Mathematics For Me

Tommy Nguyen

November 2024

Chapter 1

Linear Algebra

1.0.1 The Beginning, Systems of Linear Equations

When we first started learning about variables in middle school, one of the earliest forms of an equation we were introduced to may have looked something like this:

$$mx + b = c$$

Your first instinct is probably, "Oh look! It's the slope-intercept formula!" and you'd be right! The slope-intercept formula is an equation representing a line in 2D space (because you can move in the x and y direction).

As we move along and learn about more complex equations, we most likely encountered equations of higher dimensions (i.e., containing more variables). For example:

$$ax + +cz = d \text{ Plane in 3D space.} \tag{1.1}$$

$$ax_1 + by + cz + dx_2 = e \text{ Something in 4D space.} \tag{1.2}$$

$$a_1x_1 + a_2x_2 + ... + a_nx_n = d \ n\text{D space.} \tag{1.3}$$

As you might have guessed from the equations above, we can easily scale our **linear equations** to higher dimensions just by adding another **constant** multiplied by a new **variable**. You might have been thrown off by the number of terms introduced, so let's give them a proper definition!

- **Constant**: Any term that does not change. For example, in the equation $2x + 3$, 2 and 3 are constants because we know their value. We also know that 2 and 3 are definite values, meaning they do not change at all, regardless of what x is. In the equations provided above, a, b, c, a_1, a_2, etc are constants.
- **Variable**: Any value that will change. Typically, these are represented by an alphabetic letter (x, y, z, etc). We are usually trying to solve the

equation to find the value for these variables. For example, we might be provided a question that asks us to solve for x in: 10 = 2x + 5. In this case, x is the variable we are solving for.

- **Linear Equation**: A simple equation that *does not involve any exponents or square roots of a variable.* Basically, if you see equations like: $x^2+x=0$ or $x*y+x=2$, then they are NOT linear equations. Equations like: $x+2=10$ or $x+y+z=10$ would be considered linear equations (because they do not involve any powers or square roots, which would alter the **linearity** of the equation, basically it will no longer be a straight line).

In any linear equation, the *constants cannot all be 0's.* "Why not?" well, because if they were, then the equation would just be 0 = some number (remember, 0 times anything is going to be 0)! That's not going to be a straight line, or even a point! However, the d and e (shown in equation 1.3) CAN be 0. Any linear equation where the other side of the = is 0 is called a **homogeneous linear equation**.

Generally, a **linear equation in n variables** (where n is the number of variables for the linear equation) can be expressed via the equation shown in 1.3 (the nD one)!

There are more of them now!

Just like how we can add more variables, we can also add more equations.

A finite set of linear equations is called a **systems of linear equations** or **linear system** (we will be calling them linear systems from now on for ease of typing). In a linear system, the variables may also be called **unknowns**.

The equation below is an example of a simple linear system in 2 variables (x_1 and y_1):

$$\begin{aligned} 4x_1 + 10y_1 &= 6 \\ 6x_1 - y_1 &= 10 \end{aligned} \tag{1.4}$$

Again, like the number of variables...we can scale the number of equations as much as we want!

$$\begin{aligned} a_{11}x_1 + a_{12}x_2 + a_3x_3 + ... + a_{1n}x_n &= b_1 \\ &\vdots \\ a_{m1}x_1 + a_{m2}x_2 + ... + a_{mn}x_n &= b_m \end{aligned} \tag{1.5}$$

For a general linear system, we can say that a **solution** in n unknowns (x_1, x_2, ..., x_n) is a sequence of n numbers that, when substituted into unknowns x_1, x_2, ..., x_n, will make all linear equations in the linear system to be true.

Let's walk through an example. Given the linear system below, solve for x and y:

$$\begin{aligned} 2x - y &= 7 \\ 3x - 2y &= 10 \end{aligned} \tag{1.6}$$

Let's use the good old elimination method (if you need review, here is a great link: Khan Academy is great!). What variables can we easily eliminate? $-2y$ seems like a good candidate! Let's multiple $2x - y = 7$ by -2 and add the results to $3x - 2y = 10$.

$$\begin{aligned} 2x - y &= 7 \\ 3x - 4x - 2y + 2y &= 10 - 14 \\ \texttt{=>} \quad -x &= -4 \\ \texttt{=>} \quad x &= 4 \end{aligned} \tag{1.7}$$

Let's plug $x = 4$ back into our first equation to find y!

$$\begin{aligned} 2 * (4) - y &= 7 \\ \texttt{=>} \quad 8 - y &= 7 \\ \texttt{=>} \quad y &= 1 \end{aligned} \tag{1.8}$$

Now we know equation 1.6 has a solution: $x = 4$ and $y = 1$. We can also write the solution as coordinates too: (4, 1). More often times than not, we'll be writing the solution as coordinates because it is a lot easier to write out and visualize. Furthermore, we can think of the solutions to our linear systems as coordinates where all of the equations in the linear system intersect (cross). For example:

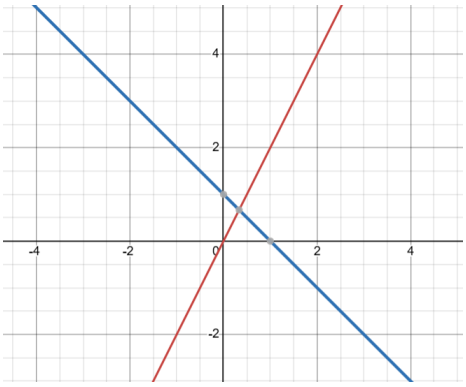

Figure 1.1: The two lines shown intersect only at point (0.33333, 0.66667).

Figure 1.1 shows two lines intersecting at one point. That is, it would only have **1 solution**. Okay, that is easy to understand...but what about other

scenarios? What if there are so many solutions, we can't really pick one and be done? In that scenario, you have **infinitely many solutions**! You can think of it like *two lines that are on top of each other* or *two planes stacked on top of one another*.

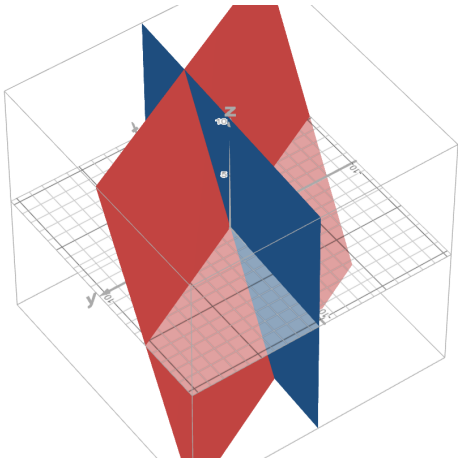

Figure 1.2: The red plane and the blue plane share an entire line where they both intersect each other. If you think back to calculus (specifically limits), you'll remember that there are infinitely many steps you can take along a line. In turn, the two planes share an infinite number of points of intersection. Likewise, if you stacked the plane on top of each other, they will also have infinitely many points of intersection, because they are basically crossing at every point!

Figure 1.2 showcases an example of infinitely many solutions. The two planes cross along an entire line, this indicates that the two planes share **MANY** points of intersections. You can think of it like this: If we stretched the planes out into infinity, the points of intersection will ALSO stretch to infinity.

In situations where we are expected to give an answer, but there are just too many answers to give, mathematicians created a neat way to provide a solution without explicitly mentioning a point! We call this way of writing our solution, **parametric equations**. Long story short, we rewrite one of the linear equations in our linear system. For example, the equation below has infinitely many solutions:

$$\begin{aligned} 4x + 2y &= 1 \\ 16x + 8y &= 4 \end{aligned} \tag{1.9}$$

If you tried solving for the linear system above, you would ultimately get $0 = 0$. This means that any solution that solves for $4x + 2y = 1$, will also solve for $16x + 8y = 4$ and vise versa! If you looked closely, you'll also notice that the *two equations are just scalar multiples of each other*!

Okay, now that we know for sure the linear system has infinitely many solutions, let's write a parametric equation! The steps are easy, really:

1. Rewrite the equation so that one variable is isolated to one side.
2. Substitute all other variables (except for the isolated variables) with a different letter.

For the first linear equation in 1.9, one of the parametric equation can be:

$$\begin{aligned} 4x + 2y &= 1 \\ \texttt{=>}\ 4x &= 1 - 2y \\ \texttt{=>}\ x &= \frac{1}{4} - \frac{y}{4} \\ \texttt{=>}\ x &= \frac{1}{4} - \frac{t}{4}, y = t \end{aligned} \tag{1.10}$$

If you had more unknowns, it's basically the same thing:

$$\begin{aligned} 2x - 3y + z - 9d &= 10 \\ \texttt{=>}\ 2x &= 10 + 3y - z + 9d \\ \texttt{=>}\ x &= \frac{10}{2} + \frac{3y}{2} - \frac{z}{2} + \frac{9d}{2} \\ \texttt{=>}\ x &= \frac{10}{2} + \frac{3t}{2} - \frac{r}{2} + \frac{9s}{2}, y = t, z = r, d = s \end{aligned} \tag{1.11}$$

Basically what we're doing in the last step is assigning a **parameter** to every unknown variable (except for the isolated variable).

There are some scenarios where you might also get **no solutions** too!

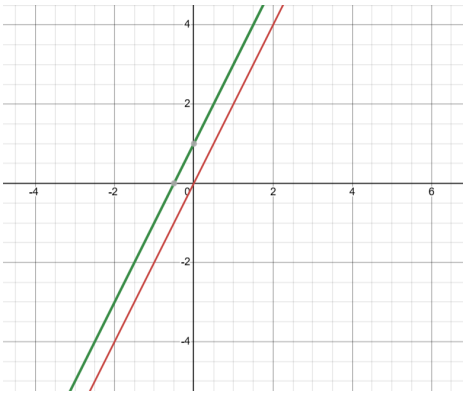

Figure 1.3: The lines are parallel, they will never cross.

Based on our investigation of one solution and infinitely many solutions, can you guess why Figure 1.3 has no solutions? If you said it was because they never cross, then you would be correct!

The two lines in Figure 1.3 never cross, they never intersect. This means that there are no solutions for x and y that satisfies both linear equations.

An example of a linear system with no solutions would be:

$$\begin{aligned} x + y &= 4 \\ 2x + 2y &= 12 \end{aligned} \tag{1.12}$$

If we tried solving for that linear system, we would end up with $0 = 4$, which will never be true.

Okay...at this point you may be thinking, "Tommy, WHY are you saying all of this now? This seems like its coming out of left field!". Well, my curious and mathematically driven reader, the reason why I bring this up now is because of an interesting rule/property that comes with linear systems: **Every linear systems has zero, one or infinitely many solutions. There are no in betweens!** As such, a linear system can be **consistent** (if it has at least 1 solution) or **inconsistent** (if it has no solutions).

The Matrix: Not the 1999 Film

As we add more equations and more unknowns, it becomes increasingly more difficult to perform computations on them, both in terms of manual computation and on the computer.

For one, keeping track of all of the "+", "-", unknowns, etc will become increasingly more difficult the more equations and unknowns we have. To combat the visual complexity of a large linear system, we introduce **augmented matrices** (**augmented matrix** for 1 matrix).

Long story short, we can convert our long and tedious-to-write-out linear system into a matrix of numbers. So our long equation below:

$$\begin{aligned} 2x_1 + 3x_2 + 4x_3 + ... + 10x_n &= 20 \\ &. \\ &. \\ &. \\ 20x_1, +30x_2 + 12x_3 + ... + x_n &= 198 \end{aligned} \tag{1.13}$$

Can turn into the matrix:

$$\begin{bmatrix} 2 & 3 & 4 & ... & 10 & 20 \\ . & & & & & \\ . & & & & & \\ . & & & & & \\ 20 & 30 & 12 & ... & 1 & 198 \end{bmatrix}$$

As you may have noticed, the last column on the right represents the numbers on the right-hand side of the equation 1.13. Broadly, all numbers on the right-most column will be the c in $mx + b = c$, and all other columns will have the coefficients of the unknowns.

Here are a few more examples so you can see what an augmented matrix may look like.

$$\begin{aligned} 2x - 3y &= 6 \\ x + 10y &= 7 \end{aligned} \tag{1.14}$$

$$\begin{bmatrix} 2 & -3 & 6 \\ 1 & 10 & 7 \end{bmatrix}$$

$$\begin{aligned} 3x - 1y - 20z &= 6 \\ 10x + 5y + 8z &= 7 \\ 10x + y + .5z &= 7 \end{aligned} \tag{1.15}$$

$$\begin{bmatrix} 3 & -1 & -20 & 6 \\ 10 & 5 & 8 & 7 \\ 10 & 1 & .5 & 7 \end{bmatrix}$$

Now that we know what an augmented matrix looks like, we can start describing how to perform operations on said matrix.

It's Elementary, My Dear Augmented Matrix!

When we perform operations on an augmented matrix to solve for unknowns, we are perform what are called **elementary row operations**. When it comes to row operations, there are 3 operations we can perform:

1. Multiply a row by a non-zero number.
2. Swap rows.
3. Add one row multiplied by a constant to another row.

As you can see, these operations are performed on rows, much like the operations we performed in previous examples were performed on one equation.

Let's explain the reason behind each operation:

- **Multiply a row by a non-zero number**: We perform this operation to get nice numbers. In other words, whenever we can multiply an entire row by a constant to get nice whole numbers, we probably should.
- **Swap rows**: We perform this operation just so we can get a matrix where the solutions are on the left diagonal (we will see an example soon). Long story short, we do this so we can easily see the answers as well. Another possible reason might be that we want to perform operations on the matrix where the left diagonal has non-zero values.
- **Add one row multiplied by a constant to another row**: Same reason for the steps we took in previous examples; we want to isolate a single unknown in one equation so we can solve for that single unknown.

Lets go through a brief example. Say we had the following augmented matrix:

$$\begin{bmatrix} 2 & 10 & 4 & 6 \\ 3 & 2 & 4 & 1 \\ 1 & 2 & 1 & 3 \end{bmatrix}$$

Here is a step-by-step walkthrough:

$$\text{Swap row 1 with row 2.} \begin{bmatrix} 1 & 2 & 1 & 3 \\ 3 & 2 & 4 & 1 \\ 2 & 10 & 4 & 6 \end{bmatrix}$$

$$\text{Swap row 2 with row 3.} \begin{bmatrix} 1 & 2 & 1 & 3 \\ 2 & 10 & 4 & 6 \\ 3 & 2 & 4 & 1 \end{bmatrix}$$

$$\text{Multiply row 1 by -2 and add to row 2.} \begin{bmatrix} 1 & 2 & 1 & 3 \\ 0 & 6 & 2 & 0 \\ 3 & 2 & 4 & 1 \end{bmatrix}$$

Multiply row 1 by -3 and add to row 3. $\begin{bmatrix} 1 & 2 & 1 & 3 \\ 0 & 6 & 2 & 0 \\ 0 & -4 & 1 & -8 \end{bmatrix}$

Multiply row 2 by 2/3 and add to row 3. $\begin{bmatrix} 1 & 2 & 1 & 3 \\ 0 & 6 & 2 & 0 \\ 0 & 0 & 7/3 & -8 \end{bmatrix}$

Multiply row 2 by -1/3 and add to row 1. $\begin{bmatrix} 1 & 0 & 1/3 & 3 \\ 0 & 6 & 2 & 0 \\ 0 & 0 & 7/3 & -8 \end{bmatrix}$

Multiply row 3 by -6/7 and add to row 2. $\begin{bmatrix} 1 & 0 & 1/3 & 3 \\ 0 & 6 & 0 & 48/7 \\ 0 & 0 & 7/3 & -8 \end{bmatrix}$

Multiply row 3 by -1/7 and add to row 1. $\begin{bmatrix} 1 & 0 & 0 & 29/7 \\ 0 & 6 & 0 & 48/7 \\ 0 & 0 & 7/3 & -8 \end{bmatrix}$

Thhe final augmented matrix shows a diagonal containing the numbers 1, 6, and 5/3. As you can probably infer, those are the coefficients for the unknowns in our linear system. We can go ahead and solve for the unknowns by dividing the values on the right-most columns by the coefficients of our unknowns (i.e., 48/7 will be divided by 6 and -8 will be divided by 7/3. First row does not need to be divided because the coefficient of the unknown is 1).

While writing out the equation, messed up a few times because I had incorrectly added/multiplied fractions. As you can probably tell, this would be where we would multiply a row by a constant, so that we can get nice round numbers to work with!

Like with linear systems, we can see if there are infinite or no solutions if the unknowns in the augmented matrix are 0 for an entire row.

No solution. 0 will never equal to 1. $\begin{bmatrix} 1 & 2 & 3 & 10 \\ 2 & 4 & 1 & -9 \\ 0 & 0 & 0 & 1 \end{bmatrix}$

Infinite solutions. Any value can be multiplied by 0 to equal 0. $\begin{bmatrix} 1 & 2 & 3 & 10 \\ 2 & 4 & 1 & -9 \\ 0 & 0 & 0 & 0 \end{bmatrix}$

Summary

Let's summarize everything we have gone through so far!

A linear system is a finite (limited) set of linear equations. Each linear equation has the potential to be a homogeneous linear equation (sum of unknowns = 0) or not (sum of unknowns = non-zero value). When we try to solve for a

linear system, we are trying to find the values for all of the unknowns, which we call solutions. For a linear system, there are three potential outcomes: the linear system has one solution, zero solutions, or infinitely many solutions. If the linear system has at least one solution, then it is consistent. If it has no solutions, then it is inconsistent. If a linear system has infinitely many solutions, we can't provide a specific point as a solution. In this scenario, we would rewrite our equation into a parametric equation, with our unknown replaced with a parameter (basically replacing it with a different letter(s)).

Outside of mathematics, the way linear systems are typically organized is in the form of an augmented matrix. When it comes to augmented matrix, we use elementary row operations to solve for the linear system. There are three operations we can perform: swap rows, multiply a row by a constant, multiply a row by a constant and add it to another row.

Practical Applications

Linear systems are a fundamental part of AI/ML. Essentially, all AI/ML models follow a similar formula:

$$W * x + b \tag{1.16}$$

Where W are the weights, b the biases, and x the inputs. Each of these values are vectors (which are just 1D matrices). ML models are trained with the expectation that the weights and biases found brings the model as **close** to the expected output as possible. In other words, we can think of training ML models as solving for the linear system $W * x + b$.

Extending it beyond simply training ML models, solving for linear systems are also used in a plethora of other ML use cases such as dimensionality reduction (PCA and SVD, where SVD is a Linear Algebra concept in-and-of itself)! Its also used in recommender systems as well, specifically matrix factorization (which usually involves linear systems). Some of these concepts (SVD and matrix factorization) we will explore later on! Just know that linear systems and solving for them are highly relevant in AI/ML!

1.0.2 Gaussian Elimination

Now that we have shown you HOW you can solve via matrices, let's provide some terminologies to get you accustomed.

There are usually two forms for a solved augmented matrix: **row echelon form** and **reduced row echelon form**.

Reduced row echelon form would be the augmented matrix we are pretty accustomed to at this point, with values only on the left-diagonal (not including the right-most columns).

$$\begin{bmatrix} 1 & 0 & 0 & 22 \\ 0 & 1 & 0 & 47 \\ 0 & 0 & 1 & 1 \end{bmatrix}$$

One thing you might have noticed from the augmented matrix above, is that the left-diagonal only has 1s in them. These 1s are called **leading 1**s. They're called "leading 1"s because they are the first 1 in their row (from left to right). Leading 1s are a major property of reduced row echelon matrices. There are other properities that are used to identify reduced row echelon matrices:

1. All rows containing only zeroes are grouped together at the bottom of the matrix.
2. Every subsequent row has a leading 1 further to the right than the previous row (i.e., if the previous row has a leading 1 in column two, then the current row will have a leading 1 in column 3, if it has a leading 1).
3. Each row has only one leading 1. All other values in the column **MUST** be 0s.

Row echelon forms are a lot more lenient. Unlike reduced row echelon form, row echelon form can have multiple leading 1s in each column. I.e., the following augmented matrix is a valid row echelon form (but **NOT** a valid reduced row echelon form):

$$\begin{bmatrix} 1 & 0 & 0 & 22 \\ 1 & 1 & 0 & 47 \\ 0 & 2 & 1 & 1 \end{bmatrix}$$

Leading 1s tend to generally be called **leading variables**. All other variables in the same row that are *not on the right most column and are not the leading variables*, are called **free variables**.

Now lets look at what a row echelon form matrix that has infinite solutions look like:

$$\begin{bmatrix} 1 & 20 & 0 & 22 \\ 1 & 1 & 23 & 47 \\ 0 & 0 & 0 & 0 \end{bmatrix}$$

As you can see, the equation has a row of just 0s! Now we can write each equation into a set of parametric equations!

$$\begin{aligned} x &= 22 - 20y \\ x &= 47 - y - 23z \end{aligned}$$

We can then replace y and z with any letters to represent them as parameters.

$$\begin{aligned} x &= 22 - 20t \\ x &= 47 - t - 23s \\ y &= t \\ z &= s \end{aligned}$$

Now we have a set of parametric equations! We can generally call a set of parametric equations as a **general solution**.

Okay...now that we have gotten some formal definitions out of the way, we can go over 2 types of **elimination procedures**. There are two ways in which we can solve a linear system using an augmented matrix, via **Gauss-Jordan Elimination** or **Gaussian Elimination**.

- **Gauss-Jordan Elimination**: Solves an augmented matrix by bringing it to reduced row echelon form. This type of elimination method uses both a *forward phase* and a *backward phase*.
- **Gaussian Elimination**: Solves an augmented matrix by bringing it to row echelon form. Typically uses a forward phase followed by *back-substitution*.

In that small bulletin, we introduced 3 new terminologies: forward phase, backward phase, and back-substitution. They may sound complex or new, but they are things you are very familiar with at this point!

- **forward phase**: You can think of this as solving the augmented matrix from top to bottom. The goal of this phase is to remove values below the leading variables.
- **backward phase**: This phase focuses on cleaning up our augmented matrix. In this phase, we move from bottom to top; we are essentially ensuring that, by the end of this phase, only the *left diagonal* contains nonzero numbers (excluding the right-most column, which will contain values).
- **back-substitution**: We learned about back-substitution when we took Algebra decades ago! Let's say you had the following equations: $x+y+z = 10$, $x + 20y + z = 3$, and $z = 10$. Since we know what z is, we can plug it into either of the other 2 equations to solve for y. We can then plug in what we found for y to solve for x! Basically, we are literally "*substituting*" backwards!

But...Why?

At this point, you might be wondering, "Tommy, why do we have **TWO** different elimination procedures? Why don't we just stick to one?". To answer that, we must understand the pros and cons of the Gauss-Jordan and Gaussian Elimination.

There is an obvious reason why we might choose to use Gauss-Jordan over Gaussian Elimination: *Super easy to see the answer*! Say you had the following augmented matrix:

$$\begin{bmatrix} 1 & 0 & 0 & 7 \\ 0 & 8 & 0 & 0 \\ 0 & 0 & 5 & 10 \end{bmatrix}$$

If 1, 8, and 5 represented the coefficients of variables x, y, and z respectively, we can easily see what the values of x, y, and z are simply by dividing the values in the right column by the values in the same row (7 / 1, 0 / 8, and 10 / 5)! Plus, performing forward phase + backward phase is easier than performing forward phase + back-substitution!

...but that only applies for small matrices. Once we get to larger ones– especially ones that are large enough to require computational power to compute– this process becomes arduous. Not only that, but if we look at the way computations are performed in the Gauss-Jordan elimination procedure, we will ultimately realize how much more computational power and operations are performed in that elimination method than in Gaussian elimination (roughly 50% more)!

To further emphasize, we are iterating through the augmented matrix **TWICE**, while performing multiplications, additions, subtractions on **ALL** rows and variables. This results in essentially 50% more operations since we are performing row operations while moving forward and backwards.

In the realm of AI, this could mean significantly slower inference or training time due to the forward phase + backward phase pipeline in the Gauss-Jordan elimination procedure. Even worse than slow inference time or training time, this 50% more operation very likely means more chances for errors.

Errors...Errors Everywhere!

In computers, every operation performed has the potential for rounding errors because computers typically approximate values. These rounding errors could be dangerous, as slight rounding can lead to severely incorrect outputs (in Python, this might mean rounding a very small value to 0 or a very large value to NaN or inf). So as you may have guessed, the more operations that are performed, the more likely it is that we will run into these types of rounding issues. Furthermore, the more operations that are performed, the more the number will round until it eventually reaches an unusable value (NaN, inf, -inf, etc).

These types of rounding issues have been a major debate when converting mathematical concepts/theories into practice, as it brings into question the vast quantity of operations typically performed in highly complex algorithms. This ultimately lead to disparity between mathematical theory and practical implementations. In this sense, algorithms that have a chance of facing these types of rounding issues (or other issues) are labeled as **unstable algorithms**. These types of algorithms are considered unstable because they produce **roundoff** errors (where values are rounded).

All in all, Gaussian Elimination seems to be the better choice in solving for larger augmented matrices as it reduces computations leading to: faster inference time and lower roundoff error rates.

TODOS:

- ~~reduced row echelon form~~

- ~~leading 1~~
- ~~rows with all 0's are grouped together at the bottom of the matrix~~
- ~~any row that contains a 1, all subsequent rows, if they have a 1, must occur further to the right~~
- ~~Each column has one leading 1 and 0s in all other locations in the column~~
- ~~row echelon form does not need to have only one leading 1 in each column~~
- ~~leading variables~~
- ~~free variables~~
- ~~general solution~~
- ~~elimination procedure~~
- ~~Gauss-Jordan Elimination~~
- ~~forward phase~~
- ~~backward phase~~
- ~~Gaussian elimination~~
- homogeneous linear system
- trivial solution
- homogeneous linear systems either only has the trivial solution or has infinitely many solutions including the trivial solution
- nontrivial solutions
- when constructing parametric equations for a homogeneous linear system in reduced row echelon form, we ignore any rows of only 0s, because they don't tell us anything (does not impose any restrictions)
- Free Variable Theorem for Homogeneous Systems: If the augmented matrix has n unknowns, its reduced row echelon form as r nonzero rows, then the system has $n - r$ free variables.
- **NOTE:** Only applies to homogeneous systems. Nonhomogeneous systems can still have no solutions (i.e., be inconsistent) if it has more unknowns than leading 1s. If a nonhomogeneous equation has more unknowns than equations AND is consistent, then it will have infinitely many solutions (will prove later on).
  - If there are more unknowns than there are equations, then it is implied that there are less leading 1s than there are free variables.

  - This also implies that there is at least 1 free variable, meaning that it will always have an infinitely many solution scenario.

- A homogeneous system that has more unknowns than linear equations has infinitely many solutions.

- for small systems, using Gauss-Jordan elimination is sufficient (forward pass + backward pass). But for larger systems that require compute power, Guassian Elimination + **back-substitution** is more efficient. ¡- I.e., in AI

- Some cool facts about row echelon and reduced row echelon matrices that I will try to prove in this text:

  - Regardless what techniques you use to bring the matrices to reduced row echelon form, they will all look the same (have the same rows, unknown values, etc).
  - Row echelon forms are not unique. Different row operation forms can produce different row echelon forms. ¡- i.e., difference may be scaling of each row, combinations of rows, etc. But if you bring these row echelon form matrices to reduced row echelon form, regardless of the operations/order of operations, it will result in the same reduced row echelon form.
  - All row echelon forms of a matrix A has the same number of zero rows and all leading 1s are in the same positions.
  - pivot positions (**pivot positions of** A)
  - pivot columns (**pivot columns of** A.

- Disparity between theory and implementation:

  - roundoff errors in computers
  - **unstable** algorithms
  - For large systems, Gauss-Jordan elimination involves 50% more operations than Gaussian elimination, so most computer algorithms resort to using Gaussian elimination (to reduce roundoff errors).