$R(w) = \frac{1}{n}\sum_{i=1}^{n}(4y_i - 3w)^2$$

To find the w that minimizes $R(w)$, we take the derivative of $R(w)$ with respect to w and set it equal to 0.

$$\begin{aligned}\frac{\mathrm{d}R}{\mathrm{d}w} &= \frac{1}{n}\sum_{i=1}^{n}\frac{\mathrm{d}}{\mathrm{d}w}(4y_i - 3w)^2 \\ &= \frac{1}{n}\sum_{i=1}^{n}2(4y_i - 3w)(-3) \\ &= \frac{-6}{n}\sum_{i=1}^{n}(4y_i - 3w)\end{aligned}$$

Setting the derivative equal to 0 and solving for w gives

$$\begin{aligned}\frac{-6}{n}\sum_{i=1}^{n}(4y_i - 3w) &= 0 \\ \sum_{i=1}^{n}(4y_i - 3w) &= 0 \\ 4\sum_{i=1}^{n}y_i - 3\sum_{i=1}^{n}w &= 0 \\ 4\sum_{i=1}^{n}y_i &= 3nw \\ w &= \frac{4\sum_{i=1}^{n}y_i}{3n} = \frac{4}{3}\bar{y}\end{aligned}$$

So, the optimal constant prediction is $w^* = \frac{4}{3}\bar{y}$. Notice that this is the mean multiplied by $\frac{4}{3}$. Is there an easy way we could have arrived at this answer without having to take the derivative?
Here's a "shortcut": let's do a substitution. Let $z_i = 4y_i$ and $t = 3w$. Then, average loss looks like:

$$R(t) = \frac{1}{n}\sum_{i=1}^{n}(z_i - t)^2$$

This just looks like mean squared error for the constant model, which means that

$$t* = \bar{z}$$

But since each $z_i = 4y_i$, we have $\bar{z} = 4\bar{y}$, and so $t^* = 4\bar{y}$. Since $t = 3w$, we have

$$3w^* = 4\bar{y} \implies w^* = \frac{4}{3}\bar{y}$$

Absolute Loss

When we first introduced the idea of a loss function, we first started by computing the error, e_i, of each prediction:

$$e_i = y_i - h(x_i)$$

where y_i is the actual value and $h(x_i)$ is the predicted value.

The issue was that some errors were positive and some were negative, and so it was hard to compare them directly. We wanted the value of the loss function to be large for bad predictions and small for good predictions.

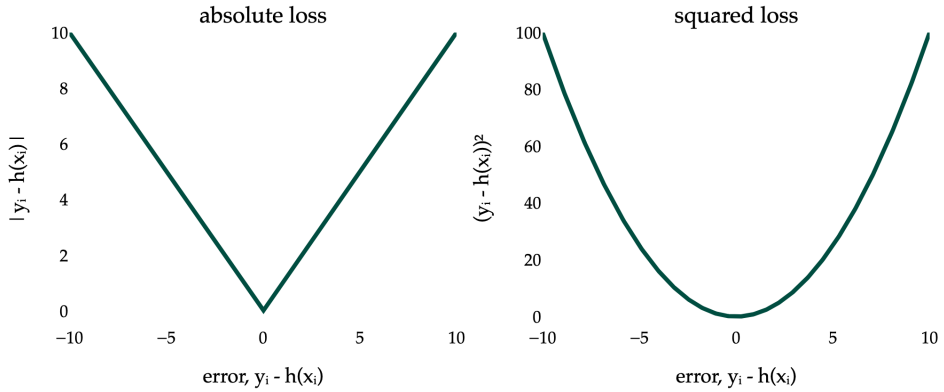
To get around this, we squared the errors, which gave us squared loss:

$$L_{\text{sq}}(y_i, h(x_i)) = (y_i - h(x_i))^2$$

But, instead, we could have taken the absolute value of the errors. Doing so gives us **absolute loss**:

$$L_{\text{abs}}(y_i, h(x_i)) = |y_i - h(x_i)|$$

Below, I've visualized the absolute loss and squared loss for **just a single data point**.



You should notice two key differences between the two loss functions:

1. The absolute loss function **is not differentiable** when $y_i = h(x_i)$. The absolute value function, $f(x) = |x|$, does not have a derivative at $x = 0$, because its slope to the left of $x = 0$ (-1) is different from its slope to the right of $x = 0$ (1). For more on this idea, see Appendix 2.
2. The squared loss function **grows much faster** than the absolute loss function, as the prediction $h(x_i)$ gets further away from the actual value y_i.

We know the optimal constant prediction, w^*, when using squared loss, is the mean. **What is the optimal constant prediction when using absolute loss?** The answer is not still the mean; rather, the answer reflects some of these differences between squared loss and absolute loss.

Let's find that new optimal constant prediction, w^*, by revisiting the three-step modeling recipe.

1. **Choose a model.**

   We'll stick with the constant model, $h(x_i) = w$.

2. **Choose a loss function.**

   We'll use absolute loss:

$$L_{\text{abs}}(y_i, h(x_i)) = |y_i - h(x_i)|$$

   For the constant model, since $h(x_i) = w$, we have:

$$L_{\text{abs}}(y_i, w) = |y_i - w|$$

3. **Minimize average loss to find optimal model parameters.**

   The average loss – also known as **mean absolute error** here – is:

$$R_{\text{abs}}(w) = \frac{1}{n}\sum_{i=1}^{n}|y_i - w|$$

In Chapter 1.2, we minimized $R_{\text{sq}}(w) = \frac{1}{n}\sum_{i=1}^{n}(y_i - w)^2$ by taking the derivative of $R_{\text{sq}}(w)$ with respect to w and setting it equal to 0. That will be more challenging in the case of $R_{\text{abs}}(w)$, because the absolute value function is not differentiable when its input is 0, as we just discussed.

Mean Absolute Error for the Constant Model

We need to minimize the mean absolute error, $R_{\text{abs}}(w)$, for the constant model, $h(x_i) = w$, but we have to address the fact that $R_{\text{abs}}(w)$ is not differentiable across its entire domain.

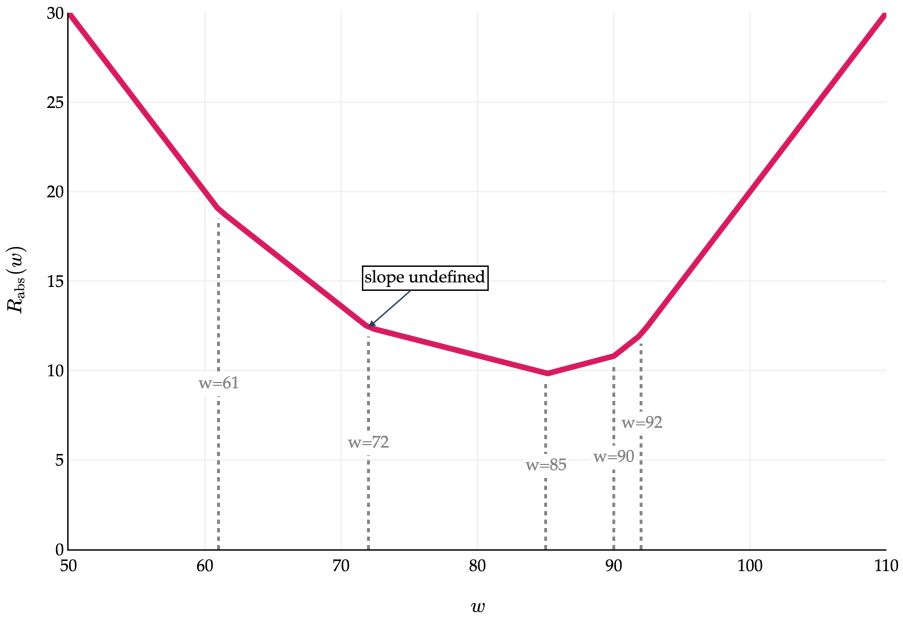
$$R_{\text{abs}}(w) = \frac{1}{n}\sum_{i=1}^{n}|y_i - w|$$

Graphing Mean Absolute Error. I think it'll help to visualize what $R_{\text{abs}}(w)$ looks like. To do so, let's reintroduce the small dataset of 5 values we used in Chapter 1.2.

$$y_1 = 72, \quad y_2 = 90, \quad y_3 = 61, \quad y_4 = 85, \quad y_5 = 92$$

Then, $R_{\text{abs}}(w)$ is:

$$R_{\text{abs}}(w) = \frac{1}{5}(|72 - w| + |90 - w| + |61 - w| + |85 - w| + |92 - w|)$$



This is a piecewise linear function. Where are the "bends" in the graph? Precisely where the data points, $y_1, y_2, \ldots, y_5$, are! Its at exactly these points where $R_{\text{abs}}(w)$ is not differentiable. At each of those points, the slope of the line segment approaching from the left is different from the slope of the line segment approaching from the right, and for a function to be differentiable at a point, the slope of the tangent line must be the same when approaching from the left and the right.

The graph of $R_{\text{abs}}(w)$ above, while not differentiable at any of the data points, still shows us something about the optimal constant prediction. If there is a bend at each data point, and at each bend the slope increases – that is, becomes more positive – then the optimal constant prediction seems to be in the middle, when the slope goes from negative to positive. I'll make this more precise in a moment.

For now, you might notice the value of w that minimizes the graph of $R_{\text{abs}}(w)$ above is a familiar summary statistic, but not the mean. I won't spell it out just yet, since I'd like for you to reason about it yourself.

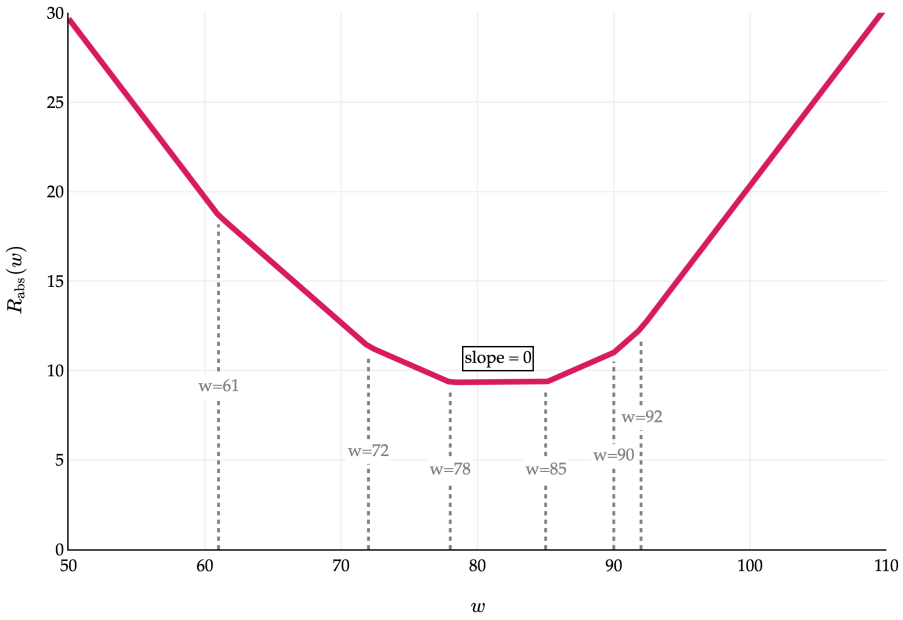
Let me show you one more graph of $R_{\text{abs}}(w)$, but this time, in a case where there are an even number of data points. Suppose we have a sixth point, $y_6 = 78$.

$$y_1 = 72, \quad y_2 = 90, \quad y_3 = 61, \quad y_4 = 85, \quad y_5 = 92, \quad y_6 = 78$$

Then, $R_{\text{abs}}(w)$ is:

$$R_{\text{abs}}(w) = \frac{1}{6}(|72 - w| + |90 - w| + |61 - w| + |85 - w| + |92 - w| + |78 - w|)$$

And its graph is:



This graph is broken into 7 segments, with 6 bends (one per data point). Between the 3rd and 4th bends – that is, the 3rd and 4th data points – the slope is 0, and **all values in that interval** minimize $R_{\text{abs}}(w)$. So, it seems that the value of w^* doesn't have to be unique!

Minimizing Mean Absolute Error. From the two graphs above, you may have a clear picture of what the optimal constant prediction, w^*, is. But, to avoid relying too heavily on visual intuition and just a single set of example data points, let's try and minimize $R_{\text{abs}}(w)$ mathematically, for an arbitrary set of data points.

To be clear, the goal is to minimize:

$$R_{\text{abs}}(w) = \frac{1}{n}\sum_{i=1}^{n}|y_i - w|$$

To do so, we'll take the derivative of $R_{\text{abs}}(w)$ with respect to w and set it equal to 0.

$$\frac{\mathrm{d}}{\mathrm{d}w}R_{\text{abs}}(w) = \frac{\mathrm{d}}{\mathrm{d}w}\left(\frac{1}{n}\sum_{i=1}^{n}|y_i - w|\right)$$

Using the familiar facts that the derivative of a sum is the sum of the derivatives, **and** that constants can be pulled out of the derivative, we have:

$$\frac{\mathrm{d}}{\mathrm{d}w}R_{\text{abs}}(w) = \frac{1}{n}\sum_{i=1}^{n}\frac{\mathrm{d}}{\mathrm{d}w}|y_i - w|$$

Here's where the challenge comes in. What is $\frac{\mathrm{d}}{\mathrm{d}w}|y_i - w|$?

Let's start by remembering the derivative of the absolute value function. The absolute value function itself can be thought of as a piecewise function:

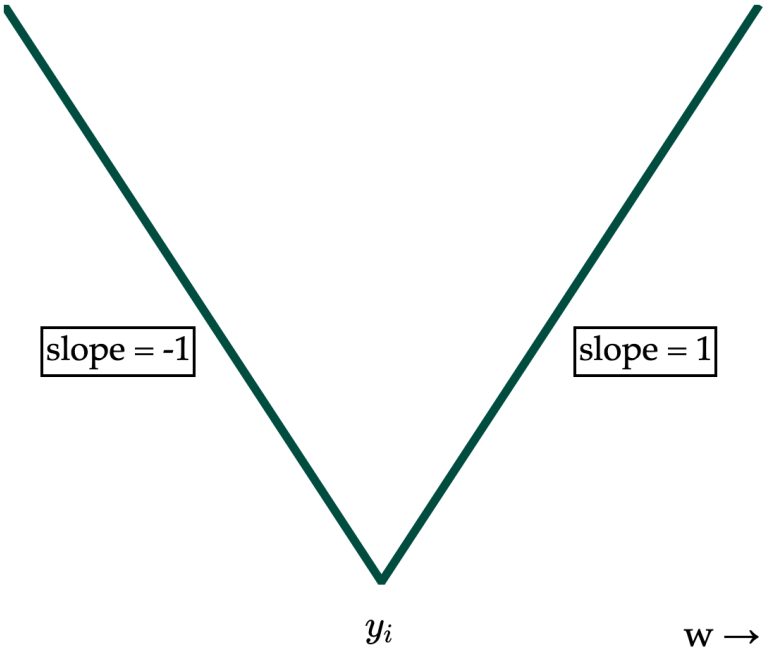
$$|x| = \begin{cases} x & x \geq 0 \\ -x & x < 0 \end{cases}$$

Note that the $x = 0$ case can either lumped in either the x or $-x$ case, since 0 and -0 are both 0.

Using this logic, I'll write $|y_i - w|$ as a piecewise of w:

$$|y_i - w| = \begin{cases} y_i - w & w \leq y_i \\ w - y_i & w > y_i \end{cases}$$

I have written the two conditions with w on the left, since it's easier to think in terms of w in my mind, but this means that the inequalities are flipped relative to how I presented them in the definition of $|x|$. Remember, $|y_i - w|$ is a function of w; we're treating y_i as some constant. If it helps, replace every instance of y_i with a concrete number, like 5, then reason through the resulting graph.



Now we can take the derivative of each piece:

$$\frac{\mathrm{d}}{\mathrm{d}w}|y_i - w| = \begin{cases} -1 & w < y_i \\ \text{undefined} & w = y_i \\ 1 & w > y_i \end{cases}$$

Great. Remember, this is the derivative of the absolute loss for a single data point. But our main objective is to find the derivative of the average absolute loss, $R_{\text{abs}}(w)$. Using this piecewise definition of $\frac{\mathrm{d}}{\mathrm{d}w}|y_i - w|$, we have:

$$\begin{aligned}\frac{\mathrm{d}}{\mathrm{d}w}R_{\text{abs}}(w) &= \frac{1}{n}\sum_{i=1}^{n}\frac{\mathrm{d}}{\mathrm{d}w}|y_i - w| \\ &= \frac{1}{n}\sum_{i=1}^{n}\begin{cases}-1 & w < y_i \\ \text{undefined} & w = y_i \\ 1 & w > y_i\end{cases}\end{aligned}$$

At any point where $w = y_i$, for any value of i, $\frac{\mathrm{d}}{\mathrm{d}w}R_{\text{abs}}(w)$ is undefined. (This makes any point where $w = y_i$ a critical point.) Let's exclude those values of w from our consideration. In all other cases, the sum in the expression above involves only two possible values: -1 and 1.

- The sum adds -1 for all data points greater than w, i.e. where $w < y_i$.
- The sum adds 1 for all data points less than w, i.e. where $w > y_i$.

Using some creative notation, I'll re-write $\frac{\mathrm{d}}{\mathrm{d}w}R_{\text{abs}}(w)$ as:

$$\frac{\mathrm{d}}{\mathrm{d}w}R_{\text{abs}}(w) = \frac{1}{n}\left(\sum_{w<y_i} -1 + \sum_{w>y_i} 1\right)$$

The sum $\sum_{w<y_i} -1$ is the sum of -1 for all data points greater than w, so perhaps a more intuitive way to write it is:

$$\sum_{w<y_i} -1 = \underbrace{(-1) + (-1) + \ldots + (-1)}_{\text{add once per data pointto the right of } w} = -(\#\text{ right of } w)$$

Equivalently, $\sum_{w>y_i} 1 = (\#\text{ left of } w)$, meaning that:

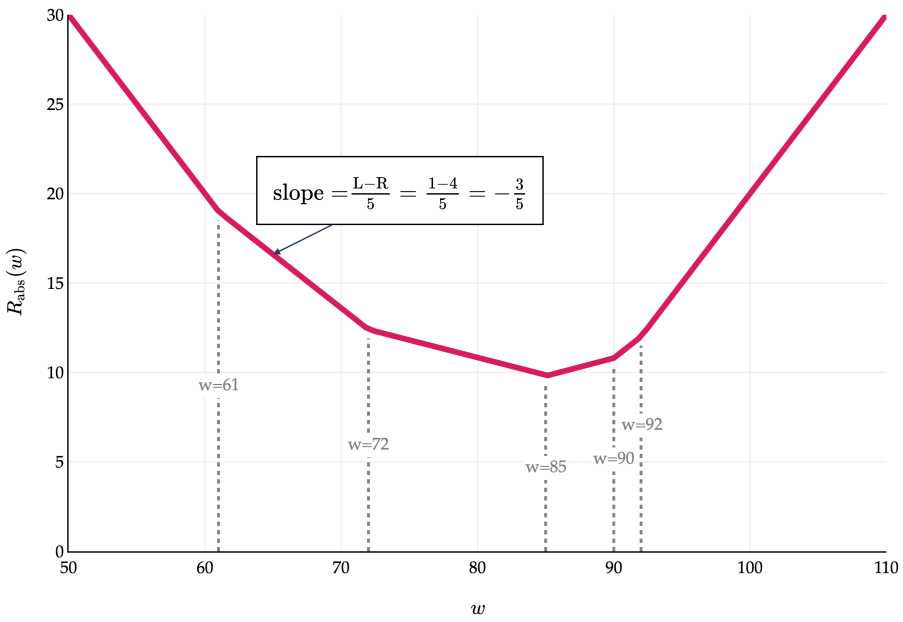
$$\begin{aligned}\frac{\mathrm{d}}{\mathrm{d}w}R_{\text{abs}}(w) &= \frac{1}{n}\left(-(\#\text{ right of } w) + (\#\text{ left of } w)\right) \\ &= \boxed{\frac{\#\text{ left of } w - \#\text{ right of } w}{n}}\end{aligned}$$

By "left of w", I mean less than w.

This boxed form gives us the slope of $R_{\text{abs}}(w)$, for any point w that is **not** an original data point. To put it in perspective, let's revisit the first graph we saw in this section, where we plotted $R_{\text{abs}}(w)$ for the dataset:

$$y_1 = 72, \quad y_2 = 90, \quad y_3 = 61, \quad y_4 = 85, \quad y_5 = 92$$

$$R_{\text{abs}}(w) = \frac{1}{5}(|72 - w| + |90 - w| + |61 - w| + |85 - w| + |92 - w|)$$



Now that we have a formula for $\frac{\text{d}}{\text{d}w} R_{\text{abs}}(w)$, the easy thing to claim is that we could set it to 0 and solve for w. Doing so would give us:

$$\frac{\text{\# left of } w - \text{\# right of } w}{n} = 0$$

Which yields the condition:

$$\text{\# left of } w = \text{\# right of } w$$

The optimal value of w is the one that satisfies this condition, and that's precisely the **median** of the data, as you may have noticed earlier.

This logic isn't fully rigorous, however, because the formula for $\frac{\text{d}}{\text{d}w} R_{\text{abs}}(w)$ is only valid for w's that aren't original data points, and if we have an odd number of data points, the median is indeed one of the original data points. In the graph above, there is never a point where the slope is 0.

To fully justify why the median minimizes mean absolute error even when there are an odd number of data points, I'll say that:

- If w is just to the left of the median, there are more points to the right of w than to the left of w, so (# left of w) < (# right of w) and $\frac{(\text{\# left of } w) - (\text{\# right of } w)}{n}$ is negative.
- If w is just to the right of the median, there are more points to the left of w than to the right of w, so (# left of w) > (# right of w) and $\frac{(\text{\# left of } w) - (\text{\# right of } w)}{n}$ is positive.

So even though the slope is undefined at the median, we know it is a point at which the sign of the derivative switches from negative to positive, and as we discussed in Appendix 2, this sign change implies at least a local minimum.

To summarize:

- If n is odd, the median minimizes mean absolute error.

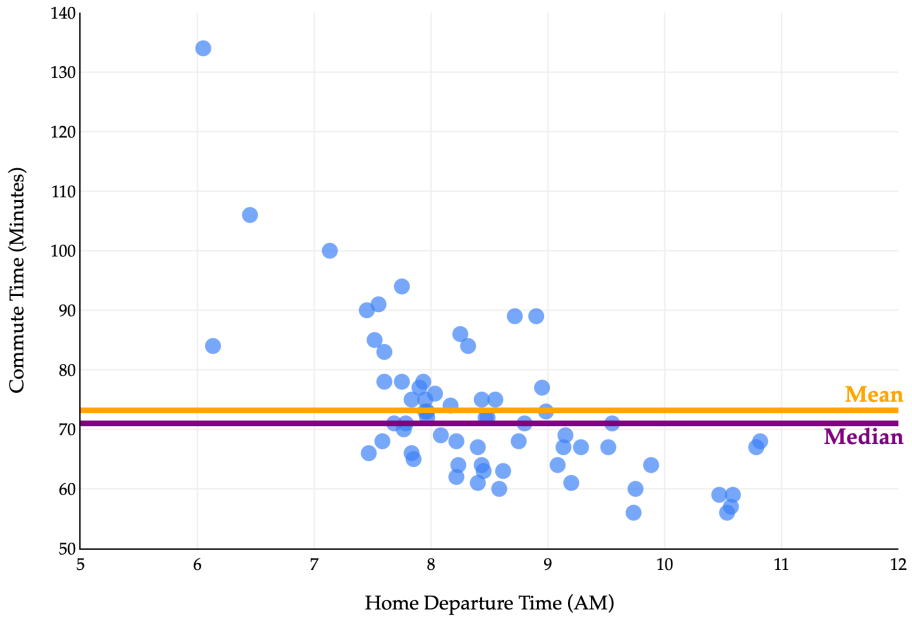
- If n is even, **any value** between the middle two values (when sorted) minimizes mean absolute error. (It's common to call the mean of the middle two values the median.)

We've just made a second pass through the three-step modeling recipe:

1. **Choose a model.**
   $h(x_i) = w$
2. **Choose a loss function.**
   $R_{\text{abs}}(w) = \frac{1}{n}\sum_{i=1}^{n} |y_i - w|$
3. **Minimize average loss to find optimal model parameters.**
   $R_{\text{abs}}(w) = \frac{1}{n}\sum_{i=1}^{n} |y_i - w| \implies w^* = \text{Median}(y_1, y_2, \ldots, y_n)$

Conclusion. What we've now discovered is that the **optimal model parameter** (in this case, the optimal constant prediction) **depends on the loss function we choose**!

In the context of the commute times dataset from Chapter 1.2, our two optimal constant predictions can be visualized as flat lines, as shown below.



Depending on your criteria for what makes a good or bad prediction (i.e., the loss function you choose), optimal model parameters may change.

Activity 2 (video walkthrough video!)

Activity 2 (video walkthrough video!)

Suppose we have a dataset of $n = 13$ numbers, such that:

$$0 < y_1 \leq y_2 \leq \ldots \leq y_{13}$$

Given that $y_8 - y_7 > 1$ and $y_9 - y_8 > 1$, how does the value of $R_{\text{abs}}(y_8 - 1)$ compare to the value of $R_{\text{abs}}(y_8 + 1)$?

Can you determine which is bigger, and by how much?

Next, we'll compare absolute loss to squared loss and see how different loss choices change the optimal constant model.

1.4. Comparing Loss Functions

We now know that:

- The mean is the constant prediction that minimizes mean squared error.

$$R_{\text{sq}}(w) = \frac{1}{n}\sum_{i=1}^{n}(y_i - w)^2 \implies w^* = \text{Mean}(y_1, y_2, \ldots, y_n)$$

- The median is the constant prediction that minimizes mean absolute error.

$$R_{\text{abs}}(w) = \frac{1}{n}\sum_{i=1}^{n}|y_i - w| \implies w^* = \text{Median}(y_1, y_2, \ldots, y_n)$$

Let's compare the behavior of the mean and median, and reason about how their differences in behavior are related to the differences in the loss functions used to derive them.

Outliers and Balance

Let's consider our example dataset of 5 commute times, with a mean of 85 and median of 80:

$$61 \qquad 72 \qquad 85 \qquad 90 \qquad 92$$

Suppose 200 is added to the largest commute time:

$$61 \qquad 72 \qquad 85 \qquad 90 \qquad 292$$

The median is still 85, but the mean is now $80 + \frac{200}{5} = 120$. This example illustrates the fact that the mean is sensitive to outliers, while the median is **robust** to outliers.

But why? I like to think of the mean and median as different "balance points" of a dataset, each satisfying a different "balance condition".

Summary Statistic	Minimizes	Balance Condition (comes from setting $\frac{\text{d}}{\text{d}w}R(w) = 0$)
Median	$R_{\text{abs}}(w) = \frac{1}{n}\sum_{i=1}^{n}\|y_i - w\|$	# left of w = # right of w
Mean	$R_{\text{sq}}(w) = \frac{1}{n}\sum_{i=1}^{n}(y_i - w)^2$	$\sum_{i=1}^{n}(y_i - w) = 0$

In both cases, the "balance condition" comes from setting the derivative of empirical risk, $\frac{\text{d}}{\text{d}w}R(w)$, to 0. The logic for the median and mean absolute error is more fresh from this section, so let's think in terms of the mean and mean squared error, from Chapter 1.2. There, we found that:

$$\frac{\text{d}}{\text{d}w}R_{\text{sq}}(w) = -\frac{2}{n}\sum_{i=1}^{n}(y_i - w)$$

Setting this to 0 gave us the balance equation above.

$$\sum_{i=1}^{n}(y_i - w^*) = 0$$

In English, this is saying that the sum of **deviations** from each data point to the mean is 0. ("Deviation" just means "difference from the mean.) Or, in other words, the positive differences and negative differences cancel each other out at the mean, and the mean is the unique point where this happens.

Let me illustrate using our familiar small example dataset.

$$61 \quad 72 \quad 85 \quad 90 \quad 92$$

The mean is 80. Then:

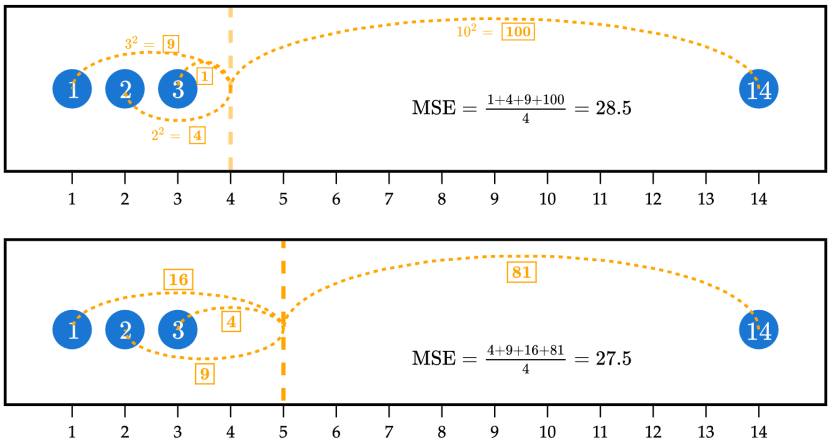
$$\underbrace{(61-80)}_{-19} + \underbrace{(72-80)}_{-8} + \underbrace{(85-80)}_{5} + \underbrace{(90-80)}_{10} + \underbrace{(92-80)}_{12} = 0$$

Note that the negative deviations and positive deviations both total 27 in magnitude.

While the mean balances the positive and negative deviations, the median balances the number of points on either side, without regard to how far the values are from the median.

Here's another perspective: the squared loss more heavily penalizes outliers, and so the resulting predictions "cater to" or are "pulled" towards these outliers.

We just derived the absolute-loss solution for the constant model; now we'll compare it to squared loss and examine how the choice of loss changes what "best" means.

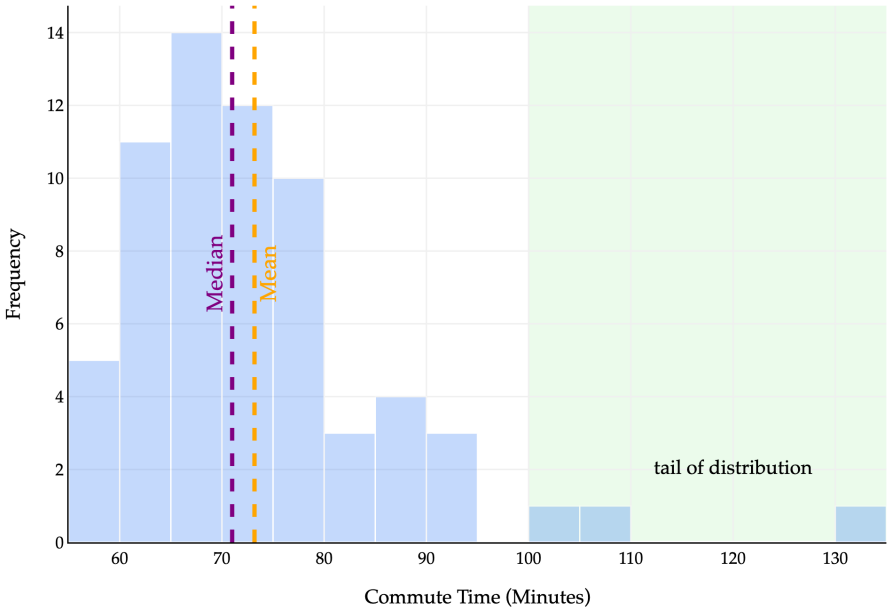


In the example above, the top plot visualizes the squared loss for the constant prediction of $w = 4$ against the dataset 1, 2, 3, and 14. While it has relatively small squared losses to the three points on the left, it has a very large squared loss to the point at 14, of $(14-4)^2 = 100$, which causes the mean squared error to be large.

In efforts to reduce the overall mean squared error, the optimal w^* is pulled towards 14. $w^* = 5$ has larger squared losses to the points at 1, 2, and 3 than $w = 4$ did, but a much smaller squared loss to the point at 14, of $(14-5)^2 = 81$. The "savings" from going from a squared loss of $10^2 = 100$ to $9^2 = 81$ more than makes up for the additional squared losses to the points at 1, 2, and 3.

In short: models that are fit using squared loss are strongly pulled towards outliers, in an effort to keep mean squared error low. Models fit using absolute loss don't have this tendency.

To conclude, let me visualize the behavior of the mean and median with a larger dataset – the full dataset of commute times we first saw at the start of Chapter 1.2.



The median is the point at which half the values are below it and half are above it. In the histogram above, **half of the area is to the left of the median and half is to the right**.

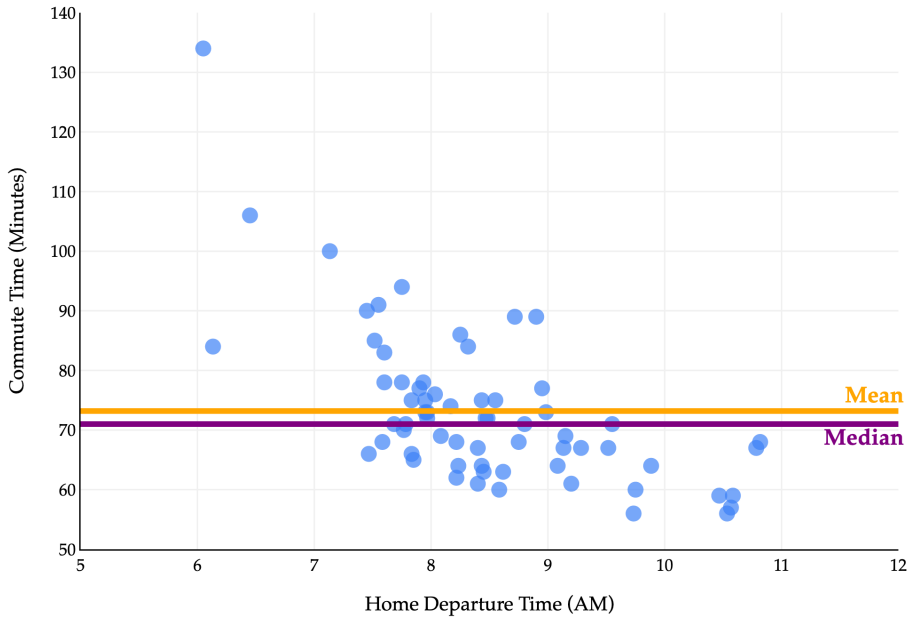
The mean is the point at which the sum of deviations from each value to the mean is 0. Another interpretation: if you placed this histogram on a playground see-saw, the mean would be the point at which the see-saw is balanced. Wikipedia has a good illustration of this general idea.

We say the distribution above is **right-skewed** or **right-tailed** because the tail is on the right side of the distribution. (This is counterintuitive to me, because most of the data is on the left of the distribution.)

In general, the mean is pulled in the direction of the tail of a distribution:

- If a distribution is symmetric, i.e. has roughly the same-shaped tail on the left and right, the mean and median are similar.
- If a distribution is right-skewed, the mean is pulled to the right of the median, i.e. Mean > Median.
- If a distribution is left-skewed, the mean is pulled to the left of the median, i.e. Mean < Median.

This explains why Mean > Median in the histogram above, and equivalently, in the scatter plot below.



Many common distributions in the real world are right-skewed, including incomes and net worths, and in such cases, the mean doesn't tell the full story.

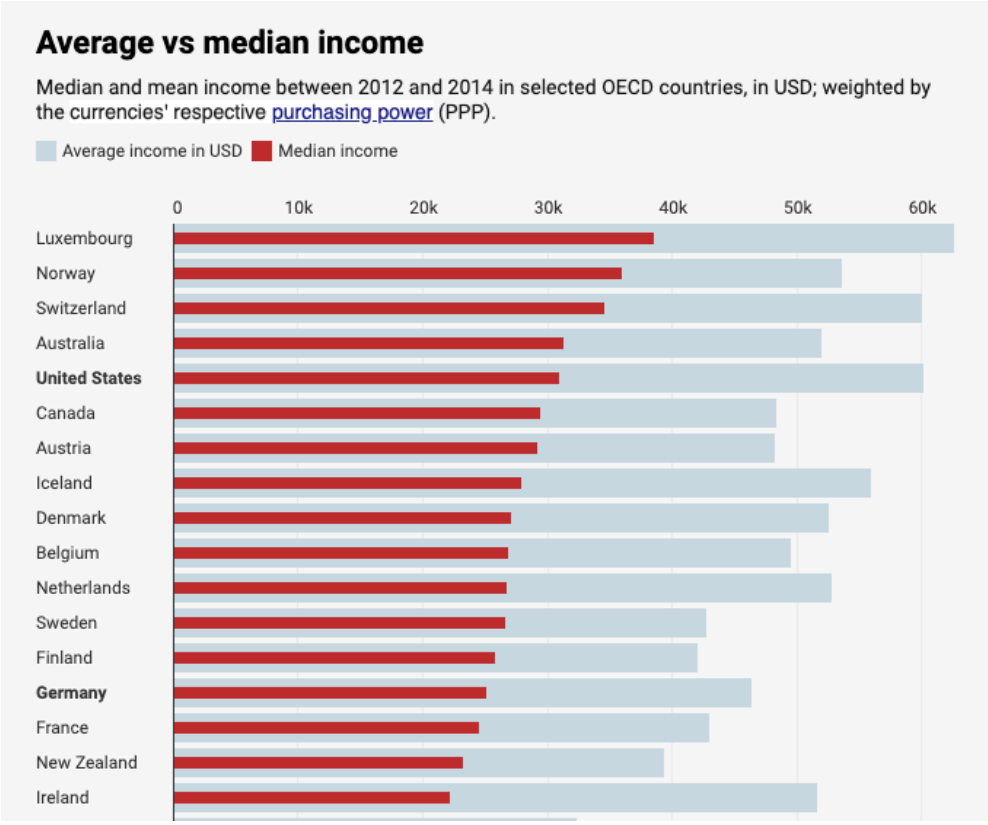


Figure 1.6: *
Mean (average) and median incomes for several countries (source).

When we move to more sophisticated models with (many) more parameters, the optimal parameter values won't

be as easily interpretable as the mean and median of our data, but the effects of our choice of loss function will still be felt in the predictions we make.

Beyond Absolute and Squared Loss

You may have noticed that the absolute loss and squared loss functions both look relatively similar:

$$L_{\text{abs}}(y_i, h(x_i)) = |y_i - h(x_i)|$$

$$L_{\text{sq}}(y_i, h(x_i)) = (y_i - h(x_i))^2$$

Both of these loss functions are special cases of a more general class of loss functions, known as L_p loss functions. For any $p \geq 1$, define the L_p loss as follows:

$$L_p(y_i, h(x_i)) = |y_i - h(x_i)|^p$$

Suppose we continue to use the constant model, $h(x_i) = w$. Then, the corresponding empirical risk for L_p loss is:

$$R_p(w) = \frac{1}{n} \sum_{i=1}^{n} |y_i - w|^p$$

We've studied, in depth, the minimizers of $R_p(w)$ for $p = 1$ (the median) and $p = 2$ (the mean). What about when $p = 3$, or $p = 4$, or $p = 100$? What happens as $p \to \infty$?

Let me be a bit less abstract. Suppose we have $p = 6$. Then, we're looking for the constant prediction w that minimizes the following:

$$R_6(w) = \frac{1}{n} \sum_{i=1}^{n} |y_i - w|^6 = \frac{1}{n} \sum_{i=1}^{n} (y_i - w)^6$$

Note that I dropped the absolute value, because $(y_i - w)^6$ is always non-negative, since 6 is an even number.

To find w^* here, we need to take the derivative of $R_6(w)$ with respect to w and set it equal to 0.

$$\frac{\text{d}}{\text{d}w} R_6(w) = -\frac{6}{n} \sum_{i=1}^{n} (y_i - w)^5$$

Setting the above to 0 gives us a new balance condition: $\sum_{i=1}^{n} (y_i - w)^5 = 0$. The minimizer of $R_2(w)$ was the point at which the balance condition $\sum_{i=1}^{n} (y_i - w) = 0$ was satisfied; equivalently, the minimizer of $R_6(w)$ is the point at which the balance condition $\sum_{i=1}^{n} (y_i - w)^5 = 0$ is satisfied. You'll notice that the degree of the differences in the balance condition is one lower than the degree of the differences in the loss function — this comes from the power rule of differentiation.

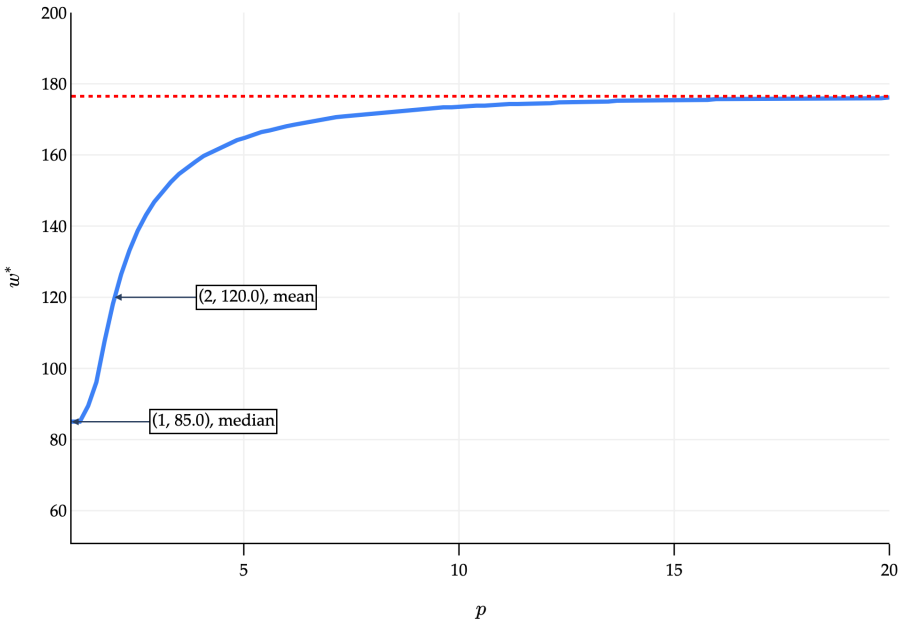
At what point w^* does $\sum_{i=1}^{n}(y_i - w^*)^5 = 0$? It's challenging to determine the value by hand, but the computer can approximate solutions for us, as you'll see in Lab 2.

Below, you'll find a computer-generated graph where:

- The x-axis is p.
- The y-axis represents the value of w^* that minimizes $R_p(w)$, for the dataset

$$61 \quad 72 \quad 85 \quad 90 \quad 292$$

that we saw earlier. Note the maximum value in our dataset is 292.



As $p \to \infty$, w^* approaches some value.

Activity 1

Activity 1

As $p \to \infty$, what value does w^* approach, and why?

Solution

As $p \to \infty$, w^* approaches:

$$\text{midrange} = \frac{\min + \max}{2}$$

Intuitively, as $p \to \infty$, we're minimizing the **worst-case distance** to any point in the dataset, i.e. the maximum distance to any point in the dataset. To keep the maximum distance as small as possible, we need to be directly in between the two extreme points of the dataset. Think of this as a "tug-of-war" between the minimum and maximum values of the dataset.

On the other extreme end, let me introduce yet another loss function, **0-1 loss**:

$$L_{0,1}(y_i, h(x_i)) = \begin{cases} 0 & y_i = h(x_i) \\ 1 & y_i \neq h(x_i) \end{cases}$$

The corresponding empirical risk, for the constant model $h(x_i) = w$, is:

$$R_{0,1}(w) = \frac{1}{n} \sum_{i=1}^{n} L_{0,1}(y_i, w)$$

This is the sum of 0s and 1s, divided by n. A 1 is added to the sum each time $y_i \neq w$. So, in other words, $R_{0,1}(w)$ is:

$$R_{0,1}(w) = \frac{\text{number of points not equal to } w}{n}$$

To minimize empirical risk, we want the number of points not equal to w to be as small as possible. So, w^* is the **mode (i.e. most frequent value)** of the dataset. If all values in the dataset are unique, they all minimize average 0-1 loss. This is not a useful loss function for regression, since our predictions are drawn from the continuous set of real numbers, but is useful for classification.

Center and Spread

Prior to taking EECS 245, you knew about the mean, median, and mode of a dataset. What you now know is that each one of these **summary statistics** comes from minimizing empirical risk (i.e. average loss) for a different loss function. All three measure the **center** of the dataset in some way.

Loss	Minimizer of Empirical Risk	Always Unique?	Robust to Outliers?	Empirical Risk Differentiable?
L_{sq}	mean	yes True	no False	yes True
L_{abs}	median	no False	yes True	no False
L_{∞}	midrange	yes True	no False	no False
$L_{0,1}$	mode	no False	no False	no False

So far, we've focused on finding model parameters that minimize empirical risk. But, we never stopped to think

about what the minimum empirical risk itself is! Consider the empirical risk for squared loss and the constant model:

$$R_{\text{sq}}(w) = \frac{1}{n}\sum_{i=1}^{n}(y_i - w)^2$$

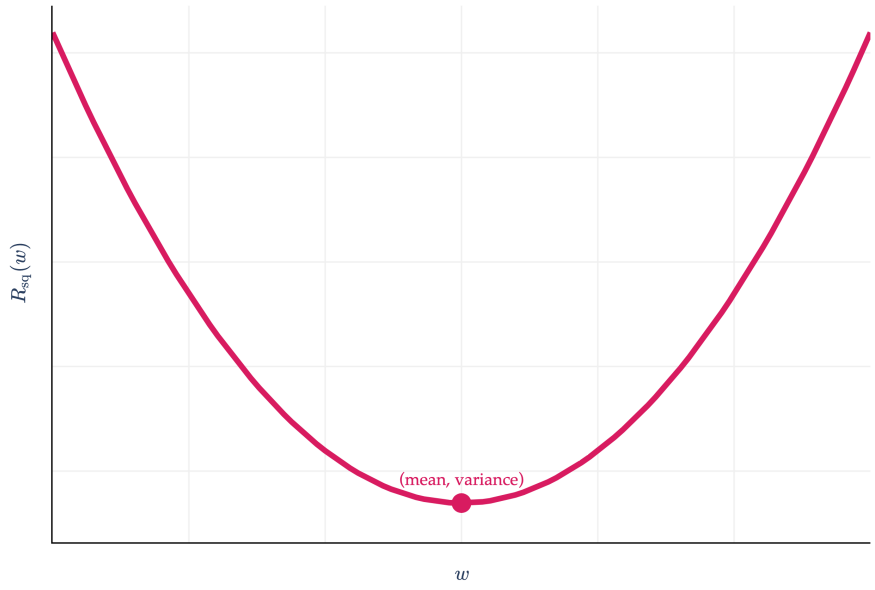
$R_{\text{sq}}(w)$ is minimized when w^* is the mean, which I'll denote with $\bar{y}$. What happens if I plug $w^* = \bar{y}$ back into R_{sq}?

$$R_{\text{sq}}(w^*) = R_{\text{sq}}(\bar{y}) = \frac{1}{n}\sum_{i=1}^{n}(y_i - \bar{y})^2$$

This is the **variance** of the dataset $y_1, y_2, ..., y_n$! The variance is nothing but the average squared deviation of each value from the mean of the dataset.

This gives context to the y-axis value of the vertex of the parabola we saw in Chapter 1.2.

$$R_{\text{sq}}(w) = \frac{1}{n}\sum_{i=1}^{n}(y_i - w)^2$$



Practically speaking, this gives us a nice "worst-case" mean squared error of any regression model on a dataset. If we learn how to build a sophisticated regression model, and its mean squared error is somehow greater than the variance of the dataset, we know that we're doing something wrong, since we could do better just by predicting the mean!

The units of the variance are the square of the units of the y-values. So, if the y_i's represent commute times in minutes, the variance is in minutes^2. This makes it a bit difficult to interpret. So, we typically take the square root of the variance, which gives us the standard deviation, σ:

$$\sigma = \sqrt{\text{variance}} = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(y_i - \bar{y})^2}$$

The standard deviation has the same units as the y-values themselves, so it's a more interpretable measure of spread.

How does this work in the context of absolute loss?

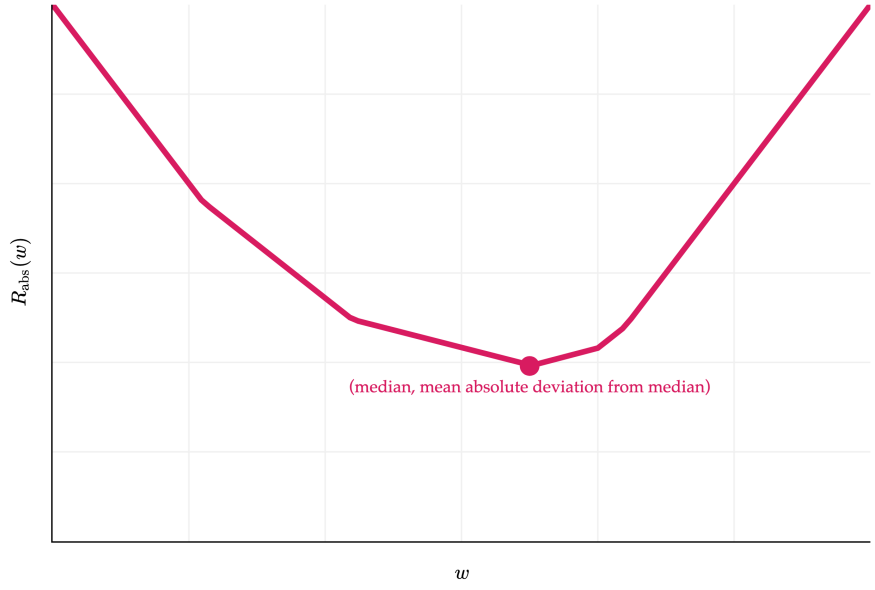
$$R_{\text{abs}}(w) = \frac{1}{n}\sum_{i=1}^{n}|y_i - w|$$

Plugging in $w^* = \text{Median}(y_1, y_2, ..., y_n)$ into $R_{\text{abs}}(w)$ gives us:

$$\begin{aligned} R_{\text{abs}}(w^*) &= \frac{1}{n}\sum_{i=1}^{n}|y_i - w^*| \\ &= \frac{1}{n}\sum_{i=1}^{n}|y_i - \text{Median}(y_1, y_2, ..., y_n)| \end{aligned}$$

I'll admit, this result doesn't have a special name. It is the **mean absolute deviation from the median**. And, like the variance and standard deviation, it measures roughly how far **spread out** the data is from its center. Its units are the same as the y-values themselves (since there's no squaring involved).

$$R_{\text{abs}}(w) = \frac{1}{n}\sum_{i=1}^{n}|y_i - w|$$



Activity 2

Activity 2

Consider the dataset:

$$1 \quad 3 \quad 5 \quad 7 \quad 64$$

Compute the following:

1. The variance.
2. The mean squared error of the **median**.
3. The mean absolute deviation from the **median**.
4. The mean absolute deviation from the **mean**.

What do you notice about the results to (1) and (2)? What about the results to (3) and (4)?

In the real-world, be careful when you hear the term "mean absolute deviation", as sometimes it's used to refer to the median absolute deviation from the mean, not the median as above.

Activity 3

Activity 3

What is the value of $R_{0,1}(w^*)$ for the constant model $h(x_i) = w$ and 0-1 loss? How does it measure the spread of the data?

To reiterate, in practice, our models will have many, many more parameters than just one, as is the case for the constant model. But, by deeply studying the effects of choosing squared loss vs. absolute loss vs. other loss functions in the context of the constant model, we can develop a better intuition for how to choose loss functions in more complex situations.

Simple Linear Regression

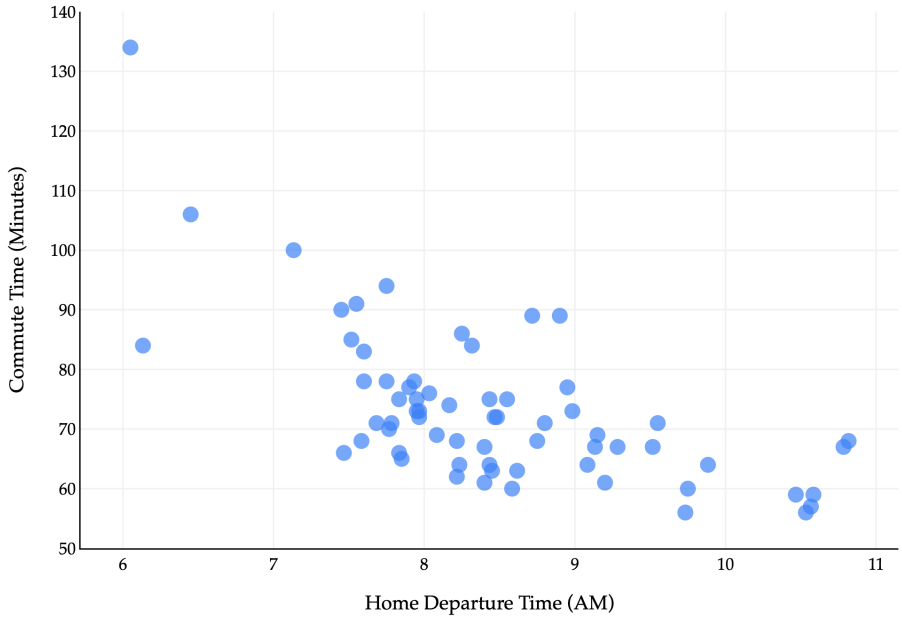
2.1. Overview

Introduction

The time has finally come: let's apply what we've learned about loss functions and the modeling recipe to "upgrade" from the constant model to the simple linear regression model.

To recap, our goal is to find a hypothesis function h such that:

$$\text{predicted commute time}_i = h(\text{departure hour}_i)$$



So far, we've studied the constant model, where the hypothesis function is a horizontal line:

$$h(x_i) = w$$

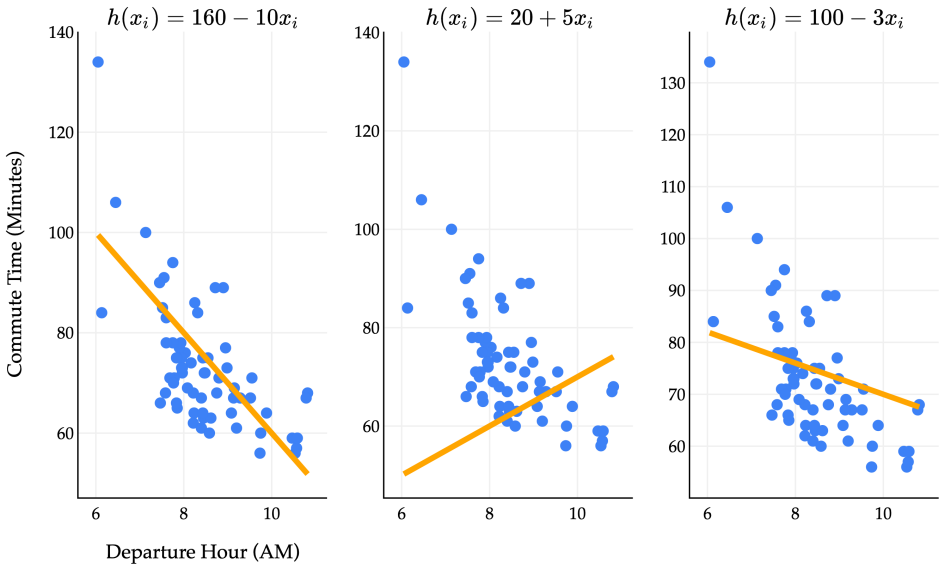
The sole parameter, w, controlled the height of the line. Up until now, "parameter" and "prediction" were interchangeable terms, because our sole parameter w controlled what our constant prediction was.

Now, the simple linear regression model has two parameters:

$$h(x_i) = w_0 + w_1 x_i$$

w_0 controls the intercept of the line, and w_1 controls its slope. No longer is it the case that "parameter" and "prediction" are interchangeable terms, because w_0 and w_1 control different aspects of the prediction-making process.

How do we find the optimal parameters, w_0^* and w_1^*? Different values of w_0 and w_1 give us different lines, each of which fit the data with varying degrees of accuracy.



Activity 1

Activity 1

Consider a dataset with two points, $(3, 5)$ and $(15, 53)$. What are the optimal parameters, w_0^* and w_1^*, for the line $h(x_i) = w_0 + w_1 x_i$ that minimizes mean squared error for this dataset?

To make things precise, let's turn to the three-step modeling recipe from Chapter 1.3.

1. Choose a model.

$$h(x_i) = w_0 + w_1 x_i$$

2. Choose a loss function.

We'll stick with squared loss:

$$L_{\text{sq}}(y_i, h(x_i)) = (y_i - h(x_i))^2$$

3. Minimize average loss (also known as empirical risk) to find optimal parameters.

Average squared loss – also known as mean squared error – for any hypothesis function h, takes the form:

$$\frac{1}{n} \sum_{i=1}^{n} (y_i - h(x_i))^2$$

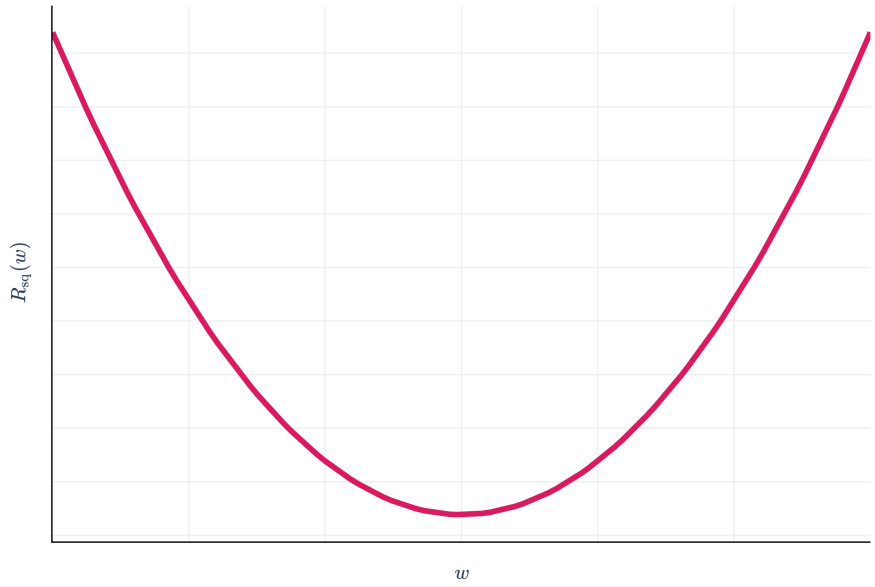
For the simple linear regression model, this becomes:

$$R_{\text{sq}}(w_0, w_1) = \frac{1}{n} \sum_{i=1}^{n} (y_i - (w_0 + w_1 x_i))^2$$

Now, we need to find the values of w_0 and w_1 that together minimize $R_{\text{sq}}(w_0, w_1)$. But what does that even

mean?

In the case of the context model and squared loss, where we had to minimize $R_{\text{sq}}(w) = \frac{1}{n}\sum_{i=1}^{n}(y_i - w)^2$, we did so by taking the derivative with respect to w and setting it to 0.



$R_{\text{sq}}(w)$ was a function with just a single input variable (w), so the problem of minimizing $R_{\text{sq}}(w)$ was straightforward, and resembled problems we solved in Calculus 1.

The function $R_{\text{sq}}(w_0, w_1)$ we're minimizing now has **two** input variables, w_0 and w_1. In mathematics, sometimes we'll write $R_{\text{sq}} : \mathbb{R}^2 \to \mathbb{R}$ to say that R_{sq} is a function that takes in two real numbers and returns a single real number.

$$R_{\text{sq}}(w_0, w_1) = \frac{1}{n}\sum_{i=1}^{n}(y_i - (w_0 + w_1 x_i))^2$$

Remember, we should treat the x_i's and y_i's as constants, as these are known quantities once we're given a dataset.

What does $R_{\text{sq}}(w_0, w_1)$ even look like? We need three dimensions to visualize it – one axis for w_0, one for w_1, and one for the output, $R_{\text{sq}}(w_0, w_1)$.

The graph above is called a **loss surface**, even though it's a graph of empirical risk, i.e. average loss, not the loss for a single data point. The plot is interactive, so you should drag it around to get a sense of what it looks like. It looks like a parabola with added depth, similar to how cubes look like squares with added depth. Lighter regions above correspond to low mean squared error, and darker regions correspond to high mean squared error.

Think of the "floor" of the graph – in other words, the w_0-w_1 plane – as all the set of possible combinations of intercept and slope. The height of the surface at any point (w_0, w_1) is the mean squared error of the hypothesis $h(x_i) = w_0 + w_1 x_i$ on the commute times dataset.

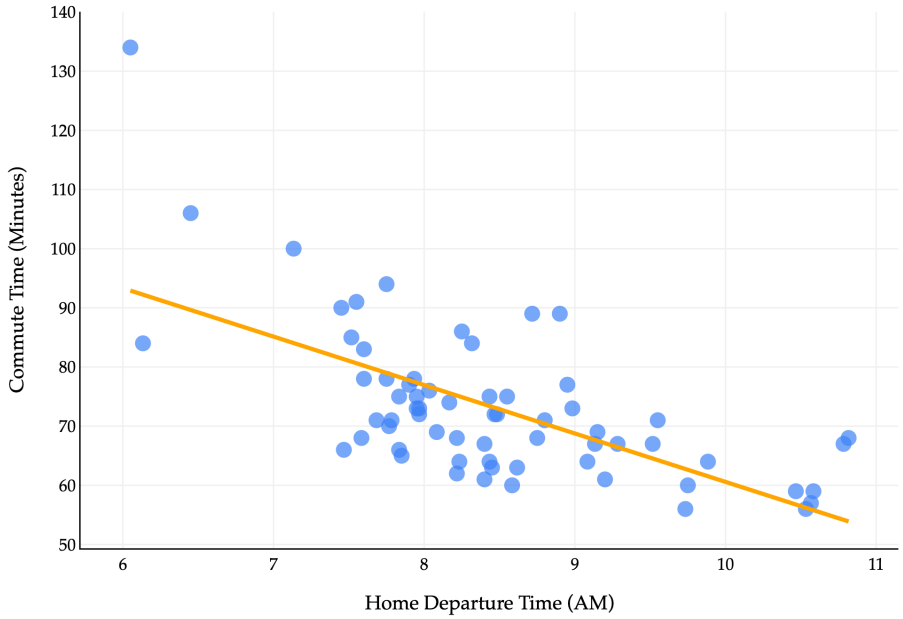
Our goal is to find the combination of w_0 and w_1 that get us to the bottom of the surface, marked by the gold point in the plot. This will involve calculus and derivatives, but we'll need to extend our single variable approach: we'll need to take **partial derivatives** with respect to w_0 and w_1. Chapter 2.2 is a detour that describes how these work; in Chapter 2.3, we'll use them to find the optimal parameters.

A Preview

Just so you have them, though, here's what we'll end up finding:

$$\boxed{w_1^* = \frac{\sum_{i=1}^n (x_i - \bar{x})(y_i - \bar{y})}{\sum_{i=1}^n (x_i - \bar{x})^2}, \qquad w_0^* = \bar{y} - w_1^* \bar{x}}$$

These are formulas that describe the optimal slope, w_1^*, and intercept, w_0^*, for the simple linear regression model, given a dataset $(x_1, y_1), (x_2, y_2), \ldots, (x_n, y_n)$. They are chosen to minimize mean squared error. On our commute times dataset, the resulting line looks like this:



2.2. Detour: Partial Derivatives

Partial Derivatives

How do we take the derivative of a function with multiple input variables?

$$R_{\text{sq}}(w_0, w_1) = \frac{1}{n}\sum_{i=1}^{n}(y_i - (w_0 + w_1 x_i))^2$$

To illustrate, let's focus on a simpler function with two input variables:

$$f(x, y) = \frac{x^2 + y^2}{9}$$

This is a quadratic function of two variables, and its graph is known as a paraboloid.

In the single-input case – i.e., for functions of the form $f : \mathbb{R} \to \mathbb{R}$ – the derivative $\frac{\mathrm{d}}{\mathrm{d}x} f(x)$ captured $f(x)$'s rate of change along the x-axis, which was the only axis of motion.

The function $f(x, y)$ has two input variables, and so there are two directions along which we can move. As such, we need two "derivatives" to describe the rate of change of $f(x, y)$ – one for the x-axis and one for the y-axis. Think of this as a science experiment, where we need control variables to isolate changes to a single variable. Our solution to this dilemma comes in the form of partial derivatives.

Definition: Partial Derivative

Suppose $f : \mathbb{R}^n \to \mathbb{R}$ is a function of n input variables, $x, y, z, \ldots$.
The **partial derivative of f with respect to x**, denoted $\frac{\partial f}{\partial x}(x, y, z, \ldots)$ – or simply $\frac{\partial f}{\partial x}$ – is the derivative of f with respect to x, **when treating all other input variables as constants.**

If f has n input variables, it has n partial derivatives, one for each axis. The function $f(x, y) = \frac{x^2+y^2}{9}$ has two partial derivatives, $\frac{\partial f}{\partial x}(x, y)$ and $\frac{\partial f}{\partial y}(x, y)$. (The symbol you're seeing, ∂, is the lowercase Greek letter delta, and is used specifically for partial derivatives.)

Let me show you how to compute partial derivatives before we visualize them. We'll start with $\frac{\partial f}{\partial x}(x, y)$.

$$\begin{aligned}
f(x, y) &= \frac{x^2 + y^2}{9} \\
\frac{\partial f}{\partial x}(x, y) &= \frac{\partial}{\partial x}\left(\frac{x^2 + y^2}{9}\right) \\
&= \frac{1}{9}\frac{\partial}{\partial x}(x^2 + y^2) \\
&= \frac{1}{9}\left(\frac{\partial}{\partial x}x^2 + \underbrace{\frac{\partial}{\partial x}y^2}_{=0}\right) \\
&= \frac{1}{9}(2x + 0) \\
&= \frac{2x}{9}
\end{aligned}$$

The result, $\frac{\partial f}{\partial x}(x, y) = \frac{2x}{9}$, is a function of x and y. It tells us the rate of change of $f(x, y)$ along the x axis, at any point (x, y). It just so happens that this function doesn't involve y since we chose a relatively simple function f, but we'll see more sophisticated examples soon.

Following similar steps, you'll see that $\frac{\partial f}{\partial y}(x, y) = \frac{2y}{9}$. This gives us:

$$\frac{\partial f}{\partial x}(x, y) = \frac{2x}{9}, \quad \frac{\partial f}{\partial y}(x, y) = \frac{2y}{9}$$

Let's pick an arbitrary point and see what the partial derivatives tell us about it. Consider, say, $(-3, 0.5)$:

- $\frac{\partial f}{\partial x}(-3, 0.5) = \frac{2(-3)}{9} = -\frac{2}{3}$, so if we hold y constant, f decreases as x increases.
- $\frac{\partial f}{\partial y}(-3, 0.5) = \frac{2(0.5)}{9} = \frac{1}{9}$, so if we hold x constant, f increases as y increases.

Above, we've shown the tangent lines in both the x and y directions at the point $(-3, 0.5)$. After all, the derivative of a function at a point tells us the slope of the tangent line at that point; that interpretation remains true with partial derivatives.

Let's look at a more complex example. Consider:

$$g(x, y) = x^3 - 3xy^2 + 2\sin(x)\cos(y)$$

Both partial derivatives are functions of **both** x and y, which is typically what we'll see.

$$\begin{aligned} g(x, y) &= x^3 - 3xy^2 + 2\sin(x)\cos(y) \\ \frac{\partial g}{\partial x}(x, y) &= 3x^2 - 3y^2 + 2\cos(x)\cos(y) \\ \frac{\partial g}{\partial y}(x, y) &= -6xy - 2\sin(x)\sin(y) \end{aligned}$$

To compute $\frac{\partial g}{\partial x}(x, y)$, we treated y as a constant. Let me try and make more sense of this.

To help visualize, we've drawn the function $g(x, y)$, along with the plane $y = a$. The slider lets you change the value of a being considered, i.e., it lets you change the constant value that we're assigning to y.

The intersection of $g(x, y)$ and $y = a$ is marked as a gold curve and is a function of x alone.

Drag the slider to $y = 1.40$, for example, and look at the gold curve that results. The expression below tells you **the derivative of that** gold curve with respect to x.

$$\frac{\partial g}{\partial x}(x, 1.40) = 3x^2 - 3(1.40)^2 + 2\cos(x)\cos(1.40) = \underbrace{3x^2 - 0.34\cos(x) - 5.88}_{\text{derivative of } \textbf{gold curve} \text{ w.r.t. } x}$$

Thinking in three dimensions can be difficult, so don't fret if you're confused as to what all of these symbols mean – this is all a bit confusing to me too. (Are professors allowed to say this?) Nonetheless, I hope these interactive visualizations are helping you make some sense of the formulas, and if there's anything I can do to make them clearer, please do tell me!

Activity 1

Activity 1

Find all three partial derivatives of the function:

$$g(x, y, z) = 2x^2 + y^2 + 3z^2 + 2xy^2 - 4yz + 6x - 4z - 10$$

Optimization

To minimize (or maximize) a function $f : \mathbb{R} \to \mathbb{R}$, we solved for critical points, which were points where the (single variable) derivative was 0, and used the second derivative test to classify them as minima or maxima (or neither, like in the case of $f(x) = x^3$ at $x = 0$).

The analog in the $\mathbb{R}^2 \to \mathbb{R}$ case is solving for the points where **both partial derivatives are 0**, which corresponds to the points where the function is neither increasing nor decreasing along either axis.

In the case of our first example,

$$f(x, y) = \frac{x^2 + y^2}{9}$$

the partial derivatives were relatively simple,

$$\frac{\partial f}{\partial x} = \frac{2x}{9}, \quad \frac{\partial f}{\partial y} = \frac{2y}{9}$$

and both are 0 when $x = y = 0$. So, $(0, 0, f(0))$ is a critical point, and we can see visually that it's a global minimum.

(Notice that above I wrote $\frac{\partial f}{\partial x}$ and $\frac{\partial f}{\partial y}$ instead of $\frac{\partial f}{\partial x}(x, y)$ and $\frac{\partial f}{\partial y}(x, y)$ to save space, but don't forget that both partial derivatives are functions of both x and y in general.)

There is a second derivative test for functions of multiple variables, but it's a bit more complicated than the single variable case, and to give you an honest explanation of it, I'll need to introduce you to quite a bit of linear algebra first. So, we'll table that thought for now.

The function $g(x, y) = x^3 - 3xy^2 + 2\sin(x)\cos(y)$ has much more complicated partial derivatives, and so it's difficult to solve for its critical points by hand. Fear not – in Chapter 8, when we discover the technique of gradient descent, we'll learn how to minimize such functions just by using their partial derivatives, even when we can't solve for where they're 0.

Activity 2

Activity 2

Find the values of x_1 and x_2 that minimize the function:

$$g(x_1, x_2) = 100(x_2 - x_1^2)^2 + (1 - x_1)^2$$

Here, we've used x_1 and x_2 to denote the two input variables, rather than x and y.

With partial derivatives in hand, we can now minimize mean squared error to solve for the optimal regression parameters.

2.3. Finding Optimal Parameters

Finding the Partial Derivatives

Let's return to the simple linear regression problem. Recall, the function we're trying to minimize is:

$$R_{\text{sq}}(w_0, w_1) = \frac{1}{n}\sum_{i=1}^{n}(y_i - (w_0 + w_1 x_i))^2$$

Why? By minimizing $R_{\text{sq}}(w_0, w_1)$, we're finding the intercept (w_0^*) and slope (w_1^*) of the line that best fits the data. **Don't forget that this goal is the point of all of these mathematical ideas.**

We've learned that to minimize $R_{\text{sq}}(w_0, w_1)$, we'll need to find both of its partial derivatives, and solve for the point $(w_0^*, w_1^*, R_{\text{sq}}(w_0^*, w_1^*))$ at which they're both 0.

Let's start with the partial derivative with respect to w_0:

$$\begin{aligned}
R_{\text{sq}}(w_0, w_1) &= \frac{1}{n}\sum_{i=1}^{n}(y_i - (w_0 + w_1 x_i))^2 \\
\frac{\partial R_{\text{sq}}}{\partial w_0} &= \frac{\partial}{\partial w_0}\left[\frac{1}{n}\sum_{i=1}^{n}(y_i - (w_0 + w_1 x_i))^2\right] \\
&= \frac{1}{n}\sum_{i=1}^{n}\frac{\partial R_{\text{sq}}}{\partial w_0}\left(y_i - (w_0 + w_1 x_i)\right)^2 \\
&= \frac{1}{n}\sum_{i=1}^{n}2(y_i - (w_0 + w_1 x_i)) \cdot \underbrace{\frac{\partial R_{\text{sq}}}{\partial w_0}(y_i - (w_0 + w_1 x_i))}_{\text{chain rule}} \\
&= \frac{1}{n}\sum_{i=1}^{n}2(y_i - (w_0 + w_1 x_i)) \cdot (-1) \\
&= -\frac{2}{n}\sum_{i=1}^{n}(y_i - (w_0 + w_1 x_i))
\end{aligned}$$

Onto w_1:

$$
\begin{aligned}
R_{\text{sq}}(w_0, w_1) &= \frac{1}{n}\sum_{i=1}^{n}(y_i - (w_0 + w_1 x_i))^2 \\
\frac{\partial R_{\text{sq}}}{\partial w_1} &= \frac{\partial}{\partial w_1}\left[\frac{1}{n}\sum_{i=1}^{n}(y_i - (w_0 + w_1 x_i))^2\right] \\
&= \frac{1}{n}\sum_{i=1}^{n}\frac{\partial R_{\text{sq}}}{\partial w_1}\left(y_i - (w_0 + w_1 x_i)\right)^2 \\
&= \frac{1}{n}\sum_{i=1}^{n} 2(y_i - (w_0 + w_1 x_i)) \cdot \underbrace{\frac{\partial R_{\text{sq}}}{\partial w_1}(y_i - (w_0 + w_1 x_i))}_{\text{chain rule}} \\
&= \frac{1}{n}\sum_{i=1}^{n} 2(y_i - (w_0 + w_1 x_i)) \cdot (-x_i) \\
&= -\frac{2}{n}\sum_{i=1}^{n} x_i(y_i - (w_0 + w_1 x_i))
\end{aligned}
$$

All in one place now:

$$
\begin{aligned}
\frac{\partial R_{\text{sq}}}{\partial w_0} &= -\frac{2}{n}\sum_{i=1}^{n}(y_i - (w_0 + w_1 x_i)) \\
\frac{\partial R_{\text{sq}}}{\partial w_1} &= -\frac{2}{n}\sum_{i=1}^{n} x_i(y_i - (w_0 + w_1 x_i))
\end{aligned}
$$

These look very similar – it's just $\frac{\partial R_{\text{sq}}}{\partial w_1}$ has an added x_i in the summation.

Remember, both partial derivatives are functions of two variables: w_0 and w_1. We're treating the x_i's and y_i's as constants. If I already have a dataset, you can pick an intercept w_0 and slope w_1 and I can use these formulas to compute the partial derivatives of R_{sq} for that combination of intercept and slope.

In case it helps you put things in perspective, here's how I might implement these formulas in code, assuming that `x` and `y` are arrays:

```
# Assume x and y are defined somewhere above this function.
def partial_R_w0(w0, w1):
    # Sub -optimal technique, since it uses a for -loop.
    total = 0
    for i in range(len(x)):
        total += (y[i] - (w0 + w1 * x[i]))
    return -2 * total / len(x)
    # Returns a single number!

def partial_R_w1(w0, w1):
    # Better technique, as it uses vectorized operations.
    return -2 * np.mean(x * (y - (w0 + w1 * x)))
    # Also returns a single number!
```

Before we solve for where both $\frac{\partial R_{\text{sq}}}{\partial w_0}$ and $\frac{\partial R_{\text{sq}}}{\partial w_1}$ are 0, let's visualize them in the context of our loss surface.

Click "Slider for values of w_0". No matter where you drag that slider, the resulting gold curve is a function of w_1 only. Every gold curve you see when dragging the w_0 slider will have a minimum at some value of w_1.

Then, click "Slider for values of w_1". No matter where you drag that slider, the resulting gold curve is a function of w_0 only, and has some minimum value.

But there is only one combination of w_0 and w_1 where the gold curves have minimums at the exact same intersecting point. That is the combination of w_0 and w_1 that minimizes R_{sq}, and it's who we're searching for.

Solving for the Optimal Parameters

Now, it's time to analytically (that is, on paper) find the values of w_0^* and w_1^* that minimize R_{sq}. We'll do so by solving the following system of two equations and two unknowns:

$$\frac{\partial R_{\text{sq}}}{\partial w_0} = -\frac{2}{n}\sum_{i=1}^{n}(y_i - (w_0 + w_1 x_i)) = 0$$

$$\frac{\partial R_{\text{sq}}}{\partial w_1} = -\frac{2}{n}\sum_{i=1}^{n} x_i(y_i - (w_0 + w_1 x_i)) = 0$$

Here's my plan:

1. In the first equation, try and isolate for w_0; this value will be called w_0^*.
2. Plug the expression for w_0^* into the second equation to solve for w_1^*.

Let's start with the **first step**.

$$-\frac{2}{n}\sum_{i=1}^{n}(y_i - (w_0 + w_1 x_i)) = 0$$

Multiplying both sides by $-\frac{n}{2}$ gives us:

$$\sum_{i=1}^{n}(\underbrace{y_i}_{\text{actual}} - \underbrace{(w_0 + w_1 x_i)}_{\text{predicted}}) = 0$$

Before I continue, I want to highlight that this itself is an importance balance condition, much like those we discussed in Chapter 1.3. It's saying that the sum of the errors of the optimal line's predictions – that is, the line with intercept w_0^* and slope w_1^* – is 0.

Let's continue with the first step – I'll try and keep the commentary to a minimum. **It's important to try and replicate these steps yourself, on paper.**

$$\sum_{i=1}^{n}(y_i - (w_0 + w_1 x_i)) = 0$$
$$\sum_{i=1}^{n}(y_i - w_0 - w_1 x_i) = 0$$
$$\sum_{i=1}^{n} y_i - \sum_{i=1}^{n} w_0 - \sum_{i=1}^{n} w_1 x_i = 0$$
$$\sum_{i=1}^{n} y_i - n w_0 - w_1 \sum_{i=1}^{n} x_i = 0$$
$$\sum_{i=1}^{n} y_i - w_1 \sum_{i=1}^{n} x_i = n w_0$$
$$\frac{\sum_{i=1}^{n} y_i}{n} - w_1 \frac{\sum_{i=1}^{n} x_i}{n} = w_0$$
$$w_0^* = \bar{y} - w_1^* \bar{x}$$

Awesome! We're halfway there. We have a formula for the optimal slope, w_0^*, in terms of the optimal intercept, w_1^*. Let's use $w_0^* = \bar{y} - w_1^* \bar{x}$ and see where it gets us in the second equation.

$$-\frac{2}{n}\sum_{i=1}^{n} x_i(y_i - (w_0 + w_1 x_i)) = 0$$
$$\sum_{i=1}^{n} x_i(y_i - (w_0 + w_1 x_i)) = 0$$
$$\sum_{i=1}^{n} x_i(y_i - (\underbrace{\bar{y} - w_1^* \bar{x}}_{w_0^*} + w_1^* x_i)) = 0$$
$$\sum_{i=1}^{n} x_i \underbrace{(y_i - \bar{y} + w_1^* \bar{x} - w_1^* x_i)}_{\text{distribute negation}} = 0$$
$$\sum_{i=1}^{n} x_i \left((y_i - \bar{y}) - w_1^*(x_i - \bar{x})\right) = 0$$
$$\underbrace{\sum_{i=1}^{n} x_i(y_i - \bar{y}) - w_1^* \sum_{i=1}^{n} x_i(x_i - \bar{x}) = 0}_{\text{expand summation}} \quad \sum_{i=1}^{n} x_i(y_i - \bar{y}) = w_1^* \sum_{i=1}^{n} x_i(x_i - \bar{x})$$
$$w_1^* = \frac{\sum_{i=1}^{n} x_i(y_i - \bar{y})}{\sum_{i=1}^{n} x_i(x_i - \bar{x})}$$

Rewriting and Using the Formulas

We're done! We have formulas for the optimal slope and intercept. But, before we celebrate, I'm going to try and rewrite w_1^* in an equivalent, more symmetrical form that is easier to interpret.

Claim:

$$w_1^* = \underbrace{\frac{\sum_{i=1}^n x_i(y_i - \bar{y})}{\sum_{i=1}^n x_i(x_i - \bar{x})}}_{\text{formula we derived above}} = \underbrace{\frac{\sum_{i=1}^n (x_i - \bar{x})(y_i - \bar{y})}{\sum_{i=1}^n (x_i - \bar{x})^2}}_{\text{nicer looking formula}}$$

Proof of equivalent, nicer looking formula

To show that these two formulas are equal, I'll start by recapping the fact that the sum of deviations from the mean is 0, in other words:

$$\sum_{i=1}^{n}(x_i - \bar{x}) = 0$$

This has come up in homeworks and past sections of the notes, but for completeness, here's the proof:

$$\begin{aligned}
&\sum_{i=1}^{n} x_i - \bar{x} \\
&\sum_{i=1}^{n} x_i - \sum_{i=1}^{n} \bar{x} \\
&= n\bar{x} - n\bar{x} \\
&= 0
\end{aligned}$$

Equipped with this fact, I can show that the new, more symmetric version of the formula is equal to the original one I derived.

numerator of new formula: $\sum_{i=1}^{n}(x_i - \bar{x})(y_i - \bar{y})$
$= \sum_{i=1}^{n}\left(x_i(y_i - \bar{y}) - \bar{x}(y_i - \bar{y})\right)$
$= \sum_{i=1}^{n} x_i(y_i - \bar{y}) - \sum_{i=1}^{n} \bar{x}(y_i - \bar{y})$
$= \sum_{i=1}^{n} x_i(y_i - \bar{y}) - \bar{x} \underbrace{\sum_{i=1}^{n}(y_i - \bar{y})}_{=0}$
$= \sum_{i=1}^{n} x_i(y_i - \bar{y})$
= numerator of original formula!

denominator of new formula: $\sum_{i=1}^{n}(x_i - \bar{x})^2$
$= \sum_{i=1}^{n}(x_i - \bar{x})(x_i - \bar{x})$
$= \sum_{i=1}^{n} x_i(x_i - \bar{x}) - \underbrace{\bar{x} \sum_{i=1^n}(x_i - \bar{x})}_{\text{same logic as in the denominator case}}$
$= \sum_{i=1}^{n} x_i(x_i - \bar{x})$
= denominator of original formula!

We skipped some steps in the denominator case, since many of them are the same as in the numerator case. Nevertheless, since we've shown that the numerators of both formulas are the same and the denominators of both formulas are the same, well, both formulas are the same!

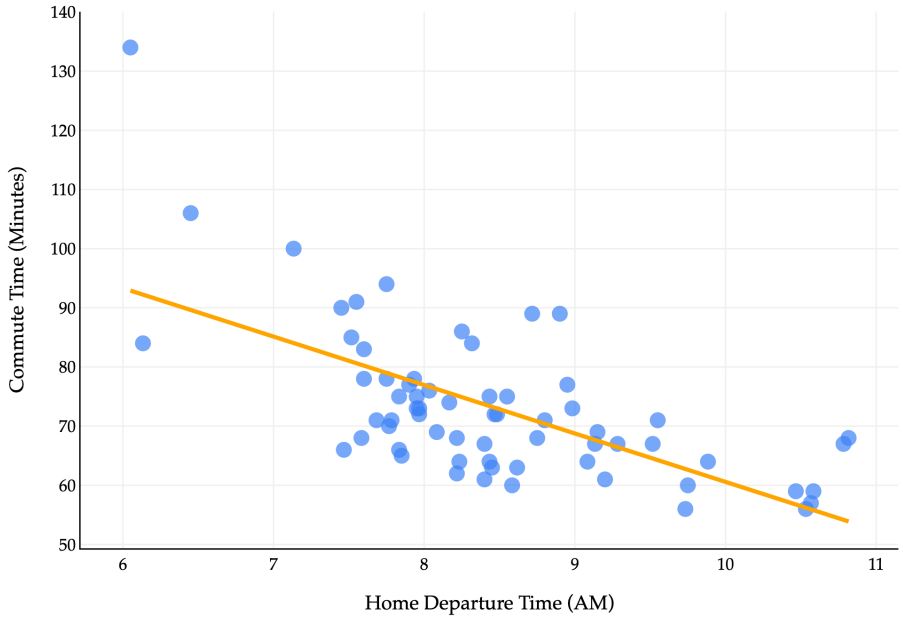
This is not the only other equivalent formula for the slope; for instance, $w_1^* = \frac{\sum_{i=1}^{n}(x_i - \bar{x})y_i}{\sum_{i=1}^{n}(x_i - \bar{x})^2}$ too, and you can verify this using the same logic as in the proof above.

To summarize, the parameters that minimize mean squared error for the simple linear regression model, $h(x_i) = w_0 + w_1 x_i$, are:

$$\boxed{w_1^* = \frac{\sum_{i=1}^n (x_i - \bar{x})(y_i - \bar{y})}{\sum_{i=1}^n (x_i - \bar{x})^2}, \qquad w_0^* = \bar{y} - w_1^* \bar{x}}$$

This is an important result, and you should remember it. There are a lot of symbols above, but just note that given a dataset $(x_1, y_1), (x_2, y_2), \ldots, (x_n, y_n)$, you *could* apply the formulas above by hand to find the optimal parameters yourself.

What does this line look like on the commute times data?



The line above goes by many names:

- The simple linear regression line that minimizes mean squared error.
- The simple linear regression line (if said without context).
- The regression line.
- The **least squares** regression line (because it has the *least* mean *squared* error).
- The line of best fit.

Whatever you'd like to call it, now that we've found our optimal parameters, we can use them to make predictions.

$$h(x_i) = w_0^* + w_1^* x_i$$

On the dataset of commute times:

```
# Assume x is an array with departure hours and y is an array with commute times.
w1_star = np.sum((x - np.mean(x)) * (y - np.mean(y))) / np.sum((x - np.mean(x)) ** 2)
w0_star = np.mean(y) - w1_star * np.mean(x)
```

```
w0_star, w1_star
```

```
(142.4482415877287, -8.186941724265552)
```

So, our specific **fit**, or **trained** hypothesis function is:

$$\begin{aligned}\text{predicted commute time}_i &= h(\text{departure hour}_i) \\ &= 142.45 - 8.19 \cdot \text{departure hour}_i\end{aligned}$$

This trained hypothesis function is **not** saying that leaving later *causes* you to have shorter commutes. Rather, that's just the best linear pattern it observed in the data for the purposes of minimizing mean squared error. In reality, there are other factors that affect commute times, and we haven't performed a thorough-enough analysis to say anything about the *causal* relationship between departure time and commute time.

To predict how long it might take to get to school tomorrow, plug in the time you'd like to leave for departure hour$_i$ and out will come your prediction. The slope, -8.19, is in units $\frac{\text{units of } y}{\text{units of } x} = \frac{\text{minutes}}{\text{hour}}$, and is telling us that for every hour later you leave, your predicted commute time decreases by 8.19 minutes.

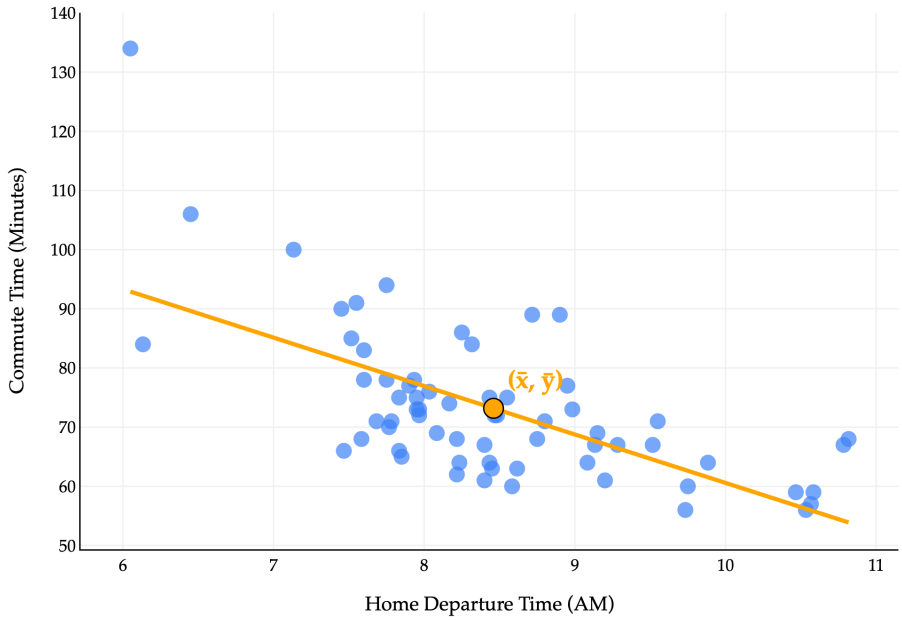
In Python, I can define a `predict` function as follows:

```
def predict(x_new):
    return w0_star + w1_star * x_new

# Predicted commute time if I leave at 8:30AM.
predict(8.5)
```

```
72.8592369314715
```

Regression Line Passes Through the Mean

There's an important property that the regression line satisfies: **for any dataset, the line that minimizes mean squared error passes through the point (mean of x, mean of y).**




```
predict(np.mean(x))
```

```
73.18461538461538
```

```
# Same!
np.mean(y)
```

```
73.18461538461538
```

Our commute times regression line passes through the point $(\bar{x}, \bar{y})$, even if that was not necessarily one of the original points in the dataset.

Intuitively, this says that for an average input, the line that minimizes mean squared error will **always** predict an average output.

Why is this fact true? See if you can reason about it yourself, then check the solution once you've attempted it.

Proof that the regression line passes through $(\bar{x}, \bar{y})$

Our regression line is:

$$h(x_i) = w_0^* + w_1^* x_i$$

We know that the optimal intercept of the regression line is:

$$w_0^* = \bar{y} - w_1^* \bar{x}$$

Substituting this in yields:

$$h(x_i) = \underbrace{\bar{y} - w_1^* \bar{x}}_{w_0^*} + w_1^* x_i$$

If we plug in $x_i = \bar{x}$, we have:

$$h(\bar{x}) = \bar{y} \underbrace{- w_1^* \bar{x} + w_1^* \bar{x}}_{\text{cancels out}} = \bar{y}$$

My interpretation of this is that the intercept is chosen to "vertically adjust" the line so that it passes through $(\bar{x}, \bar{y})$.

The Modeling Recipe

To conclude, let's run through the three-step modeling recipe.

1. Choose a model.

$$h(x_i) = w_0 + w_1 x_i$$

2. Choose a loss function.

We chose squared loss:

$$L_{\text{sq}}(y_i, h(x_i)) = (y_i - h(x_i))^2$$

3. Minimize average loss to find optimal parameters.

For the simple linear regression model, empirical risk is:

$$R_{\text{sq}}(w_0, w_1) = \frac{1}{n} \sum_{i=1}^{n} (y_i - (w_0 + w_1 x_i))^2$$

We showed that:

$$w_1^* = \frac{\sum_{i=1}^{n} (x_i - \bar{x})(y_i - \bar{y})}{\sum_{i=1}^{n} (x_i - \bar{x})^2}, \qquad w_0^* = \bar{y} - w_1^* \bar{x}$$

While the process of minimizing R_{sq} was much, much more complex than in the case of our single parameter model, the conceptual backing of the process was still this three-step recipe, and hopefully now you see its value.

2.4. Correlation

Sometimes, we're not necessarily interested in making *predictions*, but instead want to be *descriptive* about patterns that exist in data.

In a scatter plot of two variables, if there is **any** pattern, we say the variables are **associated**. If the pattern resembles a straight line, we say the variables are **correlated**, i.e. linearly associated. We can measure *how much* a scatter plot resembles a straight line using the correlation coefficient. We'll shortly see how the correlation coefficient relates to the slope of the regression line we found in Chapter 2.3.

Definition

Definition: Correlation Coefficient

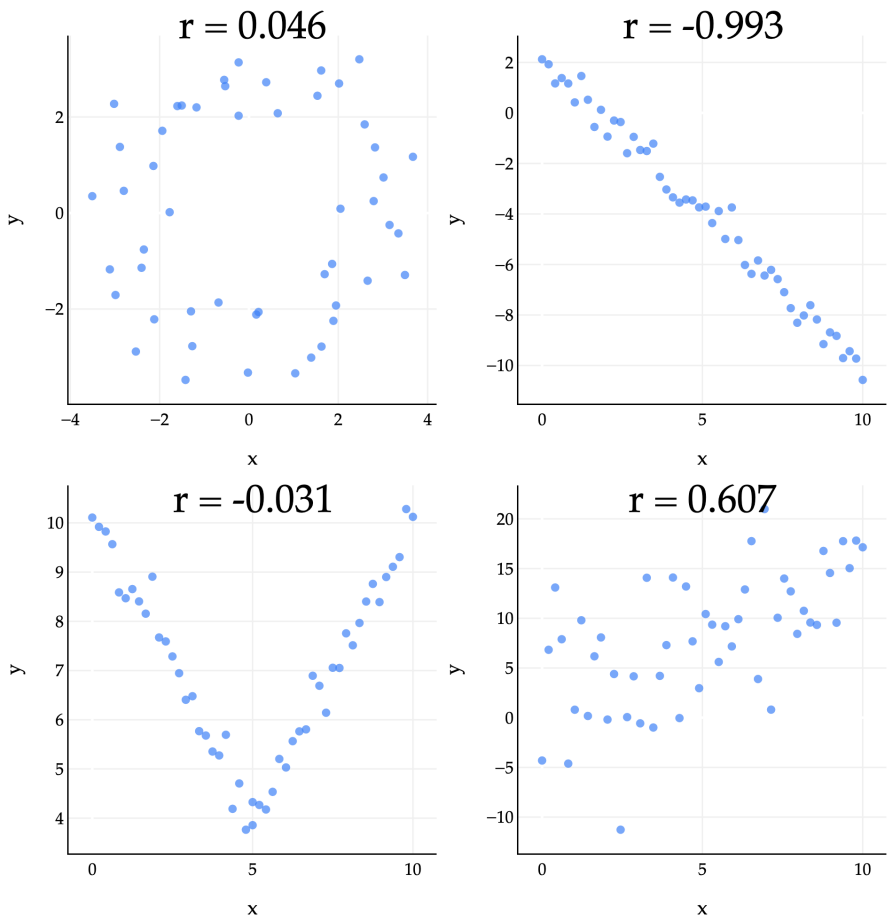
The **correlation coefficient**, denoted r, is defined as:

$$r = \frac{1}{n}\sum_{i=1}^{n}\left(\frac{x_i - \bar{x}}{\sigma_x}\right)\left(\frac{y_i - \bar{y}}{\sigma_y}\right)$$

where $\bar{x}$ and $\bar{y}$ are the means of x and y, respectively, and $\sigma_x = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(x_i - \bar{x})^2}$ and σ_y are the standard deviations of x and y, respectively.
Intuitively, r measures the **strength of the linear association** between x and y.

There are actually many different correlation coefficients; this is the most common one, and it's sometimes called the **Pearson's correlation coefficient**, after the British statistician Karl Pearson.

No matter the values of $x_1, x_2, \ldots x_n$ and $y_1, y_2, \ldots y_n$, **the value of r is bounded between -1 and 1**. The closer $|r|$ is to 1, the stronger the linear association. The sign of r tells us the direction of the trend – upwards (positive) or downwards (negative). r is a unitless quantity – it's not measured in hours, or dollars, or minutes, or anything else that depends on the units of x and y.



The plots above give us some examples of what the correlation coefficient can look like in practice.

- **Top left ($r = 0.046$):** There's some loose circle-like pattern, but it mostly looks like a random cloud of points. $|r|$ is close to 0, but just happens to be positive.
- **Top right ($r = -0.993$):** The points are very tightly clustered around a line with a negative slope, so r is close to -1.
- **Bottom left ($r = -0.031$):** While the points are certainly associated, they are not linearly associated, so the value of r is close to 0. (The shape looks more like a V or parabola than a straight line.)
- **Bottom right ($r = 0.607$):** The points are loosely clustered and follow a roughly linear pattern trending upwards. r is positive, but not particularly large.

The correlation coefficient has some useful properties to be aware of. For one, it's **symmetric**: $r(x, y) = r(y, x)$. If you swap the x_i's and y_i's in its formula, you'll see the result is the same.

$$r = \frac{1}{n}\sum_{i=1}^{n}\left(\frac{x_i - \bar{x}}{\sigma_x}\right)\left(\frac{y_i - \bar{y}}{\sigma_y}\right)$$

One way to think of r is that it's the mean of the product of x and y, once both variables have been **standardized**. To standardize a collection of numbers $x_1, x_2, \ldots x_n$, you first find the mean $\bar{x}$ and standard deviation σ_x of the collection. Then, for each x_i, you compute:

$$z_i = \frac{x_i - \bar{x}}{\sigma_x}$$

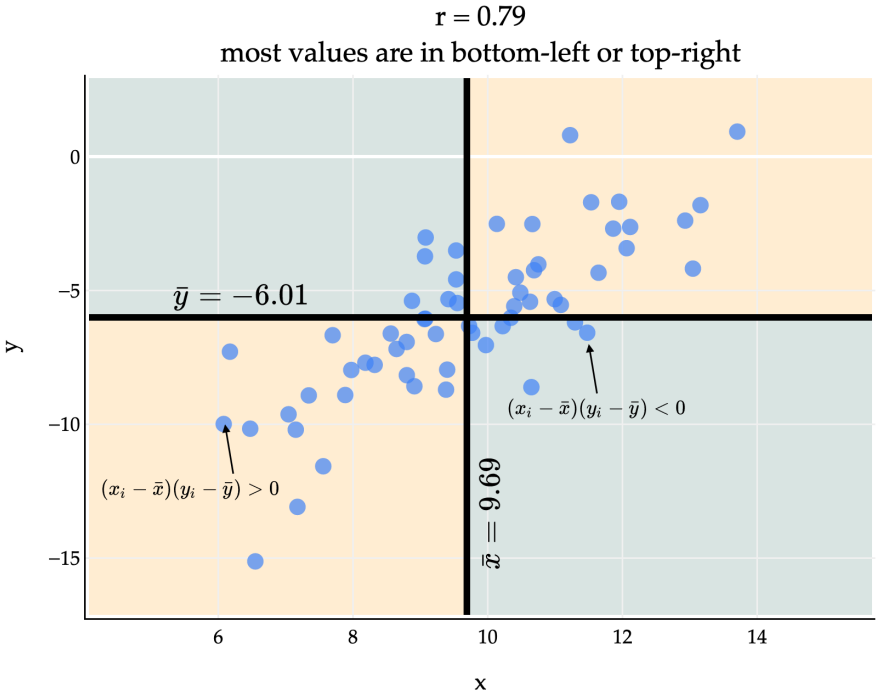
This tells you how many standard deviations away from the mean each x_i is. For example, if $z_i = -1.5$, that means x_i is 1.5 standard deviations below the mean of x. The value of x_i once it's standardized is sometimes called its **z-score**; you *may* have heard of z-scores in the context of curved exam scores.

Intuition. With this in mind, I'll again state that r is the mean of the product of x and y, once both variables have been standardized:

$$r = \frac{1}{n}\sum_{i=1}^{n} \underbrace{\left(\frac{x_i - \bar{x}}{\sigma_x}\right)}_{x_i\text{'s } z\text{-score}} \times \underbrace{\left(\frac{y_i - \bar{y}}{\sigma_y}\right)}_{y_i\text{'s } z\text{-score}}$$

This interpretation of r makes it a bit easier to see *why* r measures the strength of linear association – because up until now, it must seem like a formula I pulled out of thin air.

If there's positive linear association, then x_i and y_i will usually either both be above their averages, or both be below their averages, meaning that $x_i - \bar{x}$ and $y_i - \bar{y}$ will usually have the same sign. If we multiply two numbers with the same sign – either both positive or both negative – then the product will be positive.



Since most points are in the bottom-left and top-right quadrants, most of the products $(x_i - \bar{x})(y_i - \bar{y})$ are positive. This means that r, which is the average of these products divided by the standard deviations of x and y, will be positive too. (We divide by the standard deviations to ensure that $-1 \leq r \leq 1$.)

Above, r is positive but not exactly 1, since there are several points in the bottom-right and top-left quadrants, who would have a negative product $(x_i - \bar{x})(y_i - \bar{y})$ and bring down the average product.

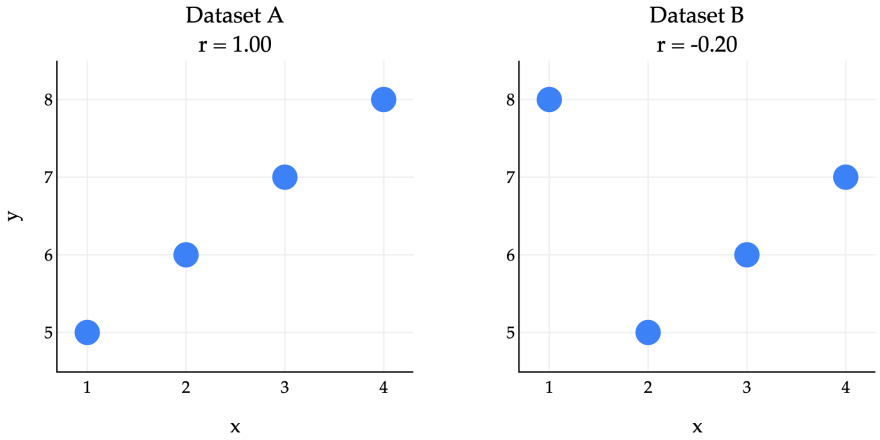
If there's negative linear association, then typically it'll be the case that x_i is above average while y_i is below average, or vice versa. This means that $x_i - \bar{x}$ and $y_i - \bar{y}$ will usually have opposite signs, and when they have opposite signs, their product will be negative. If most points have a negative product, then r will be negative

too.

Another Example. Let me show you another example that gets at the heart of how the correlation formula is defined. Consider the following two datasets.

- Dataset A: $(1, 5), (2, 6), (3, 7), (4, 8)$
- Dataset B: $(1, 8), (2, 5), (3, 6), (4, 7)$

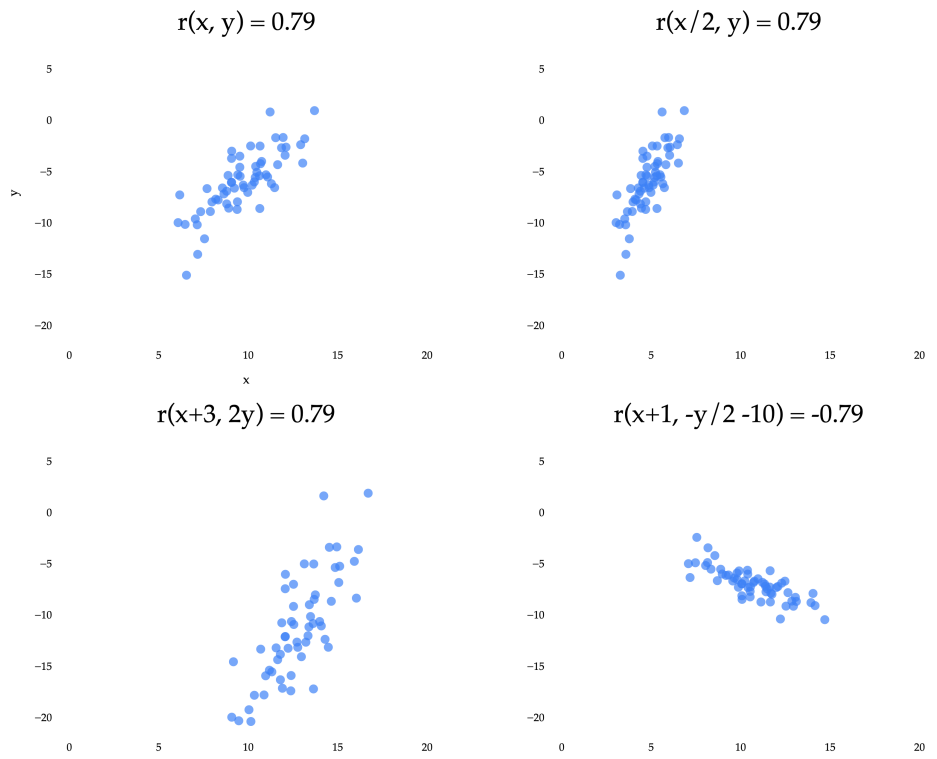
Both datasets use the same x and y values, but the pairings are different, and so their scatter plots look quite different.



In both datasets, $\bar{x}$, $\bar{y}$, σ_x, and σ_y are the same. But, their correlation coefficients are quite different, because the pairings between x_i's and y_i's are different.

Preserving Correlation

Since r measures how closely points cluster around a line, it is invariant to units of measurement. Whether you use feet or meters to measure distance, or dollars or yen to measure price, the correlation between two variables will be the same.



The top left scatter plot is the same as in the previous example, where we reasoned about why r is positive. The other three plots result from applying **linear transformations** to the x and/or y variables independently. A linear transformation of x is any function of the form $ax + b$, and a linear transformation of y is any function of the form $cy + d$. (This is an idea we'll revisit more in Chapter 6.1.)

Notice that three of the four plots have the same r of approximately 0.79. The bottom right plot has an r of approximately -0.79, because the y coordinates were multiplied by a negative constant. What we're seeing is that **the correlation coefficient is invariant to linear transformations of the two variables independently**.

Put in real-world terms: it doesn't matter if you measure commute times in hours, minutes, or seconds, the correlation between departure time and commute time will be the same in all three cases.

Correlation and the Regression Line

Since r measures how closely points cluster around a line, it shouldn't be all that surprising that r has something to do with w_1^*, the slope of the regression line.

It turns out that:

$$w_1^* = \underbrace{\frac{\sum_{i=1}^{n}(x_i - \bar{x})(y_i - \bar{y})}{\sum_{i=1}^{n}(x_i - \bar{x})^2}}_{\text{from earlier}} = \boxed{r\frac{\sigma_y}{\sigma_x}}$$

This is my preferred version of the formula for the optimal slope – it's easy to use and interpret. I've hidden the proof behind a dropdown menu below, but you really should attempt it on your own (and then read it), since it helps build familiarity with how the various components of the formula for r and w_1^* are related.

Proof that $w_1^* = r\frac{\sigma_y}{\sigma_x}$

First, let's show that we can express $\sum_{i=1}^n (x_i - \bar{x})^2$ in terms of σ_x:

$$\sigma_x = \sqrt{\frac{1}{n}\sum_{i=1}^n (x_i - \bar{x})^2}$$

$$\sigma_x^2 = \frac{1}{n}\sum_{i=1}^n (x_i - \bar{x})^2$$

$$n\sigma_x^2 = \sum_{i=1}^n (x_i - \bar{x})^2$$

Now, let's prove that $w_1^* = r\frac{\sigma_y}{\sigma_x}$:

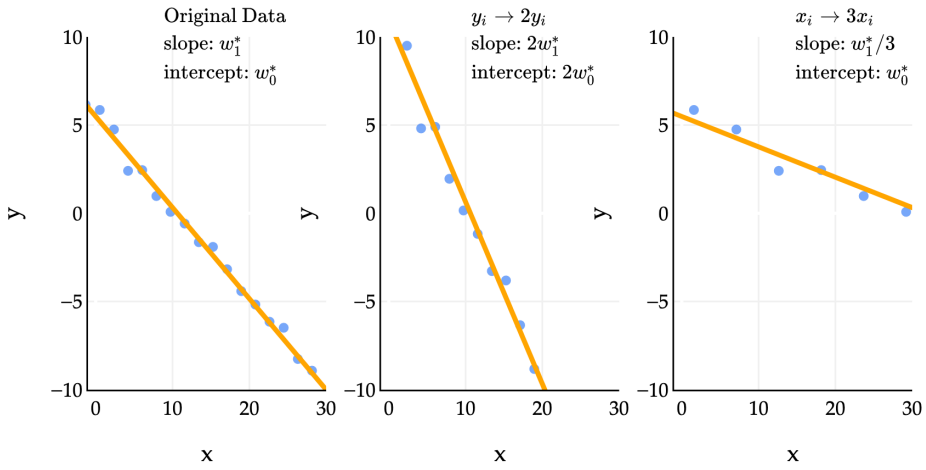
$$\begin{aligned}
r\frac{\sigma_y}{\sigma_x} &= \frac{1}{n}\sum_{i=1}^n \left(\frac{x_i - \bar{x}}{\sigma_x}\right)\left(\frac{y_i - \bar{y}}{\sigma_y}\right) \cdot \frac{\sigma_y}{\sigma_x} \\
&= \frac{1}{n}\sum_{i=1}^n \frac{(x_i - \bar{x})(y_i - \bar{y})}{\sigma_x^2} \\
&= \frac{\sum_{i=1}^n (x_i - \bar{x})(y_i - \bar{y})}{n\sigma_x^2} \\
&= \frac{\sum_{i=1}^n (x_i - \bar{x})(y_i - \bar{y})}{\sum_{i=1}^n (x_i - \bar{x})^2} \\
&= w_1^*
\end{aligned}$$

The simpler formula above implies that the sign of the slope is the same as the sign of r, which seems reasonable: if the direction of the linear association is negative, the best-fitting slope should be, too.

So, all in one place:

$$w_1^* = \frac{\sum_{i=1}^n (x_i - \bar{x})(y_i - \bar{y})}{\sum_{i=1}^n (x_i - \bar{x})^2} = r\frac{\sigma_y}{\sigma_x}, \qquad w_0^* = \bar{y} - w_1^*\bar{x}$$

This new formula for the slope also gives us insight into how the spread of x (σ_x) and y (σ_y) affects the slope. If y is more spread out than x, the points on the scatter plot will be stretched out vertically, which will make the best-fitting slope steeper.



In the middle example above, $y_i \to 2y_i$ means that we replaced each y_i in the dataset with $2y_i$. In that example, the slope and intercept of the regression line both doubled. In the third example, where we replaced each x_i with $3x_i$, the slope was divided by 3, while the intercept remained. One of the problems in Homework 2 has you prove these sorts of results, and you can do so by relying on the formula for w_1^* that involves r; **note that all three datasets above have the same** r.

Activity 1

Activity 1

This activity is an old exam question, taken from an exam that used to allow calculators. Part of it also appears in Lab 3.

1. First, suppose we minimize mean squared error to fit a simple linear regression line that **uses the square footage of a house to predict its price**. The resulting line has an intercept of w_0^* and a slope of w_1^*. In other words:

$$\text{predicted price}_i = w_0^* + w_1^* \cdot \text{square footage}_i$$

   We're now interested in minimizing mean squared error to find a simple linear regression line that uses **price to predict square footage**. Suppose this new regression line has an intercept of β_0^* and a slope of β_1^*.

   What is β_1^*? Give your answer as an expression in terms of n, r, w_0^*, and/or w_1^*.

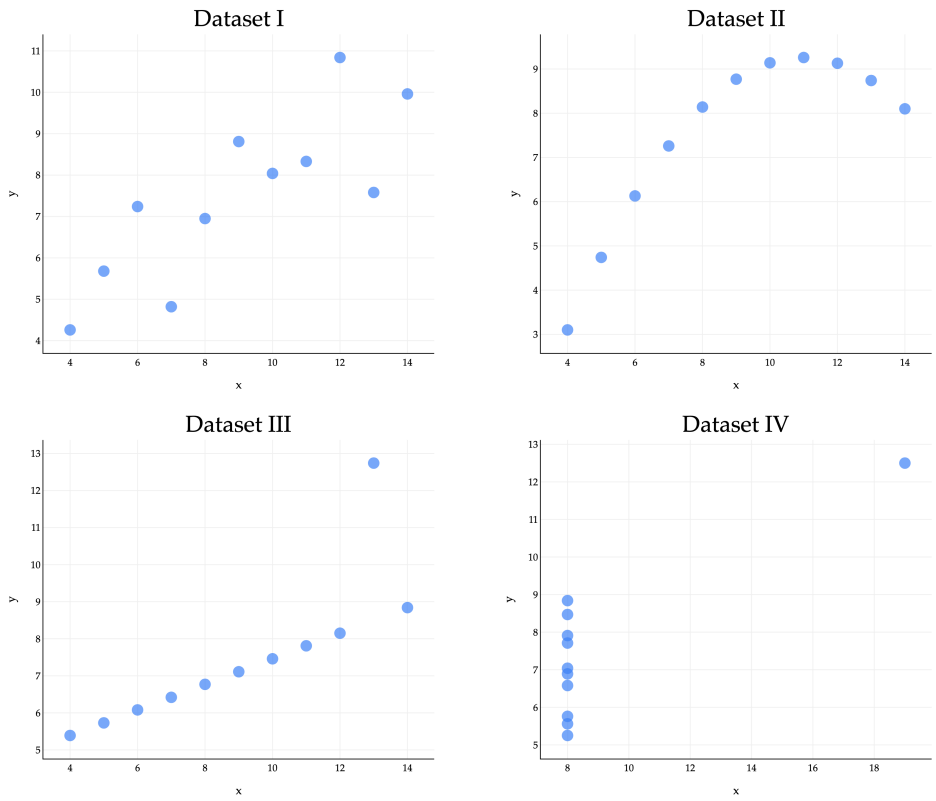
2. Given that:

- $n = 100$
- $r = 0.6$
- $w_0^* = 1000$
- $w_1^* = 250$ |
- The average square footage of houses in the dataset is 2000

What is β_0^*? Your answer should be a constant with no variables. (Once you're able to express β_0^* in terms of constants only, you can stop simplifying your answer.)

Example: Anscombe's Quartet. The correlation coefficient is just one number that describes the linear association between two variables; it doesn't tell us *everything*.

Consider the famous example of Anscombe's quartet, which consists of four datasets that all have the same mean, standard deviation, and correlation coefficient, but look very different.

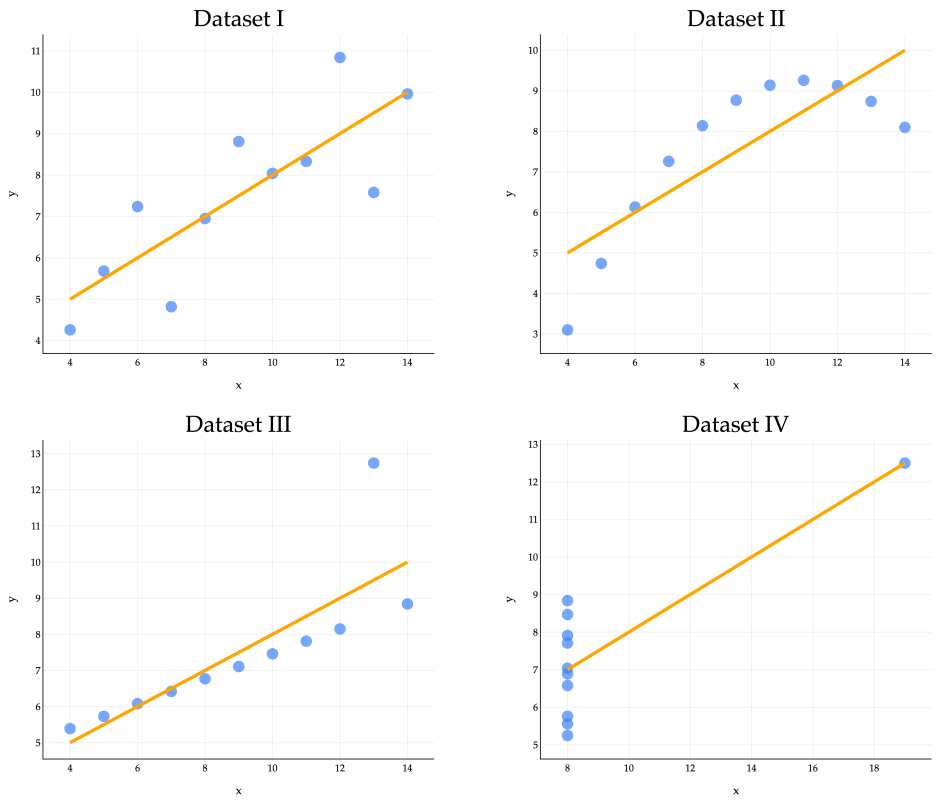


In all four datasets:

$$\bar{x} = 9, \bar{y} = 7.5, \sigma_x = 3.16, \sigma_y = 1.94, r = 0.82$$

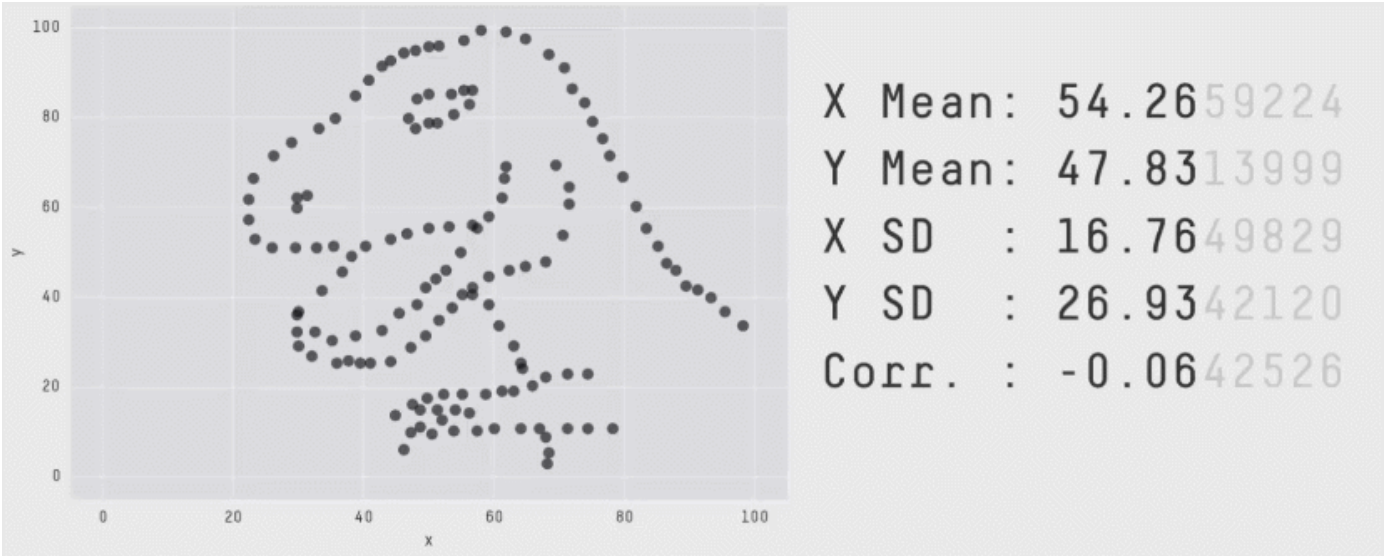
Because they all share the same five values of these key quantities, they also share the same regression lines, since the optimal slope and intercept are determined just using those five quantities.

$$w_1^* = r\frac{\sigma_y}{\sigma_x} = 0.82\frac{1.94}{3.16} = 0.52 \qquad w_0^* = \bar{y} - w_1^*\bar{x} = 7.5 - 0.52 \cdot 9 = 2.82$$



The regression line clearly looks better for some datasets than others, with Dataset IV looking particularly off. A high $|r|$ may be evidence of a strong linear association, but it cannot guarantee that a linear model is suitable for a dataset. Moral of the story - **visualize your data before trying to fit a model!** Don't *just* trust the numbers.

You might like the Datasaurus Dozen, another similar collection of 13 datasets that all have the same mean, standard deviation, and correlation coefficient, but look very different. (One looks like a dinosaur!)



2.5. Least Squares

The Modeling Recipe

We've now made several passes over the three-step modeling recipe, first introduced in Chapter 1.3.

1. **Choose a model.**
2. **Choose a loss function.**
3. **Minimize average loss to find optimal parameters.**

Our very first pass through the recipe involved:

1. The constant model, $h(x_i) = w$.
2. Squared loss, $L_{\text{sq}}(y_i, h(x_i)) = (y_i - h(x_i))^2$.
3. The above two choices gave us average loss, or empirical risk, of

$$R_{\text{sq}}(w) = \frac{1}{n}\sum_{i=1}^{n}(y_i - w)^2$$

   which we showed is minimized by $w^* = \bar{y}$.

Then, in Chapter 2.3, we considered:

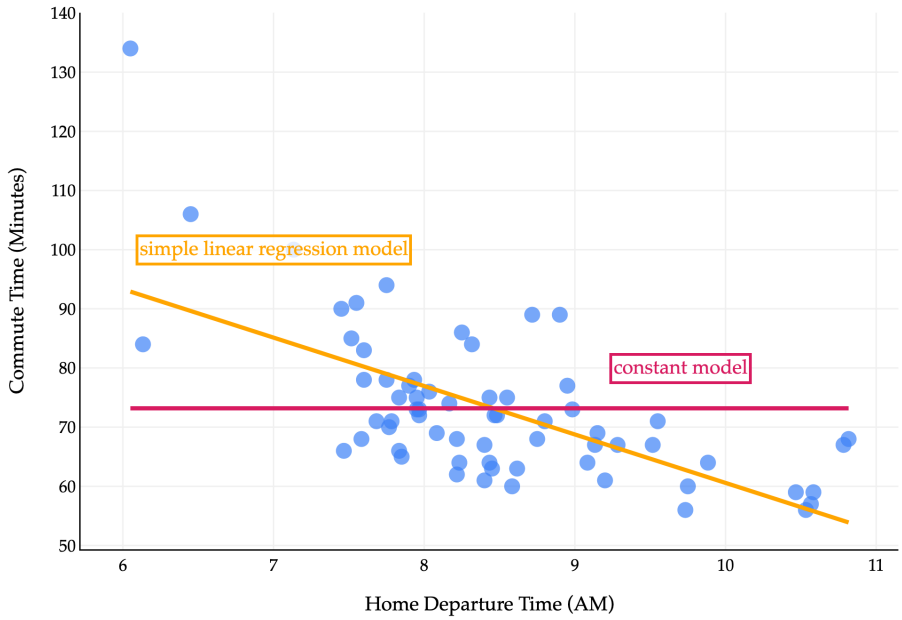
1. The simple linear regression model, $h(x_i) = w_0 + w_1 x_i$.
2. Squared loss, $L_{\text{sq}}(y_i, h(x_i)) = (y_i - h(x_i))^2$.
3. The above two choices gave us average loss, or empirical risk, of

$$R_{\text{sq}}(w_0, w_1) = \frac{1}{n}\sum_{i=1}^{n}(y_i - (w_0 + w_1 x_i))^2$$

   which we showed – using partial derivatives – is minimized by

$$w_1^* = \frac{\sum_{i=1}^{n}(x_i - \bar{x})(y_i - \bar{y})}{\sum_{i=1}^{n}(x_i - \bar{x})^2} = r\frac{\sigma_y}{\sigma_x}, \qquad w_0^* = \bar{y} - w_1^*\bar{x}$$

Let's visualize both the best constant model and the best simple linear regression model on the commute times dataset.



Activity 1

Activity 1

What are the coordinates of the point of intersection of the two lines above?

A fact we mentioned earlier – and you restated in Homework 1 – is that the mean squared error of the best constant model is the variance of the data. We can see this by substituting $w^* = \bar{y}$ into mean squared error:

$$R_{\text{sq}}(w) = \frac{1}{n}\sum_{i=1}^{n}(y_i - h)^2 \implies R_{\text{sq}}(\bar{y}) = \underbrace{\frac{1}{n}\sum_{i=1}^{n}(y_i - \bar{y})^2}_{\text{variance of } y} = \sigma_y^2$$

This gives us a nice baseline from which to compare the performance of more sophisticated models. **If a model we create has a mean squared error that's higher than the variance of the data, we know we've done something wrong, because a constant model would be better!**

The mean squared error of the best simple linear regression model on the data is:

$$R_{\text{sq}}(w_0^*, w_1^*) = \frac{1}{n}\sum_{i=1}^{n}(y_i - (w_0^* + w_1^* x_i))^2$$

This formula alone isn't particularly enlightening, but it turns out this is equivalent to $\sigma_y^2(1 - r^2)$, where r is the correlation coefficient between x and y. (The proof of this is an optional component of Homework 2.)

This means that the best simple linear regression model **can't** perform worse than the best constant model. If $r = 0$, then the best simple linear regression model has a slope of 0, and is just the best constant model! Otherwise, the best simple linear regression model will have a non-zero slope, which is chosen to minimize mean squared error.

In this particular dataset:

MSE of best constant model = $\sigma_y^2 = 167.54$ minutes^{2}MSE of best simple linear regression model = $\sigma_y^2 \underbrace{(1 - r^2)}_{\text{here, } r=-0.65} = 97.05$

```
# Assume x and y are arrays containing
# departure hours and commute times, respectively.
np.var(y, ddof=0)
```

```
167.535147928994
```

```
np.corrcoef(x, y)[0, 1]
```

```
-0.6486426165832004
```

```
np.var(y, ddof=0) * (1 - np.corrcoef(x, y)[0, 1]**2)
```

```
97.04687150819171
```

Least Squares

Least squares is the general term for minimizing mean squared error – or equivalently, the sum of squared errors – to find optimal model parameters. Least squares appears quite frequently outside of machine learning contexts, too.

Activity 2

Activity 2

I glossed over something important above. I said that minimizing mean squared error is equivalent to minimizing the sum of squared errors.
Why is that true? Let's consider a particular example, of finding w_0^* and w_1^* for the simple linear regression model $h(x_i) = w_0 + w_1 x_i$.
Why are the w_0^* and w_1^* that minimize

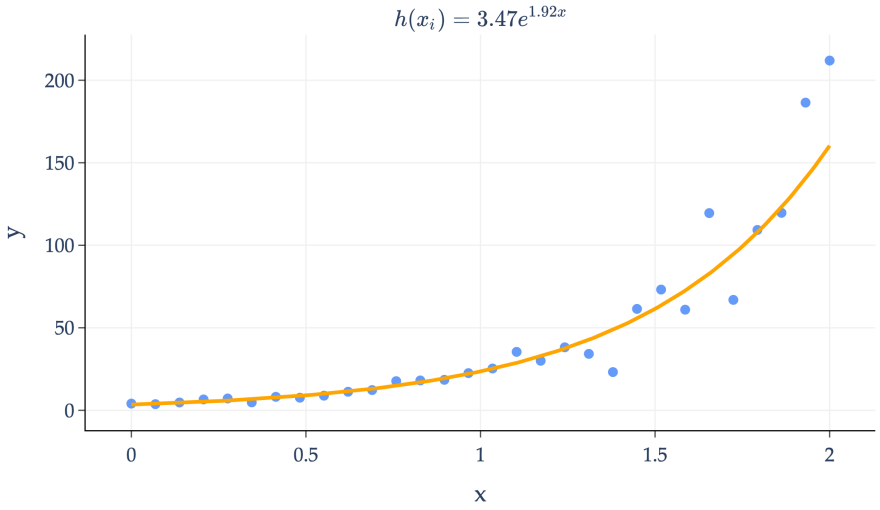
$$R_{\text{sq}}(w_0, w_1) = \frac{1}{n} \sum_{i=1}^{n} (y_i - (w_0 + w_1 x_i))^2$$

the same as the w_0^* and w_1^* that minimize

$$S_{\text{sq}}(w_0, w_1) = \sum_{i=1}^{n} (y_i - (w_0 + w_1 x_i))^2$$

This idea was also reinforced in Lab 1, and in Activity 3 of Appendix 2.

For example, population ecologists use least squares to model the growth of populations (say, of an animal species) over time.



The orange curve above is of the form $h(x_i) = ae^{bx_i}$, where a and b are parameters. I've chosen the form ae^{bx_i} because it's a good fit for the data; it's how we model exponential growth.

a and b **could** be chosen by minimizing mean squared error directly:

$$R_{\text{sq}}(a, b) = \frac{1}{n}\sum_{i=1}^{n}(y_i - ae^{bx_i})^2$$

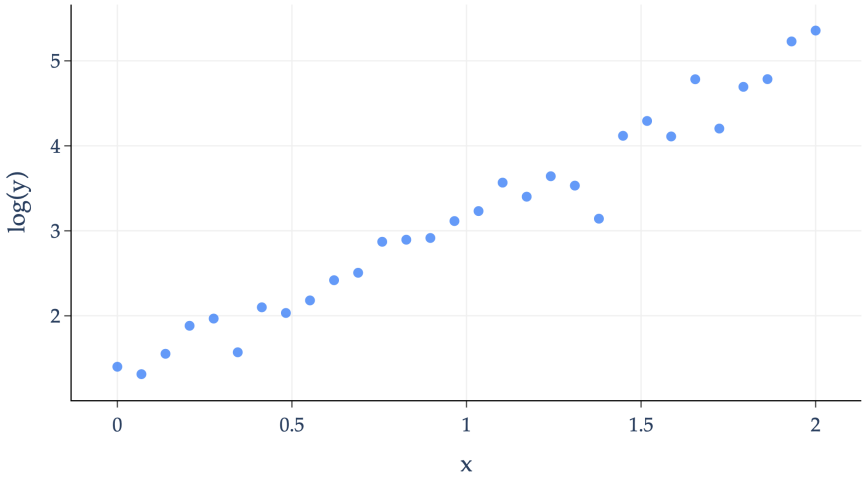
If we wanted to minimize $R_{\text{sq}}(a, b)$, we'd need to take partial derivatives with respect to a and b, set them to 0, and solve the resulting system of equations. However, the function $h(x_i) = ae^{bx_i}$ involves a parameter (b) in the exponent, which means the resulting partial derivatives of R_{sq} will be complicated and the resulting system of equations will be challenging to solve. (Don't believe me? Try finding $\frac{\partial R_{\text{sq}}}{\partial b}$.)

In this particular case, there's a transformation we can apply to the data so that we can reduce the minimization problem to something we've seen before! Consider what happens if we take the logarithm of both sides of the equation $y_i \approx ae^{bx_i}$:

$$y_i \approx ae^{bx_i} \implies \log(y_i) \approx \log(a) + bx_i$$

I've written the $\approx$ symbol to remind us that y_i is not exactly equal to ae^{bx_i}, but rather, the goal is to choose a and b so that ae^{bx_i} is approximately equal to y_i, across all points i.

When we take the logarithm of each y-value in our dataset, the resulting plot looks like:



The transformed data looks roughly linear!

How does this transformation help us find a and b? Now, instead of minimizing:

$$R_{\text{sq}}(a, b) = \frac{1}{n} \sum_{i=1}^{n} (y_i - ae^{bx_i})^2$$

I'm claiming that since $\log(y_i) \approx \log(a) + bx_i$, we can instead minimize:

$$R'_{\text{sq}}(a, b) = \frac{1}{n} \sum_{i=1}^{n} (\log(y_i) - (\underbrace{\log(a)}_{w_0} + \underbrace{b}_{w_1} x_i))^2$$

R'_{sq} is not the derivative of R_{sq}; it is a different empirical risk function, with different optimal parameters.

That said, R'_{sq} should look familiar: it looks remarkably similar to the mean squared error of the simple linear regression model.

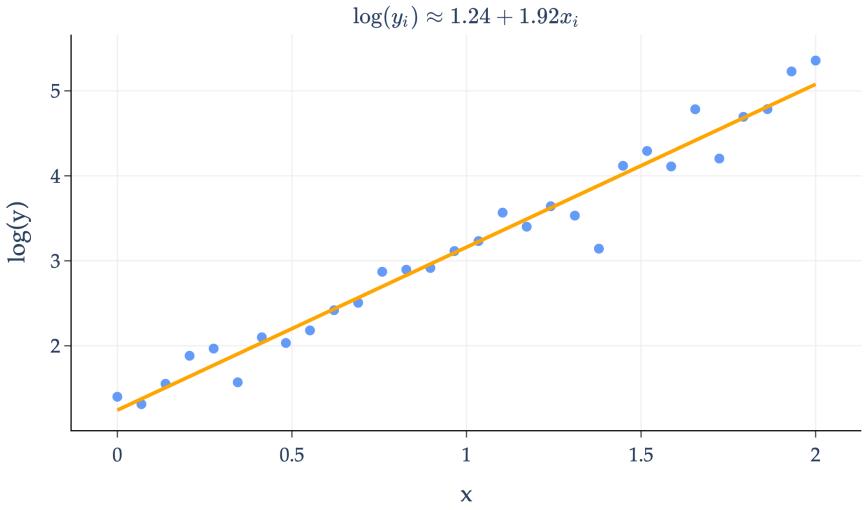
Let's define $z_i = \log(y_i)$, $w_0 = \log(a)$, and $w_1 = b$. Then, we can rewrite R'_{sq} as:

$$R'_{\text{sq}}(w_0, w_1) = \frac{1}{n} \sum_{i=1}^{n} (z_i - (w_0 + w_1 x_i))^2$$

We just spent all of Chapter 2.3 finding the optimal values of w_0^* and w_1^* in the empirical risk function above, so the optimal values of w_0^* and w_1^* are well known to us!

$$w_1^* = \frac{\sum_{i=1}^{n} (x_i - \bar{x})(z_i - \bar{z})}{\sum_{i=1}^{n} (x_i - \bar{x})^2}, \qquad w_0^* = \bar{z} - w_1^* \bar{x}$$

Note that I've computed $w_0^* = 1.24$ and $w_1^* = 1.92$ using the transformed data, $(x_1, z_1), \ldots, (x_n, z_n)$. On the transformed data, this line looks like:



How do we turn this back into a curve on the original data? To transform the data, we applied the following substitutions that we need to invert:

$$w_0 = \log(a) \implies a = e^{w_0} b = w_1$$

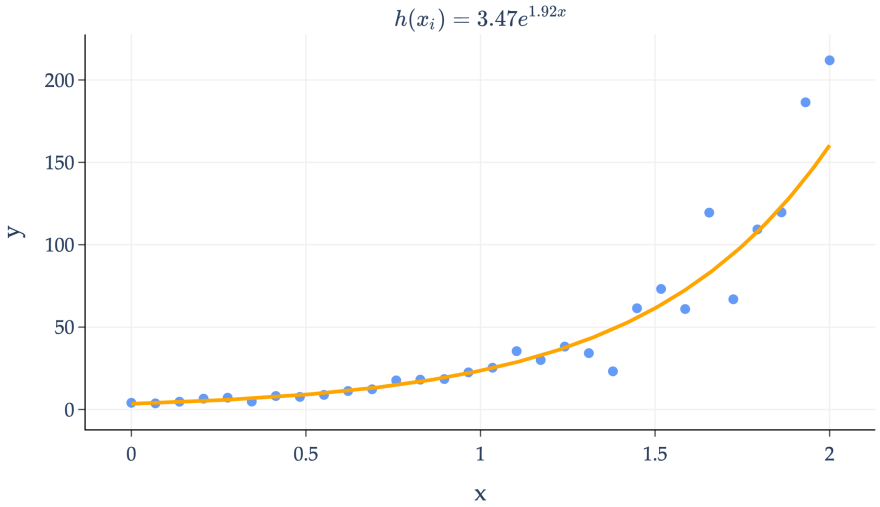
So, using the above, we can find:

$$a^* = e^{w_0^*} = e^{1.24} = 3.46 b^* = w_1^* = 1.92$$

So, the curve we're looking for is:

$$h(x_i) = 3.46 e^{1.92 x_i}$$

And this is the equation of the orange curve I first showed you!



Not every least squares problem can be reduced to the simple linear regression problem we've been studying. But, in the rare cases that it can, it's satisfying to be able to reuse our work.

What's Next?

Following Chapter 2.3, the next natural question to ask is, how do we use **multiple features** (input variables) to make predictions?

When we first introduced the commute times example back in Chapter 1.2, we saw that the dataset we're working with has a few features beyond the departure hour.

	date	day	departure_hour	minutes
0	5/15/2023	Mon	10.816667	68.0
1	5/16/2023	Tue	7.750000	94.0
2	5/22/2023	Mon	8.450000	63.0
3	5/23/2023	Tue	7.133333	100.0
4	5/30/2023	Tue	9.150000	69.0

Up until now, we've been using the `departure_hour` feature to predict the `minutes` column.

It would be great if we could figure out a way to use the day of the week as a feature, since our intuition tells us this might have an effect on commute times (traffic is known to be bad on Monday mornings and Friday afternoons). But, the `day` column currently contains strings, not numbers, so we'll have to think critically about how to encode this data numerically. (You'll see how to do this in a later homework.)

For now, let's extract the day of the **month** from the `date` column, and try and use `day_of_month` along with `departure_hour` as our two features.

	departure_hour	day_of_month	minutes
0	10.816667	15	68.0
1	7.750000	16	94.0
2	8.450000	22	63.0
3	7.133333	23	100.0
4	9.150000	30	69.0

We want to use the first two columns to predict the third. What would the hypothesis function for such a model look like? Previously, we had:

$$\underbrace{h(\text{departure hour}_i)}_{\text{predicted commute time } i} = w_0 + w_1 \cdot \text{departure hour}_i$$

Now, a linear model that involves **both** features would look like:

$$h(\text{departure hour}_i, \text{dom}_i) = w_0 + w_1 \cdot \text{departure hour}_i + w_2 \cdot \text{dom}_i$$

This is called a **multiple linear regression** model, since it uses multiple features. (To keep the equation short, I've used dom to refer to the day of the month.)

The first hypothesis function, with one input variable, looked like a line in two dimensions. Our new multiple linear regression model looks like a **plane in three dimensions!**

In two dimensional space, a line has a constant rate of change, described by its slope.

In three dimensional space, a plane has a constant rate of change **along both the x and y axes**. The plane's rate of change in the x (departure time) direction is given by w_1^*, and its rate of change in the y (day of month) direction is given by w_2^*.

Finding the optimal values of w_0^*, w_1^*, and w_2^* involves minimizing, yet again, mean squared error:

$$R_{\text{sq}}(w_0, w_1, w_2) = \frac{1}{n} \sum_{i=1}^{n} \left(y_i - (w_0 + w_1 \cdot \text{departure hour}_i + w_2 \cdot \text{dom}_i)\right)^2$$

Mean squared error here is a function of three variables, so we'd need four dimensions to visualize it, which, of course, we cannot do. To minimize R_{sq} above, we'd need to find three partial derivatives, set them all to 0, and solve the resulting system of 3 equations and 3 unknowns.

While we *could* do this, eventually, we'll want to consider a third feature, a fourth, and so on. In general, if we want d features, we'll have $d + 1$ parameters to optimize (one per feature, plus one for the intercept, w_0), and we'll need to solve a system of $d + 1$ equations and $d + 1$ unknowns to find them. Continuing this approach sounds painful.

This is where linear algebra comes in. Linear algebra provides a compact, efficient way to represent and manipulate systems of linear equations, and to find solutions to them.

Chapters 3 through 6 are dedicated to introducing the ideas from linear algebra that we'll need to solve this problem: **the problem of finding the parameters that minimize mean squared error for the multiple linear regression model**. It will take some time to get there, and at times, it'll seem like what we're doing is unrelated to this goal of finding optimal parameters.

But, I promise that we will close the loop in Chapter 7, when we return to multiple linear regression.

Vectors

3.1. Vectors and Linear Combinations

Linear algebra can be thought of as the study of vectors, matrices, and **linear** transformations, all of which are ideas we'll need to use in our journey to understand machine learning. We'll start with vectors, which are the building blocks of linear algebra.

Definition

There are many ways to define vectors, but I'll give you the most basic and practically relevant definition of a vector for now. I'll introduce more abstract definitions later if we need them.

Definition: Vector

A **vector** is an ordered list of numbers.
In this class, we'll typically use lowercase letters to denote vectors, drawn with arrows above the letters. For example:

$$\vec{v} = \begin{bmatrix} 4 \\ -3 \\ 15 \end{bmatrix}$$

In other classes or textbooks, you might see vectors written as boldface letters, like $\mathbf{v}$, or simply v.

By *ordered list*, I mean that the order of the numbers in the vector matters.

- For example, the vector $\vec{v} = \begin{bmatrix} 4 \\ -3 \\ 15 \end{bmatrix}$ is not the same as the vector $\vec{w} = \begin{bmatrix} 15 \\ -3 \\ 4 \end{bmatrix}$, even though they have the same components.
- $\vec{v}$ is also different from the vector $\vec{u} = \begin{bmatrix} 4 \\ -3 \\ 15 \\ 1 \end{bmatrix}$, even though their first three components are the same.

In general, we're mostly concerned with vectors in $\mathbb{R}^n$, which is the **set** of all vectors with n **components** or **elements**, each of which is a real number. It's possible to consider vectors with complex components (the set of all vectors with complex components is denoted $\mathbb{C}^n$), but we'll stick to real vectors for now.

The vector $\vec{v}$ defined in the box above is in $\mathbb{R}^3$, which we can express as $\vec{v} \in \mathbb{R}^3$. This is pronounced as "v is an element of R three", or "v is in R three".

A general vector in $\mathbb{R}^n$ can be expressed in terms of its n components:

$$\vec{v} = \begin{bmatrix} v_1 \\ v_2 \\ \vdots \\ v_n \end{bmatrix}$$

Subscripts can be used for different, sometimes conflicting purposes:

- In the definition of $\vec{v}$ above, the components of the vector are denoted $v_1, v_2, \ldots, v_n$. Each of these individual components is a single real number – known as a **scalar** – not a vector.
- But in the near future, we may want to consider multiple vectors at once, and may use subscripts to refer to them as well. For instance, I might have d different vectors, $\vec{v}_1, \vec{v}_2, \ldots, \vec{v}_d$, each corresponding to some

feature I care about.

The meaning of the subscript depends on the context, so just be careful!

While we'll use the definition of a vector as a list of numbers for now, I hope you'll soon appreciate that vectors are more than just a list of numbers – they encode remarkable amounts of information and beauty.

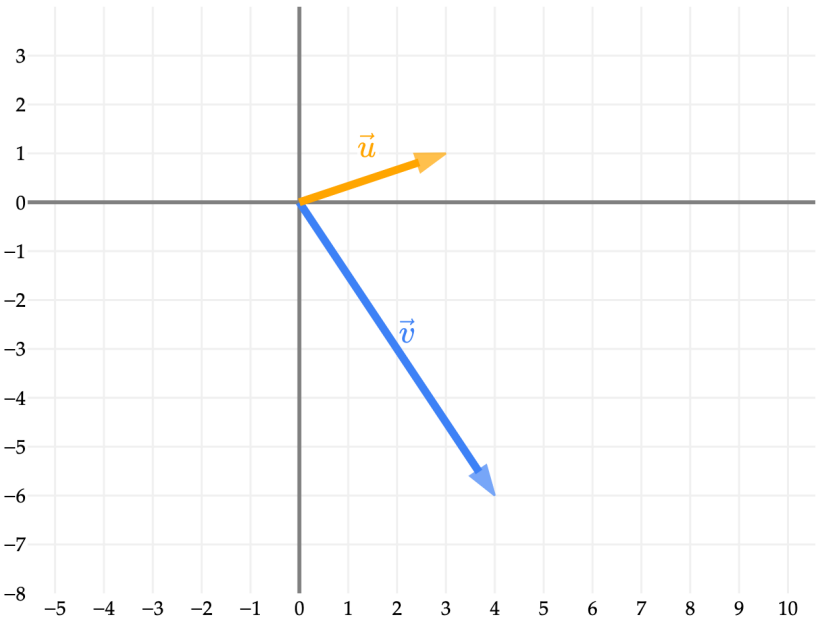
Norm (i.e. Length or Magnitude)

In the context of physics, vectors are often described as creatures with "a magnitude and a direction". While this is not a physics class – this is EECS 245, after all! – this interpretation has some value for us too.

To illustrate what we mean, let's consider some concrete vectors in $\mathbb{R}^2$, since it is easy to visualize vectors in 2 dimensions on a computer screen. Suppose:

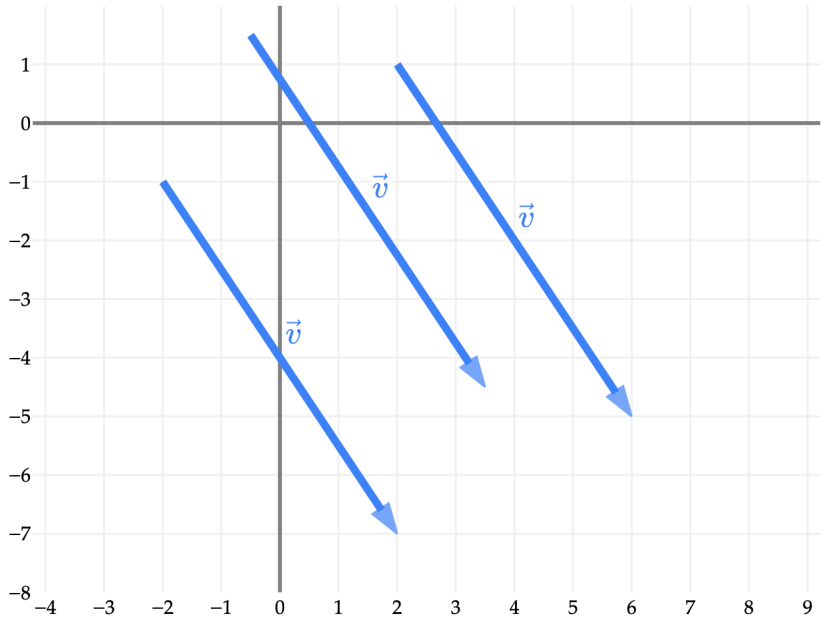
$$\vec{u} = \begin{bmatrix} 3 \\ 1 \end{bmatrix}, \quad \vec{v} = \begin{bmatrix} 4 \\ -6 \end{bmatrix}$$

Then, we can visualize $\vec{u}$ and $\vec{v}$ as arrows pointing from the origin $(0, 0)$ to the points $(3, 1)$ and $(4, -6)$ in the two dimensional Cartesian plane, respectively.

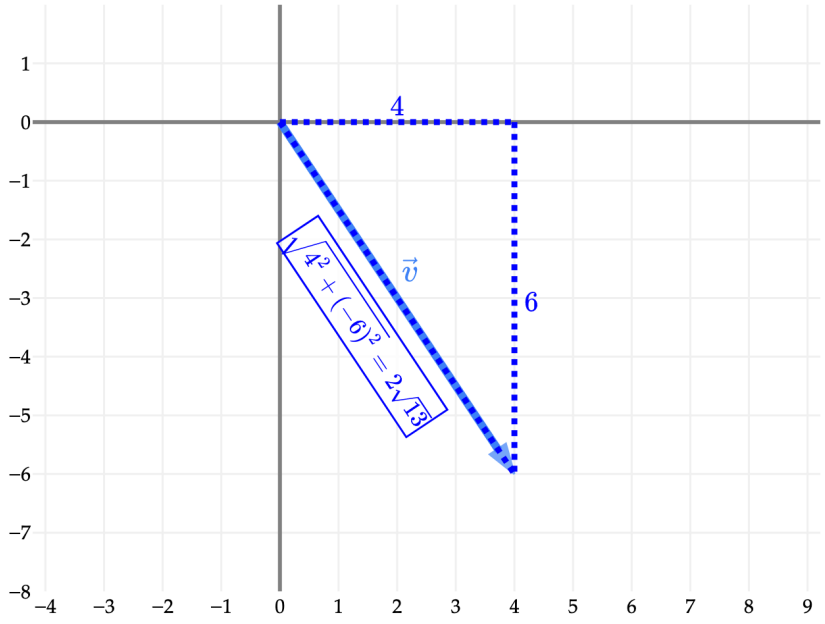


The vector $\vec{v} = \begin{bmatrix} 4 \\ -6 \end{bmatrix}$ moves 4 units to the right and 6 units down, which we know by reading the components of the vector. In Chapter 7.3, we'll see how to describe the direction of $\vec{v}$ in terms of the angle it makes with the x-axis (and you may remember how to calculate that angle using trigonometry).

It's worth noting that $\vec{v}$ isn't "fixed" to start at the origin – vectors don't have fixed positions. All three vectors in the figure below are the same vector, $\vec{v}$.



To compute the length of $\vec{v}$ – i.e. the distance between $(0, 0)$ and $(4, -6)$ – we should remember the Pythagorean theorem, which states that if we have a right triangle with legs of length a and b, then the length of the hypotenuse is $\sqrt{a^2 + b^2}$. Here, that's $\sqrt{4^2 + (-6)^2} = \sqrt{16 + 36} = \sqrt{52} = 2\sqrt{13}$.



Definition: Vector Norm

The **norm** of a vector $\vec{v} \in \mathbb{R}^n$ is defined as follows:

$$\|\vec{v}\| = \sqrt{v_1^2 + v_2^2 + \cdots + v_n^2} = \sqrt{\sum_{i=1}^{n} v_i^2}$$

The norm of a vector is also called its **length** or **magnitude**. This particular formula for the norm is also called the L_2 **norm** or **Euclidean norm** (after the Greek mathematician Euclid), and is the most common and "default" norm used in linear algebra.

Note that the norm involves a sum of *squares*, much like mean *squared* error . This connection will be made more explicit in Chapter 7, when we return to studying linear regression.

Shortly, we'll see other norms, which describe different ways of measuring the "length" of a vector.

Activity 1

Activity 1

As we'll see later in this section, a **unit vector** is a vector with norm 1. It's common to use unit vectors to describe directions. For instance, there are infinitely many vectors in $\mathbb{R}^2$ that point in the same direction as $\vec{v} = \begin{bmatrix} 4 \\ -6 \end{bmatrix}$ from above, like $\begin{bmatrix} 2 \\ -3 \end{bmatrix}$ and $\begin{bmatrix} 40 \\ -60 \end{bmatrix}$. (If you don't believe me, draw it out!)

Find a unit vector that points in the same direction as the vector $\vec{x} = \begin{bmatrix} 12 \\ 5 \end{bmatrix}$, and verify that it has norm 1.

Technically, to answer this, you'll need to use the fact that vectors can be multiplied by a scalar, which we haven't yet discussed, but see how far your intuition takes you!

What may not be immediately obvious is *why* the Pythagorean theorem seems to extend to higher dimensions. The two dimensional case seems reasonable, but why is the length of the vector $\vec{w} = \begin{bmatrix} 6 \\ -2 \\ 3 \end{bmatrix}$ in $\mathbb{R}^3$ equal to $\sqrt{6^2 + (-2)^2 + 3^2}$?

There are two right angle triangles in the picture above:

- One triangle has legs of length 6 and 2, with a hypotenuse of h; this triangle is shaded light blue above.
- Another triangle has legs of length 3 and h, with a hypotenuse of $\|\vec{w}\|$; this triangle is shaded dark pink above.

To find $\|\vec{w}\|$, we can use the Pythagorean theorem twice:

$$h^2 = 6^2 + (-2)^2 = 36 + 4 = 40 \implies h = \sqrt{40}$$

Then, we can use the Pythagorean theorem again to find $\|\vec{w}\|$:

$$\|\vec{w}\| = \sqrt{h^2 + 3^2} = \sqrt{40 + 9} = 7 = \sqrt{6^2 + (-2)^2 + 3^2}$$

So, to find $\|\vec{w}\|$, we used the Pythagorean theorem twice, and ended up computing the square root of the sum of the squares of the components of the vector, which is what the definition above states.

This argument naturally extends to higher dimensions. We will do this often: build intuition in the dimensions we can visualize (two dimensions, and with the help of interactive graphics, three dimensions), and then rely on the power of abstraction to extend our understanding to higher dimensions, even when we can't visualize. **Thinking in higher dimensions** is one of the key objectives of this course.

Length ≠ Number of Components

Don't confuse the length of a vector with the number of components in a vector!
If $\vec{v} \in \mathbb{R}^n$, then the length of $\vec{v}$ is $\|\vec{v}\| = \sqrt{v_1^2 + v_2^2 + \cdots + v_n^2}$, while the number of components in $\vec{v}$ is n.

Vector norms satisfy several interesting properties, which we will introduce shortly once we have more context.

Addition and Scalar Multiplication

Vectors support two core operations: addition and scalar multiplication. These two operations are core to the study of linear algebra – so much so, that sometimes vectors are defined abstractly as "things that can be added and multiplied by scalars".

Addition.

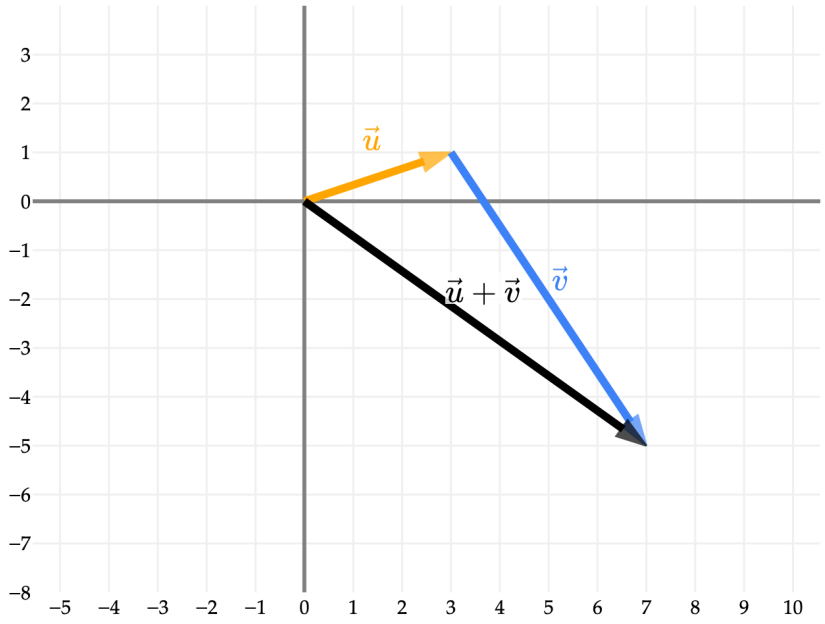
Definition: Vector Addition

Suppose $\vec{u}, \vec{v} \in \mathbb{R}^n$, i.e. both are vectors with n components.
Then, the **sum** of $\vec{u}$ and $\vec{v}$ is defined as follows:

$$\vec{u} + \vec{v} = \begin{bmatrix} u_1 \\ u_2 \\ \vdots \\ u_n \end{bmatrix} + \begin{bmatrix} v_1 \\ v_2 \\ \vdots \\ v_n \end{bmatrix} = \begin{bmatrix} u_1 + v_1 \\ u_2 + v_2 \\ \vdots \\ u_n + v_n \end{bmatrix}$$

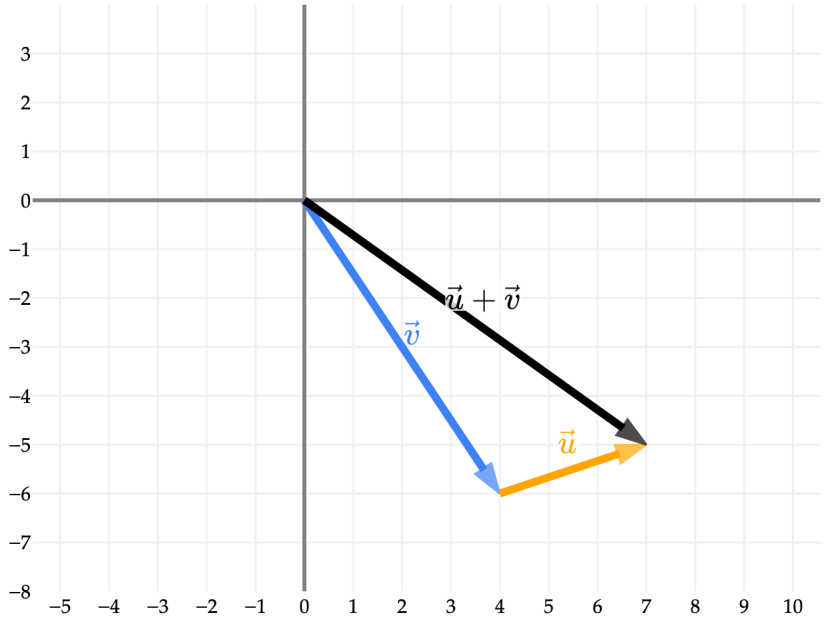
Using our examples from earlier, $\vec{u} = \begin{bmatrix} 3 \\ 1 \end{bmatrix}$ and $\vec{v} = \begin{bmatrix} 4 \\ -6 \end{bmatrix}$, we have that $\vec{u} + \vec{v} = \begin{bmatrix} 7 \\ -5 \end{bmatrix}$.

Geometrically, we can arrive at the vector $\begin{bmatrix} 7 \\ -5 \end{bmatrix}$ by drawing $\vec{u}$ at the origin, then placing $\vec{v}$ at the tip of $\vec{u}$.



Vector addition is commutative, i.e. $\vec{u} + \vec{v} = \vec{v} + \vec{u}$, for any two vectors $\vec{u}, \vec{v} \in \mathbb{R}^n$. Algebraically, this should not be a surprise, since $u_i + v_i = v_i + u_i$ for all i.

Visually, this means that we can instead start with $\vec{v}$ at the origin and then draw $\vec{u}$ starting from the tip of $\vec{v}$, and we should land in the same place.



We cannot, however, add $\vec{w} = \begin{bmatrix} 6 \\ -2 \\ 3 \end{bmatrix}$ to $\vec{u}$, since $\vec{u}$ and $\vec{w}$ have different numbers of components.

In Python, we define vectors using `numpy` arrays, and addition occurs element-wise by default.

Activity 2

Activity 2

In the cell above, try and define $\vec{w} = \begin{bmatrix} 6 \\ -2 \\ 3 \end{bmatrix}$ as an array and add it to `u`. What error do you see?

Scalar Multiplication.

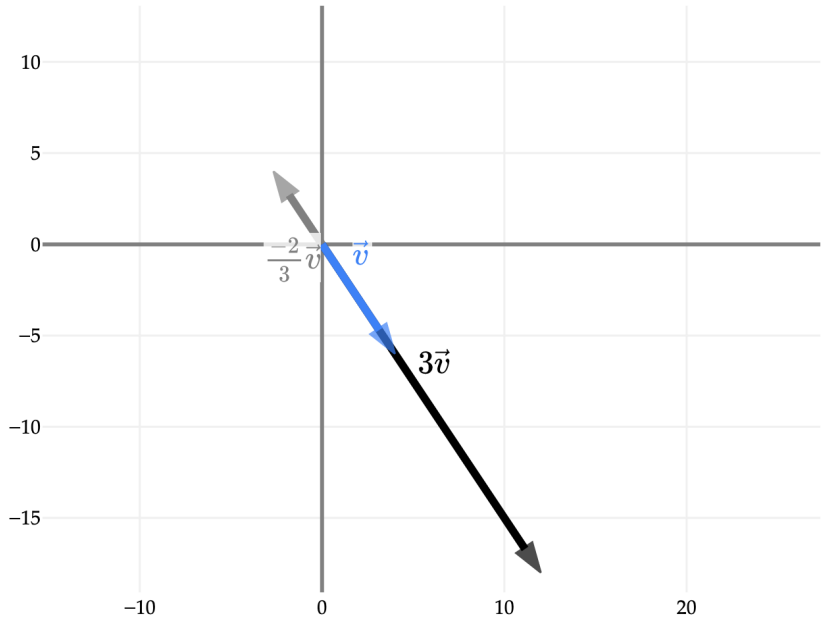
Definition: Scalar Multiplication

Suppose $\vec{v} \in \mathbb{R}^n$. The **scalar multiple** of $\vec{v}$ by a scalar $c \in \mathbb{R}$ is defined as follows:

$$c\vec{v} = c\begin{bmatrix} v_1 \\ v_2 \\ \vdots \\ v_n \end{bmatrix} = \begin{bmatrix} cv_1 \\ cv_2 \\ \vdots \\ cv_n \end{bmatrix}$$

Using our examples from earlier, $\vec{v} = \begin{bmatrix} 4 \\ -6 \end{bmatrix}$ as an example, $3\vec{v} = \begin{bmatrix} 12 \\ -18 \end{bmatrix}$. Note that I've deliberately defined this operation as **scalar** multiplication, not just "multiplication" in general, as there's more nuance to the definition of multiplication in linear algebra.

Visually, a scalar multiple is equivalent to stretching or compressing a vector by a factor of the scalar. If the scalar is negative, the direction of the vector is reversed. Below, $-\frac{2}{3}\vec{v}$ points opposite to $\vec{v}$ and $3\vec{v}$.



An important observation is that $\vec{v}$, $3\vec{v}$, and $-\frac{2}{3}\vec{v}$ **all lie on the same line**.

Activity 3

Activity 3

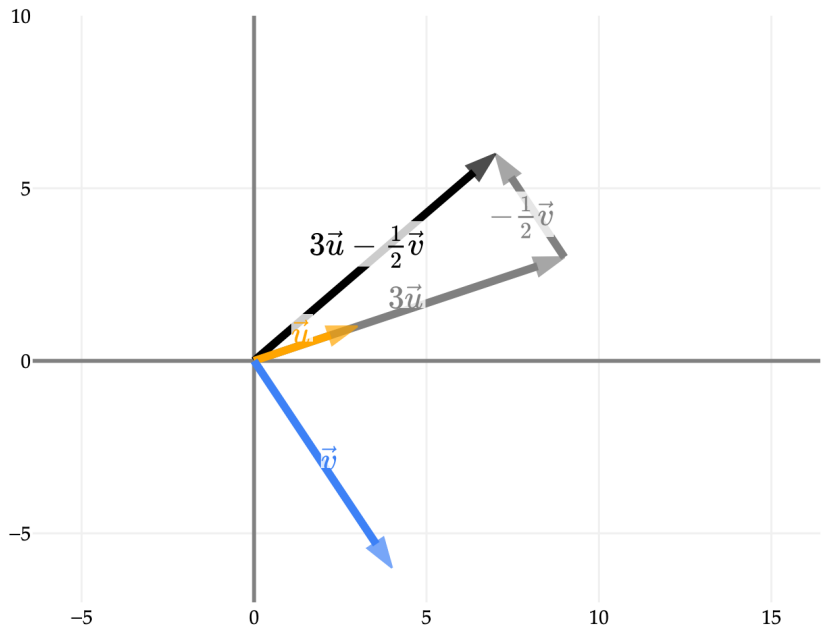
Addition and scalar multiplication can be used at the same time, as we're about to see in the next subsection. Find the vector $\vec{x}$ such that:

$$3\begin{bmatrix}7\\3\\-2\end{bmatrix} + 4\vec{x} = \begin{bmatrix}9\\4\\2\end{bmatrix}$$

Linear Combinations

Motivation and Definition. The two operations we've defined – vector addition and scalar multiplication – are the building blocks of linear algebra, and are often used in conjunction. For example, if we stick with the same vectors $\vec{u}$ and $\vec{v}$ from earlier, what might the vector $3\vec{u} - \frac{1}{2}\vec{v}$ look like?

$$3\vec{u} - \frac{1}{2}\vec{v} = 3\begin{bmatrix}3\\1\end{bmatrix} - \frac{1}{2}\begin{bmatrix}4\\-6\end{bmatrix} = \begin{bmatrix}9\\3\end{bmatrix} - \begin{bmatrix}2\\-3\end{bmatrix} = \begin{bmatrix}7\\6\end{bmatrix}$$



Definition: Linear Combination

Suppose $\vec{v}_1, \vec{v}_2, \ldots, \vec{v}_d$ are all vectors in $\mathbb{R}^n$ and $a_1, a_2, \ldots, a_d$ are scalars (sometimes called coefficients). Then, a **linear combination** of the vectors $\vec{v}_1, \vec{v}_2, \ldots, \vec{v}_d$ is any vector that can be written as:

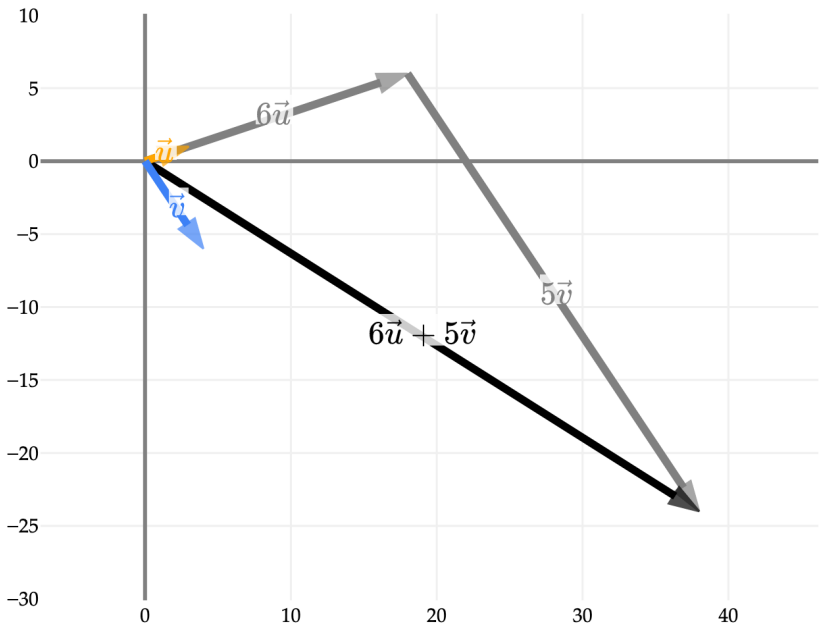
$$a_1\vec{v}_1 + a_2\vec{v}_2 + \cdots + a_d\vec{v}_d$$

The vector $\begin{bmatrix} 7 \\ 6 \end{bmatrix}$, drawn in black above, is a linear combination of $\vec{u}$ and $\vec{v}$, since it can be written in the form $3\vec{u} - \frac{1}{2}\vec{v}$. 3 and $-\frac{1}{2}$ are the scalars that the definition above refers to as a_1 and a_2, and we've used $\vec{u}$ and $\vec{v}$ in place of $\vec{v}_1$ and $\vec{v}_2$. (I've tried to make the definition a bit more general – here, we're just working with $d = 2$ vectors in $n = 2$ dimensions, but in practice d and n could both be much larger.)

Example in 2D. Here's another linear combination of $\vec{u}$ and $\vec{v}$, namely $6\vec{u} + 5\vec{v}$. Algebraically, this is:

$$6\vec{u} + 5\vec{v} = 6\begin{bmatrix} 3 \\ 1 \end{bmatrix} + 5\begin{bmatrix} 4 \\ -6 \end{bmatrix} = \begin{bmatrix} 38 \\ -24 \end{bmatrix}$$

Visually:



I like thinking of a linear combination as taking "a little bit of the first vector, a little bit of the second vector, etc." and then adding them all together. (By "little bit", I mean some amount of, e.g. $6\vec{u}$ is a little bit of $\vec{u}$.) Another useful analogy is to think of the original vectors as "building blocks" that we can use to create new vectors through addition and scalar multiplication.

This idea, of creating new vectors by scaling and adding existing vectors, is **so important** that it's essentially what our multiple linear regression problem boils down to.

In the context of our commute times example, imagine $\vec{\text{dept}}$ contains the home departure time, in hours, for each row in our dataset, and $\vec{\text{dom}}$ contains the day of the month for each row in our dataset. If we want to use these two features in a linear model to predict commute time, our problem boils down to finding the optimal coefficients w_0, w_1, and w_2 in a linear combination of $\vec{1}$, $\vec{\text{dept}}$ and $\vec{\text{dom}}$ that best predicts commute times.

$$\text{vector of predicted commute times} = w_0 \underbrace{\begin{bmatrix} 1 \\ 1 \\ \vdots \\ 1 \end{bmatrix}}_{\vec{1}} + w_1 \vec{\text{dept}} + w_2 \vec{\text{dom}}$$

Think about why $\vec{1}$ is necessary.

The Three Questions. We're going to spend a **lot** of time thinking about linear combinations. Specifically:

The Three Questions with Linear Combinations

Given a set of vectors $\vec{v}_1, \vec{v}_2, \ldots, \vec{v}_d$ and a vector $\vec{b}$, all in $\mathbb{R}^n$:

1. **Can we** write $\vec{b}$ as a linear combination of $\vec{v}_1, \vec{v}_2, \ldots, \vec{v}_d$?

2. If so, are the values of the scalars $a_1, a_2, \ldots, a_d$ **unique**?
3. What is the **shape** of the set of all possible linear combinations of $\vec{v}_1, \vec{v}_2, \ldots, \vec{v}_d$?

Again, just as an example, suppose the two vectors we're dealing with are our familiar friends:

$$\vec{u} = \begin{bmatrix} 3 \\ 1 \end{bmatrix}, \quad \vec{v} = \begin{bmatrix} 4 \\ -6 \end{bmatrix}$$

These are $d = 2$ vectors in $n = 2$ dimensions. With regards to the Three Questions:

1. **Can we** write $\vec{b}$ as a linear combination of $\vec{u}$ and $\vec{v}$?

   If $\vec{b} = \begin{bmatrix} 7 \\ 6 \end{bmatrix}$, then the answer to the first question is yes, because we've shown that:

   $$3\vec{u} - \frac{1}{2}\vec{v} = \begin{bmatrix} 7 \\ 6 \end{bmatrix}$$

   Similarly, if $\vec{b} = \begin{bmatrix} 38 \\ -24 \end{bmatrix}$, then the answer to the first question is also yes, because we've shown that:

   $$6\vec{u} + 5\vec{v} = \begin{bmatrix} 38 \\ -24 \end{bmatrix}$$

   If $\vec{b}$ is some other vector, the answer may be yes or no, for all we know right now.

2. If so, are the values of the scalars on $\vec{u}$ and $\vec{v}$ **unique**?

   Not sure! It's true that $\begin{bmatrix} 7 \\ 6 \end{bmatrix} = 3\vec{u} - \frac{1}{2}\vec{v}$, but for all I know at this point, there *could be* other scalars $a_1 \neq 3$ and $a_2 \neq -\frac{1}{2}$ such that:

   $$a_1\vec{u} + a_2\vec{v} = \begin{bmatrix} 7 \\ 6 \end{bmatrix}$$

   (As it turns out, the answer is that the values 3 and $-\frac{1}{2}$ **are** unique – you'll show why this is the case in a following activity.)

3. What is the **shape** of the set of all possible linear combinations of $\vec{u}$ and $\vec{v}$?

   Also not sure! I know that $\begin{bmatrix} 7 \\ 6 \end{bmatrix}$ and $\begin{bmatrix} 38 \\ -24 \end{bmatrix}$ are both linear combinations of $\vec{u}$ and $\vec{v}$, and presumably there are many more, but I don't know what they are.

   (It turns out that **any** vector in $\mathbb{R}^2$ can be written as a linear combination of $\vec{u}$ and $\vec{v}$! Again, you'll show this in an activity.)

We'll more comprehensively study the "Three Questions" starting in Chapter 4.1. I just wanted to call them out for you here so that you know where we're heading.

Example in 3D. Let's move to 3D. Consider the vectors:

$$\vec{w} = \begin{bmatrix} 12 \\ -4 \\ 6 \end{bmatrix}, \quad \vec{r} = \begin{bmatrix} 7 \\ 1 \\ 10 \end{bmatrix}$$

These are $d = 2$ vectors, as before, but now in $n = 3$ dimensions. What do some of their linear combinations look like?

Rotate the plot above. You'll see that all linear combinations of $\vec{w}$ and $\vec{r}$ lie on the same plane! Think of this plane as a flat sheet of paper that is tilted to align with $\vec{w}$ and $\vec{r}$, that extends infinitely in all directions. Soon, we will learn to call this plane the **span** of $\vec{w}$ and $\vec{r}$.

Is it the case that any two vectors in $\mathbb{R}^3$ together span a plane? Not necessarily: if the two vectors point in the same direction, then they both lie on the same line, and so the set of all linear combination of the two vectors is just a line, not a plane. The vectors $\begin{bmatrix}1\\2\\3\end{bmatrix}$ and $\begin{bmatrix}2\\4\\6\end{bmatrix}$ are a good example of this: any vector you can reach with a linear combination of these two vectors is a scalar multiple of $\begin{bmatrix}1\\2\\3\end{bmatrix}$, since $\begin{bmatrix}2\\4\\6\end{bmatrix} = 2\begin{bmatrix}1\\2\\3\end{bmatrix}$. More on this in Chapter 4.

The Systems of Equations Perspective. To illuminate the connection between linear combinations and systems of linear equations, I'd like to introduce another example. Let $\vec{x} = \begin{bmatrix}3\\-1\\2\end{bmatrix}$ and $\vec{y} = \begin{bmatrix}1\\4\\3\end{bmatrix}$. $\vec{x}$ and $\vec{y}$ are both vectors in $\mathbb{R}^3$ and don't point in the same direction (i.e. $\vec{y}$ is not a scalar multiple of $\vec{x}$, and vice versa), so they span a plane.

Suppose we'd like to write $\vec{b} = \begin{bmatrix}9\\-16\\-1\end{bmatrix}$ as a linear combination of $\vec{x}$ and $\vec{y}$. In other words, we want to find scalars c and d such that:

$$c\vec{x} + d\vec{y} = \begin{bmatrix}9\\-16\\-1\end{bmatrix}$$

To do so, we can write the vector equation above as:

$$\begin{bmatrix}3c\\-c\\2c\end{bmatrix} + \begin{bmatrix}d\\4d\\3d\end{bmatrix} = \begin{bmatrix}9\\-16\\-1\end{bmatrix}$$

or

$$\begin{bmatrix}3c + d\\-c + 4d\\2c + 3d\end{bmatrix} = \begin{bmatrix}9\\-16\\-1\end{bmatrix}$$

This gives us a system of three equations with two unknowns, c and d. There may or may not be a solution to this system, but we can try to find one.

$$\begin{aligned}3c + d &= 9\\-c + 4d &= -16\\2c + 3d &= -1\end{aligned}$$

We can solve this system using any method we'd like. Here, I'll use substitution.

First, solve the first equation for d:

$$d = 9 - 3c$$

Next, substitute this expression for d into the second equation:

$$\begin{aligned} -c + 4(9 - 3c) &= -16 \\ -c + 36 - 12c &= -16 \\ -13c + 36 &= -16 \\ -13c &= -52 \\ c &= 4 \end{aligned}$$

Now, substitute $c = 4$ back into the expression for d:

$$d = 9 - 3(4) = 9 - 12 = -3$$

Is the solution $\boxed{(c, d) = (4, -3)}$ consistent? We can verify by substituting $c = 4$ and $d = -3$ back into the original equation:

$$4\begin{bmatrix} 3 \\ -1 \\ 2 \end{bmatrix} - 3\begin{bmatrix} 1 \\ 4 \\ 3 \end{bmatrix} = \begin{bmatrix} 9 \\ -16 \\ -1 \end{bmatrix}$$

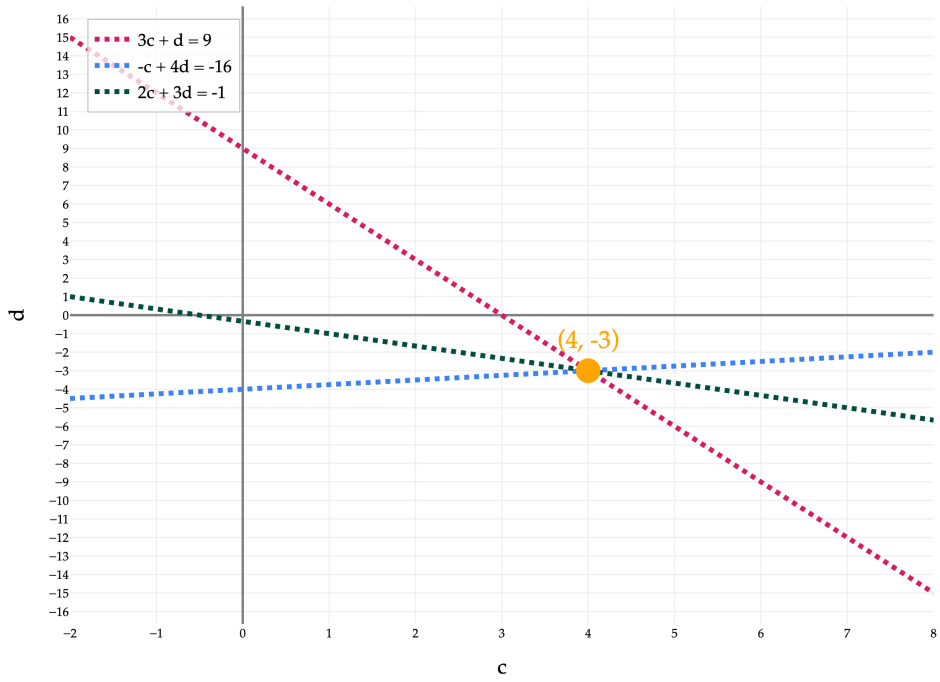
So, indeed, $c = 4$ and $d = -3$ are the coefficients that allow us to write $\begin{bmatrix} 9 \\ -16 \\ -1 \end{bmatrix}$ as a linear combination of $\begin{bmatrix} 3 \\ -1 \\ 2 \end{bmatrix}$ and $\begin{bmatrix} 1 \\ 4 \\ 3 \end{bmatrix}$. This tells us how much to take of $\begin{bmatrix} 3 \\ -1 \\ 2 \end{bmatrix}$ and $\begin{bmatrix} 1 \\ 4 \\ 3 \end{bmatrix}$ to make $\begin{bmatrix} 9 \\ -16 \\ -1 \end{bmatrix}$.

To find these coefficients, we solved a system of three equations with two unknowns.

$$\begin{aligned} 3c + d &= 9 \\ -c + 4d &= -16 \\ 2c + 3d &= -1 \end{aligned}$$

Each of the three equations above is a line in the cd-plane. The first one, $3c+d = 9$, can be rewritten as $d = -3c+9$, which is a line with slope -3 and y-intercept 9, for instance.

What does this system **look like** in the cd-plane?



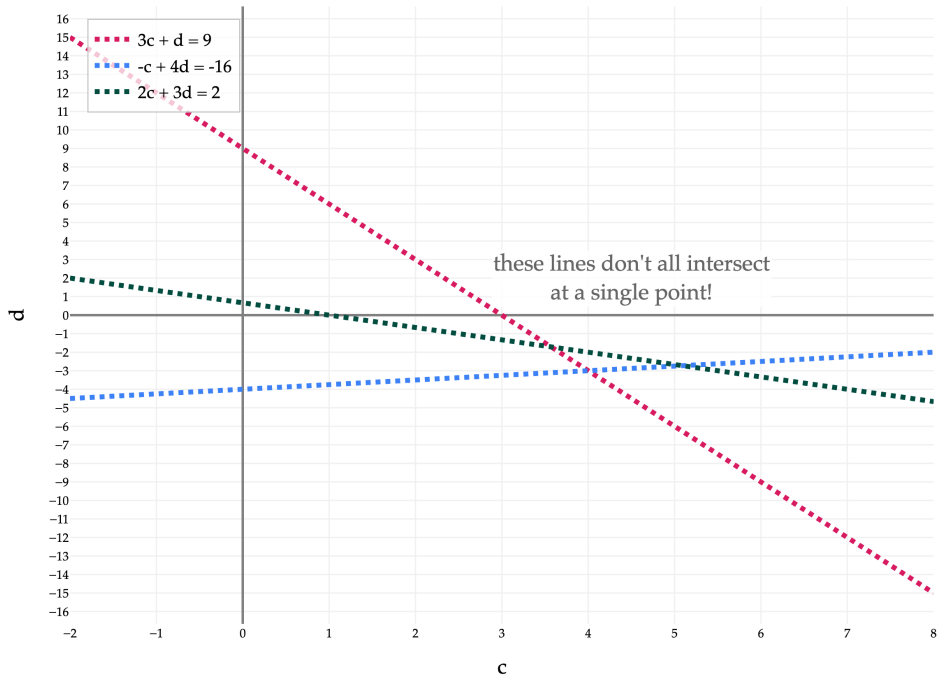
The three lines in this system intersect at the point $(4, -3)$, which is the solution to the system of equations. Because all three lines intersect at a single point, there is a unique solution, meaning there is **only one way** to write $\vec{b} = \begin{bmatrix} 9 \\ -16 \\ -1 \end{bmatrix}$ as a linear combination of $\vec{x} = \begin{bmatrix} 3 \\ -1 \\ 2 \end{bmatrix}$ and $\vec{y} = \begin{bmatrix} 1 \\ 4 \\ 3 \end{bmatrix}$.

Another Example. To illustrate what could go wrong, let's look at one final example. We'll keep the same $\vec{x} = \begin{bmatrix} 3 \\ -1 \\ 2 \end{bmatrix}$ and $\vec{y} = \begin{bmatrix} 1 \\ 4 \\ 3 \end{bmatrix}$, but choose a different "right-hand side" vector to create, say

$$\vec{b} = \begin{bmatrix} 9 \\ -16 \\ 2 \end{bmatrix}$$

Then the constraint equations are

$$\begin{aligned} 3c + d &= 9 \\ -c + 4d &= -16 \\ 2c + 3d &= 2 \end{aligned}$$



Each pair of lines still intersects in the cd-plane, but all three points of intersection are different. As such, there are no scalars c and d such that

$$c\begin{bmatrix}3\\-1\\2\end{bmatrix} + d\begin{bmatrix}1\\4\\3\end{bmatrix} = \begin{bmatrix}9\\-16\\2\end{bmatrix}$$

meaning that $\begin{bmatrix}9\\-16\\2\end{bmatrix}$ **cannot** be written as a linear combination of $\begin{bmatrix}3\\-1\\2\end{bmatrix}$ and $\begin{bmatrix}1\\4\\3\end{bmatrix}$.

So far, we've seen two cases:

- $\vec{b}$ could be written **exactly one way** as a linear combination of $\vec{x}$ and $\vec{y}$
- $\vec{b}$ **could not** be written as a linear combination of $\vec{x}$ and $\vec{y}$

In Chapter 4, we'll see that there's a third case: $\vec{b}$ could be written in **infinitely many ways** as a linear combination of $\vec{x}$ and $\vec{y}$.

3.2. Norms

In Chapter 3.1, we defined the norm of a vector $\vec{v}$ as:

$$\|\vec{v}\| = \sqrt{v_1^2 + v_2^2 + \cdots + v_n^2} = \sqrt{\sum_{i=1}^{n} v_i^2}$$

Now that we know how to add and scale vectors, we should think about how the norm – i.e., the (geometric) length of a vector (**not** the number of components it has) – behaves under these operations.

Properties of the Norm

The Three Properties

Recall, for $\vec{v} \in \mathbb{R}^n$:

$$\|\vec{v}\| = \sqrt{v_1^2 + v_2^2 + \cdots + v_n^2} = \sqrt{\sum_{i=1}^{n} v_i^2}$$

Then, $\|\vec{v}\|$ satisfies:

1. $\|\vec{v}\| \geq 0$, and $\|\vec{v}\| = 0$ if and only if $\vec{v}$ is the **zero vector**, $\vec{0} = \begin{bmatrix} 0 \\ 0 \\ \vdots \\ 0 \end{bmatrix}$.
2. $\|c\vec{v}\| = |c|\|\vec{v}\|$ for all $c \in \mathbb{R}$.
3. $\|\vec{u} + \vec{v}\| \leq \|\vec{u}\| + \|\vec{v}\|$.

Properties 1 and 2 are intuitive enough:

1. Property 1 states that it's impossible for a vector to have a negative norm. To calculate the norm a vector, we sum the squares of each of the vector's components. As long as each component v_i is a real number, then $v_i^2 \geq 0$, and so $\sum_{i=1}^{n} v_i^2 \geq 0$. The square root of a non-negative number is always non-negative, so the norm of a vector is always non-negative. The only case in which $\sum_{i=1}^{n} v_i^2 = 0$ is when each $v_i = 0$, so the only vector with a norm of 0 is the zero vector.
2. Property 2 states that scaling a vector by a scalar scales its norm by the absolute value of the scalar. For instance, it's saying that both $2\vec{v}$ and $-2\vec{v}$ should be double the length of $\vec{v}$. **See, at this point, if you can *prove* why this is the case.**

Activity 1

Activity 1

Prove Property 2 of vector norms, i.e. that:

$$\|c\vec{v}\| = |c|\|\vec{v}\|$$

Start by using the definition of the norm of the vector $c\vec{v}$.
Once you've given it a shot, read the solution below.

Solution

As I suggested, let's start by expanding the definition of $\|c\vec{v}\|$.

$$\begin{aligned}\|c\vec{v}\| &= \sqrt{(cv_1)^2 + (cv_2)^2 + \cdots + (cv_n)^2} \\ &= \sqrt{c^2v_1^2 + c^2v_2^2 + \cdots + c^2v_n^2} \\ &= \sqrt{c^2(v_1^2 + v_2^2 + \cdots + v_n^2)}\end{aligned}$$

At this point, we'd like to remove the c^2 from the square root. The issue is that c^2 is a non-negative number, but c could either have been positive or negative. So, we need to take the absolute value of c. (Remember, $\sqrt{x^2} = |x|$, not $\sqrt{x} = x$.)
Then:

$$\begin{aligned}\|c\vec{v}\| &= \sqrt{c^2(v_1^2 + v_2^2 + \cdots + v_n^2)} \\ &= |c| \underbrace{\sqrt{v_1^2 + v_2^2 + \cdots + v_n^2}}_{\text{This is just } \|\vec{v}\|!} \\ &= |c|\|\vec{v}\|\end{aligned}$$

So, we've shown that $\|c\vec{v}\| = |c|\|\vec{v}\|$. Even though this fact may seem like it "just should be true", we need to be able to rigorously justify it, and we've now done just that.

The Triangle Inequality

Property 3 is a bit more interesting. As a reminder, it states that:

$$\|\vec{u} + \vec{v}\| \leq \|\vec{u}\| + \|\vec{v}\|$$

This is a famous inequality, generally known as the **triangle inequality**, and it comes up all the time in proofs. Intuitively, it says that the length of a sum of vectors cannot be greater than the sum of the lengths of the individual vectors – or, more philosophically, a sum cannot be more than its parts. It's called the triangle inequality because it's a generalization of the fact that in a triangle, the sum of the lengths of any two sides is greater than the length of the third side.

Above:

$$\|\vec{u}\| = \sqrt{3^2 + 1^2} = \sqrt{10}$$

$$\|\vec{v}\| = \sqrt{4^2 + (-6)^2} = \sqrt{52}$$

$$\|\vec{u} + \vec{v}\| = \sqrt{7^2 + (-5)^2} = \sqrt{74}$$

And indeed, $\sqrt{74} \approx 8.6$ is less than $\sqrt{10} + \sqrt{52} \approx 10.4$.

To prove that the triangle inequality holds in general, for **any** two vectors $\vec{u}, \vec{v} \in \mathbb{R}^n$, we'll need to wait until Chapter 3.3. We don't currently have any way to expand the norm $\|\vec{u} + \vec{v}\|$ – but we'll develop the tools to do so soon. Just keep it in mind for now.

Activity 2

Activity 2

The triangle inequality says that:

$$\|\vec{u} + \vec{v}\| \leq \|\vec{u}\| + \|\vec{v}\|$$

In the example above, we had $\sqrt{74} < \sqrt{10} + \sqrt{52}$. In other words, there was **strict** inequality.
Find a pair of vectors $\vec{u}, \vec{v}$ (say, in $\mathbb{R}^2$) such that the triangle inequality achieves **equality**, i.e. $\|\vec{u} + \vec{v}\| = \|\vec{u}\| + \|\vec{v}\|$.

Unit Vectors and the Norm Ball

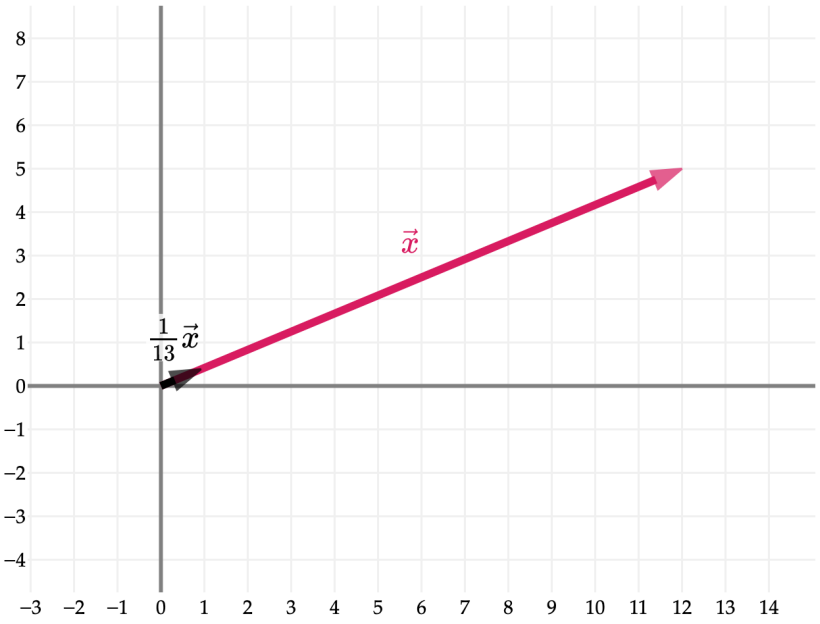
Definition: Unit Vectors

A **unit vector** is a vector with a norm of 1, i.e. $\|\vec{v}\| = 1$.

It's common to use unit vectors to describe directions. I'll use the same example as in Activity 1, when this idea was first introduced. Consider the vector $\vec{x} = \begin{bmatrix} 12 \\ 5 \end{bmatrix}$. Its norm is $\|\vec{x}\| = \sqrt{12^2 + 5^2} = \sqrt{169} = \boxed{13}$. (You *might* remember the $(5, 12, 13)$ Pythagorean triple from high school algebra– but that's not important.)

There are plenty of vectors that point in the same direction as $\vec{x}$ – any vector $c\vec{x}$ for $c > 0$ does. (If $c < 0$, then the vector $c\vec{x}$ points in the opposite direction of $\vec{x}$.)

But among all those, the only one with a norm of 1 is $\frac{1}{\boxed{13}}\vec{x}$. Property 2 of the norm tells us this.



In general, if $\vec{v}$ is any vector, then:

$$\frac{\vec{v}}{\|\vec{v}\|}$$

is a unit vector in the same direction as $\vec{v}$. Sometimes, we say that $\frac{\vec{v}}{\|\vec{v}\|}$ is a **normalized** version of $\vec{v}$. Here's where things get interesting. Let's visualize a few vectors and their normalized versions:

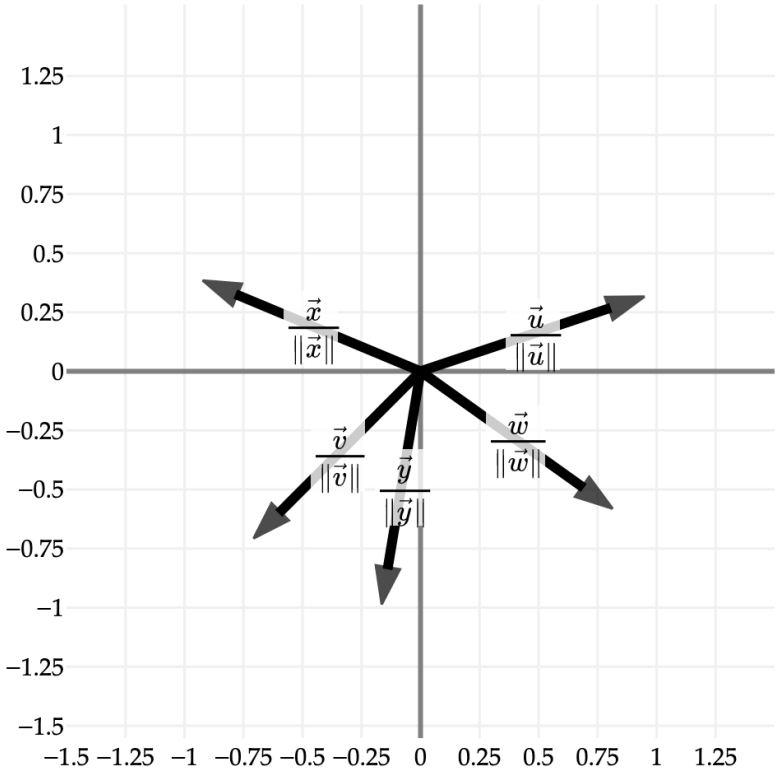
$$\vec{u} = \begin{bmatrix} 3 \\ 1 \end{bmatrix} \implies \frac{\vec{u}}{\|\vec{u}\|} = \begin{bmatrix} \frac{3}{\sqrt{10}} \\ \frac{1}{\sqrt{10}} \end{bmatrix}$$

$$\vec{v} = \begin{bmatrix} -6 \\ -6 \end{bmatrix} \implies \frac{\vec{v}}{\|\vec{v}\|} = \begin{bmatrix} \frac{-1}{\sqrt{2}} \\ \frac{-1}{\sqrt{2}} \end{bmatrix}$$

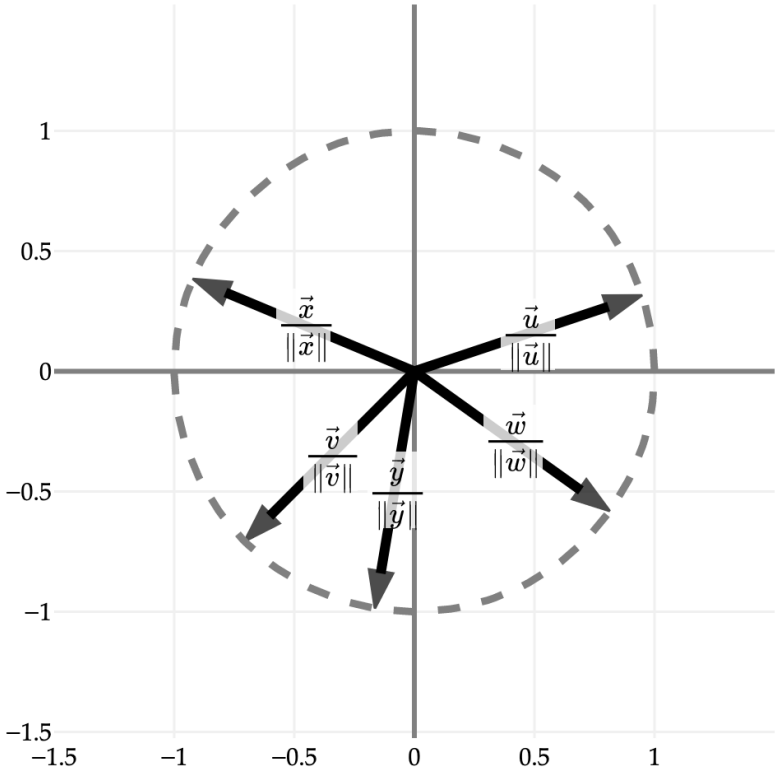
$$\vec{w} = \begin{bmatrix} 7 \\ -5 \end{bmatrix} \implies \frac{\vec{w}}{\|\vec{w}\|} = \begin{bmatrix} \frac{7}{\sqrt{74}} \\ \frac{-5}{\sqrt{74}} \end{bmatrix}$$

$$\vec{x} = \begin{bmatrix} -12 \\ 5 \end{bmatrix} \implies \frac{\vec{x}}{\|\vec{x}\|} = \begin{bmatrix} \frac{-12}{13} \\ \frac{5}{13} \end{bmatrix}$$

$$\vec{y} = \begin{bmatrix} -1 \\ -6 \end{bmatrix} \implies \frac{\vec{y}}{\|\vec{y}\|} = \begin{bmatrix} \frac{-1}{\sqrt{37}} \\ \frac{-6}{\sqrt{37}} \end{bmatrix}$$



What do these vectors all have in common, other than being unit vectors? **They all lie on a circle of radius 1, centered at** $(0, 0)$**!**



The circle shown above is called the **norm ball** of radius 1 in $\mathbb{R}^2$. It shows the set of all vectors $\vec{v} \in \mathbb{R}^2$ such that $\|\vec{v}\| = 1$. Using set notation, we might say:

$$\{\vec{v} : \|\vec{v}\| = 1, \vec{v} \in \mathbb{R}^2\}$$

That this looks like a circle is no coincidence. The condition $\|\vec{v}\| = 1$ is equivalent to $\sqrt{v_1^2 + v_2^2} = 1$. Squaring both sides, we get $v_1^2 + v_2^2 = 1$. This is the equation of a circle with radius 1 centered at the origin.
In $\mathbb{R}^3$, the norm ball of radius 1 is a sphere, and in general, in $\mathbb{R}^n$, the norm ball of radius 1 is an n-dimensional sphere.

Activity 3

Activity 3

Find the unit vector that points in the same direction as $\vec{z} = \begin{bmatrix} 4 \\ 0 \\ -3 \\ -3 \end{bmatrix}$.

Clearly write out all four of its components.

Other Norms

So far, we've only discussed one "norm" of a vector, sometimes called the L_2 norm or Euclidean norm. In general, if $\vec{v} \in \mathbb{R}^n$ is one vector, $\vec{v} = \begin{bmatrix} v_1 \\ v_2 \\ \vdots \\ v_n \end{bmatrix}$, then its norm is:

$$\|\vec{v}\| = \sqrt{v_1^2 + v_2^2 + \cdots + v_n^2} = \sqrt{\sum_{i=1}^{n} v_i^2}$$

This is, by far, the most common and most relevant norm, and in many linear algebra classes, it's the only norm you'll see. But in machine learning, a few other norms are relevant, too, so I'll briefly discuss them here.

- The L_1 or Manhattan norm of $\vec{v}$ is:

$$\|\vec{v}\|_1 = |v_1| + |v_2| + \cdots + |v_n| = \sum_{i=1}^{n} |v_i|$$

  It's called the Manhattan norm because it's the distance you would travel if you walked from the origin to $\vec{v}$ in a grid of streets, where you can only move horizontally or vertically.

- The L_∞ or maximum norm of $\vec{v}$ is:

$$\|\vec{v}\|_\infty = \max_i |v_i|$$

  This is largest absolute value of any component of $\vec{v}$.

- For any $p \geq 1$, the L_p norm of $\vec{v}$ is:

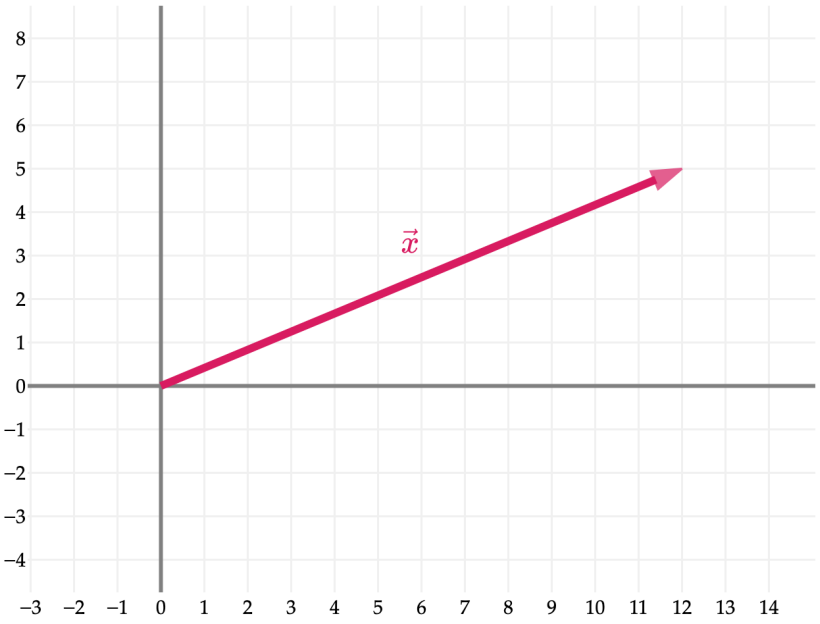
$$\|\vec{v}\|_p = \left(\sum_{i=1}^{n} |v_i|^p \right)^{\frac{1}{p}}$$

  Note that when $p = 2$, this is the same as the L_2 norm. For other values of p, this is a generalization. Something to think about: why is there an absolute value in the definition?

All of these norms measure the length of a vector, but in different ways. This might ring a bell: we saw very similar tradeoffs between squared and absolute losses in Chapter 1.

Believe it or not, all three of these norms satisfy the same "Three Properties" we discussed earlier.

Back to $\vec{x} = \begin{bmatrix} 12 \\ 5 \end{bmatrix}$. What are the L_2, L_1, and L_∞ norms of $\vec{x}$?



Here:

- $\|\vec{x}\|_2 = \sqrt{12^2 + 5^2} = \sqrt{144 + 25} = \sqrt{169} = 13$
- $\|\vec{x}\|_1 = |12| + |5| = 12 + 5 = 17$
- $\|\vec{x}\|_\infty = \max(|12|, |5|) = 12$

Let's revisit the idea of a norm ball. Using the standard L_2 norm, the norm ball in $\mathbb{R}^2$ is a circle. What does the norm ball look like for the L_1 and L_∞ norms? Or L_p with an arbitrary p?

The L_1 norm ball looks like a diamond. Any vector with an L_1 norm of 1 will lie on the boundary of the ball. The L_∞ norm ball looks like a square, and the $L_{1.3}$ norm ball looks like a diamond with rounded corners.

This is not the last you'll see of these norm balls – in particular, in future machine learning courses, you'll see them again in the context of **regularization**, which is a technique for preventing **overfitting** in our models.

The Default Norm is Still L_2!

In general, assume that when we refer to the "norm" of a vector with no other specifications or subscripts, we're talking about the standard L_2 norm that you've become familiar with here.
That is,

$$\|\vec{v}\| = \sqrt{v_1^2 + v_2^2 + \cdots + v_n^2} = \sqrt{\sum_{i=1}^{n} v_i^2}$$

is the default norm.

Activity 4

Activity 4

In the case of $\vec{x} = \begin{bmatrix} 12 \\ 5 \end{bmatrix}$, the L_1 norm ($12 + 5 = 17$) is greater than the L_2 norm ($\sqrt{12^2 + 5^2} = \sqrt{169} = 13$), i.e. $\|\vec{x}\|_1 > \|\vec{x}\|_2$.

1. Find a vector $\vec{y} \in \mathbb{R}^2$ such that $\|\vec{y}\|_1 = \|\vec{y}\|_2$.
2. Try and find a vector $\vec{z} \in \mathbb{R}^2$ such that $\|\vec{z}\|_1 < \|\vec{z}\|_2$. What do you encounter?
3. **Prove** that $\|\vec{x}\|_2 \leq \sqrt{n}\|\vec{x}\|_\infty$ for any vector $\vec{x} \in \mathbb{R}^n$. (Hint: Start with the definition of the L_2 norm of $\vec{x}$, square it, and try and compare each element in the sum to the largest element in the vector.)

`np.linalg.norm` and Vectorization

It's been a while since we've experimented with `numpy`. A few things:

- As we've seen, arrays can be added element-wise by default.
- Arrays can also be multiplied by scalars out-of-the-box, meaning that linear combinations of arrays (vectors) are easy to compute. The above two facts mean that array operations are **vectorized**: they are applied to each element of the array in parallel, without needing to use a `for`-loop.
- To compute the (L_2) norm of an array (vector), we can use `np.linalg.norm`.

Suppose you didn't know about `np.linalg.norm`. There's another way to compute the norm of an array (vector), that **doesn't** involve a `for`-loop. Follow the activity to discover it.

Activity 5

Activity 5

In the cell above:

1. Write `u ** 2`. Squaring a vector is not an operation we've discussed (and isn't an operation that exists in math), but `numpy` gives you back another array. What does this array contain?
2. Using `np.sum` and the new array you just created, find the norm of `u` without using `np.linalg.norm`.
3. Find the norm of `3 * u - 0.5 * v` using the same technique, and make sure you get the same result as was already displayed for you.

In general, we'll want to avoid Python `for`-loops in our code when there are `numpy`-native alternatives, as these `numpy` functions are optimized to use C (the programming language) under the hood for speed and memory efficiency.

3.3. The Dot Product

Definitions

In Chapter 3.1, we learned how to add and scale vectors. The next natural operation is to consider how to multiply two vectors together. Let's start with a definition, and then make sense of it.

The Computational Definition.

Definition: Dot Product

Suppose $\vec{u}, \vec{v} \in \mathbb{R}^n$, i.e. both are vectors with n components.
Then, the **dot product** of $\vec{u}$ and $\vec{v}$, denoted by $\vec{u} \cdot \vec{v}$, is the **scalar** defined as follows:

$$\vec{u} \cdot \vec{v} = \begin{bmatrix} u_1 \\ u_2 \\ \vdots \\ u_n \end{bmatrix} \cdot \begin{bmatrix} v_1 \\ v_2 \\ \vdots \\ v_n \end{bmatrix} = u_1 v_1 + u_2 v_2 + \cdots + u_n v_n$$

I call this the **computational definition** because it's the definition that's easiest to compute. Let's work out an example. Consider the vectors $\vec{u} = \begin{bmatrix} 6 \\ 2 \end{bmatrix}$ and $\vec{v} = \begin{bmatrix} 5 \\ -3 \end{bmatrix}$. Their dot product is:

$$\vec{u} \cdot \vec{v} = \begin{bmatrix} 6 \\ 2 \end{bmatrix} \cdot \begin{bmatrix} 5 \\ -3 \end{bmatrix} = (6)(5) + (2)(-3) = 30 - 6 = 24$$

Note that the dot product is **one number** (24 here), **not** another vector.

Activity 1

Activity 1

Activity 1.1

Let $\vec{z} = \begin{bmatrix} 5 \\ 3 \\ -1 \end{bmatrix}$ and $\vec{1} = \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix}$.

1. Find $\vec{z} \cdot \vec{1}$.
2. In general, if $z \in \mathbb{R}^n$ is any vector, and $\vec{1} = \begin{bmatrix} 1 \\ 1 \\ \vdots \\ 1 \end{bmatrix}$ is a vector of all 1s with the same number of components as $\vec{z}$, what is the value of:

$$\vec{z} \cdot \vec{1}$$

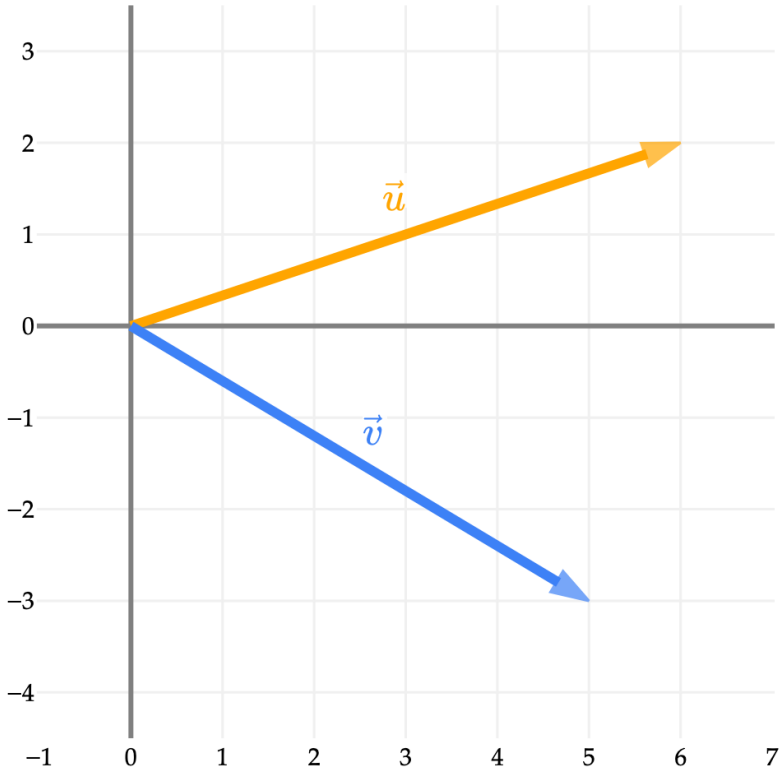
Activity 1.2

Dot products are useful for computing weighted averages. Let's illustrate that here. In your freshman fall semester, you took the following courses and earned the following grades:

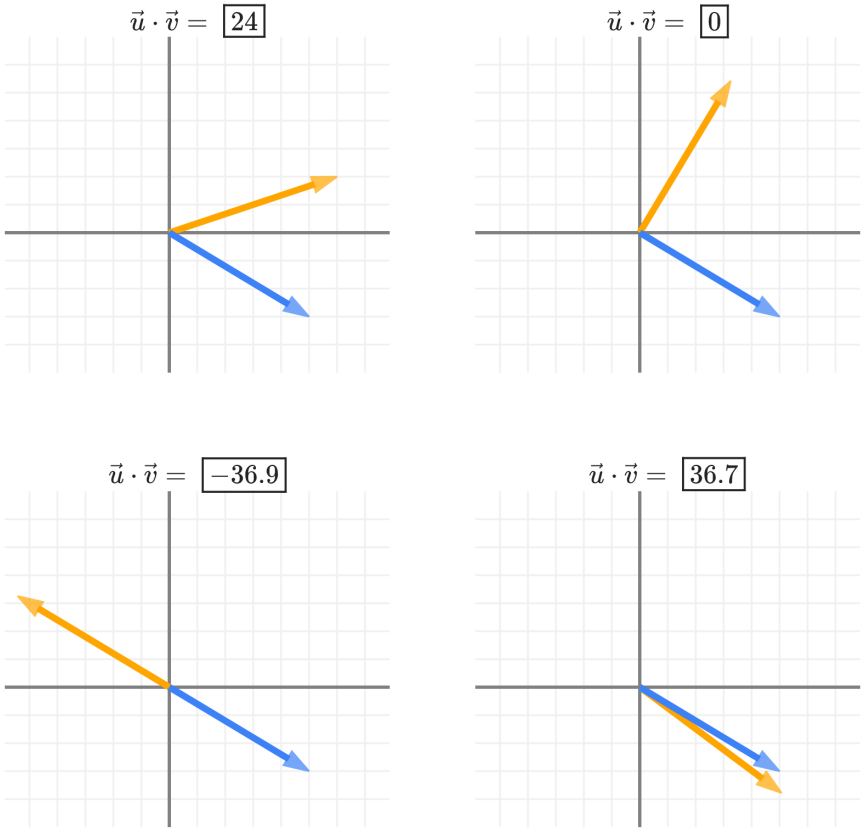
Course	Grade	Credits
EECS 245	4 (A+)	4
MATH 116	3.7 (A-)	3
EECS 201	0 (F)	1
DATASCI 101	3.3 (B+)	4

Find your GPA for the semester, and **express it as a dot product** between a grades vector $\vec{g}$ and a weights vector $\vec{w}$.

What does the dot product of 24 tell us about $\vec{u}$ and $\vec{v}$?



On its own, 24 doesn't mean much. Let's imagine we keep $\vec{v}$ fixed, and move $\vec{u}$ around. What do you notice about the resulting dot products?



It seems like the dot product has something to do with the **angle** between $\vec{u}$ and $\vec{v}$:

- When $\vec{u} \cdot \vec{v}$ is large and positive, it seems like the two vectors are pointing in the same direction. (The larger the dot product, the more aligned they are.)
- When $\vec{u} \cdot \vec{v}$ is large and negative, it seems like the two vectors are pointing in opposite directions.
- When $\vec{u} \cdot \vec{v}$ is 0, it seems like the two vectors are perpendicular.

In fact, there's another **equivalent** definition of the dot product that makes this relationship explicit.

The Geometric Definition.

Definition: Geometric Definition of Dot Product

Suppose $\vec{u}, \vec{v} \in \mathbb{R}^n$, i.e. both are vectors with n components.
The dot product can **also** be computed as:

$$\vec{u} \cdot \vec{v} = \|\vec{u}\| \, \|\vec{v}\| \cos\theta$$

where θ is the angle between $\vec{u}$ and $\vec{v}$.

Activity 2

Activity 2

The fact that the two definitions of the dot product are equivalent allows us to use the dot product to find the angle between two vectors.

Let $\vec{w} = \begin{bmatrix} 5 \\ 0 \\ -4 \end{bmatrix}$ and $\vec{x} = \begin{bmatrix} 9 \\ 1 \\ 2 \end{bmatrix}$.

1. Find the dot product between $\vec{w}$ and $\vec{x}$.
2. Find the angle between $\vec{w}$ and $\vec{x}$. Leave your answer in the form $\cos^{-1}(\cdot)$.

Solution

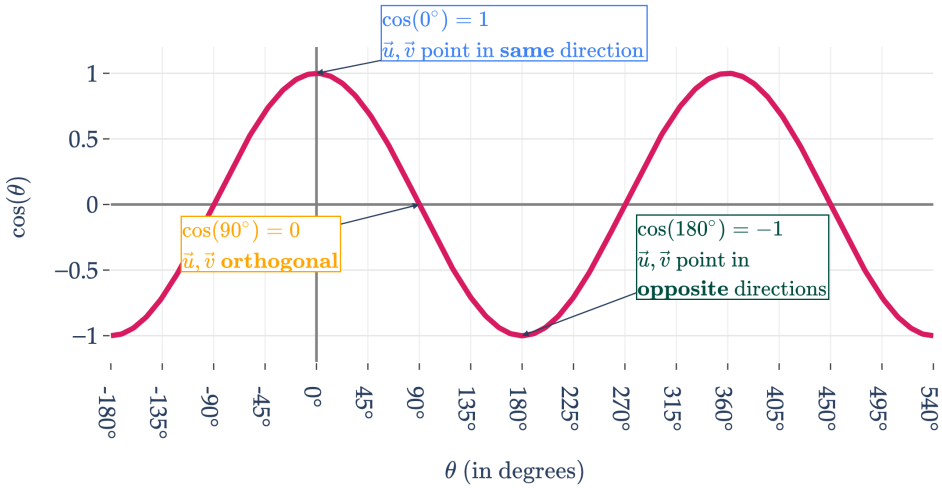
1. $\vec{w} \cdot \vec{x} = \begin{bmatrix} 5 \\ 0 \\ -4 \end{bmatrix} \cdot \begin{bmatrix} 9 \\ 1 \\ 2 \end{bmatrix} = 5 \cdot 9 + 0 \cdot 1 + (-4) \cdot 2 = 45 - 8 = 37.$
2. Since $\vec{w} \cdot \vec{x} = 37$, and $\vec{w} \cdot \vec{x} = \left\|\vec{w}\right\| \left\|\vec{x}\right\| \cos\theta$, we have

$$\begin{aligned} 37 &= \|\vec{w}\| \|\vec{x}\| \cos\theta \\ \implies \cos\theta &= \frac{37}{\|\vec{w}\| \|\vec{x}\|} \\ &= \frac{37}{\sqrt{5^2 + 0^2 + (-4)^2}\sqrt{9^2 + 1^2 + 2^2}} \\ &= \frac{37}{\sqrt{41}\sqrt{86}} \\ \implies \theta &= \cos^{-1}\left(\frac{37}{\sqrt{41}\sqrt{86}}\right) \end{aligned}$$

To **prove** why these two definitions are equivalent, we'll need to learn a bit more about the properties of the dot product. For now, let's just try and interpret this new formula. Here are both definitions of the dot product, for two vectors $\vec{u}$ and $\vec{v}$:

$$\begin{aligned} \vec{u} \cdot \vec{v} &= u_1 v_1 + u_2 v_2 + \cdots + u_n v_n \\ &= \left\|\vec{u}\right\| \left\|\vec{v}\right\| \cos\theta \end{aligned}$$

How does the function $\cos\theta$ behave? Remember, θ is the angle **between** the two vectors.



This explains what we saw in the earlier grid, which contained four pairs of vectors along with their dot products. To recap the logic:

$$\text{two vectors point in similar directions} \implies \theta \text{ small} \implies \cos\theta \text{ close to } 1 \implies \text{dot product large}$$

Broadly, the larger the dot product of two vectors is, the more similar they are!

There's another **hugely** important property that the plot of $\cos\theta$ reveals. Hugely.

Orthogonal Vectors.

Definition: Orthogonal Vectors

Vectors $\vec{u}, \vec{v} \in \mathbb{R}^n$ are **orthogonal** if their dot product is 0:

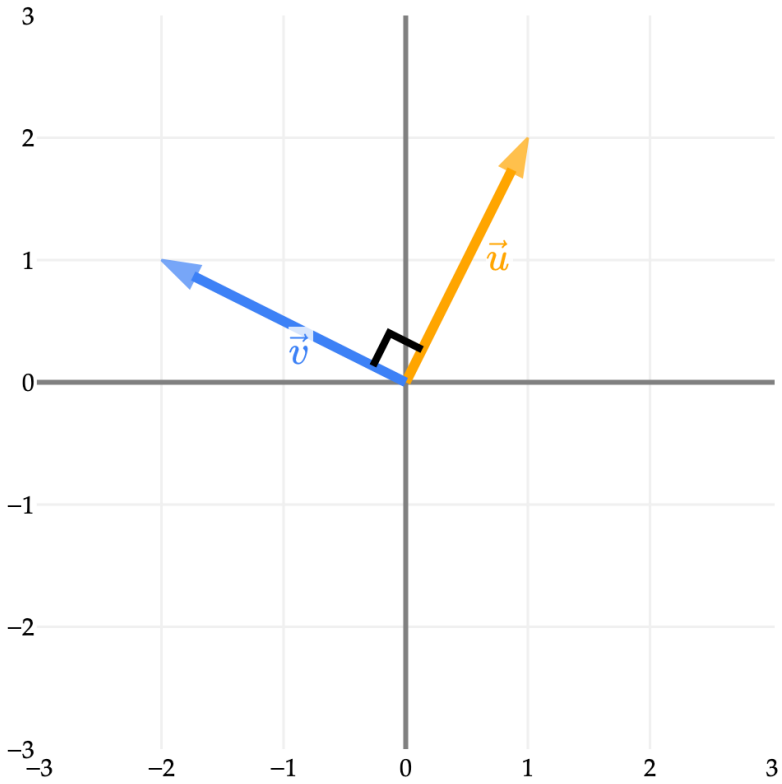
$$\vec{u} \cdot \vec{v} = 0$$

Orthogonal is just a fancy word for **perpendicular**. Both words mean that the two vectors are at a right angle (90º) to each other.

As an example in $\mathbb{R}^2$, the vectors $\vec{u} = \begin{bmatrix} 1 \\ 2 \end{bmatrix}$ and $\vec{v} = \begin{bmatrix} -10 \\ 5 \end{bmatrix}$ are orthogonal:

- Computationally, $\vec{u} \cdot \vec{v} = (1)(-10) + (2)(5) = -10 + 10 = 0$.
- Geometrically, the angle between them is 90 degrees, so $\cos\theta = 0$, meaning $\vec{u} \cdot \vec{v} = \|\vec{u}\|\|\vec{v}\| \cos\theta = 0$.

$$\vec{u} \cdot \vec{v} = \boxed{0}$$



For an example in $\mathbb{R}^3$, the vectors $\vec{w} = \begin{bmatrix} 3 \\ 6 \\ 2 \end{bmatrix}$ and $\vec{r} = \begin{bmatrix} -5 \\ 2 \\ \frac{3}{2} \end{bmatrix}$ are also orthogonal:

- Computationally, $\vec{w} \cdot \vec{r} = (3)(-5) + (6)(2) + (2)(\frac{3}{2}) = -15 + 12 + 3 = 0$.
- Geometrically, the angle between them is 90 degrees, so $\cos\theta = 0$, meaning $\vec{w} \cdot \vec{r} = \|\vec{w}\|\|\vec{r}\| \cos\theta = 0$.

What does a right angle look like in 4 or higher dimensions? I'm not sure, but that's the beauty of abstraction once again – this definition of orthogonality works in any dimension, just like our definitions of the dot product. If two vectors are orthogonal, I like thinking of them as being "as different as possible", in contrast to two vectors that point in the same direction.

Orthogonality, as it turns out, is crucial to our goal of framing the linear regression problem in terms of linear algebra. I'm not a big proponent of asking you to memorize things (I'd rather you internalize them through practice!), but the definition of orthogonality is one that you **need** to remember.

Activity 3

Activity 3

Activity 3.1

When you say "orthogonal" instead of "perpendicular" pic.twitter.com/1pJidKFIrq

Figure 3.1: *
MathMatize Memes (@MathMatize) January 22, 2026

Find a value of k such that the vectors $\vec{u} = \begin{bmatrix} 9 \\ -2 \\ 1 \end{bmatrix}$ and $\vec{v} = \begin{bmatrix} 1 \\ k \\ 3 \end{bmatrix}$ are orthogonal.

Is this value of k unique?

Activity 3.2

Find a vector that is orthogonal to **both** $\vec{u} = \begin{bmatrix} 1 \\ -2 \\ 4 \end{bmatrix}$ and $\vec{v} = \begin{bmatrix} 3 \\ -1 \\ 9 \end{bmatrix}$. In $\mathbb{R}^3$, what does this new vector look like, relative to $\vec{u}$ and $\vec{v}$?

Properties of the Dot Product

Properties of the Dot Product

Recall, for any vectors $\vec{u}, \vec{v} \in \mathbb{R}^n$, the dot product is defined as:

$$\begin{aligned} \vec{u} \cdot \vec{v} &= u_1 v_1 + u_2 v_2 + \cdots + u_n v_n \\ &= \|\vec{u}\| \, \|\vec{v}\| \cos\theta \end{aligned}$$

The dot product satisfies the following properties:

- **Commutative property**: $\vec{u} \cdot \vec{v} = \vec{v} \cdot \vec{u}$
- **Distributive property**: $\vec{u} \cdot (\vec{v} + \vec{w}) = \vec{u} \cdot \vec{v} + \vec{u} \cdot \vec{w}$
- **Associative property with respect to a scalar**: $c(\vec{v} \cdot \vec{w}) = (c\vec{v}) \cdot \vec{w} = \vec{v} \cdot (c\vec{w})$

We've already taken the commutative property for granted (the angle between $\vec{u}$ and $\vec{v}$ is the same as the angle between $\vec{v}$ and $\vec{u}$), and we're about to see a powerful application of the distributive property (though it's a good exercise to **see if you can verify it yourself**).

Let me comment on the last property, the associativity of the dot product with respect to a scalar. In standard multiplication, the associativity property for scalars $a, b, c \in \mathbb{R}$ says that $abc = (ab)c = a(bc)$. However, this **does not** hold for the dot product, because **the dot product of three vectors has no meaning!** Instead, the modified associativity property for the dot product concerns itself with two vectors and a scalar.

Activity 4

Activity 4

Suppose the dot product of $\vec{x}$ and $\vec{y}$ is 10, and the angle between $\vec{x}$ and $\vec{y}$ is 30°.

1. What is the dot product of $2\vec{x}$ and $3\vec{y}$?

2. What is the angle between $2\vec{x}$ and $3\vec{y}$?
3. What is the angle between $2\vec{x}$ and $-3\vec{y}$?

Dot Product and the Vector Norm. What is the dot product of a vector with itself? If $\vec{v} \in \mathbb{R}^n$, then:

$$\begin{aligned}\vec{v} \cdot \vec{v} &= v_1 v_1 + v_2 v_2 + \cdots + v_n v_n \\ &= {v_1}^2 + {v_2}^2 + \cdots + {v_n}^2 \\ &= \|\vec{v}\|^2\end{aligned}$$

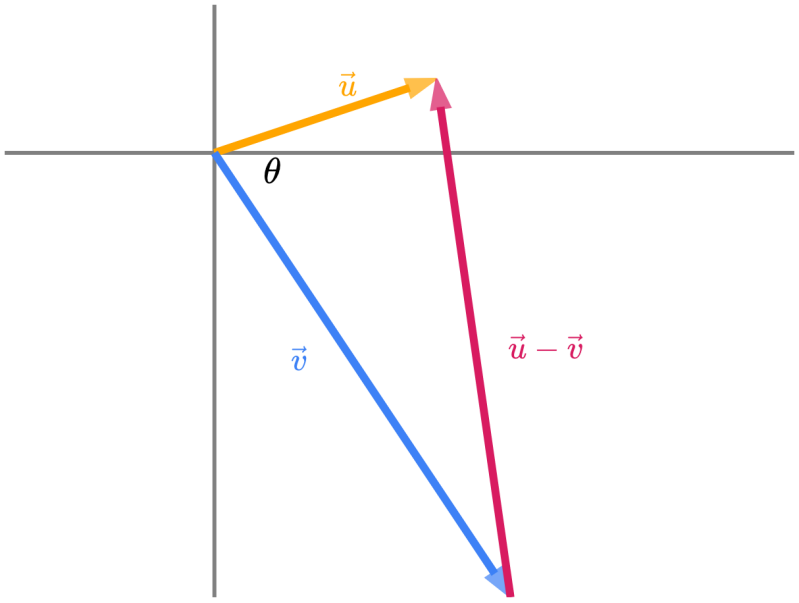
The fact that $\boxed{\vec{v} \cdot \vec{v} = \|\vec{v}\|^2}$ unlocks a variety of powerful analyses, and it's such a core definition that I've boxed it.

For example, we now have the tools to prove that the geometric cosine definition of the dot product is equal to the computational definition! Let's try and show that:

$$u_1 v_1 + u_2 v_2 + \cdots + u_n v_n = \|\vec{u}\| \|\vec{v}\| \cos\theta$$

Let's consider two arbitrary vectors in $\mathbb{R}^n$, $\vec{u}$ and $\vec{v}$. (I've drawn them below as vectors in $\mathbb{R}^2$, but we won't assume anything in particular about two dimensional space, and we won't put specific numbers to them, since our proof should be general.)

Along with them, let's consider their **difference**, $\vec{u} - \vec{v}$. This step may seem arbitrary, but we'll see why it's useful soon.



A confusing concept is whether the tip of the vector $\vec{u} - \vec{v}$ should be at the tip of $\vec{u}$ or the tip of $\vec{v}$. To verify that the above diagram is correct, note that if you walk along the length of $\vec{v}$, then along the length of $\vec{u} - \vec{v}$, you end up at the tip of $\vec{u}$, which matches what we'd expect from the expression $\vec{v} + (\vec{u} - \vec{v}) = \vec{u}$.

I'd like to try and find an expression involving θ (the angle between $\vec{u}$ and $\vec{v}$) and the dot product of $\vec{u}$ and $\vec{v}$, without using the cosine definition of the dot product (since that's what I'm trying to prove).

1. First, let's consider a rule we perhaps haven't touched in a few years: the **cosine law**. The cosine law says that for any triangle with sides of length a, b, and c, with an angle of C opposite side c,

$$c^2 = a^2 + b^2 - 2ab\cos C$$

   We can apply this rule to the triangle formed by $\vec{u}$, $\vec{v}$, and $\vec{u} - \vec{v}$ (the dashed line in the diagram above). The cosine law tells us that

$$\|\vec{u} - \vec{v}\|^2 = \|\vec{u}\|^2 + \|\vec{v}\|^2 - 2\|\vec{u}\|\|\vec{v}\|\cos\theta$$

   There's not much more I can do with this right now.

2. Above, it'd be nice to have an expression for $\|\vec{u} - \vec{v}\|^2$ that involves the dot product of $\vec{u}$ and $\vec{v}$. Let's try and find one. I will use the fact that $\|\vec{u} - \vec{v}\|^2 = (\vec{u} - \vec{v}) \cdot (\vec{u} - \vec{v})$.

$$\begin{aligned}
\|\vec{u} - \vec{v}\|^2 &= (\vec{u} - \vec{v}) \cdot (\vec{u} - \vec{v}) \\
&= \vec{u} \cdot \vec{u} \quad \underbrace{-\vec{u} \cdot \vec{v} - \vec{v} \cdot \vec{u}}_{\text{why are both terms are the same?}} \quad + \vec{v} \cdot \vec{v} \\
&= \vec{u} \cdot \vec{u} - 2\vec{u} \cdot \vec{v} + \vec{v} \cdot \vec{v} \\
&= \|\vec{u}\|^2 + \|\vec{v}\|^2 - 2\vec{u} \cdot \vec{v}
\end{aligned}$$

Let's take a step back. Independently, we've found two expressions for $\|\vec{u} - \vec{v}\|^2$:

1. $\|\vec{u} - \vec{v}\|^2 = \|\vec{u}\|^2 + \|\vec{v}\|^2 - 2\|\vec{u}\|\|\vec{v}\|\cos\theta$
2. $\|\vec{u} - \vec{v}\|^2 = \|\vec{u}\|^2 + \|\vec{v}\|^2 - 2\vec{u} \cdot \vec{v}$

These must be equal! Equating the two expressions on the right-hand sides gives us

$$\|\vec{u}\|^2 + \|\vec{v}\|^2 - 2\|\vec{u}\|\|\vec{v}\|\cos\theta = \|\vec{u}\|^2 + \|\vec{v}\|^2 - 2\vec{u} \cdot \vec{v}$$

Subtracting the common terms from both sides gives us

$$-2\|\vec{u}\|\|\vec{v}\|\cos\theta = -2\vec{u} \cdot \vec{v}$$

And finally, dividing both sides by -2 gives us

$$\boxed{\|\vec{u}\|\|\vec{v}\|\cos\theta = \vec{u} \cdot \vec{v}}$$

This completes the proof that the two formulas for the dot product are equivalent! **This is an extremely important proof, and proofs of this type will appear in labs, homeworks, and exams moving forward.** You're not expected to remember the cosine law from memory, but given the cosine law, you eventually will need to be able to produce something like this on your own.

An implication of this equality, as we saw in Activity 2, is that the angle between $\vec{u}$ and $\vec{v}$ can be found by using

$$\cos\theta = \frac{\vec{u}\cdot\vec{v}}{\|\vec{u}\|\,\|\vec{v}\|}$$

There are some important interpretations of the expression $\frac{\vec{u}\cdot\vec{v}}{\|\vec{u}\|\|\vec{v}\|}$.

In machine learning, we often call it the **cosine similarity** between $\vec{u}$ and $\vec{v}$, because it is the cosine of the angle between the two vectors. A cosine similarity value of 1 implies that the vectors point in the same direction; a value of -1 implies that they point in opposite directions, and a value of 0 implies that they are orthogonal. **The main idea is that dot products help measure similarity.**

The cosine similarity can also be thought of as a **normalized dot product**, where we start by the dot product and divide by the product of the norms of the two vectors. You can also view it as the dot product of unit vectors $\vec{U} = \frac{\vec{u}}{\|\vec{u}\|}$ and $\vec{V} = \frac{\vec{v}}{\|\vec{v}\|}$.

Note that cosine similarity ranges between -1 and 1 (like the correlation coefficient!), while the dot product ranges between $-\|\vec{u}\|\|\vec{v}\|$ and $\|\vec{u}\|\|\vec{v}\|$. This means that we can compare the cosine similarities of several pairs of vectors together; the "normalization" by the norms of the vectors allows us to make meaningful comparisons, independent of the norms of the vectors. (If we didn't normalize, then the dot product of two vectors with large norms would always be larger than the dot product of two vectors with small norms, even if the two vectors with large magnitudes weren't very similar.)

Activity 5

Activity 5

If $\|\vec{x}\| = 5$ and $\|\vec{y}\| = 12$, what is the largest possible value of $\vec{x}\cdot\vec{y}$? What is the smallest possible value? (Taken from Gilbert Strang's Linear Algebra book.)

Cauchy-Schwarz and Triangle Inequalities. In Chapter 3.1, I stated – without proof! – that the vector norm satisfies the **triangle inequality**, which says that for any vectors $\vec{u}$ and $\vec{v}$ in $\mathbb{R}^n$,

$$\|\vec{u}+\vec{v}\| \le \|\vec{u}\| + \|\vec{v}\|$$

Remember, intuitively, this says that the length of one side of a triangle cannot be longer than the sum of the lengths of the other two sides, if you consider the triangle formed by $\vec{u}$, $\vec{v}$, and $\vec{u}+\vec{v}$.

We now have the tools to prove this. But first, let me start by introducing a new inequality, the **Cauchy-Schwarz inequality**. This inequality says that for any vectors $\vec{u}$ and $\vec{v}$ in $\mathbb{R}^n$,

$$|\vec{u}\cdot\vec{v}| \le \|\vec{u}\|\,\|\vec{v}\|$$

Gilbert Strang's book calls the Cauchy-Schwarz inequality the most important inequality in mathematics, and the instructors of future machine learning courses specifically requested for us to put it in EECS 245.

Why is the Cauchy-Schwarz inequality true? Try and reason about it yourself.

Solution

The geometric definition of the dot product says

$$\vec{u} \cdot \vec{v} = \|\vec{u}\| \|\vec{v}\| \cos\theta$$

Since $-1 \leq \cos\theta \leq 1$, we have that:

$$-\|\vec{u}\| \|\vec{v}\| \leq \vec{u} \cdot \vec{v} \leq \|\vec{u}\| \|\vec{v}\|$$

Or, in equivalently,

$$|\vec{u} \cdot \vec{v}| \leq \|\vec{u}\| \|\vec{v}\|$$

Equipped with the Cauchy-Schwarz inequality, we can now prove the triangle inequality. I want to show that $\|\vec{u} + \vec{v}\| \leq \|\vec{u}\| + \|\vec{v}\|$.

Let me start by expanding $\|\vec{u} + \vec{v}\|^2$:

$$\begin{aligned} \|\vec{u} + \vec{v}\|^2 &= (\vec{u} + \vec{v}) \cdot (\vec{u} + \vec{v}) \\ &= \|\vec{u}\|^2 + 2\vec{u} \cdot \vec{v} + \|\vec{v}\|^2 \end{aligned}$$

We can't use the Cauchy-Schwarz inequality here just yet, because it says something about $|\vec{u} \cdot \vec{v}|$ but there isn't an absolute value around $\vec{u} \cdot \vec{v}$ above. But, we can use the fact that:

$$x \leq |x|$$

for any and all $x \in \mathbb{R}$. (The absolute value is just x if x is positive, and is still positive even when x is negative, so $|x|$ can never be less than x.)

Applying this to $\vec{u} \cdot \vec{v}$, we get:

$$\vec{u} \cdot \vec{v} \leq |\vec{u} \cdot \vec{v}|$$

Then, from the Cauchy-Schwarz inequality, we know that:

$$|\vec{u} \cdot \vec{v}| \leq \|\vec{u}\| \|\vec{v}\|$$

Putting these two inequalities together, we get:

$$\vec{u} \cdot \vec{v} \leq \|\vec{u}\| \|\vec{v}\|$$

Back to the main proof. Using the most recent inequality above, we have:

$$\begin{aligned}\|\vec{u}+\vec{v}\|^2 &= (\vec{u}+\vec{v})\cdot(\vec{u}+\vec{v}) \\ &= \|\vec{u}\|^2 + 2\vec{u}\cdot\vec{v} + \|\vec{v}\|^2 \\ &\leq \|\vec{u}\|^2 + 2\left\|\vec{u}\right\|\left\|\vec{v}\right\| + \|\vec{v}\|^2 \\ &= \left(\|\vec{u}\| + \|\vec{v}\|\right)^2\end{aligned}$$

Taking the square root of both sides, we get:

$$\|\vec{u}+\vec{v}\| \leq \|\vec{u}\| + \|\vec{v}\|$$

This completes the proof of the triangle inequality!

Recap

- The **Cauchy-Schwarz inequality** says that $|\vec{u}\cdot\vec{v}| \leq \|\vec{u}\|\|\vec{v}\|$.
- The **triangle inequality** says that $\|\vec{u}+\vec{v}\| \leq \|\vec{u}\| + \|\vec{v}\|$.

Both of these inequalities are true for any vectors $\vec{u}, \vec{v} \in \mathbb{R}^n$. However:

- The **Cauchy-Schwarz inequality**, the way we've presented it, is true only in general for the L_2 norm. The dot product and L_2 norm have a close relationship, and we derived the Cauchy-Schwarz inequality using the geometric definition of the dot product.
- The **triangle inequality** is true for any vector norm. Here, we've discussed it for the L_2 norm, but it's true for L_1, L_∞, and any other norm. (In fact, it's one of the defining properties of a norm!)

Activity 6

Activity 6

1. What must be true about $\vec{u}$ and $\vec{v}$ if equality holds in the Cauchy-Schwarz inequality? That is, if $|\vec{u}\cdot\vec{v}| = \|\vec{u}\|\|\vec{v}\|$, what relationship do $\vec{u}$ and $\vec{v}$ have?
2. What must be true about $\vec{u}$ and $\vec{v}$ if equality holds in the triangle inequality? That is, if $\|\vec{u}+\vec{v}\| = \|\vec{u}\| + \|\vec{v}\|$, what relationship do $\vec{u}$ and $\vec{v}$ have? **In what case is there equality for Cauchy-Schwarz but NOT the triangle inequality?**

Activity 7

Activity 7

Let $\vec{u} = \begin{bmatrix}\sqrt{a}\\ \sqrt{b}\end{bmatrix}$ and $\vec{v} = \begin{bmatrix}\sqrt{b}\\ \sqrt{a}\end{bmatrix}$.
Using the Cauchy-Schwarz inequality, prove that the geometric mean of a and b is less than or equal to the arithmetic mean of a and b.

3.4. Projecting onto a Single Vector

The Approximation Problem

Suppose $\vec{u}$ and $\vec{v}$ are any two vectors in $\mathbb{R}^n$. (When I say this, I just mean that they both have the same number of components.)

Let's think about all possible vectors of the form $k\vec{v}$, where k can be any scalar. Any vector of the form $k\vec{v}$ is a scalar multiple of $\vec{v}$, and points in the same direction as $\vec{v}$ (if $k > 0$) or the opposite direction (if $k < 0$). What is different about these scalar multiples is how long they are.

The Approximation Problem

Among all vectors of the form $k\vec{v}$, which one is closest to $\vec{u}$?

To get a sense of what I mean by this, play with the slider for k below. There are three vectors being visualized: some $\vec{u}$, some $\vec{v}$, and $k\vec{v}$, which depends on the value of k you choose.

Notice that the set of vectors of the form $k\vec{v}$ fill out a **line**. So really, what we're asking is which vector on this line is closest to $\vec{u}$.

In terms of angles, if θ is the angle between $\vec{u}$ and $\vec{v}$, then the angle between $\vec{u}$ and $k\vec{v}$ is either θ (if $k > 0$) or $180° - \theta$ (if $k < 0$). So changing k doesn't change how "similar" $\vec{u}$ and $k\vec{v}$ are in the cosine similarity sense.

But, some choices of k will make $k\vec{v}$ closer to $\vec{u}$ than others. I call this the **approximation problem**: how well can we recreate, or approximate, $\vec{u}$ using a scalar multiple of $\vec{v}$? It turns out that linear regression is intimately related to this problem. Previously, we were trying to approximate commute times as best as we could using a linear function of departure times.

Let's be more precise by what we mean by "closer". For any value of k, we can measure the **error** of our approximation by the length of the error vector, $\vec{e} = \vec{u} - k\vec{v}$.

Continue to play with the slider above for k. How do you get the length of the error vector to be as small as possible?

Intuitively, it seems that to get the error vector $\vec{e}$ to be as short as possible, we should make it **orthogonal** to $\vec{v}$. Since we can control k, we can control $\vec{u} - k\vec{v}$, so we can make the error vector orthogonal to $\vec{v}$ by choosing the right k.

Orthogonal Projections

Our goal is to minimize the length of the error vector $\|\vec{e}\|$.

$$\|\vec{e}\|$$

This is the same as minimizing

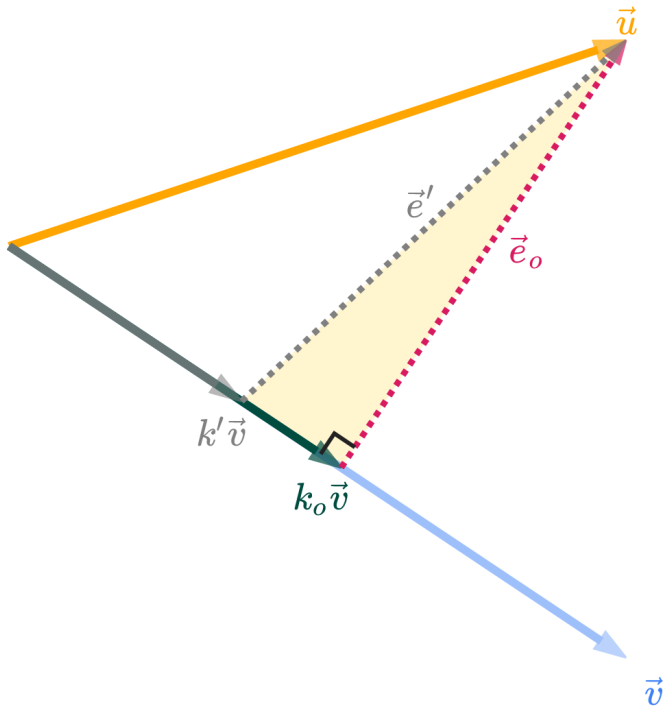
$$\|\vec{u} - k\vec{v}\|$$

One way to approach this problem is to treat the above expression as a function of k and find the value of k that minimizes it through calculus. **You'll do this in Homework 3.** I'll show you a more geometric approach here.

We've *guessed, but not yet shown* that the shortest possible error vector is the one that is orthogonal to $\vec{v}$. Let k_o be the value of k that makes the error vector $\vec{e}_o = \vec{u} - k_o\vec{v}$ **orthogonal** to $\vec{v}$. Here, we'll prove that k_o is the "best"

choice of k by showing that any other choice of k will result in an error vector that is longer than $\vec{e}_o$.

For comparison, let k' be some other value of k, and let its error vector be $\vec{e}' = \vec{u} - k'\vec{v}$.



I've drawn $k'\vec{v}$ in gray. Arbitrarily, I've shown it as being shorter than $k_o\vec{v}$, but I could have drawn it as being longer and the argument would be the same. **The prime has nothing to do with derivatives**, by the way – it's just a new variable.

The vectors $k'\vec{v}$ and $k_o\vec{v}$, along with their corresponding error vectors, create a right-angled triangle, shaded in gold above. This triangle has a hypotenuse of $\vec{e}'$ and legs of $\vec{e}_o$ and $k'\vec{v} - k_o\vec{v}$.

Applying the Pythagorean theorem to this triangle gives us

$$\|\vec{e}'\|^2 = \|\vec{e}_o\|^2 + \underbrace{\|k'\vec{v} - k_o\vec{v}\|^2}_{>0}$$

$\|k'\vec{v} - k_o\vec{v}\|^2 \geq 0$, with equality only when we choose $k' = k_o$, but we've assumed that $k' \neq k_o$.

This implies that

$$\|\vec{e}'\|^2 = \|\vec{e}_o\|^2 + \text{some positive number}$$

which means that $\|\vec{e}'\|^2 > \|\vec{e}_o\|^2$.

In other words, if $k' \neq k_o$, then $\|\vec{e}'\| > \|\vec{e}_o\|$. Thus, the error vector $\vec{e}_o$ is shorter than any other error vector $\vec{e}'$, and the best choice of k is k_o!

What was the point of all this again?

The Approximation Problem

Among all vectors of the form $k\vec{v}$, which one is closest to $\vec{u}$?

We know that the answer is $k^*\vec{v}$, where k^* is the value of k that makes the error vector $\vec{e} = \vec{u} - k^*\vec{v}$ orthogonal to $\vec{v}$. (I've switched from calling this optimal scalar k_o to k^*; k_o was a name I used in the proof above, but more generally, "optimal" values are starred for our purposes).

Let's now find the value of k^*, in terms of just $\vec{u}$ and $\vec{v}$. If $\vec{e}$ is orthogonal to $\vec{v}$, then $\vec{v} \cdot \vec{e} = 0$.

$\cdot\vec{e} = 0$

$\cdot(\vec{u} - k^*\vec{v}) = 0$

$\cdot\vec{u} - \vec{v} \cdot k^*\vec{v} = 0$

$\cdot\vec{u} - k^*(\vec{v} \cdot \vec{v}) = 0$

$\cdot\vec{u} = k^*(\vec{v} \cdot \vec{v})$

$$k^* = \boxed{\frac{\vec{u} \cdot \vec{v}}{\vec{v} \cdot \vec{v}}}$$

The boxed value above is a **scalar**. It tells us the optimal amount to multiply $\vec{v}$ by to get the best approximation of $\vec{u}$. Once we multiply that boxed scalar, k^*, by the vector $\vec{v}$, we get what's called the **orthogonal projection of $\vec{u}$ onto $\vec{v}$**.

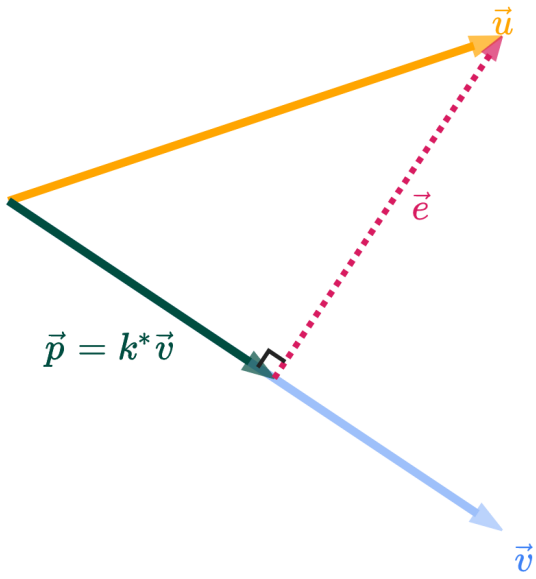
Definition: Orthogonal Projection

Suppose $\vec{u}$ and $\vec{v}$ are vectors in $\mathbb{R}^n$. The **orthogonal projection of $\vec{u}$ onto $\vec{v}$** is the vector

$$\vec{p} = \left(\frac{\vec{u} \cdot \vec{v}}{\vec{v} \cdot \vec{v}}\right)\vec{v}$$

Among all vectors of the form $k\vec{v}$, $\vec{p}$ above is the one that is closest to $\vec{u}$.

Why "orthogonal projection"? "Orthogonal" comes from the fact that $\vec{p}$'s error vector $\vec{e} = \vec{u} - \vec{p}$ is orthogonal to $\vec{v}$. "Projection" comes from the intuition you should have that $\vec{p}$ is the **shadow** of $\vec{u}$ onto $\vec{v}$.



We've defined the error vector as $\vec{e} = \vec{u} - \vec{p}$, and we know that $\vec{e}$ is orthogonal to $\vec{v}$. Rearranging the definition of the error vector gives us

$$\vec{u} = \underbrace{\vec{p}}_{\text{parallel to } \vec{v}} + \underbrace{\vec{e}}_{\text{orthogonal to } \vec{v}}$$

All this says is that $\vec{u}$ is the sum of:

- $\vec{p}$, which is parallel to $\vec{v}$ (by definition of orthogonal projection)
- $\vec{e}$, which is orthogonal to $\vec{v}$ (by definition of error vector)

Sometimes, we call this the **orthogonal decomposition** of $\vec{u}$ with respect to $\vec{v}$. I'll speak more about decompositions later in this section.

Examples

Let's make things concrete by working through several examples. Each one was carefully chosen to illustrate something in particular.

We will work through several of these examples in lecture; attempt the ones that we don't on your own.

Example: Fundamentals. Let $\vec{u} = \begin{bmatrix} 1 \\ 2 \\ 2 \end{bmatrix}$ and $\vec{v} = \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix}$.

1. Find the orthogonal projection of $\vec{u}$ onto $\vec{v}$.
2. Find the error vector, i.e. the vector $\vec{e} = \vec{u} - \vec{p}$, and verify that it is orthogonal to $\vec{v}$.
3. What is the length of the error vector (i.e. the **projection error**)?

Solution

Part 1
The orthogonal projection of $\vec{u}$ onto $\vec{v}$ is given by

$$\vec{p} = \frac{\vec{u} \cdot \vec{v}}{\vec{v} \cdot \vec{v}} \vec{v}$$

Following the same steps as in the previous example, we have

$$\vec{u} \cdot \vec{v} = \begin{bmatrix} 1 \\ 2 \\ 2 \end{bmatrix} \cdot \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix} = 1 + 2 + 2 = 5$$

$$\vec{v} \cdot \vec{v} = \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix} \cdot \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix} = 1 + 1 + 1 = 3$$

So, the orthogonal projection of $\vec{u}$ onto $\vec{v}$ is

$$\vec{p} = \frac{5}{3} \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix} = \begin{bmatrix} 5/3 \\ 5/3 \\ 5/3 \end{bmatrix}$$

Part 2
The error vector is

$$\vec{e} = \vec{u} - \vec{p} = \begin{bmatrix} 1 \\ 2 \\ 2 \end{bmatrix} - \begin{bmatrix} 5/3 \\ 5/3 \\ 5/3 \end{bmatrix} = \begin{bmatrix} 1 - 5/3 \\ 2 - 5/3 \\ 2 - 5/3 \end{bmatrix} = \begin{bmatrix} -2/3 \\ 1/3 \\ 1/3 \end{bmatrix}$$

To check if it's orthogonal to $\vec{v}$, we compute their dot product; we're hoping it's 0.

$$\vec{e} \cdot \vec{v} = \begin{bmatrix} -2/3 \\ 1/3 \\ 1/3 \end{bmatrix} \cdot \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix} = -2/3 + 1/3 + 1/3 = 0$$

So, the error vector $\vec{e}$ is orthogonal to $\vec{v}$.
Part 3
The length of the error vector is

$$\begin{aligned} \|\vec{e}\| &= \sqrt{(-2/3)^2 + (1/3)^2 + (1/3)^2} \\ &= \sqrt{4/9 + 1/9 + 1/9} \\ &= \sqrt{6/9} = \sqrt{2/3} \end{aligned}$$

We might say $\sqrt{2/3}$ is the **projection error**. Another way of thinking of it is as the shortest distance from the point $(1, 2, 2)$ to the line that passes through $(0, 0, 0)$ and $(1, 1, 1)$.

Example: The Line Perspective. Consider the points $p_1 = (1, 2, 2)$ and $p_2 = (1, 1, 1)$ in $\mathbb{R}^3$.
What is the shortest distance between p_1 and the **line** that passes through p_2 and the origin, $(0, 0, 0)$?

Solution

The answer is $\sqrt{2/3}$. This example didn't require any addition math beyond the previous example; it just serves to remind you of the geometry of the situation. The set of all possible scalar multiples of $\vec{v}$ fill out a **line** in $\mathbb{R}^3$, and that line passes through $(0, 0, 0)$ and $p_2 = (1, 1, 1)$.
Why does that line pass through $(0, 0, 0)$? Consider the vector $0\vec{v}$ – it's the zero vector!

Example: Which Order?. Let $\vec{u} = \begin{bmatrix} 1 \\ 2 \\ 2 \end{bmatrix}$ and $\vec{v} = \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix}$.

In the first example, we found the orthogonal projection of $\vec{u}$ onto $\vec{v}$.

Now, do the opposite: find the orthogonal projection of $\vec{v}$ onto $\vec{u}$.

Solution

Now, we're projecting $\vec{v}$ onto $\vec{u}$, which means our answer is going to be a multiple of $\vec{u}$, not $\vec{v}$ as in the first part.
The orthogonal projection of $\vec{v}$ onto $\vec{u}$ is given by

$$\vec{p} = \frac{\vec{v} \cdot \vec{u}}{\vec{u} \cdot \vec{u}} \vec{u}$$

The formula for the scalar in front of $\vec{u}$ is the same as in Part 1, but with all $\vec{v}$'s replaced by $\vec{u}$'s and vice versa. The numerator is the same, since $\vec{u} \cdot \vec{v} = \vec{v} \cdot \vec{u}$. The denominator is different; **just remember that the denominator is the squared norm of the vector you're projecting onto**.

$$\vec{v} \cdot \vec{u} = \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix} \cdot \begin{bmatrix} 1 \\ 2 \\ 2 \end{bmatrix} = 1 + 2 + 2 = 5$$

$$\vec{u} \cdot \vec{u} = \begin{bmatrix} 1 \\ 2 \\ 2 \end{bmatrix} \cdot \begin{bmatrix} 1 \\ 2 \\ 2 \end{bmatrix} = 1 + 4 + 4 = 9$$

So, the orthogonal projection of $\vec{v}$ onto $\vec{u}$ is

$$\vec{p} = \frac{\vec{v} \cdot \vec{u}}{\vec{u} \cdot \vec{u}} \vec{u} = \frac{5}{9} \begin{bmatrix} 1 \\ 2 \\ 2 \end{bmatrix} = \begin{bmatrix} 5/9 \\ 10/9 \\ 10/9 \end{bmatrix}$$

Note that the corresponding error vector, $\vec{e} = \vec{v} - \vec{p}$, is orthogonal to $\vec{u}$ (**not** $\vec{v}$), since $\vec{u}$ is the vector we projected onto.

Example: Unit Vectors. Let $\vec{u} = \begin{bmatrix} 4 \\ -1 \\ 2 \end{bmatrix}$, $\vec{v} = \begin{bmatrix} 2 \\ 6 \\ -3 \end{bmatrix}$ and $\vec{V} = \begin{bmatrix} 2/7 \\ 6/7 \\ -3/7 \end{bmatrix}$.

1. Find the orthogonal projection of $\vec{u}$ onto $\vec{v}$.

2. Find the orthogonal projection of $\vec{u}$ onto $\vec{V}$.
3. What do you notice about your answers to the above two parts?

Solution

Part 1
We know that the orthogonal projection of $\vec{u}$ onto $\vec{v}$ is given by

$$\vec{p} = \frac{\vec{u} \cdot \vec{v}}{\vec{v} \cdot \vec{v}} \vec{v}$$

Let's compute the relevant dot products.

$$\vec{u} \cdot \vec{v} = \begin{bmatrix} 4 \\ -1 \\ 2 \end{bmatrix} \cdot \begin{bmatrix} 2 \\ 6 \\ -3 \end{bmatrix} = 4 \cdot 2 + (-1) \cdot 6 + 2 \cdot (-3) = 8 - 6 - 6 = -4$$

$$\vec{v} \cdot \vec{v} = \begin{bmatrix} 2 \\ 6 \\ -3 \end{bmatrix} \cdot \begin{bmatrix} 2 \\ 6 \\ -3 \end{bmatrix} = 2 \cdot 2 + 6 \cdot 6 + (-3) \cdot (-3) = 4 + 36 + 9 = 49$$

So, the orthogonal projection of $\vec{u}$ onto $\vec{v}$ is

$$\vec{p} = \frac{-4}{49} \begin{bmatrix} 2 \\ 6 \\ -3 \end{bmatrix} = \begin{bmatrix} -8/49 \\ -24/49 \\ 12/49 \end{bmatrix}$$

Part 2
Now, we need to find $\frac{\vec{u} \cdot \vec{V}}{\vec{V} \cdot \vec{V}} \vec{V}$.
Let's compute the relevant dot products.

$$\vec{u} \cdot \vec{V} = \begin{bmatrix} 4 \\ -1 \\ 2 \end{bmatrix} \cdot \begin{bmatrix} 2/7 \\ 6/7 \\ -3/7 \end{bmatrix} = \frac{8}{7} - \frac{6}{7} - \frac{6}{7} = -\frac{4}{7}$$

$$\vec{V} \cdot \vec{V} = \begin{bmatrix} 2/7 \\ 6/7 \\ -3/7 \end{bmatrix} \cdot \begin{bmatrix} 2/7 \\ 6/7 \\ -3/7 \end{bmatrix} = \frac{4}{49} + \frac{36}{49} + \frac{9}{49} = \frac{49}{49} = 1$$

So, the orthogonal projection of $\vec{u}$ onto $\vec{V}$ is

$$\vec{p} = \frac{-\frac{4}{7}}{1} \begin{bmatrix} 2/7 \\ 6/7 \\ -3/7 \end{bmatrix} = \begin{bmatrix} -8/49 \\ -24/49 \\ 12/49 \end{bmatrix}$$

Part 3
Notice that in both parts, the orthogonal projection $\vec{p}$ is the same! This is not a coincidence. Both vectors point in the same direction, meaning the set of possible vectors of the form $k\vec{V}$ is the same as the set of possible vectors of the form $k\vec{v}$. Another way to think about this is that they both span the same **line** in $\mathbb{R}^3$ through the origin.
The difference between $\vec{v}$ and $\vec{V}$ is that $\vec{V}$ is a unit vector in the direction of $\vec{v}$, meaning that it points in the same direction as $\vec{v}$ but has $\|\vec{V}\| = 1$ rather than $\|\vec{v}\| = 7$.
What's different is the scalar we need to multiply each vector by to get the orthogonal projection. In the case of the unit vector $\vec{V}$, the number in front of $\vec{V}$ is $\frac{\vec{u} \cdot \vec{V}}{\vec{V} \cdot \vec{V}}$, but since $\vec{V} \cdot \vec{V} = \|\vec{V}\|^2 = 1$, this simplifies to $\vec{u} \cdot \vec{V}$.

Example: Unit Vectors, Continued. Suppose $\vec{u}, \vec{v} \in \mathbb{R}^n$. Let θ be the angle between $\vec{u}$ and $\vec{v}$. Show that the orthogonal projection of $\vec{u}$ onto $\vec{v}$ is equal to

$$\vec{p} = \left(\|\vec{u}\| \cos\theta\right) \frac{\vec{v}}{\|\vec{v}\|}$$

This is not a formula we'd use to actually compute $\vec{p}$, since finding $\cos\theta$ is harder than using the dot product-based formula from above. But, what does this formula tell us about the relationship between $\vec{u}$ and $\vec{p}$?

Solution

Let's start with the original formula for the orthogonal projection of $\vec{u}$ onto $\vec{v}$.

$$\vec{p} = \frac{\vec{u} \cdot \vec{v}}{\vec{v} \cdot \vec{v}} \vec{v}$$

Using the fact that $\vec{u} \cdot \vec{v} = \|\vec{u}\| \|\vec{v}\| \cos\theta$ and $\vec{v} \cdot \vec{v} = \|\vec{v}\|^2$, we can rewrite the formula as

$$\vec{p} = \frac{\|\vec{u}\| \|\vec{v}\| \cos\theta}{\|\vec{v}\|^2} \vec{v} = \frac{\|\vec{u}\| \cos\theta}{\|\vec{v}\|} \vec{v} = \left(\|\vec{u}\| \cos\theta\right) \frac{\vec{v}}{\|\vec{v}\|}$$

The parentheses around $\|\vec{u}\| \cos\theta$ don't change the calculation, but they help with the interpretation. This shows us that we can think of the orthogonal projection of $\vec{u}$ onto $\vec{v}$ as a vector with:

- a length of $\|\vec{u}\| \cos\theta$
- in the direction of $\frac{\vec{v}}{\|\vec{v}\|}$, which is a unit vector in the direction of $\vec{v}$

Example: Projecting onto an Orthogonal Vector. Let $\vec{u} = \begin{bmatrix} 4 \\ -1 \\ 2 \end{bmatrix}$ and $\vec{v} = \begin{bmatrix} -3 \\ 0 \\ 6 \end{bmatrix}$.

$\vec{u}$ and $\vec{v}$ are orthogonal. What does this say about the orthogonal projection of $\vec{u}$ onto $\vec{v}$?

Solution

Since $\vec{u} \cdot \vec{v} = 0$, the orthogonal projection of $\vec{u}$ onto $\vec{v}$ is the zero vector, $\vec{0}$.

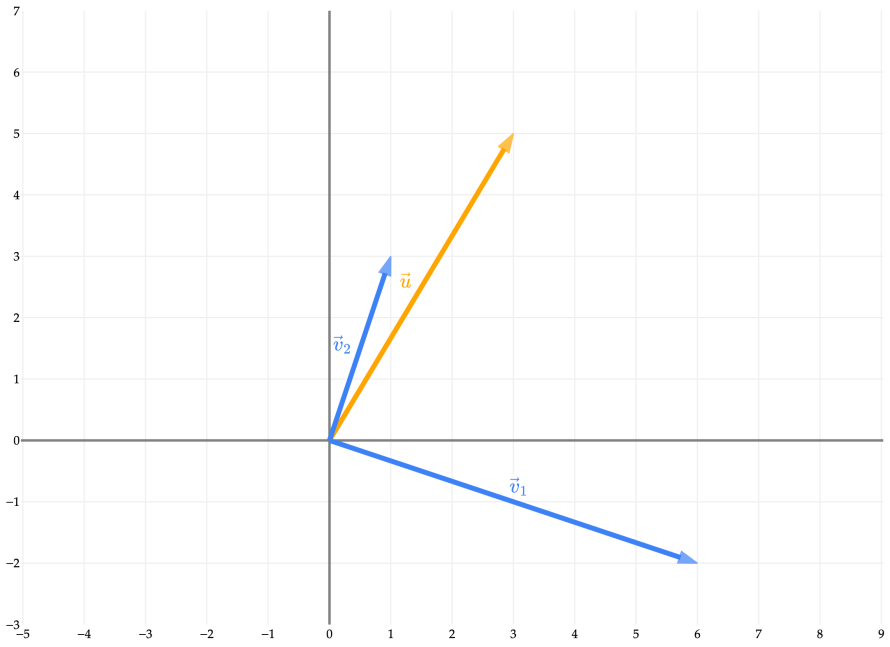
$$p = \frac{\vec{u} \cdot \vec{v}}{\vec{v} \cdot \vec{v}} \vec{v} = \frac{0}{36} \vec{v} = \vec{0}$$

Intuitively, $\vec{u}$ and $\vec{v}$ travel in totally different different directions. Travelling any amount of $\vec{v}$ will take you further away from $\vec{u}$. So, it's best to stick with the zero vector, $\vec{0}$.

Orthogonal Decomposition

To motivate the next section, let's consider another example.

Let $\vec{u} = \begin{bmatrix} 3 \\ 5 \end{bmatrix}$, $\vec{v}_1 = \begin{bmatrix} 6 \\ -2 \end{bmatrix}$, and $\vec{v}_2 = \begin{bmatrix} 1 \\ 3 \end{bmatrix}$.



Notice that $\vec{v}_1$ and $\vec{v}_2$ are orthogonal – that's important to what we're about to discover.

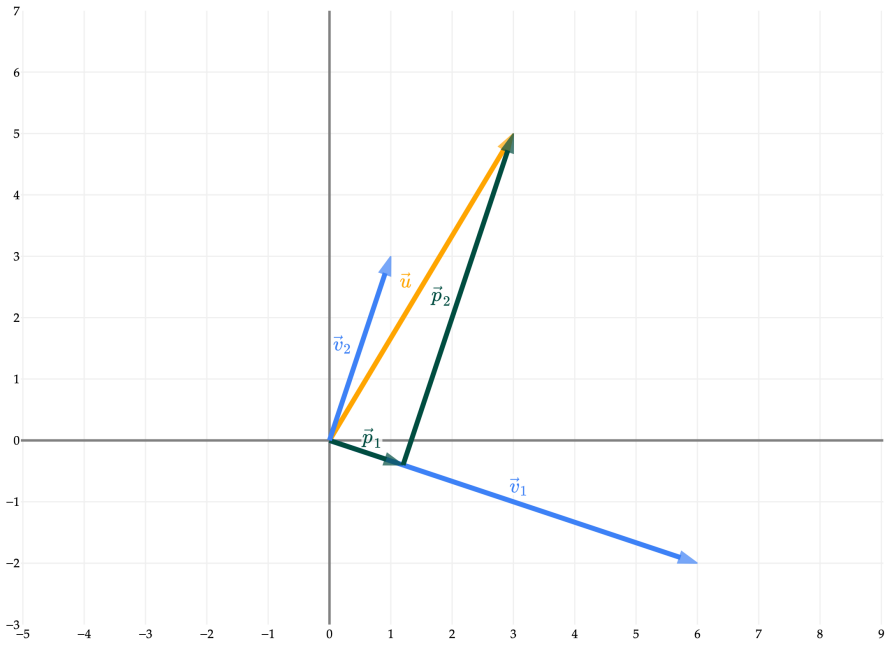
Let's find the orthogonal projection of $\vec{u}$ onto $\vec{v}_1$ (called $\vec{p}_1$) and the orthogonal projection of $\vec{u}$ onto $\vec{v}_2$ (called $\vec{p}_2$).

$$\vec{p}_1 = \frac{\vec{u} \cdot \vec{v}_1}{\vec{v}_1 \cdot \vec{v}_1}\vec{v}_1 = \frac{8}{40}\vec{v}_1 = \frac{1}{5}\vec{v}_1 = \begin{bmatrix} 6/5 \\ -2/5 \end{bmatrix}$$

$$\vec{p}_2 = \frac{\vec{u} \cdot \vec{v}_2}{\vec{v}_2 \cdot \vec{v}_2}\vec{v}_2 = \frac{18}{10}\vec{v}_2 = \frac{9}{5}\vec{v}_2 = \begin{bmatrix} 9/5 \\ 27/5 \end{bmatrix}$$

Notice that $\vec{u}$ is the sum of $\vec{p}_1$ and $\vec{p}_2$!

$$\vec{u} = \vec{p}_1 + \vec{p}_2 = \underbrace{\frac{1}{5}\vec{v}_1 + \frac{9}{5}\vec{v}_2}_{\text{linear combination}} = \begin{bmatrix} 6/5 \\ -2/5 \end{bmatrix} + \begin{bmatrix} 9/5 \\ 27/5 \end{bmatrix} = \begin{bmatrix} 3 \\ 5 \end{bmatrix}$$



Why is the sum of $\vec{p}_1$ and $\vec{p}_2$ equal to $\vec{u}$? Earlier, I mentioned that we can use orthogonal projections to **decompose** vectors. Here, when we project $\vec{u}$ onto $\vec{v}_1$, the corresponding error vector $\vec{e}_1$ is orthogonal to $\vec{v}_1$.

$$\vec{u} = \underbrace{\vec{p}_1}_{\text{parallel to } \vec{v}_1} + \underbrace{\vec{e}_1}_{\text{orthogonal to } \vec{v}_1}$$

By projecting $\vec{u}$ onto $\vec{v}_2$, we can recreate the error vector exactly, meaning $\vec{e}_1 = \vec{p}_2$.

Taking a step back, the fact that v_1 and v_2 are orthogonal meant that writing $\vec{u}$ as a linear combination of v_1 and v_2 was easy.

$$a_1 \underbrace{\begin{bmatrix} 6 \\ -2 \end{bmatrix}}_{\vec{v}_1} + a_2 \underbrace{\begin{bmatrix} 1 \\ 3 \end{bmatrix}}_{\vec{v}_2} = \underbrace{\begin{bmatrix} 3 \\ 5 \end{bmatrix}}_{\vec{u}}$$

If v_1 and v_2 were not orthogonal, then writing $\vec{u}$ as a linear combination of v_1 and v_2 would have involved solving a system of 2 equations and 2 unknowns, as we've had to do in previous sections.

For instance, if we keep $\begin{bmatrix} 3 \\ 5 \end{bmatrix}$ and look at $\vec{x}_1 = \begin{bmatrix} 1 \\ 2 \end{bmatrix}$ and $\vec{x}_2 = \begin{bmatrix} 2 \\ 1 \end{bmatrix}$, we have that

- the projection of $\vec{u}$ onto $\vec{x}_1$ is $\left(\frac{\vec{u} \cdot \vec{x}_1}{\vec{x}_1 \cdot \vec{x}_1}\right) \vec{x}_1 = \frac{13}{5} \vec{x}_1$
- the projection of $\vec{u}$ onto $\vec{x}_2$ is $\left(\frac{\vec{u} \cdot \vec{x}_2}{\vec{x}_2 \cdot \vec{x}_2}\right) \vec{x}_2 = \frac{11}{5} \vec{x}_2$

but

$$\frac{13}{5}\vec{x}_1 + \frac{11}{5}\vec{x}_2 = \frac{13}{5}\begin{bmatrix}1\\2\end{bmatrix} + \frac{11}{5}\begin{bmatrix}2\\1\end{bmatrix} = \begin{bmatrix}13/5 + 22/5\\26/5 + 11/5\end{bmatrix} = \begin{bmatrix}7\\7.4\end{bmatrix} \neq \vec{u}$$

I'd like to provide a more general "theorem", on when you can use orthogonal projections to more easily write a vector $\vec{u}$ as a linear combination of vectors $\vec{v}_1$, $\vec{v}_2$, ..., $\vec{v}_d$, but we'll need to first study the idea of a **basis**. That's to come.

This section has been light on activities, since it provided many examples that we'll work through in lecture. But, here's one to tie this last point together.

Activity 1

Activity 1

Let $\vec{v}_1 = \begin{bmatrix}-1\\2\\2\end{bmatrix}$, $\vec{v}_2 = \begin{bmatrix}2\\2\\-1\end{bmatrix}$, and $\vec{v}_3 = \begin{bmatrix}2\\-1\\2\end{bmatrix}$.

Write $\vec{u} = \begin{bmatrix}1\\1\\1\end{bmatrix}$ as a linear combination of $\vec{v}_1$, $\vec{v}_2$, and $\vec{v}_3$, **without** solving a system of equations.

What's Next?

The motivating problem for this section was the approximation problem, which asked us to find the best approximation of a vector $\vec{u}$ using only a scalar multiple of a vector $\vec{v}$.

The next natural step is to consider the case where we want to approximate $\vec{u}$ using a **linear combination** of more than one vector, $\vec{v}_1$, $\vec{v}_2$, ..., $\vec{v}_d$. Why? Remember, this all connects back to the problem of linear regression. The more vectors we have as "building blocks" in our linear combination, the more features our model will be able to use. (I haven't made the connection from linear algebra to linear regression yet, but just know this is *why* we're studying projections.)

For example, let's consider $\vec{u} = \begin{bmatrix}1\\3\\6\end{bmatrix}$, $\vec{v}_1 = \begin{bmatrix}1\\0\\1\end{bmatrix}$, and $\vec{v}_2 = \begin{bmatrix}2\\2\\1\end{bmatrix}$. **Among all linear combinations of $\vec{v}_1$ and $\vec{v}_2$, which one is closest to $\vec{u}$?**

To answer this question, we'd need to find the scalars a_1 and a_2 such that the error vector

$$\vec{e} = \vec{u} - (a_1\vec{v}_1 + a_2\vec{v}_2)$$

has minimal length, which presumably happens when $\vec{e}$ is orthogonal to both $\vec{v}_1$ and $\vec{v}_2$.

Travelling down this road, we might be able to find the values of a_1 and a_2 that minimize the length of $\vec{e}$. But then we'll want to ask how we can do this for any $\vec{u}$ and any set of vectors $\vec{v}_1$, $\vec{v}_2$, ..., $\vec{v}_d$, and it seems like we'll need a more general solution. In general, to find the "best" approximation of $\vec{u}$ using a linear combination of $\vec{v}_1$, $\vec{v}_2$, ..., $\vec{v}_d$, we'll need to know about **matrices**. We'll introduce matrices in Chapter 5.1.

Instead, in Chapter 4.1, we will set aside the goal of projections temporarily, and instead focus on truly understanding the set of possible linear combinations of a given set of vectors. For example, the vectors $\vec{v}_1 = \begin{bmatrix}1\\0\\0\end{bmatrix}$,

$\vec{v}_2 = \begin{bmatrix} -5 \\ 1 \\ 8 \end{bmatrix}$ from earlier define a **plane**. So, asking which linear combination of $\vec{v}_1$ and $\vec{v}_2$ is closest to $\vec{u}$ is equivalent to asking which point on the plane is closest to $\vec{u}$.

Chapter 4.1 will answer the questions, "why do $\vec{v}_1$ and $\vec{v}_2$ define a plane?", "which plane do they define?", and "in general, what do $\vec{v}_1$, $\vec{v}_2$, ..., $\vec{v}_d$, all in $\mathbb{R}^n$, define?"

Linear Independence

4.1. Span

Overview

At the end of Chapter 3.4, we considered the problem of approximating $\vec{u} = \begin{bmatrix} 1 \\ 3 \\ 6 \end{bmatrix}$ using a linear combination of $\vec{v}_1 = \begin{bmatrix} 1 \\ 0 \\ 1 \end{bmatrix}$ and $\vec{v}_2 = \begin{bmatrix} 2 \\ 2 \\ 1 \end{bmatrix}$.

$\vec{v}_1$ and $\vec{v}_2$ define a plane in $\mathbb{R}^3$. But not all pairs of vectors in $\mathbb{R}^3$ define a plane – for example, the vectors $\begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix}$ and $\begin{bmatrix} 2 \\ 0 \\ 0 \end{bmatrix}$ define a line. How do we know if a set of vectors define a plane, line, or something else?

To motivate our discussion, let me recap "The Three Questions" involving linear combinations from Chapter 3.1.

The Three Questions with Linear Combinations

Given a set of vectors $\vec{v}_1, \vec{v}_2, \ldots, \vec{v}_d$ and a vector $\vec{b}$, all in $\mathbb{R}^n$:

1. **Can we** write $\vec{b}$ as a linear combination of $\vec{v}_1, \vec{v}_2, \ldots, \vec{v}_d$?
2. If so, are the values of the scalars $a_1, a_2, \ldots, a_d$ **unique**?
3. What is the **shape** of the set of all possible linear combinations of $\vec{v}_1, \vec{v}_2, \ldots, \vec{v}_d$?

So far, we've informally answered these questions in the context of various examples and problems. Here, I want to introduce a unifying framework for addressing these questions.

Definition: Span

The **span** of a set of vectors $\vec{v}_1, \vec{v}_2, ..., \vec{v}_d$, each of which is in $\mathbb{R}^n$, is the **set of all possible linear combinations** of those vectors.

$$\text{span}(\{\vec{v}_1, \vec{v}_2, ..., \vec{v}_d\}) = \left\{a_1\vec{v}_1 + a_2\vec{v}_2 + \ldots + a_d\vec{v}_d \mid a_1, a_2, \ldots, a_d \in \mathbb{R}\right\}$$

In this context, we might call the vectors $\vec{v}_1, \vec{v}_2, ..., \vec{v}_d$ the **spanning set** of the span.

Above, I've used set builder notation to describe the span of a set of vectors. You'll notice that I refer to $\vec{v}_1, \vec{v}_2, \ldots, \vec{v}_d$ as a set of vectors, but you'll notice that I don't always write them inside {set brackets}; this is just to save space. But when referring to a span, I'll always write the spanning set inside {set brackets}.

Let's look at several examples of spans, introducing important geometrical objects and ideas as we go.

Read Chapter 4.4!

Chapters 4.1-4.3 are slightly more theoretical in nature, and continue the main storyline of the course. Chapter 4.4 is a slight detour that covers how to describe lines, planes, and hyperplanes in $\mathbb{R}^n$, and is designed to complement this section. It isn't explicitly listed as a reading for any of the lectures, but it's

important and necessary to read. You'll refer to it in Lab 5.

Span of a Single Vector

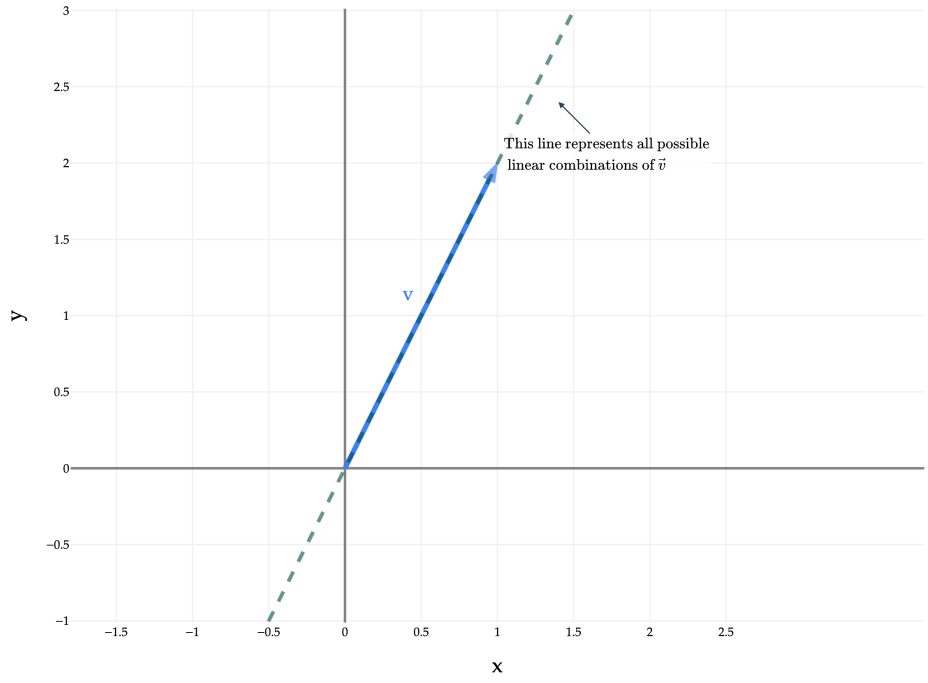
Suppose we have just a single vector $\vec{v} \in \mathbb{R}^n$.

$\text{span}(\{\vec{v}\})$ is the set of all linear combinations of just $\vec{v}$. Since there is only vector in the spanning set, there aren't any other vectors to add to $\vec{v}$, so the span is just the set of all scalar multiples of $\vec{v}$.

$$\text{span}(\{\vec{v}\}) = \{a\vec{v} \mid a \in \mathbb{R}\}$$

The span of a single vector is always a **line that passes through the origin and the vector's coordinates**. Another way of saying this is that a single vector **spans** a line. In Chapter 3.4, we learned how to project one vector onto the **span** of another vector.

Example in $\mathbb{R}^2$. For example, if we consider $\vec{v} = \begin{bmatrix} 1 \\ 2 \end{bmatrix}$ in $\mathbb{R}^2$, the span is the line through the origin and $(1, 2)$, which you might also recognize as the line $y = 2x$.



There are infinitely many vectors in $\text{span}(\{\begin{bmatrix} 1 \\ 2 \end{bmatrix}\})$, including $\begin{bmatrix} 2 \\ 4 \end{bmatrix}$ and $\begin{bmatrix} -1 \\ -2 \end{bmatrix}$, and the line shown above contains all of them. Notationally, we could say

$$\begin{bmatrix} 2 \\ 4 \end{bmatrix} \in \text{span}(\{\begin{bmatrix} 1 \\ 2 \end{bmatrix}\})$$

Example in $\mathbb{R}^3$. As another example, consider $\vec{v} = \begin{bmatrix} 2 \\ -1 \\ 3 \end{bmatrix}$ in $\mathbb{R}^3$. The span of $\vec{v}$ is the line through the origin and $(2, -1, 3)$.

The two marked points above are at the origin and $(2, -1, 3)$; I've marked them to emphasize that $\text{span}(\{\begin{bmatrix} 2 \\ -1 \\ 3 \end{bmatrix}\})$ is the unique line that passes through both $(2, -1, 3)$ and the origin. There are infinitely many lines that pass through $(2, -1, 3)$, but only one of those also passes through the origin.

Note that **lines are 1-dimensional objects**, regardless of the dimension of the space they live in. The line shown above is 1-dimensional in the sense that **any point on the line can be described using a single variable**. That variable is a from the definition of the span:

$$\text{span}(\{\begin{bmatrix} 2 \\ -1 \\ 3 \end{bmatrix}\}) = \left\{ a \begin{bmatrix} 2 \\ -1 \\ 3 \end{bmatrix} \mid a \in \mathbb{R} \right\}$$

Give me an a, and I'll give you a point on the line.

Generalization to $\mathbb{R}^n$. The fact that a single vector spans a line is true, regardless of the dimension of the vectors themselves. The span of the vector $\begin{bmatrix} 1 \\ 2 \\ \vdots \\ 100 \end{bmatrix} \in \mathbb{R}^{100}$ is the line through the origin and $(1, 2, \ldots, 100)$ in $\mathbb{R}^{100}$. We can't visualize what a line in 100-dimensional space looks like, but we know that it exists in this abstract sense.

Finally, I'll say that in the edge case where $\vec{v} = \begin{bmatrix} 0 \\ 0 \\ \vdots \\ 0 \end{bmatrix} = \vec{0}$, $\text{span}(\{\vec{0}\}) = \{\vec{0}\}$ is just the single point corresponding to the origin, not a line.

The fact that a single vector spans a line motivates a discussion in Chapter 4.4 about how to describe lines in $\mathbb{R}^n$. The short answer is that the form $y = mx + b$ only applies to lines in $\mathbb{R}^2$, and lines in $\mathbb{R}^3$ and higher must be described in **parametric form**.

$$L = \vec{p}_0 + tv, \qquad t \in \mathbb{R}$$

Again, Chapter 4.4 has more details.

Span of Two Vectors

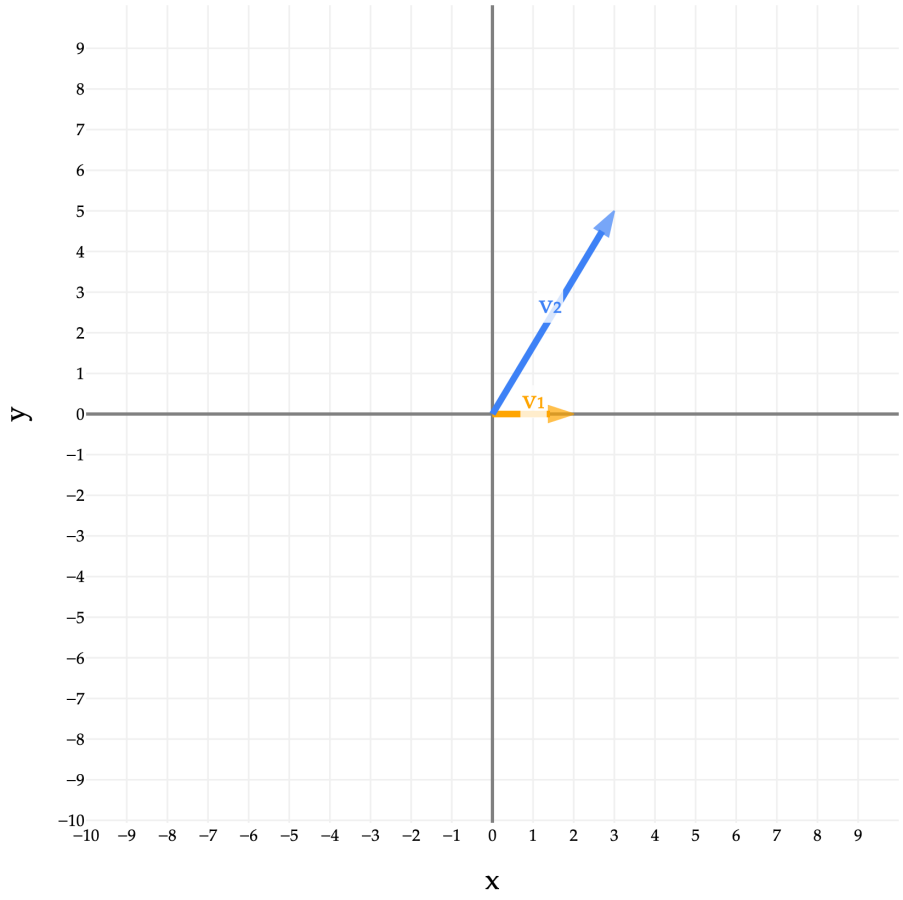
Instead of considering just a single vector, let's now consider the span of two vectors.

Examples in $\mathbb{R}^2$. Let's start with a simple example: the span of two vectors in $\mathbb{R}^2$. Consider

$$\vec{v}_1 = \begin{bmatrix} 2 \\ 0 \end{bmatrix}, \qquad \vec{v}_2 = \begin{bmatrix} 3 \\ 5 \end{bmatrix}$$

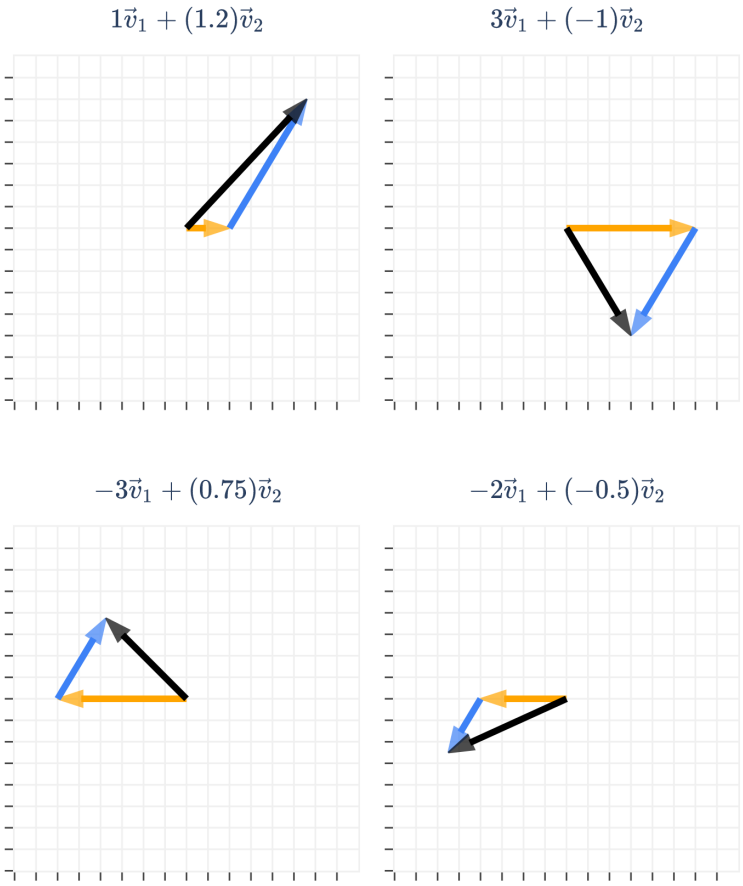
span($\{\vec{v}_1, \vec{v}_2\}$) is the set of all possible linear combinations of $\vec{v}_1$ and $\vec{v}_2$.

$$\text{span}(\{\vec{v}_1, \vec{v}_2\}) = \{a_1\vec{v}_1 + a_2\vec{v}_2 \mid a_1, a_2 \in \mathbb{R}\}$$



What is the set of all possible linear combinations of $\vec{v}_1$ and $\vec{v}_2$? Using $\vec{v}_1$ and $\vec{v}_2$ as building blocks, what are all possible vectors we can reach?

Since $\vec{v}_1$ and $\vec{v}_2$ point in different directions, we can reach any point in $\mathbb{R}^2$ by choosing appropriate values of a_1 and a_2.

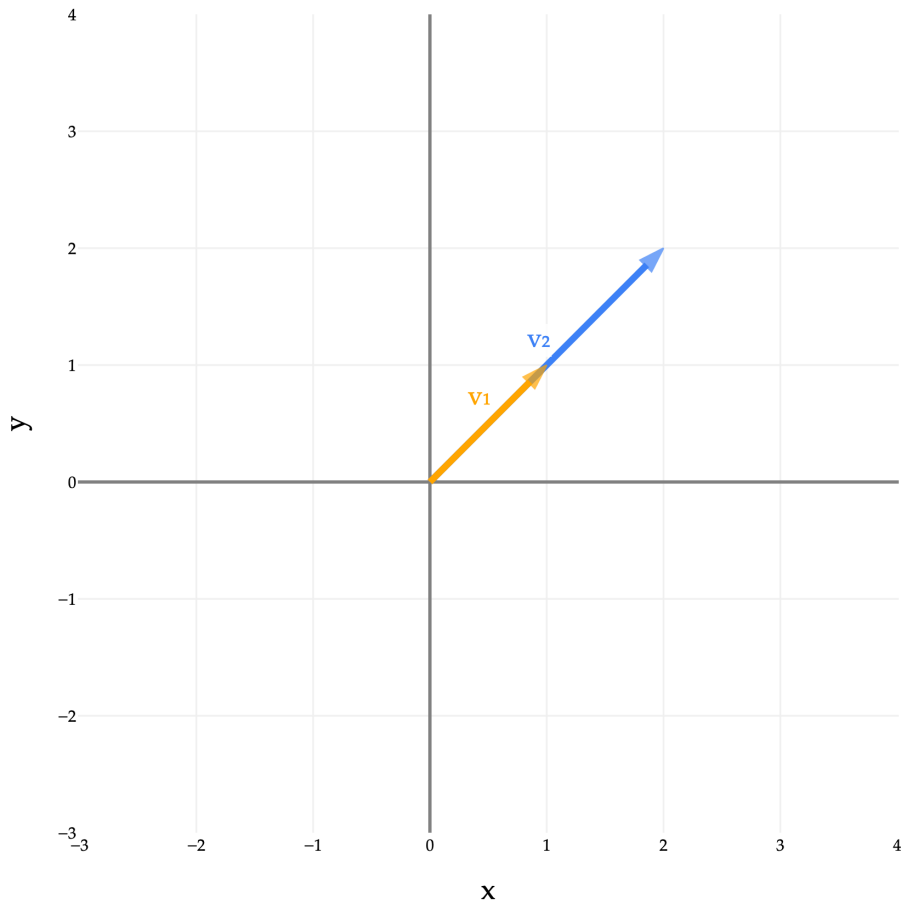


So, if $\vec{v}_1 = \begin{bmatrix} 2 \\ 0 \end{bmatrix}$ and $\vec{v}_2 = \begin{bmatrix} 3 \\ 5 \end{bmatrix}$, $\text{span}(\{\vec{v}_1, \vec{v}_2\}) = \mathbb{R}^2$, meaning that $\vec{v}_1$ and $\vec{v}_2$ span the entire xy-**plane**.

Equivalently, for **any** vector $\vec{b} \in \mathbb{R}^2$, there exist a_1 and a_2 such that $a_1\vec{v}_1 + a_2\vec{v}_2 = \vec{b}$.

But, not every pair of vectors in $\mathbb{R}^2$ spans the entire plane. For example, consider the vectors

$$\vec{v}_1 = \begin{bmatrix} 1 \\ 1 \end{bmatrix}, \qquad \vec{v}_2 = \begin{bmatrix} 2 \\ 2 \end{bmatrix}$$



$\vec{v}_1$ and $\vec{v}_2$ are scalar multiples of each other – i.e. they are **collinear** – so they only span a line, despite there being two vectors. They span the same line that either one of them spans individually, which is the line through the origin, $(1, 1)$, and $(2, 2)$.

If we start with just $\vec{v}_1$, the vector $\vec{v}_2$ doesn't "unlock" or "contribute" any new vectors to the span, since $\vec{v}_2$ is just a scalar multiple of $\vec{v}_1$. As we will soon see, this means that the vectors $\vec{v}_1$ and $\vec{v}_2$ are **linearly dependent**.

If we didn't immediately recognize that $\vec{v}_2$ is just a scalar multiple of $\vec{v}_1$ and tried to write (for example)

$$\vec{b} = \begin{bmatrix} 8 \\ 11 \end{bmatrix}$$

as a linear combination of $\vec{v}_1$ and $\vec{v}_2$, we might think it's possible, since we're working with a system of two equations and two unknowns.

$$a_1 \begin{bmatrix} 1 \\ 1 \end{bmatrix} + a_2 \begin{bmatrix} 2 \\ 2 \end{bmatrix} = \begin{bmatrix} 8 \\ 11 \end{bmatrix}$$

But, when you go to solve, you'll find

$$a_1 + 2a_2 = 8$$

$$a_1 + 2a_2 = 11$$

which is a contradiction, since $8 \neq 11$, meaning we can't make $\vec{b}$ as a linear combination of $\vec{v}_1$ and $\vec{v}_2$.

And, for the vectors $\vec{b}$ that are in span($\{\vec{v}_1, \vec{v}_2\}$), there are **infinitely many** ways to write them as linear combinations of $\vec{v}_1$ and $\vec{v}_2$. For example,

$$1\begin{bmatrix}1\\1\end{bmatrix} + 3\begin{bmatrix}2\\2\end{bmatrix} = \begin{bmatrix}7\\7\end{bmatrix}$$

$$7\begin{bmatrix}1\\1\end{bmatrix} + 0\begin{bmatrix}2\\2\end{bmatrix} = \begin{bmatrix}7\\7\end{bmatrix}$$

$$-35\begin{bmatrix}1\\1\end{bmatrix} + 21\begin{bmatrix}2\\2\end{bmatrix} = \begin{bmatrix}7\\7\end{bmatrix}$$

When solving for a_1 and a_2, I like to avoid situations where there are infinitely many solutions, since I'd like to make sure that if we all solve the same problem, we'll all get the same answer. Remember that all of this connects back to linear regression and machine learning; the analog of a_1 and a_2 are the parameters of a linear model, and we'd like to have interpretable parameter values so that we can see how the inputs of a model relate to the outputs.

Back to the main idea. In $\mathbb{R}^2$, the span of two vectors is a line if the vectors are collinear, and the entire xy-plane otherwise.

Example in $\mathbb{R}^3$. Similar results hold for the span of two vectors in $\mathbb{R}^3$. Hopefully I've convinced you that

$$\text{span}(\{\begin{bmatrix}5\\2\\1\end{bmatrix}, \begin{bmatrix}10\\4\\2\end{bmatrix}\}) = \underbrace{\text{span}(\{\begin{bmatrix}5\\2\\1\end{bmatrix}\})}_{\text{a line in } \mathbb{R}^3}$$

since $\begin{bmatrix}10\\4\\2\end{bmatrix}$ is just a scalar multiple of $\begin{bmatrix}5\\2\\1\end{bmatrix}$.

Let's see a more interesting example. Consider the vectors

$$\vec{v}_1 = \begin{bmatrix}5\\2\\1\end{bmatrix}, \quad \vec{v}_2 = \begin{bmatrix}-2\\3\\0\end{bmatrix}$$

You'll see them below, along with some of their linear combinations.

As was the case in $\mathbb{R}^2$, since $\vec{v}_1$ and $\vec{v}_2$ aren't collinear, they span a plane. In $\mathbb{R}^2$, there was only one possible plane that two vectors could span, because all of $\mathbb{R}^2$ itself is a plane, but in $\mathbb{R}^3$ there are infinitely many planes.

A plane is a flat surface that extends infinitely in all directions, with the property that if you connect any two points on the plane, the line connecting them lies entirely on the plane.

So,

$$\text{span}(\{\vec{v}_1, \vec{v}_2\}) = \underbrace{\{a_1\vec{v}_1 + a_2\vec{v}_2 \mid a_1, a_2 \in \mathbb{R}\}}_{\text{the plane above}}$$

I hope to have convinced you earlier that lines are 1-dimensional objects, since you only need a single "variable" to describe any point on a given line.

Similarly, **planes are 2-dimensional objects**, since you need two "variables" to describe any point on a given plane. In the standard xy-plane, all you need to describe a point is an x coordinate and a y coordinate. Effectively, every vector in $\mathbb{R}^2$ can be written as a linear combination of the "default" (formally **basis**) vectors $\begin{bmatrix}1\\0\end{bmatrix}$ and $\begin{bmatrix}0\\1\end{bmatrix}$.

$$\text{any point in } \mathbb{R}^2 = x\begin{bmatrix}1\\0\end{bmatrix} + y\begin{bmatrix}0\\1\end{bmatrix}$$

The plane shown above is also 2-dimensional, despite living in $\mathbb{R}^3$, since all I need are two variables to describe any point on it. The variables I need are a_1 and a_2, which are the multipliers on $\vec{v}_1$ and $\vec{v}_2$, respectively.

$$\text{any point in } \text{span}(\{\vec{v}_1, \vec{v}_2\}) = a_1\begin{bmatrix}5\\2\\1\end{bmatrix} + a_2\begin{bmatrix}-2\\3\\0\end{bmatrix}$$

Think of $\vec{v}_1$ and $\vec{v}_2$ as defining a new coordinate system for the plane that they span, where the coordinates themselves are the scalars a_1 and a_2. a_1 and a_2 can be arbitrarily large or small, which is what allows the plane to extend infinitely.

The picture above makes clear that there are many vectors in $\text{span}(\{\vec{v}_1, \vec{v}_2\})$, but also many vectors in $\mathbb{R}^3$ that are not in the span. $a_1\vec{v}_1 + a_2\vec{v}_2 = \vec{b}$ has a solution for a_1 and a_2 if and only if $\vec{b}$ lies in the plane above.

Again, Read Chapter 4.4!

To actually find the equation of the plane above in standard form,

$$ax + by + cz + d = 0$$

we need a tool called the **cross product**. We talk about this, and other details about lines and planes, in Chapter 4.4. (I know I already said this, but it's really important!)

Subspaces of $\mathbb{R}^n$. What if we're dealing with two vectors in some arbitrary $\mathbb{R}^n$? Consider

$$\vec{v}_1 = \begin{bmatrix}5\\1\\3\\2\\-8\end{bmatrix}, \qquad \vec{v}_2 = \begin{bmatrix}0\\1\\4\\0\\1\end{bmatrix}$$

These vectors live in $\mathbb{R}^5$, but what they span is a "plane" in $\mathbb{R}^5$. The more typical way of phrasing this is that these vectors span a **2-dimensional subspace** of $\mathbb{R}^5$.

We will formally define vector spaces and subspaces in Chapter 4.3, but for now, **think of a subspace as a flat**

object that passes through the origin and contains all linear combinations of some set of vectors.

- A line through the origin is a 1-dimensional subspace.
- A plane through the origin is a 2-dimensional subspace.
- In higher dimensions, we'll have 3-dimensional subspaces, 4-dimensional subspaces, and so on.

(A line that doesn't pass through the origin is still a line, but isn't a subspace, since subspaces must pass through the origin.)

The dimension of a subspace is the number of coordinates you need to describe any point in the subspace.

So although $\vec{v}_1 = \begin{bmatrix} 5 \\ 1 \\ 3 \\ 2 \\ -8 \end{bmatrix}$ and $\vec{v}_2 = \begin{bmatrix} 0 \\ 1 \\ 4 \\ 0 \\ 1 \end{bmatrix}$ sit inside $\mathbb{R}^5$, what they span is not all of $\mathbb{R}^5$, but rather a 2-dimensional "slice" of it, consisting of all linear combinations of $\vec{v}_1$ and $\vec{v}_2$. Any point in that subspace can be described using two coordinates, like a_1 and a_2 in

$$\text{any point in } \text{span}(\{\vec{v}_1, \vec{v}_2\}) = a_1 \begin{bmatrix} 5 \\ 1 \\ 3 \\ 2 \\ -8 \end{bmatrix} + a_2 \begin{bmatrix} 0 \\ 1 \\ 4 \\ 0 \\ 1 \end{bmatrix}$$

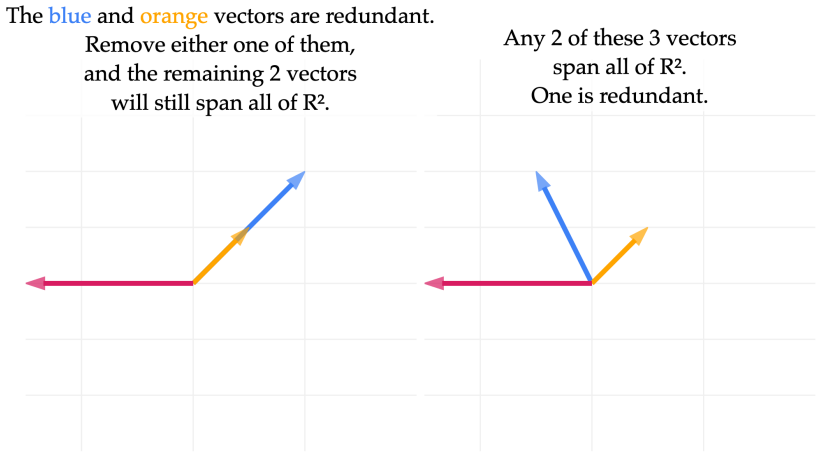
Span of Three Vectors

So far, as special cases, we've considered the span of one vector and the span of two vectors. The case of three vectors is the last special case I'll cover, and then we'll generalize to any number of vectors in any $\mathbb{R}^n$.

Examples in $\mathbb{R}^2$. Suppose we have three vectors in $\mathbb{R}^2$. What are their possible arrangements?

1. All three are the zero vector, $\vec{0}$.
2. All three are collinear, meaning they lie on the same line.
3. Two are collinear, and the third points in a different direction.
4. All three point in different directions.

In all of these cases, the "largest" their span can be is all of $\mathbb{R}^2$. Case 3 and Case 4 are shown below, on the left and right, respectively.



In the left example, $\vec{v}_3 = \begin{bmatrix} -3 \\ 0 \end{bmatrix}$ travels in a direction that $\vec{v}_1 = \begin{bmatrix} 1 \\ 1 \end{bmatrix}$ and $\vec{v}_2 = \begin{bmatrix} 2 \\ 2 \end{bmatrix}$ don't, so removing $\vec{v}_3$ would impact the span (it would drop from $\mathbb{R}^2$ to a line). But, we don't need both $\vec{v}_1$ and $\vec{v}_2$ to span $\mathbb{R}^2$, since $\vec{v}_1$ already travels in the direction of $\vec{v}_2$.

In the right example, where

$$\vec{v}_1 = \begin{bmatrix} 1 \\ 1 \end{bmatrix}, \quad \vec{v}_2 = \begin{bmatrix} -1 \\ 2 \end{bmatrix}, \quad \vec{v}_3 = \begin{bmatrix} -3 \\ 0 \end{bmatrix}$$

Any two of these vectors will span the entirety of $\mathbb{R}^2$, meaning that if you pick any two of them, you can recreate the third one. These three vectors are linearly dependent.

$$\text{span}(\{\vec{v}_1, \vec{v}_2, \vec{v}_3\}) = \text{span}(\{\vec{v}_1, \vec{v}_2\}) = \text{span}(\{\vec{v}_1, \vec{v}_3\}) = \text{span}(\{\vec{v}_2, \vec{v}_3\}) = \mathbb{R}^2$$

Examples in $\mathbb{R}^3$. Moving on up to $\mathbb{R}^3$, instead of considering all possible arrangements, let me enumerate their possible spans.

1. All three are the zero vector, $\vec{0}$. (Annoying edge case, but I'm including it for completeness.)
2. All three are collinear, meaning they span a line.
3. All three are on the same plane, meaning they span that same plane.
4. None of the above, meaning they span all of $\mathbb{R}^3$.

As before, Case 1 and Case 2 are easy enough to reason about. First, let's consider Case 3. Suppose

$$\vec{v}_1 = \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix}, \quad \vec{v}_2 = \begin{bmatrix} 1 \\ -2 \\ -3 \end{bmatrix}, \quad \vec{v}_3 = \begin{bmatrix} -3 \\ 0 \\ 1 \end{bmatrix}$$

Notice that all three vectors lie on the same plane, even though they all "point in different directions". As we saw earlier, you only need **two** vectors to span a plane. If you remove any one of these three vectors, the remaining two will still span the exact same plane.

This is a consequence of the fact that you can write $\vec{v}_3$ as a linear combination of $\vec{v}_1$ and $\vec{v}_2$.

$$\vec{v}_1 = \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix}, \quad \vec{v}_2 = \begin{bmatrix} 1 \\ -2 \\ -3 \end{bmatrix}, \quad \vec{v}_3 = \begin{bmatrix} -3 \\ 0 \\ 1 \end{bmatrix}$$

$$\vec{v}_3 = -2\vec{v}_1 - \vec{v}_2$$

Starting from $\vec{v}_1$ and $\vec{v}_2$ alone, adding $\vec{v}_3$ doesn't unlock any new directions, since it's already a linear combination of $\vec{v}_1$ and $\vec{v}_2$.

There is nothing "especially wrong" about $\vec{v}_3$. We can rearrange the above to get

$$\vec{v}_2 = -2\vec{v}_1 - \vec{v}_3$$

and

$$\vec{v}_1 = -\frac{1}{2}\vec{v}_2 - \frac{1}{2}\vec{v}_3$$

too; any one of these three vectors can be written as a linear combination of the other two, meaning if we remove any one of them, the span of the remaining two will still be the plane you see above.

As we saw earlier in the case of two collinear vectors in $\mathbb{R}^2$, having a vector that can be written as a linear combination of the other vectors in the set means that when we go to write other vectors as linear combinations of the vectors in the set, **if it's possible**, there are infinitely many solutions.

If we let $\vec{b} = \begin{bmatrix} b_1 \\ b_2 \\ b_3 \end{bmatrix}$ be any arbitrary vector in $\mathbb{R}^3$, then the system

$$a_1\vec{v}_1 + a_2\vec{v}_2 + a_3\vec{v}_3 = \vec{b}$$

either has **no solutions**, if $\vec{b}$ is not on the plane defined by $\text{span}(\{\vec{v}_1, \vec{v}_2, \vec{v}_3\})$, or it has **infinitely many**.

Activity 1

Activity 1

Let's work with the same 3 vectors from above,

$$\vec{v}_1 = \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix}, \quad \vec{v}_2 = \begin{bmatrix} 1 \\ -2 \\ -3 \end{bmatrix}, \quad \vec{v}_3 = \begin{bmatrix} -3 \\ 0 \\ 1 \end{bmatrix}$$

Suppose that $\vec{b} \in \text{span}(\{\vec{v}_1, \vec{v}_2, \vec{v}_3\})$. Here's one way of expressing $\vec{b}$ as a linear combination of $\vec{v}_1$, $\vec{v}_2$, and $\vec{v}_3$.

$$\vec{b} = 2\vec{v}_1 + 3\vec{v}_2 - 4\vec{v}_3$$

Find another way of expressing $\vec{b}$ as a linear combination of $\vec{v}_1$, $\vec{v}_2$, and $\vec{v}_3$, where the coefficient on $\vec{v}_2$ is 5.

Solution

We'd like the coefficient on $\vec{v}_2$ to be 5. Right now, it's 3. So, to boost it to 5, we can add and subtract $2\vec{v}_2$ to the right-hand side above.

$$\begin{aligned}\vec{b} &= 2\vec{v}_1 + 3\vec{v}_2 - 4\vec{v}_3 + \underbrace{2\vec{v}_2 - 2\vec{v}_2}_{\text{adding 0}} \\ &= 2\vec{v}_1 + 5\vec{v}_2 - 4\vec{v}_3 - 2\vec{v}_2\end{aligned}$$

To get rid of the $-2\vec{v}_2$ term, we can replace $\vec{v}_2$ with $-2\vec{v}_1 - \vec{v}_3$, which is a linear combination of $\vec{v}_1$ and $\vec{v}_3$. Substituting this above gives us

$$\begin{aligned}\vec{b} &= 2\vec{v}_1 + 5\vec{v}_2 - 4\vec{v}_3 - 2(-2\vec{v}_1 - \vec{v}_3) \\ &= 6\vec{v}_1 + 5\vec{v}_2 - 2\vec{v}_3\end{aligned}$$

Let's now consider Case 4, from the 4 cases above.

1. All three are the zero vector, $\vec{0}$. (Annoying edge case, but I'm including it for completeness.)
2. All three are collinear, meaning they span a line.
3. All three are on the same plane, meaning they span that same plane.
4. **None of the above, meaning they span all of $\mathbb{R}^3$.**

Consider

$$\vec{v}_1 = \begin{bmatrix}1\\1\\1\end{bmatrix}, \quad \vec{v}_2 = \begin{bmatrix}1\\4\\-3\end{bmatrix}, \quad \vec{v}_3 = \begin{bmatrix}-3\\0\\1\end{bmatrix}$$

Drag the plot around to see the space from different angles. If we look at any pair of these vectors, we see they span a plane. But, since neither is a linear combination of the other two, all three of them bring something new to the span; none are redundant.

So, the span of these three vectors is **all of** $\mathbb{R}^3$! You can think of these three vectors as defining a new coordinate system for $\mathbb{R}^3$.

The default coordinate system in $\mathbb{R}^3$ uses three numbers to describe any point in $\mathbb{R}^3$. Those three numbers are used to take a linear combination of the default basis vectors,

$$\text{any point in } \mathbb{R}^3 = x\begin{bmatrix}1\\0\\0\end{bmatrix} + y\begin{bmatrix}0\\1\\0\end{bmatrix} + z\begin{bmatrix}0\\0\\1\end{bmatrix}$$

The coordinate system defined by $\vec{v}_1$, $\vec{v}_2$, and $\vec{v}_3$ also uses three numbers to describe any point in $\mathbb{R}^3$, it's just that the coordinates used multiply vectors other than the default ones.

$$\text{any point in } \mathbb{R}^3 = a_1\begin{bmatrix}1\\1\\1\end{bmatrix} + a_2\begin{bmatrix}1\\4\\-3\end{bmatrix} + a_3\begin{bmatrix}-3\\0\\1\end{bmatrix}$$

So, any vector $\vec{b} \in \mathbb{R}^3$ can be written as a linear combination of $\vec{v}_1$, $\vec{v}_2$, and $\vec{v}_3$.

Generalization to $\mathbb{R}^n$. Consider the following vectors in $\mathbb{R}^5$.

$$\vec{v}_1 = \begin{bmatrix}5\\1\\3\\2\\-8\end{bmatrix}, \qquad \vec{v}_2 = \begin{bmatrix}0\\1\\4\\0\\1\end{bmatrix}, \qquad \vec{v}_3 = \begin{bmatrix}1\\0\\0\\0\\0\end{bmatrix}$$

None of these vectors are a linear combination of the other two. So what do they span? **A 3-dimensional subspace of $\mathbb{R}^5$.**

That subspace is a 3-dimensional slice of the 5-dimensional space that is $\mathbb{R}^5$, and any vector in that subspace can be written using three coordinates.

$$\text{any point in } \text{span}(\{\vec{v}_1, \vec{v}_2, \vec{v}_3\}) = a_1\vec{v}_1 + a_2\vec{v}_2 + a_3\vec{v}_3$$

If, say, $\vec{v}_3$ was a linear combination of $\vec{v}_1$ and $\vec{v}_2$, then the span would be a 2-dimensional subspace of $\mathbb{R}^5$.

Thinking in Higher Dimensions

Let's generalize what we've discussed so far. What I'm about to say is abstract, but try and keep track of how it relates to the examples we've seen so far. If ever in doubt, plug in numbers for d and n to help make sense of it.

Remember that I've told you to think of a subspace as a flat object that passes through the origin and contains all linear combinations of some set of vectors. The dimension of a subspace is the number of coordinates you need to describe any point in the subspace; for instance, a line through the origin is a 1-dimensional subspace, and a plane through the origin is a 2-dimensional subspace.

Suppose we have d vectors, all in $\mathbb{R}^n$, labeled $\vec{v}_1$, $\vec{v}_2$, ..., $\vec{v}_d$. Then,

- **If we have fewer than n vectors**, i.e. $d < n$, then the vectors span a 0, or 1, or 2, or 3, ..., or d-dimensional subspace of $\mathbb{R}^n$.
- **If we have at least n vectors**, i.e. $d \geq n$, then the vectors span a 0, or 1, or 2, or 3, ..., or n-dimensional subspace of $\mathbb{R}^n$. (An n-dimensional subspace of $\mathbb{R}^n$ is all of $\mathbb{R}^n$.)

Or, put succinctly:

The dimension of a subspace

The dimension of the subspace spanned by the vectors is between 0 and $\min(d, n)$.

Activity 2

Activity 2

Find an example of 6 vectors in $\mathbb{R}^4$ that span a 3-dimensional subspace of $\mathbb{R}^4$.

Solution

One idea is to pick three vectors that we know aren't linear combinations of each other, and then add three vectors that are linear combinations of the first three.
For instance,

$$\begin{bmatrix}1\\0\\0\\0\end{bmatrix}, \begin{bmatrix}3\\0\\0\\0\end{bmatrix}, \begin{bmatrix}0\\1\\0\\0\end{bmatrix}, \begin{bmatrix}0\\5\\0\\0\end{bmatrix}, \begin{bmatrix}0\\0\\100\\0\end{bmatrix}, \begin{bmatrix}0\\0\\-15\\0\end{bmatrix}$$

The first three vectors are linearly independent, and the last three are linear combinations of the first three.

In general, given a set of d vectors in $\mathbb{R}^n$, actually finding the dimension of the subspace they span involves solving a system of equations and unknowns. Chapter 4.2 introduces an algorithm for finding this dimension by hand. But for now, know that `numpy` can help us.

For example, suppose we have $d = 5$ vectors in $\mathbb{R}^7$, given by

$$\vec{v}_1 = \begin{bmatrix}5\\3\\2\\0\\14\\3\\1\end{bmatrix}, \quad \vec{v}_2 = \begin{bmatrix}2\\0\\0\\3\\19\\0\\0\end{bmatrix}, \quad \vec{v}_3 = \begin{bmatrix}3\\3\\2\\-3\\-5\\3\\1\end{bmatrix}, \quad \vec{v}_4 = \begin{bmatrix}3\\2\\-3\\1\\0\\0\\5\end{bmatrix}, \quad \vec{v}_5 = \begin{bmatrix}0\\-3\\11\\3\\52\\3\\-14\end{bmatrix}$$

If we store all 5 vectors as arrays, and use the magical `np.linalg.matrix_rank` function, we get back the dimension of the subspace they span.

```
import numpy as np

v1 = np.array([5, 3, 2, 0, 14, 3, 1])
v2 = np.array([2, 0, 0, 3, 19, 0, 0])
v3 = np.array([ 3, 3, 2, -3, -5, 3, 1])
v4 = np.array([3, 2, -3, 1, 0, 0, 5])
v5 = np.array([ 0, -3,  11,   3,  52,   3, -14])

np.linalg.matrix_rank(np.array([v1, v2, v3, v4, v5]).T)
```

```
3
```

We'll explore matrices soon. The **rank** of a matrix is the dimension of the subspace spanned by its columns. Since `np.linalg.matrix_rank` returned 3, the vectors $\vec{v}_1, \vec{v}_2, \vec{v}_3, \vec{v}_4, \vec{v}_5$ span a 3-dimensional subspace of $\mathbb{R}^7$.

Don't stop reading here: Chapter 4.2 directly follows from this section. It formalizes an idea we've implicitly used throughout this section, that of **linear independence**.

One Final Reminder: Read Chapter 4.4!

Chapter 4.2 directly follows from this section, but don't forget to also read Chapter 4.4, which you are now ready to read too.

4.2. Linear Independence

Introduction

There's an idea we used all throughout Chapter 4.1 that we haven't given a name to.

Definitions.

Definition: Linear Independence

A set of vectors $\vec{v}_1$, $\vec{v}_2$, ..., $\vec{v}_d$ is **linearly independent** if either of the following equivalent conditions hold:

1. None of the vectors can be written as a linear combination of the others.
2. The only way to create the zero vector as a linear combination of the vectors is if all the coefficients are zero. In other words, the only solution to

$$a_1\vec{v}_1 + a_2\vec{v}_2 + \ldots + a_d\vec{v}_d = \vec{0}$$

is $a_1 = a_2 = \ldots = a_d = 0$.

If a set of vectors is not linearly independent, it is **linearly dependent**.

Intuitively, if a set of vectors is linearly **dependent**, then **at least one** of the vectors is a linear combination of the others. If we think of vectors as building blocks, if a set of vectors is linearly dependent, then at least one of the building blocks is redundant, because you can create it from the other building blocks.

Equivalently, if a set of vectors is linearly **dependent**, there's a **non-trivial** linear combination of the vectors that equals the zero vector (by non-trivial, I mean that at least one of the coefficients is non-zero).

Why are these two conditions equivalent? Here's one way to see it. Suppose $\vec{v}_1 = \alpha\vec{v}_2 + \beta\vec{v}_3$, meaning that $\vec{v}_1$ can be written as a linear combination of $\vec{v}_2$ and $\vec{v}_3$. Rearranging the equation above gives us

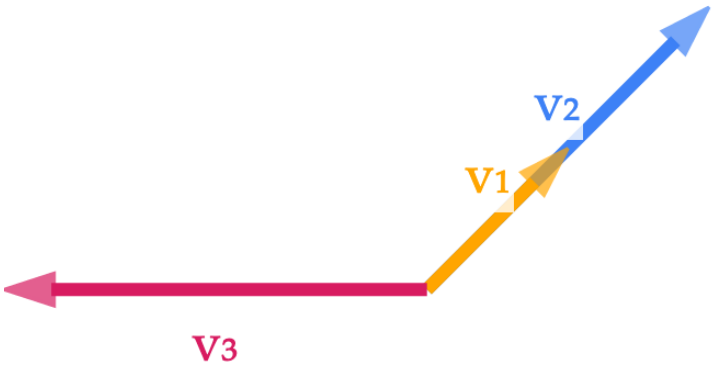
$$\vec{v}_1 - \alpha\vec{v}_2 - \beta\vec{v}_3 = \vec{0}$$

which shows us a non-trivial linear combination of $\vec{v}_1, \vec{v}_2, \vec{v}_3$ that gives $\vec{0}$. The converse (reverse direction) is true too: if you start with a non-trivial linear combination of $\vec{v}_1, \vec{v}_2, \ldots, \vec{v}_d$ that gives $\vec{0}$, then you can rearrange it to get $\vec{v}_1$ = some linear combination of $\vec{v}_2, \ldots, \vec{v}_d$.

Examples. Let's look at several sets of vectors and comment on their linear independence (or lack thereof).

Vectors	Linearly...	Why?
$\begin{bmatrix}1\\0\\3\end{bmatrix}, \begin{bmatrix}2\\1\\5\end{bmatrix}$	**Independent**	Neither is a multiple of the other.
$\begin{bmatrix}1\\0\\3\end{bmatrix}, \begin{bmatrix}2\\1\\5\end{bmatrix}, \begin{bmatrix}5\\2\\10\end{bmatrix}, \begin{bmatrix}1\\1\\1\end{bmatrix}$	**Dependent**	These vectors live in $\mathbb{R}^3$, which is a universe that only has 3 independent directions, so you only need 3 vectors to span it. Give 4 vectors, we can write at least one of them as a linear combination of the others.
$\begin{bmatrix}1\\0\\3\end{bmatrix}, \begin{bmatrix}2\\1\\5\end{bmatrix}, \begin{bmatrix}5\\2\\13\end{bmatrix}$	**Dependent**	first vector + 2(second vector) = third vector
$\begin{bmatrix}1\\0\\3\end{bmatrix}, \begin{bmatrix}2\\1\\5\end{bmatrix}, \begin{bmatrix}5\\2\\10\end{bmatrix}$	**Independent**	The first two were already linearly independent from the first example, and we can't write the third as a linear combination of the first two.

Note that if a set of vectors is linearly dependent, it doesn't mean that **every** vector in the set can be written as a linear combination of the others. It just means that there's at least one vector that can be written as a linear combination of the others. A good go-to example for this is the one below – $\vec{v}_1$ and $\vec{v}_2$ are scalar multiples of each other, making the entire set of three vectors linearly dependent, but $\vec{v}_3$ is not a linear combination of $\vec{v}_1$ and $\vec{v}_2$.



Unique Linear Combinations

Fact: If a set of vectors $\vec{v}_1, \vec{v}_2, \ldots, \vec{v}_d \in \mathbb{R}^n$ is linearly independent, then any vector $\vec{b} \in \mathbb{R}^n$ in the span of the vectors can be written as a **unique** linear combination of the vectors.

We've built intuition for this above; now let's give a formal proof. But first, note that the statement assumes that $\vec{b} \in \text{span}(\{\vec{v}_1, \vec{v}_2, \ldots, \vec{v}_d\})$ to begin with: it is not saying that if $\vec{v}_1, \vec{v}_2, \ldots, \vec{v}_d \in \mathbb{R}^n$ are linearly independent, then any $\vec{b} \in \mathbb{R}^n$ can be written as a linear combination of the vectors. That's **not** true. The vectors $\vec{v}_1 = \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix}$ and $\vec{v}_2 = \begin{bmatrix} 0 \\ 1 \\ 0 \end{bmatrix}$ are linearly independent, but $\vec{b} = \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix}$ is not a linear combination of the first two. Hopefully, the fact above is a little more clear now. (Re-read it again before proceeding to the next paragraph.)

Let's imagine an alternate universe where $\vec{b} \in \text{span}(\{\vec{v}_1, \vec{v}_2, \ldots, \vec{v}_d\})$ can be written as two different linear combinations of the vectors. (We're doing a proof by contradiction, if you're familiar with the idea.) In other words, suppose

$$a_1\vec{v}_1 + a_2\vec{v}_2 + \ldots + a_d\vec{v}_d = \vec{b}$$

and

$$c_1\vec{v}_1 + c_2\vec{v}_2 + \ldots + c_d\vec{v}_d = \vec{b}$$

and **not** all of the a_i and c_i are equal, meaning there's at least one i such that $a_i \neq c_i$.

What happens if we subtract the two equations?

$$(a_1 - c_1)\vec{v}_1 + (a_2 - c_2)\vec{v}_2 + \ldots + (a_d - c_d)\vec{v}_d = \vec{0}$$

Since $\vec{v}_1, \vec{v}_2, \ldots, \vec{v}_d$ are linearly independent, the only way this equation can hold is if all of the coefficients on the $\vec{v}_i$ are zero. In other words, we'd need

$$a_1 - c_1 = 0 a_2 - c_2 = 0 \ldots a_d - c_d = 0$$

But if that's the case, then $a_i = c_i$ for all i, which contradicts our assumption that not all of the a_i and c_i are equal.

So, this means that it **can't** be the case that $\vec{b}$ can be written as two different linear combinations of the vectors. In other words, if $\vec{b} \in \text{span}(\{\vec{v}_1, \vec{v}_2, \ldots, \vec{v}_d\})$ and $\vec{v}_1, \vec{v}_2, \ldots, \vec{v}_d$ are linearly independent, then $\vec{b}$ can be written as a **unique** linear combination of the vectors.

Linearly independent vectors are desirable, since there's only one way to use them as building blocks to create any other vector in their span.

Activity 1

Activity 1

Suppose $\vec{v}_1, \vec{v}_2, \ldots, \vec{v}_d \in \mathbb{R}^n$ are an orthogonal set of vectors, meaning that for any $i \neq j$, $\vec{v}_i \cdot \vec{v}_j = 0$. (Assume that the vectors are non-zero.)

In Lab 5, you'll show that any vector $\vec{b} \in \text{span}(\{\vec{v}_1, \vec{v}_2, \ldots, \vec{v}_d\})$ can be written as a linear combination of the vectors by projecting $\vec{b}$ onto each of the vectors $\vec{v}_i$.

Here, **prove that the vectors $\vec{v}_1, \vec{v}_2, \ldots, \vec{v}_d$ are linearly independent**. By doing this, you are showing that orthogonality is a stronger condition than linear independence.

Solution

In order for $\vec{v}_1, \vec{v}_2, \ldots, \vec{v}_d$ to be linearly independent, we need to show that the only solution to

$$a_1\vec{v}_1 + a_2\vec{v}_2 + \ldots + a_d\vec{v}_d = \vec{0}$$

is $a_1 = a_2 = \ldots = a_d = 0$. Otherwise, there must exist some other non-zero solution for the a_i's.
Let's start with the equation above and take the dot product of both sides with $\vec{v}_1$.

$$(a_1\vec{v}_1 + a_2\vec{v}_2 + \ldots + a_d\vec{v}_d) \cdot \vec{v}_1 = \vec{0} \cdot \vec{v}_1$$

This expands to

$$a_1(\vec{v}_1 \cdot \vec{v}_1) + a_2(\vec{v}_2 \cdot \vec{v}_1) + \ldots + a_d(\vec{v}_d \cdot \vec{v}_1) = 0$$

Since $\vec{v}_1, \vec{v}_2, \ldots, \vec{v}_d$ are orthogonal, we know that $\vec{v}_i \cdot \vec{v}_j = 0$ for all $i \neq j$. This means that the only non-zero term in the equation above is $a_1(\vec{v}_1 \cdot \vec{v}_1)$. So, we're left with

$$a_1(\vec{v}_1 \cdot \vec{v}_1) = 0$$

Since $\vec{v}_1$ is non-zero, we can divide both sides by $\vec{v}_1 \cdot \vec{v}_1$ to get $a_1 = 0$.
We can repeat this process for each of the vectors $\vec{v}_2, \ldots, \vec{v}_d$ to show that $a_2 = \ldots = a_d = 0$, meaning that the only solution to $a_1\vec{v}_1 + a_2\vec{v}_2 + \ldots + a_d\vec{v}_d = \vec{0}$ is $a_1 = a_2 = \ldots = a_d = 0$, meaning that $\vec{v}_1, \vec{v}_2, \ldots, \vec{v}_d$ are linearly independent.

Algorithm for Finding Linearly Independent Subsets with the Same Span

Given a set of vectors $\vec{v}_1, \vec{v}_2, \ldots, \vec{v}_d \in \mathbb{R}^n$, we'd like to find a subset of the vectors that is **linearly independent and has the same span** as the original set of vectors. In Chapter 4.3, we'll call this subset a **basis** for the span of the original set of vectors.

In other words, we'd like to "drop" the vectors that are linearly dependent on the others. For example, if we have 3 vectors in $\mathbb{R}^3$ and they span a plane, we can drop one of them and still span **the same** plane (not just any plane), because a plane is a 2-dimensional space, and you only need 2 vectors to span a plane.

In the example below, we can remove any one of the three vectors (which all point in different directions), and the span of the remaining two is still **the exact same plane**!

Dropping any "unnecessary" vectors will give us the desirable property that any vector in the span of the original set of vectors can be written as a **unique** linear combination of the vectors in the subset. (Remember that this has a connection to finding optimal model parameters in linear regression – this is not just an arbitrary exercise in theory.)

One way to produce a linear independent subset is to execute the following algorithm:

```
given v_1, v_2, ..., v_d
initialize linearly independent set S = {v_1}
for i = 2 to d:
    if v_i is not a linear combination of S:
```

```
        add v_i to S
```

The vectors we add to `S` are a basis for the span of the original set of vectors. The number of vectors we're left with is the dimension of the span of the original set of vectors. Again, these are ideas we formalize in Chapter 4.3.

Let's evaluate this algorithm on the following set of vectors:

$$\vec{v}_1 = \begin{bmatrix} 3 \\ 4 \\ 0 \end{bmatrix}, \quad \vec{v}_2 = \begin{bmatrix} 0 \\ 1 \\ 1 \end{bmatrix}, \quad \vec{v}_3 = \begin{bmatrix} -3 \\ -2 \\ 2 \end{bmatrix}, \quad \vec{v}_4 = \begin{bmatrix} 6 \\ 5 \\ -3 \end{bmatrix}, \quad \vec{v}_5 = \begin{bmatrix} 2 \\ 5 \\ 1 \end{bmatrix}$$

First, we start with $S = \{\vec{v}_1\}$.

- **Iteration 1 ($i = 2$)**: Is $\vec{v}_2$ a linear combination of the vectors in S?
  No, since $\vec{v}_2$ is not a multiple of $\vec{v}_1$. The first components (3 in $\vec{v}_1$, 0 in $\vec{v}_2$) imply that if $\vec{v}_2$ were a multiple of $\vec{v}_1$ it'd need to be $0\vec{v}_1$, but the other components of $\vec{v}_2$ are non-zero.

  Outcome: Add $\vec{v}_2$ to S. Now, $S = \{\vec{v}_1, \vec{v}_2\}$.

- **Iteration 2 ($i = 3$)**: Is $\vec{v}_3$ a linear combination of the vectors in S?
  To determine the answer, we need to try and find scalars a_1 and a_2 such that $a_1\vec{v}_1 + a_2\vec{v}_2 = \vec{v}_3$.

$$a_1 \begin{bmatrix} 3 \\ 4 \\ 0 \end{bmatrix} + a_2 \begin{bmatrix} 0 \\ 1 \\ 1 \end{bmatrix} = \begin{bmatrix} -3 \\ -2 \\ 2 \end{bmatrix} \implies \begin{cases} 3a_1 + 0a_2 = -3 \\ 4a_1 + 1a_2 = -2 \\ 0a_1 + 1a_2 = 2 \end{cases}$$

  The first equation implies $a_1 = -1$ and the third equation implies $a_2 = 2$. Plugging both into the second equation gives $4(-1) + 1(2) = -2$, which is consistent. This means that $\vec{v}_3 = -\vec{v}_1 + 2\vec{v}_2$, so $\vec{v}_3$ is a linear combination of $\vec{v}_1$ and $\vec{v}_2$, and we should not add it to S.

  Outcome: Leave S unchanged. Now, $S = \{\vec{v}_1, \vec{v}_2\}$.

- **Iteration 4 ($i = 4$)**: Is $\vec{v}_4$ a linear combination of the vectors in S?
  To determine the answer, we need to try and find scalars a_1 and a_2 such that $a_1\vec{v}_1 + a_2\vec{v}_2 = \vec{v}_4$.

$$a_1 \begin{bmatrix} 3 \\ 4 \\ 0 \end{bmatrix} + a_2 \begin{bmatrix} 0 \\ 1 \\ 1 \end{bmatrix} = \begin{bmatrix} 6 \\ 5 \\ -3 \end{bmatrix} \implies \begin{cases} 3a_1 + 0a_2 = 6 \\ 4a_1 + 1a_2 = 5 \\ 0a_1 + 1a_2 = -3 \end{cases}$$

  Similarly, we see that $a_1 = 2$ (from the first equation) and $a_2 = -3$ (from the third equation) are consistent with the second equation. This means that $\vec{v}_4 = 2\vec{v}_1 - 3\vec{v}_2$, so $\vec{v}_4$ is a linear combination of $\vec{v}_1$ and $\vec{v}_2$, and we should not add it to S.

  Outcome: Leave S unchanged. Now, $S = \{\vec{v}_1, \vec{v}_2\}$.

- **Iteration 5 ($i = 5$)**: Is $\vec{v}_5$ a linear combination of the vectors in S?
  To determine the answer, we need to try and find scalars a_1 and a_2 such that $a_1\vec{v}_1 + a_2\vec{v}_2 = \vec{v}_5$.

$$a_1\begin{bmatrix}3\\4\\0\end{bmatrix} + a_2\begin{bmatrix}0\\1\\1\end{bmatrix} = \begin{bmatrix}2\\5\\1\end{bmatrix} \implies \begin{cases}3a_1 + 0a_2 = 2\\4a_1 + 1a_2 = 5\\0a_1 + 1a_2 = 1\end{cases}$$

  The first equation implies $a_1 = \frac{2}{3}$ and the third equation implies $a_2 = 1$. Plugging both into the second equation gives $4(\frac{2}{3}) + 1(1) = \frac{11}{3} \neq 5$, which means the system is inconsistent. So, $\vec{v}_5$ is not a linear combination of $\vec{v}_1$ and $\vec{v}_2$, and we should add it to S.

  Outcome: Add $\vec{v}_5$ to S. Now, $S = \{\vec{v}_1, \vec{v}_2, \vec{v}_5\}$.

$\vec{v}_1, \vec{v}_2, \vec{v}_5$ are linearly independent vectors that have the same span as the original set of vectors. And since these are 3 linearly independent vectors in $\mathbb{R}^3$, their span is all of $\mathbb{R}^3$, since $\mathbb{R}^3$ is 3-dimensional and only has 3 independent directions to begin with! If there were more vectors in our list, they would surely be linearly dependent on $S = \{\vec{v}_1, \vec{v}_2, \vec{v}_5\}$, and so we wouldn't need to consider them.

Note that the subset that this algorithm produces is **not unique**, meaning that there exist other subsets of 3 of $\{\vec{v}_1, \vec{v}_2, \vec{v}_3, \vec{v}_4, \vec{v}_5\}$ that are also linearly independent and have the same span as all of $\{\vec{v}_1, \vec{v}_2, \vec{v}_3, \vec{v}_4, \vec{v}_5\}$ do (which is also the span of $\{\vec{v}_1, \vec{v}_2, \vec{v}_5\}$) If you started with $\vec{v}_5$, then considered $\vec{v}_4$, then considered $\vec{v}_3$, and so on, you'd end up with a subset that includes $\vec{v}_4$, for instance.

What is fixed, though, is **how many** linearly independent vectors you need to span the entire subspace that these five vectors span, and the answer to that is 3.

Homework 4 and Lab 5 will have you practice this algorithm several times, though – as mentioned above – we'll use the power of Python to handle some of this for us, soon.

Activity 2

Activity 2

To recap what we've covered in this section, answer the following questions.

1. Can any three vectors in $\mathbb{R}^2$ be linearly independent?
2. **Must** any two vectors in $\mathbb{R}^2$ be linearly independent?
3. If two vectors in $\mathbb{R}^3$ are linearly independent, what do they span?
4. If three vectors in $\mathbb{R}^3$ are linearly independent, what do they span?
5. Given d vectors in $\mathbb{R}^n$, what must be true about d and n for it to be possible for the vectors to be linearly independent?

Solutions

1. Can any three vectors in $\mathbb{R}^2$ be linearly independent? **No**. Any three vectors in $\mathbb{R}^2$ must be linearly dependent, since $\mathbb{R}^2$ is only 2-dimensional. Intuitively, $\mathbb{R}^2$ only has two independent directions, and so you only need two vectors to reach every vector in it. Given a third, you can always write it as a linear combination of the first two.
2. **Must** any two vectors in $\mathbb{R}^2$ be linearly independent? **No**. They could be collinear, like with $\begin{bmatrix}1\\2\end{bmatrix}$ and $\begin{bmatrix}2\\4\end{bmatrix}$.
3. If two vectors in $\mathbb{R}^3$ are linearly independent, what do they span? **A plane**.
4. If three vectors in $\mathbb{R}^3$ are linearly independent, what do they span? **All of** $\mathbb{R}^3$.
5. Given d vectors in $\mathbb{R}^n$, what must be true about d and n for it to be possible for the vectors to be linearly independent? $d \leq n$. If $d > n$, then at least one of them must be a linear combination of the others, since $\mathbb{R}^n$ only has n independent directions.

Here's one final abstract activity to think about.

Activity 3

Activity 3

1. Suppose we have 8 vectors in $\mathbb{R}^{17}$.
   - Could they be linearly independent?
   - Could they span all of $\mathbb{R}^{17}$?
2. Suppose we have 17 vectors in $\mathbb{R}^8$.
   - Could they be linearly independent?
   - Could they span all of $\mathbb{R}^8$?

4.3. Vector Spaces, Basis, and Dimension

In Chapter 4.1 and Chapter 4.2, I introduced the key ideas of span and linear independence. In this section, we'll revisit these ideas from a more abstract and rigorous perspective.

Context

You might be wondering what the point of all of this is – spans, linear independence, vector spaces, etc. What does any of this have to do with machine learning, or more specifically, that problem of predicting commute times given departure times from Chapter 1.2?

Remember, what we're working towards is being able to build hypothesis functions that use multiple features to make predictions.

$$\text{predicted commute time}_i = h(\text{departure hour}_i, \text{day of month}_i, \text{day of week}_i, \text{gas left in tank}_i, ...)$$

As we'll see in Chapter 7, such a hypothesis function involves computing a **linear combination** of the features. To find the best possible model, we'll need to find the best possible linear combination of the features, and to do so, we'll need to think about the space of all possible linear combinations of the features – or the **span** of those features.

At times, what I'm discussing will sound abstract, pedantic, and disconnected from our overarching focus on machine learning. But bear with me – I promise it'll all come together soon.

Vector Spaces

When we first introduced vectors back in Chapter 3.1, I mentioned that they support two key operations: addition and scalar multiplication. These two operations combine through **linear combinations**, as we've seen repeatedly throughout Chapter 3 so far.

So far, we've mostly dealt with **Euclidean vectors**, which are the vectors that we've treated as ordered lists of numbers in $\mathbb{R}^n$. (The term "Euclidean" refers to the fact that the vectors have some geometric meaning; we have thought of a vector as an arrow from the origin to a point in space. The term Euclidean is named after the Greek mathematician Euclid, who was one of the pioneers of the study of geometry.)

$$\underbrace{\vec{v} = \begin{bmatrix} 3 \\ -2 \\ 8 \\ 5 \\ 3 \end{bmatrix}, \quad \vec{x} = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix}}_{\text{Euclidean vectors}}$$

However, it's sometimes useful to consider vectors in a more abstract setting. Euclidean vectors are not the only mathematica objects that support addition and scalar multiplication!

Definition: Vector Space

A **vector space** is a set of objects V where:

1. We can add any two elements of V and the result is also an element of V.

$$\vec{u}, \vec{v} \in V \implies \vec{u} + \vec{v} \in V$$

2. We can multiply any element of V by a scalar c and the result is also an element of V.

$$\vec{v} \in V, \quad c \in \mathbb{R} \implies c\vec{v} \in V$$

Together, the two properties above tell us that **any linear combination of vectors in V is also a vector in V.**

$$\vec{u}, \vec{v} \in V, \quad c, d \in \mathbb{R} \implies c\vec{u} + d\vec{v} \in V$$

$\mathbb{R}^2$ is a vector space: if you take any two vectors in $\mathbb{R}^2$ and add them together, you will still have a vector in $\mathbb{R}^2$. Multiply a vector in $\mathbb{R}^2$ by a scalar, and you'll still have a vector in $\mathbb{R}^2$.

$$\underbrace{\vec{u} = \begin{bmatrix} 3 \\ 4 \end{bmatrix}, \quad \vec{v} = \begin{bmatrix} -1 \\ 2 \end{bmatrix}}_{\text{two vectors in } \mathbb{R}^2}$$

$$\underbrace{2\vec{u} - 3\vec{v} = \begin{bmatrix} 9 \\ 2 \end{bmatrix}}_{\text{also a vector in } \mathbb{R}^2!}$$

The same holds for $\mathbb{R}^3$, $\mathbb{R}^4$, and more generally, $\mathbb{R}^n$. These are all examples of vector spaces.

What might be a vector space that isn't made up of our regular Euclidean vectors? One possible vector space is the set of all polynomials with a **degree less than or equal to 3** (i.e. degree "at most 3"). This vector space, denoted by $\mathcal{P}_3$, is the set of all polynomials of the form

$$p(x) = a_0 + a_1 x + a_2 x^2 + a_3 x^3$$

where a_3, a_2, a_1, a_0 are real numbers (potentially equal to 0).

$$u(x) = 2x^3 - 16x^2 + 7x - 1$$

$$v(x) = 3x^3 - 4x^2 + 2x + 5$$

$$w(x) = x + 4$$

$u(x)$, $v(x)$, and $w(x)$ are all elements of $\mathcal{P}_3$; think of each of these polynomials as a vector in $\mathcal{P}_3$.

Why is $\mathcal{P}_3$ a vector space? If you add any two polynomials with a degree of at most 3 together, you will still have a polynomial with degree at most 3. Likewise, if you multiply a polynomial with degree at most 3 by a scalar, you will still have a polynomial with degree at most 3.

Activity 1

Activity 1

Why are the set of polynomials of degree **exactly** 3 **not** a vector space?

Solution

It's possible to add two polynomials with degree exactly 3 together, and get a polynomial with degree 2, which would mean that the result is not in the set of polynomials with degree exactly 3.

$$u(x) = x^3 + x^2 - 15$$

$$v(x) = -x^3 + 4$$

Here, $u(x) + v(x) = x^2 + 11$, which is not a polynomial with degree exactly 3.

There are other examples of abstract vector spaces, too, like the set of all matrices with real entries, or the set of all continuous functions from $\mathbb{R}$ to $\mathbb{R}$. In the definition of a vector space, I mentioned that the objects in the space should support addition and scalar multiplication. We know how these operations work for our traditional definition of vectors, but what do these operations look like for generic mathematical objects?

Vector Space Properties

In order for a set of objects V to be a vector space, their definitions of addition and scalar multiplication must satisfy the following 8 properties, for elements $\vec{u}, \vec{v}, \vec{w} \in V$ and scalars $c, d \in \mathbb{R}$.

	Property	Description
1	$\vec{u} + \vec{v} = \vec{v} + \vec{u}$	**Addition is commutative**, i.e. you can swap the order of addition
2	$(\vec{u} + \vec{v}) + \vec{w} = \vec{u} + (\vec{v} + \vec{w})$	**Addition is associative**, i.e. it doesn't matter how you group the vectors you're adding
3	There exists an element $\vec{0} \in V$ such that $\vec{v} + \vec{0} = \vec{v}$ for all $\vec{v} \in V$	There exists an **additive identity**, i.e. a zero vector
4	For every $\vec{v} \in V$, there exists an element $-\vec{v} \in V$ such that $\vec{v} + (-\vec{v}) = \vec{0}$	There exist **additive inverses**, i.e. subtracting a vector is the same as adding its negative
5	$1\vec{u} = \vec{u}$	Existence of a **multiplicative identity**, i.e. multiplying a vector by the scalar 1 doesn't change the vector
6	$(cd)\vec{v} = c(d\vec{v})$	**Scalar multiplication is associative**, i.e. it doesn't matter how you group the scalars you're multiplying by
7	$c(\vec{u} + \vec{v}) = c\vec{u} + c\vec{v}$	**Scalar multiplication distributes over vector addition**, i.e. you can distribute a scalar over a sum of vectors
8	$(c + d)\vec{v} = c\vec{v} + d\vec{v}$	**Scalar addition distributes over scalar multiplication**, i.e. you can distribute a sum of scalars over a vector

Technically, we could be even more abstract and not restrict the scalars to be real numbers, and instead allow them to be complex numbers. Such a vector space is called a **complex vector space**. But, we'll stick with **real vector spaces** in this class, where the scalars involved in scalar multiplication are real numbers.

And even more practically, we'll stick to focusing on the vector space $\mathbb{R}^n$. Still, I wanted to provide you with some context for how to generalize these ideas.

Activity 2

Activity 2

1. Suppose we *try* to create a new vector space in which vector addition is defined in the following way:

$$\begin{bmatrix} u_1 \\ u_2 \end{bmatrix} + \begin{bmatrix} v_1 \\ v_2 \end{bmatrix} = \begin{bmatrix} u_1 + v_2 \\ u_2 + v_1 \end{bmatrix}$$

So for example $\begin{bmatrix} 1 \\ 2 \end{bmatrix} + \begin{bmatrix} 3 \\ 4 \end{bmatrix} = \begin{bmatrix} 1+4 \\ 2+3 \end{bmatrix} = \begin{bmatrix} 5 \\ 5 \end{bmatrix}$.

Suppose we stick with the usual definition of scalar multiplication. Which of the 8 conditions above

are not satisfied?

2. Suppose we consider the set of vectors with two components, where scalar multiplication is defined as

$$k\begin{bmatrix} v_1 \\ v_2 \end{bmatrix} = \begin{bmatrix} kv_1 \\ v_2 \end{bmatrix}$$

Suppose we stick with the usual definition of vector addition. Which of the 8 conditions above are not satisfied?

Subspaces

As I emphasized above, we'll stick to working with vector spaces of the form $\mathbb{R}^n$, made up of Euclidean vectors. A theme that we've studied over the past few sections is that of a subspace: a subset of a vector space.

- The span of one vector in $\mathbb{R}^2$ is a line through the origin, which is a 1-dimensional subspace of $\mathbb{R}^2$.
- The span of two non-collinear vectors in $\mathbb{R}^3$ is a plane through the origin, which is a 2-dimensional subspace of $\mathbb{R}^3$.

Let me be a bit more precise with what a subspace is, given our new framing of vector spaces.

Definition: Subspace

A subspace S of a vector space V is a set of vectors in V where:

1. The zero vector is in S.

$$\vec{0} \in S$$

2. We can add any two elements of S and the result is also an element of S.

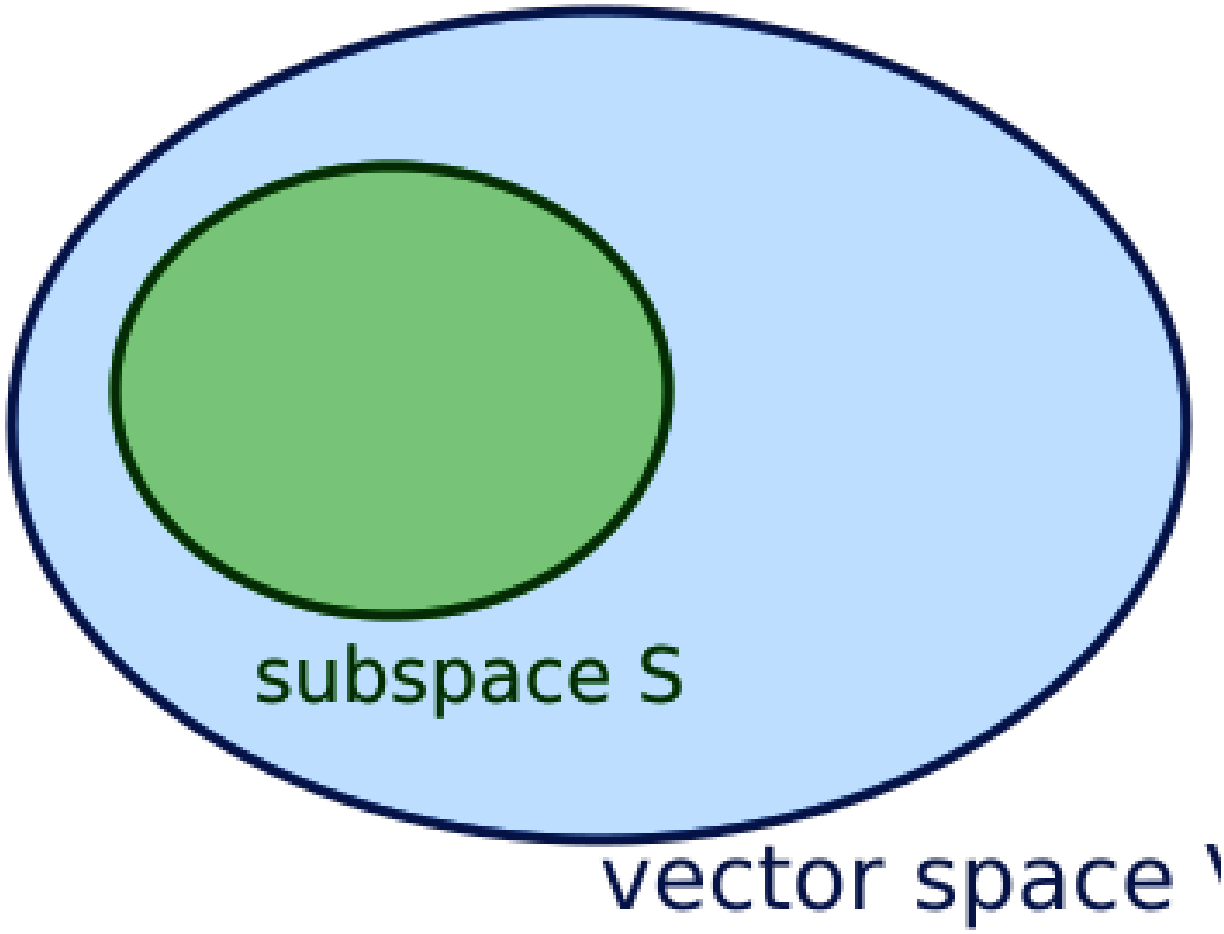
$$\vec{u}, \vec{v} \in S \implies \vec{u} + \vec{v} \in S$$

3. We can multiply any element of S by a scalar c and the result is also an element of S.

$$\vec{u} \in S, c \in \mathbb{R} \implies c\vec{u} \in S$$

The last two properties together tell us that S **must be "closed" under addition and scalar multiplication**, i.e. the must satisfy the "closure" property.

The simpler way of thinking of Properties 2 and 3 above is that if you take any two vectors $\vec{u}, \vec{v} \in S$, then any linear combination $c\vec{u} + d\vec{v}$ must also be in S. If you combine elements in a subspace, they can't "escape" the subspace.



Properties 2 and 3 above sound familiar, since they were the defining properties of a vector space. **Indeed, a subspace is a vector space on its own, it's just that it's also a subset of a larger vector space**, meaning all of its elements are also in V. The "sub" in "subspace" means "under" or "within". A subspace can also just be the entire vector space itself – $\mathbb{R}^2$ is a subspace of $\mathbb{R}^2$, for example – which is why I'll usually phrase definitions and theorems in terms of subspaces, rather than vector spaces.

Sometimes, other math texts will use set notation to denote subspaces, like $S \subseteq V$, but I'll try and avoid unnecessary notation like that here.

Many years ago, one of the great math teachers I had used the following example to explain what the term **closure** meant:

> If you put two elephants in a cage, and one day a third animal comes out, that third animal must still be an elephant. - J. Bruce White, 1938-2025

The point is not just that elephants are animals, but that if you take any two elephants and add them together, you'll still have an elephant.

Let's look at several *possible* subspaces of larger vector spaces, and see if they are subspaces (and if not, why not).

Try these examples yourself before looking at the solutions!

While these examples aren't framed as activities, if you're reviewing after lecture or lab, try and determine whether each of these sets are subspaces *without* reading my answer.

Example: First Component is 1 False. Is the set of all vectors in $\mathbb{R}^2$ whose first component is 1 a subspace of $\mathbb{R}^2$? (Notice that when I ask whether a set is a subspace, I have to specify the vector space it's possibly a subspace **of**.)

In set notation, I could write this set as

$$\left\{\begin{bmatrix}1\\y\end{bmatrix} \mid y \in \mathbb{R}\right\}$$

When I'm presented with a set like this, I like to think of some example vectors in the set, and see if those examples satisfy the properties of a subspace. **If** those examples satisfy the properties, then I'll try and prove that the properties hold for any two vectors in the set.

This set contains vectors like $\begin{bmatrix}1\\2\end{bmatrix}$ and $\begin{bmatrix}1\\-5\end{bmatrix}$. Remember, for this set to be a subspace, it must be closed under addition and scalar multiplication. But, if I add these two example vectors together, I get

$$\begin{bmatrix}1\\2\end{bmatrix} + \begin{bmatrix}1\\-5\end{bmatrix} = \begin{bmatrix}2\\-3\end{bmatrix}$$

which is not a vector in the set, since its first component is not 1. So, this set is not closed under addition, and so it is not a subspace of $\mathbb{R}^2$. I don't need to prove anything in a more general form here, since I've found a counterexample.

Also, this set doesn't contain the zero vector $\begin{bmatrix}0\\0\end{bmatrix}$, which it needs to in order to be a subspace.

Example: First Component is 0 True. Is the set of all vectors in $\mathbb{R}^2$ whose first component is 0 a subspace of $\mathbb{R}^2$? This is the set

$$\left\{\begin{bmatrix}0\\y\end{bmatrix} \mid y \in \mathbb{R}\right\}$$

This set *does* contain the zero vector $\begin{bmatrix}0\\0\end{bmatrix}$, which is a good start.

Some example vectors in this set are $\begin{bmatrix}0\\2\end{bmatrix}$ and $\begin{bmatrix}0\\-5\end{bmatrix}$. Adding these vectors together gives $\begin{bmatrix}0\\-3\end{bmatrix}$, which is also in the set.

That's a good start – but can we verify that **any linear combination** of vectors in the set is another vector in the set?

To do so, let's pick two arbitrary vectors in the set, $\vec{u} = \begin{bmatrix}0\\x\end{bmatrix}$ and $\vec{v} = \begin{bmatrix}0\\y\end{bmatrix}$. A linear combination of these two vectors is $c\vec{u} + d\vec{v}$, or

$$c\begin{bmatrix}0\\x\end{bmatrix} + d\begin{bmatrix}0\\y\end{bmatrix} = \begin{bmatrix}0\\cx+dy\end{bmatrix}$$

No matter what scalars c and d we pick, and no matter what the values of scalars x and y are, the sum $c\vec{u} + d\vec{v}$ is always a vector in $\mathbb{R}^2$ with a first component of 0 and a second component of $cx + dy$. In other words, any linear combination of vectors in the set is also a vector in the set. So, this set **is** closed under addition and scalar multiplication, and so it is a subspace of $\mathbb{R}^2$.

Example: Unit Vectors False. Is the set of all unit vectors in $\mathbb{R}^5$ a subspace of $\mathbb{R}^5$? (There's nothing special about $\mathbb{R}^5$; I just wanted to move away from examples in $\mathbb{R}^2$.) This is the set

$$\{\vec{v} \in \mathbb{R}^5 \mid \|\vec{v}\| = 1\}$$

This is not a subspace of $\mathbb{R}^5$, which we can identify quickly by realizing it doesn't contain the zero vector $\begin{bmatrix}0\\0\\0\\0\\0\end{bmatrix}$.

If we instead changed the set to be all vectors in $\mathbb{R}^5$ with a norm of less than or equal to 1, i.e.

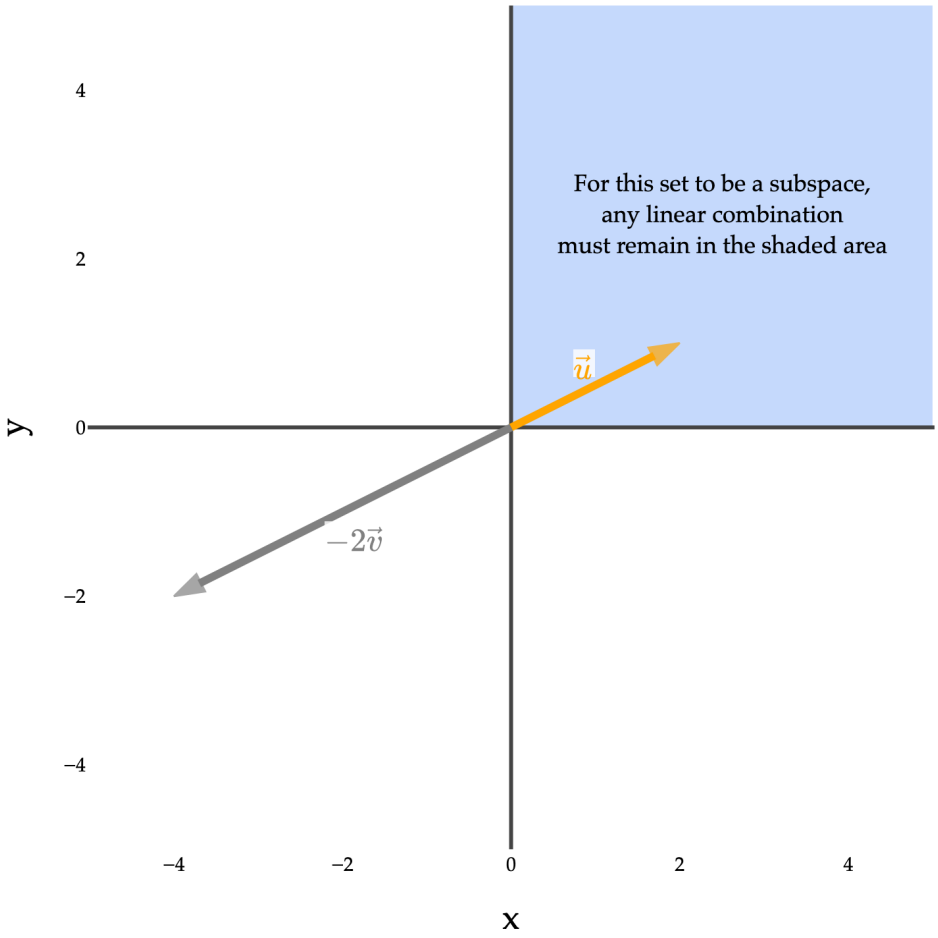
$$\{\vec{v} \in \mathbb{R}^5 \mid \|\vec{v}\| \leq 1\}$$

the resulting set would contain the zero vector in $\mathbb{R}^5$, but still wouldn't be a subspace, since it's not closed under addition or scalar multiplication. $\begin{bmatrix}1\\0\\0\\0\\0\end{bmatrix}$ is in the set, but if we multiply it by 2, we get $\begin{bmatrix}2\\0\\0\\0\\0\end{bmatrix}$, which is not in the set.

Example: Non-Negative Coordinates False. Is the set of all vectors in $\mathbb{R}^2$ with non-negative coordinates a subspace of $\mathbb{R}^2$? This is the set

$$\left\{\begin{bmatrix}x\\y\end{bmatrix} \mid x \geq 0, y \geq 0\right\}$$

Graphically, this set represents the top-right quadrant (also known as the first quadrant) of the xy-plane. If we add two vectors in this set, the third vector will also be in the set, since the sum of two non-negative numbers is non-negative. However, if we multiply a vector in this set by a negative scalar, the resulting vector will not be in the set.



Example: Line through Origin True. Consider the set of all vectors in $\mathbb{R}^3$ that lie on the line spanned by $\begin{bmatrix} 1 \\ -3 \\ 2 \end{bmatrix}$.

Is this a set a subspace of $\mathbb{R}^3$? This is the set

$$\left\{ \begin{bmatrix} 1 \\ -3 \\ 2 \end{bmatrix} \mid t \in \mathbb{R} \right\}$$

Visually, this line looks like

If we pick any two vectors on this line, any of their linear combinations will also lie on this line. Any vector on this line is a scalar multiple of $\begin{bmatrix} 1 \\ -3 \\ 2 \end{bmatrix}$, so two arbitrary vectors on this line might look like

$$\vec{u} = a \begin{bmatrix} 1 \\ -3 \\ 2 \end{bmatrix}, \quad \vec{v} = b \begin{bmatrix} 1 \\ -3 \\ 2 \end{bmatrix}$$

Then, a linear combination of $\vec{u}$ and $\vec{v}$ is

$$c\vec{u} + d\vec{v} = ca\begin{bmatrix}1\\-3\\2\end{bmatrix} + db\begin{bmatrix}1\\-3\\2\end{bmatrix} = (ca + db)\begin{bmatrix}1\\-3\\2\end{bmatrix}$$

which is just another vector on the line! So, since the set is closed under linear combinations, it is a subspace of $\mathbb{R}^3$.

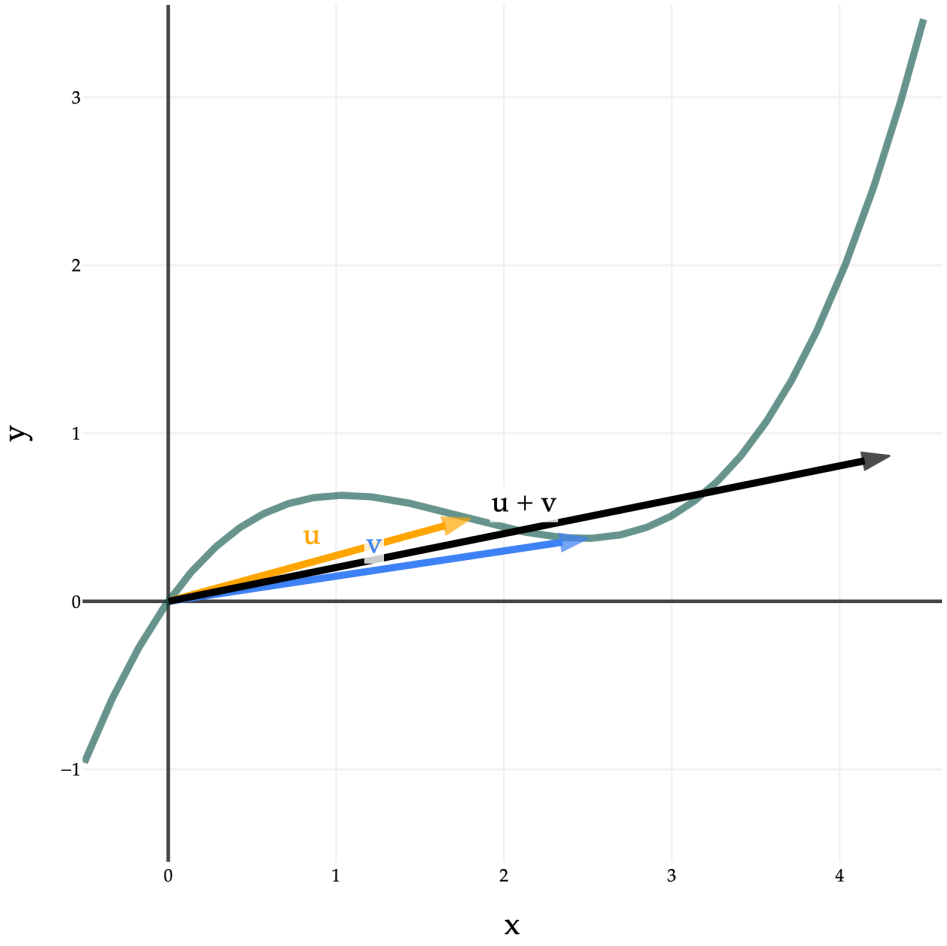
Example: Cubic Polynomials False. We've spent time considering lines that do and don't pass through the origin. What about functions in $\mathbb{R}^2$ that **aren't** lines? Consider the degree 3 polynomial

$$f(x) = 0.18x^3 - 0.95x^2 + 1.4x$$

The set of all points in $\mathbb{R}^2$ that lie on the graph of the function is

$$C = \left\{ \begin{bmatrix} x \\ 0.18x^3 - 0.95x^2 + 1.4x \end{bmatrix} \mid x \in \mathbb{R} \right\}$$

C is a subset of $\mathbb{R}^2$. Again, we're sort of blending the definition of a vector and a point. For every point on the curve, include the vector whose tail is at the origin and whose endpoint is that point. (This is implicitly what we did in the previous example too. By saying that a vector lives on a line, I meant that its endpoint is on that line.)



Even though this is a subset of $\mathbb{R}^2$ and it contains $\vec{0} = \begin{bmatrix} 0 \\ 0 \end{bmatrix}$, it is not a subspace of $\mathbb{R}^2$.

Why? **It is not closed under addition.** If you pick two vectors that live on this curve and add them, the result is not necessarily on the curve, as shown above.

Spans are Subspaces, and Vice Versa

The last example we walked through above, $\left\{ \begin{bmatrix} 1 \\ -3 \\ 2 \end{bmatrix} \mid t \in \mathbb{R} \right\}$, is a subspace of $\mathbb{R}^3$. It is also the span of the vector $\begin{bmatrix} 1 \\ -3 \\ 2 \end{bmatrix}$. It turns out that this correspondence between spans and subspaces is true in general.

Spans are Subspaces

If $\vec{v}_1, \vec{v}_2, \ldots, \vec{v}_d$ are vectors in $\mathbb{R}^n$, then the **span** (i.e. set of all possible linear combinations) of these vectors is a subspace of $\mathbb{R}^n$.
In other words,

$$\text{span}(\{\vec{v}_1, \vec{v}_2, \ldots, \vec{v}_d\}) = \left\{ a_1\vec{v}_1 + a_2\vec{v}_2 + \cdots + a_d\vec{v}_d \;\middle|\; a_1, a_2, \ldots, a_d \in \mathbb{R} \right\}$$

is a subspace of $\mathbb{R}^n$.

This is true essentially by the definition of a span. The span of a collection of vectors is defined to be the set of all of their possible linear combinations. If you pick any two vectors in the span, they're both made of the same building blocks (spanning vectors), so if you make a linear combination of these two new vectors, that'll still be made up of the same building blocks.

Example: Span of Two Vectors in $\mathbb{R}^3$ True. As an example, consider the span of the vectors

$$\vec{u} = \begin{bmatrix} 2 \\ 6 \\ -3 \end{bmatrix}, \quad \vec{v} = \begin{bmatrix} 1 \\ 5 \\ 0 \end{bmatrix}$$

$\text{span}(\{\vec{u}, \vec{v}\})$ contains all vectors of the form $c\vec{u} + d\vec{v}$. Let's pick two such vectors, say

$$\vec{x} = 2\vec{u} + 3\vec{v}, \quad \vec{y} = 4\vec{u} - 2\vec{v}$$

Both $\vec{x}$ and $\vec{y}$ are in $\text{span}(\{\vec{u}, \vec{v}\})$. Now, let's take a linear combination of these two new vectors, say, $5\vec{x} - \frac{1}{2}\vec{y}$, and check if that's also in $\text{span}(\{\vec{u}, \vec{v}\})$. This linear combination is

$$5\vec{x} - \frac{1}{2}\vec{y} = 5(2\vec{u} + 3\vec{v}) - \frac{1}{2}(4\vec{u} - 2\vec{v})$$

But, simplifying this expression shows us that it's equivalent to

$$5\vec{x} - \frac{1}{2}\vec{y} = 10\vec{u} + 15\vec{v} - 2\vec{u} + \vec{v} = 8\vec{u} + 16\vec{v}$$

which clearly shows that a linear combination of $\vec{x}$ and $\vec{y}$ is really just a linear combination of $\vec{u}$ and $\vec{v}$.

All of this is to say, if you pick any two vectors in $\text{span}(\{\vec{u}, \vec{v}\})$, their linear combinations will also be in $\text{span}(\{\vec{u}, \vec{v}\})$, so $\text{span}(\{\vec{u}, \vec{v}\})$ is closed under addition and scalar multiplication. It also contains the zero vector (since you could take $0\vec{u} + 0\vec{v} = \vec{0}$), so this span is a subspace of $\mathbb{R}^3$.

Let's not forget that the span of two non-collinear vectors in $\mathbb{R}^3$ is a plane. So, since $\text{span}(\{\vec{u}, \vec{v}\})$ is a subspace, the corresponding plane is also a subspace of $\mathbb{R}^3$ (because it's the same thing as the span of the vectors).

Chapter 4.4 shows us how to find the equation of a plane in standard form ($ax + by + cz + d = 0$) given two non-collinear vectors in the plane. That specific equation in this case happens to be $15x - 3y + 4z = 0$, which means the set of vectors $\begin{bmatrix} x \\ y \\ z \end{bmatrix}$ that satisfy this equation form a subspace of $\mathbb{R}^3$.

Example: Plane $x + y + z = 0$ True. What if we're instead given a plane, rather than the span of a collection of vectors?

Subspaces are Spans, Too!

Given any subspace S of a vector space V, there exists some collection of vectors in V such that $S = \text{span}(\{\vec{v}_1, \vec{v}_2, \ldots, \vec{v}_d\})$.

It turns out that not only is every span a subspace, but every subspace can be spanned by some collection of vectors. The plane $x + y + z = 0$ is a subspace of $\mathbb{R}^3$. Can we write it as the span of a collection of vectors? Sure we can. We know that it only takes two non-collinear vectors to span a plane, so all we need to do is pick two vectors that lie in $x + y + z = 0$.

Arbitrarily, let's pick the vectors $\begin{bmatrix} 1 \\ -1 \\ 0 \end{bmatrix}$ and $\begin{bmatrix} 2 \\ 1 \\ -3 \end{bmatrix}$. These vectors aren't collinear (as a reminder, this means that one is not a scalar multiple of the other), so they span a plane.

So, the plane $x + y + z = 0$ is the span of the vectors $\begin{bmatrix} 1 \\ -1 \\ 0 \end{bmatrix}$ and $\begin{bmatrix} 2 \\ 1 \\ -3 \end{bmatrix}$, and so it's a subspace of $\mathbb{R}^3$. These are not the only two vectors that span this plane; there are infinitely many pairs of vectors that can be said to span this plane. I just picked a pair that happens to have nice numbers.

I've plotted the plane below, without the vectors $\begin{bmatrix} 1 \\ -1 \\ 0 \end{bmatrix}$ and $\begin{bmatrix} 2 \\ 1 \\ -3 \end{bmatrix}$, to emphasize that you should think of planes **through the origin** as subspaces in their own right.

Example: Plane $x + y + z = 5$ False. What about a plane that **doesn't** pass through the origin? Is the plane $x + y + z = 5$ a subspace of $\mathbb{R}^3$?

Remember, subspaces must contain the zero vector, but the vector $\begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix}$ doesn't satisfy $x + y + z = 5$. So, the plane $x + y + z = 5$ is not a subspace of $\mathbb{R}^3$.

Equivalently, you know that the span of any set of vectors must also always contain the zero vector, since you're always able to create the zero vector by taking 0 of each of the spanning vectors, $0\vec{v}_1 + 0\vec{v}_2 + \cdots + 0\vec{v}_d = \vec{0}$. So, if a set does not contain the zero vector, it can't be the span of any set of vectors, but as we just saw, all subspaces

must be spanned by some set of vectors. **If it can't be a span, it can't be a subspace**, and so the plane $x+y+z=5$ is not a subspace of $\mathbb{R}^3$.

Basis and Dimension

In Chapter 4.2, we looked at how to find linearly independent subsets with the same span as a given set of vectors. We like the vectors we're dealing with to be linearly independent, so that any of their linear combinations are unique and can only be created in one way. (This was proved in Chapter 4.2.)

The specific example we looked at was

$$\vec{v}_1 = \begin{bmatrix} 3 \\ 4 \\ 0 \end{bmatrix}, \quad \vec{v}_2 = \begin{bmatrix} 0 \\ 1 \\ 1 \end{bmatrix}, \quad \vec{v}_3 = \begin{bmatrix} -3 \\ -2 \\ 2 \end{bmatrix}, \quad \vec{v}_4 = \begin{bmatrix} 6 \\ 5 \\ -3 \end{bmatrix}, \quad \vec{v}_5 = \begin{bmatrix} 2 \\ 5 \\ 1 \end{bmatrix}$$

And, through the algorithm introduced at the bottom of that section, we found that the vectors $\vec{v}_1, \vec{v}_2, \vec{v}_5$ alone have the same span as the original set of 5 vectors, but are also linearly independent.

$$\text{span}(\{\vec{v}_1, \vec{v}_2, \vec{v}_5\}) = \text{span}(\{\vec{v}_1, \vec{v}_2, \vec{v}_3, \vec{v}_4, \vec{v}_5\})$$

The vectors $\vec{v}_3$ and $\vec{v}_4$ can be written as linear combinations of $\vec{v}_1, \vec{v}_2, \vec{v}_5$, so they are not needed to describe the span of the original set of 5 vectors. In particular,

$$\vec{v}_3 = -\vec{v}_1 + 2\vec{v}_2, \quad \vec{v}_4 = 2\vec{v}_1 - 3\vec{v}_2$$

There is a special name for a set of vectors that are both linearly independent and span a subspace, like $\vec{v}_1, \vec{v}_2, \vec{v}_5$ above.

Definition: Basis

A **basis** for a subspace S is a set of vectors that:

1. span all of S.
2. are linearly independent.

In the example above, the vectors $\vec{v}_1, \vec{v}_2, \vec{v}_5$ are a basis for the span of the original set of 5 vectors, or equivalently, a basis for the subspace of $\mathbb{R}^3$ containing all of the vectors that can be written as linear combinations of the original 5 vectors.

Think of a basis as the minimum set of vectors you need to describe every vector in a subspace:

- If you remove a vector from a basis, you'll no longer be able to reach every vector in the subspace.
- If you add an extra vector to a basis, the new set will no longer be linearly independent.

Definition: Dimension

The **dimension** of a subspace S, denoted $\dim(S)$, is the number of vectors in a basis for S.
All bases for a subspace have the same number of vectors; this number is the dimension.

Again, continuing with the example above, since we've found a basis containing 3 vectors for the span of the original set of 5 vectors, the dimension of the span of the original set of 5 vectors is **3** (not 5). So, I might say

$$\dim(\text{span}(\{\vec{v}_1, \vec{v}_2, \vec{v}_3, \vec{v}_4, \vec{v}_5\})) = 3$$

A basis for a subspace is not unique. Sure, $\vec{v}_1, \vec{v}_2, \vec{v}_5$ is a basis for the span of the original set of 5 vectors, but so is $\vec{v}_3, \vec{v}_4, \vec{v}_5$, another collection of 3 vectors.

And, since both of these bases ($\vec{v}_1, \vec{v}_2, \vec{v}_5$ and $\vec{v}_3, \vec{v}_4, \vec{v}_5$) involve 3 linearly independent vectors in $\mathbb{R}^3$, they span **all** of $\mathbb{R}^3$. What's another set of linearly independent vectors in $\mathbb{R}^3$ that span $\mathbb{R}^3$? There's the easy choice, of $\begin{bmatrix}1\\0\\0\end{bmatrix}, \begin{bmatrix}0\\1\\0\end{bmatrix}, \begin{bmatrix}0\\0\\1\end{bmatrix}$.

Standard Basis for $\mathbb{R}^n$. We say the **standard basis** for $\mathbb{R}^n$ is the set of vectors $\begin{bmatrix}1\\0\\0\\\vdots\\0\end{bmatrix}, \begin{bmatrix}0\\1\\0\\\vdots\\0\end{bmatrix}, \ldots, \begin{bmatrix}0\\0\\0\\\vdots\\1\end{bmatrix}$. Sometimes these vectors are written as $\vec{e}_1, \vec{e}_2, \ldots, \vec{e}_n$. In addition to being a basis for $\mathbb{R}^n$, they're also unit vectors, and orthogonal.

Since the standard basis for $\mathbb{R}^n$ is, well, a basis, and any basis for a subspace has the same number of elements, we know that $\mathbb{R}^n$ is n-dimensional.

In $\mathbb{R}^2$, the standard basis consists of $\vec{e}_1 = \begin{bmatrix}1\\0\end{bmatrix}$ and $\vec{e}_2 = \begin{bmatrix}0\\1\end{bmatrix}$. $\vec{e}_1$ and $\vec{e}_2$ are linearly independent (neither is a scalar multiple of the other), and they span all of $\mathbb{R}^2$, which means that any other vector in $\mathbb{R}^2$ can be written as a linear combination of them.

Pick your favorite vector with two components, say $\begin{bmatrix}5\\-12\end{bmatrix}$, and you'll be able to write it as a linear combination of the standard basis.

$$\begin{bmatrix}5\\-12\end{bmatrix} = 5\begin{bmatrix}1\\0\end{bmatrix} - 12\begin{bmatrix}0\\1\end{bmatrix} = \underbrace{5\vec{v}_1 - 12\vec{v}_2}_{\text{linear combination of standard basis}}$$

Let's look at a few more examples of possible bases for given subspaces.

Quick Examples.

Vectors	Are they a basis for...	Answer	Why?
$\begin{bmatrix}1\\0\end{bmatrix}, \begin{bmatrix}1\\5\end{bmatrix}$	$\mathbb{R}^2$	Yes	They're linearly dent, and span
$\begin{bmatrix}1\\0\\0\end{bmatrix}, \begin{bmatrix}0\\1\\0\end{bmatrix}$	$\mathbb{R}^3$	No	They're linearly dent, but don't s $\mathbb{R}^3$, since no line nation of them $\begin{bmatrix}0\\0\\1\end{bmatrix}$ (or any c tor with a non-z component).
$\begin{bmatrix}1\\0\\0\end{bmatrix}, \begin{bmatrix}0\\1\\0\end{bmatrix}, \begin{bmatrix}3\\4\\5\end{bmatrix}$	$\mathbb{R}^3$	Yes	They're linearly dent, and span They're not the basis, but they basis.
$\begin{bmatrix}1\\0\\0\end{bmatrix}, \begin{bmatrix}0\\1\\0\end{bmatrix}, \begin{bmatrix}0\\0\\1\end{bmatrix}, \begin{bmatrix}3\\4\\5\end{bmatrix}$	$\mathbb{R}^3$	No	They span all they're not line pendent. $\mathbb{R}^3$ h sion 3, so a bas must have 3 vec

Finding a Basis for a Subspace. Often, we'll need to find a basis for a given subspace. If the subspace in question is the span of a set of vectors, we can use the algorithm from the bottom of Chapter 4.2 to find a basis.

```
given v_1, v_2, ..., v_d
initialize linearly independent set S = {v_1}
for i = 2 to d:
    if v_i is not a linear combination of S:
        add v_i to S
```

All this algorithm does is remove linearly dependent vectors from left to right, so that the remaining vectors are linearly independent.

For example, if

$$\vec{v}_1 = \begin{bmatrix}1\\1\\1\end{bmatrix}, \quad \vec{v}_2 = \begin{bmatrix}1\\-2\\-3\end{bmatrix}, \quad \vec{v}_3 = \begin{bmatrix}-3\\0\\1\end{bmatrix}$$

A basis for span $\left(\{\vec{v}_1, \vec{v}_2, \vec{v}_3\}\right)$ is $\{\vec{v}_1, \vec{v}_2\}$, since these two vectors are linearly independent and span the same subspace as the original three vectors; $\vec{v}_3$ is a linear combination of $\vec{v}_1$ and $\vec{v}_2$, specifically $\vec{v}_3 = -2\vec{v}_1 - \vec{v}_2$.

Equivalently, $\{\vec{v}_1, \vec{v}_3\}$ is also a basis for the same subspace, since we can equivalently say $\vec{v}_2 = -2\vec{v}_1 - \vec{v}_3$. Either way, $\dim\left(\text{span}\left(\{\vec{v}_1, \vec{v}_2, \vec{v}_3\}\right)\right) = 2$.

See the bottom of Chapter 4.2 for another worked example.

What about subspaces that aren't expressed explicitly as a span – how do we find a basis for them? For instance, consider the subspace of $\mathbb{R}^3$ containing vectors $\begin{bmatrix} x \\ y \\ z \end{bmatrix}$ such that $2x - 3y + 4z = 0$. This is a plane through the origin, and so we know it's spanned by two non-collinear vectors in $\mathbb{R}^3$.

Some subspaces aren't explicitly expressed as a span, but we can still find a basis for them. We looked at an example of this earlier in this section, before we knew what a basis was. Specifically, we saw the plane of points $\begin{bmatrix} x \\ y \\ z \end{bmatrix}$ such that $x + y + z = 0$ is spanned by $\begin{bmatrix} 1 \\ -1 \\ 0 \end{bmatrix}$ and $\begin{bmatrix} 2 \\ 1 \\ -3 \end{bmatrix}$. Since these two vectors are linearly independent, they form a basis for the subspace (a basis of dimension 2).

Example: Equal Second and Fourth Components True. Consider the set

$$S = \left\{ \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \end{bmatrix} \in \mathbb{R}^5 \mid x_2 = x_4 \right\}$$

S contains all vectors in $\mathbb{R}^5$ whose second and fourth components are equal. S is a subspace of $\mathbb{R}^5$. Any vector in S can be written as

$$\begin{bmatrix} a \\ \boxed{b} \\ c \\ \boxed{b} \\ d \end{bmatrix}$$

for some scalars a, b, c, d.

- The zero vector is in S, since $\begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix}$ has equal second and fourth components.
- If $\vec{u} = \begin{bmatrix} a \\ \boxed{b} \\ c \\ \boxed{b} \\ d \end{bmatrix}$ and $\vec{v} = \begin{bmatrix} e \\ \boxed{f} \\ g \\ \boxed{f} \\ h \end{bmatrix}$, then for any scalars α, β,

$$\alpha\vec{u} + \beta\vec{v} = \begin{bmatrix} \alpha a + \beta e \\ \boxed{\alpha b + \beta f} \\ \alpha c + \beta g \\ \boxed{\alpha b + \beta f} \\ \alpha d + \beta h \end{bmatrix}$$

and the second and fourth components are still equal. This means that S is closed under addition and scalar multiplication.

Now that we know S is a subspace, let's find a basis for it. Since vectors in S have the form $\begin{bmatrix} a \\ b \\ c \\ b \\ d \end{bmatrix}$, any vector in S can be decomposed as follows:

$$\begin{bmatrix} a \\ b \\ c \\ b \\ d \end{bmatrix} = a \begin{bmatrix} 1 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix} + b \begin{bmatrix} 0 \\ 1 \\ 0 \\ 1 \\ 0 \end{bmatrix} + c \begin{bmatrix} 0 \\ 0 \\ 1 \\ 0 \\ 0 \end{bmatrix} + d \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 1 \end{bmatrix}$$

Notice above that only the second vector has a non-zero second and fourth component, which forces the second and fourth components of any vector written in the form above to be equal – the other three vectors contribute 0 to the second and fourth components.

So,

$$S = \text{span}\left(\underbrace{\left\{ \begin{bmatrix} 1 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix}, \begin{bmatrix} 0 \\ 1 \\ 0 \\ 1 \\ 0 \end{bmatrix}, \begin{bmatrix} 0 \\ 0 \\ 1 \\ 0 \\ 0 \end{bmatrix}, \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 1 \end{bmatrix} \right\}}_{\text{basis for } S} \right)$$

The collection of vectors $\left\{ \begin{bmatrix} 1 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix}, \begin{bmatrix} 0 \\ 1 \\ 0 \\ 1 \\ 0 \end{bmatrix}, \begin{bmatrix} 0 \\ 0 \\ 1 \\ 0 \\ 0 \end{bmatrix}, \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 1 \end{bmatrix} \right\}$ is linearly independent, so they form a basis for S. Hence,

$$\dim(S) = 4$$

Activity 3

Activity 3

For each of the following subspaces, find **two** possible bases and state the dimension.

1. The subspace of $\mathbb{R}^3$ containing vectors $\begin{bmatrix} x \\ y \\ z \end{bmatrix}$ such that $2x - 3y + 4z = 0$.
2. The subspace of $\mathbb{R}^6$ containing vectors in which the third and fourth coordinates are 0.

Orthogonal Complements

To wrap up, let me introduce an idea that we'll rely on heavily once we get to Chapter 5.4.

Definition: Orthogonal Complement

If S is a subspace of $\mathbb{R}^n$, then the **orthogonal complement** of S, denoted $S^\perp$, is

$$S^\perp = \{\vec{x} \in \mathbb{R}^n \mid \vec{x} \cdot \vec{s} = 0 \text{ for every } \vec{s} \in S\}$$

$S^\perp$ contains all vectors that are orthogonal to **every** vector in S; $S^\perp$ is also a subspace of $\mathbb{R}^n$.

For example, suppose S is the span of the vectors $\begin{bmatrix} -2 \\ 3 \\ 1 \end{bmatrix}$ and $\begin{bmatrix} -2 \\ 0 \\ 1 \end{bmatrix}$.

$$S = \text{span}\left(\left\{\begin{bmatrix} -2 \\ 3 \\ 1 \end{bmatrix}, \begin{bmatrix} -2 \\ 0 \\ 1 \end{bmatrix}\right\}\right)$$

S is a plane through the origin. Then, $S^\perp$ is the set of vectors perpendicular to this plane. What is $S^\perp$? It is a line through the origin. The direction vector that defines this line is the cross product of the two spanning vectors of S.

Any vector that lives on the line is orthogonal to any vector that lives on the plane, which is what makes the line and plane orthogonal complements of each other.

When we get to Chapter 5.4, we'll see that matrices contain several subspaces and their orthogonal complements. Our knowledge of orthogonal complements will help us understand these subspaces better, which will be useful when we revisit the problem of finding optimal model parameters for regression models.

The gist of the connection for now is this: given a subspace S, **any** vector in $\mathbb{R}^n$ can be written as the sum of a vector in S and a vector in $S^\perp$. This relates to the idea of orthogonal projections from Chapter 3.4, where the error vector when projecting $\vec{u}$ onto $\vec{v}$ is orthogonal to $\vec{v}$. More in due time!

Activity 4

Activity 4

For any subspace S, what is $S \cap S^\perp$, i.e. the intersection of S and its orthogonal complement?

Solution

$S \cap S^\perp = \{\vec{0}\}$.

Activity 5

Activity 5

Find the orthogonal complement of the span of $\begin{bmatrix} 1 \\ 2 \\ 3 \end{bmatrix}$.

Solution

We're looking for the subspace containing all vectors in $\mathbb{R}^3$ that are orthogonal to $\begin{bmatrix}1\\2\\3\end{bmatrix}$ (and all scalar multiples of this vector). This is the plane

$$x + 2y + 3z = 0$$

Why? If we inspect the equation above, we see that any vector $\begin{bmatrix}x\\y\\z\end{bmatrix}$ that satisfies this equation is orthogonal to $\begin{bmatrix}1\\2\\3\end{bmatrix}$, since $x + 2y + 3z = 0$ is the dot product of $\begin{bmatrix}x\\y\\z\end{bmatrix}$ and $\begin{bmatrix}1\\2\\3\end{bmatrix}$.

4.4. Lines, Planes, Hyperplanes, and the Cross Product

Think of this section as a brief detour from the main storyline of the course. Here, I'll cover how to describe lines, planes, and hyperplanes in $\mathbb{R}^n$, and how they relate to our understanding of spans and subspaces from earlier in this chapter.

Lines

Intuition in $\mathbb{R}^2$ and $\mathbb{R}^3$. We're very familiar with lines in $\mathbb{R}^2$. The line $y = 2x + 3$, in $\mathbb{R}^2$, is the set of all (x, y) coordinates that satisfy the equation $y = 2x + 3$.

The line $y = 2x + 3$ is in **slope-intercept form**, which more generally looks like $y = w_0 + w_1 x$. Sometimes, we may write lines in **standard form**, like $2x - y + 3 = 0$, or more generally, $ax + by + c = 0$.

Let's kick things up a notch and consider lines in $\mathbb{R}^3$. **What is the equation of the** line below?

As we saw in Chapter 4.1, the line shown above is the span of the vector $\vec{v} = \begin{bmatrix} 2 \\ -1 \\ 3 \end{bmatrix}$. It passes through the origin, $(0, 0, 0)$, and passes through the point $(2, -1, 3)$.

Ideally, we'd be able to express the line as a function

$$z = f(x, y)$$

as we did in the $\mathbb{R}^2$ case, where $y = f(x) = mx + b$.

Unfortunately, there is no way to express the lines in $\mathbb{R}^3$, or $\mathbb{R}^4$, or $\mathbb{R}^n$ for $n > 2$ as a simple function. Why not? If there were some formula for z (the height of the line) in terms of x and y, that would imply that we should be able to plug in **any** x and **any** y to get an output z. But, the line above only works for very specific combinations of x and y. For instance, there's no point on the line above that has $x = 1$ and $y = 1$. Rather, when $x = 1$, y is forced to be $-\frac{1}{2}$, and z is forced to be $\frac{3}{2}$.

The key idea that I stressed in Chapter 4.1 is that **lines are 1-dimensional objects**, meaning that the location of any point on the line can be described using a single free variable.

So, the equation of the line above is

$$L = t \begin{bmatrix} 2 \\ -1 \\ 3 \end{bmatrix}, t \in \mathbb{R}$$

t here is a free variable – sometimes called a parameter (though this term is confusing in the context of our course) – meaning we can set it to whatever we'd like. The line is the set of all points that can be reached by plugging in different values of t.

Since the line is really a set of points, I should have written it as

$$L = \{t \begin{bmatrix} 2 \\ -1 \\ 3 \end{bmatrix} \mid t \in \mathbb{R}\}$$

but I'll use the former notation for brevity.

Equivalently, you can think of the line as three separate functions of t. Pick a t. Then, L is

$$x = 2t$$
$$y = -t$$
$$z = 3t$$

Drag the value of t below to see how t allows us to move along the line.

The line L above passes through the origin, since if we set $t = 0$, we get the point $(0, 0, 0)$. This matches what we'd expect out of the span of a single vector, since $0\vec{v} = \vec{0}$.

But how do we express a line that passes through some other fixed point that **isn't** the origin? Such a line might not be the span of a single vector, since the span of a single vector is always a line that passes through the origin. But, it's good to know how to think about lines in this more general form.

Lines in Parametric Form.

Definition: Line

A **line** in $\mathbb{R}^n$ that passes through the point $\vec{p}_0$ and is parallel to the vector $\vec{v}$ can be described as:

$$L = \vec{p}_0 + t\vec{v}, t \in \mathbb{R}$$

This is sometimes called the **parametric form** of the line, since t can be thought of as a parameter.

The definition above is not specific to 2-dimensional or 3-dimensional space – it works in any $\mathbb{R}^n$. (Technically, I'm mixing the meaning of a point and a vector here, but as long as we remember that points describe positions and vectors describe directions, we should be fine.) Here's a line in $\mathbb{R}^{100}$:

$$L = \begin{bmatrix} 1 \\ 2 \\ 3 \\ 4 \\ \vdots \\ 100 \end{bmatrix} + t \begin{bmatrix} -11 \\ 12 \\ -13 \\ 14 \\ \vdots \\ 110 \end{bmatrix}, t \in \mathbb{R}$$

Note that the parametric form of a line is not unique! Since the parametric definition of a line depends on a "starting point" $\vec{p}_0$, we can pick any starting point we'd like. We can also scale the direction vector by any non-zero scalar. So,

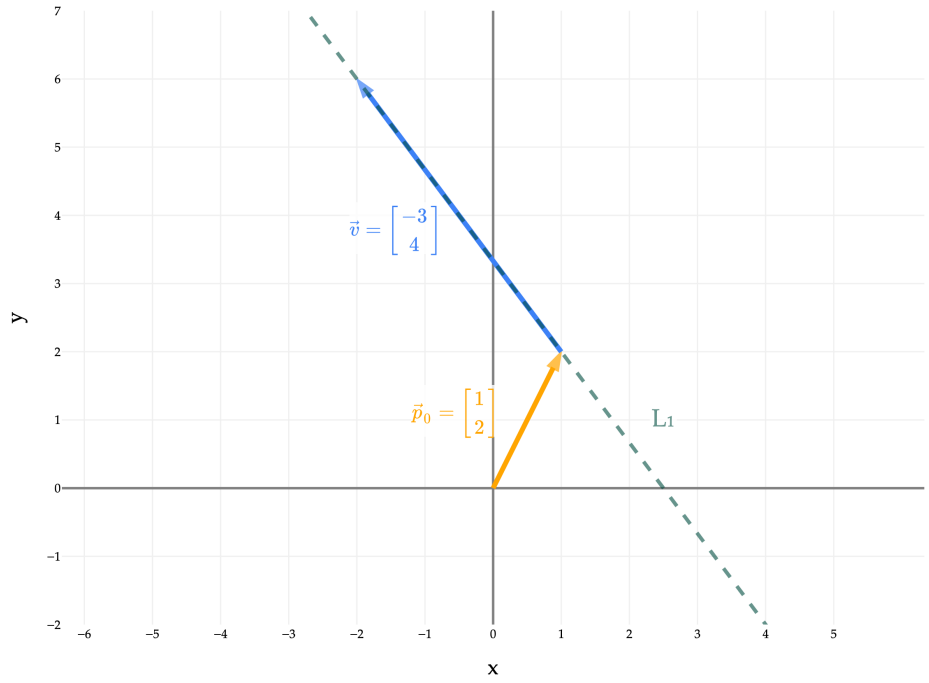
$$L_1 = \begin{bmatrix} 1 \\ 2 \end{bmatrix} + t \begin{bmatrix} -3 \\ 4 \end{bmatrix}, t \in \mathbb{R}$$

is the same line as

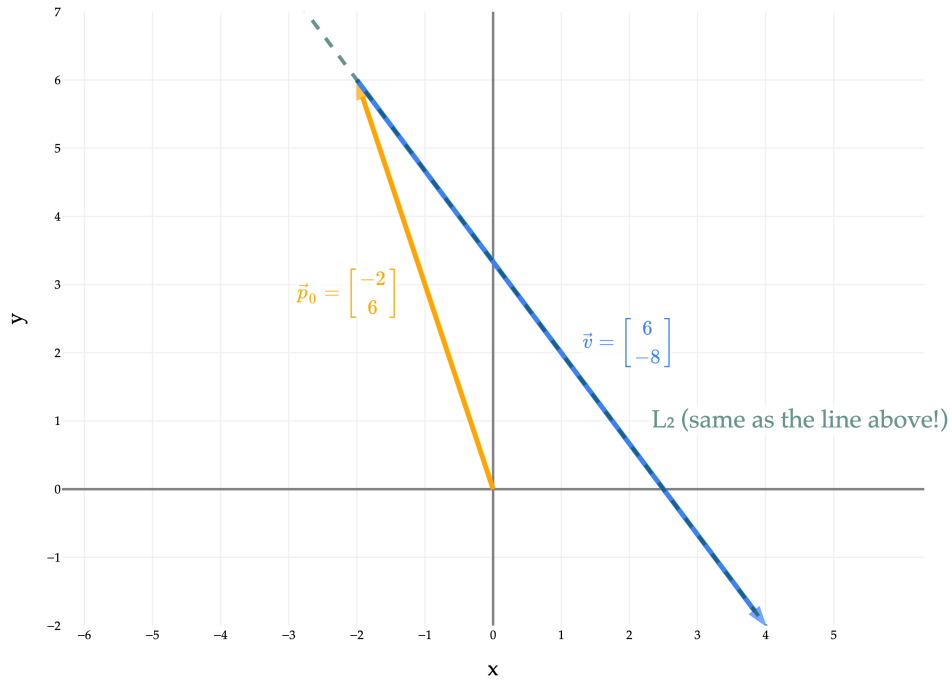
$$L_2 = \begin{bmatrix} -2 \\ 6 \end{bmatrix} + t \begin{bmatrix} 6 \\ -8 \end{bmatrix}, t \in \mathbb{R}$$

once you consider all possible values of t in both cases. (I know this is a little confusing, since plugging the same value of t into L_1 and into L_2 will give you different points, but remember that L_1 and L_2 are sets, and so we need to consider all possible values of t.)

Below is a plot of $L_1 = \begin{bmatrix} 1 \\ 2 \end{bmatrix} + t \begin{bmatrix} -3 \\ 4 \end{bmatrix}, t \in \mathbb{R}$.



And here's $L_2 = \begin{bmatrix} -2 \\ 6 \end{bmatrix} + t \begin{bmatrix} 6 \\ -8 \end{bmatrix}, t \in \mathbb{R}$.



Note that we end up with the same line, despite the different starting points and direction vectors!

The proceeding activities give you some practice with the parametric form of a line.

Activity 1

Activity 1

Write the line $y = -3x + 5$ in parametric form. There are multiple (infinitely many!) possible answers; give just one.

Solution

The line above passes through the point $(0, 5)$ and is parallel to the vector $\begin{bmatrix} 1 \\ -3 \end{bmatrix}$, since for every 1 unit we move in the x-direction, we move -3 units in the y-direction. So, the line is:

$$L = \begin{bmatrix} 0 \\ 5 \end{bmatrix} + t \begin{bmatrix} 1 \\ -3 \end{bmatrix}, t \in \mathbb{R}$$

To verify that we got the right line, let's plug in a few values of t and verify that they match the equation $y = -3x + 5$.

- When $t = 0$, we get the point $(0, 5)$, and $-3(0) + 5 = 5$ True.
- When $t = 1$, we get the point $(1, 2)$, and $-3(1) + 5 = 2$ True.
- When $t = -10$, we get the point $(-10, 35)$, and $-3(-10) + 5 = 35$ True.

(We only need to check two points, since if two lines have the same two points, they must be the same line.)

Activity 2

Activity 2

Find the equation of the line, in parametric form, of the line in $\mathbb{R}^4$ that passes through the points $(5, -1, 3, 2)$ and $(10, -2, 3, 0)$.

Solution

We know one point on the line; we just need to find a direction vector. One way to do this is to subtract the coordinates of the two points:

$$\begin{bmatrix} 10 \\ -2 \\ 3 \\ 0 \end{bmatrix} - \begin{bmatrix} 5 \\ -1 \\ 3 \\ 2 \end{bmatrix} = \begin{bmatrix} 5 \\ -1 \\ 0 \\ -2 \end{bmatrix}$$

So, the line is:

$$L = \begin{bmatrix} 5 \\ -1 \\ 3 \\ 2 \end{bmatrix} + t \begin{bmatrix} 5 \\ -1 \\ 0 \\ -2 \end{bmatrix}, t \in \mathbb{R}$$

Plugging in $t = 0$ and $t = 1$ should give us the two points we know are on the line.

Activity 3

Activity 3

Could the line you found in Activity 2 be described as the span of a single vector? Why or why not?

Solution

No, because it doesn't pass through the origin, and the span of a single vector is always a line that passes through the origin.
Let's look at L once more:

$$L = \begin{bmatrix} 5 \\ -1 \\ 3 \\ 2 \end{bmatrix} + t \begin{bmatrix} 5 \\ -1 \\ 0 \\ -2 \end{bmatrix}, t \in \mathbb{R}$$

Just by looking at the equation above, we don't know for a fact that it doesn't pass through the origin, $(0, 0, 0, 0)$. The fixed point that it is defined relative to, $(5, -1, 3, 2)$, is not the origin, but the origin might still be on the line if we pick the right value of t.
But, we can verify that that's not the case by plugging in $t = -1$, which gives us a 0 in the first two coordinates, but non-zero values in the other coordinates. At $t = -1$:

$$L = \begin{bmatrix} 5 \\ -1 \\ 3 \\ 2 \end{bmatrix} - 1 \begin{bmatrix} 5 \\ -1 \\ 0 \\ -2 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 3 \\ 4 \end{bmatrix}$$

There's no value of t other than -1 such that $5 + t(5) = 0$, so no other value of t will give us the point $(0, 0, 0, 0)$.

Activity 4

Activity 4

Find the equation of the line, in standard form, that is **orthogonal** to the line

$$3x + 4y + 12 = 0$$

and passes through the point $(9, 5)$.

Solution

What does it mean for two lines to be orthogonal? In $\mathbb{R}^2$ or $\mathbb{R}^3$, it's sufficient to say that they're orthogonal if they intersect at a right angle, because this is an idea we can visualize.
But more generally, we should think of two lines as orthogonal if **their direction vectors are orthogonal** when written in parametric form.
That is, $L_1 = \vec{p}_0 + t\vec{v}_1$ and $L_2 = \vec{q}_0 + s\vec{v}_2$ are orthogonal if $\vec{v}_1 \cdot \vec{v}_2 = 0$. The values of the starting points don't change whether the lines are orthogonal, that just changes where they intersect.
Let's return to our problem, which involves finding a line orthogonal to $3x + 4y + 12 = 0$. What is this line in parametric form? Rearranging it to slope-intercept form gives us

$$y = -\frac{3}{4}x - 3$$

which means that for every 1 unit we move in the x-direction, we move $-\frac{3}{4}$ units in the y-direction. So, the direction vector of the line is $\begin{bmatrix} 1 \\ -\frac{3}{4} \end{bmatrix}$, or equivalently, $\begin{bmatrix} 4 \\ -3 \end{bmatrix}$ (I multiplied by 4 to get nice numbers in the direction vector; as we saw earlier, any non-zero scalar multiple of a direction vector will give us the same line).
You might notice a general pattern from this: the direction vector for the line $ax + by + c = 0$ is $\begin{bmatrix} b \\ -a \end{bmatrix}$. I'd avoid memorizing this, though, and would rather you derive it from scratch every time (it's not a good use of your memory to memorize this).
We want a line orthogonal to $3x + 4y + 12 = 0$, so we need a direction vector orthogonal to $\begin{bmatrix} 4 \\ -3 \end{bmatrix}$. A natural choice is to create a direction vector of $\begin{bmatrix} 3 \\ 4 \end{bmatrix}$, since $\begin{bmatrix} 3 \\ 4 \end{bmatrix} \cdot \begin{bmatrix} 4 \\ -3 \end{bmatrix} = 0$.
So, in parametric form, one way to express the line we're looking for is

$$L = \begin{bmatrix} 9 \\ 5 \end{bmatrix} + t \begin{bmatrix} 3 \\ 4 \end{bmatrix}, t \in \mathbb{R}$$

But we want our new line in standard form, i.e. $ax + by + c = 0$. To get this, we can look at the direction vector of the new line, $\begin{bmatrix} 3 \\ 4 \end{bmatrix}$ and recognize that it's saying to move 4 units in the y-direction for every 3 units we move in the x-direction, implying a slope of $\frac{4}{3}$. The new line we're looking for then is $y = \frac{4}{3}x + w_0$, or equivalently $4x - 3y + c = 0$.
To find c, we can plug in the point $(9, 5)$ into the equation:

$$4(9) - 3(5) + c = 0 \implies 36 - 15 + c = 0 \implies c = -21$$

So, the line we're looking for is $\boxed{4x - 3y - 21 = 0}$.
You might also notice that the original line $3x + 4y + 12 = 0$ has coefficients of 3 and 4 on x and y, and the direction vector of the **line orthogonal to it** is $\begin{bmatrix} 3 \\ 4 \end{bmatrix}$. Keep this in mind, as it'll be useful in the section below on planes.

Planes

Lines are 1-dimensional objects, whether they exist in $\mathbb{R}^2$, or $\mathbb{R}^3$, or $\mathbb{R}^{47}$, or in general $\mathbb{R}^n$.

Similarly, planes are 2-dimensional objects. In $\mathbb{R}^2$, since there only exist two dimensions in the first place, the entirety of the coordinate system is one single plane, which we call the xy**-plane**.

Let's start by building intuition for planes in $\mathbb{R}^3$, the most natural setting for them, and then generalize.

Planes in $\mathbb{R}^3$.

Definition: Standard Form of a Plane in $\mathbb{R}^3$

A plane in $\mathbb{R}^3$ is the set of all points (x, y, z) that satisfy the equation

$$ax + by + cz + d = 0$$

where $a, b, c, d \in \mathbb{R}$.
Note that the above equation can also be written in the form

$$z = Ax + By + C$$

where $A = -\frac{a}{c}$, $B = -\frac{b}{c}$, and $C = -\frac{d}{c}$.

For example, let's draw:

1. $3x + 4y - 5z - 12 = 0$, or equivalently $z = \frac{3}{5}x + \frac{4}{5}y + \frac{12}{5}$
2. $-5x - 3y - z = 0$, or equivalently $z = -5x - 3y$

You'll notice that they intersect at a line. This is not a coincidence; any two **non-parallel** planes in $\mathbb{R}^3$ will intersect at a line.

Note that both planes are flat surfaces **that extend infinitely in all directions**. The fact that the blue plane is cut off at the edges is just due to how I'm plotting the planes, **not** that there's some boundary within which the plane is defined.

You'll notice that the blue plane is relatively shallow, while the orange plane is relatively steep. Why?

I find the form $z = Ax + By + C$ easier to understand intuitively, since it shows the rate of change of z with respect to x and y more clearly. Starting with $z = Ax + By + C$, we have that

$$\frac{\partial z}{\partial x} = A, \quad \frac{\partial z}{\partial y} = B$$

In this example, the blue plane has $A = \frac{3}{5}$ and $B = \frac{4}{5}$, while the orange plane has $A = -5$ and $B = -3$, which explains their relative steepness.

That said, be careful, since a plane need not have a non-zero coefficient on z. For example, $3x + 4y = 0$ and $3x + 4y = 5$ is are perfectly valid planes, and they happen to be parallel.

A key property of planes is that they are **flat**. Sure, we know that intuitively, but what does it actually mean?

Planes Contain all Secant Lines

Recall from Appendix 2 that a **secant line** is a line that connects two points on the graph of a function.
A key property of planes is that **if you pick any two points on the plane, the secant line connecting them is entirely on the plane**. This is true for planes in any $\mathbb{R}^n$, not just planes in $\mathbb{R}^3$.

This property is not true in general for other surfaces.

The Cross Product. I first mentioned planes back in Chapter 3.1, when we intuitively discussed the fact that the set of all linear combinations of two non-collinear vectors in $\mathbb{R}^3$ forms a plane. We discussed this idea at length in Chapter 4.1, too.

So, given two vectors $\vec{u} = \begin{bmatrix} u_1 \\ u_2 \\ u_3 \end{bmatrix}, \vec{v} = \begin{bmatrix} v_1 \\ v_2 \\ v_3 \end{bmatrix} \in \mathbb{R}^3$, how do we find the equation of the plane they span, in standard form?

The standard form of a plane in $\mathbb{R}^3$ is

$$ax + by + cz + d = 0$$

We know that the plane spanned by two vectors in $\mathbb{R}^3$ must contain the zero vector, since $0\vec{u} + 0\vec{v} = \vec{0}$. This means that the point $(x, y, z) = (0, 0, 0)$ must satisfy the equation of the plane. Plugging in $(x, y, z) = (0, 0, 0)$ into $ax + by + cz + d = 0$ gives us $d = 0$.

So, I'm searching for a plane of the form $ax + by + cz = 0$. Plugging in $\vec{u} = \begin{bmatrix} u_1 \\ u_2 \\ u_3 \end{bmatrix}$ tells me that a, b, and c must satisfy

$$au_1 + bu_2 + cu_3 = 0$$

Similarly, a, b, and c must also satisfy

$$av_1 + bv_2 + cv_3 = 0$$

Look closely. The left-hand side of both equations looks a lot like the dot product of $\begin{bmatrix} a \\ b \\ c \end{bmatrix}$ with each of $\vec{u}$ and $\vec{v}$. Since those dot products must both be 0 (coming from the right-hand side of each equation), we're really just looking for a vector $\begin{bmatrix} a \\ b \\ c \end{bmatrix}$ that's orthogonal to both $\vec{u}$ and $\vec{v}$.

There are infinitely many vectors orthogonal to a particular pair of vectors $\vec{u}$ and $\vec{v}$, meaning there are infinitely many possible values of a, b, and c that satisfy the above equations. (There are 2 equations but 3 unknowns, so we'd expect there to be infinitely many solutions.)

But, one property that all of these vectors share is that they all point in the same direction – if $\begin{bmatrix} a \\ b \\ c \end{bmatrix}$ is orthogonal to $\vec{u}$ and $\vec{v}$, then so is any non-zero scalar multiple of $\begin{bmatrix} a \\ b \\ c \end{bmatrix}$.

One particular vector (i.e. set of coefficients a, b, and c) that satisfies the above equations is the **cross product** of $\vec{u}$ and $\vec{v}$.

Definition: Cross Product

Suppose $\vec{u} = \begin{bmatrix} u_1 \\ u_2 \\ u_3 \end{bmatrix}$ and $\vec{v} = \begin{bmatrix} v_1 \\ v_2 \\ v_3 \end{bmatrix}$ are in $\mathbb{R}^3$. (Note that the cross product is only defined for two vectors in $\mathbb{R}^3$!)
Then, the cross product of $\vec{u}$ and $\vec{v}$ is given by

$$\vec{u} \times \vec{v} = \begin{bmatrix} u_2v_3 - u_3v_2 \\ u_3v_1 - u_1v_3 \\ u_1v_2 - u_2v_1 \end{bmatrix}$$

There's a lot of meaning baked into the definition of the cross product, but most of it is more relevant in a traditional engineering or physics context. For example, the cross product is **anticommutative**, meaning that the order you compute it in matters.

$$\vec{u} \times \vec{v} = -(\vec{v} \times \vec{u})$$

That's the type of statement we won't bother investigating further here. The key fact that is relevant for us right now is that the vector $\vec{u} \times \vec{v}$ is orthogonal to both $\vec{u}$ and $\vec{v}$.

Activity 5

Activity 5

Verify that the cross product of $\vec{u}$ and $\vec{v}$ is orthogonal to both $\vec{u}$ and $\vec{v}$.

Activity 6

Activity 6

Suppose $\vec{u}, \vec{v}, \vec{w}$ are non-zero vectors in $\mathbb{R}^3$. Show that $\vec{u}, \vec{v}, \vec{w}$ are linearly independent if and only if $(\vec{u} \times \vec{v}) \cdot \vec{w} \neq 0$.

Let's use the cross product to concretely find the equation of the plane planned by two vectors in $\mathbb{R}^3$. Suppose $\vec{u} = \begin{bmatrix} 5 \\ 2 \\ 1 \end{bmatrix}$ and $\vec{v} = \begin{bmatrix} -2 \\ 3 \\ 0 \end{bmatrix}$.
The cross product of $\vec{u}$ and $\vec{v}$ is given by

$$\vec{u} \times \vec{v} = \begin{bmatrix} 2 \cdot 0 - 1 \cdot 3 \\ 1 \cdot (-2) - 5 \cdot 0 \\ 5 \cdot 3 - 2 \cdot (-2) \end{bmatrix} = \begin{bmatrix} -3 \\ -2 \\ 19 \end{bmatrix}$$

The equation of the plane spanned by $\vec{u}$ and $\vec{v}$ is then given by

$$-3x - 2y + 19z = 0$$

The vector that the cross product returns is sometimes called the **normal vector** of the plane. Normal is another

term for orthogonal or perpendicular. For the plane $-3x-2y+19z=0$, the normal vector is $\begin{bmatrix}-3\\-2\\19\end{bmatrix}$, as that vector is orthogonal to the two vectors $\vec{u}$ and $\vec{v}$ that span the plane. When we're looking at the standard form of the equation of a plane in $\mathbb{R}^3$, the normal vector is just the coefficients of x, y, and z in the equation $ax+by+cz=0$. There are infinitely many normal vectors for a given plane, since we can multiply any normal vector by a scalar and still get a normal vector. For example, $\begin{bmatrix}-3\\-2\\19\end{bmatrix}$ is a normal vector for the plane $-3x-2y+19z=0$, and so is $\begin{bmatrix}-6\\-4\\38\end{bmatrix}$ and $\begin{bmatrix}1\\2/3\\-19/3\end{bmatrix}$. Equivalently, $-6x-4y+38z=0$ and $x+\frac{2}{3}y-\frac{19}{3}z=0$ are ways to write the same plane we've been looking at.

Activity 7

Activity 7

1. Find the equation, in standard form, of the plane spanned by $\begin{bmatrix}3\\2\\0\end{bmatrix}$ and $\begin{bmatrix}-1\\1\\0\end{bmatrix}$. Why did you not need to compute the cross product?
2. Find the equation, in standard form, of the plane spanned by $\begin{bmatrix}3\\2\\4\end{bmatrix}$ and $\begin{bmatrix}-1\\1\\2\end{bmatrix}$.

Planes in Parametric Form. The cross product is a construct that only exists in 3-dimensions. Why is that? The cross product relies on the fact that the vectors $\vec{u}$ and $\vec{v}$ are linearly independent, meaning they span a plane, and that there is **only one direction** in $\mathbb{R}^3$ that is orthogonal to that plane. The cross product returns a vector in that direction. But, given two vectors in $\mathbb{R}^4$, for instance, there are infinitely many directions that are orthogonal to both of those two vectors, so it's hard to think of an operation that returns any one of them.

All of that is to say, in $\mathbb{R}^4$ and above, we can't express planes in standard form, the same way we can't express lines in $\mathbb{R}^3$ in standard form. Instead, we'll need to resort to their parametric form.

Definition: Parametric Form of a Plane

A plane in $\mathbb{R}^n$, $n\geq 3$, can be described as the set of all points $(x_1, x_2, \ldots, x_n)$ that can be written as

$$P=\vec{p}_0+s\vec{u}+t\vec{v},\quad s,t\in\mathbb{R}$$

where $\vec{p}_0, \vec{u}, \vec{v}\in\mathbb{R}^n$. $\vec{p}_0$ is a point on the plane, and $\vec{u}$ and $\vec{v}$ are two **linearly independent** direction vectors.

Again, the formal way of stating this definition is to treat the plane like a set of points that obeys an inclusion condition.

$$P=\left\{\vec{p}_0+s\vec{u}+t\vec{v}\mid s,t\in\mathbb{R}\right\}$$

This definition is very similar to the definition of the parametric form of a line in $\mathbb{R}^n$, it's just that instead of one

direction vector, we have two. For instance,

$$P = \begin{bmatrix} 3 \\ 8 \\ 1 \\ 2 \\ -7 \\ \pi \end{bmatrix} + s \begin{bmatrix} 1 \\ 0 \\ 2 \\ -1 \\ 0 \\ 0 \end{bmatrix} + t \begin{bmatrix} 5 \\ 2 \\ -1 \\ 3 \\ 1 \\ 0 \end{bmatrix}$$

is a plane in $\mathbb{R}^6$, and you should think of it as a 2-dimensional "slice" of 6-dimensional space.

Activity 8

Activity 8

What would happen if the two direction vectors that define a plane weren't linearly independent?

Activity 9

Activity 9

Prove that if you pick any two points on a plane in $\mathbb{R}^n$, the line connecting the two points is contained entirely on the plane.
Hint: Start by picking two points on the plane. Both of them must satisfy the parametric equation above, just with different values of s and t. Then, using what you've learned about parametric equations of lines, find the equation of the line connecting the two. What do you notice about that line?

Activity 10

Activity 10

Consider the points $(3, 4, 5)$, $(1, 9, -2)$, and $(2, 2, 0)$. Find the equation of the plane that passes through all three points, and express that plane in both parametric form and standard form, $ax + by + cz + d = 0$.

Solution

Start by picking one of the three points; we'll use $(3, 4, 5)$. Then, subtract this point from the other two points to find two direction vectors on the plane:

$$\vec{u} = \begin{bmatrix} 1 \\ 9 \\ -2 \end{bmatrix} - \begin{bmatrix} 3 \\ 4 \\ 5 \end{bmatrix} = \begin{bmatrix} -2 \\ 5 \\ -7 \end{bmatrix}$$

$$\vec{v} = \begin{bmatrix} 2 \\ 2 \\ 0 \end{bmatrix} - \begin{bmatrix} 3 \\ 4 \\ 5 \end{bmatrix} = \begin{bmatrix} -1 \\ -2 \\ -5 \end{bmatrix}$$

So, the plane in parametric form is

$$P = \begin{bmatrix} 3 \\ 4 \\ 5 \end{bmatrix} + s \begin{bmatrix} -2 \\ 5 \\ -7 \end{bmatrix} + t \begin{bmatrix} -1 \\ -2 \\ -5 \end{bmatrix}, \quad s, t \in \mathbb{R}$$

To find the standard form, compute the cross product of the two direction vectors:

$$\vec{u} \times \vec{v} = \begin{bmatrix} 5(-5) - (-7)(-2) \\ (-7)(-1) - (-2)(-5) \\ (-2)(-2) - 5(-1) \end{bmatrix} = \begin{bmatrix} -39 \\ -3 \\ 9 \end{bmatrix}$$

This gives a vector orthogonal to the plane, so the plane has the form

$$-39x - 3y + 9z + d = 0$$

Plug in $(3, 4, 5)$ to solve for d:

$$-39(3) - 3(4) + 9(5) + d = 0$$

$$-117 - 12 + 45 + d = 0$$

$$d = 84$$

So, one standard-form equation is

$$-39x - 3y + 9z + 84 = 0$$

Equivalently, dividing by -3 gives

$$\boxed{13x + y - 3z - 28 = 0}$$

Hyperplanes

So far, we've learned how to think of lines and planes in arbitrarily high dimensions. We can't visualize a plane in $\mathbb{R}^{76}$, but we have some intuition that it's a 2-dimensional "slice" of 76-dimensional space.

On the topic of slices:

- A line is a **1**-dimensional "slice" of 2-dimensional space.
- A plane is a **2**-dimensional "slice" of 3-dimensional space.

In general, a **hyperplane** is an $(n-1)$-dimensional "slice" of n-dimensional space.

Definition: Hyperplane

A **hyperplane** in $\mathbb{R}^n$ is the set of all points $(x_1, x_2, \ldots, x_n)$ that satisfy the equation

$$a_1x_1 + a_2x_2 + \cdots + a_nx_n + b = \vec{a} \cdot \vec{x} + b = 0$$

where $\vec{a}, \vec{x} \in \mathbb{R}^n$ and $b \in \mathbb{R}$.

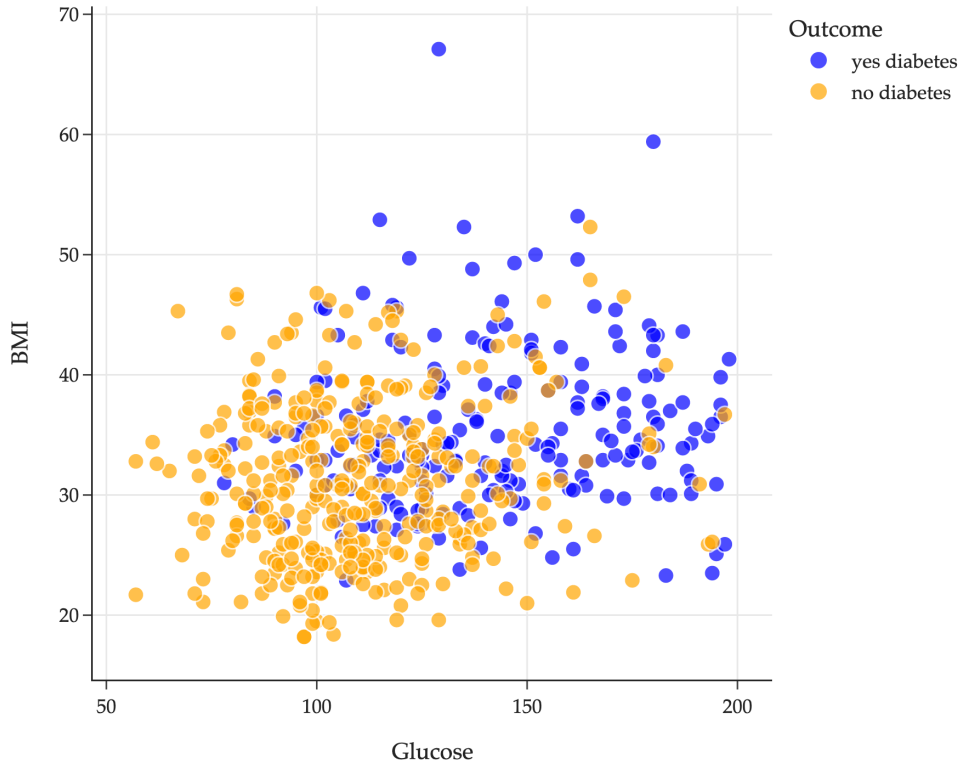
The most common way of representing a hyperplane is the form $\vec{a} \cdot \vec{x} + b = 0$.

- **Example**: $2x_1 + 3x_2 - 5 = 0$ is a hyperplane in $\mathbb{R}^2$, defined by the vector $\vec{a} = \begin{bmatrix} 2 \\ 3 \end{bmatrix}$ and $b = -5$. This is just a line in $\mathbb{R}^2$. (If it helps to see that this is a line, relabel x_1 and x_2 as x and y.)
- **Example**: $x_1 + x_2 + x_3 = 0$ is a hyperplane in $\mathbb{R}^3$, defined by the vector $\vec{a} = \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix}$ and $b = 0$. This is just a plane in $\mathbb{R}^3$.

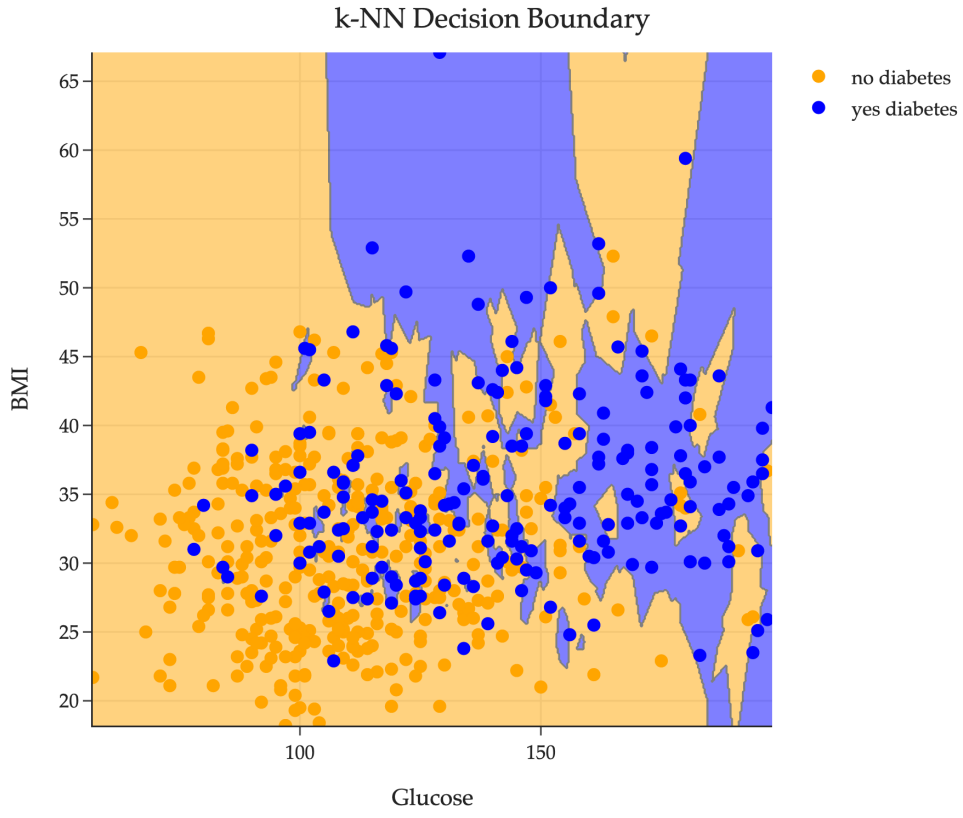
Hyperplanes are **hugely** important in machine learning, particularly in the context of **classification**. You should think of a hyperplane in $\mathbb{R}^n$ as a boundary that divides all of $\mathbb{R}^n$ into two halves: everything is either above it or below it.

For example, the hyperplane $\begin{bmatrix} -3 \\ -2 \\ 19 \end{bmatrix} \cdot \vec{x} = 0$ is shown below. Any point in $\mathbb{R}^3$ is either above it, meaning $\begin{bmatrix} -3 \\ -2 \\ 19 \end{bmatrix} \cdot \vec{x} > 0$, or below it, meaning $\begin{bmatrix} -3 \\ -2 \\ 19 \end{bmatrix} \cdot \vec{x} < 0$. (Yes, this hyperplane is just the plane $-3x - 2y + 19z = 0$ from earlier!)

Another more concrete example of a hyperplane comes from looking at the diabetes classification problem first introduced in Homework 3. There, we explored a dataset of several patients, each of which had two features – a glucose level and a body mass index (BMI) – along with a binary label indicating whether they have diabetes or not.

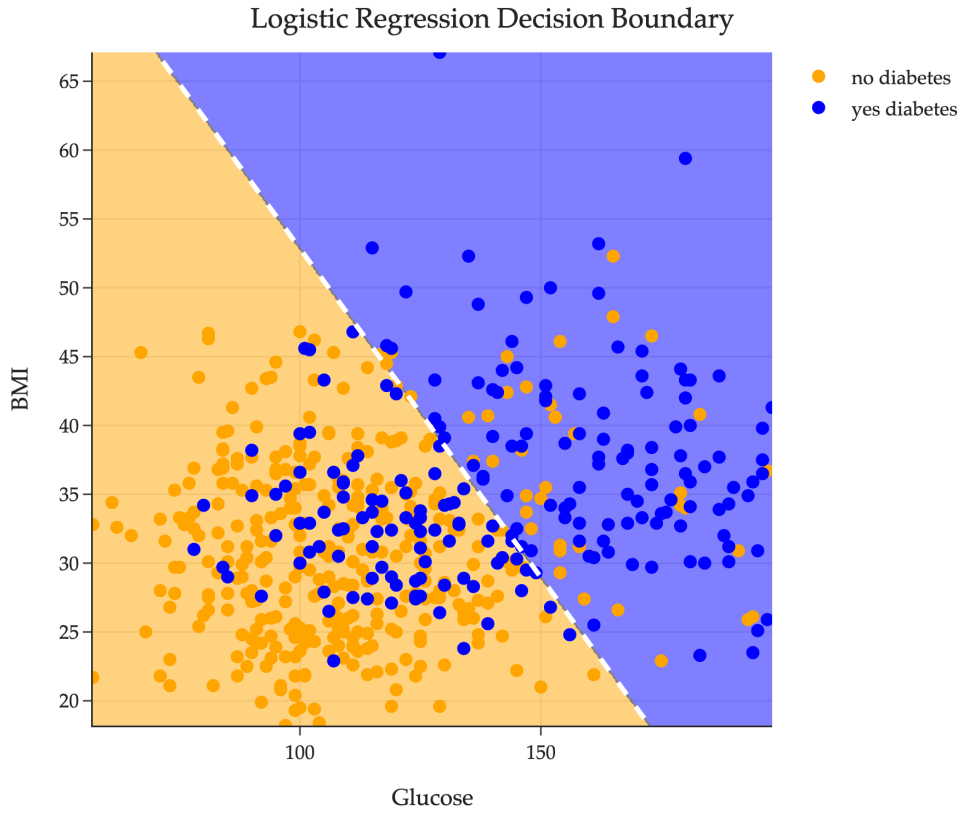


In a homework assignment, we introduce the k-nearest neighbors (k-NN) classifier. You might recall that the decision boundary of a k-NN classifier looks like a bunch of irregularly shaped blobs in the feature space ($\mathbb{R}^2$ here).



Another common family of classifiers is **linear classifiers**, where the decision boundary is a **hyperplane**. One

such linear classifier is the **logistic regression** classifier. On this dataset, its decision boundary is plotted below.



Here the decision boundary looks like a **line** because the data is only 2-dimensional, but in general (with more than two features) a linear classifier's decision boundary is a hyperplane in $\mathbb{R}^n$. The $\vec{w}$ in the decision boundary equation $\vec{w} \cdot \vec{x} + b = 0$ comes from minimizing empirical risk, for some model and loss function!

We can even peek at the decision boundary:

```
model = LogisticRegression()
model.fit(X_train, y_train)
```

```
LogisticRegression()
```

In a Jupyter environment, please rerun this cell to show the HTML representation or trust the notebook. On GitHub, the HTML representation is unable to render, please try loading this page with nbviewer.org.

[x] LogisticRegression?Documentation for LogisticRegressioniFitted

```
LogisticRegression()
```

```
model.coef_
```

```
array([[0.04, 0.08]])
```

```
model.intercept_
```

```
array([ -7.85])
```

This is telling us that the decision boundary is of the form

$$0.04 \cdot \text{Glucose}_i + 0.08 \cdot \text{BMI}_i - 7.85 = 0$$

or

$$\underbrace{\begin{bmatrix} 0.04 \\ 0.08 \end{bmatrix}}_{\vec{w}^*} \cdot \underbrace{\begin{bmatrix} \text{Glucose}_i \\ \text{BMI}_i \end{bmatrix}}_{\vec{x}_i} - 7.85 = 0$$

If this classifier used more features, then the decision boundary would involve more terms. Either way, it would be a hyperplane in $\mathbb{R}^d$, where d is the number of features used. Here, $d = 2$, so the decision boundary is a $(d-1)$-dimensional hyperplane in $\mathbb{R}^2$, i.e. a line in $\mathbb{R}^2$.

The specifics of logistic regression and how it works are beyond the scope of our course, and certainly not relevant to this section of the notes. I've provided this example here just to give you context for where hyperplanes come up in machine learning.

Matrices

5.1. Matrix Operations

Introduction

In Chapter 4, we focused on understanding the subspace of $\mathbb{R}^n$ **spanned** by a set of vectors. I've been alluding to the fact that to solve the regression problem in higher dimensions, we'd need to be able to **project** a vector onto the span of a set of vectors. (To be clear, we haven't done this yet.) Matrices will allow us to achieve this goal elegantly. It will take a few sections to get there:

- In Chapter 5.1, we'll define matrices and learn to perform basic operations on them.
- In Chapter 5.3, we'll define the column space, row space, null space, and rank of a matrix, while introducing the CR decomposition along the way.
- In Chapter 6.1, we'll understand matrix-vector multiplication as a transformation of the vector, and in Chapter 6.2 we'll learn how to invert a matrix and compute its determinant.
- In Chapter 6.3, we'll learn to project a vector onto the span of the columns of a matrix, closing the loop with Chapter 3.4.

Just remember that all of this is *for* machine learning.

For now, let's consider the following three vectors in $\mathbb{R}^4$:

$$\vec{v}_1 = \begin{bmatrix} 3 \\ 2 \\ 0 \\ 2 \end{bmatrix}, \quad \vec{v}_2 = \begin{bmatrix} 1 \\ 1 \\ -1 \\ -2 \end{bmatrix}, \quad \vec{v}_3 = \begin{bmatrix} 4 \\ 9 \\ 0 \\ 0 \end{bmatrix}$$

If we stack these vectors horizontally, we produce a **matrix**:

$$\begin{bmatrix} | & | & | \\ \vec{v}_1 & \vec{v}_2 & \vec{v}_3 \\ | & | & | \end{bmatrix} = \begin{bmatrix} 3 & 1 & 4 \\ 2 & 1 & 9 \\ 0 & -1 & 0 \\ 2 & -2 & 0 \end{bmatrix}$$

Definition: Matrix

A **matrix** is a rectangular array of numbers, organized into rows and columns.
In this class, we'll typically use uppercase letters to denote matrices. For example:

$$A = \begin{bmatrix} 3 & 1 & 4 \\ 2 & 1 & 9 \\ 0 & -1 & 0 \\ 2 & -2 & 0 \end{bmatrix}$$

A is a matrix with 4 rows and 3 columns, so we might say that A is a 4×3 matrix, or that $A \in \mathbb{R}^{4\times 3}$.

It's common to use the notation A_{ij} to denote the entry in the i-th row and j-th column of A, e.g. $A_{23} = 9$.

In `numpy`, you can access the entry in the 2nd row and 3rd column of the 2D matrix `A` using `A[1, 2]`, once we account for the fact that `numpy` uses 0-based indexing.

Activity 1

Activity 1

In the cell above, try and find the bottom-right entry of `A`, first using positive indexing and then using negative indexing.

I've introduced matrices in a very particular way, because I'd like for you to think of them as a collection of vectors, sometimes called **column vectors**. We'll come back to this in just a moment. You can also think of a matrix as a collection of row vectors; the example A defined above consists of four row vectors in $\mathbb{R}^3$ stacked horizontally.

When we introduced vectors, n was typically the placeholder we'd use for the number of components of a vector. With matrices, I like using n for the number of rows, and d for the number of columns. This means that, in general, a matrix is of shape $n \times d$ and is a member of the set $\mathbb{R}^{n \times d}$. **This makes clear that when we've collected data, we store each of our n observations in a row of our matrix.** Just be aware that using $n \times d$ to denote the shape of a matrix is a little unorthodox; most other textbooks will use $m \times n$ to denote the shape of a matrix. Of course, this is all arbitrary.

Suppose A has n rows and d columns, i.e. $A \in \mathbb{R}^{n \times d}$.

- If $n > d$, we say that A is **tall**.
- If $n = d$, we say that A is **square**.
- If $n < d$, we say that A is **wide**.

$$\underbrace{\begin{bmatrix} 5 & 3 \\ 2 & 1 \\ -1 & \frac{1}{3} \\ 3 & 6 \\ 0 & 1 \end{bmatrix}}_{\text{tall}} \qquad \underbrace{\begin{bmatrix} 3 & 0 & 4 \\ 4 & -\pi & 0 \\ \frac{1}{2} & 0 & 1 \end{bmatrix}}_{\text{square}} \qquad \underbrace{\begin{bmatrix} 1 & 0 & 3 & 2 & -1 \\ \frac{1}{9} & 3 & 0 & 0 & 2 \\ 9 & 0 & 0 & 6 & -3 \end{bmatrix}}_{\text{wide}}$$

As we'll see in the coming sections, square matrices are the most flexible – there are several properties and operations that are only defined for them. Unfortunately, not every matrix is square. Remember, matrices are used for storing data, and the number of observations, n and number of features, d, don't need to be the same. (In practice, we'll often have very tall matrices, i.e. $n \gg d$, but this is not always the case.)

Addition and Scalar Multiplication

Like vectors, matrices support addition and scalar multiplication out-of-the-box, and both behave as you'd expect.

Definition: Addition

Suppose A and B are matrices with the same shape, i.e. $A, B \in \mathbb{R}^{n \times d}$. Then, the **sum** of A and B is the matrix C with entries $C_{ij} = A_{ij} + B_{ij}$ for all i, j.
That is, the sum is performed element-wise. The sum is also commutative:

$$A + B = B + A$$

Definition: Scalar Multiplication

Suppose A is a matrix and c is a scalar. Then, the **scalar multiple** of A by c is the matrix B with entries $B_{ij} = cA_{ij}$ for all i, j.

Let's see an example. Consider the matrices A and B (where A is the same as in the previous example):

$$A = \begin{bmatrix} 3 & 1 & 4 \\ 2 & 1 & 9 \\ 0 & -1 & 0 \\ 2 & -2 & 0 \end{bmatrix}, \quad B = \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \\ 10 & 11 & 12 \end{bmatrix}$$

Then, the operation $3A - B$ is well-defined – it is a linear combination of A and B. Its result is the matrix

$$3A - B = 3\begin{bmatrix} 3 & 1 & 4 \\ 2 & 1 & 9 \\ 0 & -1 & 0 \\ 2 & -2 & 0 \end{bmatrix} - \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \\ 10 & 11 & 12 \end{bmatrix} = \begin{bmatrix} 8 & 1 & 9 \\ 2 & -2 & 21 \\ -7 & \boxed{-11} & -9 \\ -4 & -17 & -12 \end{bmatrix}$$

I've colored the entry at position $(3, 2)$ blue to help you trace the computation.

$$3 \cdot (-1) - 8 = -11$$

Matrix-Vector Multiplication

Great – we know how to add two matrices, and how to multiply a matrix by a scalar. The natural next step is to figure out how – and why – to multiply two matrices together.

First, a definition.

Golden Rule of Matrix Multiplication

Suppose A and B are two matrices. In order for the product AB to be valid, **the number of columns in A must equal the number of rows in B**.

$$\#\text{ columns in } A = \#\text{ rows in } B$$

In other words, **the inner dimensions of A and B must match**.

Let's use the Golden Rule. First – as the title of this subsection suggests – we'll start by computing the product of a matrix and a vector.

Let's suppose A is the same 4×3 matrix we've become familiar with:

$$A = \begin{bmatrix} 3 & 1 & 4 \\ 2 & 1 & 9 \\ 0 & -1 & 0 \\ 2 & -2 & 0 \end{bmatrix}$$

And let's suppose $\vec{x}$ is some vector. Note that we can think of an n-dimensional vector as a matrix with n rows and 1 column. In order for the product $A\vec{x}$ to be valid, $\vec{x}$ must have 3 elements in it, by the Golden Rule above. To make the example concrete, let's suppose:

$$\vec{x} = \begin{bmatrix} 1 \\ 0 \\ 3 \end{bmatrix}$$

How do we multiply $A\vec{x}$? The following key definition will help us.

Definition: Matrix-Vector Multiplication

If A is a $n \times d$ matrix and $\vec{x} \in \mathbb{R}^d$ is a vector, then the product $A\vec{x}$ is a vector in $\mathbb{R}^n$ that contains **the dot product of each *row* of A with $\vec{x}$.**

Let me say a little more about the dimensions of the output. Below, I've written the dimensions of A (4×3) and $\vec{x}$ (3×1) next to each other. By the Golden Rule, the **inner dimensions, both of which are bolded**, must be equal in order for the multiplication to be valid. The dimensions of the output will be the result of looking at the outer dimensions, which here are 4×1.

$$\underbrace{A}_{\boxed{4}\times 3} \quad \underbrace{\vec{x}}_{3\times\boxed{1}} = \underbrace{\text{output}}_{4\times 1}$$

So, the result of multiplying $A\vec{x}$ will be 4×1 matrix, or in other words, a vector in $\mathbb{R}^4$. Indeed, the result of multiply a matrix by a vector always results in another vector, and this act of multiplying a matrix by a vector is often thought of as **transforming** the vector from $\mathbb{R}^d$ to $\mathbb{R}^n$.

So, how do we find those 4 components? As mentioned earlier, we compute each component by taking the dot product of a row in A with $\vec{x}$.

$$A = \begin{bmatrix} 3 & 1 & 4 \\ 2 & 1 & 9 \\ 0 & -1 & 0 \\ 2 & -2 & 0 \end{bmatrix}, \quad \vec{x} = \begin{bmatrix} 1 \\ 0 \\ 3 \end{bmatrix}$$

Let's start with the top row of A, $\begin{bmatrix} 3 \\ 1 \\ 4 \end{bmatrix}$. The dot product of two vectors is only defined if they have equal lengths.

This is why we've instituted the Golden Rule! The Golden Rule tells us we can only multiply A and $\vec{x}$ if the number of columns in A is the same as the number of components in $\vec{x}$, which is true here (both of those numbers are 3).

Then, the dot product of the first row of A with $\vec{x}$ is:

$$\begin{bmatrix} 3 \\ 1 \\ 4 \end{bmatrix} \cdot \begin{bmatrix} 1 \\ 0 \\ 3 \end{bmatrix} = 3 \cdot 1 + 1 \cdot 0 + 4 \cdot 3 = 15$$

Nice! We're a quarter of the way there. Now, we just need to compute the remaining three dot products:

- The dot product of the second row of A with $\vec{x}$ is $\begin{bmatrix}2\\1\\9\end{bmatrix} \cdot \begin{bmatrix}1\\0\\3\end{bmatrix} = 29$.
- The dot product of the third row of A with $\vec{x}$ is $\begin{bmatrix}0\\-1\\0\end{bmatrix} \cdot \begin{bmatrix}1\\0\\3\end{bmatrix} = 0$.
- And finally, the dot product of the fourth row of A with $\vec{x}$ is $\begin{bmatrix}2\\-2\\0\end{bmatrix} \cdot \begin{bmatrix}1\\0\\3\end{bmatrix} = 2$.

The result of our matrix-vector multiplication, then, is the result of stacking all 4 dot products together into the vector $\begin{bmatrix}15\\29\\0\\2\end{bmatrix}$. To summarize:

$$A\vec{x} = \begin{bmatrix}3 & 1 & 4\\2 & 1 & 9\\0 & -1 & 0\\2 & -2 & 0\end{bmatrix}\begin{bmatrix}1\\0\\3\end{bmatrix} = \begin{bmatrix}15\\29\\0\\2\end{bmatrix}$$

Refer to this example if you're ever confused on how to multiply a matrix by a vector.

We can't visualize the output vector as it's in 4 dimensions, but we *can* look at the 4 row vectors of A and the vector $\vec{x}$ in 3D space.

Above, you'll notice that $\vec{x}$ and the third row vector, $\begin{bmatrix}0\\-1\\0\end{bmatrix}$, are orthogonal (rotate the plot so that you can see this), so the third component in

$$A\vec{x} = \begin{bmatrix}15\\29\\0\\2\end{bmatrix}$$

is 0.

Activity 2

Activity 2

We'll try not to bore you with mundane calculations in the future, but it's important to perform matrix-vector multiplication by hand a few times to understand how it works.
In each part, perform the matrix-vector multiplication by hand or state that it cannot be done.

1. $$\begin{bmatrix}7 & 8 & -1\\0 & 1 & 2\\9 & 0 & \frac{1}{2}\end{bmatrix}\begin{bmatrix}5\\4\\8\end{bmatrix}$$

2. $$\begin{bmatrix}7 & 8 & -1\\0 & 1 & 2\end{bmatrix}\begin{bmatrix}5\\4\end{bmatrix}$$

3.
$$\begin{bmatrix} 4 & 2 & 3 \\ 1 & 0 & 1 \\ 5 & 4 & 3 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 5 \\ 4 \\ 8 \end{bmatrix}$$

4.
$$\begin{bmatrix} 4 & 2 & 3 \\ 1 & 0 & 1 \\ 5 & 4 & 3 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 7 & 8 & -1 \\ 0 & 1 & 2 \\ 9 & 0 & \frac{1}{2} \end{bmatrix} \begin{bmatrix} 5 \\ 4 \\ 8 \end{bmatrix}$$

(While we haven't yet looked at how to compute the product of two matrices, you can still answer this just using what you know about matrix-vector multiplication.)

In the cell below, use `numpy` to verify your answers. You'll need to define the matrices and vectors as `numpy` arrays, and use the `@` operator to perform the matrix-vector multiplication.

The Linear Combination Interpretation

We've described matrix-vector multiplication as the result of taking the dot product of each row of A with $\vec{x}$, and indeed this is the easiest way to actually compute the output. But, there's another more important interpretation. In the above dot products, you may have noticed:

- Entries in the first column of A (3, 2, 0, and 2) were always multiplied by the first element of $\vec{x}$ (1).
- Entries in the second column of A (1, 1, -1, and -2) were always multiplied by the second element of $\vec{x}$ (0).
- Entries in the third column of A (4, 9, 0, and 0) were always multiplied by the third element of $\vec{x}$ (3).

In other words:

$$A\vec{x} = \begin{bmatrix} 3 & 1 & 4 \\ 2 & 1 & 9 \\ 0 & -1 & 0 \\ 2 & -2 & 0 \end{bmatrix} \begin{bmatrix} 1 \\ 0 \\ 3 \end{bmatrix} = \underbrace{1 \begin{bmatrix} 3 \\ 2 \\ 0 \\ 2 \end{bmatrix} + 0 \begin{bmatrix} 1 \\ 1 \\ -1 \\ -2 \end{bmatrix} + 3 \begin{bmatrix} 4 \\ 9 \\ 0 \\ 0 \end{bmatrix}}_{\text{linear combination of columns of A}} = \begin{bmatrix} 15 \\ 29 \\ 0 \\ 2 \end{bmatrix}$$

At the start of this section, we defined A by stacking the vectors $\vec{v}_1$, $\vec{v}_2$, and $\vec{v}_3$ side-by-side, and I told you to think of a matrix as a collection of column vectors. The above result is precisely why – it's because when we multiply A by $\vec{x}$, we're computing a linear combination of the columns of A, where the weights are the components of $\vec{x}$!

Since $A\vec{x}$ produces a linear combination of the columns of A, a natural question to ask at this point is whether the columns of A are all linearly independent. A only has 3 columns, each of which is in $\mathbb{R}^4$, so while they may or may not be linearly independent (are they?), we know they cannot span all of $\mathbb{R}^4$, as we'd need at least 4 vectors to reach every element in $\mathbb{R}^4$.

This is the type of thinking we'll return to in Chapter 5.3. This will lead us to define the **rank** of a matrix, perhaps the single most important number associated with a matrix.

The Two Pictures

Any time you see a matrix-vector product, like $A\vec{x}$, you should think of it not as a random operation, but as:

1. (**More Important**) A linear combination of the columns of A, where the weights are the components of $\vec{x}$.
$$A\vec{x} = x_1\,(\text{column 1 of } A) + x_2\,(\text{column 2 of } A) + \ldots + x_d\,(\text{column } d \text{ of } A)$$
2. A dot product of the rows of A with $\vec{x}$.
$$A\vec{x} = \begin{bmatrix} (\text{row 1 of } A)\cdot\vec{x} \\ (\text{row 2 of } A)\cdot\vec{x} \\ \vdots \\ (\text{row n of } A)\cdot\vec{x} \end{bmatrix}$$

Activity 3

Activity 3

Consider the matrix M defined below.

$$M = \begin{bmatrix} 2 & -1 & 3 & 0 & 4 \\ 1 & 5 & -2 & 1 & 0 \end{bmatrix}$$

In each of the following parts, write out $\vec{u}$ concretely, compute $M\vec{u}$, and explain the result in English.

1. A vector whose second component is 1, and whose other components are 0.
2. A vector containing all 1s.
3. A vector containing all $\frac{1}{5}$s.
4. A vector whose components sum to 1, whose first component is $\frac{3}{5}$, and whose other components are all equal to one another.

Matrix Multiplication

Matrix-matrix multiplication – or just "matrix multiplication" – is a generalization of matrix-vector multiplication. Let's present matrix multiplication in its most general terms.

Definition.

Definition: Matrix Multiplication

Suppose:

- A is a $n \times d$ matrix.
- B is a $d \times p$ matrix.

Then, AB is a $n \times p$ matrix such that

- the element in row i and column j of AB is
- the **dot product of row i of A and column j of B**, for $i = 1, 2, \ldots, n$ and $j = 1, 2, \ldots, p$.

In other words,

$$(AB)_{ij} = (\text{row } i \text{ of } A) \cdot (\text{column } j \text{ of } B) = \sum_{k=1}^{d} A_{ik} B_{kj}$$

Note that if $p = 1$, this reduces to the matrix-vector multiplication case from before. In that case, the only possible value of j is 1, since the output only has 1 column, and the element in row i of the output vector is the dot product of row i in A and the vector B (which we earlier referred to as $\vec{x}$ in the less general case).

For a concrete example, suppose A and B are defined below:

$$A = \begin{bmatrix} 3 & 1 & 4 \\ 2 & 1 & 9 \\ 0 & -1 & 0 \\ 2 & -2 & 0 \end{bmatrix} \quad B = \begin{bmatrix} 1 & 2 \\ 0 & 7 \\ 3 & 2 \end{bmatrix}$$

The number of columns of A must equal the number of rows of B in order for the product AB to be defined, as the Golden Rule tells us. That is fortunately the case here. Since A has shape $\boxed{4} \times 3$ and B has shape $3 \times \boxed{2}$, the output matrix will have shape 4×2. Each of those $4 \cdot 2 = 8$ elements will be the dot product of a row in A with a column in B.

Here is the product of A and B:

$$AB = \begin{bmatrix} 3 & 1 & 4 \\ 2 & 1 & 9 \\ 0 & -1 & 0 \\ 2 & -2 & 0 \end{bmatrix} \begin{bmatrix} 1 & 2 \\ 0 & 7 \\ 3 & 2 \end{bmatrix} = \begin{bmatrix} 15 & 21 \\ 29 & 29 \\ 0 & -7 \\ 2 & -10 \end{bmatrix}$$

Let's see if we can audit where these numbers came from. Let's consider $(AB)_{32}$, which is the element in row 3 and column 2 of the output. It should have come from the dot product of row 3 of A and column 2 of B.

$$AB = \begin{bmatrix} 3 & 1 & 4 \\ 2 & 1 & 9 \\ \boxed{0} & \boxed{-1} & \boxed{0} \\ 2 & -2 & 0 \end{bmatrix} \begin{bmatrix} 1 & \boxed{2} \\ 0 & \boxed{7} \\ 3 & \boxed{2} \end{bmatrix} = \begin{bmatrix} 15 & 21 \\ 29 & 29 \\ 0 & \boxed{-7} \\ 2 & -10 \end{bmatrix}$$

And indeed, -7 is the dot product of $\begin{bmatrix} 0 \\ -1 \\ 0 \end{bmatrix}$ and $\begin{bmatrix} 2 \\ 7 \\ 2 \end{bmatrix}$.

You should notice that many of the numbers in the output AB look familiar. That's because we used the same A as we did earlier in the section, and the first column of B is just $\vec{x}$ from the matrix-vector example. So, the first column in AB is the same as the vector $A\vec{x} = \begin{bmatrix} 15 \\ 29 \\ 0 \\ 2 \end{bmatrix}$ as we computed earlier. The difference now is that the

output AB isn't just a single vector, but is a matrix with 2 columns. The second column, $\begin{bmatrix} 21 \\ 29 \\ -7 \\ -10 \end{bmatrix}$, comes from multiplying A by the second column in B, $\begin{bmatrix} 2 \\ 7 \\ 2 \end{bmatrix}$.

Note that as we add columns to B, we'd add columns to the output. If B had 10 columns, then AB would have 10 columns, too, without A needing to change. As long as the Golden Rule – that the number of columns in A equals the number of rows in B – holds, the product AB can be computed, and it has shape (number of rows in A)× (number of columns in B).

Properties.

Properties of Matrix Multiplication

- Matrix multiplication is **associative**. That is, $(AB)C = A(BC)$ for any matrices A, B, and C that are of the appropriate shapes.
- Matrix multiplication is **distributive**. That is, $A(B + C) = AB + AC$ for any matrices A, B, and C that are of the appropriate shapes.
- Matrix multiplication is **not commutative**! That is, in general, $AB \neq BA$ for any matrices A and B that are of the appropriate shapes.

The first two properties – associativity and distributivity – match standard arithmetic properties that we've become accustomed to. The associative property allows you to, for example, compute $AB\vec{x}$ only by using matrix-vector multiplications, since you can first multiply $B\vec{x}$, which results in a vector, and then multiply A by that vector. (I had you do this in Activity 2 earlier in this section – I hope you did it!)

The fact that matrix multiplication is **not** commutative may come as a surprise, as every other form of multiplication you've learned about up until this point has been commutative (including the dot product).

In general, $AB \neq BA$.

In fact, if AB exists, BA may or may not! If A is $n \times d$ and B is $d \times p$, then BA only exists if $n = p$. But even then, $AB \neq BA$ in general.

For example, if A is 2×3 and B is 3×2, then AB is 2×2 and BA is 3×3; here, both products exist, but they cannot be equal since they have different shapes.

Even if A and B are both square matrices with the same shape, $AB \neq BA$ in general. For illustration, consider:

$$A = \begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix}, \quad B = \begin{bmatrix} 5 & 6 \\ 7 & 8 \end{bmatrix}$$

Then,

$$AB = \begin{bmatrix} 19 & 22 \\ 43 & 50 \end{bmatrix} \neq BA = \begin{bmatrix} 23 & 34 \\ 31 & 46 \end{bmatrix}$$

Activity 4

Activity 4

Activity 4.1

Let $P = \begin{bmatrix} 0 & 0 & 1 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \end{bmatrix}$, $S = \begin{bmatrix} 4 & 0 & 0 \\ 0 & \frac{1}{2} & 0 \\ 0 & 0 & 3 \end{bmatrix}$, and $\vec{x} = \begin{bmatrix} 4 \\ 6 \\ 12 \end{bmatrix}$.

1. Evaluate $P\vec{x}$ and $S\vec{x}$. Then, explain in words what multiplying P and S by $\vec{x}$ does to $\vec{x}$.
2. Evaluate $PS\vec{x}$ and $SP\vec{x}$. The results should be different, as we'd expect, since matrix multiplication is not commutative in general. Explain the difference intuitively, given the "operations" P and S perform on $\vec{x}$.

P is called a permutation matrix, and S is called a diagonal matrix.

Activity 4.2

The famous Fibonacci sequence of integers, $F_0, F_1, F_2, \ldots$, is defined as follows:

$$F_0 = 0, \quad F_1 = 1, \quad F_n = F_{n-1} + F_{n-2} \text{ for } n \geq 2$$

The first few terms in the sequence are $0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144, \ldots$.

It turns out you can compute the nth term in the sequence using matrix multiplication.

1. Find a 2×2 matrix A such that $A \begin{bmatrix} F_{n-1} \\ F_{n-2} \end{bmatrix} = \begin{bmatrix} F_n \\ F_{n-1} \end{bmatrix}$. The answer uses relatively small numbers.
2. Compute $A \begin{bmatrix} 1 \\ 0 \end{bmatrix}$, i.e. the product of A and the vector $\begin{bmatrix} 1 \\ 0 \end{bmatrix}$.
3. Since A is square, we can multiply it by itself. Compute $A^2 \begin{bmatrix} 1 \\ 0 \end{bmatrix}$ and $A^3 \begin{bmatrix} 1 \\ 0 \end{bmatrix}$.

If you continue this process, you'll find that $A^n \begin{bmatrix} 1 \\ 0 \end{bmatrix}$ is a vector containing the nth and $(n-1)$th terms in the Fibonacci sequence!

Activity 4.3

Using the same matrices P and S from Activity 4.1, compute $(P - S)\vec{x}$ and $P\vec{x} - S\vec{x}$. Are both the results the same? If so, what property of matrix multiplication guarantees this?

Is the result of $(P - S)\vec{x}$ interpretable, in the same way that the results of $P\vec{x}$ and $S\vec{x}$ were in Activity 4.1?

Computation. I've shown you the naïve – and by far most common – algorithm for matrix multiplication. If A and B are both square $n \times n$ matrices, then the runtime of the naïve algorithm is $O(n^3)$.

However, there exist more efficient algorithms for matrix multiplication. Strassen's algorithm is one such example; it describes how to multiply two square $n \times n$ matrices in $O(n^{2.807})$ time. The study of efficient algorithms for matrix multiplication is an active area of research; if you're interested in learning more, look here.

Matrix multiplication, I might argue, is one of the reasons NVIDIA is the most valuable company in the world. Modern machine learning is built on matrix multiplication, and GPUs are optimized for it. Why? This comment from Reddit does a good job of explaining:

> Imagine you have 1 million math assignments to do, they are very simple assignments, but there are a lot that need to be done, they are not dependent on each other so they can be done on any order.
>
> You have two options, distribute them to 10 thousand people to do it in parallel or give them to 10

math experts. The experts are very fast, but hey, there are only 10 of them, the 10 thousand are more suitable for the task because they have the "brute force" for this.

GPUs have thousands of cores, CPUs have tens.

On that note, the `@` operator in `numpy` is used for matrix multiplication; it is a shorthand for `np.matmul`. You can also use it to multiply a matrix by a vector.

The * Operator is Not Matrix Multiplication!

The `*` operator in `numpy` is used for element-wise multiplication, not matrix multiplication. If you want to multiply two matrices, you must use the `@` operator.
In the cell above, try and compute the product `A * B` using the `*` operator. What happens?

Chapter 5.2 has two goals: first, it'll introduce the **tranpose** of a matrix, an important operation we've yet to cover. Then, it will briefly introduce several "special" matrices that we'll use throughout the rest of the course.

5.2. Transpose and Special Matrices

There's an important operation on matrices that we haven't discussed yet. Let me introduce it, and then walk through a few types of matrices that are important enough to be given names.

Transpose

Definition: Transpose

The **transpose** of a matrix A is the matrix A^T with entries $A^T_{ij} = A_{ji}$ for all i, j.
Intuitively, the transpose results from replacing the rows of A with the columns of A and vice-versa.

To illustrate, let's start with our familiar matrix A:

$$A = \begin{bmatrix} 3 & 1 & 4 \\ 2 & 1 & 9 \\ 0 & -1 & 0 \\ 2 & -2 & 0 \end{bmatrix}$$

The transpose of A is:

$$A^T = \begin{bmatrix} 3 & 2 & 0 & 2 \\ 1 & 1 & -1 & -2 \\ 4 & 9 & 0 & 0 \end{bmatrix}$$

Note that $A \in \mathbb{R}^{4\times 3}$ and $A^T \in \mathbb{R}^{3\times 4}$.

Why would we ever need to do this? To illustrate, suppose $\vec{u} = \begin{bmatrix} u_1 \\ u_2 \\ u_3 \\ u_4 \end{bmatrix}$, and that we'd like to compute the product $A^T\vec{u}$. (Note that $\vec{u}$ must be in $\mathbb{R}^4$ in order for $A^T\vec{u}$ to be defined, unlike $\vec{x} \in \mathbb{R}^3$ in the product $A\vec{x}$). Then:

$$A^T\vec{u} = \begin{bmatrix} 3 & 2 & 0 & 2 \\ 1 & 1 & -1 & -2 \\ 4 & 9 & 0 & 0 \end{bmatrix} \begin{bmatrix} u_1 \\ u_2 \\ u_3 \\ u_4 \end{bmatrix}$$
$$= u_1 \begin{bmatrix} 3 \\ 1 \\ 4 \end{bmatrix} + u_2 \begin{bmatrix} 2 \\ 1 \\ 9 \end{bmatrix} + u_3 \begin{bmatrix} 0 \\ -1 \\ 0 \end{bmatrix} + u_4 \begin{bmatrix} 2 \\ -2 \\ 0 \end{bmatrix}$$

This is a linear combination of the **rows** of A, where the weights are the components of $\vec{u}$. Remember, the standard product $A\vec{x}$ is a linear combination of the columns of A, so the transpose helps us if we want to compute a linear combination of the rows of A. (Equivalently, it helps us if we want to compute the dot product of the **columns of** A with $\vec{u}$ – see the linear combination interpretation section of Chapter 5.1.)

The transpose also gives us another way of expressing the dot product of two vectors. If $\vec{u}$ and $\vec{v}$ are two vectors in $\mathbb{R}^n$, then $\vec{u}^T$ is a row vector with 1 row and n columns. Multiplying $\vec{u}^T$ by $\vec{v}$ results in a 1×1 matrix, which is just the scalar $\vec{u} \cdot \vec{v}$.

$$\vec{u}^T\vec{v} = \begin{bmatrix} u_1 & u_2 & \dots & u_n \end{bmatrix} \begin{bmatrix} v_1 \\ v_2 \\ \vdots \\ v_n \end{bmatrix} = u_1v_1 + u_2v_2 + \dots + u_nv_n = \vec{u} \cdot \vec{v} = \vec{v} \cdot \vec{u} = \vec{v}^T\vec{u}$$

Transpose, or Dot, but Not Both!

The following are all valid ways of computing the dot product of $\vec{u}$ and $\vec{v}$, assuming they have the same number of components:

$$\vec{u} \cdot \vec{v} = \vec{u}^T\vec{v} = \vec{v}^T\vec{u} = \vec{v} \cdot \vec{u}$$

However, $\vec{u}^T \cdot \vec{v}$ **is not defined**. The dot product is only defined for two vectors of the same dimensions, but $\vec{u}^T$ is a row vector with n columns, and $\vec{v}$ is a column vector with n rows.

The benefit of using the transpose to express the dot product is that it allows us to write the dot product of two vectors in terms of matrix multiplication, rather than being an entirely different type of operation. (In fact, as we've seen here, matrix multiplication is just a generalization of the dot product.)

There are other uses for the transpose, too, so it's a useful tool to have in your toolbox.

Properties.

Properties of the Transpose

- $(A^T)^T = A$, i.e. the transpose of the transpose of a matrix is the original matrix.
- $(A + B)^T = A^T + B^T$.
- $(cA)^T = cA^T$ for any scalar $c \in \mathbb{R}$.
- $(AB)^T = B^TA^T$.

The first three properties are relatively straightforward. The last property is a bit more subtle. Try and reason as to why it's true on your own, then peek into the box below to verify your reasoning and to see an example.

Why is $(AB)^T = B^TA^T$?

Let's start with just $(AB)^T$ and reason our way from there. Define $C = (AB)^T$. C is presumably the product of two matrices X and Y, we just don't know what X and Y are. By the definition of matrix multiplication, we know that C_{ij} is the dot product of the ith row of X and the jth column of Y. How can we express the product $C = XY$ in terms of A and B?
Let's work backwards. Since $C_{ij} = (AB)^T_{ij} = (AB)_{ji}$ by the definition of the transpose, we know that:

$$C_{ij} = (AB)_{ji} = \big(\text{row } j \text{ of } A\big) \cdot (\text{column } i \text{ of } B) = (\text{column } i \text{ of } B) \cdot \big(\text{row } j \text{ of } A\big)$$

This is a little backwards relative to the definition of matrix multiplication, which says that:

$$C_{ij} = (XY)_{ij} = (\text{row } i \text{ of } X) \cdot \big(\text{column } j \text{ of } Y\big)$$

In order for the two definitions of C_{ij} to be consistent, we must have:

$$(\text{column } i \text{ of } B) \cdot (\text{row } j \text{ of } A) = (\text{row } i \text{ of } X) \cdot (\text{column } j \text{ of } Y)$$

- Row i of X is the same as column i of B, if $X = B^T$.
- Column j of Y is the same as row j of A, if $Y = A^T$.

Putting this together, we have:

$$C = (AB)^T = B^T A^T$$

as we hoped!
To make things concrete, let's consider two new matrices A and B:

$$A = \begin{bmatrix} 0 & -1 \\ 4 & 2 \\ 3 & 9 \\ 0 & 1 \end{bmatrix} \quad B = \begin{bmatrix} 1 & 2 & 3 \\ -1 & -2 & 4 \end{bmatrix}$$

Then,

$$AB = \begin{bmatrix} 0 & -1 \\ 4 & 2 \\ 3 & 9 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 2 & 3 \\ -1 & -2 & 4 \end{bmatrix} = \begin{bmatrix} 1 & 2 & -4 \\ 2 & 4 & 20 \\ -6 & -12 & \boxed{45} \\ -1 & -2 & 4 \end{bmatrix}$$

And:

$$B^T A^T = \begin{bmatrix} 1 & -1 \\ 2 & -2 \\ 3 & 4 \end{bmatrix} \begin{bmatrix} 0 & 4 & 3 & 0 \\ -1 & 2 & 9 & 1 \end{bmatrix} = \begin{bmatrix} 1 & 2 & -6 & -1 \\ 2 & 4 & -12 & -2 \\ -4 & 20 & \boxed{45} & 4 \end{bmatrix}$$

$B^T A^T$ is the transpose of AB. Both AB and $B^T A^T$ have 12 elements, both computed using the same 12 dot products.

The fact that $(AB)^T = B^T A^T$ comes in handy when finding the norm of a matrix-vector product. If A is an $n \times d$ matrix and $\vec{x} \in \mathbb{R}^d$, then:

$$\|A\vec{x}\|^2 = (A\vec{x})^T (A\vec{x}) = \vec{x}^T A^T A \vec{x}$$

As we'll soon see, some matrices A have special properties that make this computation particularly easy.

In `numpy`, the `T` attribute is used to compute the transpose of a 2D array.

Activity 1

Activity 1

Activity 1.1
In the cell above:

1. Define `x` to be an array corresponding to the vector $\vec{x} = \begin{bmatrix} 1 \\ 0 \\ 3 \end{bmatrix}$.
2. Find the norm of the product $A\vec{x}$ using `np.linalg.norm`.
3. Find the norm of the product $A\vec{x}$ using the fact that $\|A\vec{x}\|^2 = \vec{x}^T A^T A\vec{x}$, and verify that you get the same answer.

Activity 1.2
Suppose $M \in \mathbb{R}^{n \times d}$ is a matrix, $\vec{v} \in \mathbb{R}^d$ is a vector, and $s \in \mathbb{R}$ is a scalar.
Determine whether each of the following quantities is a matrix, vector, scalar, or undefined. If the result is a matrix or vector, determine its dimensions.

1. $M\vec{v}$
2. $\vec{v}M$
3. $\vec{v}^2$
4. $M^T M$
5. MM^T
6. $\vec{v}^T M\vec{v}$
7. $(sM\vec{v}) \cdot (sM\vec{v})$
8. $(s\vec{v}^T M^T)^T$
9. $\vec{v}^T M^T M\vec{v}$
10. $\vec{v}\vec{v}^T + M^T M$
11. $\frac{M\vec{v}}{\|\vec{v}\|} + (\vec{v}^T M^T M\vec{v})M\vec{v}$

Activity 1.3
Let $A = \begin{bmatrix} 2 & 1 \\ 3 & 4 \\ -1 & 1 \end{bmatrix}$, $B = \begin{bmatrix} 1 & 0 & 2 \\ 2 & 1 & 3 \end{bmatrix}$, and $C = \begin{bmatrix} 1 & 0 & 2 & -1 \\ 0 & 1 & 1 & 1 \\ 1 & 1 & 0 & -1 \end{bmatrix}$.

1. Compute AB, then multiply the result by C.
2. Compute A, then multiply the result by BC. Do you get the same result as above? If so, what property of matrix multiplication guarantees this?
3. Determine a formula for $(ABC)^T$, and verify that your result works. (Hint: Start with the fact that $(AB)^T = B^T A^T$.)

The Identity Matrix

Now, I'll introduce several "special" types of matrices that will come in handy at various points throughout our journey.

Definition: Identity Matrix

The **identity matrix** is the square matrix I with ones on the diagonal and zeros everywhere else.

$$I = \begin{bmatrix} 1 & 0 & \dots & 0 \\ 0 & 1 & \dots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \dots & 1 \end{bmatrix}$$

Saying "the identity matrix" is a bit ambiguous, as there are infinitely many identity matrices – there's a 1×1 identity matrix, a 2×2 identity matrix, a 3×3 identity matrix, and so on. Often, the dimension of the identity matrix is implied by context, and if not, we might provide it as a subscript, e.g. I_n for the $n \times n$ identity matrix.

Why is the identity matrix defined this way? It's the matrix equivalent of the number 1 in scalar multiplication, also known as the multiplicative identity. If c is a scalar, then $c \cdot 1 = c$ and $1 \cdot c = c$. (0 is known as the additive identity in scalar multiplication.)

Similarly, if A is **square** $n \times n$ matrix and $\vec{x} \in \mathbb{R}^n$ is a vector, then the $n \times n$ identity matrix I is the unique matrix that satisfies:

- $I\vec{x} = \vec{x}$ for all $\vec{x} \in \mathbb{R}^n$.
- $IA = AI = A$ for all $A \in \mathbb{R}^{n \times n}$.

A good exercise is to verify that the identity matrix satisfies these properties.

Activity 2

Activity 2

Let $X = \begin{bmatrix} 1 & -2 \\ -1 & 3 \\ 2 & 0 \\ 0 & -1 \\ 3 & 2 \end{bmatrix}$.

1. Compute $X^T X$.
2. Then, compute the transpose of $X^T X$. What do you notice? ($X^T X$ is called a *symmetric* matrix, as we'll discuss below.)
3. Compute $X^T X + \frac{1}{2} I$.

Symmetric Matrices

Definition: Symmetric Matrix

A matrix A is **symmetric** if $A = A^T$. More formally, A is symmetric if $A_{ij} = A_{ji}$ for all i, j.
In order for $A = A^T$, A and A^T must have the same dimensions, so A must be square.

For example, A below is symmetric, but B is not.

$$\underbrace{A = \begin{bmatrix} 1 & 4 \\ 4 & 3 \end{bmatrix}, \quad A^T = \begin{bmatrix} 1 & 4 \\ 4 & 3 \end{bmatrix} = A}_{\text{symmetric}}$$

$$\underbrace{B = \begin{bmatrix} 1 & 4 \\ -2 & 3 \end{bmatrix}, \quad B^T = \begin{bmatrix} 1 & -2 \\ 4 & 3 \end{bmatrix} \neq B}_{\text{not symmetric}}$$

A symmetric matrix is such that row 1 is the same as column 1, row 2 is the same as column 2, and so on. We'll see several applications of symmetric matrices later in the course – for example, they are easy to work with in multivariate calculus.

But, where do they come from? Data usually isn't symmetric: if X is a matrix in which each row is a data point and each column is a feature, then usually X is very tall, and thus can't be symmetric.

Here's the key: for any $n \times d$ matrix X, X^TX is a symmetric $d \times d$ matrix.We can verify this using the fact that $(AB)^T = B^TA^T$.

$$(X^TX)^T = X^T(X^T)^T = X^TX$$

I think of X^TX as the **dot product matrix** of X, because its entries are the **dot products of the pairs of columns of** X. For example, let

$$X = \begin{bmatrix} 2 & -6 & 1 \\ 3 & 1 & -1 \end{bmatrix}$$

Then,

$$X^TX = \begin{bmatrix} 2 & 3 \\ -6 & 1 \\ 1 & -1 \end{bmatrix} \begin{bmatrix} 2 & -6 & 1 \\ 3 & 1 & -1 \end{bmatrix} = \begin{bmatrix} 13 & -9 & -1 \\ -9 & 37 & -7 \\ -1 & -7 & 2 \end{bmatrix}$$

X^TX contains the dot products of the columns of X with each other. For instance, -9 in position $(1, 2)$ is the dot product of

- row 1 of X^T, which is column 1 of X, with
- column 2 of X

The elements along the **diagonal** of X^TX – 13, 37, and 2 – are the dot products of the columns of X with themselves, meaning they are the squared norms of the columns.

Activity 3

Activity 3

Above, we defined X^TX as the dot product matrix, since it contained the dot products of the columns of X with each other.
Now, let's consider XX^T.

- What is the shape of XX^T, i.e. how many rows and columns does it have?

- What is the English meaning of XX^T?

Diagonal Matrices

Definition: Diagonal Matrix

A diagonal matrix is a matrix where all of the off-diagonal elements are zero. More formally, A is diagonal if $A_{ij} = 0$ for all $i \neq j$.

Usually, diagonal matrices are square, like the examples below.

$$\begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}, \quad \begin{bmatrix} 2 & 0 \\ 0 & 3 \end{bmatrix}, \quad \begin{bmatrix} -\pi & 0 & 0 \\ 0 & 15 & 0 \\ 0 & 0 & 3 \end{bmatrix}$$

Notice that the diagonal is **down and to the right** – position (1, 1) is the first diagonal element, position (2, 2) is the second diagonal element, and so on.

Diagonal matrices don't have to be square. We could also call the following a diagonal matrix.

$$\begin{bmatrix} 1 & 0 & 0 \\ 0 & 2 & 0 \end{bmatrix}$$

What is their significance? Let's observe what happens when we multiply a diagonal matrix by a vector, as we did in Activity 4. of Chapter 5.1.

$$\underbrace{\begin{bmatrix} -3 & 0 & 0 \\ 0 & \pi & 0 \\ 0 & 0 & 5 \end{bmatrix}}_{D} \underbrace{\begin{bmatrix} 1 \\ 2 \\ 3 \end{bmatrix}}_{\vec{x}} = \begin{bmatrix} -3 \\ 2\pi \\ 15 \end{bmatrix}$$

What happened to $\vec{x}$ after being multiplied by D? Each component was **stretched**. Element 1 of $\vec{x}$ was stretched by -3, element 2 was stretched by π, and element 3 was stretched by 5. The 0's in the off-diagonal elements allowed D to scale each component of $\vec{x}$ **independently**.

Triangular Matrices

Definition: Upper Triangular and Lower Triangular Matrices

An **upper triangular matrix** is a square matrix where all of the elements below the diagonal are zero.
A **lower triangular matrix** is a square matrix where all of the elements above the diagonal are zero.

For instance, $\begin{bmatrix} 1 & 2 & 3 \\ 0 & 4 & 5 \\ 0 & 0 & 6 \end{bmatrix}$ is upper triangular, and $\begin{bmatrix} 1 & 0 & 0 \\ 2 & 3 & 0 \\ 4 & 5 & 6 \end{bmatrix}$ is lower triangular. I'll have more to say on these matrices later in the term; for now, I just want you to be aware of their existence.

Orthogonal Matrices

The final category of matrix I want to introduce here is the orthogonal matrix. It's the first type of matrix here whose defining property can't be determined just by looking at the individual elements of the matrix; it involves some computation.

Definition: Orthogonal Matrix

A square matrix A is **orthogonal** if $A^TA = AA^T = I$.

Let's consider the matrix

$$A = \begin{bmatrix} \frac{\sqrt{3}}{2} & -\frac{1}{2} \\ \frac{1}{2} & \frac{\sqrt{3}}{2} \end{bmatrix}$$

It is orthogonal, because A^TA and AA^T are both equal to the 2×2 identity matrix. What does this really tell us? When I discussed symmetric matrices above, I said that A^TA for any matrix A contains the dot products of the columns of A with each other. So, if

$$A^TA = \begin{bmatrix} \frac{\sqrt{3}}{2} & \frac{1}{2} \\ -\frac{1}{2} & \frac{\sqrt{3}}{2} \end{bmatrix} \begin{bmatrix} \frac{\sqrt{3}}{2} & -\frac{1}{2} \\ \frac{1}{2} & \frac{\sqrt{3}}{2} \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$$

this tells us that A's **columns** are:

- **unit vectors**, since the dot product of each column with itself is 1
- **orthogonal**, since the dot product of each column with every other column is 0

Since $AA^T = I$, too, this means that A's **rows** are also unit vectors that are orthogonal to each other.

So, orthogonal matrices are matrices whose

- columns are unit vectors
- rows are unit vectors
- columns are orthogonal to each other
- rows are orthogonal to each other

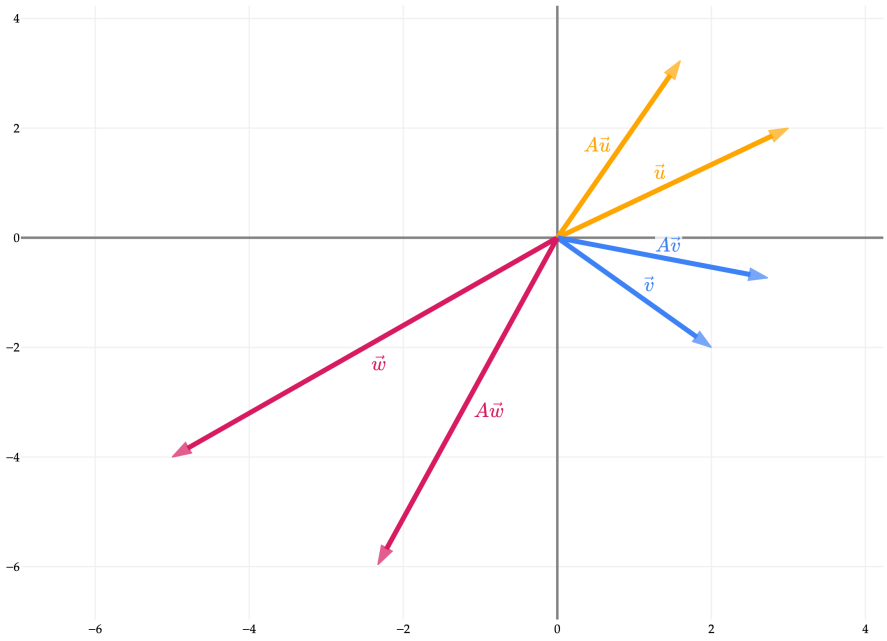
If a collection of vectors are all unit vectors and orthogonal, we may call that collection **orthonormal**. So, an orthogonal matrix has orthonormal columns **and** orthonormal rows.

Let's preview some ideas from Chapter 6.1, let's visualize six vectors in $\mathbb{R}^2$.

- $\vec{u} = \begin{bmatrix} 3 \\ 2 \end{bmatrix}$ and $A\vec{u}$

- $\vec{v} = \begin{bmatrix} 2 \\ -2 \end{bmatrix}$ and $A\vec{v}$
- $\vec{w} = \begin{bmatrix} -5 \\ -4 \end{bmatrix}$ and $A\vec{w}$

What do you notice about the vectors $A\vec{u}$, $A\vec{v}$, and $A\vec{w}$? and how they relate to $\vec{u}$, $\vec{v}$, and $\vec{w}$?



A corresponds to a **rotation**, since it rotates vectors by a certain angle (in this case, $\frac{\pi}{6}$ radians, or $30°$) but doesn't change their length. Why $\frac{\pi}{6}$? More on that in Chapter 6.1.

A question we can answer now: why does multiplying an $\vec{x}$ by A preserve its length? Let's prove this, using the fact that $\|\vec{v}\|^2 = \vec{v} \cdot \vec{v}$, which is also equal to $\vec{v}^T\vec{v}$ using our knowledge of the transpose operator.

$$\|A\vec{x}\|^2 = (A\vec{x})^T(A\vec{x}) = \vec{x}^T A^T A\vec{x} = \vec{x}^T I\vec{x} = \vec{x}^T\vec{x} = \|\vec{x}\|^2$$

The length of $A\vec{x}$ is the same as the length of $\vec{x}$, because $A^TA = I$! So, multiplying a vector by an orthogonal matrix preserves its length, and thus the only thing that *could* change is its direction.

Orthogonal matrices perform a **rotation**, which is a type of **linear transformation**. There exist plenty of different types of linear transformations, like reflections, sheers, and projections (which sound familiar). These will all become familiar in Chapter 6.1.

All I wanted to show you for now is that matrix multiplication may *look* like a bunch of random number crunching, but there's a lot of meaning baked in.

The last thing I'll note on orthogonal matrices is that just because a matrix satisfies $A^TA = I$, it doesn't mean that it is orthogonal: it just means that its **columns** are ortho**normal**. A may not even be square, which is a prerequisite for a matrix to be orthogonal. For instance,

$$A = \begin{bmatrix} 3/5 & 4/5 \\ 0 & 0 \\ 4/5 & -3/5 \end{bmatrix}$$

Then,

$$A^TA = \begin{bmatrix} 3/5 & 0 & 4/5 \\ 4/5 & 0 & -3/5 \end{bmatrix} \begin{bmatrix} 3/5 & 4/5 \\ 0 & 0 \\ 4/5 & -3/5 \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} = I_2$$

But AA^T is not equal to the above. It's not even equal to the 3×3 identity matrix!

$$AA^T = \begin{bmatrix} 3/5 & 4/5 \\ 0 & 0 \\ 4/5 & -3/5 \end{bmatrix} \begin{bmatrix} 3/5 & 0 & 4/5 \\ 4/5 & 0 & -3/5 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix} \neq I_2$$

Activity 4

Activity 4

Suppose A is an $n \times d$ matrix whose columns are orthogonal to each other, though not necessarily orthonormal. What type of matrix is A^TA?

Solution

A^TA is a **diagonal** matrix. Recall, A^TA contains the dot products of the columns of A with each other. The diagonal elements are the dot products of the columns with themselves, which are the squared norms of the columns. The off-diagonal elements (i.e. the elements of A^TA where $i \neq j$) are the dot products of the columns with each other, which are 0 because the columns are orthogonal. Since the columns aren't necessarily orthonormal, the diagonal elements are not necessarily 1.
For example, if

$$A = \begin{bmatrix} 3 & 2 \\ 0 & 1 \\ 6 & -1 \end{bmatrix}$$

then

$$A^TA = \begin{bmatrix} 3 & 0 & 6 \\ 2 & 1 & -1 \end{bmatrix} \begin{bmatrix} 3 & 2 \\ 0 & 1 \\ 6 & -1 \end{bmatrix} = \begin{bmatrix} 45 & 0 \\ 0 & 6 \end{bmatrix}$$

5.3. Rank and Column Space

In Chapter 5.1, I encouraged you to think of a matrix as a collection of vectors stacked together. Soon, these matrices will be made up of observations and features from some dataset that we'd like to build a regression model with. The content of this section will be a crucial building block for helping us return to the problem of regression.

Column Space and Rank

Let's keep working with the matrix A from Chapter 5.1.

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

A is a 5×3 matrix. It can be thought of as either:

- 3 vectors in $\mathbb{R}^5$ (the columns of A)
- 5 vectors in $\mathbb{R}^3$ (the rows of A)

Let's start with the column perspective. We saw in Chapter 5.1 that if $x \in \mathbb{R}^3$, then $A\vec{x}$ is a new vector in $\mathbb{R}^5$ that is a **linear combination** of the columns of A. For instance, if we take $\vec{x} = \begin{bmatrix} 2 \\ 0 \\ -1 \end{bmatrix}$, then

$$A\vec{x} = \begin{bmatrix} 5 & 3 & 2 \\ 0 & -1 & 1 \\ 3 & 4 & -1 \\ 6 & 2 & 4 \\ 1 & 0 & 1 \end{bmatrix} \begin{bmatrix} 2 \\ 0 \\ -1 \end{bmatrix} = 2 \begin{bmatrix} 5 \\ 0 \\ 3 \\ 6 \\ 1 \end{bmatrix} + 0 \begin{bmatrix} 3 \\ -1 \\ 4 \\ 2 \\ 0 \end{bmatrix} - 1 \begin{bmatrix} 2 \\ 1 \\ -1 \\ 4 \\ 1 \end{bmatrix} = \begin{bmatrix} 8 \\ -1 \\ 7 \\ 8 \\ 1 \end{bmatrix}$$

By definition, the vector $A\vec{x}$ is in the span of the columns of A, since it's just a linear combination of A's columns. Given what we've learned in Chapter 4.1 and Chapter 4.3 about spans and vector spaces, it's natural to try and describe the span of A's columns.

Definition: Column Space

If A is an $n \times d$ matrix, then the **column space** of A, denoted colsp(A), is the span of the columns of A. Equivalently, it is the set of all possible results of $A\vec{x}$ for $\vec{x} \in \mathbb{R}^d$.
So, if A's columns are $\vec{a}^{(1)}, \vec{a}^{(2)}, \ldots, \vec{a}^{(d)}$, like in

$$A = \begin{bmatrix} | & | & & | \\ \vec{a}^{(1)} & \vec{a}^{(2)} & \ldots & \vec{a}^{(d)} \\ | & | & & | \end{bmatrix}$$

then

$$\underbrace{\text{colsp}(A) = \text{span}\left(\vec{a}^{(1)}, \vec{a}^{(2)}, \ldots, \vec{a}^{(d)}\right) = \{A\vec{x} \mid \vec{x} \in \mathbb{R}^d\}}_{\text{subspace of } \mathbb{R}^n}$$

Notice that I've intentionally chosen not to use subscripts to refer to columns; this is so that when we switch back to focusing on datasets and machine learning, we keep consistent the fact that subscripts refer to different rows/data points, not columns/features.

"Column space" is just a new term for a concept we're already familiar with: the span of a set of vectors. In the example A we've been working with, the column space is

$$\text{colsp}(A) = \text{span}\left(\left\{\begin{bmatrix}5\\0\\3\\6\\1\end{bmatrix}, \begin{bmatrix}3\\-1\\4\\2\\0\end{bmatrix}, \begin{bmatrix}2\\1\\-1\\4\\1\end{bmatrix}\right\}\right)$$

This column space is a **2-dimensional subspace** of $\mathbb{R}^5$. Why is it 2-dimensional? The last column is a linear combination of the first two columns. Specifically,

$$\text{column 3} = \text{column 1} - \text{column 2}$$

Remember that:

- the **dimension** of a subspace is the number of vectors in a basis for the subspace, and
- a **basis** for a subspace is a linearly independent set of vectors that spans the entire subspace.

The first two columns of A alone span the column space, and are linearly independent, and so $\dim(\text{colsp}(A)) = 2$. This number, 2, is the most important number associated with the matrix A, so much so that we give it a special name.

Definition: Rank

The **rank** of a matrix A, denoted $\text{rank}(A)$, is the **number of linearly independent columns of** A. Equivalently, it is the dimension of the column space of A.

$$\text{rank}(A) = \dim(\text{colsp}(A))$$

The rank of a matrix tells us how "large" the space of possible linear combinations of the columns of A is. We care about this because ultimately, our predictions in regression are just linear combinations of the columns of some data matrix.

To get a feel for the idea of rank, let's work through some examples.

Use these examples as activities - we won't cover them all in lecture!

That's why the solutions are intentionally hidden.

Example: Creating Matrices. Find three 3×4 matrices: one with rank 1, one with rank 2, and one with rank 3. Is it possible to have a 3×4 matrix with rank 4?

Solution

- To create a 3×4 matrix with rank 1, we need there to only be one linearly independent column. For instance,

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

  has rank 1 because all columns are multiples of the first column.

- To create a 3×4 matrix with rank 2, we need there to be two linearly independent columns. One way to construct such a matrix is to make the first two columns linearly independent, and make the last two columns linear combinations of the first two. For instance, in

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

  column 3 is a scalar multiple of column 1, and column 4 is column 1 + column 2.

- To create a 3×4 matrix with rank 3, we need there to be three linearly independent columns, and the fourth column can be anything. One solution is

$$A = \begin{bmatrix} 1 & 0 & 0 & 9 \\ 0 & 1 & 0 & 8 \\ 0 & 0 & 1 & 7 \end{bmatrix}$$

- A 3×4 matrix cannot have rank 4, because it's impossible to have four linearly independent vectors in $\mathbb{R}^3$. Any three linearly independent vectors in $\mathbb{R}^3$ span all of $\mathbb{R}^3$, so a fourth vector in $\mathbb{R}^3$ would have to be a linear combination of the first three vectors.

Example: 2×2 Matrices. Let

$$A = \begin{bmatrix} a & b \\ c & d \end{bmatrix}$$

Find a condition on a, b, c, d that ensures $\text{rank}(A) = 2$.

Solution

In order for $\text{rank}(A) = 2$, the columns of A must be linearly independent. This means that the second column cannot be a scalar multiple of the first column.
$\frac{b}{a}$ is the number we multiply a by to get b. So, we just need $\frac{b}{a} \cdot c$ to be different from d.

$$\frac{b}{a} \cdot c \neq d \implies ad - bc \neq 0$$

So, if $ad - bc = 0$, then $\text{rank}(A) = 1$, and otherwise, $\text{rank}(A) = 2$.
This expression, $ad - bc$, is called the **determinant** of A. We'll learn more about determinants in Chapter 6.2.

Example: Diagonal Matrices. Suppose $d_1, d_2, \ldots, d_n$ are real numbers. What is the rank of the $n \times n$ **diagonal** matrix

$$D = \begin{bmatrix} d_1 & 0 & \cdots & 0 \\ 0 & d_2 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & d_n \end{bmatrix}$$

if

1. $d_i = i$, for $i = 1, 2, \ldots, n$?
2. $d_1 = d_2 = \cdots = d_n = 0$?
3. k of the d_i are equal to 0, and the rest are positive?

Solution

1. rank$(D) = n$: If $d_i = i$, for $i = 1, 2, \ldots, n$, meaning $d_1 = 1, d_2 = 2, \ldots, d_n = n$, then the rank is n, because all columns are linearly independent. None can be written as a linear combination of any others, since they all have their sole non-zero entry in a different row.
2. rank$(D) = 0$: If $d_1 = d_2 = \cdots = d_n = 0$, then the rank is 0, because all columns are the zero vector.
3. rank$(D) = n - k$: If k of the d_i are equal to 0, and the rest are positive, then the rank is $n - k$, because we can throw out the zero columns and the remaining columns are linearly independent.

Example: Vector Outer Product. Let $\vec{u} = \begin{bmatrix} 1 \\ -3 \\ 4 \end{bmatrix}$ and $\vec{v} = \begin{bmatrix} 2 \\ 5 \\ -1 \end{bmatrix}$.

As we've seen before, the dot product $\vec{u} \cdot \vec{v} = \vec{u}^T \vec{v}$ is a **scalar**, equal to -17 here.

The **outer product** of $\vec{u}$ and $\vec{v}$ is the matrix

$$\vec{u}\vec{v}^T = \begin{bmatrix} 1 \\ -3 \\ 4 \end{bmatrix} \begin{bmatrix} 2 & 5 & -1 \end{bmatrix} = \begin{bmatrix} 2 & 5 & -1 \\ -6 & -15 & 3 \\ 8 & 20 & -4 \end{bmatrix}$$

In general, for any two vectors $\vec{u}, \vec{v} \in \mathbb{R}^n$, what is the rank of $\vec{u}\vec{v}^T$?

Solution

The rank of $\vec{u}\vec{v}^T$ is 1, since each column in $\vec{u}\vec{v}^T$ is a scalar multiple of $\vec{u}$. In the example above, column 2 is $\frac{5}{2}$ times column 1, and column 3 is $-\frac{1}{2}$ times column 1.
In Chapter 10, we'll see that any rank r matrix can be written as a sum of r rank 1 matrices, each of which is of the form $\vec{u}\vec{v}^T$. So, rank 1 matrices can be thought of as the building blocks of all matrices!

Example: Basis for Column Space. Find a basis for the column space of

$$A = \begin{bmatrix} 3 & 6 & 0 & 9 & 3 \\ 2 & 4 & 0 & 6 & 2 \\ 0 & 0 & 1 & -5 & 0 \\ 1 & 2 & 0 & 3 & 1 \end{bmatrix}$$

Note that we've already seen plenty of problems of this form in earlier homeworks. It's just that there, the spanning set of vectors was given to you directly, and here, they're stored as columns in a matrix. The idea is the same.

Solution

- Column 2 is a multiple of column 1.
- Column 3 is independent from column 1.
- Column 4 is 3 times column 1 plus -5 times column 3.
- Column 5 is just column 1.

A only has two linearly independent columns - columns 1 and 3 - and so $\text{rank}(A) = 2$, and a basis for the column space is

$$\left\{ \begin{bmatrix} 3 \\ 2 \\ 0 \\ 1 \end{bmatrix}, \begin{bmatrix} 0 \\ 0 \\ 1 \\ 0 \end{bmatrix} \right\}$$

That's not the only possible basis for the column space; two others are

$$\left\{ \begin{bmatrix} 30 \\ 20 \\ 0 \\ 10 \end{bmatrix}, \begin{bmatrix} 0 \\ 0 \\ -15 \\ 0 \end{bmatrix} \right\}$$

and

$$\left\{ \begin{bmatrix} 0 \\ 0 \\ 1 \\ 0 \end{bmatrix}, \begin{bmatrix} 9 \\ 6 \\ -5 \\ 3 \end{bmatrix} \right\}$$

Finding the Rank Using Python. Shortly, we'll learn a new technique for finding the rank of matrix by hand. But for the most part, we'll not need to do this, and instead can use the power of Python to help us.

Returning to

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

we have

```
import numpy as np

A = np.array([[5, 3, 2],
              [0, -1, 1],
              [3, 4, -1],
              [6, 2, 4],
              [1, 0, 1]])

np.linalg.matrix_rank(A)
```

```
2
```

Python makes it easy to experiment with how different operations affect the rank of a matrix.
For instance, in Chapter 5.4, we'll prove that the matrix $A^T A$ has the same rank as A, for any $n \times d$ matrix A.

```
A.T @ A
```

```
array([[71, 39, 32],
       [39, 30,  9],
       [32,  9, 23]])
```

```
# Same as rank of A from above!
np.linalg.matrix_rank(A.T @ A)
```

```
2
```

Row Space

So far, we've focused on thinking of a matrix as a collection of "column" vectors written next to each other. This is the more common perspective, since – as I've harped on – $A\vec{x}$ is a linear combination of the columns of A.
But we can also think of a matrix as a collection of "row" vectors written on top of each other.

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

A contains 5 vectors in $\mathbb{R}^3$ in its rows, each in $\mathbb{R}^3$. These vectors also have a span, which in this case is a subspace of $\mathbb{R}^3$.

Definition: Row Space

If A is an $n \times d$ matrix, then the **row space** of A, denoted $\text{rowsp}(A)$, is the span of the rows of A. Equivalently, it is the set of all possible results of $A^T \vec{y}$ for $\vec{y} \in \mathbb{R}^n$.
So, if A's rows are $\vec{a}_1, \vec{a}_2, \ldots, \vec{a}_n$, like in

$$A = \begin{bmatrix} -- & \vec{a}_1 & -- \\ -- & \vec{a}_2 & -- \\ & \vdots & \\ -- & \vec{a}_n & -- \end{bmatrix}$$

then

$$\underbrace{\text{rowsp}(A) = \text{span}\left(\vec{a}_1, \vec{a}_2, \ldots, \vec{a}_n\right) = \{A^T\vec{y} \mid \vec{y} \in \mathbb{R}^n\}}_{\text{subspace of } \mathbb{R}^d}$$

Where did $A^T\vec{y}$ come from? Remember, $A\vec{x}$ is a linear combination of the columns of A. If we transpose A, then $A^T\vec{y}$ is a linear combination of the columns of A^T, which are the rows of A.

$$A^T\vec{y} = \begin{bmatrix} 5 & 0 & 3 & 6 & 1 \\ 3 & -1 & 4 & 2 & 0 \\ 2 & 1 & -1 & 4 & 1 \end{bmatrix} \begin{bmatrix} y_1 \\ y_2 \\ y_3 \\ y_4 \\ y_5 \end{bmatrix}$$

$$= \underbrace{y_1 \begin{bmatrix} 5 \\ 3 \\ 2 \end{bmatrix} + y_2 \begin{bmatrix} 0 \\ -1 \\ 1 \end{bmatrix} + y_3 \begin{bmatrix} 3 \\ 4 \\ -1 \end{bmatrix} + y_4 \begin{bmatrix} 6 \\ 2 \\ 4 \end{bmatrix} + y_5 \begin{bmatrix} 1 \\ 0 \\ 1 \end{bmatrix}}_{\text{linear combination of rows of A}}$$

Remember from Chapter 5.1 that $(A^T\vec{y})^T = \vec{y}^T A$. The product $\vec{y}^T A$ is also a linear combination of the rows of A; it just returns a row vector with shape $1 \times d$ rather than a vector with shape $d \times 1$.

$$\vec{y}^T A = \begin{bmatrix} y_1 & y_2 & y_3 & y_4 & y_5 \end{bmatrix} \begin{bmatrix} 5 & 3 & 2 \\ 0 & -1 & 1 \\ 3 & 4 & -1 \\ 6 & 2 & 4 \\ 1 & 0 & 1 \end{bmatrix}$$

$$= \underbrace{y_1 \begin{bmatrix} 5 & 3 & 2 \end{bmatrix} + y_2 \begin{bmatrix} 0 & -1 & 1 \end{bmatrix} + \ldots + y_5 \begin{bmatrix} 1 & 0 & 1 \end{bmatrix}}_{\text{linear combination of rows of A}}$$

In $\vec{y}^T A$, we **left-multiplied** A by a vector; in $A\vec{x}$, we **right-multiplied** A by a vector. These are distinct types of multiplication, as they involve vectors of different shapes.

Since the columns of A^T are the rows of A, the row space of A is the column space of A^T, meaning

$$\text{rowsp}(A) = \text{colsp}(A^T)$$

To avoid carrying around lots of notation, I'll often just use $\text{colsp}(A^T)$ to refer to the row space of A.

Dimension of the Row Space. **What is the dimension of the row space of A?** Other ways of phrasing this question are:

- How many linearly independent rows does A have?
- What is the rank of A^T?

We know the answer can't be more than 3, since the rows of A are vectors in $\mathbb{R}^3$. We could use the algorithm first presented in Chapter 4.1 to find a linearly independent set of rows with the same span as all 5 rows.

An easy way to see that $\dim(\text{colsp}(A^T)) = 2$ is to pick out two of the five vectors, row $2 = \vec{a}_2 = \begin{bmatrix} 0 \\ -1 \\ 1 \end{bmatrix}$ and row $5 = \vec{a}_5 = \begin{bmatrix} 1 \\ 0 \\ 1 \end{bmatrix}$, and show that the other three vectors can be written as linear combinations of them. I chose $\vec{a}_2$ and $\vec{a}_5$ just because they have the simplest numbers, and because they're linearly independent from one another.

- $\vec{a}_1 = \begin{bmatrix} 5 \\ 3 \\ 2 \end{bmatrix} = -3\vec{a}_2 + 5\vec{a}_5$
- $\vec{a}_3 = \begin{bmatrix} 3 \\ 4 \\ -1 \end{bmatrix} = -4\vec{a}_2 + 3\vec{a}_5$
- $\vec{a}_4 = \begin{bmatrix} 6 \\ 2 \\ 4 \end{bmatrix} = -2\vec{a}_2 + 6\vec{a}_5$

You might notice that the number of linearly independent rows of A and the number of linearly independent columns of A were both 2.

Equivalence of Column Rank and Row Rank. Sometimes, we say the dimension of $\text{colsp}(A)$ is the **column rank** of A, and the dimension of $\text{colsp}(A^T)$ is the **row rank** of A. For the example A we've been working with, both the column rank and row rank are 2.

But, there's no need for separate names.

Column Rank = Row Rank!

For any matrix A,

$$\text{number of linearly independent columns of } A = \text{number of linearly independent rows of } A$$

Both of these are equal to $\text{rank}(A)$. This implies that

$$\text{rank}(A) = \text{rank}(A^T)$$

for any matrix A!

It's not immediately obvious why this is true, and honestly, most of the proofs of it I've found aren't all that convincing for a first-time linear algebra student. Still, I'll *try* to prove this fact for you in just a little bit.

So, in general, what is $\text{rank}(A)$? Since the rank is equal to both the number of linearly independent columns and the number of linearly independent rows, then the largest the rank can be is the **smaller** of the number of rows and columns.

For example, if A is a 7×9 matrix, then its rank is at most 7, since it cannot have more than 7 linearly independent rows. So in general, if A is an $n \times d$ matrix, then

$$0 \leq \text{rank}(A) \leq \min(n, d)$$

$$\underbrace{\begin{bmatrix} 5 & 3 \\ 2 & 1 \\ -1 & \frac{1}{3} \\ 3 & 6 \\ 0 & 1 \end{bmatrix}}_{\text{if } n>d\text{, rows can't be independent}} \qquad \underbrace{\begin{bmatrix} 1 & 0 & 3 & 2 & -1 \\ \frac{1}{9} & 3 & 0 & 0 & 2 \\ 9 & 0 & 0 & 6 & -3 \end{bmatrix}}_{\text{if } n<d\text{, columns can't be independent}}$$

We say a matrix is **full rank** if it has the largest possible rank for a matrix of its shape, i.e. **if** $\text{rank}(A) = \min(n, d)$. We'll mostly use this term when refering to square matrices, which we'll focus more on in Chapter 6.2.

What if a matrix **isn't** full rank? (What a cliffhanger!) Keep reading Chapter 5.4 for the full story.

5.4. Null Space and the Rank-Nullity Theorem

Overview

As we saw in Chapter 5.3, colsp(A) is a subspace of $\mathbb{R}^n$ that contains all possible results of $A\vec{x}$ for $\vec{x} \in \mathbb{R}^d$. Think of colsp(A) as the "reach" of the columns of A.

If the columns of A are linearly independent, then the only way to create $\vec{0}$ through a linear combination of the columns of A is if all the coefficients in the linear combination are 0, i.e. if $\vec{x} = \vec{0}$ in $A\vec{x}$.

But, if A's columns aren't all linearly independent, then there will be some **non-zero** vectors $\vec{x}$ where $A\vec{x} = \vec{0}$. This holds from the definition of linear independence, which says that if there's a non-zero linear combination of a collection of vectors that produces $\vec{0}$, the vectors aren't linearly independent.

It turns out that it's worthwhile to study the set of vectors $\vec{x}$ that **get sent to** $\vec{0}$ when multiplied by A.

Null Space

Definition: Null Space

If A is an $n \times d$ matrix, then the **null space** of A, denoted nullsp(A), is the set of all vectors $\vec{x} \in \mathbb{R}^d$ where $A\vec{x} = \vec{0}$.

$$\underbrace{\text{nullsp}(A) = \{\vec{x} \in \mathbb{R}^d \mid A\vec{x} = \vec{0}\}}_{\text{subspace of } \mathbb{R}^d}$$

Sometimes, the null space is also called the **kernel** of A, though we will mostly avoid that term in our class, since kernel often means something different in the context of machine learning.

Important: nullsp(A) is a subspace of $\mathbb{R}^{\boxed{d}}$, since it is made up of vectors that get multiplied by A, an $n \times d$ matrix. Vectors in nullsp(A) are in $\mathbb{R}^d$, while vectors in colsp(A) are in $\mathbb{R}^n$.

Let's return to the example A from earlier.

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

nullsp(A) is the set of all vectors $\vec{x} \in \mathbb{R}^3$ where $A\vec{x} = \vec{0}$. rank(A) = 2, which is less than 3 (the number of columns of A), so there will be some non-zero vectors $\vec{x}$ in the null space. But what are they?

The Systems of Equations Approach. To find them, we need to find the general solution to

$$\underbrace{\begin{bmatrix} 5 & 3 & 2 \\ 0 & -1 & 1 \\ 3 & 4 & -1 \\ 6 & 2 & 4 \\ 1 & 0 & 1 \end{bmatrix}}_{A} \underbrace{\begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix}}_{\vec{x}} = \underbrace{\begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix}}_{\vec{0}}$$

One way to proceed is to solve the system of equations

$$\begin{aligned} 5x_1 + 3x_2 &\quad +2x_3 = 0 \\ -x_2 &\quad +x_3 = 0 \\ 3x_1 + 4x_2 &\quad -x_3 = 0 \\ 6x_1 + 2x_2 &\quad +4x_3 = 0 \\ x_1 &\quad +x_3 = 0 \end{aligned}$$

Equation (2) tells us $x_2 = x_3$, and equation (5) tells us $x_1 = -x_3$. So, any vector in nullsp(A) is of the form

$$\begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} -x_3 \\ x_3 \\ x_3 \end{bmatrix} = x_3 \begin{bmatrix} -1 \\ 1 \\ 1 \end{bmatrix}, \quad x_3 \in \mathbb{R}$$

So,

$$\text{nullsp}(A) = \text{span}\left(\left\{\begin{bmatrix} -1 \\ 1 \\ 1 \end{bmatrix}\right\}\right)$$

For example, $\begin{bmatrix} -5 \\ 5 \\ 5 \end{bmatrix} \in \text{nullsp}(A)$. Remember that vectors in nullsp(A) are in $\mathbb{R}^3$, since A is a 5×3 matrix, while vectors in colsp(A) are in $\mathbb{R}^5$.

The Shortcut. There's a shortcut to finding the null space of a matrix, which works if you happen to already know the relationship between the columns of A, or if the relationship is simple enough to eyeball. In Chapter 5.3, we saw that

$$\text{column 3} = \text{column 1} - \text{column 2}$$

or, in other words,

$$\text{column 1} - \text{column 2} - \text{column 3} = \vec{0}$$

This, right away, tells us that taking 1 of column 1, subtracting 1 of column 2, and subtracting 1 of column 3 will give us the zero vector. But $A\vec{x}$ is just a linear combination of the columns in A using the coefficients in $\vec{x}$, so this means that

$$A\begin{bmatrix}1\\-1\\-1\end{bmatrix} = \vec{0}$$

So, we've found a vector in nullsp(A). Any scalar multiple of this vector will also be in nullsp(A).

In this example, nullsp(A) is a 1-dimensional subspace of $\mathbb{R}^3$. We also know from earlier that rank(A) = 2. And curiously, $1 + 2 = 3$, the number of columns in A. This is not a coincidence, and sheds light on an important theorem.

Rank-Nullity Theorem

Rank-Nullity Theorem

The **rank-nullity theorem** states that for any $n \times d$ matrix A,

$$\text{rank}(A) + \dim(\text{nullsp}(A)) = \underbrace{\text{number of columns of } A}_{d}$$

The proof of this theorem is beyond the scope of our course. But, this is such an important theorem that it's sometimes called the **fundamental theorem of linear algebra**. It tells us, for one, that the dimension of the null space is equal to the number of columns minus the rank. "Nullity" is just another word for the dimension of the null space.

Let's see how it can be used in practice. Some of these examples are taken from Gilbert Strang's book.

Examples

Example: Linearly Independent Columns. Let $A = \begin{bmatrix}3 & 1\\9 & -3\\0 & 6\\3 & 2\\1 & 1\end{bmatrix}$.

What is nullsp(A)?

Solution

Since A's two columns are linearly independent, rank(A) = 2, and by the rank-nullity theorem, $\dim(\text{nullsp}(A)) = 2 - 2 = 0$.

So, nullsp(A) = $\{\vec{0}\}$, meaning that the only vector in the null space of A is the zero vector. No other vector $\vec{x}$ satisfies $A\vec{x} = \vec{0}$.

If A's columns are linearly independent, then nullsp(A) = $\{\vec{0}\}$

The above example illustrates an important fact: if A's columns are all linearly independent, then the only vector in the null space of A is the zero vector, i.e. nullsp(A) = $\{\vec{0}\}$. This is called the **trivial null space**,

and has a dimension of 0.
If A's columns aren't linearly independent, then nullsp(A) will contain more than just the zero vector – in fact, it'll be a subspace, which contains infinitely many vectors, spanned by some basis – and we'll say the null space is **non-trivial**.

Example: Describing Spaces. Describe the column space, row space, and null space of the matrix

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

Additionally, find a basis for the null space.

Solution

A only has one linearly independent column; columns 2 and 3 are both just multiples of column 1. So, $\text{rank}(A) = 1$.
The **column space** of A is the span of the first column, i.e.

$$\text{colsp}(A) = \text{span}\left(\left\{\begin{bmatrix}1\\2\end{bmatrix}\right\}\right)$$

This is a **1-dimensional subspace** of $\mathbb{R}^2$; 1 comes from the rank of A, and 2 comes from the fact that each column of A is a vector in $\mathbb{R}^2$.
The **row space** of A is the span of the first row, i.e.

$$\text{colsp}(A^T) = \text{span}\left(\left\{\begin{bmatrix}1\\2\\3\end{bmatrix}\right\}\right)$$

This is a **1-dimensional subspace** of $\mathbb{R}^3$. Remember that the number of linearly independent columns and rows of a matrix are always the same, which is why we know the dimensions of the column space and row space are the same.
The **null space** of A is the set of all vectors $\vec{x} \in \mathbb{R}^3$ where $A\vec{x} = \vec{0}$. This is a **2-dimensional subspace** of $\mathbb{R}^3$. 2 came from the rank-nullity theorem, which says that the dimension of the null space is equal to the number of columns minus the rank, or

$$3 - 1 = 2$$

here.
Can we say more about the null space? It is the set of all vectors $\vec{x} \in \mathbb{R}^3$ where $A\vec{x} = \vec{0}$. This is equivalent to the set of all vectors $\vec{x} = \begin{bmatrix}x\\y\\z\end{bmatrix}$ where

$$x + 2y + 3z = 02x + 4y + 6z = 0$$

The two equations above are equivalent. So, the null space is the set of all vectors $\vec{x} = \begin{bmatrix}x\\y\\z\end{bmatrix}$ where $x+2y+3z = 0$. This is a **plane** in $\mathbb{R}^3$, as we'd expect from a 2-dimensional subspace.
Can we find a **basis** for the null space? Yes, we can. My preferred method is to leverage the fact that we know the relationship between the columns of A:

$$\text{column 2} = 2(\text{column 1}) \implies 2(\text{column 1}) - \text{column 2} = \vec{0}$$

$$\text{column 3} = 3(\text{column 1}) \implies 3(\text{column 1}) - \text{column 3} = \vec{0}$$

So, this tells us that

$$A\begin{bmatrix}2\\-1\\0\end{bmatrix} = \begin{bmatrix}1&2&3\\2&4&6\end{bmatrix}\begin{bmatrix}2\\-1\\0\end{bmatrix} = \begin{bmatrix}0\\0\end{bmatrix}$$

and

$$A\begin{bmatrix}3\\0\\-1\end{bmatrix} = \begin{bmatrix}1&2&3\\2&4&6\end{bmatrix}\begin{bmatrix}3\\0\\-1\end{bmatrix} = \begin{bmatrix}0\\0\end{bmatrix}$$

So, we've found two linearly independent vectors in the null space; since $\text{nullsp}(A)$ is 2-dimensional, these must be a basis.

The easy way to find a basis for the null space

The above example illustrates a general method for finding a basis for the null space of a matrix. If you already know the relationship between the columns of A, you can use that to find a basis for the null space.

Example: Thinking Abstractly. Suppose A is a 3×4 matrix with rank 3. Describe colsp(A) and nullsp(A^T). The latter is the null space of A^T, and is sometimes called the **left null space** of A, as it's a null space of A when multiplied by vectors on the left, like in $\vec{y}^T A$ (which performs the same calculation as $A^T \vec{y}$; the results are just transposed).

Solution

When faced with a problem like this, I like drawing out a rectangle that roughly shows me the dimensions of the matrix.

$$A = \begin{bmatrix} \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot \end{bmatrix}$$

The rank tells us that there are 3 linearly independent columns. Since the columns themselves are in $\mathbb{R}^3$, this must mean that

$$\text{colsp}(A) = \text{all of } \mathbb{R}^3$$

since any 3 linearly independent vectors in $\mathbb{R}^3$ will span the entire space. (The existence of the 4th linearly dependent column doesn't change this fact, it just means that the linear combinations of the four columns won't be unique.)
The null space of A^T is the set of all vectors $\vec{y}$ where

$$\vec{A}^T\vec{y} = \begin{bmatrix} \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot \end{bmatrix} \begin{bmatrix} y_1 \\ y_2 \\ y_3 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \end{bmatrix}$$

$\text{nullsp}(A^T)$ is a collection of vectors in $\mathbb{R}^3$. What is the dimension of this space? Rank-nullity tells us that

$$\text{rank}(A) + \dim(\text{nullsp}(A)) = \text{number of columns of } A$$

We can't use this directly, since the matrix we're dealing with is A^T, not A. So, replacing A with A^T in the equation above, we get

$$\text{rank}(A^T) + \dim(\text{nullsp}(A^T)) = \text{number of columns of } A^T$$

But, $\text{rank}(A^T) = \text{rank}(A)$, and the number of columns of A^T is the same as the number of rows of A. So,

$$\text{rank}(A) + \dim(\text{nullsp}(A^T)) = \text{number of rows of } A \dim(\text{nullsp}(A^T)) = \text{number of rows of } A - \text{rank}(A)$$

But since A has 3 rows and a rank of 3, we know that

$$\dim(\text{nullsp}(A^T)) = 3 - 3 = 0$$

So, $\text{nullsp}(A^T)$ is a 0-dimensional subspace of $\mathbb{R}^3$, meaning it only contains the zero vector.

$$\text{nullsp}(A^T) = \{\vec{0}\}$$

More intuitively, using the results of the previous example, we know that A^T's columns are linearly independent (since there are 3 of them and $\text{rank}(A^T) = 3$). So, the only vector in $\text{nullsp}(A^T)$ is the zero vector.

Example: Thinking Even More Abstractly. If A is a 7×9 matrix with rank 5, find the dimensions of each of the following.

1. colsp(A)
2. colsp(A^T)
3. nullsp(A)
4. nullsp(A^T)

Solution

$$A = \begin{bmatrix} \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \end{bmatrix}$$

$$\dim(\text{colsp}(A)) = \text{rank}(A) = 5$$

So, colsp(A) is a 5-dimensional subspace of $\mathbb{R}^7$.

$$\dim(\text{colsp}(A^T)) = \text{rank}(A^T) = \text{rank}(A) = 5$$

So, colsp(A^T) is a 5-dimensional subspace of $\mathbb{R}^9$.

$$\dim(\text{nullsp}(A)) = \underbrace{9 - \text{rank}(A)}_{\text{rank-nullity theorem}} = 9 - 5 = 4$$

So, nullsp(A) is a 4-dimensional subspace of $\mathbb{R}^9$.

$$\dim(\text{nullsp}(A^T)) = \underbrace{7 - \text{rank}(A)}_{\text{rank-nullity theorem applied to } A^T} = 7 - 5 = 2$$

So, nullsp(A^T) is a 2-dimensional subspace of $\mathbb{R}^7$.

Example: Constructing a Matrix. Find a matrix A such that $\begin{bmatrix} 3 \\ 1 \end{bmatrix} \in \text{colsp}(A)$ and $\begin{bmatrix} 1 \\ 3 \end{bmatrix} \in \text{nullsp}(A)$.

Solution

A needs to satisfy both of the following conditions.

$$\begin{bmatrix}3\\1\end{bmatrix} \in \text{colsp}(A) \qquad \begin{bmatrix}1\\3\end{bmatrix} \in \text{nullsp}(A)$$

Since A can be multiplied by a vector in $\mathbb{R}^2$, and elements in colsp(A) are in $\mathbb{R}^2$, we know that A must be a 2×2 matrix. So, let

$$A = \begin{bmatrix}a & b\\c & d\end{bmatrix}$$

The second condition gives

$$A\begin{bmatrix}1\\3\end{bmatrix} = \begin{bmatrix}a+3b\\c+3d\end{bmatrix} = \begin{bmatrix}0\\0\end{bmatrix}$$

which implies

$$a = -3b \qquad c = -3d$$

Hence

$$A = \begin{bmatrix}-3b & b\\-3d & d\end{bmatrix}$$

A's columns are scalar multiples of one another, and are not linearly independent. This is what we'd expect, given that A has a non-trivial null space. (Remember, the trivial null space is just $\{\vec{0}\}$; since A's null space has more than just $\vec{0}$ in it, its columns are not linearly independent.)

Now, we just need to make sure we pick b and d such that $\begin{bmatrix}3\\1\end{bmatrix} \in$ colsp(A). The easy solution is to choose $b = 3$ and $d = 1$, which makes $\begin{bmatrix}3\\1\end{bmatrix}$ literally one of the columns of A, which means it must be in colsp(A). This gives

$$A = \begin{bmatrix}-9 & 3\\-3 & 1\end{bmatrix}$$

This is not the only solution; another one came from $b = 6$ and $d = 2$, which would have made

$$A = \begin{bmatrix}-18 & 6\\-6 & 2\end{bmatrix}$$

In both (and in the infinitely many other) cases, $\begin{bmatrix}3\\1\end{bmatrix} \in$ colsp(A) and $\begin{bmatrix}1\\3\end{bmatrix} \in$ nullsp(A).

Example: Rank of AB vs. Rank of A or B. Suppose A is an $n \times d$ matrix, and B is a $d \times p$ matrix. Explain why the column space of AB is a **subset** of the column space of A, and the row space of AB is a **subset** of the row space of B. What does this imply about the rank of AB?

Solution

To show that colsp(AB) is a subset of colsp(A), i.e.

$$\text{colsp}(AB) \subseteq \text{colsp}(A)$$

we need to show that any vector in colsp(AB) is also in colsp(A). AB is a matrix of shape $n \times p$, so to multiply it by a vector $\vec{x}$, $\vec{x}$ must be in $\mathbb{R}^p$.
Suppose $\vec{y} \in \mathbb{R}^n$ is in colsp(AB). Then, $\vec{y}$ can be written as

$$\vec{y} = AB\vec{x}$$

for some $\vec{x} \in \mathbb{R}^p$.
But,

$$\vec{y} = AB\vec{x} = A \underbrace{(B\vec{x})}_{\text{vector in } \mathbb{R}^d}$$

meaning that $\vec{y}$ is also a linear combination of the columns of A (since we wrote it as $A\vec{z}$, for some vector $\vec{z} \in \mathbb{R}^d$).
So, we've shown that if $\vec{y}$ is in colsp(AB), then $\vec{y}$ is also in colsp(A). Therefore, colsp$(AB) \subseteq$ colsp(A). This tells us that rank$(AB) \leq$ rank(A), since the rank of a matrix is the dimension of its column space.
Using similar logic, any vector in rowsp(AB) = colsp$((AB)^T)$ = colsp(B^TA^T) is of the form

$$B^TA^T\vec{x}$$

But, $B^TA^T\vec{x}$ is of the form $B^T\vec{y}$ for some $\vec{y} \in \mathbb{R}^d$, meaning that $B^TA^T\vec{x}$ is in the column space of B^T or row space of B. So, rowsp$(AB) \subseteq$ rowsp(B), meaning rank$(AB) \leq$ rank(B).
Putting these two results together, we have that

$$\text{rank}(AB) \leq \text{rank}(A) \text{ and } \text{rank}(AB) \leq \text{rank}(B)$$

But, since both must be true, then

$$\text{rank}(AB) \leq \min(\text{rank}(A), \text{rank}(B))$$

So intuitively, when we multiply two matrices, the rank of the resulting matrix can't be greater than the rank of either of the two matrices we started with, but it can "drop".

Example: Orthogonal Complements video. Suppose $\vec{r} \in$ colsp(A^T) and $\vec{n} \in$ nullsp(A), meaning $\vec{r}$ is in the row space of A and $\vec{n}$ is in the null space of A.

Prove that $\vec{r}$ and $\vec{n}$ must be orthogonal.

Solution

The row space of A, colsp(A^T), is the set of all vectors $\vec{r}$ where $\vec{r} = A^T\vec{y}$ for some $\vec{y} \in \mathbb{R}^n$. Note that if A is an $n \times d$ matrix, then A^T is a $d \times n$ matrix, and $\vec{r}$ is in $\mathbb{R}^d$.
The null space of A, nullsp(A), is the set of all vectors $\vec{n}$ where $A\vec{n} = \vec{0}$. Note that if A is an $n \times d$ matrix, then $\vec{n}$ is in $\mathbb{R}^d$.
So, $\vec{r}$ and $\vec{n}$ are both in $\mathbb{R}^d$, which means they exist in the same universe (they have the same number of components), and so we *can* ask if they're orthogonal. (If they had different numbers of components, this question would be a non-starter.)
In order to show that they're orthogonal, we need to show that their dot product is 0.

$$\begin{aligned}
\vec{r} \cdot \vec{n} &= (A^T\vec{y}) \cdot \vec{n} \\
&= \underbrace{(A^T\vec{y})^T\vec{n}}_{\vec{u}\cdot\vec{v}=\vec{u}^T\vec{v}} \\
&= y^T \underbrace{A\vec{n}}_{\vec{0}} \\
&= y^T\vec{0} \\
&= 0
\end{aligned}$$

So, every vector in the row space of A is orthogonal to every vector in the null space of A!

The row space and null space are orthogonal complements!

Above, we proved that the row space and null space are orthogonal complements, in $\mathbb{R}^d$. This means that every element in the row space is orthogonal to every element in the null space. The concept of an orthogonal complement was first introduced in Chapter 4.3.
It is also true that the column space and left null space are orthogonal complements, in $\mathbb{R}^n$. Meaning, if

$$\vec{x} \in \text{colsp}(A), \qquad \vec{y} \in \text{nullsp}(A^T)$$

then it must be the case that $\vec{x} \cdot \vec{y} = 0$.
To summarize:

- The row space, colsp(A^T), and null space, nullsp(A), are orthogonal complements; both are subspaces of $\mathbb{R}^d$.
- The column space, colsp(A), and left null space, nullsp(A^T), are orthogonal complements; both are subspaces of $\mathbb{R}^n$.

Example: Rank of X^TX video. Prove that rank(X^TX) = rank(X) for any $n \times d$ matrix X.

The matrix X^TX is hugely important for our regression problem, and you'll also see in a homework that it helps define the **covariance** matrix of our data.

Solution

First, let's think about the shape of X^TX. If X is an $n \times d$ matrix, then X^T is a $d \times n$ matrix, and X^TX is a $d \times d$ matrix. So, X and X^TX have the same number of columns (d), but X^TX is also square, which X doesn't have to be.

The way we'll proceed is to show that **both matrices have the same null space**. If we can show they both have the same null space, then the dimensions of both null spaces must be the same. Since the rank-nullity theorem tells us that

$$\text{rank}(A) + \dim(\text{nullsp}(A)) = \text{number of columns of } A$$

if we can show that $\dim(\text{nullsp}(X^TX)) = \dim(\text{nullsp}(X))$, then we'll have shown that $\text{rank}(X^TX) = \text{rank}(X)$, since both X and X^TX have the same number of columns.

To show that both X and X^TX have the same null space, we need to show that any vector $\vec{x}$ in the null space of X is also in the null space of X^TX, and vice versa.

Part 1: Show that $\vec{v} \in \text{nullsp}(X) \implies \vec{v} \in \text{nullsp}(X^TX)$

Suppose $\vec{v} \in \text{nullsp}(X)$. Then, $X\vec{v} = \vec{0}$. Multiplying both sides on the left by X^T, we get

$$X^TX\vec{v} = X^T\vec{0} = \vec{0}$$

So, $\vec{v}$ is also in the null space of X^TX.

(This was the easier part.)

Part 2: Show that $\vec{v} \in \text{nullsp}(X^TX) \implies \vec{v} \in \text{nullsp}(X)$

Suppose $\vec{v} \in \text{nullsp}(X^TX)$. Then, $X^TX\vec{v} = \vec{0}$. It's not immediately clear how to use this to show that $\vec{v}$ is in the null space of X, so let's try something different.

What if we left-multiply both sides of the equation by $\vec{v}^T$? This is equivalent to taking the dot product of both sides with $\vec{v}$.

$$X^TX\vec{v} = \vec{0}\vec{v}^TX^TX\vec{v} = \vec{v}^T\vec{0} = 0$$

Okay, so $\vec{v}^TX^TX\vec{v} = 0$. What does this tell us? If we look closely, the left-hand side is really just $(X\vec{v})^TX\vec{v}$, which is also just $(X\vec{v}) \cdot (X\vec{v})$, which we also know is equal to $\|X\vec{v}\|^2$.

So, we have

$$\|X\vec{v}\|^2 = 0$$

But, the only vector with a norm of 0 is the zero vector. So, $X\vec{v} = \vec{0}$. So, we've now shown that if $X^TX\vec{v} = \vec{0}$, then it **has to be the case that** $X\vec{v} = \vec{0}$ also.

Now that we've shown both directions, we've shown that $\text{nullsp}(X^TX) = \text{nullsp}(X)$, because any vector in one of these sets is also in the other set, and so the two sets must be equal.

Summary.

The Dimensions of the "Four Fundamental Subspaces"

If A is an $n \times d$ matrix with rank r, then:

1. The column space, $\text{colsp}(A)$, is an r-dimensional subspace of $\mathbb{R}^n$.
2. The row space, $\text{colsp}(A^T)$, is an r-dimensional subspace of $\mathbb{R}^d$.

3. The null space, nullsp(A), is a $(d-r)$-dimensional subspace of $\mathbb{R}^d$.
4. The left null space, nullsp(A^T), is an $(n-r)$-dimensional subspace of $\mathbb{R}^n$.

Matrix Decompositions

In the coming chapters, we'll spend a lot of our time decomposing matrices into smaller pieces, each of which gives us some insight into the data stored in the matrix. This is not a new concept: in earlier math courses, you've learned to write a number as a product of prime factors, and to factor quadratics like $x^2 - 5x + 6$ into $(x-2)(x-3)$.

The "ultimate" decomposition for the purposes of machine learning is the **singular value decomposition** (SVD), which decomposes a matrix into the product of three other matrices.

$$X = U\Sigma V^T$$

where U and V are orthogonal matrices and Σ is a diagonal matrix. This decomposition will allow us to solve the **dimensionality reduction** problem we first alluded to in Chapter 1.1.

We're not yet ready for that; the SVD will be the final topic we study in this course. For now, we'll introduce a decomposition that ties together ideas from this section, and allows us to understand why the number of linearly independent columns of A is equal to the number of linearly independent rows of A, i.e. that rank(A) = rank(A^T).

CR Decomposition. Suppose A is an $n \times d$ matrix with rank r. This tells us that A has r linearly independent columns, and the remaining $d-r$ columns can be written as linear combinations of the r "good" columns.

Let's continue thinking about

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

Recall, rank(A) = 2, and

$$\text{column 3} = \text{column 1} - \text{column 2}$$

Define the matrix C as containing just the linearly independent **columns** of A **when taken from left to right**, i.e. the columns that are obtained by running the algorithm in Chapter 4.2.

```
given v_1, v_2, ..., v_d # columns of A
initialize linearly independent set S = {v_1}
for i = 2 to d:
    if v_i is not a linear combination of S:
        add v_i to S
```

This means that C's columns are a basis for colsp(A). Notice that C is a 5×2 matrix; its number of columns is equal to the rank of A.

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

I'd like to produce a matrix R such that $A = CR$. R will tell us how to "mix" the columns of C (which are linearly independent) to produce the columns of A. Since A is 5×3 and C is 5×2, R must be 2×3 in order for the dimensions to work out.

- Column 1 of A is just 1(column 1 of C)
- Column 2 of A is just 1(column 2 of C)
- Column 3 of A is 1(column 1 of C) $-$ 1(column 2 of C)

So, R must be

$$R = \begin{bmatrix} 1 & 0 & 1 \\ 0 & 1 & -1 \end{bmatrix}$$

So, a **CR decomposition** of A is

$$A = \underbrace{\begin{bmatrix} 5 & 3 & 2 \\ 0 & -1 & 1 \\ 3 & 4 & -1 \\ 6 & 2 & 4 \\ 1 & 0 & 1 \end{bmatrix}}_{A} = \underbrace{\begin{bmatrix} 5 & 3 \\ 0 & -1 \\ 3 & 4 \\ 6 & 2 \\ 1 & 0 \end{bmatrix}}_{C} \underbrace{\begin{bmatrix} 1 & 0 & 1 \\ 0 & 1 & -1 \end{bmatrix}}_{R}$$

We defined C such that its columns are linearly independent and a basis for colsp(A). But, observe: R's **rows** are linearly independent too!

This is because the first two columns of R are $\begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$, which is I_2, the 2×2 identity matrix. It's impossible for any scalar multiple of the first row of R to produce the second row of R, because the second row has a non-zero value in a position where the first row has a zero value.

Why is rank(A) = rank(A^T)?. Not only are R's rows linearly independent, **they're also a basis for the row space of** A, colsp(A^T). Every row of A can be written as a linear combination of R's rows. The way to see why this is the case is to transpose $A = CR$ to get $A^T = (CR)^T = R^T C^T$.

$$A = \underbrace{\begin{bmatrix} 5 & 3 & 2 \\ 0 & -1 & 1 \\ 3 & 4 & -1 \\ 6 & 2 & 4 \\ 1 & 0 & 1 \end{bmatrix}}_{A} = \underbrace{\begin{bmatrix} 5 & 3 \\ 0 & -1 \\ 3 & 4 \\ 6 & 2 \\ 1 & 0 \end{bmatrix}}_{C} \underbrace{\begin{bmatrix} 1 & 0 & 1 \\ 0 & 1 & -1 \end{bmatrix}}_{R}$$

$$A^T = \underbrace{\begin{bmatrix} 5 & 0 & 3 & 6 & 1 \\ 3 & -1 & 4 & 2 & 0 \\ 2 & 1 & -1 & 4 & 1 \end{bmatrix}}_{A^T} = \underbrace{\begin{bmatrix} 1 & 0 \\ 0 & 1 \\ 1 & -1 \end{bmatrix}}_{R^T} \underbrace{\begin{bmatrix} 5 & 0 & 3 & 6 & 1 \\ 3 & -1 & 4 & 2 & 0 \end{bmatrix}}_{C^T}$$

The rows of R are the columns of R^T. The above decomposition of $A^T = R^T C^T$ tells us how to mix the rows of R to make the rows of A! For instance, it tells us that the first row of A is

$$\underbrace{5\begin{bmatrix} 1 \\ 0 \\ 1 \end{bmatrix} + 3\begin{bmatrix} 0 \\ 1 \\ -1 \end{bmatrix} = \begin{bmatrix} 5 \\ 3 \\ 2 \end{bmatrix}}_{\text{first row of } A}$$

To me, this is a bit magical: we started by picking off linearly independent **columns** and placing them in C, but the R we multiplied C by to recreate A ended up having linearly independent **rows**, which span the row space.

Sitting in front of us is an argument that the number of linearly independent columns of A is equal to the number of linearly independent rows of A! In other words, we've just argued that

$$\text{rank}(A) = \text{rank}(A^T)$$

The gist of it was that if $A = CR$ is a CR decomposition of A, then $A^T = (CR)^T = R^T C^T$ is a CR decomposition of A^T, and both CR decompositions use the same rank, r.

Non-Uniqueness of the CR Decomposition. I never formally defined the CR decomposition; I started with an example.

Definition: CR Decomposition

Suppose A is an $n \times d$ matrix with rank r, meaning A has r linearly independent columns. Then, $A = CR$ is a CR decomposition of A if

- C is an $n \times r$ matrix whose columns are linearly independent, and
- R is an $r \times d$ matrix whose rows are linearly independent, and
- $A = CR$

Each column of R contains the coefficients needed to write the corresponding column of A as a linear combination of the columns of C. The result is that

- the columns of C are a basis for $\text{colsp}(A)$, and
- the rows of R are a basis for $\text{rowsp}(A) = \text{colsp}(A^T)$

In the first example we saw above, we constructed C by taking the first r linearly independent columns in A to form C, and placing the coefficients needed to write each column of A as a linear combination of the columns of C in R. When constructed this way, C's columns are automatically linearly independent, and R's rows are linearly independent too (using the identity matrix and positioning of 0's argument from earlier).

A matrix has infinitely many CR decompositions, because we can choose different bases for $\text{colsp}(A)$. Both of the following are CR decompositions of our A.

```
C = np.array([[5, 2],
```

```
              [0, 1],
              [3, -1],
              [6, 4],
              [1, 1]])

R = np.array([[1, 1, 0],
              [0, -1, 1]])

C @ R

array([[ 5,  3,  2],
       [ 0, -1,  1],
       [ 3,  4, -1],
       [ 6,  2,  4],
       [ 1,  0,  1]])

# Original decomposition
C = np.array([[5, 3],
              [0, -1],
              [3, 4],
              [6, 2],
              [1, 0]])

R = np.array([[1, 0, 1],
              [0, 1, -1]])

C @ R

array([[ 5,  3,  2],
       [ 0, -1,  1],
       [ 3,  4, -1],
       [ 6,  2,  4],
       [ 1,  0,  1]])
```

You can think of $r = \text{rank}(A)$ as being the **minimum** possible number of columns in C to make $A = CR$ possible, where C is $n \times r$ and R is $r \times d$.

Here, since $\text{rank}(A) = 2$, the minimum possible number of columns in C is 2. No C with just a single column would allow for $A = CR$ to work, and while a C with 3 columns would work, 2 is the minimum number of possible columns (and if C had more than 2 columns, C's columns would no longer be linearly independent).

Finding a CR Decomposition

There are infinitely many CR decompositions of an $n \times d$ matrix A. But, if you're ever asked to find a CR decomposition of a matrix, you should:

1. Find a basis for $\text{colsp}(A)$ by running the algorithm in Chapter 4.2. This will give you the first r linearly independent columns of A; these form the columns of C.
2. Write each of the remaining $d - r$ columns of A as a linear combination of the columns of C. The coefficients for these linear combinations form the rows of R. Again, R's rows are a basis for $\text{rowsp}(A) = \text{colsp}(A^T)$.

If you construct C and R this way, the first r columns of R will be the $r \times r$ identity matrix.

What's the Point?. Why did I introduce the CR decomposition? In truth, it's not used to solve practical problems. Instead, I think it gives us a nice way to understand what the rank of a matrix really means. For one, it gave us one way to see why $\text{rank}(A) = \text{rank}(A^T)$.

Let me try and get you excited about the CR decomposition with a more practical example. Suppose we have a large 1000×50 matrix with rank $r = 8$. Storing the entire matrix requires storing $1000 \times 50 = 50000$ numbers. But, since its rank is 8, there exists a CR decomposition of A into $A = CR$ where C is 1000×8 and R is 8×50. Storing C and R requires storing

$$\underbrace{1000 \times 8}_{\text{size of } C} + \underbrace{8 \times 50}_{\text{size of } R} = 8400$$

numbers, far fewer than the 50,000 numbers required to store the entire matrix.

So, the CR decomposition is one way to **compress** a matrix; here, it relied on the fact that the rank of the matrix was much smaller than its number of columns. Hopefully this gives you some appreciation for why the rank of a matrix is such a fundamental property. Compression – of images, say – will be a recurring theme in Chapter 10, once we eventually make our way to the singular value decomposition.

Linear Transformations and Projections

6.1. Linear Transformations

Overview

The big concept introduced in Chapter 5.3 was the **rank** of a matrix, rank(A), which is the number of linearly independent columns of a matrix. The rank of a matrix told us the dimension of its column space, colsp(A), which is the set of all vectors that can be created with linear combinations of A's columns.

The ideas of rank, column space, and linear independence are all closely related, and also all apply for matrices of any shape – wide, tall, or square. This is good, because data from the real world is rarely square: we usually have many more observations (n) than features (d).

The big idea in Chapters 6.1 and 6.2 is that of the **inverse** of a matrix, A^{-1}, which **only applies to square matrices**. But, inverses and square matrices can still be useful for our rectangular matrices with real data, since the matrix A^TA is square, no matter the shape of A. And, as we discussed in Chapter 5.3 and you're seeing in Homework 6, the matrix A^TA has close ties to the correlations between the columns of A, which you already know has some connection to linear regression.

Linear Transformations

But first, I want us to think about matrix-vector multiplication as something more than just number crunching.

In Chapter 5.3, the running example was the matrix

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

To multiply A by a vector on the right, that vector must be in $\mathbb{R}^3$, and the result will be a vector in $\mathbb{R}^5$.

Put another way, if we consider the function $T(\vec{x}) = A\vec{x}$, T maps elements of $\mathbb{R}^3$ to elements of $\mathbb{R}^5$, i.e.

$$T : \mathbb{R}^3 \to \mathbb{R}^5$$

I've chosen the letter T to denote that T is a **linear transformation**.

Definition: Linear Transformation

A function $T : \mathbb{R}^d \to \mathbb{R}^n$ is a **linear transformation** if for any $\vec{x}, \vec{y} \in \mathbb{R}^d$ and any $c \in \mathbb{R}$,

1. $T(\vec{x} + \vec{y}) = T(\vec{x}) + T(\vec{y})$
2. $T(c\vec{x}) = cT(\vec{x})$

$\mathbb{R}^d$ is the **domain** of T, and $\mathbb{R}^n$ is the **codomain** of T.

Every linear transformation is of the form $T(\vec{x}) = A\vec{x}$. For our purposes, linear transformations and matrix-vector multiplication are the same thing, though in general linear transformations are a more abstract concept (just like how vector spaces can be made up of functions, for example).

For example, the function

$$f(\vec{x}) = f\left(\begin{bmatrix} x_1 \\ x_2 \end{bmatrix}\right) = \begin{bmatrix} 2x_1 + 3x_2 \\ x_1 \\ x_2 \end{bmatrix}$$

is a linear transformation from $\mathbb{R}^2$ to $\mathbb{R}^3$, and is equivalent to

$$f(\vec{x}) = \begin{bmatrix} 2x_1 + 3x_2 \\ x_1 \\ x_2 \end{bmatrix} = \underbrace{\begin{bmatrix} 2 & 3 \\ 1 & 0 \\ 0 & 1 \end{bmatrix}}_{\text{matrix } A} \underbrace{\begin{bmatrix} x_1 \\ x_2 \end{bmatrix}}_{\vec{x}}$$

The function $g(x) = 3x$ is also a linear transformation, from $\mathbb{R}$ to $\mathbb{R}$.
A non-example of a linear transformation is

$$h(\vec{x}) = \begin{bmatrix} x_1^2 \\ x_2^2 \end{bmatrix}$$

because no matrix multiplied by $\vec{x}$ will produce $\begin{bmatrix} x_1^2 \\ x_2^2 \end{bmatrix}$.
Another non-example, perhaps surprisingly, is

$$k(x) = -2x + 5$$

This is the equation of a line in $\mathbb{R}^2$, which is linear in some sense, but it's not a linear transformation, since it doesn't satisfy the two properties of linearity. For $k(x)$ to be a linear transformation, we'd need

$$k(cx) = ck(x)$$

for any $c, x \in \mathbb{R}$. But, if we consider $c = 3$ and $x = 1$ as an example, we get

$$k(cx) = k(3 \cdot 1) = k(3) = -2 \cdot 3 + 5 = -1ck(x) = 3k(1) = 3(-2 \cdot 1 + 5) = 3(3) = 9$$

which are not equal. $k(x) = -2x + 5$ is an example of an **affine transformation**, which in general is any function $f : \mathbb{R}^d \to \mathbb{R}^n$ that can be written as $f(\vec{x}) = A\vec{x} + \vec{b}$, where A is an $n \times d$ matrix and $\vec{b} \in \mathbb{R}^n$.

Activity 1

Activity 1

Activity 1.1
True or false: if T is a linear transformation, then $T(\vec{0}) = \vec{0}$.
Activity 1.2
For each of the following functions, determine whether it is a linear transformation. If it is, write it in the form $T(\vec{x}) = A\vec{x}$. If it is not, explain why not.

1. $T(\vec{x}) = \begin{bmatrix} x_1 + x_2 \\ x_1 - x_2 \end{bmatrix}$

2. $T(\vec{x}) = \begin{bmatrix} 3x_3 - x_1 \\ x_2 + 2x_3 \\ 5x_1 + 7x_2 \end{bmatrix}$
3. $T(\vec{x}) = \begin{bmatrix} 3x_3 - 5 \\ x_2 + 2x_3 \\ 5x_1 + 7x_2 \end{bmatrix}$
4. $T(\vec{x}) = \vec{x} + \begin{bmatrix} 1 \\ 2 \end{bmatrix}$
5. T takes $\vec{x}$, multiplies the first component by 2 and second component by 3, and returns the difference between the two, as a scalar.

Solutions

Activity 1.1
True or false: if T is a linear transformation, then $T(\vec{0}) = \vec{0}$.
True. Since T is a linear transformation, we have

$$T(c\vec{x}) = cT(\vec{x})$$

for any $c \in \mathbb{R}$ and $\vec{x} \in \mathbb{R}^d$. Plugging in $c = 0$, we get

$$T(0\vec{x}) = 0T(\vec{x}) \implies T(\vec{0}) = \vec{0}$$

Activity 1.2

1. $T(\vec{x}) = \begin{bmatrix} x_1 + x_2 \\ x_1 - x_2 \end{bmatrix}$ **is** a linear transformation.

$$T(\vec{x}) = \begin{bmatrix} x_1 + x_2 \\ x_1 - x_2 \end{bmatrix} = \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix}$$

2. $T(\vec{x}) = \begin{bmatrix} 3x_3 - x_1 \\ x_2 + 2x_3 \\ 5x_1 + 7x_2 \end{bmatrix}$ **is** a linear transformation.

$$T(\vec{x}) = \begin{bmatrix} 3x_3 - x_1 \\ x_2 + 2x_3 \\ 5x_1 + 7x_2 \end{bmatrix} = \begin{bmatrix} -1 & 0 & 3 \\ 0 & 1 & 2 \\ 5 & 7 & 0 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix}$$

3. $T(\vec{x}) = \begin{bmatrix} 3x_3 - 5 \\ x_2 + 2x_3 \\ 5x_1 + 7x_2 \end{bmatrix}$ **is not** a linear transformation. Notice the constant term of -5 in the first component! This is instead an affine transformation.

4. $T(\vec{x}) = \vec{x} + \begin{bmatrix} 1 \\ 2 \end{bmatrix}$ is **not** a linear transformation once again; it's an affine transformation. If you don't believe that it's not a linear transformation, plug in both $\begin{bmatrix} 1 \\ 1 \end{bmatrix}$ and $\begin{bmatrix} 2 \\ 2 \end{bmatrix}$; if T were a linear transformation, the latter output would be exactly double the former output.

5. This description tells us that $T(\vec{x}) = 2x_1 - 3x_2$, and this **is** a perfectly valid linear transformation, from $\mathbb{R}^2$ to $\mathbb{R}$.

$$T(\vec{x}) = \begin{bmatrix} 2 & -3 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix}$$

Linear Transformations from $\mathbb{R}^n$ to $\mathbb{R}^n$

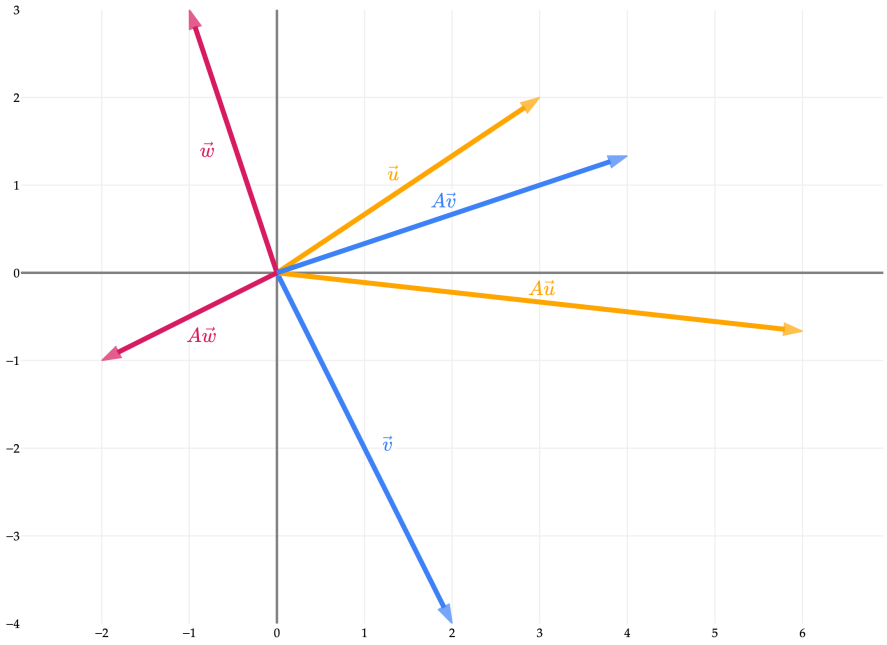
While linear transformations exist from $\mathbb{R}^2$ to $\mathbb{R}^5$ or $\mathbb{R}^{99}$ to $\mathbb{R}^4$, it's in some ways easiest to think about linear transformations with the same domain and codomain, i.e. transformations of the form $T : \mathbb{R}^n \to \mathbb{R}^n$. This will

allow us to explore how transformations stretch, rotate, and reflect vectors in the same space. Linear transformations with the same domain ($\mathbb{R}^n$) and codomain ($\mathbb{R}^n$) are represented by $n \times n$ matrices, which gives us a useful setting to think about the invertibility of square matrices, beyond just looking at a bunch of numbers.

To start, let's consider the linear transformation defined by the matrix

$$A = \begin{bmatrix} 2 & 0 \\ 0 & -1/3 \end{bmatrix}$$

What happens to a vector in $\mathbb{R}^2$ when we multiply it by A? Let's visualize the effect of A on several vectors in $\mathbb{R}^2$.



A scales, or stretches, the input space by a factor of 2 in the x-direction and a factor of $-1/3$ in the y-direction.

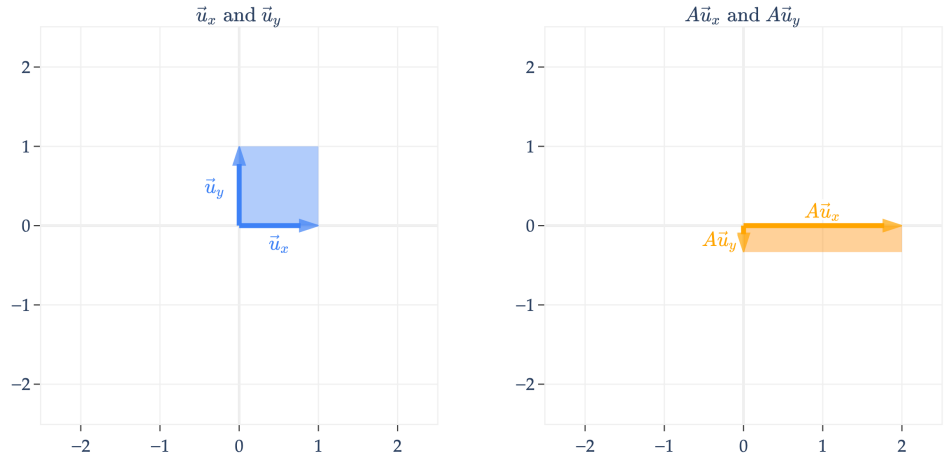
Scaling. Another way of visualizing A is to think about how it transforms the two standard basis vectors of $\mathbb{R}^2$, which are

$$\vec{u}_x = \begin{bmatrix} 1 \\ 0 \end{bmatrix}, \quad \vec{u}_y = \begin{bmatrix} 0 \\ 1 \end{bmatrix}$$

(In the past I've called these $\vec{e}_1$ and $\vec{e}_2$, but I'll use $\vec{u}_x$ and $\vec{u}_y$ here since I'll also use E to represent a matrix shortly.)

Note that $A\vec{u}_x = \begin{bmatrix} 2 \\ 0 \end{bmatrix}$ is just the first column of A, and similarly $A\vec{u}_y = \begin{bmatrix} 0 \\ -1/3 \end{bmatrix}$ is the second column of A.

$$A = \begin{bmatrix} 2 & 0 \\ 0 & -1/3 \end{bmatrix}$$



In addition to drawing $\vec{u}_x$ and $\vec{u}_y$ on the left and their transformed counterparts $A\vec{u}_x$ and $A\vec{u}_y$ on the right, I've also shaded in how the **unit square**, which is the square containing $\vec{u}_x$ and $\vec{u}_y$, gets transformed. Here, it gets stretched from a square to a rectangle.

Remember that any vector $\vec{v} \in \mathbb{R}^2$ is a linear combination of $\vec{u}_x$ and $\vec{u}_y$. For instance,

$$\begin{bmatrix} 2 \\ -1 \end{bmatrix} = 2\begin{bmatrix} 1 \\ 0 \end{bmatrix} - \begin{bmatrix} 0 \\ 1 \end{bmatrix} = \underbrace{2\vec{u}_x - \vec{u}_y}_{\vec{v}}$$

So, multiplying A by $\vec{v}$ is equivalent to multiplying A by a linear combination of $\vec{u}_x$ and $\vec{u}_y$.

$$A\begin{bmatrix} 2 \\ -1 \end{bmatrix} = A(2\vec{u}_x - \vec{u}_y) = 2A\vec{u}_x - A\vec{u}_y$$

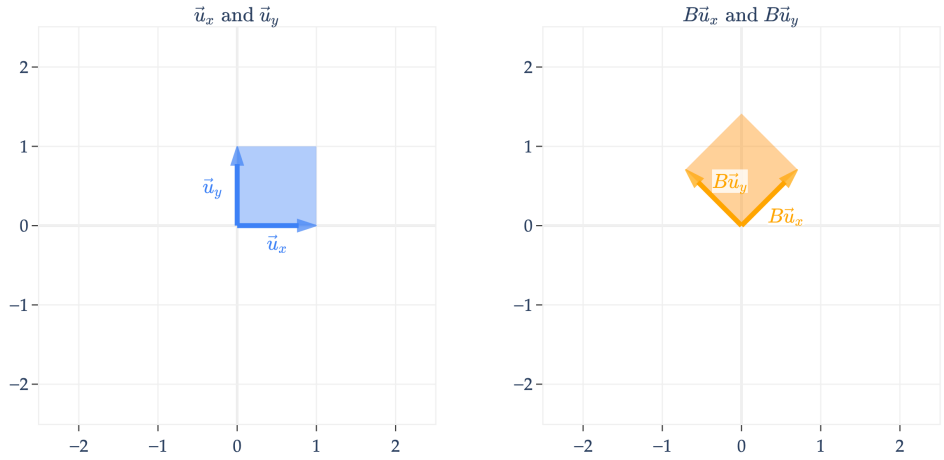
and the result is a linear combination of $A\vec{u}_x$ and $A\vec{u}_y$ with the same coefficients! **If this sounds confusing, just remember that $A\vec{u}_x$ and $A\vec{u}_y$ are just the first and second columns of A, respectively.**

So, as we move through the following examples, think of the transformed basis vectors $A\vec{u}_x$ and $A\vec{u}_y$ as a new set of "building blocks" that define the transformed space (which is the column space of A).

A is a diagonal matrix, which means it **scales** vectors. Note that **any** vector in $\mathbb{R}^2$ can be transformed by A, not just vectors on or within the unit square; I'm just using these two basis vectors to visualize the transformation.

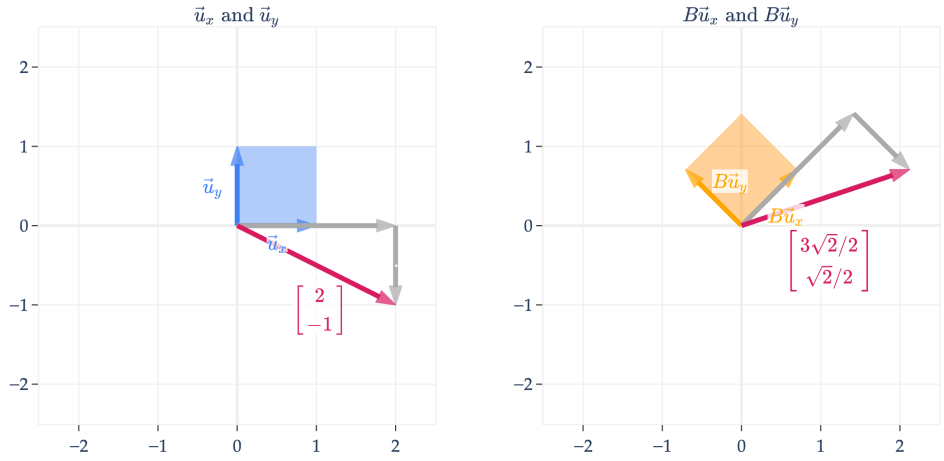
Rotations and Orthogonal Matrices. What might a non-diagonal matrix do? Let's consider

$$B = \begin{bmatrix} \sqrt{2}/2 & -\sqrt{2}/2 \\ \sqrt{2}/2 & \sqrt{2}/2 \end{bmatrix}$$



Just to continue the previous example, the vector $\begin{bmatrix} 2 \\ -1 \end{bmatrix}$ is transformed into

$$B\begin{bmatrix} 2 \\ -1 \end{bmatrix} = \begin{bmatrix} \sqrt{2}/2 & -\sqrt{2}/2 \\ \sqrt{2}/2 & \sqrt{2}/2 \end{bmatrix}\begin{bmatrix} 2 \\ -1 \end{bmatrix} = 2\underbrace{\begin{bmatrix} \sqrt{2}/2 \\ \sqrt{2}/2 \end{bmatrix}}_{B\vec{u}_x} - \underbrace{\begin{bmatrix} -\sqrt{2}/2 \\ \sqrt{2}/2 \end{bmatrix}}_{B\vec{u}_y} = \begin{bmatrix} 3\sqrt{2}/2 \\ \sqrt{2}/2 \end{bmatrix}$$



B is an **orthogonal matrix**, which means that its columns are unit vectors and are orthogonal to one another.

$$\underbrace{B^TB = BB^T = I}_{\text{condition for an orthogonal matrix}}$$

Orthogonal matrices **rotate** vectors in the input space. In general, a matrix that rotates vectors by θ (radians) counterclockwise in $\mathbb{R}^2$ is given by

$$R(\theta) = \begin{bmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{bmatrix}$$

$B = R(\frac{\pi}{4})$ rotates vectors by $\frac{\pi}{4}$ radians, i.e. $45°$.

Rotations are more difficult to visualize in $\mathbb{R}^3$ and higher dimensions, but in Homework 5, you'll prove that orthogonal matrices preserve **norms**, i.e. if Q is an $n \times n$ orthogonal matrix and $x \in \mathbb{R}^n$, then $\|Qx\| = \|x\|$. So, even though an orthogonal matrix might be doing something harder to describe in $\mathbb{R}^{15}$, we know that it isn't

changing the lengths of the vectors it's multiplying.

To drive home the point I made earlier, any vector $\vec{v} = \begin{bmatrix} x \\ y \end{bmatrix}$, once multiplied by B, ends up transforming into

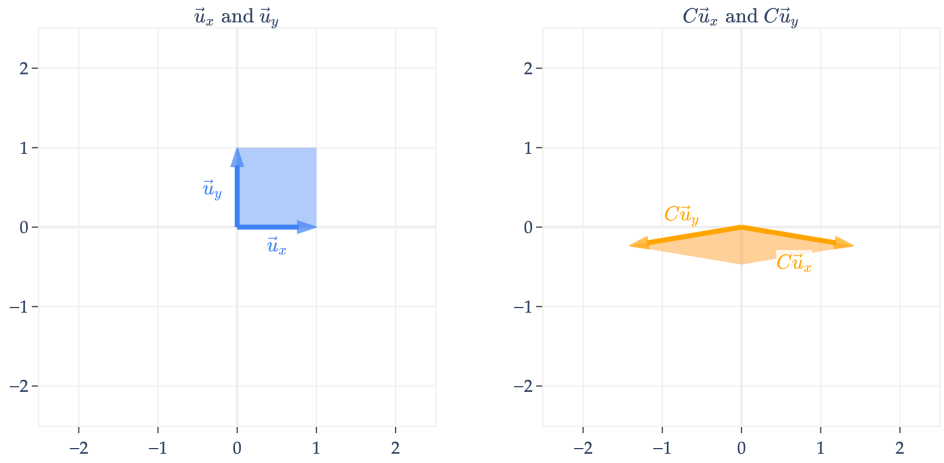
$$B \underbrace{\begin{bmatrix} x \\ y \end{bmatrix}}_{\vec{v}} = x(B\vec{u}_x) + y(B\vec{u}_y)$$

Composing Transformations. We can even apply multiple transformations one after another. This is called **composing** transformations. For instance,

$$C = \begin{bmatrix} \sqrt{2} & -\sqrt{2} \\ -\sqrt{2}/6 & -\sqrt{2}/6 \end{bmatrix}$$

is just

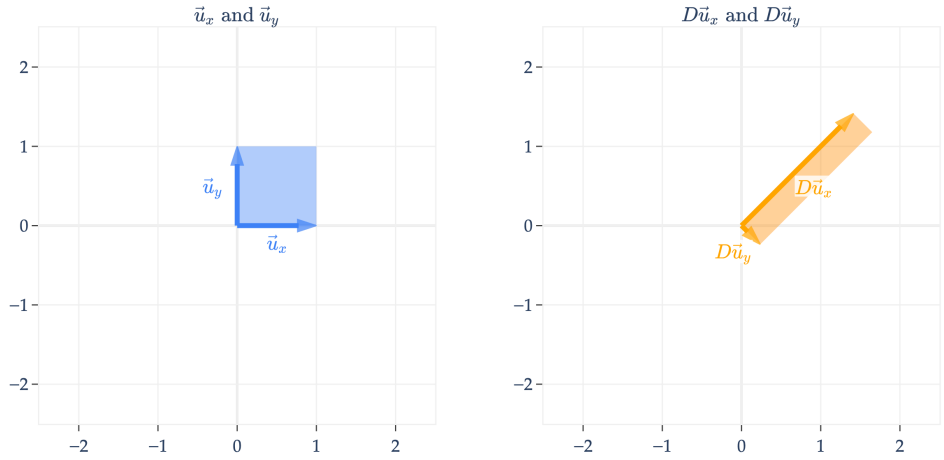
$$C = AB = \underbrace{\begin{bmatrix} 2 & 0 \\ 0 & -1/3 \end{bmatrix}}_{\text{scale}} \underbrace{\begin{bmatrix} \sqrt{2}/2 & -\sqrt{2}/2 \\ \sqrt{2}/2 & \sqrt{2}/2 \end{bmatrix}}_{\text{rotate}}$$



Note that C rotates the input vector, **and then** scales it. Read the operations from right to left, since $C\vec{x} = AB\vec{x} = A(B\vec{x})$.

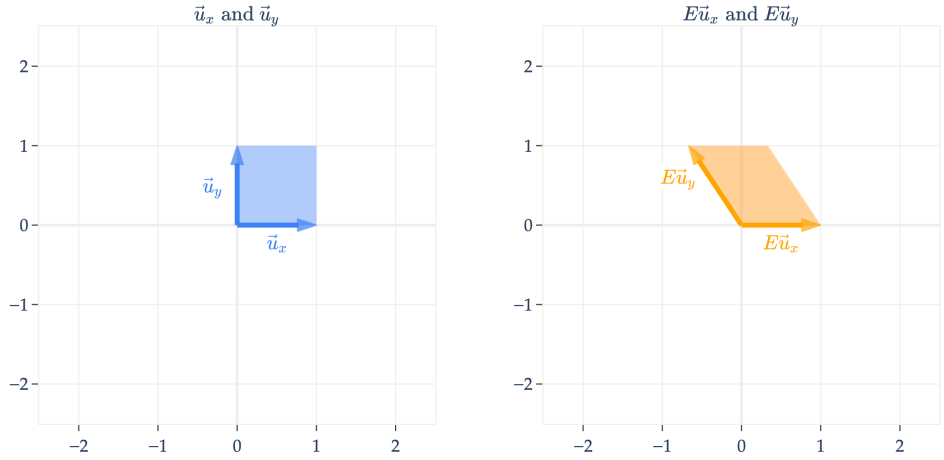
$C = AB$ is different from

$$D = \begin{bmatrix} \sqrt{2} & \sqrt{2}/6 \\ \sqrt{2} & -\sqrt{2}/6 \end{bmatrix} = BA$$



Shears.

$$E = \begin{bmatrix} 1 & -2/3 \\ 0 & 1 \end{bmatrix}$$



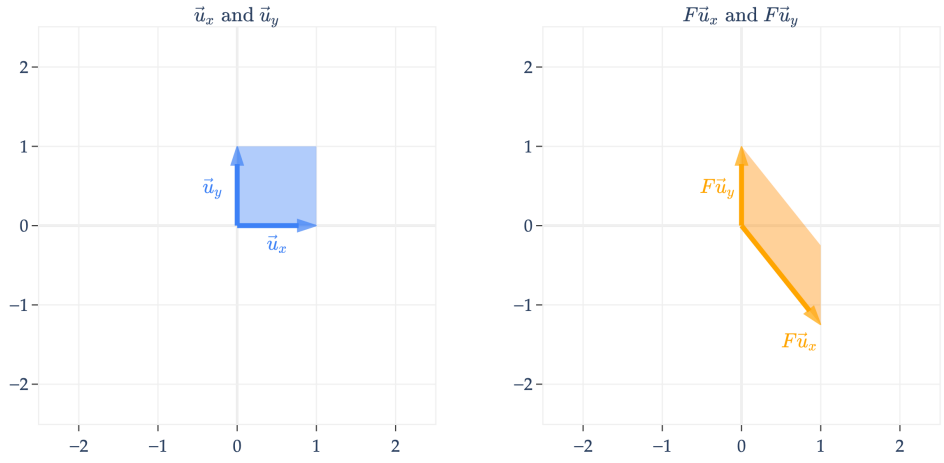
E is a **shear matrix**. Think of a shear as a transformation that slants the input space along one axis, while keeping the other axis fixed. What helps me interpret shears is looking at them formulaically.

$$E\begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} 1 & -2/3 \\ 0 & 1 \end{bmatrix}\begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} x - 2/3y \\ y \end{bmatrix}$$

Note that the y-coordinate of input vectors in $\mathbb{R}^2$ remain unchanged, while the x-coordinate is shifted by $-\frac{2}{3}y$, which results in a slanted shape.

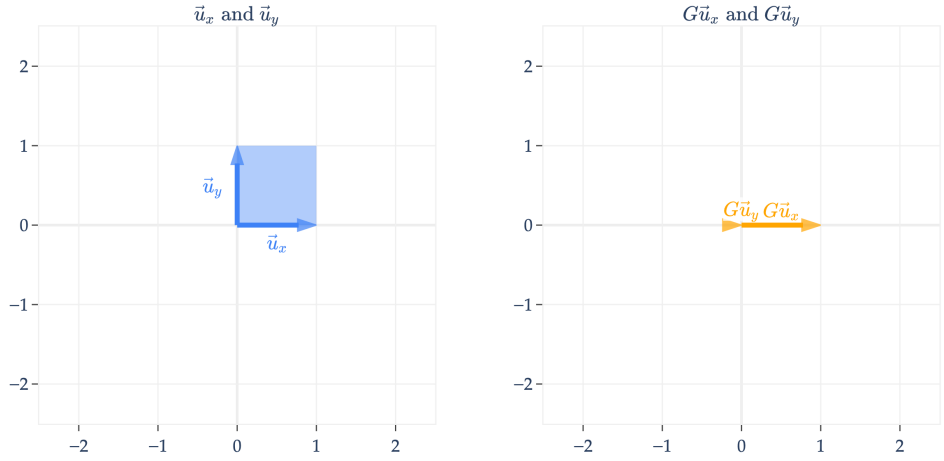
Similarly, F is a shear matrix that keeps the x-coordinate fixed, but shifts the y-coordinate, resulting in a slanted shape that is tilted downwards.

$$F = \begin{bmatrix} 1 & 0 \\ -5/4 & 1 \end{bmatrix}$$



Non-Invertible Transformations. So far we've looked at **scaling**, **rotation**, and **shear** matrices. Let's look at another type of transformation that behaves differently from what we've seen so far. Consider the matrix G below.

$$G = \begin{bmatrix} 1 & 0 \\ 0 & 0 \end{bmatrix}$$

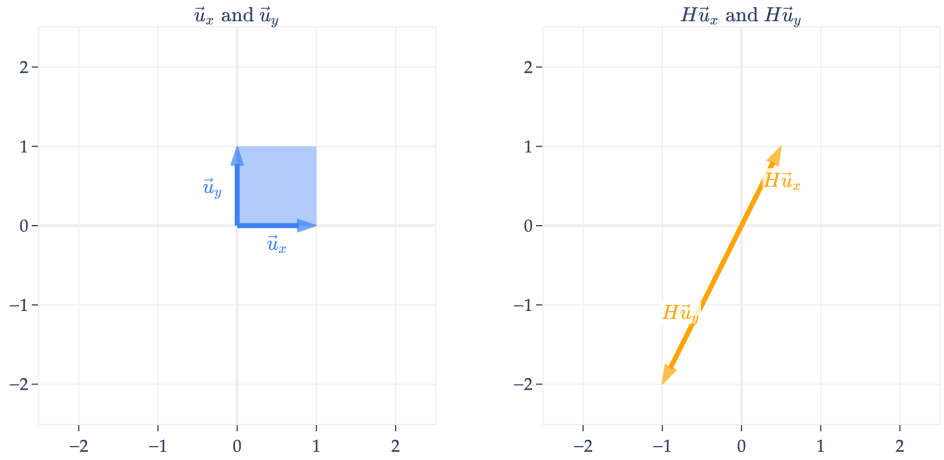


$G\vec{x}$ keeps the horizontal component of $\vec{x}$ and throws away the vertical component. Note that G maps the unit square to a **line**, not another four-sided shape.

Why have I called this a non-invertible transformation? This is the key idea that we're building towards: given that $G\vec{x} = \begin{bmatrix} 2 \\ 0 \end{bmatrix}$, **I have no way of knowing what $\vec{x}$ is!** It could be that $\vec{x} = \begin{bmatrix} 2 \\ 5 \end{bmatrix}$, or $\vec{x} = \begin{bmatrix} 2 \\ -1 \end{bmatrix}$, or $\vec{x} = \begin{bmatrix} 2 \\ 100 \end{bmatrix}$, etc. These $\vec{x}$'s all get mapped to the same vector, $\begin{bmatrix} 2 \\ 0 \end{bmatrix}$, by G. This is because, unlike the matrices we've seen so far, colsp(G) is **not** all of $\mathbb{R}^2$, but rather it's just a line in $\mathbb{R}^2$, since G's columns are not linearly independent.

H below works similarly.

$$H = \begin{bmatrix} 1/2 & -1 \\ 1 & -2 \end{bmatrix}$$



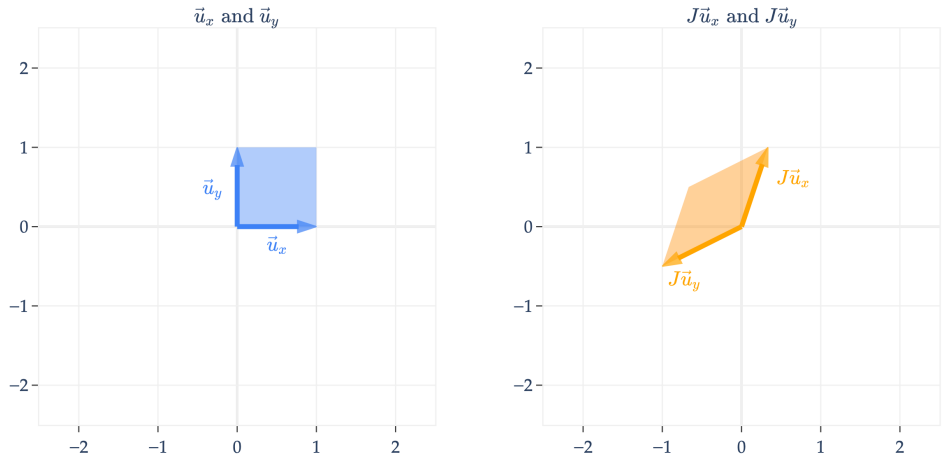
colsp(H) is the line spanned by $\begin{bmatrix} 1/2 \\ 1 \end{bmatrix}$, so $H\vec{x}$ will always be some vector on this line. Put another way, if $\vec{v} = \begin{bmatrix} x \\ y \end{bmatrix}$, then $H\vec{v}$ is

$$H \underbrace{\begin{bmatrix} x \\ y \end{bmatrix}}_{\vec{v}} = x(H\vec{u}_x) + y(H\vec{u}_y)$$

but since $H\vec{u}_x$ and $H\vec{u}_y$ are both on the line spanned by $\begin{bmatrix} 1/2 \\ 1 \end{bmatrix}$, $H\vec{v}$ is really just a scalar multiple of $\begin{bmatrix} 1/2 \\ 1 \end{bmatrix}$.

Arbitrary Matrices. Finally, I'll comment that not all linear transformations have a nice, intuitive interpretation. For instance, consider

$$J = \begin{bmatrix} 1/3 & -1 \\ 1 & -1/2 \end{bmatrix}$$



J turns the unit square into a **parallelogram**. In fact, so did A, B, C, D, E, and F; all of these transformations map the unit square to a parallelogram, with some additional properties (e.g. A's parallelogram was a rectangle, B's had equal sides, etc.).

There's no need to memorize the names of these transformations – after all, they only apply in $\mathbb{R}^2$ and perhaps $\mathbb{R}^3$ where we can visualize.

Speaking of $\mathbb{R}^3$, an arbitrary 3×3 matrix can be thought of as a transformation that maps the unit cube to a parallelepiped (the generalization of a parallelogram to three dimensions).

$$K = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 2 & 1/2 \\ 0 & -1 & 1/2 \end{bmatrix}$$

What do you notice about the transformation defined by L, and how it relates to L's columns? (Drag the plot around to see the main point.)

$$L = \begin{bmatrix} 1 & 1/2 & 0 \\ 1 & 1/2 & 0 \\ 1 & 1/2 & 1 \end{bmatrix}$$

Since L's columns are linearly **dependent**, L maps the unit cube to a flat parallelo**gram**.

The Determinant

It turns out that there's a formula for

- **area of the parallelogram** formed by transforming the unit square by a 2×2 matrix A
- **volume of the parallelepiped** formed by transforming the unit cube by a 3×3 matrix A
- in general, the n-dimensional "volume" of the object formed by transforming the unit n-cube by an $n \times n$ matrix A

That formula is called the **determinant** of A, and is denoted $\det(A)$.

Why do we care? Remember, the goal of this section is to find the inverse of a square matrix A, if it exists, and the determinant will give us one way to check if it does.

In the case of the matrices G and H above, their columns were linearly dependent, and so the transformations G and H mapped the unit square to a **line with no area**. Similarly above, L mapped the unit cube to a flat parallelogram with **no volume**. In all other transformations, the matrices' columns were linearly independent, so the resulting object had a non-zero area (in the case of 2×2 matrices) or volume (in the case of 3×3 matrices).

If $\det(A) = 0$, A's columns are linearly dependent!

Equivalently, if $\det(A) \neq 0$, A's columns are linearly **independent**. This will help us check if A is invertible in just a moment.

So, how do we find $\det(A)$? Unfortunately, the formula is only convenient for 2×2 matrices.

Definition: Determinant of a 2×2 matrix

If $A = \begin{bmatrix} a & b \\ c & d \end{bmatrix}$, its determinant is

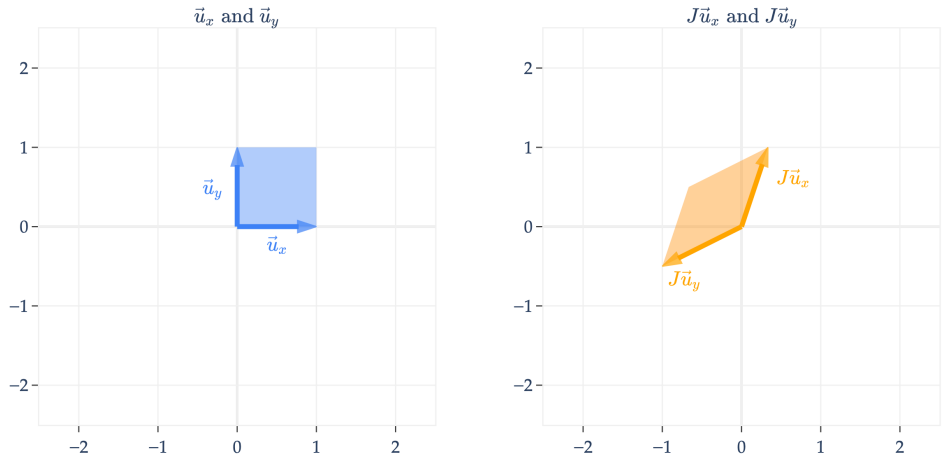
$$\det(A) = \begin{vmatrix} a & b \\ c & d \end{vmatrix} = ad - bc$$

For example, in the transformation

$$J = \begin{bmatrix} 1/3 & -1 \\ 1 & -1/2 \end{bmatrix}$$

the area of the parallelogram formed by transforming the unit square is

$$\det(J) = \frac{1}{3}\left(-\frac{1}{2}\right) - (-1)(1) = -\frac{1}{6} + 1 = \frac{5}{6}$$



Note that a determinant **can be negative**! So, to be precise, $|\det(A)|$ describes the n-dimensional volume of the object formed by transforming the unit n-cube by A.

The sign of the determinant depends on the order of the columns of the matrix. For example, swap the columns of J and its determinant would be $-\frac{5}{6}$. (If $A\vec{u}_x$ is "to the right" of $A\vec{u}_y$, the determinant is positive, like with the standard basis vectors; if $A\vec{u}_x$ is "to the left" of $A\vec{u}_y$, the determinant is negative.)

The determinant of an $n \times n$ matrix can be expressed recursively using a weighted sum of determinants of smaller $n-1 \times n-1$ matrices, called minors. For example, if A is the 3×3 matrix

$$A = \begin{bmatrix} a & b & c \\ d & e & f \\ g & h & i \end{bmatrix}$$

then $\det(A)$ is

$$\det(A) = a\begin{vmatrix} e & f \\ h & i \end{vmatrix} - b\begin{vmatrix} d & f \\ g & i \end{vmatrix} + c\begin{vmatrix} d & e \\ g & h \end{vmatrix}$$

The matrix $\begin{bmatrix} e & f \\ h & i \end{bmatrix}$ is a minor of A; it's formed by deleting the first row and first column of A. Note the alternating signs in the formula. This formula generalizes to $n \times n$ matrices, but is not practical for anything larger than 3×3. (What is the runtime of this algorithm?)

The computation of the determinant is not super important. **The big idea is that the determinant of A is a single number that tells us whether A's transformation "loses a dimension" or not.**

Some useful properties of the determinant are that, for any $n \times n$ matrices A and B,

1. $\det(A) = \det(A^T)$
2. $\det(AB) = \det(A)\det(B)$

3. $\det(cA) = c^n \det(A)$ (notice the exponent!)
4. If B results from swapping two of A's columns (or rows), then

$$\det(B) = -\det(A)$$

The proofs of these properties are beyond the scope of our course. But, it's worthwhile to think about what they mean in English in the context of linear transformations.

1. $\det(A) = \det(A^T)$ implies that the rows of A (which are the columns of A^T) create the same "volume" as the columns of A.
2. $\det(AB) = \det(A)\det(B)$ matches our intuition that linear transformations can be composed. $AB\vec{x}$ is the result of applying B to $\vec{x}$, then A to the result.
3. $\det(cA) = c^n \det(A)$ multiplies each column of A by c, and so the volume of the resulting object is scaled by $c \times c \times \cdots \times c = c^n$.
4. If B results from swapping two of A's columns (or rows), then $\det(B) = -\det(A)$ reflects that swapping two columns reverses the orientation of the transformation, flipping the signed volume from positive to negative (or vice versa) while preserving its magnitude.

This YouTube playlist contains walkthrough videos of the following activity, which has 5 subparts. The first video, embedded below, contains a brief overview of determinants, summarizing the content above.

Activity 2

Activity 2

Activity 2.1
Find the determinant of the following matrices:

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

Activity 2.2
Suppose A and B are both $n \times n$ matrices. Is $\det(A + B) = \det(A) + \det(B)$ in general?
Activity 2.3
Suppose we multiply A's 2nd column by 3. What happens to $\det(A)$?
Activity 2.4
If A's columns are linearly dependent, then find $\det(AB)$.
Activity 2.5

1. Find the determinant of $R(\theta) = \begin{bmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{bmatrix}$ (Hint: The answer does not depend on θ!).

2. $R(\theta)$ is a 2×2 orthogonal matrix. If Q is an $n \times n$ orthogonal matrix, then what is $\det(Q)$?

Now that we have a better understanding of the linear transformations that are implicitly defined by square matrices, we can return to the question of inverses, when they exist, and how to find them.

6.2. Inverses

Chapter 6.1 was a bit of a detour into the world of linear transformations. In theory, one could read Chapter 6.2 here without that knowledge, but the knowledge of linear transformations (in my mind) will help you better appreciate (and understand) the concept of a matrix inverse.

What is an Inverse, Really?

Scalar Addition and Multiplication. In "regular" addition and multiplication, you're already familiar with the idea of an inverse. In addition, the inverse of a number a is another number a' that satisfies

$$a + a' = a' + a = 0$$

0 is the "identity" element in addition, since adding it to any number a doesn't change the value of a. Of course, $a' = -a$, so $-a$ is the **additive inverse** of a. Any number a has an additive inverse; for example, the additive inverse of 2 is -2, and the additive inverse of -2 is 2.

In multiplication, the inverse of a number a is another number a' that satisfies

$$a \cdot a' = a' \cdot a = 1$$

1 is the "identity" element in multiplication, since multiplying it by any number a doesn't change the value of a. Most numbers have a **multiplicative inverse** of $a' = \frac{1}{a}$, but 0 does not!

$$0 \cdot a' = 0$$

is not achieved by any a'.

When a multiplicative inverse exists, we can use it to solve equations like

$$2x = 5$$

by multiplying both sides by the multiplicative inverse of 2, which is $\frac{1}{2}$.

$$\frac{1}{2}(2x) = \frac{1}{2}(5) \implies x = \frac{5}{2}$$

Matrices. If A is an $n \times d$ matrix, its additive inverse is just $-A$, which comes from negating each element of A. That part is not all that interesting. What we're more interested in is the **multiplicative inverse** of a matrix, which is just referred to as the **inverse** of the matrix.

Suppose A is some $n \times d$ matrix. We'd like to find an inverse, A', such that when A is multiplied by A' **in any order**, the result is the identity element for matrix multiplication, I. Recall, the $n \times n$ identity matrix is a matrix with 1s on the diagonal (moving from the top left to the bottom right), and 0s everywhere else. The 3×3 identity matrix is

$$I_3 = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

$I_3\vec{x} = \vec{x}$ for any $\vec{x} \in \mathbb{R}^3$, and $BI_3 = I_3B = B$ for any $B \in \mathbb{R}^{3\times 3}$.

$$\begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix}$$

If A is an $n \times d$ matrix, we'd need a single matrix A' that satisfies

$$AA' = A'A = I$$

In AA', A' is the **right inverse** of A, and in $A'A$, A' is the **left inverse** of A. Right now, we're interested in finding matrices that have both a left and right inverse, that happen to be the same matrix.

In order to evaluate AA', we'd need A' to be of shape $d \times$ something, and in order to evaluate $A'A$, we'd need A' to be of shape something else $\times\, n$. The solution to these constraints is to have A' be a $d \times n$ matrix (like A^T).

If we try that, then

- $A'A$ has shape $(d \times n) \cdot (n \times d) = d \times d$
- AA' has shape $(n \times d) \cdot (d \times n) = n \times n$

but, we want $A'A$ and AA' to both be **the same matrix**, not just separate valid matrices. So, we need $n = d$, which requires A to be $n \times n$, like its inverse.

Only square matrices have inverses!

The idea of invertibility only makes sense for square matrices.
Non-square matrices can have left or right inverses, which we'll discuss in Chapter 6.4, but in general they can't be "invertible".

For example, consider the 2×2 matrix

$$A = \begin{bmatrix} 2 & 4 \\ 3 & 5 \end{bmatrix}$$

It turns out that A does have an inverse! Its inverse, denoted by A^{-1}, is

$$A^{-1} = \begin{bmatrix} -5/2 & 2 \\ 3/2 & -1 \end{bmatrix}$$

Because A^{-1} is the matrix that satisfies **both**

$$AA^{-1} = \begin{bmatrix} 2 & 4 \\ 3 & 5 \end{bmatrix} \begin{bmatrix} -5/2 & 2 \\ 3/2 & -1 \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} = I$$

and

$$A^{-1}A = \begin{bmatrix} -5/2 & 2 \\ 3/2 & -1 \end{bmatrix} \begin{bmatrix} 2 & 4 \\ 3 & 5 \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} = I$$

`numpy` is good at finding inverses, when they exist.

```
import numpy as np

A = np.array([[2, 4],
              [3, 5]])

np.linalg.inv(A)

array([[ -2.5,  2. ],
       [ 1.5, -1. ]])

# The top -left and bottom -right elements are 1s,
# and the top -right and bottom -left elements are 0s.
# 4.4408921e -16 is just 0, with some floating point error baked in.
A @ np.linalg.inv(A)

array([[1.0000000e+00, 0.0000000e+00],
       [4.4408921e -16, 1.0000000e+00]])
```

But, not all square matrices are invertible, just like not all numbers have multiplicative inverses. What happens if we try and invert

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

```
B = np.array([[1, 2],
              [2, 4]])

np.linalg.inv(B)

- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
LinAlgError                               Traceback (most recent call last)
Cell In[17], line 4
      1 B = np.array([[1, 2],
      2               [2, 4]])
 - - - -> 4 np.linalg.inv(B)

File ~/miniforge3/envs/pds/lib/python3.10/site -packages/numpy/linalg/linalg.py:561, in inv(a)
    559 signature = 'D ->D' if isComplexType(t) else 'd ->d'
    560 extobj = get_linalg_error_extobj(_raise_linalgerror_singular)
 - -> 561 ainv = _umath_linalg.inv(a, signature=signature, extobj=extobj)
    562 return wrap(ainv.astype(result_t, copy=False))

File ~/miniforge3/envs/pds/lib/python3.10/site -packages/numpy/linalg/linalg.py:112, in _raise_linal
    111 def _raise_linalgerror_singular(err, flag):
```

```
 - -> 112     raise LinAlgError("Singular matrix")

LinAlgError: Singular matrix
```

We're told the matrix is **singular**, meaning that it's not invertible.

The point of this section is to understand **why** some square matrices are invertible and others are not, and what the inverse of an invertible matrix really means.

The Goal: Solving Systems of Equations

In general, suppose A is an $n \times d$ matrix and $\vec{b} \in \mathbb{R}^n$. Then, the equation

$$A\vec{x} = \vec{b}$$

is a system of n equations in d unknowns.

$$\underbrace{\begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1d} \\ a_{21} & a_{22} & \cdots & a_{2d} \\ \vdots & \vdots & \ddots & \vdots \\ a_{n1} & a_{n2} & \cdots & a_{nd} \end{bmatrix}}_{A} \underbrace{\begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_d \end{bmatrix}}_{\vec{x}} = \underbrace{\begin{bmatrix} b_1 \\ b_2 \\ \vdots \\ b_n \end{bmatrix}}_{\vec{b}}$$

There may be no solution, a unique solution, or infinitely many solutions for the vector $\vec{x}$ that satisfies the system, depending on $\text{rank}(A)$ and whether $\vec{b}$ is in $\text{colsp}(A)$.

If we assume A is square, i.e. $n = d$, then $A\vec{x} = \vec{b}$ is a system of n equations in n unknowns. If A is invertible (which, remember, only square matrices can be), then there is a unique solution to the system, and we can find it by multiplying both sides by A^{-1} on the left.

$$A\vec{x} = \vec{b} \implies A^{-1}A\vec{x} = A^{-1}\vec{b} \implies \vec{x} = A^{-1}\vec{b}$$

$\vec{x} = A^{-1}\vec{b}$ is the unique solution to the system of equations, and we can find it without having to manually solve the system. Thinking back to the example above, we used `numpy` to find that the inverse of

$$A = \begin{bmatrix} 2 & 4 \\ 3 & 5 \end{bmatrix}$$

is

$$A^{-1} = \begin{bmatrix} -5/2 & 2 \\ 3/2 & -1 \end{bmatrix}$$

We can use A^{-1} to solve the system

$$\begin{aligned} 2x_1 + 4x_2 &= b_1 \\ 3x_1 + 5x_2 &= b_2 \end{aligned}$$

For any $b_1, b_2 \in \mathbb{R}$, the unique solution for x_1 and x_2 is

$$\begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} -5/2 & 2 \\ 3/2 & -1 \end{bmatrix} \begin{bmatrix} b_1 \\ b_2 \end{bmatrix} = \begin{bmatrix} -5/2b_1 + 2b_2 \\ 3/2b_1 - b_2 \end{bmatrix}$$

That said, as we'll discuss at the bottom of this section, actually finding the inverse of a matrix is very computationally intensive, so usually we don't actually compute A^{-1}. Knowing that it exists, and understanding the properties it satisfies, is the important part.

Inverting a Transformation

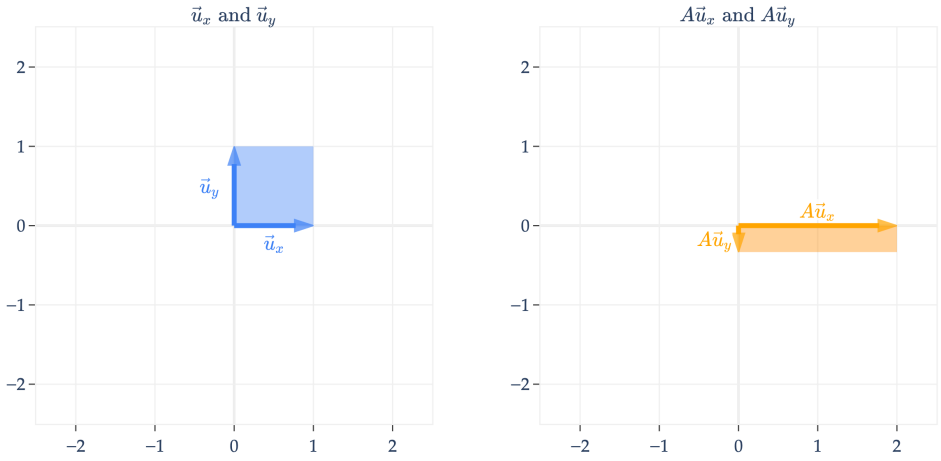
Remember, the big goal of this section is to find the inverse of a square matrix A.

Since a square matrix A corresponds to a linear transformation, we can think of A^{-1} as "reversing" or "undoing" the transformation.

For example, if A scales vectors, A^{-1} should scale by the reciprocal, so that applying A and then A^{-1} returns the original vector.

The simplest case involves a diagonal matrix, like

$$A = \begin{bmatrix} 2 & 0 \\ 0 & -1/3 \end{bmatrix}$$



To undo the effect of A, we can apply the transformation

$$A^{-1} = \begin{bmatrix} 1/2 & 0 \\ 0 & -3 \end{bmatrix}$$

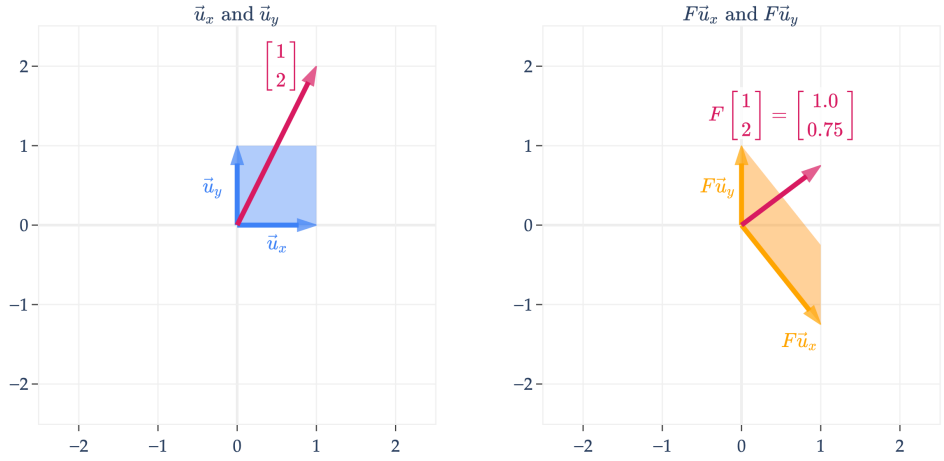
Many of the transformations we looked at involving 2×2 matrices are reversible, and hence the corresponding matrices are invertible. If the matrix rotates by θ, the inverse is a rotation by $-\theta$. If the matrix shears by "dragging to the right", the inverse is a shear by "dragging to the left".

Another way of visualizing whether a transformation is reversible is whether given a vector on the right, there is exactly one corresponding vector on the left. By exactly one, I mean not 0, and not multiple.

Here, we visualize

$$F = \begin{bmatrix} 1 & 0 \\ -5/4 & 1 \end{bmatrix}$$

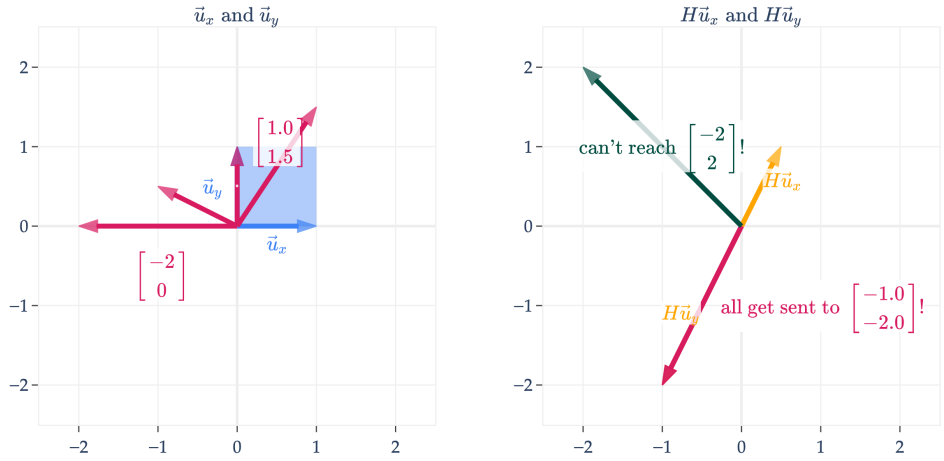
Given any vector $\vec{b}$ of the form $\vec{b} = F\vec{x}$, there is **exactly one** vector $\vec{x}$ that satisfies this equation.



On the other hand, if we look at

$$H = \begin{bmatrix} 1/2 & -1 \\ 1 & -2 \end{bmatrix}$$

the same does not hold true. Given any vector $\vec{b} \in \text{colsp}(H)$ on the right, there are infinitely many vectors $\vec{x}$ such that $H\vec{x} = \vec{b}$. The vectors in pink on the left are all sent to the same vector on the right, $\begin{bmatrix} -1 \\ -2 \end{bmatrix}$.



And, there are no vectors on the left that get sent to $\begin{bmatrix} -2 \\ 2 \end{bmatrix}$ on the right. Any vector in $\mathbb{R}^2$ that isn't on the line spanned by $\begin{bmatrix} -1 \\ -2 \end{bmatrix}$ is unreachable.

You may recall the following key ideas from discrete math:

- A function is **invertible** if and only if it is both **one-to-one** (injective) and **onto** (surjective).

- A function is **one-to-one** if and only if no two inputs get sent to the same output, i.e. $f(x_1) = f(x_2)$ implies $x_1 = x_2$.
- A function is **onto** if every element of the codomain is an output of the function, i.e. for every $y \in Y$, there exists an $x \in X$ such that $f(x) = y$.

The transformation represented by H is neither one-to-one (because $\begin{bmatrix}0\\1\end{bmatrix}$ and $\begin{bmatrix}1\\1.5\end{bmatrix}$ get sent to the same point) nor onto (because $\begin{bmatrix}-2\\0\end{bmatrix}$ isn't mapped to by any vector in $\mathbb{R}^2$).

In order for a linear transformation to be invertible, it must be **both one-to-one and onto**, i.e. it must be a **bijection**. Again, don't worry if these terms seem foreign: I've provided them here to help build connections to other courses *if* you've taken them. If not, the rest of my coverage should still be sufficient.

Inverting a Matrix

The big idea I'm trying to get across is that an $n \times n$ matrix A is invertible if and only if the corresponding linear transformation can be "undone". In other words:

What makes A invertible?

A is invertible if and only if given any vector $\vec{b} \in \mathbb{R}^n$, **there is** exactly one **vector** $\vec{x} \in \mathbb{R}^n$ **such that** $A\vec{x} = \vec{b}$.

If the visual intuition from earlier didn't make this clear, here's another concrete example. Consider

$$A = \begin{bmatrix}1 & 2 & 0\\1 & 2 & 0\\1 & 2 & 1\end{bmatrix}$$

$\text{rank}(A) = 2$, since A's first two columns are scalar multiples of one another. Let's consider two possible $\vec{b}$'s, each depicting a different case.

1. $\vec{b} = \begin{bmatrix}3\\3\\3\end{bmatrix}$. This $\vec{b}$ is in $\text{colsp}(A)$. The issue with A is that there are infinitely many linear combinations of the columns of A that equal this $\vec{b}$.

$$\underbrace{\begin{bmatrix}1 & 2 & 0\\1 & 2 & 0\\1 & 2 & 1\end{bmatrix}}_{A}\underbrace{\begin{bmatrix}1\\1\\0\end{bmatrix}}_{\text{an } \vec{x}} = \underbrace{\begin{bmatrix}1 & 2 & 0\\1 & 2 & 0\\1 & 2 & 1\end{bmatrix}}_{A}\underbrace{\begin{bmatrix}-5\\4\\0\end{bmatrix}}_{\text{another } \vec{x}} = \ldots = \underbrace{\begin{bmatrix}3\\3\\3\end{bmatrix}}_{\vec{b}}$$

2. $\vec{b} = \begin{bmatrix}1\\0\\0\end{bmatrix}$. This $\vec{b}$ is **not** in $\text{colsp}(A)$, meaning there is no $\vec{x} \in \mathbb{R}^3$ such that $A\vec{x} = \vec{b}$.

$$\underbrace{\begin{bmatrix}1 & 2 & 0\\ 1 & 2 & 0\\ 1 & 2 & 1\end{bmatrix}}_{A}\ \underbrace{\begin{bmatrix}x_1\\ x_2\\ x_3\end{bmatrix}}_{\text{no such } \vec{x}} = \underbrace{\begin{bmatrix}1\\ 0\\ 0\end{bmatrix}}_{\vec{b}}$$

But, if A's columns were linearly independent, they'd span all of $\mathbb{R}^3$, and so $A\vec{x} = \vec{b}$ would have a **unique** solution $\vec{x}$ for any $\vec{b}$ we think of.

Definition.

Definition: Inverse of a Matrix

An $n \times n$ matrix A is **invertible** if and only if A's columns are linearly independent.
The sentence above is enough to define invertibility, but there are plenty of other equivalent conditions that guarantee invertibility:

- $\text{rank}(A) = n$ (i.e., A is **full rank**)
- A's columns are linearly independent (and hence $\text{colsp}(A) = \mathbb{R}^n$)
- A's rows are linearly independent (and hence $\text{rowsp}(A) = \text{colsp}(A^T) = \mathbb{R}^n$)
- A's null space contains only the zero vector
- $\det(A) \neq 0$

If one of the above properties hold, they all hold. If one doesn't, they all don't.
If A is invertible, its inverse A^{-1} is the **unique** $n \times n$ matrix such that

$$AA^{-1} = I = A^{-1}A$$

If A is not invertible, we say that A is **singular**.

The properties in the box above are sometimes together called the **invertible matrix theorem**. This is not an exhaustive list, either, and we'll see other equivalent properties as time goes on.

Inverse of a 2×2 Matrix. As was the case with the determinant, the general formula for the inverse of a matrix is only convenient for 2×2 matrices.

Definition: Inverse of a 2×2 matrix

If $A = \begin{bmatrix}a & b\\ c & d\end{bmatrix}$ and A is invertible, then

$$A^{-1} = \frac{1}{ad - bc}\begin{bmatrix}d & -b\\ -c & a\end{bmatrix}$$

You could solve for this formula by hand, by finding scalars e, f, g, and h such that

$$\underbrace{\begin{bmatrix}a & b\\ c & d\end{bmatrix}}_{A}\begin{bmatrix}e & f\\ g & h\end{bmatrix} = \begin{bmatrix}1 & 0\\ 0 & 1\end{bmatrix}$$

Note that the formula above involves division by $ad - bc$. If $ad - bc = 0$, then A is not invertible, but $ad - bc$ is just the determinant of A! This should give you a bit more confidence in the equivalence of the statements "A is invertible" and "$\det(A) \neq 0$".

Let's test out the 2×2 formula on some examples. If

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

then

$$A^{-1} = \frac{1}{1 \cdot 4 - 2 \cdot 3} \begin{bmatrix} 4 & -2 \\ -3 & 1 \end{bmatrix} = \frac{1}{-2} \begin{bmatrix} 4 & -2 \\ -3 & 1 \end{bmatrix} = \begin{bmatrix} -2 & 1 \\ 3/2 & -1/2 \end{bmatrix}$$

and indeed both

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

and

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

hold.

On the other hand,

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

is not invertible, since its columns are linearly dependent.

Activity 1

Activity 1

Suppose A is an invertible $n \times n$ matrix.

1. Is A^T invertible? If so, what is its inverse?
2. What is $\det(A^{-1})$?
3. Is A^2 invertible? If so, what is its inverse?

Beyond 2×2 Matrices. For matrices larger than 2×2, the calculation of the inverse is not as straightforward; there's no simple formula. In the 3×3 case, we'd need to find 9 scalars c_{ij} such that

$$\underbrace{\begin{bmatrix} 3 & 7 & 1 \\ -2 & 5 & 0 \\ 4 & 2 & 0 \end{bmatrix}}_{A} \underbrace{\begin{bmatrix} c_{11} & c_{12} & c_{13} \\ c_{21} & c_{22} & c_{23} \\ c_{31} & c_{32} & c_{33} \end{bmatrix}}_{C=A^{-1}} = \underbrace{\begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}}_{I}$$

This involves solving a system of 3 equations in 3 unknowns, 3 times – one per column of the identity matrix. One such system is

$$\begin{aligned} 3c_{11} + 7c_{21} + c_{31} &= 1 \\ -2c_{11} + 5c_{21} &= 0 \\ 4c_{11} + 2c_{21} &= 0 \end{aligned}$$

You can quickly see how this becomes a pain to solve by hand. Instead, we can use one of two strategies:

1. Using **row reduction**, also known as **Gaussian elimination**, which is an efficient method for solving systems of linear equations without needing to write out each equation explicitly. Row reduction can be used to both find the rank and inverse of a matrix, among other things. More traditional linear algebra courses spend a considerable amount of time on this concept, though I've intentionally avoided it in this course to instead spend time on conceptual ideas most relevant to machine learning. That said, you'll get some practice with it in a future homework.
2. Using a **pre-built function** in `numpy` that does the row reduction for us.

At the end of this section, I give you some advice on how to (and not to) compute the inverse of a matrix in code.

More Examples

As we've come to expect, let's work through some examples that illustrate important ideas.

Example: Inverting a Product.

1. Suppose A and B are both invertible $n \times n$ matrices. Is AB invertible? If so, what is its inverse?
2. Suppose $A = BC$, and A, B, and C are all invertible $n \times n$ matrices. What is the inverse of B?
3. If A, B, and AB are all $n \times n$ matrices, and AB is invertible, must A and B both be invertible?
4. In general, if AB is invertible, must A and B both be invertible?

Solutions

1. If A and B are both invertible $n \times n$ matrices, then AB is indeed invertible, with inverse

$$(AB)^{-1} = B^{-1}A^{-1}$$

To confirm, we should check that $B^{-1}A^{-1}$ is both the left inverse of AB, meaning $B^{-1}A^{-1}AB = I$, and that it is the right inverse of AB, meaning $ABB^{-1}A^{-1} = I$.

$$B^{-1}\underbrace{A^{-1}A}_{I}B = B^{-1}B = I$$

$$ABB^{-1}A^{-1} = AA^{-1} = I$$

2. Since we know that A, B, and C are all invertible,

$$A = BC \implies AA^{-1} = BCA^{-1} \implies I = BCA^{-1}$$

This tells us that CA^{-1} is the inverse of B, since multiplying B by it gets us back to I. (For square matrices, it doesn't matter whether we multiply on the left or right; the inverse is the same and is unique.)

3. If A, B, and AB are all $n \times n$ matrices, and AB is invertible, then A and B must individually be invertible, too. One way to argue about this is using facts about determinants. Earlier, we learned

$$\det(AB) = \det(A)\det(B)$$

If AB is invertible, $\det(AB) \neq 0$, but that must mean $\det(A)\det(B) \neq 0$ too, and so neither $\det(A)$ nor $\det(B)$ can be 0 either, meaning A and B are both invertible.

4. In general, if AB is invertible, but we don't know anything about the shapes of A and B, it's not necessarily true that A and B are individually invertible, because they may not be square! Consider

$$AB = \underbrace{\begin{bmatrix} 3 & 0 & 0 \\ 0 & 2 & 0 \end{bmatrix}}_{A}\underbrace{\begin{bmatrix} 3 & 0 \\ 0 & 2 \\ 0 & 0 \end{bmatrix}}_{B} = \begin{bmatrix} 9 & 0 \\ 0 & 4 \end{bmatrix}$$

AB is an invertible 2×2 matrix, but neither A nor B are individually invertible, since neither is square.

Example: Inverting a Sum. Suppose A and B are both invertible $n \times n$ matrices. Is $A + B$ invertible? If so, what is its inverse?

Solution

In general, **no**, $A + B$ is not guaranteed to be invertible.
As a counterexample, suppose A is invertible, and that $B = -A$. Then, $B^{-1} = -A^{-1}$, but

$$A + B = 0$$

and a matrix of all 0's is not invertible (since it's columns don't span all of $\mathbb{R}^n$).

Example: Inverting X^TX. Suppose X is an $n \times d$ matrix. Note that X is not square, and so it is not invertible. However, X^TX is a square matrix, and so it is possible that it is invertible.

Explain why X^TX is invertible if and only if X's columns are linearly independent.

Solution

Recall that if X is $n \times d$, then X^TX is $d \times d$.
In Chapter 5.4, we proved that

$$\text{rank}(X^TX) = \text{rank}(X)$$

The only way for X^TX to be invertible is if $\text{rank}(X^TX) = d$. But, $\text{rank}(X^TX) = d$ if and only if $\text{rank}(X) = d$, which happens when all d of X's columns are linearly independent.

Example: Orthogonal Matrices. Recall, an $n \times n$ matrix Q is **orthogonal** if $Q^TQ = QQ^T = I$.

What is the inverse of Q? Explain how this relates to the formula for rotation matrices in $\mathbb{R}^2$ from earlier,

$$R(\theta) = \begin{bmatrix} \cos(\theta) & -\sin(\theta) \\ \sin(\theta) & \cos(\theta) \end{bmatrix}$$

Solution

Looking at $Q^TQ = QQ^T = I$, we see that

$$Q^{-1} = Q^T$$

since multiplying Q by Q^T on either side gets us back to I.
In $\mathbb{R}^2$, if

$$Q = R(\theta) = \begin{bmatrix} \cos(\theta) & -\sin(\theta) \\ \sin(\theta) & \cos(\theta) \end{bmatrix}$$

then

$$Q^T = \begin{bmatrix} \cos(\theta) & \sin(\theta) \\ -\sin(\theta) & \cos(\theta) \end{bmatrix} = \begin{bmatrix} \cos(-\theta) & -\sin(-\theta) \\ \sin(-\theta) & \cos(-\theta) \end{bmatrix} = R(-\theta)$$

since $\cos(-\theta) = \cos(\theta)$ and $\sin(-\theta) = -\sin(\theta)$. In English, if Q corresponds to rotating by θ in the counterclockwise direction, Q^T corresponds to rotating by $-\theta$ in the counterclockwise direction, i.e. by θ in the opposite direction.

Example: Householder Reflection. Suppose $\vec{u} \in \mathbb{R}^n$ is a **unit vector**. Let

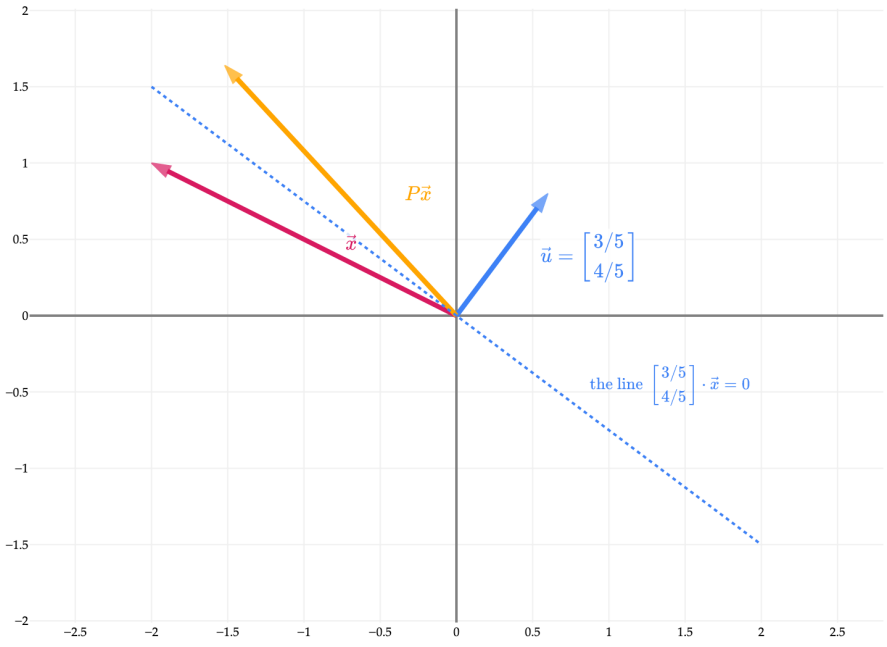
$$P = I - 2\vec{u}\vec{u}^T$$

P is called the **Householder** matrix. $f(\vec{x}) = P\vec{x}$ reflects $\vec{x}$ across the line in $\mathbb{R}^2$ (or plane in $\mathbb{R}^3$, or in general, hyperplane in $\mathbb{R}^n$) $\vec{u}^T\vec{x} = 0$.

For example, suppose $\vec{u} = \begin{bmatrix} 3/5 \\ 4/5 \end{bmatrix}$. Then,

$$P = I - 2\begin{bmatrix} 3/5 \\ 4/5 \end{bmatrix}\begin{bmatrix} 3/5 & 4/5 \end{bmatrix} = \begin{bmatrix} 7/25 & -24/25 \\ -24/25 & 7/25 \end{bmatrix}$$

reflects $\vec{x}$ across the line $\begin{bmatrix} 3/5 \\ 4/5 \end{bmatrix} \cdot \vec{x} = 0$, i.e. $\frac{3}{5}x_1 + \frac{4}{5}x_2 = 0$, or $x_2 = -\frac{3}{4}x_1$, or in more interpretable syntax, $y = -\frac{3}{4}x$.



1. Show, just using the definition $P = I - 2\vec{u}\vec{u}^T$, that P is an orthogonal matrix.
2. Find P^{-1}. (Why does the fact that P is orthogonal imply that P is invertible?)

Solution

If we want to show that P is orthogonal, then we need to show that $P^TP = PP^T = I$. Let's start by showing that $P^TP = I$. Note that $(\vec{u}\vec{u}^T)^T = \vec{u}\vec{u}^T$.

$$\begin{aligned}
P^TP &= (I - 2\vec{u}\vec{u}^T)^T(I - 2\vec{u}\vec{u}^T) \\
&= (I^T - 2(\vec{u}\vec{u}^T)^T)(I - 2\vec{u}\vec{u}^T) \\
&= (I - 2(\vec{u}^T)^T\vec{u}^T)(I - 2\vec{u}\vec{u}^T) \\
&= (I - 2\vec{u}\vec{u}^T)(I - 2\vec{u}\vec{u}^T) \\
&= I - 2\vec{u}\vec{u}^T - 2\vec{u}\vec{u}^T + 4(\vec{u}\vec{u}^T)(\vec{u}\vec{u}^T) \\
&= I - 4\vec{u}\vec{u}^T + 4(\vec{u}\vec{u}^T)(\vec{u}\vec{u}^T) \\
&= I - 4\vec{u}\vec{u}^T + 4\vec{u}\underbrace{\vec{u}^T\vec{u}}_{\text{unit vector!}}\vec{u}^T \\
&= I - 4\vec{u}\vec{u}^T + 4\vec{u}(1)\vec{u}^T \\
&= I
\end{aligned}$$

This means that P's inverse is just P^T, since we multipled P by P^T to get I. This also means that $PP^T = I$ too, since a square matrix's inverse is unique and can be multiplied on either side to get I. So, P is indeed orthogonal.

Note that in the final step above, the $-4\vec{u}\vec{u}^T$ and $4\vec{u}\vec{u}^T$ cancelled out. But, the -4 came from $-2-2$, while the 4 came from $(-2)\cdot(-2)$. All of this is to say that if we changed the -2 to some other number in the definition of the Householder matrix, we'd no longer have $P^TP = I$, because -2 is the only non-zero solution to $-(c+c) = c \cdot c$.

As mentioned above, P is invertible: $P^{-1} = P^T$. But, implicitly we also showed that $P^T = P$, meaning P is symmetric:

$$P^T = (I - 2\vec{u}\vec{u}^T)^T = I - 2\vec{u}\vec{u}^T = P$$

So, P is both **orthogonal and symmetric**! This means that $P^2 = I$, or i.e. $P = P^{-1}$.

Intuitively, P involves reflecting across a line, so $P^TP = P^2$ involves reflecting twice, which is the same as doing nothing, and the inverse of P is the same as P itself.

Example: A^2 and A. Prove that if A^2 is invertible, then A is invertible. Do so in three different ways:

1. By constructing A's inverse directly.
2. By arguing about the null spaces of A^2 and A.
3. By using the determinant.

Solution

Solution 1: By direct construction
One way to prove that A is invertible is to find its inverse. Since A^2 is invertible, there must exist some matrix B such that

$$A^2B = I, \quad BA^2 = I$$

Note that the inclusion of both orders is important, since in general, order matters for matrix multiplication. Writing $A^2 = AA$ in the first equation gives us

$$A(AB) = I$$

This tells us that AB is the inverse of A, so A is invertible.
If we used the second equation instead, we'd have

$$(BA)A = I$$

So, we'd call AB the right inverse of A (since it's the inverse that multiplies on the right) and BA the left inverse. A general fact is that if A has both a left and right inverse, they must be the same, meaning that AB must be equal to BA here (remember this is not true in general for arbitrary matrices A and B). A quick proof of this is

$$BA = BAI = BA(AAB) = BAAAB = (BAA)AB = IAB = AB$$

So, in short, $A^{-1} = AB = BA$, and since we've found A's inverse, A must be invertible.
Solution 2: The null space perspective
Let's argue by contradiction. Suppose A^2 is invertible, but A is not. This would imply that A^2 null space is just the zero vector, $\vec{0}$, while A's null space is not just $\vec{0}$. Suppose $\vec{x}$ is some vector \textbf{other than $\vec{0}$} in nullsp(A), meaning that

$$A\vec{x} = \vec{0}$$

But, left-multiplying both sides by A gives us

$$A(A\vec{x}) = A\vec{0} \implies A^2\vec{x} = \vec{0}$$

This tells us that $\vec{x}$ is in the null space of A^2, since $A^2\vec{x} = \vec{0}$. But, this contradicts our assumption that A^2 is invertible, because the only vector in the null space of an invertible matrix is the zero vector. So, our original assumption that A is not invertible must be false, and A must be invertible.
Solution 3: The determinant perspective
A fact from Chapter 6.1 is that if A and B are two $n \times n$ matrices, then

$$\det(A)\det(B) = \det(AB)$$

So,

$$\det(A^2) = \det(A \cdot A) = \det(A) \cdot \det(A) = \det(A)^2$$

But, we also know that invertible matrices have non-zero determinants, so we're given that $\det(A^2) \neq 0$. But, since $\det(A^2) = \det(A)^2$, this implies that $\det(A)^2 \neq 0$, and so $\det(A) \neq 0$ too, and so A is invertible.

Example: Matrix-Vector Products. If $A\begin{bmatrix}3\\1\\-1\end{bmatrix} = \begin{bmatrix}9\\4\\3\end{bmatrix}$ and $A\begin{bmatrix}4\\0\\2\end{bmatrix} = \begin{bmatrix}0\\0\\0\end{bmatrix}$, is A invertible?

Solution

No, A is not invertible. This is because its columns are linearly dependent, or equivalently, it has a non-trivial null space. This comes from the fact that

$$A\begin{bmatrix}4\\0\\2\end{bmatrix} = \vec{0}$$

which means that $\begin{bmatrix}4\\0\\2\end{bmatrix}$ is in the null space of A.

Computing the Inverse in Code

Recall that if A is an $n \times n$ matrix and $\vec{b} \in \mathbb{R}^n$, then

$$A\vec{x} = \vec{b}$$

is a system of n equations in n unknowns. One of the big (conceptual) usages of the inverse is to solve such a system, as I mentioned at the start of this section. If A is invertible, then we can solve for $\vec{x}$ by multiplying both sides on the left by A^{-1}:

$$A\vec{x} = \vec{b} \implies A^{-1}A\vec{x} = A^{-1}\vec{b} \implies \vec{x} = A^{-1}\vec{b}$$

A^{-1} has encoded within it the solution to $A\vec{x} = \vec{b}$, no matter what $\vec{b}$ is. This makes A^{-1} very powerful. But, that power doesn't come for free: in practice, finding A^{-1} is **less efficient and more prone to floating point errors** than solving the **one** system of n equations in n unknowns directly for the specific $\vec{b}$ we care about.

- Solving $A\vec{x} = \vec{b}$ involves solving just one system of n equations in n unknowns.
- Finding A^{-1} involves solving a system of n equations in n unknowns, n times! Each system has the same coefficient matrix A but a different right-hand side $\vec{b}$, corresponding to the columns of the identity matrix (which are the standard basis vectors in $\mathbb{R}^n$).

The more floating point operations we need to do, the more error is introduced into the final results.

Let's run an experiment. Suppose we'd like to find the solution to the system

$$\begin{aligned} x_1 + x_2 &= 7 \\ x_1 + 1.000000001x_2 &= 7.000000004 \end{aligned}$$

This corresponds to $A\vec{x} = \vec{b}$, with

$$A = \begin{bmatrix} 1 & 1 \\ 1 & 1.000000001 \end{bmatrix}, \quad \vec{b} = \begin{bmatrix} 7 \\ 7.000000004 \end{bmatrix}$$

Here, we can eyeball the solution as being $\vec{x} = \begin{bmatrix} 3 \\ 4 \end{bmatrix}$. How close to this "true" $\vec{x}$ do each of the following techniques get?

Option 1: Using `np.linalg.inv`

As we saw at the start of the section, `np.linalg.inv(A)` finds the inverse of `A`, assuming it exists. The following cell finds the solution to $A\vec{x} = \vec{b}$ by first computing A^{-1}, and then $A^{-1}\vec{b} = \vec{x}$.

```
# Prevents unnecessary rounding.
np.set_printoptions(precision=16, suppress=True)

A = np.array([[1, 1],
              [1, 1.000000001]])

b = np.array([[7],
              [7.000000004]])

x_inv = np.linalg.inv(A) @ b
x_inv
```

```
array([[2.9999998807907104],
       [4.0000001192092896]])
```

Option 2: Using `np.linalg.solve`

The following cell solves the same problem as the previous cell, but rather than asking for the inverse of A, it asks for the solution to $A\vec{x} = \vec{b}$, which doesn't inherently require inverting A.

```
x_solve = np.linalg.solve(A, b)
x_solve
```

```
array([[3.],
       [4.]])
```

This small example already illustrates the big idea: inverting can introduce more numerical error than is necessary. If we cast the problem more abstractly, consider the matrix

$$A = \begin{bmatrix} 1 & 1 \\ 1 & 1+\epsilon \end{bmatrix}$$

where ϵ is a small constant, e.g. $\epsilon = 0.000000001$ in the example above. The inverse of A is

$$A^{-1} = \frac{1}{1(1+\epsilon) - 1 \cdot 1} \begin{bmatrix} 1+\epsilon & -1 \\ -1 & 1 \end{bmatrix} = \frac{1}{\epsilon} \begin{bmatrix} 1+\epsilon & -1 \\ -1 & 1 \end{bmatrix}$$

The closer ϵ is to 0, the larger $\frac{1}{\epsilon}$ becomes, making any floating point errors all the more costly.

Use `np.linalg.solve` instead of `np.linalg.inv`!

Conceptually, both `x = np.linalg.inv(A) @ b` and `x = np.linalg.solve(A, b)` do the same thing, but we **much prefer** the latter.

6.3. Projecting onto the Column Space

We're *almost* ready to return to our original motivation for studying linear algebra, which was to perform linear regression using multiple input variables. This section outlines the final piece of the puzzle.

Approximating using a Single Vector

In Chapter 3.4, we introduced the **approximation problem**, which asked:

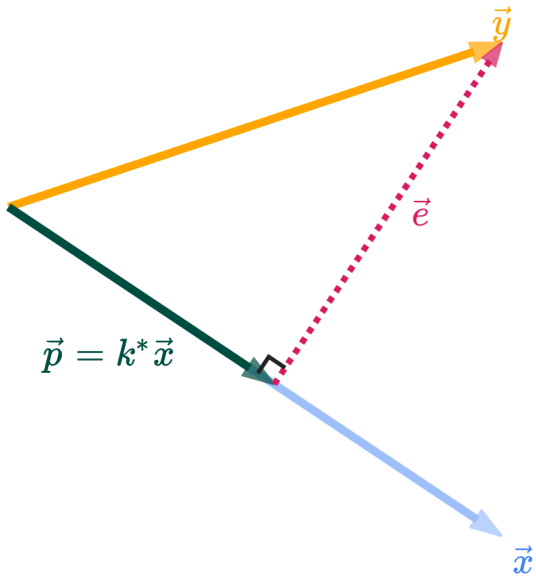
> **Among all vectors of the form $k\vec{x}$, which one is closest to $\vec{y}$?**

We now know the answer is the vector $\vec{p}$, where

$$\vec{p} = \left(\frac{\vec{y} \cdot \vec{x}}{\vec{x} \cdot \vec{x}} \right) \vec{x}$$

$\vec{p}$ is called the **orthogonal projection** of $\vec{y}$ onto $\vec{x}$.

Note that I've used $\vec{y}$ and $\vec{x}$ here rather than $\vec{u}$ and $\vec{v}$, just to make the notation more consistent with the notation we'll use as we move back into the world of machine learning.



As we've studied, the resulting error vector,

$$\vec{e} = \vec{y} - \vec{p}$$

is **orthogonal** to $\vec{x}$.

In our original look at the approximation problem, we were approximating $\vec{y}$ using a scalar multiple of just a single vector, $\vec{x}$. The set of all scalar multiples of $\vec{x}$, denoted by $\text{span}(\{\vec{x}\})$, is a line in $\mathbb{R}^n$.

Key idea: instead of projecting onto the subspace spanned by just a single vector, how might we project onto the subspace spanned by multiple vectors?

Approximating using Multiple Vectors

Equipped with our understanding of linear independence, spans, subspaces, and column spaces, we're ready to tackle a more advanced version of the approximation problem.

The Approximation Problem

Suppose $\vec{x}^{(1)}, \vec{x}^{(2)}, \ldots, \vec{x}^{(d)} \in \mathbb{R}^n$, and $\vec{y} \in \mathbb{R}^n$ **is not necessarily** in $\text{span}(\{\vec{x}^{(1)}, \vec{x}^{(2)}, \ldots, \vec{x}^{(d)}\})$.
We can construct the matrix X by placing the $\vec{x}^{(i)}$'s in its columns:

$$X = \begin{bmatrix} | & | & & | \\ \vec{x}^{(1)} & \vec{x}^{(2)} & \cdots & \vec{x}^{(d)} \\ | & | & & | \end{bmatrix}$$

Then, the following three statements are all **equivalent** ways of asking the approximation problem:

1. Among all vectors in $\text{span}(\{\vec{x}^{(1)}, \vec{x}^{(2)}, \ldots, \vec{x}^{(d)}\})$, which is closest to $\vec{y}$?
2. Among all vectors in $\text{colsp}(X)$, which is closest to $\vec{y}$?
3. Among all vectors of the form $X\vec{w}$, where $\vec{w} \in \mathbb{R}^d$, which is closest to $\vec{y}$?

All three statements at the bottom of the box above are asking the exact same question; I've presented all three forms so that you see more clearly how the ideas of spans, column spaces, and matrix-vector multiplication fit together. I will tend to refer to the latter two versions of the problem the most. In what follows, suppose X is an $n \times d$ matrix whose columns $\vec{x}^{(1)}, \vec{x}^{(2)}, \ldots, \vec{x}^{(d)}$ are the building blocks we want to approximate $\vec{y}$ with.

First, let's get the trivial case out of the way. **If** $\vec{y} \in \text{colsp}(X)$, then the vector in $\text{colsp}(X)$ that is closest to $\vec{y}$ is just $\vec{y}$ itself. If that's the case, there exists some $\vec{w}$ such that $\vec{y} = X\vec{w}$ exactly. This $\vec{w}$ is unique only if X's columns are linearly independent; otherwise, there will be infinitely many good $\vec{w}$'s.

But, that's not the case I'm really interested in. I care more about when $\vec{y}$ is **not** in $\text{colsp}(X)$. (Remember, this is the case we're interested in when we're doing linear regression: usually, it's not possible to make our predictions 100% correct, and we'll have to settle for some error.)

Then what?

In general, $\text{colsp}(X)$ is an r-dimensional subspace of $\mathbb{R}^n$, where $r = \text{rank}(X)$. In the diagram below, I've used a plane to represent $\text{colsp}(X)$; just remember that X may have more than 3 rows or columns.

Remember that $\text{colsp}(X)$ is the set of linear combinations of X's columns, so it's the set of all vectors that can be written as $X\vec{w}$, where $\vec{w} \in \mathbb{R}^d$.

Let's consider two possible vectors of the form $X\vec{w}$, and look at their corresponding **error vectors**, $\vec{e} = \vec{y} - X\vec{w}$. I won't draw the columns of X, since those would clutter up the picture.

Our problem boils down to finding the $\vec{w}$ that minimizes the norm of the error vector. Since it's a bit easier to work with squared norms (remember that $\|\vec{x}\|^2 = \vec{x} \cdot \vec{x}$), we'll minimize the squared norm of the error vector instead; this is an equivalent problem, since the norm is non-negative to begin with.

$$\underbrace{\|\vec{e}\|^2 = \|\vec{y} - X\vec{w}\|^2}_{\text{which } \vec{w} \text{ minimizes this?}}$$

Think of $\|\vec{y} - X\vec{w}\|^2$ as a function of $\vec{w}$ only; X and $\vec{y}$ should be thought of as fixed. This is a **least squares** problem: we're looking for the $\vec{w}$ that minimizes the **sum of squared errors** between $\vec{y}$ and $X\vec{w}$.

There are two ways we'll minimize this function of $\vec{w}$:

1. Using a geometric argument, as we did in the single vector case.
2. Using calculus. This is more involved than before, since the input variable is a vector, not a scalar, but it can be done, as we'll see in Chapter 8.

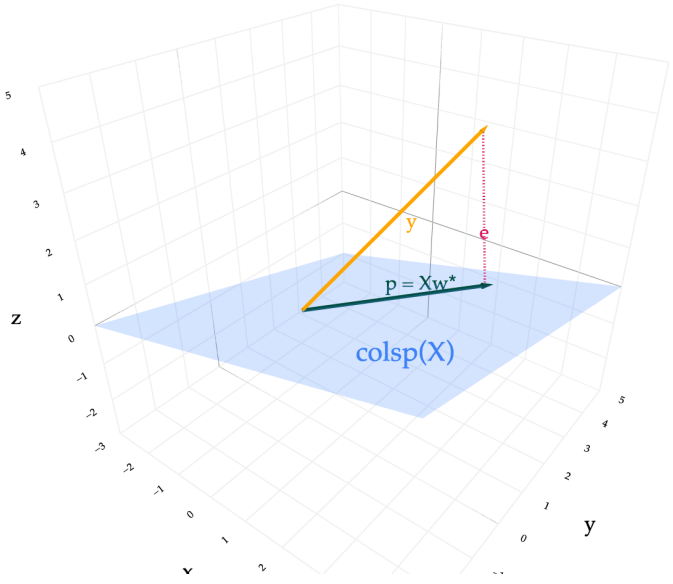
Let's focus on the geometric argument. What does our intuition tell us? Extending the single vector case, we expect the vector in colsp(X) that is closest to $\vec{y}$ to be the **orthogonal projection** of $\vec{y}$ onto colsp(X): that is, its error should be orthogonal to colsp(X).

We could see this intuitively in the visual above. $\vec{w}_o$ was chosen to make $\vec{e}_o$ orthogonal to colsp(X), meaning that $\vec{e}_o$ is orthogonal to every vector in colsp(X). (The subscript "o" stands for "orthogonal".) $\vec{w}'$ was some other arbitrary vector, leading $\vec{e}'$ to not be orthogonal to colsp(X). Clearly $\vec{e}_o$ is shorter than $\vec{e}'$.

To **prove** that the optimal choice of $\vec{w}$ comes from making the error vector orthogonal to colsp(X), you could use the same argument as in the single vector case: if you consider two vectors, $\vec{w}_o$ with an orthogonal error vector $\vec{e}_o$, and $\vec{w}'$ with an error vector $\vec{e}'$ that is not orthogonal to colsp(X), then we can draw a right triangle with $\vec{e}'$ as the hypotenuse and $\vec{e}_o$ as one of the legs, making

$$\|\vec{e}'\|^2 > \|\vec{e}_o\|^2$$

This is such an important idea that I want to redraw the picture above with just the orthogonal projection. Note that I've replaced $\vec{w}_o$ with $\vec{w}^*$ to indicate that this is the optimal choice of $\vec{w}$.



Understand the diagram above!

In my mind, the diagram above is the single most important takeaway from this semester. It shows how lots of ideas tie together, including spans, column spaces, orthogonality, and projections. (There should

be a little right angle marker between the tip of $\vec{p}$ and $\vec{e}$.)
Screenshot it, save it, draw it yourself; do whatever you need to do to internalize it.

A Proof that the Orthogonal Error Vector is the Shortest. In office hours, a student asked for more justification that the shortest possible error vector is one that's orthogonal to the column space of X, especially because it's hard to visualize what orthogonality looks like in higher dimensions. Remember, all it means for two vectors to be orthogonal is that their dot product is 0.

Given that, let's assume **only** that

- $\vec{w}_o$ is chosen so that $\vec{e}_o = \vec{y} - X\vec{w}_o$ is orthogonal to colsp(X), and
- $\vec{w}'$ is any other choice of $\vec{w}$, with a corresponding error vector $\vec{e}' = \vec{y} - X\vec{w}'$.

Just with these facts alone, we can show that $\vec{e}_o$ is the shortest possible error vector. To do so, let's start by considering the (squared) magnitude of $\vec{e}'$:

$$
\begin{aligned}
\|\vec{e}'\|^2 &= \|\vec{y} - X\vec{w}'\|^2 \\
&= \underbrace{\|\vec{y} - X\vec{w}' + X\vec{w}_o - X\vec{w}_o\|^2}_{\text{seems arbitrary, but it's a legal operation that brings } \vec{w}_o \text{ back in}} \\
&= \|\underbrace{\vec{y} - X\vec{w}_o}_{\text{“}a\text{”}} + \underbrace{X(\vec{w}_o - \vec{w}')}_{\text{“}b\text{”}}\|^2 \\
&= \underbrace{\|\vec{y} - X\vec{w}_o\|^2 + \|X(\vec{w}_o - \vec{w}')\|^2 + 2(\vec{y} - X\vec{w}_o) \cdot X(\vec{w}_o - \vec{w}')}_{\|\vec{a}+\vec{b}\|^2 = \|\vec{a}\|^2 + \|\vec{b}\|^2 + 2\vec{a}\cdot\vec{b}} \\
&= \|\vec{e}_o\|^2 + \|X(\vec{w}_o - \vec{w}')\|^2 + \underbrace{2\vec{e}_o \cdot X(\vec{w}_o - \vec{w}')}_{0, \text{ because } \vec{e}_o \text{ is orthogonal to the columns of } X} \\
&= \|\vec{e}_o\|^2 + \|X(\vec{w}_o - \vec{w}')\|^2 \\
&\geq \|\vec{e}_o\|^2
\end{aligned}
$$

So, **no matter what choice of $\vec{w}'$ we make**, the magnitude of $\vec{e}'$ can't be smaller than the magnitude of $\vec{e}_o$. This means that the error vector that is orthogonal to the column space of X is the shortest possible error vector.

This is really just the same proof as in Chapter 3.4, where we argued that $\vec{e}_o$, $X(\vec{w}_o - \vec{w}')$, and $\vec{e}'$ form a right triangle, where $\vec{e}'$ is the hypotenuse.

The Normal Equation

We've come to the conclusion that in order to find the $\vec{w}$ that minimizes

$$\|\vec{e}\|^2 = \|\vec{y} - X\vec{w}\|^2$$

we need to find the $\vec{w}$ that makes the error vector $\vec{e} = \vec{y} - X\vec{w}$ orthogonal to colsp(X). colsp(X) is the set of all linear combinations of X's columns. So, if we can find an $\vec{e}$ that is orthogonal to every column of X, then it must be orthogonal to any of their linear combinations, too.

To see why, take any vector in colsp(X). It has the form $a_1\vec{x}^{(1)} + a_2\vec{x}^{(2)} + \ldots + a_d\vec{x}^{(d)}$. Then

$$\begin{aligned}\left(a_1\vec{x}^{(1)} + a_2\vec{x}^{(2)} + \ldots + a_d\vec{x}^{(d)}\right) \cdot \vec{e} &= a_1(\vec{x}^{(1)} \cdot \vec{e}) + a_2(\vec{x}^{(2)} \cdot \vec{e}) + \ldots + a_d(\vec{x}^{(d)} \cdot \vec{e}) \\ &= a_1(0) + a_2(0) + \ldots + a_d(0) \\ &= 0\end{aligned}$$

So checking orthogonality against the columns of X is enough to guarantee orthogonality against everything in colsp(X).

So, we're looking for a $\vec{e} = \vec{y} - X\vec{w}$ that satisfies

$$\begin{aligned}\vec{x}^{(1)} \cdot (\vec{y} - X\vec{w}) &= 0 \\ \vec{x}^{(2)} \cdot (\vec{y} - X\vec{w}) &= 0 \\ &\vdots \\ \vec{x}^{(d)} \cdot (\vec{y} - X\vec{w}) &= 0\end{aligned}$$

As you might have guessed, there's an easier way to write these d equations simultaneously. Above, we're taking the dot product of $\vec{y} - X\vec{w}$ with each of the **columns** of X. We've learned that $A\vec{v}$ contains the dot products of $\vec{v}$ with the **rows** of A. So how do we get the dot products of $\vec{y} - X\vec{w}$ with the **columns** of X? **Transpose it!**

$$X^T(\vec{y} - X\vec{w}) = \begin{bmatrix} - & \vec{x}^{(1)T} & - \\ - & \vec{x}^{(2)T} & - \\ & \vdots & \\ - & \vec{x}^{(d)T} & - \end{bmatrix} (\vec{y} - X\vec{w}) = \begin{bmatrix} \vec{x}^{(1)} \cdot (\vec{y} - X\vec{w}) \\ \vec{x}^{(2)} \cdot (\vec{y} - X\vec{w}) \\ \vdots \\ \vec{x}^{(d)} \cdot (\vec{y} - X\vec{w}) \end{bmatrix}$$

So, if we want $\vec{y} - X\vec{w}$ to be orthogonal to each of the columns of X, then we need $X^T(\vec{y} - X\vec{w}) = \vec{0}$ (note that this is the vector $\vec{0} \in \mathbb{R}^d$, not the scalar 0.) Another way of saying this is that we need the error vector to be in the left null space of X, i.e. $\vec{e} \in \text{null}(X^T)$.

$$\begin{aligned}X^T\vec{e} &= \vec{0} \\ X^T(\vec{y} - X\vec{w}) &= \vec{0} \\ X^T\vec{y} - X^TX\vec{w} &= \vec{0} \\ X^TX\vec{w} &= X^T\vec{y}\end{aligned}$$

The final equation above is called the **normal equation**. "Normal" means "orthogonal". Sometimes it's called the normal equation**s** to reference the fact that it's a system of d equations and d unknowns, where the unknowns are components of $\vec{w}$ ($w_1, w_2, \ldots, w_d$.) I myself will use the terms "normal equation" and "normal equations" interchangeably.

Note that $X^TX\vec{w} = X^T\vec{y}$ looks a lot like $X\vec{w} = \vec{y}$, with added factors of X^T on the left. Remember that if $\vec{y}$ is in colsp(X), then $X\vec{w} = \vec{y}$ has a solution, but that's usualy not the case, hence why we're attempting to approximate $\vec{y}$ with a linear combination of X's columns.

Is there a unique vector $\vec{w}$ that satisfies the normal equation? That depends on whether X^TX is invertible. X^TX is a $d \times d$ matrix with the same rank as the $n \times d$ matrix X, as we proved in Chapter 5.4.

$$\text{rank}(X^T X) = \text{rank}(X)$$

So, $X^T X$ is invertible if and only if $\text{rank}(X) = d$, meaning all of X's columns are linearly independent. In that case, the best choice of $\vec{w}$ is the unique vector

$$\boxed{\vec{w}^* = (X^T X)^{-1} X^T \vec{y}}$$

$\vec{w}^*$ has a star on it, denoting that it is the best choice of $\vec{w}$. **I don't ask you to memorize much (you get to bring a notes sheet into your exams, after all), but this equation is perhaps the most important of the semester!** It might even look familiar: back in the single vector case in Chapter 3.4, the optimal coefficient on $\vec{x}$ was $\frac{\vec{x} \cdot \vec{y}}{\vec{x} \cdot \vec{x}} = \frac{\vec{x}^T \vec{y}}{\vec{x}^T \vec{x}}$, which looks similar to the one above. The difference is that here, $X^T X$ is a matrix, not a scalar. (But, if X is just a matrix with a single column, then $X^T X$ is just the dot product of X with itself, which is a scalar, and the boxed formula above reduces to the formula from Chapter 3.4.)

What if $X^T X$ **isn't** invertible? Then, there are infinitely many $\vec{w}^*$'s that satisfy the normal equation,

$$X^T X \vec{w} = X^T \vec{y}$$

It's not immediately obvious what it means for there to be infinitely many solutions to the normal equation; I'll explore this idea in Chapter 6.4.

First, let's start with a straightforward example. Let

$$X = \begin{bmatrix} 1 & 0 \\ 2 & 1 \\ 1 & 1 \\ 0 & -1 \end{bmatrix}, \quad \vec{y} = \begin{bmatrix} 1 \\ 0 \\ 4 \\ 5 \end{bmatrix}$$

The vector in $\text{colsp}(X)$ that is closest to $\vec{y}$ is the vector $X\vec{w}^*$, where $\vec{w}^*$ is the solution to the normal equations,

$$X^T X \vec{w}^* = X^T \vec{y}$$

The first step is to compute $X^T X$, which is a 2×2 matrix of **dot products of the columns of** X.

$$X^T X = \begin{bmatrix} 6 & 3 \\ 3 & 3 \end{bmatrix}$$

$X^T X$ is invertible, so we can solve for $\vec{w}^*$ uniquely. Remember that in practice, we'd ask Python to solve `np.linalg.solve(X.T @ X, X.T @ y)`, but here $X^T X$ is small enough that we can invert it by hand.

$$X^T X = \begin{bmatrix} 6 & 3 \\ 3 & 3 \end{bmatrix} \implies (X^T X)^{-1} = \frac{1}{9} \begin{bmatrix} 3 & -3 \\ -3 & 6 \end{bmatrix} = \begin{bmatrix} 1/3 & -1/3 \\ -1/3 & 2/3 \end{bmatrix}$$

Then,

$$\vec{w}^* = (X^T X)^{-1} X^T \vec{y} = \underbrace{\begin{bmatrix} 1/3 & -1/3 \\ -1/3 & 2/3 \end{bmatrix}}_{(X^T X)^{-1}} \underbrace{\begin{bmatrix} 1 & 2 & 1 & 0 \\ 0 & 1 & 1 & -1 \end{bmatrix}}_{X^T} \begin{bmatrix} 1 \\ 0 \\ 4 \\ 5 \end{bmatrix} = \begin{bmatrix} 2 \\ -7/3 \end{bmatrix}$$

The magic is in the interpretation of the numbers in $\vec{w}^*$, 2 and $-\frac{7}{3}$. These are the coefficients of the columns of X in the linear combination that is closest to $\vec{y}$. Meaning,

$$X\vec{w}^* = 2 \underbrace{\begin{bmatrix} 1 \\ 2 \\ 1 \\ 0 \end{bmatrix}}_{\vec{x}^{(1)}} - \frac{7}{3} \underbrace{\begin{bmatrix} 0 \\ 1 \\ 1 \\ -1 \end{bmatrix}}_{\vec{x}^{(2)}} = \begin{bmatrix} 2 \\ 5/3 \\ -1/3 \\ 7/3 \end{bmatrix}$$

is the vector in colsp(X) that is closest to $\vec{y}$. This vector is the orthogonal projection of $\vec{y}$ onto colsp(X).

Examples

The first example above is the most concrete. The examples that follow will build our understanding of how orthogonal projections really work. Then, Chapter 6.4 will explore what happens when there are infinitely many solutions to the normal equation, and introduce the idea of the projection matrix.

Example: Point and Plane. Find the point on the plane $6x - 3y + 2z = 0$ that is closest to the point $(1, 1, 1)$.

Solution

At first, this doesn't seem like a projection problem, but it is. The plane

$$6x - 3y + 2z = 0$$

is a 2-dimensional subspace of $\mathbb{R}^3$, meaning it can be described as the span of two non-collinear vectors. So, all we need to do is find some matrix X with those two vectors as columns, and then use the formula we derived above to project $\vec{y} = \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix}$ onto colsp(X). The fact that we said "point" instead of "vector" doesn't change the problem: in settings like these, points and vectors are equivalent.

Two vectors that lie in the plane (but point in different directions) are $\begin{bmatrix} 1 \\ 2 \\ 0 \end{bmatrix}$ and $\begin{bmatrix} 1 \\ 0 \\ -3 \end{bmatrix}$. There's nothing special about these two vectors, other than that they have relatively small integer components. Let's use them as columns of X:

$$X = \begin{bmatrix} 1 & 1 \\ 2 & 0 \\ 0 & -3 \end{bmatrix}$$

Now, to formulate the normal equations, $X^T X \vec{w}^* = X^T \vec{y}$, we need to compute $X^T X$ and $X^T \vec{y}$.

$$X^T X = \begin{bmatrix} 1 & 2 & 0 \\ 1 & 0 & -3 \end{bmatrix} \begin{bmatrix} 1 & 1 \\ 2 & 0 \\ 0 & -3 \end{bmatrix} = \begin{bmatrix} 5 & 1 \\ 1 & 10 \end{bmatrix}$$

$$X^T \vec{y} = \begin{bmatrix} 1 & 2 & 0 \\ 1 & 0 & -3 \end{bmatrix} \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix} = \begin{bmatrix} 3 \\ -2 \end{bmatrix}$$

So, the normal equations are

$$\begin{bmatrix} 5 & 1 \\ 1 & 10 \end{bmatrix} \underbrace{\begin{bmatrix} w_0^* \\ w_1^* \end{bmatrix}}_{\vec{w}^*} = \begin{bmatrix} 3 \\ -2 \end{bmatrix}$$

We could use $\vec{w}^* = (X^T X)^{-1} X^T \vec{y}$, but often it's easier to solve the system directly.

$$\begin{aligned} 5w_0^* + w_1^* &= 3 \\ w_0^* + 10w_1^* &= -2 \end{aligned}$$

Solving this system gives us $w_0^* = \frac{32}{49}$ and $w_1^* = -\frac{13}{49}$. (Sorry, I know the numbers aren't pretty in this example. But that's what happens in the real world.)

Now that we've solved for $\vec{w}^* = \begin{bmatrix} \frac{32}{49} \\ -\frac{13}{49} \end{bmatrix}$, the projection of $\vec{y}$ onto colsp(X) - which is the point on the plane that's closest to $\vec{y}$ - is

$$\begin{aligned} X\vec{w}^* &= \begin{bmatrix} 1 & 1 \\ 2 & 0 \\ 0 & -3 \end{bmatrix} \begin{bmatrix} \frac{32}{49} \\ -\frac{13}{49} \end{bmatrix} \\ &= \begin{bmatrix} \frac{32}{49} - \frac{13}{49} \\ \frac{64}{49} \\ \frac{39}{49} \end{bmatrix} \end{aligned}$$

Example: What if $\vec{y} \in$ colsp(X)? Find the orthogonal projection of $\vec{y} = \begin{bmatrix} 1 \\ 3 \\ 2 \\ -1 \end{bmatrix}$ onto colsp(X), where $X = \begin{bmatrix} 1 & 0 \\ 2 & 1 \\ 1 & 1 \\ 0 & -1 \end{bmatrix}$.

Solution

First, notice that $\vec{y}$ is just the sum of the two columns of colsp(X). So intuitively, because $\vec{y}$ is already in the column space of X, the projection is just $\vec{y}$ itself.
But let's make sure the math works out that way.
First, we solve the normal equations for $\vec{w}^*$.

$$\begin{aligned} X^T X &= \begin{bmatrix} 1 & 2 & 1 & 0 \\ 0 & 1 & 1 & -1 \end{bmatrix} \begin{bmatrix} 1 & 0 \\ 2 & 1 \\ 1 & 1 \\ 0 & -1 \end{bmatrix} \\ &= \begin{bmatrix} 1+4+1 & 2+1 \\ 2+1 & 1+1+1 \end{bmatrix} \\ &= \begin{bmatrix} 6 & 3 \\ 3 & 3 \end{bmatrix} \end{aligned}$$

$$\begin{aligned} X^T \vec{y} &= \begin{bmatrix} 1 & 2 & 1 & 0 \\ 0 & 1 & 1 & -1 \end{bmatrix} \begin{bmatrix} 1 \\ 3 \\ 2 \\ -1 \end{bmatrix} \\ &= \begin{bmatrix} 9 \\ 6 \end{bmatrix} \end{aligned}$$

So, $\vec{w}^*$ is the solution to

$$\underbrace{\begin{bmatrix} 6 & 3 \\ 3 & 3 \end{bmatrix}}_{X^T X} \vec{w}^* = \underbrace{\begin{bmatrix} 9 \\ 6 \end{bmatrix}}_{X^T \vec{y}}$$

and here, it's pretty clear that $\vec{w}^* = \begin{bmatrix} 1 \\ 1 \end{bmatrix}$. (Again, we could go through the hassle of inverting $X^T X$ to get the same answer, but there's no need to.)
This means that the projection of $\vec{y}$ onto colsp(X) is

$$X\vec{w}^* = \begin{bmatrix} 1 & 0 \\ 2 & 1 \\ 1 & 1 \\ 0 & -1 \end{bmatrix} \begin{bmatrix} 1 \\ 1 \end{bmatrix} = \begin{bmatrix} 1 \\ 3 \\ 2 \\ -1 \end{bmatrix} = \vec{y}$$

as we predicted from the start.

Example: Orthogonality with the Columns of X. Let $X = \begin{bmatrix} 1 & 2 \\ 3 & 2 \\ 0 & 2 \\ 1 & 2 \end{bmatrix}$, $Z = \begin{bmatrix} 1 & 0 \\ 0 & -1 \\ 0 & 1 \\ 3 & 0 \end{bmatrix}$, and $\vec{y}$ be any vector in $\mathbb{R}^4$.

Let $\vec{p}_X$ and $\vec{p}_Z$ be the orthogonal projections of $\vec{y}$ onto colsp(X) and colsp(Z), respectively.

Explain why it is **guaranteed** that the components of the vector

$$\vec{e}_X = \vec{y} - \vec{p}_X$$

sum to zero, but the components of the vector $\vec{e}_Z = \vec{y} - \vec{p}_Z$ do not necessarily.

Solution

The error vectors $\vec{e}_X$ and $\vec{e}_Z$ are orthogonal to the column spaces of their respective matrices. In other words,

$$X^T\vec{e}_X = \vec{0}, \qquad Z^T\vec{e}_Z = \vec{0}$$

Suppose $\vec{e}_X$'s components are e_1, e_2, e_3, e_4. In the case of X, we have that

$$\underbrace{\begin{bmatrix} 1 & 3 & 0 & 1 \\ 2 & 2 & 2 & 2 \end{bmatrix}}_{X^T} \underbrace{\begin{bmatrix} e_1 \\ e_2 \\ e_3 \\ e_4 \end{bmatrix}}_{\vec{e}_X} = \begin{bmatrix} e_1 + 3e_2 + e_4 \\ 2e_1 + 2e_2 + 2e_3 + 2e_4 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \end{bmatrix}$$

The second component of the above equation implies that $2e_1 + 2e_2 + 2e_3 + 2e_4 = 0$, so $e_1 + e_2 + e_3 + e_4 = 0$, meaning the components of $\vec{e}_X$ sum to 0.
In the case of Z, we don't have that same assurance.

$$\underbrace{\begin{bmatrix} 1 & 0 & 0 & 3 \\ 0 & -1 & 1 & 0 \end{bmatrix}}_{Z^T} \underbrace{\begin{bmatrix} e_1 \\ e_2 \\ e_3 \\ e_4 \end{bmatrix}}_{\vec{e}_Z} = \begin{bmatrix} e_1 + 3e_4 \\ -e_2 + e_3 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \end{bmatrix}$$

The condition $Z^T\vec{e}_Z = \begin{bmatrix} 0 \\ 0 \end{bmatrix}$ gives us some relationships involving e_1, e_2, e_3, e_4, but not enough to guarantee that the components sum to 0.

Key takeaway: Remember that $X^T\vec{e}_X = \begin{bmatrix} 0 \\ 0 \end{bmatrix}$ implies that $\vec{e}_X$ is orthogonal to **any linear combination** of the columns of X. So, if you can create a column of all 1's using a linear combination of X's columns, then the components of $\vec{e}_X$ will sum to 0, **no matter which vector $\vec{y}$ you choose** to project onto colsp(X).

This may seem like a small algebraic fact, but it will be very important when we return to regression in Chapter 7.1. There, our matrix X will usually contain a column of all 1s, because our regression model will usually include an intercept term. When that happens, the error vector being orthogonal to the columns of X guarantees that the components of the error vector sum to 0. In other words, our fit regression model will satisfy a concrete constraint: the positive and negative errors must balance out overall.

Example: X with Orthogonal Columns. Let $X = \begin{bmatrix} 3/13 & -4/5 \\ 4/13 & 3/5 \\ 12/13 & 0 \end{bmatrix}$ and $\vec{y} = \begin{bmatrix} 1 \\ 0 \\ 1 \end{bmatrix}$.

Find the value of $\vec{w}^*$ that minimizes $\|\vec{y} - X\vec{w}\|^2$. What about X makes this easier than in other examples?

Solution

Notice that both of X's columns are orthogonal to each other, and are unit vectors, i.e. they are **orthonormal**. This means that

$$X^T X = \begin{bmatrix} 3/13 & 4/13 & 5/13 \\ -4/5 & 3/5 & 0 \end{bmatrix} \begin{bmatrix} 3/13 & -4/5 \\ 4/13 & 3/5 \\ 5/13 & 0 \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$$

is the identity matrix, so

$$\vec{w}^* = \underbrace{(X^T X)^{-1}}_{I} X^T \vec{y} = X^T \vec{y} = \begin{bmatrix} 15/13 \\ -4/5 \end{bmatrix}$$

Usually, our data doesn't come to us with orthogonal columns. But, using the Gram-Schmidt process introduced to you in Homework 8 and Chapter 6.5 (coming soon), you can convert a set of linearly independent vectors into an orthogonal set with the same span, allowing you to leverage that $Q^T Q = I$ and simplify the projection process.

Key takeaway: Orthonormal vectors are very easy to work with!

Example: Why Can't We Separate?. Why can't we separate

$$\vec{w}^* = (X^T X)^{-1} X^T \vec{y}$$

into

$$\vec{w}^* = X^{-1} \underbrace{(X^T)^{-1} X^T}_{I} \vec{y} = X^{-1} \vec{y}$$

in general?

Solution

In general, X **is not a square matrix**, so it can't be invertible!

If X is square and invertible, then the steps above are valid. But, if X is a square and invertible $n \times n$ matrix, then $\text{colsp}(X) = \mathbb{R}^n$, meaning that any $\vec{y}$ in $\mathbb{R}^n$ can be written as a linear combination of X's columns, meaning that $\vec{y}$ is already in $\text{colsp}(X)$, and there is no projection error (or need to do a projection in the first place).

Don't stop here: keep reading Chapter 6.4 to see what happens when there are infinitely many solutions to the normal equation. **It also contains a great summary of the ideas in Chapter 6.3, too.**

Chapter 6.4: The Complete Solution to the Normal Equations

6.4. The Complete Solution to the Normal Equations

In Chapter 6.3, we explored the idea of projecting a vector onto the column space of a matrix. If X has linearly independent columns, then the vector in colsp(X) that is closest to $\vec{y}$ is the vector $X\vec{w}^*$, where $\vec{w}^*$ is the unique solution to the normal equation,

$$X^T X\vec{w}^* = X^T\vec{y}$$

When X has linearly independent columns, then $X^T X$ is invertible, so we can solve for $\vec{w}^*$ uniquely.

$$\vec{w}^* = (X^T X)^{-1} X^T\vec{y}$$

What if X's Columns are Linearly Dependent?. In the case where X's columns are linearly dependent, we can't invert $X^T X$ to solve for $\vec{w}^*$. This means that

$$X^T X\vec{w} = X^T\vec{y}$$

has infinitely many solutions. Let's give more thought to what these solutions actually are.

First, note that all of these solutions for $\vec{w}^*$ **correspond to the same projection,** $\vec{p} = X\vec{w}^*$. The "best approximation" of $\vec{y}$ in colsp(X) is always just one vector; if there are infinitely many $\vec{w}^*$'s, that just means there are infinitely many ways of describing that one best approximation. Remember, if vectors are linearly independent, then any of their linear combinations can only be expressed in one way; if they are linearly dependent, then their linear combinations can be expressed in infinitely many ways.

In other words, if X has linearly dependent columns, then there are infinitely many $\vec{w}^*$'s that satisfy the normal equation, but they all correspond to the same projection $\vec{p} = X\vec{w}^*$ in the figure below.

Let me drive this point home further. Let's suppose both $\vec{w}_1$ and $\vec{w}_2$ satisfy

$$X^T X\vec{w} = X^T\vec{y}$$

Then,

$$X^T X\vec{w}_1 - X^T X\vec{w}_2 = \vec{y} - \vec{y} = \vec{0}$$

which means that

$$(X^T X)(\vec{w}_1 - \vec{w}_2) = \vec{0}$$

i.e. the difference between the two vectors, $\vec{w}_1 - \vec{w}_2$, is in nullsp($X^T X$). But, back in Chapter 5.3, we proved that $X^T X$ and X have the same null space, meaning any vector that gets sent to $\vec{0}$ by X also gets sent to $\vec{0}$ by $X^T X$, and vice versa.

So,

$$X(\vec{w}_1 - \vec{w}_2) = \vec{0}$$

too, but that just means

$$X\vec{w}_1 = X\vec{w}_2$$

meaning that even though $\vec{w}_1$ and $\vec{w}_2$ are different-looking coefficient vectors, they both still correspond to the **same** linear combination of X's columns!

Let's see how we can apply this to an example. Let $X = \begin{bmatrix} 3 & 1 & 0 \\ 6 & 2 & 1 \\ 3 & 1 & 0 \end{bmatrix}$ and $\vec{y} = \begin{bmatrix} 2 \\ 1 \\ 3 \end{bmatrix}$. This is an example of a matrix with linearly dependent columns, so there's no unique $\vec{w}^*$ that satisfies the normal equations.

Finding One Solution One way to find a possible vector $\vec{w}^*$ is to solve the normal equations. X^TX is not invertible, so we can't solve for $\vec{w}^*$ uniquely, but we can still try and find a solution.

Here's one approach: let's just toss out the linearly dependent columns of X and solve for $\vec{w}^*$ using the remaining columns. Then, $\vec{w}^*$ for the full X can use the same coefficients for the linearly independent columns, but 0s for the dependent ones. **Removing the linearly dependent columns does not change colsp(X) (i.e. the set of all linear combinations of X's columns), so the projection is the same.**

The easy solution is to keep columns 2 and 3, since their numbers are smallest. So, for now, let's say

$$X' = \begin{bmatrix} 1 & 0 \\ 2 & 1 \\ 1 & 0 \end{bmatrix}, \quad \vec{y} = \begin{bmatrix} 2 \\ 1 \\ 3 \end{bmatrix}$$

Here, $\vec{w}' = (X'^TX')^{-1}X'^T\vec{y} = \begin{bmatrix} 5/2 \\ -4 \end{bmatrix}$. I won't bore you with the calculations; you can verify them yourself.

Now, **one** possible $\vec{w}^*$ for the full X is $\begin{bmatrix} 0 \\ 5/2 \\ -4 \end{bmatrix}$, which keeps the same coefficients on columns 2 and 3 as in $\vec{w}'$, but 0 for the column we didn't use.

Finding All Solutions As I mentioned above, if there are infinitely many solutions to the normal equation, then the **difference** between any two solutions is in nullsp(X^TX), which is also nullsp(X). Put another way, if $\vec{w}_s$ satisfies the normal equations, then so does $\vec{w}_s + \vec{n}$ for any $\vec{n} \in$ nullsp(X^TX).

$$X^TX\vec{w}_s = X^T\vec{y}$$

$$X^TX(\vec{w}_s + \vec{n}) = X^TX\vec{w}_s + \underbrace{X^TX\vec{n}}_{\vec{0}\text{, by definition of null space}} = X^T\vec{y} + \vec{0}$$

So, once we have one $\vec{w}^*$, to get the rest, just add any vector in nullsp(X^TX) or nullsp(X) (since those are the same subspaces).

What is nullsp(X)? It's the set of vectors $\vec{v}$ such that $X\vec{v} = \vec{0}$.

In our particular example,

$$X = \begin{bmatrix} 3 & 1 & 0 \\ 6 & 2 & 1 \\ 3 & 1 & 0 \end{bmatrix}$$

we see that $\text{rank}(X) = 2$, so $\text{nullsp}(X)$ has a dimension of $3 - 2 = 1$ (by the rank-nullity theorem), so it's going to be the span of a single vector. All we need to do now is find **one** vector in $\text{nullsp}(X)$, and we will know that the null space is the set of scalar multiples of that vector.

Since column 1 is three times column 2, the vector $\vec{v} = \begin{bmatrix} 1 \\ -3 \\ 0 \end{bmatrix}$ must be in $\text{nullsp}(X)$.

$$X \begin{bmatrix} 1 \\ -3 \\ 0 \end{bmatrix} = \begin{bmatrix} 3 & 1 & 0 \\ 6 & 2 & 1 \\ 3 & 1 & 0 \end{bmatrix} \begin{bmatrix} 1 \\ -3 \\ 0 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix}$$

So, since $\text{nullsp}(X) = \text{nullsp}(X^T X) = \text{span}\left(\left\{\begin{bmatrix} 1 \\ -3 \\ 0 \end{bmatrix}\right\}\right)$, we know that the set of all possible $\vec{w}^*$'s is

$$\underbrace{\begin{bmatrix} 0 \\ 5/2 \\ -4 \end{bmatrix} + t \begin{bmatrix} 1 \\ -3 \\ 0 \end{bmatrix}, t \in \mathbb{R}}_{\text{there are infinitely many solutions to the normal equations, but they're all of this form}}$$

This is **not** a subspace, since it doesn't contain the zero vector.

Finding all solutions when there are infinitely many

If X has linearly dependent columns, and $\vec{w}'$ satisfies the normal equation, then $\vec{w}' + \vec{n}$ is also a solution to the normal equation for any $\vec{n} \in \text{nullsp}(X^T X)$, or equivalently, $\vec{n} \in \text{nullsp}(X)$.

There's another way to arrive at this set of possible $\vec{w}^*$'s: we can solve the normal equations directly. I wouldn't recommend this second approach since it's much longer, but I'll add it here for completeness.

$$X^T X = \begin{bmatrix} 3 & 6 & 3 \\ 1 & 2 & 1 \\ 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} 3 & 1 & 0 \\ 6 & 2 & 1 \\ 3 & 1 & 0 \end{bmatrix} = \begin{bmatrix} 54 & 18 & 6 \\ 18 & 6 & 2 \\ 6 & 2 & 1 \end{bmatrix}$$

$$X^T \vec{y} = \begin{bmatrix} 3 & 6 & 3 \\ 1 & 2 & 1 \\ 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} 2 \\ 1 \\ 3 \end{bmatrix} = \begin{bmatrix} 21 \\ 7 \\ 1 \end{bmatrix}$$

Then, the normal equations $X^T X \vec{w}^* = X^T \vec{y}$ give us

$$\begin{aligned} 54w_1^* + 18w_2^* + 6w_3^* &= 21 \\ 18w_1^* + 6w_2^* + 2w_3^* &= 7 \\ 6w_1^* + 2w_2^* + w_3^* &= 1 \end{aligned}$$

The first and second equations are just scalar multiples of each other, so we can disregard one of them, and solve for a form where we can use one unknown as a parameter for the other two. To illustrate, let's pick $w_1^* = t$.

$$\begin{aligned} 18t + 6w_2^* + 2w_3^* &= 7 \\ 6t + 2w_2^* + w_3^* &= 1 \end{aligned}$$

$(2) - 3 \cdot (3)$ gives us $w_3^* = -4$. Plugging this into both equations gives us

$$\begin{aligned} 18t + 6w_2^* - 8 = 7 &\implies 18t + 6w_2^* &&= 15 \\ 6t + 2w_2^* - 4 = 1 &\implies 6t + 2w_2^* &&= 5 \end{aligned}$$

These are now both the same equation; the first one is just 3 times the second. So, we can solve for w_2^* in terms of t:

$$w_2^* = \frac{5 - 6t}{2}$$

which gives us the complete solution

$$\vec{w}^* = \begin{bmatrix} t \\ \frac{5-6t}{2} \\ -4 \end{bmatrix}, t \in \mathbb{R}$$

This is the exact same line as using the null space approach! Plug in $t = 0$ to get $\begin{bmatrix} 0 \\ 5/2 \\ -4 \end{bmatrix}$, for example. The set of all possible $\vec{w}^*$'s is **not a subspace**, since it doesn't contain the zero vector.

The Projection Matrix. So far, we've established that the vector in colsp(X) that is closest to $\vec{y}$ is the vector $X\vec{w}^*$, where $\vec{w}^*$ is the solution to the normal equations,

$$X^T X \vec{w}^* = X^T \vec{y}$$

If $X^T X$ is invertible, then $\vec{w}^*$ is the unique vector

$$\vec{w}^* = (X^T X)^{-1} X^T \vec{y}$$

meaning that the vector in colsp(X) that is closest to $\vec{y}$ is

$$\vec{p} = X\vec{w}^* = X(X^T X)^{-1} X^T \vec{y}$$

You'll notice that the above expression also looks like a linear transformation applied to $\vec{y}$, where $\vec{y}$ is being multiplied by the matrix

$$P = X(X^TX)^{-1}X^T$$

The matrix P is called the **projection matrix**. In other classes, it is called the "hat matrix", because they might use $\hat{\text{w}}$ instead of $\vec{w}^*$ and $\hat{\text{y}}$ instead of $\vec{p}$, and in that notation, $\hat{\text{y}} = P\text{y}$, so P puts a "hat" on y. (I don't use hat notation in this class because drawing a hat on top of a vector is awkward. Doesn't $\hat{\vec{w}}$ look strange?)

So,

$$\vec{p} = X\vec{w}^* = P\vec{y}$$

shows us that there are two ways to interpret the act of projecting $\vec{y}$ onto colsp(X):

1. The resulting vector is some optimal linear combination of X's columns.
2. The resulting vector is the result of applying the linear transformation P to $\vec{y}$.

Let's work out an example. Suppose

$$X = \begin{bmatrix} 3 & 0 \\ 0 & 154 \\ 6 & 0 \end{bmatrix}, \quad \vec{y} = \begin{bmatrix} 1 \\ 2 \\ 3 \end{bmatrix}$$

X's columns are linearly independent, so X^TX is invertible, and

$$P = X(X^TX)^{-1}X^T$$

is well-defined.

```
X = np.array([[3, 0],
              [0, 154],
              [6, 0]])

P = X @ np.linalg.inv(X.T @ X) @ X.T
P
```

```
array([[0.2, 0. , 0.4],
       [0. , 1. , 0. ],
       [0.4, 0. , 0.8]])
```

```
P @ np.array([1, 2, 3])
```

```
array([1.4, 2. , 2.8])
```

$P = \begin{bmatrix} 0.2 & 0 & 0.4 \\ 0 & 1 & 0 \\ 0.4 & 0 & 0.8 \end{bmatrix}$ contains the information we need to project $\vec{y}$ onto colsp(X). Each row of P tells us the right mixture of $\vec{y}$'s components we need to construct the projection.

Notice that P's second row is $\begin{bmatrix} 0 & 1 & 0 \end{bmatrix}^T$. This came from the fact that X's first column had a second component of 0 while its second column had a non-zero second component but zeros in the other two components, meaning that we can scale X's second column to exactly match $\vec{y}$'s second component. Change the 154 in X to any other non-zero value and P won't change!

Additionally, if we consider some $\vec{y}$ that is already in colsp(X), then multiplying it by P doesn't change it! For example, if we set $\vec{y} = \begin{bmatrix} 3 \\ 154 \\ 6 \end{bmatrix}$ (the sum of X's columns), then $P\vec{y} = \vec{y}$.

```
P @ np.array([3, 154, 6])
```

```
array([  3., 154.,   6.])
```

Let's work through some examples that develop our intuition for P.

Example: Is P invertible? Suppose $P = X(X^TX)^{-1}X^T$ exists, meaning X^TX is invertible. Is P invertible? If so, what is its inverse?

Solution

Before we do any calculations, intuitively the answer should be **no**. Once we've projected onto colsp(X), we've lost information, since we went from an arbitrary vector in $\mathbb{R}^n$ to a vector in a smaller subspace, so it shouldn't be possible to reverse the projection. Put another way, two different vectors in $\mathbb{R}^n$ might have the same "shadow" onto colsp(X).

Even in the most recent example, $P = \begin{bmatrix} 0.2 & 0 & 0.4 \\ 0 & 1 & 0 \\ 0.4 & 0 & 0.8 \end{bmatrix}$ is not invertible, since column 3 is a multiple of column 1.

Let's think about this a bit more from the perspective of ranks. It turns out that rank(P) = rank(X); I've provided a proof of this at the bottom of the solutions box, but you might want to attempt it on your own for practice.

Remember that X is an $n \times d$ matrix, meaning rank(X) $\leq \min(n, d)$. X doesn't need to have a rank of n for X^TX to be invertible; it just needs to have a rank of d.

Since P is an $n \times n$ matrix, in general it won't be the case that rank(P) = n. To illustrate, in the example above where $P = \begin{bmatrix} 0.2 & 0 & 0.4 \\ 0 & 1 & 0 \\ 0.4 & 0 & 0.8 \end{bmatrix}$, X was a 3×2 matrix with rank 2.

The only case in which rank(P) = n is when rank(X) = n, which also only happens when X is an $n \times n$ square matrix that is also invertible. In such a case, colsp(X) = $\mathbb{R}^n$, and so we probably wouldn't set out to project onto colsp(X) in the first place, since any vector in $\mathbb{R}^n$ is already a linear combination of X's columns.

Extra: Proof that rank(P) = rank(X)

We can show that this is the case by showing that P and X both have the same column spaces. **This proof also helps explain why the normal equation, $X^TX\vec{w} = X^T\vec{y}$, always has at least one solution for $\vec{w}$, even when X^TX isn't invertible.**

1. **Show colsp(P) $\subseteq$ colsp(X)**, i.e. show that any vector in the column space of P is also in the column space of X.

   If $\vec{v} \in$ colsp(P), then $\vec{v}$ can be written as a linear combination of P's columns. Say $\vec{v} = P\vec{u}$ for some $\vec{u} \in \mathbb{R}^n$. Then,

   $$\vec{v} = P\vec{u} = X \underbrace{\left((X^TX)^{-1}X^T\vec{u}\right)}_{\text{some vector}}$$

   Here, we see that if $\vec{v} = P\vec{u}$, then $\vec{v}$ is also in X's column space. So,

   $$\vec{v} \text{ is a lin. comb. of } P\text{'s columns} \implies \vec{v} \text{ is a lin. comb. of } X\text{'s columns}$$

2. **Show colsp(X) $\subseteq$ colsp(P)**, i.e. show that any vector in the column space of X is also in the column space of P.

   This direction is a bit more involved. Let's start by considering some vector $\vec{v} \in$ colsp(X), meaning

   $$\vec{v} = X\vec{u}$$

   for some $\vec{u} \in \mathbb{R}^d$. What happens if we multiply both sides by P?

   $$\begin{aligned} \vec{v} &= X\vec{u} \\ P\vec{v} &= PX\vec{u} \\ P\vec{v} &= \underbrace{X(X^TX)^{-1}X^T}_{P} X\vec{u} \end{aligned}$$

Example: Is P orthogonal? Is P orthogonal?

Solution

No. Orthogonal matrices Q have the property that $Q^TQ = QQ^T = I$, meaning that

$$Q^T = Q^{-1}$$

But, as we saw, P is not invertible in general, so it can't satisfy this property. This tells us that P **does not perform a rotation**; projections are not rotations. Rotations can be undone but projections can't.

Example: Is P symmetric? Is P symmetric?

Solution

Yes. Symmetric matrices A have the property that $A^T = A$. We can show that P satisfies this property; to do so, we'll need to use the fact that $(AB)^T = B^TA^T$.

$$\begin{aligned} P^T &= \left(X(X^TX)^{-1}X^T\right)^T \\ &= \left(X^T\right)^T \left((X^TX)^{-1}\right)^T X^T \\ &= X(X^TX)^{-1}X^T \\ &= P \end{aligned}$$

Going from the second to the third line, we used the fact that X^TX is symmetric, and so is its inverse. Remember that X^TX is a square matrix consisting of the dot products of the columns of X with themselves.

Example: Is P idempotent? Recall, an idemponent matrix A satisfies $A^2 = A$. Is P idempotent?

Solution

Yes.

$$\begin{aligned} P^2 &= P \cdot P \\ &= \left(X(X^TX)^{-1}X^T\right) X(X^TX)^{-1}X^T \\ &= X \underbrace{\left((X^TX)^{-1}X^TX\right)}_{I}(X^TX)^{-1}X^T \\ &= X(X^TX)^{-1}X^T \\ &= P \end{aligned}$$

Intuitively, this means that $P^2\vec{y}$ is the same as $P\vec{y}$, meaning that once we've projected $\vec{y}$ onto colsp(X), projecting its projection $\vec{p}$ again onto colsp(X) gives us back the same $\vec{p}$, since $\vec{p}$ is already in colsp(X).

Example: What is PX, and why? What is PX? What does the result mean?

Solution

$$\begin{aligned} PX &= X(X^TX)^{-1}X^TX \\ &= X(X^TX)^{-1}X^TX \\ &= XI \\ &= X \end{aligned}$$

Interpret PX as a matrix made up of $P\vec{x}^{(1)}$, $P\vec{x}^{(2)}$, ..., $P\vec{x}^{(d)}$ as its columns. $P\vec{x}^{(i)}$ is the projection of $\vec{x}^{(i)}$ onto colsp(X), but since $\vec{x}^{(i)}$ is already in colsp(X), projecting it again onto colsp(X) gives us back the same $\vec{x}^{(i)}$. So, PX should just be X again.

Example: Rotations, Reflections, and Projections Suppose A is an arbitrary $n \times d$ matrix. Describe the conditions on A that make the corresponding linear transformation $f(\vec{x}) = A\vec{x}$ a...

1. Rotation
2. Reflection
3. Projection

Solution

1. **Rotation**: A should be **orthogonal**, meaning that

$$A^T A = AA^T = I$$

All orthogonal matrices have a determinant of 1 or -1, since $\det(A^T A) = \det(I)$ implies that $\det(A)^2 = 1$ (remember that $\det(A^T) = \det(A)$). If we want $f(\vec{x}) = A\vec{x}$ to be a rotation rather than a reflection, we also need

$$\det(A) = 1$$

To summarize: a rotation matrix is **orthogonal** and has determinant 1. Implicit here is the fact that A is square, meaning that $n = d$. It doesn't make sense to talk about a rotation that also changes the dimension of the space.

2. **Reflection**: The distinction between a reflection and a rotation is subtle in higher dimensions. We've mostly thought of "orthogonal" and "rotation" as being the same thing. But, for example,

$$A = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}$$

is orthogonal, but doesn't rotate vectors – it reflects them across the line $x_2 = x_1$ in $\mathbb{R}^2$. The determinant of A is -1.

In general, if A is orthogonal and has a determinant of -1, then it either involves a reflection or a reflection and a rotation.

Recall, the formula for a rotation matrix – as discussed in Example: Householder Reflection in Chapter 6.2 – is given by

$$A = I - 2\vec{u}\vec{u}^T$$

for some unit vector $\vec{u}$. This matrix reflects vectors across the hyperplane $\vec{u}^T\vec{x} = 0$ (remmeber, in $\mathbb{R}^2$ a line is a hyperplane; in $\mathbb{R}^3$ a plane is a hyperplane). If A is of that form, then it indeed satisfies the properties above: it is orthogonal and has determinant -1. (To show that its determinant is -1, we need knowledge of eigenvalues, which we'll introduce in Chapter 9.) It is furthermore symmetric, since

$$A^T = (I - 2\vec{u}\vec{u}^T)^T = I - 2\vec{u}\vec{u}^T = A$$

To summarize: a reflection matrix is **orthogonal, symmetric,** and has determinant -1. But, just because a matrix is orthogonal and has determinant -1, it doesn't have to be a reflection matrix: it could represent a reflection followed by a rotation.

3. **Projection**: To reason about the properties of a projection matrix, let's think about the matrix P from earlier in this section. Suppose we have a subspace, for which the columns of X are a basis (in other words, suppose X's columns are linearly independent, and the subspace in question is colsp(X)). Then, the matrix P that projects vectors onto that subspace is given by

$$P = X(X^T X)^{-1} X^T$$

As we saw above, P is both **idempotent**, meaning $P^2 = P$, and **symmetric**, meaning $P^T = P$.

To summarize: a projection matrix is **idempotent** and **symmetric.**

There are matrices – like $\begin{bmatrix} 1 & 1 \\ 0 & 0 \end{bmatrix}$ that are idempotent but not symmetric, and you can think of them as representing non-orthogonal projections. **This is not a kind of projection we've studied yet.** Let's

To summarize:

Transformation	Properties
Rotation	Orthogonal ($A^TA = I$), determinant = 1
Reflection	Orthogonal ($A^TA = I$), symmetric ($A^T = A$) if strict reflection, determinant = -1
Projection	Idempotent ($A^2 = A$), symmetric ($A^T = A$)

Summary. Let's take a step back and walk through our logic from Chapter 6.3 and here in Chapter 6.4 once more, since it's that important.

Suppose X is an $n \times d$ matrix and $\vec{y}$ is some vector in $\mathbb{R}^n$.

Orthogonal Projections

1. Our goal is to find the linear combination of X's columns that is closest to $\vec{y}$.
2. This boils down to finding the vector $\vec{w}$ that minimizes $\|\vec{y} - X\vec{w}\|^2$.
3. The vector $\vec{w}^*$ that minimizes $\|\vec{y} - X\vec{w}\|^2$ makes the resulting error vector,

$$\vec{e} = \vec{y} - X\vec{w}^*$$

   orthogonal to the columns of X.
4. The $\vec{w}^*$ that makes the error vector orthogonal to the columns of X is the one that satisfies the normal equation,

$$X^TX\vec{w}^* = X^T\vec{y}$$

5. If X^TX is invertible, which happens if and only if X's columns are linearly independent, then $\vec{w}^*$ is the unique vector

$$\vec{w}^* = (X^TX)^{-1}X^T\vec{y}$$

   Otherwise, there are infinitely many solutions to the normal equation. **All of these infinitely many solutions correspond to the same projection, $\vec{p} = X\vec{w}^*$.** If $\vec{w}'$ is one solution (which can be found by removing the linearly dependent columns of X), then all other solutions are of the form $\vec{w}' + \vec{n}$, where $\vec{n}$ is any vector in nullsp(X) = nullsp(X^TX).

The Projection Matrix Assuming X has linearly independent columns, the projection matrix is

$$P = X(X^TX)^{-1}X^T$$

P is defined such that $P\vec{y}$ is the vector in colsp(X) that is closest to $\vec{y}$. P is **symmetric** and **idemponent**, but not invertible nor orthogonal.

We're now **finally** ready to head back to the land of machine learning.

6.5. The Gram-Schmidt Process

This section can be thought of as a detour through the main storyline of the course. Chapter 7.1 is where I connect the idea of projecting $\vec{y}$ onto the column space of X to that of linear regression.

Instead, here I'll introduce a new algorithm that can be used to turn a collection of vectors into a more convenient form, and see how this can make the act of projecting $\vec{y}$ onto the column space of X much easier.

Why Orthogonalize?

Recall, a set of vectors $\vec{q}_1, \vec{q}_2, \ldots, \vec{q}_d \in \mathbb{R}^n$ are **orthonormal** if they are both:

1. pairwise orthogonal, meaning $\vec{q}_i \cdot \vec{q}_j = 0$ for all $i \neq j$, and
2. each vector is a unit vector, meaning $\|\vec{q}_i\| = 1$ for all i

Orthonormal vectors (and matrices containing them) are convenient to work with. For example, if Q's columns are the vectors $\vec{q}_1, \vec{q}_2, \ldots, \vec{q}_d$, then Q^TQ, the matrix containing the dot products of the columns of Q, is the identity matrix.

$$Q^TQ = \begin{bmatrix} \vec{q}_1 \cdot \vec{q}_1 & \vec{q}_1 \cdot \vec{q}_2 & \cdots & \vec{q}_1 \cdot \vec{q}_d \\ \vec{q}_2 \cdot \vec{q}_1 & \vec{q}_2 \cdot \vec{q}_2 & \cdots & \vec{q}_2 \cdot \vec{q}_d \\ \vdots & \vdots & \ddots & \vdots \\ \vec{q}_d \cdot \vec{q}_1 & \vec{q}_d \cdot \vec{q}_2 & \cdots & \vec{q}_d \cdot \vec{q}_d \end{bmatrix} = \begin{bmatrix} 1 & 0 & \cdots & 0 \\ 0 & 1 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & 1 \end{bmatrix} = I$$

(I know that I promised to use superscripts like $\vec{q}^{(1)}$ to denote columns of a matrix, but I have my reasons for doing it this way in this section.)

As we've seen in Chapter 6.3, when projecting $\vec{y}$ onto the column space of X, the matrix X^TX – and its inverse – plays a big role in finding the optimal coefficients $\vec{w}^*$ to multiply each column of X by. Most matrices, by default, **don't** have orthonormal columns. But if they did, then some of these calculations would be much, much simpler!

So, the goal here is to learn how to **turn a linearly independent set of vectors into an orthonormal set of vectors with the same span**, i.e. how to "orthogonalize" a set of vectors.

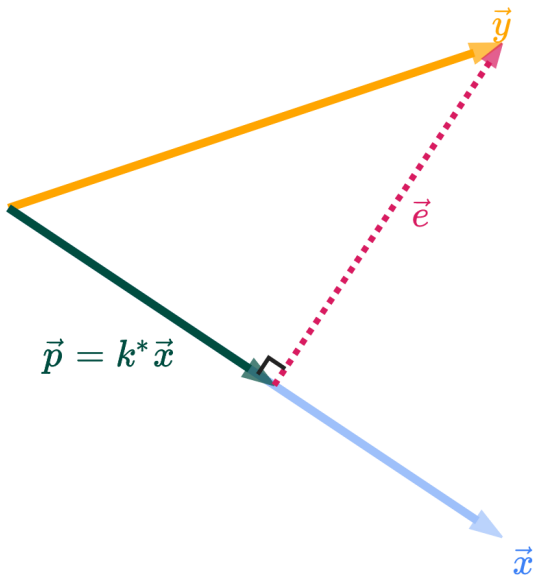
$$\underbrace{\vec{v}_1, \vec{v}_2, \ldots, \vec{v}_d}_{\text{linearly independent set of vectors}} \quad \rightarrow \quad \underbrace{\vec{q}_1, \vec{q}_2, \ldots, \vec{q}_d}_{\text{orthonormal set of vectors with the same span}}$$

The Algorithm

The algorithm that produces this orthonomal set of vectors is called the **Gram-Schmidt process**. It exploits the fact that when you project $\vec{y}$ onto $\vec{x}$, the error vector

$$\vec{e} = \vec{y} - \vec{p}$$

is orthogonal to $\vec{x}$.



If you look in the figure above, the vectors $\vec{x}$ and $\vec{e}$ are orthogonal, and have the same span as $\vec{y}$ and $\vec{x}$. The key takeaway is that if you'd like to "invent" vectors that are orthogonal to each other, you can construct them by iteratively projecting!

To illustrate how the algorithm works, let's use as an example the vectors

$$\vec{v}_1 = \begin{bmatrix} 1 \\ -1 \\ 1 \end{bmatrix}, \quad \vec{v}_2 = \begin{bmatrix} 1 \\ 0 \\ 1 \end{bmatrix}, \quad \vec{v}_3 = \begin{bmatrix} 1 \\ 1 \\ 2 \end{bmatrix}$$

These are three linearly independent vectors in $\mathbb{R}^3$, though they are not orthogonal. These vectors span some subspace S. Our goal is to find an orthonormal set of vectors that spans the same S. (In this case, S is all of $\mathbb{R}^3$, but in general this process works even if $d < n$.)

In what follows, let $\text{proj}_{\vec{x}}(\vec{y})$ be the projection of $\vec{y}$ onto $\vec{x}$, i.e. $\text{proj}_{\vec{x}}(\vec{y}) = \frac{\vec{y} \cdot \vec{x}}{\vec{x} \cdot \vec{x}} \vec{x}$.

- **Iteration 1**: Set $\vec{Q}_1 = \vec{v}_1$.

  In the first iteration, we simply take the first vector $\vec{v}_1$ and copy it to $\vec{Q}_1$. From now on, each new vector will be constructed to be orthogonal to all previously constructed $\vec{Q}_i$'s.

$$\vec{Q}_1 = \begin{bmatrix} 1 \\ -1 \\ 1 \end{bmatrix}$$

- **Iteration 2**: Set $\vec{Q}_2 = \vec{v}_2 - \text{proj}_{\vec{Q}_1}(\vec{v}_2)$.

  $\vec{Q}_2$ is the same as the error vector from projecting $\vec{v}_2$ onto $\vec{Q}_1$, which we know is orthogonal to $\vec{Q}_1$.

Even without actually calculating $\vec{Q}_2$, we can see that it, by definition, must be orthogonal to $\vec{Q}_1$.

$$\begin{aligned}
\vec{Q}_2 \cdot \vec{Q}_1 &= \left(\vec{v}_2 - \text{proj}_{\vec{Q}_1}(\vec{v}_2)\right) \cdot \vec{Q}_1 \\
&= \left(\vec{v}_2 - \frac{\vec{v}_2 \cdot \vec{Q}_1}{\vec{Q}_1 \cdot \vec{Q}_1}\vec{Q}_1\right) \cdot \vec{Q}_1 \\
&= \vec{v}_2 \cdot \vec{Q}_1 - \frac{\vec{v}_2 \cdot \vec{Q}_1}{\vec{Q}_1 \cdot \vec{Q}_1}\vec{Q}_1 \cdot \vec{Q}_1 \\
&= \vec{v}_2 \cdot \vec{Q}_1 - \vec{v}_2 \cdot \vec{Q}_1 \\
&= 0
\end{aligned}$$

With that understanding in mind, let's evaluate $\vec{Q}_2$ explicitly.

$$\vec{Q}_2 = \underbrace{\begin{bmatrix}1\\0\\1\end{bmatrix}}_{\vec{v}_2} - \underbrace{\frac{\begin{bmatrix}1\\0\\1\end{bmatrix} \cdot \begin{bmatrix}1\\-1\\1\end{bmatrix}}{\begin{bmatrix}1\\-1\\1\end{bmatrix} \cdot \begin{bmatrix}1\\-1\\1\end{bmatrix}} \begin{bmatrix}1\\-1\\1\end{bmatrix}}_{\text{proj}_{\vec{Q}_1}(\vec{v}_2)} = \begin{bmatrix}1\\0\\1\end{bmatrix} - \frac{2}{3}\begin{bmatrix}1\\-1\\1\end{bmatrix} = \begin{bmatrix}1/3\\2/3\\1/3\end{bmatrix}$$

- **Iteration 3**: Set $\vec{Q}_3 = \vec{v}_3 - \text{proj}_{\vec{Q}_1}(\vec{v}_3) - \text{proj}_{\vec{Q}_2}(\vec{v}_3)$.

  When constructed this way, $\vec{Q}_3$ is orthogonal to both $\vec{Q}_1$ and $\vec{Q}_2$. Think of it this way: $\text{span}(\{\vec{Q}_1, \vec{Q}_2\})$ is a plane in $\mathbb{R}^3$; after projecting $\vec{v}_3$ onto this plane, the remaining part of $\vec{v}_3$ that is orthogonal to the plane is $\vec{Q}_3$. If that doesn't make sense, execute $\vec{Q}_3 \cdot \vec{Q}_1$ and $\vec{Q}_3 \cdot \vec{Q}_2$ using the same general steps I followed in Iteration 2.

$$\begin{aligned}
\vec{Q}_3 &= \vec{v}_3 - \text{proj}_{\vec{Q}_1}(\vec{v}_3) - \text{proj}_{\vec{Q}_2}(\vec{v}_3) \\
&= \begin{bmatrix}1\\1\\2\end{bmatrix} - \underbrace{\frac{\begin{bmatrix}1\\1\\2\end{bmatrix} \cdot \begin{bmatrix}1\\-1\\1\end{bmatrix}}{\begin{bmatrix}1\\-1\\1\end{bmatrix} \cdot \begin{bmatrix}1\\-1\\1\end{bmatrix}} \begin{bmatrix}1\\-1\\1\end{bmatrix}}_{\text{proj}_{\vec{Q}_1}(\vec{v}_3)} - \underbrace{\frac{\begin{bmatrix}1\\1\\2\end{bmatrix} \cdot \begin{bmatrix}1/3\\2/3\\1/3\end{bmatrix}}{\begin{bmatrix}1/3\\2/3\\1/3\end{bmatrix} \cdot \begin{bmatrix}1/3\\2/3\\1/3\end{bmatrix}} \begin{bmatrix}1/3\\2/3\\1/3\end{bmatrix}}_{\text{proj}_{\vec{Q}_2}(\vec{v}_3)} \\
&= \begin{bmatrix}-1/2\\0\\1/2\end{bmatrix}
\end{aligned}$$

If there were more $\vec{v}_i$'s, we'd continue this process, each time constructing a new $\vec{Q}_i$ that is orthogonal to all previously constructed $\vec{Q}_i$'s by "subtracting off" the parts we've already accounted for through the earlier $\vec{Q}_i$'s.

Now, $\vec{Q}_1, \vec{Q}_2, \vec{Q}_3$ are ortho**gonal** to one another, but they are not yet unit vectors. To make them unit vectors, we simply need to divide each by its length.

$$\vec{q}_1 = \frac{\vec{Q}_1}{\|\vec{Q}_1\|} = \begin{bmatrix} 1/\sqrt{3} \\ -1/\sqrt{3} \\ 1/\sqrt{3} \end{bmatrix} \quad \vec{q}_2 = \frac{\vec{Q}_2}{\|\vec{Q}_2\|} = \begin{bmatrix} 1/\sqrt{6} \\ 2/\sqrt{6} \\ 1/\sqrt{6} \end{bmatrix} \quad \vec{q}_3 = \frac{\vec{Q}_3}{\|\vec{Q}_3\|} = \begin{bmatrix} -1/\sqrt{2} \\ 0 \\ 1/\sqrt{2} \end{bmatrix}$$

Now, the vectors $\vec{q}_1, \vec{q}_2, \vec{q}_3$ are ortho**normal** to one another, and they span the same subspace S as the vectors $\vec{v}_1, \vec{v}_2, \vec{v}_3$!

A quick note on signs: you could multiply any of the $\vec{q}_i$'s by -1 without changing the span of the collection, so if using a computer to compute the $\vec{q}_i$'s, you might see them come out with different signs.

Notice above that $\vec{v}_1$ in the top and $\vec{q}_1$ in the bottom are parallel, it's just that $\vec{q}_1$ is a unit vector. The three $\vec{v}_i$'s in the top figure are linearly independent but not orthogonal; the three $\vec{q}_i$'s in the bottom figure are, however, orthonormal.

Gram-Schmidt and Projection

In general, we're presumed to have a vector $\vec{y}$ that we'd like to approximate as a linear combination of matrix X's columns. Generally, X's columns are not orthonormal – they may not even be linearly independent.

If we:

1. Remove the linearly **de**pendent columns from X
2. Use the Gram-Schmidt process to orthonormalize the remaining columns, and store them in the columns of Q

Then, the matrix Q that results has the **same column space** as X, but solving the normal equations for Q and $\vec{y}$ is much simpler.

Our problem now is slightly different: we're trying to find the best linear combination of the columns of Q (not the columns of X) that approximates $\vec{y}$, i.e. we're projecting $\vec{y}$ onto the column space of Q. If we adjust our objective to this goal, then the best $\vec{w}^*$ – the one that minimizes $\|\vec{y} - Q\vec{w}\|^2$ – satisfies

$$Q^T Q\vec{w} = Q^T\vec{y}$$

But, $Q^T Q = I$ as I showed you at the start of this section, so this just reduces to

$$\vec{w}^* = Q^T\vec{y}$$

That's it! No inversion required: we can compute $\vec{w}^*$ with just a single matrix-vector multiplication.

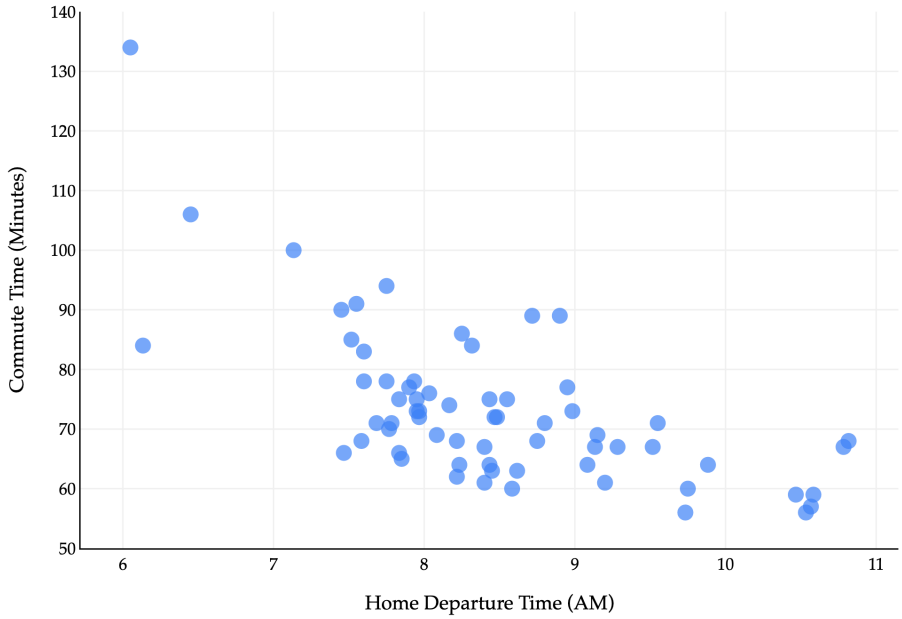
Regression using Linear Algebra

7.1. Regression using Linear Algebra

At times, Chapters 3-6 may have seemed like a bit of a detour from the main storyline of the course. It's now time to apply our understanding of vectors and matrices to the problem of linear regression.

Recap: The Modeling Recipe

In Chapter 1.2, we introduced the problem of predicting the length of our commute to school (y_i) as a function of the time we leave home (x_i).



That function was called a hypothesis function, denoted $h(x_i)$. Remember, the output of h is a predicted y-value.

$$\text{predicted commute time}_i = h(\text{departure hour}_i)$$

We looked at two types of hypothesis function:

- The constant model, $h(x_i) = w$
- The simple linear regression model, $h(x_i) = w_0 + w_1 x_i$

We'll focus on the latter, which was first introduced in Chapter 2.3. To find **optimal model parameters**, w_0^* (the best intercept) and w_1^* (the best slope), we followed the three-step modeling recipe:

1. Choose a model.

$$h(x_i) = w_0 + w_1 x_i$$

2. Choose a loss function. Our default choice has been squared loss:

$$L_{\text{sq}}(y_i, h(x_i)) = (y_i - h(x_i))^2$$

3. Minimize average loss (also known as empirical risk) to find optimal parameters. Average *squared* loss – also known as mean *squared* error – for any hypothesis function h, takes the form:

$$\frac{1}{n}\sum_{i=1}^{n}(y_i - h(x_i))^2$$

For the simple linear regression model, this becomes:

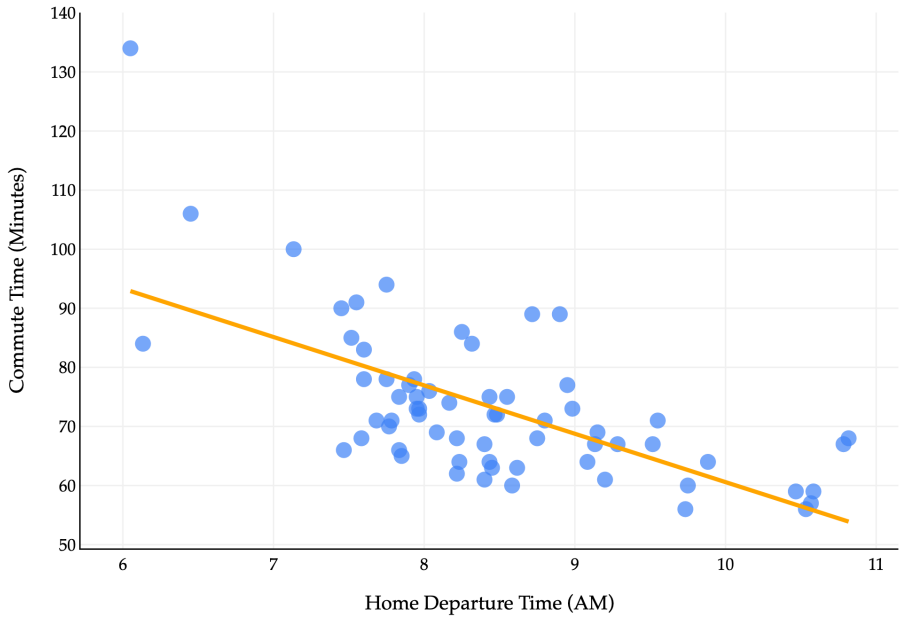
$$R_{\text{sq}}(w_0, w_1) = \frac{1}{n}\sum_{i=1}^{n}(y_i - (w_0 + w_1 x_i))^2$$

In Chapter 2.3, we used calculus to minimize $R_{\text{sq}}(w_0, w_1)$ to find the optimal parameters, w_0^* and w_1^*. This involved taking two partial derivatives, $\frac{\partial R_{\text{sq}}}{\partial w_0}$ and $\frac{\partial R_{\text{sq}}}{\partial w_1}$, setting them equal to zero, and solving the resulting system of equations. At the end of that process, we found

$$\underbrace{w_1^* = \frac{\sum_{i=1}^{n}(x_i - \bar{x})(y_i - \bar{y})}{\sum_{i=1}^{n}(x_i - \bar{x})^2} = r\frac{\sigma_y}{\sigma_x}}_{\text{optimal slope}}, \quad \underbrace{w_0^* = \bar{y} - w_1^*\bar{x}}_{\text{optimal intercept}}$$

where r is the correlation coefficient between x and y, and σ_x and σ_y are the standard deviations of x and y, respectively.

Those formulas, when applied to the dataset of commute times, describe the following line.



Here's the plan:

- In Chapter 7.1 (that's here), we'll see another way to find w_0^* and w_1^* that **doesn't** involve calculus, but rather leverages our new knowledge of vectors and matrices.
- In Chapter 7.2, we'll look at how to use this linear algebraic approach to add multiple input variables to our model. The orange line above only uses one input variable, departure time, but we might want to incorporate day of the week, weather, etc.
- In homework, you will get a *taste* of how to approach the process of adding new features, and why adding

lots of features doesn't necessarily lead to better predictions on real-world, unseen data.

The Design Matrix

Big idea: How can we express mean squared error,

$$R_{\text{sq}}(w_0, w_1) = \frac{1}{n}\sum_{i=1}^{n} \underbrace{(y_i - (w_0 + w_1 x_i))^2}_{\text{(actual−predicted)}^2}$$

in terms of vectors and matrices? If we can do so, then perhaps there will be another way to find w_0^* and w_1^* without needing to take partial derivatives. This will make our life a lot easier when we add more features to our model.

Remember that if $\vec{x} \in \mathbb{R}^n$, then $\|\vec{x}\|^2 = \sum_{i=1}^{n} x_i^2$. In the formula for R_{sq} above, I've colored $\sum_{i=1}^{n}$ and $\cdot^2$ in red to try and make the case that R_{sq} **looks a lot like the squared norm of vector that contains the errors of our predictions**. Let's try and define this vector.

Consider a dataset of n points, $(x_1, y_1), (x_2, y_2), \ldots, (x_n, y_n)$, on which we'd like to fit a simple linear regression model $h(x_i) = w_0 + w_1 x_i$.

- The **observation vector**, $\vec{y}$, is a vector in $\mathbb{R}^n$ with the n y-values from the dataset.

$$\vec{y} = \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_n \end{bmatrix} = \underbrace{\begin{bmatrix} \text{actual commute time}_1 \\ \text{actual commute time}_2 \\ \vdots \\ \text{actual commute time}_n \end{bmatrix}}_{\text{for the commute times example}}$$

  This vector contains our "right answers" that we are trying to predict as best as we can.

- The **prediction vector**, $\vec{p}$, is a vector in $\mathbb{R}^n$ with the n predicted values from the model.

$$\vec{p} = \begin{bmatrix} h(x_1) \\ h(x_2) \\ \vdots \\ h(x_n) \end{bmatrix} = \begin{bmatrix} w_0 + w_1 x_1 \\ w_0 + w_1 x_2 \\ \vdots \\ w_0 + w_1 x_n \end{bmatrix}$$

  We want $\vec{p}$ to be as close as possible to $\vec{y}$.

We can express the prediction vector, $\vec{p}$, as a matrix-vector product!

$$\vec{p} = \begin{bmatrix} w_0 + w_1 x_1 \\ w_0 + w_1 x_2 \\ \vdots \\ w_0 + w_1 x_n \end{bmatrix} = w_0 \begin{bmatrix} 1 \\ 1 \\ \vdots \\ 1 \end{bmatrix} + w_1 \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix} = \begin{bmatrix} 1 & x_1 \\ 1 & x_2 \\ \vdots & \vdots \\ 1 & x_n \end{bmatrix} \begin{bmatrix} w_0 \\ w_1 \end{bmatrix} = X\vec{w}$$

In $\vec{p} = X\vec{w}$,

- X, called the **design matrix**, is an $n \times 2$ matrix with its first column being all 1s and its second column being the inputs $x_1, x_2, \ldots, x_n$. It's the most important among these definitions, hence why this section is titled "The Design Matrix".

$$X = \begin{bmatrix} 1 & x_1 \\ 1 & x_2 \\ \vdots & \vdots \\ 1 & x_n \end{bmatrix} = \underbrace{\begin{bmatrix} 1 & \text{departure time}_1 \\ 1 & \text{departure time}_2 \\ \vdots & \vdots \\ 1 & \text{departure time}_n \end{bmatrix}}_{\text{for the commute times example}}$$

  I haven't been able to find a good reason for why this is called the design matrix; my best explanation is that it has to do with how we designed our model. **The column of 1s is there for our intercept term, w_0.**

- The **parameter vector**, $\vec{w}$, is a 2×1 vector containing our model's parameters, w_0 and w_1.

$$\vec{w} = \begin{bmatrix} w_0 \\ w_1 \end{bmatrix}$$

  We're trying to find the best choice of $\vec{w}$.

Remember, our goal is to convert

$$R_{\text{sq}}(w_0, w_1) = \frac{1}{n} \sum_{i=1}^{n} (y_i - (w_0 + w_1 x_i))^2$$

into an expression that involves vectors and matrices. Using our new definitions, we're almost there! R_{sq} involves a sum of squared errors. So, let's define the **error vector**, $\vec{e}$, as the difference between $\vec{y}$ and $\vec{p} = X\vec{w}$.

$$\vec{e} = \vec{y} - \vec{p} = \vec{y} - X\vec{w} = \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_n \end{bmatrix} - \begin{bmatrix} w_0 + w_1 x_1 \\ w_0 + w_1 x_2 \\ \vdots \\ w_0 + w_1 x_n \end{bmatrix} = \begin{bmatrix} y_1 - (w_0 + w_1 x_1) \\ y_2 - (w_0 + w_1 x_2) \\ \vdots \\ y_n - (w_0 + w_1 x_n) \end{bmatrix}$$

The components of $\vec{e}$ are the quantities being summed and squared in R_{sq}. How do we get that sum and square? By taking the norm, as I alluded to earlier.

$$\|\vec{e}\|^2 = \sum_{i=1}^{n} e_i^2 = \sum_{i=1}^{n} (y_i - (w_0 + w_1 x_i))^2$$

$\|\vec{e}\|^2$ is *almost* R_{sq}, it's just missing a $\frac{1}{n}$ up front. So,

$$\boxed{R_{\text{sq}}(\vec{w}) = \frac{1}{n} \|\vec{e}\|^2 = \frac{1}{n} \|\vec{y} - X\vec{w}\|^2}$$

We've completed our "conversion" of R_{sq} into a vector-based expression.

The Normal Equations Return

Now, our goal is to find the vector $\vec{w}^*$ that minimizes

$$R_{\text{sq}}(\vec{w}) = \frac{1}{n}\|\vec{y} - X\vec{w}\|^2$$

This sounds like a familiar problem. $X\vec{w}$ is a vector in colsp(X), so it looks like we want to find the vector in colsp(X) that is closest to $\vec{y}$. The extra $\frac{1}{n}$ up front doesn't change the optimization problem; as we studied in our calculus review at the start of the semester (and in the examples in Appendix 2), the minimizer of $cf(w)$ is the same as the minimizer of $f(w)$ when $c > 0$.

This is exactly the problem we solved in Chapter 6.3!

In Chapter 6.3, we found that the vector in colsp(X) that is closest to $\vec{y}$ is the **orthogonal projection** of $\vec{y}$ onto colsp(X), which is the vector

$$\vec{p} = X\vec{w}^*$$

where $\vec{w}^*$ is chosen to satisfy the **normal equation**,

$$X^TX\vec{w}^* = X^T\vec{y}$$

Note that both **p**rojection and **p**rediction start with "p", making it easy to remember that they're related, and the vector $\vec{p}$ can be interpreted as either. Predictions are projections onto the column space of the design matrix.

What makes the projection "orthogonal" is that the projection $\vec{p} = X\vec{w}^*$ has an error vector $\vec{e} = \vec{y} - X\vec{w}^*$ that is orthogonal to every vector in colsp(X).

$$X^T\vec{e} = \vec{0}$$

When X's columns are linearly independent, then X^TX is invertible, and the unique solution $\vec{w}^*$ to the normal equation is

$$\vec{w}^* = (X^TX)^{-1}X^T\vec{y}$$

The vector $\vec{w}^*$, in this context, is called our **optimal parameter vector**, since it contains our optimal choices of parameters, w_0^* and w_1^*. Note that

$$X = \begin{bmatrix} 1 & x_1 \\ 1 & x_2 \\ \vdots & \vdots \\ 1 & x_n \end{bmatrix}$$

is invertible as long as **not** all of the x_i's are the same. If they are all the same, then X's second column is a multiple of its first column, and so X is not of full rank, and neither is X^TX (since both matrices have the same rank). This corresponds to the case where the data points all lie on a single **vertical** line, so no line of the form $h(x_i) = w_0^* + w_1^*x_i$ can pass through them.

So, to summarize,

$$R_{\text{sq}}(\vec{w}) = \frac{1}{n}\|\vec{y} - X\vec{w}\|^2$$

is minimized when

$$X^TX\vec{w}^* = X^T\vec{y} \implies \underbrace{\vec{w}^* = (X^TX)^{-1}X^T\vec{y}}_{\text{if } X\text{'s columns are linearly independent}}$$

To be clear, when X is the $n \times 2$ design matrix and $\vec{y}$ is a vector with our n y-values, then $\vec{w}^*$ as defined above contains **the exact same values as** our calculus-based formulas for w_0^* and w_1^*!

Errors Sum to 0. Since the first column of the design matrix X is a column of all 1s, the condition $X^T\vec{e} = \vec{0}$ tells us that the error vector $\vec{e}$ must be orthogonal to $\begin{bmatrix} 1 \\ 1 \\ \vdots \\ 1 \end{bmatrix}$. That means

$$\begin{bmatrix} 1 \\ 1 \\ \vdots \\ 1 \end{bmatrix} \cdot \begin{bmatrix} e_1 \\ e_2 \\ \vdots \\ e_n \end{bmatrix} = e_1 + e_2 + \cdots + e_n = 0$$

So, for a fit simple linear regression model with an intercept term, the components of the error vector are guaranteed to sum to 0. This is worth caring about because it gives us a concrete constraint that our best fit model must satisfy: some errors may be positive and some negative, but overall they must balance out exactly.

In Chapter 7.2, we'll explore the process of introducing additional features to our models. In that process, we might sometimes choose to build a model without an intercept term. If we do, then the design matrix no longer has to contain a column of all 1s, and this guarantee no longer has to hold. So this is not a universal fact about every linear model; it is a consequence of fitting by least squares **and** including an intercept term.

Implementing $\vec{w}^*$ in Code

While you may have to wait until you complete Homework 7 to see a proof that

$$\vec{w}^* = (X^TX)^{-1}X^T\vec{y} = \begin{bmatrix} \bar{y} - r\frac{\sigma_y}{\sigma_x}\bar{x} \\ r\frac{\sigma_y}{\sigma_x} \end{bmatrix}$$

I think it's worthwhile for me to give you a preview that these are equivalent, through the lens of code. The `commutes` DataFrame in `pandas`, stored below, contains our commute times data. The first column, `departure_hour`, is our x-variable, while `minutes` is

```
commutes
```

	departure_hour	minutes
0	10.816667	68.0
1	7.750000	94.0
2	8.450000	63.0
3	7.133333	100.0
4	9.150000	69.0
...	...	...
60	7.516667	85.0
61	7.550000	91.0
62	7.583333	68.0
63	7.450000	90.0
64	7.600000	83.0

65 rows × 2 columns

Approach 1: The Old Formulas. We've done this before, but let's implement our original formulas for w_0^* and w_1^* in code.

$$w_1^* = \frac{\sum_{i=1}^n (x_i - \bar{x})(y_i - \bar{y})}{\sum_{i=1}^n (x_i - \bar{x})^2} = r\frac{\sigma_y}{\sigma_x}, \quad w_0^* = \bar{y} - w_1^* \bar{x}$$

```
def optimal_slope(x, y):
    r = np.corrcoef(x, y)[0, 1]
    return r * np.std(y) / np.std(x)

optimal_slope(commutes['departure_hour'], commutes['minutes'])
```

```
-8.186941724265557
```

```
def optimal_intercept(x, y):
    return np.mean(y) - optimal_slope(x, y) * np.mean(x)

optimal_intercept(commutes['departure_hour'], commutes['minutes'])
```

```
142.44824158772875
```

To actually use these optimal parameters to make predictions, we need to execute $w_0^* + w_1^* x_i$ ourselves.

```
def predicted_commute(departure_hour):
    w0_star = optimal_intercept(commutes['departure_hour'], commutes['minutes'])
    w1_star = optimal_slope(commutes['departure_hour'], commutes['minutes'])
    return w0_star + w1_star * departure_hour

predicted_commute(15)
```

```
19.644115723745387
```

Approach 2: The Normal Equations. Now, we'll use the fact that

$$\vec{w}^* = (X^T X)^{-1} X^T \vec{y}$$

to find the optimal slope and intercept. Remember, Chapter 6.2 told us that it's generally a bad idea to use `np.linalg.inv` directly; instead, we should use `np.linalg.solve` to solve the normal equations.

```
# We need to make the n x 2 design matrix X, which has a column of 1's
# to account for the intercept.
commutes.loc[:, '1'] = 1
commutes[['1', 'departure_hour']]
```

	1	departure_hour
0	1	10.816667
1	1	7.750000
2	1	8.450000
3	1	7.133333
4	1	9.150000
...	...	...
60	1	7.516667
61	1	7.550000
62	1	7.583333
63	1	7.450000
64	1	7.600000

65 rows × 2 columns

```
X = commutes[['1', 'departure_hour']]
y = commutes['minutes']

w_star = np.linalg.solve(X.T @ X, X.T @ y)
w_star
```

```
array([142.44824159,  -8.18694172])
```

`w_star` contains the same values as our calculus-based formulas for w_0^* and w_1^* from above!
To use `w_star` to make predictions, we eventually need to evaluate $w_0^* + w_1^* x_i$, but it turns out this can be expressed as a **dot product** between $\vec{w}^*$ and $\begin{bmatrix} 1 \\ x_i \end{bmatrix}$. More on this in Chapter 7.2.

```
# Same as w0_star + w1_star * 15
np.dot(w_star, np.array([1, 15]))
```

```
19.644115723744356
```

```
# Also the same
w_star @ np.array([1, 15])
```

```
19.644115723744356
```

Approach 3: `sklearn`. Finally, we can use `sklearn`'s `LinearRegression` class to find w_0^* and w_1^*.

```
from sklearn.linear_model import LinearRegression

model = LinearRegression()

# By default, sklearn knows to minimize mean squared error,
# and knows to add an intercept term to the model.
# If you want to turn off the intercept, you can set `fit_intercept=False`.
model.fit(commutes[['departure_hour']].to_numpy(), commutes['minutes'].to_numpy())

print(model.intercept_, model.coef_)

142.4482415877287 [ -8.18694172]
```

Once again, we get the same values as the prior two approaches! None of this should be a surprise, but it's reassuring to see that (1) our calculus and linear algebra approaches are consistent, and (2) both of those are equivalent to using `sklearn`. All three approaches involve the same three step modeling recipe.

And finally, to use `model` to make predictions, we don't need to do any of the math ourselves: we can use the `predict` method.

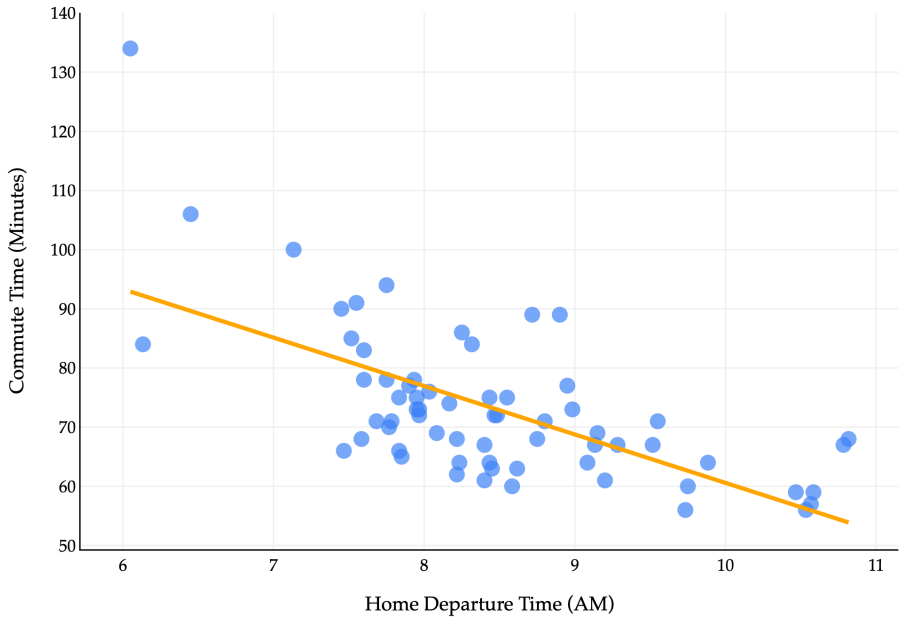
```
# The same, once again!
# Notice that the input is a 2D array.
# More on this in Chapter 7.2.
model.predict([[15]])

array([19.64411572])
```

The Three Pictures

Throughout Chapter 2.3 and here in Chapter 7.1, we've seen three different diagrams involving the simple linear regression model, and it's important to understand what each one depicts.

Picture 1: The Data and Model ($\mathbb{R}^2$). The first and most intuitive diagram is the one that shows the original points $(x_1, y_1), (x_2, y_2), \ldots, (x_n, y_n)$ in $\mathbb{R}^2$ with the fit line $h(x_i) = w_0^* + w_1^* x_i$.

In the scatter plot below, the vertical errors from the fitted line – some of which are positive, some of which are negative – add up to 0, as we discussed in the section above titled Errors Sum to 0.



Picture 2: The Projection View ($\mathbb{R}^n$). Then, there is the diagram that shows that the optimal predictions, $\vec{p} = X\vec{w}^*$, are the orthogonal projections of $\vec{y}$ onto colsp(X). Unlike the above plot, which is in $\mathbb{R}^2$, this one is in $\mathbb{R}^n$, where n is our number of data points.

Picture 3: The Loss Surface ($\mathbb{R}^3$). The final relevant picture is that of the graph of mean squared error, i.e. the graph whose x-axis is $\vec{w}_0$, y-axis is $\vec{w}_1$, and whose z-axis is $R_{\text{sq}}(\vec{w})$. The w_0 and w_1 coordinates of the "bottom" of the graph correspond to the optimal parameters in $\vec{w}^*$, which are the weights in the linear combination of the columns of X that is closest to $\vec{y}$.

You should take time to understand how these are all related.

- Our goal is to find the best fitting line in Picture 1.
- To do that, we minimize mean squared error in Picture 3.
- To do that, we project $\vec{y}$ onto colsp(X) in Picture 2.

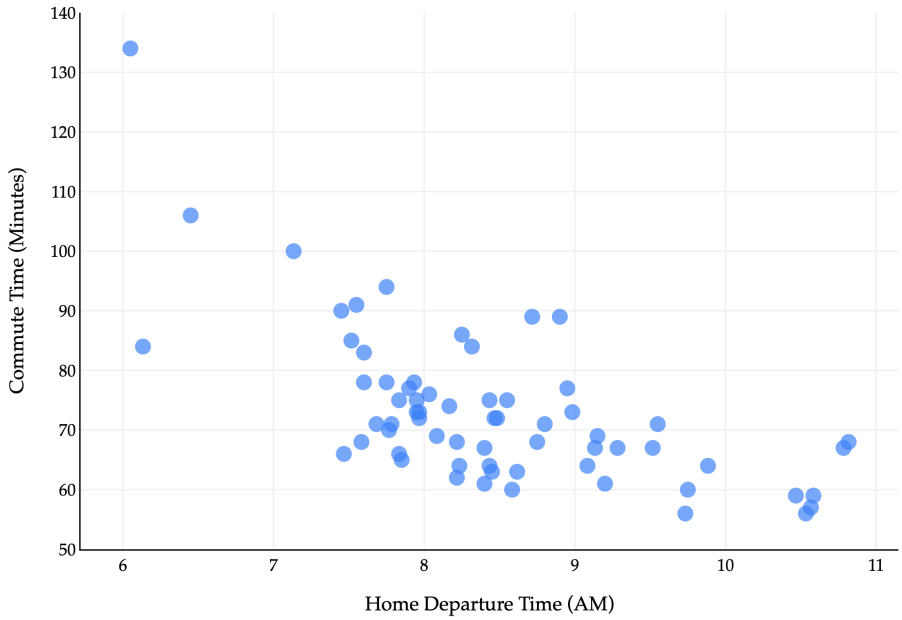
With this in mind, let's move to Chapter 7.2, where we'll see how to extend our new linear algebraic approach to multiple input variables, in what's called **multiple linear regression**.

7.2. Multiple Linear Regression

In Chapter 7.1, we learned how to use our knowledge of spans and projections to recast the problem of finding optimal model parameters for the simple linear regression model, $h(x_i) = w_0 + w_1 x_i$. Along the way, we defined new characters, most importantly, the design matrix

$$X = \begin{bmatrix} 1 & x_1 \\ 1 & x_2 \\ \vdots & \vdots \\ 1 & x_n \end{bmatrix}$$

Here, we'll look at how and why to extend X to include multiple input features, in what's called **multiple linear regression**.



Incorporating Multiple Features

Motivation. The commute times dataset that we were reintroduced to in Chapter 7.1 has several columns in it, but we've only used one – `departure_hour` – as an input variable.

```
commutes_full.head()
```

	date	day	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]
0	5/15/2023	Mon	2023-05-15 10:49:00	[illegible]	2023-05-15 11:57:00	[illegible]	stopped for coffee	2023-05-15 17:10:00	[illegible]	2023-05-15 18:22:00	[illegible]	NaN	68.0	10.816667	[illegible]	72.0	17.166667	[illegible]	15
1	5/16/2023	Tue	2023-05-16 07:45:00	[illegible]	2023-05-16 09:19:00	[illegible]	stopped for gas by exit 10 of state route 16	NaN	NaN	NaN	NaN	NaN	94.0	7.750000	[illegible]	NaN	NaN	NaN	16
2	5/22/2023	Mon	2023-05-22 08:27:00	[illegible]	2023-05-22 09:30:00	[illegible]	NaN	2023-05-22 15:54:00	[illegible]	2023-05-22 17:16:00	[illegible]	NaN	63.0	8.450000	[illegible]	82.0	15.900000	[illegible]	22
3	5/23/2023	Tue	2023-05-23 07:08:00	[illegible]	2023-05-23 08:48:00	[illegible]	NaN	NaN	NaN	NaN	NaN	NaN	100.0	7.133333	[illegible]	NaN	NaN	NaN	23
4	5/30/2023	Tue	2023-05-30 09:09:00	[illegible]	2023-05-30 10:18:00	[illegible]	South Gate did not open, drove around to North...	2023-05-30 17:07:00	[illegible]	2023-05-30 18:23:00	[illegible]	NaN	69.0	9.150000	[illegible]	76.0	17.116667	[illegible]	30

Let's take a look at a subset of them.

```
commutes_full[['departure_hour', 'day', 'day_of_month', 'minutes']].head()
```

	departure_hour	day	day_of_month	minutes
0	10.816667	Mon	15	68.0
1	7.750000	Tue	16	94.0
2	8.450000	Mon	22	63.0
3	7.133333	Tue	23	100.0
4	9.150000	Tue	30	69.0

For example, the first row above tells us that the first recorded commute was on a Monday, which was the 15th of that month.

Using `day` seems like it would be a good idea, since traffic patterns likely differ based on the day of the week. But, it's not clear how to use strings, like `'Tue'`, in a linear model. We'll address this in a moment. For now,

suppose we'd like to fit a linear model that predicts commute time in `minutes` using **both** `departure_hour` and `day_of_month` ("dom" for short).

Such a hypothesis function is of the form

$$\underbrace{h(\text{departure hour}_i, \text{dom}_i)}_{=\text{pred. commute}_i} = w_0 + w_1 \cdot \text{departure hour}_i + w_2 \cdot \text{dom}_i$$

which is a **plane** in $\mathbb{R}^3$. Once we figure out how to find w_0^*, w_1^*, and w_2^*, we'll visualize the resulting plane, not to worry.

(To be crystal clear, the $\cdot$'s in the above equations refer to "regular" scalar multiplication, not the dot product. There are no vectors in the equation above.)

But, **how** do we find these three optimal model parameters? If we consult the modeling recipe, and choose squared loss, we need to find the values of w_0^*, w_1^*, and w_2^* that minimize

$$R_{\text{sq}}(w_0, w_1, w_2) = \frac{1}{n}\sum_{i=1}^{n}(y_i - (w_0 + w_1 \cdot \text{departure hour}_i + w_2 \cdot \text{dom}_i))^2$$

The solution is to use what we know about spans, projections, and the design matrix. The design matrix for this problem will have **3** columns: a column of 1's for the intercept, a column of `departure_hour` values, and a column of `dom` values.

$$\begin{aligned}\vec{p} &= \begin{bmatrix} w_0 + w_1 \cdot \text{departure hour}_1 + w_2 \cdot \text{dom}_1 \\ w_0 + w_1 \cdot \text{departure hour}_2 + w_2 \cdot \text{dom}_2 \\ \vdots \\ w_0 + w_1 \cdot \text{departure hour}_n + w_2 \cdot \text{dom}_n \end{bmatrix} \\ &= \underbrace{\begin{bmatrix} 1 & \text{departure hour}_1 & \text{dom}_1 \\ 1 & \text{departure hour}_2 & \text{dom}_2 \\ \vdots & \vdots & \vdots \\ 1 & \text{departure hour}_n & \text{dom}_n \end{bmatrix}}_{X} \underbrace{\begin{bmatrix} w_0 \\ w_1 \\ w_2 \end{bmatrix}}_{\vec{w}}\end{aligned}$$

To find $\vec{w}^*$, all we need to do is define the observation vector, $\vec{y} = \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_n \end{bmatrix}$, and find the $\vec{w}^*$ that minimizes

$$R_{\text{sq}}(\vec{w}) = \frac{1}{n}\left\|\vec{y} - X\vec{w}\right\|^2$$

which is a solved problem at this point: it's the $\vec{w}^*$ that satisfies the normal equation, $X^T X \vec{w}^* = X^T \vec{y}$.

Generalized Notation. Let me try and state this problem in slightly more general terms, where we have d features rather than just 1 or 2.

As before, subscripts distinguish between **individuals (rows)** in our dataset, and we will try and use superscripts (in parentheses) to distinguish between **features (columns)**. Specifically, we'll use the notation $x_i^{(j)}$ to

represent the j-th feature for the i-th individual. In the example above, $x_5^{(1)}$ is the departure hour for the 5th row in the dataset. Think of $x^{(1)}$, $x^{(2)}$, ... as new variable names, like new letters.

Consider the following example dataset.

	departure_hour	day_of_month	minutes
0	10.816667	15	68.0
1	7.750000	16	94.0
2	8.450000	22	63.0

There are $n = 3$ rows, each of which has $d = 2$ features (`minutes` is **not** a feature, it's what we're trying to predict).

We can represent each row (day) with a **feature vector**, $\vec{x}_i = \begin{bmatrix} x_i^{(1)} \\ x_i^{(2)} \end{bmatrix}$. A feature vector contains all of the information we're using to make predictions for the i-th individual.

$$\vec{x}_1 = \begin{bmatrix} 10.816667 \\ 15 \end{bmatrix}, \qquad \vec{x}_2 = \begin{bmatrix} 7.75 \\ 16 \end{bmatrix}, \qquad \vec{x}_3 = \begin{bmatrix} 8.45 \\ 22 \end{bmatrix}$$

In general,

$$\vec{x}_i = \begin{bmatrix} x_i^{(1)} \\ x_i^{(2)} \\ \vdots \\ x_i^{(d)} \end{bmatrix}$$

Note that if our model has d features, then there are $d + 1$ parameters, w_0, w_1, w_2, ..., w_d. There is one parameter for each feature, **plus** one for the intercept. So, the generalized **multiple** linear regression model has a hypothesis function of the form

$$h(\vec{x}_i) = w_0 + w_1 x_i^{(1)} + w_2 x_i^{(2)} + \ldots + w_d x_i^{(d)}$$

It'd be great if we could express our hypothesis function h as a dot product between a parameter vector $\vec{w}$ and a feature vector $\vec{x}_i$, but we can't: $\vec{w}$ has $d + 1$ components while $\vec{x}_i$ has d components.

To address this issue, we'll define the **augmented feature vector**, $\text{Aug}(\vec{x}_i)$, which is the vector obtained by adding a 1 to the front of feature vector $\vec{x}_i$ (much like the design matrix X has a column of 1's).

$$\text{Aug}(\vec{x}_i) = \begin{bmatrix} 1 \\ x_i^{(1)} \\ x_i^{(2)} \\ \vdots \\ x_i^{(d)} \end{bmatrix}$$

Then, the multiple linear regression model can be written as

$$\boxed{h(\vec{x}_i) = w_0 + w_1 x_i^{(1)} + w_2 x_i^{(2)} + \ldots + w_d x_i^{(d)} = \vec{w} \cdot \text{Aug}(\vec{x}_i)}$$

The General Problem. Now we can finally state the problem of multiple linear regression in its most general form. Suppose we have n data points, $(\vec{x}_1, y_1), (\vec{x}_2, y_2), \ldots, (\vec{x}_n, y_n)$, where each $\vec{x}_i$ is a feature vector of d features:

$$\vec{x}_i = \begin{bmatrix} x_i^{(1)} \\ x_i^{(2)} \\ \vdots \\ x_i^{(d)} \end{bmatrix}$$

We'd like to find a good **linear** model,

$$h(\vec{x}_i) = w_0 + w_1 x_i^{(1)} + w_2 x_i^{(2)} + \ldots + w_d x_i^{(d)} = \vec{w} \cdot \text{Aug}(\vec{x}_i)$$

By **good**, we mean that we'd like to find optimal parameters w_0^*, w_1^*, ..., w_d^* that minimize mean squared error:

$$\begin{aligned} R_{\text{sq}}(\vec{w}) &= \frac{1}{n} \sum_{i=1}^{n} (y_i - h(\vec{x}_i))^2 \\ &= \frac{1}{n} \sum_{i=1}^{n} \left(y_i - (w_0 + w_1 x_i^{(1)} + w_2 x_i^{(2)} + \ldots + w_d x_i^{(d)}) \right)^2 \\ &= \frac{1}{n} \sum_{i=1}^{n} \left(y_i - \vec{w} \cdot \text{Aug}(\vec{x}_i) \right)^2 \\ &= \frac{1}{n} \| \vec{y} - X\vec{w} \|^2 \end{aligned}$$

all equivalent ways of writing mean squared error!

The solution is to define the $n \times (d+1)$ design matrix X and n-dimensional observation vector $\vec{y}$:

$$X = \begin{bmatrix} 1 & x_1^{(1)} & x_1^{(2)} & \ldots & x_1^{(d)} \\ 1 & x_2^{(1)} & x_2^{(2)} & \ldots & x_2^{(d)} \\ \vdots & \vdots & \vdots & & \vdots \\ 1 & x_n^{(1)} & x_n^{(2)} & \ldots & x_n^{(d)} \end{bmatrix} = \begin{bmatrix} \text{Aug}(\vec{x}_1)^T \\ \text{Aug}(\vec{x}_2)^T \\ \vdots \\ \text{Aug}(\vec{x}_n)^T \end{bmatrix}, \qquad \vec{y} = \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_n \end{bmatrix}$$

and to solve the normal equation to find the optimal parameter vector, $\vec{w}^*$:

$$X^T X \vec{w}^* = X^T \vec{y}$$

The $\vec{w}^*$ that satisfies the equations above minimizes mean squared error, $R_{\text{sq}}(\vec{w})$, so use this $\vec{w}^*$ to make predictions in $h(\vec{x}_i) = \vec{w}^* \cdot \text{Aug}(\vec{x}_i)$. Once you find that $\vec{w}^*$,

- $\vec{p} = X\vec{w}^*$ is a prediction vector of predicted y-values for the entire dataset. It is also the projection of $\vec{y}$ onto the span of the columns of X.

- $h(\vec{x}_i) = \vec{w}^* \cdot \text{Aug}(\vec{x}_i)$ is the predicted y-value for just the input $\vec{x}_i$.

Using `sklearn`. Earlier in the course, we saw how to use `sklearn` to fit a simple linear regression model. Performing multiple linear regression is just as easy.

```
from sklearn.linear_model import LinearRegression

model_multiple = LinearRegression()
model_multiple.fit(X=commutes_full[['departure_hour', 'day_of_month']], y=commutes_full['minutes'])
```

```
LinearRegression()
```

In a Jupyter environment, please rerun this cell to show the HTML representation or trust the notebook.
On GitHub, the HTML representation is unable to render, please try loading this page with nbviewer.org.

[x] LinearRegression?Documentation for LinearRegressioniFitted

```
LinearRegression()
```

After calling `fit`, we can access $\vec{w}^*$.

```
model_multiple.intercept_, model_multiple.coef_
```

```
(141.86402699471932, array([ -8.2233821 ,  0.05615985]))
```

The outputs above tell us that the "best way" (according to squared loss) to make commute time predictions using a linear model is using:

$$\text{pred. commute}_i = 141.86 - 8.22 \cdot \text{departure hour}_i + 0.06 \cdot \text{day of month}_i$$

This is the **plane of best fit** given historical data; it is not a causal statement.

Fit `LinearRegression` objects have a `predict` method, which can be used to predict commute times given a `departure_hour` and `day_of_month`.

```
# What if I leave at 9:15AM on the 26th of the month?
# To supress the warning below, we should convert X and y to numpy arrays before calling .fit.
model_multiple.predict([[9.25, 26]])
```

```
/Users/surajrampure/miniforge3/envs/pds/lib/python3.10/site -packages/sklearn/base.py:493: UserWarni

X does not have valid feature names, but LinearRegression was fitted with feature names
```

```
array([67.25789852])
```

```
model_multiple.predict(pd.DataFrame({'departure_hour': [9.25], 'day_of_month': [26]}))
```

```
array([67.25789852])
```

Comparing Models. While it's not difficult to implement ourselves, `sklearn` has a built-in `mean_squared_error` function.

```
from sklearn.metrics import mean_squared_error
mean_squared_error(commutes_full['minutes'], model_multiple.predict(commutes_full[['departure_hour',

96.78730488437492
```

So, the mean squared error of our plane of best fit is 96.78 minutes^2.

Throughout this section, we'll fit various different models to the commute times dataset, and it'll be useful to keep track of their mean squared errors in one place. I'll do so in a Python dictionary. (I'm intentionally showing you more code than I typically have, so you have some sense of how to do this all yourself – I hope you don't mind!)

```
mse_dict = {}

# Multiple linear regression model.
mse_dict['departure_hour + day_of_month'] = mean_squared_error(commutes_full['minutes'], model_multi

# Simple linear regression model.
model_simple = LinearRegression()
model_simple.fit(X=commutes_full[['departure_hour']], y=commutes_full['minutes'])
mse_dict['departure_hour'] = mean_squared_error(commutes_full['minutes'], model_simple.predict(commu

# Constant model.
model_constant = commutes_full['minutes'].mean()
mse_dict['constant'] = mean_squared_error(commutes_full['minutes'], np.ones(commutes_full.shape[0])

mse_dict

{'departure_hour + day_of_month': 96.78730488437492,
 'departure_hour': 97.04687150819183,
 'constant': 167.535147928994}
```

As you can see, adding `day_of_month` **barely** reduced our model's mean squared error, which matches our intuition that knowing whether it's the 15th or 19th or 3rd of the month doesn't seem like it would be helpful in predicting commute time. The activity below has you think about *why* the mean squared error still did decrease, and whether this is always the case.

Activity 1

Activity 1

Recall, **training data** is the data that we use to fit/train/create the model. **Test data** refers to any other data that we might use to evaluate the model's performance.

1. As we add more features, the mean squared error of a model's predictions **on the training data** will **never increase**. Why?
2. Could a model's mean squared error **on test data** increase as we add more features? When would it?

Feature Engineering

So far, we've limited ourselves to using `departure_hour` and `day_of_month`, two features that already came to us in the dataset. In practice, we'll often need to **engineer** features, which means creating new features from existing ones, or "transforming" raw data into good features. We might do this to improve model performance, for example, in case the relationship between the features and output variable isn't truly linear, or maybe even to enable the use of categorical features.

Example: One Hot Encoding. Suppose we'd like to use the `day` column from `commutes_full` as a feature.

```
commutes_full[['departure_hour', 'day', 'day_of_month']].head()
```

	departure_hour	day	day_of_month
0	10.816667	Mon	15
1	7.750000	Tue	16
2	8.450000	Mon	22
3	7.133333	Tue	23
4	9.150000	Tue	30

One naive approach would be to convert each `day` value to a number, e.g. `"Mon"` is 1, `"Tue"` is 2, `"Wed"` is 3, etc. **The reason this is a bad idea** is that this approach implies that Monday is somehow "less than" Tuesday and should impact our model's predictions less than Tuesday (remember that this column will be multiplied by a parameter in order to make predictions). This is what's called an **ordinal encoding**, and these only make sense when the features have some inherent order that is meaningful to the prediction problem.

Instead, we'll perform **one hot encoding**, which turns a categorical feature into several binary features: one per unique value of the categorical feature.

day	day == Mon	day == Tue	day == Wed	day == Thu	day == Fri
Mon	1	0	0	0	0
Tue	0	1	0	0	0
Mon	1	0	0	0	0
...	...	...	...	...	...
Mon	1	0	0	0	0
Tue	0	1	0	0	0
Thu	0	0	0	1	0

Since `day` has 5 unique values, we'll need to create 5 new binary features.

```
# pandas Series have a value_counts method, which describes the frequency of each unique value in th
commutes_full['day'].value_counts()
```

```
day
Tue    25
Mon    20
Thu    15
Wed     3
Fri     2
Name: count, dtype: int64
```

Suppose we want to build a model that predicts `minutes` using `departure_hour`, `day_of_month`, **and** one hot encoded `day`. That model would have $1 + 1 + 5 = 7$ features, so its design matrix would have $7 + 1 = 8$ columns!

	1	departure_hour	day_of_month	day == Mon	day == Tue	day == Wed	day == Thu	day == Fri
0	1	10.816667	15	1	0	0	0	0
1	1	7.750000	16	0	1	0	0	0
2	1	8.450000	22	1	0	0	0	0
3	1	7.133333	23	0	1	0	0	0
4	1	9.150000	30	0	1	0	0	0

The actual process of one hot encoding can be done efficiently using `sklearn.preprocessing.OneHotEncoder`. You'll see how to use this in a homework assignment.

Note that while it looks like the `day == Wed`, `day == Thu`, and `day == Fri` columns are all 0's, this is just because there were no Wednesday, Thursday, or Friday values **in the first five rows** of the dataset. Somewhere later on in the dataset, there were rows with Wednesday, Thursday, or Friday values, and so those columns have non-zero values later on. If there were never any Wednesday, Thursday, or Friday values, then we wouldn't make those columns in the first place (just like we never made a Saturday column, or a Zebra column, etc.).

Question: What is the rank of the design matrix, whose first five rows are shown above?

You might notice that among the 5 binary columns, in a particular row, **exactly** one of them is 1, and the other 4 are 0. This means that if we sum together the 5 binary columns, we'll get a column of 1's – which is already the first column in our design matrix!

This means that, by default, when we perform one hot encoding, the design matrix doesn't have full rank. This doesn't stop us from projecting $\vec{y}$ onto colsp(X), but it does mean that there are infinitely many possible optimal parameters $\vec{w}^*$. Again, all of these would give us the same predictions and same mean squared error, but it's annoying to have to deal with this setting.

```
np.linalg.matrix_rank(X_for_ohe) # Not 8!
```

```
7
```

```
X_for_ohe['day == Mon'] + X_for_ohe['day == Tue'] + X_for_ohe['day == Wed'] + X_for_ohe['day == Thu'
```

```
0     1
1     1
2     1
3     1
4     1
     ..
60    1
```

```
61    1
62    1
63    1
64    1
Length: 65, dtype: int64
```

So, a common solution when performing one hot encoding is to **drop one of the binary columns**. This doesn't change the information conveyed by the other features, and doesn't change colsp(X), which is what matters for the quality of the model's predictions. And intuitively, even if we don't have the value of `day == Fri`, we know it's Friday if `day == Mon`, `day == Tue`, `day == Wed`, and `day == Thu` are all 0.

Now that we've converted `day` to a numerical variable, we can use it as input in a regression model. Here's the model we'll try to fit:

$$\begin{aligned}\text{pred. commute time}_i = w_0 &+ w_1 \cdot \text{departure hour}_i \\ &+ w_2 \cdot \text{day of month}_i \\ &+ w_3 \cdot \text{day}_i == \text{Mon} \\ &+ w_4 \cdot \text{day}_i == \text{Tue} \\ &+ w_5 \cdot \text{day}_i == \text{Wed} \\ &+ w_6 \cdot \text{day}_i == \text{Thu}\end{aligned}$$

This model has 6 features (1 + 1 + 4) and hence 7 parameters. Its design matrix is $n \times 7$.

```
X_for_ohe = for_ohe[['1', 'departure_hour', 'day_of_month', 'day == Mon', 'day == Tue', 'day == Wed'
X_for_ohe.head()
```

	1	departure_hour	day_of_month	day == Mon	day == Tue	day == Wed	day == Thu
0	1	10.816667	15	1	0	0	0
1	1	7.750000	16	0	1	0	0
2	1	8.450000	22	1	0	0	0
3	1	7.133333	23	0	1	0	0
4	1	9.150000	30	0	1	0	0

Notice that below, I've set `fit_intercept=False`, because I manually added a column of 1's to the design matrix, so I don't need `sklearn` to add another one.

```
model_with_ohe = LinearRegression(fit_intercept=False)
model_with_ohe.fit(X=X_for_ohe, y=commutes_full['minutes'])
```

```
LinearRegression(fit_intercept=False)
```

In a Jupyter environment, please rerun this cell to show the HTML representation or trust the notebook. On GitHub, the HTML representation is unable to render, please try loading this page with nbviewer.org.

[x] LinearRegression?Documentation for LinearRegressioniFitted

```
LinearRegression(fit_intercept=False)
```

```
model_with_ohe.coef_

array([ 1.34043066e+02, -8.41714813e+00, -2.52657944e -02,  5.09024617e+00,
        1.63763100e+01,  5.11983892e+00,  1.14972088e+01])
```

So, our linear model to predict commute time given `departure_hour`, `day_of_month`, and `day` (Mon, Tue, Wed, or Thu) is:

$$\begin{aligned}\text{pred. commute time}_i = 134 &- 8.42 \cdot \text{departure hour}_i \\ &- 0.03 \cdot \text{day of month}_i \\ &+ 5.09 \cdot \text{day}_i == \text{Mon} \\ &+ 16.38 \cdot \text{day}_i == \text{Tue} \\ &+ 5.12 \cdot \text{day}_i == \text{Wed} \\ &+ 11.5 \cdot \text{day}_i == \text{Thu}\end{aligned}$$

While this model has 6 features, and thus requires 7 dimensions to graph, it's really a collection of **five parallel planes** in 3D, all with slightly different z-intercepts! Remember that $\text{day}_i == \text{Mon}$, $\text{day}_i == \text{Tue}$, etc. aren't all free variables: if one of them is 1, the others must be 0. There are 5 cases to consider, and each one corresponds to one of these 5 parallel planes. You're exploring this idea in Homework 7, Question 3.

If we do want to visualze the model in $\mathbb{R}^2$, we need to pick a single feature to place on the x-axis.

Despite being a linear model, this model doesn't look like a straight line, since it's linear in terms of **all 6 features**, but when you project it into 2D, it no longer appears linear.

How does this new model compare to our previous models?

```
mse_dict['departure_hour + day_of_month + day'] = mean_squared_error(
    commutes_full['minutes'],
    model_with_ohe.predict(X_for_ohe)
)

mse_dict

{'departure_hour + day_of_month': 96.78730488437492,
 'departure_hour': 97.04687150819183,
 'constant': 167.535147928994,
 'departure_hour + day_of_month + day': 70.21791287461917}
```

Note that adding the day of the week decreased our model's mean squared error **significantly**: this one hot encoded model is our best yet.

Activity 2

Activity 2

Suppose we use the code below to build a multiple linear regression model to predict the width of a fish, given its height and weight.

```
model = LinearRegression()
model.fit(X, y)
```

```
# Used in the answer choices below.
ws = np.append(model.intercept_, model.coef_)
preds = model.predict(X)
squares = X.shape[0] * mean_squared_error(y, preds)
```

Assume that:

- X and `X` refer to the **full rank** $n \times 3$ design matrix used to fit the model.
- $\vec{y}$ and `y` refer to the obdservation vector.
- $\vec{w}^*$ refers to the optimal parameter vector found by `sklearn`.

In **each column** of the grid below, **select all** mathematical expressions that have the same value as the expression provided in the column header. Some rows may end up empty, but every column should have at least one expression picked. Assume that 0 and $\vec{0}$ are the same for the purposes of this question.

	`preds`	`ws`	`squares`	`np.sum(y - preds)`
0				
$\|\vec{y} - X\vec{w}^*\|^2$				
$X^TX\vec{w}^* - X^T\vec{y}$				
$\vec{1}^T(\vec{y} - X\vec{w}^*)$				
$(X^TX)^{-1}X^T\vec{y}$				
$X(X^TX)^{-1}X^T\vec{y}$				

Solution

	`preds`	`ws`	`squares`	`np.sum(y - preds)`
0				×
$\|\vec{y} - X\vec{w}^*\|^2$			×	
$X^TX\vec{w}^* - X^T\vec{y}$				×
$\vec{1}^T(\vec{y} - X\vec{w}^*)$				×
$(X^TX)^{-1}X^T\vec{y}$		×		
$X(X^TX)^{-1}X^T\vec{y}$	×			

Let's look at each **column** one by one.

- `preds`, in math, is a vector of predicted values, $\vec{p} = X\vec{w}^*$. Since X's columns are linearly independent, $\vec{w}^*$ is the unique vector $\vec{w}^* = (X^TX)^{-1}X^T\vec{y}$, and so $\vec{p} = X(X^TX)^{-1}X^T\vec{y}$.
- `ws` is $(X^TX)^{-1}X^T\vec{y}$, as discussed above.
- `squares` is the mean squared error of the predictions, multiplied by n (i.e. it's the sum of squared errors). Mean squared error is $\frac{1}{n}\|\vec{y} - X\vec{w}^*\|^2$, so `squares` is $\|\vec{y} - X\vec{w}^*\|^2$.
- `np.sum(y - preds)` is the sum of the components in $\vec{y} - X\vec{w}^*$, which we know is $\mathbf{0}$. This is because:
  - X contains a column of 1's, and
  - the error vector $\vec{e} = \vec{y} - X\vec{w}^*$ must be orthogonal to each of X's columns, so
  - $\vec{e} \cdot \begin{bmatrix} 1 \\ \vdots \\ 1 \end{bmatrix} = 0$, so
  - $e_1 + e_2 + \ldots + e_n = 0$. This was introduced in Chapter 6.3 and revisited in Chapter 7.1.

  So, any other expression on the left that is equal to 0 (or $\vec{0}$) should be selected.
  - 0, the first row, should be seleccted.
  - $X^TX\vec{w}^* - X^T\vec{y}$ is equal to $\vec{0}$, since $\vec{w}^*$ satisfies the normal equations, $X^TX\vec{w}^* = X^T\vec{y}$.
  - $\vec{1}^T(\vec{y} - X\vec{w}^*)$ is equal to 0, using the logic above.

Example: Numerical Transformations. Another common step in feature engineering is to produce new numerical features out of existing ones.

	departure_hour	day	day_of_month
0	10.816667	Mon	15
1	7.750000	Tue	16
2	8.450000	Mon	22
3	7.133333	Tue	23
4	9.150000	Tue	30

As an example, I could fit a degree 2 polynomial model

$$h(x_i) = w_0 + w_1 x_i + w_2 x_i^2$$

by producing the design matrix

$$X = \begin{bmatrix} 1 & x_1 & x_1^2 \\ 1 & x_2 & x_2^2 \\ \vdots & \vdots & \vdots \\ 1 & x_n & x_n^2 \end{bmatrix}$$

Below, we'll apply this thinking to the `departure_hour` feature. We'll create these features manually, though in homework assignments you'll see how to do this easily and more efficiently using `sklearn.preprocessing.PolynomialF`

```
X_for_quadratic = commutes_full[['departure_hour']].copy()
X_for_quadratic['departure_hour^2'] = X_for_quadratic['departure_hour'] ** 2

model_quadratic = LinearRegression()
model_quadratic.fit(X=X_for_quadratic, y=commutes_full['minutes'])
model_quadratic.intercept_, model_quadratic.coef_
```

```
(351.324480895268, array([ -57.14500476,   2.8270457 ]))
```

This tells us that the best-fitting quadratic model is

$$h(x_i) = 351.32 - 57.14 \cdot \text{departure hour}_i + 2.83 \cdot (\text{departure hour}_i)^2.$$

which looks like:

This is a linear model, even though the features themselves are made up of non-linear functions.

If we treat x_i and x_i^2 as two separate features, then each data point lives in $\mathbb{R}^3$ with coordinates

$$(x_i, x_i^2, y_i)$$

In that feature space, the model is

$$\text{predicted commute}_i = w_0 + w_1 \cdot \underbrace{\text{departure hour}_i}_{\text{feature 1}} + w_2 \cdot \underbrace{(\text{departure hour}_i)^2}_{\text{feature 2}}$$

so the predictions lie on a plane. It only looks curved when we project back down to the usual 2D plot of `departure_hour` versus `minutes`, because the points we draw must satisfy the relationship $x_i^2 = (x_i)^2$.

We could follow the same idea to fit a degree 3 polynomial model, or a degree 4 polynomial model, or a degree 5 polynomial model, and so on. Just to try this one more time, let's fit a degree 3 model.

```
X_for_cubic = commutes_full[['departure_hour']].copy()
X_for_cubic['departure_hour^2'] = X_for_cubic['departure_hour'] ** 2
X_for_cubic['departure_hour^3'] = X_for_cubic['departure_hour'] ** 3

model_cubic = LinearRegression()
```

```
model_cubic.fit(X=X_for_cubic, y=commutes_full['minutes'])
model_cubic.intercept_, model_cubic.coef_
```

```
(816.0936290109973, array([ -227.63718864,   23.33213694,   -0.80864398]))
```

Overfitting. Is this model **better** than the simple linear regression model or quadratic model from earlier?

75.27, the mean squared error of the cubic model, is lower than the simple linear regression model's mean squared error of 97.04 (but worse than we got when one hot encoding `day`) and lower than the quadratic model's mean squared error of 75.32.

Keep in mind that adding complex features **doesn't always** equate to a "better model", even if it lowers the mean squared error on the training data. To illustrate what I mean, what if we fit a degree 10 polynomial to the data?

Without computing it, we can tell this model's mean squared error is lower than that of the simple linear, quadratic, or cubic models, since it passes closer to the training data. But, this model is **overfit** to the training data. Remember, our eventual goal is to build models that **generalize well to new data**, and it seems unlikely that this degree 10 polynomial reflects the true relationship between departure hour and commute time in nature.

The question, then, is how to choose the right model, e.g. how to choose the right polynomial degree, if we're committed to making polynomial features. The solution is to intentionally split our data into **training data** and **test data**, and use only the training data to pick the features we'd like to include (e.g. polynomial degree). Then, we pick the combination of features that performed best on the test data, since that combination of features seems most likely to generalize well on unseen data. You'll explore this paradigm in homework assignments.

In the above examples, I created polynomial features out of `departure_hour` only, but there's nothing stopping me from creating features out of `day_of_month` too, or out of both of them simultaneously. A perfectly valid model is

$$\text{pred. commute time}_i = w_0 + w_1 \cdot \text{departure hour}_i + w_2 \cdot (\text{departure hour}_i \cdot \text{dom}_i) + w_3 \cdot \log(\text{dom}_i)$$

which, using our more generic syntax, would look like

$$h(\vec{x}_i) = w_0 + w_1 x_i^{(1)} + w_2 \left(x_i^{(1)} x_i^{(2)} \right) + w_3 \log(x_i^{(2)})$$

You may wonder, **how do I know which features to create**? Through a variety of techniques:

- Domain knowledge
- Trial and error
- By visualizing the data

In the polynomial examples above, the data didn't particularly look like it was quadratic or cubic, so a linear model sufficed. But if we're given a dataset that clearly has some non-linear relationship, visualization *can* help us identify what features might improve model performance.

As the dimensionality of our data increases, though, visualization becomes less possible. (How do we visualize a dataset with 100 features?) Strategies for reducing the dimensionality of our data will be explored in Chapter 10.

Linear in the Parameters. What is the extent to which I can use linear regression to fit a model? As long as a model can be expressed in the form

$$h(\vec{x}_i) = \vec{w} \cdot \text{Aug}(\vec{x}_i)$$

for **some** parameter vector $\vec{w}$ and **some** feature vector $\vec{x}_i$, then it can be fit using linear regression (i.e. by creating a design matrix X and finding $\vec{w}^*$ by solving the normal equations). We say such a model is **linear in the parameters**. The choice of features to include in $\vec{x}_i$ is up to us.

For example,

- If x_i is a scalar, then

$$h(x_i) = w_0 + w_1 x_i + w_2 \cos(x_i) + w_3 e^{x_i} = \vec{w} \cdot \begin{bmatrix} 1 \\ x_i \\ \cos(x_i) \\ e^{x_i} \end{bmatrix}$$

  is linear in the parameters. It would be linear in the parameters even if the w_0 term wasn't there; then, we wouldn't need to augment $\vec{x}_i$ with a 1.

- If $x_i^{(1)}$ and $x_i^{(2)}$ are scalars (e.g. height and weight), then

$$h(\vec{x}_i) = w_0 + w_1 x_i^{(1)} + w_2 x_i^{(2)} + w_3 x_i^{(1)} x_i^{(2)} + w_4 \frac{e^{x_i^{(1)}}}{x_i^{(2)}} = \vec{w} \cdot \begin{bmatrix} 1 \\ x_i^{(1)} \\ x_i^{(2)} \\ x_i^{(1)} x_i^{(2)} \\ \frac{e^{x_i^{(1)}}}{x_i^{(2)}} \end{bmatrix}$$

  is linear in the parameters.

- If x_i is a scalar, then

$$h(x_i) = w_0 + w_1 \cos(x_i) + \sin(w_2 x_i)$$

  is not linear in the parameters, because w_2 is within the $\sin$ function. This model can't be written as $\vec{w} \cdot \text{Aug}(\vec{x}_i)$ for any choice of $\vec{x}_i$.

Gradients

8.1. The Gradient Vector

The big theme in our course so far has been the three-step modeling recipe, of choosing a model, choosing a loss function, and then minimizing empirical risk (i.e. average loss) to find optimal model parameters.

In Chapter 2.3, we used calculus to find the slope w_1^* and intercept w_0^* that minimized mean squared error,

$$R_{\text{sq}}(w_0, w_1) = \frac{1}{n}\sum_{i=1}^{n}(y_i - (w_0 + w_1 x_i))^2$$

by computing $\frac{\partial R_{\text{sq}}}{\partial w_0}$ (the partial derivative with respect to w_0) $\frac{\partial R_{\text{sq}}}{\partial w_1}$, setting both to zero, and solving the resulting system of equations.

Then, in Chapters 7.1 and 7.2, we focused on the multiple linear regression model and the squared loss function, which saw us minimize

$$R_{\text{sq}}(\vec{w}) = \frac{1}{n}\|\vec{y} - X\vec{w}\|^2 = \frac{1}{n}\sum_{i=1}^{n}(y_i - \vec{w} \cdot \text{Aug}(\vec{x}_i))^2$$

where X is the $n \times (d+1)$ design matrix, $\vec{y} \in \mathbb{R}^n$ is the observation vector, and $\vec{w} \in \mathbb{R}^{d+1}$ is the parameter vector we're trying to pick. In Chapter 6.3, we minimized $R_{\text{sq}}(\vec{w})$ by arguing that the optimal $\vec{w}^*$ had to create an error vector, $\vec{e} = \vec{y} - X\vec{w}^*$, that was orthogonal to colsp(X), which led us to the normal equations.

It turns out that there's a way to use our calculus-based approach from Chapter 2.3 to minimize the more general version of R_{sq} for any d, that **doesn't** involve computing d partial derivatives. To see how this works, we need to define a new object, the **gradient vector**, which we'll do here in Chapter 8.1. After we're familiar with how the gradient vector works, we'll use it to build a new approach to function minimization, one that works even when there isn't a closed-form solution for the optimal parameters: that technique is called **gradient descent**, which we'll see in Chapter 8.3.

Domain and Codomain

As we saw in Chapter 6.2 when we first introduced the concept of the inverse of a matrix, the notation

$$f : \mathbb{R}^d \to \mathbb{R}^n$$

means that f is a function whose inputs are vectors with d components and whose outputs are vectors with n components. $\mathbb{R}^d$ is the **domain** of the function, and $\mathbb{R}^n$ is the **codomain**. I've used d and n to match the notation we've used for matrices and linear transformations. In general, if A is an $n \times d$ matrix, then any vector $\vec{x}$ multiplied by A (on the right) must be in $\mathbb{R}^d$ and the result $A\vec{x}$ will be in $\mathbb{R}^n$.

Given this framing, consider the following four types of functions.

Type	Domain and Codomain	Examples
Scalar-to-scalar	$f : \mathbb{R} \to \mathbb{R}$	$f(x) = x^2 + \sin(x)$ $R_{\text{sq}}(w) = \frac{1}{n}\sum_{i=1}^{n}(y_i - w)^2$
Vector-to-scalar	$f : \mathbb{R}^d \to \mathbb{R}$	$f(\vec{x}) = \vec{x}^T\vec{x}$ $f(\vec{x}) = (x_1 - 1)^2 + (x_1 - x_2)^2 + 3$ $R_{\text{sq}}(\vec{w}) = \frac{1}{n}\lVert\vec{y} - X\vec{w}\rVert^2$
Scalar-to-vector	$f : \mathbb{R} \to \mathbb{R}^n$	$f(x) = \underbrace{\begin{bmatrix}1\\2\\1\end{bmatrix} + x\begin{bmatrix}3\\0\\-4\end{bmatrix}}_{\text{parametric form of a line}}$ $f(x) = \begin{bmatrix}x^2 + e^x\\-3\end{bmatrix}$
Vector-to-vector	$f : \mathbb{R}^d \to \mathbb{R}^n$	$f(\vec{x}) = \underbrace{\begin{bmatrix}3&4\\2&0\\0&1\end{bmatrix}\vec{x}}_{\text{linear transformation}}$ $f(\vec{x}) = \begin{bmatrix}x_1^2 + x_1x_2 + \cos(x_1^4)\\3x_1x_2 + 4\\5x_1\\x_2/x_3\end{bmatrix}$

The first two types of functions are "scalar-valued", while the latter two are "vector-valued". These are not the only types of functions that exist; for instance, the function $f(A) = \text{rank}(A)$ is a matrix-to-scalar function.

The type of function we're most concerned with at the moment are **vector-to-scalar** functions, i.e. functions that take in a vector (or equivalently, multiple scalar inputs) and output a single scalar.

$$R_{\text{sq}}(\vec{w}) = \frac{1}{n}\lVert\vec{y} - X\vec{w}\rVert^2$$

is one such function, and it's the focus of this section.

Rates of Change

Let's think from the perspectives of **rates of change**, since ultimately what we're building towards is a technique for minimizing functions. We're most familiar with the concept of rates of change for scalar-to-scalar functions.

If

$$f(x) = x^2\sin(x)$$

then its **derivative**,

$$\frac{\text{d}f}{\text{d}x} = 2x\sin(x) + x^2\cos(x)$$

itself is a scalar-to-scalar function, which describes how quickly f is changing at any point x in the domain of f. At $x = 3$, for instance, the instantaneous rate of change is

$$\frac{df}{dx}(3) = 2 \cdot 3\sin(3) + 3^2\cos(3) \approx -8.06$$

meaning that at $x = 3$, f is decreasing at a rate of (approximately) 8.06 per unit change in x. Perhaps a more intuitive way of thinking about the instantaneous rate of change is to think of it as the **slope of the tangent line** to f at $x = 3$.

The steeper the slope, the faster f is changing at that point; the sign of the slope tells us whether f is increasing or decreasing at that point.

In Chapter 2.3, we saw how to compute derivatives of functions that take in multiple scalar inputs, like

$$f(x, y, z) = x^2 + 2xy + 3xz + 4(y - z)^2$$

In the language of Chapter 8.1, we'd call such a function a **vector-to-scalar** function, and might use the notation

$$f(\vec{x}) = x_1^2 + 2x_1x_2 + 3x_1x_3 + 4(x_2 - x_3)^2$$

This function has three **partial derivatives**, each of which describes the instantaneous rate of change of f with respect to one of its inputs, while holding the other two inputs constant. There's a good animation of what it means to hold an input constant in Chapter 2.2 that is worth revisiting.

Here,

$$\frac{\partial f}{\partial x_1} = 2x_1 + 2x_2 + 3x_3, \quad \frac{\partial f}{\partial x_2} = 2x_1 + 8x_2 - 8x_3, \quad \frac{\partial f}{\partial x_3} = 3x_1 - 8x_2 + 8x_3$$

The big idea of this section, the **gradient vector**, packages all of these partial derivatives into a single vector. This will allow us to think about the **direction** in which f is changing, rather than just looking at its rates of change in each dimension independently.

The Gradient Vector

Definition: Gradient Vector

Suppose $f : \mathbb{R}^d \to \mathbb{R}$ is a vector-to-scalar function. The **gradient vector** of f, denoted $\nabla f(\vec{x})$, is the vector in $\mathbb{R}^d$ of partial derivatives of f:

$$\nabla f(\vec{x}) = \begin{bmatrix} \frac{\partial f}{\partial x_1} \\ \frac{\partial f}{\partial x_2} \\ \vdots \\ \frac{\partial f}{\partial x_d} \end{bmatrix}$$

$\nabla f(\vec{x})$ itself is a vector-to-vector function; it takes in a vector $\vec{x} \in \mathbb{R}^d$ and outputs a new vector in $\mathbb{R}^d$, describing the rates of change of f along each dimension. The gradient, when evaluated at a point $\vec{x}_0$ describes the **direction of steepest ascent** of f at $\vec{x}_0$, i.e. the direction in which f is increasing most quickly.

Let's start with a straightforward example where the partial derivatives are easy to compute. Let

$$f(\vec{x}) = x_1^2 + x_2^2 - 3x_1x_2$$

Then

$$\frac{\partial f}{\partial x_1} = 2x_1 - 3x_2, \quad \frac{\partial f}{\partial x_2} = 2x_2 - 3x_1$$

so

$$\nabla f(\vec{x}) = \begin{bmatrix} 2x_1 - 3x_2 \\ 2x_2 - 3x_1 \end{bmatrix}$$

If we evaluate the gradient at $\vec{x} = \begin{bmatrix} 2 \\ 1 \end{bmatrix}$, we get

$$\nabla f\left(\begin{bmatrix} 2 \\ 1 \end{bmatrix}\right) = \begin{bmatrix} 2(2) - 3(1) \\ 2(1) - 3(2) \end{bmatrix} = \begin{bmatrix} 1 \\ -4 \end{bmatrix}$$

What does the fact that $\nabla f\left(\begin{bmatrix} 2 \\ 1 \end{bmatrix}\right) = \begin{bmatrix} 1 \\ -4 \end{bmatrix}$ tell us? It tells us that the direction of steepest ascent of f at $\begin{bmatrix} 2 \\ 1 \end{bmatrix}$ is $\begin{bmatrix} 1 \\ -4 \end{bmatrix}$. To put this into context, let's consider another example.

Visualizing the Gradient Vector

Let's look at another example and use it to understand what the gradient of a function tells us visually. Suppose $\vec{x} \in \mathbb{R}^2$, and let

$$f(\vec{x}) = x_1 e^{-x_1^2 - x_2^2}$$

To find $\nabla f(\vec{x})$, we need to compute the partial derivatives of f with respect to each component of $\vec{x}$. The "input variables" to f are x_1 and x_2, so we need to compute $\frac{\partial f}{\partial x_1}$ and $\frac{\partial f}{\partial x_2}$, but if you'd like, replace x_1 and x_2 with x and y if it makes the algebra a little cleaner, and then replace x and y with x_1 and x_2 at the end.

$$f(\vec{x}) = x_1 e^{-x_1^2 - x_2^2}$$

$$\frac{\partial f}{\partial x_1} = \frac{\partial}{\partial x_1}\left(x_1 e^{-x_1^2 - x_2^2}\right) = \underbrace{1 \cdot e^{-x_1^2 - x_2^2} + x_1 \cdot e^{-x_1^2 - x_2^2} \cdot (-2x_1)}_{\text{product rule}} = (1 - 2x_1^2)e^{-x_1^2 - x_2^2}$$

$$\frac{\partial f}{\partial x_2} = \frac{\partial}{\partial x_2}\left(x_1 e^{-x_1^2 - x_2^2}\right) = \underbrace{x_1 \cdot e^{-x_1^2 - x_2^2} \cdot (-2x_2)}_{\text{chain rule}} = -2x_1x_2 e^{-x_1^2 - x_2^2}$$

Putting these together, we have

$$\nabla f(\vec{x}) = \begin{bmatrix} (1-2x_1^2)e^{-x_1^2-x_2^2} \\ -2x_1x_2e^{-x_1^2-x_2^2} \end{bmatrix}$$

Remember, $\nabla f(\vec{x})$ itself is a function. If we plug in a value of $\vec{x}$, we get a new vector back.

$$\nabla f\left(\begin{bmatrix} -1 \\ 0 \end{bmatrix}\right) = \begin{bmatrix} (1-2(-1)^2)e^{-(-1)^2-0^2} \\ -2(-1)(0)e^{-(-1)^2-0^2} \end{bmatrix} = \begin{bmatrix} -1/e \\ 0 \end{bmatrix}$$

What does $\nabla f\left(\begin{bmatrix} -1 \\ 0 \end{bmatrix}\right) = \begin{bmatrix} -1/e \\ 0 \end{bmatrix}$ really tell us? In order to visualize it, let me introduce another way of visualizing f, called a **contour plot**.

I think of the contour plot as a bird's-eye view of f when you look at the surface from above. Notice the correspondence between the colors in both graphs.

The circle-like traces in the contour plot are called **level curves**; they represent slices through the surface at a constant height. On the right, the circle labeled 0.1 represents the set of points where $f(x_1, x_2) = 0.1$.

Visualizing the fact that $\nabla f\left(\begin{bmatrix} -1 \\ 0 \end{bmatrix}\right) = \begin{bmatrix} -1/e \\ 0 \end{bmatrix}$ is easier to do in the contour plot, since the contour plot is 2-dimensional, like the gradient vector is. Remember that red values are high and blue values are low.

At the point $\begin{bmatrix} -1 \\ 0 \end{bmatrix}$, which is at the tail of the vector drawn in gold, f is near the global minimum, meaning there are lots of directions in which we can move to increase f. But, the gradient vector at this point is $\begin{bmatrix} -1/e \\ 0 \end{bmatrix}$, which points in the **direction of steepest ascent** starting at $\begin{bmatrix} -1 \\ 0 \end{bmatrix}$. The gradient describes the "quickest way up".

As another example, consider the fact that $\nabla f\left(\begin{bmatrix} 1.25 \\ -0.5 \end{bmatrix}\right) \approx \begin{bmatrix} -0.347 \\ 0.204 \end{bmatrix}$.

Again, the gradient at $\begin{bmatrix} 1.25 \\ -0.5 \end{bmatrix}$ gives us the direction in which f is increasing the quickest **at that very point**. If we move even a little bit in any direction (in the direction of the gradient or some other direction), the gradient will change.

One way to see this more globally is to draw many gradient vectors at once, forming a **gradient field**.

Each arrow in this gradient field shows the gradient vector at a different point, and the arrow lengths are proportional to the magnitude of the gradient there. Longer arrows indicate places where f increases more steeply, while shorter arrows indicate places where the rate of increase is smaller. Note that the arrows don't necessarily all point to the "top" of the function, located at $\begin{bmatrix} \frac{\sqrt{2}}{2} \\ 0 \end{bmatrix} \approx \begin{bmatrix} 0.707 \\ 0 \end{bmatrix}$ – instead, they point in the direction of steepest ascent at each point.

At the two critical points $\begin{bmatrix} -\frac{\sqrt{2}}{2} \\ 0 \end{bmatrix}$ and $\begin{bmatrix} \frac{\sqrt{2}}{2} \\ 0 \end{bmatrix}$, the gradient really is $\begin{bmatrix} 0 \\ 0 \end{bmatrix}$, so those locations are shown as points instead of arrows.

As you might guess, to find the **critical points** of a function - that is, places where it is neither increasing nor decreasing - we need to find points where the gradient is zero. Hold that thought.

In our course, most of the functions we'll work with won't be defined in terms of the individual components of the input vector $\vec{x}$, like in the case of $f(\vec{x}) = x_1^2 + x_2^2 - 3x_1x_2$. Instead, they'll be defined in terms of **matrix-vector operations**, like $f(\vec{x}) = \vec{x}^T A\vec{x}$. Chapter 8.2 explores how to compute gradients of functions like this.

8.2. Gradients of Matrix-Vector Operations

More typically, the functions we'll need to take the gradient of will themselves be defined in terms of matrix and vector operations. In all of these examples, remember that we're working with vector-to-scalar functions.

The Big Three Rules

Example: Dot Product. Let $\vec{a} \in \mathbb{R}^d$ be some fixed vector (the equivalent of a constant in this context). Let's find the gradient of

$$f(\vec{x}) = \vec{a} \cdot \vec{x}$$

I find it helpful to think about $f(\vec{x})$ in its expanded form,

$$f(\vec{x}) = \vec{a} \cdot \vec{x} = a_1 x_1 + a_2 x_2 + \cdots + a_d x_d$$

Remember, $\nabla f(\vec{x})$ contains all of the partial derivatives of f, which we now need to compute.

- What is $\frac{\partial f}{\partial x_1}$? To me, that looks like a_1, since the first term is $a_1 x_1$ and none of the other terms involve x_1.
- Similarly, $\frac{\partial f}{\partial x_2} = a_2$.
- In general, $\frac{\partial f}{\partial x_i} = a_i$.

Putting these together, we get

$$\nabla f(\vec{x}) = \begin{bmatrix} \frac{\partial f}{\partial x_1} \\ \frac{\partial f}{\partial x_2} \\ \vdots \\ \frac{\partial f}{\partial x_d} \end{bmatrix} = \begin{bmatrix} a_1 \\ a_2 \\ \vdots \\ a_d \end{bmatrix} = \vec{a}$$

To visualize this, let $\vec{a} = \begin{bmatrix} 2 \\ -3 \end{bmatrix}$. Then

$$f(\vec{x}) = \vec{a}^T \vec{x} = 2x_1 - 3x_2$$

so the graph of f is the plane $z = 2x_1 - 3x_2$ in $\mathbb{R}^3$. The two arrows below show the gradient at two different points on the plane. They point in the same direction because $\nabla f(\vec{x}) = \vec{a}$ everywhere, so the direction of steepest ascent doesn't change.

Example: Norm and Chain Rule. Here's an extremely important example that shows up everywhere in machine learning. Find the gradients of:

1. $f(\vec{x}) = \|\vec{x}\|^2$
2. $f(\vec{x}) = \|\vec{x}\|$

Solution

1. As we did in the previous example, we can expand $f(\vec{x}) = \|\vec{x}\|^2$ to get

$$f(\vec{x}) = \vec{x} \cdot \vec{x} = x_1^2 + x_2^2 + \cdots + x_d^2$$

For each i, $\frac{\partial f}{\partial x_i} = 2x_i$. So,

$$\nabla f(\vec{x}) = \begin{bmatrix} 2x_1 \\ 2x_2 \\ \vdots \\ 2x_d \end{bmatrix} = \boxed{2\vec{x}}$$

Think of this as the equivalent of the "power rule" for vectors.

2. There are two ways to find the gradient of $f(\vec{x}) = \|\vec{x}\|$: directly, or by using the chain rule. It's not immediately obvious how the chain rule should work here, so we'll start with the direct method and reason about how the chain rule may arise.

Direct method: Let's start by expanding $f(\vec{x}) = \|\vec{x}\|$ like we did above.

$$f(\vec{x}) = \sqrt{\vec{x} \cdot \vec{x}} = (\vec{x} \cdot \vec{x})^{1/2} = (x_1^2 + x_2^2 + \cdots + x_d^2)^{1/2}$$

For each i, the (regular, scalar-to-scalar) chain rule tells us that

$$\frac{\partial f}{\partial x_i} = \frac{1}{2}(x_1^2 + x_2^2 + \cdots + x_d^2)^{-1/2} \cdot 2x_i = \frac{x_i}{\sqrt{x_1^2 + x_2^2 + \cdots + x_n^2}} = \frac{x_i}{\|\vec{x}\|}$$

So,

$$\nabla f(\vec{x}) = \begin{bmatrix} \frac{x_1}{\|\vec{x}\|} \\ \frac{x_2}{\|\vec{x}\|} \\ \vdots \\ \frac{x_d}{\|\vec{x}\|} \end{bmatrix} = \boxed{\frac{\vec{x}}{\|\vec{x}\|}}$$

Chain rule method: Let me start by writing $f(\vec{x})$ in terms of a composition of two functions.

$$f(\vec{x}) = \|\vec{x}\| = \sqrt{\|\vec{x}\|^2} = h(g(\vec{x}))$$

where $g(\vec{x}) = \|\vec{x}\|^2$ and $h(x) = \sqrt{x}$. Note that $g : \mathbb{R}^d \to \mathbb{R}$ is the vector-to-scalar function we found the gradient of above, and $h : \mathbb{R} \to \mathbb{R}$ is a scalar-to-scalar function.

Then, generalizing the calculation we did with the first method, we have a "chain rule" for a function $h(g(\vec{x}))$ (where h is scalar-to-scalar and g is vector-to-scalar):

$$\nabla f(\vec{x}) = \underbrace{\left(\frac{\mathrm{d}h}{\mathrm{d}x}(g(\vec{x}))\right)}_{h'(g(\vec{x}))} \nabla g(\vec{x})$$

Remember that $h(x) = \sqrt{x}$, so $\frac{\mathrm{d}h}{\mathrm{d}x}(x) = \frac{1}{2\sqrt{x}}$ and $\frac{\mathrm{d}h}{\mathrm{d}x}(g(\vec{x})) = \frac{1}{2\sqrt{g(\vec{x})}} = \frac{1}{2\|\vec{x}\|}$. This means

$$\nabla f(\vec{x}) = \left(\frac{\mathrm{d}h}{\mathrm{d}x}(g(\vec{x}))\right) \nabla g(\vec{x}) = \left(\frac{1}{2\|\vec{x}\|}\right) 2\vec{x} = \frac{\vec{x}}{\|\vec{x}\|}$$

Chain Rule for Vector-to-Scalar Functions. To recap from the previous example, suppose $f(\vec{x}) = h(g(\vec{x}))$, where $h : \mathbb{R} \to \mathbb{R}$ is a scalar-to-scalar function (meaning h has a derivative) and $g : \mathbb{R}^d \to \mathbb{R}$ is a vector-to-scalar function (meaning g has a gradient). Then,

$$\nabla f(\vec{x}) = \left(\frac{\mathrm{d}h}{\mathrm{d}x}(g(\vec{x}))\right) \nabla g(\vec{x})$$

Example: Norm to an Exponent. Find the gradient of $f(\vec{x}) = \|\vec{x}\|^p$, where p is some real number.

Solution

We can treat this as a composition of two functions, $g(\vec{x}) = \|\vec{x}\|$ and $h(x) = x^p$, and use the chain rule introduced in the solution to the previous example.
$\frac{\mathrm{d}h}{\mathrm{d}x}(x) = px^{p-1}$ and $\nabla g(\vec{x}) = \frac{\vec{x}}{\|\vec{x}\|}$. Putting these together yields

$$\begin{aligned}
\nabla f(\vec{x}) &= \left(\frac{\mathrm{d}h}{\mathrm{d}x}(g(\vec{x}))\right) \nabla g(\vec{x}) \\
&= p\, g(\vec{x})^{p-1} \frac{\vec{x}}{\|\vec{x}\|} \\
&= p\, \|\vec{x}\|^{p-1} \frac{\vec{x}}{\|\vec{x}\|} \\
&= p\, \|\vec{x}\|^{p-2} \vec{x}
\end{aligned}$$

Example: Log Sum Exp. If $\vec{x} \in \mathbb{R}^d$, we can define the **log sum exp** function as

$$f(\vec{x}) = \log\left(\sum_{i=1}^{d} e^{x_i}\right)$$

What is $\nabla f(\vec{x})$? (The answer is called the **softmax function**, and comes up all the time in machine learning, when we want our models to output predicted **probabilities** in a classification problem.)

Solution

Let's look at the partial derivatives with respect to each x_i.

$$\frac{\partial f}{\partial x_i} = \frac{\partial}{\partial x_i}\left(\log\left(\sum_{j=1}^{d} e^{x_j}\right)\right) = \left(\frac{1}{\sum_{j=1}^{d} e^{x_j}}\right)\frac{\partial}{\partial x_i}\left(\sum_{j=1}^{d} e^{x_j}\right) = \frac{e^{x_i}}{\sum_{j=1}^{d} e^{x_j}}$$

Then,

$$\nabla f(\vec{x}) = \begin{bmatrix} \frac{e^{x_1}}{\sum_{j=1}^{d} e^{x_j}} \\ \frac{e^{x_2}}{\sum_{j=1}^{d} e^{x_j}} \\ \vdots \\ \frac{e^{x_n}}{\sum_{j=1}^{d} e^{x_j}} \end{bmatrix} = \frac{1}{\sum_{j=1}^{d} e^{x_j}} \begin{bmatrix} e^{x_1} \\ e^{x_2} \\ \vdots \\ e^{x_d} \end{bmatrix}$$

There isn't really a way to simplify the expression using matrix-vector operations, so I'll leave it as-is. As mentioned above, the gradient we're looking at is called the **softmax function**. The softmax function maps $\mathbb{R}^d \to \mathbb{R}^d$, meaning it's a vector-to-vector function.

Let's suppose we have the matrix $\begin{bmatrix} 3 \\ 5 \\ -1 \end{bmatrix}$. What does passing it through the softmax function yield?

$$\text{softmax}\left(\begin{bmatrix} 3 \\ 5 \\ -1 \end{bmatrix}\right) = \frac{1}{e^3 + e^5 + e^{-1}} \begin{bmatrix} e^3 \\ e^5 \\ e^{-1} \end{bmatrix} \approx \begin{bmatrix} 0.119 \\ 0.879 \\ 0.0002 \end{bmatrix}$$

The output vector has the same number of elements as the input vector, but each element is between 0 and 1, and the sum of elements is 1, meaning that we can interpret the outputted vector as a probability distribution. Larger values in the output correspond to larger values in the input, and almost all of the "mass" is concentrated at the maximum element of the input vector (position 2), hence the name "soft" max. (The "hard" max might be $\begin{bmatrix} 0 \\ 1 \\ 0 \end{bmatrix}$ in this case.)

Example: Quadratic Forms. Suppose $x \in \mathbb{R}^n$ and A is an $n \times n$ matrix. The function

$$f(\vec{x}) = \vec{x}^T A \vec{x}$$

is called a **quadratic form**, and its gradient is given by

$$\nabla f(\vec{x}) = (A + A^T)\vec{x}$$

We won't directly cover the proof of this formula here; one place to find it is here. Instead, we'll focus our energy on understanding how it works, since it's **extremely important**.

1. Let $A = \begin{bmatrix} a & b \\ c & d \end{bmatrix}$. Expand out $f(\vec{x}) = \vec{x}^T A \vec{x}$ and compute $\nabla f(\vec{x})$ directly by computing partial derivatives, and verify that the result you get matches the formula above.
2. In quadratic forms, we typically assume that A is symmetric, meaning that $A = A^T$. Why do you think this assumption is made (what does it help with)?

- Hint: Let $A = \begin{bmatrix} 3 & 2 \\ 6 & 1 \end{bmatrix}$ and $B = \begin{bmatrix} 3 & 4 \\ 4 & 1 \end{bmatrix}$. Compute $\nabla(\vec{x}^T A\vec{x})$ and $\nabla(\vec{x}^T B\vec{x})$.

3. If A is any symmetric $n \times n$ matrix, what is $\nabla f(\vec{x})$?
4. Suppose A is symmetric and $n \times n$, $\vec{b} \in \mathbb{R}^n$, and $c \in \mathbb{R}$. Find the gradient of

$$f(\vec{x}) = \vec{x}^T A\vec{x} + \vec{b} \cdot \vec{x} + c$$

Solution

1. If $A = \begin{bmatrix} a & b \\ c & d \end{bmatrix}$, then

$$\begin{aligned} f(\vec{x}) &= \vec{x}^T A \vec{x} \\ &= \begin{bmatrix} x_1 & x_2 \end{bmatrix} \begin{bmatrix} a & b \\ c & d \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} \\ &= \begin{bmatrix} x_1 & x_2 \end{bmatrix} \begin{bmatrix} ax_1 + bx_2 \\ cx_1 + dx_2 \end{bmatrix} \\ &= ax_1^2 + (b+c)x_1x_2 + dx_2^2 \end{aligned}$$

   Then,

$$\frac{\partial f}{\partial x_1} = 2ax_1 + (b+c)x_2, \quad \frac{\partial f}{\partial x_2} = (b+c)x_1 + 2dx_2$$

$$\nabla f(\vec{x}) = \begin{bmatrix} 2ax_1 + (b+c)x_2 \\ (b+c)x_1 + 2dx_2 \end{bmatrix} = \begin{bmatrix} 2a & b+c \\ b+c & 2d \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = (A + A^T)\vec{x}$$

   since $A^T = \begin{bmatrix} a & c \\ b & d \end{bmatrix}$, meaning $A + A^T = \begin{bmatrix} 2a & b+c \\ b+c & 2d \end{bmatrix}$.

2. For a particular quadratic form, there are infinitely many choices of matrices A that represent it. To illustrate, let's look at $A = \begin{bmatrix} 3 & 2 \\ 6 & 1 \end{bmatrix}$ and $B = \begin{bmatrix} 3 & 4 \\ 4 & 1 \end{bmatrix}$ as provided in the hint.

$$\vec{x}^T A \vec{x} = \begin{bmatrix} x_1 & x_2 \end{bmatrix} \begin{bmatrix} 3 & 2 \\ 6 & 1 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = 3x_1^2 + (2+6)x_1x_2 + x_2^2$$

$$\vec{x}^T B \vec{x} = \begin{bmatrix} x_1 & x_2 \end{bmatrix} \begin{bmatrix} 3 & 4 \\ 4 & 1 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = 3x_1^2 + (4+4)x_1x_2 + x_2^2$$

   Note that both $\vec{x}^T A \vec{x}$ and $\vec{x}^T B \vec{x}$ are equal to the expression $3x_1^2 + 8x_1x_2 + x_2^2$. In fact, any matrix of the form $\begin{bmatrix} 3 & b \\ c & 1 \end{bmatrix}$ where $b + c = 8$ would produce the same quadratic form.

   So, to avoid this issue of having infinitely many choices of the matrix A, we pick the **symmetric** matrix A, where $A = A^T$. As we're about to see, this choice of A simplifies the calculation of the gradient.

3. If A is any symmetric $n \times n$ matrix, then $A = A^T$, and $A + A^T = 2A$. So,

$$\nabla(\vec{x}^T A \vec{x}) = (A + A^T)\vec{x} = 2A\vec{x}$$

   This is also an important rule; don't forget it.

4. Think of $f(\vec{x}) = \vec{x}^T A \vec{x} + \vec{b} \cdot \vec{x} + c$ as the matrix-vector equivalent of a quadratic function, $ax^2 + bx + c$. The derivative of $ax^2 + bx + c$ is $2ax + b$. Check out what the gradient of $f(\vec{x})$ ends up being!

$$\begin{aligned} f(\vec{x}) &= \vec{x}^T A \vec{x} + \vec{b} \cdot \vec{x} + c \\ \nabla f(\vec{x}) &= (A + A^T)\vec{x} + \vec{b} \\ &= 2A\vec{x} + \vec{b} \qquad \text{(since } A \text{ is symmetric)} \end{aligned}$$

Summary of the Big Three Rules. These are the main three rules you **need** to know moving forward, not just because we're about to use them in an important proof, but because they'll come up repeatedly in your future machine learning work.

Function	Name	Gradient
$f(\vec{x}) = \vec{a} \cdot \vec{x}$	dot product	$\nabla f(\vec{x}) = \vec{a}$
$f(\vec{x}) = \|\vec{x}\|^2$	squared norm	$\nabla f(\vec{x}) = 2\vec{x}$
$f(\vec{x}) = \vec{x}^T A\vec{x}$	quadratic form	$\nabla f(\vec{x}) = (A + A^T)\vec{x}$ if A is symmetric, $\nabla f(\vec{x}) = 2A\vec{x}$

Optimization

In the calculus of scalar-to-scalar functions, we have a well-understood procedure for finding the extrema of a function. The general strategy is to take the derivative, set it to zero, and solve for the inputs (called **critical points**) that satisfy that condition. To be thorough, we'd perform a second derivative test to check whether each critical point is a maximum, minimum, or neither.

In the land of vector-to-scalar functions, the equivalent is to solve for where the **gradient** is zero, which corresponds to finding where all partial derivatives are zero. Assessing whether we've arrived at a maximum or minimum is more difficult to do in the vector-to-scalar case, and we will save a discussion of this for Chapter 8.5.

As an example, consider

$$f(\vec{x}) = \vec{x}^T \begin{bmatrix} 3 & 4 \\ 4 & 1 \end{bmatrix} \vec{x} + \begin{bmatrix} 1 \\ 2 \end{bmatrix} \cdot \vec{x} + 3$$

As we computed earlier, the gradient of $f(\vec{x}) = \vec{x}^T A\vec{x} + \vec{b} \cdot \vec{x} + c$ is $\nabla f(\vec{x}) = 2A\vec{x} + \vec{b}$ for symmetric A. So,

$$\nabla f(\vec{x}) = 2 \begin{bmatrix} 3 & 4 \\ 4 & 1 \end{bmatrix} \vec{x} + \begin{bmatrix} 1 \\ 2 \end{bmatrix} = \begin{bmatrix} 6x_1 + 8x_2 + 1 \\ 8x_1 + 2x_2 + 2 \end{bmatrix}$$

To find the critical points, we set the gradient to zero and solve the resulting system. We can also accomplish this by using the inverse of A, if we happen to have it:

$$\nabla f(\vec{x}) = 0 \implies 2A\vec{x} + \vec{b} = 0 \implies \vec{x}^* = -\frac{1}{2} A^{-1}\vec{b}$$

Either way, we find that $\vec{x}^* = \begin{bmatrix} -7/26 \\ 1/13 \end{bmatrix}$ satisfies $\nabla f(\vec{x}^*) = 0$, which corresponds to a local minimum.

Minimizing Mean Squared Error

Remember, the goal of this section is to minimize mean squared error,

$$R_{\text{sq}}(\vec{w}) = \frac{1}{n} \|\vec{y} - X\vec{w}\|^2$$

In the general case, X is an $n \times (d+1)$ matrix, $y \in \mathbb{R}^n$, and $\vec{w} \in \mathbb{R}^{d+1}$.

We're now equipped with the tools to minimize $R_{\text{sq}}(\vec{w})$ by taking its gradient and setting it to zero. Hopefully, we end up with the same conditions on $\vec{w}^*$ that we derived in Chapter 6.3.

In the most recent example we saw, the optimal vector $\vec{x}^*$ corresponded to a **local minimum**. We know that we won't run into such an issue here since $R_{\text{sq}}(\vec{w})$ cannot output a negative number (it is the average of squared losses), so its minimum possible output is 0, meaning that there will be *some* global minimizer $\vec{w}^*$.

Let's start by rewriting the squared norm as a dot product and eventually matrix multiplication.

$$\begin{aligned} R_{\text{sq}}(\vec{w}) = \frac{1}{n}\|\vec{y} - X\vec{w}\|^2 &= \frac{1}{n}(\vec{y} - X\vec{w}) \cdot (\vec{y} - X\vec{w}) \\ &= \underbrace{\frac{1}{n}(\vec{y} - X\vec{w})^T(\vec{y} - X\vec{w})}_{\text{since } \vec{u}\cdot\vec{v} = \vec{u}^T\vec{v}} \\ &= \frac{1}{n}\left(\vec{y}^T - (X\vec{w})^T\right)(\vec{y} - X\vec{w}) \\ &= \frac{1}{n}\left(\vec{y}^T\vec{y} - \vec{y}^T X\vec{w} - (X\vec{w})^T\vec{y} + (X\vec{w})^T X\vec{w}\right) \end{aligned}$$

Let's focus on the two terms in orange. They are both equal: they are both the dot product of $\vec{y}$ and $X\vec{w}$. Ideally, I want to express each term as a dot product of $\vec{w}$ with something, since I'm taking the gradient with respect to $\vec{w}$. Remember, the dot product is a **scalar**, and the transpose of a scalar is just that same scalar. So,

$$\vec{y}^T X\vec{w} = (\vec{y}^T X\vec{w})^T = \vec{w}^T X^T\vec{y} = \vec{w}^T(X^T\vec{y})$$

so, performing this substitution in for both orange terms gives us

$$\begin{aligned} R_{\text{sq}}(\vec{w}) &= \frac{1}{n}\left(\vec{y}^T\vec{y} - \vec{w}^T(X^T\vec{y}) - \vec{w}^T X^T\vec{y} + \vec{w}^T(X^T X)\vec{w}\right) \\ &= \frac{1}{n}\left(\vec{y}^T\vec{y} - 2\vec{w}^T(X^T\vec{y}) + \vec{w}^T(X^T X)\vec{w}\right) \end{aligned}$$

Now, we're ready to take the gradient, which we'll do term by term.

- $\nabla\left(\vec{y}^T\vec{y}\right) = \vec{0}$, since $\vec{y}^T\vec{y}$ is a constant with respect to $\vec{w}$
- $\nabla\left(2\vec{w}^T(X^T\vec{y})\right) = 2X^T\vec{y}$ using the dot product rule, since this is the dot product between $2X^T\vec{y}$ (a vector) and $\vec{w}$ (a vector)
- $\nabla\left(\vec{w}^T(X^T X)\vec{w}\right) = 2X^T X\vec{w}$, using the quadratic form rule, since $X^T X$ is a symmetric matrix

Plugging these terms in gives us

$$\begin{aligned} R_{\text{sq}}(\vec{w}) &= \frac{1}{n}\left(\vec{y}^T\vec{y} - 2\vec{w}^T(X^T\vec{y}) + \vec{w}^T(X^T X)\vec{w}\right) \\ \nabla R_{\text{sq}}(\vec{w}) &= \frac{1}{n}\left(\nabla\left(\vec{y}^T\vec{y}\right) - \nabla\left(2\vec{w}^T(X^T\vec{y})\right) + \nabla\left(\vec{w}^T(X^T X)\vec{w}\right)\right) \\ &= \frac{1}{n}\left(0 - 2X^T\vec{y} + 2X^T X\vec{w}\right) \\ &= \boxed{\frac{2}{n}(X^T X\vec{w} - X^T\vec{y})} \end{aligned}$$

Finally, to find the minimizer $\vec{w}^*$, we set the gradient to zero and solve.

$$\frac{2}{n}(X^T X\vec{w}^* - X^T\vec{y}) = 0 \implies X^T X\vec{w}^* = X^T\vec{y}$$

Stop me if this feels familiar... these are the **normal equations** once again! It shouldn't be a surprise that we ended up with the same conditions on $\vec{w}^*$ that we derived in Chapter 6.3, since we were solving the same problem.

We've now shown that the minimizer of

$$R_{\text{sq}}(\vec{w}) = \frac{1}{n}\|\vec{y} - X\vec{w}\|^2$$

is given by solving $X^T X\vec{w}^* = X^T\vec{y}$. These equations have a unique solution if $X^T X$ is invertible, and infinitely many solutions otherwise. If $\vec{w}^*$ satisfies the normal equations, then $X\vec{w}^*$ is the vector in colsp(X) that is closest to $\vec{y}$. All of that interpretation from Chapter 6.3 and Chapter 7 carry over; we've just introduced a new way of finding the solution.

Heads up: In Homework 10, you'll follow similar steps to minimize a new **objective** function, that resembles $R_{\text{sq}}(\vec{w})$ but involves another term. There, you'll minimize

$$R_{\text{ridge}}(\vec{w}) = \|\vec{y} - X\vec{w}\|^2 + \lambda\|\vec{w}\|^2$$

where $\lambda > 0$ is a constant, called the **regularization hyperparameter**. (Notice the missing $\frac{1}{n}$.) A good way to practice what you've learned (and to get a head start on the homework) is to compute the gradient of $R_{\text{ridge}}(\vec{w})$ and set it to zero. We'll walk through what the significance of $R_{\text{ridge}}(\vec{w})$ is in the homework.

8.3. Gradient Descent

In Chapter 8.1, we learned about the **gradient**, the equivalent of the derivative for **vector-to-scalar** functions. The big takeaway was that $\nabla f(\vec{x})$ describes the direction in which f is increasing the quickest, at the point $\vec{x}$.

At the end of Chapter 8.1, we **minimized**

$$R_{\text{sq}}(\vec{w}) = \frac{1}{n}\|\vec{y} - X\vec{w}\|^2$$

by computing its gradient, setting it to 0, and solving for $\vec{w}^*$, which gave us another way of deriving the normal equations.

So far, most of the empirical risk functions that we've minimized had **closed-form solutions** – that is, formulas for the optimal parameters $w_0^*, w_1^*, \ldots$ that we could find by hand using algebra.

$$R_{\text{sq}}(w) = \frac{1}{n}\sum_{i=1}^{n}(y_i - w)^2 \quad \Longrightarrow \quad w^* = \bar{y}$$

$$R_{\text{sq}}(w_0, w_1) = \frac{1}{n}\sum_{i=1}^{n}(y_i - (w_0 + w_1 x_i))^2 \quad \Longrightarrow \quad w_1^* = r\frac{\sigma_y}{\sigma_x}, \quad w_0^* = \bar{y} - w_1^*\bar{x}$$

$$R_{\text{sq}}(\vec{w}) = \frac{1}{n}\|\vec{y} - X\vec{w}\|^2 \quad \Longrightarrow \quad \vec{w}^* = (X^TX)^{-1}X^T\vec{y}$$

But, soon we'll encounter combinations of models and loss functions whose empirical risk functions that have *some* global minimum, but that global minimum isn't described by a formula that we can write by hand. There was an example of such an empirical risk function from a previous homework; see if you can remember what it was!

As another example, the **logistic regression** model, used for predicting the probability of a binary event (e.g. whether or not a patient has a disease) makes predictions using

$$h(\vec{x}_i) = P(y_i = 1 \mid \vec{x}_i) = \frac{1}{1 + e^{-\vec{w}\cdot\text{Aug}(\vec{x}_i)}}$$

Logistic regression typically uses the **cross-entropy** loss function,

$$L_{\text{ce}}(y_i, h(\vec{x}_i)) = -\left(y_i \log h(\vec{x}_i) + (1 - y_i)\log(1 - h(\vec{x}_i))\right)$$

resulting in the empirical risk function

$$R_{\text{ce}}(\vec{w}) = -\frac{1}{n}\sum_{i=1}^{n}\left(y_i \log\left(\frac{1}{1 + e^{-\vec{w}\cdot\text{Aug}(\vec{x}_i)}}\right) + (1 - y_i)\log\left(1 - \frac{1}{1 + e^{-\vec{w}\cdot\text{Aug}(\vec{x}_i)}}\right)\right)$$

(What a mess!)

$R_{\text{ce}}(\vec{w})$ is a vector-to-scalar function, and it has a gradient, $\nabla R_{\text{ce}}(\vec{w})$. The issue is that the solutions to $\nabla R_{\text{ce}}(\vec{w}) = \vec{0}$ can't be found by hand. But, $\nabla R_{\text{ce}}(\vec{w})$ still means **something** – and we can use it to estimate $\vec{w}^*$ without solving for it explicitly.

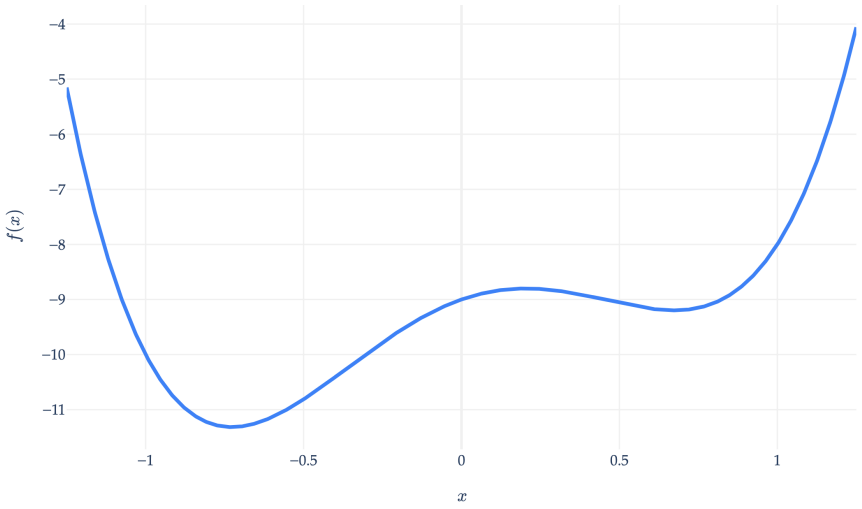
Throughout this section, keep the following thought exercise in mind:

> Suppose you're at the top of a mountain and **need to get to the bottom**. And, perhaps it's really cloudy, meaning you can only see a few feet around you. **How** would you get to the bottom?

What do Derivatives Tell Us?

Let's start with a simpler, scalar-to-scalar example. Suppose we'd like to **minimize** the function

$$f(x) = 5x^4 - x^3 - 5x^2 + 2x - 9$$



There are some not-so-elegant techniques for minimizing f. For instance, we *could* evaluate f at dozens (or hundreds) of possible x's, and pick the one that had the smallest output. But, that's an inefficient way to go about things, especially as the number of input variables (i.e. the d in $\vec{x} \in \mathbb{R}^d$) increases.

Instead, we'll use the fact that f is **differentiable**. Its derivative is

$$\frac{\mathrm{d}f}{\mathrm{d}x}(x) = 20x^3 - 3x^2 - 10x + 2$$

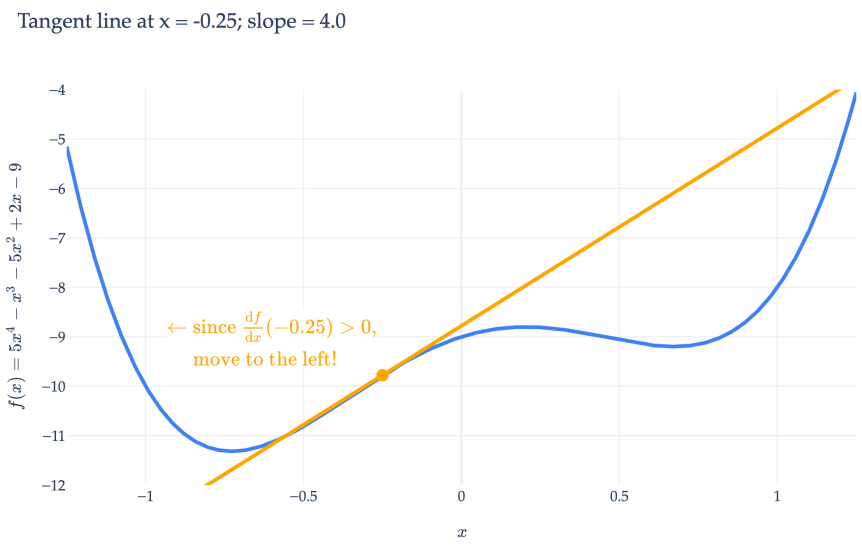
Typically, to minimize f, we'd

1. Find $\frac{\mathrm{d}f}{\mathrm{d}x}(x)$.
2. Solve for the input x^* such that $\frac{\mathrm{d}f}{\mathrm{d}x}(x^*) = 0$.

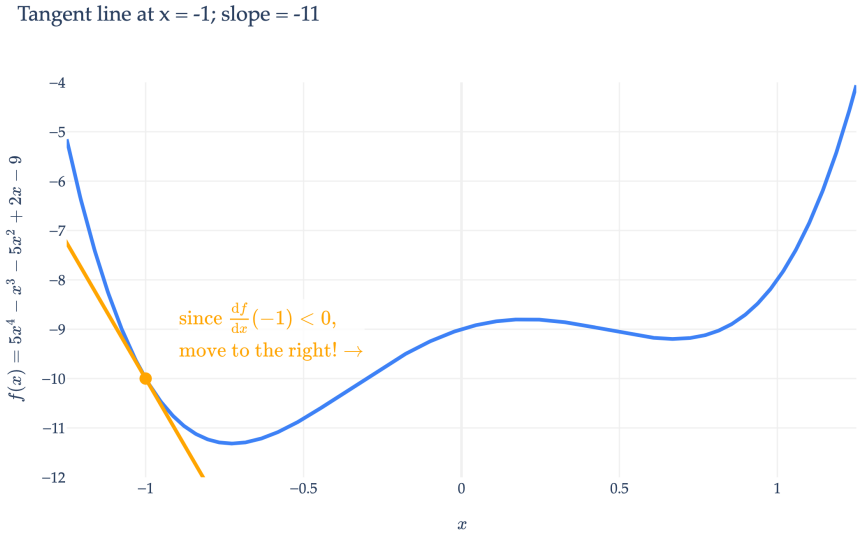
But, $\frac{\mathrm{d}f}{\mathrm{d}x}(x) = 0$ is a cubic equation, which is difficult to solve by hand (there is actually a "cubic formula", the same way there's a "quadratic formula", but higher-degree polynomials don't have similar formulas). What can we do with its derivative?

The key idea is that we'll take an iterative approach. Suppose we start with an initial guess for x^*, say, $x^{(0)}$.

Case 1: If the derivative at $x^{(0)}$ is positive , then f is increasing at $x^{(0)}$, which means to decrease f, we should move to the **left** of $x^{(0)}$.



Case 2: If the derivative at $x^{(0)}$ is negative , then f is decreasing at $x^{(0)}$, which means to decrease f, we should move to the **right** of $x^{(0)}$.



Remember that at a minimum (or maximum), the derivative is 0 (if it exists), and gradually approaches 0, at least for the types of functions we'll consider. So, if the derivative at our current guess is large, we must be far away from a minimum, and should take a larger step than if the derivative is small (which must mean we're close to the minimum already).

This intuition is the basic idea behind **gradient descent**. The fact that "gradient" is in the name implies that it's typically used for minimizing vector-to-scalar functions $f : \mathbb{R}^d \to \mathbb{R}$, which we will use it for shortly. I just figured a scalar-to-scalar example would help build intuition.

Gradient Descent

Definition: Gradient Descent

Suppose $f : \mathbb{R}^d \to \mathbb{R}$ is a **differentiable** vector-to-scalar function, meaning that all of its partial derivatives are defined everywhere.
To find $\vec{x}^*$, the minimizer of f:

1. Choose a positive number, α. This number is called the **learning rate**, or **step size**.
2. Choose an **initial guess** for the minimizer, $\vec{x}^{(0)}$.
3. Then, repeatedly update the guess using the **update rule**:

$$\vec{x}^{(t+1)} = \vec{x}^{(t)} - \alpha \nabla f(\vec{x}^{(t)})$$

4. Terminate once the algorithm converges, which happens when the norm of the gradient, $\|\nabla f(\vec{x}^{(t)})\|$, is below some small **tolerance** level, e.g. 0.001 (since this must mean we're very close to a minimum).

Gradient descent is a **numerical method** for finding the input to a function f that **minimizes** the function. Ultimately, that's what this is about: minimizing functions.

A numerical method is a technique for approximating the solution to a mathematical problem, often by using the computer. Since it only involves first (partial) derivatives, it's called a **first-order method**; numerical methods that use second (partial) derivatives are called **second-order methods**.

Gradient descent is *the* workhorse of machine learning. As I stated earlier, it's used to minimize empirical risk, $R(\vec{w})$, in the general case, where it can't be minimized by hand. (In the first few examples we'll see, we'll use gradient descent to minimize arbitrary functions of the form $f(\vec{x})$, but in Chapter 8.4 we'll return to minimizing empirical risk functions specifically.) This is especially true for state-of-the-art models, like neural networks and transformers, which have billions of parameters. So, when you hear about companies spending billions of dollars on training models, they're spending money to run gradient descent on their oceans of data and models with billions of parameters.

Example Implementation. To get a feel for how it works, let's implement gradient descent ourselves on the scalar-to-scalar function we looked at earlier,

$$f(x) = 5x^4 - x^3 - 5x^2 + 2x - 9, \qquad \frac{\mathrm{d}f}{\mathrm{d}x}(x) = 20x^3 - 3x^2 - 10x + 2$$

For scalar-to-scalar functions, we can replace the gradient with the derivative, and the update rule becomes

$$x^{(t+1)} = x^{(t)} - \alpha \frac{\mathrm{d}f}{\mathrm{d}x}(x^{(t)})$$

Let's start with an initial guess of $x^{(0)} = 0$ and a learning rate of $\alpha = 0.01$.

```
def df(x):
    return 20 * (x**3) - 3 * (x**2) - 10 * x + 2

def minimize_f(x0, alpha, tol=0.0001):
    x = x0 # Initial guess
    t = 0 # Iteration counter (just for tracking)
```

```
    while np.abs(df(x)) > tol:
        # The core logic is in this one line
        x = x - alpha * df(x)

        # Everything below is for us to track the progress of the algorithm
        t += 1
        if t % 10 == 0:
            print(f't = {t}: x = {x}, df/dx = {df(x)}')
    print(f"Converged in {t} iterations")

minimize_f(x0=0, alpha=0.01)
```

```
t = 10: x = -0.3019961066782706, df/dx = 4.195505266352445
t = 20: x = -0.6539833552473451, df/dx = 1.6626527284020556
t = 30: x = -0.7227462183452047, df/dx = 0.10967130332536357
t = 40: x = -0.7267760078170167, df/dx = 0.005439315124045052
t = 50: x = -0.7269745284952817, df/dx = 0.0002654471448240159
Converged in 54 iterations
```

With our initial guess of $x^{(0)} = 0$ and a learning rate of $\alpha = 0.01$, the algorithm converges to $x^* \approx -0.727$ in 54 iterations.

Animations. Let's visualize the execution of the algorithm on this $f(x)$, with several different choices of initial guess and learning rate. In each animation, click the " Start animation" button to see the algorithm in action. (If the algorithm stops somewhere that isn't a minimum, it's because I set them to only show 50 iterations.)

Initial guess = 0, step size = 0.01

The algorithm converges to the true global minimum, but it takes a while.

What if we choose a different initial guess?

Initial guess = 1.1, step size = 0.01

Uh oh: the algorithm gets trapped in a local minimum that isn't the global minimum! From its perspective, local and global minima are the same, in that they have derivatives of 0. **Gradient descent doesn't seem well-suited for functions with multiple local minima.**

What if we try a different step size?

Initial guess = 1.1, step size = 0.1

At first, seemingly by luck, the algorithm jumps over to the neighborhood of the global minimum. But our step size appears to be too large, causing the algorithm to keep jumping back and forth across the global minimum.

The choice of step size is critical. In future courses, you may encounter some theoretical results that give you insight on how to choose the step size, but in practice, we often just try different step sizes and see what works. Another technique is to choose a **decaying** learning rate, in which the value of α decreases over time.

Vector-to-Scalar Functions

Let's look at another more complex example,

$$f(\vec{x}) = 3\sin(2x_1)\cos(2x_2) + x_1^2 + x_2^2$$

Note that I've chosen a function $f : \mathbb{R}^2 \to \mathbb{R}$, since the resulting function exists in three dimensions, which is the largest space we can visualize. If f took in vectors with 3, or 4, or more components, we couldn't visualize how gradient descent operations on it.

That said, we'll visualize $f(\vec{x})$ in two ways: as a surface and as a contour plot. (Remember, you should think of the latter as a "birds-eye view" of the former.)

This is the type of function that gradient descent is often used on in practice: functions with multiple (remember, billions! of) input variables, often with local minima.

We can see that $f(\vec{x})$ has a global minimum around $\vec{x}^* \approx \begin{bmatrix} -0.6 \\ 0 \end{bmatrix}$. But, **computers don't have eyes**, and instead need to rely on gradient descent. Recall, gradient descent updates are given by

$$\vec{x}^{(t+1)} = \vec{x}^{(t)} - \alpha \nabla f(\vec{x}^{(t)})$$

$f(\vec{x})$'s gradient is

$$\nabla f(\vec{x}) = \begin{bmatrix} 6\cos(2x_1)\cos(2x_2) + 2x_1 \\ -6\sin(2x_1)\sin(2x_2) + 2x_2 \end{bmatrix}$$

Just to make sure we're 100% on the same page, even though we have a formula for f's gradient, the reason we're using gradient descent is that it is impossible to solve for the vector $\vec{x}^*$ where $\nabla f(\vec{x}^*) = \vec{0}$ by hand: the algebra doesn't work.

So, gradient descent updates are given by

$$\vec{x}^{(t+1)} = \vec{x}^{(t)} - \alpha \begin{bmatrix} 6\cos(2x_1)\cos(2x_2) + 2x_1 \\ -6\sin(2x_1)\sin(2x_2) + 2x_2 \end{bmatrix}$$

The Gradient Describes the Direction of Steepest Ascent

Remember that the gradient vector at a point describes the direction of the steepest **ascent**. So, if we want to **minimize** the function, we should move in the direction opposite to the gradient, hence the negative sign!

The negative of the gradient vector is the direction we want to move in! The amount we move in this direction is determined by both $\|\nabla f(\vec{x})\|$ and α.

Let's visualize the path of gradient descent on this function, again at several different initial guesses and learning rates.

Initial guess = $\begin{bmatrix} -1.5 \\ -1 \end{bmatrix}$**, step size = 0.1**

$\vec{x}^{(t)}$ converged to a local minimum, of which there are many in this function. Also, notice that this step size ($\alpha = 0.1$) worked here even though a step size that large caused the earlier polynomial example to diverge. There is no universal best step size.

Initial guess = $\begin{bmatrix} -1.5 \\ -1 \end{bmatrix}$**, step size = 0.25**

We can visualize the same path on the surface itself.

We can even step through one of these runs interactively: **drag the slider** to move forward one iteration at a time and watch the path unfold.

Initial guess = $\begin{bmatrix} 1 \\ -0.5 \end{bmatrix}$**, step size = 0.05**

Initial guess = $\begin{bmatrix} 1 \\ -0.5 \end{bmatrix}$**, step size = 0.25**

Remember not to rely **too** heavily on visual intuition, since practical examples will take us into higher dimensions, where we can't visualize. But I think both the surface and contour plots are helpful.

Activity 1

Activity 1

Consider the following function.

$$f(\vec{x}) = (x_1 - 2)^2 + 2x_1 - (x_2 - 3)^2$$

1. Is $f(\vec{x})$ a quadratic form?
2. Given an initial guess of $\vec{x}^{(0)} = \begin{bmatrix} 0 \\ 0 \end{bmatrix}$ and a step size of $\alpha = \frac{1}{3}$, perform **two** iterations of gradient descent. What is $\vec{x}^{(2)}$?

It seems that the ability for gradient descent to converge on the global minimum depends on lots of factors:

- The existence of "traps" – that is, local minima that aren't the global minimum.
- The step size, α.
- The initial guess.

If there are no local minima, then gradient descent will converge to the global minimum, given a sufficiently small step size. The type of function that has no local minima is called **convex**. Intuitively, a convex function has a "bowl-like" shape, as you have seen in calculus. In Chapter 8.5, we'll study the idea of convexity in more detail.

8.4. Gradient Descent for Empirical Risk Minimization

While gradient descent can be used to (attempt to) minimize any differentiable function $f(\vec{x})$, we typically use it to minimize empirical risk functions, $R(\vec{w})$. As I said in Chapter 8.3, gradient descent is **the tool** in practice for finding optimal model parameters. This is because most empirical risk functions in practice don't have closed-form solutions, i.e. a formula for $\vec{w}^*$ that we can derive algebraically by hand.

Simple Linear Regression

Let's try using gradient descent to fit a linear regression model - that is, let's use it to minimize

$$R_{\text{sq}}(\vec{w}) = \frac{1}{n}\|\vec{y} - X\vec{w}\|^2$$

This function has a closed-form minimizer, $\vec{w}^* = (X^TX)^{-1}X^T\vec{y}$, so we don't *need* gradient descent. Still, it's worthwhile to see how gradient descent works on it.

In Chapter 8.2, we found that the gradient of $R_{\text{sq}}(\vec{w})$ is

$$\nabla R_{\text{sq}}(\vec{w}) = \frac{2}{n}(X^TX\vec{w} - X^T\vec{y})$$

so, the update rule is

$$\vec{w}^{(t+1)} = \vec{w}^{(t)} - \alpha\frac{2}{n}(X^TX\vec{w}^{(t)} - X^T\vec{y})$$

Let's start by using gradient descent to fit a simple linear regression model to predict commute times in `minutes` from `departure_hour` – a problem we've solved many times.

First, for reference, we'll compute $\vec{w}^*$ using the closed-form solution,

$$\vec{w}^* = (X^TX)^{-1}X^T\vec{y}$$

```
x = df["departure_hour"].to_numpy(dtype=float)
y = df["minutes"].to_numpy(dtype=float)
n = len(df)

X = np.column_stack([np.ones(n), x])

w_star_closed_form = np.linalg.solve(X.T @ X, X.T @ y)
w_star_closed_form
```

```
array([142.44824159,  -8.18694172])
```

The code below implements gradient descent. At each iteration, it computes the MSE, its gradient, and logs the current $\vec{w}^{(t)}$ vector. That current $\vec{w}^{(t)}$ vector is sometimes called an "iterate". For every 500 iterations, it displays the current values of MSE, the norm of the gradient vector, and $\vec{w}^{(t)}$. I've collapsed the code since it's relatively lengthy.

```
t =     0 | w^(0) = [   0.000,    0.000] | mse = 5523.5231 | ||grad|| = 1229.842640
t =   500 | w^(500) = [  19.804,    6.102] | mse =  314.8511 | ||grad|| =  3.527944
t =  1000 | w^(1000) = [  36.133,    4.200] | mse =  260.7135 | ||grad|| =  3.058219
t =  1500 | w^(1500) = [  50.289,    2.551] | mse =  220.0323 | ||grad|| =  2.651035
t =  2000 | w^(2000) = [  62.559,    1.121] | mse =  189.4630 | ||grad|| =  2.298065
t =  2500 | w^(2500) = [  73.196,   -0.118] | mse =  166.4919 | ||grad|| =  1.992091
t =  3000 | w^(3000) = [  82.416,   -1.193] | mse =  149.2306 | ||grad|| =  1.726856
t =  3500 | w^(3500) = [  90.409,   -2.124] | mse =  136.2598 | ||grad|| =  1.496935
t =  4000 | w^(4000) = [  97.338,   -2.931] | mse =  126.5130 | ||grad|| =  1.297627
t =  4500 | w^(4500) = [ 103.344,   -3.631] | mse =  119.1889 | ||grad|| =  1.124856
t =  5000 | w^(5000) = [ 108.551,   -4.237] | mse =  113.6852 | ||grad|| =  0.975088
t =  5500 | w^(5500) = [ 113.064,   -4.763] | mse =  109.5496 | ||grad|| =  0.845261
t =  6000 | w^(6000) = [ 116.976,   -5.219] | mse =  106.4419 | ||grad|| =  0.732719
t =  6500 | w^(6500) = [ 120.368,   -5.614] | mse =  104.1067 | ||grad|| =  0.635162
t =  7000 | w^(7000) = [ 123.308,   -5.957] | mse =  102.3519 | ||grad|| =  0.550594
t =  7500 | w^(7500) = [ 125.856,   -6.254] | mse =  101.0333 | ||grad|| =  0.477285
t =  8000 | w^(8000) = [ 128.065,   -6.511] | mse =  100.0424 | ||grad|| =  0.413738
t =  8500 | w^(8500) = [ 129.980,   -6.734] | mse =   99.2978 | ||grad|| =  0.358651
t =  9000 | w^(9000) = [ 131.640,   -6.928] | mse =   98.7383 | ||grad|| =  0.310899
t =  9500 | w^(9500) = [ 133.079,   -7.095] | mse =   98.3179 | ||grad|| =  0.269504
t = 10000 | w^(10000) = [ 134.327,   -7.241] | mse =   98.0020 | ||grad|| =  0.233621
t = 10500 | w^(10500) = [ 135.408,   -7.367] | mse =   97.7646 | ||grad|| =  0.202516
t = 11000 | w^(11000) = [ 136.345,   -7.476] | mse =   97.5862 | ||grad|| =  0.175552
t = 11500 | w^(11500) = [ 137.158,   -7.571] | mse =   97.4521 | ||grad|| =  0.152179
t = 12000 | w^(12000) = [ 137.862,   -7.653] | mse =   97.3514 | ||grad|| =  0.131917
t = 12500 | w^(12500) = [ 138.473,   -7.724] | mse =   97.2757 | ||grad|| =  0.114353
t = 13000 | w^(13000) = [ 139.002,   -7.785] | mse =   97.2188 | ||grad|| =  0.099127
t = 13500 | w^(13500) = [ 139.461,   -7.839] | mse =   97.1761 | ||grad|| =  0.085929
t = 14000 | w^(14000) = [ 139.859,   -7.885] | mse =   97.1440 | ||grad|| =  0.074488
t = 14500 | w^(14500) = [ 140.204,   -7.925] | mse =   97.1198 | ||grad|| =  0.064571
t = 15000 | w^(15000) = [ 140.502,   -7.960] | mse =   97.1017 | ||grad|| =  0.055973
t = 15500 | w^(15500) = [ 140.761,   -7.990] | mse =   97.0881 | ||grad|| =  0.048521
t = 16000 | w^(16000) = [ 140.986,   -8.017] | mse =   97.0778 | ||grad|| =  0.042061
t = 16500 | w^(16500) = [ 141.181,   -8.039] | mse =   97.0701 | ||grad|| =  0.036460
t = 17000 | w^(17000) = [ 141.350,   -8.059] | mse =   97.0644 | ||grad|| =  0.031606
t = 17500 | w^(17500) = [ 141.496,   -8.076] | mse =   97.0600 | ||grad|| =  0.027398
t = 18000 | w^(18000) = [ 141.623,   -8.091] | mse =   97.0567 | ||grad|| =  0.023750
t = 18500 | w^(18500) = [ 141.733,   -8.104] | mse =   97.0543 | ||grad|| =  0.020588
t = 19000 | w^(19000) = [ 141.828,   -8.115] | mse =   97.0524 | ||grad|| =  0.017847
t = 19500 | w^(19500) = [ 141.910,   -8.124] | mse =   97.0511 | ||grad|| =  0.015470
t = 20000 | w^(20000) = [ 141.982,   -8.133] | mse =   97.0500 | ||grad|| =  0.013411
t = 20500 | w^(20500) = [ 142.044,   -8.140] | mse =   97.0492 | ||grad|| =  0.011625
t = 21000 | w^(21000) = [ 142.098,   -8.146] | mse =   97.0486 | ||grad|| =  0.010077
t = 21027 | w^(21027) = [ 142.101,   -8.146] | mse =   97.0486 | ||grad|| =  0.010000
```

As you can see in the first line, we started with an initial guess of $\vec{w}^{(0)} = \begin{bmatrix} 0 & 0 \end{bmatrix}$. We also chose a step size of $\alpha = 0.01$ arbitrarily.

By the final iteration (number 21027), gradient descent has essentially matched the closed-form least-squares solution. After ~10000 iterations, the norm of the gradient vector is already close to 0, and the MSE stops changing very much.

Instead of just looking at printed logs, here's an interactive figure. Drag the slider from left to right: the left panel tracks the MSE over time, while the right panel shows the regression line corresponding to the selected iteration. The title reports the MSE of the model at that iteration number. This is the sort of figure that machine learning practitioners draw frequently when training large-scale models.

To be clear, the function we are actually minimizing doesn't appear in either of the plots above. That function, $R_{\text{sq}}(\vec{w}) = \frac{1}{n}\|\vec{y} - X\vec{w}\|^2$, is a vector-to-scalar function that we'd need to draw in $\mathbb{R}^3$ here.

Gradient descent is beautiful: once we can write down a model, a loss for a single example, and the gradient of the resulting average loss, we can search for optimal parameters for far more complicated model-loss combinations. Least squares is just one example. As long as the average loss is differentiable, gradient descent gives us a general recipe for finding the model's optimal parameters.

Another Example: Logistic Regression

Coming soon!

Issues of Scale

When I get a chance, I'll flesh this section out more. For now, refer to this blog post by Sebastian Ruder. It discusses **stochastic gradient descent**, a variant of gradient descent used in practical machine learning applications.

8.5. Convexity

Attention

This chapter is currently under development and has more bugs and typos than usual.

As we saw in Chapter 8.3, gradient descent is prone to getting stuck in local minima.

When I use gradient descent to minimize an empirical risk function, getting stuck at a local minimum that is not global means landing on a parameter vector $\vec{w}$ that is not actually optimal. That is a real problem.

In modern machine learning, the loss surfaces people care about are often huge and messy, and dealing with bad local minima is part of the game. I am not going to get into those fixes here. Instead, I want to focus on a special family of functions where life is much nicer: **convex functions**.

The following video, recorded in an earlier semester, summarizes the key ideas of this section.

Formal Definition of Convexity

Let me start with the picture, because the picture is the whole point.

If you need a refresher on what a secant line is, take a quick look at the appendix. For a scalar-to-scalar function, I want every secant line between two points on the graph to lie on or above the graph itself. If that happens no matter which two points I pick, the function has the familiar bowl-shaped behavior I want.

The figure below is the geometry I have in mind.

That picture turns almost directly into algebra. Suppose I pick two inputs x and y, and some $t \in [0, 1]$.

- The input that is a fraction t of the way from x to y is $(1-t)x + ty$.
- The height of the secant line at that same horizontal location is $(1-t)f(x) + tf(y)$.

So saying "the secant line lies above the graph" is the same as saying

$$f((1-t)x + ty) \leq (1-t)f(x) + tf(y).$$

That is the formal definition in one dimension. In higher dimensions, nothing really changes: I just replace the scalar inputs x and y with vectors $\vec{x}$ and $\vec{y}$, and think about the line segment connecting them.

Definition: Convex Function

Suppose $f : \mathbb{R}^d \to \mathbb{R}$. I say that f is **convex** if for all $\vec{x}, \vec{y} \in \mathbb{R}^d$ and all $t \in [0, 1]$,

$$f((1-t)\vec{x} + t\vec{y}) \leq (1-t)f(\vec{x}) + tf(\vec{y}).$$

For $d = 1$, this is exactly the secant-line picture above. For $d > 1$, it says the same thing along **every** line segment in the domain.

This is one of those definitions that is much more useful than it first looks. I do not just want a fancy way to say "bowl-shaped." I want an inequality that I can actually plug into proofs.

Once I know that every point on every line segment satisfies a weighted-average inequality, I can stop arguing from a picture and start choosing $\vec{x}$, $\vec{y}$, and t strategically. That is what makes the next result work.

Local Minimums are Global Minimums

This is the payoff.

Suppose f is convex, and suppose $\vec{x}^*$ is a local minimum of f. I want to show that $\vec{x}^*$ is automatically a global minimum.

Take any other point $\vec{z}$. Since $\vec{x}^*$ is a local minimum, points on the line segment from $\vec{x}^*$ toward $\vec{z}$ that stay close enough to $\vec{x}^*$ cannot have smaller function value. So for all sufficiently small $t > 0$,

$$f((1-t)\vec{x}^* + t\vec{z}) \geq f(\vec{x}^*).$$

But convexity also gives

$$f((1-t)\vec{x}^* + t\vec{z}) \leq (1-t)f(\vec{x}^*) + tf(\vec{z}).$$

Putting those together,

$$f(\vec{x}^*) \leq (1-t)f(\vec{x}^*) + tf(\vec{z}).$$

Subtract $(1-t)f(\vec{x}^*)$ from both sides:

$$tf(\vec{x}^*) \leq tf(\vec{z}).$$

Since $t > 0$, I can divide by t and get

$$f(\vec{x}^*) \leq f(\vec{z}).$$

And since $\vec{z}$ was arbitrary, $\vec{x}^*$ beats every other point in the domain. So $\vec{x}^*$ is a global minimum.

This is why convexity matters so much in optimization. It does not magically make minimization easy, but it does remove the possibility of bad local minima.

There is one important caveat, though: convex functions do **not** need to have global minima in the first place. A standard example is $f(x) = e^x$. It is convex, and it keeps getting smaller as I move left, but it never actually attains its infimum of 0.

Strict Convexity

Convexity rules out bad local minima, but it does not rule out ties.

A convex function can have a completely flat bottom, in which case every point in that flat region is a local minimum and a global minimum. The example below does exactly that.

So if I want a guarantee of a **single** best point, I need to strengthen the definition a little.

Definition: Strictly Convex Function

Suppose $f : \mathbb{R}^d \to \mathbb{R}$. I say that f is **strictly convex** if for all distinct $\vec{x}, \vec{y} \in \mathbb{R}^d$ and all $t \in (0, 1)$,

$$f((1-t)\vec{x} + t\vec{y}) < (1-t)f(\vec{x}) + tf(\vec{y}).$$

The only difference is that the inequality is now strict once I move away from the endpoints. Geometrically, the secant line is allowed to touch the graph at the two endpoints, but not in between.

Now I can prove the uniqueness statement I really want. Suppose a strictly convex function has a global minimum, but that there are two different minimizers, $\vec{x}^*$ and $\vec{y}^*$. Let their common minimum value be m.

For any $t \in (0, 1)$, strict convexity says

$$f((1-t)\vec{x}^* + t\vec{y}^*) < (1-t)f(\vec{x}^*) + tf(\vec{y}^*) = (1-t)m + tm = m.$$

But that says there is a point with function value **smaller** than the global minimum value m, which is impossible. So a strictly convex function can have **at most one** global minimum. In other words: if a global minimum exists, it is unique.

Strict Convexity and Mean Squared Error. This is exactly the distinction I want you to keep in mind for mean squared error.

Recall that

$$R_{\text{sq}}(\vec{w}) = \frac{1}{n}\|\vec{y} - X\vec{w}\|^2.$$

When the columns of X are linearly independent, this risk surface curves upward in every direction, so there is a single best parameter vector. When the columns of X are linearly dependent, there are flat directions: I can move in some directions without changing the predictions, and the minimum need not be unique.

The contour plots below show both behaviors.

The left-hand plot is the nice case: one bowl, one bottom, one minimizer. The right-hand plot still describes a convex function, but not a strictly convex one, because there is a whole line of minimizers.

I am not proving the full criterion for R_{sq} here just yet. The clean explanation comes from the Hessian, and one chapter from now we will phrase that explanation using eigenvalues of $X^T X$. But the geometry is already visible: full rank gives curvature in every direction; linear dependence creates flat directions.

Second Derivative Test

The formal definition is the ground truth, but it is not always the fastest way to check that a function is convex.

For scalar-to-scalar functions, you may already know the second derivative test from calculus: if a twice-differentiable function satisfies

$$\frac{d^2 f}{dx^2}(x) > 0$$

for all x in its domain, then f is convex. In fact, the slightly weaker condition $\frac{d^2f}{dx^2}(x) \geq 0$ is enough for convexity, while > 0 points toward strict convexity.

What does this look like for vector-to-scalar functions? Now there is no single second derivative. There are many of them.

For example, if

$$f(x_1, x_2) = x_1^2 + x_1 x_2 + 2x_2^2,$$

then the first partial derivatives are

$$\frac{\partial f}{\partial x_1} = 2x_1 + x_2, \qquad \frac{\partial f}{\partial x_2} = x_1 + 4x_2,$$

and the second partial derivatives are

$$\frac{\partial^2 f}{\partial x_1^2} = 2, \qquad \frac{\partial^2 f}{\partial x_1 \partial x_2} = 1, \qquad \frac{\partial^2 f}{\partial x_2^2} = 4.$$

The natural thing to do is collect all of those second partial derivatives into a matrix.

Definition: Hessian

Suppose $f : \mathbb{R}^d \to \mathbb{R}$ is twice differentiable. The **Hessian** of f at $\vec{x}$ is the matrix of second partial derivatives,

$$H_f(\vec{x}) = \begin{bmatrix} \frac{\partial^2 f}{\partial x_1^2} & \frac{\partial^2 f}{\partial x_1 \partial x_2} & \cdots & \frac{\partial^2 f}{\partial x_1 \partial x_d} \\ \frac{\partial^2 f}{\partial x_2 \partial x_1} & \frac{\partial^2 f}{\partial x_2^2} & \cdots & \frac{\partial^2 f}{\partial x_2 \partial x_d} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial^2 f}{\partial x_d \partial x_1} & \frac{\partial^2 f}{\partial x_d \partial x_2} & \cdots & \frac{\partial^2 f}{\partial x_d^2} \end{bmatrix}.$$

For the example above,

$$H_f(\vec{x}) = \begin{bmatrix} 2 & 1 \\ 1 & 4 \end{bmatrix},$$

which does not even depend on $\vec{x}$.

The vector-valued second derivative test says: a twice-differentiable function f is convex exactly when its Hessian is **positive semidefinite** everywhere, meaning

$$\vec{v}^T H_f(\vec{x}) \vec{v} \geq 0 \qquad \text{for all } \vec{x} \in \mathbb{R}^d \text{ and all } \vec{v} \in \mathbb{R}^d.$$

If the Hessian is positive definite everywhere, then I get strict convexity.

This brings us back to mean squared error. For

$$R_{\text{sq}}(\vec{w}) = \frac{1}{n} \|\vec{y} - X\vec{w}\|^2,$$

we already computed

$$\nabla R_{\text{sq}}(\vec{w}) = \frac{2}{n}(X^T X\vec{w} - X^T\vec{y}).$$

Differentiate once more, and the Hessian is

$$H_{R_{\text{sq}}}(\vec{w}) = \frac{2}{n}X^T X.$$

That is a really nice conclusion: the Hessian is constant, and it is always positive semidefinite. So mean squared error is always convex. If the columns of X are linearly independent, then $X^T X$ is positive definite, which is why R_{sq} becomes strictly convex and has a unique minimizer. If the columns are dependent, there is a zero-curvature direction, which is exactly the flat valley we saw above.

Aside: Tangent Hyperplanes

For scalar-to-scalar functions, there is another way to recognize convexity: a differentiable convex function always lies above each of its tangent lines.

For vector-to-scalar functions, the same story holds, except the tangent line becomes a **tangent hyperplane**. When the input has two coordinates, that hyperplane is literally just a plane in 3D.

The next figure shows a convex surface together with its tangent plane at one point.

Suppose I fix a point $\vec{a}$. The tangent hyperplane to f at $\vec{a}$ is the linear approximation

$$L_{\vec{a}}(\vec{x}) = f(\vec{a}) + \nabla f(\vec{a})^T(\vec{x} - \vec{a}).$$

This is the vector-valued version of the tangent-line formula from calculus.

First-order characterization of convexity

If $f : \mathbb{R}^d \to \mathbb{R}$ is differentiable, then f is convex if and only if

$$f(\vec{y}) \geq f(\vec{x}) + \nabla f(\vec{x})^T(\vec{y} - \vec{x})$$

for all $\vec{x}, \vec{y} \in \mathbb{R}^d$.

In words: **the graph of a differentiable convex function lies above every tangent hyperplane**.

I like this characterization because it tells me that the local linear information in the gradient is globally trustworthy. On a non-convex function, a tangent line or tangent plane can point you in a misleading direction. On a convex function, the tangent hyperplane never overshoots the graph, which is a big part of why gradient-based optimization behaves so nicely in this setting.

Eigenvalues and Eigenvectors

9.1. Eigenvalues and Eigenvectors

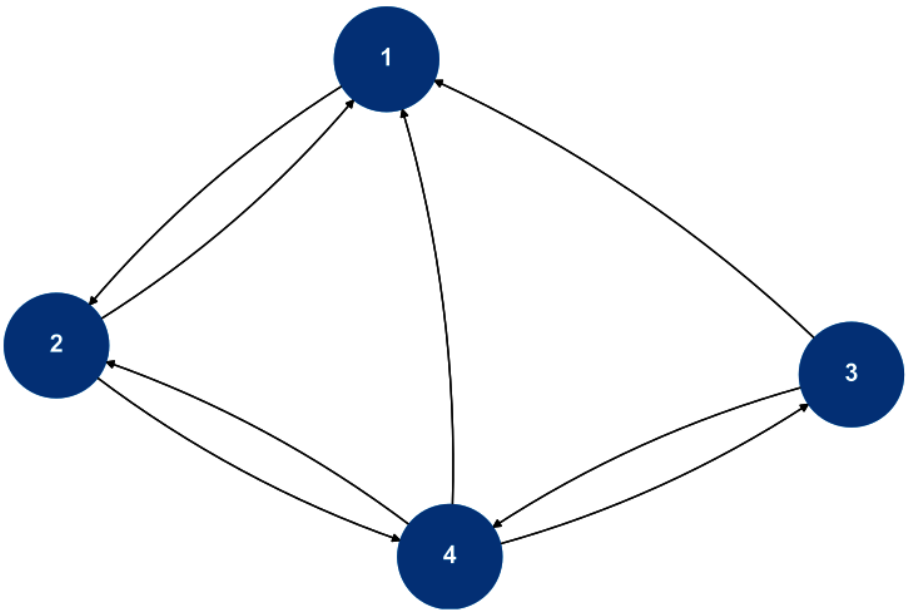
Overview

In Chapter 9 – which will last us the majority of the remainder of the semester – we're going to introduce a new lens through which we can view the information stored in a matrix: through its **eigenvalues** and **eigenvectors**. As Gilbert Strang says in his book, eigenvalues and eigenvectors allow us to look into the "heart" of a matrix and see what it's really doing.

Throughout this chapter, we'll see how eigen-things can help us more deeply understand the topics we've already covered, like linear regression, the normal equations, gradient descent, and convexity.

- The eigenvalues of the Hessian matrix (matrix of partial second derivatives) of a vector-to-scalar function will tell us whether or not the function is convex, i.e. they will give us a "second derivative test" for vector-to-scalar functions, which we haven't yet seen.
- Eigenvalues will explain why the matrix $X^TX + \lambda I$ is always invertible, even if X's columns aren't linearly independent.
- And eventually, in Chapter 10.4, we'll use eigenvalues and eigenvectors to address the dimensionality reduction problem, first introduced in Chapter 1.1.

On top of all of that, eigenvalues and eigenvectors will unlock a new set of applications – those that involve some element of **time**. My favorite such example is Google's PageRank algorithm. The algorithm, first published in 1998 by Sergey Brin and Larry Page (Google's cofounders, the latter of whom is a Michigan alum), is used to rank pages on the internet based on their relative importance.



From the research paper linked above:

> PageRank or PR(A) can be calculated using a simple iterative algorithm, and corresponds to the principal eigenvector of the normalized link matrix of the web.

We'll make sense of the algorithm in Homework 10: just know that this is where we're heading.

Introduction

Before we get started, keep in mind that everything we're about to introduce only applies to square matrices. This was also true when we first studied invertibility, and for the same reason: we should think of eigenvalues and eigenvectors as properties of a linear transformation from $\mathbb{R}^n$ to $\mathbb{R}^n$ (that is, from a vector space to itself), not between vector spaces of different dimensions. Rectangular matrices will have their moment in Chapter 10.1.

Definition: Eigenvalues and Eigenvectors

Suppose A is an $n \times n$ matrix. A **non-zero** vector $\vec{v} \in \mathbb{R}^n$ is an **eigenvector** of A if:

$$A\vec{v} = \lambda\vec{v}$$

for some scalar $\lambda \in \mathbb{R}$. The scalar λ is called the **eigenvalue** of A corresponding to $\vec{v}$.

This definition is a bit hard to parse when you first look at it. But here's the intuitive interpretation.

If $\vec{v}$ is an eigenvector of A, then $A\vec{v}$ is a vector that points in the same direction as $\vec{v}$, just stretched by a factor of λ!

As we've seen, the function $f(\vec{v}) = A\vec{v}$ is a **linear transformation** that maps vectors in $\mathbb{R}^n$ to vectors in $\mathbb{R}^n$. Eigenvectors of A are vectors whose **directions are unchanged** by the linear transformation f. The corresponding eigenvalues are the scalars by which the eigenvectors are scaled.

"Eigen" is a German word meaning "own", as in "one's own". So, an eigenvector is a vector who still points in its own direction when transformed by A.

A First Example. Let's start with a 2×2 matrix:

$$A = \begin{bmatrix} 1 & 2 \\ 2 & 1 \end{bmatrix}$$

I've chosen the numbers in A to be small enough that we can roughly eyeball the eigenvectors. Here's how I look at A:

- First, notice that both rows of A sum to 3, meaning that

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

  This tells me that $\begin{bmatrix} 1 \\ 1 \end{bmatrix}$ is an eigenvector of A with eigenvalue 3. But, $\begin{bmatrix} 2 \\ 2 \end{bmatrix}$ is also an eigenvector of A with the same eigenvalue, since

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

  Indeed, if $\vec{v}$ is an eigenvector of A with eigenvalue λ, then so is $c\vec{v}$ for any non-zero scalar c. So really, eigenvectors define **directions**.

- Additionally, noticing that there'd be *some* symmetry if I took the difference of the entries in each row of A, consider

$$\begin{bmatrix} 1 & 2 \\ 2 & 1 \end{bmatrix} \begin{bmatrix} -1 \\ 1 \end{bmatrix} = \begin{bmatrix} 1 \\ -1 \end{bmatrix} = -1 \begin{bmatrix} -1 \\ 1 \end{bmatrix}$$

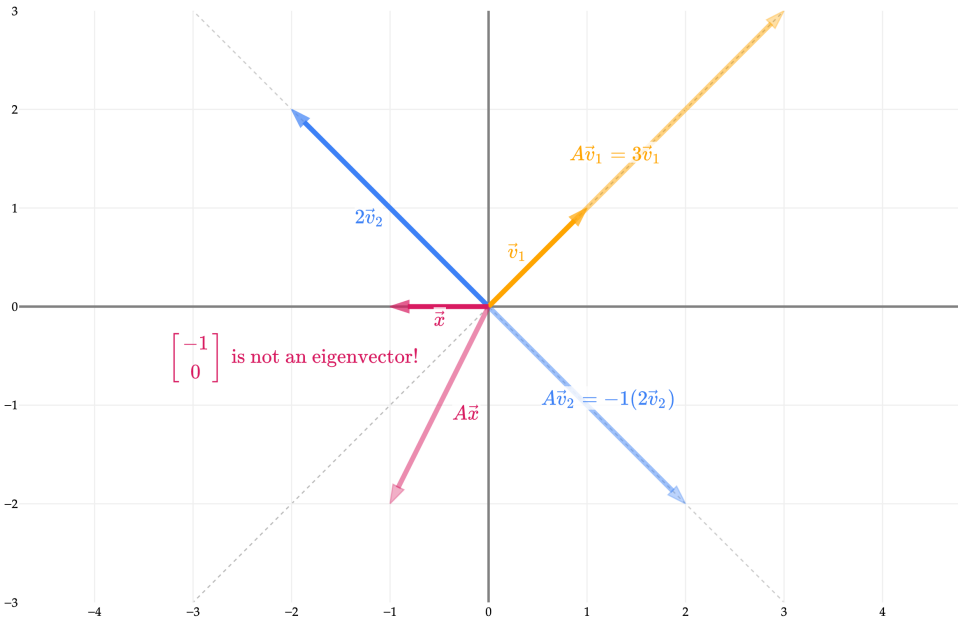
  This tells me that $\begin{bmatrix} -1 \\ 1 \end{bmatrix}$ is an eigenvector of A with eigenvalue -1.

So, $A = \begin{bmatrix} 1 & 2 \\ 2 & 1 \end{bmatrix}$ has two eigenvalues, 3 and -1, and two corresponding eigenvectors, $\begin{bmatrix} 1 \\ 1 \end{bmatrix}$ and $\begin{bmatrix} -1 \\ 1 \end{bmatrix}$. In general, an $n \times n$ matrix has n eigenvalues, but some of them may be the same, and some of them may not be real numbers. We'll see how to systematically find these eigenvalues and eigenvectors in just a bit.

Visually, this means that $\vec{v}_1 = \begin{bmatrix} 1 \\ 1 \end{bmatrix}$ lives on the same line as $A\vec{v}_1$, which is also the line that $c\vec{v}_1$ and $A(c\vec{v}_1)$ live on (for any non-zero scalar c). And, $\vec{v}_2 = \begin{bmatrix} -1 \\ 1 \end{bmatrix}$ lives on the same line as $A\vec{v}_2$.

But, if a vector isn't already on one of the two aforementioned lines – like $\vec{x} = \begin{bmatrix} -1 \\ 0 \end{bmatrix}$ – then it will change directions when multiplied by A, and it is not an eigenvector.

Visualizing the eigenvectors of $A = \begin{bmatrix} 1 & 2 \\ 2 & 1 \end{bmatrix}$



Just to be 100% clear, I've used $2\vec{v}_2$ instead of $\vec{v}_2$ above just to illustrate the fact that eigenvectors are only defined up to a scalar multiple; $2\vec{v}_2$ is just as good as an eigenvector as $\vec{v}_2$ is, and both correspond to the same eigenvalue, $\lambda = 1$.

You might notice that the two eigenvectors of A corresponding to the two different eigenvalues are orthogonal in the example above. This is not true in general for any 2×2 matrix, but there's a specific reason it's true for A: it's **symmetric**. I'll elaborate more on this idea in Chapter 9.5, but for now, just remember that symmetric matrices have orthogonal eigenvectors.

Finding Eigenvalues using `numpy`. Just to show you another example, consider

$$B = \begin{bmatrix} 2 & 5 \\ 1 & 0 \end{bmatrix}$$

Its eigenvalues and eigenvectors aren't particularly nice, and since we don't yet have a way to find them by hand, now is as good as a time as any to use `numpy`:

```
B = np.array([[2, 5],
              [1, 0]])

np.linalg.eig(B)
```

```
EigResult(eigenvalues=array([ 3.44948974, -1.44948974]), eigenvectors=array([[ 0.96045535, -0.82311
       [ 0.27843404,  0.56786837]]))
```

So, B has eigenvalues of ≈ 3.45 and ≈ -1.45. **Note that the eigenvectors are the columns of the matrix returned**, not the rows!

```
eigvals, eigvecs = np.linalg.eig(B)
for i in range(eigvecs.shape[1]):
    print(f"Eigenvector {i+1}: {eigvecs[:, i]}")
    print(f"Eigenvalue {i+1}: {eigvals[i]}")
    print()
```

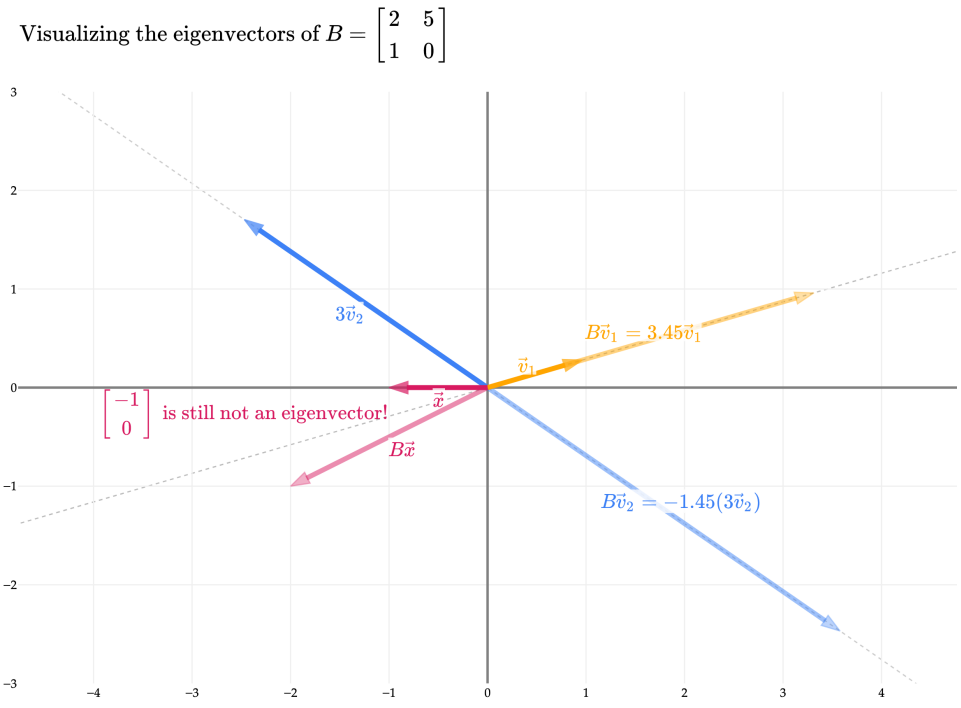
```
Eigenvector 1: [0.96045535 0.27843404]
Eigenvalue 1: 3.4494897427831783

Eigenvector 2: [ -0.82311938  0.56786837]
Eigenvalue 2: -1.449489742783178
```

`eigvecs` is a matrix where each column is an eigenvector of B. For now, call this matrix P. Below, I calculate P^TP, which contains the dot products of all pairs of eigenvectors of B. The diagonal of this matrix contains the dot products of each eigenvector with itself; since these are 1, this tells us that the returned eigenvectors are unit vectors. This was a design decision by the implementors of `np.linalg.eig` – remember that we can scale an eigenvector by any non-zero scalar and it is still an eigenvector. The off-diagonal entries of -0.632 tell us that the two eigenvectors are not orthogonal.

```
eigvecs.T @ eigvecs
```

```
array([[ 1.        , -0.63245553],
       [ -0.63245553,  1.        ]])
```

Let's take a look at the directions of the eigenvectors for B.



While the eigenvectors of B are not orthogonal, they are still linearly independent and span all of $\mathbb{R}^2$. This was also the case in the A example above. **The fact that the eigenvectors of an $n \times n$ matrix are linearly independent and span all of $\mathbb{R}^n$ is also not guaranteed to be true, though it's a desirable property.** The class of matrices that have this property are called **diagonalizable**, which are the focus of Chapter 9.4.

Let's consider a few more examples, each of which is meant to highlight a different key property of eigenvalues and eigenvectors.

Example: Matrix Powers. Let $A = \begin{bmatrix} 1 & 2 \\ 2 & 1 \end{bmatrix}$ be the matrix from the first example above. What are the eigenvalues and eigenvectors of A^2? Find them manually. You should notice the property below.

Solution

$$A^2 = \begin{bmatrix} 1 & 2 \\ 2 & 1 \end{bmatrix} \begin{bmatrix} 1 & 2 \\ 2 & 1 \end{bmatrix} = \begin{bmatrix} 5 & 4 \\ 4 & 5 \end{bmatrix}$$

Similar to the original example, A^2's rows both sum to the same number, 9, meaning that

$$\begin{bmatrix} 5 & 4 \\ 4 & 5 \end{bmatrix} \begin{bmatrix} 1 \\ 1 \end{bmatrix} = \begin{bmatrix} 9 \\ 9 \end{bmatrix} = 9 \begin{bmatrix} 1 \\ 1 \end{bmatrix}$$

We can do the same thing with $A^2 \begin{bmatrix} 1 \\ -1 \end{bmatrix}$:

$$A^2 \begin{bmatrix} 1 \\ -1 \end{bmatrix} = \begin{bmatrix} 1 \\ -1 \end{bmatrix} = 1 \begin{bmatrix} 1 \\ -1 \end{bmatrix}$$

So the eigenvalues of A^2 are 9 and 1, and their respective eigenvectors are $\begin{bmatrix} 1 \\ 1 \end{bmatrix}$ and $\begin{bmatrix} 1 \\ -1 \end{bmatrix}$.

But, these are the same two eigenvectors that A has, and the corresponding eigenvalues are the squares of A's eigenvalues, since $3^2 = 9$ and $(-1)^2 = 1$. Is this true more generally? **Yes!**

If λ is an eigenvalue of A with eigenvector $\vec{v}$, then λ^k is an eigenvalue of A^k with the same eigenvector $\vec{v}$. (Click to see the proof.)

A quick proof: suppose $A\vec{v} = \lambda\vec{v}$. Then,

$$A^2\vec{v} = A(A\vec{v}) = A(\lambda\vec{v}) = \lambda(A\vec{v}) = \lambda^2\vec{v}$$

This logic can be extended to A^3, then A^4, and so on. (If you are familiar with induction from EECS 203, you can think of this as an inductive proof.)

Note that the converse of this statement is not necessarily true, meaning it's possible for A^2 to have an eigenvector that is not an eigenvector of A. For example, if A corresponds to a rotation by 90º degrees (or $\frac{\pi}{2}$ radians), then A^2 corresponds to a rotation by 180º degrees (or π radians). No vector lies on the same line after rotation by 90º degrees, but all vectors lie on the same line after rotation by 180º degrees. So, in this example, A^2 has plenty of eigenvectors that A does not have.

Example: Non-Invertible Matrices. Let $A = \begin{bmatrix} 1 & 4 \\ 3 & 12 \end{bmatrix}$. Notice that $\text{rank}(A) = 1$. A has an eigenvalue of 13 with eigenvector $\begin{bmatrix} 1 \\ 3 \end{bmatrix}$ – verify that this is the case. Does it have another eigenvalue? What is the corresponding eigenvector?

Solution

Since $\text{rank}(A) = 1$, A has a non-trivial null space, and any vector in the null space will get sent to 0 times itself when multiplied by A. Since $4 \cdot \text{column } 1 = \text{column } 2$, the null space of A is spanned by the vector $\begin{bmatrix} 4 \\ -1 \end{bmatrix}$. So,

$$A\begin{bmatrix} 4 \\ -1 \end{bmatrix} = \begin{bmatrix} 1 & 4 \\ 3 & 12 \end{bmatrix}\begin{bmatrix} 4 \\ -1 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \end{bmatrix} = 0\begin{bmatrix} 4 \\ -3 \end{bmatrix}$$

$\vec{0}$ can't be an eigenvector, but 0 is a perfectly good eigenvalue.
Our intuition tells us that A should have another eigenvalue, since it's a 2×2 matrix. It happens to be 13, corresponding to the eigenvector $\begin{bmatrix} 1 \\ 3 \end{bmatrix}$. In Chapter 9.2: The Characteristic Polynomial, we'll see how to find this eigenvalue-eigenvector pair without guesswork.

0 is an eigenvalue of A if and only if A is not invertible. (Click to see more details.)

If A is invertible, then the only solution to $A\vec{v} = 0\vec{v} = \vec{0}$ is the trivial solution where $\vec{v}$ is the zero vector itself. But, we defined eigenvectors to be non-zero vectors. So, 0 can't be an eigenvalue of an invertible matrix. If A is not invertible, then A has a non-trivial null space, and any non-zero vector in the null space is an eigenvector corresponding to the eigenvalue 0.

Example: The Identity Matrix. What are the eigenvalues and eigenvectors of $I = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$?

Solution

The identity matrix multiplied by any vector $\vec{v}$ returns that same vector back, meaning that all vectors in $\mathbb{R}^2$ are eigenvectors of I, all with eigenvalue 1.
This is the first example we've seen so far where there exist multiple "lines" or "directions" of eigenvectors for a single eigenvalue. The vectors $\begin{bmatrix} 2 \\ 3 \end{bmatrix}$ and $\begin{bmatrix} -1 \\ 4 \end{bmatrix}$ are both eigenvectors of I with eigenvalue 1 but they don't lie on the same line. We will study this idea – of having multiple eigenvector directions for a single eigenvalue – more precisely in Chapter 9.4.

A question on your mind might be, how do we find the eigenvalues of a generic matrix, when they aren't easy to eyeball? Keep reading! The bottom of Chapter 9.2 also has a great summary of the key ideas from this section.

9.2. The Characteristic Polynomial

So far, we've found the eigenvalues and eigenvectors of a matrix by eyeballing them or reasoning about them geometrically, but this is not a sustainable strategy. Let's develop a more systematic approach.

Deriving the Characteristic Polynomial

We're looking for combinations of scalars, λ, and vectors, $\vec{v}$, such that

$$A\vec{v} = \lambda\vec{v}$$

In some ways, we're trying to "solve" for λ and $\vec{v}$ given A. Let's experiment a little:

$$\begin{aligned} A\vec{v} &= \lambda\vec{v} \\ A\vec{v} - \lambda\vec{v} &= \vec{0} \\ (A - \lambda I)\vec{v} &= \vec{0} \end{aligned}$$

What the above says is that **if $\vec{v}$ is an eigenvector of A with eigenvalue λ, then $\vec{v}$ is in the null space of $A - \lambda I$.** But, since eigenvectors can't be the zero vector, this means that $A - \lambda I$ is not invertible, since it has a non-trivial null space!

Thinking back to Chapter 6.2, we know that there are several equivalent ways to check if a matrix is not invertible. Perhaps the most computational approach is to compute its determinant; if a (square) matrix's determinant is 0, then it is not invertible, otherwise it is.

So, since $A - \lambda I$ is not invertible, its determinant must be 0!

$$A\vec{v} = \lambda\vec{v} \implies \det(A - \lambda I) = 0$$

Definition: Characteristic Polynomial

The **characteristic polynomial** of an $n \times n$ matrix A is defined as

$$p(\lambda) = \det(A - \lambda I)$$

This is a polynomial in λ of degree n.
The eigenvalues of A are the roots of its characteristic polynomial, i.e. the values of λ that satisfy $\det(A - \lambda I) = 0$.

(We won't always use the symbol $p(\lambda)$, I just introduced it above to make it clear that $\det(A - \lambda I)$ is a polynomial function of λ.)

Let's revisit our example $A = \begin{bmatrix} 1 & 2 \\ 2 & 1 \end{bmatrix}$.

The matrix $A - \lambda I$ is

$$A - \lambda I = \begin{bmatrix} 1 & 2 \\ 2 & 1 \end{bmatrix} - \lambda \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} = \begin{bmatrix} 1 - \lambda & 2 \\ 2 & 1 - \lambda \end{bmatrix}$$

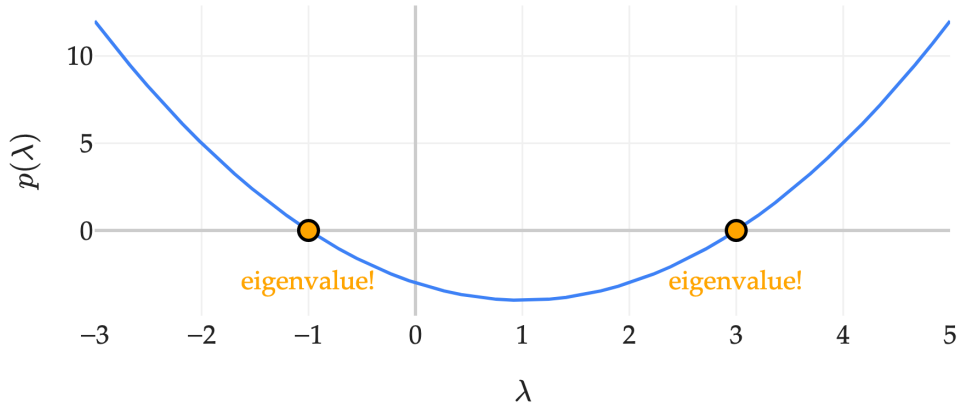
Note that $A - \lambda I$ involves subtracting λ from the diagonal elements of A, and leaving all other elements unchanged.

The determinant of $A - \lambda I$ is

$$\det(A - \lambda I) = (1 - \lambda)(1 - \lambda) - 2 \cdot 2 = \lambda^2 - 2\lambda + 1 - 4 = \underbrace{\lambda^2 - 2\lambda - 3}_{p(\lambda)}$$

The eigenvalues of A are the values of λ where $\lambda^2 - 2\lambda - 3 = 0$.

$p(\lambda) = \lambda^2 - 2\lambda - 3$



(In general, we're not going to plot the characteristic polynomial each time, but I think it's useful to see once or twice to give the idea some context.)

By factoring, I can write this equation as

$$(\lambda + 1)(\lambda - 3) = 0$$

which tells me that the eigenvalues of A are $\lambda_1 = -1$ and $\lambda_2 = 3$. If this equation weren't factorable, I'd need to use the quadratic formula to find the eigenvalues. For now, there's no particular "ordering" to the eigenvalues, i.e. I could have said $\lambda_1 = 3$ and $\lambda_2 = -1$; all that matters is that I stay consistent throughout a particular example.

Once we find the eigenvalues by solving $p(\lambda) = 0$, we can find the eigenvectors by solving $(A - \lambda I)\vec{v} = \vec{0}$ for each eigenvalue.

- For $\lambda_1 = -1$, we're looking for vectors $\vec{v}$ such that $A\vec{v} = -1\vec{v}$, i.e. if $\vec{v} = \begin{bmatrix} a \\ b \end{bmatrix}$, then

$$\begin{bmatrix} 1 & 2 \\ 2 & 1 \end{bmatrix} \begin{bmatrix} a \\ b \end{bmatrix} = \begin{bmatrix} -a \\ -b \end{bmatrix}$$

  (I'm using a and b instead of v_1 and v_2 because I'll refer to the vectors $\vec{v}_1$ and $\vec{v}_2$ in just a moment.) As a system of equations, this says

$$\begin{aligned} a + 2b &= -a \\ 2a + b &= -b \end{aligned}$$

  The first and second equations both tell us that $b = -a$. Remember, we'd expect there to be infinitely many solutions to this system, since any scalar multiple of an eigenvector is still an eigenvector. So, the "simple"

solution is $\vec{v}_1 = \begin{bmatrix} 1 \\ -1 \end{bmatrix}$, but $\begin{bmatrix} 2 \\ -2 \end{bmatrix}$, $\begin{bmatrix} 3 \\ -3 \end{bmatrix}$, etc. are also solutions.

- For $\lambda_2 = 3$, let me introduce another way of finding the corresponding eigenvector. There's nothing wrong with the system of equations approach, but it's useful to have multiple techniques for solving problems in our toolkit. $\lambda_2 = 3$ tells us that $A\vec{v} = 3\vec{v}$, or equivalently, $(A - 3I)\vec{v} = \vec{0}$. So, the eigenvector we're looking for is in the null space of $A - 3I$.

$$A - 3I = \begin{bmatrix} 1-3 & 2 \\ 2 & 1-3 \end{bmatrix} = \begin{bmatrix} -2 & 2 \\ 2 & -2 \end{bmatrix}$$

Here, we notice that both rows of $A - 3I$ sum to 0, meaning that

$$(A - 3I)\begin{bmatrix} 1 \\ 1 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \end{bmatrix}$$

So, $\begin{bmatrix} 1 \\ 1 \end{bmatrix}$ is an eigenvector for $\lambda_2 = 3$.

Examples

Example: 2×2 **Matrices.** Find the eigenvalues and eigenvectors of $A = \begin{bmatrix} 3 & 1 \\ 2 & 4 \end{bmatrix}$.

Solution

$$\begin{aligned} A - \lambda I &= \begin{bmatrix} 3-\lambda & 1 \\ 2 & 4-\lambda \end{bmatrix} \\ \det(A - \lambda I) &= (3-\lambda)(4-\lambda) - (1 \cdot 2) \\ &= 12 - 7\lambda + \lambda^2 - 2 \\ &= \lambda^2 - 7\lambda + 10 \\ &= (\lambda - 2)(\lambda - 5) \end{aligned}$$

So our eigenvalues are $\lambda_1 = 2$ and $\lambda_2 = 5$. Next, let's find eigenvectors for them. As I did in the first example, I'll show two different techniques for finding eigenvectors.

- For $\lambda_1 = 2$, we're looking for a vector $\vec{v} = \begin{bmatrix} a \\ b \end{bmatrix}$ such that $A\vec{v} = 2\vec{v}$:

$$\begin{aligned} A\vec{v} &= 2\vec{v} \\ \begin{bmatrix} 3 & 1 \\ 2 & 4 \end{bmatrix} \begin{bmatrix} a \\ b \end{bmatrix} &= \begin{bmatrix} 2a \\ 2b \end{bmatrix} \\ 3a + b &= 2a \\ 2a + 4b &= 2b \end{aligned}$$

  As expected, there are infinitely many choices for a and b (since both equations tell us that $b = -a$), and the resulting eigenvectors $\vec{v}$ all lie on the same line. So, if we let $a = 1$, then we have that $\vec{v}_1 = \begin{bmatrix} 1 \\ -1 \end{bmatrix}$ is an eigenvector for λ_1.

- For $\lambda_2 = 5$, I'll try the null space approach, just to illustrate it once more. Again, I'm looking for a vector $\vec{v}$ such that $A\vec{v} = 5\vec{v}$, or equivalently, $(A - 5I)\vec{v} = \vec{0}$.

$$A - 5I = \begin{bmatrix} 3-5 & 1 \\ 2 & 4-5 \end{bmatrix} = \begin{bmatrix} -2 & 1 \\ 2 & -1 \end{bmatrix}$$

  The vector $\vec{v}_2 = \begin{bmatrix} 1 \\ 2 \end{bmatrix}$ is orthogonal to both rows of $A - 5I$, meaning $(A - 5I)\begin{bmatrix} 1 \\ 2 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \end{bmatrix}$. So, $\vec{v}_2 = \begin{bmatrix} 1 \\ 2 \end{bmatrix}$ is an eigenvector for $\lambda_2 = 5$, though again any non-zero scalar multiple of $\begin{bmatrix} 1 \\ 2 \end{bmatrix}$ will also be an eigenvector for $\lambda_2 = 5$.

So, A has eigenvalues $\lambda_1 = 2$ and $\lambda_2 = 5$, and corresponding eigenvectors $\vec{v}_1 = \begin{bmatrix} 1 \\ -1 \end{bmatrix}$ and $\vec{v}_2 = \begin{bmatrix} 1 \\ 2 \end{bmatrix}$.

Example: Diagonal Matrices. Find the eigenvalues and eigenvectors of $A = \begin{bmatrix} 3 & 0 \\ 0 & -4 \end{bmatrix}$. What do you notice about the eigenvalues? The eigenvectors?

Solution

The key is to notice that for diagonal matrices – that is, matrices where all the entries off the diagonal are 0 – the eigenvalues are simply the entries on the diagonal.

$$A - \lambda I = \begin{bmatrix} 3-\lambda & 0 \\ 0 & -4-\lambda \end{bmatrix}$$
$$\det(A - \lambda I) = (3-\lambda)(-4-\lambda)$$

The eigenvalues $\lambda_1 = 3$ and $\lambda_2 = -4$ are the same as the values on the diagonal! The corresponding eigenvectors are $\vec{v}_1 = \begin{bmatrix} 1 \\ 0 \end{bmatrix}$ and $\vec{v}_2 = \begin{bmatrix} 0 \\ 1 \end{bmatrix}$. Just to illustrate one of those cases out,

$$A\vec{v}_2 = \begin{bmatrix} 3 & 0 \\ 0 & -4 \end{bmatrix} \begin{bmatrix} 0 \\ 1 \end{bmatrix} = \begin{bmatrix} 0 \\ -4 \end{bmatrix} = -4 \begin{bmatrix} 0 \\ 1 \end{bmatrix}$$

The fact that eigenvalues and eigenvectors are so easy to find for diagonal matrices, coupled with the fact that diagonal matrices are really easy to compute powers of (e.g. A^2 is just the diagonal matrix with each entry squared), makes them very useful in practice.

Example: Sharing a Characteristic Polynomial. Find two **different** 2×2 matrices who have the characteristic polynomial

$$p(\lambda) = \lambda^2 - 4\lambda + 3$$

Solution

Recall that for a 2×2 matrix $A = \begin{bmatrix} a & b \\ c & d \end{bmatrix}$:

$$\begin{aligned} p(\lambda) &= \det(A - \lambda I) \\ &= \begin{vmatrix} a-\lambda & b \\ c & d-\lambda \end{vmatrix} \\ &= (a-\lambda)(d-\lambda) - bc \end{aligned}$$

A simple approach to this problem is to find 2 diagonal matrices $D = \begin{bmatrix} a & 0 \\ 0 & d \end{bmatrix}$ and $D' = \begin{bmatrix} d & 0 \\ 0 & a \end{bmatrix}$. That way, all we have to do is factor $p(\lambda)$ into the form $(a-\lambda)(d-\lambda)$:

$$\lambda^2 - 4\lambda + 3 = (3-\lambda)(1-\lambda)$$

So, our two different matrices are $D = \begin{bmatrix} 3 & 0 \\ 0 & 1 \end{bmatrix}$ and $D' = \begin{bmatrix} 1 & 0 \\ 0 & 3 \end{bmatrix}$.
There exist non-diagonal matrices with the same characteristic polynomial, too. For instance,

$$\begin{bmatrix} 2 & 1 \\ 1 & 2 \end{bmatrix}$$

also has eigenvalues of 1 and 3, and its characteristic polynomial is also $p(\lambda) = \lambda^2 - 4\lambda + 3$.

Example: 3×3 Matrices. Recall, if A is a 3×3 matrix, like

$$A = \begin{bmatrix} a & b & c \\ d & e & f \\ g & h & i \end{bmatrix}$$

then $\det(A)$ is

$$\det(A) = a \begin{vmatrix} e & f \\ h & i \end{vmatrix} - b \begin{vmatrix} d & f \\ g & i \end{vmatrix} + c \begin{vmatrix} d & e \\ g & h \end{vmatrix}$$

where $|\cdot|$ denotes the determinant of the matrix.

Find the eigenvalues and eigenvectors of the **upper triangular** matrix

$$A = \begin{bmatrix} 3 & 3 & 0 \\ 0 & 1 & 5 \\ 0 & 0 & 2 \end{bmatrix}$$

What do you notice about the eigenvalues?

Solution

We'll start by calculating $\det(A - \lambda I)$ using the formula for the 3×3 determinant provided for us:

$$A - \lambda I = \begin{bmatrix} 3-\lambda & 3 & 0 \\ 0 & 1-\lambda & 5 \\ 0 & 0 & 2-\lambda \end{bmatrix}$$

$$\begin{aligned} \det(A - \lambda I) &= (3-\lambda)\begin{vmatrix} 1-\lambda & 5 \\ 0 & 2-\lambda \end{vmatrix} - 3\begin{vmatrix} 0 & 5 \\ 0 & 2-\lambda \end{vmatrix} + 0\begin{vmatrix} 0 & 1-\lambda \\ 0 & 0 \end{vmatrix} \\ &= (3-\lambda)\big[(1-\lambda)(2-\lambda) - 0\big] - 3\big[0-0\big] + 0\big[0-0\big] \\ &= (3-\lambda)(1-\lambda)(2-\lambda) \end{aligned}$$

So, the eigenvalues are the values along the diagonal: $\lambda_1 = 3$, $\lambda_2 = 1$, and $\lambda_3 = 2$. This is a convenient property of triangular matrices (which is why I've planted this example here). Next, let's find their eigenvectors.

- For $\lambda_1 = 3$, I'll do it the null space way: I'm looking for a vector in the null space of $A - 3I$. (I say "the null space way" but it's really just a different way of writing the system of equations approach, and one that can sometimes be easier to eyeball.)

  $$A - 3I = \begin{bmatrix} 0 & 3 & 0 \\ 0 & -2 & 5 \\ 0 & 0 & -1 \end{bmatrix}$$

  The zeros in the first column tell me that $(A - 3I)\begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix}$, so $\vec{v}_1 = \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix}$ is an eigenvector for $\lambda_1 = 3$.

  Because I have three unique eigenvalues and my matrix is 3×3, I know that this is the only possible eigenvector direction for $\lambda_1 = 3$; in Chapter 9.4, we'll run into situations where the null space of $A - 3I$ (for instance) is spanned by more than one vector, but that's not the case here.

- For $\lambda_2 = 1$, I'll do it the system of equations way: I'm looking for a vector $\vec{v} = \begin{bmatrix} a \\ b \\ c \end{bmatrix}$ such that $A\vec{v} = 1\vec{v}$.

  $$\begin{aligned} A\vec{v} &= \vec{v} \\ \begin{bmatrix} 3 & 3 & 0 \\ 0 & 1 & 5 \\ 0 & 0 & 2 \end{bmatrix}\begin{bmatrix} a \\ b \\ c \end{bmatrix} &= \begin{bmatrix} a \\ b \\ c \end{bmatrix} \\ 3a + 3b &= a \\ b + 5c &= b \\ 2c &= c \end{aligned}$$

  Once again, there are infinitely many solutions to this system, which we'd expect since there's a whole line of eigenvectors.

  - The third equation above tells us that $c = 0$.
  - Substituting that into the second equation, we have that $b + 5 \cdot 0 = b$, i.e. $b = b$, so let's treat b as a free variable for now.
  - The first equation above tells us that $3a + 3b = a$, i.e. $a = -\frac{3}{2}b$.

  So, any vector of the form $\begin{bmatrix} -\frac{3}{2}b \\ b \\ 0 \end{bmatrix}$ is an eigenvector for $\lambda_2 = 1$. But we just need to find one, so let's

For upper triangular and lower triangular matrices, the eigenvalues are the values along the diagonal!

Recall, an upper triangular matrix has all zeros below the diagonal, and a lower triangular matrix has all zeros above the diagonal.
This comes from the fact that the determinant of an upper triangular matrix is the product of the diagonal entries, and the determinant of a lower triangular matrix is the product of the diagonal entries.
So, for example, the eigenvalues of

$$\begin{bmatrix} 10 & 3 & 2 & -50 \\ 0 & -94 & 2 & 1 \\ 0 & 0 & 0 & 3 \\ 0 & 0 & 0 & 12 \end{bmatrix}$$

are 10, -94, 0, and 12. And, the eigenvalues of

$$\begin{bmatrix} 2 & 0 & 0 \\ -1 & 5 & 0 \\ 4 & 3 & -7 \end{bmatrix}$$

are 2, 5, and -7.

```
# When in doubt, check with numpy!
np.linalg.eig(
    np.array([
        [2, 0, 0],
        [ -1, 5, 0],
        [4, 3, -7]
    ])
)
```

```
EigResult(eigenvalues=array([ -7.,  5.,  2.]), eigenvectors=array([[0.        , 0.        , 0.839254
       [0.        , 0.9701425 , 0.27975144],
       [1.        , 0.24253563, 0.4662524 ]]))
```

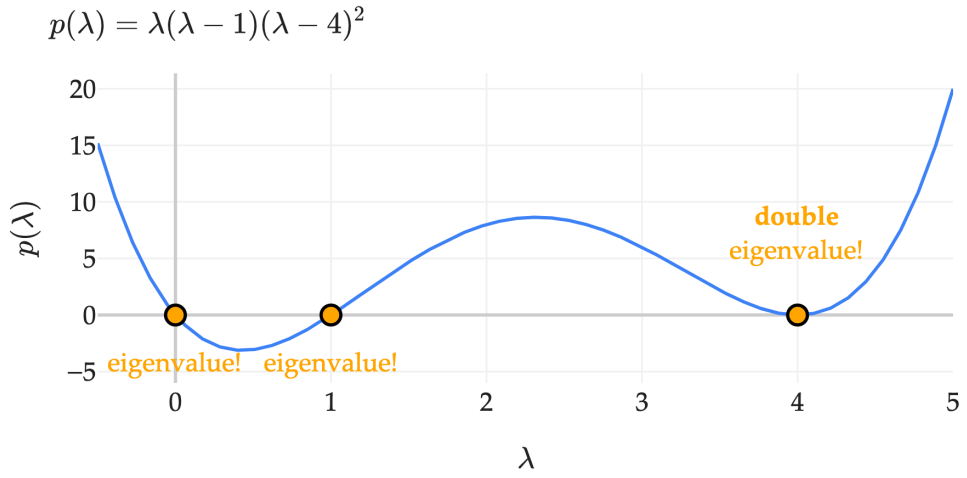
Number of Eigenvalues. The characteristic polynomial of an $n \times n$ matrix is a polynomial of degree n. A fact from algebra is that a polynomial of degree n has exactly n roots. The intuitive way of thinking about this is that degree n polynomials can have at most $n-1$ "bends" (a line can't bend, a quadratic can bend once, a cubic can bend twice, etc.), and each time it bends, it can change directions to cross the x-axis again.

The issue is that some of these roots may be repeated, and some may be complex numbers, meaning that they don't actually cross the x-axis in the standard xy-plane (or in our case, the λ-axis and $\lambda, p(\lambda)$-plane).

For example, the matrix $A = \begin{bmatrix} 4 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 4 \end{bmatrix}$ is diagonal, meaning its easy to read off its characteristic polynomial:

$$p(\lambda) = \det(A - \lambda I) = \begin{vmatrix} 4-\lambda & 0 & 0 & 0 \\ 0 & 1-\lambda & 0 & 0 \\ 0 & 0 & -\lambda & 0 \\ 0 & 0 & 0 & 4-\lambda \end{vmatrix} = \lambda(\lambda-1)(\lambda-4)^2$$

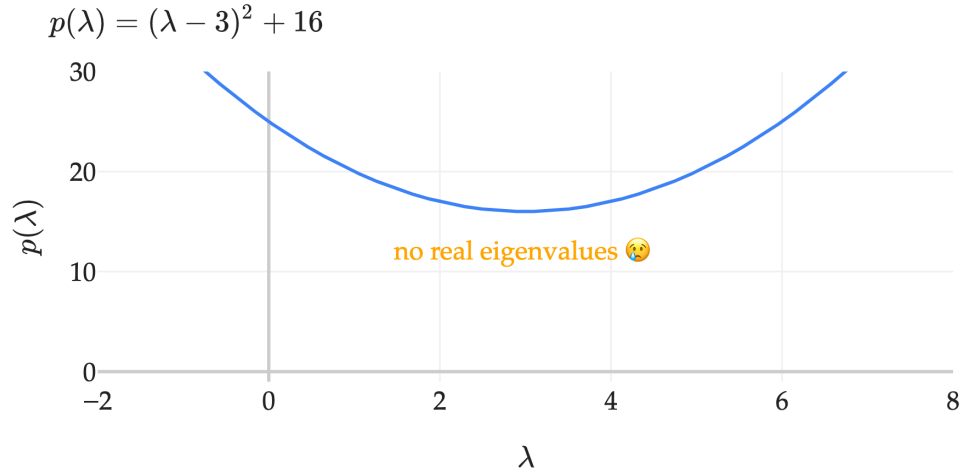
This characteristic polynomial has a double root at $\lambda = 4$, a single root at $\lambda = 1$, and a single root at $\lambda = 0$. This A has 3 distinct eigenvalues, but one of them, $\lambda = 2$, is repeated, and has an **algebraic multiplicity** of 2.



As another example, the matrix $A = \begin{bmatrix} 3 & -4 \\ 4 & 3 \end{bmatrix}$ has the characteristic polynomial

$$p(\lambda) = \begin{vmatrix} 3-\lambda & -4 \\ 4 & 3-\lambda \end{vmatrix} = (\lambda-3)^2 + 16$$

I can expand this out to get $\lambda^2 - 6\lambda + 25$, but in a case like this I think the above form is more telling. Visually, $p(\lambda)$ here is a parabola that sits entirely above the λ-axis, meaning it has no real roots, so A has no real eigenvalues.



It should not be all that surprising that $A = \begin{bmatrix} 3 & -4 \\ 4 & 3 \end{bmatrix}$ has no real eigenvalues. A is a rotation matrix, scaled by a factor of 5:

$$A = \begin{bmatrix} 3 & -4 \\ 4 & 3 \end{bmatrix} = 5\begin{bmatrix} \frac{3}{5} & -\frac{4}{5} \\ \frac{4}{5} & \frac{3}{5} \end{bmatrix} = 5\begin{bmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{bmatrix}$$

where $\theta = \cos^{-1}\left(\frac{3}{5}\right)$. This matrix rotates vectors by θ radians counterclockwise, which means that no real-valued vector will remain in the same direction after being multiplied by A. In the 2×2 case, the only way to get a real-valued eigenvalue out of a rotation matrix is if the matrix rotates by an integer multiple of π radians (180 degrees), which either corresponds to reflecting/negating the vector (like in $\begin{bmatrix} -1 & 0 \\ 0 & -1 \end{bmatrix}$) or returning back the same vector itself (like in $\begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$, the identity matrix, which can be thought of as a rotation by 2π).

The solutions to $p(\lambda) = (\lambda - 3)^2 + 16 = 0$ are the complex numbers $\lambda = 3 + 4i$ and $\lambda = 3 - 4i$, again where i is the imaginary unit, defined by $i^2 = -1$. The corresponding eigenvectors are complex too, and we won't worry about finding them. (If you're familiar with complex numbers, you might recognize these eigenvalues as $5e^{i\theta}$ and $5e^{-i\theta}$, where $\theta = \cos^{-1}\left(\frac{3}{5}\right)$. This comes from Euler's formula, $e^{i\theta} = \cos\theta + i\sin\theta$.)

Trace and Determinant. As a final example, consider the arbitrary 2×2 matrix

$$A = \begin{bmatrix} a & b \\ c & d \end{bmatrix}$$

Its characteristic polynomial is

$$p(\lambda) = \begin{vmatrix} a-\lambda & b \\ c & d-\lambda \end{vmatrix} = (a-\lambda)(d-\lambda) - bc = \lambda^2 - (a+d)\lambda + ad - bc$$

The coefficient on λ is $-(a+d)$, which is the negative of the **sum of the diagonal entries of** A. A term often used to refer to the sum of the diagonal entries of a matrix is its **trace**, $\text{trace}(A)$. And, the constant term of $p(\lambda)$ above is just $\det(A)$! For 2×2 matrices, this says that the characteristic polynomial is of the form

$$p(\lambda) = \lambda^2 - \text{trace}(A)\lambda + \det(A)$$

For example, consider $A = \begin{bmatrix} 3 & 1 \\ 7 & 9 \end{bmatrix}$. The fact above says that A's characteristic polynomial is

$$p(\lambda) = \lambda^2 - (3+9)\lambda + (3 \cdot 9 - 1 \cdot 7) = \lambda^2 - 12\lambda + 20$$

which, with enough practice, is a calculation you might be able to do in your head. But, this gives way to a more general fact, that is true for any $n \times n$ matrix.

Eigenvalues multiply to the determinant and add to the trace!

That is, the product of the eigenvalues of an $n \times n$ matrix is equal to its determinant, $\det(A)$, and the sum of the eigenvalues of a matrix is equal to the sum of the diagonal, $\text{trace}(A)$.
These are two conditions, and if I have a matrix that's larger than 2×2, they're not enough to determine the eigenvalues. But, they're helpful in indicating whether the eigenvalues you've found are correct or not.

Example 1: $A = \begin{bmatrix} 3 & 1 \\ 7 & 9 \end{bmatrix}$'s eigenvalues add up to $3 + 9 = 12$ and multiply to $\det(A) = 3 \cdot 9 - 1 \cdot 7 = 20$. This gives me a quick way of figuring out that A's eigenvalues are 10 and 2.

Example 2: $A = \begin{bmatrix} 0.5 & 0.5 & 0 \\ 0.4 & 0.3 & 0.3 \\ 0 & 0.5 & 0.5 \end{bmatrix}$ has an eigenvalue of 1, corresponding to the eigenvector $\begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix}$, since the sum of each row of A is 1. Let λ_2 and λ_3 be the other two eigenvalues. The trace fact tells me that

$$1 + \lambda_2 + \lambda_3 = 0.5 + 0.3 + 0.5 = 1.3 \implies \lambda_2 + \lambda_3 = 0.3$$

and the determinant fact tells me that

$$\begin{aligned} 1 \cdot \lambda_2 \cdot \lambda_3 &= \begin{vmatrix} 0.5 & 0.5 & 0 \\ 0.4 & 0.3 & 0.3 \\ 0 & 0.5 & 0.5 \end{vmatrix} \\ &= 0.5(0.3 \cdot 0.5 - 0.3 \cdot 0.5) - 0.5(0.4 \cdot 0.5 - 0.3 \cdot 0) + 0(0.4 \cdot 0.5 - 0.3 \cdot 0) \\ &= -0.5 \cdot 0.4 \cdot 0.5 \qquad\qquad = -0.1 \end{aligned}$$

In other words, the other two eigenvalues sum to 0.3 and multiply to -0.1. This sounds like a job for 0.5 and -0.2, which are indeed the other two eigenvalues of A (other than 1).

Example 3: $A = \begin{bmatrix} 1 & 2 \\ 2 & 4 \end{bmatrix}$ has a determinant of $1 \cdot 4 - 2 \cdot 2 = 0$, which means that its eigenvalues multiply to 0, which is another reminder that A has an eigenvalue of 0. The trace fact tells me that the other eigenvalue λ_2 must satisfy $1 + 4 = 0 + \lambda_2$, so $\lambda_2 = 5$.

Key Takeaways

Let's summarize the key ideas from both this section and the last.

Suppose A is an $n \times n$ matrix.

1. λ is an eigenvalue of A, corresponding to the eigenvector $\vec{v}$, if

$$A\vec{v} = \lambda\vec{v}$$

   The English interpretation of this is that $\vec{v}$ is an eigenvector of A if, when multiplied by A, $\vec{v}$ still points in the same direction, and is just stretched by a factor of λ.

2. An $n \times n$ matrix A has exactly n eigenvalues. Some of these may be complex, and some of these may be repeated, but all of them are roots to the **characteristic polynomial** of A,

$$p(\lambda) = \det(A - \lambda I)$$

3. A quick way to check if you've found the right eigenvalues is that

- The product of the eigenvalues of A is equal to $\det(A)$, i.e. $\lambda_1\lambda_2 \cdots \lambda_n = \det(A)$.
- The sum of the eigenvalues of A is equal to the trace of A, which is the sum of the diagonal elements of A.

4. 0 is an eigenvalue of A if and only if A is not invertible.
5. If λ is an eigenvalue of A with eigenvector $\vec{v}$, then λ^k is an eigenvalue of A^k with **the same** eigenvector $\vec{v}$.

9.3. Markov Chains and Adjacency Matrices

To extend what we've seen over the past two sections, I'll introduce a simplified version of the PageRank problem, first discussed in Chapter 9.1. This will allow me to highlight a key property of eigenvalues and eigenvectors: that they can be used to understand the long-term behavior of a system.

The Game

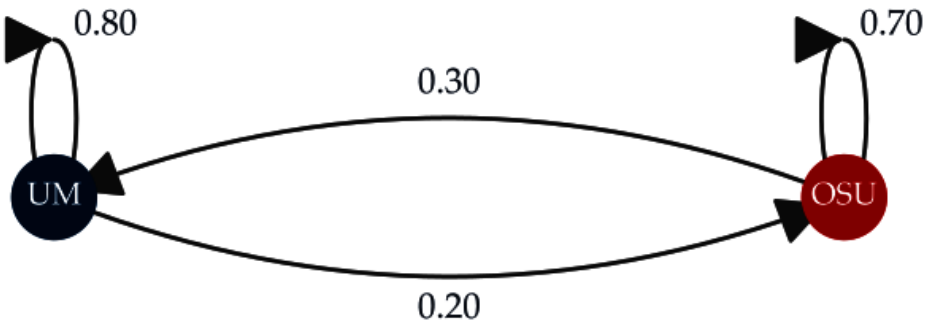
That sounds abstract – what is a system, and what is long-term behavior? Let's illustrate with an example. "The Game" refers to the annual football game between Michigan and Ohio State. Let's suppose that:

- If **Michigan** wins in year k, there's an 80% chance we win again in year $k+1$, and a 20% chance that Ohio State wins in year $k+1$.
- If **Ohio State** wins in year k, there's a 70% chance they win again in year $k+1$, and a 30% chance that Michigan wins in year $k+1$.

Question: In the long-run, what percentage of the time does Michigan win? Perhaps a more approachable way of phrasing this is, what is the (unconditional) probability that Michigan wins a particular game?

This system is an example of a **Markov chain**, in which the probabilities of transitioning between states don't change over time. In words, the **Markov property** says that once we know the current state, the next state depends only on where we are now, not on the full history of how we got there. So for this example, once we know who won this year, the probability of next year's winner doesn't depend on who won two years ago, three years ago, and so on. Here, a "state" is the outcome of the game in a particular year. The probabilities described above are fixed: the chance Michigan wins in year 100 given they won in year 99 is 80%, and the chance they win in year 55555 given they won in year 55554 is also 80%.

It's often useful to visualize Markov chains using a **state diagram**, which is a directed graph in which each node represents a state, and each edge represents a transition between states. Each edge is labeled with the probability of the transition.



Notice that the sum of the probabilities of the outgoing edges from each node (state) is 1. Intuitively, this says we must account for all possible transitions from a given state.

This Markov chain can be described by a 2×2 **adjacency matrix**, A:

$$A = \begin{bmatrix} p(\text{UM} \to \text{UM}) & p(\text{OSU} \to \text{UM}) \\ p(\text{UM} \to \text{OSU}) & p(\text{OSU} \to \text{OSU}) \end{bmatrix} = \begin{bmatrix} 0.8 & 0.3 \\ 0.2 & 0.7 \end{bmatrix}$$

Note that in an adjacency matrix, for our purposes:

- Each **column** describes the movement **from** a given node; the sums of the columns in an adjacency matrix are equal to 1.
- Each **row** describes the movement **into** a given node; the sums of the rows in an adjacency matrix don't necessarily add to 1.

That is, a_{ij} is the probability of transitioning from node j to node i.

Recall, our goal is to determine the long-run fraction of games that Michigan wins. How does the adjacency matrix A help us with this?

Simulating a Chain. Let's use it to perform a **simulation** of this system. Let's seed our simulation in the year $t = 0$, in which Michigan wins the first game, i.e.

$$\vec{x}_0 = \begin{bmatrix} 1 \\ 0 \end{bmatrix}$$

where $\vec{x}_k$ is a vector with two components, the first of which is the probability that Michigan wins in year k, and the second of which is the probability that Ohio State wins in year k.

To find the **distribution** of wins in year 1, we can multiply $\vec{x}_0$ by A:

$$\vec{x}_1 = A\vec{x}_0 = \begin{bmatrix} 0.8 & 0.3 \\ 0.2 & 0.7 \end{bmatrix} \begin{bmatrix} 1 \\ 0 \end{bmatrix} = \begin{bmatrix} 0.8 \\ 0.2 \end{bmatrix}$$

So, in year 1, we'd expect Michigan to win 80% of the time, and Ohio State to win 20% of the time. What about in year 2?

$$\vec{x}_2 = A\vec{x}_1 = \begin{bmatrix} 0.8 & 0.3 \\ 0.2 & 0.7 \end{bmatrix} \begin{bmatrix} 0.8 \\ 0.2 \end{bmatrix} = \begin{bmatrix} 0.8 \cdot 0.8 + 0.3 \cdot 0.2 \\ 0.2 \cdot 0.8 + 0.7 \cdot 0.2 \end{bmatrix} = \begin{bmatrix} 0.7 \\ 0.3 \end{bmatrix}$$

Or year 3?

$$\vec{x}_3 = A\vec{x}_2 = \begin{bmatrix} 0.8 & 0.3 \\ 0.2 & 0.7 \end{bmatrix} \begin{bmatrix} 0.7 \\ 0.3 \end{bmatrix} = \begin{bmatrix} 0.8 \cdot 0.7 + 0.3 \cdot 0.3 \\ 0.2 \cdot 0.7 + 0.7 \cdot 0.3 \end{bmatrix} = \begin{bmatrix} 0.65 \\ 0.35 \end{bmatrix}$$

In general, the distribution of wins in year t is given by

$$\vec{x}_k = A\vec{x}_{k-1} = A^2\vec{x}_{k-2} = \cdots = \boxed{A^k\vec{x}_0}$$

If you look at $\vec{x}_1$, $\vec{x}_2$, and $\vec{x}_3$ above, they don't seem to be exploding to some really large value. Despite repeatedly multiplying by A, the values seem to be stabilizing. Are these vectors converging? And, if so, to what?

Python can help verify our intuition.

```
def simulate_steps(A, x0, num_steps=15):
    x = x0
    for k in range(1, num_steps+1):
        # Note that np.linalg.matrix_power(A, k) is the same as A @ A @ ... @ A (k times)
        # A ** k raises each element of A to the kth power, which is not what we want
        x_k = np.linalg.matrix_power(A, k) @ x
        print(f'x_{k} = {x_k.flatten()}')

A = np.array([[0.8, 0.3],
              [0.2, 0.7]])

x0 = np.array([[1], [0]])

simulate_steps(A, x0)
```

```
x_1 = [0.8 0.2]
x_2 = [0.7 0.3]
x_3 = [0.65 0.35]
x_4 = [0.625 0.375]
x_5 = [0.6125 0.3875]
x_6 = [0.60625 0.39375]
x_7 = [0.603125 0.396875]
x_8 = [0.6015625 0.3984375]
x_9 = [0.60078125 0.39921875]
x_10 = [0.60039063 0.39960938]
x_11 = [0.60019531 0.39980469]
x_12 = [0.60009766 0.39990234]
x_13 = [0.60004883 0.39995117]
x_14 = [0.60002441 0.39997559]
x_15 = [0.60001221 0.39998779]
```

It seems that the sequence of $\vec{x}_k$'s is converging to $\begin{bmatrix} 0.6 \\ 0.4 \end{bmatrix}$, which would imply that in the long-run, **Michigan wins 60% of the time**.

An Eigenvalue Problem in Disguise

Could we have found the vector $\begin{bmatrix} 0.6 \\ 0.4 \end{bmatrix}$ without simulating the system? You sure bet. **This is the eigenvector of A corresponding to the eigenvalue 1!** Intuitively, since I'm searching for a long-run distribution, I'm really looking for a vector $\vec{x}$ such that

$$A\vec{x} = \vec{x}$$

meaning that advancing $\vec{x}$ one step in time won't change it. This is precisely the definition of an eigenvector with eigenvalue $\lambda = 1$. Eventually, we will prove that all adjacency matrices have an eigenvalue of 1, and that the corresponding eigenvector is the long-run distribution we're searching for.

But for now, let's verify that 1 is indeed an eigenvalue of A for this specific A by computing A's eigenvalues and eigenvectors ourselves.

$$A = \begin{bmatrix} 0.8 & 0.3 \\ 0.2 & 0.7 \end{bmatrix}$$

You might be able to spot the eigenvalues from the trace and determinant facts (they need to add to 1.5 and multiply to 0.5). If not, there's nothing wrong with explicitly writing out the characteristic polynomial.

$$\begin{aligned} p(\lambda) &= \begin{vmatrix} 0.8-\lambda & 0.3 \\ 0.2 & 0.7-\lambda \end{vmatrix} \\ &= (0.8-\lambda)(0.7-\lambda) - 0.2 \cdot 0.3 \\ &= \lambda^2 - 1.5\lambda + 0.5 \\ &= (\lambda - 1)(\lambda - 0.5) \end{aligned}$$

This tells us that A's eigenvalues are $\lambda_1 = 1$ and $\lambda_2 = 0.5$.

- For $\lambda_1 = 1$, we can find an eigenvector $\vec{v}_1 = \begin{bmatrix} a \\ b \end{bmatrix}$ by solving $A \begin{bmatrix} a \\ b \end{bmatrix} = \begin{bmatrix} a \\ b \end{bmatrix}$.

  $$A \begin{bmatrix} a \\ b \end{bmatrix} = \begin{bmatrix} a \\ b \end{bmatrix} \implies \begin{bmatrix} 0.8a + 0.3b \\ 0.2a + 0.7b \end{bmatrix} = \begin{bmatrix} a \\ b \end{bmatrix}$$

  Both components of the equation above give the relationship

  $$0.8a + 0.3b = a \implies 0.2a = 0.3b \implies b = \frac{2}{3}a$$

  So, one eigenvector is $\vec{v}_1 = \begin{bmatrix} 3 \\ 2 \end{bmatrix}$. **But**, since we'd like to interpret the components of $\vec{v}_1$ as a probability distribution, we should scale it so that its components sum to 1. Since $3 + 2 = 5$, this gives $\vec{v}_1 = \begin{bmatrix} 3/5 \\ 2/5 \end{bmatrix} = \begin{bmatrix} 0.6 \\ 0.4 \end{bmatrix}$, which is precisely what our simulated distributions converged to!

- For $\lambda_2 = 0.5$, notice that

  $$A \begin{bmatrix} 1 \\ -1 \end{bmatrix} = \begin{bmatrix} 0.8 & 0.3 \\ 0.2 & 0.7 \end{bmatrix} \begin{bmatrix} 1 \\ -1 \end{bmatrix} = \begin{bmatrix} 0.8 - 0.3 \\ 0.2 - 0.7 \end{bmatrix} = \begin{bmatrix} 0.5 \\ -0.5 \end{bmatrix} = 0.5 \begin{bmatrix} 1 \\ -1 \end{bmatrix}$$

  So, $\vec{v}_2 = \begin{bmatrix} 1 \\ -1 \end{bmatrix}$ is an eigenvector of A corresponding to the eigenvalue $\lambda_2 = 0.5$. There's no way to scale $\vec{v}_2$ so that its components are all positive and sum to 1, but there's no need to, since we're only interested in interpreting $\lambda_1 = 1$'s eigenvector as a probability distribution.

I want to revisit our Python simulation from above. Using the initial vector $\vec{x}_0 = \begin{bmatrix} 1 \\ 0 \end{bmatrix}$, the state vectors $\vec{x}_k$ converged to $\vec{v}_1 = \begin{bmatrix} 0.6 \\ 0.4 \end{bmatrix}$.

```
simulate_steps(A, x0=np.array([[1], [0]]))
```

```
x_1 = [0.8 0.2]
x_2 = [0.7 0.3]
x_3 = [0.65 0.35]
x_4 = [0.625 0.375]
x_5 = [0.6125 0.3875]
x_6 = [0.60625 0.39375]
x_7 = [0.603125 0.396875]
x_8 = [0.6015625 0.3984375]
x_9 = [0.60078125 0.39921875]
x_10 = [0.60039063 0.39960938]
x_11 = [0.60019531 0.39980469]
x_12 = [0.60009766 0.39990234]
x_13 = [0.60004883 0.39995117]
x_14 = [0.60002441 0.39997559]
x_15 = [0.60001221 0.39998779]
```

But, there was nothing special about our choice of $\vec{x}_0$. Had we started with, say, $\vec{x}_0 = \begin{bmatrix} 0.5 \\ 0.5 \end{bmatrix}$ or even $\vec{x}_0 = \begin{bmatrix} 0 \\ 1 \end{bmatrix}$, the state vectors would have still converged to the eigenvector $\vec{v}_1 = \begin{bmatrix} 0.6 \\ 0.4 \end{bmatrix}$.

```
simulate_steps(A, x0=np.array([[0.5], [0.5]]))
```

```
x_1 = [0.55 0.45]
x_2 = [0.575 0.425]
x_3 = [0.5875 0.4125]
x_4 = [0.59375 0.40625]
x_5 = [0.596875 0.403125]
x_6 = [0.5984375 0.4015625]
x_7 = [0.59921875 0.40078125]
x_8 = [0.59960938 0.40039063]
x_9 = [0.59980469 0.40019531]
x_10 = [0.59990234 0.40009766]
x_11 = [0.59995117 0.40004883]
x_12 = [0.59997559 0.40002441]
x_13 = [0.59998779 0.40001221]
x_14 = [0.5999939 0.4000061]
x_15 = [0.59999695 0.40000305]
```

```
simulate_steps(A, x0=np.array([[0], [1]]))
```

```
x_1 = [0.3 0.7]
x_2 = [0.45 0.55]
x_3 = [0.525 0.475]
x_4 = [0.5625 0.4375]
x_5 = [0.58125 0.41875]
x_6 = [0.590625 0.409375]
x_7 = [0.5953125 0.4046875]
x_8 = [0.59765625 0.40234375]
x_9 = [0.59882813 0.40117188]
```

```
x_10 = [0.59941406 0.40058594]
x_11 = [0.59970703 0.40029297]
x_12 = [0.59985352 0.40014648]
x_13 = [0.59992676 0.40007324]
x_14 = [0.59996338 0.40003662]
x_15 = [0.59998169 0.40001831]
```

What if we start with a $\vec{x}_0$ that isn't a probability distribution, like $\vec{x}_0 = \begin{bmatrix} 50 \\ -15 \end{bmatrix}$?

```
simulate_steps(A, x0=np.array([[50], [ -15]]))
```

```
x_1 = [35.5 -0.5]
x_2 = [28.25  6.75]
x_3 = [24.625 10.375]
x_4 = [22.8125 12.1875]
x_5 = [21.90625 13.09375]
x_6 = [21.453125 13.546875]
x_7 = [21.2265625 13.7734375]
x_8 = [21.11328125 13.88671875]
x_9 = [21.05664063 13.94335938]
x_10 = [21.02832031 13.97167969]
x_11 = [21.01416016 13.98583984]
x_12 = [21.00708008 13.99291992]
x_13 = [21.00354004 13.99645996]
x_14 = [21.00177002 13.99822998]
x_15 = [21.00088501 13.99911499]
```

The state vector is converging on $\begin{bmatrix} 21 \\ 14 \end{bmatrix}$, which is still on the line described by $\vec{v}_1 = \begin{bmatrix} 0.6 \\ 0.4 \end{bmatrix}$. So, it seems that no matter where we start, the state vectors will converge to the eigenvector $\vec{v}_1 = \begin{bmatrix} 0.6 \\ 0.4 \end{bmatrix}$ or some scalar multiple of it.

The Dominant Eigenvalue

Why is it the case that $\vec{x}_k = A^k\vec{x}_0$ converges to an eigenvector for $\lambda_1 = 1$, and not to an eigenvector for the other eigenvalue of $\lambda_2 = 0.5$?

The answer is that the **dominant eigenvalue**, i.e. the one with the largest magnitude, is the one that determines the long-run behavior of the system. Let's see why this is the case.

First, recall that A's eigenvalues are $\lambda_1 = 1$ and $\lambda_2 = 0.5$, and corresponding eigenvectors are $\vec{v}_1 = \begin{bmatrix} 0.6 \\ 0.4 \end{bmatrix}$ and $\vec{v}_2 = \begin{bmatrix} 1 \\ -1 \end{bmatrix}$. Note that these two eigenvectors are linearly independent, so any other vector $\vec{x} \in \mathbb{R}^2$ can be written as a linear combination of $\vec{v}_1$ and $\vec{v}_2$. **The fact that these eigenvectors are linearly independent is key to the argument below.**

So, suppose we start with some vector $\vec{x} = c_1\vec{v}_1 + c_2\vec{v}_2$. Then, what we're really interested in is the behavior of the vectors $A\vec{x}$, $A^2\vec{x}$, $A^3\vec{x}$, and so on. What is $A\vec{x}$?

$$A\vec{x} = A(c_1\vec{v}_1 + c_2\vec{v}_2) = c_1A\vec{v}_1 + c_2A\vec{v}_2 = c_1\lambda_1\vec{v}_1 + c_2\lambda_2\vec{v}_2$$

Above, I used the fact that $A\vec{v}_1 = \lambda_1\vec{v}_1$ and $A\vec{v}_2 = \lambda_2\vec{v}_2$ by the definition of eigenvectors.
Recall from the original Chapter 9.1 that if λ is an eigenvalue of A with eigenvector $\vec{v}$, then $A^k\vec{v} = \lambda^k\vec{v}$ also. So,

$$A^2\vec{x} = A(c_1\lambda_1\vec{v}_1 + c_2\lambda_2\vec{v}_2) = c_1\lambda_1^2\vec{v}_1 + c_2\lambda_2^2\vec{v}_2$$

and more generally, if k is a positive integer,

$$A^k\vec{x} = c_1\lambda_1^k\vec{v}_1 + c_2\lambda_2^k\vec{v}_2$$

So, $A^k\vec{x}$ is a linear combination of $\vec{v}_1$ and $\vec{v}_2$ also, with weights $c_1\lambda_1^k$ and $c_2\lambda_2^k$. Here, $\lambda_1 = 1$ and $\lambda_2 = 0.5$, so $\lambda_1^k = 1$ and $\lambda_2^k = 0.5^k$.

$$A^k\vec{x} = c_1(1)^k\begin{bmatrix}0.6\\0.4\end{bmatrix} + c_2(0.5)^k\begin{bmatrix}1\\-1\end{bmatrix} = c_1\begin{bmatrix}0.6\\0.4\end{bmatrix} + c_2(0.5)^k\begin{bmatrix}1\\-1\end{bmatrix}$$

But, as k increases, 0.5^k approaches 0, while $1^k = 1$. So,

$$A^k\vec{x} = c_1\begin{bmatrix}0.6\\0.4\end{bmatrix} + c_2(0.5)^k\begin{bmatrix}1\\-1\end{bmatrix} \approx c_1\begin{bmatrix}0.6\\0.4\end{bmatrix} \text{ as } k \to \infty$$

This explains why the state vectors $\vec{x}_k$ converge to (some scalar multiple of) $\vec{v}_1 = \begin{bmatrix}0.6\\0.4\end{bmatrix}$, no matter where we start!

Why do Adjacency Matrices Have an Eigenvalue of 1?. Let's prove that all adjacency matrices have an eigenvalue of 1. Recall that an adjacency matrix is defined such that:

- Each **column** describes the movement **from** a given node; the sums of the columns in an adjacency matrix are equal to 1.
- Each **row** describes the movement **into** a given node; the sums of the rows in an adjacency matrix don't necessarily add to 1.

Here's the key fact: **for any square matrix A, the eigenvalues of A are the same as the eigenvalues of A^T.** The eigenvectors themselves are usually different, but the eigenvalues are the same. This is because A and A^T both have the same characteristic polynomial. Remember from Chapter 6.2 that $\det(A) = \det(A^T)$. So, A's characteristic polynomial is

$$p(\lambda) = \det(A - \lambda I)$$

but this is the same as $\det((A - \lambda I)^T) = \det(A^T - \lambda I)$, which is the characteristic polynomial of A^T. So, A and A^T have the same characteristic polynomial, and thus the same eigenvalues.

Back to the original question: why does an adjacency matrix have an eigenvalue of 1? Remember, the **columns** of an adjacency matrix, which describe the movement **from** a given node, sum to 1. If A is an adjacency matrix,

then

$$A^T \begin{bmatrix} 1 \\ 1 \\ \vdots \\ 1 \end{bmatrix} = \begin{bmatrix} \text{sum of column 1 of } A \\ \text{sum of column 2 of } A \\ \vdots \\ \text{sum of column } n \text{ of } A \end{bmatrix} = \begin{bmatrix} 1 \\ 1 \\ \vdots \\ 1 \end{bmatrix}$$

which shows that A^T has an eigenvector (the all ones vector) with an eigenvalue of 1.

For instance, if $A = \begin{bmatrix} 0.8 & 0.3 \\ 0.2 & 0.7 \end{bmatrix}$, then $A^T = \begin{bmatrix} 0.8 & 0.2 \\ 0.3 & 0.7 \end{bmatrix}$, and $A^T \begin{bmatrix} 1 \\ 1 \end{bmatrix} = \begin{bmatrix} 1 \\ 1 \end{bmatrix}$, which is an eigenvector of A^T with eigenvalue 1.

But, since A and A^T have the same eigenvalues, A must have an eigenvalue of 1.

The followup question is why it's guaranteed that the **largest** eigenvalue of an adjacency matrix is 1, because this seems to be crucial in guaranteeing that the state vectors $\vec{x}_k$ converge to the eigenvector corresponding to the eigenvalue 1. Proving this fact is beyond the scope of our course, but it follows from the **Perron-Frobenius theorem** if you'd like to read more.

Example: Non-Adjacency Matrices. You might wonder, if A's largest eigenvalue is greater than 1, what happens to the vector $A^k\vec{x}$ as k increases? (I won't call $A^k\vec{x}$ a state vector, since it's not a probability distribution.)

Let's consider $A = \begin{bmatrix} 3 & 1 \\ 2 & 4 \end{bmatrix}$, which was used as an example in Chapter 9.2. A has eigenvalues $\lambda_1 = 2$ and $\lambda_2 = 5$, and corresponding eigenvectors $\vec{v}_1 = \begin{bmatrix} 1 \\ -1 \end{bmatrix}$ and $\vec{v}_2 = \begin{bmatrix} 1 \\ 2 \end{bmatrix}$. These eigenvectors are also linearly independent, so any $\vec{x} \in \mathbb{R}^2$ can be written as a linear combination of $\vec{v}_1$ and $\vec{v}_2$. Then, following the logic from the previous example,

$$A^k\vec{x} = c_1(2)^k \begin{bmatrix} 1 \\ -1 \end{bmatrix} + c_2(5)^k \begin{bmatrix} 1 \\ 2 \end{bmatrix}$$

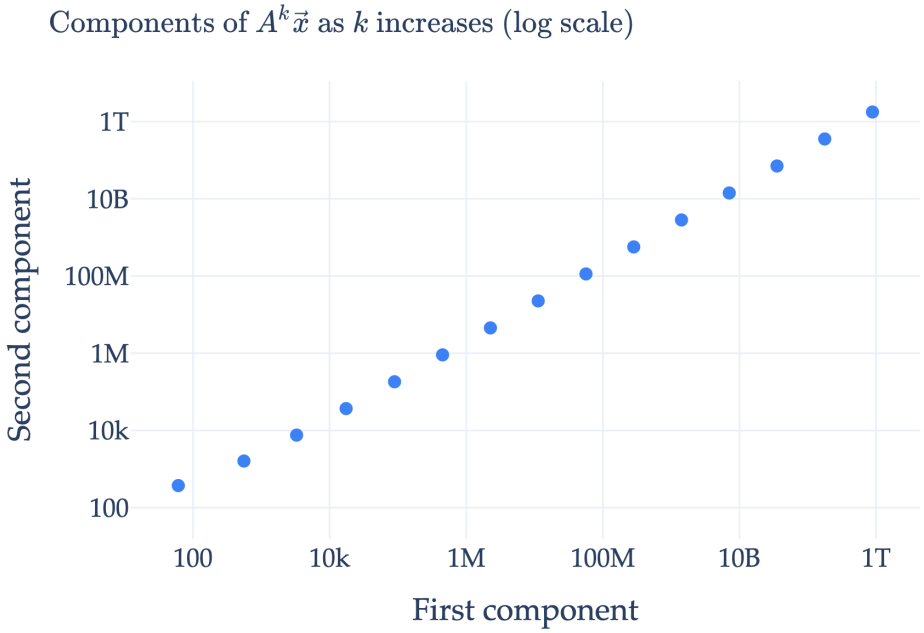
The issue is that both 5^k and 2^k grow without bound as k increases. So, $A^k\vec{x}$ will not converge. But, the **direction** of $A^k\vec{x}$ is still predictable. Let me start with with $\vec{x} = \begin{bmatrix} -13 \\ 100 \end{bmatrix}$ (nothing special about these values) and evaluate $A^k\vec{x}$ for $k = 1, 2, \ldots, 15$.

```
simulate_steps(
    A = np.array([[3, 1],
                  [2, 4]]),
    x0 = np.array([[ -13], [100]])
)
```

```
x_1 = [ 61 374]
x_2 = [ 557 1618]
x_3 = [3289 7586]
x_4 = [17453 36922]
x_5 = [ 89281 182594]
x_6 = [450437 908938]
x_7 = [2260249 4536626]
x_8 = [11317373 22667002]
x_9 = [ 56619121 113302754]
x_10 = [283160117 566449258]
```

```
x_11 = [1415929609 2832117266]
x_12 = [ 7079906093 14160328282]
x_13 = [35400046561 70801125314]
x_14 = [177001264997 354004594378]
x_15 = [ 885008389369 1770020907506]
```

The numbers are indeed getting bigger and bigger, but a relationship is emerging between the components of $A^k\vec{x}$. Let's plot the coordinates of the vector $A^k\vec{x}$ for $k = 1, 2, \ldots, 15$.



I had to log-scale the axes, since the components of $A^k\vec{x}$ are growing exponentially. But, you should notice that the vectors $A^k\vec{x}$ all lie on the same line – the line described by the **eigenvector for the dominant eigenvalue**, $\vec{v}_2 = \begin{bmatrix}1\\2\end{bmatrix}$! Hover over any point above to see the coordinates of the vector $A^k\vec{x}$ for a particular k, and you'll see the y-coordinate tends to be roughly double the x-coordinate.

So, when the dominant eigenvalue (that is, the eigenvalue with the largest magnitude) is greater than 1, the vector $A^k\vec{x}$ doesn't converge, but its direction converges to the direction of the eigenvector corresponding to the dominant eigenvalue. Through the lens of our most recent equation for $A^k\vec{x}$,

$$A^k\vec{x} = c_1(2)^k\begin{bmatrix}1\\-1\end{bmatrix} + c_2(5)^k\begin{bmatrix}1\\2\end{bmatrix}$$

Notice that if we divide both sides by 5^k, we see

$$\frac{A^k\vec{x}}{5^k} = c_1\left(\frac{2}{5}\right)^k\begin{bmatrix}1\\-1\end{bmatrix} + c_2\begin{bmatrix}1\\2\end{bmatrix}$$

And as k increases, the term $c_1\left(\frac{2}{5}\right)^k$ approaches 0. So, the direction of $\frac{A^k\vec{x}}{5^k}$ approaches the direction of $\begin{bmatrix}1\\2\end{bmatrix}$, but $A^k\vec{x}$ and $\frac{A^k\vec{x}}{5^k}$ have the same direction (but different magnitudes).

(I view our approach above as being similar to an approach used in limit problems in Calculus 1. For example, to find $\lim_{x\to\infty}\frac{x^2}{3x^2-4x}$, you divide the numerator and denominator by the highest power to get $\lim_{x\to\infty}\frac{1}{3-4/x} = 1/3$. This is similar to what we did above, where we divided by the highest power of 5^k to make the dominant

behavior clear.)

You can view the process of computing $A^k\vec{x}$ for larger and larger k until the direction stabilizes as being a technique for **approximating** the eigenvector corresponding to the dominant eigenvalue. This is called the **power method**.

Here's the cliffhanger: our analysis of the behavior of $A^k\vec{x}$ as k increases has depended on the eigenvectors of A (that is, $\vec{v}_1$ and $\vec{v}_2$) being linearly independent. What if the eigenvectors of a particular matrix aren't linearly independent? Time for Chapter 9.4 to shine.

9.4. Multiplicities and Diagonalization

In Chapter 9.1 and Chapter 9.3, we saw that eigenvalues and eigenvectors allow us to better understand the behavior of the linear transformation from $\mathbb{R}^n$ to $\mathbb{R}^n$ given by an $n \times n$ matrix A.

When understanding matrix powers and adjacency matrices, we often decomposed some arbitrary vector $\vec{x}$ as a linear combination of eigenvectors of A. But sometimes, the eigenvectors of A don't span all of $\mathbb{R}^n$, meaning we can't write every other vector as a linear combination of eigenvectors of A. When does this happen, and why do we really care?

The Eigenvalue Decomposition

Recall that if A is an $n \times n$ matrix, then A has n eigenvalues, it's just that some of the eigenvalues may be complex (leading to complex-valued eigenvectors), and some eigenvalues may be repeated. These n eigenvalues are all solutions to the characteristic polynomial of A,

$$p(\lambda) = \det(A - \lambda I)$$

The corresponding eigenvectors are the solutions to the system of equations $(A - \lambda I)\vec{v} = \vec{0}$.

Suppose A has n linearly independent eigenvectors $\vec{v}_1, \vec{v}_2, \cdots, \vec{v}_n$, with eigenvalues $\lambda_1, \lambda_2, \cdots, \lambda_n$. What I'm about to propose next will seem a little arbitrary, but bear with me – you'll see the power of this idea shortly. What happens if we multiply A by a **matrix**, say V, whose columns are the eigenvectors of A?

$$\begin{aligned} AV &= A\begin{bmatrix} | & | & & | \\ \vec{v}_1 & \vec{v}_2 & \cdots & \vec{v}_n \\ | & | & & | \end{bmatrix} \\ &= \begin{bmatrix} | & | & & | \\ A\vec{v}_1 & A\vec{v}_2 & \cdots & A\vec{v}_n \\ | & | & & | \end{bmatrix} \\ &= \begin{bmatrix} | & | & & | \\ \lambda_1\vec{v}_1 & \lambda_2\vec{v}_2 & \cdots & \lambda_n\vec{v}_n \\ | & | & & | \end{bmatrix} \end{aligned}$$

AV is a matrix whose columns are eigenvectors of A, each scaled by the corresponding eigenvalue! Is there another way to write the last line above?

$$\begin{aligned} AV &= \begin{bmatrix} | & | & & | \\ \lambda_1\vec{v}_1 & \lambda_2\vec{v}_2 & \cdots & \lambda_n\vec{v}_n \\ | & | & & | \end{bmatrix} \\ &= \begin{bmatrix} | & | & & | \\ \vec{v}_1 & \vec{v}_2 & \cdots & \vec{v}_n \\ | & | & & | \end{bmatrix} \begin{bmatrix} \lambda_1 & 0 & \cdots & 0 \\ 0 & \lambda_2 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & \lambda_n \end{bmatrix} \\ &= V\Lambda \end{aligned}$$

Here, $*$ (the capitalized Greek letter for λ) is a diagonal matrix of eigenvalues **in the same order as the eigenvectors in** V.

$$AV = V\Lambda$$

We can take this a step further. **If** V is invertible – which it is here, since we assumed A has n linearly independent eigenvectors – then we can multiply both sides of the equation above by V^{-1} on the right to get

$$A = V\Lambda V^{-1}$$

The existence of this decomposition is contingent on V being invertible, **which happens when A has n linearly independent eigenvectors**. If A doesn't have "enough" eigenvectors, then V wouldn't be invertible, and we wouldn't be able to decompose A in this way.

Diagonalizable Matrices.

Definition: Eigenvalue Decomposition

The **eigenvalue decomposition** (or eigenvector decomposition) of a matrix A is a decomposition of the form

$$A = V\Lambda V^{-1}$$

where V is a matrix containing the eigenvectors of A as columns, and $*$ is a diagonal matrix of eigenvalues in the same order.

Definition: Diagonalizable Matrix

A matrix A is **diagonalizable** if it has an eigenvalue decomposition

$$A = V\Lambda V^{-1}$$

using the same definitions of V and $*$ above. A is diagonalizable if and only if it has n linearly independent eigenvectors (which allows for V to be invertible). Otherwise, A is not diagonalizable, and is sometimes called **defective**.
Often, A is defined to be diagonalizable if and only if it can be written in the form $A = PDP^{-1}$ where P is *some* invertible matrix and D is *some* diagonal matrix. This is a general definition of diagonalizability. But, **the only matrices that can be written in $A = PDP^{-1}$ have eigenvalue decompositions and vice versa**. The eigenvalue decomposition tells you how to actually diagonalize A.

A First Example. Let's start with a familiar example, $A = \begin{bmatrix} 1 & 2 \\ 2 & 1 \end{bmatrix}$, which was the first example we saw in Chapter 9.1. A has eigenvalues $\lambda_1 = 3$ and $\lambda_2 = -1$, and corresponding eigenvectors $\vec{v}_1 = \begin{bmatrix} 1 \\ 1 \end{bmatrix}$ and $\vec{v}_2 = \begin{bmatrix} 1 \\ -1 \end{bmatrix}$. These eigenvectors are linearly independent, so A is diagonalizable, and we can write

$$V = \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix}, \qquad \Lambda = \begin{bmatrix} 3 & 0 \\ 0 & -1 \end{bmatrix}$$

which tells us that

$$V\Lambda V^{-1} = \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix} \begin{bmatrix} 3 & 0 \\ 0 & -1 \end{bmatrix} \begin{bmatrix} 1/2 & 1/2 \\ 1/2 & -1/2 \end{bmatrix} = \begin{bmatrix} 1 & 2 \\ 2 & 1 \end{bmatrix} = A$$

Let's check with `numpy`.

```
V = np.array([[1, 1],
              [1, -1]])

Lambda = np.diag([3, -1]) # New tool.
V_inv = np.linalg.inv(V)
V @ Lambda @ V_inv # The same as A!

array([[1., 2.],
       [2., 1.]])
```

Is this decomposition unique? No, because we could have chosen a different set of eigenvectors, or wrote them in a different order (in which case $*$ would be different). We'll keep the eigenvectors in the same order, but instead let's consider

$$V = \begin{bmatrix} 2 & -5 \\ 2 & 5 \end{bmatrix}$$

which is also a valid eigenvector matrix for A. (Remember that any scalar multiple of an eigenvector is still an eigenvector!)

It is **still** true that $A = V\Lambda V^{-1}$. This may seem a little unbelievable (I wasn't convinced at first), but remember that V^{-1} "undoes" any changes in scaling we introduce to V.

$$V\Lambda V^{-1} = \begin{bmatrix} 2 & -5 \\ 2 & 5 \end{bmatrix} \begin{bmatrix} 3 & 0 \\ 0 & -1 \end{bmatrix} \begin{bmatrix} 1/4 & 1/4 \\ -1/10 & 1/10 \end{bmatrix} = \begin{bmatrix} 1 & 2 \\ 2 & 1 \end{bmatrix} = A$$

```
V = np.array([[2, -5],
              [2, 5]])

Lambda = np.diag([3, -1])
V_inv = np.linalg.inv(V)
V @ Lambda @ V_inv # Same as above!

array([[1., 2.],
       [2., 1.]])
```

Activity 1

Activity 1

Suppose $\vec{v}_1$, $\vec{v}_2$, and $\vec{v}_3$ are eigenvectors of A with eigenvalues λ_1, λ_2, and λ_3, respectively, and all three of λ_1, λ_2, and λ_3 are distinct.

1. Prove that $\vec{v}_1$ and $\vec{v}_2$ are linearly independent.
2. Prove that $\vec{v}_1$, $\vec{v}_2$, and $\vec{v}_3$ are linearly independent.

Solution

Part 1 We'll start by assuming that $A\vec{v}_1 = \lambda_1\vec{v}_1$, $A\vec{v}_2 = \lambda_2\vec{v}_2$, and $\lambda_1 \neq \lambda_2$.
If $\vec{v}_1$ and $\vec{v}_2$ are linearly independent, it means that the only solution to the equation

$$c_1\vec{v}_1 + c_2\vec{v}_2 = \vec{0}$$

is $c_1 = c_2 = 0$. (Otherwise, there are non-zero scalars c_1 and c_2 that make the equation true.)
Let's start with $c_1\vec{v}_1 + c_2\vec{v}_2 = \vec{0}$ and do two things to it:

1. Multiply both sides (on the left) by A, yielding

$$\begin{aligned} A(c_1\vec{v}_1 + c_2\vec{v}_2) &= A\vec{0} \\ c_1A\vec{v}_1 + c_2A\vec{v}_2 &= \vec{0} \\ c_1\lambda_1\vec{v}_1 + c_2\lambda_2\vec{v}_2 &= \vec{0} \end{aligned}$$

2. Multiply both sides by λ_1, yielding

$$\lambda_1(c_1\vec{v}_1 + c_2\vec{v}_2) = \lambda_1\vec{0} \implies c_1\lambda_1\vec{v}_1 + c_2\lambda_1\vec{v}_2 = \vec{0}$$

Now, we have two similar equations:

$$\begin{aligned} c_1\lambda_1\vec{v}_1 + c_2\lambda_2\vec{v}_2 &= \vec{0} \\ c_1\lambda_1\vec{v}_1 + c_2\lambda_1\vec{v}_2 &= \vec{0} \end{aligned}$$

Subtracting the second equation from the first, we get

$$c_2(\lambda_2 - \lambda_1)\vec{v}_2 = \vec{0}$$

Since λ_1 and λ_2 are distinct, $\lambda_2 - \lambda_1 \neq 0$, so we must have $c_2 = 0$.
But if $c_2 = 0$, then $c_1 = 0$ also! We can see this by substituting $c_2 = 0$ into the first equation we started working with.

$$c_1\vec{v}_1 + c_2\vec{v}_2 = \vec{0} \implies c_1\vec{v}_1 = \vec{0} \implies c_1 = 0$$

So, the only solution to $c_1\vec{v}_1 + c_2\vec{v}_2 = \vec{0}$ is $c_1 = c_2 = 0$. This means that $\vec{v}_1$ and $\vec{v}_2$ are linearly independent.

Part 2
Now we need to show that if $\vec{v}_1$, $\vec{v}_2$, and $\vec{v}_3$ are eigenvectors of A with distinct eigenvalues (λ_1, λ_2, and λ_3, respectively), then they are linearly independent. We need to show that the only solution to

$$c_1\vec{v}_1 + c_2\vec{v}_2 + c_3\vec{v}_3 = \vec{0}$$

is $c_1 = c_2 = c_3 = 0$. The general idea is the same as in Part 1: we need to use the fact that λ_1, λ_2, and λ_3 are different.
Instead of multiplying $c_1\vec{v}_1 + c_2\vec{v}_2 + c_3\vec{v}_3 = \vec{0}$ by A and then λ_1 and then subtracting the two equations, here's an idea: why don't we multiply by $A - \lambda_1 I$ and see what happens?
Doing so gives

Key Applications

Rightfully, you might be asking what the point of this is. There are (at least) two main uses of the eigenvalue decomposition.

Application 1: Matrix Powers. The first is that it makes it easy to compute powers of A, which we know is a useful concept in understanding the long-run behavior of a Markov chain.

Suppose $A = V\Lambda V^{-1}$. Then,

$$A^2 = (V\Lambda V^{-1})(V\Lambda V^{-1}) = V\Lambda(V^{-1}V)\Lambda V^{-1} = V\Lambda^2 V^{-1}$$

What does this say about A^3?

$$A^3 = (V\Lambda V^{-1})(V\Lambda V^{-1})(V\Lambda V^{-1}) = V\Lambda(V^{-1}V)\Lambda(V^{-1}V)\Lambda V^{-1} = V\Lambda^3 V^{-1}$$

In general, if k is a positive integer,

$$\boxed{A^k = V\Lambda^k V^{-1}}$$

So, to compute A^k, we don't need to multiply k matrices together (which would be a computational nightmare for large k). Instead, all we need to do is compute $V\Lambda^k V^{-1}$. And remember, $*$ is a diagonal matrix, so computing Λ^k is easy: we just raise the diagonal entries to the power in question.

For example, $A^{10} = \begin{bmatrix} 1 & 2 \\ 2 & 1 \end{bmatrix}^{10}$ is

$$\begin{aligned} A^{10} &= V\Lambda^{10}V^{-1} \\ &= \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix} \begin{bmatrix} 3^{10} & 0 \\ 0 & (-1)^{10} \end{bmatrix} \begin{bmatrix} 1/2 & 1/2 \\ 1/2 & -1/2 \end{bmatrix} \\ &= \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix} \begin{bmatrix} 3^{10}/2 & 3^{10}/2 \\ 1/2 & -1/2 \end{bmatrix} \\ &= \begin{bmatrix} (3^{10}+1)/2 & (3^{10}-1)/2 \\ (3^{10}-1)/2 & (3^{10}+1)/2 \end{bmatrix} \end{aligned}$$

Pretty neat!

```
np.linalg.matrix_power(A, 10)
```

```
array([[29525., 29524.],
       [29524., 29525.]])
```

Application 2: Understanding Linear Transformations. Remember that if A is an $n \times n$ matrix, then $f(\vec{x}) = A\vec{x}$ is a linear transformation from $\mathbb{R}^n$ to $\mathbb{R}^n$. But, if $A = V\Lambda V^{-1}$, then

$$f(\vec{x}) = A\vec{x} = V\Lambda V^{-1}\vec{x}$$

This allows us to understand the effect of f on $\vec{x}$ in three stages. Remember that V's columns are the eigenvectors of A, so the act of multiplying a vector $\vec{y}$ by V is equivalent to taking a linear combination of V's columns (A's eigenvectors) using the weights in $\vec{y}$.

$$V\vec{y} = \begin{bmatrix} | & | & \cdots & | \\ \vec{v}_1 & \vec{v}_2 & \cdots & \vec{v}_n \\ | & | & \cdots & | \end{bmatrix} \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_n \end{bmatrix} = y_1\vec{v}_1 + y_2\vec{v}_2 + \cdots + y_n\vec{v}_n$$

For example, $V \begin{bmatrix} 3 \\ -4 \end{bmatrix}$ says take 3 of the first eigenvector, and -4 of the second eigenvector, and add them together to get a new vector. If $V \begin{bmatrix} 3 \\ -4 \end{bmatrix} = \begin{bmatrix} 1 \\ 2 \end{bmatrix}$, this says that taking 3 of the first eigenvector and -4 of the second eigenvector is the same as taking 1 of the first standard basis vector, $\begin{bmatrix} 1 \\ 0 \end{bmatrix}$ and 2 of the second standard basis vector, $\begin{bmatrix} 0 \\ 1 \end{bmatrix}$.

Intuitively, think of V as a matrix that takes in "amounts of each eigenvector" and outputs "amounts of each standard basis vector", where the standard basis vectors are the columns of I.

So, if

$$V : \text{eigenvector amounts} \to \text{standard basis vector amounts}$$

then V^{-1} does the opposite, and maps

$$V^{-1} : \text{standard basis vector weights} \to \text{eigenvector amounts}$$

i.e. multiplying V^{-1} by $\vec{x}$ **expresses $\vec{x}$ as a linear combination of the eigenvectors of A.** If $\vec{z}$ is the output of $V^{-1}\vec{x}$, then $\vec{x} = V\vec{z}$, meaning $\vec{z}$ contains the amounts of each eigenvector needed to produce $\vec{x}$.

(This is non-standard notation, since V is not a function and eigenvector amounts and standard basis vector amounts are not sets but concepts, but I hope this helps convey the role of V and V^{-1} in this context.)

Taking a step back, recall

$$f(\vec{x}) = A\vec{x} = V\Lambda V^{-1}\vec{x}$$

What this is saying is f does three things:

1. First, it takes $\vec{x}$ and expresses it as a linear combination of the eigenvectors of A, which is $V^{-1}\vec{x}$. (This contains the "amounts of each eigenvector" needed to produce $\vec{x}$.)
2. Then, it takes that linear combination $V^{-1}\vec{x}$ and scales each eigenvector by its corresponding eigenvalue, i.e. it **scales** or **stretches** each eigenvector by a different amount. **Remember that diagonal matrices only scale, they don't do anything else!**
3. Finally, it takes the resulting scaled vector $\Lambda V^{-1}\vec{x}$ and expresses it as a linear combination of the standard basis vectors, i.e. it combines the correct amounts of the stretched eigenvectors to get the final result.

$$\underbrace{\text{express in eigenvector basis}}_{\text{multiply by } V^{-1}} \to \underbrace{\text{scale eigenvectors}}_{\text{multiply by } \Lambda} \to \underbrace{\text{express in standard basis}}_{\text{multiply by } V}$$

This is better understood visually, as you'll see in Chapter 9.5: Symmetric Matrices and the Spectral Theorem.

Examples

Example: Non-Diagonalizable Matrices. Most matrices we've seen in Chapter 9.1 and here so far in this section have been diagonalizable. I've tried to shield you from the messy world of non-diagonalizable matrices until now, but we're ready to dive deeper.

Consider $A = \begin{bmatrix} 1 & 1 \\ 0 & 1 \end{bmatrix}$. The characteristic polynomial of A is

$$p(\lambda) = \det(A - \lambda I) = \det \begin{bmatrix} 1-\lambda & 1 \\ 0 & 1-\lambda \end{bmatrix} = (1-\lambda)^2$$

which has a double root of $\lambda = 1$. So, A has a single eigenvalue $\lambda = 1$. What are all of A's eigenvectors? They're solutions to $A\vec{v} = 1\vec{v}$, i.e. $(A - I)\vec{v} = \vec{0}$. Let's look at the null space of $A - I$:

$$A - I = \begin{bmatrix} 1-1 & 1 \\ 0 & 1-1 \end{bmatrix} = \begin{bmatrix} 0 & 1 \\ 0 & 0 \end{bmatrix}$$

The null space of $A - I$ is spanned by the single vector $\begin{bmatrix} 1 \\ 0 \end{bmatrix}$. So, A has a single line of eigenvectors, spanned by $\vec{v}_1 = \begin{bmatrix} 1 \\ 0 \end{bmatrix}$.

So, does A have an eigenvalue decomposition? **No**, because we don't have enough eigenvectors to form V. If we **try** and form V by using $\vec{v}_1$ in the first column and a column of zeros in the second column, we'd have

$$V = \begin{bmatrix} 1 & 0 \\ 0 & 0 \end{bmatrix}$$

but this is not an invertible matrix, so V is not invertible, and we can't write $A = V\Lambda V^{-1}$. A **is not diagonalizable**! This means that it's harder to interpret A as a linear transformation through the lens of eigenvectors, and it's harder to understand the long-run behavior of $A^k\vec{x}$ for an arbitrary $\vec{x}$.

Let me emphasize what I mean by A needing to have n **linearly independent** eigenvectors. Above, we found that $\begin{bmatrix} 1 \\ 0 \end{bmatrix}$ is an eigenvector of A, which means that any scalar multiple of $\begin{bmatrix} 1 \\ 0 \end{bmatrix}$ is also an eigenvector of A. But, the matrix

$$\begin{bmatrix} 1 & 2 \\ 0 & 0 \end{bmatrix}$$

is not invertible, meaning it can't be the V in $A = V\Lambda V^{-1}$.

Example: The Identity Matrix. The 2×2 identity matrix $I = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$ has the characteristic polynomial $p(\lambda) = (\lambda - 1)^2$. So, I has a single eigenvalue $\lambda = 1$, just like the last example.

But, $\lambda = 1$ corresponds to two **different** eigenvector directions! I can pick any two linearly independent vectors in $\mathbb{R}^2$ and they'll both be eigenvectors for I. For example, let $\vec{v}_1 = \begin{bmatrix} 1 \\ -3 \end{bmatrix}$ and $\vec{v}_2 = \begin{bmatrix} 11 \\ 98 \end{bmatrix}$, meaning

$$V = \begin{bmatrix} 1 & 11 \\ -3 & 98 \end{bmatrix}$$

The matrix $*$ is just $\begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$ which is the same as I itself. This means that as long as V is invertible (i.e. as long as I pick two linearly independent vectors in $\mathbb{R}^2$),

$$V\Lambda V^{-1} = VIV^{-1} = VV^{-1} = I$$

which means that indeed, I is diagonalizable. (Any matrix that is diagonal to begin with is diagonalizable: in $A = PDP^{-1}$, P is just the identity matrix.)

If you'd rather look at this example through the lens of solving systems of equations, an eigenvector $\vec{v} = \begin{bmatrix} a \\ b \end{bmatrix}$ of I with eigenvalue $\lambda = 1$ satisfies

$$I\vec{v} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} a \\ b \end{bmatrix} = \begin{bmatrix} a \\ b \end{bmatrix} = 1 \begin{bmatrix} a \\ b \end{bmatrix}$$

But, as a system this just says $a = a$ and $b = b$, which is true for any $\vec{v}$. Think of a and b both as independent variables; the set of possible $\vec{v}$'s, then, is two dimensional (and is all of $\mathbb{R}^2$).

Example: Non-Invertible Matrices. Let $A = \begin{bmatrix} 1 & 2 \\ 2 & 4 \end{bmatrix}$.

- A is not invertible (column 2 is double column 1), so it has an eigenvalue of $\lambda_1 = 0$, which corresponds to the eigenvector $\vec{v}_1 = \begin{bmatrix} -2 \\ 1 \end{bmatrix}$.
- It also has an eigenvalue of $\lambda_2 = 5$, corresponding to the eigenvector $\vec{v}_2 = \begin{bmatrix} 1 \\ 2 \end{bmatrix}$. (Remember, the quick way to spot this eigenvalue is to remember that the sum of the eigenvalues of A is equal to the trace of A, which is $1 + 4 = 5$; since 0 is an eigenvalue, the other eigenvalue must be 5.)

Does A have an eigenvalue decomposition? Let's see if anything goes wrong when we try and construct the eigenvector matrix V and the diagonal matrix $*$ and multiplying $V\Lambda V^{-1}$.

$$V = \begin{bmatrix} -2 & 1 \\ 1 & 2 \end{bmatrix}, \qquad \Lambda = \begin{bmatrix} 0 & 0 \\ 0 & 5 \end{bmatrix}$$

$$V\Lambda V^{-1} = \begin{bmatrix} -2 & 1 \\ 1 & 2 \end{bmatrix} \begin{bmatrix} 0 & 0 \\ 0 & 5 \end{bmatrix} \begin{bmatrix} -2/5 & 1/5 \\ 1/5 & 2/5 \end{bmatrix} = \begin{bmatrix} 1 & 2 \\ 2 & 4 \end{bmatrix}$$

Everything worked out just fine! A is diagonalizable, even though it's not invertible.

Diagonalizability is not related to invertibility!

An $n \times n$ matrix can be either, both, or neither. Knowing that a matrix is diagonalizable tells us nothing about its invertibility, and vice versa.
Remember, a matrix is diagonalizable if and only if it has n linearly independent eigenvectors, and is

invertible if and only if it has n linearly independent columns. But columns and eigenvectors are two different things!

Algebraic and Geometric Multiplicity

As we saw in an earlier example, the matrix $\begin{bmatrix} 1 & 1 \\ 0 & 1 \end{bmatrix}$ does not have two linearly independent eigenvectors, and so is not diagonalizable. I'd like to dive deeper into identifying **when** a matrix is diagonalizable.

Definition: Algebraic Multiplicity

The **algebraic multiplicity** of an eigenvalue λ_i is the number of times λ_i appears as a root of the characteristic polynomial of A.
That is, if A's characteristic polynomial is

$$p(\lambda) = (\lambda - \lambda_1)^{m_1}(\lambda - \lambda_2)^{m_2} \cdots (\lambda - \lambda_k)^{m_k}$$

then the algebraic multiplicity of λ_i is $\text{AM}(\lambda_i) = m_i$.
Note that in the above, $m_1 + m_2 + \cdots + m_k = n$.

Activity 2

Activity 2

Suppose an $n \times n$ matrix A has the characteristic polynomial

$$p(\lambda) = (\lambda + 1)^2\lambda(\lambda - 1)^3(\lambda - 4)^2(\lambda - 5)(\lambda - 12)^2$$

1. What is n (i.e. the number of rows/columns of A)?
2. What is the determinant of A?
3. What are all of A's eigenvalues and their algebraic multiplicities?

Solution

1. n is equal to the sum of the exponents of the polynomial, which is $2 + 1 + 3 + 2 + 1 + 2 = 11$. Remember that $p(\lambda)$ is a polynomial of degree n.
2. The determinant of A is the product of the eigenvalues. Notice that $p(\lambda)$ has a factor of λ, i.e. $p(0) = 0$, meaning 0 is an eigenvalue, so $\det(A) = 0$.
3. A has eigenvalues:
   - -1 with multiplicity $\text{AM}(-1) = 2$
   - 0 with multiplicity $\text{AM}(0) = 1$
   - 1 with multiplicity $\text{AM}(1) = 3$
   - 4 with multiplicity $\text{AM}(4) = 2$
   - 5 with multiplicity $\text{AM}(5) = 1$
   - 12 with multiplicity $\text{AM}(12) = 2$

   These algebraic multiplicities come from the exponents of the factors in $p(\lambda)$.

The algebraic multiplicity of an eigenvalue, as the name suggests, is purely a property of the characteristic polynomial. Alone, they don't tell us whether or not a matrix is diagonalizable. Instead, we'll need to look at another form of multiplicity alongside the algebraic multiplicity.

Definition: Eigenspace and Geometric Multiplicity

The **eigenspace** of an eigenvalue λ_i is the set of all eigenvectors with an eigenvalue of λ_i. Equivalently:

- It is the set of all vectors $\vec{v}$ such that $A\vec{v} = \lambda_i\vec{v}$.
- It is the **subspace** $\text{nullsp}(A - \lambda_i I)$, though $\vec{0}$ (which is in the null space of every matrix, including $A - \lambda_i I$) is not technically an eigenvector.

The **geometric multiplicity** of λ is the **dimension of the eigenspace** of λ.

$$\text{GM}(\lambda_i) = \dim(\text{nullsp}(A - \lambda_i I))$$

Equivalently, the geometric multiplicity of λ_i is **the number of linearly independent eigenvectors corresponding to** λ_i.

The geometric multiplicity of an eigenvalue can't be determined from the characteristic polynomial alone – instead, it involves finding $\text{nullsp}(A - \lambda_i I)$ and finding its dimension, i.e. the number of linearly independent vectors needed to span it. But in general,

$$1 \leq \text{GM}(\lambda_i) \leq \text{AM}(\lambda_i)$$

Think of the algebraic multiplicity of an eigenvalue as the "potential" number of linearly independent eigenvectors for an eigenvalue, sort of like the number of slots we have for that eigenvalue. The geometric multiplicity, on the other hand, is the number of linearly independent eigenvectors we actually have for that eigenvalue. As we'll see in the following examples, when the geometric multiplicity is less than the algebraic multiplicity for any eigenvalue, the matrix in question is not diagonalizable.

In each of the following matrices A, we'll

1. Find the algebraic multiplicity of each of A's eigenvalues.
2. For each eigenvalue, find the geometric multiplicity and a basis for the eigenspace.
3. Conclude whether A is diagonalizable.

As usual, attempt these examples on your own first before peeking at the solutions.

A First Example.

$$A = \begin{bmatrix} 1 & 1 & 0 \\ 0 & 1 & 1 \\ 0 & 0 & 1 \end{bmatrix}$$

We'll walk through this one together.

1. First, we'll find the characteristic polynomial of A:

$$p(\lambda) = \det(A - \lambda I) = \begin{vmatrix} 1-\lambda & 1 & 0 \\ 0 & 1-\lambda & 1 \\ 0 & 0 & 1-\lambda \end{vmatrix} = (1-\lambda)^3$$

   (You should practice quickly finding the characteristic polynomial of a 3×3 matrix; Chapter 9.2 has relevant examples.)

   This tells us that A has a single eigenvalue $\lambda = 1$, with algebraic multiplicity $\boxed{\text{AM}(1) = 3}$.

2. The geometric multiplicity of $\lambda = 1$ is $\dim(\text{nullsp}(A - I))$. Let's look at $A - I$:

$$A - I = \begin{bmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \\ 0 & 0 & 0 \end{bmatrix}$$

   $A - I$ has rank 2, so $\dim(\text{nullsp}(A - I)) = 3 - 2 = 1$ (from the rank-nullity theorem), so the geometric multiplicity of $\lambda = 1$ is $\boxed{\text{GM}(1) = 1}$.

   The vector $\vec{v}_1 = \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix}$ is an eigenvector for $\lambda = 1$, and so is any scalar multiple of $\vec{v}_1$; I found this by noticing that the first column of $A - I$ is all zeros. So,

$$\boxed{\text{nullsp}(A - I) = \text{span}\left(\left\{\begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix}\right\}\right)}$$

3. Since $\text{GM}(1) = 1 < \text{AM}(1) = 3$, A is not diagonalizable. A only has one linearly independent eigenvector!

A is diagonalizable if and only if $\text{AM}(\lambda_i) = \text{GM}(\lambda_i)$ for all eigenvalues λ_i!

If every eigenvalue "lives up to its potential", then the matrix has n linearly independent eigenvectors, and so is diagonalizable (because the V in $A = V\Lambda V^{-1}$ can be inverted).
If any eigenvalue has a geometric multiplicity that isn't equal to its algebraic multiplicity, then that eigenvalue will be missing eigenvectors, and A won't have n linearly independent eigenvectors total.

Now, it's your turn.

Example: Adjacency Matrices. Consider this adjacency matrix of a Markov chain with two states. (This is the same adjacency matrix we introduced in Chapter 9.3.)

$$A = \begin{bmatrix} 0.8 & 0.3 \\ 0.2 & 0.7 \end{bmatrix}$$

Is A diagonalizable? If so, find V and Λ such that $A = V\Lambda V^{-1}$.

Solution

1. In Chapter 9.3, we found that A has eigenvalues $\lambda_1 = 1$ and $\lambda_2 = 0.5$, but we'll find the characteristic polynomial again for clarity:

$$\begin{aligned} p(\lambda) &= \det(A - \lambda I) \\ &= \begin{vmatrix} 0.8 - \lambda & 0.3 \\ 0.2 & 0.7 - \lambda \end{vmatrix} \\ &= (0.8 - \lambda)(0.7 - \lambda) - (0.3)(0.2) \\ &= \lambda^2 - 1.5\lambda + 0.5 \\ &= (1 - \lambda)(0.5 - \lambda) \end{aligned}$$

   So, the eigenvalues are $\lambda_1 = 1$ and $\lambda_2 = 0.5$, each with algebraic multiplicity $\boxed{\text{AM}(1) = \text{AM}(0.5) = 1}$.

2. The fact that A has two distinct eigenvalues alone tells us that A is diagonalizable, since each eigenvalue has exactly one independent eigenvector (which really means a line of eigenvectors, or a 1-dimensional eigenspace) and eigenvectors for different eigenvalues are always linearly independent. Let's find those eigenvectors to be sure.

   - For $\lambda_1 = 1$, we solve $(A - I)\vec{v} = \vec{0}$:

$$A - I = \begin{bmatrix} -0.2 & 0.3 \\ 0.2 & -0.3 \end{bmatrix}$$

     The null space of this matrix is spanned by $\begin{bmatrix} 3 \\ 2 \end{bmatrix}$, so $\boxed{\text{nullsp}(A - I) = \text{span}\left(\left\{\begin{bmatrix} 3 \\ 2 \end{bmatrix}\right\}\right)}$ and $\boxed{\text{GM}(1) = 1}$.

   - For $\lambda_2 = 0.5$, we solve $(A - 0.5I)\vec{v} = \vec{0}$:

$$A - 0.5I = \begin{bmatrix} 0.3 & 0.3 \\ 0.2 & 0.2 \end{bmatrix}$$

     The null space of this matrix is spanned by $\begin{bmatrix} -1 \\ 1 \end{bmatrix}$, so $\boxed{\text{nullsp}(A - 0.5I) = \text{span}\left(\left\{\begin{bmatrix} -1 \\ 1 \end{bmatrix}\right\}\right)}$ and $\boxed{\text{GM}(0.5) = 1}$.

3. Since A has two linearly independent eigenvectors, it is diagonalizable:

$$A = V\Lambda V^{-1} = \begin{bmatrix} 3 & -1 \\ 2 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 \\ 0 & 0.5 \end{bmatrix} \left(\begin{bmatrix} 3 & -1 \\ 2 & 1 \end{bmatrix} \right)^{-1}$$

Example: Another Diagonalizable Matrix.

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

Is A diagonalizable? If so, find V and Λ such that $A = V\Lambda V^{-1}$.

Solution

1. The characteristic polynomial of A is

$$\begin{aligned} p(\lambda) &= \det(A - \lambda I) \\ &= \begin{vmatrix} 2-\lambda & 1 & 0 \\ 1 & 2-\lambda & 0 \\ 0 & 0 & 3-\lambda \end{vmatrix} \\ &= (2-\lambda)\left((2-\lambda)(3-\lambda) - 0\cdot 0\right) - 1\left((1)(3-\lambda) - 0\cdot 0\right) + 0 \\ &= (2-\lambda)(2-\lambda)(3-\lambda) - (3-\lambda) \\ &= (3-\lambda)\left((2-\lambda)^2 - 1\right) \\ &= (3-\lambda)(\lambda^2 - 4\lambda + 3) \\ &= (3-\lambda)^2(1-\lambda) \end{aligned}$$

So, A has eigenvalues $\lambda_1 = 3$ with $\boxed{\text{AM}(3) = 2}$ and $\lambda_2 = 1$ with $\boxed{\text{AM}(1) = 1}$. There's a quicker way we could have spotted A's eigenvalues, discussed after the solutions box.

2. • For $\lambda_1 = 3$, we solve $(A - 3I)\vec{v} = \vec{0}$:

$$A - 3I = \begin{bmatrix} -1 & 1 & 0 \\ 1 & -1 & 0 \\ 0 & 0 & 0 \end{bmatrix}$$

$\text{rank}(A - 3I) = 1$ (since all columns are multiples of $\begin{bmatrix} -1 \\ 1 \\ 0 \end{bmatrix}$), so $\dim(\text{nullsp}(A - 3I)) = 3 - 1 = 2$, which means $\boxed{\text{GM}(3) = 2}$. At this point, we can conclude that A is diagonalizable! But, let's find a basis for the eigenspace $\text{nullsp}(A - 3I)$ to be thorough.

Two linearly independent vectors in $\text{nullsp}(A - 3I)$ are $\begin{bmatrix} 1 \\ 1 \\ 0 \end{bmatrix}$ and $\begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix}$, so $\boxed{\text{nullsp}(A - 3I) = \text{span}\left(\left\{\begin{bmatrix} 1 \\ 1 \\ 0 \end{bmatrix}, \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix}\right\}\right)}$. This is just one possible basis for the eigenspace; it's not the only one.

What this says is **any linear combination of $\begin{bmatrix} 1 \\ 1 \\ 0 \end{bmatrix}$ and $\begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix}$ is also an eigenvector for $\lambda = 3$.** For example, $\begin{bmatrix} 2 & 1 & 0 \\ 1 & 2 & 0 \\ 0 & 0 & 3 \end{bmatrix}\begin{bmatrix} -1 \\ -1 \\ 13 \end{bmatrix} = 3\begin{bmatrix} -1 \\ -1 \\ 13 \end{bmatrix}$.

• For $\lambda_2 = 1$, we solve $(A - I)\vec{v} = \vec{0}$:

$$A - I = \begin{bmatrix} 1 & 1 & 0 \\ 1 & 1 & 0 \\ 0 & 0 & 2 \end{bmatrix}$$

Since $\text{AM}(1) = 1$, we know that there could only be one linearly independent eigenvector for $\lambda_2 = 1$, so $\boxed{\text{GM}(1) = 1}$ too. One such eigenvector is $\begin{bmatrix} 1 \\ -1 \\ 0 \end{bmatrix}$, so $\boxed{\text{nullsp}(A - I) = \text{span}\left(\left\{\begin{bmatrix} 1 \\ -1 \\ 0 \end{bmatrix}\right\}\right)}$.

3. Since A has three linearly independent eigenvectors, it is diagonalizable:

$$\begin{bmatrix} 1 & 0 & 1 \\ 1 & 0 & -1 \\ 0 & 1 & 0 \end{bmatrix}\begin{bmatrix} 3 & 0 & 0 \\ 0 & 3 & 0 \\ 0 & 0 & 1 \end{bmatrix}\left(\begin{bmatrix} 1 & 0 & 1 \\ 1 & 0 & -1 \\ 0 & 1 & 0 \end{bmatrix}\right)^{-1}$$

In the solutions above, I found A's eigenvalues the traditional way, by solving for the roots of A's characteristic polynomial. But there's a quicker way to find A's eigenvalues in this case that I want to highlight. A is a **block diagonal** matrix:

$$A = \begin{bmatrix} 2 & 1 & 0 \\ 1 & 2 & 0 \\ 0 & 0 & 3 \end{bmatrix} = \begin{bmatrix} A_1 & \vec{0}_{2\times 1} \\ \vec{0}_{1\times 2} & A_2 \end{bmatrix}$$

In this case, $A_1 = \begin{bmatrix} 2 & 1 \\ 1 & 2 \end{bmatrix}$ and $A_2 = \begin{bmatrix} 3 \end{bmatrix}$. Think of this as us placing A_1 and A_2 on the diagonal of A, and then filling the rest of A with zeros.

In general, if A is a block diagonal matrix, then the eigenvalues of A are the eigenvalues of the blocks on the diagonal combined. To see why this is true, suppose you know that λ is an eigenvalue of one of A's diagonal blocks with a corresponding eigenvector of $\vec{v}$. To get a corresponding eigenvector of A, just extend $\vec{v}$ by placing 0's in the positions of the other blocks. In other words, if $\vec{v}$ is an eigenvector of A_1, then $\begin{bmatrix} \vec{v} \\ 0 \\ \vdots \\ 0 \end{bmatrix}$ is an eigenvector of A.

Here, the eigenvalues of A_1 are 1 and 3, and the sole eigenvalue of A_2 is 3, so A's eigenvalues are 1 and 3 (with algebraic multiplicity 2).

Example: Another Non-Diagonalizable Matrix.

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

Is A diagonalizable? If so, find V and Λ such that $A = V\Lambda V^{-1}$. *Tip: Use the fact that A is upper triangular to find its eigenvalues quickly. We first discussed this trick in Chapter 9.2.*

Solution

1. A is upper triangular, so its eigenvalues are the entries on the diagonal: $\lambda_1 = 2$, $\lambda_2 = 2$, $\lambda_3 = 3$. So, $\boxed{\text{AM}(2) = 2}$ and $\boxed{\text{AM}(3) = 1}$.

2. - For $\lambda_1 = 2$, we solve $(A - 2I)\vec{v} = \vec{0}$:

$$A - 2I = \begin{bmatrix} 0 & 1 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

   $\text{rank}(A - 2I) = 2$, so $\dim(\text{nullsp}(A - 2I)) = 1$, so $\boxed{\text{GM}(2) = 1}$. At this point, we can conclude that A is not diagonalizable, since $\text{GM}(2) < \text{AM}(2)$. Noticing that the first column of $A - 2I$ is all zeros, $\vec{v}_1 = \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix}$ is an eigenvector, and

$$\boxed{\text{nullsp}(A - 2I) = \text{span}\left(\left\{\begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix}\right\}\right)}$$

   - For $\lambda_3 = 3$, we solve $(A - 3I)\vec{v} = \vec{0}$:

$$A - 3I = \begin{bmatrix} -1 & 1 & 0 \\ 0 & -1 & 0 \\ 0 & 0 & 0 \end{bmatrix}$$

   Using similar logic, $\boxed{\text{GM}(3) = 1}$ and

$$\boxed{\text{nullsp}(A - 3I) = \text{span}\left(\left\{\begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix}\right\}\right)}$$

3. A only has two linearly independent eigenvectors, one for $\lambda = 2$ and one for $\lambda = 3$, but is a 3×3 matrix, and hence its not diagonalizable. (Also, for $\lambda_1 = 2$, the geometric multiplicity, 1, is less than the algebraic multiplicity, 2, but we need equality for all eigenvalues in order for A to be diagonalizable.)

Example: Working Backwards. In the previous examples, we were given a matrix A and then found its eigenvalues and eigenvectors. Let's go in the other direction.

Suppose A is a 3×3 matrix with:

- eigenvalue $\lambda_1 = 3$ with eigenvector $\vec{v}_1 = \begin{bmatrix} 2 \\ 0 \\ 1 \end{bmatrix}$
- eigenvalue $\lambda_2 = -2$ with eigenspace

$$\text{span}\left(\left\{\begin{bmatrix}1\\1\\0\end{bmatrix}, \begin{bmatrix}0\\1\\2\end{bmatrix}\right\}\right)$$

Find A.

Solution

If we're given eigenvalues and enough linearly independent eigenvectors, we can build A directly from the decomposition $A = V\Lambda V^{-1}$.

Since the eigenspace for $\lambda = -2$ is 2-dimensional, we can use $\begin{bmatrix}1\\1\\0\end{bmatrix}$ and $\begin{bmatrix}0\\1\\2\end{bmatrix}$ as two linearly independent eigenvectors for $\lambda = -2$. So, one valid choice is

$$V = \begin{bmatrix}2 & 1 & 0\\0 & 1 & 1\\1 & 0 & 2\end{bmatrix}, \quad \Lambda = \begin{bmatrix}3 & 0 & 0\\0 & -2 & 0\\0 & 0 & -2\end{bmatrix}$$

where the columns of V are the eigenvectors, and the diagonal entries of Λ are the corresponding eigenvalues in the same order.
Now we compute

$$A = V\Lambda V^{-1} = \begin{bmatrix}2 & 1 & 0\\0 & 1 & 1\\1 & 0 & 2\end{bmatrix}\begin{bmatrix}3 & 0 & 0\\0 & -2 & 0\\0 & 0 & -2\end{bmatrix}\left(\begin{bmatrix}2 & 1 & 0\\0 & 1 & 1\\1 & 0 & 2\end{bmatrix}\right)^{-1} = \boxed{\begin{bmatrix}2 & -4 & 2\\0 & -2 & 0\\2 & -2 & -1\end{bmatrix}}$$

You can quickly verify that this works:

- $A\begin{bmatrix}2\\0\\1\end{bmatrix} = 3\begin{bmatrix}2\\0\\1\end{bmatrix}$
- $A\begin{bmatrix}1\\1\\0\end{bmatrix} = -2\begin{bmatrix}1\\1\\0\end{bmatrix}$
- $A\begin{bmatrix}0\\1\\2\end{bmatrix} = -2\begin{bmatrix}0\\1\\2\end{bmatrix}$

The main takeaway is that **if you know a full eigenbasis and the matching eigenvalues, you can reconstruct the original matrix** by putting the eigenvectors into V, the eigenvalues into Λ, and multiplying $V\Lambda V^{-1}$.

In Chapter 9.5, I want to build on our understanding of diagonalizability by focusing on a special class of matrices: symmetric matrices. It turns out that symmetric matrices are always diagonalizable, and beyond that, their corresponding eigenvectors obey a really nice property. Chapter 9.5 also has an overall summary of diagonalization.

Activity 3

Activity 3

For each statement, determine whether it is true or false and provide a brief justification. (For false statements, provide a counterexample.)

1. True or False: If all of A's eigenvalues are distinct, then A is diagonalizable.
2. True or False: If all of A's eigenvalues are positive, then A is diagonalizable.
3. True or False: If A is diagonalizable, then A is invertible.
4. True or False: If A is invertible, then A is diagonalizable.

Solution

1. If all of A's eigenvalues are distinct, then A is diagonalizable. **True**. Remember that the minimum possible geometric multiplicity is 1, so if all eigenvalues are distinct, they each have an algebraic and geometric multiplicity of 1. Eigenvectors for different eigenvalues are always linearly independent; think of it this way, an eigenvector can only ever be paired with a single eigenvalue, so eigenvectors for different eigenvalues can't be linearly dependent. Pairing these two facts, A must have n linearly independent eigenvectors, and so is diagonalizable.
2. If all of A's eigenvalues are positive, then A is diagonalizable. **False**. In the example above titled "Another Non-Diagonalizable Matrix", A had eigenvalues of 2, 2, and 3, all of which are positive, but A was not diagonalizable.
3. If A is diagonalizable, then A is invertible. **False**. We saw an example of a diagonalizable matrix that was not invertible earlier in this section:

$$\begin{bmatrix} 1 & 2 \\ 2 & 4 \end{bmatrix} = \underbrace{\begin{bmatrix} 2 & 1 \\ -1 & 2 \end{bmatrix}}_{V} \underbrace{\begin{bmatrix} 0 & 0 \\ 0 & 5 \end{bmatrix}}_{\Lambda} \underbrace{\left(\begin{bmatrix} 2 & 1 \\ -1 & 2 \end{bmatrix}\right)^{-1}}_{V^{-1}}$$

4. If A is invertible, then A is diagonalizable. **False**. The matrix from the example "Another Non-Diagonalizable Matrix" above had eigenvalues of 2, 2, and 3, none of which are 0, meaning A is invertible, but A was not diagonalizable.

Most people with "AI/ML" in their bios don't even know a real symmetric matrix always has real eigenvalues.

Figure 9.1: *
vixhaℓ (@TheVixhal) March 29, 2026

9.5. Symmetric Matrices and the Spectral Theorem

Symmetric matrices – that is, square matrices where $A = A^T$ – behave **really** nicely through the lens of eigenvectors, and understanding exactly how they work is key to Chapter 10.1, when we generalize beyond square matrices.

While editing these notes, I came across a fitting tweet:

The Spectral Theorem

The Spectral Theorem

If A is an $n \times n$ symmetric matrix with real entries (like all matrices we've studied in this course), then the **spectral theorem** states that:

1. A has exactly n real eigenvalues, counting multiplicities (i.e. none are complex numbers).
2. The eigenvectors for different eigenvalues are orthogonal.
3. For every eigenvalue λ_i, $\text{GM}(\lambda_i) = \text{AM}(\lambda_i)$.
4. Putting (1), (2), and (3) together, there exists an orthogonal matrix Q and a diagonal matrix $*$ such that

$$A = Q*Q^T$$

Equivalently, "A can be diagonalized by an orthogonal matrix".

If you search for the spectral theorem online, you'll often just see Statement 4 above; I've broken the theorem into smaller substatements to see how they are chained together.

The proof of Statement 1 is beyond our scope, since it involves fluency with complex numbers. If the term "complex conjugate" means something to you, read the proof here – it's relatively short.

The key idea to prove is Statement 2: that for a symmetric matrix, eigenvectors corresponding to different eigenvalues are orthogonal. Suppose $\vec{v}_i$ is an eigenvector of A with eigenvalue λ_i and $\vec{v}_j$ is an eigenvector of A with eigenvalue λ_j, where $\lambda_i \neq \lambda_j$. Then

$$A\vec{v}_i = \lambda_i\vec{v}_i \qquad \text{and} \qquad A\vec{v}_j = \lambda_j\vec{v}_j$$

Consider the dot product $\vec{v}_i \cdot (A\vec{v}_j)$. Using the fact that $\vec{v}_j$ is an eigenvector, we get

$$\vec{v}_i \cdot (A\vec{v}_j) = \vec{v}_i \cdot (\lambda_j\vec{v}_j) = \lambda_j(\vec{v}_i \cdot \vec{v}_j)$$

But we can also rewrite the same quantity using the fact that A is symmetric:

$$\vec{v}_i \cdot (A\vec{v}_j) = \vec{v}_i^T A\vec{v}_j = \vec{v}_i^T A^T \vec{v}_j = (A\vec{v}_i)^T \vec{v}_j = \lambda_i \vec{v}_i^T \vec{v}_j = \lambda_i(\vec{v}_i \cdot \vec{v}_j)$$

So,

$$\lambda_j(\vec{v}_i \cdot \vec{v}_j) = \lambda_i(\vec{v}_i \cdot \vec{v}_j)$$

which means

$$(\lambda_j - \lambda_i)(\vec{v}_i \cdot \vec{v}_j) = 0$$

Since $\lambda_i \neq \lambda_j$, the first factor is non-zero, so we must have $\vec{v}_i \cdot \vec{v}_j = 0$. Therefore, eigenvectors corresponding to different eigenvalues are orthogonal.

For a given eigenvector direction, we can pick any vector in that direction to be the eigenvector we store in the V that we use to diagonalize A – if $\vec{v}$ is an eigenvector, so is $2\vec{v}$, $-3\vec{v}$, $\frac{\vec{v}}{\|\vec{v}\|}$, and so on. The convenient choice is to pick unit vectors in each direction. If we take these n unit eigenvectors and place them in the columns of a matrix, that matrix is an orthogonal matrix! Orthogonal matrices Q satisfy $Q^TQ = QQ^T = I$, meaning their columns (and rows) are ortho**normal**, not just orthogonal to one another. The fact that $Q^TQ = QQ^T = I$ means that $Q^T = Q^{-1}$, so taking the transpose of a matrix is the same as taking its inverse.

$$A = V\Lambda V^{-1}$$

we've "upgraded" to

$$A = Q\Lambda Q^T$$

This is the main takeaway of the spectral theorem: that symmetric matrices can be diagonalized by an orthogonal matrix. Sometimes, $A = Q\Lambda Q^T$ is called the **spectral decomposition** of A, but all it is is a special case of the eigenvalue decomposition for symmetric matrices.

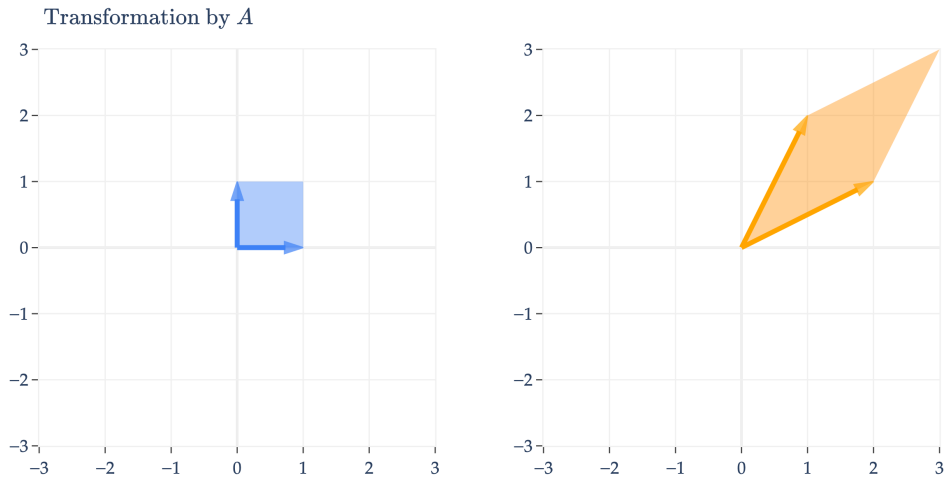
Visualizing the Spectral Theorem. Why do we prefer $Q\Lambda Q^T$ over $V\Lambda V^{-1}$? Taking the transpose of a matrix is much easier than inverting it, so actually working with $Q\Lambda Q^T$ is easier.

$$\underbrace{A = Q\Lambda Q^T \implies A^k = Q\Lambda^k Q^T}_{\text{no inversion needed!}}$$

But it's also an improvement in terms of **interpretation**: remember that orthogonal matrices are matrices that represent rotations. So, if A is symmetric, then the linear transformation $f(\vec{x}) = A\vec{x}$ is a sequence of rotations and stretches.

$$f(\vec{x}) = A\vec{x} = Q\Lambda Q^T\vec{x}$$

Let's make sense of this visually. Consider the symmetric matrix $A = \begin{bmatrix} 1 & 2 \\ 2 & 1 \end{bmatrix}$.



A appears to perform an arbitrary transformation; it turns the unit square into a parallelogram, as we first saw in Chapter 6.1.

But, since A is symmetric, it can be diagonalized by an orthogonal matrix, $A = Q\Lambda Q^T$.

$$A = \begin{bmatrix} 1 & 2 \\ 2 & 1 \end{bmatrix}$$

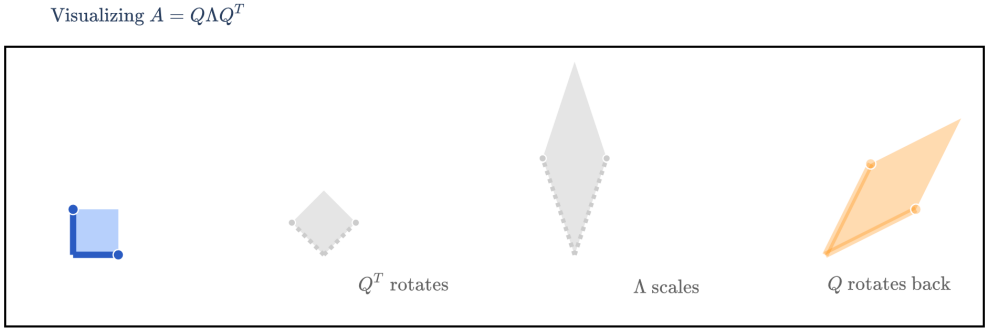
has eigenvalues $\lambda_1 = 3$ with eigenvector $\vec{v}_1 = \begin{bmatrix} 1 \\ 1 \end{bmatrix}$ and $\lambda_2 = -1$ with eigenvector $\vec{v}_2 = \begin{bmatrix} -1 \\ 1 \end{bmatrix}$. But, the $\vec{v}_i$'s I've written aren't unit vectors, which they need to be in order for Q to be orthogonal. So, we normalize them to get $\vec{q}_1 = \begin{bmatrix} \frac{1}{\sqrt{2}} \\ \frac{1}{\sqrt{2}} \end{bmatrix}$ and $\vec{q}_2 = \begin{bmatrix} -\frac{1}{\sqrt{2}} \\ \frac{1}{\sqrt{2}} \end{bmatrix}$. Placing these $\vec{q}_i$'s as columns of Q, we get

$$Q = \begin{bmatrix} \frac{1}{\sqrt{2}} & -\frac{1}{\sqrt{2}} \\ \frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} \end{bmatrix}$$

and so

$$A = Q\Lambda Q^T = \underbrace{\begin{bmatrix} \frac{1}{\sqrt{2}} & -\frac{1}{\sqrt{2}} \\ \frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} \end{bmatrix}}_{Q} \underbrace{\begin{bmatrix} 3 & 0 \\ 0 & -1 \end{bmatrix}}_{\Lambda} \underbrace{\begin{bmatrix} \frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} \\ -\frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} \end{bmatrix}}_{Q^T}$$

We're visualizing how $\vec{x}$ turns into $A\vec{x}$, i.e. how $\vec{x}$ turns into $Q\Lambda Q^T\vec{x}$. This means that we first need to consider the effect of Q^T on $\vec{x}$, then the effect of $*$ on that result, and finally the effect of Q on that result – that is, read the matrices from right to left.

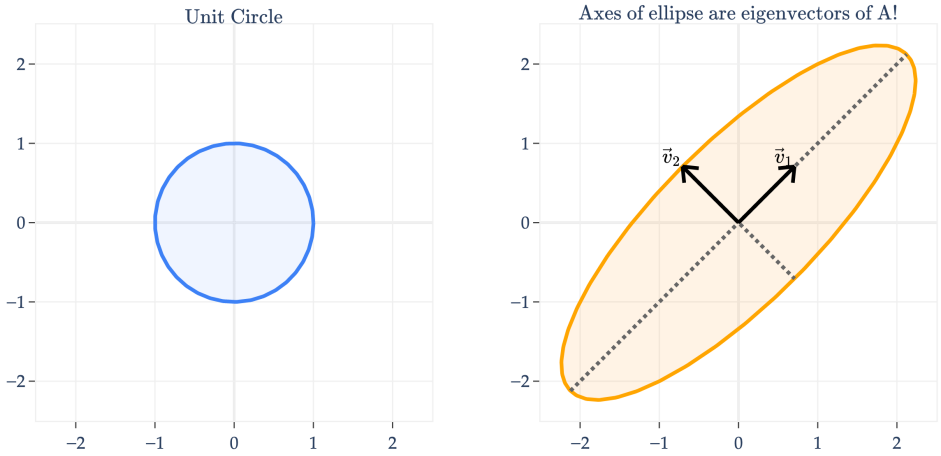


Symmetric matrices rotate, stretch along eigenvectors, and then rotate back!

$$\underbrace{\text{rotate to align with eigenvectors}}_{\text{multiply by } Q^T} \rightarrow \underbrace{\text{scale eigenvectors}}_{\text{multiply by } \Lambda} \rightarrow \underbrace{\text{rotate back}}_{\text{multiply by } Q}$$

This interpretation is **crucial** to understanding more advanced machine learning techniques, as we'll see in Chapter 10.1.

The Ellipse Perspective. Another way of visualizing the linear transformation of a symmetric matrix is to consider its effect on the unit **circle**, not the unit square. Below, I'll apply $A = \begin{bmatrix} 1 & 2 \\ 2 & 1 \end{bmatrix}$ to the unit circle.



Notice that A transformed the unit circle into an ellipse. What's more, the **axes of the ellipse are the eigenvector directions of A!**

Why is one axis longer than the other? As you might have guessed, the longer axis – the one in the direction of the eigenvector $\vec{v}_1 = \begin{bmatrix} 1 \\ 1 \end{bmatrix}$ – corresponds to the larger eigenvalue. Remember that A has $\lambda_1 = 3$ and $\lambda_2 = -1$, so the "up and to the right" axis is three times longer than the "down and to the right" axis, defined by $\vec{v}_2 = \begin{bmatrix} 1 \\ -1 \end{bmatrix}$.

Why does this happen? Since A is symmetric, it has a spectral decomposition $A = Q\Lambda Q^T$, where Q is orthogonal and Λ is diagonal. For any vector $\vec{x}$, let $\vec{y} = Q^T\vec{x}$. Then

$$\vec{x}^T A\vec{x} = \vec{x}^T(Q\Lambda Q^T)\vec{x} = \vec{y}^T\Lambda\vec{y} = \sum_{i=1}^{n} \lambda_i y_i^2$$

Now imagine that $\vec{x}$ lies on one of A's eigenvector directions, say the direction of $\vec{v}_1$. In that case, after rotating by Q^T, the vector $\vec{y}$ has only one non-zero coordinate, namely the coordinate corresponding to λ_1. So the sum above collapses to

$$\vec{x}^T A\vec{x} = \lambda_1 y_1^2$$

Similarly, along the eigenvector direction of $\vec{v}_2$, we get $\vec{x}^T A\vec{x} = \lambda_2 y_2^2$. The size of the output along each principal axis is therefore controlled by the corresponding eigenvalue. Larger eigenvalues produce longer axes, and smaller eigenvalues produce shorter axes. Here, $\lambda_1 = 3$ and $\lambda_2 = -1$, so the axis in the $\vec{v}_1$ direction is longer in

magnitude than the axis in the $\vec{v}_2$ direction.

Key Takeaways

1. The **eigenvalue decomposition** of a matrix A is a decomposition of the form

$$A = V\Lambda V^{-1}$$

   where V is a matrix containing the eigenvectors of A as columns, and $*$ is a diagonal matrix of eigenvalues in the same order. Only diagonalizable matrices can be decomposed in this way.

2. The **algebraic multiplicity** of an eigenvalue λ_i is the number of times λ_i appears as a root of the characteristic polynomial of A.

3. The **geometric multiplicity** of λ is the **dimension of the eigenspace** of λ, i.e. $\dim(\text{nullsp}(A - \lambda I))$.

4. The $n \times n$ matrix is **diagonalizable** if and only if any of these equivalent conditions are true:

   - A has n linearly independent eigenvectors.
   - For every eigenvalue λ_i, $\text{GM}(\lambda_i) = \text{AM}(\lambda_i)$. When A is diagonalizable, it has an eigenvalue decomposition, $A = V\Lambda V^{-1}$.

5. If A is a symmetric matrix, then the **spectral theorem** tells us that A can be diagonalized by an orthogonal matrix Q such that

$$A = Q\Lambda Q^T$$

   and that all of A's eigenvalues are guaranteed to be real.

What's next? There's the question of **how any of this relates to real data**. Real data comes in rectangular matrices, not square matrices. And even it were square, how does any of this enlighten us?

9.6. Positive Semidefinite Matrices and the Rayleigh Quotient

Attention

This chapter is currently under development and has more bugs and typos than usual.

In Chapter 9.5, we saw that symmetric matrices can be diagonalized by an orthogonal matrix. In this section, we will use that fact to understand two closely related ideas:

- **positive semidefinite matrices**, which control when a quadratic form has a global minimum and is convex, and
- the **Rayleigh quotient**, which is a normalized quadratic form that isolates the effect of direction.

Quadratic Forms and Positive Semidefinite Matrices

If A is an $n \times n$ matrix, then the function

$$f(\vec{x}) = \vec{x}^T A \vec{x}$$

is called a **quadratic form**. When A is symmetric, quadratic forms are especially nice because the spectral theorem lets us understand them completely in terms of eigenvalues and eigenvectors.

Definition: Positive Semidefinite Matrices

Suppose A is a symmetric $n \times n$ matrix. We say that A is **positive semidefinite** if

$$\vec{x}^T A \vec{x} \geq 0 \quad \text{for every } \vec{x} \in \mathbb{R}^n$$

Equivalently, A is positive semidefinite if all of its eigenvalues are non-negative.
Symbolically, a positive semidefinite matrix is often written $A \succeq 0$.

A positive **definite** matrix is even stronger: it satisfies

$$\vec{x}^T A \vec{x} > 0 \quad \text{for every } \vec{x} \neq \vec{0},$$

which is equivalent to saying that all eigenvalues of A are strictly positive.

Quadratic forms already appeared earlier in the course. For example, the mean-squared error from Chapter 8.1 can be expanded into a quadratic expression in the weight vector $\vec{w}$.

$$R_{\text{sq}}(\vec{w}) = \frac{1}{n} \| \vec{y} - X\vec{w} \|^2$$

So understanding when a quadratic form has a minimum is not just abstract linear algebra; it is directly tied to optimization.

Why are the two definitions of positive semidefiniteness equivalent? Since A is symmetric, we can write

$$A = Q \Lambda Q^T$$

where Q is orthogonal and Λ is diagonal with eigenvalues $\lambda_1, \ldots, \lambda_n$ on the diagonal. For any vector $\vec{x}$, let $\vec{y} = Q^T\vec{x}$. Then

$$\vec{x}^T A\vec{x} = \vec{x}^T(Q\Lambda Q^T)\vec{x} = \vec{y}^T\Lambda\vec{y} = \sum_{i=1}^{n} \lambda_i y_i^2$$

This formula is the key. Each y_i^2 is non-negative, so if every eigenvalue $\lambda_i \geq 0$, then every term $\lambda_i y_i^2$ is non-negative and therefore $\vec{x}^T A\vec{x} \geq 0$ for every $\vec{x}$. Conversely, if some eigenvalue were negative, then plugging in the corresponding eigenvector would make $\vec{x}^T A\vec{x} < 0$.

The picture on the left comes from the positive definite matrix

$$A = \begin{bmatrix} 2 & 1 \\ 1 & 2 \end{bmatrix},$$

whose eigenvalues are 3 and 1. Every output is non-negative, and the level curves are ellipses surrounding a single global minimum at the origin.

The picture on the right comes from the symmetric matrix

$$A = \begin{bmatrix} 1 & 5 \\ 5 & 1 \end{bmatrix},$$

whose eigenvalues are 6 and -4. Because one eigenvalue is negative, the quadratic form takes both positive and negative values, so it cannot be positive semidefinite and it is not convex.

For a symmetric quadratic form, convexity is controlled exactly by positive semidefiniteness. If

$$f(\vec{x}) = \vec{x}^T A\vec{x},$$

then

$$\nabla f(\vec{x}) = 2A\vec{x} \qquad \text{and} \qquad H_f = 2A,$$

because $A = A^T$. From multivariable calculus, a function is convex exactly when its Hessian is positive semidefinite. So for symmetric quadratic forms,

$$f(\vec{x}) = \vec{x}^T A\vec{x} \text{ is convex} \iff A \succeq 0$$

This is one reason positive semidefinite matrices show up constantly in optimization: they tell us that the landscape bends upward in every direction.

But there is still one issue. The value of $\vec{x}^T A\vec{x}$ depends on both the **direction** of $\vec{x}$ and its **length**. If we double $\vec{x}$, then the value quadruples. To study the effect of direction alone, we normalize by the squared length of the vector.

The Rayleigh Quotient

Suppose A is a symmetric $n \times n$ matrix. The **Rayleigh quotient** of A is the function

$$g(\vec{v}) = \frac{\vec{v}^T A\vec{v}}{\vec{v}^T\vec{v}}$$

for all non-zero vectors $\vec{v}$.

You should think of this as a **normalized quadratic form**. The numerator $\vec{v}^T A\vec{v}$ measures the output of the quadratic form, while the denominator $\vec{v}^T\vec{v} = \|\vec{v}\|^2$ removes the effect of scale.

Indeed, if $c \neq 0$, then

$$g(c\vec{v}) = \frac{(c\vec{v})^T A(c\vec{v})}{(c\vec{v})^T(c\vec{v})} = \frac{c^2\vec{v}^T A\vec{v}}{c^2\vec{v}^T\vec{v}} = g(\vec{v})$$

So the Rayleigh quotient depends only on the **direction** of $\vec{v}$, not on its magnitude. In particular, if $\|\vec{v}\| = 1$, then

$$g(\vec{v}) = \vec{v}^T A\vec{v},$$

so the Rayleigh quotient is just the quadratic form restricted to the unit sphere.

In Homework 9, Problem 4, you showed that

$$\nabla g(\vec{v}) = \frac{2}{\vec{v}^T\vec{v}}\left(A\vec{v} - g(\vec{v})\vec{v}\right)$$

If $\vec{v}$ is a critical point of g, then $\nabla g(\vec{v}) = \vec{0}$, which forces

$$A\vec{v} = g(\vec{v})\vec{v}$$

That means every critical point of the Rayleigh quotient is an eigenvector of A, and the corresponding value of the Rayleigh quotient is the associated eigenvalue.

The dashed lines mark the eigenvector directions of A. Notice what changed compared to the earlier quadratic form plot: the extreme values no longer occur farther and farther away from the origin. After normalization, only **direction** matters.

The reddest direction is the eigenvector direction corresponding to the largest eigenvalue, which is 6. The bluest direction is the eigenvector direction corresponding to the smallest eigenvalue, which is -4. On the unit circle, those are exactly the maximum and minimum values of the quadratic form.

So for a symmetric matrix A,

- the **largest possible value** of the Rayleigh quotient is the largest eigenvalue of A, and
- the **smallest possible value** of the Rayleigh quotient is the smallest eigenvalue of A.

This gives a geometric interpretation of eigenvectors: they are the directions where the normalized quadratic form is stationary, and in fact extremized.

Activity 1

Activity 1

Suppose $x, y \in \mathbb{R}$. What are the largest and smallest possible values of

$$f(x, y) = \frac{2x^2 + 12xy + 7y^2}{x^2 + y^2}$$

Solution

First, we should write $f(x, y)$ as a Rayleigh quotient. Let $\vec{x} = \begin{bmatrix} x \\ y \end{bmatrix}$ and

$$A = \begin{bmatrix} 2 & 6 \\ 6 & 7 \end{bmatrix}$$

Then,

$$f(x, y) = \frac{\vec{x}^T A \vec{x}}{\vec{x}^T \vec{x}} = g(\vec{x})$$

Before going any further, you should verify that $\vec{x}^T A \vec{x}$ indeed is equal to $2x^2 + 12xy + 7y^2$.
Using the logic introduced before this activity, the largest possible value of $f(x, y)$ is the largest eigenvalue of A, and the smallest possible value is the smallest eigenvalue of A.
The eigenvalues of A must

- sum to $\text{trace}(A) = 2 + 7 = 9$, and
- multiply to $\det(A) = 2 \cdot 7 - 6^2 = -22$.

So, A's eigenvalues are 11 and -2. Thus, the largest possible value of $f(x, y)$ is 11, and the smallest possible value of $f(x, y)$ is -2.

The Rayleigh quotient will reappear in Chapter 10.3, where we'll apply it with $A = \tilde{X}^T \tilde{X}$ to find the best direction for dimensionality reduction.

Singular Value Decomposition

10.1. Computing the Singular Value Decomposition

Eigenvalues and eigenvectors are powerful concepts, but unfortunately, they only apply to square matrices. It would be nice if we could extend some of their utility to non-square matrices, like matrices containing real data, which typically have many more rows than columns.

Introduction

This is precisely where the **singular value decomposition** (SVD) comes in. You should think of it as a generalization of the eigenvalue decomposition, $A = V\Lambda V^{-1}$, to non-square matrices. When we first introduced eigenvalues and eigenvectors, we said that an eigenvector $\vec{v}$ of A is a vector whose direction is unchanged when multiplied by A – all that happens to $\vec{v}$ is that it's scaled by a factor of λ.

$$A\vec{v} = \lambda\vec{v}$$

But now, suppose X is some $n \times d$ matrix. For any vector $\vec{v} \in \mathbb{R}^d$, the vector $X\vec{v}$ is in $\mathbb{R}^n$, **not** $\mathbb{R}^d$, meaning $\vec{v}$ and $X\vec{v}$ live in different universes. So, it can't make sense to say that $X\vec{v}$ results from stretching $\vec{v}$.

Instead, we'll find pairs of **singular vectors**, $\vec{v} \in \mathbb{R}^d$ and $\vec{u} \in \mathbb{R}^n$, such that

$$X\vec{v} = \sigma\vec{u}$$

where σ is a **singular value** of A (not a standard deviation!). Intuitively, this says that when X is multiplied by $\vec{v}$, the result is a scaled version of $\vec{u}$. These singular values and vectors are the focus of this chapter. Applications of the singular value decomposition will start in Chapter 10.2 – but go scroll to the bottom of that section if you want to see what we're working towards.

Think of $\vec{u}$, $\vec{v}$, and σ – or perhaps better $\vec{u}_i$, $\vec{v}_i$, and σ_i – as a trio of friends that appear together in the SVD of X; each matrix X has r such trios, where r is the rank of X.

I'll start by giving you the definition of the SVD, and then together we'll figure out where it came from.

Definition: Singular Value Decomposition

Suppose X is any $n \times d$ matrix (that is, **not necessarily square**). Then, there exists a **singular value decomposition** of X of the form

$$X = U\Sigma V^T$$

where:

- U is an $n \times n$ orthogonal matrix, whose columns are called the **left singular vectors** of X
- $\pm$ is an $n \times d$ diagonal matrix with non-negative real numbers on the diagonal and zeros elsewhere
- V is a $d \times d$ orthogonal matrix, whose columns are called the **right singular vectors** of X

The diagonal entries of $\pm$ are called the **singular values** of X. Typically, the singular values are sorted in decreasing order, i.e. $\sigma_1 \geq \sigma_2 \geq \cdots \geq \sigma_r > 0$, where $r = \text{rank}(X)$.

Firstly, note that the SVD exists, no matter what X is: it can be non-square, and it doesn't even need to be full rank. But note that unlike in the eigenvalue decomposition, where we decomposed A using just one eigenvector

matrix V and a diagonal matrix $*$, here we need to use two **singular vector** matrices U and V and a diagonal matrix $\pm$.

There's a lot of notation and new concepts above. Let's start with an example, then try and understand where each piece in $X = U\Sigma V^T$ comes from and means. Suppose

$$X = \begin{bmatrix} 3 & 2 & 5 \\ 2 & 3 & 5 \\ 2 & -2 & 0 \\ 5 & 5 & 10 \end{bmatrix}$$

X is a 4×3 matrix with $\text{rank}(X) = 2$, since its third column is the sum of the first two. Its singular value decomposition is given by $X = U\Sigma V^T$:

$$X = \underbrace{\begin{bmatrix} \frac{1}{\sqrt{6}} & \frac{1}{3\sqrt{2}} & -\frac{1}{\sqrt{3}} & -\frac{2}{3} \\ \frac{1}{\sqrt{6}} & -\frac{1}{3\sqrt{2}} & -\frac{1}{\sqrt{3}} & \frac{2}{3} \\ 0 & \frac{2\sqrt{2}}{3} & 0 & \frac{1}{3} \\ \frac{2}{\sqrt{6}} & 0 & \frac{1}{\sqrt{3}} & 0 \end{bmatrix}}_{U} \underbrace{\begin{bmatrix} 15 & 0 & 0 \\ 0 & 3 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix}}_{\Sigma} \underbrace{\begin{bmatrix} \frac{1}{\sqrt{6}} & \frac{1}{\sqrt{6}} & \frac{2}{\sqrt{6}} \\ \frac{1}{\sqrt{2}} & -\frac{1}{\sqrt{2}} & 0 \\ \frac{1}{\sqrt{3}} & \frac{1}{\sqrt{3}} & -\frac{1}{\sqrt{3}} \end{bmatrix}}_{V^T}$$

Two important observations:

1. U and V are both orthogonal matrices, meaning $U^TU = UU^T = I_{4\times4}$ and $V^TV = VV^T = I_{3\times3}$.
2. $\pm$ contains the singular values of X on the diagonal, arranged in decreasing order. We have that $\sigma_1 = 15$, $\sigma_2 = 3$, and $\sigma_3 = 0$. X has three singular values, but only two are non-zero. **In general, the number of non-zero singular values is equal to the rank of** X.

Where did all of these numbers come from?

Discovering the SVD

The SVD of X depends heavily on the matrices X^TX and XX^T. While X itself is 4×3,

- X^TX is a **symmetric** 3×3 **matrix**, containing the dot products of X's columns
- XX^T is a **symmetric** 4×4 **matrix**, containing the dot products of X's rows

Since X^TX and XX^T are both square matrices, they have eigenvalues and eigenvectors. And since they're both symmetric, their eigenvectors for different eigenvalues are orthogonal to each other, as the spectral theorem $A = Q\Lambda Q^T$ guarantees for any symmetric matrix A.

Singular Values and Singular Vectors. The singular value decomposition involves creatively using the eigenvalues and eigenvectors of X^TX and XX^T. Suppose $X = U\Sigma V^T$ is the SVD of X. Then, using the facts that $U^TU = I$ and $V^TV = I$, we have:

$$X^TX = (U\Sigma V^T)^T(U\Sigma V^T) = V\Sigma^T \underbrace{U^TU}_{I} \Sigma V^T = \underbrace{V\Sigma^T\Sigma V^T}_{\text{looks like } Q\Lambda Q^T}$$

$$XX^T = (U\Sigma V^T)(U\Sigma V^T)^T = U\Sigma \underbrace{V^T V}_{I} \Sigma^T U^T = \underbrace{U\Sigma\Sigma^T U^T}_{\text{looks like } P\Lambda P^T}$$

This just looks like we diagonalized X^TX and XX^T! The expressions above are saying that:

- V's columns are the eigenvectors of X^TX
- U's columns are the eigenvectors of XX^T

X^TX and XX^T usually have different sets of eigenvectors, which is why U and V are generally not the same matrix (they don't even have the same shape).

The eigen**values** of X^TX and XX^T **are the same**, though: those are the non-zero entries of $\Sigma^T\Sigma$ and $\Sigma\Sigma^T$. Since $\pm$ is an $n \times d$ matrix, $\Sigma^T\Sigma$ is a $d \times d$ matrix and $\Sigma\Sigma^T$ is an $n \times n$ matrix. But, when you work out both products, you'll notice that their non-zero values are the same.

Suppose for example that $\Sigma = \begin{bmatrix} \sigma_1 & 0 & 0 \\ 0 & \sigma_2 & 0 \\ 0 & 0 & \sigma_3 \\ 0 & 0 & 0 \end{bmatrix}$. Then,

$$\Sigma^T\Sigma = \begin{bmatrix} \sigma_1 & 0 & 0 & 0 \\ 0 & \sigma_2 & 0 & 0 \\ 0 & 0 & \sigma_3 & 0 \end{bmatrix} \begin{bmatrix} \sigma_1 & 0 & 0 \\ 0 & \sigma_2 & 0 \\ 0 & 0 & \sigma_3 \\ 0 & 0 & 0 \end{bmatrix} = \begin{bmatrix} \sigma_1^2 & 0 & 0 \\ 0 & \sigma_2^2 & 0 \\ 0 & 0 & \sigma_3^2 \end{bmatrix}$$

$$\Sigma\Sigma^T = \begin{bmatrix} \sigma_1 & 0 & 0 \\ 0 & \sigma_2 & 0 \\ 0 & 0 & \sigma_3 \\ 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} \sigma_1 & 0 & 0 & 0 \\ 0 & \sigma_2 & 0 & 0 \\ 0 & 0 & \sigma_3 & 0 \end{bmatrix} = \begin{bmatrix} \sigma_1^2 & 0 & 0 & 0 \\ 0 & \sigma_2^2 & 0 & 0 \\ 0 & 0 & \sigma_3^2 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix}$$

But, these matrices with the squared terms are precisely the $*$'s in the spectral decompositions of $X^TX = Q\Lambda Q^T$ and $XX^T = P\Lambda P^T$. This means that

$$\sigma_i^2 = \lambda_i \implies \sigma_i = \sqrt{\lambda_i}$$

where σ_i is a singular value of X and λ_i is an eigenvalue of X^TX or XX^T.

The above derivation is enough to justify that the eigenvalues of X^TX and XX^T are never negative, but for another perspective, note that both X^TX and XX^T are **positive semidefinite**, meaning their eigenvalues are non-negative. (You're wrestling with this fact in Lab 11 and Homework 10.)

The relationship between singular values/vectors and eigenvalues/vectors

To summarize: in $X = U\Sigma V^T$,

- The columns of U – called the **left singular vectors** of X – are the eigenvectors of XX^T
- The columns of V – called the **right singular vectors** of X – are the eigenvectors of X^TX
- The **singular values** of X, which live on the diagonal of $\pm$, are the **square roots of the eigenvalues** of X^TX and XX^T:

$$\sigma_i = \sqrt{\lambda_i}$$

Another proof that X^TX and XX^T have the same non-zero eigenvalues

Above, we implicitly used the fact that X^TX and XX^T have the same non-zero eigenvalues. Here's a proof of this fact that has nothing to do with the SVD.
Suppose $\vec{v}_i$ is an eigenvector of X^TX with eigenvalue λ_i.

$$X^TX\vec{v}_i = \lambda_i\vec{v}_i$$

What happens if we multiply both sides on the left by X?

$$XX^TX\vec{v}_i = X\lambda_i\vec{v}_i$$

Creatively adding parentheses gives us

$$XX^T(X\vec{v}_i) = \lambda_i(X\vec{v}_i)$$

This shows that $X\vec{v}_i$ is an eigenvector of XX^T with eigenvalue λ_i. The important thing is that the eigenvalue is shared. This logic can be reversed too, to show that if $\vec{u}_i$ is an eigenvector of XX^T with eigenvalue λ_i, then λ_i is also an eigenvalue of X^TX.

Computing the SVD. To find U, $\pm$, and V^T, we don't actually need to compute both X^TX and XX^T: all of these quantities can be uncovered just with one of them.

Let's return to our example, $X = \begin{bmatrix} 3 & 2 & 5 \\ 2 & 3 & 5 \\ 2 & -2 & 0 \\ 5 & 5 & 10 \end{bmatrix}$. Using what we've just learned, let's find U, $\pm$, and V^T ourselves. X^TX has fewer entries than XX^T, so let's start with it. I will delegate *some* number crunching to `numpy`.

```
import numpy as np

X = np.array([[3, 2, 5],
              [2, 3, 5],
              [2, -2, 0],
              [5, 5, 10]])
X.T @ X
```

```
array([[ 42,  33,  75],
       [ 33,  42,  75],
       [ 75,  75, 150]])
```

X^TX is a 3×3 matrix, but its rank is 2, meaning it will have an eigenvalue of 0. What are its other eigenvalues?

```
eigvals, eigvecs = np.linalg.eig(X.T @ X)
eigvals
```

```
array([225.,   9.,   0.])
```

The eigenvalues of $X^T X$ are 225, 9, and 0. This tells us that the singular values of X are $\sqrt{225} = 15$, $\sqrt{9} = 3$, and 0. So far, we've discovered that $\Sigma = \begin{bmatrix} 15 & 0 & 0 \\ 0 & 3 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix}$. Remember that Σ always has the same shape as X (both are $n \times d$), and all of its entries are 0 except for the singular values, which are arranged in decreasing order on the diagonal, starting with the largest singular value in the top left corner.

Let's now find the eigenvectors of $X^T X$ – that is, the right singular vectors of X – which we should store in V. **We expect the eigenvectors $\vec{v}_i$ of $X^T X$ to be orthogonal**, since $X^T X$ is symmetric.

```
X.T @ X
```

```
array([[ 42,  33,  75],
       [ 33,  42,  75],
       [ 75,  75, 150]])
```

- For $\lambda_1 = 225$, one eigenvector is $\vec{v}_1 = \begin{bmatrix} 1 \\ 1 \\ 2 \end{bmatrix}$, since

$$\underbrace{\begin{bmatrix} 42 & 33 & 75 \\ 33 & 42 & 75 \\ 75 & 75 & 150 \end{bmatrix}}_{X^T X} \begin{bmatrix} 1 \\ 1 \\ 2 \end{bmatrix} = \begin{bmatrix} 225 \\ 225 \\ 450 \end{bmatrix} = 225 \begin{bmatrix} 1 \\ 1 \\ 2 \end{bmatrix}$$

- For $\lambda_2 = 9$, one eigenvector is $\vec{v}_2 = \begin{bmatrix} 1 \\ -1 \\ 0 \end{bmatrix}$, since

$$\begin{bmatrix} 42 & 33 & 75 \\ 33 & 42 & 75 \\ 75 & 75 & 150 \end{bmatrix} \begin{bmatrix} 1 \\ -1 \\ 0 \end{bmatrix} = \begin{bmatrix} 9 \\ -9 \\ 0 \end{bmatrix} = 9 \begin{bmatrix} 1 \\ -1 \\ 0 \end{bmatrix}$$

- For $\lambda_3 = 0$, one eigenvector is $\vec{v}_3 = \begin{bmatrix} 1 \\ 1 \\ -1 \end{bmatrix}$, since

$$\begin{bmatrix} 42 & 33 & 75 \\ 33 & 42 & 75 \\ 75 & 75 & 150 \end{bmatrix} \begin{bmatrix} 1 \\ 1 \\ -1 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix}$$

Note that $\vec{v}_3$ is a basis for nullsp(X), since dim(nullsp(X)) = 1 and $X\vec{v}_3 = \vec{0}$. Hold that thought for now.

To create V, all we need to do is turn $\vec{v}_1$, $\vec{v}_2$, and $\vec{v}_3$ into unit vectors.

$$\begin{bmatrix} 1 \\ 1 \\ 2 \end{bmatrix} \to \underbrace{\begin{bmatrix} \frac{1}{\sqrt{6}} \\ \frac{1}{\sqrt{6}} \\ \frac{2}{\sqrt{6}} \end{bmatrix}}_{\vec{v}_1}, \quad \begin{bmatrix} 1 \\ -1 \\ 0 \end{bmatrix} \to \underbrace{\begin{bmatrix} \frac{1}{\sqrt{2}} \\ -\frac{1}{\sqrt{2}} \\ 0 \end{bmatrix}}_{\vec{v}_2}, \quad \begin{bmatrix} 1 \\ 1 \\ -1 \end{bmatrix} \to \underbrace{\begin{bmatrix} \frac{1}{\sqrt{3}} \\ \frac{1}{\sqrt{3}} \\ -\frac{1}{\sqrt{3}} \end{bmatrix}}_{\vec{v}_3}$$

Stacking these unit vectors together gives us V:

$$V = \begin{bmatrix} \frac{1}{\sqrt{6}} & \frac{1}{\sqrt{2}} & \frac{1}{\sqrt{3}} \\ \frac{1}{\sqrt{6}} & -\frac{1}{\sqrt{2}} & \frac{1}{\sqrt{3}} \\ \frac{2}{\sqrt{6}} & 0 & -\frac{1}{\sqrt{3}} \end{bmatrix}$$

And indeed, since X^TX is symmetric, V is orthogonal: $V^TV = VV^T = I_{3\times3}$.

Great! We're almost done computing the SVD. So far, we have

$$\underbrace{\begin{bmatrix} 3 & 2 & 5 \\ 2 & 3 & 5 \\ 2 & -2 & 0 \\ 5 & 5 & 10 \end{bmatrix}}_{X} = U \underbrace{\begin{bmatrix} 15 & 0 & 0 \\ 0 & 3 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix}}_{\Sigma} \underbrace{\begin{bmatrix} \frac{1}{\sqrt{6}} & \frac{1}{\sqrt{6}} & \frac{2}{\sqrt{6}} \\ \frac{1}{\sqrt{2}} & -\frac{1}{\sqrt{2}} & 0 \\ \frac{1}{\sqrt{3}} & \frac{1}{\sqrt{3}} & -\frac{1}{\sqrt{3}} \end{bmatrix}}_{V^T}$$

$XV = U\Sigma$ and $X\vec{v}_i = \sigma_i\vec{u}_i$. Ideally, we can avoid having to compute the eigenvectors of XX^T to stack into U. And we can. If we start with

$$X = U\Sigma V^T$$

and multiply both sides on the right by V, we uncover a relationship between the columns of U and the columns of V.

$$XV = U\Sigma$$

Let's unpack this. On the left, the matrix XV is made up of multiplying X by each column of V.

$$XV = X_{4\times3}\begin{bmatrix} | & | & | \\ \vec{v}_1 & \vec{v}_2 & \vec{v}_3 \\ | & | & | \end{bmatrix}_{3\times3} = \begin{bmatrix} | & | & | \\ X\vec{v}_1 & X\vec{v}_2 & X\vec{v}_3 \\ | & | & | \end{bmatrix}_{4\times3}$$

On the right, $U\Sigma$ is made up of stretching each column of U by the corresponding singular value in the diagonal of ±.

$$U\Sigma = \begin{bmatrix} | & | & | & | \\ \vec{u}_1 & \vec{u}_2 & \vec{u}_3 & \vec{u}_4 \\ | & | & | & | \end{bmatrix}_{4\times4}\begin{bmatrix} \sigma_1 & 0 & 0 \\ 0 & \sigma_2 & 0 \\ 0 & 0 & \sigma_3 \\ 0 & 0 & 0 \end{bmatrix}_{4\times3} = \begin{bmatrix} | & | & | \\ \sigma_1\vec{u}_1 & \sigma_2\vec{u}_2 & \sigma_3\vec{u}_3 \\ | & | & | \end{bmatrix}_{4\times3}$$

But, since $XV = U\Sigma$, we have

$$\begin{bmatrix} | & | & | \\ X\vec{v}_1 & X\vec{v}_2 & X\vec{v}_3 \\ | & | & | \end{bmatrix} = \begin{bmatrix} | & | & | \\ \sigma_1\vec{u}_1 & \sigma_2\vec{u}_2 & \sigma_3\vec{u}_3 \\ | & | & | \end{bmatrix}$$

Singular values represent stretches, too!

A consequence of the above relationship is that if X is $n \times d$, then for $i = 1, 2, \ldots, d$,

$$X\vec{v}_i = \sigma_i \vec{u}_i$$

The above is saying that when X is multiplied by $\vec{v}_i$, the result is a scaled version of $\vec{u}_i$ (not $\vec{v}_i$)! As we said at the start of the section, $X\vec{v}_i$ lives in a different universe ($\mathbb{R}^n$) than $\vec{v}_i$ does ($\mathbb{R}^d$), so $\vec{v}_i$ and $X\vec{v}_i$ can't point in the same direction; instead, $X\vec{v}_i$ points in the same direction as $\vec{u}_i$, which lives in $\mathbb{R}^n$. I like to think of each $\vec{v}_i$ having a "partner" $\vec{u}_i$; multiplying X by $\vec{v}_i$ results in stretching $\vec{u}_i$.
$X\vec{v}_i = \sigma_i\vec{u}_i$ is the singular value/vector analog of $A\vec{v} = \lambda\vec{v}$ for eigenvalues/eigenvectors.

Crucially though, $X\vec{v}_i = \sigma_i\vec{u}_i$ only holds when we've arranged the singular values and vectors in U, $\pm$, and V^T consistently. This is one reason why we always arrange the singular values in $\pm$ in decreasing order.

Back to our example. Again, we currently have

$$\underbrace{\begin{bmatrix} 3 & 2 & 5 \\ 2 & 3 & 5 \\ 2 & -2 & 0 \\ 5 & 5 & 10 \end{bmatrix}}_{X} = U \underbrace{\begin{bmatrix} 15 & 0 & 0 \\ 0 & 3 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix}}_{\Sigma} \underbrace{\begin{bmatrix} \frac{1}{\sqrt{6}} & \frac{1}{\sqrt{6}} & \frac{2}{\sqrt{6}} \\ \frac{1}{\sqrt{2}} & -\frac{1}{\sqrt{2}} & 0 \\ \frac{1}{\sqrt{3}} & \frac{1}{\sqrt{3}} & -\frac{1}{\sqrt{3}} \end{bmatrix}}_{V^T}$$

We know X, and we know each $\vec{v}_i$ and σ_i. Rearranging $X\vec{v}_i = \sigma_i\vec{u}_i$ gives us

$$\vec{u}_i = \frac{1}{\sigma_i} X\vec{v}_i$$

which is a recipe for computing $\vec{u}_1$, $\vec{u}_2$, and $\vec{u}_3$.

1. $\vec{u}_1 = \frac{1}{15} X \begin{bmatrix} \frac{1}{\sqrt{6}} \\ \frac{1}{\sqrt{6}} \\ \frac{2}{\sqrt{6}} \end{bmatrix} = \begin{bmatrix} \frac{1}{\sqrt{6}} \\ \frac{1}{\sqrt{6}} \\ 0 \\ \frac{2}{\sqrt{6}} \end{bmatrix}$
2. $\vec{u}_2 = \frac{1}{3} X \begin{bmatrix} \frac{1}{\sqrt{2}} \\ -\frac{1}{\sqrt{2}} \\ 0 \end{bmatrix} = \begin{bmatrix} \frac{1}{3\sqrt{2}} \\ -\frac{1}{3\sqrt{2}} \\ \frac{2\sqrt{2}}{3} \\ 0 \end{bmatrix}$
3. $\vec{u}_3 = \frac{1}{0} X \begin{bmatrix} \frac{1}{\sqrt{3}} \\ \frac{1}{\sqrt{3}} \\ -\frac{1}{\sqrt{3}} \end{bmatrix}$ = ...wait, what?

Something is not quite right: $\sigma_3 = 0$, which means we can't use $\vec{u}_i = \frac{1}{\sigma_i} X\vec{v}_i$ to compute $\vec{u}_3$. And, even if we could, this recipe tells us nothing about $\vec{u}_4$, which we also need to find, since U is 4×4.

Null Spaces Return. So far, we've found two of the four columns of U.

$$U = \begin{bmatrix} \frac{1}{\sqrt{6}} & \frac{1}{3\sqrt{2}} & ? & ? \\ \frac{1}{\sqrt{6}} & -\frac{1}{3\sqrt{2}} & ? & ? \\ 0 & \frac{2\sqrt{2}}{3} & ? & ? \\ \frac{2}{\sqrt{6}} & 0 & ? & ? \end{bmatrix}$$

The issue is that we don't have a recipe for what $\vec{u}_3$ and $\vec{u}_4$ should be. This problem stems from the fact that $\text{rank}(X) = 2$.

$$X = \begin{bmatrix} 3 & 2 & 5 \\ 2 & 3 & 5 \\ 2 & -2 & 0 \\ 5 & 5 & 10 \end{bmatrix}$$

If we were to compute XX^T, whose eigenvectors are the columns of U, we would have seen that it has an eigenvalue of 0 with geometric (and algebraic) multiplicity 2.

```
X @ X.T
```

```
array([[ 38,  37,   2,  75],
       [ 37,  38,  -2,  75],
       [  2,  -2,   8,   0],
       [ 75,  75,   0, 150]])
```

```
eigvals, eigvecs = np.linalg.eig(X @ X.T)
eigvals
```

```
array([225.,   9.,  -0.,   0.])
```

So, all we need to do is fill $\vec{u}_3$ and $\vec{u}_4$ with two orthonormal vectors that span the eigenspace of XX^T for eigenvalue 0. $\vec{u}_1$ is already an eigenvector for eigenvalue 225 and $\vec{u}_2$ is already an eigenvector for eigenvalue 9. Bringing in $\vec{u}_3$ and $\vec{u}_4$ will complete U, making it a **basis** for $\mathbb{R}^4$, which it needs to be to be an invertible square matrix. (Remember, U must satisfy $U^TU = UU^T = I$, which means U is invertible.)

But, the eigenspace of XX^T for eigenvalue 0 is nullsp(XX^T)!

Since XX^T is symmetric, $\vec{u}_3$ and $\vec{u}_4$ will be orthogonal to $\vec{u}_1$ and $\vec{u}_2$ no matter what, since the eigenvectors for different eigenvalues of a symmetric matrix are always orthogonal. (Another perspective on why this is true is that $\vec{u}_3$ and $\vec{u}_4$ will be in nullsp(XX^T) which is equivalent to nullsp(X^T), and any vector in nullsp(X^T) is orthogonal to any vector in colsp(X), which $\vec{u}_1$ and $\vec{u}_2$ are in. There's a long chain of reasoning that leads to this conclusion: make sure you're familiar with it.)

Observe that

- $\begin{bmatrix} -1 \\ -1 \\ 0 \\ 1 \end{bmatrix}$ is in nullsp(XX^T), since $\begin{bmatrix} 38 & 37 & 2 & 75 \\ 37 & 38 & -2 & 75 \\ 2 & -2 & 8 & 0 \\ 75 & 75 & 0 & 150 \end{bmatrix} \begin{bmatrix} -1 \\ -1 \\ 0 \\ 1 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \end{bmatrix}$
- $\begin{bmatrix} -2 \\ 2 \\ 1 \\ 0 \end{bmatrix}$ is in nullsp(XX^T), since $\begin{bmatrix} 38 & 37 & 2 & 75 \\ 37 & 38 & -2 & 75 \\ 2 & -2 & 8 & 0 \\ 75 & 75 & 0 & 150 \end{bmatrix} \begin{bmatrix} -2 \\ 2 \\ 1 \\ 0 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \end{bmatrix}$

The vectors $\begin{bmatrix}-1\\-1\\0\\1\end{bmatrix}$ and $\begin{bmatrix}-2\\2\\1\\0\end{bmatrix}$ are orthogonal to each other, so they're good candidates for $\vec{u}_3$ and $\vec{u}_4$, we just need to normalize them first.

$$\begin{bmatrix}-1\\-1\\0\\1\end{bmatrix} \to \underbrace{\begin{bmatrix}-\frac{1}{\sqrt{3}}\\-\frac{1}{\sqrt{3}}\\0\\\frac{1}{\sqrt{3}}\end{bmatrix}}_{\vec{u}_3}, \quad \begin{bmatrix}-2\\2\\1\\0\end{bmatrix} \to \underbrace{\begin{bmatrix}-\frac{2}{3}\\\frac{2}{3}\\\frac{1}{3}\\0\end{bmatrix}}_{\vec{u}_4}$$

Before we place $\vec{u}_3$ and $\vec{u}_4$ in U, it's worth noticing that $\vec{u}_3$ and $\vec{u}_4$ are both in $\text{nullsp}(X^T)$ too.

$$X^T\vec{u}_3 = \begin{bmatrix}3&2&2&5\\2&3&-2&5\\5&5&0&10\end{bmatrix}\begin{bmatrix}-\frac{1}{\sqrt{3}}\\-\frac{1}{\sqrt{3}}\\0\\\frac{1}{\sqrt{3}}\end{bmatrix} = \begin{bmatrix}0\\0\\0\end{bmatrix}$$

$$X^T\vec{u}_4 = \begin{bmatrix}3&2&2&5\\2&3&-2&5\\5&5&0&10\end{bmatrix}\begin{bmatrix}-\frac{2}{3}\\\frac{2}{3}\\\frac{1}{3}\\0\end{bmatrix} = \begin{bmatrix}0\\0\\0\end{bmatrix}$$

In fact, remember from Chapter 5.4 that

$$\text{nullsp}(X^T) = \text{nullsp}(XX^T)\text{nullsp}(X) = \text{nullsp}(X^TX)$$

So, we have

$$U = \begin{bmatrix}\frac{1}{\sqrt{6}}&\frac{1}{3\sqrt{2}}&-\frac{1}{\sqrt{3}}&-\frac{2}{3}\\\frac{1}{\sqrt{6}}&-\frac{1}{3\sqrt{2}}&-\frac{1}{\sqrt{3}}&\frac{2}{3}\\0&\frac{2\sqrt{2}}{3}&0&\frac{1}{3}\\\frac{2}{\sqrt{6}}&0&\frac{1}{\sqrt{3}}&0\end{bmatrix}$$

And finally, we have computed the SVD of X!

$$\underbrace{\begin{bmatrix}3&2&5\\2&3&5\\2&-2&0\\5&5&10\end{bmatrix}}_{X} = \underbrace{\begin{bmatrix}\frac{1}{\sqrt{6}}&\frac{1}{3\sqrt{2}}&-\frac{1}{\sqrt{3}}&-\frac{2}{3}\\\frac{1}{\sqrt{6}}&-\frac{1}{3\sqrt{2}}&-\frac{1}{\sqrt{3}}&\frac{2}{3}\\0&\frac{2\sqrt{2}}{3}&0&\frac{1}{3}\\\frac{2}{\sqrt{6}}&0&\frac{1}{\sqrt{3}}&0\end{bmatrix}}_{U}\underbrace{\begin{bmatrix}15&0&0\\0&3&0\\0&0&0\\0&0&0\end{bmatrix}}_{\Sigma}\underbrace{\begin{bmatrix}\frac{1}{\sqrt{6}}&\frac{1}{\sqrt{6}}&\frac{2}{\sqrt{6}}\\\frac{1}{\sqrt{2}}&-\frac{1}{\sqrt{2}}&0\\\frac{1}{\sqrt{3}}&\frac{1}{\sqrt{3}}&-\frac{1}{\sqrt{3}}\end{bmatrix}}_{V^T}$$

For the most part, we will use `numpy` to compute the SVD of a matrix. But, it's important to understand the steps we took to compute the SVD manually.

Computing the SVD By Hand

To conclude, we found $X = U\Sigma V^T$ by:

1. Computing $X^T X$.
2. Finding the eigenvalues of $X^T X$; their square roots are the singular values of X.

$$\sigma_i = \sqrt{\lambda_i}$$

   These singular values are placed in the diagonal of $\pm$ in decreasing order.

3. For each eigenvalue λ_i, finding an orthonormal eigenvector $\vec{v}_i$ of $X^T X$ and placing it in the ith column of V.
4. For each $i = 1, 2, ..., r$, finding $\vec{u}_i$ by solving

$$\vec{u}_i = \frac{1}{\sigma_i} X\vec{v}_i$$

   and placing it in the ith column of U.

5. Filling the rest of U with orthonormal vectors that form a basis for nullsp(X^T).

This is not the only possible sequence of steps to follow; for instance, once you find the singular values $\sigma_1, \sigma_2, ..., \sigma_r$, you can independently find orthonormal eigenvectors of $X^T X$ and XX^T and use them to form U and V. Just make sure you place the $\vec{u}_i$'s and $\vec{v}_i$'s in the correct order, corresponding to the order of the singular values in Σ.

Here's a summary of everything we've gathered so far.

Examples

Let's work through a few computational problems. Most of our focus on the SVD in this class is conceptual, but it's important to have some baseline computational fluency.

Example: SVD of a 2×2 Matrix. We developed the SVD to decompose non-square matrices. But, it still works for square matrices too. Find the SVD of $X = \begin{bmatrix} 5 & 4 \\ 0 & 3 \end{bmatrix}$.

Solution

Let's follow the six steps introduced in the "Computing the SVD By Hand" box above.

1. $X^TX = \begin{bmatrix} 5 & 0 \\ 4 & 3 \end{bmatrix}\begin{bmatrix} 5 & 4 \\ 0 & 3 \end{bmatrix} = \begin{bmatrix} 25 & 20 \\ 20 & 25 \end{bmatrix}$

2. Now, we need to find the eigenvalues and eigenvectors of X^TX. Notice that the sum of X^TX's rows are both the same, imploring me to verify that $\begin{bmatrix} 1 \\ 1 \end{bmatrix}$ is an eigenvector:

$$X^TX\begin{bmatrix} 1 \\ 1 \end{bmatrix} = \begin{bmatrix} 25 & 20 \\ 20 & 25 \end{bmatrix}\begin{bmatrix} 1 \\ 1 \end{bmatrix} = \begin{bmatrix} 45 \\ 45 \end{bmatrix} = 45\begin{bmatrix} 1 \\ 1 \end{bmatrix}$$

   So, $\begin{bmatrix} 1 \\ 1 \end{bmatrix}$ is an eigenvector for eigenvalue 45. Before placing it into V, I'll need to turn it into a unit vector; since the norm of $\begin{bmatrix} 1 \\ 1 \end{bmatrix}$ is $\sqrt{2}$, I'll place $\begin{bmatrix} \frac{1}{\sqrt{2}} \\ \frac{1}{\sqrt{2}} \end{bmatrix}$ into V. Since X^TX is symmetric, its other eigenvector will be orthogonal to $\begin{bmatrix} 1 \\ 1 \end{bmatrix}$, so I'll choose $\begin{bmatrix} -1 \\ 1 \end{bmatrix}$. This works because in $\mathbb{R}^2$, there's only one direction orthogonal to any given vector. This corresponds to the eigenvalue

$$X^TX\begin{bmatrix} -1 \\ 1 \end{bmatrix} = \begin{bmatrix} 25 & 20 \\ 20 & 25 \end{bmatrix}\begin{bmatrix} -1 \\ 1 \end{bmatrix} = \begin{bmatrix} -5 \\ 5 \end{bmatrix} = 5\begin{bmatrix} -1 \\ 1 \end{bmatrix}$$

   So, $\begin{bmatrix} -1 \\ 1 \end{bmatrix}$ – or, equivalently, $\begin{bmatrix} \frac{-1}{\sqrt{2}} \\ \frac{1}{\sqrt{2}} \end{bmatrix}$ – is an eigenvector for eigenvalue 5.

3. I now have enough information to form $\pm$ and V:

$$\pm = \begin{bmatrix} \sqrt{45} & 0 \\ 0 & \sqrt{5} \end{bmatrix}, \quad V = \begin{bmatrix} \frac{1}{\sqrt{2}} & -\frac{1}{\sqrt{2}} \\ \frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} \end{bmatrix}$$

4. To form U, I need to find two orthonormal vectors that are eigenvectors of XX^T for eigenvalues 45 and 5. The easy way is to use the fact that

$$\vec{u}_i = \frac{1}{\sigma_i}X\vec{v}_i$$

   Applying this to $i = 1$ and $i = 2$ gives us

   - $\vec{u}_1 = \frac{1}{\sigma_1}X\vec{v}_i = \frac{1}{\sqrt{45}}\begin{bmatrix} 5 & 4 \\ 0 & 3 \end{bmatrix}\begin{bmatrix} \frac{1}{\sqrt{2}} \\ \frac{1}{\sqrt{2}} \end{bmatrix} = \begin{bmatrix} \frac{9}{\sqrt{90}} \\ \frac{3}{\sqrt{90}} \end{bmatrix} = \begin{bmatrix} \frac{3}{\sqrt{10}} \\ \frac{1}{\sqrt{10}} \end{bmatrix}$
   - $\vec{u}_2 = \frac{1}{\sigma_2}X\vec{v}_2 = \frac{1}{\sqrt{5}}\begin{bmatrix} 5 & 4 \\ 0 & 3 \end{bmatrix}\begin{bmatrix} \frac{-1}{\sqrt{2}} \\ \frac{1}{\sqrt{2}} \end{bmatrix} = \begin{bmatrix} \frac{-1}{\sqrt{10}} \\ \frac{3}{\sqrt{10}} \end{bmatrix}$ Note that I could have also used the fact that I knew $\vec{u}_2$ would be orthogonal to $\vec{u}_1$ to find it, since $\vec{u}_1$ and $\vec{u}_2$ both live in $\mathbb{R}^2$.

5. U is 2×2, so I have all I need:

$$U = \begin{bmatrix} \frac{3}{\sqrt{10}} & \frac{-1}{\sqrt{10}} \\ \frac{1}{\sqrt{10}} & \frac{3}{\sqrt{10}} \end{bmatrix}$$

So, the singular value decomposition of X is

$$\begin{bmatrix} 5 & 4 \\ 0 & 3 \end{bmatrix} = \begin{bmatrix} \frac{3}{\sqrt{10}} & -\frac{1}{\sqrt{10}} \\ \frac{1}{\sqrt{10}} & \frac{3}{\sqrt{10}} \end{bmatrix}\begin{bmatrix} 3\sqrt{5} & 0 \\ 0 & \sqrt{5} \end{bmatrix}\begin{bmatrix} \frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} \\ -\frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} \end{bmatrix}$$

Example: SVD of a Wide Matrix. Find the SVD of $X = \begin{bmatrix} 3 & 1 & 0 \\ 4 & 0 & 1 \end{bmatrix}$.

Solution

If I were to start with X^TX, I'd have to compute a 3×3 matrix. Instead, I'll start with XX^T, which is 2×2. This will mean filling in the columns of U before I fill in the columns of V, which will require a slightly different approach.

$$XX^T = \begin{bmatrix} 3 & 1 & 0 \\ 4 & 0 & 1 \end{bmatrix} \begin{bmatrix} 3 & 4 \\ 1 & 0 \\ 0 & 1 \end{bmatrix} = \begin{bmatrix} 10 & 12 \\ 12 & 17 \end{bmatrix}$$

XX^T's eigenvalues must add to $10 + 17 = 27$ and multiply to $10 \cdot 17 - 12^2 = 26$. This quickly tells me its eigenvalues are 26 and 1.

- For $\lambda = 26$, any eigenvector is of the form $\begin{bmatrix} a \\ b \end{bmatrix}$ where

  $$\begin{bmatrix} 10 & 12 \\ 12 & 17 \end{bmatrix} \begin{bmatrix} a \\ b \end{bmatrix} = 26 \begin{bmatrix} a \\ b \end{bmatrix}$$

  This implies that $10a + 12b = 26a$, i.e. $16a = 12b$, or $a = \frac{3}{4}b$. Eventually we'll need to normalize the vector, but for now I'll pick $b = 4$ so that $a = 3$ and so $\begin{bmatrix} 3 \\ 4 \end{bmatrix}$ is an eigenvector.

- For $\lambda = 1$, the eigenvector needs to be orthogonal to $\begin{bmatrix} 3 \\ 4 \end{bmatrix}$, which is easy to solve for when working in $\mathbb{R}^2$, since there's only one possible direction for a vector orthogonal to $\begin{bmatrix} 3 \\ 4 \end{bmatrix}$ in $\mathbb{R}^2$. The easy choice is $\begin{bmatrix} -4 \\ 3 \end{bmatrix}$.

I now have enough information to form both U and $\pm$:

$$U = \begin{bmatrix} 3/5 & -4/5 \\ 4/5 & 3/5 \end{bmatrix}, \quad \Sigma = \begin{bmatrix} \sqrt{26} & 0 & 0 \\ 0 & 1 & 0 \end{bmatrix}$$

The extra column of 0s is to account for the fact that X is 2×3 and Σ has the same shape as X. Everything but the non-zero singular values along the diagonal of Σ is 0.

Finally, how do we find the columns of the 3×3 matrix V? The equation we used in earlier examples, $X\vec{v}_i = \sigma_i \vec{u}_i$, is not useful here because it would involve solving a system of three equations and three unknowns for each $\vec{v}_i$, since its the unknown vector $\vec{v}_i$ that is being multiplied by X. That equation works only when you **first** find the eigenvectors of X^TX (the columns of V). We're proceeding in a different order, so we need a different equation. (There's always the fallback of using the fact that we know that X^TX's non-zero eigenvalues are 26 and 1 and using them to solve for the eigenvectors, but let's think of another approach.)

The parallel equation to $X\vec{v}_i = \sigma_i \vec{u}_i$ is $\vec{X}^T\vec{u}_i = \sigma_i \vec{v}_i$. Where did this come from? Start with

$$X\vec{v}_i = \sigma_i \vec{u}_i$$

and multiply both sides on the left by X^T:

$$X^TX\vec{v}_i = X^T\sigma_i \vec{u}_i$$

But, $\vec{v}_i$ is an eigenvector of X^TX with eigenvalue σ_i^2, so replace $X^TX\vec{v}_i$ with $\sigma_i^2\vec{v}_i$:

$$\sigma_i^2\vec{v}_i = X^T\sigma_i\vec{u}_i \implies X^T\vec{u}_i = \sigma_i\vec{v}_i \implies \vec{v}_i = \frac{1}{\sigma_i}X^T\vec{u}_i$$

This gives us recipes for the first and second columns of the 3×3 matrix V:

Example: SVD of a Positive Semidefinite Matrix. Recall, a positive semidefinite matrix is a square, symmetric matrix whose eigenvalues are non-negative.

Consider the positive semidefinite matrix $X = \begin{bmatrix} 1 & 2 \\ 2 & 4 \end{bmatrix}$. Find the SVD of X. How does it compare to the spectral decomposition of X into $Q\Lambda Q^T$?

Solution

Let's find X's singular value decomposition first. To prevent this section from being too long, I'll do this relatively quickly:

- $X^TX = \begin{bmatrix} 5 & 10 \\ 10 & 20 \end{bmatrix}$, which has eigenvalues 25 and 0 (which we can spot quickly because X and X^TX are not invertible, so they have an eigenvalue of 0).
- For $\lambda = 0$, the corresponding eigenvector direction is $\begin{bmatrix} 2 \\ -1 \end{bmatrix}$.
- So, for $\lambda = 25$, the eigenvector direction is $\begin{bmatrix} 1 \\ 2 \end{bmatrix}$.

So, the $\pm$ and V are

$$\pm = \begin{bmatrix} \sqrt{25} & 0 \\ 0 & 0 \end{bmatrix}, \quad V = \begin{bmatrix} 1/\sqrt{5} & 2/\sqrt{5} \\ 2/\sqrt{5} & -1/\sqrt{5} \end{bmatrix}$$

What's left is U, which has the eigenvectors of XX^T. It is tempting to just say that XX^T and X^TX are the same matrix and so they have the same eigenvectors, so $U = V$. This happens to work here, but this is a bad habit to get into since it can lead us into trouble in other similar examples. (For example, if $\vec{u}_1$ is an column of U and $\vec{v}_1$ is a column of V, we could replace each with $-\vec{u}_1$ and $-\vec{v}_1$, respectively, and keep the same SVD. But if we were to find the columns of U totally independently from the columns of V, we wouldn't know which sign to use).
Instead, let's use $\vec{u}_i = \frac{1}{\sigma_i} X\vec{v}_i$ to find the columns of U, to ensure the correspondence between each $\vec{u}_i$ and its partner $\vec{v}_i$.

- $\vec{u}_1 = \frac{1}{\sigma_1} X\vec{v}_1 = \frac{1}{5}\begin{bmatrix} 1 & 2 \\ 2 & 4 \end{bmatrix}\begin{bmatrix} 1/\sqrt{5} \\ 2/\sqrt{5} \end{bmatrix} = \begin{bmatrix} 1/\sqrt{5} \\ 2/\sqrt{5} \end{bmatrix}$.
- $\vec{u}_2 = \frac{1}{\sigma_2} X\vec{v}_2$ can't be evaluated since $\sigma_2 = 0$, but we know $\vec{u}_2$ must be orthogonal to $\vec{u}_1$, so pick $\begin{bmatrix} 2/\sqrt{5} \\ -1/\sqrt{5} \end{bmatrix}$.

So, the SVD of X is

$$\underbrace{\begin{bmatrix} 1 & 2 \\ 2 & 4 \end{bmatrix}}_{X} = \underbrace{\begin{bmatrix} 1/\sqrt{5} & 2/\sqrt{5} \\ 2/\sqrt{5} & -1/\sqrt{5} \end{bmatrix}}_{U} \underbrace{\begin{bmatrix} 5 & 0 \\ 0 & 0 \end{bmatrix}}_{\pm} \underbrace{\begin{bmatrix} 1/\sqrt{5} & 2/\sqrt{5} \\ 2/\sqrt{5} & -1/\sqrt{5} \end{bmatrix}}_{V^T}$$

This is exactly the same as the spectral decomposition of X into $Q\Lambda Q^T$, just with $\Sigma = \Lambda$ and $U = V$!
Don't try and make to broad of a generalization of this: when matrices are square, generally their SVD $X = U\Sigma V^T$ is different from their eigenvalue decomposition $X = V\Lambda V^{-1}$, as we saw with $\begin{bmatrix} 5 & 4 \\ 0 & 3 \end{bmatrix}$. Even when X is symmetric, its SVD and spectral decomposition don't need to be the same. A key aspect of the equivalence in this example is that X is positive semidefinite, meaning its eigenvalues – 5 and 0 (verify this yourself, it wasn't part of our solution above) – are non-negative. If they were negative, then the SVD and spectral decompositions would be different, as we're about to see.

Example: SVD of a Symmetric Matrix with Negative Eigenvalues. Let $X = \begin{bmatrix} 2 & 3 \\ 3 & 2 \end{bmatrix}$. Find the SVD of X. How do its singular values compare to its eigenvalues?
(Remember that only square matrices have eigenvalues, but all matrices have singular values.)

Solution

X is a symmetric matrix, but it is not positive semidefinite like in the last example, since one of its eigenvalues is negative. Its eigenvalues are 5 and -1, corresponding to eigenvectors $\begin{bmatrix}1\\1\end{bmatrix}$ and $\begin{bmatrix}-1\\1\end{bmatrix}$. X's spectral decomposition, then, is

$$X = \underbrace{\begin{bmatrix}1/\sqrt{2} & -1/\sqrt{2}\\ 1/\sqrt{2} & 1/\sqrt{2}\end{bmatrix}}_{Q}\underbrace{\begin{bmatrix}5 & 0\\ 0 & -1\end{bmatrix}}_{\Lambda}\underbrace{\begin{bmatrix}1/\sqrt{2} & 1/\sqrt{2}\\ -1/\sqrt{2} & 1/\sqrt{2}\end{bmatrix}}_{Q^T}$$

So, we know X's singular values can't be the same as its eigenvalues, as singular values can't be negative. What are X's singular values? They come from the square rooting the eigenvalues of X^TX, which is the same as XX^T:

$$X^TX = XX^T = \begin{bmatrix}13 & 12\\ 12 & 13\end{bmatrix}$$

X^TX has eigenvalues 25 and 1, which means X's singular values are 5 and 1 (not -1), with corresponding eigenvectors $\begin{bmatrix}1/\sqrt{2}\\1/\sqrt{2}\end{bmatrix}$ and $\begin{bmatrix}-1/\sqrt{2}\\1/\sqrt{2}\end{bmatrix}$. So, we have

$$\Sigma = \begin{bmatrix}5 & 0\\ 0 & 1\end{bmatrix}, \quad V = \begin{bmatrix}1/\sqrt{2} & -1/\sqrt{2}\\ 1/\sqrt{2} & 1/\sqrt{2}\end{bmatrix}$$

Finally, we need to find U. Again, it's tempting to want to say that $X^TX = XX^T$ so $U = V$, **but if we did that, our U would be wrong!** Instead, we need to use the one-at-a-time relationship $\vec{u}_i = \frac{1}{\sigma_i}X\vec{v}_i$ to find the columns of U.

- $\vec{u}_1 = \frac{1}{\sigma_1}X\vec{v}_1 = \frac{1}{5}\begin{bmatrix}2 & 3\\ 3 & 2\end{bmatrix}\begin{bmatrix}1/\sqrt{2}\\1/\sqrt{2}\end{bmatrix} = \begin{bmatrix}1/\sqrt{2}\\1/\sqrt{2}\end{bmatrix}$
- $\vec{u}_2 = \frac{1}{\sigma_2}X\vec{v}_2 = \frac{1}{1}\begin{bmatrix}2 & 3\\ 3 & 2\end{bmatrix}\begin{bmatrix}-1/\sqrt{2}\\1/\sqrt{2}\end{bmatrix} = \begin{bmatrix}1/\sqrt{2}\\-1/\sqrt{2}\end{bmatrix}$

So, $U = \begin{bmatrix}1/\sqrt{2} & 1/\sqrt{2}\\ 1/\sqrt{2} & -1/\sqrt{2}\end{bmatrix}$. Note that the negative signs in U and V are in very slightly different places; this subtlely makes the whole calculation work. So, putting it all together, X's singular value decomposition is

$$X = \underbrace{\begin{bmatrix}1/\sqrt{2} & 1/\sqrt{2}\\ 1/\sqrt{2} & -1/\sqrt{2}\end{bmatrix}}_{U}\underbrace{\begin{bmatrix}5 & 0\\ 0 & 1\end{bmatrix}}_{\Sigma}\underbrace{\begin{bmatrix}1/\sqrt{2} & 1/\sqrt{2}\\ -1/\sqrt{2} & 1/\sqrt{2}\end{bmatrix}}_{V^T}$$

This looks ever so similar to the boxed equation for $X = Q\Lambda Q^T$ above, but it's slightly different, with minus signs dropped and rearranged.

In short, for non-positive semidefinite but still symmetric matrices, the singular values are the absolute values of the eigenvalues. I don't think this is a general rule worth remembering.

For non-symmetric but still square matrices, the eigenvalues of X and singular values of X don't have any direct relation, other than the general rule that σ_i is the square root of the corresponding eigenvalue of X^TX. Remember that the SVD is mostly designed to solve practical problems that arise with non-square matrices X, which don't have any eigenvalues to begin with.

Example: SVD of an Orthogonal Matrix. Suppose X itself is an $n \times n$ orthogonal matrix, meaning $X^T X = XX^T = I$. Why are all of X's singular values 1?

Solution

Recall, in $X = U\Sigma V^T$, the columns of U are the eigenvectors of XX^T and the columns of V are the eigenvectors of $X^T X$. But here, $X^T X = XX^T = I_{n\times n}$. The $n \times n$ identity matrix has the unique eigenvalue of 1 with algebraic and geometric multiplicity n, meaning

$$\lambda_1 = \lambda_2 = \cdots = \lambda_n = 1$$

so

$$\sigma_1 = \sigma_2 = \cdots = \sigma_n = \sqrt{1} = 1$$

Visualizing the SVD

In Chapter 9.5, we visualized the spectral theorem, which decomposed a symmetric matrix $A = Q\Lambda Q^T$, which we said represented a rotation, stretch, and rotation back.

$$\underbrace{U}_{\text{orthogonal!}} \quad \underbrace{\Sigma}_{\text{diagonal!}} \quad \underbrace{V^T}_{\text{orthogonal!}}$$

$X = U\Sigma V^T$ can be interpreted similarly. If we think of $f(\vec{w}) = X\vec{w}$ as a linear transformation from $\mathbb{R}^d$ to $\mathbb{R}^n$, then f operates on $\vec{w}$ in three stages:

1. a rotation by V^T (because V is orthogonal), followed by
2. a stretch by $\pm$, followed by
3. a rotation by U

To illustrate, let's consider the square (but not symmetric) matrix

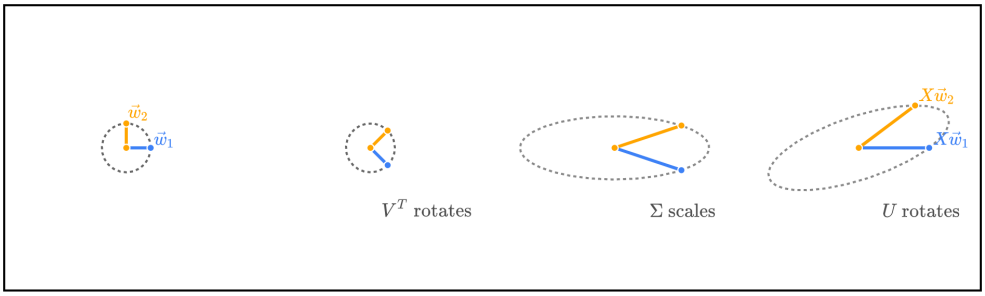
$$X = \begin{bmatrix} 5 & 4 \\ 0 & 3 \end{bmatrix}$$

from one of the earlier worked examples. Unfortunately, it's difficult to visualize the SVD of matrices larger than 2×2, since then either the $\vec{u}_i$'s or $\vec{v}_i$'s (or both) are in $\mathbb{R}^3$ or higher.

X's SVD is

$$\underbrace{\begin{bmatrix} 5 & 4 \\ 0 & 3 \end{bmatrix}}_{X} = \underbrace{\begin{bmatrix} \frac{3}{\sqrt{10}} & -\frac{1}{\sqrt{10}} \\ \frac{1}{\sqrt{10}} & \frac{3}{\sqrt{10}} \end{bmatrix}}_{U} \underbrace{\begin{bmatrix} 3\sqrt{5} & 0 \\ 0 & \sqrt{5} \end{bmatrix}}_{\Sigma} \underbrace{\begin{bmatrix} \frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} \\ -\frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} \end{bmatrix}}_{V^T}$$

Visualizing $X\vec{w} = U\Sigma V^T \vec{w}$ for two different vectors



Rotate, scale, rotate!

The intuition that the SVD decomposes a matrix into a rotation, scaling (stretching), and rotation is crucial for future machine learning applications.

In Chapter 10.2, we'll see how the singular value decomposition can be used to construct low-rank approximations of matrices. A good summary of the SVD is also found at the end of Chapter 10.2.

10.2. Low-Rank Approximation

In Chapter 10.1, we learned how to find the singular value decomposition of a matrix X, by using the eigenvalues and eigenvectors of X^TX and XX^T.

$$X = U\Sigma V^T$$

Here, we'll apply our understanding to a useful practical application: **low-rank approximation**.

Full and Compact SVD

But first, let's try and better understand *how* the SVD decomposes a matrix.

Full SVD. The version of the SVD we've constructed together is called the **full SVD**.

In the full SVD, U is $n \times n$, $\pm$ is $n \times d$, and V^T is $d \times d$. If $\text{rank}(X) = r$, then

- The first r columns of U are the left singular vectors of X and are a basis for $\text{colsp}(X)$; the last $n-r$ columns of U are a basis for $\text{nullsp}(X^T)$. Together, the columns of U span $\mathbb{R}^n$.
- The first r columns of V are the right singular vectors of X and are a basis for $\text{colsp}(X^T)$; the last $d-r$ columns of V are a basis for $\text{nullsp}(X)$. Together, the columns of V span $\mathbb{R}^d$.

The full SVD is nice for proofs, but is a little... annoying to use in real applications, because it contains a lot of 0's. The additional $n-r$ columns of U and $d-r$ columns of V are included to make U and V orthogonal matrices, but end up getting 0'd out when multiplied by the corresponding 0's in $\pm$.

Remember that $X = U\Sigma V^T$ is equivalent to $XV = U\Sigma$, which says that, for $i = 1, 2, ..., d$,

$$X\vec{v}_i = \sigma_i \vec{u}_i$$

but when $i > r$, all this says is $\vec{0} = \vec{0}$:

- $X\vec{v}_i = \vec{0}$, since $\vec{v}_r, \vec{v}_{r+1}, ..., \vec{v}_d$ are all in $\text{nullsp}(X)$, and
- $\sigma_i = 0$, so $\sigma_i\vec{u}_i = 0\vec{u}_i = \vec{0}$.

Compact SVD. The **compact SVD** throws away the 0's in $\pm$ and the corresponding columns in U and V. In the annotated figure above, it keeps only the first r columns in U, first r values in Σ, and first r rows in V^T.

That is, the compact SVD says $X = U_r\Sigma_r V_r^T$, where U_r is $n \times r$, Σ_r is $r \times r$, and V_r^T is $r \times d$.

$$X = \underbrace{\begin{bmatrix} | & \cdots & | \\ \vec{u}_1 & \cdots & \vec{u}_r \\ | & \cdots & | \end{bmatrix}}_{U_r} \underbrace{\begin{bmatrix} \sigma_1 & & \\ & \ddots & \\ & & \sigma_r \end{bmatrix}}_{\Sigma_r} \underbrace{\begin{bmatrix} — & \vec{v}_1^T & — \\ & \vdots & \\ — & \vec{v}_r^T & — \end{bmatrix}}_{V_r^T}$$

The full and compact SVDs are equivalent, in that $X = U\Sigma V^T = U_r\Sigma_r V_r^T$. But the individual components are of different sizes:

	Full SVD	Compact SVD	
U	$n \times n$	$n \times r$	First r columns of U are basis for colsp(X)
$\pm$	$n \times d$	$r \times r$	Number of non-zero σ_i's is $r = \text{rank}(X)$
V^T	$d \times d$	$r \times d$	First r rows of V^T are basis for colsp(X^T)

U_r and V_r are no longer orthogonal matrices, since only square matrices can be orthogonal. However, their columns are still orthonormal, meaning $U_r^T U_r = I_{r\times r}$ and $V_r^T V_r = I_{r\times r}$.

SVD as a Sum

The compact SVD hints at another way to think about X.

$$X = \underbrace{\begin{bmatrix} | & \cdots & | \\ \vec{u}_1 & \cdots & \vec{u}_r \\ | & \cdots & | \end{bmatrix}}_{U_r} \underbrace{\begin{bmatrix} \sigma_1 & & \\ & \ddots & \\ & & \sigma_r \end{bmatrix}}_{\Sigma_r} \underbrace{\begin{bmatrix} — & \vec{v}_1^T & — \\ & \vdots & \\ — & \vec{v}_r^T & — \end{bmatrix}}_{V_r^T}$$

$$= \begin{bmatrix} | & \cdots & | \\ \sigma_1\vec{u}_1 & \cdots & \sigma_r\vec{u}_r \\ | & \cdots & | \end{bmatrix} \begin{bmatrix} — & \vec{v}_1^T & — \\ & \vdots & \\ — & \vec{v}_r^T & — \end{bmatrix}$$

$$= \sigma_1\vec{u}_1\vec{v}_1^T + \sigma_2\vec{u}_2\vec{v}_2^T + \cdots + \sigma_r\vec{u}_r\vec{v}_r^T$$

Each term $\sigma_i\vec{u}_i\vec{v}_i^T$ is an **outer product** of $\vec{u}_i$ and $\vec{v}_i$, scaled by σ_i. Outer products are rank-one matrices: each column of $\sigma_i\vec{u}_i\vec{v}_i^T$ is a multiple of $\vec{u}_i$, and each row of it is a multiple of $\vec{v}_i^T$.

This outer product view of matrix multiplication is not one that we've emphasized a ton in this course, but it can be useful in certain contexts, as we're about to see. To see how it works, let's revisit our first example.

$$\underbrace{\begin{bmatrix} 3 & 2 & 5 \\ 2 & 3 & 5 \\ 2 & -2 & 0 \\ 5 & 5 & 10 \end{bmatrix}}_{X} = \underbrace{\begin{bmatrix} \frac{1}{\sqrt{6}} & \frac{1}{3\sqrt{2}} & -\frac{1}{\sqrt{3}} & -\frac{2}{3} \\ \frac{1}{\sqrt{6}} & -\frac{1}{3\sqrt{2}} & -\frac{1}{\sqrt{3}} & \frac{2}{3} \\ 0 & \frac{2\sqrt{2}}{3} & 0 & \frac{1}{3} \\ \frac{2}{\sqrt{6}} & 0 & \frac{1}{\sqrt{3}} & 0 \end{bmatrix}}_{U} \underbrace{\begin{bmatrix} 15 & 0 & 0 \\ 0 & 3 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix}}_{\Sigma} \underbrace{\begin{bmatrix} \frac{1}{\sqrt{6}} & \frac{1}{\sqrt{6}} & \frac{2}{\sqrt{6}} \\ \frac{1}{\sqrt{2}} & -\frac{1}{\sqrt{2}} & 0 \\ \frac{1}{\sqrt{3}} & \frac{1}{\sqrt{3}} & -\frac{1}{\sqrt{3}} \end{bmatrix}}_{V^T}$$

The summation view of the SVD says that:

$$X = 15 \begin{bmatrix} \frac{1}{\sqrt{6}} \\ \frac{1}{\sqrt{6}} \\ 0 \\ \frac{2}{\sqrt{6}} \end{bmatrix} \begin{bmatrix} \frac{1}{\sqrt{6}} & \frac{1}{\sqrt{6}} & \frac{2}{\sqrt{6}} \end{bmatrix} + 3 \begin{bmatrix} \frac{1}{3\sqrt{2}} \\ -\frac{1}{3\sqrt{2}} \\ \frac{2\sqrt{2}}{3} \\ 0 \end{bmatrix} \begin{bmatrix} \frac{1}{\sqrt{2}} & -\frac{1}{\sqrt{2}} & 0 \end{bmatrix}$$

$$= \underbrace{\begin{bmatrix} \frac{5}{2} & \frac{5}{2} & 5 \\ \frac{5}{2} & \frac{5}{2} & 5 \\ 0 & 0 & 0 \\ 5 & 5 & 10 \end{bmatrix}}_{\text{rank-one matrix}} + \underbrace{\begin{bmatrix} \frac{1}{2} & -\frac{1}{2} & 0 \\ -\frac{1}{2} & \frac{1}{2} & 0 \\ 2 & -2 & 0 \\ 0 & 0 & 0 \end{bmatrix}}_{\text{rank-one matrix}}$$

Since $15 > 3$, the first outer product contributes more to X than the second one does.

We can think of the singular values as representing the importance of the corresponding singular vectors in representing X. Since we sort singular values in decreasing order, $\sigma_1 \geq \sigma_2 \geq \ldots \geq \sigma_r > 0$, the first outer product is always the most important one, the second outer product is the second most important one, and so on.

Low-Rank Approximation. Our most recent observation is that the SVD $X = U\Sigma V^T$ allows us to write X as a sum of rank-one matrices, in **decreasing order of importance**.

$$X = \underbrace{\sigma_1 \vec{u}_1 \vec{v}_1^T}_{\text{most important}} + \underbrace{\sigma_2 \vec{u}_2 \vec{v}_2^T}_{\text{second most important}} + \cdots + \underbrace{\sigma_r \vec{u}_r \vec{v}_r^T}_{\text{least important}}$$

In many practical applications, the full matrix X can be too large to store or process. In such cases, we can produce a **low-rank approximation** of X by summing fewer than r of these rank-one matrices. In the example above, X was of rank 2, so a rank-1 approximation of X would just be the first outer product, $\sigma_1 \vec{u}_1 \vec{v}_1^T$, which is in blue above.

Definition: Rank-k Approximation

Suppose X is an $n \times d$ matrix with rank r, with SVD $X = U\Sigma V^T = \sum_{i=1}^{r} \sigma_i \vec{u}_i \vec{v}_i^T$. A **rank-$k$ approximation** of X is a matrix X_k of the form

$$X_k = \sum_{i=1}^{k} \sigma_i \vec{u}_i \vec{v}_i^T$$

where $k \leq r$.

Example: Image Compression. A common application of the SVD and low-rank approximations is to **compress** images. How so? Consider the following grayscale image of my (16 year old) dog, Junior.

This image is 300 pixels wide and 400 pixels tall. Since the image is grayscale, each pixel's intensity can be represented by a number between 0 and 255, where 0 is black and 255 is white. These intensities can be stored in a 400×300 matrix. The rank of this matrix is likely 300, since it's extremely unlikely that any of the 300 columns of the image are exactly representable as a linear combination of other columns.

But, as the SVD reveals, the image can be approximated well by a rank-k matrix, for a k that is much smaller than 300. We build this low-rank approximation of the image by summing up k rank-one matrices.

$$\text{rank } k \text{ approximation of image} = \sum_{i=1}^{k} \sigma_i \vec{u}_i \vec{v}_i^T$$

A slider should appear below, allowing you to select a value of k and see the corresponding rank-k approximation of Junior.

To store the full image, we need to store $400 \cdot 300 = 120{,}000$ numbers. But to store a rank-k approximation

of the image, we only need to store $(1 + 400 + 300)k = 701k$ numbers – only the first k singular values, along with $\vec{u}_1, \vec{u}_2, ..., \vec{u}_k$ (each of which has 400 numbers), and $\vec{v}_1, \vec{v}_2, ..., \vec{v}_k$ (each of which has 300 numbers). If we're satisfied with a rank-30 approximation, we only need to store $701 \cdot 30 = 21{,}030$ numbers, which is a compression of $\frac{120{,}000}{21{,}030} \approx 5.7$ times!

Computing the SVD

Finally, I'll show you how to use `numpy` to compute the SVD of a matrix. The key function is `np.linalg.svd`. Let's apply it to our familiar example,

$$\underbrace{\begin{bmatrix} 3 & 2 & 5 \\ 2 & 3 & 5 \\ 2 & -2 & 0 \\ 5 & 5 & 10 \end{bmatrix}}_{X} = \underbrace{\begin{bmatrix} \frac{1}{\sqrt{6}} & \frac{1}{3\sqrt{2}} & -\frac{1}{\sqrt{3}} & -\frac{2}{3} \\ \frac{1}{\sqrt{6}} & -\frac{1}{3\sqrt{2}} & -\frac{1}{\sqrt{3}} & \frac{2}{3} \\ 0 & \frac{2\sqrt{2}}{3} & 0 & \frac{1}{3} \\ \frac{2}{\sqrt{6}} & 0 & \frac{1}{\sqrt{3}} & 0 \end{bmatrix}}_{U} \underbrace{\begin{bmatrix} 15 & 0 & 0 \\ 0 & 3 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix}}_{\Sigma} \underbrace{\begin{bmatrix} \frac{1}{\sqrt{6}} & \frac{1}{\sqrt{6}} & \frac{2}{\sqrt{6}} \\ \frac{1}{\sqrt{2}} & -\frac{1}{\sqrt{2}} & 0 \\ \frac{1}{\sqrt{3}} & \frac{1}{\sqrt{3}} & -\frac{1}{\sqrt{3}} \end{bmatrix}}_{V^T}$$

```
X = np.array([[3, 2, 5],
              [2, 3, 5],
              [2, -2, 0],
              [5, 5, 10]])

u, s, vt = np.linalg.svd(X)
u.shape, s.shape, vt.shape
```

```
((4, 4), (3,), (3, 3))
```

By default, it computes the full SVD, which is why `u` is of shape 4×4 and `vt` is of shape 3×3, even though $\text{rank}(X) = 2$.

`s` is returned as a 1-dimensional array of singular values.

```
s
```

```
array([15.,  3.,  0.])
```

If we'd like to use `u`, `s`, and `vt` to reconstruct X, we need to reshape `s` into a matrix with the same shape as X.

```
ss = np.zeros(X.shape)
ss[np.arange(len(s)), np.arange(len(s))] = s
ss
```

```
array([[15.,  0.,  0.],
       [ 0.,  3.,  0.],
       [ 0.,  0.,  0.],
       [ 0.,  0.,  0.]])
```

```
u @ ss @ vt
```

```
array([[ 3.,  2.,  5.],
       [ 2.,  3.,  5.],
       [ 2., -2.,  0.],
       [ 5.,  5., 10.]])
```

Notice: The signs of the columns of U and V are not uniquely determined by the SVD. For example, we could replace $\vec{u}_1$ with $-\vec{u}_1$ and $\vec{v}_1$ with $-\vec{v}_1$ to get the same matrix X. The values for U we found had flipped signs for columns 1, 2 relative to the values returned by `np.linalg.svd`.

$$\text{our } U = \begin{bmatrix} \frac{1}{\sqrt{6}} & \frac{1}{3\sqrt{2}} & -\frac{1}{\sqrt{3}} & -\frac{2}{3} \\ \frac{1}{\sqrt{6}} & -\frac{1}{3\sqrt{2}} & -\frac{1}{\sqrt{3}} & \frac{2}{3} \\ 0 & \frac{2\sqrt{2}}{3} & 0 & \frac{1}{3} \\ \frac{2}{\sqrt{6}} & 0 & \frac{1}{\sqrt{3}} & 0 \end{bmatrix}$$

`u`

```
array([[ -0.40824829,  0.23570226, -0.45735398, -0.75405909],
       [ -0.40824829, -0.23570226, -0.68098866,  0.56038577],
       [ -0.        ,  0.94280904, -0.05590867,  0.32861122],
       [ -0.81649658, -0.        ,  0.56917132,  0.09683666]])
```

- The first column of `numpy`'s `u` is the negative of our first column of U.
- Both second columns are the same.
- The latter two columns are different. Why? There are infinitely many orthogonal bases for the null space of X^T, which are what the last two columns of U represent. We just picked a different choice than `numpy` did.

Key Takeaways

To recap:

1. All $n \times d$ matrices X have a singular value decomposition $X = U\Sigma V^T$, where U is $n \times n$, $\pm$ is $n \times d$, and V is $d \times d$.
2. The columns of U are orthonormal eigenvectors of XX^T; these are called the **left singular vectors** of X.
3. The columns of V are orthonormal eigenvectors of X^TX; these are called the **right singular vectors** of X.
4. Both XX^T and X^TX share the same non-zero eigenvalues; the singular values of X are the square roots of these eigenvalues. The number of non-zero singular values is equal to the rank of X. It's important that we sort the singular values in decreasing order, so that $\sigma_1 \geq \sigma_2 \geq \ldots \geq \sigma_r > 0$, and place the singular vectors in the columns of U and V in the same order.

$$\sigma_i = \sqrt{\lambda_i}$$

5. A typical recipe for computing the SVD is to:
   (a) Compute $X^T X$. Place its eigenvectors in the columns of V, and place the square roots of its eigenvalues in the diagonal of $\pm$.
   (b) To find each $\vec{u}_i$, use $X\vec{v}_i = \sigma_i \vec{u}_i$, i.e. $\vec{u}_i = \frac{1}{\sigma_i} X\vec{v}_i$.
   (c) The above rule only works for $\sigma_i > 0$. If $\sigma_i = 0$, then the remaining $\vec{u}_i$'s must be eigenvectors of XX^T for the eigenvalue 0, meaning they must lie in the nullspace of X^T.
6. The SVD allows us to interpret the linear transformation of multiplying by X as a composition of a rotation by V^T, a scaling/stretching by $\pm$, and a rotation by U.
7. The SVD $X = U\Sigma V^T$ can be viewed as a sum of rank-one matrices:

$$X = \sum_{i=1}^{r} \sigma_i \vec{u}_i \vec{v}_i^T$$

   Each piece $\sigma_i \vec{u}_i \vec{v}_i^T$ is a rank-one matrix, consisting of the outer product of $\vec{u}_i$ and $\vec{v}_i$. This summation view can be used to compute a low-rank approximation of X by summing fewer than r of these rank-one matrices.

$$X_k = \sum_{i=1}^{k} \sigma_i \vec{u}_i \vec{v}_i^T$$

10.3. The Best Direction

Introduction

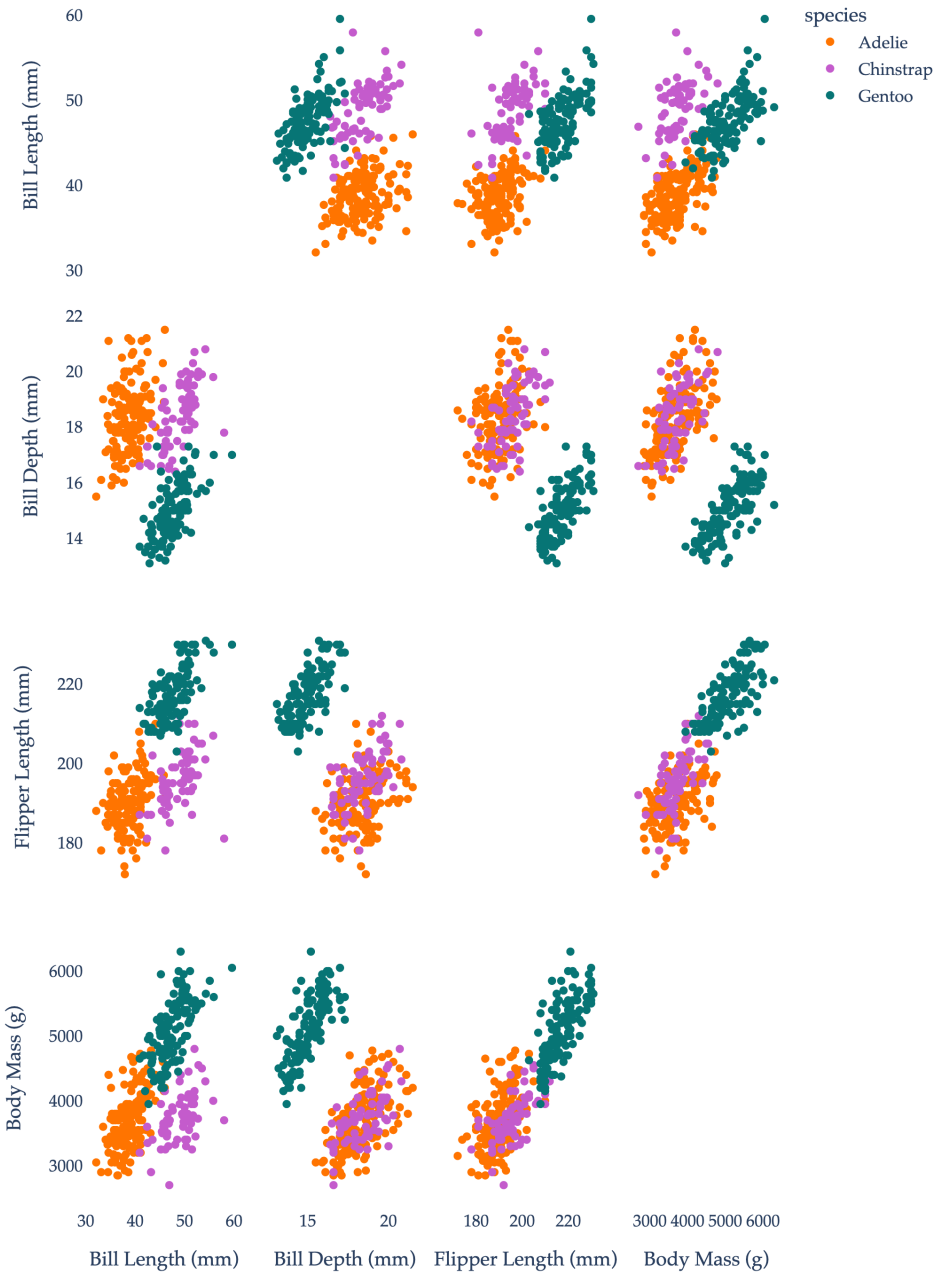
As we saw in Chapter 10.2, the singular value decomposition of a matrix X into $X = U\Sigma V^T$ can be used to find the best low-rank approximation of X, which has applications in image compression, for example. But, hiding in the singular value decomposition is another, more data-driven application: **dimensionality reduction**.

Consider, for example, the dataset below. Each row corresponds to a penguin.

	species	island	bill_length_mm	bill_depth_mm	flipper_length_mm	body_mass_g	sex
0	Adelie	Torgersen	39.1	18.7	181.0	3750.0	Male
1	Adelie	Torgersen	39.5	17.4	186.0	3800.0	Female
2	Adelie	Torgersen	40.3	18.0	195.0	3250.0	Female
4	Adelie	Torgersen	36.7	19.3	193.0	3450.0	Female
5	Adelie	Torgersen	39.3	20.6	190.0	3650.0	Male
...	...	...	...	...	...	...	...
338	Gentoo	Biscoe	47.2	13.7	214.0	4925.0	Female
340	Gentoo	Biscoe	46.8	14.3	215.0	4850.0	Female
341	Gentoo	Biscoe	50.4	15.7	222.0	5750.0	Male
342	Gentoo	Biscoe	45.2	14.8	212.0	5200.0	Female
343	Gentoo	Biscoe	49.9	16.1	213.0	5400.0	Male

333 rows × 7 columns

Each penguin has three categorical features - `species`, `island`, and `sex` - and four numerical features - `bill_length_mm`, `bill_depth_mm`, `flipper_length_mm`, and `body_mass_g`. If we restrict our attention to these numerical features, we have a 333×4 matrix, which we can think of as 333 points in $\mathbb{R}^4$. We, of course, can't visualize in $\mathbb{R}^4$. One solution is to draw scatter plots of all pairs of numerical features. Points are colored by `species`.



While we have four features (or "independent" variables) for each penguin, it seems that some of the features are strongly correlated, in the sense that the correlation coefficient

$$r = \frac{1}{n}\sum_{i=1}^{n}\left(\frac{x_i - \bar{x}}{\sigma_x}\right)\left(\frac{y_i - \bar{y}}{\sigma_y}\right)$$

between them is close to ± 1. (To be clear, the coloring of the points is by `species`, which does not impact the correlations between features.)

```
penguins[['bill_length_mm', 'bill_depth_mm', 'flipper_length_mm', 'body_mass_g']].corr()
```

	bill_length_mm	bill_depth_mm	flipper_length_mm	body_mass_g
bill_length_mm	1.000000	-0.228626	0.653096	0.589451
bill_depth_mm	-0.228626	1.000000	-0.577792	-0.472016
flipper_length_mm	0.653096	-0.577792	1.000000	0.872979
body_mass_g	0.589451	-0.472016	0.872979	1.000000

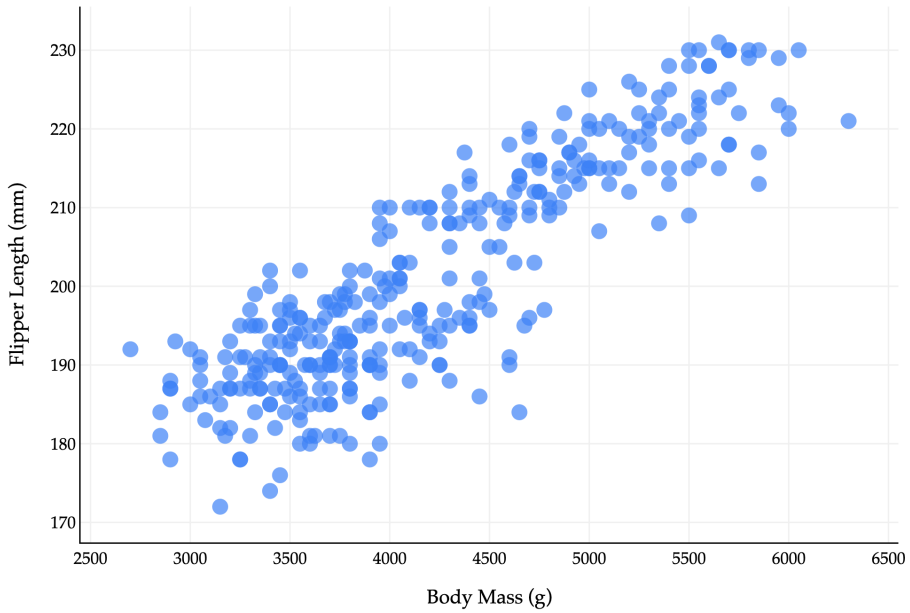
For instance, the correlation between `flipper_length_mm` and `body_mass_g` is about 0.87, which suggests that penguins with longer flippers tend to be heavier. Not only are correlated features redundant, but they can cause interpretability issues - that is, multicollinearity - when using them in a linear regression model, since the resulting optimal parameters $\vec{w}^*$ will no longer be interpretable. (Remember, in $h(\vec{x}_i) = \vec{w} \cdot \text{Aug}(\vec{x}_i)$, $\vec{w}^*$ contains the optimal coefficients for each feature, which measure rates of change when the other features are held constant. But if two features are correlated, one can't be held constant without also changing the other.)

Here's the key idea: instead of having to work with $d = 4$ features, perhaps we can instead **construct** some number $k < d$ of **new** features that:

1. capture the "most important" information in the dataset,
2. while being **uncorrelated** (i.e. linearly independent) from one another.

I'm not suggesting that we just select some of the original features and drop the others; rather, I'm proposing that we find 1 or 2 new features that are **linear combinations** of the original features. If everything works out correctly, we may be able to **reduce the number of features** we need to deal with, without losing too much information, and without the interpretability issues that come with multicollinearity.

To illustrate what I mean by constructing a new feature, let's suppose we're starting with a 2-dimensional dataset, which we'd like to reduce to 1 dimension. Consider the scatter plot below of `flipper_length_mm` vs. `body_mass_g` (with colors removed).



Above, each penguin is represented by two numbers, which we can think of as a vector

$$\vec{x}_i = \begin{bmatrix} \text{body mass}_i \\ \text{flipper length}_i \end{bmatrix}$$

The fact that each point is a vector is not immediately intuitive given the plot above: I want you to imagine drawing arrows from the origin to each point.

Now, suppose these vectors $\vec{x}_i$ are in the rows of a matrix X, i.e.

$$X = \begin{bmatrix} \text{body mass}_1 & \text{flipper length}_1 \\ \text{body mass}_2 & \text{flipper length}_2 \\ \vdots & \vdots \\ \text{body mass}_n & \text{flipper length}_n \end{bmatrix}$$

Our goal is to construct a single new feature, called a **principal component**, by taking a linear combination of both existing features. **Principal component analysis**, or **PCA**, is the process of finding the best principal components (new features) for a dataset.

$$\text{new feature}_i = a \cdot \text{body mass}_i + b \cdot \text{flipper length}_i$$

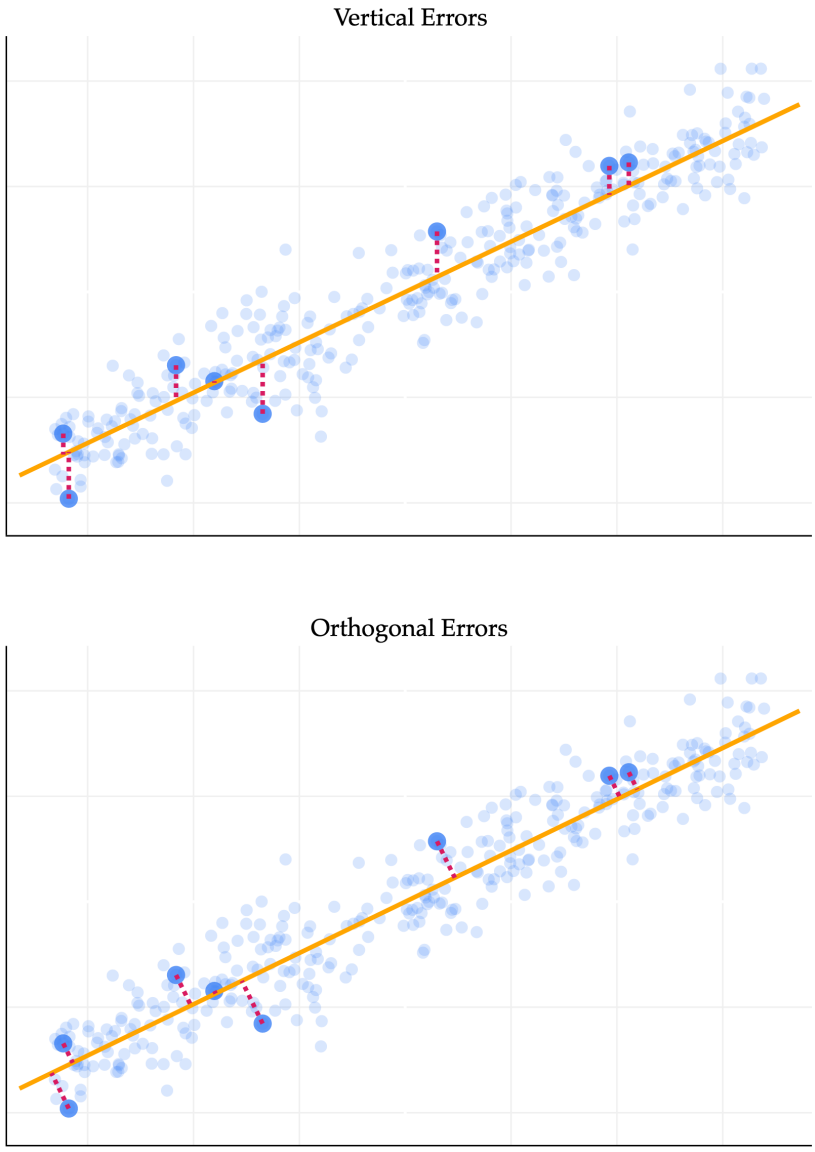
This is equivalent to approximating this 2-dimensional dataset with a single line. (The equation above is a re-arranged version of $y = mx + b$, where x is body mass_i and y is flipper length_i.)

What is the *best* linear combination of the features, or equivalently, what is the best line to approximate this 2-dimensional dataset? To make this idea precise, we need to define what we mean by "best". It may be tempting to think that we should find the best-fitting line that predicts `flipper_length_mm` from `body_mass_g`, but that's not quite what we want, since **this is not a prediction problem: it's a representation problem**. Regression lines in the way we've seen them before are fit by minimizing mean squared **vertical** error, $y_i - h(x_i)$, but don't at all capture horizontal errors, i.e. errors in the `body_mass_g` direction here.

So instead, we'd like to find the line such that, for our n vectors (points) $\vec{x}_1, \ldots, \vec{x}_n$, the mean squared **distance to the closest vector (point) on the line** is minimized. Crucially, this distance is calculated by computing the norm of the difference between the two vectors. If $\vec{p}_i$ is the closest vector (point) on the line to $\vec{x}_i$, then we'd like to minimize

$$\frac{1}{n} \sum_{i=1}^{n} \|\vec{x}_i - \vec{p}_i\|^2$$

But, as we saw in Chapter 3.4, the vector (point) on a line that is closest to another vector (point) $\vec{x}_i$ is the **orthogonal projection** of $\vec{x}_i$ onto that line. The resulting error vectors are orthogonal to the line we're projecting on. As such, I'll say that we're looking for the line that minimizes the mean squared **orthogonal projection** error.



The difference between the two lines above is key. The top line reflects the supervised nature of linear regression, where the goal is to make accurate predictions. The bottom line reflects the fact that we're looking for a new representation of the data that captures the most important information. These, in general, are not the same line, since they're chosen to minimize different objective functions.

Recall from Chapter 1 that **unsupervised learning** is the process of finding structure in unlabeled data, without any specific target variable. This is precisely the problem we're trying to solve here. To further distinguish our current task from the problem of finding regression lines, I'll now refer to finding the line with minimal orthogonal projection error as finding the **best direction**.

Activity 1

Activity 1

Consider the line $\frac{3}{5}x + \frac{4}{5}y = 0$ in $\mathbb{R}^2$. For the point $(1, 1)$,

1. What is the vertical error of projecting $(1, 1)$ onto the line?
2. What is the point on the line that is closest to $(1, 1)$?
3. What is the orthogonal error of projecting $(1, 1)$ onto the line?

Minimizing Orthogonal Error

The first section above told us that our goal is to find the best direction, which is the direction that minimizes the mean squared orthogonal error of projections onto it. How do we find this direction?

Fortunately, we spent all of Chapter 3.4 and Chapter 6.3 discussing orthogonal projections. But there, we discussed how to project **one** vector onto the span of one or more other **given** vectors (e.g. projecting the observation vector $\vec{y}$ onto the span of the columns of a design matrix X). That's not quite we're doing here, because here, we don't know which vector we're projecting onto: we're trying to find the **best** vector to project our $n = 333$ other vectors onto!

Suppose that in general, X is an $n \times d$ matrix, whose rows $\vec{x}_i$ are data points in $\mathbb{R}^d$. (In our current example, $d = 2$, but everything we're studying applies in higher dimensions.) Then,

- The orthogonal projection of $\vec{x}_i$ onto $\vec{v}$ is

$$\vec{p}_i = \frac{\vec{x}_i \cdot \vec{v}}{\vec{v} \cdot \vec{v}} \vec{v}$$

- To keep the algebra slightly simpler, if we assume that $\vec{v}$ is a unit vector (meaning $\|\vec{v}\| = \sqrt{\vec{v} \cdot \vec{v}} = 1$), then the orthogonal projection of $\vec{x}_i$ onto $\vec{v}$ is

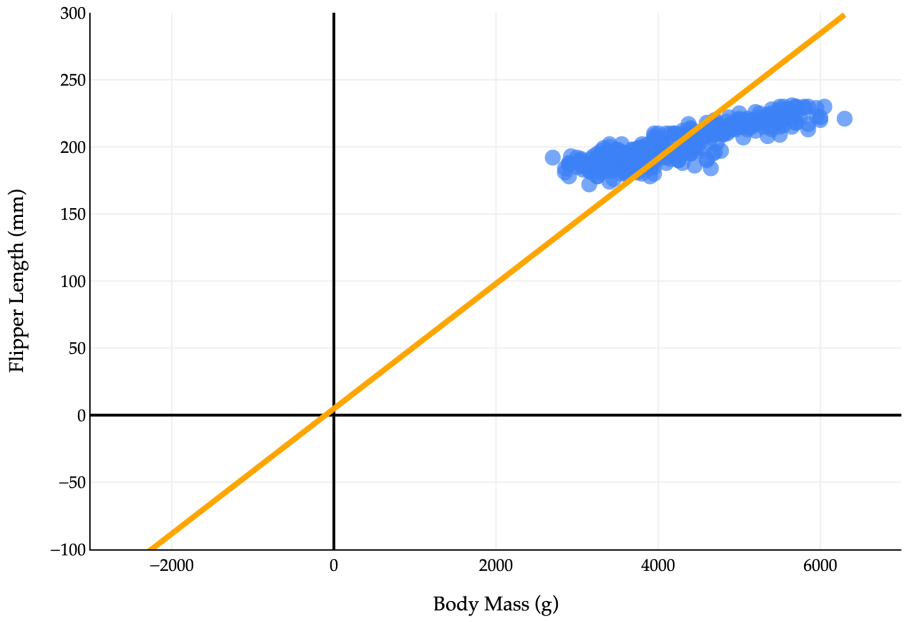
$$\vec{p}_i = (\vec{x}_i \cdot \vec{v})\vec{v}$$

  This is a reasonable choice to make, since unit vectors are typically used to describe directions.

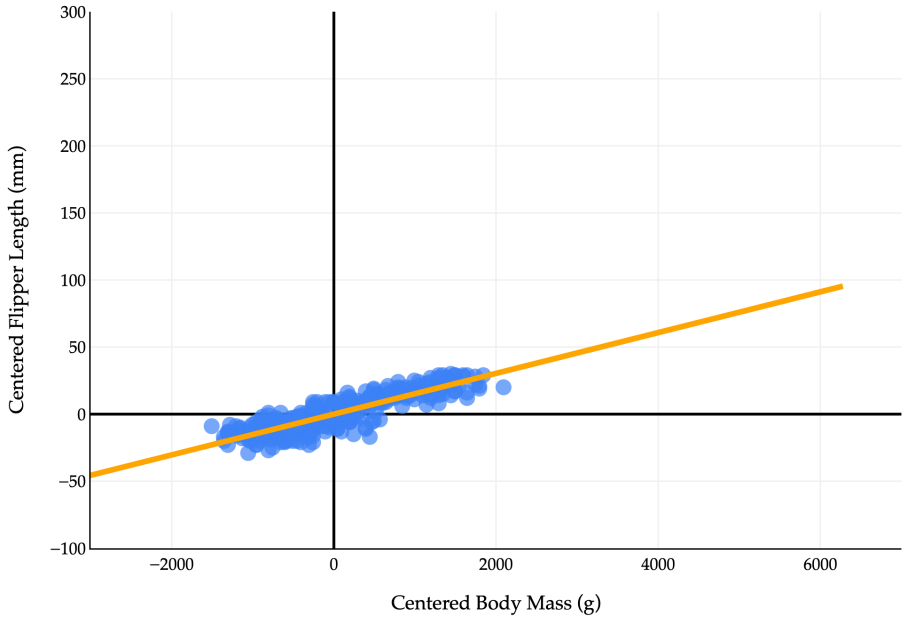
- If $\vec{v} \in \mathbb{R}^d$, then $\text{span}(\{\vec{v}\})$ is a line through the origin in $\mathbb{R}^d$.

Centering the Data. The issue is that our dataset - and most datasets in general, by default - aren't centered at the origin, but the projections $\vec{p}_i = (\vec{x}_i \cdot \vec{v})\vec{v}$ all lie on the line through the origin spanned by $\vec{v}$. So, trying to approximate our dataset using a line through the origin is not a good idea.

Our data isn't usually located near the origin...



...which is why we center the data first! This doesn't change its shape.



So, the solution is to first **center** the data by subtracting the mean of each column from each entry in that column. This gives us a new matrix $\tilde{X}$.

$$X = \begin{bmatrix} x_1^{(1)} & x_1^{(2)} & \cdots & x_1^{(d)} \\ x_2^{(1)} & x_2^{(2)} & \cdots & x_2^{(d)} \\ \vdots & \vdots & \ddots & \vdots \\ x_n^{(1)} & x_n^{(2)} & \cdots & x_n^{(d)} \end{bmatrix} \rightarrow \tilde{X} = \begin{bmatrix} x_1^{(1)} - \mu_1 & x_1^{(2)} - \mu_2 & \cdots & x_1^{(d)} - \mu_d \\ x_2^{(1)} - \mu_1 & x_2^{(2)} - \mu_2 & \cdots & x_2^{(d)} - \mu_d \\ \vdots & \vdots & \ddots & \vdots \\ x_n^{(1)} - \mu_1 & x_n^{(2)} - \mu_2 & \cdots & x_n^{(d)} - \mu_d \end{bmatrix}$$

In $\tilde{X}$, **the mean of each column is 0**, which means that it's reasonable to approximate the dataset using a line through the origin. Importantly, this doesn't change the shape of the dataset, meaning it doesn't change the

"best direction" we're searching for.

Activity 2

Activity 2

Suppose $\tilde{X}$ is a centered $n \times d$ matrix, meaning the mean of each column of $\tilde{X}$ is 0.
Prove that the entries of $\tilde{X}\vec{w}$, where $\vec{w}$ is any vector in $\mathbb{R}^d$, sum to 0. *Hint: Think about how you can use the vector* $\vec{1}$.

Finding the Best Direction. Now, $\tilde{X}$ is the $n \times d$ matrix of centered data, and we're ready to return to our goal of finding the best direction. Let $\tilde{x}_i \in \mathbb{R}^d$ refer to row i of $\tilde{X}$. Note that I've dropped the vector symbol on top of $\tilde{x}_i$, not because it's not a vector, but just because drawing a tilde on top of a vector hat is a little funky: $\vec{\tilde{x}}_i$.

$$\tilde{X} = \begin{bmatrix} x_1^{(1)} - \mu_1 & x_1^{(2)} - \mu_2 & \cdots & x_1^{(d)} - \mu_d \\ x_2^{(1)} - \mu_1 & x_2^{(2)} - \mu_2 & \cdots & x_2^{(d)} - \mu_d \\ \vdots & \vdots & \ddots & \vdots \\ x_n^{(1)} - \mu_1 & x_n^{(2)} - \mu_2 & \cdots & x_n^{(d)} - \mu_d \end{bmatrix} = \begin{bmatrix} - & \tilde{x}_1^T & - \\ - & \tilde{x}_2^T & - \\ & \vdots & \\ - & \tilde{x}_n^T & - \end{bmatrix}$$

Our goal is to find the **unit vector** $\vec{v} \in \mathbb{R}^d$ that minimizes the mean squared orthogonal projection error. That is, if the orthogonal projection of $\tilde{x}_i$ onto $\vec{v}$ is $\vec{p}_i$, then we want to find the unit vector $\vec{v}$ that **minimizes**

$$J(\vec{v}) = \frac{1}{n}\sum_{i=1}^{n} \|\tilde{x}_i - \vec{p}_i\|^2 = \frac{1}{n}\sum_{i=1}^{n} \|\tilde{x}_i - (\tilde{x}_i \cdot \vec{v})\vec{v}\|^2$$

I've chosen the letter J since there is no standard notation for this quantity, and because J is a commonly used letter in optimization problems. It's important to note that J is **not** a loss function, nor is it an empirical risk, since it has nothing to do with prediction.

How do we find the unit vector $\vec{v}$ that minimizes $J(\vec{v})$? We can start by expanding the squared norm, using the fact that $\|\vec{a} - \vec{b}\|^2 = \|\vec{a}\|^2 - 2\vec{a} \cdot \vec{b} + \|\vec{b}\|^2$:

$$
\begin{aligned}
J(\vec{v}) &= \frac{1}{n}\sum_{i=1}^{n}\|\tilde{x}_i - (\tilde{x}_i \cdot \vec{v})\vec{v}\|^2 \\
&= \frac{1}{n}\sum_{i=1}^{n}\left(\|\tilde{x}_i\|^2 - 2\tilde{x}_i \cdot (\tilde{x}_i \cdot \vec{v})\vec{v} + \left((\tilde{x}_i \cdot \vec{v})\vec{v}\right) \cdot \left((\tilde{x}_i \cdot \vec{v})\vec{v}\right)\right) \\
&= \frac{1}{n}\sum_{i=1}^{n}\left(\|\tilde{x}_i\|^2 - 2\tilde{x}_i \cdot (\tilde{x}_i \cdot \vec{v})\vec{v} + (\tilde{x}_i \cdot \vec{v})^2 \underbrace{\vec{v} \cdot \vec{v}}_{=1}\right) \\
&= \frac{1}{n}\sum_{i=1}^{n}\left(\|\tilde{x}_i\|^2 - 2(\tilde{x}_i \cdot \vec{v})^2 + (\tilde{x}_i \cdot \vec{v})^2\right) \\
&= \frac{1}{n}\sum_{i=1}^{n}\left(\|\tilde{x}_i\|^2 - (\tilde{x}_i \cdot \vec{v})^2\right) \\
&= \frac{1}{n}\sum_{i=1}^{n}\|\tilde{x}_i\|^2 - \frac{1}{n}\sum_{i=1}^{n}(\tilde{x}_i \cdot \vec{v})^2
\end{aligned}
$$

The first term, $\frac{1}{n}\sum_{i=1}^{n}\|\tilde{x}_i\|^2$, is constant with respect to $\vec{v}$, meaning we can ignore it for the purposes of minimizing $J(\vec{v})$. Our focus, then, is on minimizing the second term,

$$-\frac{1}{n}\sum_{i=1}^{n}(\tilde{x}_i \cdot \vec{v})^2$$

But, since the second term has a negative sign in front of it, minimizing it is equivalent to maximizing its **positive** counterpart,

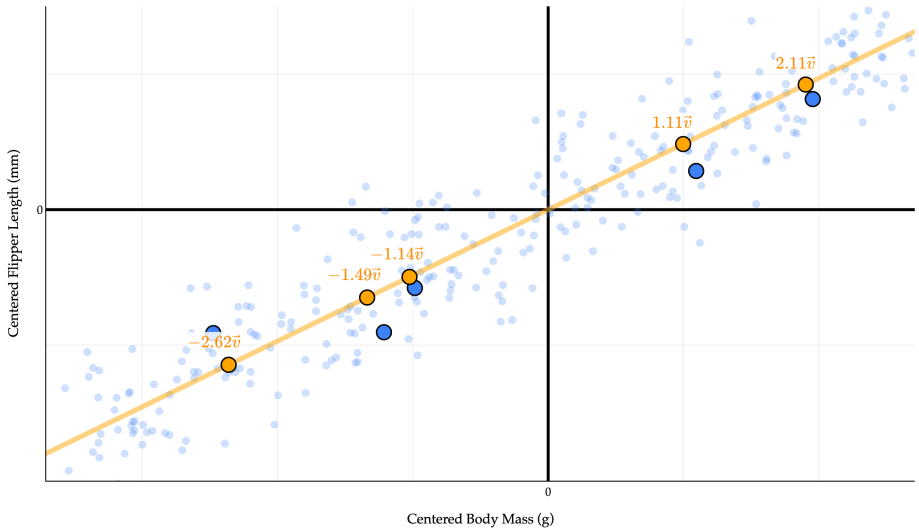
$$\frac{1}{n}\sum_{i=1}^{n}(\tilde{x}_i \cdot \vec{v})^2$$

Maximizing Variance

Finding the best direction is equivalent to finding the unit vector $\vec{v}$ that maximizes the mean squared value of $\tilde{x}_i \cdot \vec{v}$. What is this quantity, though? Remember, the projection of $\tilde{x}_i$ onto $\vec{v}$ is given by

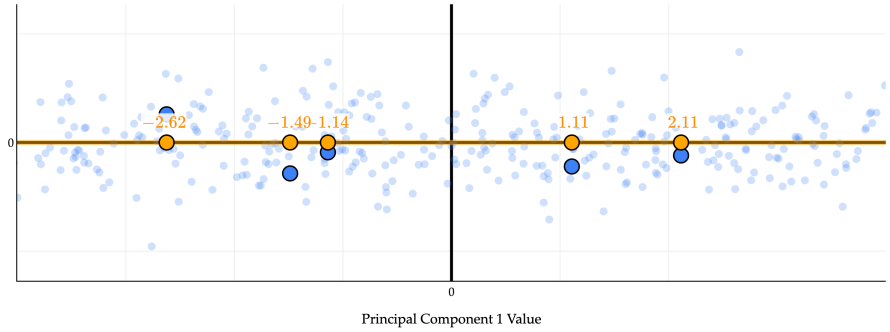
$$\vec{p}_i = (\tilde{x}_i \cdot \vec{v})\vec{v}$$

All n of our original data points are projected onto $\vec{v}$, meaning that each $\tilde{x}_i$ is approximated by some scalar multiple $(\tilde{x}_i \cdot \vec{v})$ of $\vec{v}$. These scalar multiples are our new feature values - that is, the values of the **first principal component** (first because this is the "best" direction).



The values $\tilde{x}_1 \cdot \vec{v}, \tilde{x}_2 \cdot \vec{v}, \ldots, \tilde{x}_n \cdot \vec{v}$ - -2.62, -1.49, -1.14, -1.11, and 2.11 above - describe the position of point $\tilde{x}_i$ on the line defined by $\vec{v}$. (I've highlighted just five of the points to make the picture less cluttered, but in reality every single blue point has a corresponding orange dot above!)

These coefficients on $\vec{v}$ can be thought of as positions on a number line.



Effectively, we've **rotated** the data so that the orange points are on the "new" x-axis. (Rotations... where have we heard of those before? The connection is coming.)

So, we have a grasp of what the values $\tilde{x}_1 \cdot \vec{v}, \tilde{x}_2 \cdot \vec{v}, \ldots, \tilde{x}_n \cdot \vec{v}$ describe. But why are they important?

The quantity we're trying to maximize,

$$\frac{1}{n} \sum_{i=1}^{n} (\tilde{x}_i \cdot \vec{v})^2$$

is the **variance** of the values $\tilde{x}_1 \cdot \vec{v}, \tilde{x}_2 \cdot \vec{v}, \ldots, \tilde{x}_n \cdot \vec{v}$. It's not immediately obvious why this is the case. Remember, the variance of a list of numbers is

$$\frac{1}{n} \sum_{i=1}^{n} (\text{value}_i - \text{mean})^2$$

(I'm intentionally using vague notation to avoid conflating singular values with standard deviations.)

The quantity $\frac{1}{n} \sum_{i=1}^{n} (\tilde{x}_i \cdot \vec{v})^2$ resembles this definition of variance, but it doesn't seem to involve a subtraction

by a mean. This is because the mean of $\tilde{x}_1 \cdot \vec{v}, \tilde{x}_2 \cdot \vec{v}, \ldots, \tilde{x}_n \cdot \vec{v}$ is 0! This seems to check out visually above: it seems that if we found the corresponding orange point for every single blue point, the orange points would be symmetrically distributed around 0.

Proof that the mean of $\tilde{x}_1 \cdot \vec{v}, \tilde{x}_2 \cdot \vec{v}, \ldots, \tilde{x}_n \cdot \vec{v}$ is 0

Recall that each $\tilde{x}_i$ is a vector in $\mathbb{R}^d$ and corresponds to a **row** in $\tilde{X}$. The values $\tilde{x}_1 \cdot \vec{v}, \tilde{x}_2 \cdot \vec{v}, \ldots, \tilde{x}_n \cdot \vec{v}$ can all be computed at once through the matrix-vector product $\tilde{X}\vec{v}$ (remember that to multiply a matrix by a vector, we dot product the rows of the matrix with the vector).
The **columns** of $\tilde{X}$ each have a mean of 0. As we discovered in Activity 2, this means that

$$\tilde{X}^T\vec{1} = \begin{bmatrix} \text{sum of column 1 of } \tilde{X} \\ \text{sum of column 2 of } \tilde{X} \\ \vdots \\ \text{sum of column } d \text{ of } \tilde{X} \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ \vdots \\ 0 \end{bmatrix} = \vec{0}_d$$

if $\vec{1} \in \mathbb{R}^n$ is a vector of all ones.
So, what is the mean of the values $\tilde{x}_1 \cdot \vec{v}, \tilde{x}_2 \cdot \vec{v}, \ldots, \tilde{x}_n \cdot \vec{v}$? This mean, by definition, is $\frac{1}{n}\sum_{i=1}^n \tilde{x}_i \cdot \vec{v}$, but this is equal to $\frac{1}{n}\left((\tilde{X}\vec{v})^T\vec{1}\right)$, once again, using the fact that the dot product with the ones vector is equivalent to summing the values of the other vector.

$$\frac{1}{n}\sum_{i=1}^n (\tilde{x}_i \cdot \vec{v}) = \frac{1}{n}\left((\tilde{X}\vec{v})^T\vec{1}\right) = \frac{1}{n}\left(\vec{v}^T\tilde{X}^T\vec{1}\right) = \frac{1}{n}\left(\vec{v}^T\vec{0}\right) = 0$$

So, the mean of the values $\tilde{x}_1 \cdot \vec{v}, \tilde{x}_2 \cdot \vec{v}, \ldots, \tilde{x}_n \cdot \vec{v}$ is 0.

Since the mean of the values $\tilde{x}_1 \cdot \vec{v}, \tilde{x}_2 \cdot \vec{v}, \ldots, \tilde{x}_n \cdot \vec{v}$ is 0, it is true that the quantity we're **maximizing** is the variance of the values $\tilde{x}_1 \cdot \vec{v}, \tilde{x}_2 \cdot \vec{v}, \ldots, \tilde{x}_n \cdot \vec{v}$, which I'll refer to as

$$\text{PV}(\vec{v}) = \frac{1}{n}\sum_{i=1}^n (\tilde{x}_i \cdot \vec{v})^2$$

PV stands for "projected variance"; it is the variance of the scalar projections of the rows of $\tilde{X}$ onto $\vec{v}$. This is non-standard notation, but I wanted to avoid calling this the "variance of $\vec{v}$", since it's not the variance of the components of $\vec{v}$.

The Duality. We've stumbled upon a beautiful **duality** - that is, two equivalent ways to think about the same problem. Given a centered $n \times d$ matrix $\tilde{X}$ with rows $\tilde{x}_i \in \mathbb{R}^d$, the **most important direction** $\vec{v}$ is the one that

1. **Minimizes** the mean squared orthogonal error of projections onto it,

$$J(\vec{v}) = \frac{1}{n}\sum_{i=1}^n \|\tilde{x}_i - \vec{p}_i\|^2$$

2. Equivalently, **maximizes** the variance of the projected values $\tilde{x}_i \cdot \vec{v}$, i.e. **maximizes**

$$\text{PV}(\vec{v}) = \frac{1}{n}\sum_{i=1}^n (\tilde{x}_i \cdot \vec{v})^2$$

Minimizing orthogonal error is equivalent to maximizing variance: don't forget this. In the widget below, drag the slider to see this duality in action: the shorter the orthogonal errors tend to be, the more spread out the orange points tend to be (meaning their variance is large). When the orange points are close together, the orthogonal errors tend to be quite large.

Note that the scalars $\tilde{x}_1 \cdot \vec{v}, \tilde{x}_2 \cdot \vec{v}, \ldots, \tilde{x}_n \cdot \vec{v}$ are just the components of the vector $\tilde{X}\vec{v}$! Multiplying $\tilde{X}$ by $\vec{v}$ returns a vector that contains the dot products of each row of $\tilde{X}$ (which are the $\tilde{x}_i$'s) with $\vec{v}$. This means that an equivalent formula for the projected variance is

$$\text{PV}(\vec{v}) = \frac{1}{n}\|\tilde{X}\vec{v}\|^2$$

This equivalent formulation will allow us to employ gradient rules from Chapter 8.2 to find the best direction.

The Rayleigh Quotient. Our optimization problem is

$$\text{maximize } \|\tilde{X}\vec{v}\|^2 \quad \text{subject to } \|\vec{v}\| = 1$$

I've dropped the constant factor of $\frac{1}{n}$ for convenience; it won't change the answer.

This is a **constrained** optimization problem. The constraint matters because we're trying to find a direction, not an arbitrarily long vector: if we were allowed to scale $\vec{v}$ however we wanted, then $\|\tilde{X}\vec{v}\|^2$ could be made as large as we wanted just by multiplying $\vec{v}$ by a huge constant.

Constrained optimization problems are usually harder than unconstrained ones. In an unconstrained problem, a standard strategy is to compute a gradient and set it equal to $\vec{0}$. With a constraint like $\|\vec{v}\| = 1$, that simple strategy does not apply directly, since the maximizer of $\|\tilde{X}\vec{v}\|^2$ subject to $\|\vec{v}\| = 1$ might not have a zero gradient.

Fortunately, this particular constrained problem can be converted into an equivalent unconstrained one. Define

$$f(\vec{v}) = \frac{\|\tilde{X}\vec{v}\|^2}{\|\vec{v}\|^2}$$

for all non-zero vectors $\vec{v}$. This function is called a **Rayleigh quotient**, as we discussed briefly in Chapter 9.6.

Why is maximizing $f(\vec{v})$ with no constraints equivalent to maximizing $\|\tilde{X}\vec{v}\|^2$ subject to $\|\vec{v}\| = 1$? The short answer is that $f(\vec{v})$ is invariant to scaling $\vec{v}$ by a constant factor. To see what I mean, suppose that c is a non-zero constant. Then,

$$f(c\vec{v}) = \frac{\|\tilde{X}(c\vec{v})\|^2}{\|c\vec{v}\|^2} = \frac{c^2\|\tilde{X}\vec{v}\|^2}{c^2\|\vec{v}\|^2} = f(\vec{v})$$

So $f(\vec{v})$ depends only on direction, not on length. Equivalently, every non-zero $\vec{v}$ gives the same value as the corresponding unit vector $\frac{\vec{v}}{\|\vec{v}\|}$, who makes the denominator in $f(\vec{v})$ equal to 1. That means maximizing $\|\tilde{X}\vec{v}\|^2$ subject to $\|\vec{v}\| = 1$ is equivalent to maximizing $f(\vec{v})$ over all non-zero vectors $\vec{v}$.

This is easier, because once the constraint has been absorbed into the denominator, we can treat f as an ordinary vector-to-scalar function and find its critical points by setting the gradient equal to $\vec{0}$.

In our setting,

$$f(\vec{v}) = \frac{\|\tilde{X}\vec{v}\|^2}{\|\vec{v}\|^2} = \frac{\vec{v}^T\tilde{X}^T\tilde{X}\vec{v}}{\vec{v}^T\vec{v}}$$

To find the critical points of f, it is cleaner to rewrite it as a product:

$$f(\vec{v}) = \left(\|\tilde{X}\vec{v}\|^2\right)\left(\|\vec{v}\|^{-2}\right)$$

Let

$$g(\vec{v}) = \|\tilde{X}\vec{v}\|^2 = \vec{v}^T\tilde{X}^T\tilde{X}\vec{v} \quad h(\vec{v}) = \|\vec{v}\|^{-2} = (\vec{v}^T\vec{v})^{-1}$$

so that $f(\vec{v}) = g(\vec{v})h(\vec{v})$. Since $\tilde{X}^T\tilde{X}$ is symmetric,

$$\nabla g(\vec{v}) = 2\tilde{X}^T\tilde{X}\vec{v}$$

Also, by the chain rule,

$$\nabla h(\vec{v}) = -2\frac{\vec{v}}{\|\vec{v}\|^4}$$

Now the product rule gives

$$\nabla f(\vec{v}) = g(\vec{v})\nabla h(\vec{v}) + h(\vec{v})\nabla g(\vec{v})$$

which simplifies to

$$\begin{aligned}
\nabla f(\vec{v}) &= g(\vec{v})\nabla h(\vec{v}) + h(\vec{v})\nabla g(\vec{v}) \\
&= \vec{v}^T\tilde{X}^T\tilde{X}\vec{v}\left(-2\frac{\vec{v}}{\|\vec{v}\|^4}\right) + (\vec{v}^T\vec{v})^{-1}\left(2\tilde{X}^T\tilde{X}\vec{v}\right) \\
&= -2\frac{\vec{v}^T\tilde{X}^T\tilde{X}\vec{v}}{\|\vec{v}\|^4}\vec{v} + 2\frac{\tilde{X}^T\tilde{X}\vec{v}}{\|\vec{v}\|^2} \\
&= -2\frac{f(\vec{v})}{\|\vec{v}\|^2}\vec{v} + 2\frac{\tilde{X}^T\tilde{X}\vec{v}}{\|\vec{v}\|^2} \\
&= \frac{2}{\|\vec{v}\|^2}\left(\tilde{X}^T\tilde{X}\vec{v} - f(\vec{v})\vec{v}\right)
\end{aligned}$$

If $\vec{v}$ maximizes $f(\vec{v})$, then it must be a critical point, so $\nabla f(\vec{v}) = \vec{0}$. Therefore,

$$\tilde{X}^T\tilde{X}\vec{v} - f(\vec{v})\vec{v} = \vec{0}$$

which means

$$\tilde{X}^T\tilde{X}\vec{v} = f(\vec{v})\vec{v}$$

This says exactly that $\vec{v}$ is an eigenvector of $\tilde{X}^T\tilde{X}$, and that its eigenvalue is $f(\vec{v})$. If $\vec{v}$ maximizes – or minimizes – $f(\vec{v})$, then it must be an eigenvector of $\tilde{X}^T\tilde{X}$.

So, to make $f(\vec{v})$ as large as possible, we should choose the eigenvector whose eigenvalue is as large as possible. In other words, **the best direction is the eigenvector of $\tilde{X}^T\tilde{X}$ corresponding to its largest eigenvalue!**

Alternative Derivation

There is another way to prove the same result that uses the spectral theorem more directly.
Let $\lambda_1 \geq \lambda_2 \geq \cdots \geq \lambda_d$ be the eigenvalues of $\tilde{X}^T\tilde{X}$, and let $\vec{v}_1, \vec{v}_2, \ldots, \vec{v}_d$ be corresponding unit eigenvectors.
Since $\tilde{X}^T\tilde{X}$ is symmetric, these eigenvectors form an orthonormal basis of $\mathbb{R}^d$.
Now suppose $\|\vec{v}\| = 1$. Then

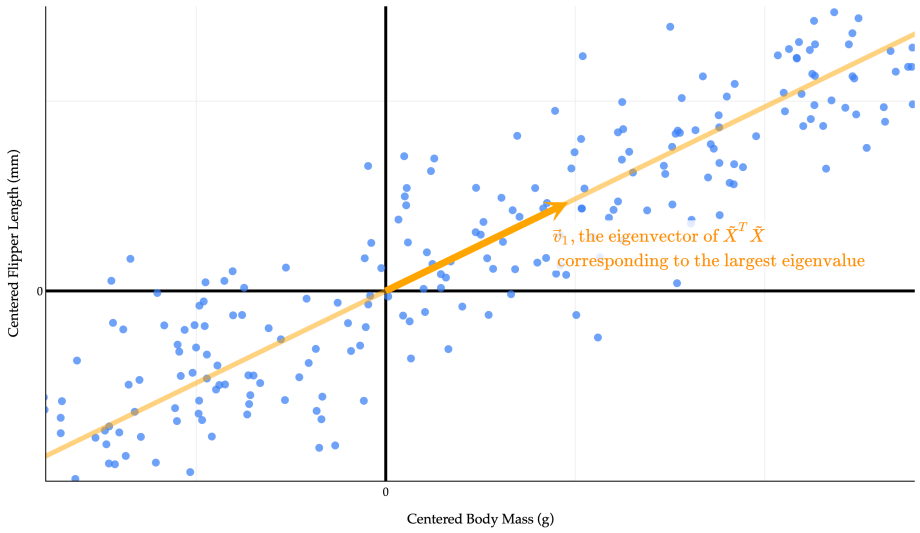
$$\vec{v} = c_1\vec{v}_1 + c_2\vec{v}_2 + \cdots + c_d\vec{v}_d, \qquad c_1^2 + c_2^2 + \cdots + c_d^2 = 1$$

Using the fact that $\tilde{X}^T\tilde{X}\vec{v}_i = \lambda_i\vec{v}_i$, we get

$$\begin{aligned} f(\vec{v}) &= \vec{v}^T\tilde{X}^T\tilde{X}\vec{v} \\ &= \lambda_1 c_1^2 + \lambda_2 c_2^2 + \cdots + \lambda_d c_d^2 \end{aligned}$$

So $f(\vec{v})$ is a weighted average of the eigenvalues of $\tilde{X}^T\tilde{X}$. A weighted average can never be larger than the largest value being averaged, so $f(\vec{v}) \leq \lambda_1$, with equality when $\vec{v} = \vec{v}_1$ (or $-\vec{v}_1$). This again shows that the maximizing direction is the eigenvector corresponding to the largest eigenvalue.

What does this really mean geometrically?



It says that the line that maximizes the variance of the projected data - which is also the line that minimizes the mean squared orthogonal error - is **spanned by the eigenvector of $\tilde{X}^T\tilde{X}$ corresponding to the largest eigenvalue of $\tilde{X}^T\tilde{X}$.**

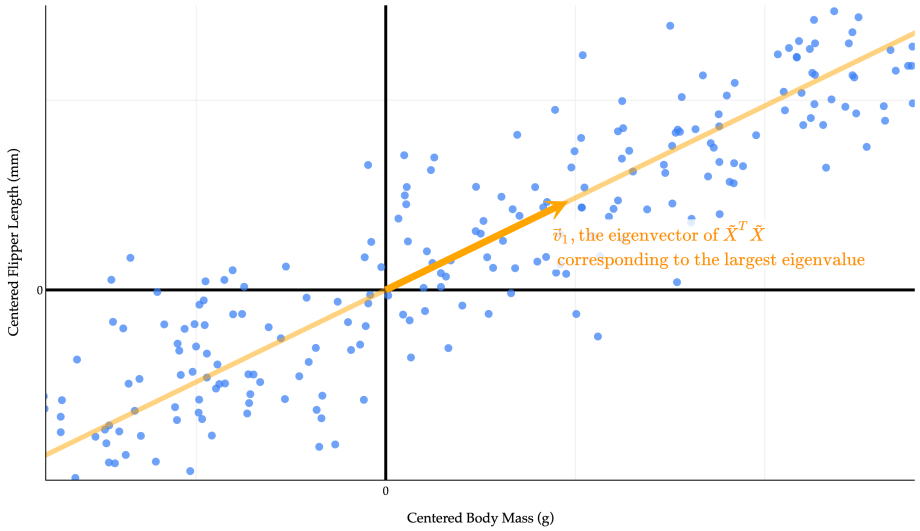
So, we've solved the "best direction" problem. But what can we do with it? And how does this relate to the singular value decomposition, which we discussed in Chapters 10.1 and 10.2 but didn't touch at all here? Let's keep reading...

10.4. Principal Components Analysis

In Chapter 10.3, we found the best direction for representing data by minimizing orthogonal projection error (equivalently, maximizing projected variance). We now use that result to define principal components and connect them to the SVD.

Principal Components

We've discovered that the single best direction to project the data onto is the eigenvector of $\tilde{X}^T\tilde{X}$ corresponding to the largest eigenvalue. Let's call this eigenvector $\vec{v}_1$, as I do in the diagram below. More $\vec{v}$'s are coming.



$$\text{PV}(\vec{v}) = \frac{1}{n}\|\tilde{X}\vec{v}\|^2 \implies \vec{v}^* = \vec{v}_1$$

The values of **the first principal component**, PC_1 (i.e. new feature 1), come from projecting each row of $\tilde{X}$ onto $\vec{v}_1$:

$$\text{PC}_1 = \begin{bmatrix} \tilde{x}_1 \cdot \vec{v}_1 \\ \tilde{x}_2 \cdot \vec{v}_1 \\ \vdots \\ \tilde{x}_n \cdot \vec{v}_1 \end{bmatrix} = \tilde{X}\vec{v}_1$$

This projection of our data onto the line spanned by $\vec{v}_1$ is the linear combination of the original features that captures the largest possible amount of variance in the data.

(Whenever you see "principal component", you should think "new feature".)

Remember, one of our initial goals was to find **multiple** principal components that were **uncorrelated** with each other, meaning their correlation coefficient r is 0. We've found the first principal component, PC_1, which came from projecting onto the best direction. This is the single best linear combination of the original features we can come up with.

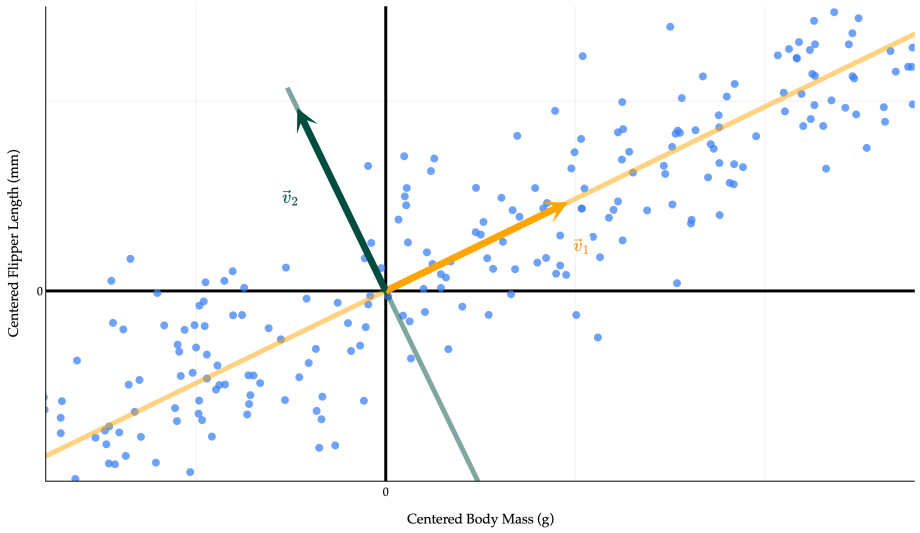
Let's take a greedy approach. I now want to find the next best principal component, PC_2, which should be uncorrelated with PC_1. PC_2 should capture all of the remaining variance in the data that PC_1 couldn't capture.

Intuitively, since our dataset is 2-dimensional, together, PC_1 and PC_2 should contain the same information as the original two features.

PC_2 comes from projecting onto the best direction **among all directions that are orthogonal to** $\vec{v}_1$. It can be shown that this "second-best direction" is the eigenvector $\vec{v}_2$ of $\tilde{X}^T\tilde{X}$ corresponding to the **second largest eigenvalue** of $\tilde{X}^T\tilde{X}$.

$$PC_2 = \tilde{X}\vec{v}_2$$

In other words, the vector $\vec{v}$ that maximizes $f(\vec{v}) = \frac{\|\tilde{X}\vec{v}\|^2}{\|\vec{v}\|^2}$ subject to the constraint that $\vec{v}$ is orthogonal to $\vec{v}_1$, is $\vec{v}_2$. The proof of this is beyond the scope of what we'll discuss here, as it involves some constrained optimization theory.



Why is $\vec{v}_2$ orthogonal to $\vec{v}_1$? They are both eigenvectors of $\tilde{X}^T\tilde{X}$ corresponding to different eigenvalues, so they must be orthogonal, thanks to the spectral theorem (which applies because $\tilde{X}^T\tilde{X}$ is symmetric). Remember that while any vector on the line spanned by $\vec{v}_2$ is also an eigenvector of $\tilde{X}^T\tilde{X}$ corresponding to the second largest eigenvalue, we pick the specific $\vec{v}_2$ that is a unit vector.

$\vec{v}_2$ and $\vec{v}_1$ being orthogonal means that $\tilde{X}\vec{v}_2$ and $\tilde{X}\vec{v}_1$ are orthogonal, too. But **because** the columns of $\tilde{X}$ are mean centered, we can show that this implies that the correlation between PC_1 and PC_2 is 0. I'll save the algebra for now, but see if you can work this out yourself. You've done similar proofs in homeworks already.

The SVD Returns. The moment I said that $\vec{v}_1$ is an eigenvector of $\tilde{X}^T\tilde{X}$, something should have been ringing in your head: $\vec{v}_1$ is a **singular vector** of $\tilde{X}$! Recall, if

$$\tilde{X} = \underbrace{\begin{bmatrix} | & \cdots & | & | & \cdots & | \\ \vec{u}_1 & \cdots & \vec{u}_r & \vec{u}_{r+1} & \cdots & \vec{u}_n \\ | & \cdots & | & | & \cdots & | \end{bmatrix}}_{U} \underbrace{\begin{bmatrix} \sigma_1 & & & 0 & \cdots & 0 \\ & \ddots & & \vdots & & \vdots \\ & & \sigma_r & 0 & \cdots & 0 \\ 0 & \cdots & 0 & 0 & \cdots & 0 \\ \vdots & & \vdots & \vdots & & \vdots \\ 0 & \cdots & 0 & 0 & \cdots & 0 \end{bmatrix}}_{\Sigma} \underbrace{\begin{bmatrix} — & \vec{v}_1^T & — \\ & \vdots & \\ — & \vec{v}_r^T & — \\ — & \vec{v}_{r+1}^T & — \\ & \vdots & \\ — & \vec{v}_d^T & — \end{bmatrix}}_{V^T}$$

is the singular value decomposition of $\tilde{X}$, then the columns of V are the eigenvectors of $\tilde{X}^T\tilde{X}$, all of which are orthogonal to each other and have unit length.

We arranged the components in the singular value decomposition in decreasing order of singular values of $\tilde{X}$, which are the square roots of the eigenvalues of $\tilde{X}^T\tilde{X}$.

$$\sigma_i = \sqrt{\lambda_i}, \quad \text{where } \lambda_i \text{ is the } i\text{-th largest eigenvalue of } \tilde{X}^T\tilde{X}$$

So,

- the first column of V is $\vec{v}_1$, the "best direction" to project the data onto,
- the second column of V is $\vec{v}_2$, the "second-best direction" to project the data onto,
- the third column of V is $\vec{v}_3$, the "third-best direction" to project the data onto,
- and so on.

Finding Principal Components

In general, principal component j comes from multiplying $\tilde{X}$ by the j-th column of V, where $\tilde{X} = U\Sigma V^T$ is the SVD of $\tilde{X}$, and $\tilde{X}$ is the mean-centered version of X.

$$\text{PC}_j = \tilde{X}\vec{v}_j$$

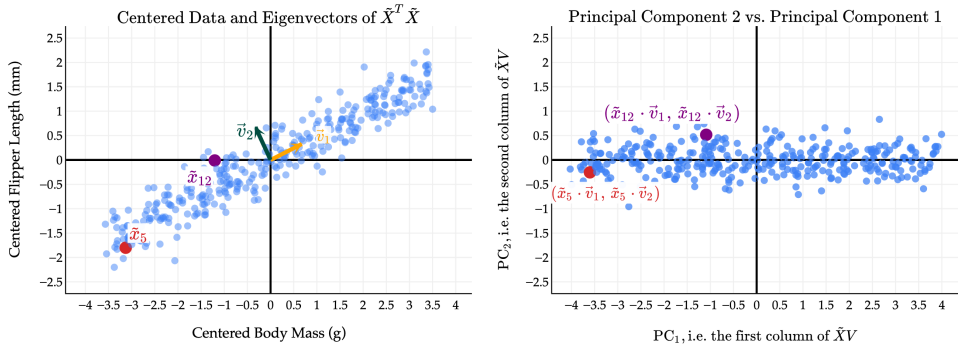
To find all principal components at once, all we need to do is multiply $\tilde{X}$ by the entire matrix V. The resulting matrix is sometimes called the principal component matrix.

$$\tilde{X}V = \begin{bmatrix} | & | & & | \\ \text{PC}_1 & \text{PC}_2 & \cdots & \text{PC}_d \\ | & | & & | \end{bmatrix}$$

Equivalently, $\tilde{X}V = U\Sigma$.
The process of creating principal components is called **principal component analysis**.

Let's put this in context with a few examples.

Example: From $\mathbb{R}^2$ to $\mathbb{R}^2$. Let's start with the 2-dimensional dataset of flipper length vs. body mass from the `penguins` dataset.



The first thing you should notice is that while the original data points seem to have some positive correlation, **the principal components are uncorrelated**! This is a good thing; it's what we wanted as a design goal. In effect, by converting from the original features to principal components, we've rotated the data to remove the correlation between the features.

I have picked two points to highlight, points 5 and 12 in the original data. The coloring in red and purple is meant to show you how original (centered) data points translate to points in the principal component space.

Notice the scale of the data: PC_1 axis is much longer than the PC_2 axis, since the first principal component captures much more variance than the second. We will make this notion - of the proportion of variance captured by each principal component - more precise soon.

(You'll notice that the body mass and flipper length values on the left are much smaller than in the original datasets; in the original dataset, the body mass values were in the thousands, which distorted the scale of the plot and made it hard to see that the two eigenvectors are indeed orthogonal.)

Example: Starting from $\mathbb{R}^3$. The real power of PCA reveals itself when we start with high-dimensional data. Suppose we start with three of the features in the `penguins` dataset: `bill_depth_mm`, `flipper_length_mm`, and `body_mass_g` - and want to reduce the dimensionality of the data to 1 or 2. Points are colored by their `species`.

Observe that penguins of the same species tend to be clustered together. This, alone, has nothing to do with PCA: we happen to have this information, so I've included it in the plot.

If X is the 333×3 matrix of three features, and $\tilde{X}$ is the mean-centered version, the best directions in which to project the data are the columns of V in $\tilde{X} = U\Sigma V^T$.

`X_three_features`

	bill_depth_mm	flipper_length_mm	body_mass_g
0	18.7	181.0	3750.0
1	17.4	186.0	3800.0
2	18.0	195.0	3250.0
4	19.3	193.0	3450.0
5	20.6	190.0	3650.0
...	...	...	...
338	13.7	214.0	4925.0
340	14.3	215.0	4850.0
341	15.7	222.0	5750.0
342	14.8	212.0	5200.0
343	16.1	213.0	5400.0

333 rows × 3 columns

```
X_three_features.mean()
```

```
bill_depth_mm          17.164865
flipper_length_mm     200.966967
body_mass_g          4207.057057
dtype: float64
```

```
# This is the mean -centered version of X_three_features!
X_three_features - X_three_features.mean()
```

	bill_depth_mm	flipper_length_mm	body_mass_g
0	1.535135	-19.966967	-457.057057
1	0.235135	-14.966967	-407.057057
2	0.835135	-5.966967	-957.057057
4	2.135135	-7.966967	-757.057057
5	3.435135	-10.966967	-557.057057
...	...	...	...
338	-3.464865	13.033033	717.942943
340	-2.864865	14.033033	642.942943
341	-1.464865	21.033033	1542.942943
342	-2.364865	11.033033	992.942943
343	-1.064865	12.033033	1192.942943

333 rows × 3 columns

```
X_three_features_centered = X_three_features - X_three_features.mean()
u, s, vt = np.linalg.svd(X_three_features_centered)
```

Now, the rows of V^T (`vt`), which are the columns of V (`vt.T`), contain the best directions.

```
vt
```

```
array([[ -1.15433983e -03,  1.51946036e -02,  9.99883889e -01],
       [ 1.02947493e -01, -9.94570148e -01,  1.52327042e -02],
       [ 9.94686122e -01,  1.02953123e -01, -4.16174416e -04]])
```

It's important to remember that these "best directions" are nothing more than linear combinations of the original features. Since the first row of V^T is $\begin{bmatrix}-0.00115 & 0.01519 & 0.99988\end{bmatrix}$, the first principal component is

$$\text{PC } 1_i = -0.00115 \cdot \text{bill depth}_i + 0.01519 \cdot \text{flipper length}_i + 0.99988 \cdot \text{body mass}_i$$

while the second is

$$\text{PC } 2_i = 0.10294 \cdot \text{bill depth}_i - 0.99457 \cdot \text{flipper length}_i + 0.01523 \cdot \text{body mass}_i$$

and third is

$$\text{PC } 3_i = 0.99468 \cdot \text{bill depth}_i + 0.10295 \cdot \text{flipper length}_i - 0.00042 \cdot \text{body mass}_i$$

where $i = 1, 2, \ldots, 333$ is the index of the penguin.

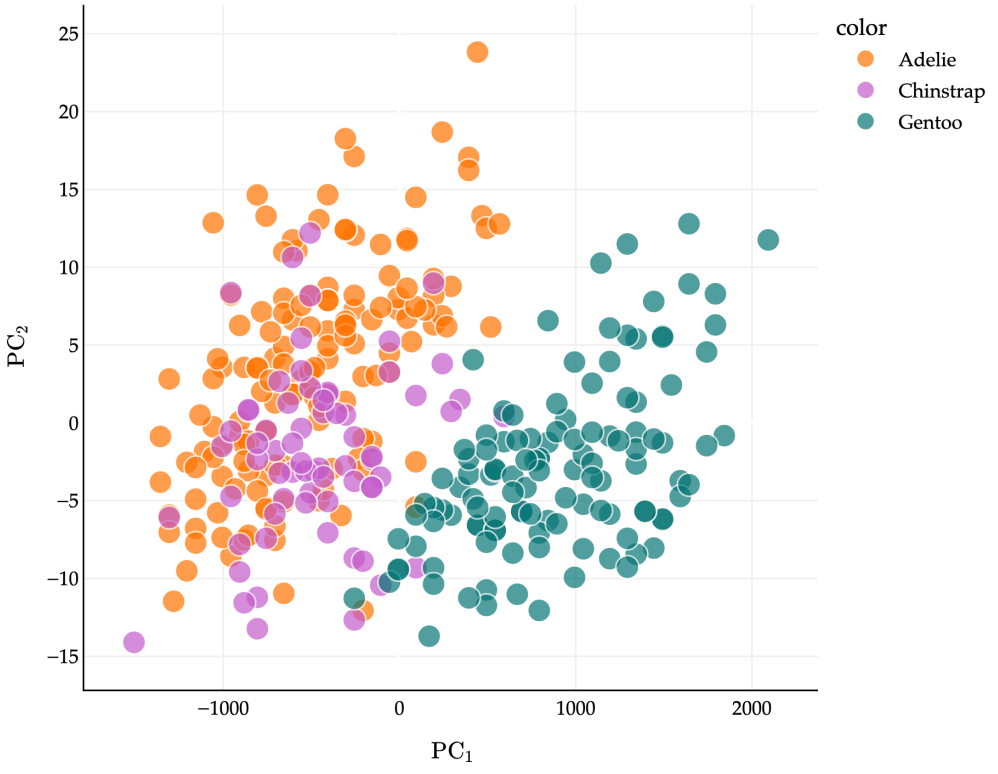
To compute all three of these principal components at once, for every penguin, we just need to compute $\tilde{X}V$.

```
pcs = X_three_features_centered @ vt.T
pcs
```

	0	1	2
0	-457.309150	13.054373	-0.338469
1	-407.237482	8.709325	-1.137604
2	-957.037562	-8.558025	0.614682
4	-757.092674	-3.388512	1.618633
5	-557.162981	2.775571	2.519631
...	...	...	...
338	718.061613	-2.382752	-2.403451
340	643.084824	-4.458007	-1.672473
341	1543.085070	2.433563	0.066202
342	992.998024	3.908624	-1.629651
343	1192.988496	6.094426	-0.316840

333 rows × 3 columns

Let's plot the first two principal components: that is, for each of our 333 penguins, we'll plot their value of $\tilde{X}\vec{v}_1$ on the x-axis and their value of $\tilde{X}\vec{v}_2$ on the y-axis.



This is the best 2-dimensional projection of our 3-dimensional scatter plot. And here's the kicker: **penguins of the same species STILL tend to be clustered together in the principal component space!**

What this tells us is that our technique for taking linear combinations of the original features is good at preserving the important information in the original dataset. We went from writing down 3 numbers per penguin to 2, but it seems that we didn't lose much information per penguin.

What do I mean by "much important information"? Let's make this idea more precise.

Explained Variance

The goal of PCA is to find new features - principal components - that capture as much of the variation in the data as possible, while being uncorrelated with each other.

PCA isn't foolproof: it works better on some datasets than others. If the features we're working with are already uncorrelated, PCA isn't useful. And, even for datasets that are well suited for PCA, we need a systematic way to decide how many principal components to use. So far, we've often used 1 or 2 principal components for the purposes of visualization, but in general, we need a more systematic approach. What if we start with an $n \times 50$ matrix and want to decide how many principal components to compute?

Let's define the **total variance** of an $n \times d$ matrix X as the sum of the variances of the columns of X. If $x_i^{(j)}$ is the i-th value of the j-th feature, and μ_j is the mean of the j-th feature, then the total variance of X is

$$\text{total variance} = \sum_{j=1}^{d} \left(\frac{1}{n} \sum_{i=1}^{n} (x_i^{(j)} - \mu_j)^2 \right)$$

Let's compute this for `X_three_features`.

```
X_three_features.var(ddof=0) # The variances of the individual columns of X.
```

```
bill_depth_mm             3.866243
flipper_length_mm       195.851762
body_mass_g          646425.423171
dtype: float64
```

```
X_three_features.var(ddof=0).sum()
```

```
646625.1411755901
```

So, the total variance of `X_three_features` is approximately 646625.

This is also equal to the **sum** of the variances of the columns of $\tilde{X}V$, i.e. the sum of the variances of the principal components!

```
# These three numbers are DIFFERENT than the numbers above,
# but their sum is the same.
(X_three_features_centered @ vt.T).var(ddof=0)
```

```
0    646575.552578
1        47.055776
2         2.532822
dtype: float64
```

```
(X_three_features_centered @ vt.T).var(ddof=0).sum() # Same sum as before.
```

```
646625.1411755901
```

Why? If we create the same number of principal components as we have original features, we haven't lost any information - we've just written the same data in a different basis. The goal, though, is to pick a number of principal components that is relatively small (smaller than the number of original features), but still captures most of the variance in the data.

Each new principal component - that is, each column of $\tilde{X}V$ - captures some amount of this total variance. What we'd like to measure is the **proportion** (that is, fraction, or percentage) of the total variance that each principal component captures. The first principal component captures the most variance, since it corresponds to the direction in which the data varies the most. The second principal component captures the second-most variance, and so on. But **how much** variance does the first, second, or j-th principal component capture?

Recall from earlier that the variance of the data projected onto a vector $\vec{v}$ is given by

$$\text{PV}(\vec{v}) = \frac{1}{n}\|\tilde{X}\vec{v}\|^2$$

$\vec{v}_1$ maximizes this quantity, which is why the first principal component is the projection of the data onto $\vec{v}_1$; $\vec{v}_2$ maximizes the quantity subject to being orthogonal to $\vec{v}_1$, and so on.

Then, the variance of PC_1 is whatever we get back when we plug $\vec{v}_1$ into the formula above, and in general, the variance of PC_j is $\text{PV}(\vec{v}_j)$ (again, where $\vec{v}_j$ is the j-th column of V in the SVD of $\tilde{X}$). Observe that if $\vec{v}_j$ is the j-th column of V in $\tilde{X} = U\Sigma V^T$, then,

$$\text{PV}(\vec{v}_j) = \frac{1}{n}\|\tilde{X}\vec{v}_j\|^2 = \frac{1}{n}\|\sigma_j\vec{u}_j\|^2 = \frac{1}{n}\sigma_j^2\|\vec{u}_j\|^2 = \frac{\sigma_j^2}{n}$$

Here, we used the ever important fact that $\tilde{X}\vec{v}_j = \sigma_j\vec{u}_j$, where σ_j is the j-th singular value of $\tilde{X}$ and $\vec{u}_j$ is the j-th column of U in the SVD of $\tilde{X}$.

What this tells us is that **the variance of the j-th principal component is $\frac{\sigma_j^2}{n}$**.

$$\boxed{\text{variance of PC } j = \frac{\sigma_j^2}{n}}$$

This is a beautiful result - it tells us that the variance of the j-th principal component is simply the square of the j-th singular value of $\tilde{X}$, divided by n. **The σ_j's in the SVD represent the amount that the data varies in the direction of $\vec{v}_j$!** We don't need any other fancy information to compute the variance of the principal components; we don't need to know the individual principal component values, or have access to a `variance` method in code.

```
u, s, vt = np.linalg.svd(X_three_features_centered)
s
```

```
array([14673.43378383,   125.1781673 ,    29.04185933])
```

```
np.set_printoptions(precision=8)
```

```
n = X_three_features_centered.shape[0]
```

```
variances_of_pcs = s ** 2 / n
np.set_printoptions(suppress=True)
variances_of_pcs
```

```
array([646575.55257751,     47.05577648,      2.5328216 ])
```

Above, we see the exact same values as if we computed the variances of the principal components directly from the data.

```
(X_three_features_centered @ vt.T).var(ddof=0)
```

```
0    646575.552578
1        47.055776
2         2.532822
dtype: float64
```

Since the total variance of X is the sum of the variances of its principal components, the total variance of X is then the sum of $\frac{\sigma_j^2}{n}$ over all $j = 1, 2, \ldots, r$, where $r = \text{rank}(X)$. (Remember that if $k > r$, then $\sigma_k = 0$.)

$$\text{total variance} = \sum_{j=1}^{r} \frac{\sigma_j^2}{n}$$

So, if we single out just one principal component, how much of the overall variation in X does it capture? The answer is given by the **proportion of variance explained** by the j-th principal component:

$$\text{proportion of variance explained by PC } j = \frac{\sigma_j^2}{\sum_{k=1}^{r} \sigma_k^2}$$

This is a number between 0 and 1, which we can interpret as a percentage.

```
s # The singular values of X_three_features.
```

```
array([14673.43378383,   125.1781673 ,    29.04185933])
```

```
(s ** 2) / (s ** 2).sum() # The proportions of variance explained by each principal component.
```

```
array([0.99992331, 0.00007277, 0.00000392])
```

The above tells us that in `X_three_features`, the first principal component captures 99.99% of the variance in the data! There's very little information lost in projecting this 3-dimensional dataset into 1-dimensional space.

Often, the proportions above are visualized in a **scree plot**, as you'll see in Homework 11. Scree plots allow us to visually decide the number of principal components to keep, based on where it seems like we've captured most of the variation in the data. We'll work on a related example in live lecture.

The PCA Recipe

Let's briefly summarize what I'll call the "PCA recipe", which describes how to find the principal components (new features) of a dataset.

1. Starting with an $n \times d$ matrix X of n data points in d dimensions, mean-center the data by subtracting the mean of each column from itself. The new matrix is $\tilde{X}$.
2. Compute the singular value decomposition of $\tilde{X}$: $\tilde{X} = U\Sigma V^T$. The columns of V (rows of V^T) describe the directions of maximal variance in the data.
3. Principal component j comes from multiplying $\tilde{X}$ by the j-th column of V.

$$\text{PC}_j = \tilde{X}\vec{v}_j = \sigma_j \vec{u}_j$$

   The resulting principal components - which are the columns of $\tilde{X}V$ - are uncorrelated with one another.

4. The variance of PC_j is $\frac{\sigma_j^2}{n}$. The proportion of variance explained by PC_j is

$$\text{proportion of variance explained by PC } j = \frac{\sigma_j^2}{\sum_{k=1}^{r} \sigma_k^2}$$

In Chapter 10.5, to wrap up our course, we'll use this recipe on images of handwritten digits. Then, we'll feed the resulting principal components into a classifier.

10.5. Conclusion

In Chapter 10.4, I introduced principal components analysis (PCA), which is the process of creating new features (called principal components) that are linear combinations of the original features that

1. capture as much of the variation in the data as possible, and
2. are uncorrelated with each other.

The main use case of PCA so far has been for **dimensionality reduction** – taking a high-dimensional dataset and representing each point as a vector in a lower-dimensional space. The main example in Chapter 10.4 was the penguin dataset, where each penguin was originally described by four features (bill length, bill depth, flipper length, and body mass), but PCA allowed us to describe each penguin using just two features.

Handwritten Digits

At the very start of the course, in Chapter 1.1, I introduced you to the MNIST dataset (MNIST stands for Modified National Institute of Standards and Technology). To wrap up the course, I'd like to revisit this dataset.

The dataset contains 70,000 labeled grayscale images of handwritten digits, each of which is 28 pixels by 28 pixels. The dataset exists to train machine learning models to recognize handwritten digits. This is a **classification** task, rather than a regression task, which we spent the majority of the course studying.

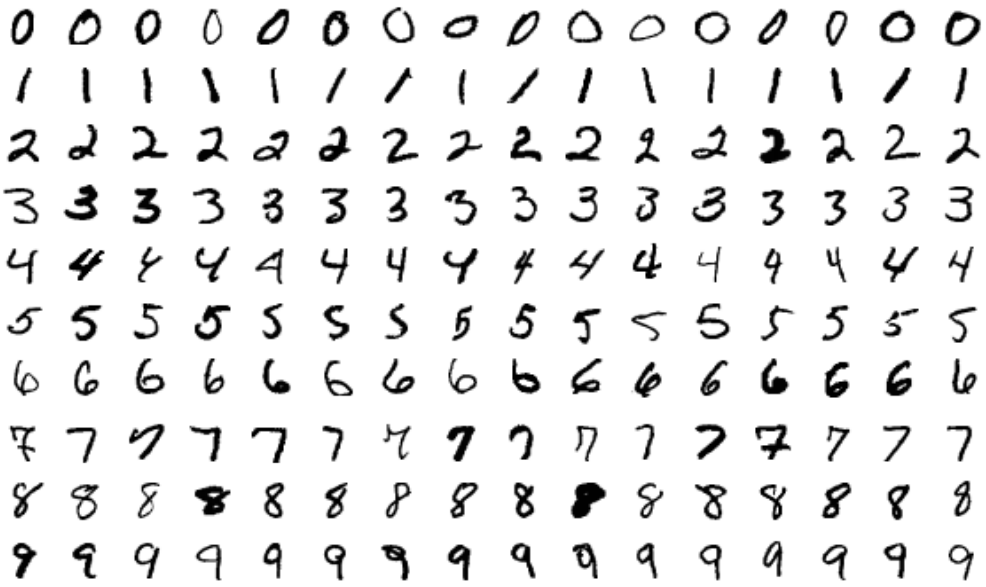


Figure 10.1: *
A few of the images in the MNIST dataset.

Each image is a 28×28 grid of pixels, and each of these pixels is an integer between 0 and 255, representing the pixel's intensity. In the example below, hover over any pixel to see its intensity value.

Each image can be stored as a vector in $\mathbb{R}^{784}$, since $28 \times 28 = 784$. A table containing all images, then, has 70,000 rows (one per image) and 784 columns (one per pixel).

	pixel1	pixel2	pixel3	pixel4	pixel5	pixel6	pixel7	pixel8	pixel9	pixel10	...	pixel775	pixel776	pixel777	pixel778	pixel779	pixel780	pixel781	pixel782	pixel783	pixel784
0	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0
2	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0
3	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0
4	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
69995	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0
69996	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0
69997	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0
69998	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0
69999	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0

70000 rows × 784 columns

Each row above corresponds to one flattened digit. While we can visualize individual images in the dataset one at a time, we can't visualize the entire dataset at once, since it's made up of vectors in $\mathbb{R}^{784}$. That's where PCA comes in!

PCA Returns

By creating principal components, we can represent each image using only 2 or 3 features, rather than 784.

Below, I've reduced the dimensionality of the 784-feature dataset to just 2 features. Rather than using `np.linalg.svd` as in Chapter 10.4, I've used `sklearn`'s PCA implementation, just to show you how it works.

```
pca_2 = PCA(n_components=2)
mnist_pca = pca_2.fit_transform(X)
```

What fraction of the variance in the full 784-dimensional data is captured by the first two principal components? It looks like about 17%, which is less than the ~99% we saw for the first two principal components in the penguin dataset. We're seeing a much lower proportion of variance explained here likely because it's hard to distinguish between 28×28 images with just two numbers per image.

```
# The proportion of variance explained by each principal component.
pca_2.explained_variance_ratio_
```

```
array([0.09746116, 0.07155445])
```

```
# The total variance explained by the first two principal components.
pca_2.explained_variance_ratio_.sum()
```

```
0.16901560509373448
```

In the scatter plot below, **each point corresponds to an image**. Its position is determined by its values in the first two principal components. Note that we've only included a small sample of the full 70,000 image dataset as to avoid crowding the plot (and slowing down your browser).

The values of principal component 1 and 2 **not** pixel intensities, as they don't range between 0 and 255. Instead, they are linear combinations of the original pixel intensities.

By construction, PCA did **not** use the labels when determining the placement of each image. All it knew about each image were the 784 pixel values. Still, the images are roughly clustered by their true digits, meaning that handwritten 0s tend to be placed near handwritten 0s, handwritten 1s near handwritten 1s, and so on.

Notice that even with just two features, the digits are roughly clustered by their true digit label. The 1's are clustered together in the top left, the 0's are clustered together in the right center, and so on. But, there's a fair amount of overlap.

Let's go one step further: in the graph below (which only uses a random sample of 200 points for speed), hovering over a point reveals the **full original 28 x 28 image itself**.

Zoom in on the graph, and look at any two points that are close to each other but have different labels. You'll run into cases like a 9 that looks like a 1, or 5 that looks like a 3. Again, with just these two features for each image, we retain a lot of information about the original image, which is remarkable!

Feature Maps. How are these principal components computed? As we saw in Chapter 10.4,

$$\text{PC}_j = \tilde{X}\vec{v}_j$$

where $\tilde{X}$ is the centered data matrix, and $\vec{v}_j$ is the jth eigenvector of $\tilde{X}^\top\tilde{X}$. In other words, $\vec{v}_j$ is the jth column of V in $\tilde{X} = U\Sigma V^T$, the SVD of $\tilde{X}$.

Here, $\vec{v}_j$ contains 784 values, one for each pixel in the original 28×28 image. Here's an idea: why don't we **visualize** $\vec{v}_j$ as a 28×28 image? Such a graph is called a **feature map**, as it shows how the original 784 features are combined to create each principal component.

Now, let's look at the scatter plot of the first two principal components again, with the digits colored by their true digit label. Notice that the 0's are in the far **right** while the 1's are in the far **left**. How does the above heat map explain this?

PCA Regression

Here's the last big idea: what if we use these new features as inputs to a model? We absolutely can. But what kind of predictive task is this: regression or classification? Classification, of course, because the goal is to predict **which digit** appears in the image.

A standard choice is **logistic regression**. Despite the name, logistic regression is a linear **classification** technique that builds on linear regression. Instead of predicting a numerical response directly, it predicts class probabilities and then turns those probabilities into decisions.

For handwritten digits, `sklearn` uses a multinomial version of logistic regression that predicts a probability for each digit 0 through 9. We won't focus on the math, the loss function, or the optimization details here; those are beyond our scope. The point is just that PCA can create new features, and a classifier can use those new features as inputs.

```
Pipeline(steps=[('pca',
                 PCA(n_components=2, random_state=245,
                     svd_solver='randomized')),
                ('logisticregression',
                 LogisticRegression(max_iter=5000, random_state=245))])
```

In a Jupyter environment, please rerun this cell to show the HTML representation or trust the notebook.
On GitHub, the HTML representation is unable to render, please try loading this page with nbviewer.org.

[ ] Pipeline?Documentation for PipelineiNot fitted

```
Pipeline(steps=[('pca',
                 PCA(n_components=2, random_state=245,
                     svd_solver='randomized')),
                ('logisticregression',
                 LogisticRegression(max_iter=5000, random_state=245))])
```

[ ] PCA?Documentation for PCA

```
PCA(n_components=2, random_state=245, svd_solver='randomized')
```

[ ] LogisticRegression?Documentation for LogisticRegression

```
LogisticRegression(max_iter=5000, random_state=245)
```

```
pipe.fit(X_train, y_train)
two_pc_accuracy = pipe.score(X_test, y_test)
two_pc_accuracy
```

```
0.4465
```

Even with just two principal components, the classifier reaches about 44.6% accuracy on the test set. That is far from perfect, but it is still much better than random guessing among 10 classes, which would only be correct about 10% of the time.

While the full MNIST dataset was too high-dimensional to visualize directly, it can be compressed into principal components, which can be visualized and then fed into a classifier. What a beautiful way to wrap up the course!

Mathematical Foundations

Appendix 1: Summation Notation and the Mean

This course will rely on your understanding of some ideas from calculus and high school algebra. The appendix of the course notes serves to review these ideas.

Sums and averages play an important role in machine learning. Here, we'll review the most relevant properties of summation notation, and use the arithmetic mean as a case study of sorts.

Introduction

Definition: Summation Notation

The $\sum$ symbol, read "sigma", is used to indicate a sum of a sequence. In general, if a and b are integers, and $x_1, x_2, \ldots$ is a collection of numbers, then:

$$\sum_{i=a}^{b} x_i = x_a + x_{a+1} + x_{a+2} + \ + x_{b-1} + x_b$$

Above, i is the *index of summation*.

For example, if we take $x_i = i^2$, then $\sum_{i=1}^{6} i^2$ represents the sum of the squares of all integers from 1 to 6:

$$\sum_{i=1}^{6} i^2 = 1^2 + 2^2 + 3^2 + 4^2 + 5^2 + 6^2$$

Notice that both the starting and ending indices (1 and 6, respectively) are included in the sum. Summation notation allows us to express sums conveniently – the left-hand side above is more compact than the right-hand side.

Often, we'll take the sum of the first n terms of a sequence. For example, the sum of the squares of the first n positive integers is:

$$\sum_{i=1}^{n} i^2$$

Note that the index of summation can be any variable name (i is just a typical choice). That is, $\sum_{j=1}^{n} j^2$, $\sum_{i=1}^{n} i^2$, and $\sum_{\text{zebra}=1}^{n} \text{zebra}^2$ all represent the same sum.

Summation notation can be thought of in terms of a `for`-loop. In Python, to compute the sum $\sum_{i=a}^{b} i^2$, we could write:

```
total = 0
for i in range(a, b + 1):
 total = total + i ** 2
```

As we mentioned above, the ending index is *inclusive* in summation notation. This is in contrast to Python, where the ending index is *exclusive*, which is why we provided `b + 1` as the second argument to the `range` function instead of `b`.

Activity 1

Activity 1

Identify what makes each of the following expressions invalid or ambiguous, and why.

- $\sum_{k=1}^{13} k^2 + k$
- $i \sum_{i=1}^{5} i$
- $\sum_{i=1}^{n} x_i + x_i$
- $\sum_{i=5}^{3} i^2$
- $\sum_{i=2}^{\pi} x_i$

Properties and Examples

To illustrate various properties of summation notation, we'll use the following fact:

$$\sum_{i=0}^{n} 2^i = 2^0 + 2^1 + 2^2 + \ldots + 2^n = 2^{n+1} - 1$$

The expression $2^{n+1} - 1$ is called a **closed form** for the summation. When closed forms exist, they make it easy to compute the value of a summation. You'll also notice that in addition to writing the sum using summation notation, I also showed the first few and last terms of the sum, with a ... to indicate that the pattern continues in between. As convenient as summation notation is, explicitly writing the first few and last terms of a sum can sometimes make it easier to understand what exactly is being summed.

For example, $\sum_{i=0}^{3} 2^i = 2^0 + 2^1 + 2^2 + 2^3 = 1 + 2 + 4 + 8 = 15$, which is indeed $2^{3+1} - 1 = 15$. The sequence $2^0, 2^1, 2^2, \ldots, 2^n$ is called a geometric sequence, and the resulting sum is called a geometric series.

For convenience, we'll define $S(n) = \sum_{i=0}^{n} 2^i$; again, $S(n) = 2^{n+1} - 1$.

The best way to learn through these examples is to try to solve them yourself before looking at the solution.

Example: Partial Sums. Determine the value of $\sum_{i=8}^{19} 2^i$. (Find an answer that doesn't involve summation notation or a sum over many terms.)

Solution

We can't use the formula for $S(n)$ directly, because the starting index is 8, not 0. But, if we look at the expansion of $S(19)$, we'll see that it contains the sum we're looking for:

$$\begin{aligned} S(19) &= \sum_{i=0}^{19} 2^i \\ &= 2^0 + 2^1 + 2^2 + \cdots + 2^{19} \\ &= (2^0 + 2^1 + 2^2 + \cdots + 2^7) + (2^8 + 2^9 + 2^{10} + \cdots + 2^{19}) \\ &= \underbrace{\sum_{i=0}^{7} 2^i}_{S(7)} + \underbrace{\sum_{i=8}^{19} 2^i}_{\text{what we're looking for}} \end{aligned}$$

So, the piece we're looking for is $S(19) - S(7)$, or:

$$\sum_{i=8}^{19} 2^i = S(19) - S(7) = (2^{20} - 1) - (2^8 - 1) = 2^{20} - 2^8$$

The specific value of the sum, $2^{20} - 2^8$, is $1048576 - 256 = 1048320$, if you're curious.

Example: Constant Multiples. Determine the value of $\sum_{i=0}^{7} 5 \cdot 2^i$.

Solution

$$\begin{aligned} \sum_{i=0}^{7} 5 \cdot 2^i &= 5 \cdot 2^0 + 5 \cdot 2^1 + 5 \cdot 2^2 + 5 \cdot 2^3 + 5 \cdot 2^4 + 5 \cdot 2^5 + 5 \cdot 2^6 + 5 \cdot 2^7 \\ &= 5(2^0 + 2^1 + 2^2 + 2^3 + 2^4 + 2^5 + 2^6 + 2^7) \\ &= 5 \sum_{i=0}^{7} 2^i \\ &= 5(2^8 - 1) \end{aligned}$$

This last line is what's most important; its specific value is $5 \cdot (2^8 - 1) = 5 \cdot 255 = 1275$.

Example: Sum of a Constant. Determine the value of $\sum_{i=7}^{20} 5$.

Solution

$$\begin{aligned}\sum_{i=7}^{20} 5 &= \underbrace{5+5+5+\cdots+5}_{14 \text{ times}} \\ &= 5 \cdot 14 \\ &= 70\end{aligned}$$

Where did 14 come from? The starting index is 7, and the ending index is 20. So, the number of terms is $20 - 7 + 1 = 14$.

Example: Shifting Indices. Determine the value of $\sum_{i=10}^{20} 2^{i-5}$.

Solution

As we did in the very first example, let's try and write the sum as a difference of two calls to $S(n)$. $S(n)$ involves the sum of terms of the form 2^i, and the sum we're looking for involves the terms 2^{i-5}, so we'll need to shift the indices.
When $i = 10$, $i - 5 = 5$, and when $i = 20$, $i - 5 = 15$. So, we can rewrite the sum in question as:

$$\sum_{i=10}^{20} 2^{i-5} = \sum_{i=5}^{15} 2^i$$

This looks like $S(15) - S(4)$, or $2^{16} - 1 - (2^5 - 1) = 2^{16} - 2^5 = 65536 - 32 = 65504$.

Example: Separating Sums. Given that $\sum_{i=1}^{n} i = \frac{n(n+1)}{2}$, determine the value of $\sum_{i=0}^{12}(2^i + 5i)$.

Solution

$$\begin{aligned}\sum_{i=0}^{12}(2^i + 5i) &= (2^0 + 5 \cdot 0) + (2^1 + 5 \cdot 1) + (2^2 + 5 \cdot 2) + \ldots + (2^{12} + 5 \cdot 12) \\ &= (2^0 + 2^1 + 2^2 + \ldots + 2^{12}) + (5 \cdot 0 + 5 \cdot 1 + 5 \cdot 2 + \ldots + 5 \cdot 12) \\ &= \sum_{i=0}^{12} 2^i + \sum_{i=0}^{12} 5i \\ &= 2^{13} - 1 + 5 \cdot \frac{12 \cdot 13}{2} \\ &= 8581\end{aligned}$$

The key here is that we can split the sum into two sums.

Key Takeaways. In the order they were introduced in the examples, here are some useful properties of summations:

1. Partial Sums:

$$\sum_{i=1}^{n} x_i = \sum_{i=1}^{k} x_i + \sum_{i=k+1}^{n} x_i$$

2. Constant Multiples:

$$\sum_{i=1}^{n} cx_i = c \sum_{i=1}^{n} x_i$$

3. Sum of a Constant:

$$\sum_{i=1}^{n} c = c \cdot n$$

4. Shifting Indices:

$$\sum_{i=k}^{n} x_i = \sum_{i=0}^{n-k} x_{i+k}$$

5. Separating Sums:

$$\sum_{i=1}^{n} (x_i + y_i) = \sum_{i=1}^{n} x_i + \sum_{i=1}^{n} y_i$$

Warning

Summation notation **does not distribute over multiplication or division**. That is,

$$\sum_{i=1}^{n} (x_i y_i) \neq \left(\sum_{i=1}^{n} x_i \right) \left(\sum_{i=1}^{n} y_i \right)$$

$$\sum_{i=1}^{n} \frac{x_i}{y_i} \neq \frac{\sum_{i=1}^{n} x_i}{\sum_{i=1}^{n} y_i}$$

For more practice, work through Activity 2 (which has 5 sub-activities).

Activity 2

Activity 2

Activity 2.1

Given that $\sum_{i=1}^{n} i^2 = \frac{n(n+1)(2n+1)}{6}$, determine the value of $\sum_{j=3}^{10} (j^2 - 7j)$.

Activity 2.2

Show that $\sum_{n=1}^{100} \frac{1}{n(n+1)} = 1 - \frac{1}{101}$.
Hint: Try writing $\frac{1}{n(n+1)}$ as a difference of two fractions.

Activity 2.3
Recall, $n! = n \cdot (n-1) \cdot (n-2) \cdot \ldots \cdot 1$.
The Taylor Series expansion for the function $f(x) = e^x$, at the point $x = 0$, is given by:

$$e^x = \sum_{n=0}^{\infty} \frac{x^n}{n!}$$

Find a closed form expression for the infinite series:

$$\frac{1}{2!} - \frac{1}{3!} + \frac{1}{4!} - \frac{1}{5!} + \frac{1}{6!} - \frac{1}{7!} + \cdots$$

Activity 2.4
$\binom{n}{k}$, pronounced "n choose k", is the number of ways to choose k items from a set of n items.

$$\binom{n}{k} = \frac{n!}{k!(n-k)!}$$

For example, $\binom{5}{2} = 10$, because there are 10 ways to choose 2 items from a set of 5 items, and $\frac{5!}{2!(5-2)!} = \frac{120}{2 \cdot 6} = 10$.
Using the fact introduced in Activity 2.1, find a closed form expression for the following sum:

$$\sum_{k=2}^{n} \binom{k}{2}$$

Activity 2.5
Argue why the following equality holds (it'll be hard to prove this algebraically):

$$\sum_{k=0}^{n} \binom{n}{k} = 2^n$$

Mean and Standard Deviation

As I mentioned at the start of this section, we'll work with sums of data points quite frequently in this class. We'll often set up a problem by saying we have a sequence of n scalar[1] values, represented by $x_1, x_2, \ldots, x_n$. For instance, perhaps there are n students in this course, and x_i represents the height of student i.

Mean. The **mean**, or **average**, of all n values is given the symbol $\bar{x}$ (pronounced "x-bar") and is defined as follows:

$$\bar{x} = \frac{x_1 + x_2 + \ldots + x_n}{n} = \frac{1}{n} \sum_{i=1}^{n} x_i$$

[1] "Scalar" just means "individual number", as opposed to a vector or matrix which can contain multiple numbers, as we'll see later in the course.

You've likely seen this definition before. But, an often-forgotten property of the mean is that the **sum of the deviations from the mean is zero**. By that, I mean (no pun intended) that if you:

1. compute the mean of a sequence of numbers,
2. compute the *signed* difference between each number and the mean, and then
3. sum all of those differences, the result will be zero.

Let's first see this in action, then show why it is true in general. Suppose there are only 4 students in the class, with heights 72, 63, 68, and 65 inches. The mean of these heights is:

$$\bar{x} = \frac{72 + 63 + 68 + 65}{4} = 67$$

The deviations from the mean are:

$$\begin{aligned} 72 - 67 &= 5 \\ 63 - 67 &= -4 \\ 68 - 67 &= 1 \\ 65 - 67 &= -2 \end{aligned}$$

The sum of the four deviations, then, is:

$$5 + (-4) + 1 + (-2) = 0$$

So, the mean deviation from the mean is zero in this example.

This is also true in general. Precisely, I'm claiming that if $x_1, x_2, ..., x_{n-1}, x_n$ are any n numbers, and $\bar{x}$ is their mean, then $\sum_{i=1}^{n}(x_i - \bar{x}) = 0$.

Let's prove it:

$$\begin{aligned} \sum_{i=1}^{n}(x_i - \bar{x}) &= \sum_{i=1}^{n} x_i - \sum_{i=1}^{n} \bar{x} \\ &= \sum_{i=1}^{n} x_i - n\bar{x} \\ &= \sum_{i=1}^{n} x_i - n\left(\frac{x_1 + x_2 + \ldots + x_n}{n}\right) \\ &= \sum_{i=1}^{n} x_i - (x_1 + x_2 + \ldots + x_n) \\ &= 0 \end{aligned}$$

So, we've shown that the sum of the deviations from the mean is 0 in general. A consequence of this is that the positive deviations and negative deviations are equal in magnitude, since they need to cancel each other out. In the 72, 63, 68, 65 example, the positive deviations are 5 and 1 and the negative deviations are -4 and -2, and both have magnitude 6. As a result, the mean is sometimes thought of as the "balance point" of the dataset – the point at which the negative deviations are balanced by the positive deviations. More on this in Chapter 1.3.

Standard Deviation. Since the sum (and average) of deviations from the mean is 0, no matter the dataset, we can't use the average deviation from the mean to measure how far values tend to be from the mean. The average deviation will be 0, whether the dataset is tightly clustered or spread out.

To measure how far values in a dataset tend to deviate from their mean, then, we'll need to address the fact that some deviations are positive and some are negative. A common approach is to:

1. Compute the mean of the dataset
2. Compute the deviation of each value from the mean
3. Square each deviation
4. Take the average of the squared deviations

The result of this process is called the **variance**, denoted s^2 or σ^2; its square root is called the **standard deviation**. Following the above steps, the variance is given by:

$$\sigma^2 = \frac{1}{n}\sum_{i=1}^{n}(x_i - \bar{x})^2$$

(If you've read Chapter 1.2, you'll notice some similarities between this formula and the formula for mean squared error.)

Activity 3

Activity 3

What is the variance of the dataset 72, 63, 68, 65?

As a final note, the variance has a convenient, equivalent form:

$$\sigma^2 = \frac{1}{n}\sum_{i=1}^{n}(x_i - \bar{x})^2 = \frac{1}{n}\sum_{i=1}^{n}x_i^2 - \bar{x}^2$$

In English, we might say the variance is "the mean of the squares of x minus the square of the mean of x".

Your turn: prove this equivalent form of the variance.

Solution

$$\begin{aligned}
\sigma^2 &= \frac{1}{n}\sum_{i=1}^{n}(x_i - \bar{x})^2 \\
&= \frac{1}{n}\sum_{i=1}^{n}(x_i^2 - 2x_i\bar{x} + \bar{x}^2) \\
&= \frac{1}{n}\sum_{i=1}^{n}x_i^2 - \frac{2}{n}\sum_{i=1}^{n}x_i\bar{x} + \frac{1}{n}\sum_{i=1}^{n}\bar{x}^2 \\
&= \frac{1}{n}\sum_{i=1}^{n}x_i^2 - 2\bar{x}\frac{1}{n}\sum_{i=1}^{n}x_i + \bar{x}^2 \\
&= \frac{1}{n}\sum_{i=1}^{n}x_i^2 - 2\bar{x}^2 + \bar{x}^2 \\
&= \frac{1}{n}\sum_{i=1}^{n}x_i^2 - \bar{x}^2
\end{aligned}$$

Activity 4

Activity 4

Suppose the dataset $x_1, x_2, \ldots, x_n$ has mean $\bar{x}$ and variance σ^2.
Find the mean and variance of the dataset $-4x_1+3, -4x_2+3, \ldots, -4x_n+3$, and justify your answer rigorously using the definitions of mean and variance.

Appendix 2: Derivatives

Calculus is the study of **rates of change**, and without it, modern machine learning would not be possible. In many ways, machine learning is about **optimizing** quantities – making the *best* possible predictions, or making prediction errors as *small* as possible – and calculus is the tool that enables us to perform this optimization.

We'll address these lofty goals throughout the semester. Here, we will review key ideas from a first course in calculus (e.g. Math 115).

Tangent Lines

Suppose f is a function that takes in a single real number and outputs a single real number, i.e. $f : \mathbb{R} \to \mathbb{R}$.

If f is a function, then the **derivative** of f is another function, sometimes denoted f', such that $f'(x)$ is the "instantaneous" rate of change of f at the input x.

To understand what I mean by "instantaneous" rate of change, let's consider an example. Suppose $f(x) = \frac{1}{4}x^2 - 3$. The graph of f is shown below, in blue, along with a slider for input values of x. Drag the slider.

At any point x, the tangent line to the graph of f is the **best linear approximation** of the graph of f near x.

For instance, consider when $x = -4$. At $x = -4$, $f(-4) = \frac{1}{4}(-4)^2 - 3 = 1$.

The tangent line at $x = -4$ is the line that passes through the point $(-4, 1)$ that best approximates f near $x = -4$, among all other lines that pass through $(-4, 1)$. In the plot above, use your mouse to zoom in on the point $(-4, 1)$, and you'll see that when zoomed in, the tangent line and original function are very difficult to distinguish.

The Derivative is the Slope of the Tangent Line

Here's the million dollar connection. The slope of the tangent line at x **is, by definition**, the derivative of f at x.
In the example above, when $x = -4$, the slope of the tangent line is -2, which is the derivative of f at $x = -4$, i.e. $f'(-4) = -2$.

This intuitive definition of the derivative – as being the slope of the tangent line – is the most important way to think about the derivative. The formal definition is also important, and we'll get there next, but you should have enough context for this first activity.

Activity 1

Activity 1

Let $f(x) = \frac{1}{4}x^4 + \frac{1}{3}x^3 - x^2 + 2$.
Given that $f'(x) = x^3 + x^2 - 2x$, find the equation of the tangent line to $f(x)$ at the following points:

- $x = -3$
- $x = -1$
- $x = 1$

As a bonus exercise, try to verify that the provided formula for $f'(x)$ is correct, using your knowledge of derivatives from Calculus 1.

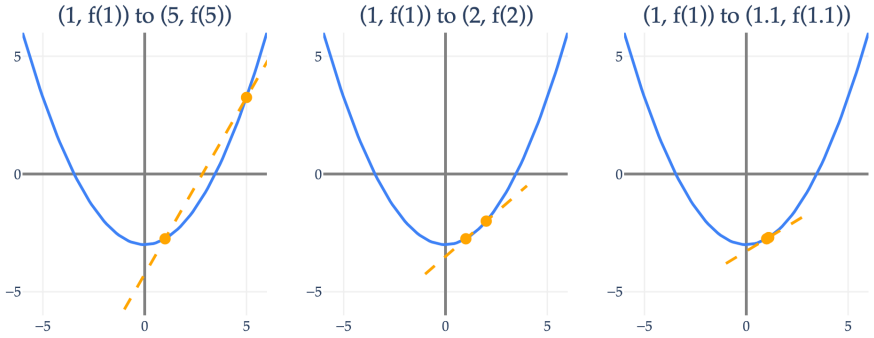
Formal Definition

Secant Lines. Let's review the more formal definition of the derivative. First, remember the general formula for the slope of the line between two points (x_1, y_1) and (x_2, y_2):

$$\frac{y_2 - y_1}{x_2 - x_1}$$

Let's say we're trying to find the slope of the tangent line at $x = a$, which is the line that passes through the point $(a, f(a))$ whose slope is the instantaneous rate of change of f at $x = a$. To find that instantaneous rate of change, we can find the slope of the line between $(a, f(a))$ and some other point $(b, f(b))$, where $b - a$ is as close to 0 as possible.

In the example below, as b approaches $a = 1$, the slope of the line between $(1, f(1))$ and $(b, f(b))$ approaches the slope of the tangent line at $x = 1$. Note that the formal name for the line between any two points on a function is a **secant line**.



Limits. Let's be more precise, using the idea of a **limit** from Calculus 1. Recall, $\lim_{x \to a} g(x) = L$ is pronounced "the limit of $g(x)$ as x approaches a is L". If $\lim_{x \to a} g(x) = L$, then as x gets closer and closer to a, $g(x)$ gets closer and closer to L. (Intuitively, you might think this must always mean that $g(a) = L$, but that's not always the case.)

The slope of the tangent line at $x = a$ is the limit of the slope of the line between $(a, f(a))$ and $(b, f(b))$ as b approaches a:

$$\text{slope of tangent line at } a = \lim_{b \to a} \frac{f(b) - f(a)}{b - a}$$

This definition alone can help us compute some common derivatives. For instance, if $f(x) = x^2$, then the slope of the tangent line at $x = a$ is:

$$\lim_{b \to a} \frac{f(b) - f(a)}{b - a} = \lim_{b \to a} \frac{b^2 - a^2}{b - a} = \lim_{b \to a} \frac{(b - a)(b + a)}{b - a} = \lim_{b \to a} (b + a) = 2a$$

Here, I used the difference of squares formula, $b^2 - a^2 = (b - a)(b + a)$. To find the slope of the tangent line using the limit definition, you'll need to use some sort of algebraic manipulation to simplify the expression, because the limit of the denominator is 0, and we can't divide by 0.

Instead of thinking of the slope of the tangent line as the limit of the slope of the line between the points $(a, f(a))$ and $(b, f(b))$ as $b \to a$, a more general (and equivalent!) definition of the derivative is the limit of the slope of the line between the points $(x, f(x))$ and $(x + h, f(x + h))$ as $h \to 0$.

Derivatives.

Definition: Derivative

Suppose $f : \mathbb{R} \to \mathbb{R}$ is a function. Then the **derivative** of f at x is defined as:

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

This is pronounced "the derivative of f with respect to x."

I will not use this formal definition much in this class, but it's good to understand where it comes from and why it works.

There are two equivalent notations for the derivative: $\frac{\mathrm{d}f}{\mathrm{d}x}(x)$ and $f'(x)$. I used the notation $f'(x)$ in the previous section since it's easier to write and more commonly used in calculus courses. However, I'll use the notation $\frac{\mathrm{d}f}{\mathrm{d}x}(x)$ from now on, as it'll make the transition to multivariable calculus more natural when we get there.

Often, for brevity, I will drop the (x) and just write $\frac{\mathrm{d}f}{\mathrm{d}x}$. As an example, suppose $g(x) = \sin^2(x) + 3\log(x)$, where $\log(\cdot)$ is the natural logarithm (with base e). Then, the derivative of g is:

$$\frac{\mathrm{d}g}{\mathrm{d}x} = 2\sin(x)\cos(x) + \frac{3}{x}$$

$\frac{\mathrm{d}g}{\mathrm{d}x}$ (equivalently, $\frac{\mathrm{d}g}{\mathrm{d}x}(x)$) is a function, not a number. To get a number as an output, we need to plug in a value for x. For example, $\frac{\mathrm{d}g}{\mathrm{d}x}(\pi)$ is the number $(\frac{3}{\pi})$ corresponding to the slope of the tangent line to g at $x = \pi$.

To actually find $\frac{\mathrm{d}g}{\mathrm{d}x}$, we used several derivative rules, which we'll now review.

Rules and Examples

Five Key Derivative Rules

1. **Constant Rule**: If $f(x) = c$, where c is some constant, then:

$$\frac{\mathrm{d}f}{\mathrm{d}x} = 0$$

2. **Addition Rule**: If $h(x) = f(x) + g(x)$, then:

$$\frac{\mathrm{d}h}{\mathrm{d}x} = \frac{\mathrm{d}f}{\mathrm{d}x} + \frac{\mathrm{d}g}{\mathrm{d}x}$$

   Equivalently, $h'(x) = f'(x) + g'(x)$.

3. **Power Rule**:

$$\frac{\mathrm{d}}{\mathrm{d}x} x^n = n x^{n-1}$$

4. **Product Rule**: If $h(x) = f(x)g(x)$, then:

$$\frac{\mathrm{d}h}{\mathrm{d}x} = f \cdot \frac{\mathrm{d}g}{\mathrm{d}x} + \frac{\mathrm{d}f}{\mathrm{d}x} \cdot g$$

Equivalently, $h'(x) = f(x)g'(x) + f'(x)g(x)$.

5. **Chain Rule**: If $h(x) = f(g(x))$, then:

$$\frac{\mathrm{d}h}{\mathrm{d}x} = \frac{\mathrm{d}f}{\mathrm{d}g} \cdot \frac{\mathrm{d}g}{\mathrm{d}x}$$

Equivalently, $h'(x) = f'(g(x))g'(x)$.

These rules can all be proved using the formal definition of the derivative, but those proofs won't be emphasized in this class.

Attempt each of the following examples before peeking at their solutions.

Example: Polynomials. Differentiate $f(x) = 4x^5 + 3x^2 + 2x + 1$.

Solution

None of the five key rules directly specify how to take the derivative of an expression like $4x^5$, and even though you *may* look at this expression and see that its derivative is $20x^4$, it's well worth our time to review the rules carefully.
Let's split $4x^5$ into two separate functions, 4 and x^5, and then use the product rule.

$$\frac{\mathrm{d}}{\mathrm{d}x}4x^5 = 4\left(\frac{\mathrm{d}}{\mathrm{d}x}x^5\right) + \left(\frac{\mathrm{d}}{\mathrm{d}x}4\right)x^5 = 4 \cdot 5x^4 + 0 = 20x^4$$

While applying the product rule, I also used the power rule to differentiate x^5, and the constant rule to differentiate 4.
With this in mind, we can return to the goal of differentiating $f(x)$.

$$\begin{aligned}\frac{\mathrm{d}f}{\mathrm{d}x} &= \frac{\mathrm{d}}{\mathrm{d}x}(4x^5 + 3x^2 + 2x + 1) \\ &= \frac{\mathrm{d}}{\mathrm{d}x}4x^5 + \frac{\mathrm{d}}{\mathrm{d}x}3x^2 + \frac{\mathrm{d}}{\mathrm{d}x}2x + \frac{\mathrm{d}}{\mathrm{d}x}1 \\ &= 20x^4 + 6x + 2\end{aligned}$$

Example: Reciprocals. Differentiate $f(x) = \frac{1}{x^2-1}$.

Solution

Note that $f(x)$ can also be written as $f(x) = (x^2-1)^{-1}$, which looks like something we can use the power rule on. To fully apply the power rule, we'll need to use the chain rule, too.

$$\begin{aligned}\frac{\mathrm{d}}{\mathrm{d}x}(x^2-1)^{-1} &= \underbrace{-1\cdot(x^2-1)^{-2}}_{\text{power rule}} \cdot \underbrace{\frac{\mathrm{d}}{\mathrm{d}x}(x^2-1)}_{\text{required by the chain rule}} \\ &= -1\cdot(x^2-1)^{-2}\cdot 2x \\ &= -\frac{2x}{(x^2-1)^2}\end{aligned}$$

Example: Quotients. Differentiate $f(x) = \frac{x^2+1}{x^2-1}$.

Solution

It looks like $f(x)$ is a quotient of two functions, x^2+1 and x^2-1, but I intentionally did not introduce a quotient rule – it's unnecessary, and can be replicated using the product rule and chain rule, since $f(x)$ can be written as a product:

$$f(x) = \frac{x^2+1}{x^2-1} = (x^2+1)\cdot\frac{1}{x^2-1}$$

You'll notice that we've already found the derivative of $\frac{1}{x^2-1}$ in Example 2, so we can use that result here while applying the product rule.

$$\begin{aligned}\frac{\mathrm{d}f}{\mathrm{d}x} &= \frac{\mathrm{d}}{\mathrm{d}x}\left((x^2+1)\cdot\frac{1}{x^2-1}\right) \\ &= \underbrace{\left(\frac{\mathrm{d}}{\mathrm{d}x}(x^2+1)\right)\cdot\frac{1}{x^2-1} + (x^2+1)\cdot\left(\frac{\mathrm{d}}{\mathrm{d}x}\frac{1}{x^2-1}\right)}_{\text{product rule}} \\ &= 2x\cdot\frac{1}{x^2-1} + (x^2+1)\cdot\underbrace{\left(-\frac{2x}{(x^2-1)^2}\right)}_{\text{from Example 2}} \\ &= \frac{2x}{x^2-1} - \frac{2x(x^2+1)}{(x^2-1)^2} \\ &= \underbrace{\frac{2x(x^2-1)-2x(x^2+1)}{(x^2-1)^2}}_{\text{common denominator}} \\ &= \frac{2x(x^2-1-x^2-1)}{(x^2-1)^2} \\ &= \frac{-4x}{(x^2-1)^2}\end{aligned}$$

Example: Trigonometric and Logarithmic Functions. Differentiate $f(x) = \log(\cos(e^x))$.

To do so, you'll need to remember a few other common derivatives that the five key rules don't cover.

Trigonometric Derivatives:

$$\frac{\mathrm{d}}{\mathrm{d}x}\sin(x) = \cos(x)$$

$$\frac{\mathrm{d}}{\mathrm{d}x}\cos(x) = -\sin(x)$$

Logarithmic Derivatives: If $\log(\cdot)$ is the natural logarithm (with base e), then:

$$\frac{\mathrm{d}}{\mathrm{d}x}\log(x) = \frac{1}{x}$$

$$\frac{\mathrm{d}}{\mathrm{d}x}\log_b(x) = \frac{1}{x\log(b)}$$

Exponential Derivatives:

$$\frac{\mathrm{d}}{\mathrm{d}x}e^x = e^x$$

$$\frac{\mathrm{d}}{\mathrm{d}x}a^x = a^x\log(a)$$

Solution

Differentiating $f(x) = \log(\cos(e^x))$ requires a few careful applications of the chain rule.

$$\begin{aligned}
\frac{\mathrm{d}f}{\mathrm{d}x} &= \frac{\mathrm{d}}{\mathrm{d}x}\log(\cos(e^x)) \\
&= \frac{1}{\cos(e^x)} \cdot \frac{\mathrm{d}}{\mathrm{d}x}\cos(e^x) \\
&= \frac{1}{\cos(e^x)} \cdot (-\sin(e^x)) \cdot \frac{\mathrm{d}}{\mathrm{d}x}e^x \\
&= \frac{1}{\cos(e^x)} \cdot (-\sin(e^x)) \cdot e^x \\
&= -\frac{e^x\sin(e^x)}{\cos(e^x)} \\
&= -e^x\tan(e^x)
\end{aligned}$$

The chain rule is **extremely** pervasive in machine learning, which is why I've included examples like the one above. This site contains dozens more examples of the chain rule in practice.

Activity 2

Activity 2

Note that the activities in this section are quite challenging, so make sure you've attempted and fully understood the examples above first.

Activity 2.1

An important function in machine learning is the **sigmoid function**, which is defined as:

$$\sigma(x) = \frac{1}{1 + e^{-x}}$$

$\sigma(x)$ has a nice S-shape, and is used for predicting probabilities.

Find the derivative of $\sigma(x)$, and show that it satisfies the following property:

$$\frac{\mathrm{d}}{\mathrm{d}x}\sigma(x) = \sigma(x)(1 - \sigma(x))$$

Activity 2.2

Find the derivative of each of the following functions:

- $f(x) = \sqrt{\sin(4\pi x)}$
- $g(x) = (2x + 1)^{3x}$ (Hint: Start by taking the natural logarithm of both sides, then take the derivative of both sides.)

Activity 2.3

Suppose x and y satisfy the following relationship:

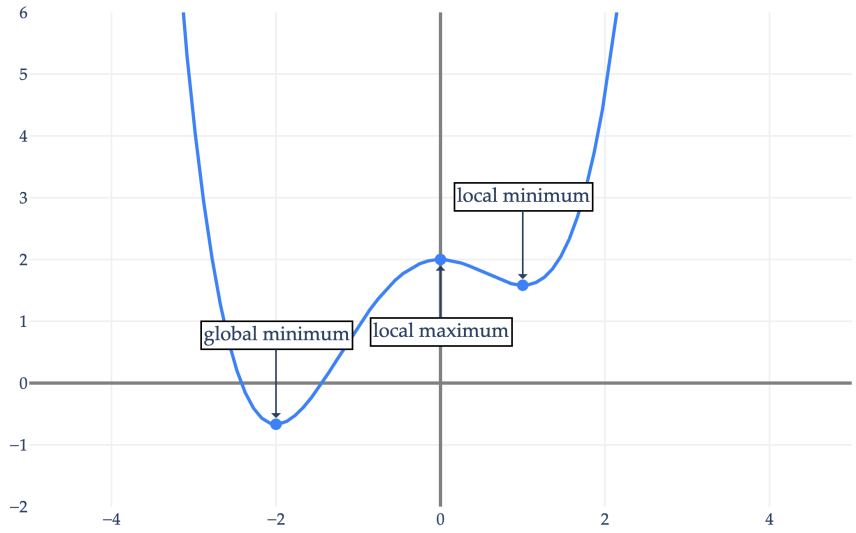
$$x^2 = y^3 - 11$$

1. Find an expression for $\frac{\mathrm{d}y}{\mathrm{d}x}$ that involves **both** x and y. To do this, don't solve for y in terms of x – instead, take the derivative of both sides of the equation with respect to x, use the power and chain rules, and re-arrange to isolate $\frac{\mathrm{d}y}{\mathrm{d}x}$.
2. Find the slope of the tangent line to the curve at the point $(x, y) = (4, 3)$.

Optimization

Maxima and Minima. As stated at the start of this section, calculus is a tool for optimization – that is, finding the inputs that maximize or minimize a function. Let's be more precise about what we mean by "maximize" and "minimize".

Consider $f(x) = \frac{1}{4}x^4 + \frac{1}{3}x^3 - x^2 + 2$, shown below.



Where is $f(x)$ maximized and minimized?

- At $x = -2$, $f(x)$ is less than it is at all other inputs. This is means $(-2, f(-2))$ is a **global minimum**.
- At $x = 0$, $f(x)$ *looks like* it is greater than all other inputs, but only if you restrict your attention to points near $x = 0$. This means $(0, f(0))$ is a **local maximum**. $(0, f(0))$ is not a **global maximum** because there are plenty of points where $f(x) > f(0)$ – they are just not immediately adjacent to $x = 0$.
- Similarly, $(1, f(1))$ is a **local minimum**.

$f(x)$ does not have a global maximum, since $f(x)$ approaches infinity as x increases beyond $x = 1$ or decreases beyond $x = -2$. If we were to restrict the domain (or set of possible inputs) of $f(x)$ to, say, $[-3, 3]$, then there would be a global maximum at $x = 3$.

Note that we usually care more about the inputs that maximize or minimize a function, rather than the actual values of the function at those inputs. In the above example, the fact that $x = -2$ is a global minimum is important; the fact that $f(-2) = -\frac{2}{3}$ is not *as* important.

To recap:

Definition: Maxima and Minima

Suppose $f : \mathbb{R} \to \mathbb{R}$ is a function.
On maxima:

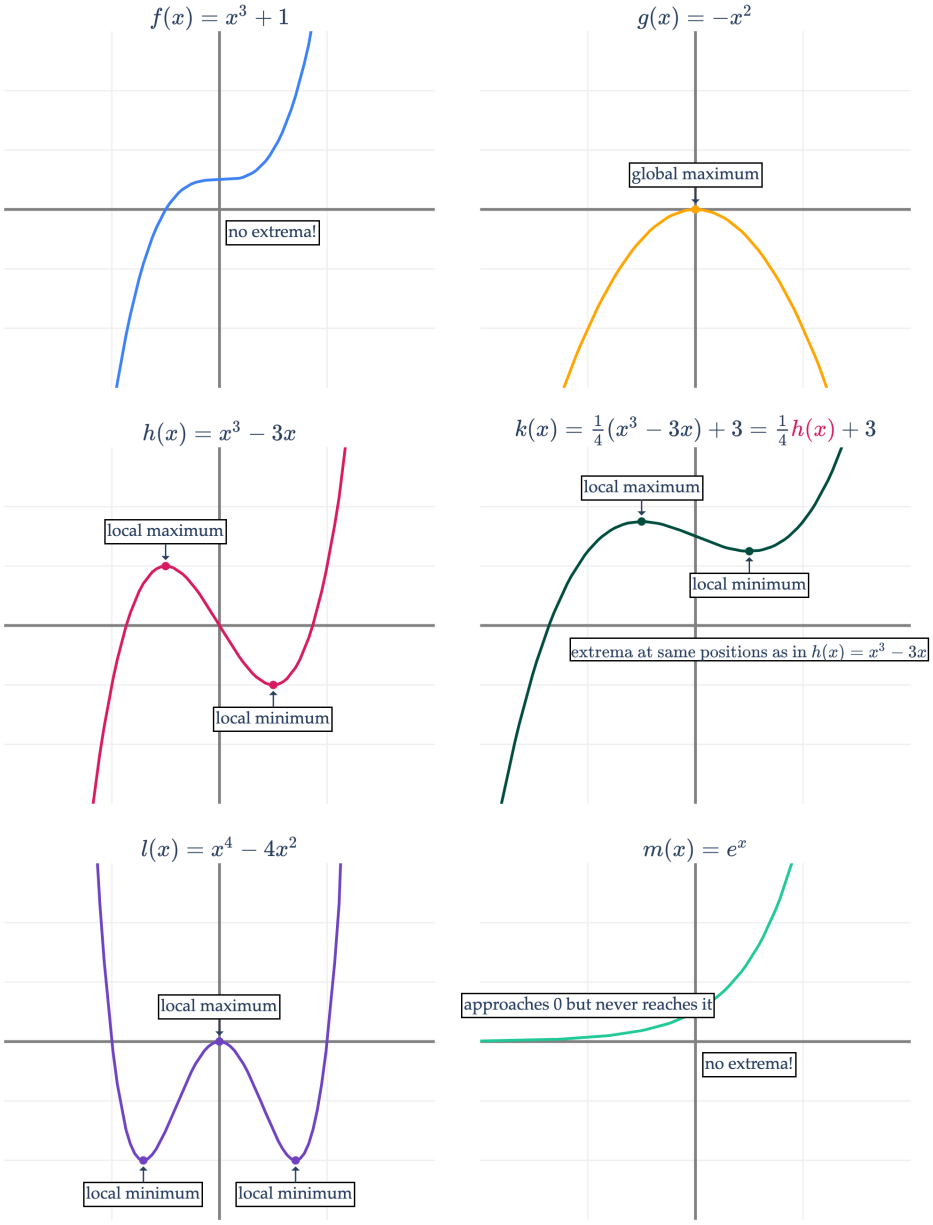
- A function $f : \mathbb{R} \to \mathbb{R}$ has a **local maximum** at $x = x^*$ if $f(x^*) \geq f(x)$ for all x in some interval around x^*.
- A function $f : \mathbb{R} \to \mathbb{R}$ has a **global maximum** at $x = x^*$ if $f(x^*) \geq f(x)$ for all x in the domain of f.

Intuitively, a local maximum is a point where the function is higher than all nearby points, and a global maximum is a point where the function is higher than all points in the domain.
On minima:

- A function $f : \mathbb{R} \to \mathbb{R}$ has a **local minimum** at $x = x^*$ if $f(x^*) \leq f(x)$ for all x in some interval around x^*.
- A function $f : \mathbb{R} \to \mathbb{R}$ has a **global minimum** at $x = x^*$ if $f(x^*) \leq f(x)$ for all x in the domain of f.

Note that maxima is the plural of maximum, minima is the plural of minimum, and extrema refers to both maxima and minima.

Below, you'll find several other examples of functions with varying amounts and types of extrema. Play close attention to the relationship between the two functions in the second row, $h(x)$ and $k(x)$. $k(x)$ results from stretching $h(x)$ vertically and shifting it up vertically, and has the same extrema.



Activity 3

Activity 3

Let $q(x) = 8x^4 - 4x$. $q(x)$ has a global minimum at $\left(\frac{1}{2}, -\frac{3}{2}\right)$.
For each of the following functions, find all extrema, and specify whether each extremum is a local maximum, global maximum, local minimum, or global minimum. Make sure to specify both the x-values and the y-values of each extremum.

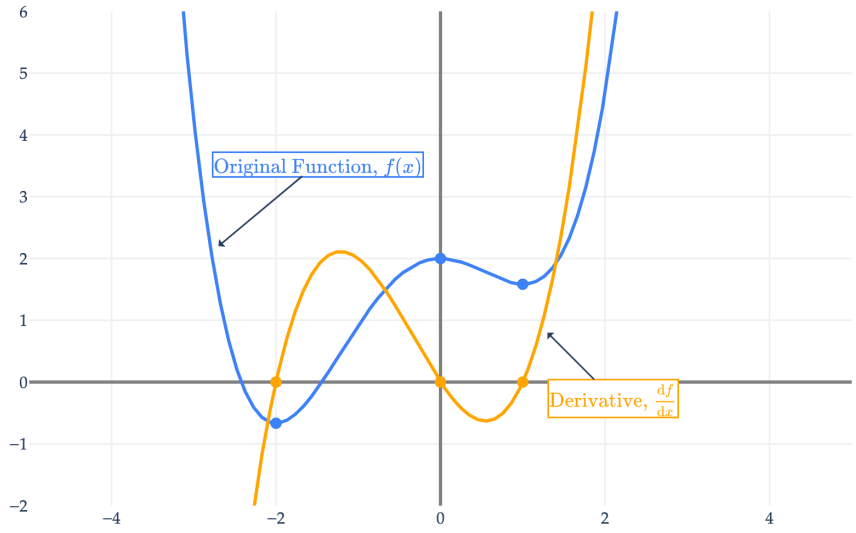
1. $f(x) = 2q(x) + 10$
2. $g(x) = -10q(x)$
3. $h(x) = |q(x)|$
4. Finding the extrema of $l(x) = q(x)^2$ is a bit more complicated than in the examples above. Why?

Critical Points. How do we actually find the critical points of a function, especially when we can't graph the function? The derivative plays a crucial role.

Definition: Critical Point

A **critical point** of a function $f : \mathbb{R} \to \mathbb{R}$ is a point $(a, f(a))$ where $\frac{\mathrm{d}f}{\mathrm{d}x}(a) = 0$ or $\frac{\mathrm{d}f}{\mathrm{d}x}(a)$ is undefined.

Let's revisit the function $f(x) = \frac{1}{4}x^4 + \frac{1}{3}x^3 - x^2 + 2$, shown in blue along with its derivative, $\frac{\mathrm{d}f}{\mathrm{d}x} = x^3 + x^2 - 2x$, shown in orange.



You should notice that the derivative is 0 at all three extrema we identified earlier – the global minimum at $x = -2$, the local maximum at $x = 0$, and the local minimum at $x = 1$. Intuitively, the derivative is 0 at a maximum or minimum because the tangent lines at these points are horizontal (with slope 0), as the function is neither increasing nor decreasing at these points.

In the region between $x = -2$ and $x = 0$, the derivative is positive, meaning the function is increasing.

Solving for the inputs that make the derivative 0 – i.e., finding the critical points – is a necessary, but not sufficient, step. If all we know is that the derivative is 0 at a point, we don't know whether the point is a maximum or minimum. It may not be either, such as in the case of $f(x) = x^3$, which has a critical point at $x = 0$ that is neither a maximum nor a minimum.

Second Derivatives. To be able to determine whether a critical point of $f(x)$ is a maximum or minimum, we need to look at the **second derivative** of $f(x)$. If the (first) derivative of $f(x)$ is a function that describes the rate at which $f(x)$ is changing, the second derivative – denoted $\frac{\mathrm{d}^2 f}{\mathrm{d}x^2}$ – is a function that describes the rate at which

the derivative is changing.

Physics provides us with an analogy that helps us understand the role of the second derivative. Suppose you're driving down a straight road, and $s(t)$ is your position on the road at time t, relative to your starting point (so a negative value of $s(t)$ means you've moved backwards).

Then, $v(t) = \frac{\mathrm{d}s}{\mathrm{d}t}$ is your velocity (the rate at which your position is changing) and $a(t) = \frac{\mathrm{d}^2 s}{\mathrm{d}t^2}$ is your acceleration (the rate at which your velocity is changing).

- If $\frac{\mathrm{d}s}{\mathrm{d}t} > 0$ and $\frac{\mathrm{d}^2 s}{\mathrm{d}t^2} = 0$, you are moving forward at a constant speed (say, on cruise control).
- If $\frac{\mathrm{d}s}{\mathrm{d}t} > 0$ and $\frac{\mathrm{d}^2 s}{\mathrm{d}t^2} > 0$, you are moving forward and your speed is increasing (you are accelerating).
- If $\frac{\mathrm{d}s}{\mathrm{d}t} > 0$ and $\frac{\mathrm{d}^2 s}{\mathrm{d}t^2} < 0$, you are moving forward, but your speed is decreasing, and **eventually, your car will come to a halt**.
- Cases where $\frac{\mathrm{d}s}{\mathrm{d}t} < 0$ correspond to driving backwards!

Activity 4

Activity 4

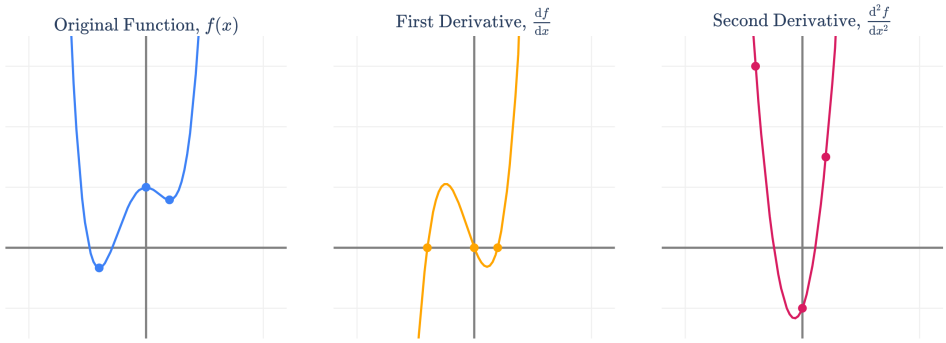
Give real-world examples (similar to those provided above) for each of the following scenarios in the context of the physics analogy:

- $\frac{\mathrm{d}s}{\mathrm{d}t} < 0$ and $\frac{\mathrm{d}^2 s}{\mathrm{d}t^2} > 0$
- $\frac{\mathrm{d}s}{\mathrm{d}t} < 0$ and $\frac{\mathrm{d}^2 s}{\mathrm{d}t^2} < 0$
- $\frac{\mathrm{d}s}{\mathrm{d}t} = 0$ and $\frac{\mathrm{d}^2 s}{\mathrm{d}t^2} < 0$

Let's put this in the context of our running example, $f(x) = \frac{1}{4}x^4 + \frac{1}{3}x^3 - x^2 + 2$. The second derivative of $f(x)$ is:

$$\begin{aligned}
\frac{\mathrm{d}^2 f}{\mathrm{d}x^2} &= \frac{\mathrm{d}}{\mathrm{d}x}\left(\frac{\mathrm{d}}{\mathrm{d}x}\left(\frac{1}{4}x^4 + \frac{1}{3}x^3 - x^2 + 2\right)\right) \\
&= \frac{\mathrm{d}}{\mathrm{d}x}\left(\underbrace{x^3 + x^2 - 2x}_{\text{first derivative of } f(x)}\right) \\
&= 3x^2 + 2x - 2
\end{aligned}$$

The second derivative, $\frac{\mathrm{d}^2 f}{\mathrm{d}x^2}$, is a function, not a number. What does the second derivative look like, relative to the original function and first derivative?



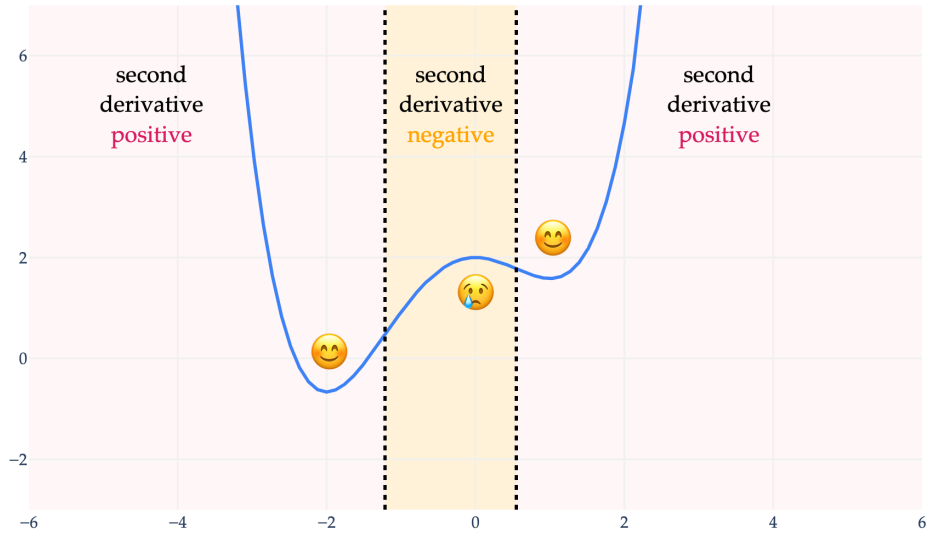
$f(x)$ is a polynomial of degree 4, $\frac{df}{dx}$ is a polynomial of degree 3, and $\frac{d^2f}{dx^2}$ is a polynomial of degree 2 – the degree drops by one each time, as a consequence of the power rule.

Recall that $f(x)$ has critical points at $x = -2$, $x = 0$, and $x = 1$, which we've highlighted in all three plots above. Our goal is to determine an algebraic approach for determining whether these points are maxima, minima, or neither; we shouldn't rely on the graph of $f(x)$ alone, since we won't always be able to see its graph.

At all three of these points, the first derivative is 0, meaning that the function is neither increasing nor decreasing at these points. But the second derivative $\frac{d^2f}{dx^2} = 3x^2 + 2x - 2$ gives us additional information:

- At $x = -2$, $\frac{d^2f}{dx^2}(-2) = 6$, which is **positive**. So, at $x = -2$, $f(x)$ is neither increasing nor decreasing, but is also "*speeding up*", since the second derivative is positive. So, as we move to the right of $x = -2$, the slope of the tangent line will increase, causing the function to increase. If the function increases to the right of $x = -2$, then $x = -2$ must correspond to a **local minimum** of $f(x)$.
- At $x = 0$, $\frac{d^2f}{dx^2}(0) = -2$, which is **negative**. So, at $x = 0$, $f(x)$ is neither increasing nor decreasing, but is also "*slowing down*". So, as we move to the right of $x = 0$, the slope of the tangent line will decrease, causing the function to decrease. If the function decreases to the right of $x = 0$, then $x = 0$ must correspond to a **local maximum** of $f(x)$.
- At $x = 1$, $\frac{d^2f}{dx^2}(1) = 3$ is **positive**, which, using the logic from the $x = -2$ case, means that $x = 1$ also corresponds to a **local minimum** of $f(x)$.

Convexity. The sign of the second derivative is useful for more than just determining whether a critical point is a local maximum or minimum. Below, we've plotted $f(x)$, along with annotations for the regions where the second derivative is positive and negative.



When the second derivative is positive, the function is **concave opening up**, also known as **convex**. You should think of convex functions as "bowl-shaped" or "smiling". When the second derivative is negative, the function is **concave opening down**, or simply **concave**; the equivalent analogy is that concave down regions are "upside-down bowls" or "sad faces".

From the perspective of finding local mimina, if a function is concave up at a critical point, then we must be at the bottom of a bowl – a local minimum – and if a function is concave down at a critical point, we must be at the top of a hill, corresponding to a local maximum.

If a function is concave up across its entire domain – **unlike** in the example above, but like in $f(x) = x^2$ – then any local minimum must be a **global minimum**. Convexity is a hugely important concept in optimization and machine learning, and we'll see it again in more detail throughout the course.

The points at which the second derivative is 0 are called **inflection points**. $f(x)$ has two inflection points, marked by vertical dotted lines above, roughly at $x = -1.22$ and $x = 0.55$. These are the roots of the quadratic equation $\frac{\mathrm{d}^2 f}{\mathrm{d}x^2} = 3x^2 + 2x - 2 = 0$.

We've implictly used a **second derivative test** for determining whether a critical point is a local maximum or minimum:

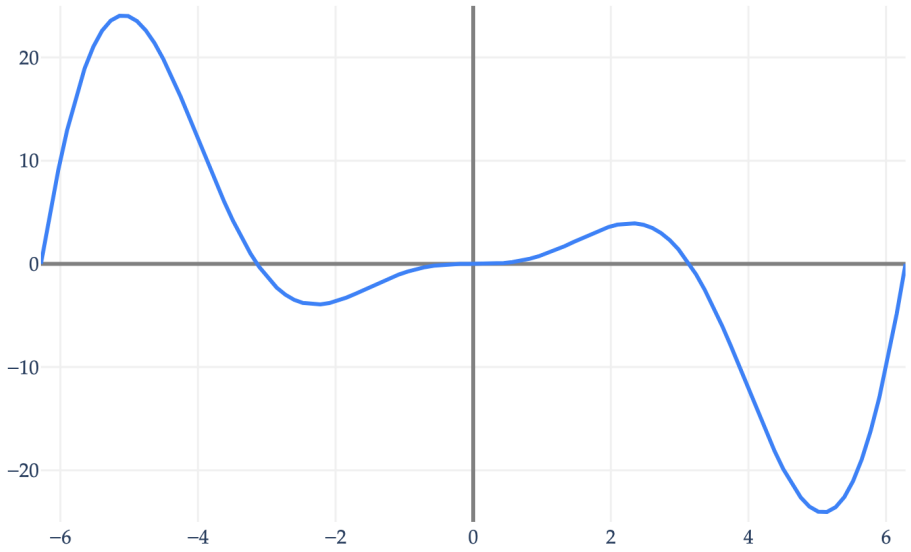
Definition: Second Derivative Test

Suppose a is a critical point of $f(x)$, i.e. $\frac{\mathrm{d}}{\mathrm{d}x} f(a) = 0$ or $\frac{\mathrm{d}}{\mathrm{d}x} f(a)$ is undefined. Then, there are 3 possibilities:

- If $\frac{\mathrm{d}^2 f}{\mathrm{d}x^2}(a) > 0$, then $f(x)$ is concave opening up (i.e. convex) at $x = a$, and $x = a$ is a local **minimum**.
- If $\frac{\mathrm{d}^2 f}{\mathrm{d}x^2}(a) < 0$, then $f(x)$ is concave opening down at $x = a$, and $x = a$ is a local **maximum**.
- If $\frac{\mathrm{d}^2 f}{\mathrm{d}x^2}(a) = 0$, then $x = a$ is an inflection point, and the second derivative test is inconclusive.

Again, the second derivative test only tries to tell us whether critical points are local maxima or minima; it **does not** tell us whether they are global maxima or minima.

Let's look at another example, particularly one where the second derivative test is inconclusive. Consider $f(x) = x^2 \sin(x)$, shown below.



$f(x) = x^2 \sin(x)$, like $\sin(x)$, is oscillatory, and has no global extrema; see **here** for a larger graph of it. Above, we've plotted $f(x)$ within the domain $[-2\pi, 2\pi]$, and we see several local maxima and minima.

The first and second derivatives of $f(x) = x^2 \sin(x)$ are given by:

$$\begin{aligned}\frac{\mathrm{d}f}{\mathrm{d}x} &= x^2\left(\frac{\mathrm{d}}{\mathrm{d}x}\sin(x)\right) + \left(\frac{\mathrm{d}}{\mathrm{d}x}x^2\right)\sin(x) \\ &= x^2\cos(x) + 2x\sin(x)\end{aligned}$$

$$\begin{aligned}\frac{\mathrm{d}^2 f}{\mathrm{d}x^2} &= \frac{\mathrm{d}}{\mathrm{d}x}\left(x^2\cos(x) + 2x\sin(x)\right) \\ &= x^2\left(\frac{\mathrm{d}}{\mathrm{d}x}\cos(x)\right) + \left(\frac{\mathrm{d}}{\mathrm{d}x}x^2\right)\cos(x) + 2x\left(\frac{\mathrm{d}}{\mathrm{d}x}\sin(x)\right) + \left(\frac{\mathrm{d}}{\mathrm{d}x}2x\right)\sin(x) \\ &= -x^2\sin(x) + 2x\cos(x) + 2x\cos(x) + 2\sin(x) \\ &= 2x\cos(x) - (x^2 - 2)\sin(x)\end{aligned}$$

Solving for the critical points of $f(x)$ by setting $\frac{\mathrm{d}f}{\mathrm{d}x} = x^2\cos(x) + 2x\sin(x) = 0$ is no easy task, as there are infinitely many solutions, most of which cannot be solved for by hand. We'll learn how to write code to approximate solutions to $\frac{\mathrm{d}f}{\mathrm{d}x} = 0$ in Chapter 8 of the course, when we study gradient descent. There are also infinitely many inflection points, since $\frac{\mathrm{d}^2 f}{\mathrm{d}x^2} = 0$ has infinitely many solutions, meaning that there are many regions where $f(x)$ is concave up and many others where it is concave down.

However, one critical point is easy to spot: $x = 0$. At $x = 0$, the derivative is 0:

$$\frac{\mathrm{d}f}{\mathrm{d}x}(0) = 0^2\cos(0) + 2 \cdot 0 \cdot \sin(0) = 0$$

$x = 0$ is also an inflection point, since the second derivative is also 0:

$$\frac{\mathrm{d}^2 f}{\mathrm{d}x^2}(0) = 2 \cdot 0 \cdot \cos(0) - (0^2 - 2)\sin(0) = 0$$

To be clear, not every inflection point is a critical point, and not every critical point is an inflection point; $x = 0$ just happens to be both.

If we look at the graph of $f(x)$ near $x = 0$, we'll see that $x = 0$ corresponds to neither a local maximum nor a local minimum, but rather, a region where $f(x)$ is very flat. If we weren't able to graph $f(x)$, we could try and determine its behavior around $(0, f(0))$ by looking at points immediately to the left and right of $x = 0$ – say, $(0.001, f(0.001))$ and $(-0.001, f(-0.001))$. If $f(0.001) > f(0)$ and $f(-0.001) > f(0)$, then $x = 0$ would be a local minimum (but that's not the case here).

Activity 5

Activity 5

Activity 5.1
Let $f(x) = x\log(x^2)$, where $\log(\cdot)$ is the natural logarithm.

1. Find the critical points of $f(x)$, and determine whether they are local maxima, minima, or neither.
2. Find the inflection points of $f(x)$, and use them to sketch a possible graph of $f(x)$.

Activity 5.2
Let $g(3) = 10$, $\frac{\mathrm{d}g}{\mathrm{d}x}(3) = -2$, and $\frac{\mathrm{d}^2 g}{\mathrm{d}x^2}(3) = 1$.

1. Describe the behavior of $g(x)$ near $x = 3$.
2. The Taylor series of a function allows us to approximate the value of a function near a point $x = a$, given the value of the function and its derivatives at $x = a$. The Taylor series of an arbitrary function $f(x)$ around $x = a$ is given by:

   $$f(x) \approx f(a) + \left(\frac{\mathrm{d}f}{\mathrm{d}x}(a)\right)(x-a) + \frac{1}{2}\left(\frac{\mathrm{d}^2 f}{\mathrm{d}x^2}(a)\right)(x-a)^2 + \frac{1}{6}\left(\frac{\mathrm{d}^3 f}{\mathrm{d}x^3}(a)\right)(x-a)^3 + \frac{1}{24}\left(\frac{\mathrm{d}^4 f}{\mathrm{d}x^4}(a)\right)(x-a)^4 + \cdots$$

   Note that this is an infinite series; the more terms we use, the more accurate the approximation.

   Use the Taylor series to approximate the value of $g(3.1)$, using only the information provided. You'll only be able to use the first 3 terms of the Taylor series.

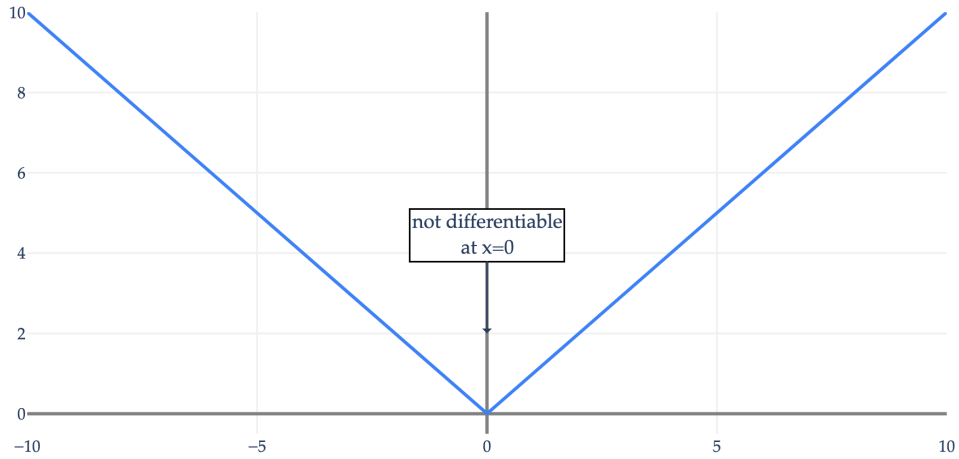
Activity 5.3
Given that $\frac{\mathrm{d}^2 h}{\mathrm{d}x^2} = 2x(x-3)(x+1)$, sketch a possible graph of $h(x)$.

Continuity and Differentiability

Finally, I'll remark that we've presented derivatives, extrema, and optimization all in the most ideal setting: where the functions we're working with are continuous and differentiable. A function is **continuous** if its graph can be drawn without lifting a pen; any point where the graph has a "jump" or "break" is a discontinuity. (Of course, there is a more formal definition of continuity, but this is a good enough illustration for now.) A function is **differentiable** if its derivative exists everywhere; otherwise, there exist some points at which the derivative does not exist.

Most relevant functions in machine learning are continuous, but non-differentiable functions do appear, so it's worth understanding what they are and how to deal with them. Let's look at a few examples.

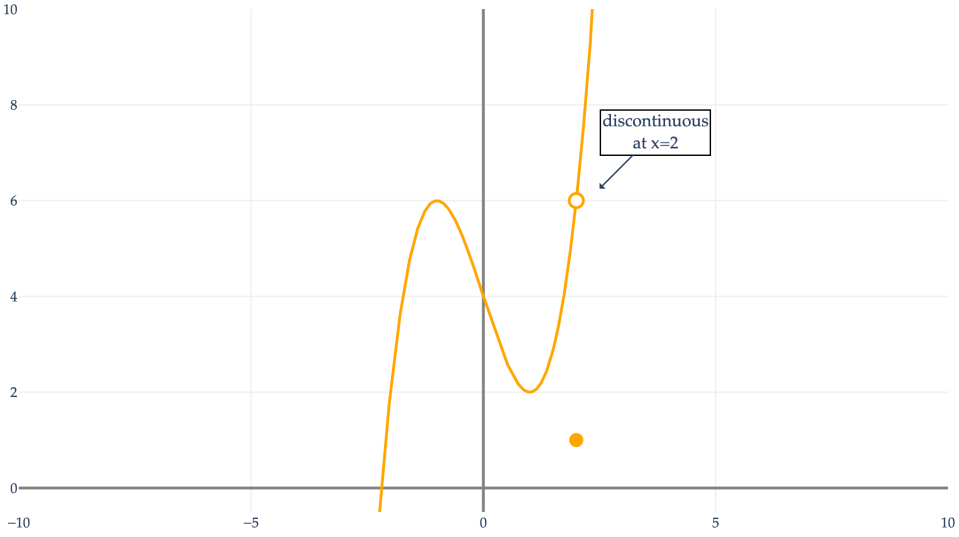
Example 1: $f(x) = |x|$



$f(x) = |x|$ is continuous everywhere, as intuitively, we can draw its graph without lifting our pen. It is differentiable everywhere **except** at $x = 0$; the reason it is not differentiable at $x = 0$ is that the slopes approaching it from the left (-1) and right (1) are different, and in order for a derivative at $x = a$ to exist, the limit of the slopes approaching a from the left and right must be the same.

$$\frac{\mathrm{d}f}{\mathrm{d}x} = \begin{cases} -1 & x < 0 \\ 1 & x > 0 \\ \text{undefined} & x = 0 \end{cases}$$

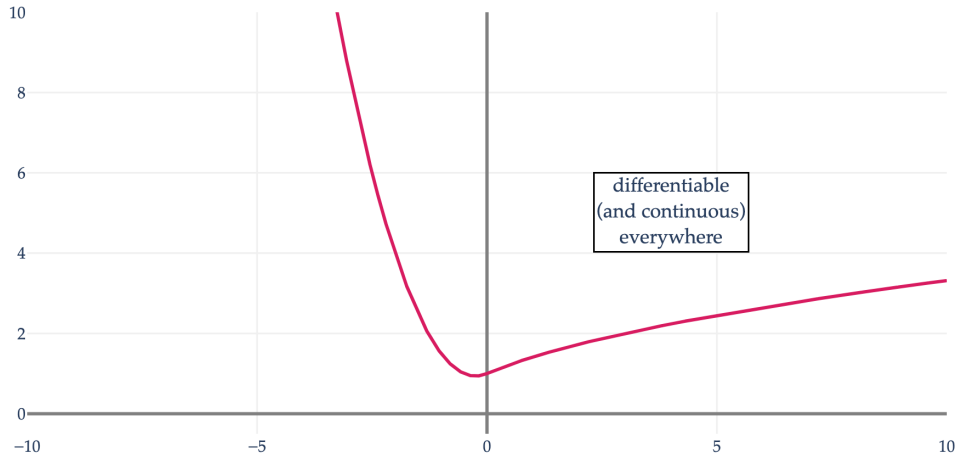
Example 2: $g(x) = \begin{cases} x^3 - 3x + 4 & x \neq 2 \\ 1 & x = 2 \end{cases}$



$g(x) = \begin{cases} x^3 - 3x + 4 & x \neq 2 \\ 1 & x = 2 \end{cases}$ is continuous and differentiable everywhere, except at $x = 2$, where it is neither.

$$\frac{dg}{dx} = \begin{cases} 3x^2 - 3 & x \neq 2 \\ \text{undefined} & x = 2 \end{cases}$$

Example 3: $h(x) = \begin{cases} x^2 + \frac{1}{2}x + 1 & x < 0 \\ \sqrt{x+1} & x \geq 0 \end{cases}$

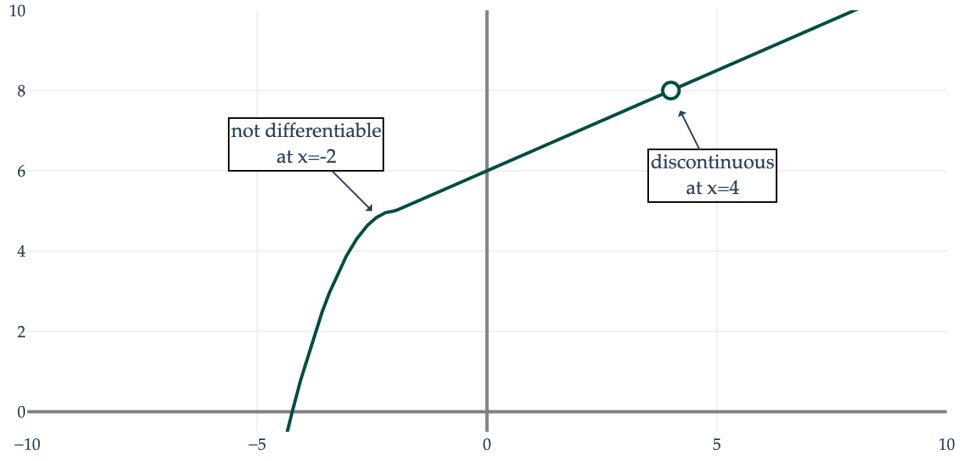


$h(x) = \begin{cases} x^2 + \frac{1}{2}x + 1 & x < 0 \\ \sqrt{x+1} & x \geq 0 \end{cases}$ is continuous and differentiable everywhere, despite being a piecewise function. Its individual pieces are continuous, and the entire function is continuous because the "left" and "right" functions at the connection point of $x = 0$ have the same value.

$$\frac{dh}{dx} = \begin{cases} 2x + \frac{1}{2} & x < 0 \\ \frac{1}{2\sqrt{x+1}} & x \geq 0 \end{cases}$$

Since the two piecewise derivatives agree at $x = 0$, $h(x)$ is differentiable at $x = 0$ (and across its entire domain).

Example 4: $k(x) = \begin{cases} -(x+2)^2 + 5 & x < -2 \\ \frac{1}{2}x + 6 & x \geq -2,\ x \neq 4 \\ \text{undefined} & x = 4 \end{cases}$



$k(x) = \begin{cases} -(x+2)^2 + 5 & x < -2 \\ \frac{1}{2}x + 6 & x \geq -2,\ x \neq 4 \\ \text{undefined} & x = 4 \end{cases}$ is continuous everywhere, except at $x = 4$, where it has a "jump" and is neither continuous nor differentiable. But in addition, $k(x)$ is not differentiable at $x = -2$ because the slopes approaching it from the left and right are different.

$$\frac{\mathrm{d}k}{\mathrm{d}x} = \begin{cases} -2(x+2) & x < -2 \\ \text{undefined} & x = -2 \\ \frac{1}{2} & x > -2,\ x \neq 4 \\ \text{undefined} & x = 4 \end{cases}$$

An important point is that any function that is differentiable everywhere is also continuous everywhere; differentiability is a stronger condition than continuity. Plenty of functions are continuous but not differentiable, like $f(x) = |x|$ in Example 1.