LibreNotebook

Last Updated: September 3, 2026

Enoch Yu

Contents

Preface

§Purpose and Motivation

LibreNotebook initially started as my personal notes. To be exact, it began with me studying for national math olympiad. Frankly, I was lost when studying for the national math olympiad as not many resources were available specifically for the high school division of our country's olympiad. Even if there are resources, it was either for middle school round or was commercial with high expense. Naturally, I began seeking for problems and resources from different countries. There, with open and public resources, I was able to learn and improve. Since then, I always wanted to return the blessings received from open education.

Over the 2026 summer break, I wrote notes on linear algebra and machine learning. With increasing lengths and details, I thought they could be more than just personal notes. With different handouts that I wrote for multiple purposes, I thought it would be possible to return the blessings received by doing the same of returning knowledge. LibreNotebook is the notebook that highlights understandings that were learned to be shared.

§About the Book

Teaching and sharing information, in my opinion, is the best way to learn. Whenever I explain the contents that I learned to others, I start to really distinguish what I know and what I do not know. Therefore, when I write my notes, I write it as if I am explaining and teaching others, sharing my thought process and how I understood topics.

Having said that, LibreNotebook is a collection of my personal notes. It is not meant to replace textbooks, courses, and dedicated resources. On the other hand, if you are also interested and learning more about the topics included in my notes, I believe LibreNotebook can serve as a great resource to see how another person on Earth understood a concept.

§Contributing

First and foremost, thank you so much for your interest and taking your time to contribute. All contributions are welcomed and encouraged with minor details to keep in mind. Any contributions ranging from a typo to writing a new note is welcomed! The source files are hosted in GitHub, and please either create an issue, submit a pull request, or write a mail using the contact form from my website.

However, I want to include a notice on AI usage. I know this topic is philosophical and political, but I want to say I really don't want LibreNotebook to be flooded with AI generated contents. Unlike AI assisted contents, many AI generated contents lack proper credits and quality checks. For transparency of contributors, I think it would be great if AI usage is disclosed in each contribution. With that, thank you again for your interest and efforts!

§About the Author

Enoch is an ardent student who retains keen interest in pure math, machine learning, and quantitative finance. His previous accomplishments includes honorable mention from Korean Mathematical Olympiad (300 in South Korea) and numerous awards from AMC. He currently dedicates his time on making YouTube videos on various math topics and Nanum Learn. More on Enoch can be found in his personal website.

Part I

Math Olympiad

Contents

Note 1

Inequalities

Inequality is perhaps the most amiable, but extremely in depth section to study in Algebra. It could range from simple inequalities such as solving for x in $x+1>2$ to problems involving numerous advanced techniques and known results. As a result, although few inequalities in advanced problems could be proven by bashing, most would require an elegant solution. With that in mind, this chapter will begin with numerous classical inequalities such as the AM-GM inequality to advanced concepts such as majorization.

§1.1 Classic Inequalities

One of the first concepts that most students, including myself, study when getting started with olympiad is I believe inequalities. Many problems are solved by tweaking known inequalities and doing some clever substitutions. In this note, we will discuss fundamental and classic inequalities as an introduction to olympiad inequalities.

1.1.1 AM-GM Inequality

Let's start with AM-GM Inequality. I assume most of you know or at least have heard of this inequality as we learn it in middle and high school. AM-GM inequality stands for Arithmetic Mean-Geometric Mean inequality and literally compares those two means.

Definition 1.1.1: AM-GM Inequality

For $a_k \geq 0$, the following inequality holds with equality if and only if $a_1 = a_2 = \cdots = a_n$.

$$\frac{\sum_{i=1}^{n} a_i}{n} \geq \sqrt[n]{\prod_{i=1}^{n} a_i}$$

You might be more familiar with $a+b \geq 2\sqrt{ab}$, but the inequality above is the generalized version.

Tip.

AM-GM as a fundamental inequality has lots of application. However in olympiads, it is used the most for cancellation and investigating the minimum.

Consider the following example.

Exercise 1.1.2

Prove the following inequality for $a, b, c, d \geq 0$.

$$(a+b)(b+c)(c+d)(d+a) \geq 16abcd$$

Solution.

Because a, b, c, d are non-negative, we could utilize AM-GM inequality. By AM-GM inequality, the following inequalities hold.

$$\begin{aligned} a+b &\geq 2\sqrt{ab} \\ b+c &\geq 2\sqrt{bc} \\ c+d &\geq 2\sqrt{cd} \\ d+a &\geq 2\sqrt{da} \end{aligned}$$

Multiplying the inequalities,

$$(a+b)(b+c)(c+d)(d+a) \geq 2\sqrt{ab} \cdot 2\sqrt{bc} \cdot 2\sqrt{cd} \cdot 2\sqrt{da} = 16abcd$$

hold and $(a+b)(b+c)(c+d)(d+a) \geq 16abcd$.

Exercise 1.1.3

SECOND PROBLEM FOR AM-GM INEQUALITY

When we discuss arithmetic and geometric mean, we normally include weighted means. Thus, it is natural to consider AM-GM inequality with weighted means.

Weighted AM-GM Inequality

Let's first get started with definitions.

Definition 1.1.4: Weighted AM-GM Inequality

For weights $w_1, w_2, \ldots, w_n \geq 0$ such that $\sum_{i=1}^{n} w_i = w$ the following inequality holds.

$$\frac{\sum_{i=1}^{n} w_i a_i}{w} \geq \sqrt[w]{\prod_{i=1}^{n} a_i^{w_i}}$$

The equality holds if and only if $a_1 = a_2 = \cdots = a_n$.

From the definition, we can notice that the original inequality is obtained with $w_1 = w_2 = \cdots = w_n = \frac{1}{n}$. Here is a problem on weighted AM-GM inequality.

Exercise 1.1.5: 2020 IMO Problem 2

Show that for $a, b, c, d \in \mathbb{R}$ such that $a \geq b \geq c \geq d > 0$ and $a + b + c + d = 1$,

$$(a + 2b + 3c + 4d)a^a b^b c^c d^d < 1$$

holds.

(Video Solution)

Solution.

First, consider the following weighted AM-GM inequality for a, b, c, d with weights a, b, c, d respectively.

$$\frac{a \cdot a + b \cdot b + c \cdot c + d \cdot d}{a + b + c + d} \geq \sqrt[a+b+c+d]{a^a b^b c^c d^d}$$
$$a^2 + b^2 + c^2 + d^2 \geq a^a b^b c^c d^d$$

Therefore, it suffices to show that the following inequality holds.

$$(a + 2b + 3c + 4d)(a^2 + b^2 + c^2 + d^2) < 1$$

Consider inequalities below.

$$\begin{aligned}
a^2(a + 2b + 3c + 4d) &< a^2(a + 3b + 3c + 3d) && (\because b \geq d) \\
b^2(a + 2b + 3c + 4d) &< b^2(3a + b + 3c + 3d) && (\because 2a \geq b + d) \\
c^2(a + 2b + 3c + 4d) &< c^2(3a + 3b + c + 3d) && (\because 2a + b \geq 2c + d) \\
d^2(a + 2b + 3c + 4d) &< d^2(3a + 3b + 3c + d) && (\because 2a + b \geq 3d)
\end{aligned}$$

Integrating,

$$\begin{aligned}
(a + 2b + 3c + 4d)(a^2 + b^2 + c^2 + d^2) &< a^2(a + 3b + 3c + 3d) + b^2(3a + b + 3c + 3d) \\
&\quad + c^2(3a + 3b + c + 3d) + d^2(3a + 3b + 3c + d) \\
&< (a + b + c + d)^3 = 1
\end{aligned}$$

and the inequality holds.

With the ideas of weighted AM-GM inequality in mind, we can further generalize our results for different means.

1.1.2 Power Mean and Weighted Power Mean Inequality

Power Mean inequality, also known as QM-AM-GM-HM inequality, compares the four major means with generalization of $AM - GM$ inequality. Without simple notations, we can represent the Power Mean inequality as $M_2 \geq M_1 \geq M_0 \geq M_{-1}$.

Definition 1.1.6: Power Mean Inequality

For positive real numbers a_i, the inequality

$$\sqrt{\frac{\sum_{i=1}^{n} a_i^2}{n}} \geq \frac{\sum_{i=1}^{n} a_i}{n} \geq \sqrt[n]{\prod_{i=1}^{n} a_i} \geq \frac{n}{\sum_{i=1}^{n} \frac{1}{a_i}}$$

holds with equality if and only if $a_1 = \cdots = a_n$.

With the definition in mind, we can naturally apply the same for weighted power means.

Definition 1.1.7: Weighted Power Mean Inequality

For positive real numbers a_i, the inequality

$$\sqrt{\frac{\sum_{i=1}^{n} w_i a_i^2}{w}} \geq \frac{\sum_{i=1}^{n} w_i a_i}{w} \geq \sqrt[w]{\prod_{i=1}^{n} a_i^{w_i}} \geq \frac{w}{\sum_{i=1}^{n} \frac{w_i}{a_i}}$$

holds with equality if and only if $a_1 = \cdots = a_n$.

Continuing from AM-GM inequality and its variations, the second most taught inequality in middle and high schools is probably Cauchy-Schwarz Inequality.

1.1.3 Cauchy-Schwarz Inequality

As always, let's get started with the definition.

Definition 1.1.8: Cauchy-Schwarz Inequality

For all real numbers a_i and b_i,

$$\left(\sum_{i=1}^{n} a_i^2\right)\left(\sum_{i=1}^{n} b_i^2\right) \geq \left(\sum_{i=1}^{n} a_i b_i\right)^2$$

with equality if and only if $\frac{a_i}{b_i} = k$ for all integer $1 \leq i \leq n$, where $k > 0$ and $a_i b_i \neq 0$.

When $n = 2$ we get the form $(a^2 + b^2)(x^2 + y^2) \geq (ax + by)^2$ that we initially learn. Continuing, we can obtain an interesting direct consequence. The lemma below known as Titu's lemma, T2 Lemma, Sedrakyan's inequality, and Engel's form and was named after one of the greatest mathematician Titu Andreescu.

Corollary 1.1.9: Titu's Lemma

For positive real numbers a_i and b_i,

$$\sum_{i=1}^{n} \frac{a_i^2}{b_i} \geq \frac{\left(\sum_{i=1}^{n} a_i\right)^2}{\sum_{i=1}^{n} b_i}$$

Proof.

Substituting $x_i = \frac{a_i}{\sqrt{b_i}}$ and $y_i = \sqrt{b_i}$ to the Cauchy-Schwarz inequality, the following inequalities are obtained.

$$\left(\sum_{i=1}^{n} x_i^2\right)\left(\sum_{i=1}^{n} y_i^2\right) \geq \left(\sum_{i=1}^{n} x_i y_i\right)^2$$

$$\left(\sum_{i=1}^{n} \left(\frac{a_i}{\sqrt{b_i}}\right)^2\right)\left(\sum_{i=1}^{n} \left(\sqrt{b_i}\right)^2\right) \geq \left(\sum_{i=1}^{n} \frac{a_i}{\sqrt{b_i}} \cdot \sqrt{b_i}\right)^2$$

Recasting,

$$\left(\sum_{i=1}^{n} \frac{a_i^2}{b_i}\right)\left(\sum_{i=1}^{n} b_i\right) \geq \left(\sum_{i=1}^{n} a_i\right)^2$$

and

$$\sum_{i=1}^{n} \frac{a_i^2}{b_i} \geq \frac{\left(\sum_{i=1}^{n} a_i\right)^2}{\sum_{i=1}^{n} b_i}$$

is obtained. Thus, the lemma holds.

This lemma comes especially handy if an inequality involves fractions with square numbers. Now that we discussed a direct consequence of Cauchy-Schwarz inequality, we can consider its generalization.

1.1.4 Hölder's Inequality

Cauchy-Schwarz inequality is actually a special case of Hölder's inequality. Both inequalities can be used in multiple ways such as eliminating radicals and fractions. Below is the definition of Hölder's inequality.

Definition 1.1.10: Hölder's Inequality

Let $a_i, b_i, \ldots, z_i > 0$ and $\lambda_a, \lambda_b, \ldots, \lambda_z \geq 0$ such that $\lambda_a + \lambda_b + \cdots + \lambda_z = 1$. Then,

$$\left(\sum_{i=1}^{n} a_i\right)^{\lambda_a}\left(\sum_{i=1}^{n} b_i\right)^{\lambda_b} \cdots \left(\sum_{i=1}^{n} z_i\right)^{\lambda_z} \geq \sum_{i=1}^{n} a_i^{\lambda_a} b_i^{\lambda_b} \cdots z_i^{\lambda_z}$$

holds with equality if $a_1 : a_2 : \cdots : a_n \equiv b_1 : b_2 : \cdots : b_n \equiv \cdots \equiv z_1 : z_2 : \cdots : z_n$.

Indeed, Cauchy-Schwarz inequality does resemble Hölder's Inequality! To derive Cauchy-

Scharz inequality from Hölder's, it suffices to show that a specific case of Hölder's inequality will lead to Cauchy-Schwarz inequality.

Let $\lambda_a = \lambda_b = \frac{1}{2}$. Substituting, the following inequalities hold.

$$\left(\sum_{i=1}^{n} a_i\right)^{\frac{1}{2}} \left(\sum_{i=1}^{n} b_i\right)^{\frac{1}{2}} \left(\sum_{i=1}^{n} c_i\right)^{0} \cdots \left(\sum_{i=1}^{n} z_i\right)^{0} \geq \sum_{i=1}^{n} a_i^{\frac{1}{2}} b_i^{\frac{1}{2}} \cdot c_i^0 \cdots z_i^0$$

$$\left(\sum_{i=1}^{n} a_i\right)^{\frac{1}{2}} \left(\sum_{i=1}^{n} b_i\right)^{\frac{1}{2}} \geq \sum_{i=1}^{n} a_i^{\frac{1}{2}} b_i^{\frac{1}{2}}$$

Let $\alpha_i = \sqrt{a_i}$ and $\beta_i = \sqrt{b_i}$. Substituting,

$$\left(\sum_{i=1}^{n} \alpha_i^2\right)^{\frac{1}{2}} \left(\sum_{i=1}^{n} \beta_i^2\right)^{\frac{1}{2}} \geq \sum_{i=1}^{n} \alpha_i \beta_i$$

and

$$\left(\sum_{i=1}^{n} \alpha_i^2\right) \left(\sum_{i=1}^{n} \beta_i^2\right) \geq \left(\sum_{i=1}^{n} \alpha_i \beta_i\right)^2$$

is obtained. This is Cauchy-Schwarz inequality! Below is a very classic problem on Hölder's Inequality

Exercise 1.1.11

Show that the following inequality holds for all $a, b, c > 0$.

$$(a^3+2)(b^3+2)(c^3+2) \geq (a+b+c)^3$$

(Video Solution)

Solution.

The key to this problem is changing the left-hand side to apply Hölder's Inequality. Consider the following inequality by Hölder.

$$(a^3+1+1)^{\frac{1}{3}}(1+b^3+1)^{\frac{1}{3}}(1+1+c^3)^{\frac{1}{3}} \geq a+b+c$$

Thus, cubing both sides leads to the desired result.

Continuing from Cauchy-Schwarz inequality and its variation, another fundamental inequality of different type is rearrangement inequality.

1.1.5 Rearrangement Inequality

Let's start with definition.

Definition 1.1.12: Rearrangement Inequality

Let a_i and b_i be real numbers such that $a_1 \geq a_2 \geq \cdots \geq a_n$ and $b_1 \geq b_2 \geq \cdots \geq b_n$. Moreover, let $b_{\sigma(i)}$ be the i^{th} term of the permutations of $b_1, \cdots, b_n$. Then, the following inequalities hold.

$$\sum_{i=1}^{n} a_i b_i \geq \sum_{i=1}^{n} a_i b_{\sigma(i)} \geq \sum_{i=1}^{n} a_i b_{n-i+1}$$

The equality holds if at least one sequence is constant.

With the definition, we can obtain a special consequence.

Corollary 1.1.13: Chebyshev's Inequality

For real numbers $a_1 \geq a_2 \geq \cdots \geq a_n$ and $b_1 \geq b_2 \geq \cdots \geq b_n$,

$$n\left(\sum_{i=1}^{n} a_i b_i\right) \geq \left(\sum_{i=1}^{n} a_i\right)\left(\sum_{i=1}^{n} b_i\right) \geq n\left(\sum_{i=1}^{n} a_i b_{n-i+1}\right)$$

holds with the equality if at least one sequence is constant.

Proof.

Consider the following cases where $b_1 \geq b_2 \geq \cdots \geq b_n$ are cycled through in the rearrangement inequality.

$$\begin{aligned}
\sum_{i=1}^{n} a_i b_i &\geq a_1 b_1 + a_2 b_2 + \cdots + a_n b_n \geq \sum_{i=1}^{n} a_i b_{n-i+1} \\
\sum_{i=1}^{n} a_i b_i &\geq a_1 b_2 + a_2 b_3 + \cdots + a_n b_1 \geq \sum_{i=1}^{n} a_i b_{n-i+1} \\
&\vdots \\
\sum_{i=1}^{n} a_i b_i &\geq a_1 b_n + a_2 b_1 + \cdots + a_n b_{n-1} \geq \sum_{i=1}^{n} a_i b_{n-i+1}
\end{aligned}$$

Adding the inequalities,

$$n\left(\sum_{i=1}^{n} a_i b_i\right) \geq \left(\sum_{i=1}^{n} a_i\right)\left(\sum_{i=1}^{n} b_i\right) \geq n\left(\sum_{i=1}^{n} a_i b_{n-i+1}\right)$$

and the corollary holds.

§1.2 Advanced Inequalities

§1.3 Techniques and Patterns

One of the many fields in olympiad algebra that has known techniques and patterns that could come very handy would be inequality. This section mostly includes the techniques that can be implemented with simple pattern recognition, and shares some of my personal experience that I built from solving inequality problems.

1.3.1 Telescoping

Telescoping series and generalization powered with classic inequalities like AM-GM and Cauchy Schwarz can really come handy in solving some problems.

Definition 1.3.1

Telescoping series refers to a substantially large series that cancels out to leave only the beginning and the ending terms.

The definition above is not "formal" per se, but an example of such series would be the following.

$$\sum_{k=1}^{n}(a_k - a_{k+1}) = a_1 - a_2 + a_2 - a_3 + a_4 - a_5 + \cdots + a_n - a_{n+1} = a_1 - a_{n+1}$$

Of course telescoping series do not reveal themselves in problems easily. A common form in which they are found is fraction. For instance, the following sequence telescopes.

$$\begin{aligned}\sum_{k=1}^{n}\left(\frac{1}{k(k+1)}\right) &= \sum_{k=1}^{n}\left(\frac{1}{k} - \frac{1}{k+1}\right)\\ &= \frac{1}{1} - \frac{1}{2} + \frac{1}{2} - \frac{1}{3} + \cdots + \frac{1}{n} - \frac{1}{n+1}\\ &= 1 - \frac{1}{n+1} = \frac{n}{n+1}\end{aligned}$$

With the idea in mind, let's solve a problem to see how they can be powered with fundamental inequalities to solve problems.

Exercise 1.3.2: 2007 Belarusian MO Final Round Problem 2

Show that the following inequality holds for all $a_1, \ldots, a_{n+1} > 0$.

$$\frac{1}{a_1} + \frac{a_1}{a_2} + \frac{a_1a_2}{a_3} + \cdots + \frac{a_1a_2\cdots a_n}{a_{n+1}} \geq 4(1 - a_1\cdots a_{n+1})$$

Solution.

Before applying any techniques or inequalities, let's first analyze the inequality. Notice that in our left hand side of the inequality, we have products $a_1 \cdots a_k$ that grows after each term. On our right hand side, we have the product $a_1 \cdots a_{n+1}$. Now letting $p_k = a_1 \cdots a_k$

and $p_0 = 1$, we can see that our right hand side becomes $4(p_0 - p_{n+1})$, which hints that the sequence telescopes. Indeed, consider the following expansion.

$$4(1 - a_1 \cdots a_{n+1}) = 4\left\{(1 - a_1) + (a_1 - a_1a_2) + \cdots + (a_1 \cdots a_n - a_1 \cdots a_{n+1})\right\}$$

Matching each expanded term with the terms in left hand side, it suffices to show that

$$\frac{a_1 \cdots a_k}{a_{k+1}} \geq 4(a_1 \cdots a_k - a_1 \cdots a_{k+1})$$

holds. Rearranging the terms the following inequality hold.

$$\frac{a_1^2 a_2^2 \cdots a_k^2}{a_1^2 a_2^2 \cdots a_{k+1}^2} + 4a_1 \cdots a_{k+1} \geq 4a_1 \cdots a_k$$

Because $a_k > 0$ by definition, the inequality holds by AM-GM inequality.

The problem above uses the property of telescoping series and AM-GM inequality for a proof. If you see such form, it would be worth suspecting that we can use telescoping technique.

§1.4 Practice Problems

Note 2

Euclidean Geometry

§2.1 Power of a Point

You might already be very familiar with this topic if you have math competition background. However, I wanted to revisit this topic as it is surprisingly common topic in math olympiad. However, the difference could be finding patterns and more advanced topics from power of a point. With that in mind, let's review some of the topics.

Definition 2.1.1

The **power of a point** of P with respect to a circle ω represented as $P_\omega(P) = PO^2 - r^2$ where O and r the center and radius of ω respectively.

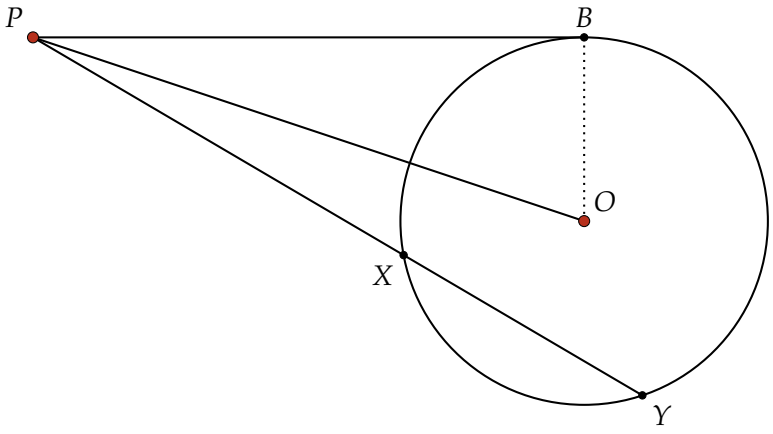
The power of a point is a numeric value that describes the relationship between the circle and the point. Before reviewing the relationship, let's first derive the formula in the definition. The theorem below is the heart of all concepts in this section.

Theorem 2.1.2: Power of a Point Theorem

Let P be a point and ω be a circle on the same space. If lines l_1 and l_2 that passes through P meets ω at A, B and C, D (not necessarily distinct) respectively, then $PA \cdot PB = PC \cdot PD$.

If you read the other notes, you may have noticed that I try my best to include proofs for all theorems included. However, let's skip the proof for now for the purpose of this note.

From the theorem above, we can deduce that $PX \cdot PY$, where X and Y are points on ω such that P, X, Y are collinear, is always consistent. With that in mind, consider the following diagram.



By the theorem, $P_\omega(P) = PX \cdot PY = PB \cdot PB$. Moreover by the definition of a line tangent to a circle, $PB \perp BO$ and $PO^2 = PB^2 + r^2$ by Pythagorean Theorem. Therefore, $P_\omega(P) = PO^2 - r^2$.

Tip.

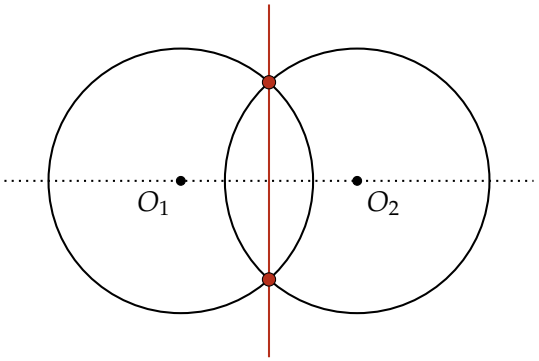
With the definition, we can derive trivial relationship between the power of a point and the location of the point. If $P_\omega(P) < 0$, then P is inside ω. If $P_\omega(P) = 0$, then P is on ω and P is outside ω if and only if $P_\omega(P) > 0$.

Continuing from the definition, we can derive another priceless definition that is one of the foundational concept in olympiad geometry.

Definition 2.1.3

The **radical axis** of two circles ω_1 and ω_2 with different center is the locus of P such that $P_{\omega_1}(P) = P_{\omega_2}(P)$.

Another way of thinking about this is that a point P is on the radical axis if and only if $O_1P^2 - r_1^2 = O_2P^2 - r_2^2$ by definition.



Continuing, there is another very properties of radical axis.

Theorem 2.1.4: Properties of Radical Axis

1. The radical axis is always a straight line.
2. The radical axis of ω_1 and ω_2 is always perpendicular to the line that pass through two origins of ω_1 and ω_2.

Let's first prove the first property.

Proof.

Continuing, we can prove the second property.

Proof.

Five special positions add more lemmas and practice problems

Note 3

Projective Geometry

§3.1 Desargues' Theorem

Desargues' Theorem is a renowned theorem that relates the axis and the centers of perspective of two triangles. Therefore, before we dive into Desargues' Theorem, we need to have a gut feeling on Projective Geometry.

First, consider the two, not necessarily similar, triangles and a point P.

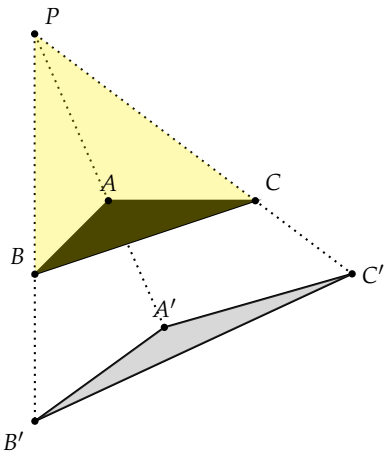


Figure 3.1

From the diagram above, we can think of it as having a light source P that shines $\triangle ABC$. There, we get our outcome $\triangle A'B'C'$. Henceforth, we say that A', B', C' are corresponding points of A, B, C respectively. Similarly, we could state that $\overline{A'B'}, \overline{B'C'}, \overline{C'A'}$ are corresponding sides of $\overline{AB}, \overline{BC}, \overline{CA}$ respectively.

Theorem 3.1.1: Desargues' Theorem

Two triangles are axially perspective if and only if they are centrally perspective. Two triangles are considered axially perspective if there exists the axis of perspectivity and they are centrally perspective if there exists the center of perspectivity.

For simpler explanation, consider the diagram below, which is simply the extension of Figure 3.1.

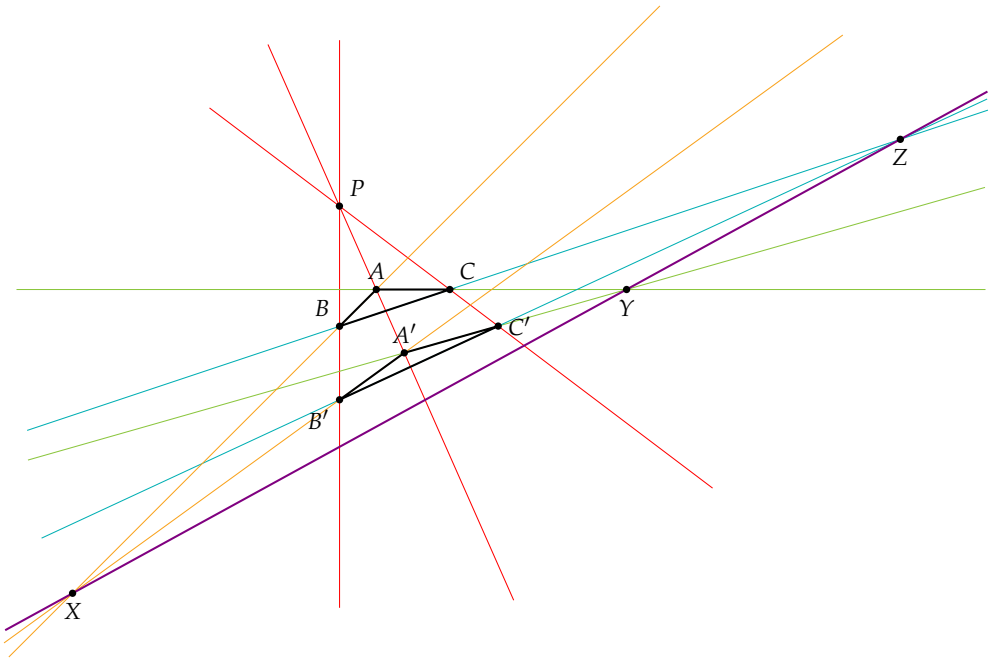


Figure 3.2

Point P, where AA', BB', CC' concur, is the center of perspectivity. Line XZ, where X, Y, Z are collinear, is the axis of perspectivity in the diagram above. If the pairs of corresponding lines are all parallel, it is said that the lines meet at a point of infinity. However, we normally don't discuss about that case.

The theorem is stating that if there exists a point P such that $P = AA' \cap BB' \cap CC'$, then the points $X = AB \cap A'B'$, $Y = BC \cap B'C'$, and $Z = CA \cap C'A'$ are collinear.

Below is a quick example problem.

Exercise 3.1.2

D is a point on $\overline{BC}$ in $\triangle ABC$. Let I_1, I_2 be the incenter of $\triangle ABD$ and $\triangle ACD$ respectively. Moreover, let I_3 and I_4 be ex-centers in respect to $\angle BAD$ and $\angle CAD$ respectively. Show that $\overline{I_1I_2}$, $\overline{I_3I_4}$, and $\overline{BC}$ intersect at one point.

(Video Solution)

Proof.

First, because I_1 and I_2 are the angle bisectors of $\angle ABD$ and $\angle ACD$ respectively, $I = BI_1 \cap CI_2$ is the incenter of $\triangle ABC$. Similarly, we know that $I' = BI_3 \cap CI_4$ is the ex-center of $\triangle ABC$ in respect that $\angle BAC$. Therefore A, I, and I' are collinear.

With similar idea, we could prove that A, I_1, I_3 are collinear as well as A, I_2, I_4. In other words, I_1I_3 and I_2I_4 intersect at point A. Therefore, $BI_1 \cap CI_2$, $I_1I_3 \cap I_2I_4$, and $I_3B \cap I_4C$ are collinear in $\triangle I_1BI_3$ and $\triangle I_2CI_4$. By Desargues' theorem, $\overline{I_1I_2}$, $\overline{I_3I_4}$, and $\overline{BC}$ concurs.

3.1.1 Proof of Desargues' Theorem

Let's try to prove the theorem. The proof involves the use of Menelaus' theorem three times [Pam23].

Proof.

First and foremost, recall that Menelaus' Theorem states the following.

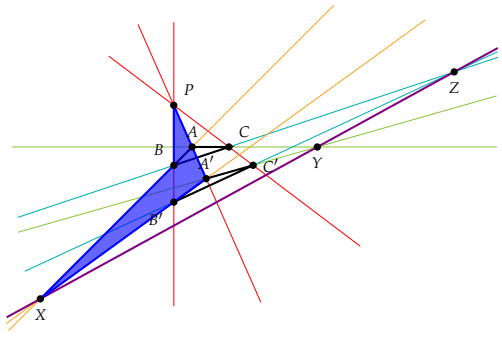
Theorem 3.1.3: Menelaus' Theorem

Assume for $\triangle ABC$, X, Y, Z are defined as $X \in \overline{AB}$, $Y \in \overline{BC}$, and $Z \in \overline{CA}$. Then, the following equation of cross-ratio holds.

$$\frac{XA}{XB} \cdot \frac{YB}{YC} \cdot \frac{ZC}{ZA} = -1$$

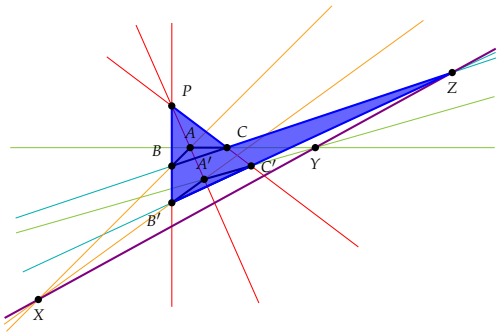
Because Desargues' Theorem must satisfy the if and only if condition, let's first prove that two triangles are axially perspective if they are centrally perspective.

Assuming that $\triangle ABC$ and $\triangle A'B'C'$ are perspective from point P, Menelaus' Theorem could be used.



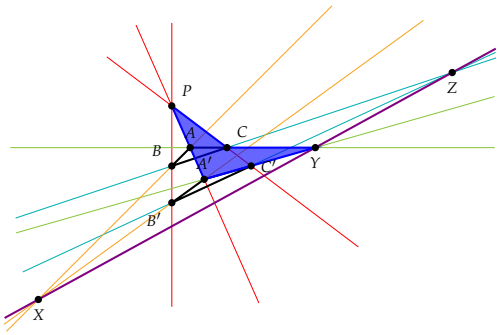
Consider $\triangle PA'B'$ with transverse line XB. Using Menelaus' Theorem, the following equation could be derived.

$$\frac{XA'}{XB'} \cdot \frac{BB'}{BP} \cdot \frac{AP}{AA'} = 1$$



Similarly, consider $\triangle PB'C'$ and transverse line ZC.

$$\frac{ZB'}{ZC'} \cdot \frac{CC'}{CP} \cdot \frac{BP}{BB'} = 1$$



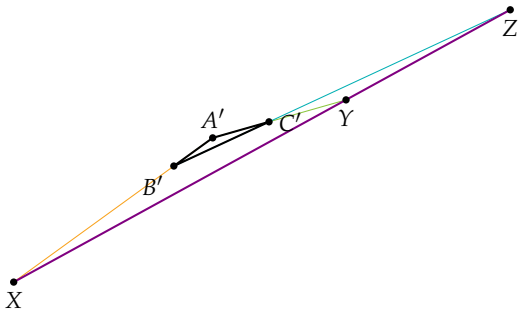
With $\triangle PC'A'$ and line YC,

$$\frac{YA'}{YC'} \cdot \frac{CC'}{CP} \cdot \frac{AP}{AA'} = 1$$
$$\frac{YC'}{YA'} \cdot \frac{CP}{CC'} \cdot \frac{AA'}{AP} = 1$$

Multiplying the equations,

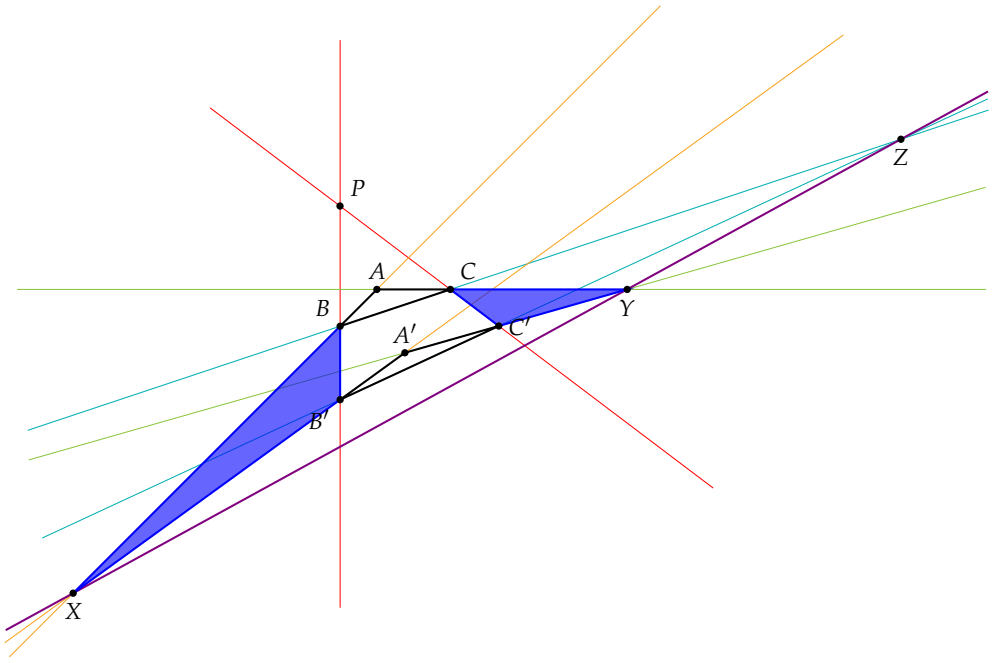
$$\left(\frac{XA'}{XB'} \cdot \frac{BB'}{BP} \cdot \frac{AP}{AA'}\right) \cdot \left(\frac{ZB'}{ZC'} \cdot \frac{CC'}{CP} \cdot \frac{BP}{BB'}\right) \cdot \left(\frac{YC'}{YA'} \cdot \frac{CP}{CC'} \cdot \frac{AA'}{AP}\right) = 1$$
$$\frac{XA'}{XB'} \cdot \frac{ZB'}{ZC'} \cdot \frac{YC'}{YA'} = 1$$

is obtained. Continuing, consider the following diagrams.



By converse of Menelaus' Theorem, it is evident that X, Y, and Z are collinear.

Now that the first statement for if and only if condition is proven, the converse must also be proven. Consider the diagram below where the same construction method is used. Note that if AA', BB', and CC' concur is yet to be known.



Notice that $\triangle XBB'$ and $\triangle YCC'$ are in perspective from point Z. Moreover, the first half of the proof manifests that P, A, and A' must be collinear. By construction, P is on line BB' and CC'. Moreover, it is also on the line AA'. Thus, $\triangle ABC$ and $\triangle A'B'C'$ are centrally perspective if they are axially perspective.

References

[Pam23] Paris Pamfilos. *Desargues' theorem and perspectivities*. https://users.math.uoc.gr/~pamfilos/eGallery/problems/Desargues.pdf. 2023.

Part II

Linear Algebra

Contents

Introducing the Notes

Linear algebra is a foundational branch of mathematics that primarily investigates in linear equations, vectors, and their relations. The application of linear algebra is so immense that numerous fields require foundational understanding of linear algebra. For instance, it would not be possible to gain profound understandings in machine learning and data science without the basics of linear algebra. This collection of notes discuss the essence of linear algebra in perspective of teaching.

One of the most effective way to learn, at least for me, is to teach. During the summer of 2026, I was enrolled in the course XM511 from Stanford Pre-Collegiate University-Level Online Math, instructed by Dr. Margarita Kanarsky [Kan26]. During the lectures, I thought it would be a great idea to write notes on linear algebra just as I wrote the notes for machine learning. Writing the teaching-style notes for machine learning allowed me to differentiate what I truly know.

That being said, my notes do not represent the course. I have included my understandings the supporting textbook [Ric07] from numerous external sources and restructured the notes itself to adhere to the Permission to Use Materials. Therefore, any errors in these notes are my responsibility.

Note 4

Vectors and Vector Spaces

If you have backgrounds on physics or computer science, you are probably familiar with what vectors are. In physics classes, we learn that vectors are quantities with magnitude and direction. In computer science, vectors are introduced as lists or arrays of data. Both definitions capture different aspects of what vectors are. Throughout the notes, we will implement both understandings. In this note, we will discuss fundamental terms for vectors and vector spaces before implementing matrices in vector spaces. Also, most of the heavy abstractions are done in modern algebra, so it will be an introduction.

§4.1 Basic Terms and Notations

First, let's get started by building from common notations and definitions.

Definition 4.1.1

$\mathbb{R}$ and $\mathbb{C}$ denote the set of real and complex numbers respectively. Epsilon symbols $\in$ and $\notin$ are used to denote whether an element is in the set or not.

Examples of using such notations would be $\pi \in \mathbb{C}$ and $i \notin \mathbb{R}$.

Definition 4.1.2

A **vector space** $\mathbb{V}$ is a set of elements and scalars with addition properties and scalar multiplication that adheres to the following ten properties, or **axioms for a vector space**. Moreover, the elements of $\mathbb{V}$ are known as **vectors**.

Vector Addition

A1. **Closure:** $\forall\, u, v \in \mathbb{V}$, $u \oplus v \in \mathbb{V}$.
A2. **Commutativity:** $\forall\, u, v \in \mathbb{V}$, $u \oplus v = v \oplus u$.
A3. **Associativity:** $\forall\, u, v, w \in \mathbb{V}$, $u \oplus (v \oplus w) = (u \oplus v) \oplus w$.
A4. **Additive Identity Element:** $\exists\, 0 \in \mathbb{V}$ such that $\forall\, u \in \mathbb{V}$, $u \oplus 0 = u$, also known as the **zero vector**.
A5. **Additive Inverses:** $\forall\, u \in \mathbb{V}, \exists\, -u \in \mathbb{V}$ such that $u \oplus (-u) = 0$.

Scalar Multiplication

S1. **Closure:** $\forall\, u \in \mathbb{V}$ and scalars α, $\alpha \odot u \in \mathbb{V}$.
S2. **Associativity:** $\forall\, u \in \mathbb{V}$ and scalars α, β, $\alpha \odot (\beta \odot u) = (\alpha \cdot \beta) \odot u$.
S3. **Identity Element:** $\forall\, u \in \mathbb{V}$, $1 \odot u = u$.
S4. **Distributivity I:** $\forall\, u \in \mathbb{V}$ and scalars α, β, $(\alpha + \beta) \odot u = \alpha \odot u \oplus \beta \odot u$.
S5. **Distributivity II:** $\forall\, u, v \in \mathbb{V}$ and scalars α, $\alpha \odot (u \oplus v) = \alpha \odot u \oplus \alpha \odot v$.

Side note here is that for this section of the note, we will be using the binary operation symbols $\oplus$ and $\odot$ to denote vector addition and scalar multiplication. However, from the next section, we will be using more standard notations like $v + u$ and λv for vector addition and scalar multiplication. The reason I decided to use this is that this is the way I first learned and it helped me to emphasize axioms for vector spaces.

Based on the scalars that we have here, we could indicate vector spaces with the following terms.

Definition 4.1.3

If the set of scalars is in $\mathbb{R}$, then the vector space is known as a **vector space over** $\mathbb{R}$, or a **real vector space**. Similarly, if the scalars are in $\mathbb{C}$, then the vector space is denoted as a **vector space over** $\mathbb{C}$, or a **complex vector space**.

One thing to note is that there is a concept called **scalar fields**; however, we will not go too deep into such fields. Before we move on to different concepts in vector spaces, I wanted to discuss about tuples and dimensions.

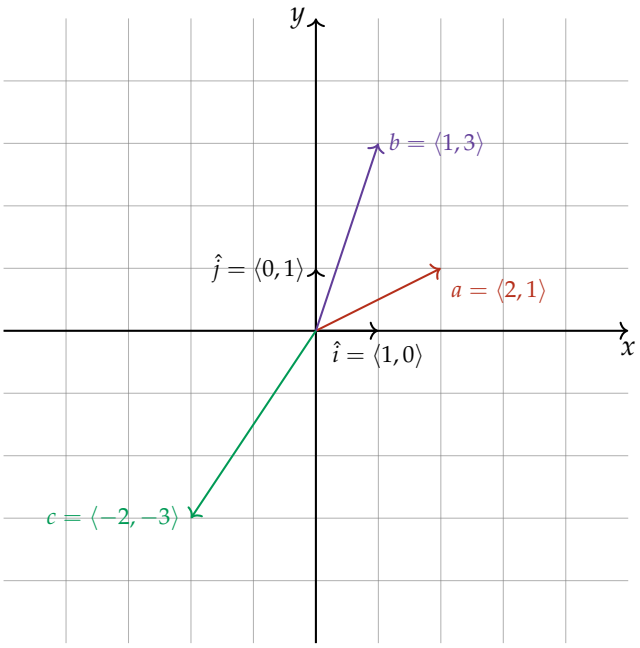
Definition 4.1.4

An n**-tuple** is an ordered array, or list, of n real numbers. For an n-tuple, its dimension is said to be n-dimensional and the set of such real tuples is n**-space**, denoted as $\mathbb{R}^n$.

For instance, the following 4-tuple is in 4-space.

$$\begin{bmatrix} 1 & 2 & 3 & 4 \end{bmatrix} \in \mathbb{R}^4$$

With tuples in mind, let's take a look at how we can represent 2-tuples, 3-tuples, and vectors in $\mathbb{R}^2$ and $\mathbb{R}^3$. The vector space $\mathbb{R}^2$ can simply be conceived as 2-dimensional space, or plane. Also, in linear algebra, we generally don't move the tail of a vector from the origin as we do in introductory physics classes.



From the graph above, there are two vectors $\hat{i}$ and $\hat{j}$. They are called "i hat" and "j hat" respectively.

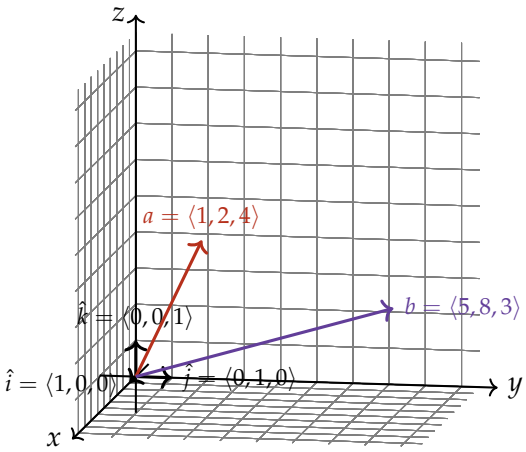
Definition 4.1.5

Unit Vectors are vectors with magnitude 1 that are used to specify direction.

Here, $\hat{i}$ and $\hat{j}$ denote the unit vectors, and they are the standard symbol. If you wondered if unit vectors are used to construct vectors, you are right! In our examples above, we can write each vector as the following.

$$\begin{aligned} a &= \langle 2,1 \rangle = 2\hat{i} + \hat{j} \\ b &= \langle 1,3 \rangle = 1\hat{i} + 3\hat{j} \\ c &= \langle -2,-3 \rangle = -2\hat{i} - 3\hat{j} \end{aligned}$$

Now, let's take a look at vector space $\mathbb{R}^3$. There are two ways of graphing 3-dimensional space: right-handed system and left-handed system. However, we will be using right-hand system throughout the notes.



Here, $\hat{i}$, $\hat{j}$, and $\hat{k}$ are the unit vectors. Going back to the definition of vector spaces, we can write the following equations for $u = \langle x_1, y_1, z_1 \rangle, v = \langle x_2, y_2, z_2 \rangle \in \mathbb{R}^3$ and $\alpha \in \mathbb{R}$.

$$u \oplus v = \langle x_1, y_1, z_1 \rangle + \langle x_2, y_2, z_2 \rangle = \langle x_1 + x_2, y_1 + y_2, z_1 + z_2 \rangle$$
$$\alpha \odot u = \alpha \odot \langle x_1, y_1, z_1 \rangle = \langle \alpha x_1, \alpha y_1, \alpha z_1 \rangle$$

Now that we saw some geometrical interpretation of vector spaces, let's discuss more information with some problems.

§4.2 Fundamental Properties

Let's first start by solving some problems.

Exercise 4.2.1

Identify if the set $\mathbb{S} = \{(x, y) \in \mathbb{R}^2 \mid x + y = 5\}$ is a vector space.

Solution.

We could see if the set satisfies all axioms for a vector space. However, just by looking at the condition $x + y = 5$, we know that there cannot exist a zero vector since $0 + 0 \neq 5$. Because the set violates A4, the set is not a vector space.

Exercise 4.2.2

Identify if the set $\mathbb{S} = \{\langle x, y \rangle \in \mathbb{R}^2 \mid x + y = 0\}$ is a vector space.

Solution.

Unlike the previous problem, we know that there exists a zero vector. For this one, let's go through each axiom. Let $u = \langle x_1, -x_1 \rangle$ and $v = \langle x_2, -x_2 \rangle$. Notice that $u \oplus v = \langle x_1 + x_2, -x_1 - x_2 \rangle \in \mathbb{R}^2$ and satisfies the condition $x_1 + x_2 + (-x_1 - x_2) = 0$. Therefore, A1

is satisfied. Moreover, consider the following equations for A2 and A3 where $w = \langle x_3, -x_3 \rangle$.

$$\begin{aligned} u \oplus v &= \langle x_1, -x_1 \rangle + \langle x_2, -x_2 \rangle = \langle x_1 + x_2, -x_1 - x_2 \rangle \\ &= \langle x_2 + x_1, -x_2 - x_1 \rangle = \langle x_2, -x_2 \rangle + \langle x_1, -x_1 \rangle \\ &= v \oplus u \\ u \oplus (v \oplus w) &= \langle x_1, -x_1 \rangle + (\langle x_2, -x_2 \rangle + \langle x_3, -x_3 \rangle) \\ &= \langle x_1, -x_1 \rangle + \langle x_2 + x_3, -x_2 - x_3 \rangle \\ &= \langle x_1 + x_2 + x_3, -x_1 - x_2 - x_3 \rangle \\ &= \langle x_1 + x_2, -x_1 - x_2 \rangle + \langle x_3, -x_3 \rangle \\ &= (u \oplus v) \oplus w \end{aligned}$$

Therefore, A2 and A3 are satisfied. Furthermore, notice that $\langle 0, 0 \rangle \in \mathbb{R}^2$ and $0 + 0 = 0$, satisfying A4. Lastly for vector addition A5 holds as $\langle -x, x \rangle$ is the additive inverse of $\langle x, -x \rangle$ that satisfies the provided conditions. Thus, all axioms for vector addition are met.

Continuing with scalar multiplication, notice that S1 and S3 satisfy as $\langle x, y \rangle \in \mathbb{R}^2$ and $1 \odot \langle x, y \rangle = \langle x, y \rangle$. Moreover, consider the following equations for $\alpha, \beta \in \mathbb{R}$.

$$\begin{aligned} \alpha \odot (\beta \odot u) &= \alpha \odot \langle \beta x_1, -\beta x_1 \rangle = \langle \alpha\beta x_1, -\alpha\beta x_1 \rangle = \alpha\beta \odot \langle x_1, -x_1 \rangle \\ &= (\alpha\beta) \odot u \\ (\alpha + \beta) \odot u &= \langle (\alpha + \beta)x_1, -(\alpha + \beta)x_1 \rangle = \langle \alpha x_1 + \beta x_1, -\alpha x_1 - \beta x_1 \rangle \\ &= \langle \alpha x_1, -\alpha x_1 \rangle + \langle \beta x_1, -\beta x_1 \rangle = \alpha \langle x_1, -x_1 \rangle + \beta \langle x_1, -x_1 \rangle \\ &= \alpha \odot u \oplus \beta \odot u \\ \alpha \odot (u \oplus v) &= \alpha \langle x_1 + x_2, -x_1 - x_2 \rangle = \langle \alpha x_1 + \alpha x_2, -\alpha x_1 - \alpha x_2 \rangle \\ &= \langle \alpha x_1, -\alpha x_1 \rangle + \langle \alpha x_2, -\alpha x_2 \rangle = \alpha \langle x_1, -x_1 \rangle + \alpha \langle x_2, -x_2 \rangle \\ &= \alpha \odot u \oplus \alpha \odot v \end{aligned}$$

Thus, S2, S4, and S5 are all satisfied. Because all ten axioms for a vector space are met, the set $S = \{\langle x, y \rangle \in \mathbb{R}^2 \mid x + y = 0\}$ is a vector space.

Exercise 4.2.3

Consider vectors in the form $\langle x, y, z \rangle$ where the vector addition is defined as $\langle x, y, z \rangle \oplus \langle x', y', z' \rangle = \langle x + x', y + y', z + z' \rangle$ and scalar multiplication as $\alpha \langle x, y, z \rangle = \langle x^\alpha, y^\alpha, z^\alpha \rangle$. Identify if the vectors form a vector space.

Solution.

First, notice that the vector addition adheres to the standard vector addition. Therefore, we only have to look at scalar multiplication if any axioms are violated. From a glance, S1, S2,

and S3 appears to be satisfied. Therefore, let's take a look at S4.

$$(\alpha + \beta) \odot u = \left(x^{\alpha+\beta}, y^{\alpha+\beta}, z^{\alpha+\beta}\right)$$

Notice that in general, $x^{\alpha+\beta} \neq x^{\alpha} + x^{\beta}$. Therefore, the set of vectors does not form a vector space.

Just as we have vector spaces $\mathbb{R}^n$, we could also define spaces depending on the objects.

Definition 4.2.4

The symbol $\mathbb{P}^n$ denotes the vector space composed of polynomials of degree at most n and adheres to the ten properties. $\mathbb{M}_{m\times n}$ represents the vector space with matrices of order $m \times n$ that adheres to the ten properties.

Therefore, we can say that $\mathbb{R}^n$ is more or less the same as $\mathbb{M}_{1\times n}$. With the understandings above, we can derive some important theorems. Although most of them are intuitive, it's always nice to formally prove them so let's do that now.

Theorem 4.2.5

For all vector spaces, zero vector, or the additive identity element, is unique.

Proof.

For the sake of contradiction, let $0_1, 0_2 \in V$ be distinct additive identity element for a vector space V. By definition, the following equations hold.

$$\begin{aligned} 0_1 \oplus 0_2 &= 0_1 \\ 0_1 \oplus 0_2 &= 0_2 \\ \therefore 0_1 \oplus 0_2 &= 0_1 = 0_2 \end{aligned}$$

Because $0_1 = 0_2$ is contradictory from the fact that 0_1 and 0_2 are distinct, the initial assumption that there exists distinct additive identity element in a vector space is false.

Continuing from the uniqueness of the zero vector in a vector space, we can assert the following theorem.

Theorem 4.2.6

For all $u \in V$ for any vector space V, $v \in V$ such that $u \oplus v = 0$ is unique.

Proof.

For the sake of contradiction, assume $v, w \in V$ are distinct additive inverses of u. By

definition and properties of vector spaces, the following equations hold.

$$v = v \oplus 0 = v \oplus (u \oplus w) = (v \oplus u) \oplus w = 0 \oplus w = w$$

Because $v = w$ contradicts the definition that v and w are unique, the initial assumption that there exists more than one additive inverse of a vector in vector spaces is false.

Just as $x = 0$ if $x + x = x$, we could do similar things in vector spaces.

Theorem 4.2.7

For all $v \in V$ and any vector space V, $v \oplus v = v$ if and only if $v = 0$.

Proof.

First, the first half of the statement can be proven. Consider the following equations.

$$\begin{aligned}(v \oplus v) \oplus (-v) &= v \oplus (-v) = 0 \\ &= v \oplus (v \oplus (-v)) = v \oplus 0 = v\end{aligned}$$

Therefore, if $v \oplus v = v$, then $v = 0$. Moreover, if $v = 0$, then $v \oplus v = 0 \oplus 0 = 0$. Therefore, the premise holds.

Now let's take a look at theorems for scalar multiplications.

Theorem 4.2.8

The following properties hold for all vector space V over $\mathbb{F}$, $u, \mathbf{0} \in V$, and any scalars α.

1. $0 \odot u = \mathbf{0}$
2. $\alpha \odot \mathbf{0} = \mathbf{0}$.
3. $(-1) \odot u = -u$
4. $u = -(-u)$
5. $\alpha \odot u = \mathbf{0}$ if and only if $\alpha = 0$ or $u = \mathbf{0}$.

Let's start with the first proof.

Proof.

Consider the following equation.

$$0 \odot u = (0 + 0) \odot u = 0 \odot u \oplus 0 \odot u$$

By Theorem 4.2.7, $0 \odot u = \mathbf{0}$.

Below is the proof for the second property.

Proof.

Consider the following equation.

$$\alpha \odot \mathbf{0} = \alpha \odot (\mathbf{0} \oplus \mathbf{0}) = \alpha \odot \mathbf{0} \oplus \alpha \odot \mathbf{0}$$

By Theorem 4.2.7, $\alpha \odot \mathbf{0} = \mathbf{0}$

Continuing, here's the proof for the third one.

Proof.

By the properties of vector spaces, $\mathbf{0} = 0 \odot u = (1-1) \odot u = 1 \odot u \oplus (-1) \odot u = u \oplus (-1) \odot u$. Because $(-1) \odot u$ is the additive inverse of u, $(-1) \odot u = -u$.

Below is the proof for the fourth one.

Proof.

Notice that $-(-u)$ is the additive inverse of $-u$. However, by definition, u is the additive inverse of $-u$. By Theorem 4.2.6, $u = -(-u)$.

Finally, let's prove the last property.

Proof.

It suffices to show the latter half of the statement as the first half is shown by the first two properties. Consider the following equations for $\alpha \neq 0$.

$$\begin{aligned} \frac{1}{\alpha} \odot (\alpha \odot u) &= \frac{1}{\alpha} \odot \mathbf{0} = \mathbf{0} \\ &= \left(\frac{1}{\alpha} \cdot \alpha\right) \odot u = 1 \odot u = u \end{aligned}$$

Therefore, if $\alpha \neq 0$, then $u = \mathbf{0}$. Similarly, if $u \neq \mathbf{0}$, then $\alpha = 0$ as the product of nonzero numbers is a nonzero number.

We have covered the fundamental concepts and properties in vector spaces. Before we move on with the next topic in vector spaces, I wanted to discuss briefly on dot product as it will frequently appear when you deal with vectors.

Definition 4.2.9

Dot product is an algebraic operation defined as the following for vectors $a = [a_1, \ldots, a_n]$ and $b = [b_1, \ldots, b_n]$

$$a \cdot b = \sum_{k=1}^{n} a_k b_k$$

Here notice that the number of elements of the vectors in dot product must be equal for the

product to be defined. Below are some examples of dot products.

$$\begin{bmatrix}1\\2\\3\end{bmatrix} \cdot \begin{bmatrix}-1\\-2\\-3\end{bmatrix} = 1(-1)+2(-2)+3(-3) = -14$$
$$\begin{bmatrix}0 & 1 & 2\end{bmatrix} \cdot \begin{bmatrix}0 & 1 & 0\end{bmatrix} = 0\cdot 0+1\cdot 1+2\cdot 0 = 1$$

Note that this is very different from the matrix multiplication that we will discuss in the next note and we must write the notation $\cdot$ to ensure that we are not doing matrix multiplication. With that in mind, let's continue our discussion with subspaces.

§4.3 Subspaces

In the previous section, we discussed that all ten axioms must be satisfied for a set to be a vector space. However, testing and proving that a set satisfies all ten axioms, to be honest, is quite tedious. The good news is that we only need to prove three conditions to show that a set is a vector space thanks to **subspaces**!

Definition 4.3.1

A **subspace** of a vector space V is a subset of V that uses the same definition of vector addition and scalar multiplication from V.

Now for a set to be a subspace, it must satisfy the following conditions.

Theorem 4.3.2

A non-empty subset S of a vector space V is a subspace of V if and only if the following conditions are satisfied.

1. **Closure for Addition:** $\forall u, v \in S, u+v \in S$
2. **Closure for Scalar Multiplication:** $\forall u \in S$ and scalars α, $\alpha u \in S$

Proof.

Notice that it suffices to show that S is a vector space, or adhere to the then axioms for a vector space. Moreover, A1 and S1 are assumed to be true by the initial condition.

By the second condition, $\mathbf{0} = 0u \in S$ and $-u = (-1)u \in S$. Therefore, A4 and A5 are satisfied. For A2 and A3, $u, v, w \in V$ is true for $u, v, w \in S$ by definition. Moreover, because V is a vector space, A2 and A3 hold. Similarly, because V is a vector space that satisfies S1, S2, S3, S4, and S5, its subset S also satisfies the conditions. Therefore, if the two closure conditions are met and S is a subset of V, then S is a subspace of V.

Now the good thing is that subspaces are vector spaces, so it suffices to show that a set is a subspace of a known vector space with the two conditions shown above to show that it is a

vector space. Let's take a look at an example.

Exercise 4.3.3

Determine whether $S = \{(x, y, z) \in \mathbb{R}^3 \mid y = 0\}$ is a vector space.

Solution.

Notice that S is a non-empty subset of $\mathbb{R}^3$, which is a known vector space. Therefore, it suffices to show that $u = (x_1, 0, z_1), v = (x_2, 0, z_2) \in S$ and scalars α satisfy the closure conditions for addition and scalar multiplication.

Notice that $u + v = (x_1 + x_2, 0, z_1 + z_2) \in S$. Moreover, $\alpha u = (\alpha x_1, 0, \alpha z_1) \in S$. Therefore, S is a vector space, more specifically a subset of $\mathbb{R}^3$.

As you can see from the solution above, subspaces really made our lives easier. However, to make it even better, we can boil it down to a single condition.

Corollary 4.3.4

For scalars α and β and $u, v \in S$ where S is a non-empty subset of a vector space V that has the same operations for vector addition and scalar multiplication, S is a subspace if and only if the following condition is satisfied.

$$\alpha u + \beta v \in S$$

Proof.

First and foremost, the first half of the statement can be proven. If $\alpha u + \beta v \in S$, then $u + v \in S$ where $\alpha = \beta = 1$ since α and β are all scalars. Similarly, $\alpha u + \beta v \in S$ implies that $\alpha u \in S$ where $\beta = 0$. By Theorem 4.3.2, S is a subspace of V.

Continuing with the second half of the statement, if S is a subset of V, then $\alpha u, \beta v \in S$. Therefore, $\alpha u + \beta v \in S$, proving the corollary.

More examples on this can be found in the practice problems below. Now let's get into the topics of **linear combination** and the **span** of a set.

4.3.1 Linear Combinations and Span

As always, let's start with definitions.

Definition 4.3.5

For scalars α_i and v_i in a vector space V, the vector u defined as the following is a **linear combination of the vectors** v_i.

$$u = \sum_{k=1}^{n} \alpha_k v_k$$

For instance, $[1,2,3]$ is a linear combination of $[2,-2,1]$, $[1,0,2]$, and $[1,-1,1]$ as it can be represented as the following.

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

Now we have a special term for the collection of such vectors.

Definition 4.3.6

The **span of the set of vectors** $\{v_1, \dots v_n\}$ is the set of all possible linear combinations of the vectors.

$$\text{span}\{v_1, \dots, v_n\} = \left\{ \sum_{k=1}^{n} \alpha_k v_k \;\middle|\; \alpha_k \in \mathbb{R} \right\}$$

With the definition, we could notice an important theorem.

Theorem 4.3.7

For a vector space V and its subset $S = \{v_1, \dots, v_n\}$, $\text{span}(S)$ is a subspace of V.

Proof.

First and foremost, notice that $\mathbf{0} \in \text{span}(S)$ since all the scalars in linear combination can equal zero. Moreover, let $u = \sum_{k=1}^{n} \alpha_k v_k$ and $v = \sum_{k=1}^{n} \alpha_k v_k$ for any scalars α_k and β_k. By definition, $\alpha u + \beta v \in \text{span}(S)$ is true. Therefore, by Corollary 4.3.4, $\text{span}(S)$ is a subspace of V.

Side note here is that there are spans of infinite sets and they adhere to the theorem above. Continuing with the definition of the span of a set and its relation with subspace, we can show the two following theorems.

Theorem 4.3.8

Consider the following statements for a vector space V and its subspace W and non-empty subset S such that $S \subset W$.

1. $\text{span}(S) \subset W$
2. The intersection of all subspaces containing S is $\text{span}(S)$.

Let's start by proving the first statement.

Proof.

Let $u = \sum_{k=1}^{n} \alpha_k v_k \in \text{span}(S)$. By definition, $v_k \in W$ for integer $k \in [1, n]$. Moreover, $u \in W$ by Corollary 4.3.4 and $\text{span}(S) \subset W$.

Below is the proof for the second one.

Proof.

First and foremost, notice that by the previous statement, it was shown that $\text{span}(S)$ is included in all subspaces that contains S. Let A be the intersection of all subspaces that contain S. Therefore, $\text{span}(S) \subset A$. Moreover, by Theorem 4.3.7 $\text{span}(S)$ is a subspace of V that contains S. Thus, $A \subset \text{span}(S)$. Because $\text{span}(S) \subset A$ and $A \subset \text{span}(S)$, $\text{span}(S) = A$.

Next, let's conclude our discussion on linear combination and span of a set of vectors by proving the following statements.

Theorem 4.3.9

For a vector space V and its subsets $S, T \subset V$, consider the following statements.

1. If $T \subset S$, then $\text{span}(T) \subset \text{span}(S)$.
2. $\text{span}(\text{span}(S)) = \text{span}(S)$.
3. If $T \subset \text{span}(S)$, then $\text{span}(T) \subset \text{span}(S)$.
4. For nonzero vectors $v_1, \ldots, v_n \in V$, scalar $\lambda \in \mathbb{R}$, and $1 \leq j, k \leq n$ such that $\lambda \neq 0$ and $j \neq k$, the following equations hold.

$$\text{span}\{v_1, \ldots, v_k, \ldots, v_n\} = \text{span}\{v_1, \ldots, v_k + \lambda v_j, \ldots, v_n\}$$
$$\text{span}\{v_1, \ldots, v_k, \ldots, v_n\} = \text{span}\{v_1, \ldots, \lambda v_k, \ldots, v_n\}$$

There's a lot to prove, but let's do them one by one starting with the first one.

Proof.

Notice that since all $t_k \in T$ satisfy $t_k \in S$, all $\sum_{k=1}^{n} c_k t_k \in \text{span}(T)$ for some appropriate n and constant c_k also satisfy $\sum_{k=1}^{n} c_k t_k \in \text{span}(S)$. Therefore, $\text{span}(T) \subset \text{span}(S)$.

Below is the proof for the second statement, which is a corollary of the first statement.

Proof.

First, notice that $S \subset \text{span}(S)$. By the first statement, $\text{span}(S) \subset \text{span}(\text{span}(S))$ holds. Moreover, notice that because $\text{span}(S)$ contains all linear combinations of S, making more linear combinations of the linear combinations of S will not add more elements. Therefore, $\text{span}(\text{span}(S)) \subset \text{span}(S)$ and $\text{span}(\text{span}(S)) = \text{span}(S)$.

An interesting fact about this property is that this applies for any subset S, finite or infinite. This is because even if $\text{span}(S)$ is infinite, it only uses finite number of vectors to create each linear combination. Continuing, let's prove the third statement.

Proof.

Notice that by the first statement, $\text{span}(T) \subset \text{span}(\text{span}(S))$. Continuing with the second statement shows that $\text{span}(T) \subset \text{span}(S)$.

Finally, here is the proof for the last statement.

Proof.

First, it is self-evident that $\text{span}\{v_1, \ldots, v_k + \lambda v_j, \ldots, v_n\} \subset \text{span}\{v_1, \ldots, v_k, \ldots, v_n\}$ since all linear combination of the set of vectors on the left can be represented as the linear combination of the set of vectors on the right. Moreover notice that v_k can be represented as $v_k = (v_k + \lambda v_j) - \lambda v_j$ even if $k = j$. Therefore, $v_k \in \text{span}\{v_1, \ldots, v_k + \lambda v_j, \ldots, v_n\}$ and $\text{span}\{v_1, \ldots, v_k, \ldots, v_n\} \subset \text{span}\{v_1, \ldots, v_k + \lambda v_j, \ldots, v_n\}$ by statement 3. Therefore, the equations in the statement hold.

With the ideas of linear combinations and the span of a set of vectors in mind, let's discuss about linear independence.

4.3.2 Linear Independence

One of the most important reasons why we use Linear Algebra is due to efficiency. As you could have guessed, most vector spaces have infinitely many vectors. Of course, there must be an efficient way of representing the vector space instead of manually listing. Determining such representative vectors involves numerous factors, and that's what we will discuss with linear independence, basis, and dimension.

Definition 4.3.10

Consider the linear combination of $V = \{v_1, \ldots, v_n\}$ in a vector space and scalars $c_1, \ldots, c_n$ that satisfy the following equation.

$$\sum_{k=1}^{n} c_k v_k = \mathbf{0}$$

The set of vectors V is **linearly dependent** if there exists scalars $c_1, \ldots, c_k$ that are not all equal to zero. The set of vectors is **linearly independent** if all scalars are zero in the equation.

Let's take a look at a quick example.

Exercise 4.3.11

Determine whether $V = \{\langle 1, 1\rangle, \langle 2, 3\rangle \in \mathbb{R}^2\}$ is linearly independent.

Solution.

By definition, for the set to be linearly independent, all scalars c_1 and c_2 must be zero for the following equation to hold.

$$c_1\langle 1, 1\rangle + c_2\langle 2, 3\rangle = \langle 0, 0\rangle$$

Writing in system of linear equations, the equation above can be written as the following.

$$c_1 + 2c_2 = 0$$
$$c_1 + 3c_2 = 0$$

Solving the equation, $c_1 = c_2 = 0$ and no other set of scalars can lead to the equation. Therefore, the set of vectors is linearly independent.

One interesting observation that we can make is that for $n \in \mathbb{Z}$, if the set of vectors in $\mathbb{R}^n$ contains more than n elements, then the set is linearly dependent. We will discuss more on this later when we discuss matrix and its application in linear systems, but for such cases, we have more variables than linear equations, which implies that there exists infinitely many solutions. Therefore, such a set of vectors must be linearly dependent. Continuing, below is a theorem that we can deduce from the definition.

Theorem 4.3.12

A set with a finite number of nonzero vectors is linearly dependent if and only if a vector in the ordering of the set can be represented as a linear combination of vectors preceding it.

Proof.

First and foremost, the first half of the theorem can be shown. Consider the set of vectors $\{v_1, \ldots, v_{j-1}, v_j, \ldots, v_n\}$. Assume for scalars c_k, the following equation is true.

$$v_j = \sum_{k=1}^{j-1} c_k v_k$$

Therefore, $\mathbf{0} = \sum_{k=1}^{j-1} c_k v_k - v_j$. Letting $c_k = 0$ for integer $k \in (j, n]$, the following equation holds.

$$\sum_{k=1}^{j-1} c_k v_k - v_j + \sum_{k=j+1}^{n} c_k v_k = \mathbf{0}$$

Because at least one scalar is nonzero, the set of vectors $\{v_1, \ldots, v_n\}$ is linearly dependent.

Continuing, let the set of vectors $\{v_1, \ldots, v_n\}$ be linearly dependent. For the corresponding scalars c_k, let c_j be the last scalar that is nonzero. Thus, the following equation holds.

$$\sum_{k=1}^{j} c_k v_k + \sum_{k=j+1}^{n} 0 v_k = \mathbf{0}$$

Rearranging the equation, $v_k = -\sum_{k=1}^{j-1} \frac{c_k}{v_j} v_k$, which is a linear combination. Because the converse holds, the theorem is true.

From this theorem, we can deduce a corollary.

Corollary 4.3.13

For a vector space $V = \text{span}(S)$ and linearly dependent set $S = \{v_1, \ldots, v_n\} \subset V$, then there exists a set $T \subset S$ such that $|T| = n - 1$ and $V = \text{span}(T)$.

Proof.

By Theorem 4.3.12, there exists element $v_j \in S$ such that v_j is the linear combination of the previous elements as listed in S.

Let $T = \{v_1, \ldots, v_{j-1}, v_{j+1}, \ldots, v_n\}$. Consider the following equations for scalars d_k and c_k.

$$\begin{aligned} V &= \sum_{k=1}^{n} d_k v_k = \sum_{k=1}^{j-1} d_k v_k + d_j v_j + \sum_{k=j+1}^{n} d_k v_k \\ &= \sum_{k=1}^{j-1} d_k v_k + d_j \left(\sum_{k=1}^{j-1} c_k v_k \right) + \sum_{k=j+1}^{n} d_k v_k \\ &= \sum_{k=1}^{j-1} (d_j c_k + d_k) v_k + \sum_{k=j+1}^{n} d_k v_k = \text{span}(T) \end{aligned}$$

Therefore, the corollary holds.

Continuing from the corollary, the following theorems could be asserted.

Theorem 4.3.14

Consider the following statements for set S.

1. If S contains a zero vector, then it is linearly dependent.
2. If $|S| = 1$, then it is linearly dependent if and only if its element is the zero vector.
3. If $|S| = 2$, then it is linearly dependent if and only if a vector is a scalar multiple of the other.
4. If S is linearly dependent, then every set that contains S is linearly dependent.
5. If S is linearly independent, then all its subsets are linearly independent.

There's quite a bit to prove! Let's start with the first one.

Proof.

Let $S = \{\mathbf{0}, v_1, \ldots, v_n\}$. Notice that $\alpha(\mathbf{0}) + \sum_{k=1}^{n} 0 v_k = \mathbf{0}$ for any scalar α. Therefore, S is linearly dependent.

Below is the proof for the second one.

Proof.

Let $S = \{v\}$. Notice that if $v \neq \mathbf{0}$, it is not possible for a nonzero scalar α to make $\alpha v = \mathbf{0}$. Therefore, if $|S| = 1$, then $v = \mathbf{0}$. For the converse, if $v = \mathbf{0}$, then any scalar α will let $\alpha v = \mathbf{0}$. Therefore the statement holds.

Here is the proof for the third statement.

> *Proof.*
>
> Let $S = \{v_1, v_2\}$. First, if $v_2 = \alpha v_1$ for any scalar α, then $\mathbf{0} = \alpha v_1 - \alpha v_1 = \alpha v_1 - v_2$. Therefore, S is linearly dependent. For the converse, $c_1 v_1 + c_2 v_2 = 0$ for scalars c_1 and c_2. Therefore, assuming that the scalars are nonzero without loss of generality, $v_2 = -\frac{c_1}{c_2} v_1$. Thus, the statement is true.

Below is the proof for the fourth one.

> *Proof.*
>
> Let $S = \{v_1, \ldots, v_n\}$ and $T = \{u_1, \ldots, u_n, v_1, \ldots, v_n\}$. By definition, there exists c_k that are not all zero such that $\sum_{k=1}^{n} c_k v_k = \mathbf{0}$. Because $\sum_{k=1}^{n} 0 u_k + \sum_{k=1}^{n} c_k v_k = \mathbf{0}$, T is also linearly dependent.

Finally, here is the proof for the last statement.

> *Proof.*
>
> Let $T \subset S$ where S is linearly independent. Assume for the sake of contradiction that there exists a subset T that is linearly dependent. However, if T is linearly dependent, then S is linearly dependent by the previous property, which is a contradiction. Therefore, by contradiction, if S is linearly independent then all its subsets T are also linearly independent.

With this, we have concluded our discussion on linear dependence and independence. Before concluding this note, let's discuss basis and dimension of a set of vectors.

4.3.3 Basis and Dimension

As always, let's start with definition.

Definition 4.3.15

For a vector space V and its subset S, if $\text{span}(S) = V$, then S is a spanning set for V.

When we say aloud, we say that S spans V. For instance, the set $\{\langle 1, 0\rangle, \langle 0, 1\rangle\}$ spans $\mathbb{R}^2$ since all vectors $\langle a, b\rangle \in \mathbb{R}^2$ can be represented as the sum $a\langle 1, 0\rangle + b\langle 0, 1\rangle$.

Definition 4.3.16

A linearly independent subset that spans V is a **basis** for V.

Continuing from the previous example, we can see that $\{\langle 1, 0\rangle, \langle 0, 1\rangle\}$ not only spans $\mathbb{R}^2$, but is also a basis for $\mathbb{R}^2$ since it is linearly independent. As you can see, the same will apply if we generalize it to $\mathbb{R}^n$.

Definition 4.3.17

The **standard basis for** $\mathbb{R}^n$ is the set $S = \{e_1, \dots e_n\}$ where e_i is an element of $\mathbb{R}^n$ with 1 in its i^{th} entry and 0 elsewhere.

We could do the same for polynomials. First, notice that the set $\{1, x, x^2\}$ spans $\mathbb{P}^2$. Moreover, the set is linearly independent. Therefore, $\{1, x, x^2\}$ is a basis for $\mathbb{P}^2$.

Definition 4.3.18

The **standard basis for** $\mathbb{P}^n$ is the set $S = \{1, x, x^2, \dots, x^n\}$.

From the two definitions above, we can notice that there are finite elements for the two standard bases for $\mathbb{R}^n$ and $\mathbb{P}^n$. We call such vector spaces **finite-dimensional**.

Definition 4.3.19

A vector space with a basis of finite elements is called **finite-dimensional**. Whereas, **infinite dimensional** vector spaces do not contain a basis with finite elements.

An example of infinite dimensional vector space would be $\mathbb{P}$ since the basis is required to be an infinite set. However, we will not be discussing infinite dimensional vector spaces. From the definitions above, we could establish a theorem.

Theorem 4.3.20

For a basis S of a vector space V such that $|S| = n$, any set in V with more than n elements is linearly dependent.

Proof.

Let $S = \{v_1, \dots, v_n\}$ and $T = \{u_1, \dots, u_m\}$ such that $S, T \in V$ and $m > n$. Notice that because $\text{span}(S) = V$ and $T \in V$, there exists a set of constants a_{ij} that satisfies the following equation (note that we will be using this indices in the next note when we discuss matrices.).

$$\begin{aligned} u_1 &= \sum_{j=1}^{n} a_{1j} v_j \\ u_2 &= \sum_{j=1}^{n} a_{2j} v_j \\ &\vdots \\ u_m &= \sum_{j=1}^{n} a_{mj} v_j \end{aligned}$$

Therefore, it suffices to show that there exists constants $c_1, \dots, c_m$ such that not all are zero

and that satisfies the following equation.

$$\sum_{i=1}^{m}\sum_{j=1}^{n} c_i a_{ij} v_j = \mathbf{0}$$

Continuing, it suffices to show that the variables c_i and constants a_{ij} satisfy the following equations where not all c_i are zero.

$$\begin{aligned}
\sum_{i=1}^{m} c_i a_{i1} &= 0 \\
\sum_{i=1}^{m} c_i a_{i2} &= 0 \\
&\vdots \\
\sum_{i=1}^{m} c_i a_{in} &= 0
\end{aligned}$$

Notice that there are m variables and n equations. It is known that for a linear system of equations, there exists an infinite number of solutions if there are more variables than the equations (don't worry we will prove this in future notes.). Because $m > n$, there exists $c_1, \ldots c_m$ that are not all zero and that satisfy the equations for linear dependence, the theorem holds.

Continuing, we can derive three corollaries from the theorem.

Corollary 4.3.21

If a basis for the vector space V contains n elements, then any linearly independent subsets of V contain at most n elements.

Proof.

By definition, a basis of a vector space V spans V. Because any linearly dependent subsets of V contain more than n elements by Theorem 4.3.20, any linearly independent subsets of V must not contain more than n elements.

Below is the second corollary.

Corollary 4.3.22

If V is a finite vector space, every basis of V will contain the same number of elements.

Proof.

Let S and T be bases of V with p and q elements respectively. By definition, the bases of a vector space are linearly independent. By the previous corollary, it is evident that $q \leq p$ and $p \leq q$ are true. Therefore, $p = q$ and the corollary holds.

Now before continuing with our third corollary, let's first define what the dimension of a vector

space is.

Definition 4.3.23

For a finite dimensional vector space V, the number of elements in any basis of V is called the **dimension** of V, denoted as $\dim(V)$.

For instance, recall the standard basis of $\mathbb{R}^n$. From the number of elements in the basis, we can write that $\dim(\mathbb{R}^n) = n$. Similarly, we can also notice that $\dim(\mathbb{P}^n) = n + 1$. Now let's continue with our third corollary.

Corollary 4.3.24

Every set S in an n-dimensional vector space such that $|S| = n + 1$ is linearly dependent.

Proof.

By definition, a basis of an n-dimensional vector space V contains n elements. By Theorem 4.3.20, since $n + 1 > n$, S must be linearly dependent.

One fun fact to note is that for the vector space $\{0\}$, we say $\dim(\{0\}) = 0$ by convention. Moreover, consider the following theorem.

Theorem 4.3.25

For a basis $S = \{v_1, \ldots, v_n\}$ of a vector space V, any vector $u \in V$ can be written as a unique linear combination of S.

Proof.

For the sake of contradiction, assume that u can be written as $u = \sum_{k=1}^{n} c_k v_k = \sum_{k=1}^{n} d_k v_k$ where $c_k \neq d_k$ for some choice of k. Therefore, the following equations can be obtained.

$$\sum_{k=1}^{n} c_k v_k = \sum_{k=1}^{n} d_k v_k$$

$$\sum_{k=1}^{n} (c_k - d_k) v_k = 0$$

Note that by definition, S is linearly independent and $c_k - d_k = 0$ for all integer $k \in [1, n]$. Because it contradicts with the assumption that $c_k \neq d_k$ for some choice of k, the theorem holds.

Now with the theorem in mind, because we know that the linear combination of a basis to represent a vector is unique, we can establish the following definition.

Definition 4.3.26

For a basis $S = \{v_1, \ldots, v_n\}$ of a vector space V and any vector $u = \sum_{k=1}^{n} c_k v_k \in V$, the **coordinates of u with respect to** S, denoted as $(u)_S$ is the list of coefficients $c_1, \ldots, c_n$.

We often represent the coordinates with n-tuple. To see that, let's take a look at the following example.

Exercise 4.3.27

Find the coordinate of $f(x) = x^2 + 2x + 3$ with respect to the basis $S = \{x^2, 2x^2 + 1, x\}$ in the vector space $\mathbb{P}^2$.

Solution.

Consider the following equation.

$$f(x) = 3\left(2x^2 + 1\right) - 5\left(x^2\right) + 2(x)$$

Therefore, the coordinate of $f(x)$ can be represented as the following 3-tuple.

$$\begin{bmatrix} -5 \\ 3 \\ 2 \end{bmatrix}_S$$

Lastly, let's conclude our discussion on basis and dimension with a theorem.

Theorem 4.3.28

Consider the following statements for a vector space V and its subset $S \subset V$.

1. There exists a basis for V that is a subset of S if $\text{span}(S) = V$.
2. There exists a basis for V that includes all elements in S if S is linearly independent.

Let's start by proving the first statement.

Proof.

Notice that by definition, a basis of a vector space V is a linearly independent set that spans V. Therefore, it suffices to show that there exists a linearly independent set that spans V that is also a subset of S.

By Theorem 4.3.12, a subset of S is linearly dependent if any only if a vector can be represented as the linear combination of the preceding vectors. Let T be a subset of S without the vectors that can be represented as the linear combination of the preceding vectors. Notice that T spans V as the vector removed can be represented as the linear combination of the other vectors. Moreover, because T is linearly independent, there exists a basis for V that is a subset of S.

Below is the proof for the second statement.

Proof.

Let T be a basis for V and $A := S \cup T$. By definition, $\text{span}(A) = V$. Construct $B \in A$ by removing all vectors that can be represented as a linear combination of the preceding

vectors. Notice that no elements from S will be removed as S is linearly independent. By construction, B spans V and is also linearly independent. Because $S \subset B$, the statement holds.

We discussed fundamental topics of vector spaces. Of course there are a lot more to discuss in vector spaces; however, to do so, we need some understanding of matrices. So let's pause our discussions on vector spaces for now, and move on to matrices and their applications in linear system of equations before looking at their implications in vector spaces.

§4.4 Practice Problems

1. Consider the set $S = \{(x, y, z) \in \mathbb{R}^3 \mid x = 0\}$. Identify whether the set is a vector space under the standard vector addition and scalar multiplication.

2. Determine whether $S = \{f \in \mathbb{R} \mid f'(0) = 0\}$ is a vector space for differentiable functions f.

3. For all $n \in \mathbb{Z}$ and $a_i \in \mathbb{R}$, determine whether

$$S = \left\{ (x_1, x_2, \ldots, x_n) \in \mathbb{R}^n \,\middle|\, \sum_{k=1}^{n} a_k x_k = 0 \right\}$$

   is a vector space.

4. Show that the intersection of subspaces of a vector space V is also a subspace of V.

5. Show that for distinct real numbers $a_1, \ldots, a_n$, the set $\{e^{a_1 x}, \ldots, e^{a_n x}\}$ is linearly independent.

6. Let $A, B \in \mathbb{R}^3$ be lines defined as $x = y = z$ and $3x = 4y = 5z$ respectively. If $C = \{a + b \mid a \in A, b \in B\}$, show that $C = A \oplus B$ where $\oplus$ represents the direct sum [Erd20].

7. If $\{a, b, c\}$ is linearly independent in a vector space V, show that $\{a + b, b + c, c + a\}$ is also linearly independent [Erd20].

8. Show that the set of all polynomials in $\mathbb{P}^3$ that includes the origin is a subspace of $\mathbb{P}^3$.

Note 5

Matrices

Matrices are one of the fundamental components that build linear algebra. This note discusses the fundamental concepts and properties of matrices.

§5.1 Basic Terms and Notations

As always, let's get started with the definition.

Definition 5.1.1

A **matrix** is a 2-dimensional array composed of entries called **elements**. A matrix is said to have **order** "m by n" and written as $m \times n$ if there are m rows and n columns.

Technically, the elements of a matrix can be anything including functions. However, the majority of the matrices discussed in linear algebra will have numeric elements. Let's take a look at some examples.

$$A = \underbrace{\begin{bmatrix} 1 & 2 & 3 & 4 \\ -1 & -2 & -3 & -4 \\ 0 & -1 & 0 & 1 \end{bmatrix}}_{4 \text{ columns}} \Bigg\} 3 \text{ rows}, \qquad B = \underbrace{\begin{bmatrix} \pi \\ e \\ 0 \\ 1 \end{bmatrix}}_{1 \text{ column}} \Bigg\} 4 \text{ rows}$$

Here, matrix A has order 3×4 and matrix B has order 4×1. The form of a matrix that has order $m \times n$ can be generalized as the following.

$$A = \underbrace{\begin{bmatrix} a_{1,1} & a_{1,2} & a_{1,3} & \cdots & a_{1,n} \\ a_{2,1} & a_{2,2} & a_{2,3} & \cdots & a_{2,n} \\ a_{3,1} & a_{3,2} & a_{3,3} & \cdots & a_{3,n} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ a_{m,1} & a_{m,2} & a_{m,3} & \cdots & a_{m,n} \end{bmatrix}}_{m \text{ columns}} \Bigg\} n \text{ rows}$$

It is also worth noting that matrix A is often represented as $[a_{ij}]$ and the elements as a_{ij} like a_{12} for $a_{1,2}$.

Definition 5.1.2

An element a_{ij} is said to have row index of i and column index of j.

As you probably have guessed, we assign special names to some matrices depending on m and n.

Definition 5.1.3

Square matrices are matrices with an equal number of rows and columns. The elements with the same row and column index are called **diagonal elements** of the matrix. The collection of such elements is called **main diagonal**, or **principal diagonal**.

What if either m or n is equal to 1? Yes, we do have special terms for them.

Definition 5.1.4

Row matrices are matrices that are composed of only one row. A **column matrix** is an array composed of one column.

If you have a machine learning background, you likely have heard of the term "high dimensional arrays or vectors." Here, the term "dimension" works the same way.

Definition 5.1.5

The **dimension** of a row or column matrix is the number of **components**, or the elements. An **n-tuple** is an n-dimensional row or column vector.

Let's take a look at an example from the previous matrix B reproduced below.

$$B = \underbrace{\begin{bmatrix} \pi \\ e \\ 0 \\ 1 \end{bmatrix}}_{\text{1 column}} \Big\} \text{4 rows}$$

Here, matrix B is a 4-dimensional matrix with four components. This column matrix can also be called as 4-tuple.

We could also define matrices by their elements.

Definition 5.1.6

A **zero matrix** is a matrix with all its elements as 0.

With the fundamental terms in mind, let's discuss the properties of matrices.

§5.2 Fundamental Properties

First, when are two matrices "equal" to each other? Consider the following definition for a formal explanation.

Theorem 5.2.1

Two matrices A and B with orders $m \times n$ and $p \times q$ respectively are said to be equal to each other if they retain the same order and corresponding elements, i.e. they satisfy the following properties.

1. $m = p$
2. $n = q$
3. For elements a_{ij} in A and b_{ij} in B, the following equation satisfies.

$$a_{ij} = b_{ij} \, \forall \, i \in [1, m] \text{ and } j \in [1, n]$$

Although this is very trivial and intuitive, it is always nice to formally define concepts!

Theorem 5.2.2

For matrices A and B with same order, the element in the (i, j)-entry of their sum $A + B$ is equivalent to the sum of elements in the (i, j)-entry of A and B.

Note that the orders of matrices being added must be identical. We do not consider additions with matrices of different orders. Below is an example of addition.

$$\begin{bmatrix} 1 & 3 & 5 \\ 2 & \pi & 4 \end{bmatrix} + \begin{bmatrix} -2 & 1 & 3 \\ 3 & \pi & -4 \end{bmatrix} = \begin{bmatrix} -1 & 4 & 8 \\ 5 & 2\pi & 0 \end{bmatrix}$$

Similarly, we could see that subtractions are defined in similar way. Subtracting the matrices above, we get the following results.

$$\begin{bmatrix} 1 & 3 & 5 \\ 2 & \pi & 4 \end{bmatrix} - \begin{bmatrix} -2 & 1 & 3 \\ 3 & \pi & -4 \end{bmatrix} = \begin{bmatrix} 3 & 2 & 2 \\ -1 & 0 & 8 \end{bmatrix}$$

Now, as we look at the additions and subtractions of matrices, you probably wondered about the application of additive properties. Intuitively, the fundamental properties of addition can be applied to matrices addition.

Theorem 5.2.3

For matrices A and B with order $m \times n$, $A + B = B + A$.

Proof.

Let $A = [a_{ij}]_{m \times n}$ and $B = [b_{ij}]_{m \times n}$. By definition, the sum $A + B = [a_{ij} + b_{ij}]$. Because

addition is commutative, the following equation holds.

$$A + B = [a_{ij} + b_{ij}] = [b_{ij} + a_{ij}] = B + A$$

Thus, the commutative property of matrix addition holds.

In addition to the commutative property, the associative property is also applicable.

Theorem 5.2.4

For matrices A, B, and C with order $m \times n$, $A + (B + C) = (A + B) + C$.

Proof.

Let $A = [a_{ij}]_{m\times n}$, $B = [b_{ij}]_{m\times n}$, and $C = [c_{ij}]_{m\times n}$. Consider the following equation.

$$\begin{aligned} A + (B + C) &= [a_{ij}] + [b_{ij} + c_{ij}] = [a_{ij} + (b_{ij} + c_{ij})] = [a_{ij} + b_{ij} + c_{ij}] \\ &= [(a_{ij} + b_{ij}) + c_{ij}] = [a_{ij} + b_{ij}] + [c_{ij}] \\ &= (A + B) + C \end{aligned}$$

Thus, the associative property of addition applies to the matrix addition.

One obvious property in addition is that when you add 0 to a number, you get the number itself. The same applies in matrix addition.

Theorem 5.2.5

The sum of matrices $A_{m\times n}$ and $0_{m\times n}$ yields the matrix A, i.e. the zero matrix is the identity elements of matrix addition.

Proof.

Let $A = [a_{ij}]_{m\times n}$. By definition, the sum $A + 0 = [a_{ij} + 0] = [a_{ij}] = A$. Therefore, the zero matrix is the identity element of matrix addition.

Now what if we want to add the same matrix multiple times? Fortunately, we already resolved the issue. For instance, we write $x + x + x$ as $3x$. We can multiply matrices with scalars.

Theorem 5.2.6

For a complex number λ and $A = [a_{ij}]$, $\lambda A = [\lambda a_{ij}]$.

Let's take a look at an example.

Exercise 5.2.7

Compute $2A - 3B$ for A and B defined as the following.

$$A = \begin{bmatrix} 2 & 3 & -1 \\ 1 & 7 & 3 \end{bmatrix}, \quad B = \begin{bmatrix} -2 & 1 & 0 \\ 5 & 3 & 8 \end{bmatrix}$$

Solution.

Before subtracting, we could perform scalar multiplication first.

$$2A - 3B = 2\begin{bmatrix} 2 & 3 & -1 \\ 1 & 7 & 3 \end{bmatrix} - 3\begin{bmatrix} -2 & 1 & 0 \\ 5 & 3 & 8 \end{bmatrix} = \begin{bmatrix} 4 & 6 & -2 \\ 2 & 14 & 6 \end{bmatrix} - \begin{bmatrix} -6 & 3 & 0 \\ 15 & 9 & 24 \end{bmatrix}$$

From here we could subtract two matrices. Therefore, $2A - 3B$ can be represented as following.

$$2A - 3B = \begin{bmatrix} 10 & 3 & -2 \\ -13 & 5 & -18 \end{bmatrix}$$

With the scalar multiplication in mind, we can expand the application of foundational concepts here with matrices.

Theorem 5.2.8

For $A = [a_{ij}]_{m\times n}$, $B = [b_{ij}]_{m\times n}$, and $\lambda_1, \lambda_2 \in \mathbb{C}$, the following properties hold.

1. $\lambda_1(A \pm B) = \lambda_1 A \pm \lambda_1 B$
2. $(\lambda_1 \pm \lambda_2)A = \lambda_1 A \pm \lambda_2 A$
3. $\lambda_1(\lambda_2 A) = (\lambda_1\lambda_2)A$

Let's prove them! Below is the proof for the first property.

Proof.

Consider the following equation.

$$\lambda_1(A \pm B) = \lambda_1([a_{ij}] \pm [b_{ij}]) = \lambda_1([a_{ij} \pm b_{ij}])$$

Note that individual elements are multiplied in scalar multiplication. Moreover, because $\lambda_1(a_{ij} \pm b_{ij}) = \lambda_1 a_{ij} \pm \lambda_1 b_{ij}$, the following equation holds.

$$\lambda_1([a_{ij} \pm b_{ij}]) = [\lambda_1 a_{ij} \pm \lambda_1 b_{ij}] = [\lambda_1 a_{ij}] \pm [\lambda_1 b_{ij}] = \lambda_1 A \pm \lambda_1 B$$

Therefore, $\lambda_1(A \pm B) = \lambda_1 A \pm \lambda_1 B$.

Now let's prove the second property.

Proof.

By definition of scalar multiplication, $(\lambda_1 \pm \lambda_2)A$ can be represented as the following.

$$(\lambda_1 \pm \lambda_2)A = (\lambda_1 \pm \lambda_2)[a_{ij}] = [(\lambda_1 \pm \lambda_2)a_{ij}]$$

Using the distributive property, the following equation holds.

$$[(\lambda_1 \pm \lambda_2)a_{ij}] = [\lambda_1 a_{ij} \pm \lambda_2 a_{ij}] = [\lambda_1 a_{ij}] \pm [\lambda_2 a_{ij}] = \lambda_1 A \pm \lambda_2 A$$

Thus, the equation $(\lambda_1 \pm \lambda_2)A = \lambda_1 A \pm \lambda_2 A$ holds.

Finally, here is the proof to the third property.

Proof.

Consider the following equations.

$$\lambda_1(\lambda_2[a_{ij}]) = \lambda_1[\lambda_2 a_{ij}] = [\lambda_1\lambda_2 a_{ij}] = [(\lambda_1\lambda_2)a_{ij}] = (\lambda_1\lambda_2)[a_{ij}] = (\lambda_1\lambda_2)A$$

Therefore, the property holds.

We have discussed fundamental properties so far. In the next section of the note, let's discuss what matrix multiplication with intuitions and computations.

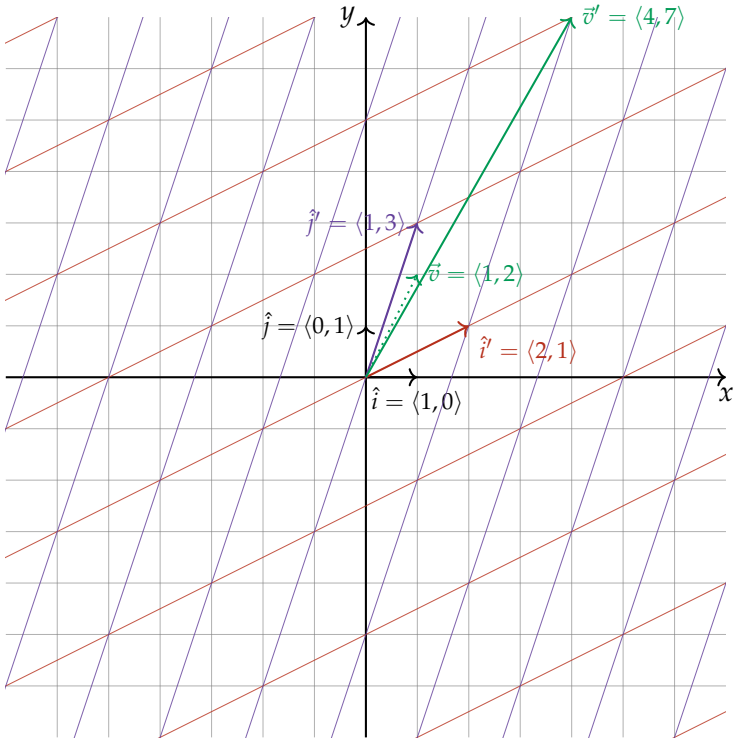
§5.3 Matrix Multiplication

Before we get into formulas and computations, let's build our intuition on what a matrix and its multiplication is supposed to represent. We will discuss **linear transformation** in depth in latter notes, but we can think of it as a function that transforms the vector space. Matrix instructs the function how to transform the space. Let's take a look at a quick example.

Let A be a matrix that has order 2×2. The first column tells how to map the $\hat{i}$ and the second column represents where our $\hat{j}$ is mapped.

$$\begin{bmatrix} 2 & 1 \\ 1 & 3 \end{bmatrix}$$

Matrix A is telling us to map the standard basis $\hat{i} = \langle 1, 0 \rangle$ to $\hat{i}' = \langle 2, 1 \rangle$ and from $\hat{j} = \langle 0, 1 \rangle$ to $\hat{j}' = \langle 1, 3 \rangle$. Consider the diagram below.



After the transformation, you can imagine mapping everything in the standard vector space with respect to the new colored space. That, is what matrix multiplication does. See how a vector $\vec{v} = \langle 1, 2 \rangle$ is now on $\langle 4, 7 \rangle$. Using matrix multiplication, we can represent the mapping as the following.

$$\begin{bmatrix} 2 & 1 \\ 1 & 3 \end{bmatrix} \begin{bmatrix} 1 \\ 2 \end{bmatrix} = \begin{bmatrix} 4 \\ 7 \end{bmatrix}$$

Because we cannot draw such planes every time, let's take a look at how the vector $\vec{v} = \langle 1, 2 \rangle$ is mapped to $\langle 4, 7 \rangle$. First, from the previous note, we have seen that the first row represents the x coordinate and the second row represents the y coordinate. Now, to find where the point $(1, 2)$ is located in the "scaled" version, we could simply look at the basis vectors in the "scaled" plane. Because the vector that we are looking for is the sum of the two "scaled" basis vector, we can represent our multiplication as the following.

$$\begin{bmatrix} 2 & 1 \\ 1 & 3 \end{bmatrix} \begin{bmatrix} 1 \\ 2 \end{bmatrix} = 1 \begin{bmatrix} 2 \\ 1 \end{bmatrix} + 2 \begin{bmatrix} 1 \\ 3 \end{bmatrix} = \begin{bmatrix} 2 \\ 1 \end{bmatrix} + \begin{bmatrix} 2 \\ 6 \end{bmatrix} = \begin{bmatrix} 4 \\ 7 \end{bmatrix}$$

Nice! We get the same result. Now that we have intuitions on what a matrix and its multiplication represents, let's take a look at algebraic properties and resulting formulas from the intuition.

5.3.1 The Method and Properties

As you can see above, matrix multiplication is not defined elementwise unlike addition and scalar multiplication. First, take a look at the following formula for individual elements.

Theorem 5.3.1

For matrices $[a_{ij}]_{m\times n}$ and $[b_{ij}]_{n\times p}$, the elements c_{ij} of the product matrix $[c_{ij}]_{m\times p}$ are defined as the following expression.

$$c_{ij} = \sum_{k=1}^{n} a_{ik}b_{kj}$$

Let's take a look at an example to see how this works.

Exercise 5.3.2

Compute the following matrix multiplication.

$$\begin{bmatrix} 2 & 3 & 4 \\ 1 & 5 & -1 \end{bmatrix} \begin{bmatrix} 1 & 2 \\ 2 & 3 \\ 4 & 0 \end{bmatrix}$$

Solution.

Consider the following computation.

$$\begin{bmatrix} 2 & 3 & 4 \\ 1 & 5 & -1 \end{bmatrix} \begin{bmatrix} 1 & 2 \\ 2 & 3 \\ 4 & 0 \end{bmatrix} = \begin{bmatrix} 2\cdot 1+3\cdot 2+4\cdot 4 & 2\cdot 2+3\cdot 3+4\cdot 0 \\ 1\cdot 1+5\cdot 2-1\cdot 4 & 1\cdot 2+5\cdot 3-1\cdot 0 \end{bmatrix}$$

$$= \begin{bmatrix} 2+6+16 & 4+9+0 \\ 1+10-4 & 2+15 \end{bmatrix} = \begin{bmatrix} 24 & 13 \\ 7 & 17 \end{bmatrix}$$

It is important to note that matrix multiplication applies under a specific condition.

Theorem 5.3.3

For matrices $A_{m\times n}$ and $B_{p\times q}$, the matrices can be multiplied if and only if $n = p$. The resulting matrix will have order $m \times q$.

I believe the condition above is somewhat self-evident. When the condition above is not met, we say the product is **undefined**. Continuing from this condition, we could get an interesting property.

Theorem 5.3.4

Matrix multiplication is not commutative, i.e. for matrices A and B, it is not necessarily true that $AB = BA$.

Proof.

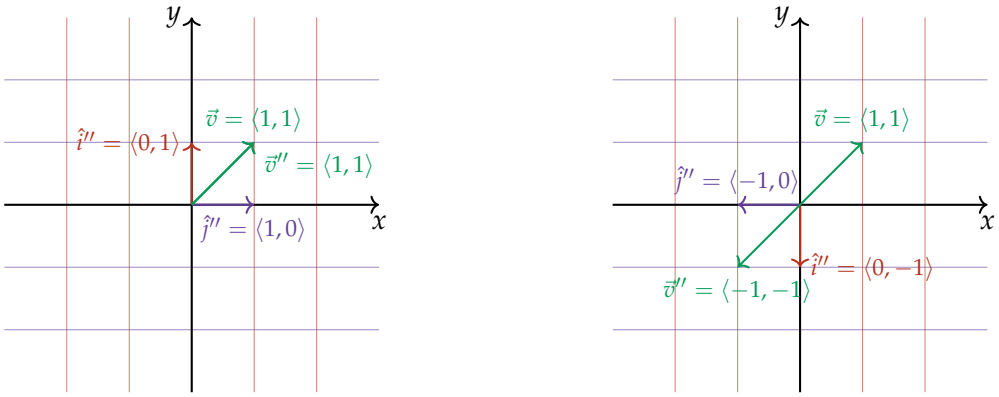
I will leave a more formal proof for the readers. However, I believe this is self-evident as for matrices $A_{m\times n}$ and $B_{n\times p}$ where $m \neq p$, AB is defined while BA is not.

A fun way of understanding commutativity is to think about transformation questions that we see in elementary or middle school competition math. You might be familiar with questions asking orders of transformation of an object. For instance, let A be defined as the 90° rotation counterclockwise and let transformation B be reflection across the y axis. Is performing AB equal to BA? Obviously, the answer is false when you visually think about it. What if we apply the same idea here with matrices?

Notice the matrices A and B can be represented as following using matrices.

$$A = \begin{bmatrix} 0 & -1 \\ 1 & 0 \end{bmatrix}, \quad B = \begin{bmatrix} -1 & 0 \\ 0 & 1 \end{bmatrix}$$

Taking a look at the visualization, the left diagram represents AB and the right diagram visualizes BA with $\vec{v} = \langle 1, 1 \rangle$ as the reference.



Let's take a look at what's happening here. $\vec{v}''$ in our left diagram can be represented as the following.

$$\vec{v}'' = \begin{bmatrix} -1 & 0 \\ 0 & 1 \end{bmatrix} \left(\begin{bmatrix} 0 & -1 \\ 1 & 0 \end{bmatrix} \begin{bmatrix} 1 \\ 1 \end{bmatrix} \right)$$

Notice that we have to write it in the order $B(A\vec{v})$ as we would be performing A first. We can think of it as reading it from left to right just as we would read from left to right when computing composite functions.

$$\vec{v}'' = \begin{bmatrix} -1 & 0 \\ 0 & 1 \end{bmatrix} \left(\begin{bmatrix} 0 & -1 \\ 1 & 0 \end{bmatrix} \begin{bmatrix} 1 \\ 1 \end{bmatrix} \right) = \begin{bmatrix} -1 & 0 \\ 0 & 1 \end{bmatrix} \left(\begin{bmatrix} -1 \\ 1 \end{bmatrix} \right) = \begin{bmatrix} 1 \\ 1 \end{bmatrix}$$

This is exactly what we have for the left diagram. We could do the same for the right side.

$$\vec{v}'' = \begin{bmatrix} 0 & -1 \\ 1 & 0 \end{bmatrix} \left(\begin{bmatrix} -1 & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} 1 \\ 1 \end{bmatrix} \right) = \begin{bmatrix} 0 & -1 \\ 1 & 0 \end{bmatrix} \left(\begin{bmatrix} -1 \\ 1 \end{bmatrix} \right) = \begin{bmatrix} -1 \\ -1 \end{bmatrix}$$

This is also supported! Alternatively, we could just compute AB and BA to see what our "final

transformation is".

$$BA = \begin{bmatrix} -1 & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} 0 & -1 \\ 1 & 0 \end{bmatrix} = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}$$
$$AB = \begin{bmatrix} 0 & -1 \\ 1 & 0 \end{bmatrix} \begin{bmatrix} -1 & 0 \\ 0 & 1 \end{bmatrix} = \begin{bmatrix} 0 & -1 \\ -1 & 0 \end{bmatrix}$$

Clearly, we could see that the mapping is different.

Another interesting property for matrices A, B, and C is that even if $AC = BC$, it does not necessarily mean that $A = B$. Think about the case when C is a zero matrix. Although matrix multiplication is not commutative, the associative and distributive property still applies.

Theorem 5.3.5

For matrices A, B, and C, the following equations are true given that the conditions for the addition and multiplication are met.

1. $A(BC) = (AB)C$
2. $A(B + C) = AB + AC$
3. $(B + C)A = BA + CA$
4. $\lambda(AB) = A(\lambda B)$

Let's start by proving the associative property.

Proof.

Let $A = [a_{ij}]_{m\times n}$, $B = [b_{ij}]_{n\times q}$, and $C = [c_{ij}]_{q\times r}$. First, notice that the elements of the product BC, or $D = [d_{ij}]_{n\times r}$ can be represented as $d_{ij} = \sum_{k=1}^{q} b_{ik}c_{kj}$. Multiplying it again with $[a_{ij}]$, the elements of the product $E = [e_{ij}]_{m\times r}$ can be represented as the following.

$$e_{ij} = \sum_{s=1}^{n} a_{is}d_{sj} = \sum_{s=1}^{n} \left(a_{is} \sum_{k=1}^{q} b_{sk}c_{kj} \right) = \sum_{s=1}^{n} \sum_{k=1}^{q} a_{is}b_{sk}c_{kj}$$

Similarly, we could do the same for the right-hand side of the equation. Note that the elements of the product AB, or $D' = [d'_{ij}]_{m\times q}$ can be represented as $d'_{ij} = \sum_{k=1}^{n} a_{ik}b_{kj}$. Continuing, the elements in the final product $E' = [e'_{ij}]_{m\times r}$ can be written as the following.

$$e'_{ij} = \sum_{s=1}^{q} d'_{is}c_{sj} = \sum_{s=1}^{q} \left(\left(\sum_{k=1}^{n} a_{ik}b_{ks} \right) c_{sj} \right) = \sum_{s=1}^{q} \sum_{k=1}^{n} a_{ik}b_{ks}c_{sj}$$
$$= \sum_{k=1}^{q} \sum_{s=1}^{n} a_{is}b_{sk}c_{kj} = \sum_{s=1}^{n} \sum_{k=1}^{q} a_{is}b_{sk}c_{kj} = e_{ij}$$

Because $e_{ij} = e'_{ij}$ for all possible combinations, $A(BC) = (AB)C$.

In my head...

You might wonder if we are allowed to say that the following equation is true.

$$\sum_{k=1}^{q}\sum_{s=1}^{n} a_{is}b_{sk}c_{kj} = \sum_{s=1}^{n}\sum_{k=1}^{q} a_{is}b_{sk}c_{kj}$$

When proving this property, I initially thought that my proof contained an error and spent my time reading the proof over and over again. This actually applies and visualizing it helped me. This part is not rigorous, so I didn't include it in the proof above, but you can think of rotating the entire summation by 90°.

$$\sum_{k=1}^{q}\sum_{s=1}^{n} a_{is}b_{sk}c_{kj} = \begin{matrix} a_{i1}b_{11}c_{1j} \\ \vdots \\ a_{in}b_{n1}c_{1j} \end{matrix} + \cdots + \begin{matrix} a_{i1}b_{1q}c_{qj} \\ \vdots \\ a_{in}b_{nq}c_{qj} \end{matrix}$$

Similarly, the second sum can be represented as the following.

$$\sum_{s=1}^{n}\sum_{k=1}^{q} a_{is}b_{sk}c_{kj} = \begin{matrix} a_{i1}b_{11}c_{1j} \\ \vdots \\ a_{i1}b_{1q}c_{qj} \end{matrix} + \cdots + \begin{matrix} a_{in}b_{n1}c_{1j} \\ \vdots \\ a_{in}b_{nq}c_{qj} \end{matrix}$$

Now imagine you write the entire terms out, and rotate it 90°. They are identical!

Next, let's prove the first distributive law.

Proof.

Let $A = [a_{ij}]_{m\times n}$, $B = [b_{ij}]_{n\times q}$, and $C = [c_{ij}]_{n\times q}$. Consider the following equation.

$$\begin{aligned}
[a_{ij}]\left([b_{ij}] + [c_{ij}]\right) = [a_{ij}][b_{ij} + c_{ij}] &= \left[\sum_{k=1}^{n} a_{ik}(b_{kj} + c_{kj})\right] \\
&= \left[\sum_{k=1}^{n} a_{ik}b_{kj} + \sum_{k=1}^{n} a_{ik}c_{kj}\right] = \left[\sum_{k=1}^{n} a_{ik}b_{kj}\right] + \left[\sum_{k=1}^{n} a_{ik}c_{kj}\right] \\
&= [a_{ij}][b_{ij}] + [a_{ij}][c_{ij}]
\end{aligned}$$

Therefore, the distributive property $A(B + C) = AB + AC$ applies.

We could prove the right distributive law in similar fashion.

Proof.

Let $A = [a_{ij}]_{n\times q}$, $B = [b_{ij}]_{m\times n}$, and $C = [c_{ij}]_{m\times n}$. Consider the following equation.

$$\begin{aligned}\left([b_{ij}] + [c_{ij}]\right)[a_{ij}] = [b_{ij} + c_{ij}][a_{ij}] &= \left[\sum_{k=1}^{n}(b_{ik} + c_{ik})a_{kj}\right] \\ &= \left[\sum_{k=1}^{n} b_{ik}a_{kj} + \sum_{k=1}^{n} c_{ik}a_{kj}\right] = \left[\sum_{k=1}^{n} b_{ik}a_{kj}\right] + \left[\sum_{k=1}^{n} c_{ik}a_{kj}\right] \\ &= [b_{ij}][a_{ij}] + [c_{ij}][a_{ij}]\end{aligned}$$

Thus, the right distributive property $(B + C)A = BA + CA$ holds.

Finally, let's prove the fourth property.

Proof.

Let $A = [a_{ij}]_{m\times n}$ and $B = [b_{ij}]_{n\times p}$. Consider the following equation.

$$\lambda(AB) = \lambda\left[\sum_{k=1}^{n} a_{ik}b_{kj}\right] = \left[\sum_{k=1}^{n} \lambda a_{ik}b_{kj}\right] = \left[\sum_{k=1}^{n} a_{ik}(\lambda b_{kj})\right] = A(\lambda B)$$

Thus, the fourth property holds.

In addition to the visual implications of a matrix and its multiplication above, matrix multiplication can be extended to systems of equations. For instance, consider the following system of equations.

$$\begin{aligned}2x + 3y &= 5 \\ -x + 7y &= 3\end{aligned}$$

Using **coefficient matrix**, we could represent the system above as the following matrix multiplication.

$$\begin{bmatrix}2 & 3 \\ -1 & 7\end{bmatrix}\begin{bmatrix}x \\ y\end{bmatrix} = \begin{bmatrix}5 \\ 3\end{bmatrix}$$

Let's take a look at more interesting facts on matrix multiplication.

Definition 5.3.6

A square matrix with 1 in all entries of its main diagonal and 0 in the other entries is an **identity matrix**, denoted as I_n for an identity matrix with order $n \times n$. Moreover, all identity matrices satisfy the following equation for square matrix A that can be multiplied.

$$AI = IA = A$$

In other word, identity matrices have the following form.

$$\begin{bmatrix} 1 & 0 & \cdots & 0 & 0 \\ 0 & 1 & \cdots & 0 & 0 \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & \cdots & 1 & 0 \\ 0 & 0 & \cdots & 0 & 1 \end{bmatrix}$$

This is called an identity matrix because when you multiply any square matrix by the identity matrix with the corresponding order, you get the square matrix itself. Consider the following example.

$$\begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{bmatrix} = \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{bmatrix}$$

From this, we can obtain an important fact about the uniqueness of the identity matrix.

Theorem 5.3.7

The identity matrix I for square matrices A is unique.

Proof.

For the sake of contradiction, let matrix $A_{m\times m}$ have two unique identity matrices $I_{m\times m}$ and $J_{m\times m}$. By definition, $AI = IA = A$ and $AJ = JA = A$. Therefore, $AI = IA = AJ = JA$ and $AI = AJ$. Because A is any square matrix with order $m \times m$, let $A = I$. Substituting, $II = IJ$, and $I = J$ is obtained by definition. By contradiction from the assumption that $I \neq J$, the identity matrix for a square matrix A is unique.

Another interesting insight on matrix multiplication is dissecting the matrices being multiplied. Let's take a look at a simple example.

$$\begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \end{bmatrix} \begin{bmatrix} b_{11} & b_{12} \\ b_{21} & b_{22} \\ b_{31} & b_{32} \end{bmatrix} = \begin{bmatrix} a_{11}b_{11} + a_{12}b_{21} + a_{13}b_{31} & a_{11}b_{12} + a_{12}b_{22} + a_{13}b_{32} \\ a_{21}b_{11} + a_{22}b_{21} + a_{23}b_{31} & a_{21}b_{12} + a_{22}b_{22} + a_{23}b_{32} \end{bmatrix}$$

$$= \begin{bmatrix} a_{11} & a_{12} & a_{13} \\ 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} b_{11} & b_{12} \\ b_{21} & b_{22} \\ b_{31} & b_{32} \end{bmatrix} + \begin{bmatrix} 0 & 0 & 0 \\ a_{21} & a_{22} & a_{23} \end{bmatrix} \begin{bmatrix} b_{11} & b_{12} \\ b_{21} & b_{22} \\ b_{31} & b_{32} \end{bmatrix}$$

$$= \begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \end{bmatrix} \begin{bmatrix} b_{11} & 0 \\ b_{21} & 0 \\ b_{31} & 0 \end{bmatrix} + \begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \end{bmatrix} \begin{bmatrix} 0 & b_{12} \\ 0 & b_{22} \\ 0 & b_{32} \end{bmatrix}$$

In the matrix multiplication above, we could see that to find the element in the entry (i, j), it is sufficient for us to see row i from the left matrix and column j from the right. In the next section of the note, we will discuss different topics about matrix.

§5.4 Additional Fundamental Topics

This section discusses frequently appearing concepts in matrices that they have their own names!

5.4.1 Transpose

Let's begin with transpose of a matrix.

Definition 5.4.1

The **transpose** of a matrix A is defined by a matrix that has switched rows and columns of A, i.e. each element is defined as following.

$$(A^\intercal)_{ij} = A_{ji}$$

Let's take a look at an example. You could think of it as holding the left top and bottom right of a matrix and rotating to see its back. For matrix A defined as the following,

$$A = \begin{bmatrix} 1 & 2 & 3 & 4 \\ 5 & 6 & 7 & 8 \end{bmatrix}$$

its transpose, or A transpose is defined as the following.

$$A^\intercal = \begin{bmatrix} 1 & 5 \\ 2 & 6 \\ 3 & 7 \\ 4 & 8 \end{bmatrix}$$

As you can see, if A has the order $m \times n$, its transpose will have the order $n \times m$. Another observation that we can make is that a transpose of a row matrix is a column matrix and vice versa. Here is a quick example. Assume matrix A defined as the following.

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

The transpose of A can be represented as the following.

$$A^\intercal = \begin{bmatrix} 1 \\ 2 \\ 3 \end{bmatrix}$$

Notice that the transpose of transpose of A is simply A.

$$(A^\intercal)^\intercal = \begin{bmatrix} 1 & 2 & 3 \end{bmatrix} = A$$

Before formally stating our observations, here are few definitions to note.

Definition 5.4.2

A positive integral powers of A, or multiplying the matrix n times can be represented as A^n. A^0 is defined to be the identity matrix I_n with corresponding order n. A square matrix is **nilpotent** if there exists $n \in \mathbb{Z}$ such that $A^n = 0$.

Other definitions from transpose is symmetry and skew-symmetry.

Definition 5.4.3

For matrix A and its transpose $A^\intercal$, matrix A is **symmetric** if $A^\intercal = A$ and **skew-symmetric** if $A^\intercal = -A$.

An example of a symmetric matrix is the identity matrix. For more general example, matrix A and B below are symmetric and skew-symmetric respectively.

$$A = \begin{bmatrix} a & b & c \\ b & d & e \\ c & e & f \end{bmatrix}, \quad B = \begin{bmatrix} 0 & a & b \\ -a & 0 & c \\ -b & -c & 0 \end{bmatrix}$$

Notice that all symmetric and skew-symmetric matrices are square matrices. Moreover, the diagonal elements must be zero for a square matrix to be skew-symmetric.

Now let's discuss the properties of transpose of a matrix.

Theorem 5.4.4

For matrix A, B, scalar λ and positive integer n, the following equations are true assuming that the addition and multiplication conditions are satisfied.

1. $(A^\intercal)^\intercal = A$
2. $(A + B)^\intercal = A^\intercal + B^\intercal$
3. $(\lambda A)^\intercal = \lambda A^\intercal$
4. $(AB)^\intercal = B^\intercal A^\intercal$
5. $(A^n)^\intercal = (A^\intercal)^n$

There are lots of properties to prove, but let's do them one by one. Below is the proof for the first property.

Proof.

Let $A = [a_{ij}]_{m\times n}$. By definition, its transpose $A^\intercal := [a_{ji}]_{n\times m}$. Continuing, $\left([a_{ji}]_{n\times m}\right)^\intercal = [a_{ij}]_{m\times n} = A$. Therefore, $(A^\intercal)^\intercal = A$.

Below is the proof for the second property.

Proof.

Let $A = [a_{ij}]_{m\times n}$ and $B = [b_{ij}]_{m\times n}$. Consider the following equation.

$$\begin{aligned}(A+B)^\intercal &= \left([a_{ij}]_{m\times n} + [b_{ij}]_{m\times n}\right)^\intercal = \left([a_{ij}+b_{ij}]_{m\times n}\right)^\intercal \\ &= [a_{ji}+b_{ji}]_{n\times m} = [a_{ji}]_{n\times m} + [b_{ji}]_{n\times m} \\ &= A^\intercal + B^\intercal\end{aligned}$$

Thus, the property holds.

Let's continue our proof for $(\lambda A)^\intercal = \lambda A^\intercal$.

Proof.

Let $A = [a_{ij}]_{m\times n}$. Because $(\lambda A)^\intercal = [\lambda a_{ij}]^\intercal_{m\times n} = [\lambda a_{ji}]_{n\times m} = \lambda[a_{ji}]_{n\times m} = \lambda A^\intercal$, the property holds.

We have two more left! Here is the proof for the fourth property.

Proof.

Let $A = [a_{ij}]_{m\times n}$ and $B = [b_{ij}]_{n\times p}$. The following equation is satisfied by definitions of matrix multiplication and transpose.

$$(AB)^\intercal = \left([a_{ij}]_{m\times n}[b_{ij}]_{n\times p}\right)^\intercal = \left(\left[\sum_{k=1}^{n} a_{ik}b_{kj}\right]_{m\times p}\right)^\intercal = \left[\sum_{k=1}^{n} a_{jk}b_{ki}\right]_{p\times m}$$

Continuing from the right hand side, the following equation is obtained.

$$B^\intercal A^\intercal = \left[b^\intercal_{ij}\right]_{p\times n}\left[a^\intercal_{ij}\right]_{n\times m} = \left[\sum_{k=1}^{n} b^\intercal_{ik}a^\intercal_{kj}\right]_{p\times m} = \left[\sum_{k=1}^{n} b_{ki}a_{jk}\right]_{p\times m} = \left[\sum_{k=1}^{n} a_{jk}b_{ki}\right]_{p\times m}$$

Because both the left-hand side and the right-hand side of the equation lead to the same matrix, the property holds.

Finally, below is the proof for the corollary of the fourth property.

Proof.

First, notice that by the fourth property, the following equation holds.

$$\left(A^2\right)^\intercal = A^\intercal A^\intercal = \left(A^\intercal\right)^2$$

Therefore, it suffices to show that $(A^n)^\intercal = (A^\intercal)^n$ holds for $n = k+1$ if it is true for $n = k$. Assume $(A^k)^\intercal = (A^\intercal)^k$. By the fourth property, the following equation is obtained.

$$\left(A^{k+1}\right)^\intercal = (A^kA)^\intercal = A^\intercal(A^k)^\intercal = A^\intercal(A^\intercal)^k = (A^\intercal)^{k+1}$$

By induction, the property holds for $k \geq 2$. For $k = 1$, it is self evident that the property

holds true. Therefore, the property holds for all positive integers n.

For the next part of the section, we will discuss partitions.

5.4.2 Partitions

As always, let's start with key definitions.

Definition 5.4.5

As the name suggests a **submatrix** is a matrix of A that is obtained by omitting certain rows or columns.

Consider matrix A defined as the following.

$$A = \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{bmatrix}$$

Below are a few example of submatrices of A.

$$A = \begin{bmatrix} 1 & 2 & 3 \\ \cancel{4} & \cancel{5} & \cancel{6} \\ 7 & 8 & 9 \end{bmatrix} \rightarrow \begin{bmatrix} 1 & 2 & 3 \\ 7 & 8 & 9 \end{bmatrix}$$

$$A = \begin{bmatrix} 1 & \cancel{2} & 3 \\ 4 & 5 & 6 \\ \cancel{7} & \cancel{8} & \cancel{9} \end{bmatrix} \rightarrow \begin{bmatrix} 4 & 6 \end{bmatrix}$$

Definition 5.4.6

We say the matrix is **partitioned** if horizontal and vertical lines are used to divide the matrix into submatrices.

Here are a few examples of different partitions of A.

$$\left[\begin{array}{ccc} 1 & 2 & 3 \\ 4 & 5 & 6 \\ \hline 7 & 8 & 9 \end{array}\right], \quad \left[\begin{array}{c|cc} 1 & 2 & 3 \\ 4 & 5 & 6 \\ \hline 7 & 8 & 9 \end{array}\right], \quad \left[\begin{array}{c|c|c} 1 & 2 & 3 \\ \hline 4 & 5 & 6 \\ \hline 7 & 8 & 9 \end{array}\right]$$

From different partitions, we can obtain different **blocks**.

Definition 5.4.7

A **block** of a partition of A is a submatrix of A divided by the horizontal and vertical partition lines.

The highlighted portion of the second partition above is an example of a block. Now, why

should we even care about partitions and blocks? Partitions of a matrix are important because it could be a very useful tool in computationally heavy matrix multiplication. Let's take a look at a generalized example AB.

$$AB = \begin{bmatrix} a_{11} & a_{12} & a_{13} & a_{14} \\ a_{21} & a_{22} & a_{23} & a_{24} \\ a_{31} & a_{32} & a_{33} & a_{34} \\ a_{41} & a_{42} & a_{43} & a_{44} \end{bmatrix} \begin{bmatrix} b_{11} & b_{12} & b_{13} \\ b_{21} & b_{22} & b_{23} \\ b_{31} & b_{32} & b_{33} \\ b_{41} & b_{42} & b_{43} \end{bmatrix}$$

At least for me, this multiplication looks very painful. Fortunately, using partitions, we can turn this monster into a baby monster.

$$\begin{aligned} AB &= \left[\begin{array}{cc|cc} a_{11} & a_{12} & a_{13} & a_{14} \\ a_{21} & a_{22} & a_{23} & a_{24} \\ a_{31} & a_{32} & a_{33} & a_{34} \\ a_{41} & a_{42} & a_{43} & a_{44} \end{array}\right] \left[\begin{array}{ccc} b_{11} & b_{12} & b_{13} \\ b_{21} & b_{22} & b_{23} \\ \hline b_{31} & b_{32} & b_{33} \\ b_{41} & b_{42} & b_{43} \end{array}\right] \\ &= \begin{bmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \\ a_{31} & a_{32} \\ a_{41} & a_{42} \end{bmatrix} \begin{bmatrix} b_{11} & b_{12} & b_{13} \\ b_{21} & b_{22} & b_{23} \end{bmatrix} + \begin{bmatrix} a_{13} & a_{14} \\ a_{23} & a_{24} \\ a_{33} & a_{34} \\ a_{43} & a_{44} \end{bmatrix} \begin{bmatrix} b_{31} & b_{32} & b_{33} \\ b_{41} & b_{42} & b_{43} \end{bmatrix} \\ &= \begin{bmatrix} a_{11}b_{11} + a_{12}b_{21} & a_{11}b_{12} + a_{12}b_{22} & a_{11}b_{13} + a_{12}b_{23} \\ a_{21}b_{11} + a_{22}b_{21} & a_{21}b_{12} + a_{22}b_{22} & a_{21}b_{13} + a_{22}b_{23} \\ a_{31}b_{11} + a_{32}b_{21} & a_{31}b_{12} + a_{32}b_{22} & a_{31}b_{13} + a_{32}b_{23} \\ a_{41}b_{11} + a_{42}b_{21} & a_{41}b_{12} + a_{42}b_{22} & a_{41}b_{13} + a_{42}b_{23} \end{bmatrix} \\ &\qquad + \begin{bmatrix} a_{13}b_{31} + a_{14}b_{41} & a_{13}b_{32} + a_{14}b_{42} & a_{13}b_{33} + a_{14}b_{43} \\ a_{23}b_{31} + a_{24}b_{41} & a_{23}b_{32} + a_{24}b_{42} & a_{23}b_{33} + a_{24}b_{43} \\ a_{33}b_{31} + a_{34}b_{41} & a_{33}b_{32} + a_{34}b_{42} & a_{33}b_{33} + a_{34}b_{43} \\ a_{43}b_{31} + a_{44}b_{41} & a_{43}b_{32} + a_{44}b_{42} & a_{43}b_{33} + a_{44}b_{43} \end{bmatrix} \\ &= \begin{bmatrix} a_{11}b_{11}+a_{12}b_{21}+a_{13}b_{31}+a_{14}b_{41} & a_{11}b_{12}+a_{12}b_{22}+a_{13}b_{32}+a_{14}b_{42} & a_{11}b_{13}+a_{12}b_{23}+a_{13}b_{33}+a_{14}b_{43} \\ a_{21}b_{11}+a_{22}b_{21}+a_{23}b_{31}+a_{24}b_{41} & a_{21}b_{12}+a_{22}b_{22}+a_{23}b_{32}+a_{24}b_{42} & a_{21}b_{13}+a_{22}b_{23}+a_{23}b_{33}+a_{24}b_{43} \\ a_{31}b_{11}+a_{32}b_{21}+a_{33}b_{31}+a_{34}b_{41} & a_{31}b_{12}+a_{32}b_{22}+a_{33}b_{32}+a_{34}b_{42} & a_{31}b_{13}+a_{32}b_{23}+a_{33}b_{33}+a_{34}b_{43} \\ a_{41}b_{11}+a_{42}b_{21}+a_{43}b_{31}+a_{44}b_{41} & a_{41}b_{12}+a_{42}b_{22}+a_{43}b_{32}+a_{44}b_{42} & a_{41}b_{13}+a_{42}b_{23}+a_{43}b_{33}+a_{44}b_{43} \end{bmatrix} \end{aligned}$$

Well, this looks good to me! If the matrices being multiplied have higher dimension, partitioning such matrices could help in computation and reducing mistakes. Now that we have seen partitions of matrices, we could derive special forms from the ideas.

5.4.3 Special Forms of Matrix

As always, let's build from definitions.

Definition 5.4.8

A **symmetrically partitioned matrix** refers to a partitioned matrix such that its arrangements of the blocks are diagonally symmetric. The **diagonal blocks** are the blocks that pass through the diagonal symmetry.

We could also define matrices with diagonal blocks that are not necessarily symmetrically partitioned.

Definition 5.4.9

A **block diagonal matrix** is a matrix with square diagonal blocks and zero on the other entries.

General form for square matrices A_i can be represented as $A = \begin{bmatrix} A_1 & 0 & \cdots & 0 \\ 0 & A_2 & \cdots & 0 \\ \vdots & \vdots & \ddots & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix}$. Continuing, we could define rows and columns based on its elements.

Definition 5.4.10

Zero rows and **zero columns** are rows and columns composed of zeros respectively.

This definition is quite straightforward, and now that we have basic terms in mind, let's discuss two forms from the definitions above.

Definition 5.4.11

Matrix in **row echelon form (REF)** satisfies the conditions below.

- All nonzero leading entry is 1. (Side note here is that some textbooks and lectures do not require the leading entry to be 1, but I actually would like to keep it this way as this is how I initially learned it.)
- All nonzero leading entries in a nonzero row in a later row are to the right of nonzero leading entries in previous rows.
- If there are both nonzero and zero rows, all zero rows must appear after the nonzero rows above.

The leading entry of matrices in REF is known as a **pivot**. Similarly, matrices in **reduced row echelon form (RREF)** are matrices that satisfy the following conditions.

- All nonzero leading entries in a nonzero row is 1.
- If there are both nonzero and zero rows, all zero rows must appear after the nonzero rows above.
- All leading 1s in the succeeding rows appear in a column after the columns with previous leading 1s.
- The leading 1 must be the only nonzero element present in each column if there exists such 1.

Below are examples of a matrices in REF and RREF respectively.

$$\begin{bmatrix} 1 & 2 & 3 & 4 \\ 0 & 0 & 1 & 2 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \end{bmatrix}, \quad \begin{bmatrix} 1 & 0 & 2 & 0 \\ 0 & 1 & 2 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \end{bmatrix}$$

Matrices that follow these forms will be very useful when solving linear systems of equations, but we will save that for the latter notes. Now let's discuss about triangular matrices.

Triangular matrices are a special type of square matrices that is determined by elements below the main diagonal. If all its elements below the main diagonal are zeros, we call it an **upper triangular** matrix. Similarly, if all elements above the main diagonal are zeros, we call them **lower triangular** matrices. For more formal statement, consider the following definition.

Definition 5.4.12

A square matrix $A = [a_{ij}]$ is **upper triangular** if $a_{ij} = 0 \,\forall\, i > j$. Similarly, the matrix is **lower triangular** if $a_{ij} = 0 \,\forall\, i < j$.

Let's take a look at an example. The left and right matrices below are upper triangular and lower triangular matrices respectively.

$$\begin{bmatrix} 1 & 2 & 3 & 4 \\ 0 & 2 & 3 & 4 \\ 0 & 0 & 3 & 4 \\ 0 & 0 & 0 & 4 \end{bmatrix}, \quad \begin{bmatrix} 1 & 0 & 0 & 0 \\ 2 & 1 & 0 & 0 \\ 3 & 2 & 1 & 0 \\ 0 & 3 & 2 & 0 \end{bmatrix}$$

Now that we have definitions of triangular matrices in mind, we can discuss a theorem from such matrices.

Theorem 5.4.13

The product of two lower triangular matrices is a lower triangular matrix, and the product of two upper triangular matrices is an upper triangular matrix.

Proof.

Let $A = [a_{ij}]_{n\times n}$ and $B = [b_{ij}]_{n\times n}$ be lower triangular matrices. Consider the following equation for their product matrix $C = [c_{ij}]_{n\times n}$.

$$C = \left[\sum_{k=1}^{n} a_{ik}b_{kj}\right] = \left[\sum_{k=1}^{j-1} a_{ik}b_{kj} + \sum_{k=j}^{n} a_{ik}b_{kj}\right]$$

From the expression, it suffices to show that $\sum_{k=1}^{j-1} a_{ik}b_{kj} + \sum_{k=j}^{n} a_{ik}b_{kj} = 0$ for $i < j$. Notice that by definition, $b_{kj} = 0$ for all $k = 1$ to $k = j - 1$. Similarly, $a_{ik} = 0$ for all integer $k \in [j, n]$.

Therefore, $c_{ij} = 0$ for all $i < j$, proving the first half of the theorem.

Similarly, let $A' = [a'_{ij}]_{n\times n}$ and $B' = [b'_{ij}]_{n\times n}$ be upper triangular matrices. The product matrix $C' = [c'_{ij}]_{n\times n}$ can be represented as following.

$$C' = \left[\sum_{k=1}^{n} a'_{ik}b'_{kj}\right] = \left[\sum_{k=1}^{j-1} a'_{ik}b'_{kj} + \sum_{k=j}^{n} a'_{ik}b'_{kj}\right]$$

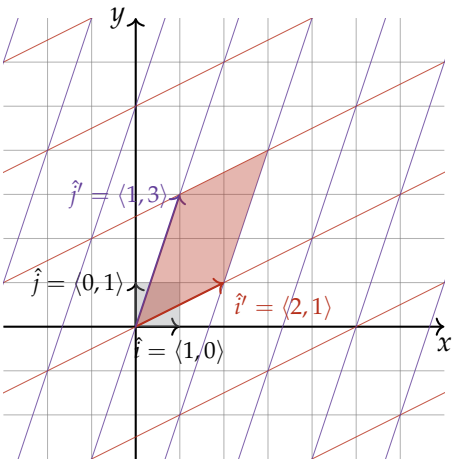
By definition, it is evident that $a'_{ik} = 0$ for integer $k \in [1, j-1]$ and $b'_{kj} = 0$ for integer $k \in [j, n]$. Therefore, for $i > j$, $c'_{ij} = 0$ and the theorem is true.

This is it for different topics on fundamentals of matrices! I know this note is quite long, but let's conclude our notes with the discussion on determinants.

§5.5 Determinants

Finally determinants! One of the most foundational concepts in linear algebra is the determinant. Being one of the foundational concepts, I also think it is a hard concept if the intuition is excluded in the discussion due to its formula heavy nature.

As you could have noticed, a linear transformation does not only change the position of the vectors. Consider the following diagram.



Notice that from the diagram above, the gray area is scaled to the new red area. Notice that the area of the new parallelogram is $3 \cdot 4 - 1 \cdot 2 \cdot \frac{1}{2} \cdot 2 - 1 \cdot 3 \cdot \frac{1}{2} \cdot 2 - 1 \cdot 2 = 12 - 2 - 3 - 2 = 5$. Because the area which was originally 1 is scaled to 5, we can say that

$$\begin{vmatrix} 2 & 1 \\ 1 & 3 \end{vmatrix} = 5$$

holds. The determinant is only defined for square matrices, and for each square matrix, the determinants tell how much the area has changed after the linear transformation. Determinants

can surely be negative, and they represent the change in orientation. With this intuition in mind, let's discuss more computational perspective of determinants.

Before we discuss the formula for determinants, let's define a notation.

Definition 5.5.1

The **minor** of matrix A, denoted as M_{ij}, is a submatrix of A without the i^{th} row and j^{th} column of A. The **cofactor** of a_{ij} is defined as $(-1)^{i+j}\det(M_{ij})$.

Consider the following examples for A defined as the following.

$$A = \begin{bmatrix} a_{11} & a_{12} & a_{13} & a_{14} \\ a_{21} & a_{22} & a_{23} & a_{24} \\ a_{31} & a_{32} & a_{33} & a_{34} \\ a_{41} & a_{42} & a_{43} & a_{44} \end{bmatrix}$$

M_{11} and M_{24} are defined as the following.

$$M_{11} = \begin{bmatrix} a_{22} & a_{23} & a_{24} \\ a_{32} & a_{33} & a_{34} \\ a_{42} & a_{43} & a_{44} \end{bmatrix}, \quad M_{24} = \begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{31} & a_{32} & a_{33} \\ a_{41} & a_{42} & a_{43} \end{bmatrix}$$

Now let's define what determinants are with a recursive formula.

Definition 5.5.2

The **determinant** of a square matrix A, denoted as $\det(A)$ or $|A|$, is a value defined with the following rules.

1. $\det(A) = a_{11}$ if $A = [a_{11}]$.
2. The following equation can be used to determine $\det(A)$ for A with order $n \times n$ for $n > 1$.
$$\det(A) = \sum_{j=1}^{n} (-1)^{1+j} a_{1j} \det(M_{1j})$$

Let's take a look at an example with a 2×2 matrix $A = [a_{ij}]$.

$$\begin{aligned} \det(A) &= \sum_{j=1}^{2} (-1)^{1+j} a_{1j} \det(M_{1j}) = (-1)^{1+1} a_{11} \det(M_{11}) + (-1)^{1+2} a_{12} \det(M_{12}) \\ &= (-1)^2 a_{11} a_{22} + (-1)^3 a_{12} a_{21} \\ &= a_{11} a_{22} - a_{12} a_{21} \end{aligned}$$

Returning to our first example, we can verify that the formula holds.

$$\begin{vmatrix} 2 & 1 \\ 1 & 3 \end{vmatrix} = 2 \cdot 3 - 1 \cdot 1 = 5$$

We could also derive the formula for higher orders with many computations but I will leave them for you! I also think it is just better to use the recursive formula and the formula for 2×2 matrices instead of memorizing complex formulas, but I don't know. I think it's up to you. Below is another example.

$$\det\left(\begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}\right) = \sum_{j=1}^{3} (-1)^{1+j} a_{1j} \det(M_{1j}) = a_{11} \det(M_{11}) = 1 \cdot 1 - 0 \cdot 0 = 1$$

With these in mind, we can establish a very important theorem in computing determinants.

Theorem 5.5.3

For any square matrix A, the cofactor expansion along the i^{th} row and the cofactor expansion along the j^{th} column equal. In other words, the following equation holds.

$$\det(A) = \sum_{j=1}^{n} (-1)^{i+j} a_{ij} \det(M_{ij}) = \sum_{i=1}^{n} (-1)^{i+j} a_{ij} \det(M_{ij})$$

Let's save the proof for the theorem for later as we need to show different properties of determinants before proving the theorem above. For now, let's discuss a special property of triangular matrices.

Theorem 5.5.4

For a triangular (upper or lower) matrix $A_{n \times n} = [a_{ij}]$, the determinant is defined as the following.

$$\det(A) = \prod_{i=1}^{n} a_{ii}$$

Proof.

First, the case for upper triangular matrix can be proven. Notice that

$$\det(A) = \sum_{i=1}^{n} (-1)^{i+1} a_{i1} \det(M_{i1}) = a_{11} \det(M_{11})$$

by Theorem 5.5.3. Continuing, notice that M_{11} is another upper triangular matrix by definition.

If such calculations are continued, it is evident that the last term would be a_{nn}. Similarly, the second to the last term would be $a_{nn} a_{n-1,n-1}$. By inductive hypothesis, it is evident that

$$\det(A) = a_{11} \det(M_{11}) = a_{11} \prod_{i=2}^{n} a_{ii} = \prod_{i=1}^{n} a_{ii}$$

and the theorem holds for upper triangular matrix.

The case for lower triangular matrix can be proven in similar manner. Note that $\det(A) = \sum_{j=1}^{n}(-1)^{1+j}a_{1j}\det(M_{1j}) = a_{11}\det(M_{11})$. With the same inductive hypothesis above, it can be concluded that

$$\det(A) = a_{11}\det(M_{11}) = a_{11}\prod_{i=2}^{n}a_{ii} = \prod_{i=1}^{n}a_{ii}$$

and the theorem holds.

As you could have guessed, we get a direct corollary from the theorem.

Corollary 5.5.5

The determinant of any diagonal matrix is the product of the diagonal entries.

Proof.

First and foremost, notice that a diagonal matrix is both an upper and lower triangular matrix. Thus by Theorem 5.5.4, the corollary holds.

Continuing from triangular matrices, we can also apply our understanding of a transpose of a matrix. Consider the following theorem.

Theorem 5.5.6

$\det(A) = \det(A^\intercal)$ holds for all square matrix A.

Proof.

First and foremost, notice that $A = A^\intercal$ if A is a 1×1 matrix. Thus, the theorem holds for any $A_{1\times1}$. Continuing, note that for $A_{n\times n}$ with $n > 1$, M_{ij} equals M'_{ij} for A^T. Moreover by definition, $M'_{ij} = M_{ji}$ and the following equation holds $A^\intercal = [a'_{ij}]$.

$$\det(A) = \sum_{i=1}^{n}(-1)^{i+1}a_{i1}\det(M_{i1}) = \sum_{j=1}^{n}(-1)^{1+j}a'_{1j}\det(M'_{1j}) = \det\left(A^\intercal\right)$$

Thus, the theorem holds.

Now that we discussed the relationships between matrices and determinants, let's conclude our discussion with products and determinants.

5.5.1 Determinant and Products

Before we conclude our discussion on determinant and this note, I wished to introduce one of the most important results of determinants. In this part of the section, we will try to prove that the product of the determinants is the same as the determinant of the product. To prove this result, we would need a lot of lemmas, so let's get started with the first one.

Lemma 5.5.7

For a matrix $A_{n\times n}$ and an elementary matrix $E_{n\times n}$ for interchanging two rows, $\det(E) = -1$ and $\det(EA) = \det(E)\det(A)$.

Proof.

First and foremost, the case when E exchanges two consecutive rows can be proven. Let $B = FA$ where F is the $n \times n$ elementary matrix that interchanges the k^{th} and $k+1^{\text{th}}$ rows of A. Therefore, for $A = [a_{ij}]$ and $B = [b_{ij}]$, $b_{k+1,j} = a_{kj}$ holds and for the minor of B as M'_{ij}, $M'_{i+1,j} = M_{ij}$. Moreover, notice that $F = -1$ for $n = 2$ and by inductive hypothesis with exchanging the use of $\det(A) = (-1)^2 a_{11}\det(M_{11})$ and $\det(A) = (-1)^{2n} a_{nn}\det(M_{nn})$, it is evident that $\det(F) = -1$. Thus, the following equations hold.

$$\begin{aligned}\det(B) &= \sum_{j=1}^{n}(-1)^{k+1+j} b_{k+1,j}\det(M'_{k+1,j}) \\ &= \sum_{j=1}^{n}(-1)^{k+1+j} a_{kj}\det(M_{kj}) = -\sum_{j=1}^{n}(-1)^{k+j} a_{kj}\det(M_{kj}) \\ &= -\det(A)\end{aligned}$$

Thus, $\det(FA) = \det(F)\cdot\det(A)$. Continuing, notice that any exchange between two consecutive rows requires one F. Moreover, any exchange between two rows with a row in between requires three Fs. Since any exchange can be performed with the combination of such two operations, it is evident that every E can be represented as odd number of Fs. Therefore, the following equations hold for odd m.

$$\begin{aligned}\det(EA) &= \det(F_1F_2\cdots F_mA) = \det(F_1)\det(F_2)\cdots\det(F_m)\det(A) \\ &= (-1)^m\det(A) = -\det(A) \\ &= \det(E)\det(A)\end{aligned}$$

Thus, the lemma holds.

Continuing from this lemma, we can establish an interesting corollary.

Corollary 5.5.8

The equation $\det(A) = 0$ holds for all square matrices A with either two identical rows or columns.

Proof.

Let k_1 and k_2 be the identical rows of the matrix A. For an elementary matrix E that interchanges the rows k_1 and k_2, the following equations hold by Lemma 5.5.7.

$$\det(A) = \det(EA) = \det(E)\det(A) = -\det(A)$$

Thus, $\det(A) = 0$. Similarly, if A has two identical columns k_1 and k_2, the same argument applies by Theorem 5.5.6 as $\det(A) = \det(A^\intercal)$.

For our second lemma, we have another elementary row operation.

Lemma 5.5.9

For a matrix $A_{n\times n}$ and an elementary matrix $E_{n\times n}$ that scales k^{th} row of A by some nonzero scalar λ, $\det(E) = \lambda$ and $\det(EA) = \det(E)\det(A)$.

Proof.

First because $E_{n\times n} = k$, it is evident that utilizing $\det(A) = (-1)^2 a_{11}\det(M_{11})$ and $\det(A) = (-1)^{2n} a_{nn}\det(M_{nn})$, appropriately will lead to $\det(E) = \lambda$ with inductive hypothesis. Moreover, consider the following equations.

$$\begin{aligned}\det(EA) &= \sum_{i=1}^{n}(-1)^{1+k}\lambda a_{ik}\det(M_{ik}) = \lambda\sum_{i=1}^{n}(-1)^{1+k}a_{ik}\det(M_{ik})\\ &= \lambda\det(A) = \det(E)\det(A)\end{aligned}$$

Thus, the lemma holds.

Just like our first lemma, we can expand this second lemma with an interesting corollary.

Corollary 5.5.10

For any matrix $A_{n\times n}$ and scalars λ, $\det(\lambda A) = \lambda^n\det(A)$.

Proof.

Let E_k be the elementary row operation that scales the k^{th} row of A by λ. Then, the following equations hold by Lemma 5.5.9.

$$\det(\lambda A) = \det(E_n E_{n-1}\cdots E_1 A) = \det(E_n)\det(E_{n-1})\cdots\det(E_1)\det(A) = \lambda^n\det(A)$$

Thus, the corollary holds.

Below is the third lemma on determinants and elementary matrices.

Lemma 5.5.11

For a matrix $A_{n\times n}$ and an elementary matrix E of the same order that adds λ times row k_1 to row k_2 of A, then $\det(E) = 1$ and $\det(EA) = \det(E)\det(A)$.

Proof.

First and foremost, notice that E is a triangular matrix and the elements in its diagonal are all 1. Thus by Theorem 5.5.4, $\det(E) = 1$.

Consider a new matrix B that is obtained by replacing k_2 with k_1 from A. Then, $b_{k_2 j} = a_{k_1 j}$ and the minor M'_{ij} of B satisfy $M'_{k_2 j} = M_{k_2 j}$. Moreover by Corollary 5.5.8, $\det(B) = 0$. Therefore, the following equations hold.

$$\begin{aligned}\det(EA) &= \sum_{j=1}^{n}(-1)^{k_2+j}(a_{k_2 j} + \lambda a_{k_1 j})\det(M_{k_2 j}) \\ &= \sum_{j=1}^{n}(-1)^{k_2+j}(a_{k_2 j})\det(M_{k_2 j}) + \lambda\sum_{j=1}^{n}(-1)^{k_2+j}(a_{k_1 j})\det(M_{k_2 j}) \\ &= \det(A) + \lambda\det(B) = \det(A)\end{aligned}$$

Thus $\det(EA) = \det(A) = \det(E)\det(A)$.

With the three lemmas above, we can establish an important theorem that links the determinants of a matrix and an elementary matrix.

Theorem 5.5.12

For a matrix $A_{n\times n}$ and any elementary matrix E of the same order, $\det(EA) = \det(E)\det(A)$.

Proof.

By definition, there exists three different elementary matrices that interchanges rows, scales a row, or add a scaled row to the other. By Lemmas 5.5.7, 5.5.9, and 5.5.11, $\det(EA) = \det(E)\det(A)$ for all elementary matrices E.

Now, I know it's lots of statements, but I promise we will get to the main result. Let's take a look at one last important result to get to our desired theorem! This theorem and its result include the topic of invertibility, which has not been introduced yet. Please feel free to read Section 3.3 and return or vice versa as it shouldn't be something too hard.

Theorem 5.5.13

A square matrix A is invertible if and only if $\det(A) \neq 0$.

Proof.

First, notice that A is invertible if and only if there exists a sequence of elementary matrices $E_1, \ldots, E_k$ such that $E_k \cdots E_1 A = U$ holds for some upper triangular matrix A. In other words,

$$\det(E_k)\cdots\det(E_1)\det(A) = \det(U)$$

where $\det(E_i) \neq 0$ and $\det(U) \neq 0$. Moreover, there exists such elementary matrices if and only if $\det(A) \neq 0$. Thus, the theorem holds.

Now finally! Let's prove the theorem!

Theorem 5.5.14

For square matrices A and B with the same order, $\det(AB) = \det(A)\det(B)$.

Proof.

First, the theorem can be shown for the case where A is not invertible. Notice that if A is not invertible, then AB must also be not invertible as if AB is invertible, then there exists a matrix C such that $(AB)C = I$ and $A(BC) = I$, which is contradictory. Thus by Theorem 5.5.13, $\det(AB) = 0 = 0 \cdot \det(B) = \det(A)\det(B)$ and the theorem holds for not invertible A.

Continuing, the case for invertible A can be proven. By definition, if A is invertible, then it can be represented as a product of sequence of elementary matrices $E_1, \ldots, E_k$. Therefore, $\det(AB) = \det(E_1 \cdots E_k B) = \det(E_1 \cdots E_k)\det(B) = \det(A)\det(B)$ by Theorem 5.5.12. Thus, the theorem holds.

With this important theorem in mind, let's wrap up our notes with two corollaries.

Corollary 5.5.15

For an invertible matrix A, $\det(A^{-1}) = \frac{1}{\det(A)}$.

Proof.

Notice that $1 = \det(AA^{-1}) = \det(A)\det(A^{-1})$ by Theorem 5.5.14. Therefore, $\det(A^{-1}) = \frac{1}{\det(A)}$.

Continuing, below is the second corollary.

Corollary 5.5.16

If two matrices A and B are similar, then $\det(A) = \det(B)$.

Proof.

By definition, if A and B are similar, then there exists an invertible matrix P that satisfy $A = P^{-1}BP$. Therefore by Theorem 5.5.14 and Corollary 5.5.15,

$$\det(A) = \det(P^{-1})\det(B)\det(P) = \frac{1}{\det(P)} \cdot \det(B)\det(P) = \det(B)$$

and the corollary holds.

This is it for determinants! I think it is an interesting way to connect the relationship between linear transformations, which we will discuss in future notes and matrix representation of their effects. In the next notes, we will discuss different applications of matrices.

§5.6 Practice Problems

1. Consider matrices A and B defined as the following.

$$A = \begin{bmatrix} 1 & 2 \\ 2 & 0 \end{bmatrix}, \quad B = \begin{bmatrix} -1 & 2 \\ 0 & 3 \end{bmatrix}$$

   Compute the following matrix operations.

   (a) $A - 2B$
   (b) AB
   (c) $(AB - BA)^\intercal$

2. Find $a_2 B$ if AB is defined as the following.

$$AB = \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{bmatrix}$$

3. Find matrix A such that the following equation holds.

$$2A - 3A^\intercal = \begin{bmatrix} -1 & -5 \\ 0 & -4 \end{bmatrix}$$

4. Show that all square matrices can be represented as the sum of symmetric and skew-symmetric matrices.

5. Show that if $AB = BA$ for nilpotent matrices A and B, then the product AB is nilpotent.

6. Show that for square matrices A and B, $AB + (AB)^\intercal$ is symmetric.

7. For square matrices A and B of order $n \times n$, show that the following equation holds [Erd20].

$$\det\left(\begin{bmatrix} A & B \\ B & A \end{bmatrix}\right) = \det(A + B)\det(A - B)$$

8. Show that if a square matrix A is skew-symmetric, then A^3 is also skew-symmetric.

Note 6

System of Linear Equations

One of the most important applications of matrices would be in linear systems of equations. With the ideas on matrices, we can open our third eye when viewing systems of linear equations. This note is dedicated to matrices in linear equations, Gaussian Elimination, and LU decompositions.

§6.1 Basics and Intuitions

As always, let's get started with fundamental definitions, concepts, and intuitions that will be used throughout.

First, I am pretty sure most of you are aware of what a system of equations is. However, because it is always nice to formally state our understanding, here is the definition of a linear system of equations.

Definition 6.1.1

A **system of linear equations** is composed of linear equations, or equations whose highest degree is 1. A system of m equations and n variables adhere to the following forms where a_{ij} and b_i are known quantities.

$$\begin{aligned}
a_{11}x_1 + a_{12}x_2 + a_{13}x_3 + \cdots + a_{1n}x_n &= b_1 \\
a_{21}x_1 + a_{22}x_2 + a_{23}x_3 + \cdots + a_{2n}x_n &= b_2 \\
a_{31}x_1 + a_{32}x_2 + a_{33}x_3 + \cdots + a_{3n}x_n &= b_3 \\
&\vdots \\
a_{m1}x_1 + a_{m2}x_2 + a_{m3}x_3 + \cdots + a_{mn}x_n &= b_m
\end{aligned}$$

Below is an example of a system of linear equations.

$$\begin{aligned} x + 2y + 3z &= 4 \\ -x + 3y + 2z &= 5 \end{aligned}$$

Notice that all equations above are linear, meaning, their degrees are 1. Now, what does it mean to find a solution to such systems?

Definition 6.1.2

A **solution to a system of linear equations** is a set of scalar values for unknown variables such that all linear equations are satisfied when substituted.

For instance, $(2, 1)$ is the solution to the following system of linear equations.

$$\begin{aligned} x - 2y &= 0 \\ x + y &= 3 \end{aligned}$$

To check it, you can simply substitute to see if the equations are satisfied.

Now, if you recall from the previous note where we discussed matrix multiplication, system of linear equations can be written as a matrix multiplication.

Definition 6.1.3

A system of linear equations

$$\begin{aligned} a_{11}x_1 + a_{12}x_2 + a_{13}x_3 + \cdots + a_{1n}x_n &= b_1 \\ a_{21}x_1 + a_{22}x_2 + a_{23}x_3 + \cdots + a_{2n}x_n &= b_2 \\ a_{31}x_1 + a_{32}x_2 + a_{33}x_3 + \cdots + a_{3n}x_n &= b_3 \\ &\vdots \\ a_{m1}x_1 + a_{m2}x_2 + a_{m3}x_3 + \cdots + a_{mn}x_n &= b_m \end{aligned}$$

can be written as $Ax = b$ where A, x, and b are matrices defined as the following.

$$A = \begin{bmatrix} a_{11} & a_{12} & a_{13} & \cdots & a_{1n} \\ a_{21} & a_{22} & a_{23} & \cdots & a_{2n} \\ a_{31} & a_{32} & a_{33} & \cdots & a_{3n} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ a_{m1} & a_{m2} & a_{m3} & \cdots & a_{mn} \end{bmatrix}, \quad x = \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ \vdots \\ x_n \end{bmatrix}, \quad b \begin{bmatrix} b_1 \\ b_2 \\ b_3 \\ \vdots \\ b_m \end{bmatrix}$$

When you actually perform matrix multiplication, you can see that $Ax = b$ is identical to the system of linear equations. Now that we have a basic understanding of the relationship between matrices and linear systems of equations, let's take a look at a theorem and an important corollary.

Theorem 6.1.4

For two distinct solutions x_1 and x_2 of $Ax = b$, $\alpha x_1 + \beta x_2$ is also a solution to the linear system given that $\alpha, \beta \in \mathbb{R}$ and $\alpha + \beta = 1$.

Proof.

From Theorem 5.3.5, it was shown that $\lambda(AB) = A(\lambda B)$ for scalar λ and matrices A and B that can be multiplied. Substituting $\alpha x_1 + \beta x_2$, the following system is obtained.

$$A(\alpha x_1 + \beta x_2) = A(\alpha x_1) + A(\beta x_2) = \alpha(Ax_1) + \beta(Ax_2) = \alpha b + \beta b = b$$

Therefore, $(\alpha x_1 + \beta x_2)$ is another solution to the system.

From this theorem, we can establish an important corollary about the number of solutions to a system of linear equations.

Corollary 6.1.5

A system of linear equations can retain 0, 1, or infinitely many solutions.

Proof.

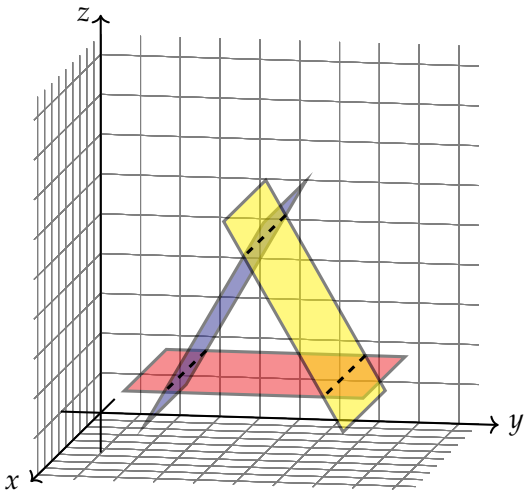
First, it is self-evident that a system can retain 0 or 1 solution. Therefore, the fact that there exist infinite solutions if there are more than 1 solution must be proven.

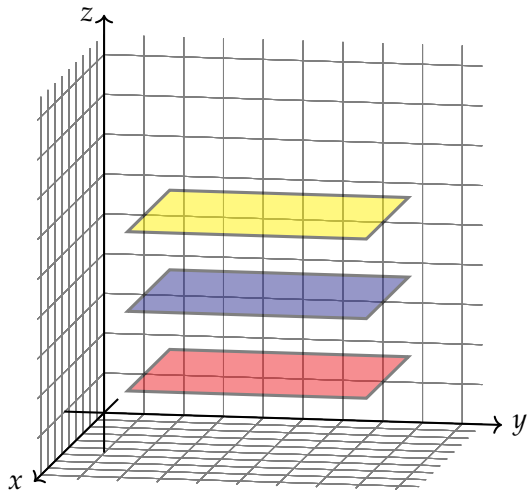
Let $f(\alpha) = \alpha x_1 + (1 - \alpha)x_2$ be a function that returns possible solutions where $x_1 \neq x_2$ are fixed values. It suffices to show that the function f is injective as there exists infinitely many values of α. Consider the following equations.

$$f(\alpha) = \alpha x_1 + (1 - \alpha)x_2 = \alpha(x_1 - x_2) + x_2$$
$$f(\alpha') = \alpha' x_1 + (1 - \alpha')x_2 = \alpha'(x_1 - x_2) + x_2$$

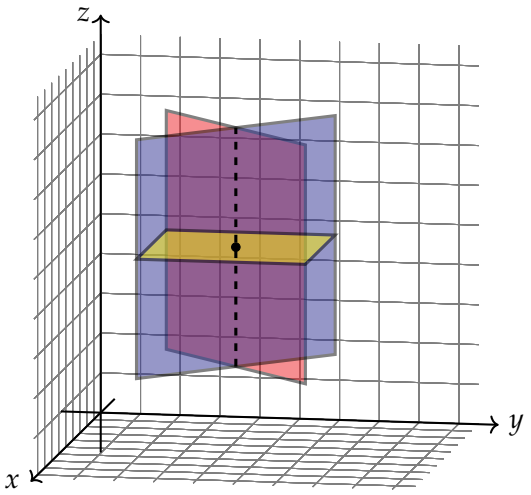
If $f(\alpha) = f(\alpha')$, then $\alpha(x_1 - x_2) + x_2 = \alpha'(x_1 - x_2) + x_2$ and $\alpha = \alpha'$, proving the injectivity. Therefore, a system of linear equations can only have 0, 1, or infinitely many solutions.

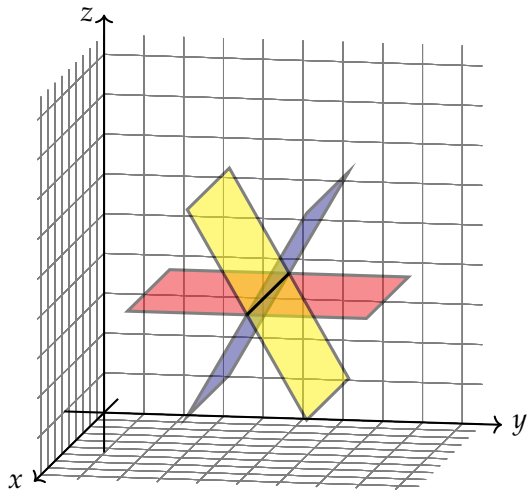
In middle school, you probably saw a visual understanding of the corollary above. Let's do the same, except in 3-dimensional space. Below are two examples of when there exists no solution. Note that not all three planes intersect at a single point.





Continuing, the left diagram represents the case where there is a single solution while the right diagram shows the system with infinite solutions.





Wow, not going to lie, it took some time to draw the diagrams above. Anyhow, with that in mind, let's discuss other fundamental terms and concepts before we move on to the next section of the note.

Definition 6.1.6

A linear system is **consistent** if it retains at least one solution, whereas a system is **inconsistent** if there is no solution.

We could also define a system based on the column matrix b.

Definition 6.1.7

A linear system is **homogeneous** if the column matrix b in the linear system $Ax = b$ is a zero matrix. If the matrix is not a zero matrix, the system is **nonhomogeneous**, or **inhomogeneous**.

Now from the definitions, we can see that all homogeneous systems are consistent. As you have

guessed, we have a special name for a special solution in such cases.

Definition 6.1.8

Trivial solution is a solution where all the entries for x in $Ax = b$ are zero.

With the fundamental terms in mind, let's take a look at how matrices can be used to solve complex linear system with Gaussian Elimination.

§6.2 Gaussian Elimination

When you solve a linear system, you probably multiply constants and add or subtract equations to cancel variables. That is exactly what we will be doing in Gaussian Elimination, but with matrices. First, define how we solve a linear system in a more rigorous way.

Definition 6.2.1

For a system of linear equations, performing the following operations will result in a linear system with an identical solution.

1. Change the order of linear equations in which they are listed
2. Multiply a nonzero constant, or nonzero scalars, to an equation
3. Add equations from the system to form a new equation

Now before we connect this idea with matrices, let's define a matrix that we will use with the property of a linear system above.

Definition 6.2.2

An **augmented matrix** for a linear system $Ax = b$ is a partitioned matrix in the form $[A \mid b]$.

Consider the following system for an example.

$$\begin{aligned} x + 2y + 3z &= 4 \\ 2x + 3y + 4z &= 5 \\ 3x + 4y + 5z &= 6 \end{aligned}$$

In the linear system above, the augmented matrix will be the following matrix.

$$\left[\begin{array}{ccc|c} 1 & 2 & 3 & 4 \\ 2 & 3 & 4 & 5 \\ 3 & 4 & 5 & 6 \end{array}\right]$$

Now that we have an augmented matrix in mind, let's take a look at how the three properties that we discussed apply to matrices.

Definition 6.2.3

For an augmented matrix for a linear system, performing the following operations will result in a linear system with an identical solution as the initial system.

R_1. Rearrangement of the rows in the augmented matrix

R_2. Multiply a nonzero scalar to a row in the augmented matrix

R_3. Add a row multiplied by a scalar with another to replace a row with new row

Such operations are referred to as **elementary row operations**.

If you compare the operations one by one, you can notice that the two sets of properties refer to the same property. Now with the set of properties of augmented matrices in mind, here are the steps for the Gaussian Elimination algorithm.

Theorem 6.2.4

Execute the following steps in order for Gaussian Elimination.

1. Construct the augmented matrix.
2. Utilize the elementary row operations to transform the augmented matrix into REF.
3. Convert the augmented matrix in REF into equations of a transformed linear system, referred to as the **derived set**.
4. Back substitute to solve the system.

Let's take a look at a few examples.

Exercise 6.2.5

Find solution(s) to the following system of linear equations if any.

$$\begin{aligned} x + 2y + z &= 1 \\ 3x - y + z &= 6 \\ x + y - z &= -2 \end{aligned}$$

Solution.

First, the augmented matrix can be found from the system. Consider the matrix below.

$$\left[\begin{array}{ccc|c} 1 & 2 & 1 & 1 \\ 3 & -1 & 1 & 6 \\ 1 & 1 & -1 & -2 \end{array}\right]$$

Using the elementary row operations, the augmented matrix can be transformed as the following.

$$\left[\begin{array}{ccc|c} 1 & 2 & 1 & 1 \\ 3 & -1 & 1 & 6 \\ 1 & 1 & -1 & -2 \end{array}\right] \rightarrow \left[\begin{array}{ccc|c} 1 & 2 & 1 & 1 \\ 0 & -4 & 4 & 12 \\ 1 & 1 & -1 & -2 \end{array}\right] \rightarrow \left[\begin{array}{ccc|c} 1 & 2 & 1 & 1 \\ 0 & -4 & 4 & 12 \\ 0 & -1 & -2 & -3 \end{array}\right]$$

$$\rightarrow \left[\begin{array}{ccc|c} 1 & 2 & 1 & 1 \\ 0 & -1 & 1 & 3 \\ 0 & 1 & 2 & 3 \end{array}\right] \rightarrow \left[\begin{array}{ccc|c} 1 & 2 & 1 & 1 \\ 0 & -1 & 1 & 3 \\ 0 & 0 & 3 & 6 \end{array}\right] \rightarrow \left[\begin{array}{ccc|c} 1 & 2 & 1 & 1 \\ 0 & 1 & -1 & -3 \\ 0 & 0 & 1 & 2 \end{array}\right]$$

Because the augmented matrix is now in REF, the transformed linear system can be rewritten.

$$\begin{aligned} x + 2y + z &= 1 \\ y - z &= -3 \\ z &= 2 \end{aligned}$$

Back substituting, it is evident that $z = 2$ and $y = -1$. Moreover, $x = 1 + 2 - 2 = 1$. Therefore, the triple $(1, -1, 2)$ is the solution to the linear system.

Try substituting to the original system! You can notice that the solution is valid. Below is the second example of using Gaussian Elimination.

Exercise 6.2.6

Find solution(s) to the following system of linear equations if any.

$$\begin{aligned} x + 2y + 3z &= 4 \\ y + 3z &= 5 \\ x + y &= 1 \end{aligned}$$

Solution.

First and foremost, the augmented matrix can be found.

$$\left[\begin{array}{ccc|c} 1 & 2 & 3 & 4 \\ 0 & 1 & 3 & 5 \\ 1 & 1 & 0 & 1 \end{array}\right] \rightarrow \left[\begin{array}{ccc|c} 1 & 2 & 3 & 4 \\ 0 & 1 & 3 & 5 \\ 0 & -1 & -3 & -3 \end{array}\right] \rightarrow \left[\begin{array}{ccc|c} 1 & 2 & 3 & 4 \\ 0 & 1 & 3 & 5 \\ 0 & 0 & 0 & 2 \end{array}\right]$$

Writing the transformed augmented matrix into a system of linear equations, the following system is obtained.

$$\begin{aligned} x + 2y + 3z &= 4 \\ y + 3z &= 5 \\ 0 &= 2 \end{aligned}$$

Notice that $0 = 2$ can never be true. Therefore, no solution exists.

We could double check our answer without using matrices. Notice that $3z - x = 4$ and $x = 3z - 4$. Similarly, $y = 5 - 3z$. Substituting to the first equation, $3z - 4 + 10 - 6z + 3z = 4$, or $6 = 4$, which is not true. Therefore, our results align! Below is the third example of using the algorithm.

Exercise 6.2.7

Find solution(s) to the following system of linear equations if any.

$$\begin{aligned} x - y - 2z &= 3 \\ 2x + y - 7z &= 3 \\ x + y - 4z &= 1 \end{aligned}$$

Solution.

Consider the following augmented matrix.

$$\left[\begin{array}{ccc|c} 1 & -1 & -2 & 3 \\ 2 & 1 & -7 & 3 \\ 1 & 1 & -4 & 1 \end{array}\right]$$

Utilizing Gaussian Elimination, the augmented matrix can be transformed as following.

$$\left[\begin{array}{ccc|c} 1 & -1 & -2 & 3 \\ 2 & 1 & -7 & 3 \\ 1 & 1 & -4 & 1 \end{array}\right] \to \left[\begin{array}{ccc|c} 1 & -1 & -2 & 3 \\ 2 & 1 & -7 & 3 \\ 0 & 2 & -2 & -2 \end{array}\right] \to \left[\begin{array}{ccc|c} 1 & -1 & -2 & 3 \\ 0 & 1 & -1 & -1 \\ 2 & 1 & -7 & 3 \end{array}\right]$$
$$\to \left[\begin{array}{ccc|c} 1 & -1 & -2 & 3 \\ 0 & 1 & -1 & -1 \\ 0 & 3 & -3 & -3 \end{array}\right] \to \left[\begin{array}{ccc|c} 1 & -1 & -2 & 3 \\ 0 & 1 & -1 & -1 \\ 0 & 0 & 0 & 0 \end{array}\right]$$

Rewriting the system, the following equations are obtained.

$$\begin{aligned} x - y - 2z &= 3 \\ y - z &= -1 \end{aligned}$$

Notice that $y = z - 1$ and $x = y + 2z + 3 = z - 1 + 2z + 3 = 3z + 2$. Because there exists infinitely many z, the system has infinitely many solutions.

When writing such solutions, it could be written as the following.

$$x = \begin{bmatrix} 3z + 2 \\ z - 1 \\ z \end{bmatrix}$$

For the next section, let's discuss the inverse of a matrix and its application in linear systems.

§6.3 The Multiplicative Inverse

As we discuss matrix multiplication, you may have wondered about matrix division. I mean it would be so nice if we could just divide the coefficient matrix to find our variables. Although

there is no "division" operation per se, we could achieve similar operation with the multiplicative inverse of a matrix, or simply the inverse matrix. In other words, some solutions to linear systems can be written as $x = A^{-1}b$.

Definition 6.3.1

The **inverse of a matrix** is a matrix that produces an identity matrix when multiplied with the original matrix. The inverse of A is denoted as A^{-1}.

We could see that matrices A and B are the inverse of each other if the following equation is satisfied.

$$AB = BA = I$$

Below is an example where $B = A^{-1}$.

$$A = \begin{bmatrix} 1 & 2 \\ 3 & 5 \end{bmatrix}, \quad B = \begin{bmatrix} -5 & 2 \\ 3 & -1 \end{bmatrix}$$

One thing to note from the definition is that A and B must be square matrices. Say we have $A_{4\times 2}$ and $B_{2\times 4}$. Even if both AB and BA return an identity matrix, the order will be different. Therefore, for a matrix to have the inverse, it must be a square matrix.

Definition 6.3.2

A matrix is **invertible**, or **nonsingular**, if it retains an inverse matrix. A matrix is **singular** if it does not have an inverse matrix.

Now from the definition above, we could establish a theorem.

Theorem 6.3.3

If all diagonal elements of a matrix are nonzero constants, then the matrix is invertible.

Proof.

Consider the matrix $A_{m\times m}$ defined below.

$$A = \begin{bmatrix} a_{11} & a_{12} & a_{13} & \cdots & a_{1m} \\ a_{21} & a_{22} & a_{23} & \cdots & a_{2m} \\ a_{31} & a_{32} & a_{33} & \cdots & a_{3m} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ a_{m1} & a_{m2} & a_{m3} & \cdots & a_{mm} \end{bmatrix}$$

It suffices to show that there exists the inverse matrix for all values of the elements. Consider

the inverse below.

$$A^{-1} = \begin{bmatrix} \frac{1}{a_{11}} & 0 & 0 & \cdots & 0 \\ 0 & \frac{1}{a_{22}} & 0 & \cdots & 0 \\ 0 & 0 & \frac{1}{a_{33}} & \cdots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & \cdots & \frac{1}{a_{mm}} \end{bmatrix}$$

Because a_{ii} for integer $i \in [1, m]$ are nonzero numbers, all elements in the matrix A^{-1} are defined, and the inverse matrix exists.

Now, it is important to note that not all square matrices have the inverse. For example, consider a matrix with zero row. The same row in the product matrices will become a zero row. By definition, it is not possible to make an identity matrix with a zero row.

Now that we discussed the existence, we could also investigate if the inverse matrix is unique just as a nonzero integer has a unique reciprocal.

Theorem 6.3.4

If there exists an inverse matrix, the inverse is unique.

Proof.

For the sake of contradiction, let B and C be different inverse of A. Consider the following equation.

$$B = BI = B(AC) = (BA)C = IC = C$$

Because the fact that $B = C$ contradicts the initial assumption that $B \neq C$, all inverse matrices are unique.

Before we discuss how we actually find an inverse, let's discuss four important properties of inverse matrices.

Theorem 6.3.5

For invertible square matrices A and B and a nonzero scalar λ, the following equations hold.

1. $(A^{-1})^{-1} = A$
2. $(A^\intercal)^{-1} = (A^{-1})^\intercal$
3. $(AB)^{-1} = B^{-1}A^{-1}$
4. $(\lambda A)^{-1} = (\frac{1}{\lambda})A^{-1}$

There's a lot to prove and let's do them one by one starting with the first one!

Proof.

Let matrix $B = A^{-1}$. Substituting, it suffices to show that $B^{-1} = A$. By definition of inverse matrix, the equation $AB = BA = 1$ holds. In other words, A is the inverse of B and the

property holds.

Below is the proof for the second property.

Proof.

Notice that $(AB)^\intercal = B^\intercal A^\intercal$ from Theorem 5.4.4. Let matrix $B = A^{-1}$ and consider the following equation.

$$I = (AB)^\intercal = B^\intercal A^\intercal = \left(A^{-1}\right)^\intercal A^\intercal$$

In other words, $(A^{-1})^\intercal$ is the inverse of $A^\intercal$, or $(A^\intercal)^{-1}$, proving the property.

Continuing, here is the proof for the third property.

Proof.

Consider the following equation that uses the associative property of matrix multiplication.

$$(B^{-1}A^{-1})(AB) = B^{-1}(A^{-1}A)B = B^{-1}(I)B = B^{-1}(IB) = B^{-1}B = I$$

Because $B^{-1}A^{-1}$ is the inverse of AB, the property is true.

Finally, here is the proof for the last property.

Proof.

Let $B = \lambda A$. Consider the following equation.

$$\frac{1}{\lambda}A^{-1}B = \frac{1}{\lambda}A^{-1}\lambda A = A^{-1}A = I$$

In other words, $\frac{1}{\lambda}A^{-1}$ is the inverse of λA, proving the fourth property.

Now that we took a look at different properties of inverse matrices, let's discuss how to actually find one with tools to consider.

Definition 6.3.6

For a matrix A that is not necessarily a square matrix, an **elementary matrix**, denoted as E, is a square matrix that performs an elementary row operation by multiplying with A.

Let's take a look at an example. Consider matrix A defined below, and we want to perform elementary row operations of changing the order of rows 1 and 3 and add two times the fourth row to the second respectively.

$$A = \begin{bmatrix} 1 & 2 & 3 & 4 \\ 1 & 0 & 2 & 0 \\ 2 & 4 & 6 & 8 \\ 1 & 1 & 1 & 1 \end{bmatrix}$$

The elementary row operations above can be complicated with words, but we can represent

that with two elementary matrices respectively. Letting E_1 and E_2 be elementary matrices for each operation, we can represent the elementary row operations as the following.

$$E_1 = \begin{bmatrix} 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}, \quad E_2 = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 2 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Let's multiply to see if we are right.

$$E_1 A = \begin{bmatrix} 2 & 4 & 6 & 8 \\ 1 & 0 & 2 & 0 \\ 1 & 2 & 3 & 4 \\ 1 & 1 & 1 & 1 \end{bmatrix}, \quad E_2 A = \begin{bmatrix} 1 & 2 & 3 & 4 \\ 3 & 2 & 4 & 2 \\ 2 & 4 & 6 & 8 \\ 1 & 1 & 1 & 1 \end{bmatrix}$$

Now we can notice three observations and tips for constructing elementary matrices. For each elementary row operation, we could do the following.

Definition 6.3.7

Methods to construct the elementary matrix for each elementary row operation:

- To interchange row a and b, write the corresponding identity matrix and interchange row a and b from the identity matrix.
- To multiply a nonzero scalar k to a^{th} row, start from the corresponding identity matrix and change the 1 from a^{th} row of the identity matrix to k.
- To add row a multiplied by the nonzero scalar k to row b, start from the corresponding identity matrix and find the b^{th} row in the matrix. Then, change the zero from the a^{th} column of the row to k.

From this, we can notice the following theorem.

Theorem 6.3.8

All elementary matrices are invertible.

Proof.

The proof is rather simple. Because all elementary row operations can be inverted with the inverse operation, all elementary matrices are invertible.

Building onto the theorem, we could prove the uniqueness.

Theorem 6.3.9

For each elementary row operation, the corresponding elementary matrix is unique.

Proof.

For the sake of contradiction, assume there exist elementary matrices E and $F \neq E$. By definition, $EA = FA$ and $A = EE^{-1}A = E^{-1}FA$. By Theorem 5.3.7, $E^{-1}F = I$ and $E = F$ is obtained by Theorem 6.3.4. By contradiction, it is shown that elementary matrices are unique.

There are more interesting lemmas and theorems, but let's discuss them in latter notes. However, let's discuss a few theorems for each elementary row operation before moving on to finding an inverse matrix.

Theorem 6.3.10

For each elementary matrix for corresponding elementary row operations, the following properties hold.

1. For an elementary matrix that changes the order of two rows, the inverse of such a matrix is itself.
2. For an elementary matrix that scales a row with nonzero constant k, the inverse can be obtained by replacing k in the elementary matrix with $\frac{1}{k}$.
3. For an elementary matrix that adds the scaled row into the other, its inverse can be obtained by replacing the nonzero scalar k with $-k$.

As always, let's start by proving the first property.

Proof.

The inverse of an elementary matrix can be considered as undoing the elementary row operation. To undo the interchanging of two rows, the two rows can be interchanged again. In other words, because the elementary row operation $EE = I$ by Theorem 5.3.7, $E = E^{-1}$.

Continuing, here is the proof for the second property.

Proof.

Let elementary matrices E and F be matrices that scale a row by a nonzero scalar k and $\frac{1}{k}$ respectively. By definition, $F(EI) = I$ and $FE = I$. Therefore, $F = E^{-1}$ and the property is shown.

Lastly, here is the proof for the final property.

Proof.

Similar to the proof for the second property, let elementary matrices E and F represent the elementary row operations that add a row scaled by nonzero constant k to another and the operation that adds a row scaled by $-k$ to another respectively. By definition, $F(EI) = I$ and $F = E^{-1}$ as demonstrated above.

Now before we actually find an inverse, we need to know which square matrices have the

inverse matrix.

Definition 6.3.11

A square matrix A is invertible if and only if it can be represented as an identity matrix after sequences of elementary row operations, i.e. there exist elementary matrices E_i such that the following equation is true.

$$E_k E_{k-1} \cdots E_1 A = I$$

This is a very important result that we will prove later in a note, so don't worry. Finally, here are the steps to find an inverse if there exists one.

Definition 6.3.12

Steps to Find the Inverse Matrix of $\mathbf{A_{m\times m}}$:

1. Write an augmented matrix $[A \mid I]$ for I_m.
2. Utilize elementary row operations on the augmented matrix for the block A to be in REF.
3. If the main diagonal of the left partition contains a zero, A is singular. If not, the next step can be continued.
4. Use elementary row operations again to transform the block A in REF to an identity matrix. The right partition block is the inverse of A.

Let's take a look at a few examples.

Exercise 6.3.13

Find the inverse of $A = \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{bmatrix}$.

Solution.

First, the augmented matrix can be constructed.

$$\left[\begin{array}{ccc|ccc} 1 & 2 & 3 & 1 & 0 & 0 \\ 4 & 5 & 6 & 0 & 1 & 0 \\ 7 & 8 & 9 & 0 & 0 & 1 \end{array}\right]$$

Consider the following elementary row operations.

$$\left[\begin{array}{ccc|ccc} 1 & 2 & 3 & 1 & 0 & 0 \\ 4 & 5 & 6 & 0 & 1 & 0 \\ 7 & 8 & 9 & 0 & 0 & 1 \end{array}\right] \to \left[\begin{array}{ccc|ccc} 1 & 2 & 3 & 1 & 0 & 0 \\ 8 & 10 & 12 & 0 & 2 & 0 \\ 7 & 8 & 9 & 0 & 0 & 1 \end{array}\right] \to \left[\begin{array}{ccc|ccc} 1 & 2 & 3 & 1 & 0 & 0 \\ 1 & 2 & 3 & 0 & 2 & -1 \\ 7 & 8 & 9 & 0 & 0 & 1 \end{array}\right]$$

$$\rightarrow \left[\begin{array}{ccc|ccc} 1 & 2 & 3 & 1 & 0 & 0 \\ 0 & 0 & 0 & -1 & 2 & -1 \\ 7 & 8 & 9 & 0 & 0 & 1 \end{array}\right]$$

It is evident that the main diagonal will contain a zero after the transformation. Therefore, the matrix is not invertible.

Exercise 6.3.14

Find the inverse of the following matrix A.

$$A = \begin{bmatrix} 1 & 0 & 0 \\ 2 & 1 & 1 \\ 0 & 1 & 2 \end{bmatrix}$$

Solution.

First and foremost, the augmented matrix can be constructed.

$$\left[\begin{array}{ccc|ccc} 1 & 0 & 0 & 1 & 0 & 0 \\ 2 & 1 & 1 & 0 & 1 & 0 \\ 0 & 1 & 2 & 0 & 0 & 1 \end{array}\right] \rightarrow \left[\begin{array}{ccc|ccc} 1 & 0 & 0 & 1 & 0 & 0 \\ 0 & 1 & 2 & 0 & 0 & 1 \\ 2 & 1 & 1 & 0 & 1 & 0 \end{array}\right] \rightarrow \left[\begin{array}{ccc|ccc} 1 & 0 & 0 & 1 & 0 & 0 \\ 0 & 1 & 2 & 0 & 0 & 1 \\ 0 & 0 & -1 & -2 & 1 & -1 \end{array}\right]$$
$$\rightarrow \left[\begin{array}{ccc|ccc} 1 & 0 & 0 & 1 & 0 & 0 \\ 0 & 1 & 2 & 0 & 0 & 1 \\ 0 & 0 & 1 & 2 & -1 & 1 \end{array}\right]$$

Continuing, the left partition can be transformed into an identity matrix with elementary row operations.

$$\left[\begin{array}{ccc|ccc} 1 & 0 & 0 & 1 & 0 & 0 \\ 0 & 1 & 2 & 0 & 0 & 1 \\ 0 & 0 & 1 & 2 & -1 & 1 \end{array}\right] \rightarrow \left[\begin{array}{ccc|ccc} 1 & 0 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & -4 & 2 & -1 \\ 0 & 0 & 1 & 2 & -1 & 1 \end{array}\right]$$

Therefore, the inverse of A is the following.

$$A^{-1} = \begin{bmatrix} 1 & 0 & 0 \\ -4 & 2 & -1 \\ 2 & -1 & 1 \end{bmatrix}$$

We could actually multiply to see if we get I_3.

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

Indeed it is the inverse! In the next section of the note, we will discuss LU decomposition before

completing our note on linear systems.

§6.4 LU Decomposition

The last popular method to solve a linear system that we will discuss is LU decomposition. As always, let's start by discussing fundamental definitions.

Definition 6.4.1

LU decomposition is the process of factorizing an invertible matrix into the product of lower and upper triangular matrices where the diagonal elements of the lower triangular matrix are 1.

Before we look at an important theorem and how it can be used to solve a linear system, here is an example of LU decomposition for matrix A defined as the following.

$$A = \begin{bmatrix} 1 & 2 & 3 \\ 2 & 6 & 9 \\ 3 & 10 & 18 \end{bmatrix}$$

From here, notice that matrix A can be transformed to upper triangular matrix with the following elementary row operations of R_3.

$$\begin{bmatrix} 1 & 2 & 3 \\ 2 & 6 & 9 \\ 3 & 10 & 18 \end{bmatrix} \to \begin{bmatrix} 1 & 2 & 3 \\ 0 & 2 & 3 \\ 3 & 10 & 18 \end{bmatrix} \to \begin{bmatrix} 1 & 2 & 3 \\ 0 & 2 & 3 \\ 0 & 4 & 9 \end{bmatrix} \to \begin{bmatrix} 1 & 2 & 3 \\ 0 & 2 & 3 \\ 0 & 0 & 3 \end{bmatrix}$$

Notice that the operations above can be represented as the matrix multiplication with R_3 that are lower triangular matrices.

$$\begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & -2 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ -3 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 \\ -2 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} A = \begin{bmatrix} 1 & 2 & 3 \\ 0 & 2 & 3 \\ 0 & 0 & 3 \end{bmatrix}$$

Therefore, the following equation is obtained.

$$A = \left(\begin{bmatrix} 1 & 0 & 0 \\ 2 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 3 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 2 & 1 \end{bmatrix} \right) \begin{bmatrix} 1 & 2 & 3 \\ 0 & 2 & 3 \\ 0 & 0 & 3 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ 2 & 1 & 0 \\ 3 & 2 & 1 \end{bmatrix} \begin{bmatrix} 1 & 2 & 3 \\ 0 & 2 & 3 \\ 0 & 0 & 3 \end{bmatrix}$$

A is now represented as the product of lower and upper triangular matrices! Now that we know how LU decomposition works, let's take a look at an important lemma on triangular matrices to prove a theorem on LU decomposition.

Lemma 6.4.2

The inverse of upper and lower triangular matrices are upper and lower triangular matrices respectively.

Proof.

First and foremost, consider an invertible upper triangular matrix $A_{m\times m} = [a_{ij}]$ and its inverse $B = [b_{ij}]$. By definition, its product $C = [c_{ij}]$ can be represented as $C = [\sum_{k=1}^{m} a_{ij}b_{ji}]$. Because $c_{ij} = 0$ for $i \neq j$ and $c_{ij} = 1$ for $i = j$, the values of b_{ji} can be found.

By definition, $a_{ij} = 0$ for $i > j$. Therefore, $c_{ij} = 0$ for $i > j$. Similarly, if $b_{ji} = 0$ for $j > i$, then $c_{ij} = 0$ for $i \neq j$. Continuing, if $b_{ij} = \frac{1}{a_{ij}}$ for $i = j$, B can be constructed, which is an upper triangular matrix. Because the inverse matrix is unique, the inverse of an upper triangular matrix is an upper triangular matrix if there exists one.

Similarly, the fact that the inverse of an lower triangular matrix is a lower triangular matrix can be proven. Let $A'_{m\times m} = [a'_{ij}]$ be a lower triangular matrix and $B' = [b'_{ij}]$ be its inverse. By definition, their product $C' = [c'_{ij}] = [\sum_{k=1}^{m} a'_{ij}b'_{ji}]$ is the identity matrix I_m. Thus, it suffices to show that $c'_{ij} = 0$ for $i \neq j$ and $c'_{ij} = 1$ for $i = j$.

Note that by definition, $a'_{ij} = 0$ for $i < j$. Therefore, $c'_{ij} = 0$ for $i < j$. Similarly, if $b'_{ji} = 0$ for $i > j$, and $b'_{ij} = \frac{1}{a'_{ij}}$ for $i = j$, then the valid inverse matrix B is constructed. Because B is a lower triangular matrix and the inverse matrix is unique, the inverse of a lower triangular matrix is a lower triangular matrix.

With the lemma in mind, let's prove the following theorem.

Theorem 6.4.3

An invertible matrix has a LU decomposition if and only if multiplying a sequence of lower triangular elementary matrices for elementary row operations can transform the matrix into an upper triangular matrix.

Proof.

First, the statement that a nonsingular matrix has a LU decomposition if multiplying a sequence of lower triangular elementary matrices for elementary row operations can transform the matrix into an upper triangular matrix can be proven. Let A be an invertible matrix that can be represented as the following.

$$E_{n,n-1} \cdots E_{31}E_{21}A = U$$

for lower matrices E and an upper matrix U. Notice that A can be written as the following.

$$A = \left(E_{21}^{-1}E_{31}^{-1} \cdots E_{n,n-1}^{-1}\right) U$$

From Lemma 6.4.2, the inverses of the elementary matrices are lower triangular matrices. Moreover, by Theorem 5.4.13, the product of the lower triangular matrices is also a lower triangular matrix. Therefore, for a lower triangular matrix L, A can be written as the following.

$$A = LU$$

Therefore, the first half of the statement holds.

For the second half of the statement, consider the following equation.

$$I = AA^{-1} = LUA^{-1} = L\left(UA^{-1}\right)$$

Therefore, L is nonsingular. From the proof for Lemma 6.4.2, it is evident that the diagonal elements of an invertible lower triangular matrix are nonzero as the diagonal elements in the inverse cannot be defined if a zero is present in the diagonal elements of the lower triangular matrix.

Let D be the diagonal matrix formed with the diagonal elements of L and define L' such that the equation $L'D = L$ is satisfied. Moreover, let $U' = DU$. Thus, the following equation is obtained.

$$A = LU = L'DU = L'U'$$

Recall that L is invertible. Because L^{-1} exists and $(L'D)^{-1} = D^{-1}L'^{-1}$ exists, L' is also invertible. In other words, $L'^{-1}A = U'$. Therefore, it suffices to show that L'^{-1} is a product of lower triangular elementary matrices. By definition of D, the diagonal elements of L' are 1. Thus, it can be interpreted as a sequence of elementary row operations. In other words, it can be represented as a product of lower triangular elementary matrices, proving the latter half of the theorem.

Now that we discussed what LU decomposition is, let's see how we can use it to solve a linear system.

6.4.1 Application in Linear Systems

When we apply LU decomposition in solving linear systems, two relatively easy systems must be solved. Consider the system $Ax = b$ where $A = LU$. First, y must be found from the following system with forward substitution.

$$Ly = b$$

Then, we can solve for x with back substitution with the following system.

$$Ux = y$$

The good news is that both systems are relatively easy to solve. The reason is simple. Consider the following equation.

$$Ax = LUx = b$$

Let $y = Ux$, then $Ly = b$. If we find $Ly = b$, then we can find $Ux = y$ as we know y. This way, we can find x while exploiting the triangular form. Let's take a look at an example.

Exercise 6.4.4

Solve the following system of linear equations.

$$\begin{aligned} x - 2y + z &= 2 \\ 2x + y - z &= 1 \\ 3x - y + 2z &= 15 \end{aligned}$$

Solution.

First and foremost, the linear system can be represented as the matrix multiplication below.

$$\begin{bmatrix} 1 & -2 & 1 \\ 2 & 1 & -1 \\ 3 & -1 & 2 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \end{bmatrix} = \begin{bmatrix} 2 \\ 1 \\ 15 \end{bmatrix}$$

Continuing, the coefficient matrix A can be transformed as the following.

$$\begin{bmatrix} 1 & -2 & 1 \\ 2 & 1 & -1 \\ 3 & -1 & 2 \end{bmatrix} \to \begin{bmatrix} 1 & -2 & 1 \\ 0 & 5 & -3 \\ 3 & -1 & 2 \end{bmatrix} \to \begin{bmatrix} 1 & -2 & 1 \\ 0 & 5 & -3 \\ 0 & 5 & -1 \end{bmatrix} \to \begin{bmatrix} 1 & -2 & 1 \\ 0 & 5 & -3 \\ 0 & 0 & 2 \end{bmatrix}$$

Using elementary matrices, the transformation above can be written as the following.

$$\begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & -1 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ -3 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 \\ -2 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} A = \begin{bmatrix} 1 & -2 & 1 \\ 0 & 5 & -3 \\ 0 & 0 & 2 \end{bmatrix}$$

Completing the LU decomposition, A can be represented as the product of the following lower and upper triangular matrices.

$$\begin{aligned} A &= \left(\begin{bmatrix} 1 & 0 & 0 \\ 2 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 3 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 1 & 1 \end{bmatrix} \right) \begin{bmatrix} 1 & -2 & 1 \\ 0 & 5 & -3 \\ 0 & 0 & 2 \end{bmatrix} \\ &= \begin{bmatrix} 1 & 0 & 0 \\ 2 & 1 & 0 \\ 3 & 1 & 1 \end{bmatrix} \begin{bmatrix} 1 & -2 & 1 \\ 0 & 5 & -3 \\ 0 & 0 & 2 \end{bmatrix} \end{aligned}$$

Continuing, the following system could be solved.

$$\begin{bmatrix} 1 & 0 & 0 \\ 2 & 1 & 0 \\ 3 & 1 & 1 \end{bmatrix} \begin{bmatrix} x' \\ y' \\ z' \end{bmatrix} = \begin{bmatrix} 2 \\ 1 \\ 15 \end{bmatrix}$$

The matrix multiplication above can be written as the following system of linear equations.

$$\begin{aligned} x' &= 2 \\ 2x' + y' &= 1 \\ 3x' + y' + z' &= 15 \end{aligned}$$

Substituting, $y' = 1 - 4 = -3$ and $z' = 15 - 6 + 3 = 12$. Thus, the final system can be written as the following.

$$\begin{bmatrix} 1 & -2 & 1 \\ 0 & 5 & -3 \\ 0 & 0 & 2 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \end{bmatrix} = \begin{bmatrix} 2 \\ -3 \\ 12 \end{bmatrix}$$

Writing in linear system, x, y, and z can be found.

$$\begin{aligned} x - 2y + z &= 2 \\ 5y - 3z &= -3 \\ 2z &= 12 \end{aligned}$$

Therefore, $z = 6$, $y = 3$, and $x = 2$.

With LU decomposition, we completed our discussion on solving systems of linear equations! Below are some practice problems for the topics that we covered.

§6.5 Practice Problems

1. Show that for an invertible square matrix A, the system $Ax = b$ is always consistent.

2. Show that the following system cannot have a unique solution for all $k \in \mathbb{R}$.

$$\begin{bmatrix} 1 & 2 \\ 2 & 4 \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} 1 \\ k \end{bmatrix}$$

3. Use Gaussian Elimination to find the solution for the following system.

$$\begin{aligned} x + y + z &= 10 \\ 3x + 4y - z &= 2 \\ -x - 3y + z &= 4 \end{aligned}$$

4. Solve the same system above using LU decomposition.

5. Find the inverse of the following matrix.

$$\begin{bmatrix} 1 & 1 & 1 \\ 3 & 4 & -1 \\ -1 & -3 & 1 \end{bmatrix}$$

   Then, find the values of x, y, z that satisfy the following system.

$$\begin{bmatrix} 1 & 1 & 1 \\ 3 & 4 & -1 \\ -1 & -3 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \end{bmatrix} = \begin{bmatrix} 10 \\ 2 \\ 4 \end{bmatrix}$$

Note 7

Matrices and Vector Spaces

Now that we discussed matrices and their application in linear systems, let's return to the vector spaces to see what matrices represent in vector spaces. This note will introduce more concepts on vector spaces and matrices like row spaces and rank of a matrix and linear transformation.

§7.1 Row Space and Rank of a Matrix

As always, let's get started with the definition.

Definition 7.1.1

The span of rows of a matrix $A_{m\times n}$ in $\mathbb{R}^n$ is the **row space** of the matrix A. The dimension of the row space is known as the **row rank**.

We can notice that by definition, the row space is a subspace of $\mathbb{R}^n$. Let's take a look at an example. Consider the following matrix A.

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

The row space $\text{row}(A)$ is represented by $\text{span}\{(1,2),(3,4),(5,6)\}$ and its rank $\text{r}(A) = \dim(\text{row}(A)) = \dim(\mathbb{R}^2) = 2$. Connecting this with our knowledge in matrices, we can establish the following theorem.

Theorem 7.1.2

A basis for the row space of a matrix A in REF is the set of nonzero rows, and the number of such rows is the row rank.

Proof.

By definition, the nonzero rows span the row space since the zero rows do not contribute the vector addition. Therefore, it suffices to show that the set of nonzero vectors is linearly independent.

By definition of matrices in REF, there exist no two rows with the same initial nonzero entry, implying its uniqueness. Therefore, by induction, all coefficients in the linear combination of the set of nonzero vectors must be zero. Thus, the set of nonzero rows is a basis for the row space of A. Moreover, by definition, the rank of A is the number of nonzero rows in A.

Continuing, we can also establish the following theorem.

Theorem 7.1.3

If A and B are matrices that can be obtained by applying elementary row operations to each other, then $\text{row}(A) = \text{row}(B)$.

Proof.

Let $A = \{v_1, \ldots, v_n\}$ and B be a matrix that is obtained by a single row operation from A. For some $j, k \in [1, n]$ such that $j \neq k$, it is evident that $\text{span}\{v_1, \ldots, v_j, \ldots, v_k, \ldots, v_n\} = \text{span}\{v_1, \ldots, v_k, \ldots, v_j, \ldots, v_n\}$. Moreover, for some scalar λ, $\text{span}\{v_1, \ldots, \lambda v_k, \ldots, v_n\} = \{v_1, \ldots, v_n\}$ and $\text{span}\{v_1, \ldots, v_k + \lambda v_j, \ldots, v_n\}$ by statement four in Theorem 4.3.9. Continuing with induction, the same applies to B that is obtained by a sequence of elementary operations from A. Thus, the theorem holds.

From the two theorems above, we can notice that for any matrix A, we can find its basis and rank by transforming it into REF with elementary row operations and observing the nonzero rows. Below is a quick example.

Exercise 7.1.4

Find the basis, row space, and rank of the following matrix.

$$\begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{bmatrix}$$

Solution.

First and foremost, the matrix can be transformed into REF with elementary row operations.

$$\begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{bmatrix} \to \begin{bmatrix} 1 & 2 & 3 \\ 0 & -3 & -6 \\ 7 & 8 & 9 \end{bmatrix} \to \begin{bmatrix} 1 & 2 & 3 \\ 0 & 1 & 2 \\ 7 & 8 & 9 \end{bmatrix} \to \begin{bmatrix} 1 & 2 & 3 \\ 0 & 1 & 2 \\ 0 & -6 & -12 \end{bmatrix} \begin{bmatrix} 1 & 2 & 3 \\ 0 & 1 & 2 \\ 0 & 0 & 0 \end{bmatrix}$$

Therefore, a basis for the row space, row space, and rank of the matrix are $\{(1,2,3),(0,1,2)\}$, $\text{span}\{(1,2,3),(0,1,2)\}$, and 2 respectively.

We could also consider the following theorem.

Theorem 7.1.5

For a vector space V and its basis $S = \{v_1, \ldots, v_n\}$ where $u, \{u_1, \ldots, u_m\} \in V$, $u \in \text{span}\{u_1, \ldots, u_m\}$ if and only if a linear combination of a coordinate representation of $\{u_1, \ldots, u_m\}$ with respect to S is the coordinate representation of u with respect to S.

Proof.

First and foremost, notice that $u \in \text{span}\{u_1, \ldots, u_m\}$ if and only if for some constants c_k, $u = \sum_{k=1}^{m} c_k u_k$. Let the coordinate representation of u_i with respect to S be $[a_{i1} \cdots a_{in}]$. Substituting, it could be inferred that $u \in \text{span}\{u_1, \ldots, u_m\}$ holds if and only if u can be represented as the following.

$$u = \sum_{i=1}^{m} \sum_{j=1}^{n} c_i(a_{ij}v_j) = \sum_{i=1}^{n} \sum_{j=1}^{m} (c_j a_{ji})v_i$$

In other words, $u \in \text{span}\{u_1, \ldots, u_m\}$ if and only if the coordinate representation of u with respect to S is the following.

$$[u]_S = \begin{bmatrix} \sum_{j=1}^{m}(c_j a_{j1}) \\ \sum_{j=1}^{m}(c_j a_{j2}) \\ \vdots \\ \sum_{j=1}^{m}(c_j a_{jn}) \end{bmatrix} = \sum_{k=1}^{m} c_k \begin{bmatrix} a_{k1} \\ \vdots \\ a_{kn} \end{bmatrix}$$

Therefore, the theorem holds.

Continuing from this theorem, we can establish another theorem.

Theorem 7.1.6

For an n-dimensional vector space V and its basis B, consider a subset $U = \{u_1, \ldots, u_m\} \subset V$. Construct a matrix $A = [a_{ij}]$ such that the coordinate representation of u_i with respect to B is the i^{th} row of A. For some set of vectors $S \subset V$, if a basis for the row space of A is the coordinate representation of vectors in S with respect to B, then S is a basis for $\text{span}(U)$.

Proof.

First and foremost, assume that the coordinate representation of vectors in S with respect to B is a basis for $\text{row}(A)$. By definition, the span of such representations is the row space of A. Notice that by Theorem 7.1.5, an arbitrary vector $v \in \text{span}(S)$ if and only if a linear combination of a coordinate representation of S with respect to B is the coordinate

representation of v with respect to B. In other words, $v \in \text{span}(S)$ if and only if the coordinate representation of v is in $\text{row}(A)$. By Theorem 7.1.5 again, the statement implies that $v \in \text{span}(U)$. Thus, $v \in \text{span}(S)$ if and only if $v \in \text{span}(U)$ and $\text{span}(S) = \text{span}(U)$.

To show that S is a basis for $\text{span}(U)$, or $\text{span}(S)$, it suffices to show that S is linearly independent. Notice that because the set of coordinate representations of S is a basis for $\text{row}(A)$, the set of coordinate representations of S is linearly independent. Moreover, by the proof for the Theorem 7.1.5, S is also linearly independent. Thus, S is a basis for $\text{span}(U)$.

There are lots of theorems and results! But here is another one that connects vectors and matrices.

Theorem 7.1.7

For $S \subset \mathbb{R}^n$ such that $|S| = k$, construct matrix $A_{k\times n}$ such that the rows are each vectors in S. S is linearly independent if and only if $\text{r}(A) = k$.

Proof.

First and foremost, notice that by definition, S is linearly independent if and only if it is a basis for its span. Moreover, by construction, $\text{span}(S) = \text{row}(A)$ and S is linearly independent if and only if it is a basis for the row space of A. Continuing, S is a basis for the row space of A if and only if $\text{r}(A) = |S|$ by definition. Therefore, S is linearly independent if and only if $\text{r}(A) = |S|$.

To be honest, I procrastinated a bit when I was learning to prove the theorems above. For some reason, the proofs were hard to understand and less intuitive than the proofs that I have done through olympiad style problems. I think what helped me to understand the proofs is to just continue looking at it over and over again until I fully knew the terms. Because the proofs above prioritize understanding of the basic concepts over creativity, I think procrastinating helped me fully understand the proofs.

So far, we discussed about row spaces and row ranks. How about columns? As you may have guessed, we could do the same for columns!

Definition 7.1.8

The span of the columns of a matrix $A_{m\times n}$ is the **column space** of A, denoted as $\text{col}(A)$. The dimension of such space is **column rank** of A.

We could notice that because rows and columns of a matrix are not necessarily the same, the row space and column space of a matrix are generally different. One special example in which they are the same would be identity matrices.

Now when we discussed about row rank of a matrix A, we used the notation $\text{row}(A)$. That is because the row rank and column rank of a matrix are always equal.

Theorem 7.1.9

The row rank and column rank of a matrix A are always equal.

Proof.

First and foremost, notice that by definition, $\text{row}(A^\intercal) = \text{col}(A)$ and $\text{row}(A) = \text{col}(A^\intercal)$. Therefore, the following equations are true.

$$\dim(\text{row}(A^\intercal)) = \dim(\text{col}(A))$$
$$\dim(\text{row}(A)) = \dim(\text{col}(A^\intercal))$$

Notice that if $\dim(\text{col}(A)) \leq \dim(\text{row}(A))$, then the following expression holds.

$$\dim(\text{row}(A)) = \dim(\text{col}(A^\intercal)) \leq \dim(\text{row}(A^\intercal)) = \dim(\text{col}(A))$$

Therefore, if $\dim(\text{col}(A)) \leq \dim(\text{row}(A))$ holds, then $\dim(\text{row}(A)) \leq \dim(\text{col}(A))$ and $\dim(\text{col}(A)) = \dim(\text{row}(A))$. Thus, it suffices to show that $\dim(\text{col}(A)) \leq \dim(\text{row}(A))$.

Consider a matrix $A = [a_{ij}]$ that has the order $m \times n$. Notice that each row can be represented as an n-tuple. Let $U = \{u_1, \ldots, u_k\} \subset \mathbb{R}^n$ be a basis for $\text{row}(A)$. By definition, each row A_i of A can be written as the following for some constants c_{ij}.

$$A_i = \sum_{j=1}^{k} c_{ij} u_j$$

Moreover, notice that each element can be written as the following.

$$a_{ij} = \sum_{n=1}^{k} c_{in} u_{nj}$$

Continuing, let $V = \{v_1, \ldots, v_k\}$ be the set of such coefficients where v_i is defined as the following.

$$v_i = \begin{bmatrix} c_{1i} \\ c_{2i} \\ \vdots \\ c_{mi} \end{bmatrix}$$

Therefore, each column of A can be written as the following m-tuple.

$$\begin{bmatrix} a_{1j} \\ a_{2j} \\ \vdots \\ a_{mj} \end{bmatrix} = \sum_{n=1}^{k} \begin{bmatrix} c_{1n} \\ c_{2n} \\ \vdots \\ c_{mn} \end{bmatrix} u_{nj} = \sum_{n=1}^{k} u_{nj} v_n$$

Because each column is in the span of V, $\dim(\text{col}(A)) \leq k = \dim(\text{row}(A))$. Thus, the

theorem holds.

Now that we have column space in mind, let's return to linear systems. Let's first start with an example before looking at the theorem.

$$\begin{aligned} x + 2y + 3z &= 4 \\ x - y + z &= -1 \\ 4x + 3y + 2z &= 1 \end{aligned}$$

Notice that we can write the system as the following.

$$x \begin{bmatrix} 1 \\ 1 \\ 4 \end{bmatrix} + y \begin{bmatrix} 2 \\ -1 \\ 3 \end{bmatrix} + z \begin{bmatrix} 3 \\ 1 \\ 2 \end{bmatrix} = \begin{bmatrix} 4 \\ -1 \\ 1 \end{bmatrix}$$

This splits the coefficient matrix A in $Ax = b$ to its columns. Now the left-hand side can be interpreted as the linear combination of the columns of A, or an element in $\text{col}(A)$. In other words, the solution exists if $b \in \text{col}(A)$ and the system is inconsistent if $b \notin \text{col}(A)$. If $b \in \text{col}(A)$, then the rank of the augmented matrix $[A \mid b]$ will equal $\text{r}(A)$ since $\text{col}(A) = \text{col}([A \mid b])$. However, if $b \notin \text{col}(A)$, then $\text{r}([A \mid b]) = \text{r}(A) + 1$ since we need an extra vector for the basis for the column space. We can now write this idea as a theorem.

Theorem 7.1.10

A linear system $Ax = b$ has at least one solution if and only if $\text{r}(A) = \text{r}([A \mid b])$.

Proof.

For a linear system $Ax = b$, let $A_{m\times n} = [a_{ij}]$, A_i be the i^{th} column of A, $x = [x_i]$, and $b = [b_i]$ where x and b are m-tuples. Notice that the system can be represented as the following.

$$\sum_{k=1}^{n} x_k A_k = b$$

The system will hold if and only if b is in the span of the columns of A. Continuing, $\text{r}(A) = \text{r}([A \mid b])$ if and only if b is in the span of the existing columns of A. Thus, a solution to the linear system exists if and only if $\text{r}(A) = \text{r}([A \mid b])$.

Though satisfying, the theorem may not be significantly useful since we anyways have to write the matrices in REF. However, we can notice another interesting result.

Theorem 7.1.11

For a consistent system with n variables, if $\text{r}(A) = k$, then the final solutions can be written with $n - k$ free variables.

Proof.

First and foremost, notice that because the system $Ax = b$ is consistent and $\mathrm{r}(A) = k$, $\mathrm{r}([A \mid b]) = k$. By Theorem 7.1.2, there exist k nonzero rows, or k variables with set values for $\mathrm{r}([A \mid b])$ in REF. Therefore, there exist $n - k$ free variables.

Continuing with this theorem, we can set a corollary for homogeneous systems.

Corollary 7.1.12

For a homogeneous system $Ax = \mathbf{0}$ with n variables, there exist nontrivial solutions if and only if $\mathrm{r}(A) \neq n$.

Proof.

By Theorem 7.1.11, there exist free variables if $\mathrm{r}(A) < n$, which are not necessarily trivial solutions. However, if $\mathrm{r}(A) = n$, then there exist zero free variables and a unique solution, which is the trivial solution.

Continuing from the system, we could also apply our knowledge to invertibility of a square matrix. Now before we prove an important theorem on the relationship between rank and invertibility, let's prove a property that we couldn't prove in our previous notes. In our previous notes, we proved the uniqueness of the inverse of a square matrix. Moreover, we also knew that if $AB = BA = I$, then A and B are inverse of each other. What if we truncate the equation to $AB = I$? The answer is that they imply the same thing! Indeed for two square matrices A and B with the same order, if $AB = I$, then $BA = I$. To prove this, let's first prove two lemmas.

Lemma 7.1.13

For two square matrices A and B with order $n \times n$ such that $AB = I$, $\mathrm{r}(A) = n$.

Proof.

First and foremost, notice that for all integers $i, j \in [1, n]$, the following equation is satisfied for A_i the i^{th} row of A and B_j the j^{th} column of B.

$$A_i B_j = I_{ij}$$

Moreover, notice that if $\sum_{i=1}^{n} c_i A_i = \mathbf{0}$ for some constants c_i, then $\sum_{i=1}^{n} c_i A_i B_j = \mathbf{0}$ for any choice of j. By definition, $A_i B_j = 1$ if and only if $i = j$. In other words, if $\sum_{i=1}^{n} c_i A_i = \mathbf{0}$, then $c_i = 0$ for all integer $i \in [1, n]$. Therefore, the rows of A are linearly independent. By definition, the rows compose the basis of $\mathrm{row}(A)$ and $\mathrm{r}(A) = n$.

Continuing, let's prove the second lemma.

Lemma 7.1.14

For a square matrix $A_{n \times n}$, if $\mathrm{r}(A) = n$, then there exists a matrix B where $BA = I$.

Proof.

Notice that because $\mathrm{r}(A) = n$, none of the elements in the main diagonal for A in REF is zero after applying series of elementary row operations. From A in REF, more elementary row operations can be applied to convert A in REF to an identity matrix. Thus, the lemma holds.

Now that we have the two lemmas in mind, let's prove the main result.

Theorem 7.1.15

For square matrices A and B with the order $n \times n$, if $AB = I$, then $BA = I$.

Proof.

First and foremost, by Lemma 7.1.13, $\mathrm{r}(A) = n$. Moreover by Lemma 7.1.14, there exists matrix C such that $CA = I$. Continuing, $C = CI = CAB = IB = B$. Thus, if $AB = I$, then $BA = I$.

This is the proof for the definition that we skipped in the earlier notes! Now with the two lemmas and the theorem in mind, we can derive the following result.

Theorem 7.1.16

A square matrix $A_{n\times n}$ is invertible if and only if $\mathrm{r}(A) = n$.

Proof.

First and foremost, the first half of the theorem can be proven. By definition, if A is invertible, then there exists a square matrix of the same order B such that $AB = I$. By Lemma 7.1.13, $\mathrm{r}(A) = n$.

Continuing, by Lemma 7.1.13, if $\mathrm{r}(A) = n$, then there exists a matrix C such that $CA = I$. By Theorem 7.1.15 and definition of the inverse matrix, A is invertible.

This is it for row space and rank of a matrix! In the next section, we will discuss about linear transformations, an important topic in Linear Algebra.

§7.2 Linear Transformations

When we discussed about matrices, we saw that they kind of "change" the vector space. When I first learned about matrices in an academy as a middle school student, all I had in mind was "this is an interesting way of organizing items!" I knew what vectors and matrices were, but I didn't really know them. That changed until I learned about linear transformations. Of course, I wouldn't expect my middle school self to learn all these and truly understand what matrices represent in a vector space, but this moment, or section really changed my understanding. Like all the other sections, this is an interesting one, and let's get started!

7.2.1 Brief Review of Functions

Before we get into linear transformations, let's review about functions. I think this would be a good refresher on the terms.

Definition 7.2.1

A **function** is a special **relation** where each input from its domain has exactly one corresponding output.

For instance, a function $f : X \to Y$ is a special relation whose domain is X and whose corresponding outputs lie in Y. As you probably know, there are special terms for X and Y.

Definition 7.2.2

The set of all inputs of a function is called **domain** of the function. The set of all corresponding outputs is known as **image**, or range, of the function denoted as $\Im(f)$.

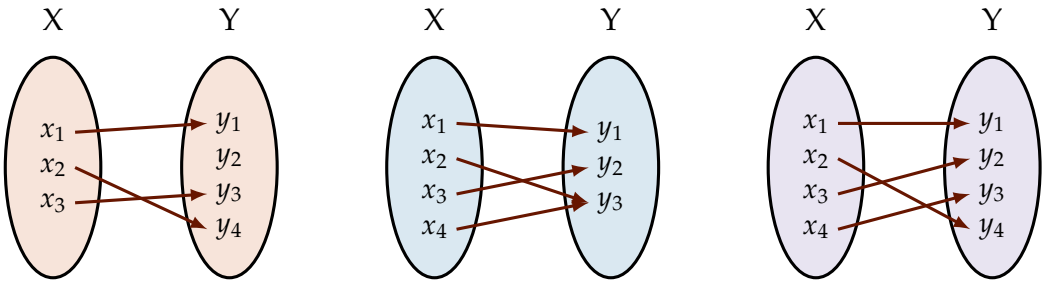
You might be familiar with the term **range**, but **image** and **range** refer to the same thing, and **image** is more commonly used in linear algebra and other courses. Now let's review about bijection, injection, and surjection.

Definition 7.2.3

A function is **injective**, or one-to-one, if each element in the domain maps to a unique element in the image. A function is **surjective**, or onto, if the image and codomain are identical. A **bijective** functions are functions that are both injective and surjective.

For more visual representation, below is a diagram that manifests the definition above.

Injective Function	**Surjective Function**	**Bijective Function**
If $f(x_1) = f(x_2)$, then $x_1 = x_2$	$\forall\, y_i \in Y,\ \exists\, x_j \in X$ $s.t.\ f(x_j) = y_i$	$\forall\, y_i \in Y,\ \exists!\, x_j \in X$ $s.t.\ f(x_j) = y_i$



Below is a fun problem on bijectivity.

Exercise 7.2.4

Show that $f : \mathbb{R} \to \mathbb{R}$ is bijective if $f(f(x)) = x$.

Solution.

First, the injectivity can be proven. Let $f(a) = f(b)$ and consider the following equation.

$$a = f(f(a)) = f(f(b)) = b$$

Therefore, if $f(a) = f(b)$, then $a = b$ and the function is injective. Continuing, let $f(x) = y$. Substituting, $f(y) = x$ is true for all $x \in \mathbb{R}$. Because the image is $\mathbb{R}$, the function is bijective.

Now that we reviewed about functions, let's discuss linear transformation.

7.2.2 Basic Terms and Notations

If we have functions over different domains, can we have a function whose domain and range are vector spaces? The answer is yes, and we have a special name for such functions.

Definition 7.2.5

A function whose domain and range are defined over vector spaces is known as a **transformation**, denoted as T.

A transformation that maps a vector space V to W is denoted as $T : V \to W$. Just like how we have $f(x)$, we do the same like $T(v)$ for $v \in V$, but it is more common to write Tv. Continuing, the definitions for image, injection, surjection, and bijection all apply in similar fashion.

There are many different transformations, but some transformation moves the vector space in "parallel" in a way that preserves the fundamental definition of vector addition and scalar multiplication.

Definition 7.2.6

A **linear transformation** is a transformation that preserves the definition of vector addition and scalar multiplication of the input vector space. In other words, for scalars α and β and input vectors v_1 and v_2, a transformation T is linear if the following equation holds.

$$T(\alpha v_1 + \beta v_2) = \alpha T(v_1) + \beta T(v_2)$$

One way to think of the equation above is preservation of the **linear** combination. Let's take a look at an example.

Exercise 7.2.7

Show that a transformation defined as $T(v) = 2v$ is linear.

Solution.

Consider the following equation for some scalars α and β.

$$T(\alpha v_1 + \beta v_2) = 2(\alpha v_1 + \beta v_2) = 2\alpha v_1 + 2\beta v_2 = \alpha T(v_1) + \beta T(v_2)$$

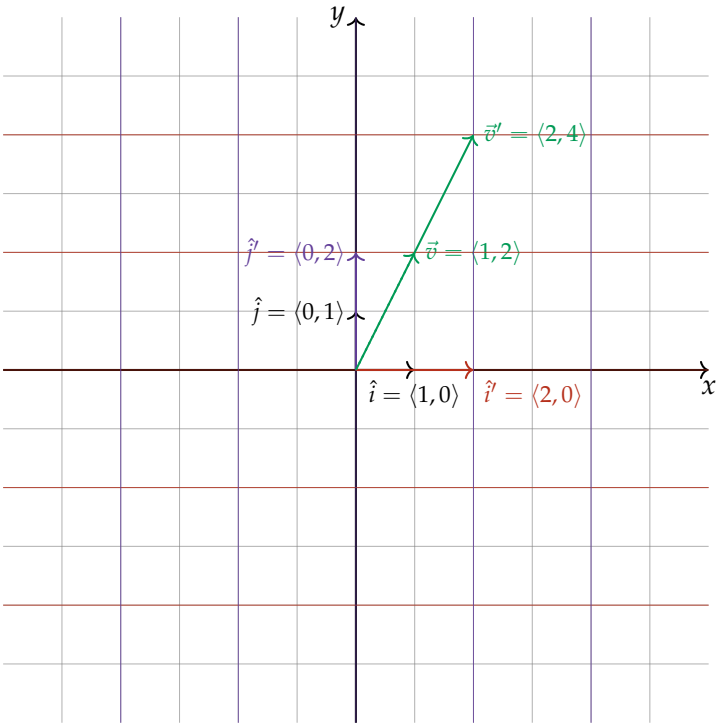
Thus, the transformation is linear.

As you can see above, if $T : \mathbb{R}^2 \to \mathbb{R}^2$, you can imagine the vectors dilating. Indeed, such linear transformations are called **dilation**

Definition 7.2.8

A **dilation** is a linear transformation that scales the vectors in the input vector space defined as $Tv = kv$ for some scalar k.

Let's take a look at the visual representation of the simple version of the example above. Consider the transformation defined as $T(\langle a, b\rangle) = \langle 2a, 2b\rangle$. We know from the previous example that the transformation is linear. Now let's try and graph it.



There are other special linear transformations that output the same vector space or the zero vector space.

Definition 7.2.9

A **zero transformation** is a linear transformation defined as $Tv = \mathbf{0}$. An **identity transformation** is a linear transformation that returns the same input vector as the output.

Let's take a look at a few more examples on linear transformations.

Exercise 7.2.10

Determine whether a transformation defined as $Tv = v^2$ is linear.

Solution.

Consider the following equation for arbitrary scalars α and β.

$$T(\alpha v_1 + \beta v_2) = (\alpha v_1 + \beta v_2)^2$$

Notice that the following equation holds.

$$= \alpha T(v_1) + \beta T(v_2) = \alpha v_1^2 + \beta v_2^2$$

Because $(\alpha v_1 + \beta v_2)^2 \neq \alpha v_1^2 + \beta v_2^2$ in general, the transformation is not linear.

Below is another example.

Exercise 7.2.11

Determine whether a transformation defined as $Ty = \frac{dy}{dx}$ for some differentiable function $y = f(x)$ is linear.

Solution.

Consider the following equation for some scalars α and β.

$$\begin{aligned} T(\alpha y_1 + \beta y_2) &= \frac{d}{dx}(\alpha y_1 + \beta y_2) = \frac{d}{dx}(\alpha y_1) + \frac{d}{dx}(\beta y_2) \\ &= \alpha \frac{d}{dx} y_1 + \beta \frac{d}{dx} y_2 = \alpha T(y_1) + \beta T(y_2) \end{aligned}$$

Thus, the transformation is linear.

One thing that we can notice from the problem above is that the transformation is not linear since changing the constants of an input will not change the output. The transformation is also not surjective as not all functions can be represented as the derivative of other functions. Continuing, we can consider the following transformation.

Exercise 7.2.12

Determine whether a transformation defined as $T(\langle a, b\rangle) = \langle -a, -b\rangle$ is linear.

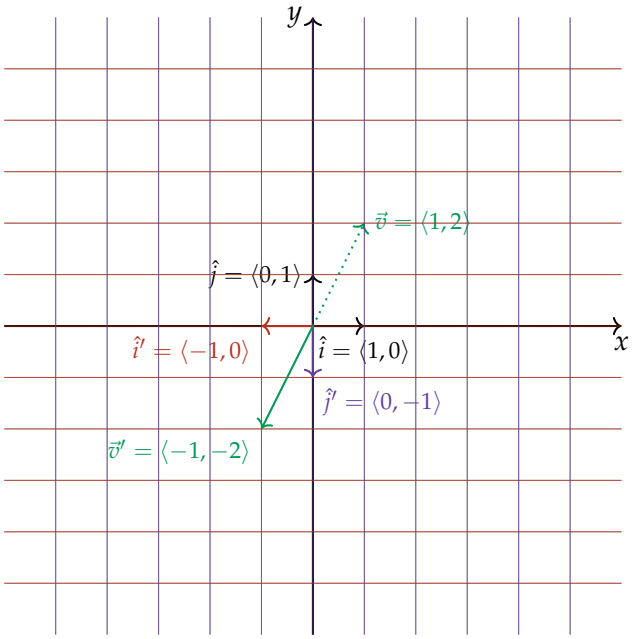
Solution.

Consider the following equation for arbitrary scalars α and β.

$$\begin{aligned} T(\alpha\langle a, b\rangle + \beta\langle a', b'\rangle) &= T(\langle \alpha a, \alpha b\rangle + \langle \beta a', \beta b'\rangle) = T(\langle \alpha a + \beta a', \alpha b + \beta b'\rangle) \\ &= \langle -\alpha a - \beta a', -\alpha b - \beta b'\rangle = \langle -\alpha a, -\alpha b\rangle + \langle -\beta a', -\beta b'\rangle \\ &= \alpha\langle -a, -b\rangle + \beta\langle -a', -b'\rangle = \alpha T(v_1) + \beta T(v_2) \end{aligned}$$

Thus, the transformation is linear.

Let's try and graph this to visually see how the vector space is transformed.



This transformation reflects the vectors across the origin! Now that we discussed what a linear transformation is, let's see how matrices relate to it.

7.2.3 Matrices and Linear Transformation

Recall that every vector in a vector space can be represented as a linear combination of the basis. We will apply our understanding of a linear transformation to vector representations in a vector space to find the relation.

Consider a basis $S = \{v_1, \ldots, v_n\}$ for a vector space V. Notice that by definition, an arbitrary vector $v \in V$ can be represented as a linear combination of a basis as shown below (where c_k are constants).

$$v = \sum_{k=1}^{n} c_k v_k$$

Therefore, the following equation holds.

$$(v)_S = \begin{bmatrix} c_1 \\ \vdots \\ c_n \end{bmatrix}$$

Now returning to the definition of a linear transformation, we can consider the following equation.

$$Tv = T\left(\sum_{k=1}^{n} c_k v_k\right) = \sum_{k=1}^{n} c_k T v_k$$

Notice that the coordinate is preserved. In other words, the following equation holds for the new basis $S' = \{Tv_1, \ldots, Tv_n\}$

$$(Tv)_{S'} = \begin{bmatrix} c_1 \\ \vdots \\ c_n \end{bmatrix}$$

This could let us geometrically understand that a linear transformation "preserves" the general shape of a vector space since the new vectors have the same coordinates as the original ones, but with different basis. With this in mind, we can establish two important theorems.

Theorem 7.2.13

For a vector space V and a basis $S = \{v_1, \ldots, v_n\}$, the transformation $T : V \to \mathbb{R}^n$ defined as $Tv = (v)_S$ is linear and bijective.

Proof.

First, the linearity of the transformation could be proven. Consider the following equations for $u_1, u_2 \in V$ and some constants c_k and d_k.

$$\begin{aligned} T(\alpha u_1 + \beta u_2) &= T\left(\alpha \sum_{k=1}^{n} c_k v_k + \beta \sum_{k=1}^{n} d_k v_k\right) = T\left(\sum_{k=1}^{n} (\alpha c_k + \beta d_k) v_k\right) \\ &= \begin{bmatrix} \alpha c_1 + \beta d_1 \\ \alpha c_2 + \beta d_2 \\ \vdots \\ \alpha c_n + \beta d_n \end{bmatrix} = \begin{bmatrix} \alpha c_1 \\ \alpha c_2 \\ \vdots \\ \alpha c_n \end{bmatrix} + \begin{bmatrix} \beta d_1 \\ \beta d_2 \\ \vdots \\ \beta d_n \end{bmatrix} = \alpha \begin{bmatrix} c_1 \\ c_2 \\ \vdots \\ c_n \end{bmatrix} + \beta \begin{bmatrix} d_1 \\ d_2 \\ \vdots \\ d_n \end{bmatrix} \\ &= \alpha T(u_1) + \beta T(u_2) \end{aligned}$$

Thus, T is linear. Continuing, notice that if $T(u_1) = T(u_2)$, then $(u_1)_S = (u_2)_S$ and $u_1 = u_2$. Therefore, the transformation is injective. The transformation is also surjective as the entries c_k in the coordinates are arbitrary real numbers. Thus, T is linear and bijective.

Continuing, below is an important theorem and definition.

Theorem 7.2.14

For vector spaces V and W with basis $B = \{v_1, \ldots, v_n\}$ and $C = \{w_1, \ldots, w_m\}$ respectively, there exists a matrix A such that the following equation holds for a linear transformation $T : V \to W$ and arbitrary $v \in V$.

$$(Tv)_C = A(v)_B$$

Proof.

First and foremost, consider the following equation for any $Tv_j \in W$ and constant a_{ij}.

$$Tv_j = \sum_{i=1}^{m} a_{ij} w_i$$

Let $A = [a_{ij}]$ and consider the following equation.

$$A(v_2)_B = A \begin{bmatrix} 0 \\ 1 \\ \vdots \\ 0 \end{bmatrix} = \begin{bmatrix} a_{12} \\ a_{22} \\ \vdots \\ a_{m2} \end{bmatrix}$$

Generalizing, $(Tv_j)_C = A(v_j)_B$ and the following equations hold for constants c_j.

$$\begin{aligned} (Tv)_C &= \left(\sum_{j=1}^{n} c_j T v_j \right)_C = \sum_{j=1}^{n} (c_j T v_j)_C = \sum_{j=1}^{n} c_j (Tv_j)_C \\ &= \sum_{j=1}^{n} c_j A(v_j)_B = A \left(\sum_{j=1}^{n} (c_j v_j)_B \right) = A \left(\sum_{j=1}^{n} c_j v_j \right)_B \\ &= A(v)_B \end{aligned}$$

Thus, there always exists such a matrix A.

The matrix A above has a special name. The matrix is known as the matrix representation of T with respect to B and C, denoted as A_B^C. In short, we also say A from B to C.

Continuing from the theorem, we can also notice that for a vector space V and its basis A and B, there exists a matrix P_A^B such that $(v)_B = P_A^B (v)_A$ holds for all $v \in V$. To prove this, consider the identity transformation $T : V \to V$. By theorem 7.2.14, there exists a matrix P such that $(v)_B = P_A^B (v)_A$. This matrix too has a special name.

Definition 7.2.15

A **transition matrix** is a matrix that converts the coordinate representation of a vector in a vector space V from a basis in V to another basis in V.

Now from the definitions, we could notice that linear transformations are highly related to the basis of a vector space. If we know how the basis is mapped, we can understand the linear transformation. Consider the following theorem.

Theorem 7.2.16

For a vector space and its bases A and B, the coordinate representation of the j vector of A with respect to B is the j^{th} column of the transition matrix P_A^B.

Proof.

Let $A = \{a_1, \ldots, a_n\}$ and $B = \{b_1, \ldots, b_n\}$. For some scalars p_{ij}, a_i can be written as $a_j = \sum_{i=1}^{n} p_{ij} b_i$. Let a vector v in the vector space be written as $v = \sum_{k=1}^{n} a'_k + a_k = \sum_{k=1}^{n} b'_k + b_k$. Continuing, the following equation holds.

$$v = \sum_{j=1}^{n} a'_j a_j = \sum_{j=1}^{n} a'_j \left(\sum_{i=1}^{n} p_{ij} b_i \right) = \sum_{i=1}^{n} \left(\sum_{j=1}^{n} p_{ij} a'_j \right) b_i$$

In other words, $b'_i = \sum_{j=1}^{n} p_{ij} a'_j$ and $(v)_B = [p_{ij}](v)_A = P_A^B (v)_A$.

Continuing with the theorem above, we can derive the following.

Theorem 7.2.17

A transition matrix P_A^B is always invertible and $(P_A^B)^{-1} = P_B^A$.

Proof.

Consider the following equations for a vector v.

$$(v)_B = P_A^B (v)_A$$
$$(v)_A = P_B^A (v)_B$$

Therefore, $(v)_B = P_A^B (v)_A = P_A^B P_B^A (v)_B$ for all $(v)_B$. Thus, $P_A^B P_B^A = I$ and the theorem holds.

Now how about matrix representations like A_S^S? We have an interesting theorem that links such matrices with transition matrices.

Theorem 7.2.18

Let A_B^B and A_C^C be the matrix representations of linear transformations for bases B and C. If P_B^C represents the transition matrix, then $A_B^B = P_C^B A_C^C P_B^C$.

Proof.

For a vector v in the same vector space, notice that $(Tv)_B = A_B^B (v)_B$ and $(v)_C = P_B^C (v)_B$ hold by definition. Consider the following equation.

$$(Tv)_B = P_C^B (Tv)_C = P_C^B A_C^C (v)_C = P_C^B A_C^C P_B^C (v)_B = A_B^B (v)_B$$

Thus, $P_C^B A_C^C P_B^C = A_B^B$.

Let's use A for A_B^B, $\tilde{A}$ for A_C^C, and P for P_B^C where $P^{-1} = P_C^B$. Notice that we can write $A = P^{-1} \tilde{A} P$ and $PA = \tilde{A} P$. Here, we say that A and $\tilde{A}$ are **similar**.

Definition 7.2.19

Two matrices A and B are **similar** if the linear transformation that they represent with different bases is identical. In other words, for transition matrix $P = P_A^B$, $PA = BP$.

Continuing with the definition, we can establish the following theorem.

Theorem 7.2.20

For a vector space V with basis B, let A be the matrix representation of the linear transformation $T : V \to V$ from B to B. If A is similar to another matrix representation $\tilde{A}$, then there exists a basis C such that $\tilde{A}$ represents the same linear transformation from C to C.

Proof.

First and foremost, let a transition matrix from a basis C to B be P and let v be a vector in the vector space. Consider the following equations.

$$\begin{aligned}(Tv)_B &= A(v)_B = AP(v)_C \\ &= P(Tv)_C\end{aligned}$$

Therefore, $AP(v)_C = P(Tv)_C$ and $(Tv)_C = A'(v)_C = P^{-1}AP(v)_C$. In other words, for each basis C, there exists a transition matrix similar to A. Moreover, because the whole process is invertible, the theorem holds.

Now that we discussed that matrices can be used to represent linear transformations, let's discuss about topics that are never left out in linear algebra.

7.2.4 Kernels and Images

As you could have noticed, sometimes it is easier to discuss a linear transformation with intuitions and sometimes matrices are much more straightforward. Recall that linear transformations are functions for vector spaces. Sometimes it is easier to look at the functions directly and sometimes it is more convenient to look at the matrix representation of such functions. That is what we are going to do with kernels and images. As always, let's get started with definitions.

Definition 7.2.21

The **kernel** of a linear transformation $T : V \to W$, also known as the **nullspace** of T, is the set of vectors in V that are mapped to zero vectors when transformed with T.

$$\ker(T) = \{v \in V \mid Tv = \mathbf{0}\}$$

Continuing with the definition, we can establish two fundamental properties of the kernel of a transformation.

Theorem 7.2.22

Consider the following statements for all linear transformations $T : V \to W$.

1. $\mathbf{0} \in \ker(T)$
2. $\ker(T)$ is a subspace of V.

Let's start by proving the first statement. The proof is rather straightforward with the properties of linear transformation.

Proof.

Consider the following equation.

$$T(\mathbf{0}) = T(0\mathbf{0}) = 0T(\mathbf{0}) = \mathbf{0}$$

Because $\mathbf{0}$ is mapped to $\mathbf{0}$ after any linear transformation T, $\mathbf{0} \in \ker(T)$ for all linear transformations T.

Continuing, below is the proof for the second statement.

Proof.

By definition, $\ker(T) \in V$. Therefore, it suffices to show that the kernel of T is closed under vector addition and scalar multiplication. Because $\ker(T)$ is not empty by the first statement, consider the following equations for $u, v \in \ker(T)$ and some scalars α and β.

$$T(\alpha u + \beta v) = \alpha Tu + \beta Tv = \alpha\mathbf{0} + \beta\mathbf{0} = \mathbf{0}$$

In other words $\alpha u + \beta v \in \ker(T)$ and $\ker(T)$ is a subspace of V.

Recall that a linear transformation can be represented as some matrix A. Therefore, kernel of A can be interpreted as the set of solutions to $Ax = \mathbf{0}$. Notice that some solutions retain finite and infinite number of vectors. Therefore, we can establish the following definition.

Definition 7.2.23

The **nullity** of T, denoted as $\text{null}(T)$, is the dimension of $\ker(T)$ for some linear transformation T.

Continuing, we can define the image of a transformation, which should be unsurprising.

Definition 7.2.24

The subset of the codomain W for a transformation $T : V \to W$ that contains all output vectors is known as the **image** of T. Below defines $\Im(T)$ with $v \in V$.

$$\Im(T) = \{w \in W \mid w = Tv\}$$

Although the kernel and the image of a transformation are very different, we can establish very similar statements for the image just as how we derived them for the kernel.

Theorem 7.2.25

Consider the following statements for all linear transformations $T : V \to W$.

1. $\mathbf{0} \in \Im(T)$
2. $\Im(T)$ is a subspace of W.

Let's start by proving the first statement.

Proof.

By theorem 7.2.22, there exists a vector in V such that it is mapped to $\mathbf{0}$ in W. Thus, the statement holds.

Continuing, below is the proof for the second statement.

Proof.

First and foremost, $\Im(T) \in W$ by definition. Therefore, it suffices to show that the image is closed under vector addition and scalar multiplication. Consider the following equation for $v_1, v_2 \in V$, $Tv_1 = w_1$, $Tv_2 = w_2$, and some scalars α and β.

$$\alpha w_1 + \beta w_2 = \alpha T v_1 + \beta T v_2 = T(\alpha v_1 + \beta v_2)$$

Because $T(\alpha v_1 + \beta v_2) \in \Im(T)$, $\alpha w_1 + \beta w_2 \in \Im(T)$ and the statement holds.

Now recall the kernel of a matrix A is the set of all solutions to the equation $Ax = \mathbf{0}$. Similarly, the image is the set of all possible outcomes of the product Ax for some $x \in V$. Consider the equations below.

$$\begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{m1} & a_{m2} & \cdots & a_{mn} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix} = \begin{bmatrix} \sum_{k=1}^{n} a_{1k} x_k \\ \sum_{k=1}^{n} a_{2k} x_k \\ \vdots \\ \sum_{k=1}^{n} a_{mk} x_k \end{bmatrix} = \sum_{k=1}^{n} x_k \begin{bmatrix} a_{1k} \\ a_{2k} \\ \vdots \\ a_{mk} \end{bmatrix}$$

In other words, the image of A is the column space of A. Therefore, we can establish the following definition.

Definition 7.2.26

The **rank** of a transformation T, denoted as $\mathrm{r}(T)$, is $\dim(\Im(T))$.

Continuing from the definition and ideas of column space and the image, we can derive the following theorem.

Theorem 7.2.27

Let B and C be n-dimensional and m-dimensional bases for V and W in a linear transformation $T : V \to W$ respectively. If the matrix representation of T is $A = A_B^C$, then $\mathrm{r}(T) = \mathrm{r}(A)$.

Proof.

First and foremost, let a basis for $\Im(T)$ be $\{w_1, \ldots, w_k\}$. Notice that if $\{(w_1)_C, \ldots, (w_k)_C\}$ is a basis for $\mathrm{col}(A)$, then $\dim(\Im(T)) = \dim(\mathrm{col}(A))$ and $\mathrm{r}(T) = \mathrm{r}(A)$. Thus, it suffices to show that $\{(w_1)_C, \ldots, (w_k)_C\}$ is a basis for $\mathrm{col}(A)$.

Let c_k be some scalars that $x \in \mathrm{col}(A)$ is represented as the linear combination of the columns A_k of A where $x = \sum_{k=1}^{n} c_k A_k$. Note that c_k can be written as an n-tuple. Thus, let $v \in V$ be a vector such that the following equation holds.

$$(v)_B = \begin{bmatrix} c_1 \\ c_2 \\ \vdots \\ c_n \end{bmatrix}$$

Continuing, $x = A(v)_B = (Tv)_C$. Because $Tv \in \Im(T)$, it is evident that x lies in $\mathrm{span}\{(w_1)_C, \ldots, (w_k)_C\}$. Moreover, because x is an arbitrary vector from $\mathrm{col}(A)$, it is evident that $\mathrm{col}(A) \subset \mathrm{span}\{(w_1)_C, \ldots, (w_k)_C\}$ and $\{(w_1)_C, \ldots, (w_k)_C\}$ spans $\mathrm{col}(A)$.

By definition, $\{w_1, \ldots, w_k\}$ is linearly independent. For some scalars d_k, for $\sum_{k=1}^{n} d_k (w_k)_C$ to be a zero vector, $(\sum_{k=1}^{n} d_k w_k)_C = \mathbf{0}$ and $\sum_{k=1}^{n} d_k w_k = \mathbf{0}$, which only happens when $d_k = 0$ for all integer $k \in [0, n]$. Thus, $\{(w_1)_C, \ldots, (w_k)_C\}$ is linearly independent and is a basis for $\mathrm{col}(A)$.

As you can see above by the theorem, although the kernel and the image of a linear transformation appear to be totally unrelated, they are somehow related. Below is another theorem that connects the bases of the kernel and the image of a linear transformation.

Theorem 7.2.28

Let $\{v_1, \ldots, v_k\}$ be the basis of the kernel of a linear transformation $T : V \to W$. If $\{v_1, \ldots, v_k, \ldots v_n\}$ is a basis for V, then a basis of $\Im(T)$ is $\{Tv_{k+1}, \ldots, Tv_n\}$.

Proof.

To prove that $\{Tv_{k+1}, \ldots, Tv_n\}$ is a basis of $\Im(T)$, it suffices to show that the set of vectors is linearly independent and spans $\Im(T)$. First, to show that the set is linearly independent,

consider the following equations for some scalars c_i.

$$\sum_{i=k+1}^{n} c_i T v_i = T\left(\sum_{i=k+1}^{n} c_i v_i\right) = \mathbf{0}$$

In other words, $\sum_{i=k+1}^{n} c_i v_i \in \ker(T)$ and it can be written as the linear combination of the basis, or $\sum_{i=k+1}^{n} c_i v_i = \sum_{i=1}^{k} c_i v_i$. By definition, because $\{v_1, \ldots, v_k, \ldots v_n\}$ is linearly independent, c_i must be all zero for integer $i \in [1, n]$ for $\sum_{i=k+1}^{n} c_i v_i - \sum_{i=1}^{k} c_i v_i = \mathbf{0}$ to satisfy. Thus, $\{Tv_{k+1}, \ldots, Tv_n\}$ is linearly independent.

Continuing, the fact that the set $\{Tv_{k+1}, \ldots, Tv_n\}$ spans $\Im(T)$ can be proven. Let $w = Tv$ for any $w \in \Im(T)$. Consider the following equations below.

$$\begin{aligned} w = Tv &= T\left(\sum_{i=1}^{n} c_i v_i\right) = \sum_{i=1}^{n} c_i T v_i \\ &= \sum_{i=1}^{k} c_i T v_i + \sum_{i=k+1}^{n} c_i T v_i = \sum_{i=k+1}^{n} c_i T v_i \in \text{span}\{Tv_{k+1}, \ldots, Tv_n\} \end{aligned}$$

In other words, $\{Tv_{k+1}, \ldots, Tv_n\}$ spans $\Im(T)$. Thus, the theorem holds.

Continuing from this theorem, we can establish another interesting corollary as a theorem.

Theorem 7.2.29

For a finite dimensional vector space V, the following equation is true for a linear transformation $T : V \to W$.

$$\text{r}(T) + \text{null}(T) = \dim(V)$$

Proof.

Let $k = \dim(\ker(T))$ and $n = \dim(V)$. By Theorem 7.2.28, there exists a basis for $\Im(T)$ with $n - k$ vectors, or $\text{r}(T) = \dim(\Im(T)) = n - k$. Thus, $\text{r}(T) + \text{null}(T) = n - k + k = n = \dim(V)$.

Moving on from the relationship with the kernel and the image of a linear transformation, let's discuss about injection, surjection, and bijection. Consider the following theorem.

Theorem 7.2.30

A linear transformation $T : V \to W$ is injective if and only if $\ker(T) = \{\mathbf{0}\}$.

Proof.

By definition, T is injective if $v_1, v_2 \in V$ such that $v_1 \neq v_2$ implies $Tv_1 \neq Tv_2$. Therefore, if $v \in \ker(T)$, then $Tv = \mathbf{0} = T\mathbf{0}$ and $\ker(T)$ must contain no nonzero vectors for T to be injective. Thus, $\ker(T)$ must only contain the zero vector.

Continuing, below is another theorem on injection and surjection.

Theorem 7.2.31

For a linear transformation $T : V \to W$ with $\dim(V) = \dim(W) = n$, T is injective if and only if T is surjective.

Proof.

By Theorem 7.2.30, T is injective if and only if $\ker(T) = \{\mathbf{0}\}$. Moreover by definition, $\ker(T) = \{\mathbf{0}\}$ if and only if $\text{null}(T) = 0$ and $\text{r}(T) = \dim(V) - \text{null}(T) = n$ by Theorem 7.2.29. Continuing by definition, $\text{r}(T) = n$ if and only if $\dim(\Im(T)) = n = \dim(W)$. Therefore, T is injective if and only if T is surjective.

With the interesting properties, we can assign a special name for bijective linear transformation.

Definition 7.2.32

A bijective linear transformation is called an **isomorphism**. Moreover, a vector space V is said to be **isomorphic** to another vector space W if there exists an isomorphism $T : V \to W$, denoted as $V \cong W$.

Continuing from the definition, we can derive the following properties.

Theorem 7.2.33

For vector spaces U, V, and W, isomorphism is reflexive, symmetric, and transitive.

1. $U \cong U$
2. If $U \cong V$, then $V \cong U$.
3. If $U \cong V$ and $V \cong W$, then $U \cong W$.

Let's start by proving the first statement.

Proof.

Notice that it suffices to show that the identity transformation is bijective. By definition, because each vector is mapped to itself, identity transformation is bijective and $U \cong U$.

Below is the proof for the second property.

Proof.

First and foremost, notice that if there exists $T : U \to V$ that is bijective, then there exists its inverse $T^{-1} : V \to U$ that is also bijective. Continuing, consider the equation below for some vectors $v_1, v_2 \in V$ and scalars α and β.

$$\begin{aligned} \alpha T^{-1}v_1 + \beta T^{-1}v_2 &= T^{-1}\left(T\left(\alpha T^{-1}v_1 + \beta T^{-1}v_2\right)\right) \\ &= T^{-1}\left(\alpha T\left(T^{-1}v_1\right) + \beta T\left(T^{-1}v_2\right)\right) \\ &= T^{-1}\left(\alpha v_1 + \beta v_2\right) \end{aligned}$$

Therefore, T^{-1} is linear and T^{-1} is an isomorphism.

Continuing, below is the proof for the last statement.

Proof.

First and foremost, let $T_1 : U \to V$ and $T_2 : V \to W$ be isomorphisms. Therefore, it suffices to show that $T_2 \circ T_1$ is an isomorphism. Notice that because T_1 and T_2 are bijective, $T_2 \circ T_1$ must also be bijective. Moreover, consider the following equation for some vectors $v_1, v_2 \in V$ and scalars α and β.

$$\begin{aligned}(T_2 \circ T_1)(\alpha v_1 + \beta v_2) &= T_2(\alpha T_1 v_1 + \beta T_1 v_2) \\ &= \alpha T_2(T_1 v_1) + \beta T_2(T_1 v_2) \\ &= \alpha(T_2 \circ T_1)(v_1) + \beta(T_2 \circ T_1)(v_2)\end{aligned}$$

Thus, $T_2 \circ T_1$ is linear and the statement is true.

As you can see above, the three statements in Theorem 7.2.33 each shows a relation called reflexive, symmetric, and transitive. Continuing from this terms, we can assign a special name for a relation.

Definition 7.2.34

An **equivalence relation** is a relation that is reflexive, symmetric, and transitive.

Below is another interesting theorem on isomorphism.

Theorem 7.2.35

For finite dimensional vector spaces V and W, $V \cong W$ if and only if $\dim(V) = \dim(W)$.

Proof.

First, the statement if $\dim(V) = \dim(W)$, then $V \cong W$ can be proven. Notice that the coordinate representations of infinitely many vectors in V and W are unique. Therefore, $V \cong \mathbb{R}^n$ and $W \cong \mathbb{R}^n$. By Theorem 7.2.33, $\mathbb{R}^n \cong W$ and $V \cong W$.

Continuing, if $V \cong W$, then there exists a linear transformation $T : V \to W$ that is bijective. By Theorem 7.2.30, $\ker(T) = \{\mathbf{0}\}$ and $\text{null}(T) = 0$. Moreover by definition, because T is surjective, $\text{r}(T) = \dim(\Im(T)) = \dim(W)$. Finally by Theorem 7.2.29, $\dim(W) + 0 = \dim(V)$ and the theorem holds.

From the theorem above, we can notice that if we consider a bijective linear transformation that maps to coordinate representation, then all n-dimensional vector spaces are isomorphic to $\mathbb{R}^n$. In other words, every n-dimensional vector space over a given field for some finite n can be considered as the same vector space, but just with different representations.

This is it for linear transformations, and let's discuss about eigenvalues and eigenvectors!

§7.3 Eigenvalues and Eigenvectors

As always, let's start with definitions.

Definition 7.3.1

For a linear transformation $T : V \to V$ with a nonzero vector $v \in V$ and some scalar λ,if $Tv = \lambda v$, then v is an **eigenvector** of T and λ is an **eigenvalue** of T.

From here, it is important to note that while v must be a nonzero vector, λ could equal zero. Let's take a look at an example.

Exercise 7.3.2

Let $T : \mathbb{R}^2 \to \mathbb{R}^2$ be a linear transformation defined as $T(a, b) = (a, -b)$. Find an eigenvector and eigenvalue of T.

Solution.

Consider the following equation.

$$T(0, 1) = (0, -1) = -1 \cdot (0, 1)$$

Therefore, $(0, 1)$ is an eigenvector of T and -1 is the corresponding eigenvalue of T.

Below is another example for a linear transformation $T : \mathbb{R}^2 \to \mathbb{R}^2$.

Exercise 7.3.3

Find, if any, eigenvectors and eigenvalues of $T : \mathbb{R}^2 \to \mathbb{R}^2$ defined as $T(a, b) = (b, -a)$.

Solution.

For some scalar λ, consider the following equation.

$$T(a, b) = (b, -a) = \lambda(a, b) = (\lambda a, \lambda b)$$

In other words $b = \lambda a$ and $-a = \lambda b$. Substituting, $-a = \lambda^2 a$ and $(\lambda^2 + 1)a = 0$. Because $\lambda \in \mathbb{R}$, $a = 0$ and $b = 0$. However, because eigenvectors are nonzero vectors by definition, T has not real eigenvalues and eigenvectors.

Now eigenvectors and eigenvalues are important because they make our lives easier for investigating matrix representation of a linear transformation. Consider the following theorem.

Theorem 7.3.4

For a linear transformation $T : V \to V$ for a finite dimensional vector space V, there exists a basis whose matrix representation is a diagonal matrix if and only if there exists a basis composed of only eigenvectors of T.

Proof.

First, let $B = \{v_1, \ldots, v_n\}$ be a basis for V and let the matrix representation of T be a diagonal matrix A whose diagonal entries are $\lambda_1, \ldots, \lambda_n$. Thus

$$(Tv_j)_B = \begin{bmatrix} 0 \\ \vdots \\ \lambda_j \\ \vdots \\ 0 \end{bmatrix}$$

hold by construction and $Tv_j = \lambda_j v_j$. In other words, v_j and λ_j are an eigenvector and an eigenvalue of T and there exists a basis composed of only eigenvectors of T.

Continuing, let $B = \{v_1, \ldots, v_n\}$ be a basis for V whose elements are all eigenvectors of T. Therefore, $Tv_j = \lambda v_j$ and the following equation holds.

$$(Tv_j)_B = \begin{bmatrix} 0 \\ \vdots \\ \lambda_j \\ \vdots \\ 0 \end{bmatrix}$$

Therefore, there exists a basis whose matrix representation is a diagonal matrix.

With this theorem and connecting vector spaces with matrix again, we can redefine eigenvectors and eigenvalues with different interpretations.

Definition 7.3.5

For a square matrix $A_{n\times n}$ and a nonzero n-tuple $x \in \mathbb{R}^n$, if there exists a scalar λ such that $Ax = \lambda x$, then x is an **eigenvector** and λ is an **eigenvalue** of A.

When you think about this definition, we can see that both definitions imply the same thing as the A is the matrix representation of a linear transformation and x is a vector in the corresponding vector space. Let's take a look at a quick example.

Exercise 7.3.6

Find an eigenvector and eigenvalue of matrix A defined as the following.

$$A = \begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix}$$

Solution.

Consider the following equation.

$$\begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix} \begin{bmatrix} 0 \\ 1 \end{bmatrix} = \begin{bmatrix} 0 \\ -1 \end{bmatrix} = -1 \cdot \begin{bmatrix} 0 \\ 1 \end{bmatrix}$$

Thus, -1 is an eigenvalue and $[01]$ is an eigenvector.

If this example looks familiar, this is actually the same question as the first example from this section. Continuing, below are two interesting theorems on eigenvectors and eigenvalues.

Theorem 7.3.7

For a linear transformation $T : V \to V$, $v \in V$ is an eigenvector with the corresponding eigenvalue λ if and only if $(v)_B$ is an eigenvector with the corresponding eigenvalue λ for A_B^B, the matrix representation of T for any basis B.

Proof.

Notice that $Tv = \lambda v$ if and only if $A_B^B(v)_B = (Tv)_B = (\lambda v)_B = \lambda(v)_B$. Thus, the theorem holds.

This theorem will come in handy as we can either find all eigenvectors and corresponding eigenvalues of the transformation or its matrix representation. Continuing, below is another theorem.

Theorem 7.3.8

For a linear transformation $T : V \to V$ and eigenvalue λ, $S_\lambda = \{v \in V | Tv = \lambda v\}$ is a subspace of V.

Proof.

First, notice that because $S_\lambda \subset V$, it suffices to show that the set is closed under vector addition and scalar multiplication. Consider the following equations for $u, v \in S_\lambda$ and some scalars α and β.

$$\begin{aligned} T(\alpha u + \beta v) &= \alpha Tu + \beta Tv = \alpha\lambda u + \beta\lambda v \\ &= \lambda(\alpha u + \beta v) \end{aligned}$$

In other words, $\alpha u + \beta v \in S_\lambda$ and the theorem holds.

As you may have guessed, we can establish a new definition from the theorem above.

Definition 7.3.9

For a transformation $T : V \to V$ and its eigenvalue λ, $S_\lambda = \{v \in V | Tv = \lambda v\}$ defined as the **eigenspace** of T for the eigenvalue λ.

Just as we redefined eigenvectors and eigenvalues with matrices, we could do the same for eigenspace. Now that we discussed what eigenvectors and eigenvalues are, let's discuss how to find one.

Theorem 7.3.10

For a square matrix $A_{n\times n}$, λ is its eigenvalue if and only if $\det(A - \lambda I_n) = 0$.

Proof.

First notice that by definition, λ is an eigenvalue of A if and only if $Ax = \lambda x$ holds for some nonzero vector x. Continuing, $Ax = \lambda x$ holds if and only if $Ax = \lambda I_n x$ and $(A - \lambda I_n)x = 0$. In other words, the columns of the matrix $A - \lambda I_n$ are linearly dependent and invertible. Therefore by Theorem 5.5.13, the theorem holds.

Being one of the most important theorems in linear algebra, we can establish another definition from the theorem.

Definition 7.3.11

The **characteristic equation** of a square matrix A is defined as $\det(A - \lambda I_n) = 0$ for scalar variable λ.

An important thing to note from this definition is that $\det(A - \lambda I_n)$ is a polynomial in λ with degree n. Moreover, the real roots of the polynomial can be interpreted as eigenvalues of A. Let's take a look at an example of how the characteristic equation can be used to find all eigenvalues of a matrix.

Exercise 7.3.12

Find all eigenvalues and the corresponding eigenspaces of matrix $A = \begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix}$.

Solution.

To find all eigenvalues of A, it suffices to find all solutions for the characteristic equation of A by Theorem 7.3.10.

$$A - \lambda I_2 = \begin{bmatrix} 1-\lambda & 0 \\ 0 & -1-\lambda \end{bmatrix}$$

Therefore, $\det(A - \lambda I_2) = (1 - \lambda)(-1 - \lambda)$ and $\lambda = 1$ or $\lambda = -1$ are the solutions to the characteristic equation.

Continuing, the corresponding eigenspaces could be found. For $\lambda = 1$, the eigenspace

is defined as $S_1 = \{x | Ax = x\}$. Solving for the equation $(A - I)x = 0$, the following equations hold.

$$\begin{bmatrix} 0 & 0 \\ 0 & -2 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = 0$$
$$-2x_2 = 0$$

Therefore, the eigenspace for the eigenvalue 1 can be represented as the following

$$S_1 = \text{span} \left\{ \begin{bmatrix} 1 \\ 0 \end{bmatrix} \right\}$$

Similarly, $S_{-1} = \{x | Ax = -x\}$ and $(A + I)x = 0$ could be solved.

$$\begin{bmatrix} 2 & 0 \\ 0 & 0 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = 0$$
$$2x_1 = 0$$

Therefore,

$$S_{-1} = \text{span} \left\{ \begin{bmatrix} 0 \\ 1 \end{bmatrix} \right\}$$

and the corresponding eigenspaces for all eigenvalues of A are found.

We got a nice matrix here, but it's worth noting that λ need not be real. As demonstrated from the simple example above, solving the characteristic equation will lead to all eigenvalues for a matrix. I will leave more complex and interesting examples in the practice problems below. In the next part of the section, we will discuss the properties of eigenvalues and eigenvectors.

7.3.1 Properties of Eigenvalues

Eigenvalues and eigenvectors have interesting properties and relationship with vector spaces. Consider the following theorem.

Theorem 7.3.13

For two square matrices A and B such that $A \cong B$, the characteristic equation and eigenvalues for each matrix are equal.

Proof.

By definition of similar matrices, there exists some invertible matrix P such that $B = P^{-1}AP$. Therefore, the following equations hold.

$$P^{-1}(A - \lambda I)P = P^{-1}AP - P^{-1}\lambda IP = B - \lambda I$$

Therefore, $\det(B - \lambda I) = \det(P^{-1}(A - \lambda I)P) = \det(A - \lambda I)$ and the characteristic equations are the same. In other words, the eigenvalues for A and B are equal and the theorem holds.

Below is another theorem on the eigenvalues of triangular matrices.

Theorem 7.3.14

The eigenvalues of triangular (upper or lower) matrices are the diagonal elements.

Proof.

By definition, notice that for a triangular matrix A, $A - \lambda I$ is also triangular. Therefore by Theorem 5.5.4,

$$\det(A - \lambda I) = \prod_{k=1}^{n} (\lambda_k - \lambda)$$

and $\det(A - \lambda I_n) = 0$ if and only if $\lambda = \lambda_k$. Because λ_k are the diagonal elements of A, the theorem holds.

Continuing, below is another theorem on the relationship between eigenvalues and invertibility.

Theorem 7.3.15

A square matrix A contains a zero eigenvalue if and only if it is not invertible.

Proof.

A square matrix A contains a zero eigenvalue if and only if $\det(A - 0I) = 0$ and $\det(A) = 0$. Because A is invertible if and only if $\det(A) \neq 0$ by Theorem 5.5.13, the theorem holds.

If you think this theorem is nice, below is another theorem on invertibility and eigenvalues.

Theorem 7.3.16

For an invertible matrix A with eigenvector x and its corresponding eigenvalue λ, x is also an eigenvalue of A^{-1} with corresponding eigenvalue $\frac{1}{\lambda}$.

Proof.

By definition, $Ax = \lambda x$ and $x = A^{-1}\lambda x$. Continuing, $A^{-1}x = \frac{1}{\lambda} \cdot x$. Therefore, x is also an eigenvector of A^{-1} with the corresponding eigenvalue $\frac{1}{\lambda}$.

Continuing, below is a theorem on the relationship between eigenvalues and linear independence.

Theorem 7.3.17

If $x_1, \ldots, x_k$ are eigenvectors of a square matrix A and the corresponding eigenvalues $\lambda_1, \ldots, \lambda_k$ are all distinct, then $\{x_1, \ldots, x_k\}$ is linearly independent.

Proof.

First, notice that $\{x_1\}$ is linearly independent. Therefore, it suffices to show that if the theorem holds for $k-1 > 0$, then the statement holds for k. Assume $\{x_1, \ldots, x_{k-1}\}$ is a linearly independent set of eigenvectors with distinct corresponding eigenvalues $\lambda_1, \ldots, \lambda_{k-1}$. Consider the following equations for some scalars c_i.

$$\sum_{i=1}^{k} c_i x_i = 0$$
$$\sum_{i=1}^{k} c_i A x_i = 0$$
$$\sum_{i=1}^{k} c_i \lambda_i x_i = 0$$
$$\sum_{i=1}^{k-1} c_i \lambda_k x_i = 0$$

Subtracting the last two equations, $\sum_{i=1}^{k-1} c_i(\lambda_i - \lambda_k)x_i = 0$ is obtained. Because the set $\{x_1, \ldots, x_{k-1}\}$ is linearly independent, $c_i(\lambda_i - \lambda_k) = 0$ for all integer $i \in [1, k)$. Continuing, because $\lambda_i - \lambda_k \neq 0$ by construction, $c_i = 0$ for all integer $i \in [1, k)$ and $c_k = 0$ since $x_k \neq 0$. Thus $\{x_1, \ldots, x_k\}$ is linearly independent and the theorem holds by induction.

Naturally, we can consider the dimension of an eigenspace.

Theorem 7.3.18

For a matrix $A_{n\times n}$ with an eigenvalue λ, the following equation holds.

$$\dim(S_\lambda) = n - \mathrm{r}(A - \lambda I)$$

Proof.

By definition, the eigenspace for λ is the set of all vectors v that satisfy $Av = \lambda v$. Because $(A - \lambda)v = 0$, S_λ can be interpreted as $\ker(A - \lambda I)$. Moreover by Theorem 7.2.29, the following equation holds.

$$\mathrm{r}(A - \lambda I) + \mathrm{null}(A - \lambda I) = n$$

Because $\dim(\ker(A - \lambda I)) = \mathrm{null}(A - \lambda I)$ holds by definition, $\dim(S_\lambda) = n - \mathrm{r}(A - \lambda I)$.

We could really go on forever with interesting theorems, but before we conclude this part of the note, I would like to discuss two more theorems.

Theorem 7.3.19

Consider the following statements for a matrix A with an eigenvector x with its corresponding eigenvalue λ.

1. For an arbitrary scalar k, x is also an eigenvector of kA with corresponding eigenvalue $k\lambda$.
2. For some $n \in \mathbb{N}$, x is also an eigenvector of A^n with corresponding eigenvalue λ^n.

Let's start by proving the first statement.

Proof.

Consider the following equation.

$$(kA)x = k(Ax) = k\lambda x = (k\lambda)x$$

Therefore, x is an eigenvector of kA and its corresponding eigenvalue is $k\lambda$.

Below is the proof for the second statement.

Proof.

Assume that if the statement holds for $n-1$, then the statement holds for $n > 1$. Consider the following equations.

$$A^n x = A\left(A^{n-1}x\right) = A\left(\lambda^{n-1}x\right) = \lambda^{n-1}Ax = \lambda^{n-1}\lambda x = \lambda^n x$$

Because $Ax = \lambda x$, the statement holds by induction.

Before we move on to the final content with the last theorem for this part, below is a definition for the theorem.

Definition 7.3.20

The **trace** of a matrix $A = [a_{ij}]$, denoted as $\text{tr}(A)$, is the sum of the diagonal elements of A.

With the definition, we can continue with the theorem.

Theorem 7.3.21

For a square matrix $A_{n\times n}$ with eigenvalues $\lambda_1, \ldots, \lambda_n$, the following equations hold.

1. $\text{tr}(A) = \sum_{k=1}^{n} \lambda_k$
2. $\det(A) = \prod_{k=1}^{n} \lambda_k$

Let's start by proving the first equation.

Proof.

By the Fundamental Theorem of Algebra, the following equation holds.

$$\det(A - \lambda I) = \prod_{k=1}^{n}(\lambda_k - \lambda) = 0$$

Therefore, all eigenvalues are the diagonal elements and by the definition of trace of A, the sum of all eigenvalues are the trace of A.

Continuing, below is the proof for the second equation.

Proof.

Substituting $\lambda = 0$ into the equation

$$\det(A - \lambda I) = \prod_{k=1}^{n}(\lambda_k - \lambda),$$

$\det(A) = \prod_{k=1}^{n} \lambda_k$ and the equation holds.

It is important to note that the second equation counts multiplicity. In other words, even if $\lambda_p = \lambda_q$, you still count both. Wow! I can't believe that we are already almost done with the notes! For the final concept of my notes on linear algebra, we will discuss diagonalization.

7.3.2 Diagonalization

Diagonalization in linear algebra is a technique that I think really makes our lives easier in calculations. To get into it, let's get started with definition and theorems.

Definition 7.3.22

A square matrix that is similar to a diagonal matrix is known to be **diagonalizable**.

Now we cannot always use a brute force method to see if two matrices are similar. Fortunately, the following theorem will help in checking whether a square matrix is diagonalizable.

Theorem 7.3.23

A square matrix $A_{n\times n}$ is diagonalizable if and only if it retains a set of n eigenvectors that is linearly independent.

Proof.

First and foremost, notice that A and a diagonal matrix D of the same order with entries λ_i are similar if and only if there exists an invertible matrix P such that $P^{-1}AP = D$ or $AP = PD$. Let x_j and e_j be the j^{th} column of P and I respectively. Then, the following equations hold.

$$Ax_j = APe_j = PDe_j = P\lambda_j e_j = \lambda_j Pe_j = \lambda_j x_j$$

Thus, x_j and λ_j are an eigenvector and the corresponding eigenvalue of A. By definition, because M is invertible, x_j are all linearly independent and A has n linearly independent eigenvectors. In other words, if A is diagonalizable, then A has n linearly independent eigenvectors.

Continuing, the converse can be proven. Let A have n linearly independent eigenvectors denoted as x_j with corresponding eigenvalues λ_j. Construct matrices P and D such that the j^{th} column of P is x_j and the diagonal element of D is λ_j. If e_j is the j^{th} column of I, then the j^{th} column of AP can be written as the following.

$$APe_j = Ax_j = \lambda_j x_j$$

Similarly, the j^{th} column of PD can be written as $PDe_j = M\lambda_j e_j = \lambda_j M e_j = \lambda_j x_j$ and $AP = PD$ holds. Moreover, because P is invertible by construction, $P^{-1}AP = D$ and A is diagonalizable.

Continuing from the theorem, we can establish the following corollary.

Corollary 7.3.24

A matrix $A_{n\times n}$ with n distinct real eigenvalues is diagonalizable.

Proof.

First and foremost, notice that by Theorem 7.3.17, the set of corresponding eigenvectors $\{x_1, \ldots, x_n\}$ of A is linearly independent. Thus by Theorem 7.3.23, the corollary holds.

This theorem also naturally leads to the question if a square matrix can be similar to more than one diagonal matrix. The answer to that question can be found with the theorem below.

Theorem 7.3.25

If a square matrix is similar to both diagonal matrices D_1 and D_2, then the diagonal entries of D_1 and D_2 are the same with possibly different orderings.

Proof.

First and foremost, notice that by Theorem 7.3.13, because A and D_1 are similar, they have the same characteristic equation. Similarly, because A and D_2 are similar, they also have the same characteristic equation and the characteristic equations of D_1 and D_2 are equal.

$$\det(D_1 - \lambda I) = \det(D_2 - \lambda I) = 0$$

By construction, notice that $D_1 - \lambda I$ and $D_2 - \lambda I$ are diagonal matrices. Thus by Corollary 5.5.5, the determinant is the product of all $(\lambda - \lambda_k$ for integer $k \in [1, n]$ where λ_k are the diagonal entries for D_1. Because multiplication is commutative, the diagonal entries for D_1 and D_2 are the same.

Naturally, we can see that if $A \cong D_1$ and $A \cong D_2$, then $D_1 \cong A$ and $D_1 \cong D_2$. In other words, if A is similar to multiple diagonal matrices, then the diagonal matrices are similar to each other.

Now that we discussed when a matrix is diagonalizable, let's discuss how to diagonalize and its applications.

Definition 7.3.26

Diagonalization is the process of writing a diagonalizable matrix A in the form of $A = PDP^{-1}$.

Diagonalization is particularly useful as it makes calculations easier. Before concluding our notes, below is the final theorem!

Theorem 7.3.27

Let $A_{n\times n}$ be a matrix with real and distinct eigenvalues $\lambda_1, \ldots, \lambda_k$. The matrix A is diagonalizable if and only if the following equation holds.

$$\sum_{i=1}^{k} \dim(S_{\lambda_i}) = n$$

Proof.

First, the statement if $\sum_{i=1}^{k} \dim(S_{\lambda_i}) = n$, then A is diagonalizable can be proven. Let B_i be a basis for eigenspace S_{λ_i}. By definition, vectors v in eigenspace S_{λ_i} satisfy $Av = \lambda_i v$. Because λ_i are all distinct, no eigenspace shares the same eigenvector and thus no basis of two different eigenspaces shares a vector. In other words, B contains n distinct eigenvectors of A. Let $v_i \in S_{\lambda_i}$ be some linear combination of vectors in B_i. Notice that if $\sum_{i=1}^{k} v_i = 0$, then the coefficients must be zero since $\{v_1, \ldots, v_k\}$ is linearly independent by Theorem 7.3.17. In other words, all coefficients for the linear combination of B_i that forms v_i must be zero for $\sum_{i=1}^{k} v_i = 0$ to hold. Thus, B is linearly independent and A is diagonalizable by Theorem 7.3.23.

Continuing, the converse can be proven. Assume that $\sum_{i=1}^{k} \dim(S_{\lambda_i}) > n$. Continuing from the previous definition, $B \subset \mathbb{R}^n$ and the inequality $\sum_{i=1}^{k} \dim(S_{\lambda_i}) > n$ leads to contradiction. Moreover, if $\sum_{i=1}^{k} \dim(S_{\lambda_i}) < n$ then A has no set that contains n linearly independent eigenvectors. Thus, A is not diagonalizable if $\sum_{i=1}^{k} \dim(S_{\lambda_i}) \neq n$.

This is it for eigenvalues and eigenvectors! This note was a particularly long one, but I think it also means that the relationships between matrices and vector spaces are one of the most essential understandings for linear algebra. Below are a few practice problems on the topics that we discussed.

§7.4 Practice Problems

1. Determine whether a transformation defined as $T(\langle a, b\rangle) = a + b$ is linear. How about $T(\langle a, b\rangle) = ab$?

2. For square matrices A and B of order $n \times n$, show that if λ is an eigenvalue of AB, then the eigenvalues of BA also include λ [Erd20].

3. Find bases for the kernel and the image of $\begin{bmatrix} 1 & 1 & 1 \\ 0 & 1 & 0 \end{bmatrix}$.

4. Show that $A \cong B$ if and only if $A^\intercal \cong B^\intercal$.

5. Find the eigenvalues and corresponding eigenspace for $\begin{bmatrix} 1 & 2 & 3 \\ 0 & 1 & 0 \\ 0 & 1 & 2 \end{bmatrix}$.

Note 8

Appendix

Here are the solutions to the practice problems at the end of each note.

§8.1 Solutions for Note 1

1. It suffices to show that $\alpha(x,y,z) + \beta(x',y',z') \in S$ for any scalars α and β. With standard vector addition and scalar multiplication, $\alpha(x,y,z) + \beta(x',y',z') = (0, \alpha y + \beta y', \alpha z + \beta z') \in S$. Thus, the set is a vector space.

2. Let f and g be functions in $\mathbb{R}$ such that $f'(0) = 0$ and $g'(0) = 0$. Notice that $(f+g)'(0) = f'(0) + g'(0) = 0$ and $f + g \in S$. Moreover, for any scalars α, $(\alpha f)'(0) = \alpha f'(0) = 0$. Thus $\alpha f \in S$ and S is a vector space.

3. First, for any scalars α, notice that $\alpha(x_1, \ldots, x_n) = (\alpha x_1, \ldots, \alpha x_n)$. If each coefficient is divided by $\alpha \neq 0$, the scalar multiplication holds. The case where $\alpha = 0$ naturally holds. Continuing, notice that $(x_1, \ldots, x_n) + (x_1', \ldots, x_n') = (x_1 + x_1', \ldots, x_n + x_n') \in S$ due to the property of addition. Thus, S is a vector space.

4. Let $V = W_1 \cap \cdots \cap W_n$ where W_i for integer $i \in [1, n]$ is a subspace of V. For $u, v \in V$, notice that since $u, v \in W_i$ for all i and $\alpha u + \beta v \in W_i$ for any scalars α and β by construction, $\alpha u + \beta v \in V$ and the intersection of subspaces of a vector space is also a subspace of the vector space.

5. First and foremost, it is evident that $e^{a_1 x}$ is linearly independent since $e^{a_1 x} > 0$ for all a_1 and x. Therefore, it suffices to show that if the set is linearly independent for $n-1$, then it

is linearly independent for n. Consider the following equations for some c_k.

$$\sum_{k=1}^{n} c_k e^{a_k x} = 0$$
$$\sum_{k=1}^{n} c_k a_k e^{a_k x} = 0$$
$$\sum_{k=1}^{n} c_k a_n e^{a_k x} = 0$$

Subtracting the third equation, which is obtained by multiplying a_n in both sides, to the second equation obtained by differentiating both sides,

$$\sum_{k=1}^{n} c_k (a_k - a_n) e^{a_k x} = \sum_{k=1}^{n-1} c_k (a_k - a_n) e^{a_k x} = 0$$

is obtained. Because $a_k \neq a_n$ for integer $k \in [1, n-1]$ and $\{e^{a_1 x}, \ldots, e^{a_{n-1} x}\}$ is linearly independent $c_1 = \cdots = c_{n-1} = 0$. In other words, $c_n e^{a_n x} = 0$ and $c_n = 0$. Therefore, $\{e^{a_1 x}, \ldots, e^{a_n x}\}$ is linearly independent by induction.

6. By definition, note that $C = A + B$. Therefore, it suffices to show that $A \cap B = \{0\}$. Notice that A and B can be written as $A = \text{span}\{(1,1,1)\}$ and $B = \text{span}\{(20,15,12)\}$. Because $\lambda(1,1,1) = (20,15,12)$ if and only if $\lambda = 0$, $A \cap B = \{0\}$ and $C = A \oplus B$.

7. Consider the following equations for some scalars k_1, k_2, k_3.

$$k_1(a+b) + k_2(b+c) + k_3(c+a) = 0$$
$$a(k_3 + k_1) + b(k_1 + k_2) + c(k_2 + k_3) = 0$$

The linear combinations of $\{a+b, b+c, c+a\}$ equal zero if and only if the linear combinations of a, b, and c equal zero. Moreover, from the equation above, $k_1 = -k_2 = k_3 = -k_1$. In other words, $k_1 = k_2 = k_3 = 0$ and $\{a+b, b+c, c+a\}$ is linearly independent.

8. Let f and g be polynomials such that $f(0) = g(0) = 0$. Therefore, $(f+g)(0) = 0$ and $f+g$ is in the set. Moreover, $(\alpha f)(0) = 0$ for any scalars α. Therefore, the set of all polynomials in $\mathbb{P}^3$ that includes the origin is a subspace of $\mathbb{P}^3$.

§8.2 Solutions for Note 2

1. (a) Consider the following computation.

$$A - 2B = \begin{bmatrix} 1 & 2 \\ 2 & 0 \end{bmatrix} - 2\begin{bmatrix} -1 & 2 \\ 0 & 3 \end{bmatrix} = \begin{bmatrix} 1 & 2 \\ 2 & 0 \end{bmatrix} + \begin{bmatrix} 2 & -4 \\ 0 & -6 \end{bmatrix} = \begin{bmatrix} 3 & -2 \\ 2 & -6 \end{bmatrix}$$

(b) Consider the following equation.

$$AB = \begin{bmatrix} 1 & 2 \\ 2 & 0 \end{bmatrix} \begin{bmatrix} -1 & 2 \\ 0 & 3 \end{bmatrix} = \begin{bmatrix} -1 & 8 \\ -2 & 7 \end{bmatrix}$$

(c) Because AB is found above, BA could be found.

$$BA = \begin{bmatrix} -1 & 2 \\ 0 & 3 \end{bmatrix} \begin{bmatrix} 1 & 2 \\ 2 & 0 \end{bmatrix} = \begin{bmatrix} 3 & -2 \\ 6 & 0 \end{bmatrix}$$

Therefore, the following equation is true.

$$(AB - BA)^\intercal = \left(\begin{bmatrix} -1 & 8 \\ -2 & 7 \end{bmatrix} - \begin{bmatrix} 3 & -2 \\ 6 & 0 \end{bmatrix} \right)^\intercal = \left(\begin{bmatrix} -4 & 10 \\ -8 & 7 \end{bmatrix} \right)^\intercal = \begin{bmatrix} -4 & -8 \\ 10 & 7 \end{bmatrix}$$

2. By the definition of matrix multiplication, $a_2 B = [4, 5, 6]$.

3. Notice that A is a 2×2 square matrix. Therefore, define A as the following.

$$A = \begin{bmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{bmatrix}$$

Therefore, the following equation is obtained.

$$2A - 3A^\intercal = \begin{bmatrix} 2a_{11} & 2a_{12} \\ 2a_{21} & 2a_{22} \end{bmatrix} - \begin{bmatrix} 3a_{11} & 3a_{21} \\ 3a_{12} & 3a_{22} \end{bmatrix} = \begin{bmatrix} 2a_{11} - 3a_{11} & 2a_{12} - 3a_{21} \\ 2a_{21} - 3a_{12} & 2a_{22} - 3a_{22} \end{bmatrix} = \begin{bmatrix} -1 & -5 \\ 0 & -4 \end{bmatrix}$$

Continuing, the elements of A could be found.

$$\begin{aligned} 2a_{11} - 3a_{11} &= -1 \\ 2a_{12} - 3a_{21} &= -5 \\ 2a_{21} - 3a_{12} &= 0 \\ 2a_{22} - 3a_{22} &= -4 \end{aligned}$$

Therefore, $a_{11} = 1$, $a_{12} = 2$, $a_{21} = 3$, and $a_{22} = 4$. Thus, A can be defined as the following.

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

4. For any square matrix S, define A and B as the following.

$$A = \frac{S + S^\intercal}{2}, \quad B = \frac{S - S^\intercal}{2}$$

Notice that $S = A + B$. Therefore, it suffices to show that A is symmetric and B is

skew-symmetric. By Theorem 5.4.4, the following equations hold.

$$A^\intercal = \frac{S + S^\intercal}{2}, \quad B^\intercal = \frac{S^\intercal - S}{2} = -B$$

Thus, all square matrices can be represented as the sum of a symmetric and skew-symmetric matrices.

5. By definition, there exist $n, m \in \mathbb{Z}$ such that $A^n = 0$ and $B^m = 0$. Because $AB = BA$, $A^{nm}B^{nm} = 0$ can be written as $(AB)^{nm} = 0$ and AB is nilpotent.

6. By Theorem 5.4.4, $(AB + (AB)^\intercal)^\intercal = (AB)^\intercal + AB = AB + (AB)^\intercal$. Therefore, $AB + (AB)^\intercal$ is symmetric.

7. An easy trap for this problem is that you might be tempted to do $\det(A^2 - B^2) = \det(A + B)\det(A - B)$. However, notice that this does not hold in general as $AB \neq BA$ in general.

   Consider the following matrix P and its inverse P^{-1}.

$$P = \begin{bmatrix} I & I \\ I & -I \end{bmatrix}, \quad P^{-1} = \frac{1}{2}\begin{bmatrix} I & I \\ I & -I \end{bmatrix}$$

   Continuing,

$$P^{-1}\begin{bmatrix} A & B \\ B & A \end{bmatrix}P = \frac{1}{2}\begin{bmatrix} A+B & B+A \\ A-B & B-A \end{bmatrix}\begin{bmatrix} I & I \\ I & -I \end{bmatrix} = \begin{bmatrix} A+B & 0 \\ 0 & A-B \end{bmatrix}$$

   holds and $\begin{bmatrix} A & B \\ B & A \end{bmatrix} \cong \begin{bmatrix} A+B & 0 \\ 0 & A-B \end{bmatrix}$. Therefore, the determinant of the original matrix is $\det(A + B)\det(A - B)$.

8. By definition, A is skew-symmetric if $A^\intercal = -A$. Notice that $(A^3)^\intercal = (A^\intercal)^3$ by Theorem 5.4.4. Therefore, $(A^3)^\intercal = (A^\intercal)^3 = -A^3$. Thus, if A is skew-symmetric, then A^3 is also skew-symmetric.

§8.3 Solutions for Note 3

1. If A is invertible, then $A^{-1}Ax = A^{-1}b$ and $x = A^{-1}b$. Therefore, the system $Ax = b$ is always consistent.

2. Consider the following representation of the system.

$$\begin{aligned} x + 2y &= 1 \\ 2x + 4y &= k \end{aligned}$$

   Notice that if $k = 2$, then there exist infinitely many solutions while $k \neq 2$ gives no solutions. Thus the system never has a unique solution.

3. First and foremost, consider the following augmented matrix.

$$\left[\begin{array}{ccc|c} 1 & 1 & 1 & 10 \\ 3 & 4 & -1 & 2 \\ -1 & -3 & 1 & 4 \end{array}\right]$$

Using elementary row operations, the augmented matrix can be transformed as following.

$$\left[\begin{array}{ccc|c} 1 & 1 & 1 & 10 \\ 3 & 4 & -1 & 2 \\ -1 & -3 & 1 & 4 \end{array}\right] \to \left[\begin{array}{ccc|c} 1 & 1 & 1 & 10 \\ 0 & 1 & -4 & -28 \\ -1 & -3 & 1 & 4 \end{array}\right] \to \left[\begin{array}{ccc|c} 1 & 1 & 1 & 10 \\ 0 & 1 & -4 & -28 \\ 0 & -2 & 2 & 14 \end{array}\right]$$
$$\to \left[\begin{array}{ccc|c} 1 & 1 & 1 & 10 \\ 0 & 1 & -4 & -28 \\ 0 & 0 & -6 & -42 \end{array}\right]$$

Therefore, $-6z = -42$ and $z = 7$. Moreover, $y - 4z = -28$ and $y = 0$. Finally, $x + y + z = 10$ and $x = 3$. Therefore, the solution to the system is $(3, 0, 7)$.

4. First and foremost, let $A = \begin{bmatrix} 1 & 1 & 1 \\ 3 & 4 & -1 \\ -1 & -3 & 1 \end{bmatrix}$. Using elementary row operations, A can be transformed as following.

$$\begin{bmatrix} 1 & 1 & 1 \\ 3 & 4 & -1 \\ -1 & -3 & 1 \end{bmatrix} \to \begin{bmatrix} 1 & 1 & 1 \\ 0 & 1 & -4 \\ -1 & -3 & 1 \end{bmatrix} \to \begin{bmatrix} 1 & 1 & 1 \\ 0 & 1 & -4 \\ 0 & -2 & 2 \end{bmatrix} \to \begin{bmatrix} 1 & 1 & 1 \\ 0 & 1 & -4 \\ 0 & 0 & -6 \end{bmatrix}$$

In other words, A can be represented as the matrix multiplication below.

$$\begin{bmatrix} 1 & 1 & 1 \\ 3 & 4 & -1 \\ -1 & -3 & 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ 3 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ -1 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & -2 & 1 \end{bmatrix} \begin{bmatrix} 1 & 1 & 1 \\ 0 & 1 & -4 \\ 0 & 0 & -6 \end{bmatrix}$$
$$= \begin{bmatrix} 1 & 0 & 0 \\ 3 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ -1 & -2 & 1 \end{bmatrix} \begin{bmatrix} 1 & 1 & 1 \\ 0 & 1 & -4 \\ 0 & 0 & -6 \end{bmatrix}$$
$$= \begin{bmatrix} 1 & 0 & 0 \\ 3 & 1 & 0 \\ -1 & -2 & 1 \end{bmatrix} \begin{bmatrix} 1 & 1 & 1 \\ 0 & 1 & -4 \\ 0 & 0 & -6 \end{bmatrix}$$

Consider the following system for equation for $Ly = b$.

$$\begin{bmatrix} 1 & 0 & 0 \\ 3 & 1 & 0 \\ -1 & -2 & 1 \end{bmatrix} \begin{bmatrix} x' \\ y' \\ z' \end{bmatrix} = \begin{bmatrix} 10 \\ 2 \\ 4 \end{bmatrix}$$

It is evident that $x' = 10$, $y' = 2 - 3x' = -28$, and $z' = 4 + x' + 2y' = -42$. Continuing, the system for $Ux = y$ can be solved.

$$\begin{bmatrix} 1 & 1 & 1 \\ 0 & 1 & -4 \\ 0 & 0 & -6 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \end{bmatrix} = \begin{bmatrix} 10 \\ -28 \\ -42 \end{bmatrix}$$

Therefore, $z = 7$, $y = 0$, and $x = 3$ holds and the solution for the system is $(3, 0, 7)$ as shown in the previous problem.

5. To find the inverse of $A = \begin{bmatrix} 1 & 1 & 1 \\ 3 & 4 & -1 \\ -1 & -3 & 1 \end{bmatrix}$, consider the following augmentation and transformations.

$$\left[\begin{array}{ccc|ccc} 1 & 1 & 1 & 1 & 0 & 0 \\ 3 & 4 & -1 & 0 & 1 & 0 \\ -1 & -3 & 1 & 0 & 0 & 1 \end{array}\right] \rightarrow \left[\begin{array}{ccc|ccc} 1 & 1 & 1 & 1 & 0 & 0 \\ 0 & 1 & -4 & -3 & 1 & 0 \\ -1 & -3 & 1 & 0 & 0 & 1 \end{array}\right]$$

$$\rightarrow \left[\begin{array}{ccc|ccc} 1 & 1 & 1 & 1 & 0 & 0 \\ 0 & 1 & -4 & -3 & 1 & 0 \\ 0 & -2 & 2 & 1 & 0 & 1 \end{array}\right]$$

$$\rightarrow \left[\begin{array}{ccc|ccc} 1 & 1 & 1 & 1 & 0 & 0 \\ 0 & 1 & -4 & -3 & 1 & 0 \\ 0 & 0 & -6 & -5 & 2 & 1 \end{array}\right]$$

$$\rightarrow \left[\begin{array}{ccc|ccc} 1 & 1 & 1 & 1 & 0 & 0 \\ 0 & 1 & -4 & -3 & 1 & 0 \\ 0 & 0 & 1 & \frac{5}{6} & -\frac{1}{3} & -\frac{1}{6} \end{array}\right]$$

$$\rightarrow \left[\begin{array}{ccc|ccc} 1 & 1 & 1 & 1 & 0 & 0 \\ 0 & 1 & 0 & \frac{1}{3} & -\frac{1}{3} & -\frac{2}{3} \\ 0 & 0 & 1 & \frac{5}{6} & -\frac{1}{3} & -\frac{1}{6} \end{array}\right]$$

$$\rightarrow \left[\begin{array}{ccc|ccc} 1 & 0 & 0 & -\frac{1}{6} & \frac{2}{3} & \frac{5}{6} \\ 0 & 1 & 0 & \frac{1}{3} & -\frac{1}{3} & -\frac{2}{3} \\ 0 & 0 & 1 & \frac{5}{6} & -\frac{1}{3} & -\frac{1}{6} \end{array}\right]$$

Therefore, $A^{-1} = \begin{bmatrix} -\frac{1}{6} & \frac{2}{3} & \frac{5}{6} \\ \frac{1}{3} & -\frac{1}{3} & -\frac{2}{3} \\ \frac{5}{6} & -\frac{1}{3} & -\frac{1}{6} \end{bmatrix}$ and the system could be solved by multiplying A^{-1} and b.

$$\begin{bmatrix} -\frac{1}{6} & \frac{2}{3} & \frac{5}{6} \\ \frac{1}{3} & -\frac{1}{3} & -\frac{2}{3} \\ \frac{5}{6} & -\frac{1}{3} & -\frac{1}{6} \end{bmatrix} \begin{bmatrix} 10 \\ 2 \\ 4 \end{bmatrix} = \begin{bmatrix} 3 \\ 0 \\ 7 \end{bmatrix}$$

Thus, the solution is $(3, 0, 7)$, which is consistent with the previous problems.

§8.4 Solutions for Note 4

1. For transformation $T(\langle a, b\rangle) = a + b$, consider the following equations for some scalars α and β.

$$\begin{aligned} T(\alpha\langle a_1, b_1\rangle + \beta\langle a_2, b_2\rangle) &= T(\langle \alpha a_1, \alpha b_1\rangle + \langle \beta a_2, \beta b_2\rangle) \\ &= \alpha a_1 + \alpha b_1 + \beta a_2 + \beta b_2 \\ &= \alpha(a_1 + b_1) + \beta(a_2 + b_2) \\ &= \alpha T(\langle a_1, b_1\rangle) + \beta T(\langle a_2, b_2\rangle) \end{aligned}$$

   Therefore by the definition of linear transformation, T is linear.

   For $T(\langle a, b\rangle) = ab$, consider the following for arbitrary scalars α and β.

$$\begin{aligned} T(\alpha\langle a_1, b_1\rangle + \beta\langle a_2, b_2\rangle) &= T(\langle \alpha a_1, \alpha b_1\rangle + \langle \beta a_2, \beta b_2\rangle) \\ &= T(\langle \alpha a_1 + \beta b_2, \alpha b_1 + \beta b_2\rangle) \\ &= (\alpha a_1 + \beta b_2)(\alpha b_1 + \beta b_2) \end{aligned}$$

   Moreover,

$$\alpha T(\langle a_1, b_1\rangle) + \beta T(\langle a_2, b_2\rangle) = \alpha a_1 b_1 + \beta a_2 b_2$$

   holds. Because $(\alpha a_1 + \beta b_2)(\alpha b_1 + \beta b_2) \neq \alpha a_1 b_1 + \beta a_2 b_2$ in general, the transformation is not linear.

2. By definition, if λ is an eigenvalue of AB, then there exists some nonzero vector v such that $ABv = \lambda v$. First, let $\lambda \neq 0$. Notice that $u = Bv \neq \mathbf{0}$. Therefore, $BAu = BA(Bv) = B(ABv) = B(\lambda v) = \lambda(Bv) = \lambda u$ and λ is also an eigenvalue of BA. Continuing if $\lambda = 0$, then $\det(AB) = \det(A)\det(B) = 0 = \det(B)\det(A) = \det(BA)$. In other words, if AB is singular then BA is also singular and contains the eigenvalue λ. Therefore, λ is an eigenvalue of AB if and only if it is an eigenvalue of BA.

3. First, to find bases for the kernel of the matrix, the following equation must be solved.

$$\begin{bmatrix} 1 & 1 & 1 \\ 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \end{bmatrix}$$

   Solving the system, $y = 0$ and $x = -z$. Therefore the basis for the kernel is $\left\{ \begin{bmatrix} -1 \\ 0 \\ 1 \end{bmatrix} \right\}$.

Continuing, the basis for the image can be found. Consider the following equation.

$$\begin{bmatrix}1&1&1\\0&1&0\end{bmatrix}\begin{bmatrix}x\\y\\z\end{bmatrix}=\begin{bmatrix}x+y+z\\y\end{bmatrix}=x\begin{bmatrix}1\\0\end{bmatrix}+y\begin{bmatrix}1\\1\end{bmatrix}+z\begin{bmatrix}1\\0\end{bmatrix}$$

Therefore,

$$\left\{\begin{bmatrix}1\\0\end{bmatrix},\begin{bmatrix}1\\1\end{bmatrix}\right\}$$

is a basis of the image of the given matrix.

4. By definition, $A\cong B$ if and only if there exists an invertible matrix P such that $A=P^{-1}BP$. Continuing, $A=P^{-1}BP$ if and only if $A^\intercal=(P^{-1}BP)^\intercal=P^\intercal B^\intercal(P^{-1})^\intercal=P^\intercal B^\intercal(P^\intercal)^{-1}$. In other words, $A\cong B$ if and only if $A^\intercal\cong B^\intercal$.

5. To find the eigenvalues for the given matrix say A, it suffices to solve $\det(A-\lambda I)=0$ where λ is an eigenvalue.

$$\det\left(\begin{bmatrix}1-\lambda&2&3\\0&1-\lambda&0\\0&1&2-\lambda\end{bmatrix}\right)=0$$

Continuing with the expansion along the first column,

$$\begin{aligned}\det\left(\begin{bmatrix}1-\lambda&2&3\\0&1-\lambda&0\\0&1&2-\lambda\end{bmatrix}\right)&=(1-\lambda)\det\left(\begin{bmatrix}1-\lambda&0\\1&2-\lambda\end{bmatrix}\right)\\&=(1-\lambda)(1-\lambda)(2-\lambda)\end{aligned}$$

Therefore, the eigenvalues are 1 and 2. For eigenvalue 1, it suffices to solve the equation $Av=v$ or $(A-I)v=0$. Consider the following system.

$$\begin{bmatrix}0&2&3\\0&0&0\\0&1&1\end{bmatrix}\begin{bmatrix}x\\y\\z\end{bmatrix}=\begin{bmatrix}0\\0\\0\end{bmatrix}$$

Therefore, $2y+3z=0$ and $y+z=0$. In other words, $y=z=0$. Thus, $S_1=\operatorname{span}\left\{\begin{bmatrix}1\\0\\0\end{bmatrix}\right\}$ holds for $\lambda=1$. Continuing with $\lambda=2$, it suffices to solve $Av=2v$ or $(A-2I)v=0$.

$$\begin{bmatrix}-1&2&3\\0&-1&0\\0&1&0\end{bmatrix}\begin{bmatrix}x\\y\\z\end{bmatrix}=\begin{bmatrix}0\\0\\0\end{bmatrix}$$

Solving the system, $y = 0$ and $x = 3z$. Thus, $S_2 = \text{span}\left\{\begin{bmatrix}3\\0\\1\end{bmatrix}\right\}$ holds for $\lambda = 2$.

§8.5 Final Thoughts

I wanted to include my final thoughts on these notes and notice on some topics that were not covered here, and I guess I can use the help of appendix.

First, as mentioned in the prefix, these are the notes that I took while learning the essence of linear algebra in 8 weeks. That being said, while most of the fundamental concepts were covered, other topics like dual space, cross product, euclidean inner product, and more advanced concepts including Markov chains and Jordan canonical forms are not included here. After the initial release, I will continue to update these notes for the second edition as I will likely take linear algebra course again in college as a student who will major in pure math. In the meantime, I highly recommend watching 3Blue1Brown series on linear algebra as it really helped me visually understand different concepts and formulas.

As I also study different concepts in machine learning, I realized how important linear algebra is outside "mathematics." Literally, vectors are used everywhere including, but not limited to encoding/decoding and backpropagation. One of the reasons why I find math so fascinating is due to duality and applications that just seems so perfect. Matrices can be interpreted with heavy computation while it can also be interpreted geometrically with change in area. The two seemingly unrelated topics actually imply the same exact concept. Similarly, what else can we use instead of matrix multiplication and addition for forward pass in machine learning? The way it seems so intuitive and "perfect" is I think one of the most beautiful side of mathematics. Really, if you completed these notes, I highly recommend learning math behind machine learning if you like finding different applications of mathematics. More information can be found in the next part of LibreNotebook.

References

[Ric07] Gabriel B. Costa Richard Bronson. *Linear Algebra*. Elsevier, 2007. ISBN: 9780080510262.

[Erd20] J.M. Erdman. *Exercises And Problems In Linear Algebra*. World Scientific Publishing Company, 2020. ISBN: 9789811220425.

[Kan26] Margarita Kanarsky. *Linear Algebra (XM511)*. 2026.

Part III

Machine Learning

Contents

What is AI and Why Should We Learn It?

Unless you've been living under a rock, you probably have heard of the term "AI" and "Machine Learning". Just as the name suggests Artificial Intelligence is any technology that mimics human intelligence. That being said, AI is a very, very broad term. Under AI, there is machine learning (ML), which quite literally means techniques, or machines, that can learn from data to better mimic human intelligence. Now, just as us humans learn from data with our brain, a subset of ML is Deep Learning (DL), which uses neural networks inspired from our brains. We could continue more, but let's save that for later since this is a brief introduction.

AI has been studied, and will continue to develop. In October 2015, a legendary match between an AI model AlphaGo developed by DeepMind and one of the greatest Go player Lee Sedol was conducted. During the game, AlphaGo demonstrated its creativity through numerous moves including Move 37. Ultimately, AlphaGo won the game with 4-1 victory. More information can be found in AlphaGo.

Despite the long history, the modern AI boom in the public occurred from late 2022 when OpenAI released its GPT-3.5. Nevertheless, LLMs have significantly transformed the lives of countless people. For instance, as the concept of "vibe coding" emerged, the concept of "coding" became more accessible to the public. Moreover, job markets are transforming with the rise of AI. Now despite all "AI bubbles" concerns and anti-AI movements, it is somewhat self-evident that AI and the concept of AGI (Artificial General Intelligence) will not suddenly disappear in near future. Then, the real question is "do you want to become a consumer or producer?"

To briefly share why I began studying ML, the reason is quite simple. As a student who wishes to advance as a quantitative researcher, not learning ML is just not in the option. Moreover, learning to develop AI will widen the future opportunities even if I decided different path than quantitative finance. Although the ideas on an AI model developing another model is rising, it is imperative that a human engineer must superintend the architecture and development. Finally, it is fun to learn. I mean what is more intriguing than to learn about your favorite tool and how to contribute to its development?

Note 9

Fundamental Math for Machine Learning

This note covers fundamental math necessary for machine learning. Please note that I won't nor have the capacity to include details on the concepts that are not strictly necessary to learn ML, which I believe are out of the scope of the notes. With that in mind, let's get started!

§9.1 Calculus

Ah yes, that one friend who's everywhere, where ever you go. Broadly speaking, calculus is one of the foundational math for machine learning. For instance, backpropagation, which we will discuss further in latter notes, is fundamentally based on the chain rule. There are actually a lot to cover on calculus and there are some advanced topics. Good news is that you probably don't need calculus concepts outside multivariable calculus (a.k.a. Calculus III) unless you are doing theoretical works, and they are out of my capacity. That being said, I will include most calculus concepts that you will need for general purpose, but concepts that are out of scope of these notes will not be included.

9.1.1 Foundations

I am assuming most of you have some background in calculus, so I will be skipping basic concepts from Calculus AB/BC (a.k.a Calculus I and II) as this section is about calculus for machine learning. Please don't worry if you don't as you probably won't need all concepts from the courses to understand the topics here. Now, let's start by building from terminologies.

Definition 9.1.1

The **limit** of a function is the value that the function approaches as the input gets arbitrarily close to a value.

Below are some examples.

$$\lim_{x\to 3} x^2 = 3^2 = 9, \quad \lim_{x\to\infty} \frac{1}{x} = 0, \quad \lim_{x\to 1} \frac{1}{x-1} \text{ DNE}$$

Definition 9.1.2

Derivative of a function respect to a variable is the measure of the rate of change of the output based on the change of the variable(s). **Gradient** is a vector of partial derivatives. **Differentiation** is the process of finding the derivative.

With equations, derivative and partial derivatives can be written as the following for function $f(x)$ and $f(x_1, \ldots, x_n)$.

$$f'(a) = \lim_{h\to 0} \frac{f(a+h) - f(a)}{h}$$
$$\frac{\partial}{\partial x_i} f(a) = \lim_{h\to 0} \frac{f(a_1, \ldots, a_{i-1}, a_i + h, a_{i+1}, \ldots, a_n) - f(a_1, \ldots, a_n)}{h}$$

For gradient ∇f, read as "nabla f", the mathematical expression follows.

$$\nabla f(a) = \begin{bmatrix} \frac{\partial f}{\partial x_1}(a) \\ \vdots \\ \frac{\partial f}{\partial x_n}(a) \end{bmatrix}$$

For instance, the gradient of $f(x, y) = x^2 + 2y$ can be written as $\nabla f = \langle 2x, 2 \rangle$ and the gradient at $(1, 1)$ can be written as $\nabla f(1, 1) = \langle 2, 2 \rangle$.

Now that we discussed about derivatives, let's take a look at common rules that will appear for machine learning. Starting with the power rule, below is the list of common rules.

Theorem 9.1.3: Common Rules for Derivatives

1. $\frac{d}{dx} x^n = nx^{n-1}$
2. $\frac{d}{dx} e^x = e^x$
3. $\frac{d}{dx} \ln(x) = \frac{1}{x}$
4. $\frac{d}{dx} a = 0$
5. $\frac{d}{dx} cf(x) = cf'(x)$
6. $\frac{d}{dx} \left(f(x) \pm g(x)\right) = \frac{d}{dx} f(x) \pm \frac{d}{dx} g(x)$
7. $\frac{d}{dx} \left(f(x)g(x)\right) = f'(x)g(x) + f(x)g'(x)$
8. $\frac{d}{dx} \left(\frac{f(x)}{g(x)}\right) = \frac{f'(x)g(x) - f(x)g'(x)}{(g(x))^2}$
9. $\frac{\partial^2 f}{\partial x^2} = \frac{\partial}{\partial x} \left(\frac{\partial f}{\partial x}\right)$
10. $\frac{\partial^2 f}{\partial y \partial x} = \frac{\partial}{\partial y} \left(\frac{\partial f}{\partial x}\right)$

Let's skip the proofs for this one! In addition to derivative rules, one of the most primary use case of calculus in machine learning is the chain rule. Indeed, the neural network is

fundamentally based on the chain rule in calculus.

Theorem 9.1.4: The Chain Rule of Calculus

The derivative of a composite function $h(x) = f \circ g(x)$ can be represented as $h'(x) = f'(g(x)) \cdot g'(x)$. Alternatively, for $y = f(u)$ and $u = g(x)$, the following equation holds.

$$\frac{dy}{dx} = \frac{dy}{du} \cdot \frac{du}{dx}$$

Proof.

Consider the following equation by the definition of derivatives.

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

Substituting $h(x) = f(g(x))$, the following equations are obtained.

$$\begin{aligned} h'(x) &= \lim_{n \to 0} \frac{f(g(x+n)) - f(g(x))}{n} \\ &= \lim_{n \to 0} \frac{f(g(x+n)) - f(g(x))}{g(x+n) - g(x)} \cdot \lim_{n \to 0} \frac{g(x+n) - g(x)}{n} \end{aligned}$$

Let $n' = g(x+n) - g(x)$. Assuming that g is continuous, as $n \to 0$, $n' \to 0$. Therefore, the following equation is obtained.

$$h'(x) = \lim_{n' \to 0} \frac{f(u+n') - f(u)}{n'} \cdot \lim_{n \to 0} \frac{g(x+n) - g(x)}{n} = \frac{dy}{du} \cdot \frac{du}{dx}$$

Thus, the chain rule holds.

For a brief example, let's take a look at an example with sigmoid function.

Exercise 9.1.5: The Derivative of Sigmoid Function

Sigmoid function is defined as $\sigma(x) = \frac{1}{1+e^{-x}}$. Find the derivative of $\sigma(x)$.

Solution.

Consider the following process with the product and chain rule.

$$\begin{aligned} \sigma'(x) &= \frac{d}{dx}\left(1+e^{-x}\right)^{-1} = -\left(1+e^{-x}\right)^{-2} \cdot \left(-e^{-x}\right) \\ &= \left(1+e^{-x}\right)^{-2} \cdot e^{-x} = \left(1+e^{-x}\right)^{-1} \cdot \frac{e^{-x}}{1+e^{-x}} \\ &= \sigma(x)\left(1-\sigma(x)\right) \end{aligned}$$

Therefore, the derivative $\sigma'(x) = \sigma(x)(1-\sigma(x))$.

In addition to the chain rule above, we could expand it for multivariable functions.

Theorem 9.1.6: The Chain Rule for One Independent Variable

For differentiable functions $x = a(t)$, $y = b(t)$, and $z = f(x, y)$, the following equation holds.

$$\frac{dz}{dt} = \frac{\partial z}{\partial x} \cdot \frac{dx}{dt} + \frac{\partial z}{\partial y} \cdot \frac{dy}{dt}$$

Similarly, we could expand it for cases where x and y are functions of two variables.

Theorem 9.1.7: The Chain Rule for Two Independent Variable

For differentiable functions $x = a(u, v)$, $y = b(u, v)$, and $z = f(x, y)$, the following equations hold.

$$\frac{\partial z}{\partial u} = \frac{\partial z}{\partial x}\frac{\partial x}{\partial u} + \frac{\partial z}{\partial y}\frac{\partial y}{\partial u}$$
$$\frac{\partial z}{\partial v} = \frac{\partial z}{\partial x}\frac{\partial x}{\partial v} + \frac{\partial z}{\partial y}\frac{\partial y}{\partial v}$$

For the purpose of these notes, let's skip the proof for the two theorems above.

§9.2 Linear Algebra

Linear algebra is a foundational building block of ML as majority of the components are written in very high dimensional arrays and tensors. Without exaggeration, training and using a model can be considered as performing billions of linear algebra operations. Therefore, it is encouraged to get familiar with the notations as they will be used throughout the future notes. That being said, because many of the contents were introduced in detail in the linear algebra part of LibreNotebook, I will keep it brief here for topics that were not introduced in the previous part.

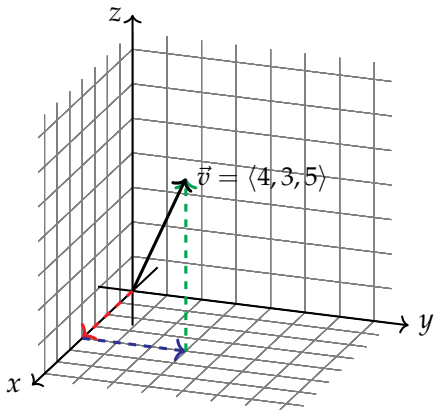
9.2.1 Foundations

Many mathematicians might not like the definition below, but I think it is worth considering different perspectives of how we can see vectors and matrices.

Definition 9.2.1

A 1-dimensional vector is simply called **vector**. A 2D vector is called **matrix**. For vectors with 3+ dimensions, the term **tensor** is used.

Now scalars are not considered vectors because it is only defined with magnitude. Geometrically speaking, the dimensions here can be interpreted as the dimension that we usually say in science. For example, a matrix can be plotted in a 2D space with x and y components. Similarly, a 3D tensor can be drawn in 3-dimensional plane with x, y, and z coordinates. Consider the following example.



Continuing, we can see an operation that vectors can perform that we have not discussed in the notes for linear algebra.

Definition 9.2.2

A **dot product** of two vectors is the sum of products of the corresponding elements.

Dot product will be quite important when we discuss attention and transformers because it can tell us how "related" two vectors are. Indeed, consider the vectors A and B with the angle between the two vectors as θ. The following equation holds.

$$A \cdot B = ||A||||B|| \cos\theta$$

Here, $||V||$ for $V \in \mathbb{R}^n$ represents the magnitude of the vector, i.e. $\sqrt{\sum_{k=1}^{n} V_k^2}$. Let's take a look at a quick example.

$$\begin{aligned} \langle 1, 0 \rangle \cdot \langle 1, \sqrt{3} \rangle &= 1 \cdot 1 + 0 \cdot \sqrt{3} = 1 \\ &= 1 \cdot 2 \cdot \cos 60^\circ = 1 \end{aligned}$$

One thing to keep in mind is that we have a special term for the sign $||V||$.

Definition 9.2.3

The **vector norm** is the non-negative value that represents the magnitude of a vector.

With that in mind, let's move on to a more abstract and advanced topics in linear algebra that are constantly used in machine learning.

9.2.2 CONTINUE

§9.3 Probability Theory

Ultimately, ML models are prediction engines. In other words, it is very crucial to understand fundamental probability theory. Therefore, let's get started with basic terminologies and

concepts.

Definition 9.3.1

Probability distribution is a process of describing probability and possible cases.

For example, using functions or tables to represents probability of getting a certain rolling sum after rolling a fair standard die two times is using probability distribution.

Few symbols to keep in mind are is used to represent the following of the probability distribution, and $\propto$ is used to illustrate direct proportionality.

Definition 9.3.2

Sample space, denoted as Ω, is a set of possible outcomes. **Probability measure** is a function that maps such outcome events to probability in $[0, 1]$. **Random Variables** are functions that map the outcomes to certain numbers.

Notice that the sum of all measures in a sample space is 1. Now sometimes, probabilities depend on the previous events.

Definition 9.3.3

If A is the event that occurs after B, the probability of A occurring after B is known as the **posterior probability**, denoted as $P(A \mid B)$. Moreover, $P(A)$ is the **prior** probability.

For instance, if A is the event where you see a snail outside and B is the event of raining, then the probability $P(A \mid B)$ is the probability of you seeing a snail outside given that it is raining. With this in mind, we can observe the following important theorem that will come very handy when we discuss language models.

Theorem 9.3.4: The Chain Rule of Probability

The chain rule of probability states the following equation.

$$P(A, B) = P(A \mid B)P(B)$$

The equation above is read as the probability of A and B is equal to the probability of A given B multiplied by the probability of B. The direct corollary of the chain rule is Bayes' Theorem.

Theorem 9.3.5: Bayes' Theorem

Conditional probabilities can be found with Bayes' Theorem defined with the equation below for $P(B) \neq 0$.

$$P(A \mid B) = \frac{P(B \mid A)P(A)}{P(B)}$$

As you may have guessed, we can scale the chain rule above with generalization. Consider the following generalized version of the chain rule.

Theorem 9.3.6: Generalized Chain Rule

The chain rule of probability states the following.

$$\begin{aligned} P(x_1, x_2, \ldots, x_n) &= P(x_1)P(x_2 \mid x_1)P(x_3 \mid x_1, x_2) \\ &\qquad \cdots P(x_n \mid x_1, x_2, \ldots, x_{n-1}) \\ &= \prod_{i=1}^{n} P\left(x_i \,\middle|\, \bigcap_{k=1}^{i-1} x_k\right) \end{aligned}$$

Proof.

Deriving this generalization from the chain rule with two variables is rather simple. We can use the definition multiple times to multiply and telescope. Here is what I mean. (Note that mathematicians like to use the standard set notations while ML engineers likes to use a more "colloquial notation". For the purpose of these notes, let's use the notations that ML engineers use.)

Consider the following equations by the definition of conditional probability.

$$\begin{aligned} P(x_1) &= P(x_1) \\ P(x_2 \mid x_1) &= \frac{P(x_1, x_2)}{P(x_1)} \\ P(x_3 \mid x_1, x_2) &= \frac{P(x_1, x_2, x_3)}{P(x_1, x_2)} \\ &\vdots \\ P(x_n \mid x_1, \ldots, x_{n-1}) &= \frac{P(x_1, \ldots, x_n)}{P(x_1, \ldots, x_{n-1})} \end{aligned}$$

Multiplying the equations, the following equation is obtained.

$$\begin{aligned} &P(x_1)P(x_2 \mid x_1)P(x_3 \mid x_1, x_2)P(x_n \mid x_1, \ldots, x_{n-1}) \\ &\qquad = P(x_1) \cdot \frac{P(x_1, x_2)}{P(x_1)} \cdot \frac{P(x_1, x_2, x_3)}{P(x_1, x_2)} \cdots \frac{P(x_1, \ldots, x_n)}{P(x_1, \ldots, x_{n-1})} \end{aligned}$$

Telescoping the right hand side, we get our final equation.

$$P(x_1, \ldots, x_n) = P(x_1)P(x_2 \mid x_1)P(x_3 \mid x_1, x_2)P(x_n \mid x_1, \ldots, x_{n-1})$$

Alternatively, we could try going backward from the final equation from the definition, but I found this proof a little more satisfying.

Note 10

Python Basics for Machine Learning

To develop a model represented in abstract math, we need a programming language. If you ask any ML engineer on the language they use the most, I can guarantee you that it's Python (well most likely). The reason is simple: syntax and libraries. Although all heavy computations in ML is mostly done by C++, Python is way easier to write than C++. You don't have to deal with segmentation faults and spend more time adding the missing semicolon than actually debugging. Moreover, Python has popular libraries like NumPy, Matplotlib, and PyTorch that will make our lives easier. Therefore, we will discuss about basic syntax, NumPy, and Matplotlib for ML in this note. Don't worry, we will discuss about PyTorch in latter notes as it requires ML concepts.

§10.1 Basic Syntax

As always, let's get started by being more familiar with the nature of Python. Unlike most other languages, indentation is a vital factor in Python. Two spaces, four spaces, doesn't really matter, let's use four spaces as it is more common and readable. Here is the Hello World example.

```
1 print("Hello World")
```

```
Hello World
```

Please note that this Jupyter Notebook can be found in `jupyter/` directory in GitHub under the root directory for these notes. Please feel free to download and follow along! The next example is a list of simple operations that we can do with Python.

```
1 import math
2 import numpy as np
3 import matplotlib.pyplot as plt
4 %matplotlib inline
```

```
5
6  print(1 + 2)        # Addition
7  print(3 - 9)        # Subtraction
8  print(4 * 3)        # Multiplication
9  print(5 ** 2)       # Power
10 print(math.log(5)) # log(a, base), default base is e
11 print(math.exp(2)) # e^2
12 print(math.e)       # Accessing constants
```

```
3
-6
12
25
1.6094379124341003
7.38905609893065
2.718281828459045
```

As you can see in Cell 2, we need module "math" to access constants and functions like e, π, and $\log_a b$.

Definition 10.1.1

We can achieve the same effect with **variables**, or a place with name that stores a value. A **function** is a reusable block of code dedicated for specific tasks.

Let's take a look at an example of how function and variable in addition to the operations above can be used to find the approximation of the derivative of $f(x) = x^2 + 5$ at $x = 3$.

```
1 def f(x):
2     return x**2 + 5
3
4 a = 3
5 h = 0.0001
6 der = (f(a + h) - f(a))/h
7 print(der)
```

```
6.000100000012054
```

Let's check if our answer is reasonable. Using the Power Rule, we know that $f'(x) = 2x$. Because $f(3) = 2 \cdot 3 = 6$ and 6.000100000012054 and 6 are somewhat close, our result is reasonable.

Definition 10.1.2

Lists are dynamic and ordered collection of elements of potentially different data types. An **index** is the position of an element in the list. The index of the first item is 0.

Below are examples of what we can do with lists.

```python
# A list can be empty
a = []
print(a)

# We can append a value at the end
a = [1, 'hi', 3]
a.append('bye')
print(a)

# Remove an element
a.remove('hi')
print(a)

# Access an element with index
print(a[2])

# Find the number of elements
print(len(a))
```

```
[]
[1, 'hi', 3, 'bye']
[1, 3, 'bye']
bye
3
```

What do we do if we need to access multiple elements? This is where **slicing** shines. Consider the following example.

```python
# a[i:j] to access from index 'i' to index 'j-1'
a = [1, 2, 3, 4, 5, 6, 7, 8, 9]
print(a[1:5])

# Omit 'i' or 'j' to define only the start or end values
print(a[:6])
print(a[3:])
```

```
[2, 3, 4, 5]
[1, 2, 3, 4, 5, 6]
[4, 5, 6, 7, 8, 9]
```

Now that we know basic operations, lists, and how we can use functions, let's discuss about conditions and loops.

Definition 10.1.3

Conditional statements are statements that performs certain tasks under specific conditions. **Loops** are structure that repeats a block of code until they are explicitly told to end. One more definition that will make our lives easier is **increment**, which is an assignment operator that augments certain values.

Here is an example of conditional statements.

```
# If, Else, Elif (else if)
banana_num = 10
if banana_num < 10:
    print("There are less than ten bananas")
elif banana_num == 10:
    print("There are ten bananas")
else:
    print("There are more than ten bananas")
```

```
There are ten bananas
```

Here, because "=" is reserved, we use "==" for "equal to." We could also do less than or equal to, greater than or equal to, and not equal to with <=, >=, and ! = respectively.

Regarding loops, "for loop" and "while loop" are two primary methods. Below are examples of how both loops can be used.

```
# While loop
love_banana = True
banana_num = 0

while love_banana and banana_num < 5:
    banana_num += 1 # Increment
    print(banana_num)

print()

# For loop
my_banana = ['yellow', 'green', 'mushy', 'sweet']

for i in range(len(my_banana)):
    print(my_banana[i])
```

```
1
2
3
4
```

```
5

yellow
green
mushy
sweet
```

First, `love_banana = True` is a Boolean value. In the while loop, what it does is while `love_banana` is True, we will increase the value `banana_num` by 1 and print them. Notice that the increment with `banana_num += 1` is a shorter form of writing `banana_num = banana_num + 1`. Moreover, we use `and` to make the loop continue if `love_banana` is true and `banana_num` is less than five.

In the second loop, we listed all elements in the list `my_banana`. The loop `for i in range()` tells Python to go through the code
`print(my_banana[i])` from $i = 0$ to range(len(my_banana)) $- 1$.

We discussed very basic Python syntax that we must know to understand the following notes. Let's discuss core libraries NumPy and Matplotlib now, and save PyTorch for later.

§10.2 Modules and Libraries

First, what are modules and libraries?

Definition 10.2.1

Modules in Python are files that consists of definitions and statements for organization and reusability. **Libraries** are collection of modules and pre-written packages that can later be imported and distributed.

One of the reason why Python is so popular in ML and data science is because of the extensive libraries. Having the pre-written libraries specifically for ML and tensors like PyTorch simply makes our lives easier.

10.2.1 NumPy

NumPy is a Python package that helps handle multidimensional arrays. Let's get started by importing NumPy to our code. We could import it as whatever alias we want, but `np` is the standard for ML, so let's stick with that.

```
1 import numpy as np
```

Assuming that you have read the previous note or have strong foundations in linear algebra, we can implement the mathematical properties in Python with NumPy. Below is an example of

how we can define vectors in different dimensions. Note that `.ndim` can be used to find the dimension of a vector.

```
arr_1d = np.array([1, 2, 3])
arr_2d = np.array([[1, 2, 3],
                   [4, 5, 6]])
arr_3d = np.array([[[1, 2], [3, 4]],
                   [[5, 6], [7, 8]],
                   [[9, 10], [11, 12]]])

print(f"Dimension: {arr_1d.ndim}, Shape: {arr_1d.shape}")
print(f"Dimension: {arr_2d.ndim}, Shape: {arr_2d.shape}")
print(f"Dimension: {arr_3d.ndim}, Shape: {arr_3d.shape}")
```

```
Dimension: 1, Shape: (3,)
Dimension: 2, Shape: (2, 3)
Dimension: 3, Shape: (3, 2, 2)
```

A simple way to understand multidimensional array in NumPy is stacking previous layers. For instance, `arr_2d` has the shape $(2,3)$. We can think of it as stacking two 1D array with three elements. Similarly, `arr_3d` has the shape $(3,2,2)$. We could think of it as stacking three layers of 2D arrays that consists of two 1D arrays. It is quite hard to visualize higher dimensional arrays, so let's take a look at different things that NumPy can do.

Below is an example where we utilize arange, reshape, dtype, and slicing techniques in NumPy. Let's create an array first.

```
# Making an array with numbers 0-17 and
# rerranging it to shape (3, 3, 2)
a = np.arange(18).reshape(3, 3, 2)
a
```

```
array([[[ 0,  1],
        [ 2,  3],
        [ 4,  5]],

       [[ 6,  7],
        [ 8,  9],
        [10, 11]],

       [[12, 13],
        [14, 15],
        [16, 17]]])
```

We can also find the data type with the attribute `.dtype`.

```
a.dtype.name
```

```
'int64'
```

This means that the individual elements in array `a` are 8 bytes integers. Just as we index and slice 1D array, we could do the same with NumPy. Consider the following example for a matrix.

```
a = np.arange(9).reshape(3, 3)
print(f"{a}\n")

# First row, second column element
print(a[0, 1])
# Entire second row
print(a[1, :])
# Entire third column
print(a[:, 2])
```

```
[[0 1 2]
 [3 4 5]
 [6 7 8]]

1
[3 4 5]
[2 5 8]
```

This was the basic introduction to NumPy. Please check out the documentation for further information.

10.2.2 Matplotlib

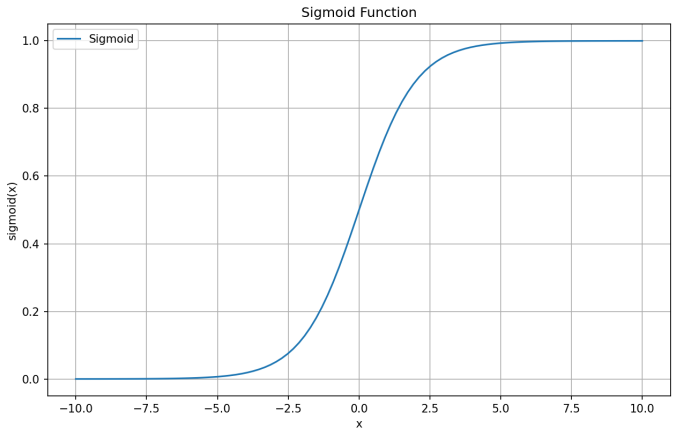
Do you want to visualize something with Python? Matplotlib is here for your need! Matplotlib is a Python library for visualization. Let's first import the library to take a look at an example.

```
import matplotlib.pyplot as plt
%matplotlib inline
```

If you are using Jupyter Notebook like the examples in these notes, it is highly recommended to keep `%matplotlib inline` to make our lives easier by plotting directly in the notebook without an external window. Although Matplotlib is an essential library for visualizing model performance and analyzing, let's stick with basic plotting for now as I believe more information would be an overkill for this note.

For this brief introduction, let's plot Sigmoid function, which we will discuss in later notes.

```python
# Define domain and the function
x = np.linspace(-10, 10, 100)
y = 1 / (1 + np.exp(-x))

plt.figure()
plt.plot(x, y, label='Sigmoid')
plt.grid(True)
plt.xlabel('x')
plt.ylabel('sigmoid(x)')
plt.title('Sigmoid Function')
plt.legend()
plt.show()
```



I know this intro was quite brief, but we will continue using amazing features in Matplotlib! For more information, please visit the official tutorial page.

In the next note, we will discuss foundational terms and concepts in ML. It will also include a high level introduction to large language model (LLM). Understanding the high level overview of how LLMs operate under the hood really helped me grasp the intuition. Without further ado, let's get started!

Note 11

Foundations and Intuitions

This note covers basic concepts as well as intuitions through the introduction of Large Language Models.

§11.1 Fundamental Terms and Concepts

This section is an introduction to basic terms and concepts that will be used throughout my notes. That being said, more advanced concepts will be introduced in latter notes. Please feel free to skip over this section if you already know the bold words.

Definition 11.1.1

Machine learning is a subset of artificial intelligence that learns from data. **Deep learning** is a subset of machine learning that utilizes neural networks.

The primary purpose of ML is to make AI "smarter". For instance, calculators are "dumb" in a sense that it requires an input and can only do certain tasks such as outputting calculated input. However, we can make it "smarter" by teaching data. We can clearly see that WolframAlpha is smarter than a normal scientific calculator. (Please note that this is just an analogy not an example because WolframAlpha is a knowledge engine, not a machine learning model. But I mean you get the point.)

On the high level, you can think of an AI model as a giant function with billions of array of numbers, called parameters.

Definition 11.1.2

Parameters are the internal values in a model composed with **weights** and **biases** that are learned by the model. After the parameters are finalized with the training phase, the models generate **predictions** through the process called **inference**.

Indeed, models are prediction machines based on the trained parameters. One fun fact here

is that it generally costs way less to run inference than to train the model. More in-depth explanation will be available in latter notes where we discuss neural nets.

11.1.1 ML Paradigms

In ML, there are three primary learning paradigms: supervised learning, unsupervised learning, and reinforcement learning. First, what does it mean to "learn" and "train"?

Definition 11.1.3

Machine learning utilizes the process of **training** to learn from data with algorithms. A common example would be dog and cat recognition from picture with **labeled** data.

There are multiple ways in which models are trained. One of the most used method out of the three paradigms is supervised learning.

Definition 11.1.4

When a model is trained with **supervised learning**, the model utilizes labeled dataset, or examples of input-output pairs, to learn.

With supervised learning, there are two primary predictions that models make.

Definition 11.1.5

Classification is a prediction where the label of an input data is being predicted. **Regression** is utilized to predict continuous values.

Classic examples of classification would be categorizing dog/cat pictures and email spam filtering. An example of regression would be the house price over years.

The second most common learning paradigm is unsupervised learning.

Definition 11.1.6

Unsupervised learning is a process of training a model with unlabeled data.

How is it possible for a model to learn without labeled data? One primary example algorithm is k-means clustering, where the model categorizes the data with relevance and similarity. To cluster high dimensional data, we often perform **dimensionality reduction** to reduce the dimension for clustering with the trust in **manifold hypothesis**. I won't go to deep into unsupervised learning and k-means clustering in this note, but you can check out this awesome visualization of k-means clustering.

The third most used learning paradigm is reinforcement learning.

Definition 11.1.7

Reinforcement learning is a process of training models through interactions with environment and reward system.

Personally, reinforcement learning is more complicated compared to the two paradigms introduced above. On the high level, an agent will take an action in an environment, where it returns **rewards** based on the action. The agent will then update its policies with the provided reward. The process is continued until the "adequate" amount is reached.

Training a model can be understood as writing a best fit mathematical function to training datasets that captures the holistic trend. There are few terms that describe the function.

Definition 11.1.8

A model is **overfit** if it learns too much from the data that it does not effectively capture that holistic pattern but small **noises**. On the other hand, an **underfitting** model is formed when it does not learn effectively from the data.

On the high level, our goal is to generalize and capture the general pattern and prevent overfitting or underfitting. We normally would stop training a model when it does not appear to improve, or is degrading, but how do we assess it?

Definition 11.1.9

To prevent overfitting and to evaluate the model during training, **validation** is utilized with **unseen data**. Unlike validation, **test sets** are used at the end of the training to evaluate the finalized model.

We have discussed the fundamental terms and concepts that would repeatedly appear in the following notes. Now let's build our high level intuitions on ML with the introduction to LLMs!

§11.2 Introduction to Large Language Models

You probably have heard of Large Language Models (LLMs) and even have your favorite ones. You may have also noticed that if you and your friend ask a same question to a LLM, you get different response, or at least outputs that are not identical from word to word. That is because LLMs are generative models. As always, let's get started by getting used to few vocabs.

Definition 11.2.1

Generative models are models that predicts and estimates the values on a probability distribution.

A generative model is also capable of taking in sequential data for a more context-rich prediction. An example of generative models would be an autoregressive model.

Definition 11.2.2

An **autoregressive model** is a generative model that predicts the future values in a sequence given the past input sequence.

For instance, when given the sequence of integers $0, 1, 1, 2, 3, 5, 8$, an autoregressive model is likely to generate 13 by using previously learned pattern and the input. In other words, it is simply a model that predicts the term $P(x_n \mid x_1, \ldots x_{n-1})$.

Definition 11.2.3

A **language model** is a model that utilize human languages and predicts the next tokens given the previous inputs.

An interesting fact about such models is that they sometimes exhibit **emergent behavior** where they actually "learn" and exhibit capabilities that were not explicitly present in the training data.

11.2.1 Tokenization

We know that at the end, language models are huge machines that predict the next numerical values. Then how do such models generate outputs in human language? This is where **tokenization** kicks in.

Definition 11.2.4

Tokenization is a process of breaking down inputs into smaller units called **tokens** which are often represented as numeric values.

To tokenize inputs, a tool called **tokenizer** is used with **Byte Pair Encoding (BPE) algorithm**. Intuitively, what we could do is assign numbers to each alphabets like 1 for a, 2 for b, and so on. However, this would be very inefficient considering the frequency of each letter and scalability. Instead, what we can do with the BPE algorithm is create a byte-efficient list of merged chunks (that are not necessarily a single letter) based on frequency. Here is an example from the OpenAI Tokenizer for GPT-5.x & O1/3.

Consider the following sentence:

Hippopotomonstrosesquippedaliophobia means fear of long words.

When tokenized, this 62 characters sentence turns into 17 tokens as following.

['H', 'ipp', 'opot', 'omon', 'st', 'ros', 'es', 'qu', 'ipped', 'ali', 'ophobia',
' means', ' fear', ' of', ' long', ' words', '.']

The token IDs of the tokens above are the following numbers.

[39, 3012, 101543, 43960, 302, 2199, 268, 351, 8193, 3010, 141868,
4748, 11747, 328, 1701, 6391, 13]

Now, to help the models "understand" the tokens and take the IDs as input, we generally turn them into vectors through the process of **One-Hot Encoding (OHE)**, though the process may differ in modern Deep Learning for efficiency.

With OHE, the token ID i can be translated to a vector with 1 in index i and 0 in other slots. However, notice that OHE itself is not highly effective. To make the tokenizer more "meaningful" and "effective", we could use the process of embeddings. **Learned embedding vectors** uses somewhat low dimensional vectors (usually a few thousand dimensions) to make the vectors "meaningful". For example, in learned embedding vector, the difference between the vectors that represent king and queen will be similar to the difference between the vectors that represent man and woman. Modern models use **relative positional embeddings** to effectively distinguish the position of the words. (Let's save this topic for latter notes on transformers.)

11.2.2 Language Model Output

As you probably have guessed, language models do not simply output the texts that we see. An autoregressive transformer language model will output a sequence of multi-dimensional vectors with probability distribution for the next token based on the sequence of multi-dimension input vectors.

Definition 11.2.5

Logits are the **scores** of the next possible token represented by individual vectors in the output sequence.

Note that logits are *raw* scores. To convert them to probability distribution, we use something called **softmax**, in which we will discuss further in notes about neural networks.

Definition 11.2.6

Decoding is the process selecting the next token to convert raw and abstract model output into a human language with **detokenization**.

There are different strategies in decoding, but the two main methods are **greedy decoding** and **sampling**. In greedy decoding, you select the token with the highest probability. Meaning, the temperature (we will cover this is neural nets notes) does not influence the selection. Unlike greedy decoding, temperature does influence the result from sampling. In sampling, you randomly select the next token with accounting the individual probability. To put it simply, imagine you have a biased die with thousands of sides. You roll the die with the bias in each side, and the next token is selected by the result.

Model outputs are random and highly organized. Take a look at the following example. In the first case, say the model input is "I love" and the expected model output is "I love potatoes". In the second case, the model input is "Veni, vidi," and the expected output is "Veni, vidi, vici". When the model returns an output, which case do you think the model has the higher probability of returning the expected output? To answer this question, we must understand that models are composed of billions of parameters that are trained with countless data. The model likely learned lots of texts that includes "I love dogs", "I love coffee", and more. However, for a model dedicated for English, it is highly unlikely that it learned another phrase that starts with "Veni, vidi," other than "Veni, vidi, vici", the famous line from Julius Caesar. The model probably assigned lots of logits and converted them to probability distribution with softmax. While the first case have somewhat flat distribution, the second case will likely assign "vici" with high probability like 0.98 or 0.99. Therefore, the second model has the higher probability of returning the expected output.

Now that we have some intuitions on model outputs, let's discuss about post-training.

§11.3 Improving Large Language Models

Up until now, we have discussed fundamental terms and very high-level overview of pre-training phase. This section discusses very high-level introduction to how a raw autocomplete model transitions to a model that we use in daily life.

11.3.1 Introduction to Post-Training

One of the most important phase of training that makes the chatbots chatbots are post-training. When a base model is developed through pre-training phase, it is simply an autocomplete model. Whereas, an aligned model is capable of generating conversation-like responses. For instance, if the input is "Why are dogs so cute?", the base model will generate responses like "What are cats doing?", which does not feel like a conversation. Whereas, an aligned model will generate responses like "People find dogs cute because humans developed a biological phenomena called baby schema."

The post-training process contains two primary phases: Supervised Fine-Tuning (SFT) and Reinforcement Learning from Human Feedback (RLHF) On the high-level, SFT is the process where human provides prompt and response samples. The model then learns from the samples to provide a chatbot-like experience. After the phase, the model can be reinforced through RLHF. There are multiple ways of doing this, but a common method is making another "reward model" to judge and provide feedback on a model's response. Instead of a human manually judging and rewarding every responses, reward model will predict how a human expert will reward (usually an integer from 0 to 5) a response to more effectively align the model. The specific process is quite out of scope for this note, and will be introduced in the following notes.

Most of the "learning" happens in pre-training. However, alignment is necessary to improve HHH, or helpfulness, honesty, and harmlessness. And yes, SFT and RLHF are popular, but they

are not the only methods. Few other examples contain LIMA, RLAIF, DPO, PPO, and URIAL.

11.3.2 Reasoning

As you probably have noticed, LLMs are predictions models. In other words, they do not "reason" like how us humans reason. Language models will predict the next likely token that will provide a plausible response to a reasoning task. One way to test if the models can actually reason is by using benchmarks. One of the foundational datasets, which is considered outdated now, is GSM8K by OpenAI. This dataset is used to evaluate if a model can reason grade school level math. The modern, highly challenging, benchmark includes FrontierMath. As you have noticed, benchmarking and reasoning is a giant hurdle for many LLMs. Modern engineers classify reasoning as another section in post-training to improve a model's logic.

Prompting

One of the most effective way to improve LLM reasoning is to use prompting strategies.

Definition 11.3.1

One technique is **K-shot prompting** where you provide k examples for the model to follow a template. Zero-shot prompting will be when $k = 0$, where you provide no examples, but instructions. The second prompting strategy is **Chain-of-Thought (CoT) prompting**. This strategy explicitly tells a model to break a complex reasoning tasks into intermediate steps.

Let's take a look at an example on how K-shot prompting and CoT prompting work together.

Model Input

Instruction:

Consider the following examples to provide a response to the final equation. Provide a step by step explanation for your answer as demonstrated in the samples.

Example 1:

Q: Henry the dog loves tomatoes. He originally had 5 tomatoes. Henry ate 3 of them, and sneaked into the owner's yard to secretly bring 4 more. How many tomatoes does Henry have now?

A: Henry started with 5, but ate 3. Therefore, he has $5 - 3 = 2$ tomatoes left. Because he brought 4 more from the owner's yard, he now has $2 + 4 = 6$ tomatoes. Henry has 6 tomatoes now.

Example 2:

Q: Bob the yak loves burgers. One day, his lizard friend Sofia told him that she can bring

three boxes of burgers than contain 2 burgers each. Bob instantly pleaded Sofia, and he received the boxes. Out of hunger, he ate three burgers. How many burgers does he have now?

A: It is assumed that Bob does not have any burgers before the interaction with Sofia. With Sofia's virtue, Bob now has $2 \cdot 3 = 6$ burgers. Because he ate three burgers from six in total, Bob now has $6 - 3 = 3$ burgers left.

Question:

Charlotte the cat has five close friends. She wishes to split her 120 kg tuna equally with her friends. After the allocation, Charlotte ate 4 kg of tuna. How much tuna in kg does Charlotte have left?

Model Output

Charlotte initially has 120 kg of tuna. Because she split it with her friends, she has $\frac{120}{5+1} = 20$ kg of tuna after the allocation. Since Charlotte ate 4 kg of tuna from 20 kg, she now has $20 - 4 = 16$ kg of tuna.

As demonstrated, CoT has the potential to increase reasoning of LLMs. However, it is important to note that although performance may significantly ameliorate, numerous potential consequences follow. For instance, because the model includes the intermediate steps, the token generation considerably increases, which will increase the cost. Moreover, when model starts to hallucinate in the middle of the reasoning, the entire output from the hallucination will no longer be useful. Furthermore, models sometimes "lie". What I mean to lie is that some models tend to provide a "plausible CoT" without actually providing one. This is more related to AI safety, and let's save that for final notes.

Self-Consistency

Another technique that could improve reasoning and performance is **self-consistency**.

Definition 11.3.2

Self-consistency is a technique where a model generates numerous possible outputs to an input, and aggregate the outputs for the final output.

The primary motivation of the technique is that just as mathematical problems have numerous paths to an answer, we could derive an "accurate" output with different reasoning. However, like any other techniques, there are limitations. One somewhat self-evident caveat is that this requires much greater computation. Because the technique requires multiple reasoning and aggregation for the final, the amount of computation drastically increase. One other issue is that this self-consistency only works on closed-solution prompts. I mean different people and different outputs can have differing opinions. Although this technique is not fully outdated, modern ML engineers use more effective and efficient techniques.

The field of reinforcement learning is one of the most rapidly developed field in ML. For instance, unlike RLHF, the emerging method Reinforcement Learning with Verifiable Rewards (RLVR) allows models to align themselves with minimum human contributions. In the next note, we will discuss about neural network from scratch!

Note 12

Deep Dive into Deep Neural Networks

Neural networks (or neural nets) can be comprehended as a complex function that does not require manual feature engineering. Due to the high level of abstraction and automation in learning, neural nets are highly scalable. This note covers neural nets with mathematical interpretations and its application with Python.

§12.1 Definitions and High Level Overview

As always, let's get started with the fundamental terms and high level overview of what neural nets are. Before the introduction of neural nets, traditional models like linear/logistic regression and decision trees were used. The problem with traditional models is that you are likely required to manually adjust the **features**. With neural nets, the features are automatically engineered, making us to focus on the architecture.

12.1.1 High Level Architecture

To understand the high level architecture, consider the diagram below.

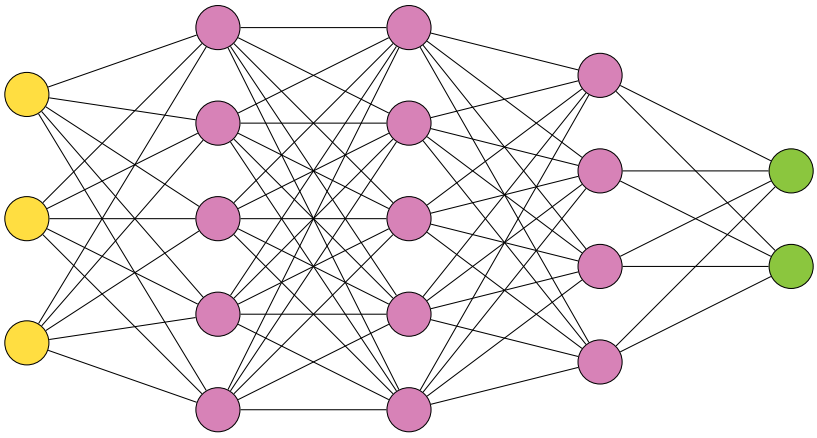


Figure 12.1: A visualization of a vanilla neural network (MLP)

From now, let's refer to the nodes as **neurons**.

Definition 12.1.1

Neurons are the fundamental unit in neural networks that process input into output with set of parameters.

From the diagram, the neurons are organized in to different columns. We call the columns **layers**.

Definition 12.1.2

The **input layer** is the layer that receives inputs for the neural network. **Hidden layers** are where the input data is processed with the learned parameters to extract information. The **output layer** is the final layer that represents the final prediction of the neural network.

In the diagram above, the input layer is represented as the layer with yellow neurons, hidden layers as the layers with purple neurons, output layer with the layer with green neurons. One of the advantages of neural network is that there is simply no limit, except computational limit. You can have unlimited amount of neurons in each layers and unlimited amount of hidden layers. The best neural net will be effectively structured in a way that there are sufficient neurons to learn from while avoiding overfitting and decreasing computational cost.

Definition 12.1.3

Connections are the links that connects the neurons with each other.

Just as our biological neurons are interconnected to each other, artificial neurons are connected to effectively extract the holistic feature from the input. Now it is worth noting that each neuron represents **feature**.

Here is a brief example. Say your neural net will classify handwritten numbers. The number 9 and 0 retains a similar feature. Both the numbers have a loop in the top. Therefore, in this scenario, the initial hidden layers may have similar features when categorizing the numbers.

In the past we used **perceptrons**, or neurons that uses step functions with binary output. However, modern multilayer perceptron (MLP) uses a more flexible neurons with continuous activation functions. Now let's take a look at what individual neurons have with the diagram below.

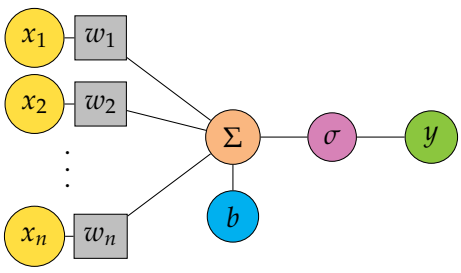


Figure 12.2: Visualization of a single neuron

On the high level, an individual neuron takes the weighted sum of the previous layer with the trained parameter. Bias is then added to make the activations more flexible. Then, an activation function is used to determine the final activation of the neuron.

Definition 12.1.4

Activation of a neuron indicates the significance of a neuron, or presence of a feature from the input data. An **activation function** is a function that generalizes each activation and introduce **non-linearity**.

In the diagram above, the purple color represents the activation function and the green color represents the final output of the neuron. Here, non-linearity is important for generalization with activation function. The activation is represented as a linear weighted sum. However, for generalization, activation functions like ReLU (Sigmoid is considered outdated) are used.

Here, the superscript represents the layer and i identifies the neuron in the layer. The function σ is an activation function. It could be sigmoid, ReLU, tanh, and other activation functions.

12.1.2 Training

Inference is rather straight forward, but how do we "train" a model? Well, that's what this part of the note is all about. There are few terminologies that we need to know, and I think we can discuss them as we go along.

Definition 12.1.5

Forward pass, also known as **forward propagation**, the process of generating output with input data.

Here, it is worth noting that forward pass is different from inference in a sense that the process of inference only applies to a complete trained model while forward pass also applies to models being trained.

Let's discuss the definitions and underlying concepts in individual process after capturing the big picture. Generally speaking, the following diagram captures the training process.

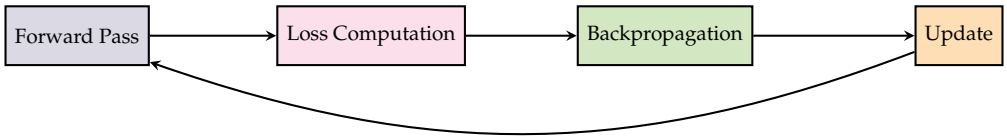


Figure 12.3: Overview of the Training Process

The first thing that we do after the forward pass is to calculate the **loss**.

Definition 12.1.6

Loss functions are algorithms that calculate the **loss**, or the performance of the prediction compared to the expected outcome.

There are many loss functions, but the two dominant functions are **mean squared error (MSE)** and **cross-entropy** (a.k.a. negative log-likelihood.) Generally, MSE is great for regression and cross-entropy excels at classification. Therefore, cross-entropy will commonly be used for LLMs for token predictions. Let's take a look at each functions.

Following the convention, let $\hat{y}_n$ and y_n be the prediction and the true value respectively. With MSE, the loss will be represented as $\frac{1}{n}\sum(\hat{y}_i - y_i)^2$. The intuition behind this formula is we always want the loss to be positive as it is comfortable to think that our goal is to reduce the loss. If the loss is negative, it will be hard to apply this sense. That's why we squared the difference. Now you might ask, why not absolute value? Indeed, there is a function Mean Absolute Error (MAE), but we like to square the values as it amplifies the loss. With cross-entropy, its a bit more complicated, but a simplified version would be $\sum(-y_i \log(\hat{y}_i))$. More on cross-entropy will be discussed soon!

We know we can calculate the loss, but how do we actually improve the model with the calculated loss?

Definition 12.1.7

An **optimizer** is an algorithm that determines the necessary changes for model weights.

The dominant optimizer type is **gradient descent** with its variants like Adam, RMSProp, and SGD. There's quite a bit of content for gradient descent for ML, so let's discuss that in a separate section of the note.

12.1.3 Gradient Descent and Learning Rate

Before we implement the ideas and concepts above with code, I wanted to discuss gradient descent and training more in-depth to actually train our model. The partial derivatives and mathematical definition of gradient descent was discussed in the first note. Here, we will discuss about gradient descent specifically for ML.

Definition 12.1.8

Gradient descent as an optimizer represents how the weights and biases of a model are updated based on the gradients of the loss function.

You would typically start from a random position to move in the opposite direction of the gradient of the loss function. Before I introduce mathematical equation, here are two famous analogies of a ball rolling down the hill and finding a way in a foggy forest. Starting with the first analogy, consider the diagram below.

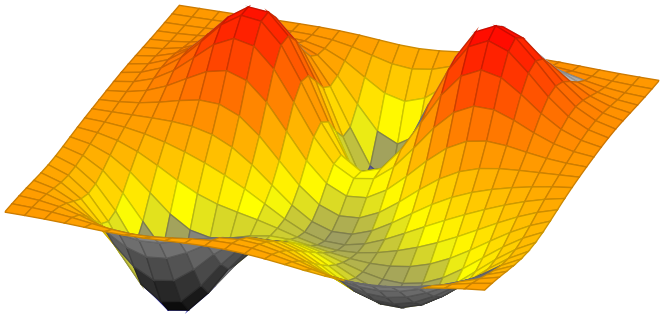


Figure 12.4: Ball Rolling Downhill Analogy

Imagine rolling a ball from the graph above. After some time, it will naturally fall into one of the pits. If we are lucky, we will find the global minimum, if not, we will fall into one of the local minima. To see how we roll a ball, consider two diagrams below.

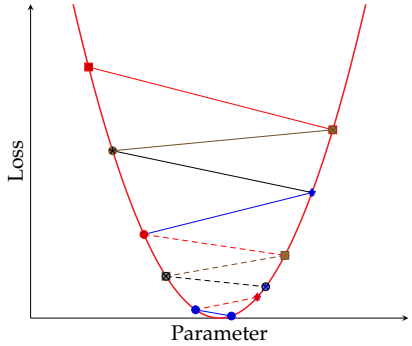


Figure 12.5: Visualization of Case I

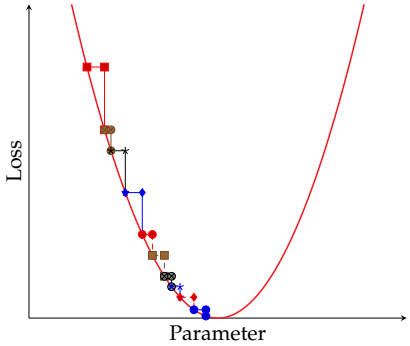


Figure 12.6: Visualization of Case II

Taking a look at Figure 12.5, imagine your random point in the red square. Because the slope of the line tangent to the point in negative, you want to move to the right, or in positive direction. This time, your learning rate is high that the slope of the line tangent to the next landing position is positive. Therefore, you want to move to the left, which is the negative direction. After continuing this process over and over again while adjusting the learning rate, you will likely find the local minimum. In Figure 12.6, our learning rate is low. Despite being low, it will still lead to the local minimum. One thing to note is that you don't want your learning rate to be low or high as you might need to train forever or overshoot. Learning rate is often denoted as η, and we normally would adjust it with the gradient, which we will see in a bit. Of course in real training, we won't be finding the slope of the tangent line as we do in the 2-dimensional plane. In higher dimension, we would find the gradient. Please refer to the first note for more information!

Another very popular analogy is foggy mountain analogy. Imagine you are in the top of the mountain, and its getting dark. You want to get to the lowest point as fast as possible. However, you can't see anything beyond your local information because the mountain is very foggy. What would you do? We would find the gradient and multiply it with the learning rate to determine our step size. Then, with your local information, we would be able to approach the local minimum of the foggy mountain. Ideally, we should be hoping that we reach the lowest

point and villages.

Now, before we continue for building our own neural net without machine learning libraries, let's conclude the concepts by discussing different types of gradient descent and their effects.

Types of Gradient Descent

There are many types of gradient descents like Batch GD, Mini-batch GD, Stochastic GD, and Momentum-Based GD. However, the three major types are Batch Gradient Descent (BGD), Stochastic Gradient Descent (SGD), and Mini-batch Gradient Descent. On the high level, consider the following definitions.

Definition 12.1.9

With **Batch Gradient Descent (BGD)**, each update will occur based on the entire training examples, which are often averaged. In contrast, **Stochastic Gradient Descent (SGD)** updates based on a single data point. **Mini-batch Gradient Descent** is the middle ground of BGD and SGD that it updates based on few data points like 32 or 64, usually in a power of 2.

From the sense, we can notice that the biggest difference in use case would be efficiency and accuracy. BGD is generally more accurate in each update than SGD as it is based on the entire data points. However, SGD is generally more efficient as it does not have to perform massive computations for each update to incorporate all the training examples. One interesting thing to note is that because SGD is "stochastic", or more noisy, it is sometimes better at optimizing the updates as it moves more randomly than BGD.

When visualizing, we often consider the following diagram.

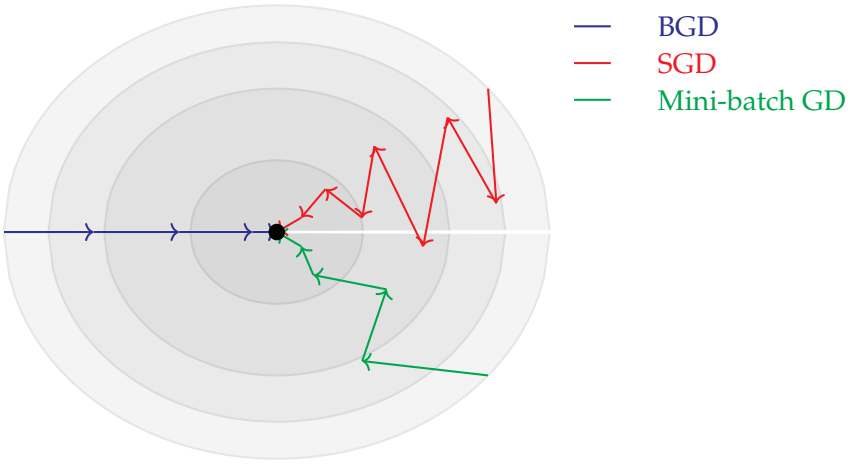


Figure 12.7: Visualization of Different Types of Gradient Descent

As you can see, BGD often heads straightly towards the minimum as the entire training examples are incorporated. Unlike BGD, SGD shows more random and noisy movement and often requires more updates to reach the local minimum. Finally, Mini-batch GD is like the

movement in between those two. Its is some what random, but less noisy SGD that it often requires less updates.

Overall, this is what artificial neural nets are about. The pre-trained and inter-connected neurons work together to predict that output as a giant function. This giant function are composed of weights and biases, which are fine-tuned during the training process. During training, you calculate the loss and tweak the parameters to minimize the loss through gradient descent. You repeat the training process until your model learned well enough, but not too much to prevent overfitting. That's pretty much it in the high level! For our next section, let's discuss more on training and gradient descent before we actually try and build a neural network from scratch without ML libraries like PyTorch and TensorFlow.

Congratulations! We just finished the high level overview of what neural nets are and what vanilla neural nets are composed of. Before we actually try and implement the concepts with Python to build a vanilla neural net without machine learing libraries like PyTorch and TensorFlow, let's take a look at mathematical equations to write them in code.

§12.2 Conventions and Mathematical Interpretations

If you have not already read the first note on math for ML, please do so as this section of the note does not include explanations for mathematical concepts, but their interpretation here with neural networks.

Starting off, let's start with some conventions with parameters for the following neural net with two inputs, two hidden layers with four neurons respectively, and a single output.

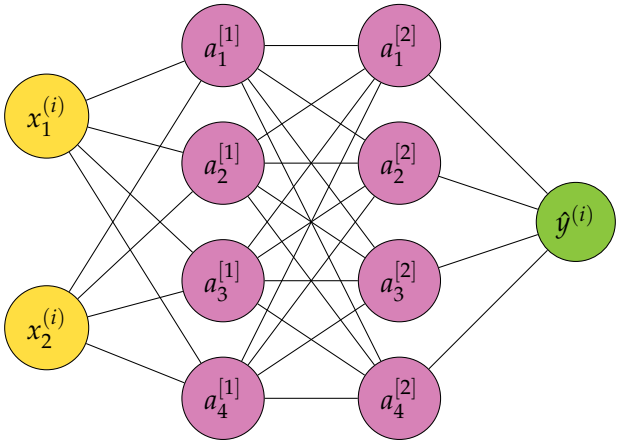


Figure 12.8: A visualization of a neural network with layers of 2, 8, 8, and 1

As you can see in the diagram above, the input of the i^{th} training example is represented with the notation $x_n^{(i)}$ for integer n as 1 through the total number of inputs. Similarly, we usually count the first hidden layer as the first layer, and the activations are represented as $a_n^{[l]}$ were l represents the layer and n again as the integer from 1 through the total number of neurons in the layer. Finally, we use $\hat{y}^{(i)}$ to represent the prediction for the i^{th} training example.

Following the convention, we could write a single neuron as the following. Note that you might be familiar with σ instead of g, but here let's use g for activation function as σ usually represents sigmoid.

Theorem 12.2.1: Formula for a Neuron

For $z_i^{[l]}$ defined as the following, the activation of a neuron can be represented as the following.

$$z_i^{[l]} = \sum_j w_{ij}^{[l]} a_j^{[l-1]} + b_i^{[l]}$$
$$a_i^{[l]} = g\left(z_i^{[l]}\right) = g\left(\sum_j w_{ij}^{[l]} a_j^{[l-1]} + b_i^{[l]}\right)$$

One reason why linear algebra is so important in machine learning is because we could represent the entire layer with simple matrix operations. Consider the following example for the second layer.

$$\begin{bmatrix} w_{11}^{[2]} & w_{12}^{[2]} & w_{13}^{[2]} & w_{14}^{[2]} \\ w_{21}^{[2]} & w_{22}^{[2]} & w_{23}^{[2]} & w_{24}^{[2]} \\ w_{31}^{[2]} & w_{32}^{[2]} & w_{33}^{[2]} & w_{34}^{[2]} \\ w_{41}^{[2]} & w_{42}^{[2]} & w_{43}^{[2]} & w_{44}^{[2]} \end{bmatrix} \begin{bmatrix} a_1^{[1]} \\ a_2^{[1]} \\ a_3^{[1]} \\ a_4^{[1]} \end{bmatrix} + \begin{bmatrix} b_1^{[2]} \\ b_2^{[2]} \\ b_3^{[2]} \\ b_4^{[2]} \end{bmatrix} = \begin{bmatrix} a_1^{[2]} \\ a_2^{[2]} \\ a_3^{[2]} \\ a_4^{[2]} \end{bmatrix}$$

Similarly, we could write different notations for updates.

Theorem 12.2.2: Adjusting Parameters with Gradient Descent

Let l_i, b_i, J, and η represent weight, bias, cost function, and learning rate respectively. The new weight and bias can be represented with the following formula.

$$w_2 = w_1 - \eta \frac{\partial J}{\partial w_1}$$
$$b_2 = b_1 - \eta \frac{\partial J}{\partial b_1}$$

As you can see above, we normally use the cost function J than the loss function L as the cost function captures the entire loss while the loss function only represent the loss per example. Now that we discussed the basic notations and mathematical formulas for the concepts that we discussed, let's take a look at math in backpropagation.

12.2.1 Deriving Equations and Backpropagation

Personally it took me a while to fully understand backprop to the state where I would be able to build a vanilla neural net from scratch. Here, I would like to share more in-depth understanding of backpropagation across numerous layers.

To be honest, it took me few days to understand it since I decided to study mainly because I didn't understand the delta rule. After reading this note, I highly recommend please try and write your own network without assistance from LLMs. I though I understood backprop, which in reality was only the last layer. That understanding is probably all you need to build the first thing that we will create in the next section. However, I was stuck when trying to do backprop across numerous layers. To discuss this, I thought it would be helpful for us to actually rebuild or derive main equations involved in a neural network and backprop, so let's start with that. If you haven't already, please read the first note if you are unfamiliar with derivatives of multivariable functions as I won't be going over them here.

Before we derive generalized equations, let's start with a simple neural net that has two hidden layers, 2 inputs, 3 neurons in each hidden layer, and two outputs. That way, we can track the indices for matrices.

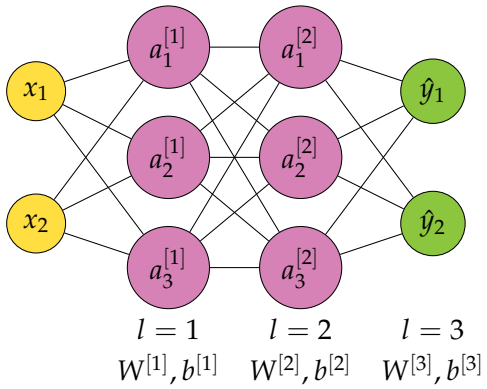


Figure 12.9: A visualization of a neural network with layers of 2, 3, 3, and 2

For simplicity, let's use MSE for our loss function, but we will be generalizing it later. Since $L_1 = \frac{1}{2}(\hat{y}_1 - y_1)^2$ and $L_2 = \frac{1}{2}(\hat{y}_2 - y_2)^2$, the following equation is obtained.

$$L = \sum_{k=1}^{2} \frac{1}{2}(\hat{y}_k - y_k)^2$$

Moreover, the preactivation output of the second hidden layer can be written as following.

$$\begin{bmatrix} z_1^{[2]} \\ z_2^{[2]} \\ z_3^{[2]} \end{bmatrix} = \begin{bmatrix} w_{11}^{[2]} & w_{12}^{[2]} & w_{13}^{[2]} \\ w_{21}^{[2]} & w_{22}^{[2]} & w_{23}^{[2]} \\ w_{31}^{[2]} & w_{32}^{[2]} & w_{33}^{[2]} \end{bmatrix} \begin{bmatrix} a_1^{[1]} \\ a_2^{[1]} \\ a_3^{[1]} \end{bmatrix} + \begin{bmatrix} b_1^{[2]} \\ b_2^{[2]} \\ b_3^{[2]} \end{bmatrix}$$

From the matrix multiplication above, we can write a single output as the following.

$$\begin{aligned} z_2^{[2]} &= w_{21}^{[2]} a_1^{[1]} + w_{22}^{[2]} a_2^{[1]} + w_{23}^{[2]} a_3^{[1]} + b_2^{[2]} \\ &= \sum_{j=1}^{3} w_{2j}^{[2]} a_j^{[1]} + b_2^{[2]} \end{aligned}$$

Now keep in mind the indices! The 2 in w_{2j} and b_2 is the same 2 as z_2. Generalizing for our cases, we can write the preactivation outputs as the following.

$$z_i^{[2]} = \sum_{j=1}^{3} w_{ij}^{[2]} a_j^{[1]} + b_i^{[2]}$$

Nice, now that we have the forward pass for our examples, let's generalize all equations that we have now. For MSE, we could write it as the following.

$$L = \sum_k \frac{1}{2}(\hat{y}_k - y_k)^2$$

For individual preactivation output, we could write it as the following equation where n represents the number of neurons in layer $l-1$.

$$z_i^{[l]} = \sum_{j=1}^{n} w_{ij}^{[l]} a_j^{[l-1]} + b_i^{[l]}$$

Now with the equations in mind, let's get into a more confusing but fun part with loss and gradient. This part is important! We are not computing the loss of individual neurons. We will be computing *gradients* instead. Loss will typically be calculated once with the final outputs.

Let's start with the last layer. Remember! This is the last layer. To make it more special, let's use l_f to denote the final layer.

$$\frac{\partial L}{\partial w_{ij}^{[l_f]}} = \frac{\partial L}{\partial a_i^{[l_f]}} \frac{\partial a_i^{[l_f]}}{\partial z_i^{[l_f]}} \frac{\partial z_i^{[l_f]}}{\partial w_{ij}^{[l_f]}}$$

In this chain rule here, we don't add other gradients because it is the last layer. The weights for the output layer does not go through other paths, but straight to the loss. That's why we don't add gradients. This would be different for different layers. Using the definition, MSE, and activation function g, the following equation is obtained.

$$\frac{\partial L}{\partial w_{ij}^{[l_f]}} = \left(a_i^{[l_f]} - y_i\right) g'\left(z_i^{[l_f]}\right) a_j^{[l_f-1]} = \delta_i^{[l_f]} a_j^{[l_f-1]}$$

Notice that $\delta_i^{[l_f]}$ will be different depending on the loss function that we use. To address that and to make our lives easier for hidden layers, we define $\delta_i^{[l]}$ as the following.

$$\delta_i^{[l]} = \frac{\partial L}{\partial z_i^{[l]}} = \frac{\partial L}{\partial a_i^{[l]}} \frac{\partial a_i^{[l]}}{\partial z_i^{[l]}}$$

Again, because the path from z_i to a_i does not require additional paths, we do not add other gradients. Now here comes the **delta rule**. Let's start deriving them together. Consider the

following equation.

$$\begin{aligned}\frac{\partial L}{\partial a_i^{[l]}} &= \sum_j \left(\frac{\partial L}{\partial z_j^{[l+1]}} \frac{\partial z_j^{[l+1]}}{\partial a_i^{[l]}} \right) \\ &= \sum_j \delta_j^{[l+1]} w_{ji}^{[l]} = \sum_j w_{ji}^{[l]} \delta_j^{[l+1]}\end{aligned}$$

Substituting back to the definition of $\delta_i^{[l]}$, the following equation is obtained by the **delta rule**.

$$\delta_i^{[l]} = g'\left(z_i^{[l]}\right) \sum_j w_{ji}^{[l]} \delta_j^{[l+1]}$$

However, writing all that for each gradient is quite tedious. Therefore, we get our general formula shown below, which will be used to build our vanilla MLP.

$$\frac{\partial L}{\partial w_{ij}^{[l]}} = \delta_i^{[l]} a_j^{[l-1]}$$

That was for the weights. Good news is that bias is more simple! Consider the following equation.

$$\frac{\partial L}{\partial b_i^{[l]}} = \frac{\partial L}{\partial a_i^{[l]}} \frac{\partial a_i^{[l]}}{\partial z_i^{[l]}} \frac{\partial z_i^{[l]}}{\partial b_i^{[l]}} = \delta_i^{[l]} \frac{\partial z_i^{[l]}}{\partial b_i^{[l]}} = \delta_i^{[l]}$$

Now with that formulas and learning rate η, we can update the weights and biases as shown in Theorem 12.2.2. There are more advanced concepts like Jacobian matrix for ease in representing, but this is pretty much it with backprop. Calculating the gradients and updating based on the loss.

§12.3 Building Vanilla Neural Network From Scratch

For our toy examples, we will create two things. The first one is technically not a neural network as it does not contain conventional neurons and structures. The second one is an actual neural net that is very simple. In this section, I will share the entire code first, then provide explanations.

12.3.1 Simple Stochastic Gradient Descent-Based Learning Engine

What we will make now is a simple stochastic gradient descent-based learning engine. This isn't technically a neural network, as you will see why in the code, but I decided to include it here as making similar examples really helped me understand backprop with code.

Our task with this learning engine is simple. We are going to predict e^x for limited range with concatenated Maclaurin Series. If you recall from your calculus courses, the following equation

holds by Taylor series.

$$e^x = \sum_{n=0}^{\infty} \frac{1}{n!} x^n = 1 + x + \frac{x^2}{2!} + \frac{x^3}{3!} + \cdots$$

Here, for limited x, we will be approximating e^x with the following polynomial where b is the bias and w_i are the weights.

$$e^x \approx b + w_1 x + w_2 x^2 + w_3 x^3 + w_4 x^4 + w_5 x^5 + w_6 x^6 + w_7 x^7 + w_8 x^8$$

We will get back to this on why our x should be limited, but let's see our structure first.

We will be "training" the engine with 200 examples from -1.00 to 0.99 inclusive. Then, we will be testing it with 50 validations from 1.00 to 1.49 inclusive. This backprop machine is stochastic as it will adhere to the following structure.

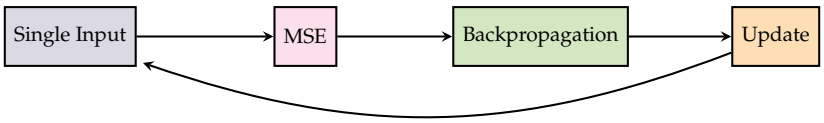


Figure 12.10: Overview of the Simple Learning Engine

As you can see, we will be using MSE for our loss function, and we do not need activation functions like sigmoid and tanh because we don't need one in our simple example. Okay enough talking and here is the entire code.

Source Code

Below is the source code for the Simple Stochastic Gradient Descent-Based Learning Engine for predicting e^x.

```
import math
import random
import matplotlib.pyplot as plt
%matplotlib inline
```

```
train_x = [i/100 for i in range(-100, 100)]
train_y = [math.exp(i/100) for i in range(-100, 100)]
test_x = [i/100 for i in range(100, 150)]
test_y = [math.exp(i/100) for i in range(100, 150)]
```

```
class SGDBLE:
    def __init__(self):
        random.seed(3141592) # So that we get the same result
```

```
        self.ws = [random.uniform(-1, 1) for _ in range(8)]
        self.b = random.random()

    def forward(self, x):
        self.ypred = sum(w * x**k for w, k in zip(self.ws,range(1, 9)
            )) + self.b
        return self.ypred

    def loss(self, y, ypred):
        self.loss_val = 0.5 * (y - ypred)**2
        return self.loss_val

    def grads(self, x, y, ypred):
        base_der = ypred - y # i.e. dL / dypred
        self.grads_w = [base_der * x**i for i in range(1, 9)]
        self.grad_b = base_der

    def update(self, lr):
        for i in range(8):
            self.ws[i] -= lr * self.grads_w[i]
        self.b -= lr * self.grad_b

    def train(self, epochs, lr):
        losses = []
        for epoch in range(epochs):
            loss_total = 0
            for i in range(len(train_x)):
                ypred = self.forward(train_x[i])
                loss_total += self.loss(train_y[i], ypred)
                self.grads(train_x[i], train_y[i], ypred)
                self.update(lr)
            loss_avg = loss_total / len(train_x)
            losses.append(loss_avg)

            if (epoch+1) % 100 == 0:
                print(f"Epoch: {epoch+1}/{epochs}, Loss: {loss_avg}")

        return losses

    def test(self):
        loss_total = 0
        for i in range(0, len(test_x), 2):
            ypred1 = self.forward(test_x[i])
            loss_total += self.loss(test_y[i], ypred1)

            if i + 1 < len(test_x):
                ypred2 = self.forward(test_x[i + 1])
```

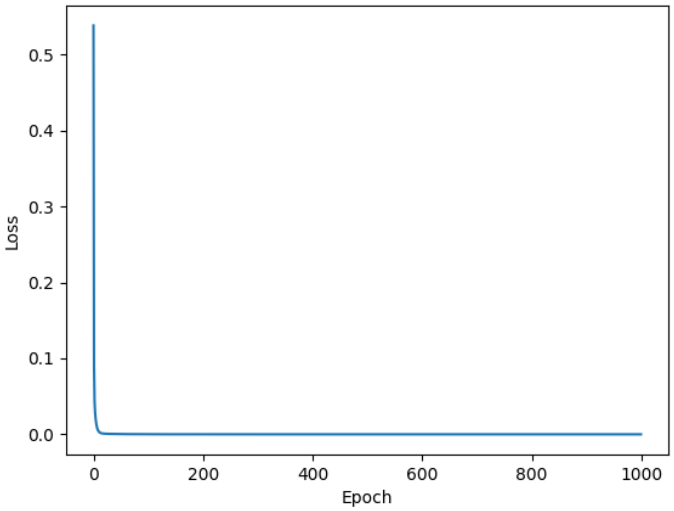
```
                loss_total += self.loss(test_y[i + 1], ypred2)

                print(f"Test {i+1}                    Test {i+2}")
                print(f"y:     {test_y[i]}      y:     {test_y[i+1]}"
                    )
                print(f"ypred: {ypred1}     ypred: {ypred2}")
                print()
            else:
                print(f"Test {i+1}")
                print(f"y:     {test_y[i]}")
                print(f"ypred: {ypred1}")

        loss_total /= len(test_x)

    def print_params(self):
        coeff = [1 / math.factorial(i+1) for i in range(8)]
        for i in range(8):
            print(f"w{i+1}: {self.ws[i]}, Expected: {coeff[i]}")
        print(f"b: {self.b}, Expected: 1")

model = SGDBLE()
```

```
def graph_loss(losses):
    plt.plot(losses)
    plt.xlabel("Epoch")
    plt.ylabel("Loss")
    plt.show()

def graph_pred():
    xs = [i / 10 for i in range(-100, 200)]
    true = [math.exp(x) for x in xs]
    pred = [model.forward(x) for x in xs]

    plt.plot(xs, true, label="e^x")
    plt.plot(xs, pred, label="Prediction")
    plt.yscale("log")
    plt.legend()
    plt.show()
```

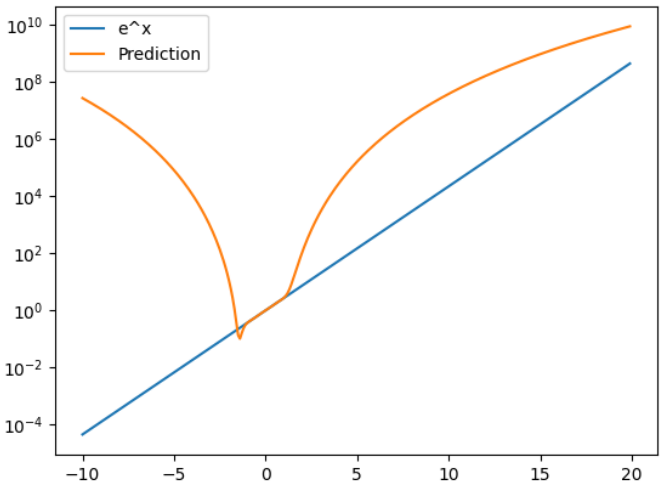
```
losses = model.train(1000, 0.005)
graph_loss(losses)
model.print_params()
graph_pred()
```

```
Epoch: 100/1000, Loss: 0.00013234985876534958
```

```
Epoch: 200/1000, Loss: 3.427320321762344e-05
Epoch: 300/1000, Loss: 2.4492707827970274e-05
Epoch: 400/1000, Loss: 2.2669862503550263e-05
Epoch: 500/1000, Loss: 2.172205317266906e-05
Epoch: 600/1000, Loss: 2.0984248749352728e-05
Epoch: 700/1000, Loss: 2.036538265870468e-05
Epoch: 800/1000, Loss: 1.9834658817269138e-05
Epoch: 900/1000, Loss: 1.937270927247297e-05
Epoch: 1000/1000, Loss: 1.8965386261105253e-05
```




```
w1: 0.9444454598562942, Expected: 1.0
w2: 0.49986306356857685, Expected: 0.5
w3: 0.620431907367758, Expected: 0.16666666666666666
w4: 0.21618569290559692, Expected: 0.041666666666666664
w5: -0.9321256839917001, Expected: 0.008333333333333333
w6: -0.48709437200615985, Expected: 0.001388888888888889
w7: 0.5594822789317027, Expected: 0.0001984126984126984
w8: 0.33002989302732855, Expected: 2.48015873015873e-05
b: 0.9983077779493585, Expected: 1
```




```
model.test()
```

```
Test 1                          Test 2
y:     2.718281828459045        y:     2.7456010150169163
ypred: 2.7495260176087557       ypred: 2.7867851769675642

Test 3                          Test 4
y:     2.7731947639642978       y:     2.801065834699079
ypred: 2.8257757734623574       ypred: 2.8666322590606814

Test 5                          Test 6
y:     2.82921701435156         y:     2.857651118063164
ypred: 2.909497162739583        ypred: 2.954521430346852

Test 7                          Test 8
y:     2.8863709892679585       y:     2.915379499976997
ypred: 3.001864774074873        ypred: 3.0516960317129964

Test 9                          Test 10
y:     2.944679551065524        y:     2.9742740725630656
ypred: 3.1041935358457007       ypred: 3.159545493165112

Test 11                         Test 12
y:     3.0041660239464334       y:     3.034358394435676
ypred: 3.2179503740678106       ypred: 3.2796173127071624

Test 13                         Test 14
y:     3.0648542032930024       y:     3.095656500124711
ypred: 3.3447665176737535       ypred: 3.4136296934778425
```

```
Test 15                      Test 16
y:     3.1267683651861553      y:     3.158192909689767
ypred: 3.4864504730090555     ypred: 3.5634848611499024

Test 17                      Test 18
y:     3.1899332761161845      y:     3.2219926385284996
ypred: 3.6450016897210107     ypred: 3.7312830839373126

Test 19                      Test 20
y:     3.2543742028896707      y:     3.287081207383118
ypred: 3.8226249405557278     ypred: 3.9193374178962683

Test 21                      Test 22
y:     3.3201169227365472      y:     3.3534846525490236
ypred: 4.02174543791974      ypred: 4.130189200546641

Test 23                      Test 24
y:     3.3871877336213347      y:     3.4212295362896734
ypred: 4.245024710403111     ypred: 4.366624316181152

Test 25                      Test 26
y:     3.4556134647626755      y:     3.4903429574618414
ypred: 4.495377262801668     ypred: 4.631690256570204

Test 27                      Test 28
y:     3.5254214873653824      y:     3.5608525623555205
ypred: 4.775988043516579     ypred: 4.928714001110956

Test 29                      Test 30
y:     3.596639725569281 7     y:     3.6327865557528094
ypred: 5.090330743550226     ypred: 5.261320740809877

Test 31                      Test 32
y:     3.6692966676192444      y:     3.706173712210199
ypred: 5.442186951657918     ypred: 5.633453470828671

Test 33                      Test 34
y:     3.7434213772608627      y:     3.781043387568781
ypred: 5.8356661905556715     ypred: 6.049393476664138

Test 35                      Test 36
y:     3.819043505366336       y:     3.8574255306969745
ypred: 6.275226859424931     ypred: 6.513781739373128

Test 37                      Test 38
y:     3.896193301795215       y:     3.9353506954704733
ypred: 6.765698108295758     ypred: 7.0316412855945165
```

```
Test 39                         Test 40
y:     3.9749016274947477       y:     4.014850052994201
ypred: 7.312302670230667        ypred: 7.6084005084606225

Test 41                         Test 42
y:     4.0551999668446745       y:     4.095955404071176
ypred: 7.9206806775719905       ypred: 8.249917485831345

Test 43                         Test 44
y:     4.137120440251392        y:     4.178699191923246
ypred: 8.596914488856152        ypred: 8.962505322624688

Test 45                         Test 46
y:     4.220695816996552        y:     4.263114515168817
ypred: 9.347554553339133        ypred: 9.752958544358306

Test 47                         Test 48
y:     4.305959528345206        y:     4.349235141062741
ypred: 10.179646340417854       ypred: 10.628580569357089

Test 49                         Test 50
y:     4.392945680918757        y:     4.437095519003664
ypred: 11.100758361572902       ypred: 11.597212287422602
```

Detailed Explanation

Now let's take a look at individual cells.

```
1 import math
2 import random
3 import matplotlib.pyplot as plt
4 %matplotlib inline
```

Here, I tried to keep it as minimal as possible. Instead of NumPy, I decided to use the math module for all math stuffs like `math.exp()`. We need `random` as it would make our lives easier than to manually set the eight weights and the bias. Matplotlib of course since we want to see how it learns for limited range of input.

```
1 train_x = [i/100 for i in range(-100, 100)]
2 train_y = [math.exp(i/100) for i in range(-100, 100)]
3 test_x = [i/100 for i in range(100, 150)]
4 test_y = [math.exp(i/100) for i in range(100, 150)]
```

Setting up the dataset for our engine is also quite simple. Instead of writing every single number,

I just decided to loop through them and store them in an array, and that is `train_x`. The values for `train_y` is also quite straightforward as it is e^x for corresponding x. The same goes for the test set. I tried to keep it 80% and 20% for training and validation, but that is not necessary and it varies by models. However, I decided to use that here because that's pretty much what most people do when they first train. Now let's get into the fun stuffs.

```
class SGDBLE:
    def __init__(self):
        random.seed(3141592) # So that we get the same result
        self.ws = [random.uniform(-1, 1) for _ in range(8)]
        self.b = random.random()
```

Now I decided to store the core functions in the class SGDBLE as we will be using `self` to store attributes and objects. Here we set the specific seed so that you will also get the same result when you run the code in somewhere. Next we set weights and biases and stored them in `self` with our set random values.

```
def forward(self, x):
    self.ypred = sum(w * x**k for w, k in zip(self.ws,range(1, 9))) +
        self.b
    return self.ypred

def loss(self, y, ypred):
    self.loss_val = 0.5 * (y - ypred)**2
    return self.loss_val
```

This is where we set some core functions. Forward is forward pass and it is somewhat straight forward as that is how we defined it. For the loss function, we are using MSE, i.e. $\frac{1}{2}(y - \hat{y})^2$.

```
def grads(self, x, y, ypred):
    base_der = ypred - y # i.e. dL / dypred
    self.grads_w = [base_der * x**i for i in range(1, 9)]
    self.grad_b = base_der

def update(self, lr):
    for i in range(8):
        self.ws[i] -= lr * self.grads_w[i]
    self.b -= lr * self.grad_b
```

This is the core part of our engine. Let's briefly look at the math and write the loop out for some

to see what's actually going on. Consider the following equation by the chain rule.

$$\frac{dL}{dw_i} = \frac{dL}{dy} \cdot \frac{dy}{dw_i}$$

Notice that $\frac{dL}{dy} = \frac{d}{dy}(\frac{1}{2}(y-\hat{y})^2) = y - \hat{y}$. This is `base_der`. Continuing with the chain rule, we can write the weights and the bias as the following.

$$\begin{aligned}
\frac{dL}{dw_1} &= \frac{dL}{dy} \cdot \frac{dy}{dw_1} = \text{base_der} \cdot x \\
\frac{dL}{dw_2} &= \frac{dL}{dy} \cdot \frac{dy}{dw_2} = \text{base_der} \cdot x^2 \\
\frac{dL}{db} &= \frac{dL}{dy} \cdot \frac{dy}{db} = \text{base_der} \cdot 1
\end{aligned}$$

Now the update function. Recall $w_2 = w_1 - \eta \frac{dL}{dw_1}$ and $b_2 = b_1 - \eta \frac{dL}{db_1}$. That is exactly what we did, but for weights we have to loop through each element.

```
def train(self, epochs, lr):
    losses = []
    for epoch in range(epochs):
        loss_total = 0
        for i in range(len(train_x)):
            ypred = self.forward(train_x[i])
            loss_total += self.loss(train_y[i], ypred)
            self.grads(train_x[i], train_y[i], ypred)
            self.update(lr)
        loss_avg = loss_total / len(train_x)
        losses.append(loss_avg)

        if (epoch+1) % 100 == 0:
            print(f"Epoch: {epoch+1}/{epochs}, Loss: {loss_avg}")

    return losses
```

This is the training loop. Of course we don't want to stop at a single epoch especially in stochastic setting. You can ignore the `losses` in the beginning as we only use them to graph. For each epoch, we want to set the total loss to 0 as they will accumulate. Then, for all x values in our training set, we will predict, calculate the loss, backprop, an update as shown. Note that we have to set `epoch` and `lr`, which is the learning rate. Now we will take the average of the total loss and print it every 100 epoch to see how well our engine is learning.

```
def test(self):
    loss_total = 0
    for i in range(0, len(test_x), 2):
```

```
        ypred1 = self.forward(test_x[i])
        loss_total += self.loss(test_y[i], ypred1)

        if i + 1 < len(test_x):
            ypred2 = self.forward(test_x[i + 1])
            loss_total += self.loss(test_y[i + 1], ypred2)

            print(f"Test {i+1}                    Test {i+2}")
            print(f"y:     {test_y[i]}      y:     {test_y[i+1]}")
            print(f"ypred: {ypred1}     ypred: {ypred2}")
            print()
        else:
            print(f"Test {i+1}")
            print(f"y:     {test_y[i]}")
            print(f"ypred: {ypred1}")

    loss_total /= len(test_x)

def print_params(self):
    coeff = [1 / math.factorial(i+1) for i in range(8)]
    for i in range(8):
        print(f"w{i+1}: {self.ws[i]}, Expected: {coeff[i]}")
    print(f"b: {self.b}, Expected: 1")

model = SGDBLE()
```

The test loop is also straight forward. It predicts, calculates the loss, and print them as shown. Also, because we know the coefficients for the Maclaurin Series, I decided to see if the parameters match with the actual coefficients. Finally, the last line is to be able use the model.

```
def graph_loss(losses):
    plt.plot(losses)
    plt.xlabel("Epoch")
    plt.ylabel("Loss")
    plt.show()

def graph_pred():
    xs = [i / 10 for i in range(-100, 200)]
    true = [math.exp(x) for x in xs]
    pred = [model.forward(x) for x in xs]

    plt.plot(xs, true, label="e^x")
    plt.plot(xs, pred, label="Prediction")
    plt.yscale("log")
    plt.legend()
    plt.show()
```

This part is plotting with Matplotlib. Please refer to the second note if you are confused! One thing though `yscale` is necessary as exponential function will get too large to show.

```
losses = model.train(1000, 0.005)
graph_loss(losses)
model.print_params()
graph_pred()
```

This part is the actual training part. We set the epoch to 1000 and learning rate to 0.005. This learning rate might be too small for the start, but I realized that this gives slightly better results than 0.001 in this setting. Then we graph and show loss, parameters, and predictions.

```
model.test()
```

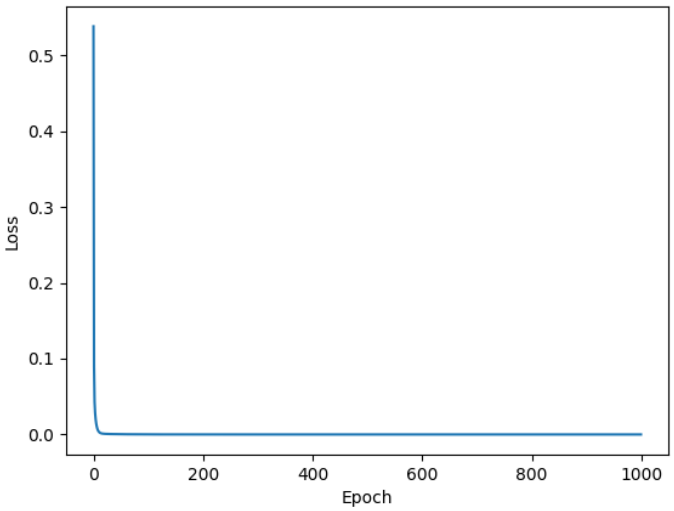
This part is for testing. Now that we have seen the entire code and explanation for it, let's now analyze our results.

Analyzing the Results

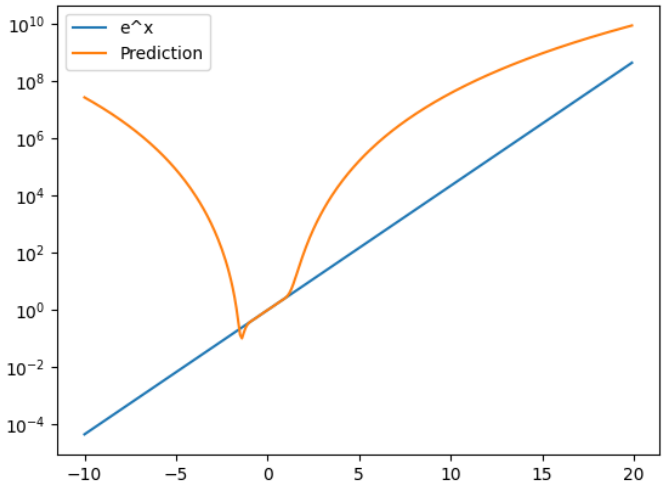
First, let's take a look at the first result.

```
losses = model.train(1000, 0.005)
graph_loss(losses)
model.print_params()
graph_pred()
```

```
Epoch: 100/1000, Loss: 0.00013234985876534958
Epoch: 200/1000, Loss: 3.427320321762344e-05
Epoch: 300/1000, Loss: 2.4492707827970274e-05
Epoch: 400/1000, Loss: 2.2669862503550263e-05
Epoch: 500/1000, Loss: 2.172205317266906e-05
Epoch: 600/1000, Loss: 2.0984248749352728e-05
Epoch: 700/1000, Loss: 2.036538265870468e-05
Epoch: 800/1000, Loss: 1.9834658817269138e-05
Epoch: 900/1000, Loss: 1.937270927247297e-05
Epoch: 1000/1000, Loss: 1.8965386261105253e-05
```




```
w1: 0.9444454598562942, Expected: 1.0
w2: 0.49986306356857685, Expected: 0.5
w3: 0.620431907367758, Expected: 0.16666666666666666
w4: 0.21618569290559692, Expected: 0.041666666666666664
w5: -0.9321256839917001, Expected: 0.008333333333333333
w6: -0.48709437200615985, Expected: 0.001388888888888889
w7: 0.5594822789317027, Expected: 0.0001984126984126984
w8: 0.33002989302732855, Expected: 2.48015873015873e-05
b: 0.9983077779493585, Expected: 1
```



This is what we got after the first training. The loss looks pretty good actually. As you can see for each 100^{th} epoch, the loss is decreasing well. From the graph, you can see the sudden drop in losses; however, we can't necessarily say that this is a grokking phenomena as it might just be model understanding the pattern from underfitting.

Looking at the weights, some are good and some are bad. We will get back to this later, but

this is also one of the reason why we cannot scale this engine. Similarly, we can see that our polynomial fits very well for the small range, but not well as the input increases.

```
model.test()
```

```
Test 1                          Test 2
y:     2.718281828459045        y:     2.7456010150169163
ypred: 2.7495260176087557       ypred: 2.7867851769675642

Test 3                          Test 4
y:     2.7731947639642978       y:     2.801065834699079
ypred: 2.8257757734623574       ypred: 2.8666322590606814

Test 5                          Test 6
y:     2.82921701435156         y:     2.857651118063164
ypred: 2.909497162739583        ypred: 2.954521430346852
...
Test 45                         Test 46
y:     4.220695816996552        y:     4.263114515168817
ypred: 9.347554553339133        ypred: 9.752958544358306

Test 47                         Test 48
y:     4.305959528345206        y:     4.349235141062741
ypred: 10.179646340417854       ypred: 10.628580569357089

Test 49                         Test 50
y:     4.392945680918757        y:     4.437095519003664
ypred: 11.100758361572902       ypred: 11.597212287422602
```

For the training set, I only listed some of them as you can refer to the previous pages. Notice that for first few tests, the predictions are not bad. However, for Test 5, the engine does not seem be effective and seems lost from Test 10. We could actually run our training loop multiple times to see what happens, and I will leave this to the readers. However, the engine does not seem to learn well.

Here is the question. Is the engine overfitting right now? If you said no, well... that is correct! Technically, we can see it as overfitting, but what I think is more accurate is to say that this is the limitation of the scalability of this engine and our method. When predicting, we only used nine terms. From the parameters, some were very close to the expectation and others were very bad. This is because we are using a polynomial with finite term to predict, whereas the real Maclaurin series uses infinite sum. Therefore, it is techinically an overfitting to certain point in a way that our engine tries to fit our polynomial in limited range as shown in the graph, but this too could improve if we have the computational ability to use infinite sum. Therefore, it may be more accurate to say that it is the limitation of the engine and scalability that causes high loss in testing.

We just completed our first model! Although it is very limited in terms of scalability and usability, I hope this example was very helpful in using backpropagation in code. For our next example, we will build an actual simple neural net to predict a nonlinear function.

12.3.2 Vanilla Neural Network

Now that we have created a simple backprop engine, let's increase the number of hidden layers to predict $x_1 x_2$ for inputs x_1 and x_2. I thought it would be great to introduce nonlinearity to our problem, and it works! Without further ado, let's get started by taking a look at the source code.

Source Code

Below is the entire source code with the results.

```
import math
import random
import matplotlib.pyplot as plt
%matplotlib inline
```

```
# Predicting x1 * x2
def tanh(x):
    return (math.exp(x) - math.exp(-x)) / (math.exp(x) + math.exp(-x)
        )

def tanh_der(x):
    return 1 - tanh(x)**2

def loss(y, ypred):
    return 0.5 * (y - ypred)**2


# Data
data_x = [[i/10, j/10] for i in range(-10, 11) for j in range(-10,
    11)]
data_y = [(x1 * x2) for x1, x2 in data_x]

data = list(zip(data_x, data_y))
random.seed(3141592)
random.shuffle(data)

split = int(0.8 * len(data))
train_data = data[:split]
test_data = data[split:]

train_x = [x for x, y in train_data]
```

```
train_y = [y for x, y in train_data]
test_x = [x for x, y in test_data]
test_y = [y for x, y in test_data]
```

```
class MLP:
    def __init__(self, layers):
        self.ws = []
        self.bs = []

        random.seed(3141592)
        for i in range(len(layers) - 1):
            layer_in = layers[i]
            layer_out = layers[i + 1]
            layer_w = [[random.uniform(-1, 1) for _ in range(layer_in
                )]
                       for _ in range(layer_out)]
            layer_b = [random.uniform(-1, 1) for _ in range(layer_out
                )]

            self.ws.append(layer_w)
            self.bs.append(layer_b)

    def forward(self, x):
        self.act = [x]
        self.zs = []

        # Layers
        for i in range(len(self.ws)):
            prev = self.act[-1]
            pre_layer = []
            act_layer = []

            # Neurons
            for j in range(len(self.ws[i])):
                pre = self.bs[i][j]
                for k in range(len(self.ws[i][j])):
                    pre += self.ws[i][j][k] * prev[k]

                pre_layer.append(pre)

                if i != len(self.ws) - 1:
                    act_layer.append(tanh(pre))
                else:
                    act_layer.append(pre)

            self.zs.append(pre_layer)
```

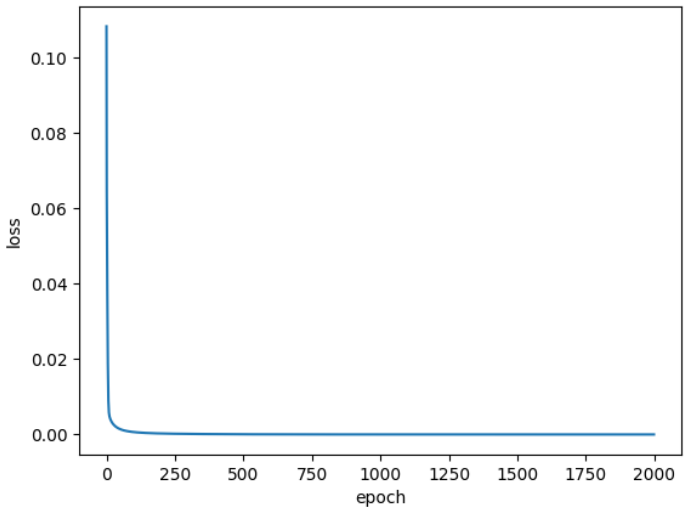
```
            self.act.append(act_layer)

        return self.act[-1][0]

    def backprop(self, y):
        self.grad_w = [[[0.0 for _ in range(len(self.ws[i][j]))]
                        for j in range(len(self.ws[i]))]
                       for i in range(len(self.ws))]
        self.grad_b = [[0.0 for _ in range(len(self.bs[i]))]
                       for i in range(len(self.bs))]
        ypred = self.act[-1][0]
        delta = [0] * len(self.ws)
        delta[-1] = [ypred - y]

        # range(start, stop, step)
        for k in range(len(self.ws) - 2, -1, -1):
            delta[k] = []
            for i in range(len(self.ws[k])):
                dlda = 0.0
                for j in range(len(self.ws[k + 1])):
                    dlda += self.ws[k + 1][j][i] * delta[k + 1][j]
                delta[k].append(dlda * tanh_der(self.zs[k][i]))

        for i in range(len(self.ws)):
            prev = self.act[i]
            for j in range(len(self.ws[i])):
                self.grad_b[i][j] = delta[i][j]
                for k in range(len(prev)):
                    self.grad_w[i][j][k] = delta[i][j] * prev[k]

    def update(self, lr):
        for i in range(len(self.ws)):
            for j in range(len(self.ws[i])):
                for k in range(len(self.ws[i][j])):
                    self.ws[i][j][k] -= lr * self.grad_w[i][j][k]
                self.bs[i][j] -= lr * self.grad_b[i][j]

    def train(self, train_x, train_y, epochs, lr):
        losses = []
        for epoch in range(epochs):
            total_loss = 0

            for x, y in zip(train_x, train_y):
                ypred = self.forward(x)
                total_loss += loss(y, ypred)
                self.backprop(y)
                self.update(lr)
```

```
88
89              average_loss = total_loss / len(train_x)
90              losses.append(average_loss)
91
92              if (epoch + 1) % 100 == 0:
93                  print(f"Epoch: {epoch+1}/{epochs}, Loss: {
                        average_loss}")
94
95          return losses
96
97
98      def show_params(self):
99          for i, (w, b) in enumerate(zip(self.ws, self.bs)):
100             layer = "OUTPUT" if i == len(self.ws) - 1 else f"LAYER {i
                    +1}"
101             print(layer)
102             for j, neuron_w in enumerate(w):
103                 weights_str = ", ".join(f"{x:.4f}" for x in neuron_w)
104                 print(f"  Neuron {j+1}:")
105                 print(f"    Weights: [{weights_str}]")
106                 print(f"    Bias = {b[j]:.4f}")
107             print()
108
109 mlp = MLP([2, 8, 8, 1])
```

```
1 losses = mlp.train(train_x, train_y, epochs=2000, lr=0.01)
2 print()
3
4 for x, y in zip(train_x[:10], train_y[:10]):
5     print(x, y)
6     print(f"     ypred: {mlp.forward(x)}")
7
8 plt.plot(losses)
9 plt.xlabel("epoch")
10 plt.ylabel("loss")
11 plt.show()
12
13 #mlp.show_params()
```

```
Epoch: 100/2000, Loss: 0.0006795237745380497
Epoch: 200/2000, Loss: 0.0002780199905385479
Epoch: 300/2000, Loss: 0.00015780133554953304
Epoch: 400/2000, Loss: 9.996884725589866e-05
Epoch: 500/2000, Loss: 6.544552166502369e-05
Epoch: 600/2000, Loss: 4.373246439583505e-05
Epoch: 700/2000, Loss: 3.032175895325618e-05
```

```
Epoch: 800/2000, Loss: 2.2295750007135315e-05
Epoch: 900/2000, Loss: 1.7559747007674843e-05
Epoch: 1000/2000, Loss: 1.4723606896641237e-05
Epoch: 1100/2000, Loss: 1.2951440585334302e-05
Epoch: 1200/2000, Loss: 1.1773900112740833e-05
Epoch: 1300/2000, Loss: 1.0936643742787753e-05
Epoch: 1400/2000, Loss: 1.0302935457567813e-05
Epoch: 1500/2000, Loss: 9.79811628836463e-06
Epoch: 1600/2000, Loss: 9.379957580934361e-06
Epoch: 1700/2000, Loss: 9.023360655331247e-06
Epoch: 1800/2000, Loss: 8.712547584218995e-06
Epoch: 1900/2000, Loss: 8.437042172730164e-06
Epoch: 2000/2000, Loss: 8.189547717119376e-06

[1.0, -1.0] -1.0
     ypred: -0.9914314084230468
[0.8, -1.0] -0.8
     ypred: -0.8026975789984329
[0.8, -0.9] -0.7200000000000001
     ypred: -0.7229659139930072
[0.1, -0.8] -0.08000000000000002
     ypred: -0.07901899113208088
[0.3, -0.3] -0.09
     ypred: -0.08931568886644314
[-0.7, -0.1] 0.06999999999999999
     ypred: 0.07003157242570901
[-0.6, -0.5] 0.3
     ypred: 0.29863205953269367
[0.7, 0.8] 0.5599999999999999
     ypred: 0.5613061391881997
[-0.5, -0.4] 0.2
     ypred: 0.19629911113434118
[0.9, 0.8] 0.7200000000000001
     ypred: 0.7218073084017129
```




```python
test_loss = 0
for x, y in zip(test_x, test_y):
    ypred = mlp.forward(x)
    test_loss += loss(y, ypred)

for x, y in zip(test_x[:10], test_y[:10]):
    ypred = mlp.forward(x)
    print(x, y)
    print(f"    ypred: {ypred}")

print("Test Loss:", test_loss / len(test_x))
```

```
[-0.9, -0.3] 0.27
    ypred: 0.27224246623233583
[0.9, 1.0] 0.9
    ypred: 0.889162612694173
[0.1, 0.5] 0.05
    ypred: 0.050824340568330406
[1.0, 1.0] 1.0
    ypred: 0.9737260427905209
[0.6, -0.8] -0.48
    ypred: -0.4811987339447982
[0.6, -0.3] -0.18
    ypred: -0.17587974783858806
[0.6, 0.4] 0.24
    ypred: 0.23577905969192914
[0.5, -0.8] -0.4
    ypred: -0.4007379739663608
[0.5, 0.2] 0.1
    ypred: 0.0970149401772119
```

```
[0.9, -0.2] -0.18000000000000002
    ypred: -0.18071247280108016
Test Loss: 1.1432408362631248e-05
```

Detailed Explanation

Now let's take a look at each block of the code to see their role in our vanilla neural net.

```
import math
import random
import matplotlib.pyplot as plt
%matplotlib inline
```

Here, I imported essential modules and libraries to do math, get a randomized value, and plot the loss.

```
# Predicting x1 * x2
def tanh(x):
    return (math.exp(x) - math.exp(-x)) / (math.exp(x) + math.exp(-x)
        )

def tanh_der(x):
    return 1 - tanh(x)**2

def loss(y, ypred):
    return 0.5 * (y - ypred)**2
```

For this problem, I decided to use tanh for our activation function as sigmoid squishes the values to 0 and 1, which is not ideal for regression problem like ours and ReLU was not effective when I tried. Since $\tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}$ and $\tanh'(x) = 1 - \tanh^2(x)$, we did the same. For our loss function, I decided to use MSE for simplicity again.

```
# Data
data_x = [[i/10, j/10] for i in range(-10, 11) for j in range(-10,
    11)]
data_y = [(x1 * x2) for x1, x2 in data_x]

data = list(zip(data_x, data_y))
random.seed(3141592)
random.shuffle(data)

split = int(0.8 * len(data))
train_data = data[:split]
test_data = data[split:]
```

```
12
13 train_x = [x for x, y in train_data]
14 train_y = [y for x, y in train_data]
15 test_x = [x for x, y in test_data]
16 test_y = [y for x, y in test_data]
```

This part is for making our datasets. I decided to divide i and j by 10 because if I don't, math overflow error will be resulted due to tanh function.

This time, I decided to shuffle data in a way that we will get the same result when you try running this, and split the data into 80% for training and 20% for validating.

```
1  class MLP:
2      def __init__(self, layers):
3          self.ws = []
4          self.bs = []
5
6          random.seed(3141592)
7          for i in range(len(layers) - 1):
8              layer_in = layers[i]
9              layer_out = layers[i + 1]
10             layer_w = [[random.uniform(-1, 1) for _ in range(layer_in
                   )]
11                        for _ in range(layer_out)]
12             layer_b = [random.uniform(-1, 1) for _ in range(layer_out
                   )]
13
14             self.ws.append(layer_w)
15             self.bs.append(layer_b)
```

Again for our example, I decided to store our core functions in a class since we want to reuse and store our attributes. When this class is first called, we will create an empty array and for `ws` and `bs` and store them in `self`. Then, for the randomized value, we will loop through each layers to store a randomized value between -1 and 1 for weights and biases. Notice that unlike the bias which only loop through the neurons in the output layer, we have to loop through the previous layer too for weights since we have different weights for each previous neuron for each output neuron. We will then append, or store, our layer to `ws` and `bs`.

```
1 def forward(self, x):
2     self.act = [x]
3     self.zs = []
4
5     # Layers
6     for i in range(len(self.ws)):
```

```
        prev = self.act[-1]
        pre_layer = []
        act_layer = []

        # Neurons
        for j in range(len(self.ws[i])):
            pre = self.bs[i][j]
            for k in range(len(self.ws[i][j])):
                pre += self.ws[i][j][k] * prev[k]

            pre_layer.append(pre)

            if i != len(self.ws) - 1:
                act_layer.append(tanh(pre))
            else:
                act_layer.append(pre)

        self.zs.append(pre_layer)
        self.act.append(act_layer)

    return self.act[-1][0]
```

This is the forward function. Here, I decided to create both `self.act` and `self.zs` since we need both of them for backprop function. As the name suggests, `self.act` is the value after tanh is applied to `self.zs`. Moreover, I stored the input in `self.act` in the beginning to start using them in the first hidden layer.

Continuing, we create arrays for each layer for preactivation and activation. Then, we will continue by looping through the neurons within each layer. Our preactivation will start from the bias, and by following the definition, we will add the products of weights and the previous activation. For the if-else condition, I decided to do this because we don't want tanh for our final output layer. Then, we will store the layers in `self.act` and `self.zs` for latter use.

```
def backprop(self, y):
    self.grad_w = [[[0.0 for _ in range(len(self.ws[i][j]))]
                    for j in range(len(self.ws[i]))]
                   for i in range(len(self.ws))]
    self.grad_b = [[0.0 for _ in range(len(self.bs[i]))]
                   for i in range(len(self.bs))]
    ypred = self.act[-1][0]
    delta = [0] * len(self.ws)
    delta[-1] = [ypred - y]

    # range(start, stop, step)
    for k in range(len(self.ws) - 2, -1, -1):
```

```
        delta[k] = []
        for i in range(len(self.ws[k])):
            dlda = 0.0
            for j in range(len(self.ws[k + 1])):
                dlda += self.ws[k + 1][j][i] * delta[k + 1][j]
            delta[k].append(dlda * tanh_der(self.zs[k][i]))

    for i in range(len(self.ws)):
        prev = self.act[i]
        for j in range(len(self.ws[i])):
            self.grad_b[i][j] = delta[i][j]
            for k in range(len(prev)):
                self.grad_w[i][j][k] = delta[i][j] * prev[k]
```

Ah the backprop. I think this is the hardest part from our code. Assuming that you have read the previous section, let's interpret the code with the corresponding math equations.

Recall although we do not calcuate the loss per individual parameter, we do have to calculate the gradients. To start, I decided to put the initial values as 0. To store them, we can do the same just as we randomized the initial weights and biases because ultimately, each gradient will have a matching parameter. Continuing, `ypred` is the prediction, and that is the first element from the last activation layer. This will vary when we have numerous outputs, but since this is a simple regression problem and we only have one output, this is good. `delta` here is the delta term for our hidden layers. Let's start by setting them as 0 at first and `ypred - y` for the last layer.

Here comes the fun part. In Python, the `range` function can be used as `range(start, stop, step)`. Here, we start from the last hidden layer and we subtract 2 due to indices. Then, we will stop when we reach -1 by going backward by 1. We will loop through them for our delta terms. Recall the formula that we derived below.

$$\delta_i^{[l]} = \frac{\partial L}{\partial a_i^{[l]}} \frac{\partial a_i^{[l]}}{\partial z_i^{[l]}} = g'\left(z_i^{[l]}\right) \sum_j w_{ji}^{[l]} \delta_j^{[l+1]}$$

As the formula suggests, we will add the weights and the corresponding following delta term with the for loop for $\frac{\partial L}{\partial a_i^{[l]}}$. Then, we will mulitply it by the derivative of the actiavtion function, which is tanh for our example. Finally, we can store our delta terms that are calculuated. Next, we can now use them to calculate the gradients.

$$\frac{\partial L}{\partial w_{ij}^{[l]}} = \delta_i^{[l]} a_j^{[l-1]}$$

$$\frac{\partial L}{\partial b_i^{[l]}} = \delta_i^{[l]} \frac{\partial z_i^{[l]}}{\partial b_i^{[l]}} = \delta_i^{[l]}$$

As the formula suggests, for each parameters, will calculate the gradients again with for the

loop. This is it for the backprop!

```
def update(self, lr):
    for i in range(len(self.ws)):
        for j in range(len(self.ws[i])):
            for k in range(len(self.ws[i][j])):
                self.ws[i][j][k] -= lr * self.grad_w[i][j][k]
            self.bs[i][j] -= lr * self.grad_b[i][j]

def train(self, train_x, train_y, epochs, lr):
    losses = []
    for epoch in range(epochs):
        total_loss = 0

        for x, y in zip(train_x, train_y):
            ypred = self.forward(x)
            total_loss += loss(y, ypred)
            self.backprop(y)
            self.update(lr)

        average_loss = total_loss / len(train_x)
        losses.append(average_loss)

        if (epoch + 1) % 100 == 0:
            print(f"Epoch: {epoch+1}/{epochs}, Loss: {average_loss}")

    return losses
```

Now we need to update. This part is similar with our previous example, but for each parameter, we will update it by subracting the product of the corresponding gradient and the learning rate. The training part is again very similar to our previous example. We will begin our loss with 0, the we will predict, accumulate the loss, backprop, update, and print every hundred epoch.

```
    def show_params(self):
        for i, (w, b) in enumerate(zip(self.ws, self.bs)):
            layer = "OUTPUT" if i == len(self.ws) - 1 else f"LAYER {i
                +1}"
            print(layer)
            for j, neuron_w in enumerate(w):
                weights_str = ", ".join(f"{x:.4f}" for x in neuron_w)
                print(f"  Neuron {j+1}:")
                print(f"    Weights: [{weights_str}]")
                print(f"    Bias = {b[j]:.4f}")
            print()

```

```
mlp = MLP([2, 8, 8, 1])
```

This part is for printing the parameters, and as you can see, it is pretty straight forward. I ended up not using it because I don't think it is that effective at this point, but why not just have it. Now we create an instance for the MLP class with two hidden layers with eight neurons each.

```
losses = mlp.train(train_x, train_y, epochs=2000, lr=0.01)
print()

for x, y in zip(train_x[:10], train_y[:10]):
    print(x, y)
    print(f"     ypred: {mlp.forward(x)}")

plt.plot(losses)
plt.xlabel("epoch")
plt.ylabel("loss")
plt.show()

#mlp.show_params()
```

We will train with the learning rate of 0.01 and 2000 epochs. We then print the first ten p9redictions and plot the loss.

```
test_loss = 0
for x, y in zip(test_x, test_y):
    ypred = mlp.forward(x)
    test_loss += loss(y, ypred)

for x, y in zip(test_x[:10], test_y[:10]):
    ypred = mlp.forward(x)
    print(x, y)
    print(f"    ypred: {ypred}")

print("Test Loss:", test_loss / len(test_x))
```

This part is for printing the loss and prediciton for testing. That was it for the explanation! Now let's take a look at the results.

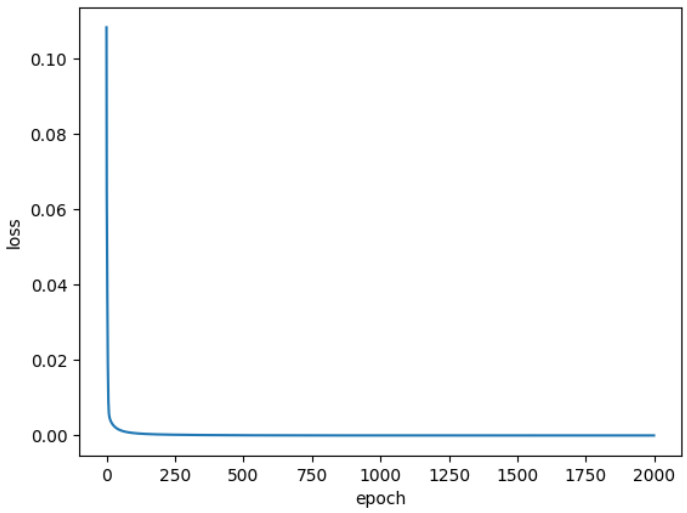
Analyzing the Results

Now that we have looked at the source code, let's analyze the result to see how our vanilla neural net performs.

```
losses = mlp.train(train_x, train_y, epochs=2000, lr=0.01)
print()

for x, y in zip(train_x[:10], train_y[:10]):
    print(x, y)
    print(f"     ypred: {mlp.forward(x)}")

plt.plot(losses)
plt.xlabel("epoch")
plt.ylabel("loss")
plt.show()

#mlp.show_params()
```

```
Epoch: 100/2000, Loss: 0.0006795237745380497
Epoch: 200/2000, Loss: 0.0002780199905385479
Epoch: 300/2000, Loss: 0.00015780133554953304
Epoch: 400/2000, Loss: 9.996884725589866e-05
Epoch: 500/2000, Loss: 6.544552166502369e-05
Epoch: 600/2000, Loss: 4.373246439583505e-05
Epoch: 700/2000, Loss: 3.032175895325618e-05
Epoch: 800/2000, Loss: 2.2295750007135315e-05
Epoch: 900/2000, Loss: 1.7559747007674843e-05
Epoch: 1000/2000, Loss: 1.4723606896641237e-05
Epoch: 1100/2000, Loss: 1.2951440585334302e-05
Epoch: 1200/2000, Loss: 1.1773900112740833e-05
Epoch: 1300/2000, Loss: 1.0936643742787753e-05
Epoch: 1400/2000, Loss: 1.0302935457567813e-05
Epoch: 1500/2000, Loss: 9.79811628836463e-06
Epoch: 1600/2000, Loss: 9.379957580934361e-06
Epoch: 1700/2000, Loss: 9.023360655331247e-06
Epoch: 1800/2000, Loss: 8.712547584218995e-06
Epoch: 1900/2000, Loss: 8.437042172730164e-06
Epoch: 2000/2000, Loss: 8.189547717119376e-06

[1.0, -1.0] -1.0
     ypred: -0.9914314084230468
[0.8, -1.0] -0.8
     ypred: -0.8026975789984329
[0.8, -0.9] -0.7200000000000001
     ypred: -0.7229659139930072
[0.1, -0.8] -0.08000000000000002
     ypred: -0.07901899113208088
[0.3, -0.3] -0.09
     ypred: -0.08931568886644314
[-0.7, -0.1] 0.06999999999999999
     ypred: 0.07003157242570901
```

```
[-0.6, -0.5] 0.3
     ypred: 0.29863205953269367
[0.7, 0.8] 0.5599999999999999
     ypred: 0.5613061391881997
[-0.5, -0.4] 0.2
     ypred: 0.19629911113434118
[0.9, 0.8] 0.7200000000000001
     ypred: 0.7218073084017129
```



I would say this is pretty good! The loss is constantly decreasing and the prediction seems to be pretty good. Looking at the graph, the model seemed to have little idea at first, but instantly learned to see the holistic view as suggested by the sudden drop.

```
test_loss = 0
for x, y in zip(test_x, test_y):
    ypred = mlp.forward(x)
    test_loss += loss(y, ypred)

for x, y in zip(test_x[:10], test_y[:10]):
    ypred = mlp.forward(x)
    print(x, y)
    print(f"    ypred: {ypred}")

print("Test Loss:", test_loss / len(test_x))
```

```
[-0.9, -0.3] 0.27
    ypred: 0.27224246623233583
[0.9, 1.0] 0.9
    ypred: 0.889162612694173
[0.1, 0.5] 0.05
    ypred: 0.050824340568330406
[1.0, 1.0] 1.0
```

```
    ypred: 0.9737260427905209
[0.6, -0.8] -0.48
    ypred: -0.4811987339447982
[0.6, -0.3] -0.18
    ypred: -0.17587974783858806
[0.6, 0.4] 0.24
    ypred: 0.23577905969192914
[0.5, -0.8] -0.4
    ypred: -0.4007379739663608
[0.5, 0.2] 0.1
    ypred: 0.0970149401772119
[0.9, -0.2] -0.18000000000000002
    ypred: -0.18071247280108016
Test Loss: 1.1432408362631248e-05
```

Our test also seems pretty good. The loss seems small because our validation set uses small numbers, but we could see that the prediction somewhat matches the real values!

Now looking back at our model, this is again good for the limited range as suggested by the result above. However, there is a reason for that. Recall that we shuffled the dataset and seletcted 80% for training and 20% for testing there. Therefore, for the limited range that the model is trained in, the model did pretty good job. However, when I first tried making dataset by splitting the training and validation as first 80% and latter 20%, our model did poorly. This is because the model learned the left side well, but it does poorly in predicting the right side since it has no idea what's going on there with absolutely no training example. However, the good news is that our neural net is scalable unlike the previous example! By increasing the range for datasets, and could see whether the model underfits or overfits.

We now covered two models from scratch! Now that we know what happens in deep neural nets, let's build our own language models!

Note 13

Building Language Models

From the third note, we discussed foundations and intuitions on language models. In our previous note, we discussed about in-depth about neural networks. Before going into different topics including AI safety, I wanted to build language models because there are a lot of concepts to discuss there. First, using our knowledge, let's build a bigram. Then, let's build a more sophisticated language model that actually sounds like a chatbot with the introduction of transformers.

§13.1 Bigram Language Model

§13.2 Attention and Transformers

Finally! Its time to discuss about transformers. If you have not already, please complete the playlist by 3Blue1Brown's highlighted in the last note and read Attention Is All You Need. More great resources can be found in the last note. *Attention Is All You Need* is considered one of the most important paper in computer science and behind its meme-like title, it makes current LLMs possible.

Although the transformer architecture is used in many places, I want to introduce it LLMs as I think it is most intuitive and classic. The problem with prior language models is that they can't remember what they were talking about seconds ago. Because the next token is predicted with limited context window, it is not possible to "remember" the entire context like us. To improve this, eight researchers from Google introduced transformers. Although the basic ideas existed before the paper, it made the transformers well known.

The content itself, when you think about it, is quite simple and very intuitive. What should we do to enhance the use of information that we have? The answer is quite simple and circular. Literally use more prior contexts with efficient methods. Think of transformer blocks, or layers, as a function that takes in and spits out vectors of the same shape. The input vector goes in

to a magical concept called multi-head attention, add and norm, and feed forward (which is basically a fancier word for MLPs that we discussed so far).

§13.3 Building an LLM with PyTorch

Note 14

AI and The Future

§14.1 AI Safety

Note 15

Further Resources

Throughout learning, I was blessed with countless resources dedicated for machine learning. Therefore, I decided to list recommended readings and other resources I used when studying and writing the notes. A huge shout out to creators for their contributions in machine learning community!

Papers

- **All Time Favorites**
  - Attention Is All You Need
  - A Mathematical Framework for Transformer Circuits
- **LLMs**
  - Chain-of-Thought Prompting Elicits Reasoning in Large Language Models
  - Chain-of-Thought Reasoning Without Prompting
  - Language Models are Few-Shot Learners
  - Scaling Laws for Neural Language Models
  - A Neural Probabilistic Language Model
  - Large Language Models: A Survey
- **Diffusion Models**
  - Hierarchical Text-Conditional Image Generation with CLIP Latents
  - Score-Based Generative Modeling through Stochastic Differential Equations
  - Denoising Diffusion Probabilistic Models
  - Learning Transferable Visual Models From Natural Language Supervision

Websites

- **Articles and Blogs**
  - Colah's Blog (Christopher Olah)
  - When AI builds itself (Anthropic)
  - Deep Double Descent (OpenAI)

  - Visualizing K-Means Clustering (Naftali Harris)
  - Jason Osajima (Jason Osajima)
  - ML Foundations: Understanding the Math Behind Backpropagation (Jordan Coblin)
- **Miscellaneous**
  - MLU-EXPLAIN (Machine Learning University)
  - FrontierMath
  - Machine Learning Mastery
  - xKiwiLabs
  - Distill
  - PyTorch Tutorial
- **GitHub**
  - ML-From-Scratch (Erik Linder-Norén)

Books and Notes

- **Books**
  - Goodfellow's Book
  - Dive into Deep Learning
  - Neural Networks and Deep Learning (Michael Nielsen)
  - Mathematics for Machine Learning
  - Deep Learning with PyTorch
  - Foundations of Computer Vision (Antonio Torralba, Phillip Isola, William Freeman)
- **Notes**
  - CS229 (Stanford University)
  - Sparse Autoencoder (Andrew Ng)

Courses and Videos

- **Courses**
  - Practical Deep Learning
  - CME 295 (Stanford University)
- **Videos**
  - Neural Networks (3Blue1Brown)
  - Neural Networks: Zero to Hero (Andrej Karpathy)

Tools and Datasets

- **Tools**
  - Weights & Biases
  - OpenRouter
- **Datasets**
- **Platforms**
  - Cirleback
  - Fireflies

Part IV

Competitive Programming

Contents

Note 16

C++ Basics

Before learning algorithms and solving problems, we need to choose a language to use. There are many programming languages that we can choose from for competitive programming such as C++, Python, and Java. However, many competitors use C++ due to its speed and popularity [Laa19]. I was debating whether to learn C++ or just stick with Python that I am already familiar with; however, I decided to learn C++ because it never hurts to learn a new language and it is also a recommended language [USA26]. With that in mind, let's get started with basic syntax from C++!

§16.1 Foundations

Let's start with the Hello World example. However to do so, we need a text editor or an integrated development environment (IDE) and a compiler. There are many different options for text editor or IDE. There are online editors that lets you write and compile there. However, I highly recommend using local editors and compilers for maintainability. Among many local editors, some popular choices are Visual Studio Code, CLion, and Vim. Please don't get mad for not including your favorite editor as it could get very philosophical. I personally use Neovim due to its low learning curve and ease in set up. Choosing an editor is up to you!

For compilers, two popular options are GNU Compiler Collection (GCC) and Clang. I personally use Clang, but this too is up to you! If you have an editor and a compiler of your choice, let's get started with printing "Hello, World!" on our screen.

Code 16.1.1: CS0201.26050-01

```cpp
#include <iostream>

int main() {
    std::cout << "Hello, World!\n";
    return 0;
}
```

I know this looks a lot, but let's first compile and see it before breaking down each line. To compile this, we can run

```
$ clang++ program.cpp -o program
```

and execute it with the following commend.

```
$ ./program
```

Then, you should see the following output in your terminal.

Output 16.1.2

```
Hello, World!
```

Now let's break down the program line by line. First, `#include <iostream>` is for including the standard input and output streams library. We need this to print our output on the screen.

Next, `int main()` defines our `main` function for execution. Next, `std::cout << "Hello, World!\n";` is to `cout`, or character out, from the standard library specified with the namespace `std`. We also have `\n` to add a line break at the end to make a pretty output. Finally, don't forget your semicolon! We need to tell our compiler that our statement has ended with the semicolon. Meaning, unlike Python, indentation is only for writing a legible code.

Lastly, `return 0;` is used to tell that the program has successfully terminated. Nowadays, it isn't strictly necessary as reaching the end of the `main` function automatically returns 0. However, it's still a good habit to keep them at the end of the `main` function.

Below is an example with many outputs.

Code 16.1.3: CS0201.26050-02

```
#include <iostream>
using namespace std;

int main() {
    cout << "I love banana. " << "Banana likes me" << endl;
    cout << "I ate five bananas today\n";

    return 0;
}
```

Compiling and executing returns the following.

Output 16.1.4

```
I love banana. Banana likes me
I ate five bananas today
```

Notice that we can add `using namespace std;` to avoid writing `std::` every time we need `cout`. We can do this for small projects and speed in competitive programming. However, this is highly discouraged in header files for large projects as it causes collisions. One more thing to

note is that both `\n` and `endl` returns a new line. However, they are **not** the same thing! We will return to this later when we discuss about optimization.

16.1.1 Variables

We know how to print a value. How about storing them? Let's first start with definitions.

Definition 16.1.5

A region of a storage that a hardware can store a value is called an **object**. **Variables** are objects with a name. The **data type** is the type of value that can be stored in the object [Pom24].

There are many data types and we need to specify the data type before declaring a variable. Below are some examples of basic data types.

Types	Stores...
int	Integers
long long	Larger 64-bit integers
float	4-bit decimals
double	8-bit decimals
char	Single character
string	Series of characters
bool	True or false value

With the data type in mind, we can declare a variable as the following.

```
dataType variableName = value;
```

Below is a quick example of declaring a variable and printing the value.

Code 16.1.6: CS0201.26050-03

```
#include <iostream>

int main() {
    std::string name = "Enoch";
    int numBananas = 12 + 34;

    std::cout << "Hi! I am " << name << ", and I ate "
              << numBananas << " bananas today.\n";

    return 0;
}
```

Notice that it is important to use quotation when declaring a string, and we can use simple operations when declaring a variable. This time, let's take a look at another example where we can take in the value of a variable and declare variables of the same type.

Code 16.1.7: CS0201.26050-04

```cpp
#include <iostream>

int main() {
    int a, b;

    std::cout << "Type two numbers to add: ";
    std::cin >> a >> b;
    std::cout << "Your sum is: " << a + b << std::endl;

    return 0;
}
```

When you compile and run it, we can get a simple calculator that can add two numbers with adequate size (32-bit).

Output 16.1.8

```
Type two numbers to add: 12345 67890
Your sum is: 80235
```

If you don't want the value of a to change, we can use `const`.

Code 16.1.9: CS0201.26050-05

```cpp
#include <iostream>

int main() {
    const int a = 5;
    int b;

    std::cout << "Type a number: ";
    std::cin >> b;
    std::cout << b << " + 5 = " << a + b << std::endl;

    return 0;
}
```

In this variation, a is fixed as 5 and cannot be changed.

Output 16.1.10

```
Type a number: 123456789
123456789 + 5 = 123456794
```

Now that we can take inputs and print outputs with various data types, let's discuss about loops and conditions.

§16.2 Conditionals and Loops

Before getting started with loops, let's briefly go over increments. For our counter i, we can **increment** or **decrement** the value, or increase or decrease, by 1 with `i++` and `i--`.

For instance if we do the following,

```
i = 5
i++
```

then it is equivalent of the following statements.

```
i = 5
i = i + 1
i = 6
```

Similarly, `i--` is identical to saying `i = i - 1`. There is also `++i` and `--i`, which first increments or decrements the value before defining, but it is just better to adjust index and use `i++` and `i--` to avoid confusion unless there is a specific reason to use them.

16.2.1 Conditionals

One more thing to note before we move on is logical operations. Unlike how we would usually write in math, we cannot use $=$ and $\neq$ for equal to and not equal to. We also won't be able to use comparisons $\geq$ and $\leq$. Instead, we would be using `==` for equal to, `!=` for not equal to, `<=` for less than or equal to, and `>=` for greater than or equal to. This is crucial when we have to compare two values in different statements. With these in mind, let's get started with if statements!

If-Else Statements

What if you want to display different statements depending on the user input? Fortunately, we could achieve that through if-else statements without manual human inspections. If, else, if-else statements have the following forms.

```
if (condition) {
    body
} else if (condition) {
    body
} else {
    body
}
```

Let's take a look at a quick example.

Code 16.2.1: CS0201.26051-01

```
#include <iostream>

int main() {
    int a;

    std::cout << "Enter a number: ";
    std::cin >> a;

    if (a > 10) {
        std::cout << "Your number is greater than ten!\n";
    } else if (a == 10) {
        std::cout << "Your number is ten!\n";
    } else {
        std::cout << "Your number is smaller than ten!\n";
    }

    return 0;
}
```

In this short example, we compare an input from the user with 10. As you can see, we must use `a == 10`, not `a = 10`, as it would be declaring a as ten. We can see that our program works as expected!

Output 16.2.2

```
$ ./program
Enter a number: -1
Your number is smaller than ten!

$ ./program
Enter a number: 10
Your number is ten!

$ ./program
Enter a number: 11
Your number is greater than ten!
```

Instead of comparing numbers, we could also use statements for comparing strings.

Code 16.2.3: CS0201.26051-02

```
#include <iostream>

int main() {
    std::string password = "password";
    std::string input;

    std::cout << "Enter your password: ";
    std::cin >> input;

    bool correct = (input == password);

    if (correct) {
        std::cout << "Correct!\n";
    } else {
        std::cout << "Incorrect!\n";
    }

    return 0;
}
```

As you can see from the example above we can take in the boolean value. An alternative way would be directly putting `input == password` in the condition for if without defining a new variable.

Output 16.2.4

```
$ ./program
Enter your password: password
Correct!

$ ./program
Enter your password: Password
Incorrect!
```

If you have fixed options, and you don't want to type every syntax for if-else statements, we can use **switch** statements.

Switch Statements

Switch statements have the following structure.

```
switch (expression) {
    case case1:
        block
    case case2:
        block
        break;
    default:
        block
}
```

Here, `break` and `default` are optional, but nice to have. Consider the following example for more intuitive explanation.

Code 16.2.5: CS0201.26051-03

```
#include <iostream>

int main() {
    char input;

    std::cout << "Choices for Breakfast:\n"
              << "  A. Two Bananas\n"
              << "  B. Three Bananas\n"
              << "  C. Four Bananas\n"
              << "What would you like for breakfast today?\n";
    std::cin >> input;

    switch (input) {
        case 'A':
            std::cout << "You chose two bananas!\n";
            break;
        case 'B':
            std::cout << "You chose three bananas!\n";
            break;
        case 'C':
            std::cout << "You chose four bananas!\n";
            break;
        default:
            std::cout << "Please choose from the options\n";
    }

    return 0;
}
```

Here, the program would go through all the options to see if there are any matching cases. If not, it will execute `default` as the fallback. If you also want to exit from the switch statement after finding the right case, we can do so with the `break` keyword.

Output 16.2.6

```
$ ./program
Choices for Breakfast:
  A. Two Bananas
  B. Three Bananas
  C. Four Bananas
What would you like for breakfast today?
B
You chose three bananas!

$ ./program
Choices for Breakfast:
  A. Two Bananas
  B. Three Bananas
  C. Four Bananas
What would you like for breakfast today?
D
Please choose from the options
```

With the basic ideas of conditions in mind, let's discuss different loops for C++.

16.2.2 Loops

There are many different fundamental loops that we can use in C++. First, let's start with for loops.

For Loops

For loops retain the following structure.

```
for (start; condition; update) {
    body
}
```

The introduction of loops really made our lives easier. One of the reason why we write program is to avoid repetitive tasks and work more efficiently. Say we want to print "I ate six bananas" five times on the screen. Without loops, we may have to use `cout` five times, which is ridiculous if the number increase to hundreds and thousands. Instead, we can use a for loop.

Code 16.2.7: CS0201.26051-04

```
#include <iostream>

int main() {
    for (int i = 0; i < 5; i++) {
        std::cout << "I ate six bananas\n";
    }

    return 0;
}
```

When we compile and run, it is telling the computer to print "I ate six bananas" if $i < 5$ starting from $i = 0$ where it increments by 1 after each pass. As expected, we will get the following result.

Output 16.2.8

```
I ate six bananas
I ate six bananas
I ate six bananas
I ate six bananas
I ate six bananas
```

A more practical use of for loops would be this.

Code 16.2.9: CS0201.26051-05

```
#include <iostream>

int main() {
    for (int i = 2; i <= 6; i++) {
        std::cout << "I ate " << i << " bananas\n";
    }

    return 0;
}
```

Instead of manually changing the numbers, the loop will display the following.

Output 16.2.10

```
I ate 2 bananas
I ate 3 bananas
I ate 4 bananas
I ate 5 bananas
I ate 6 bananas
```

With this, it is natural to consider for loops inside a for loop. This is called **nested loop** made with for loops. Consider the following example that generates all possible combinations of my breakfast assuming there exists three distinct bananas and two distinct apple where I must eat exactly one banana and exactly one apple.

Code 16.2.11: CS0201.26051-06

```
#include <iostream>

int main() {
    for (int i = 0; i < 3; i++) {
        for (int j = 0; j < 2; j++) {
            std::cout << "Banana " << i+1 << " and "
                      << "Apple " << j+1 << std::endl;
        }
    }

    return 0;
}
```

This will run the inner loop when $i = 0$, $i = 1$, and $i = 3$. We also have to start from $i = 1$ and $j = 1$ or add the displayed number by 1, or else we will get Banana 0 and Apple 0. Compiling and running, we can notice that we get $3 \cdot 2 = 6$ distinct combinations.

Output 16.2.12

```
Banana 1 and Apple 1
Banana 1 and Apple 2
Banana 2 and Apple 1
Banana 2 and Apple 2
Banana 3 and Apple 1
Banana 3 and Apple 2
```

Moving on from for loops, let's discuss while loops.

While Loops

While loops have the following form.

```
while (condition) {
    body
}
```

There are many things that we can do with while loops. Below are two simple examples.

Code 16.2.13: CS0201.26051-07

```
#include <iostream>

int main() {
    int i = 10;

    while (i > 0) {
        std::cout << i << " ";
        i--;
    }
    std::cout << std::endl;

    return 0;
}
```

The code block inside the while loop will iterate until *i* reaches 1.

Output 16.2.14

```
10 9 8 7 6 5 4 3 2 1
```

Unlike the first example, the second example uses boolean for the condition.

Code 16.2.15: CS0201.26051-08

```
#include <iostream>

int main() {
    int birthdayYear = 9999;
    int input;
    bool wrong = true;

    while (wrong) {
        std::cout << "Guess which year I was born in!\n";
        std::cin >> input;

        if (input == birthdayYear) {
            wrong = false;
            std::cout << "Correct!\n";
        } else {
            std::cout << ":(\n";
        }
    }

    return 0;
}
```

This program will ask the user the same question with feedback until the boolean value `wrong` is false, or the answer is correct. Notice that it is important to put `std::cout << ":(\n";` in `else` to prevent it showing when the answer is correct.

Output 16.2.16

```
Guess which year I was born in!
1234
:(
Guess which year I was born in!
5678
:(
Guess which year I was born in!
9999
Correct!
```

Lastly, below is an introduction to another loop.

Do/While Loops

Do/While loop is a variation of while loop that always run the first time [W3S26]. Unlike the traditional while loop that always check the condition first before running, do/while loop run the first time, then check if next run is eligible even if the first pass does not meet the condition. Below is the general structure of do/while loops.

```
do {
    block
} while (condition);
```

Consider the following example.

Code 16.2.17: CS0201.26051-09

```
#include <iostream>

int main() {
    bool correct = false;

    do {
        std::cout << "Correct!\n";
    } while (correct);

    return 0;
}
```

With the traditional while loop, no outputs are expected. However, do/while loop will give a different result.

Output 16.2.18

```
Correct!
```

Notice that even if the condition is false for the first pass, the block still runs anyhow.

This is it for the basics of conditionals and loops! Before discussing fundamental algorithms,

let's discuss more essential topics on functions and references.

§16.3 Functions and References

Up until now, we had everything in our `main` function. However, you may have noticed that dumping everything in there is not efficient nor scalable. This is where functions kick in.

16.3.1 Functions

As always, let's start with definition.

Definition 16.3.1

A reusable code block dedicated for a certain task is called a **function** [Gee26].

Functions are specially useful as we can use it multiple times just by calling them instead of copy and pasting every time we need them. Functions in C++ generally have the following format.

```
outputDataType functionName(parameters) {
    code block
    return outputValue;
}
```

Looking at a bare `main` function, we can see that we have `int` since the `main` function returns an integer value like 0. Since no arguments are taken in a bare `main` function, we have empty parenthesis. There are some functions that do not return any value. In that case, we would be using `void`. Moreover to use a function, we need to call it. Below is a quick example.

Code 16.3.2: CS0201.26052-01

```
#include <iostream>

void helloWorld() {
    std::cout << "Hello, World!\n";
}

int main() {
    helloWorld();
    helloWorld();
    helloWorld();
    return 0;
}
```

In the example above, we have our `helloWorld` function that prints the text "Hello, World!" to the console. The type is `void` since it does not return any values. By calling the function three times in `main`, we don't have to manually write long texts every time!

Output 16.3.3

```
Hello, World!
Hello, World!
Hello, World!
```

Below is a more practical use of simple functions for basic mathematical operations.

Code 16.3.4: CS0201.26052-02

```
#include <iostream>

int add(int a, int b) {
    return a + b;
}

int subtract(int a, int b) {
    return a - b;
}

int multiply(int a, int b) {
    return a * b;
}

double divide(double a, double b) {
    return a / b;
}

int mod(int a, int b) {
    return a % b;
}

int main() {
    std::cout << "1 + 2 = "     << add(1, 2)       << std::endl;
    std::cout << "3 - 4 = "     << subtract(3, 4)  << std::endl;
    std::cout << "5 * 6 = "     << multiply(5, 6)  << std::endl;
    std::cout << "7 / 8 = "     << divide(7, 8)    << std::endl;
    std::cout << "10 mod 9 = " << mod(10, 9)      << std::endl;

    std::cout << "((1 + 2) * 4) mod 10 = "
              << mod(multiply(add(1, 2), 4), 10) << std::endl;

    return 0;
}
```

Each function will take in the values a and b to perform the dedicated task. After declaring the functions, we can call them in the `main` function to display them. Many times it is better to make the functions declare a value rather than print directly to the console for reusability. As shown in the last example, we can call a function as an input of different function.

Output 16.3.5

```
1 + 2 = 3
3 - 4 = -1
5 * 6 = 30
7 / 8 = 0.875
10 mod 9 = 1
((1 + 2) * 4) mod 10 = 2
```

Another very important technique that allows us to break down hard problems into simpler problems is recursion.

Recursion

As always, let's get started with definition.

Definition 16.3.6

A technique of calling a function in the function itself is **recursion**.

The technique is rather straight forward when we look at some examples.

Exercise 16.3.7

Write a code that returns the sum of integers from 1 to n inclusive were n is any input.

Solution.

There are few ways of solving this problem. First, we can use a simple for loop. Consider the following program.

Code 16.3.8: CS0201.26052-03

```
1  #include <iostream>
2
3  int main() {
4      int n;
5      int sum = 0;
6
7      std::cout << "Enter n: ";
8      std::cin >> n;
9
10     for (int i = 1; i <= n; i++) {
11         sum += i;
12     }
13     std::cout << "Sum: " << sum << "\n";
14
15     return 0;
16 }
```

In the code above, we use a simple loop without compound assignment operator to find

the sum from 1 to any positive input n.

Output 16.3.9

```
Enter n: 3
Sum: 6
```

Below is another output.

Output 16.3.10

```
Enter n: 10
Sum: 55
```

Although this method works for simple example like this, we can also use recursion.

Code 16.3.11: CS0201.26052-04

```
#include <iostream>

int sum(int n) {
    if (n > 0) {
        return n + sum(n-1);
    } else {
        return 0;
    }
}

int main() {
    int n;

    std::cout << "Enter n: ";
    std::cin >> n;

    std::cout << "Sum: " << sum(n) << "\n";

    return 0;
}
```

In the code above, we have a function `sum` that returns `n + sum(n-1)`. We also need `return 0` as our base case. If $n = 5$, then it will return `5 + sum(4)` which continues as `5 + 4 + sum(3)` until we have $5+4+3+2+1+0$. With recursion, we broke down a hard problem of adding a range of numbers to a simple problem of adding two numbers.

Output 16.3.12

```
Enter n: 3
Sum: 6
```

Below is another output.

Output 16.3.13

```
Enter n: 10
Sum: 55
```

As shown above, both methods work.

For a simple problem like the exercise above, the could be a more simple solution than recursion. However, the technique will come very handy when the problem gets harder. Now with the ideas of functions in mind, we can continue with another important topic of references.

16.3.2 References

One of the primary goals of writing codes for competitive programming is efficiency. Indeed most, if not all, codes written for competitions are not maintained as the main purpose is to solve problems efficiently [USA26]. Moreover, it is important that your program is fast enough to execute under the time limit. One way to achieve this is by preventing making of copies, and this is where reference comes in.

References and pointers serves different purposes, although they look similar. Indeed, references are using pointers behind the scenes. However there are much more topics to discuss for pointers, and let's focus on references for now for the purpose of this note.

Definition 16.3.14

A **reference** is an alias of an existing object that allows indirect access to the object.

Let's take a look at a quick example.

Code 16.3.15: CS0201.26052-05

```cpp
#include <iostream>

int main() {
    int a = 5;
    int& ref = a; // reference to a

    std::cout << "ref: " << ref << "\n";
    std::cout << "a:   " << a   << "\n";
    ref++;
    std::cout << "After adding 1 to ref\n";
    std::cout << "ref: " << ref << "\n";
    std::cout << "a:   " << a   << "\n";

    return 0;
}
```

When declaring a reference, we use the ampersand &. Technically `int& ref` and `int &ref` does the same thing, but let's use the first one to be consistent. Here, `ref` is our reference to

`a`. Unlike declaring `int ref = a;`, which creates a copy of `a`, reference become an alias of `a` that we can use to edit `a`. Below is the output from running the code.

Output 16.3.16

```
ref: 5
a:   5
After adding 1 to ref
ref: 6
a:   6
```

As you can see above, incrementing `ref` also increments `a` as they refer to the same object. One of the most common application of reference is passing arguments in functions [Gee26]. Consider the example below.

Code 16.3.17: CS0201.26052-06

```
#include <iostream>

void increment1(int a) {
    a++;
    std::cout << a << "\n";
}

void increment2(int& a) {
    a++;
    std::cout << a << "\n";
}

int main() {
    int x = 5;
    increment1(x);
    x = 5;
    increment2(x);

    return 0;
}
```

In the example above, the difference in performance between the two functions are trivial. However, the first function creates a local copy of x while the second one doesn't. Passing arguments by reference can be especially helpful when we are dealing with large objects. Indeed, copying large objects won't be as efficient as just indirectly adjusting the original value.

Output 16.3.18

```
5
6
```

Another use case of reference in arguments is that we can directly change the value. Let's tweak our code above to further elaborate on the point above.

Code 16.3.19: CS0201.26052-07

```
#include <iostream>

void increment1(int a) {
    a++;
}

void increment2(int& a) {
    a++;
}

int main() {
    int x = 5;
    increment1(x);
    std::cout << x << "\n";

    x = 5;
    increment2(x);
    std::cout << x << "\n";

    return 0;
}
```

This time, we have `cout` in the `main` function. As mentioned previously, the first function creates a local copy and adjust the copy. Meaning, the original values does not change. However, the second function changes the original value. Therefore, the values 5 and 6 must be the outputs.

Output 16.3.20

```
5
6
```

References are frequently used for efficiency and cleaner code. Before getting into exciting data structures and algorithms, let's discuss arrays and vectors.

§16.4 Arrays and Vectors

One of the most important and essential topic in computer science in general could be arrays and vectors and they provide a way to effectively store and manipulate data. Let's get started with arrays!

16.4.1 Arrays

A simple array of n elements can be thought of as an n-tuple.

Definition 16.4.1

An **array** is a collection of elements of the same data type.

When we declare an array, we need to specify the type, name, and the number of elements [W3S26].

```
dataType arrayName[number];
```

Consider the following example.

Code 16.4.2: CS0201.26053-01

```
#include <iostream>

int main() {
    int arr[5] = {1, 2, 3, 4, 5};

    for (int i = 0; i < 5; i++) {
        std::cout << arr[i] << " ";
    }
    std::cout << "\n";

    return 0;
}
```

In the example above, we have an array `arr` with five elements. When we actually print them to the console with a for loop, we can see that we have to start from `i = 0` since the index of the first element is 0. To access an element, we can have the name of the array, bracket, and the index number.

Output 16.4.3

```
1 2 3 4 5
```

We can also edit an element by specifying it.

Code 16.4.4: CS0201.26053-02

```
#include <iostream>

int main() {
    int arr[5] = {1, 2, 3, 4, 5};
    arr[2] = -3;
    std::cout << arr[2] << "\n";

    return 0;
}
```

The line `arr[2] = -3` will override the previous assignment.

Output 16.4.5

```
-3
```

Continuing, an array need not be 1-dimensional. There exists multidimensional arrays that are made of lower dimensional arrays [Mic26]. Below is an example of a 2×3 matrix.

Code 16.4.6: CS0201.26053-03

```
#include <iostream>

int main() {
    int arr[2][3];

    for (int i = 0; i < 2; i++) {
        for (int j = 0; j < 3; j++) {
            arr[i][j] = (i + 1)*10 + (j + 1);
        }
    }

    for (int i = 0; i < 2; i++) {
        for (int j = 0; j < 3; j++) {
            std::cout << arr[i][j] << " ";
        }
        std::cout << "\n";
    }

    return 0;
}
```

In the example above, we declared a multidimensional array with two rows and three columns. Then, we used loops to assign a value in each index. Using the same way to put an input, we printed the values to the console. The index for the matrix should be familiar.

Output 16.4.7

```
11 12 13
21 22 23
```

From the examples above, you may have noticed that array is not really flexible in terms of number of elements. Sometimes we might want to add or remove elements from an array, but it won't allow that since the size is fixed. This is were vectors come to place.

16.4.2 Vectors

A simple way of thinking vectors is resizable array.

Definition 16.4.8

A **vector** is a dynamic container that contains elements of same the data type.

Vectors are provided by the Standard Template Library and we need to include directive to access them [Gee26]. Because vectors are dynamic, we only need the data type and name of the vector when declaring.

Output 16.4.9

```
std::vector<dataType> vectorName;
```

Below is a quick example of declaring a vector and two ways of printing the elements in the console.

Code 16.4.10: CS0201.26053-0

```
#include <iostream>
#include <vector>

int main() {
    std::vector<int> vec = {1, 2, 3, 4, 5};

    for (int i = 0; i < 5; i++) {
        std::cout << vec[i] << " ";
    }
    std::cout << "\n";

    for (int element : vec) {
        std::cout << element << " ";
    }
    std::cout << "\n";

    return 0;
}
```

From the example above, we declared a vector `vec` and printed the elements to the console in two different ways. Both methods are fine and the `:` symbol can be interpreted as "in."

Output 16.4.11

```
1 2 3 4 5
1 2 3 4 5
```

With the idea in mind, let's discuss few key functions that distinguishes a vector and an array.

The first function is `.push_back()` function. This function allows us to add an element as the last element. If you wish to delete the last element, we can use the `.pop_back()` function. Lastly, `.size()` will allow us to see the size of a vector. Consider the following example.

Code 16.4.12: CS0201.26053-05

```cpp
#include <iostream>
#include <vector>

int main() {
    std::vector<int> vec = {1, 2, 3, 4, 5};

    vec.push_back(6);
    for (int x : vec) {
        std::cout << x << " ";
    }
    std::cout << "\n";
    std::cout << vec.size() << "\n";

    vec.pop_back();
    for (int x : vec) {
        std::cout << x << " ";
    }
    std::cout << "\n";
    std::cout << vec.size() << "\n";

    return 0;
}
```

In this example, we declare a vector, add 6 to the end, list the elements, and find the size. Then, we will revert it to the original vector by removing the last element that was added.

Output 16.4.13

```
1 2 3 4 5 6
6
1 2 3 4 5
5
```

This is it for the C++ basics! There are so much more topics to discuss in C++, but for now, let's solve some practice problems before discussing more fundamental concepts in DSA.

§16.5 Practice Problems and Solutions

Below is the list of links to practice problems for this note and personal solutions.

1. **Source:** 2018 USACO Third Bronze
   Problem: Problem 1. Teleportation
   Solution: Enoch's Solution (Git)

2. **Source:** 2016 USACO Second Bronze
   Problem: Problem 1. Promotion Counting
   Solution: Enoch's Solution (Git)

3. **Source:** 2020 USACO First Bronze
   Problem: Problem 1. Do You Know Your ABCs?
   Solution: Enoch's Solution (Git)

4. **Source:** 2020 USACO Second Bronze
   Problem: Problem 1. Word Processor
   Solution: Enoch's Solution (Git)

5. **Source:** 2019 USACO US Open Bronze
   Problem: Problem 1. Bucket Brigade
   Solution: Enoch's Solution (Git)

6. **Source:** Codeforces
   Problem: Problem 4A. Watermelon
   Solution: Enoch's Solution (Git)

7. **Source:** Codeforces
   Problem: Problem 231A. Team
   Solution: Enoch's Solution (Git)

8. **Source:** Codeforces
   Problem: Problem 546A. Soldier and Bananas
   Solution: Enoch's Solution (Git)

Note 17

Foundations

Before getting into different techniques and methods, let's discuss fundamental topics including time complexity and basic DSA.

§17.1 Time Complexity

One of the primary goal of competitive programming and an effective program is efficiency. Indeed most, if not all, programming competitions have time limits, which varies for each language. Some problems can be solved with brute force approach while others need an elegant solution to fit in the given time frame.

Definition 17.1.1

The **Time Complexity** of a program denotes the number of operations an algorithm executes [Gee26].

Please note that the definition above is very basic, and not formal. A more formal definition involves theoretical computer science, and maybe I can include more on the formal definition and P = NP in other notes.

The approximate number of the operations for the worst-case is represented using the **big O notation**. The lower the complexity, the efficient the program is.

The six major complexities are Constant $\mathcal{O}(1)$, Linear $\mathcal{O}(n)$, Logarithmic $\mathcal{O}(n \log n)$, Quadratic $\mathcal{O}(n^2)$, Exponential $\mathcal{O}(2^n)$, and Factorial $\mathcal{O}(n!)$ [Ola22]. We will evaluate the time complexities of algorithms as we walk through them, but here are some simple examples using input/output and loops.

A simple hello world example have constant time complexity since it only prints once. One the other hand, the following snippet

```
for (int i = 0; i < 5*n; i++) {
    // constant time code
}
```

is $\mathcal{O}(n)$ [Laa19]. When we start nesting them, it gets horrible. For instance, the following snippet is $\mathcal{O}(nm)$.

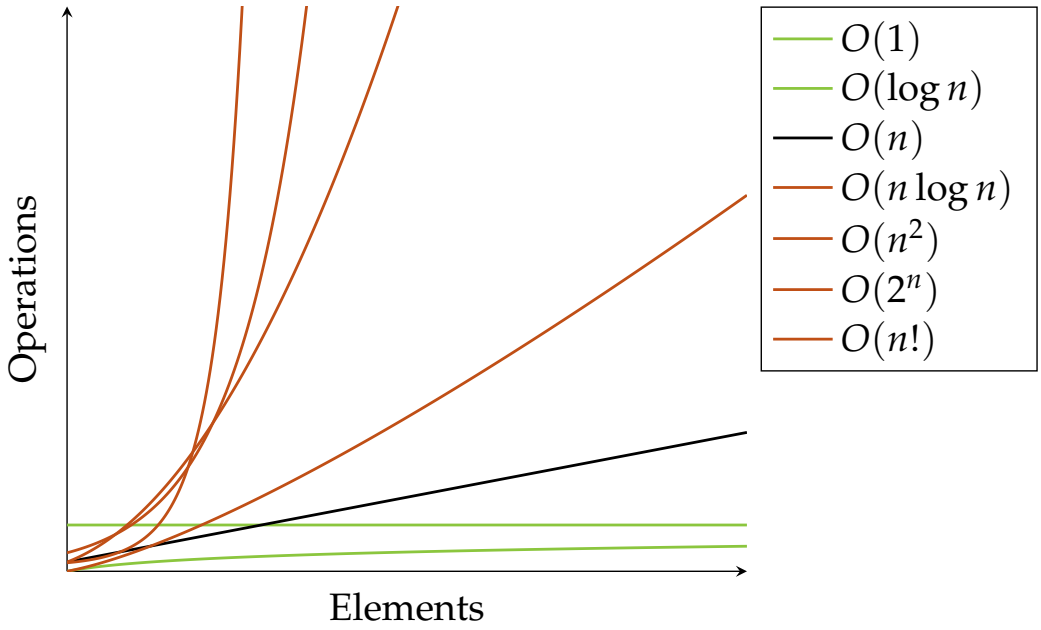
```
for (int i = 0; i < n; i++) {
    for (int j = 0; i < m; j++) {
        // constant time code
    }
}
```

Naturally, we can see that we get quadratic time if we nest code as the following.

```
for (int i = 0; i < n; i++) {
    for (int j = 0; i < n; j++) {
        // constant time code
    }
}
for (int i = 0; i < 10000*n; i++) {
    // constant time code
}
```

The snippet above is $\mathcal{O}(nn) = \mathcal{O}(n^2)$. Note that the time complexity is not linear since the big O is only interested in the worst case.

One more thing to note is the order of growth. Consider the graph below [Ola22].



From the graph above, we can see that some functions grow faster than the other. For instance,

the constant time has the lowest growth rate since it is not growing at all. Next, we can see that $\log n$ grows slower than n, which grows slower than $n \log n$ [Gee26]. Generally speaking, anything below (n) is good while we really want to avoid anything beyond the linear time.

Now big O is important since it is used to compare two algorithms that perform the same task. With **asymptotic analysis**, we can evaluate and compare algorithms without the inconsistency of inputs and machines [Gee26].

This was a short introduction to computing time complexities and big O notation. In the next section, we will be discussing fundamental data structures.

References

[Laa19] Antti Laaksonen. *Competitive Programmer's Handbook*. 2019. URL: https://github.com/pllk/cphb.

[Ola22] Joel Olawanle. *Big O Cheat Sheet – Time Complexity Chart*. 2022. URL: https://www.freecodecamp.org/news/big-o-cheat-sheet-time-complexity-chart/.

[Pom24] Alex Pomeranz. *LearnCpp*. 2024. URL: https://www.learncpp.com/.

[Gee26] GeeksforGeeks. *C++ Programming Language*. 2026. URL: https://www.geeksforgeeks.org/cpp/c-plus-plus/.

[Mic26] Microsoft Learn. *C++ documentation*. 2026. URL: https://learn.microsoft.com/en-us/cpp/cpp/.

[USA26] USACO Guide. *USACO Guide*. 2026. URL: https://usaco.guide/.

[W3S26] W3Schools. *C++ Tutorial*. 2026. URL: https://www.w3schools.com/cpp/.