An Intuitive Overview of Linear Algebra Fundamentals

Table of Contents

Introduction

Why Learn Linear Algebra?

Linear algebra is incredibly important. Do you want to be an engineer, chemist or computer scientist? Learn linear algebra. Do you want to program video games? You definitely want to be an expert in linear algebra. You can even write a cookbook in the form of a matrix! Do you want to understand how the universe works? Linear algebra is the language of quantum mechanics.

The point is this: basic linear algebra is rather simple, yet it's extremely useful in many areas and disciplines. Putting the applications aside, linear algebra itself is a fascinating, self-contained mathematical field on its own, so you should definitely take some time and at least get an intuition of what it all means!

This write up is an overview of some of the linear algebra fundamentals. It focuses on providing an intuitive / geometric review of some of the main concepts. The coverage is by no means comprehensive or complete. It's meant to serve as review material for those who haven't touched it in a very long time, or as a brief intro for those who want an over-the-top view of the landscape.

Noting the above, it's time to begin. So, what is linear algebra?

Linear algebra is the study of vectors and linear functions.

Functions are computations / transformations that take a set of inputs and produce an output. Functions of several variables are often presented in one line such as:

$$f(x, y) = 3x + 5y$$

In mathematics, the concept of linearity plays a very important role. Mathematically, a linear function, or linear map, or linear operator, f is a function that satisfies:

1. $f(x+y) = f(x)+ f(y)$, for any input x and y
2. $f(cx) = cf(x)$ for any input x and any scalar c

Put into words, the first condition means that the output of a function acting on a sum of inputs is just equal to the sum of the individual outputs. The second condition implies that the output of a scaled input is just the scaled output of the original input.

Linear functions are those whose graph is a straight line. A linear function has the following form.

$$y = f(x) = a + bx.$$

Here, x is our independent variable and y is our dependent variable (since the result (y) depends on the transformation we perform on x).

Notice that although we classify one variable as being dependent on the other, in realty, both variables are linked and dependent on each other. We simply choose to classify the variable which takes an input as the 'independent' one. In our example above, we can just as easily choose to exchange our variables and make y the independent variable and x the dependent one.

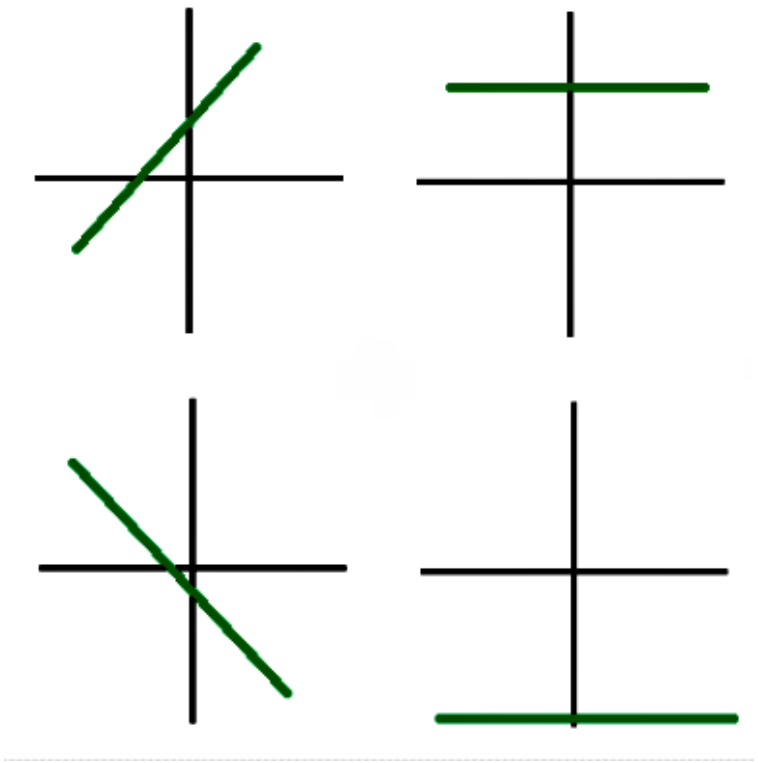
Let's take $y = f(x) = 3x + 1$ as an example. We can choose to model the above function in terms of y by taking the inverse, so we get:

$$y = 3x + 1$$
$$y - 1 = 3x$$
$$x = f(y) = y/3 - 1/3$$

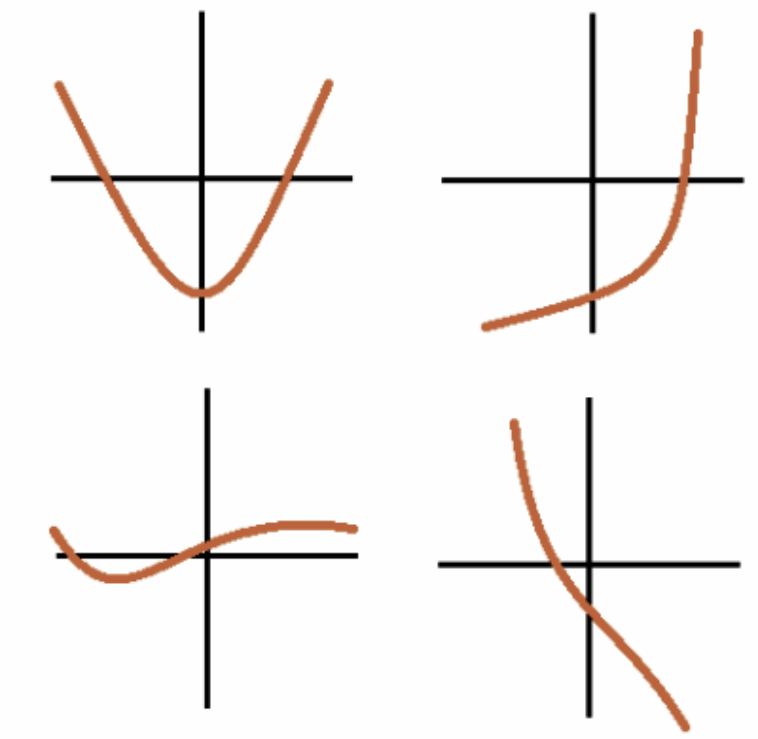
In either case, you get the point. Our linear function can be viewed as representing a linear relationship between two or more variables. The relationships between these variables are the relationships we model in linear algebra.

To make things easier to grasp, we usually represent our variable relationship visually by projecting input variable x on a horizontal axis (which we call the x-axis) and showing its mapping to variable y on a vertical axis (which we call the y-axis). This lets us get a visual intuition of how our function works in transforming inputs to outputs, and lets us model our linear transformations geometrically.

Some graphical examples of linear functions are provided below:



Of course, linear functions aren't the only types of functions. Non-linear functions don't have a linear mapping and are not represented by straight lines. Some examples of non-linear functions are provided below:



Vectors

The key starting point of linear algebra is the concept of vectors. In high school physics, chances are you've already seen that concept as being nothing more than "a number and a direction". This isn't false, but it's definitely not the whole story. It's a rather intuitive way of introducing vectors, hence why we'll use this analogy extensively.

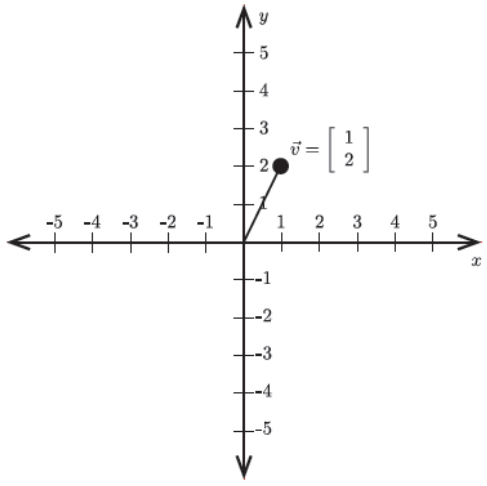
From a mathematical perspective, vectors are just a way of stacking numbers together in a column or a row. For example:

$$\begin{bmatrix} 2 \\ 3 \end{bmatrix}, \quad \begin{bmatrix} i \\ 3 \\ -3 \end{bmatrix}$$

$$\begin{bmatrix} 5 & 2.1 \end{bmatrix}, \quad \begin{bmatrix} -3 & \frac{1}{2} & 4 \end{bmatrix}$$

The first two vector examples are naturally referred to as column vectors and the last two as row vectors.

The amount of numbers is referred to as the dimension of the vector. Even if you've never explicitly learned about vectors until now, you've already seen them. A 2-dimensional vector of real numbers is analogous to the Cartesian coordinates. For example, a 2-dimensional vector looks like:



A 3-dimensional vector of real numbers is analogous to the spatial coordinates in three dimensions. You can always think of a vector with n numbers as a point in n-dimensional “hyperspace”.

For simplicity, we’ll generally use 2 and 3 dimensional vectors from now on, but everything we explain below applies to vectors of arbitrary sizes.

Vector Addition

Since we just defined a new mathematical concept, it’s natural to ask the question: can we add and multiply vectors?

Imagine the vectors v and w:

$$\vec{v} = \begin{bmatrix} 3 \\ 1 \end{bmatrix} \quad \text{and} \quad \vec{w} = \begin{bmatrix} 1 \\ 2 \end{bmatrix}$$

represent two different movements on a 2-dimensional plane. For example, v can be thought as moving along the x-axis by 3 and along the y-axis by 1. Adding two vectors is essentially the same thing as saying:

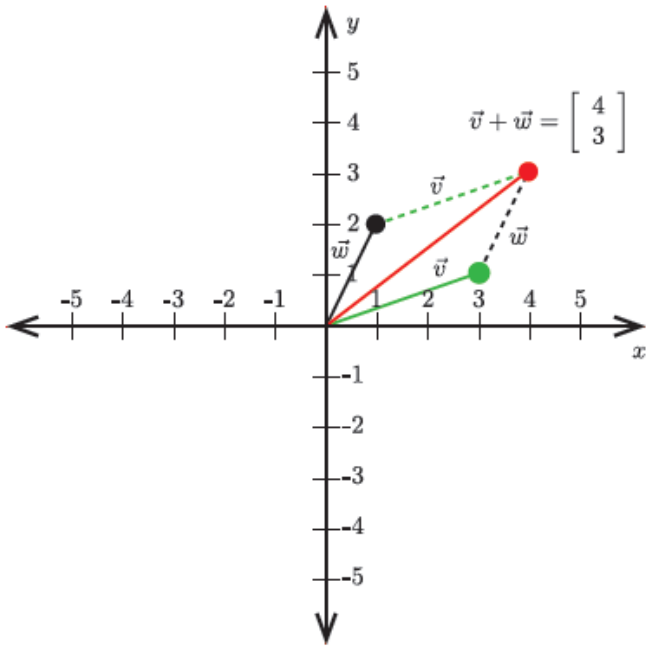
“Start at the origin. Move along x by 3, then along y by 1. After that, move along x by 1, and then along y by 2. Your final position will be at 4 along x and at 3 along y.”

In other words, adding vectors is easy, just add each corresponding component!

Let’s take 2 example vectors v and w:

$$\vec{v} + \vec{w} = \begin{bmatrix} v_1 + w_1 \\ v_2 + w_2 \end{bmatrix}$$

and demonstrate how our additions map in 2-D space capture below:



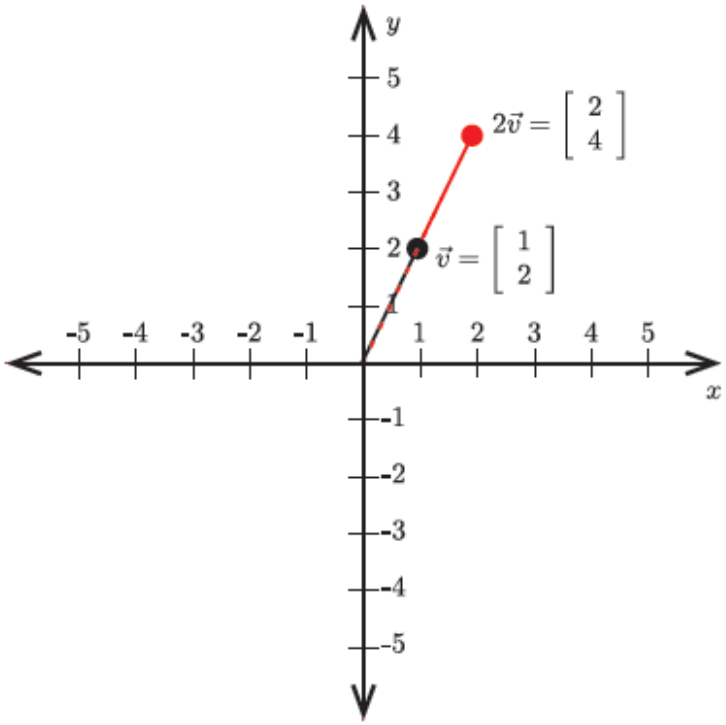
Vector Multiplication

You can scale a vector by just multiplying each entry by the scalar (the number).

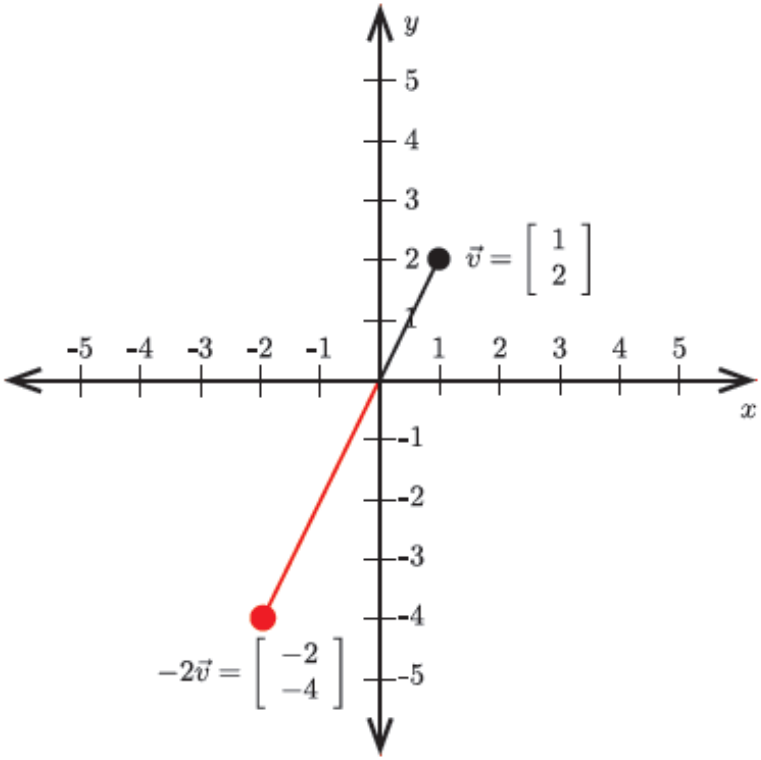
Let's assume we have vector:

$$\vec{v} = \begin{bmatrix} 1 \\ 2 \end{bmatrix}$$

Let's assume we want to scale the above vector by 2. All we have to do is multiply each component by 2 and we get:



Scaling a vector by a negative number inverts its direction:

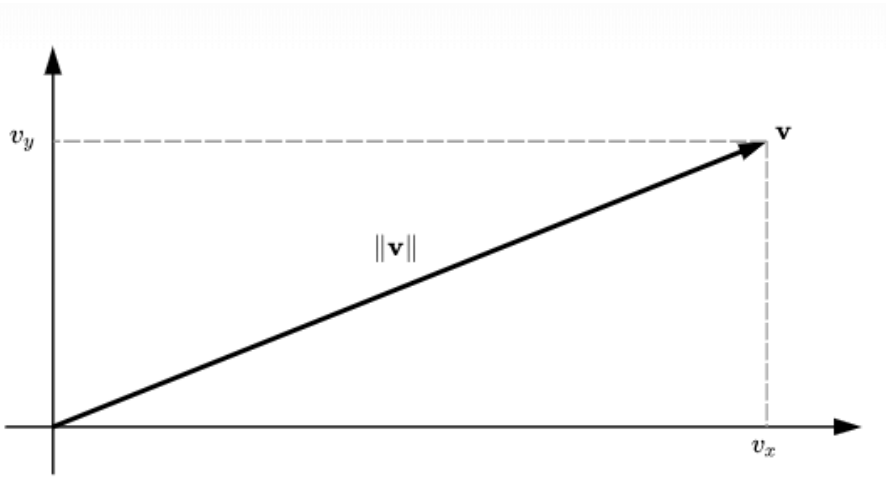


Vector Length

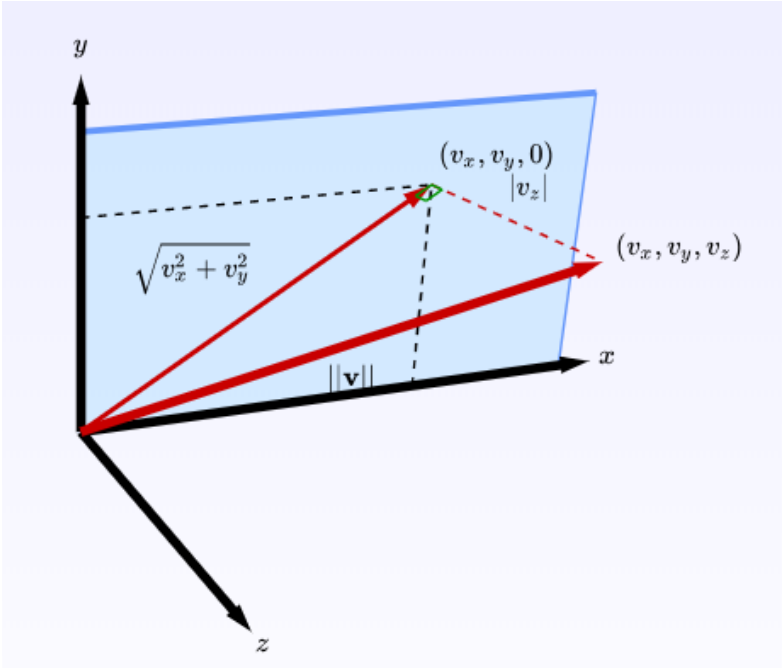
In 2-D, it's easy to visualize how to get the vector length. We can use the Pythagorean theorem in order to show that it's always going to equal to:

$$\sqrt{|v_1|^2 + |v_2|^2}$$

which we can see using the figure below:



Let's now try to get a geometric intuition of how this would translate in 3-dimensions



If we imagine our vector projecting outwards from 2-D into the z-axis, we can see that we once again get a right-angled triangle whit a hypotenuse length equal to our vector length:

$$||\mathbf{v}||^2 = (\sqrt{v_x^2 + v_y^2})^2 + v_z^2$$

and thus ||v|| must be:

$$\sqrt{v_x^2 + v_y^2 + v_z^2}.$$

We can keep adding dimensions and we get the same result.

In other words, the norm (or length) of an n-dimensional vector v, denoted by ||v||, is given by:

$$\begin{aligned}\|\vec{v}\| &= \sqrt{\vec{v} \bullet \vec{v}} \\ &= \sqrt{\vec{v}^{\dagger}\vec{v}} \\ &= \sqrt{|v_1|^2 + |v_2|^2 + \ldots + |v_n|^2}\end{aligned}$$

Unit Vectors and Normalization

By definition, unit vectors have norm equal to 1. Normalizing a non-zero vector v simply means that we're stretching it so that its direction remains the same but its length equals 1. To do this, we have to scale the vector by 1 / ||v||.

For example, let's normalize the following vector:

$$\vec{v} = \begin{bmatrix} 1 \\ -2 \end{bmatrix}$$

Since ||v|| = $\sqrt{|1|^2+|-2|^2}=\sqrt{5}$, we can scale it by $1/\sqrt{5}$ to get the unit vector:

$$\frac{1}{\sqrt{5}}\vec{v} = \frac{1}{\sqrt{5}}\begin{bmatrix} 1 \\ -2 \end{bmatrix} = \begin{bmatrix} 1/\sqrt{5} \\ -2/\sqrt{5} \end{bmatrix}$$

Dot Product

The dot product is a very useful definition in linear algebra. It's also known as the inner product or the scalar product. If we take the dot product of two vectors v and w, we get a scalar value showing the result of multiplying each of the corresponding vector components of v and w and adding them together.

Formally, we can say that if v and w are the vectors provided below:

$$\vec{v} = \begin{bmatrix} v_1 \\ v_2 \\ \vdots \\ v_n \end{bmatrix} \quad \text{and} \quad \vec{w} = \begin{bmatrix} w_1 \\ w_2 \\ \vdots \\ w_n \end{bmatrix}$$

Then the dot product of v and w gives:

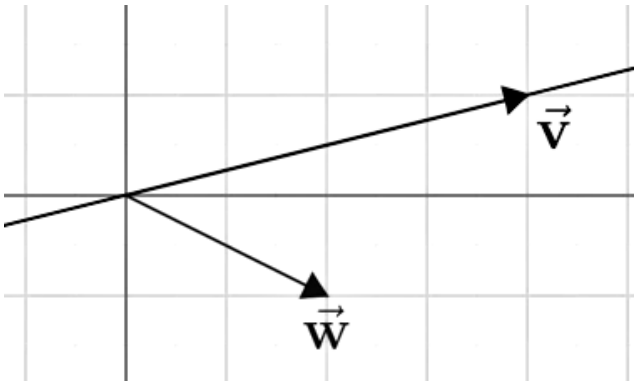
$$\vec{v} \bullet \vec{w} = \overline{v_1} w_1 + \overline{v_2} w_2 + \ldots + \overline{v_n} w_n$$

So, what does this mean?

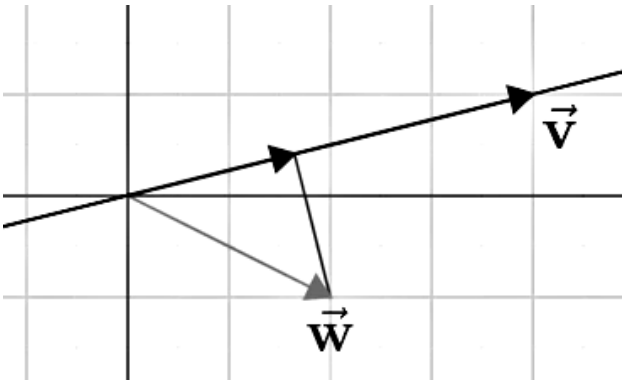
In the instance above, we can picture the dot product geometrically as taking the vector w and projecting it onto v, and then taking the length of this projection and multiplying it by v to get the scalar which represents our dot product.

Let's see a visual which might make the above easier to grasp.

Let's say that vectors v = [4, 1] ^{T} and w = [2, -1] ^{T} shown below:

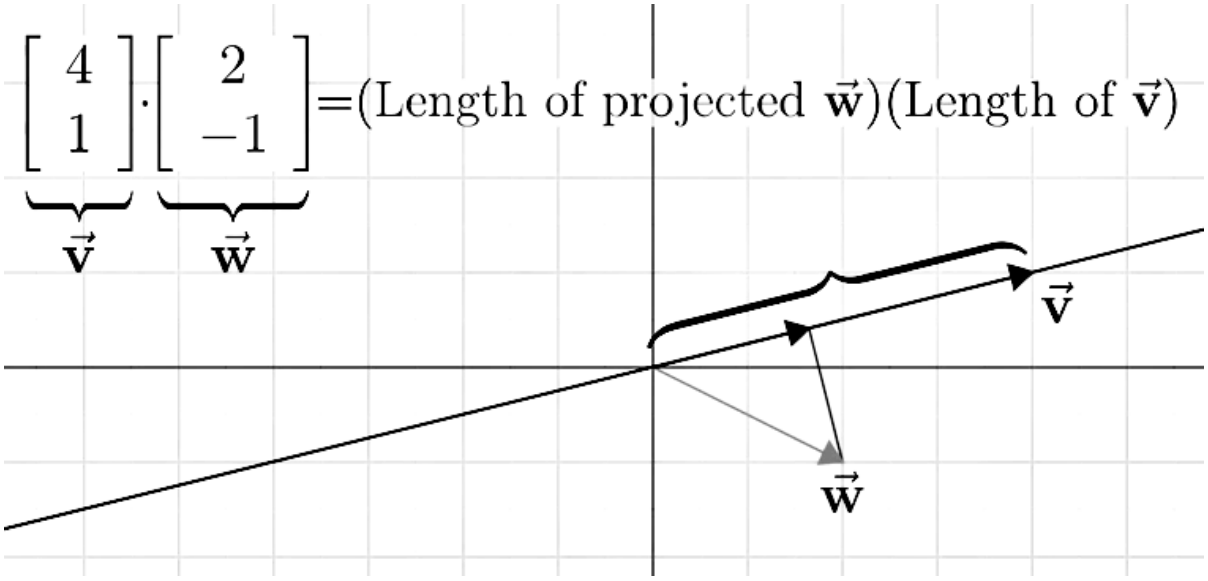


We visualize the projection of w onto v as:



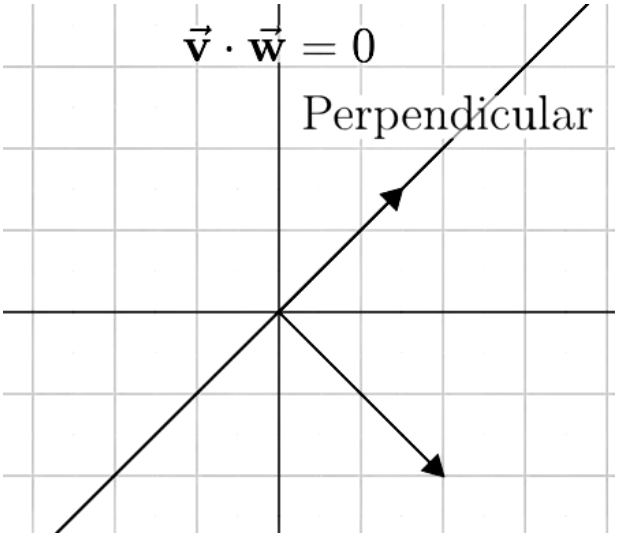
We then take the length of this projection and multiply it by the length of v to obtain our final result:

$$4(2) + 1(-1) = 7$$



As a note, if our projection of w were pointing in the opposite direction of v, we would get a negative dot product.

What would happen if we were to have w oriented 90 degrees to vector v? In other words, what happens when we get vectors which are perpendicular?



We can easily see that our projection in this instance would result in a vector of length 0. Since anything multiplied by 0 equals 0, we're ensured to always get a dot product of 0 for any vectors perpendicular to each other. We call such vectors **orthogonal vectors.**

Basis Vectors

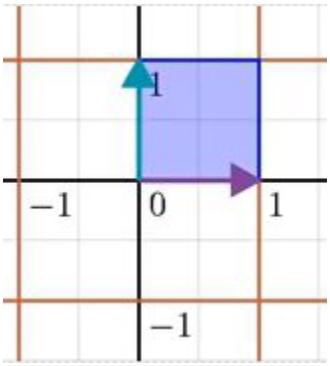
A very important concept in linear algebra is that of basis (or bases in plural). A basis is a finite set of vectors that can be used to describe any other vectors of the same dimension. For example, any 2-dimensional vector v can be decomposed as:

$$\begin{aligned}\vec{v} &= \begin{bmatrix} v_1 \\ v_2 \end{bmatrix} \\ &= v_1 \begin{bmatrix} 1 \\ 0 \end{bmatrix} + v_2 \begin{bmatrix} 0 \\ 1 \end{bmatrix}\end{aligned}$$

The set:

$$\left\{ \begin{bmatrix} 1 \\ 0 \end{bmatrix}, \begin{bmatrix} 0 \\ 1 \end{bmatrix} \right\}$$

is a basis for vectors of dimension 2 and v_1 and v_2 are the coefficients of each basis vector. We provide a simple visual showing the basis below:



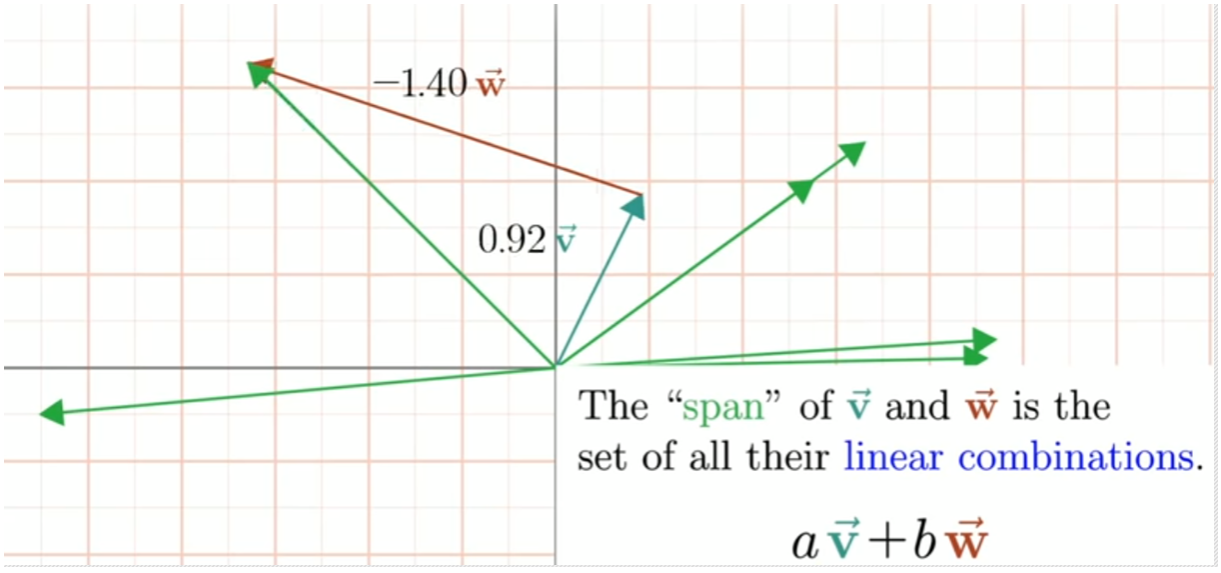
As you can see, this example clearly relates to the Cartesian plane, where we can always describe any point on the plane as “how much of x” and “how much of y”.

Can you think of a different basis? The fact of the matter is, there’s an infinite number of them! The concept of using different basis vectors and dimensions in order to transform planes plays a vital role in linear algebra, and we’ll discuss this further when we discuss the topic of matrices.

Span / Linear Dependence and Independence

The span is the set of all allowed linear combinations that are reachable through linear scaling of vectors.

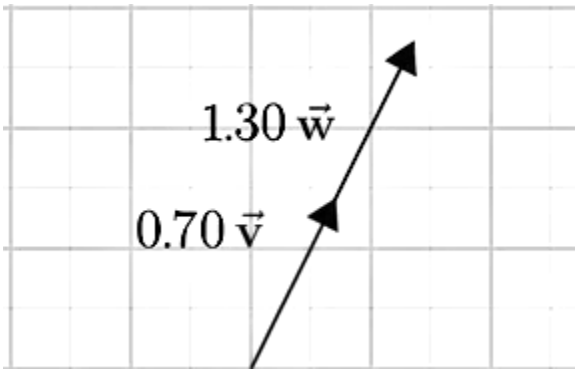
Let’s take the 2-D example below:



We can see from the example above that if we were allowed to ‘stretch’ or scale vectors v and w in any direction or by any factor that we want, the span of points these 2 vectors can reach encompasses all of 2-D space! In other words, using our 2 vectors, we can reach any 2 points we want to reach on our Cartesian plane.

Will a set of vectors with 2 dimensions always span the entire 2-d space?

Not necessarily. Let’s take an example of 2 vectors which point in the same direction:



Using the example above, we can clearly see that we’re limited in our ability to span the space. Combining the 2 vectors only allows us to reach a subset of points which lie on a straight line in this space (which we call a sub-space). Adding vector w to our basis (composed of v) doesn’t really give us the flexibility of extending our 1-dimensional plane into one with 2-dimensions. By adding this new linear mapping to our space, we’re simply adding redundancy to an already existing system, and we can thus say that vectors v and w are **linearly dependent.**

Let’s take another example.

Given the three vectors:

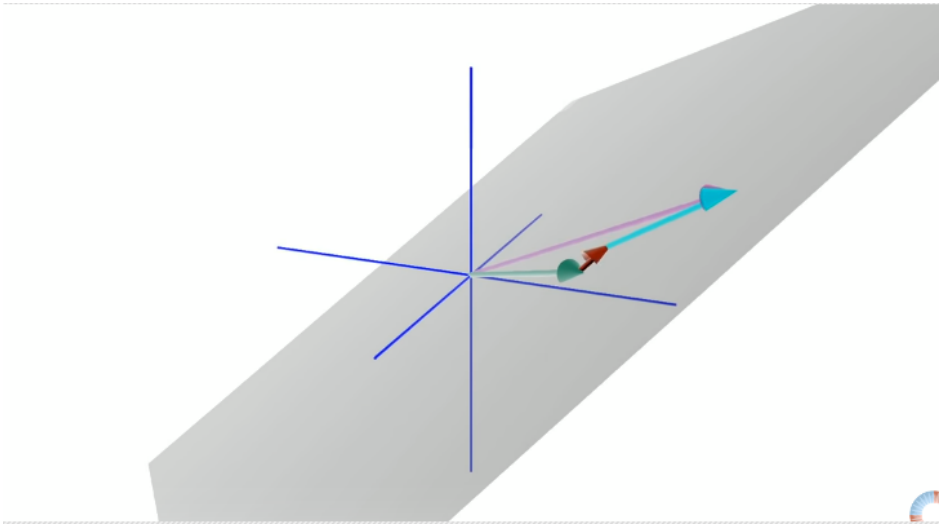
$$\left\{ \vec{v}_1 = \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix}, \quad \vec{v}_2 = \begin{bmatrix} 1 \\ 0 \\ 1 \end{bmatrix}, \quad \vec{v}_3 = \begin{bmatrix} 3 \\ 2 \\ 3 \end{bmatrix} \right\}$$

Can we say that the vectors are linearly independent?

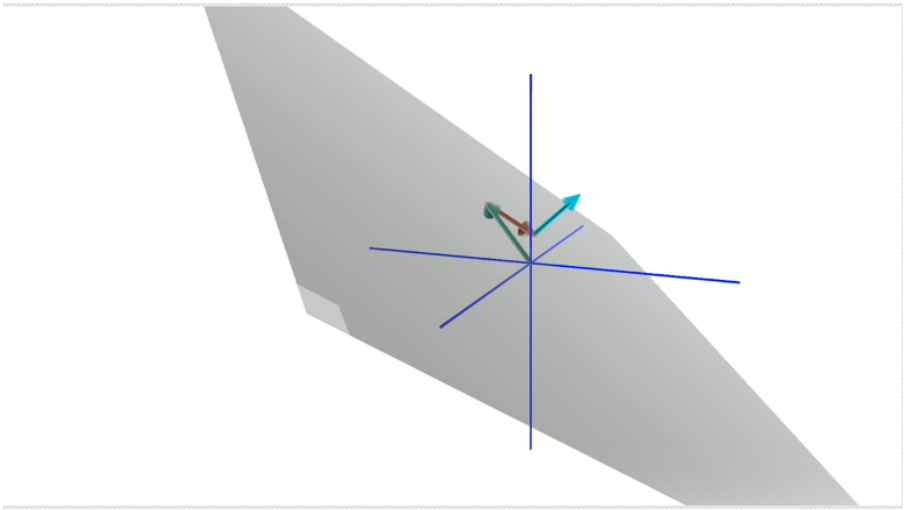
If they were, the fact that they have 3-dimensions and the fact that we have 3 vectors should give us the ability to span all of 3-d space. Looking closely at the above vectors, we can see that this isn’t the case. These 3 vectors are linearly dependent since:

$$v_3 = 2v_1 + v_2$$

We can envision the above 3 vectors giving us the ability to span a slice located in 3-d space, as shown below:

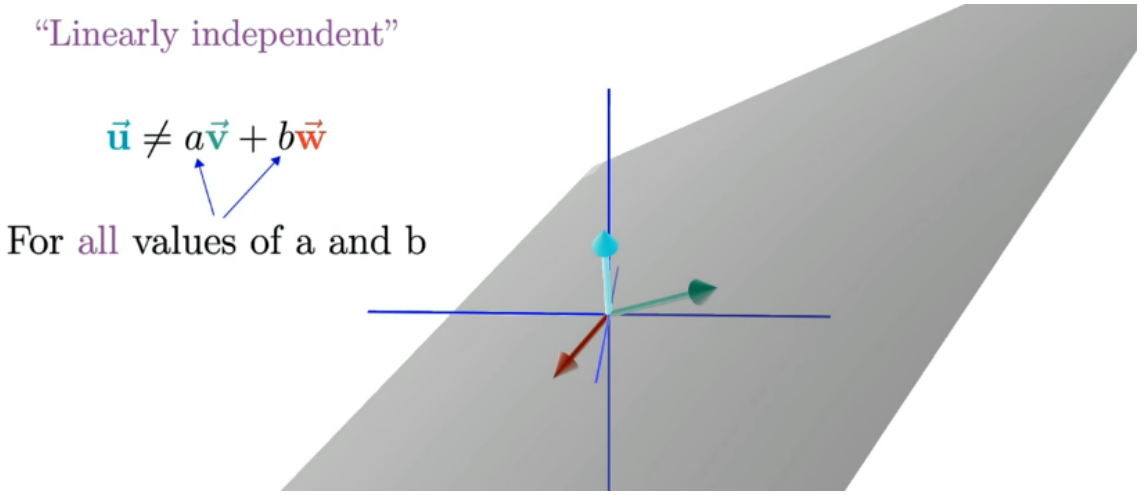


Even though we have 3 vectors, we can only transform them to cover a plane or slice through this space without having the ability to move through it. If we were to have full linear independence, we would be allowed to move our 2-d plane throughout this space by scaling one of our 3 vectors. We can visually show this by changing our blue vector so that it's independent and able to push our plane upward (or downward):



This should give you an intuitive grasp of what it means for vectors to be dependent or linearly independent. We can extend the above concept to any number of dimensions.

More formally, we say that a set of vectors is linearly dependent if at least one of the vectors can be written as a linear combination of the others. Otherwise they're linearly independent.



Why are span and linear dependence / independence important?

Simply put: they allows us to determine what types of solutions we can obtain by using our system of linear relations.

There are three separate cases to consider:

1. No solution
2. One solution
3. An infinite number of solutions

If our system is inconsistent, we know that we have no solutions. If the system is consistent and linearly independent, then we know that we can only have one solution. If the system is consistent and linearly dependent, then we know we have an infinite number of solutions.

Matrices

Let’s increase the complexity a little bit by introducing the concept of matrices.

A matrix is a box of numbers (real or complex). A square matrix is a matrix with the same number of rows and columns.

At first glance, there doesn’t seem to be much intuition about matrices. As we’ll see soon enough, a matrix can be thought of as a mathematical object “acting on vectors.” These mathematical objects can at the same time be thought of as linear transformations of space. Having a good intuition about these functions / transformations is important in having a good intuition and understanding linear algebra, so we will review this key area in our later sections.

Matrix Addition and Scalar Multiplication

Similar to vector addition, adding two matrices, as long as both matrices have the same dimensions, consists of adding each corresponding matrix element. Similarly, scalar multiplication is simply the scaling of all the elements of the matrix.

Simple matrix addition example:

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

$$= \begin{bmatrix} 8 & 9 \\ 10 & 5 \end{bmatrix}$$

Simple scalar multiplication example:

$$2A = 2 \cdot \begin{bmatrix} 10 & 6 \\ 4 & 3 \end{bmatrix}$$

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

$$= \begin{bmatrix} 20 & 12 \\ 8 & 6 \end{bmatrix}$$

Matrix Multiplication

Although the definition of matrix multiplication might seem to be a little confusing, an explicit multiplication example might clarify this simple task. We'll multiply the two following matrices to quickly show an example of matrix multiplication for anyone needing a refresher:

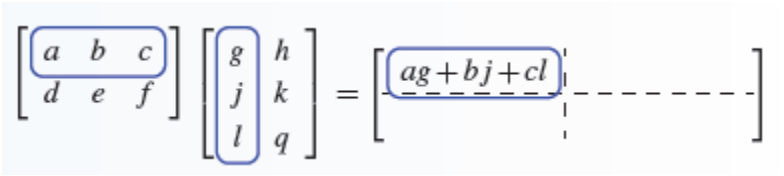
$$M = \begin{bmatrix} a & b & c \\ d & e & f \end{bmatrix}, \quad N = \begin{bmatrix} g & h \\ j & k \\ l & q \end{bmatrix}$$

Since A is a 2×3 matrix and B is a 3×2 matrix, we know that the result of the multiplication is a 2×2 matrix:

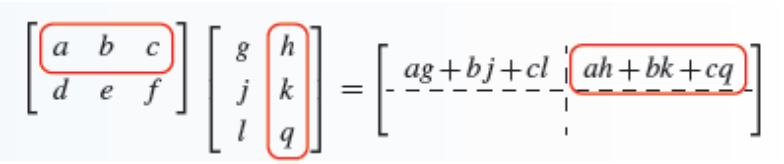
$$\begin{bmatrix} a & b & c \\ d & e & f \end{bmatrix} \begin{bmatrix} g & h \\ j & k \\ 1 & q \end{bmatrix} = \begin{bmatrix} \text{-------} & \text{-------} \\ & \end{bmatrix}$$

Let's go through the steps below:

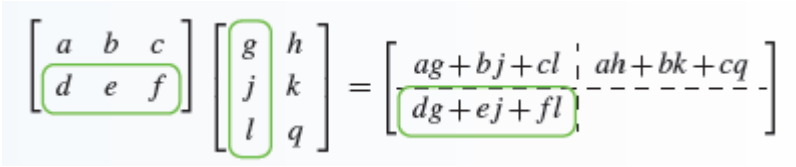
Step 1: We multiply the 1st column of M by the 1st row of N to get our entry at [1, 1]:



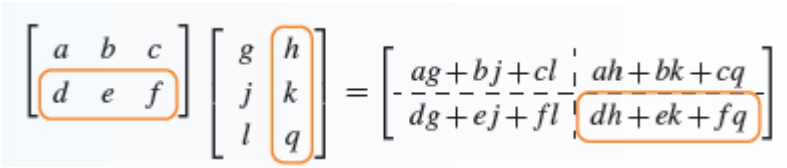
Step 2: We multiply the 1st column of M by the 2nd row of N to get our entry at [1, 2]:



Step 3: We multiply the 2nd column of M by the 1st row of N to get our entry at [2, 1]:



Step 4: We multiply the 2nd column of M by the 2nd row of N to get our entry at [2, 2]:



We can see from the above that **matrices can be thought of as functions on vectors**. We can also think of a vector as a matrix that has only one column. An m by n matrix maps an n-dimensional vector to an m-dimensional vector.

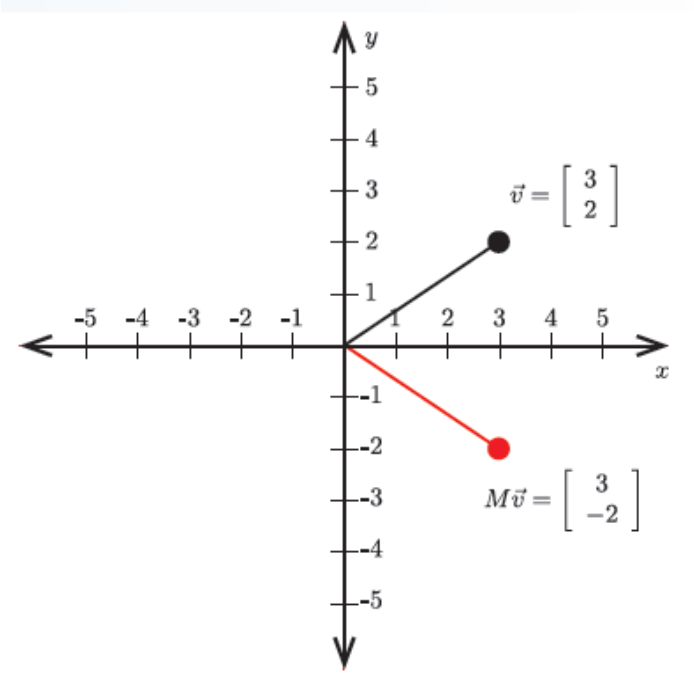
As an example, consider:

$$\vec{v} = \begin{bmatrix} 3 \\ 2 \end{bmatrix} \quad \text{and} \quad M = \begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix}$$

The matrix M acts as:

$$M\vec{v} = M\begin{bmatrix} 3 \\ 2 \end{bmatrix}$$
$$= \begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix}\begin{bmatrix} 3 \\ 2 \end{bmatrix}$$
$$= \begin{bmatrix} 3 \\ -2 \end{bmatrix}$$

If you visualize v as a vector in the Cartesian plane, the matrix M performs a reflection of the x-axis:



Non-Commutativity of Matrix Multiplication

An interesting property of matrix multiplication is its non-commutativity. In mathematics, we say an operation is commutative if the order of operations is irrelevant.

For example, if a and b are any scalar number, then a + b = b + a and ab = ba.

Clearly, matrix addition and matrix scalar multiplication are commutative operations, but what about matrix multiplication? First, we must notice that asking the question about commutativity only makes sense for square matrices. Let’s look at the following example, where we multiply 2 matrices in different order to obtain 2 different results:

$$MN = \begin{bmatrix} 1 & 2 \\ 3 & 1 \end{bmatrix} \begin{bmatrix} 4 & 3 \\ 2 & 1 \end{bmatrix}$$
$$= \begin{bmatrix} 8 & 5 \\ 14 & 10 \end{bmatrix}$$
$$NM = \begin{bmatrix} 4 & 3 \\ 2 & 1 \end{bmatrix} \begin{bmatrix} 1 & 2 \\ 3 & 1 \end{bmatrix}$$
$$= \begin{bmatrix} 13 & 9 \\ 5 & 5 \end{bmatrix}$$

These are clearly not the same matrices, therefore MN is not equal to NM. There are cases where matrices will commute, but in general they do not.

We can think of matrix multiplication in terms of function composition. Lets imagine that after applying a function g on an input, we then apply another function f using our g output as f's input. Abstractly, we denote the composition of function f and g as f(g(x)).

Transposing this to matrices and vectors, lets say that f(v) = Mv and g(v) = Nv for any vector v.

We thus have:

$$f(g(v)) = f(Nv) = MNv$$

Therefore, applying the function f after the function g is the same as applying the operator N, and then the operator M. Remember, matrices do not commute under matrix multiplication. Therefore, it's crucial to understand that matrix composition goes from right to left. That is, the rightmost matrix is the one being applied first, followed by the matrices prior to this one, and so on.

Properties of Matrix Arithmetic

For the sake of listing properties of matrix addition and multiplication, we'll assume the dimensions of M, N and P are compatible for the given operations performed:

1. M+N = N +M
2. (M+N)+P = M+(N +P)
3. c(M+N) = cM+cN for any scalar c

4. (c+d)M = cM+dM, for any scalar c and d
5. c(MN) = (cM)N = M(cN) = (MN)c, for any scalar c
6. (MN)P = M(NP)
7. (M+N)P = MP+NP
8. M(N +P) = MN +MP
9. MN != NM, in most cases

Matrix / Vector Transpose

The transpose of a matrix M, denoted M^T is such that the nth row of M^T is the same as the nth column of M.

Note that the transpose of an m by n matrix is an n by m matrix. It follows that the transpose of a column vector v, denoted v^T is just a row vector with the same entries as v. For example, if:

$$\vec{v} = \begin{bmatrix} a \\ b \end{bmatrix}, \quad M = \begin{bmatrix} c & d \\ f & g \end{bmatrix}$$

then:

$$\vec{v}^t = \begin{bmatrix} a & b \end{bmatrix}, \quad M^t = \begin{bmatrix} c & f \\ d & g \end{bmatrix}$$

Matrices as Linear Transformations in Space

In our previous overview of matrices, we showed that a matrix can be thought of as a a function which takes vectors as inputs and outputs a vector which may have a different dimensionality than our input vector. Here, we want to focus on a different interpretation which works in a similar vein, although the below representation will give us a better view on how our matrix function does its magic behind the scenes in order to perform its vector transformations.

We can do this by modeling matrices as transformations in space, whereby the first column transforms our 1st basis vector into a different basis vector, the 2nd column of our 2nd basis into another, etc…

We can thus think of matrices as functions which warp our Cartesian coordinate basis vectors into a different set of basis vectors which are represented by a new coordinate system.

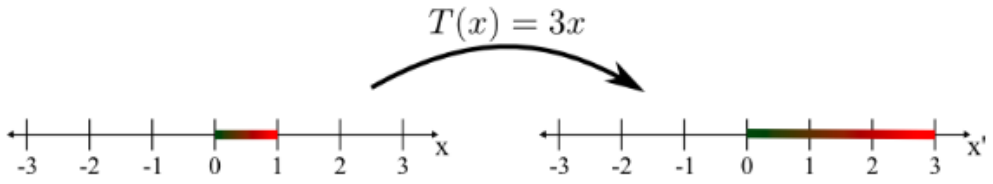
One-Dimensional Linear Transformations

A one-dimensional linear transformation is a function T(x) = ax for some scalar a.

An example one-dimensional linear transformation is the function T(x) = 3x. We could visualize this function by its graph, which is a line through the origin with slope 3. However, instead, let's view it as a mapping from the real line R back onto the real line R.

In this case, we think of the function as x′ = T(x), which maps the number x on the x-axis to the new number T(x) on an x′-axis.

T for example takes the number 1 and transforms / maps it to 3. We can also view these transformations as linear functions.



Two-dimensional linear transformations

A two-dimensional linear transformation is a function $T:R^2 \to R^2$ of the form

$$\mathbf{T}(x, y) = (ax + by, cx + dy) = \begin{bmatrix} a & b \\ c & d \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix},$$

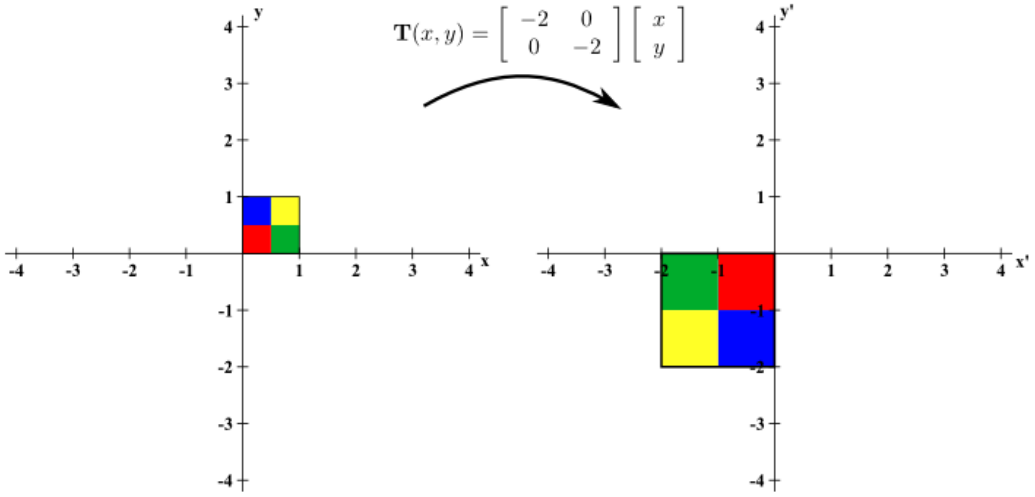
where a, b, c, and d are numbers defining the linear transformation. We can write this more succinctly as T(x) = Ax, where x=(x, y) and A is the 2×2 matrix containing the constants that define the linear transformation,

$$A = \begin{bmatrix} a & b \\ c & d \end{bmatrix}.$$

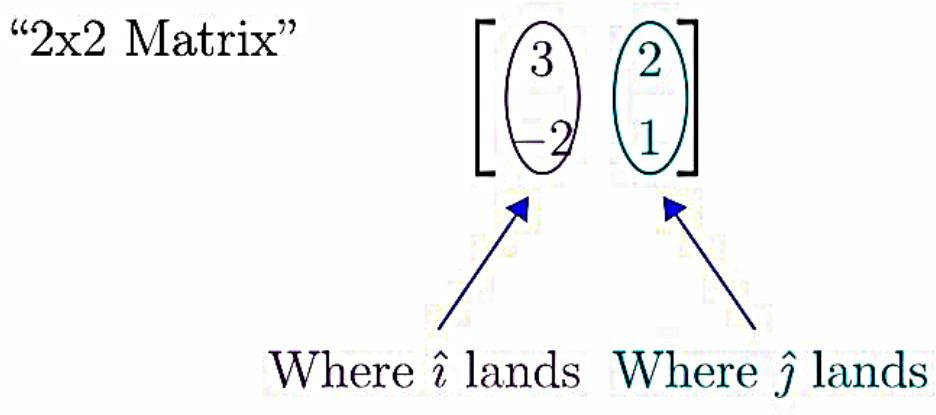
As an example, let’s look at the linear transformation:

$$\mathbf{T}(x, y) = \begin{bmatrix} -2 & 0 \\ 0 & -2 \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix}.$$

We can get a feel for the behavior of T by looking at its action on the standard unit vectors, i = (1, 0) and j = (0, 1). T maps (1, 0) to (−2, 0), and it maps (0, 1) to (0, −2). It stretched both vectors by a factor of 2 and rotated them in space:



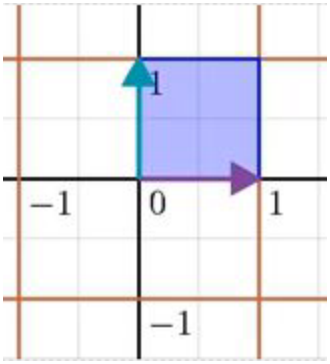
As another example, let’s take the matrix below:



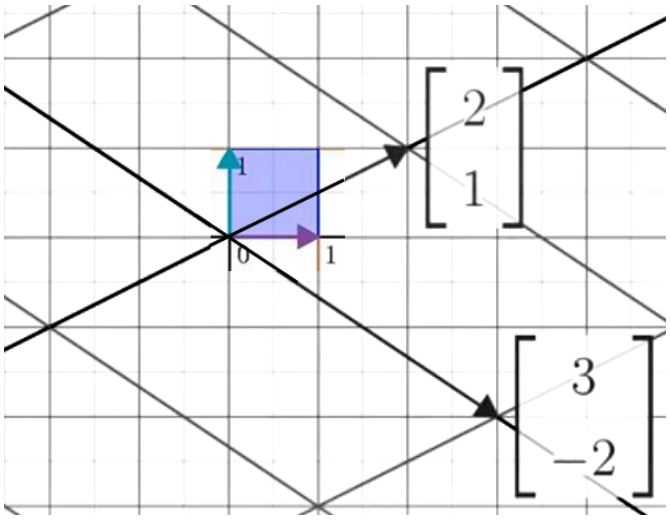
In linear transformations, we're simply stretching space such that:

- Our origin remains at the origin and
- Our lines remain evenly spaced after stretching.

In the above instance, we're transforming our normal Cartesian basis shown below:



into one with a set of basis vectors and grid lines shown:

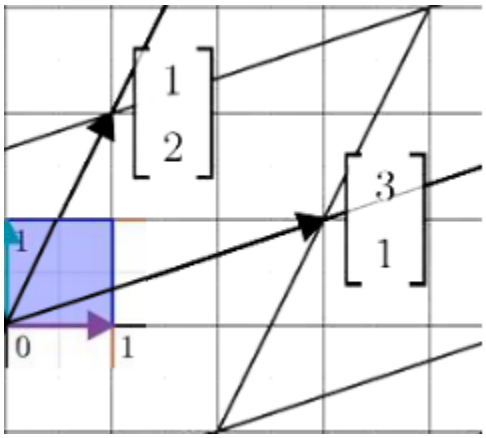


Pay attention to the regular basis vector shown in the above figure. What we're doing is essentially transforming our i basis vector from the Cartesian plane $[1, 0]^T$ into a new basis with i equal to $[3, -2]^T$. In a similar manner, we're transforming our j basis from $[0, 1]^T$ into $[2, 1]^T$. We thus end up with a new basis space which transforms our regular 1x1 Cartesian plane grid into a different grid expressed by our new basis vectors.

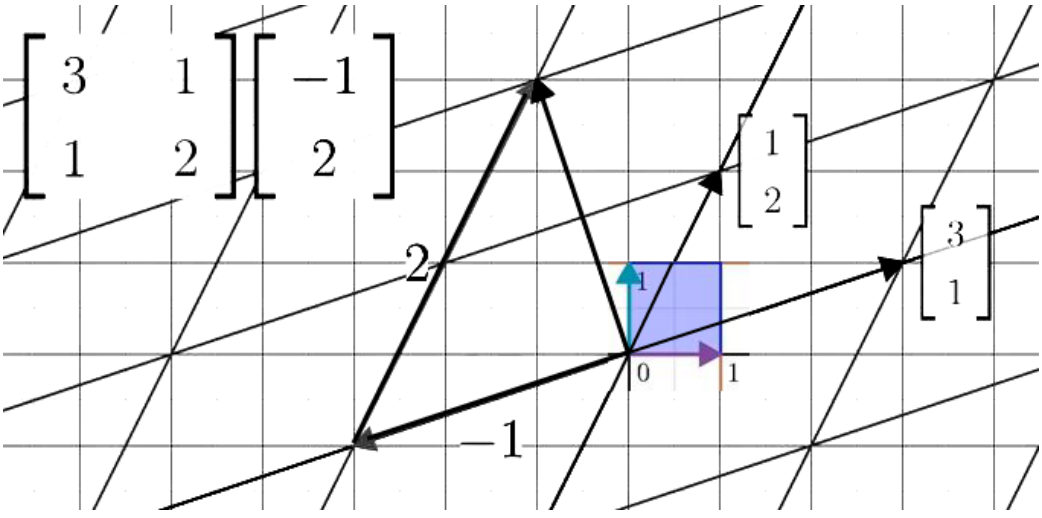
Let's do another example and use the below matrix:

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

We can express this new basis space using a new grid which translates our original i and j Cartesian vectors into one which is represented by vectors shown below:



Let's express a vector in this new basis space. Let's say that we want to show the vector $[-1, 2]^T$. To translate our vector, we simply scale our new i basis vector by our x-coordinate (-1) resulting in a vector pointing to $[-3, -1]^T$. We then add our new j vector $[1, 2]^T$ scaled by a factor of our y-coordinate (2) which results in a vector pointing to $[-1, 3]^T$. We can show this linear transformation using the diagram outlined below:

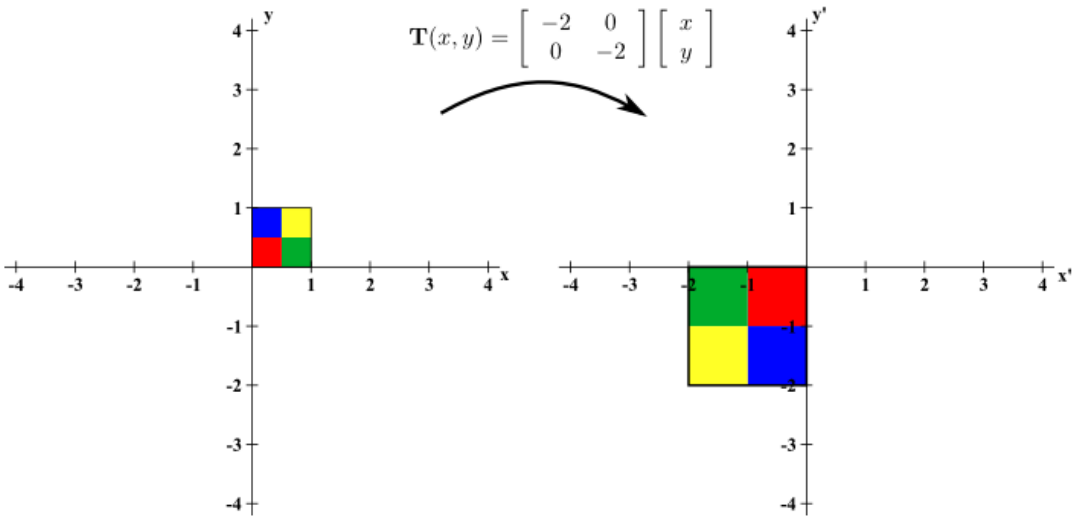


We can confirm that our transformation is correct by performing the actual matrix multiplication, which will yield the same results.

Determinants

We can think of a determinant as showing how much of a factor a matrix stretches our 'space' or basis. The determinant of a matrix tells us important geometrical properties of its associated linear transformation.

In the below case, we're stretching our original unit basis which has an area of 1 into a new basis which has an area of 4:



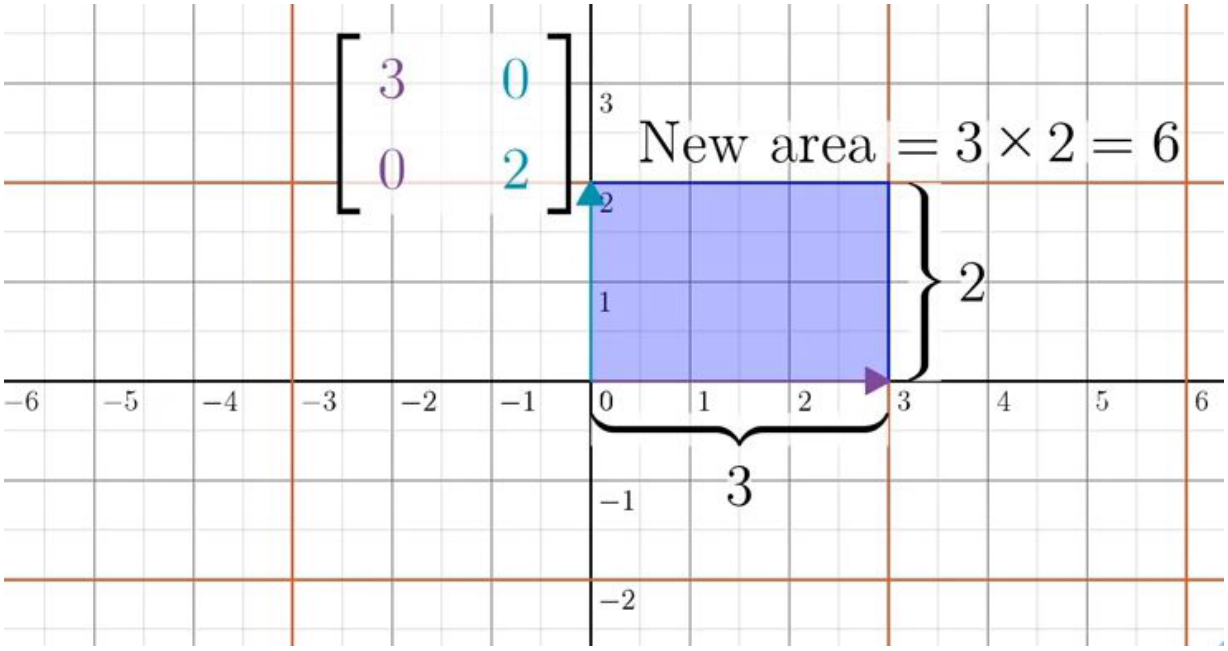
We can confirm this by doing a manual calculation and showing the determinant. The formula for a determinant in 2-D space is:

$$\det\left(\begin{bmatrix} a & b \\ c & d \end{bmatrix}\right) = ad - bc$$

Using our formula, we can see that

$$\det(A) = (-2)(-2) - (0)(0) = 4.$$

Let's take another example matrix and show how it relates to our determinant:



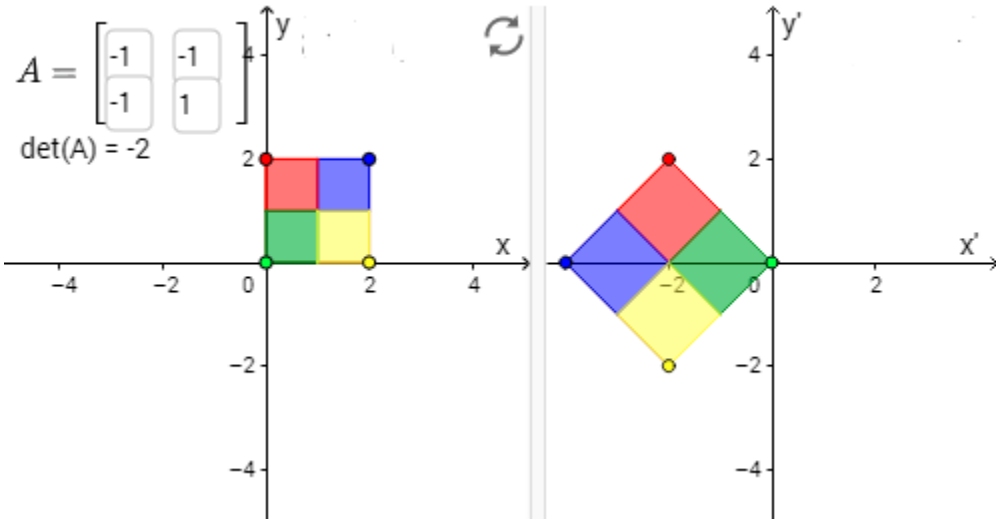
In the above example, we can see that the matrix in question turns our original 1x1 grid into a 3×2 grid, and we can safely say that our determinant here is 6, since we're stretching the basis by a factor of 6.

As a note, if we take a closer look at our formula, we should be able to easily see that it's possible for us to have a negative determinant. So, if we are trying to generalize a determinant as the 'stretch' factor that's applied to a linear space, how can we get a negative value?

Another way of visualizing 2-D linear transformations is by imagining a 2-D grid as being a sheet of space. We can create a matrix which changes our basis where the j vector (y axis) is now to to the right of our i vector (x-axis), and we can imagine this as flipping our sheet of paper to face the other side.

This is essentially why we have negative determinants. It means that we're flipping space to the other orientation / side and changing the orientation of our basis vectors.

Let's take another example which transforms our i basis into $[-1, -1]^T$ and j basis into $[-1, 1]^T$:



In the above example,

$$\det(A) = (-1)(1) - (-1)(-1) = \mathbf{-2}$$

so we know that we're stretching our basis by a factor of 2.

We can see that in our transformation above by simply inspecting the size of our new basis – but can you see why our determinant is now negative?

Notice that we're not able to simply stretch and rotate our original normal basis vector in a clock-wise or counter-clockwise direction in order to position it into its newly transformed basis. In the above instance, we could rotate the normal basis into its new position and stretch it, but the resulting grid wouldn't match up since we've now inverted the 2 basis vectors. We should be able to see this by noticing that our yellow / red colored squares are now inverted and face the opposite direction. To get them to match our transformation, we would have to 'flip' our 2-D projection over in 3-D space by 180 degrees first and then perform our stretch / rotation operations.

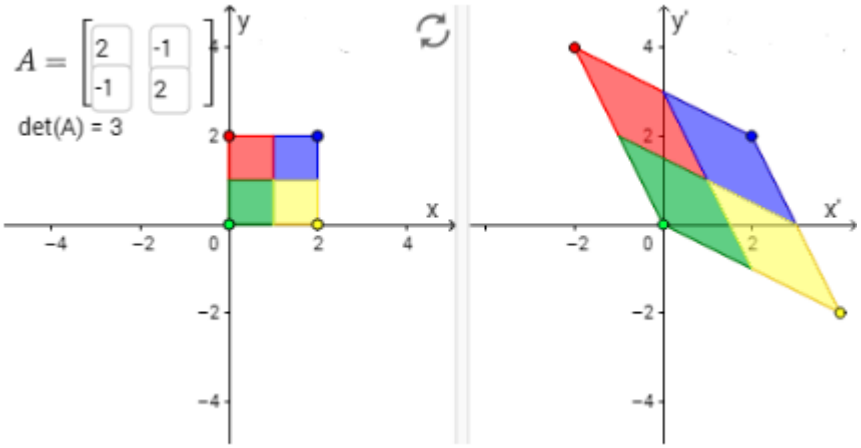
This same principle applies to vectors in higher dimensional spaces, including 3-D space. We can get a negative determinant in 3-D space if our resulting basis space is transformed in such a way as to not obey a right-hand rule. Although we won't go into the full details of specifying the full rules and how they apply to determinants here, it's a good point to note.

Let's visualize one more example showing how a matrix transformation works in 2-D space, just so we can get a good grasp on this important topic.

Let's take a linear transformation represented by the matrix:

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

We can visualize this transformation using the below diagram:



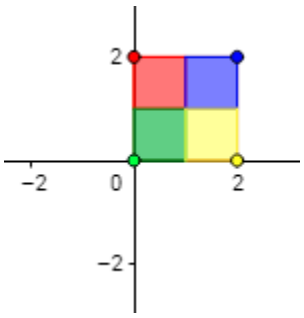
Once again, we can visualize linear transformations as changing our original basis which is pictured below into a new one represented by the column vectors of our matrix. We thus end up with a new spacial ‘grid’ which is stretched and represents a new vector space we can operate on diagrammed by our new basis. Since we don’t have to flip our resulting 2-D projection in 3-D space in order to get our new basis in the example above, we end up with a positive determinant. The 3 tells us that we’re stretching our original grid / basis by a factor of 3.

Common Linear Transforms

Identity Matrix

The identity matrix is a matrix that, when multiplied by anything (either a vector or a matrix) produces the thing it was multiplied by.

$$I = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$$

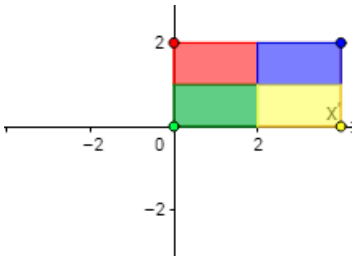


Scale Matrix

A scale matrix may either be a uniform scale (scale by the same amount in the x and y directions) or a non-uniform scale (scale by different amounts in each direction). The matrix for a scale transform is:

$$S = \begin{bmatrix} s_x & 0 \\ 0 & s_y \end{bmatrix}$$

Let's visualize a non-uniform scale matrix which scales x by 2:

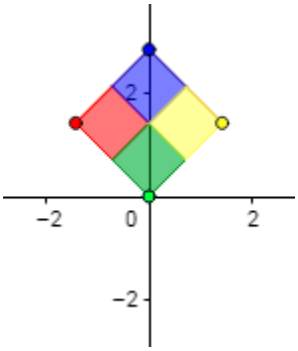


Rotation Matrix

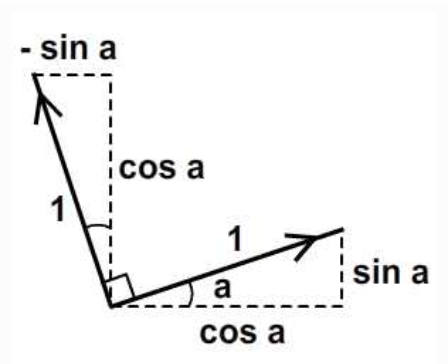
A rotation matrix transforms a vector by rotating it counter-clockwise about the origin. If the angle for rotation is α, then the matrix for a rotation transform is:

$$R_\alpha = \begin{bmatrix} \cos(\alpha) & -\sin(\alpha) \\ \sin(\alpha) & \cos(\alpha) \end{bmatrix}$$

Let's visualize a rotation matrix which rotates our basis vector by 45 degrees:



We can get an intuitive feel of this by taking a look at the diagram visualized below, and seeing how our angles are translated into a newly rotated basis, where x becomes (cos a, sin a) and y becomes (-sin a, cos a):

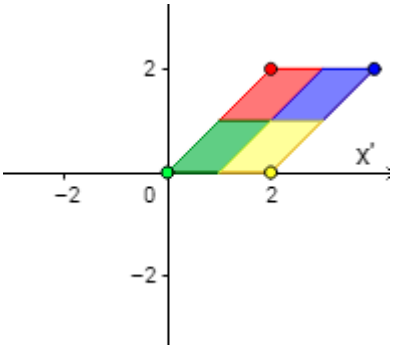


Shear Matrix

A shear matrix transforms a vector by "pulling" it along the either x-axis or the y-axis. A shear has a factor k which determines how much the image is pulled. The shear matrix is given by:

$$S_x = \begin{bmatrix} 1 & k \\ 0 & 1 \end{bmatrix} \quad \text{and} \quad S_y = \begin{bmatrix} 1 & 0 \\ k & 1 \end{bmatrix}$$

An example of a matrix sheared along the x axis (with k = 1) is shown below:

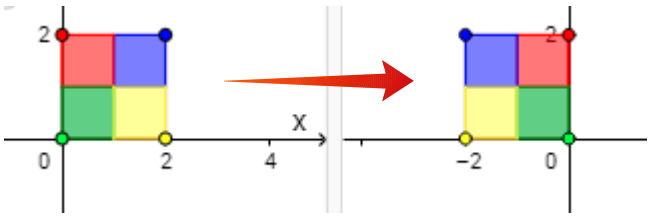


Reflection Matrix

A reflection matrix "flips" a vector across a line. To reflect about either the x-axis or the y-axis, we use the matrices

$$F_x = \begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix} \quad \text{and} \quad F_y = \begin{bmatrix} -1 & 0 \\ 0 & 1 \end{bmatrix}$$

An example showing a matrix reflected about the y-axis is provided below:



Cross Product

Earlier on, we showed the we can visualize determinants as being the area ‘stretch factor’ we get by transforming the basis vectors (i: [1, 0] ^{T} and j: [0, 1] ^{T}) into new basis vectors defined by our matrix.

The cross product is similar to the determinant, but instead of obtaining a ‘stretch factor,’ we obtain a new vector which is scaled according to this stretch factor and points in the direction perpendicular to our 2 vectors.

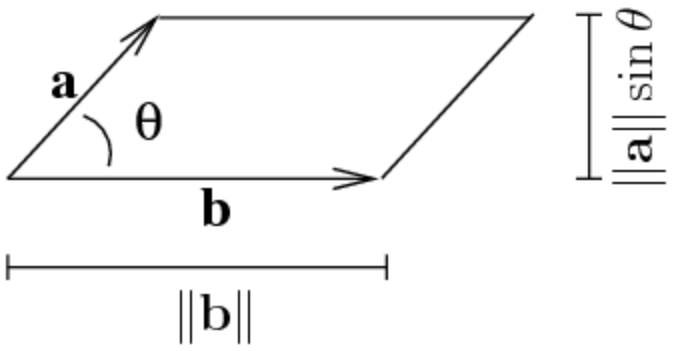
To put it precisely, if a and b are two vectors, then their cross product, written as a×b meets the following three requirements:

1. $a \times b$ is a vector that is perpendicular to both a and b.
2. The magnitude (or length) of the vector $a \times b$, written as $\|a \times b\|$, is the area of the parallelogram spanned by a and b and equates to our determinant.
3. The direction of $a \times b$ is determined by the right-hand rule. (This means that if we curl the fingers of the right hand from a to b, then the thumb points in the direction of $a \times b$.)

The below figure illustrates how, using trigonometry, we can calculate that the area of the parallelogram spanned by a and b is:

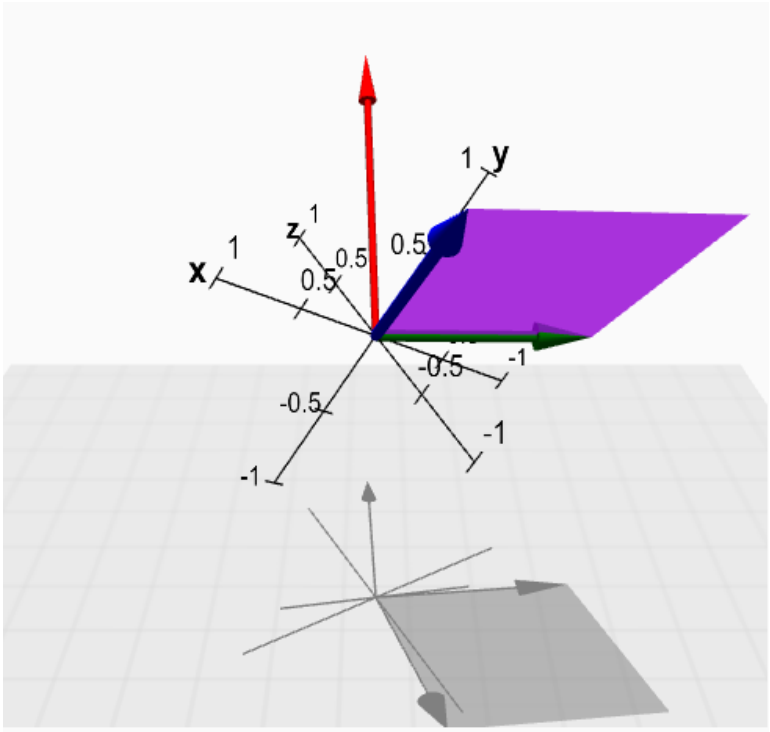
$$\|a\| \|b\| \sin\theta,$$

where θ is the angle between a and b. The figure shows the parallelogram as having a base of length $\|b\|$ and perpendicular height $\|a\| \sin\theta$.



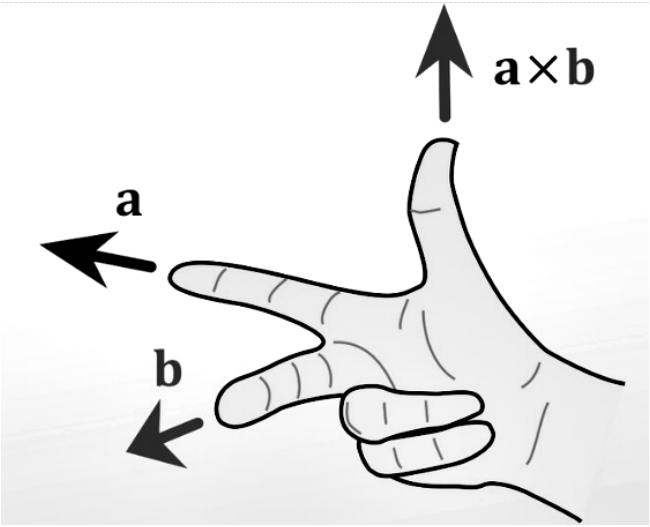
Let's try to visualize this.

In the figure below, we see that the cross product (red arrow) is perpendicular to the parallelogram formed by our 2 vectors:

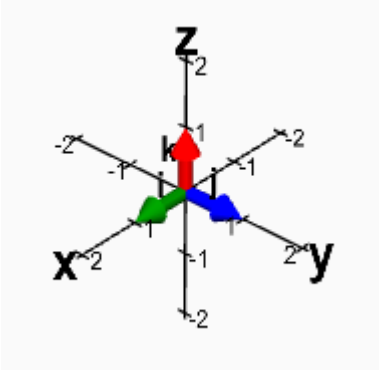


The length of our cross product vector (red arrow) will be equal to the area of the parallelogram shown, which will be equal to the determinant of our 2 vectors (a and b).

We can obtain the direction of our resulting vector by using the right hand rule. We see the rule demonstrated in the figure below:



Let’s show a simpler example using our standard basis vectors. Let i, j, and k be the standard unit vectors pointing in the x, y, and z planes respectively, and which we show in the figure below:



The parallelogram spanned by any two of these standard unit vectors is a unit square, which has area one. Hence, by the geometric definition, the cross product must be a unit vector. Since the cross product must be perpendicular to the two unit vectors, it must be equal to the other unit vector or the opposite of that unit vector. Looking at the above graph, we can use the right-hand rule to show that the cross products of our unit vectors can equate to the definitions provided below:

$$i \times j = k$$

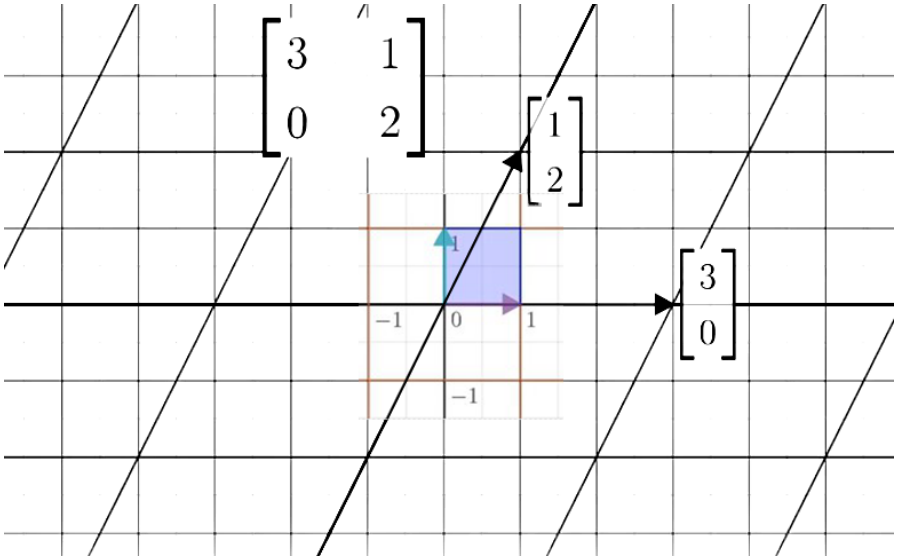
$$j \times k = i$$

$$k \times i = j$$

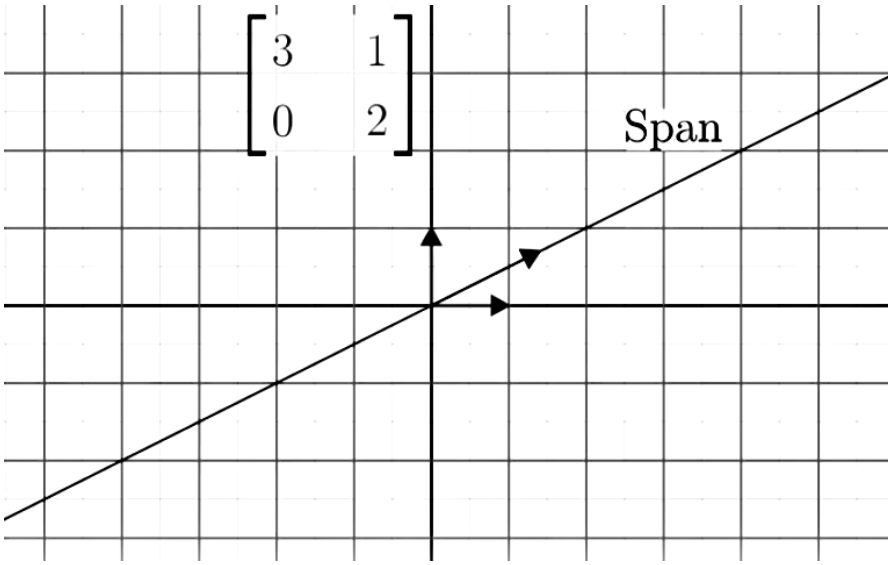
We won’t go any further, although we want to once again emphasize the fact that the cross product and determinant are similar concepts, with the cross product being an extension of the determinant showing a resulting vector (instead of a simple scalar) in a new dimension perpendicular to the plane spanned by our 2 cross product vectors and scaled by the determinant.

Eigenvalues and Eigenvectors

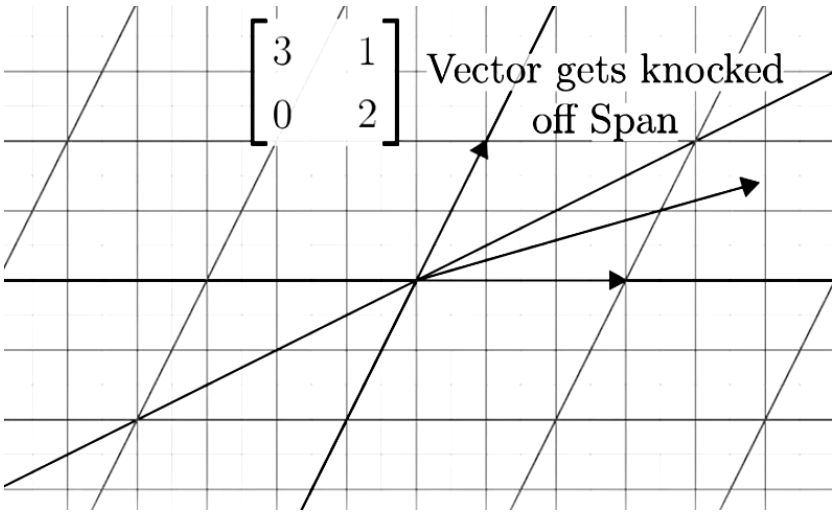
Let's consider some linear transformation in 2-dimensions. Let's say the we want to use the 2-D transformation shown below, which turns our i basis into $[3, 0]^T$ and our j basis into $[1, 2]^T$:



Focus in on what any particular transformation does to a vector in our basis space, as well the span of this vector and how it's transformed in our new basis. Let's take an example vector shown below:

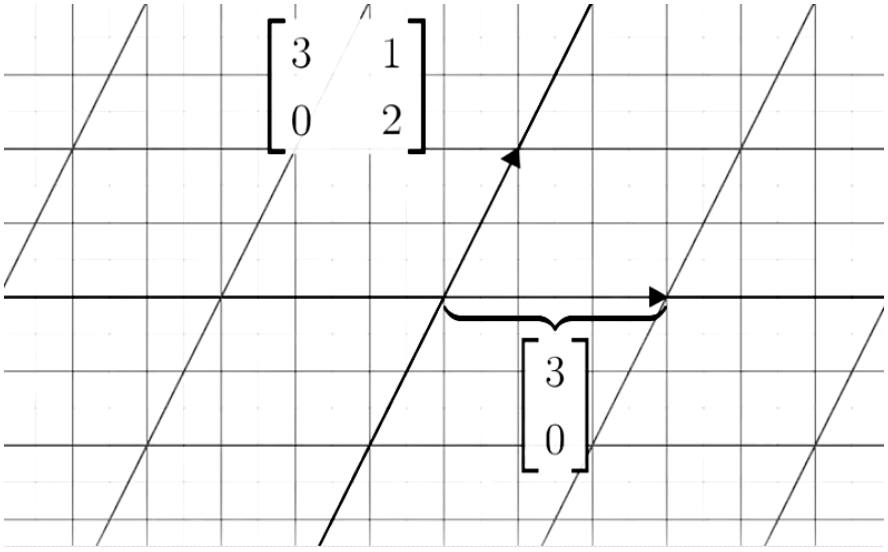


Most vectors from our old basis space (where i = [1, 0] ^{T} and j = [0, 1] ^{T}) are going to get knocked off of their span during our transformation, including our example vector:

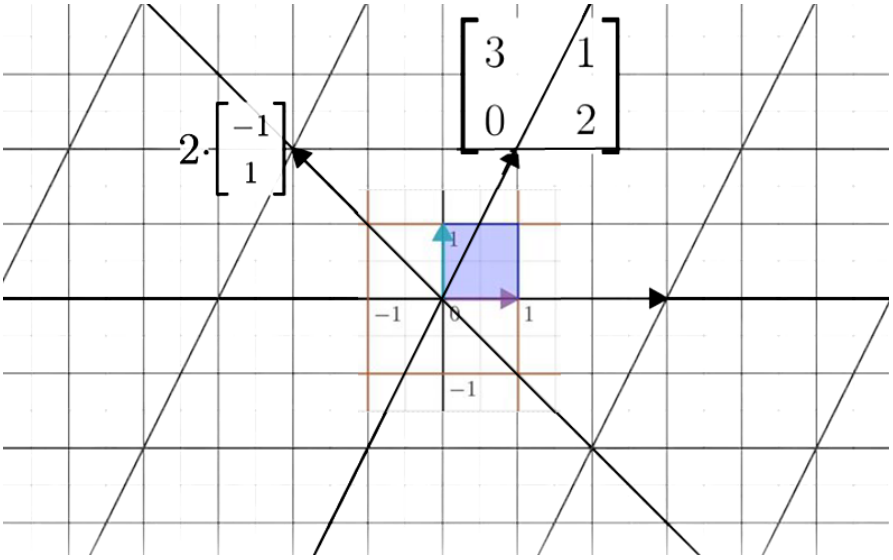


There exist a class of special vectors though which do remain on their span, meaning that the effect our basis transformation has on the vector is to squish or stretch it, like a scalar without altering its direction.

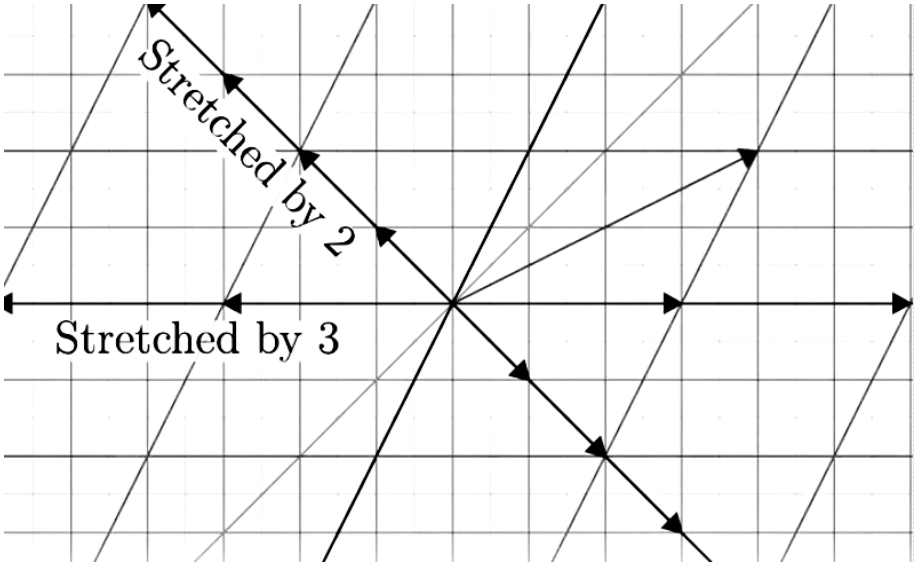
As an example, let's once again focus on our linear transformation above. Notice that our actual i basis vector remains in the same orientation even after we apply our linear transformation. The only difference between this vector and our old one is that we stretch our original Cartesian vector ($[1, 0]^{T}$) by a factor of 3, as shown below:



Another less obvious vector which remains the same is [-1, 1] ^{T}, which ends up getting stretched by a factor of 2 without having its span altered:

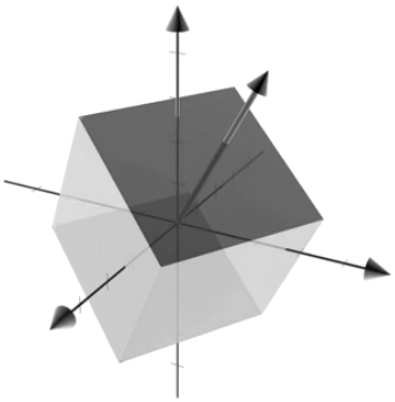


Any vectors which do not lie on the lines spanning the vectors above are going to get knocked off of their span after our linear transformation. Any vectors which span our lines above will continue to span the same lines – the only difference is that our vectors covering this space are going to get stretched by a factor of 3, or 2, depending on which line they're oriented on.

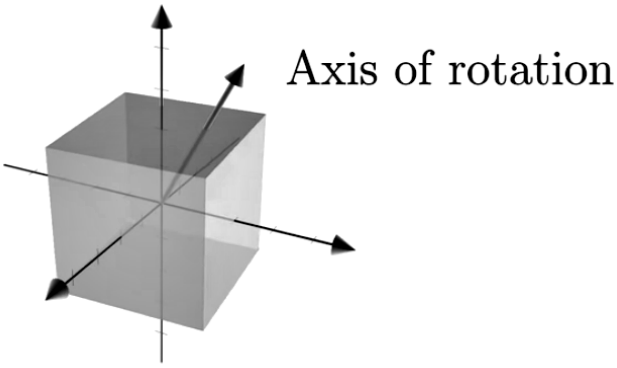


The above vectors are called the **eigenvectors** of our transformation, and each one of our eigenvectors has an associated **eigenvalue**, which is simply by the factor that it's 'stretched' or 'squished' by our linear transformation.

Let's go even further and consider what eigenvalues and eigenvectors represent in 3-d space. Let's try to picture a 3-d rotation:



If we can find the eigenvector of our rotation, what we actually find is our axis of rotation:

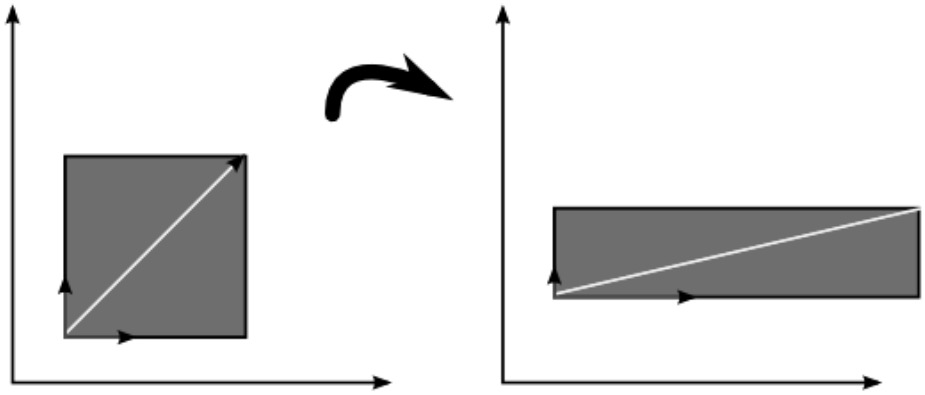


It's much easier to think of 3-d rotations in terms of the axis' of their rotations rather than having to consider their full 3 by 3 matrix transformations. Our eigenvalues and eigenvectors simplify our problems in 3-d space by quite a large factor.

Let's go through another simple example. Let's do one where we simply scale our normal basis vector, and we use the matrix shown below:

$$A = \begin{bmatrix} 2 & 0 \\ 0 & 0.5 \end{bmatrix}.$$

In our above case, we can visualize the transformation as scaling our horizontal direction (i basis) by a factor of 2 while our vertical direction (j basis) is scaled by a factor of 0.5:



In the above instance, we can intuitively see that we will get 2 eigenvectors, one having the same orientation as our i basis ($[1, 0]^T$) with an eigenvalue equal to 2, and the other which is oriented on our j basis ($[0, 1]^T$) with an eigenvalue equal to 0.5.

Rather than using our geometric intuition, it helps to define what our terms mean more formally, just so we can find the eigenvectors and eigenvalues in instances where we may not be able to form a visual picture of how our basis space is 'warped' by our matrix. We can say that the eigenvector v of a matrix A is the vector for which the following holds:

$$A\vec{v} = \lambda\vec{v}$$

We can re-write this equation as follows:

$$\begin{aligned} A\vec{v} - \lambda\vec{v} &= 0 \\ \Rightarrow \vec{v}(A - \lambda I) &= 0, \end{aligned}$$

where I is the identity matrix with the same dimensions as A.

To find the eigenvectors of A, we simply have to solve the following equation:

$$Det(A - \lambda I) = 0.$$

Example:

Let's find the eigenvalues and eigenvectors of matrix A defined below:

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

To determine the eigenvalues for this example, we simply use our determinant equation and plug our matrix values in to get:

$$Det \begin{pmatrix} 2-\lambda & 3 \\ 2 & 1-\lambda \end{pmatrix} = 0.$$

Calculating the determinant gives:

$$\begin{aligned} &(2-\lambda)(1-\lambda) - 6 = 0 \\ \Rightarrow &2 - 2\lambda - \lambda - \lambda^2 - 6 = 0 \\ \Rightarrow &\lambda^2 - 3\lambda - 4 = 0. \end{aligned}$$

From the above, we can see that the solution to our equation is (- 4)(+ 1) = 0 which gives us eigenvalues of λ = 4 and λ = -1.

Let's use these values to find our eigenvectors.

To find the solution for λ = -1, we plug in our value into the equation below:

$$\begin{bmatrix} 2 & 3 \\ 2 & 1 \end{bmatrix} \begin{bmatrix} x_{11} \\ x_{12} \end{bmatrix} = -1 \begin{bmatrix} x_{11} \\ x_{12} \end{bmatrix}.$$

Since this is simply the matrix notation for a system of equations, we can write it in its equivalent form:

$$\begin{cases} 2x_{11} + 3x_{12} = -x_{11} \\ 2x_{11} + x_{12} = -x_{12} \end{cases}$$

and solve the first equation as a function of x_{12} , resulting in:

$$x_{11} = -x_{12}.$$

For this example, we arbitrarily choose x_{12} = 1, such that x_{11} = -1. Therefore, the eigenvector that corresponds to eigenvalue λ = -1 is:

$$\vec{v_1} = \begin{bmatrix} -1 \\ 1 \end{bmatrix}.$$

To calculate our 2nd eigenvector, we go through the same process we went to get our first eigenvector. We first substitute eigenvalue λ = 4 into equation below yielding:

$$\begin{bmatrix} 2 & 3 \\ 2 & 1 \end{bmatrix} \begin{bmatrix} x_{21} \\ x_{22} \end{bmatrix} = 4 * \begin{bmatrix} x_{21} \\ x_{22} \end{bmatrix}.$$

Written as a system of equations, this is equivalent to:

$$\begin{cases} 2x_{21} + 3x_{22} = 4x_{21} \\ 2x_{21} + x_{22} = 4x_{22} \end{cases}$$

Solving this results in:

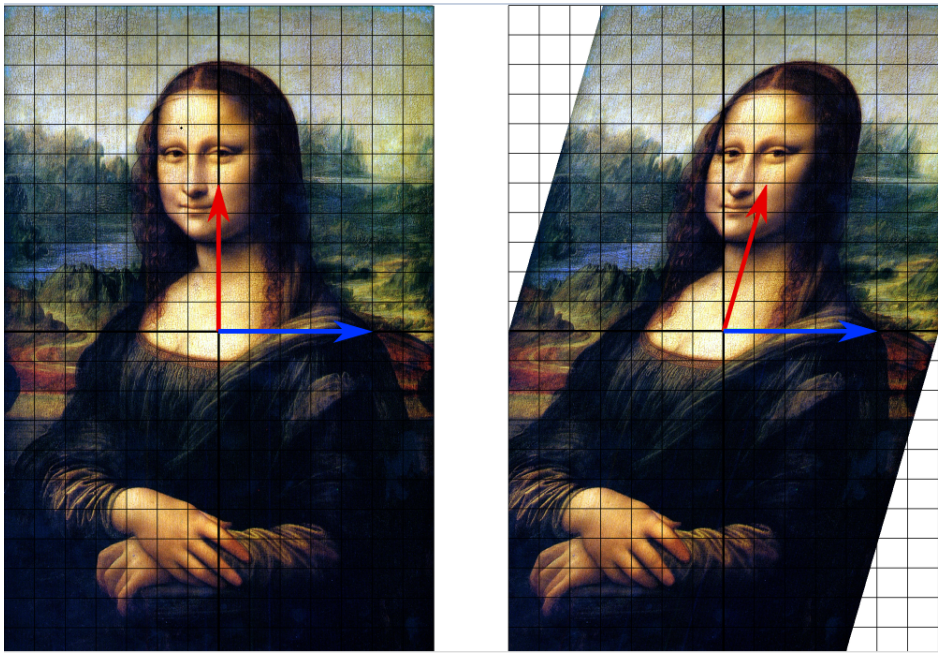
$$x_{22} = \frac{3}{2} x_{21}$$

We then arbitrarily choose x_{21} = 2, and find x_{22} = 3. Therefore, the eigenvector that corresponds to eigenvalue λ = 4 is:

$$\vec{v}_2 = \begin{bmatrix} 3 \\ 2 \end{bmatrix}.$$

Why are eigenvectors / eigenvalues important?

Eigenvectors / eigenvalues make understanding linear transformations easy. As we already explained, eigenvectors show they're the directions along which a linear transformation acts simply by stretching or compressing our basis vectors, and eigenvalues give us the factors by which this compression occurs. In the below image for example, our blue vector is an eigenvector while the red vector is not, since its direction is changed by our transform:



Eigenvalues characterize important properties of linear transformations, such as whether a system of linear equations has a unique solution or not. In many applications eigenvalues also describe physical properties of a mathematical model. They're used in a lot of important applications, including principal components analysis (PCA), dimensionality reduction, physics, market risk analysis, and even PageRank which Google uses to rank search results.

More Vector / Matrix Terminology

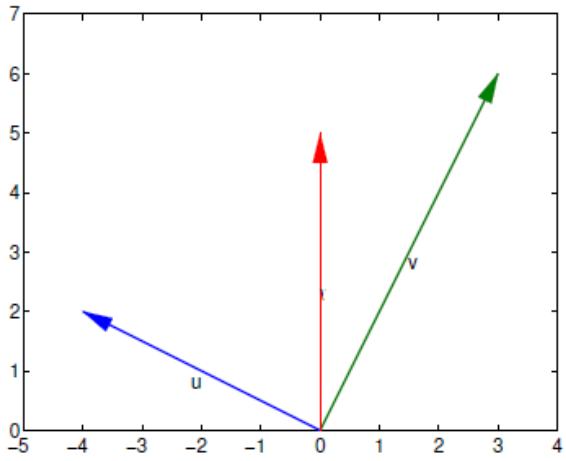
Before we dive into the full details of how vectors / matrices are used, let's review a bit of terminology. We're going to limit our coverage of linear algebra to its essential points, so the overview below will only cover the base essentials without diving into the details. To get a full overview of the topics, I recommend that the reader review the resources listed at the end of the document.

Vector Orthogonality

Two vectors v and u are said to be orthogonal to each other if:

$$v^T u = 0$$

In 2-D, it's easy to see what this means (our vectors are perpendicular to each other):



Vector Orthonormality

Two vectors v and w are said to be orthonormal to each other if they are orthogonal to each other and each has a length of 1 (unit length):

$$v^T w = 0 \quad \text{and} \quad \|v\| = \|w\| = 1$$

Let's take an example below and show that our vectors are orthonormal:

$$u = \frac{1}{\sqrt{5}} \begin{pmatrix} -2 \\ 1 \end{pmatrix} \quad \text{and} \quad v = \frac{1}{\sqrt{5}} \begin{pmatrix} 1 \\ 2 \end{pmatrix}$$

We first verify that they are orthogonal (perpendicular) to each other through simple vector multiplication:

$$u^{\mathrm{T}} v = \frac{1}{\sqrt{5}} \begin{pmatrix} -2 \\ 1 \end{pmatrix}^{\mathrm{T}} \frac{1}{\sqrt{5}} \begin{pmatrix} 1 \\ 2 \end{pmatrix} = \frac{1}{5} \begin{pmatrix} -2 \\ 1 \end{pmatrix}^{\mathrm{T}} \begin{pmatrix} 1 \\ 2 \end{pmatrix} = 0$$

We then verify that our vectors have a length of 1 (unit length) and we are done:

$$\|u\| = \sqrt{\frac{1}{5}((-2)^2 + 1^2)} = 1$$
$$\|v\| = \sqrt{\frac{1}{5}((1)^2 + 2^2)} = 1$$

Matrix Orthogonality

An orthogonal matrix is a square matrix, for which every column is a unit vector, and every pair of columns is orthogonal. This means that the transpose of the matrix multiplied by itself gives the identity matrix: $O^{T}O = I$. This means that applying an orthogonal matrix to a vector does not change its length. Similarly, applying an orthogonal matrix to two vectors does not change the angle between them.

Once again, a matrix A is orthogonal if $AA^{T} = A^{T}A = I$.

As an example, if our matrix A is defined below:

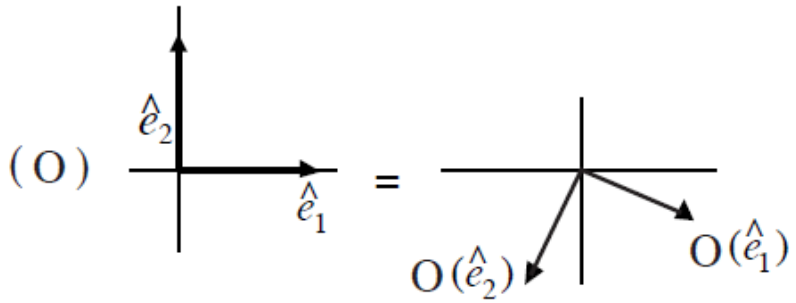
$$A = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 3/5 & -4/5 \\ 0 & 4/5 & 3/5 \end{bmatrix}$$

we can say that it is orthogonal since:

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

Since the columns of a matrix describe the response of the system to the standard basis, an orthogonal matrix maps the standard basis onto a new set of N orthogonal axes, which form an alternative basis for the space.

Putting this all together, we can think of orthogonal matrices as performing a generalized rotation: a rigid physical rotation of the space and possibly negation of some axes. Note that the product of two orthogonal matrices is also orthogonal.



$$O = \begin{bmatrix} \cos\theta & \sin\theta \\ \sin\theta & \cos\theta \end{bmatrix}$$

Diagonal Matrix

A diagonal matrix A is a matrix where all the entries a_{ij} are 0 when i is not equal to j. In other words, the only non-zero values run along the main dialog from the upper left corner to the lower right corner:

$$A = \begin{bmatrix} a_{11} & 0 & \dots & 0 \\ 0 & a_{22} & & 0 \\ . & & . & . \\ 0 & \dots & & a_{mm} \end{bmatrix}$$

Examples of diagonal matrices A and B are provided below:

$$A = \begin{pmatrix} 1 & 0 \\ 0 & 3 \end{pmatrix}, \quad B = \begin{pmatrix} 2 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 7 \end{pmatrix}$$

Projection

In linear algebra and functional analysis, a projection is a linear transformation P from a vector space to itself such that $P^2 = P$. In other words, the projection leaves its image unchanged. Though abstract, this definition of "projection" formalizes and generalizes the idea of graphical projection.

Using projection, we take a higher dimensional object as input and map it onto a lower dimensional one.

A real world example of this can be seen in computer games. They are almost all generated in a 3D environment where they interact with polygons and computational operations simulating a 3-D environment. However, our monitors are only 2-dimensional, so all of that mathematical stuff must be projected onto a 2-D surface to be displayed.

Simple Example

For example, the function which maps the point (x, y, z) in three-dimensional space R^3 to the point (x, y, 0) is a projection onto the x-y plane. This function is represented by the matrix:

$$P = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix}.$$

The action of this matrix on an arbitrary vector is:

$$P \begin{pmatrix} x \\ y \\ z \end{pmatrix} = \begin{pmatrix} x \\ y \\ 0 \end{pmatrix}.$$

Solving Linear Equations

Earlier on, we talked about linear algebra being about linear functions and linear transformations. So why the need for vectors and matrices? Why did we spend so much time describing their properties and what they mean?

Simply put: all linear equations can be translated in terms of vectors / matrix operations.

We can think of vectors as representing one set of linear relationships, and a matrix as representing a function which maps multiple relationships using a row by column representations, whereby each row can be thought of as modeling the relationships present in our equations, and each column can be thought of as the variable inter-relationships between the set of ALL equations.

The fundamental problem of linear algebra really is to solve N linear equations with N unknowns; for example:

$$2x - y = 0$$

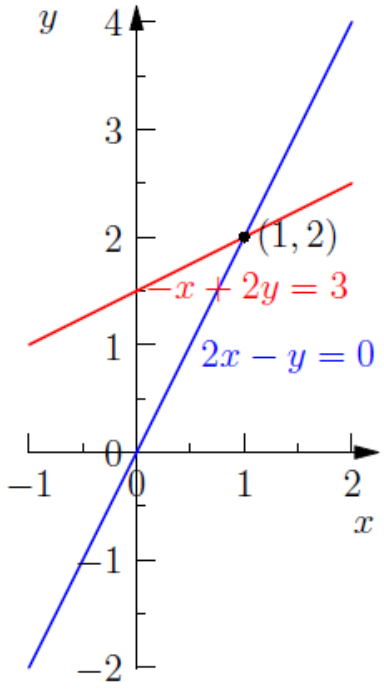
$$-x + 2y = 3$$

The system above is two dimensional (N = 2). By adding a third variable z, we could expand it to three dimensions.

Linear algebra simply lets us represent a set of equations / functions in geometric terms by translating our mathematical relationships using vectors / matrices. This is usually done through a column formulation representing our relationships, but prior to describing this, let's take a look at the regular row picture.

Row Picture

To solve our equation, we can plot the points that satisfy each one and find the intersection. This intersection represents the solution to the system of equations. Looking at the figure below, we see that the solution to this system of equations is x = 1, y = 2:



We plug this into the original system of equations to check our work:

$$2 \cdot 1 - 2 = 0$$
$$-1 + 2 \cdot 2 = 3.$$

The solution in a 3 dimensional system of equations is the common point of intersection of 3 planes (if there is one), and we can expand this view into any amount of dimensions that we want, although visualizing this is harder to grasp, so we'll stick to using 2-dimensional spaces for now.

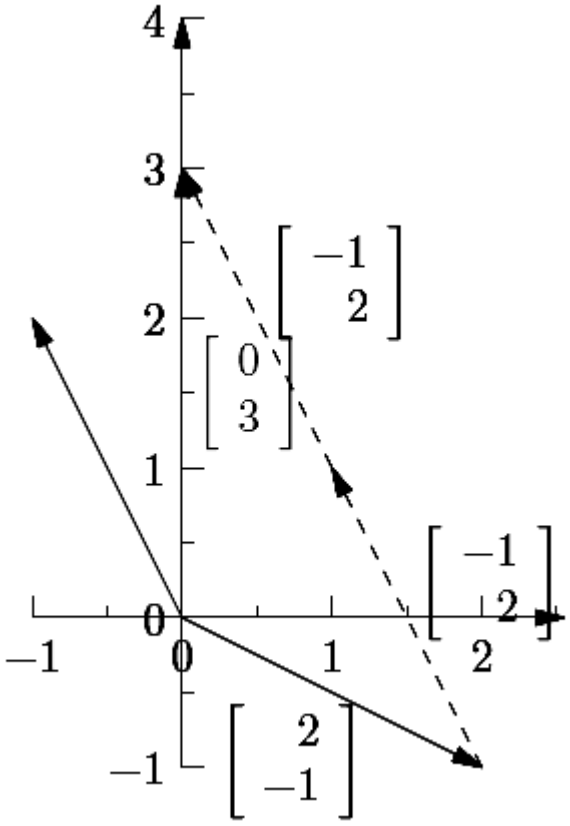
Column Picture

In the column picture we rewrite the system of linear equations as a single equation by turning the coefficients in the columns of the system into vectors:

$$x \begin{bmatrix} 2 \\ -1 \end{bmatrix} + y \begin{bmatrix} -1 \\ 2 \end{bmatrix} = \begin{bmatrix} 0 \\ 3 \end{bmatrix}.$$

Geometrically, we can picture this as wanting to find numbers x and y so that x copies of vector [2, -1] ^{T} are added to y copies of vector [-1, 2] ^{T} to equal to vector [0, 3] ^{T}.

We can see this concept being applied by looking at the figure below:



We can get an intuitive sense how the above mapping works by once again thinking of matrices as functions which map linear relations from one space into another.

In regards to our functions, we know that we have a relation mapping 2x in our 1st equation to -1x in our 2nd equation. Let's take a look at our equations again:

$$2x - y = 0$$

$$-x + 2y = 3$$

We know that for every value of x which we plug into our first equation, our 2nd equation must have the same value divided by 2 and opposite magnitude. As an example, for the value of x = 1, our first equation x coordinate becomes 2 while our 2nd equation x coordinate becomes -1. For x = 2, we get a value of 4 in our first equation and a value of -2 in our 2nd equation, and so on.

As you can see, we have a linear relationship in regards to these variables, and we can represent this linear relationship using a new basis vector. In this instance, we're simply

creating a new basis for i which maps our 1st equation input (2x) onto our x axis and our 2nd equation input (-1x) onto the y-axis. We can model this dependency by using a basis vector which warps our original Cartesian plane into one which has a new i basis vector of $[2, -1]^T$.

We make a similar analogy in regards to our y variable. We know that any input for y in our first equation scales it by -1, and that any input in our 2nd equation scales it by 2, so we choose to map this relationship by using a new basis vector for j which maps $[0, 1]^T$ onto $[-1, 2]^T$.

We can imagine then that we're transforming our problem from one which can be viewed in terms of equations into one which maps linear relationships onto basis vectors representing these relationships, and then using these basis vectors to compose a solution vector or vectors which map onto a point containing our solution using this new basis space.

Think about this closely, since understanding the intuition of this is very important. In linear algebra, we're simply translating equations and linear relationships into a form of geometric construction which allows us to translate these linear relationships onto a space which keeps them intact. To find the equation solutions, we use this new basis in this new space and we try to construct our solution point / vector within this space in order to obtain our solution (assuming one exists).

In our above example, we translated our x variable relationships to a new i basis, and our y variable relationships to a new j basis to construct a new basis space. We then used these vectors in order to find a mapping onto our solution point which was represented by $[0, 3]^T$.

The above generality doesn't' just extend to 2-dimensional spaces. We can make the same generalization and construction in 3 or more dimensions. In 3 dimensions, the column picture would require us to find new basis vectors mapping the variable relationships onto new basis vector set in 3 dimensions. We would thus have 1 basis vector which represents the x variable inter-relationships, one vector which models the y-variable inter-relationships, and a 3rd one modeling variable z. We could then use the same methodology we used above in trying to use these new basis vectors in order to obtain a mapping to a point in this space to a point containing our solution, and this would enable us to find a solution to any linear problem we want!

The Matrix Cookbook Example

Let's formulate a cookbook using matrices to show how versatile they can be. Once again, we can use matrices to model and system of linear equations or relationships. Below, we'll write a small cookbook using only numbers in a matrix to show you an example of applied linear algebra.

To make things simple, we'll use a finite number of ingredients, say sugar, flour, salt, milk and water, and we'll use three different dough recipes:

	Sugar (cup)	**Flour** (cup)	**Salt** (tsp)	**Milk** (cup)	**Water** (cup)
Pizza	0	2	1	0	1
Cake	1	1	0	0.5	0
Bagel	1	1	2	0.25	0.25

First, we need to determine the input and output vectors and which operation the cookbook matrix M will represent. Our input shall be the answer to the question "what types of dough do you want to make?".

Our output will contain a list of ingredients showing what we need to use in order to make our dough recipes.

Input:

So, as our input, we'll have 3 separate variables which we can specify, so our basis here will be composed in 3-dimensions. Each dough is independent of each other, so we will assign each dough to a basis vector:

$$\overrightarrow{pizza} = \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix}, \quad \overrightarrow{cake} = \begin{bmatrix} 0 \\ 1 \\ 0 \end{bmatrix}, \quad \overrightarrow{bagel} = \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix}$$

Output:

We have a list of 5 ingredients, which are independent of each other, and so we aren't allowed to substitute in place of another. For our output, we'll thus use a basis of 5 vectors, and we'll assign:

$$\overrightarrow{sugar} = \begin{bmatrix} 1 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix}, \quad \overrightarrow{flour} = \begin{bmatrix} 0 \\ 1 \\ 0 \\ 0 \\ 0 \end{bmatrix}, \quad \overrightarrow{salt} = \begin{bmatrix} 0 \\ 0 \\ 1 \\ 0 \\ 0 \end{bmatrix}, \quad \overrightarrow{milk} = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 1 \\ 0 \end{bmatrix}, \quad \overrightarrow{water} = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 1 \end{bmatrix}$$

From the recipes, we know the list of ingredients we'll need for each one of our doughs, so we can compose vectors mapping the list of each dough:

$$M\overrightarrow{pizza} = \begin{bmatrix} 0 \\ 2 \\ 1 \\ 0 \\ 1 \end{bmatrix}, \quad M\overrightarrow{cake} = \begin{bmatrix} 1 \\ 1 \\ 0 \\ 0.5 \\ 0 \end{bmatrix}, \quad M\overrightarrow{bagel} = \begin{bmatrix} 1 \\ 1 \\ 2 \\ 0.25 \\ 0.25 \end{bmatrix}$$

We'll use matrix M provided below to model our cookbook:

$$M = \begin{bmatrix} 0 & 1 & 1 \\ 2 & 1 & 1 \\ 1 & 0 & 2 \\ 0 & 0.5 & 0.25 \\ 1 & 0 & 0.25 \end{bmatrix}$$

As a rule of thumb, it's good to remember that we want to create an m by n matrix to model our linear relationships so that m (number of rows) equates to the number of output dimensions we're looking to show while n (number of columns) equates to the number of inputs / variables we're using.

In the above instance for example, we're looking to list the number of 3 types of dough we want to input while the output represents the combined ingredient list showing the total of each ingredient we'll need to use in order to create our recipes. We will thus have a resulting 5 by 3 matrix modeling our linear relationships which we'll use to create a map from one basis space into another (input space of dough → output space of ingredients).

Let's show an example. Suppose we want to create 2 pizzas, half a cake, and 2 bagels. All we have to do is plug these values into our 3-dimensional input vector / space, and

we can output our list of ingredients by doing simple matrix multiplication using our cookbook matrix:

$$\begin{bmatrix} 0 & 1 & 1 \\ 2 & 1 & 1 \\ 1 & 0 & 2 \\ 0 & 0.5 & 0.25 \\ 1 & 0 & 0.25 \end{bmatrix} \begin{bmatrix} 2 \\ 0.5 \\ 3 \end{bmatrix} = \begin{bmatrix} 3.5 \\ 7.5 \\ 7 \\ 1 \\ 2.75 \end{bmatrix}$$

We thus know that to create our different types of dough, we'll need 3.5 cups of sugar, 7.5 cups of flour, 7 tea spoons of salt, 1 cup of milk and 2.75 cups of water.

We can once again see that matrices make our 'cookbooks' and transformations easy to grasp. In the example above, we're mapping a 3 dimensional input space into a 5 dimensional list of ingredients.

Formulating our problem in terms of matrix notation and linear projections was very simple!

Gauss-Jordan Elimination

Suppose that rather than given a set of input variables and a cookbook we can use to obtain the output, we're instead given the cookbook with an output, and we have to find an input mapping to this output using our system of linear relationships and equations.

This is essentially what we have to do any time we are tasked with solving a system of linear equations.

Earlier on, we went through one example of mapping this system of linear equations onto our system of linear relations. Here, we'll go through another example whereby we'll demonstrate how we can transform our cookbook matrix into a notation which makes obtaining our input variable mapping to the given output easy to find. The algorithm / process used to arrive or transform our matrix cookbook is called Gauss-Jordan elimination.

Suppose we're asked to solve the following system of equations:

$$x + 2y = 5$$

$$3x + 9y = 21$$

Gauss–Jordan elimination is a systematic procedure for solving systems of equations based the following row operations:

1. Adding a multiple of one row to another row.
2. Swapping two rows.
3. Multiplying a row by a constant.

To illustrate the Gauss–Jordan elimination procedure, we'll show the sequence of row operations required to solve the system above.

We start by constructing an augmented matrix as follows:

$$\left[\begin{array}{cc|c} 1 & 2 & 5 \\ 3 & 9 & 21 \end{array}\right].$$

The first column in the augmented matrix corresponds to the coefficients of the variable x, the second column corresponds to the coefficients of y, and the third column contains the constants from the right-hand side.

The Gauss-Jordan elimination procedure consists of two phases. During the first phase, we proceed left-to-right by choosing a row with a leading one in the leftmost column (called a pivot) and systematically subtracting that row from all rows below it to get zeros below in the entire column. In the second phase, we start with the rightmost pivot and use it to eliminate all the numbers above it in the same column. Let's see this in action:

1. The first step is to use the pivot in the first column to eliminate the variable x in the second row. We do this by subtracting three times the first row from the second row, denoted by $R_2 = R_2 - 3R_1$,

$$\left[\begin{array}{cc|c} 1 & 2 & 5 \\ 0 & 3 & 6 \end{array}\right]$$

2. Next, we create a pivot in the second row using $R_2 = 1/3\ R_2$:

$$\left[\begin{array}{cc|c} 1 & 2 & 5 \\ 0 & 1 & 2 \end{array}\right].$$

3. We now start the backward phase and eliminate the second variable from the first row. We do this by subtracting two times the second row from the first row $R_1 = R_1 - 2R_2$:

$$\left[\begin{array}{cc|c} 1 & 0 & 1 \\ 0 & 1 & 2 \end{array}\right]$$

We say that the matrix is now in reduced row echelon form (RREF), which is its "simplest" form, and we can easily obtain the solutions mapping the output to our input:

$$x = 1, y = 2$$

Matrix Inverse

Let's say that instead of modeling our problem above in terms of mapping our input basis space to a new space containing our solution, we want to instead to do the inverse and map the output basis space back to our input space. Using this approach, we can map our set of outputs back to our input space and reveal which points from our output lead to our solution points directly.

Let's once again go through our example above and show how we can use this approach to solve our linear equations. We start by modeling our equations using the augmented matrix below:

$$\left[\begin{array}{cc|c} 1 & 2 & 5 \\ 3 & 9 & 21 \end{array}\right].$$

We need to find a matrix which maps our output points $[5, 21]^T$ onto our solution space. That way, we can figure out our solution through simple matrix multiplication.

We can find this by finding an inverse matrix – which is essentially a matrix which 'undoes' our original basis mapping from our regular space to a new space and performs an inverse mapping of our new space back to our original space.

How can we compute our inverse matrix?

Here, we'll use the same methodology we used above, which is Gauss-Jordan elimination. We start by creating an array containing the entries of the matrix A on the left side and the identity matrix on the right side:

$$\left[\begin{array}{cc|cc} 1 & 2 & 1 & 0 \\ 3 & 9 & 0 & 1 \end{array}\right]$$

Then, we perform the Gauss-Jordan elimination procedure on this array.

1. The first row operation is to subtract three times the first row from the second row: $R_2 = R_2 - 3R_1$. We obtain:

$$\left[\begin{array}{cc|cc} 1 & 2 & 1 & 0 \\ 0 & 3 & -3 & 1 \end{array}\right]$$

2. The second row operation is divide the second row by 3: $R_2 = 1/3\ R_2$

$$\left[\begin{array}{cc|cc} 1 & 2 & 1 & 0 \\ 0 & 1 & -1 & \frac{1}{3} \end{array}\right]$$

3. The third row operation is $R_1 = R_1 - 2R_2$

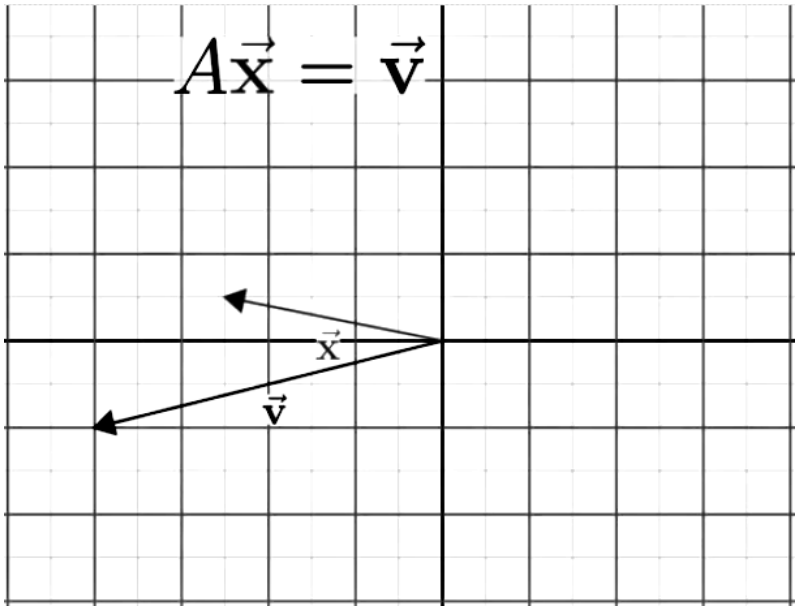
$$\left[\begin{array}{cc|cc} 1 & 0 & 3 & -\frac{2}{3} \\ 0 & 1 & -1 & \frac{1}{3} \end{array}\right]$$

The array is now in reduced row echelon form (RREF). The inverse matrix appears on the right side of the array.

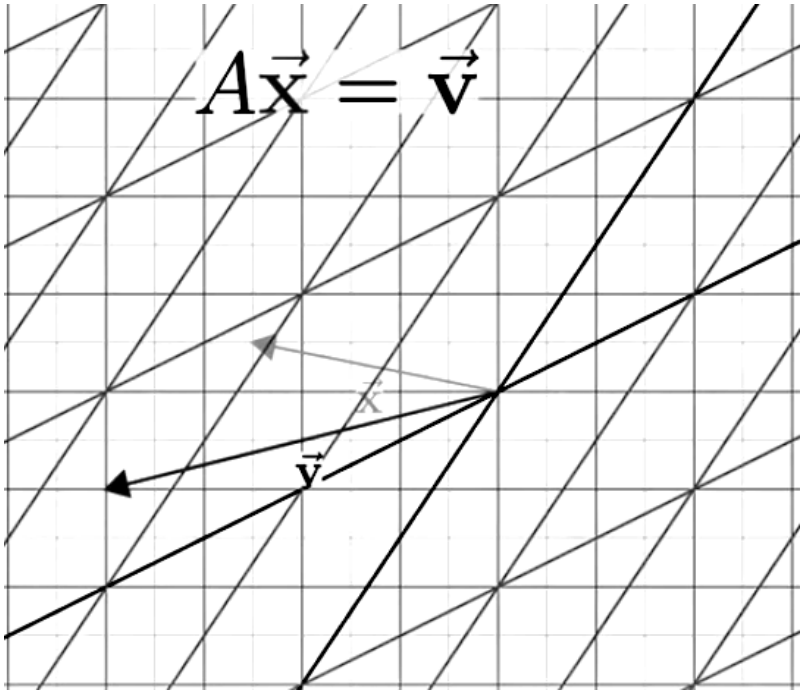
Observe that the sequence of row operations we used to solve the specific system of equations in Ax = b in the previous section are the same as the row operations we used in this section to find the inverse matrix, except that this time, instead of mapping our input space directly to a new output points / vector, we're instead mapping our original input space to a new 2×2 output space we can use to map any output points back to our inputs.

Let's go through this again, just so we can get a good intuitive of this concept, and which we previously went over before.

Any time we have a linear transformation represented by Ax = v and which we can use to model our system of linear equation, we can geometrically picture this system as the matrix A representing some linear transformation which maps vector x onto vector v (our solution):



We can furthermore think of A as a matrix which warps our original basis space ($[1, 0]^T$ and $[0, 1]^T$) onto a new one which 'warps' our input vector x onto v:



This warping of space represents our system of linear transformations originally represented in our system of linear equations.

Now, let's say that we want to map our output vector v back to our input vector x. How do we do this?

We do this by computing the inverse of A!! This allows us to ‘play back’ the transformation of our original input space to our output space. We can thus map any points / vectors present in our output space back onto our original basis vector / space easily.

What would happen if we were to play our original transformation A followed by our inverse transformation A^{-1}? Think about this closely – by applying A, all we are doing is transforming out input basis space to a new one representing our linear relationships and which we can use to map our input to our output. The A inverse undoes this operation, and maps this output basis back to our input basis, and allows us to map an output back to our input space. What are we left with?

We’re left with our original basis, so $A^{-1}A$ always equates to matrix which does nothing! By multiplying the 2, we’re always left with a matrix which we call the identity transformation / matrix which simply leaves all of our original basis vectors where they are:

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

Let’s go back to our original example.

Now that have the inverse of our matrix, we can map our solution to the basis containing our inputs through simple matrix multiplication:

$$\begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = A^{-1}\vec{b} = \begin{bmatrix} 3 & -\frac{2}{3} \\ -1 & \frac{1}{3} \end{bmatrix} \begin{bmatrix} 5 \\ 21 \end{bmatrix} = \begin{bmatrix} 1 \\ 2 \end{bmatrix}$$

We can once again observe that our solution equates to x = 1 and y = 2.

Once again, all we’re doing here is transforming our solution space back to our original input basis space so we can map our solution vector back to an input vector. The resulting vector contains the x and y values which represent our solution.

Linear Algebra Applications

Singular Value Decomposition

One application of the SVD is data compression. Consider some matrix A with rank five hundred; that is, the columns of this matrix span a 2000-dimensional space. Encoding this matrix on a computer is going to take quite a lot of memory! We might be interested in approximating this matrix with one of lower rank - how close can we get to this matrix if we only approximate it as a matrix with rank one hundred, so that we only have to store a hundred columns? What if we use a matrix of rank 20? Can we summarize all of the information in this very dense, 2000-rank matrix with only a rank twenty matrix?

SVD is essentially about dimensionality and reducing our dimensions. Dimensionality reduction is about converting data of very high dimensionality into data of much lower dimensionality such that each of the lower dimensions convey generalized information representing our higher dimensions. This is typically used in machine learning problems for extracting features in order to perform classification or regression tasks.

Here's a simple example: suppose we have a list of 200 movies and we have 1000 people for which we know whether they like or dislike each of the 200 movies. In this case, for each person, we have a binary vector of length 200 [position i is 0 if that person dislikes the ith movie, and 1 otherwise].

We can perform our machine learning task on these vectors directly, or instead, we could decide to compress the movie information from 200 into 5 genres of movies. Using the data we already have, we figure out whether the person likes or dislikes the entire genre and in this way, we reduce our data from a vector of size 200 into a vector of size 5. The vector of length 5 can be thought of as a good representative of the vector of length 200 because most people might like movies only belonging to a specific genre.

The downside of using the above approach is that we’re not going to have an exact mapping of our user preferences. We might have instances where a person hates all movies of a genre except one – on the other hand, the simplicity of modeling our problem space using this reduced dimensionality lets us use less space and lets us compute our results faster.

We can do the above reduction by performing matrix factorization. Matrix factorization takes a matrix and breaks it down into some product of smaller matrices (its factors).

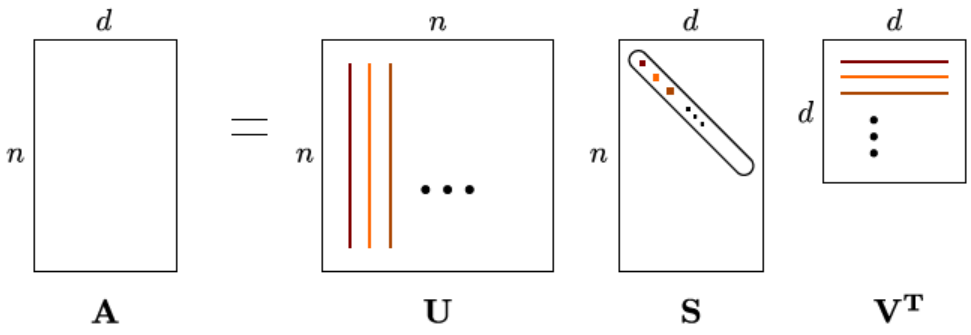
It's very similar to how we did factoring in elementary school. We took a big number like 12 and broke it down into its factors (1, 12), (2, 6), (3, 4), where each pair yields the number 12 when multiplied together. Factorizing matrices is exactly the same but since we are breaking something like a matrix that is inherently more complex, there are many, many ways to perform this break down.

How you factor a matrix basically comes down to what constraints you put on the factors (the matrices that when multiplied together form the original). Do you want just 2 factors? Do you want more? Do you want them to have particular characteristics like orthogonality? Do you want their eigenvectors to have any specific things?

Our high-level plan for computing an approximation of a matrix A will be:

1. Express A as a list of its ingredients, ordered by importance
2. Keep only the k most important ingredients.

The non-trivial step 1 is made easy by the singular value decomposition (SVD), and it's essentially a standard form for factorizing a matrix.



The result essentially decomposes the matrix A into a product of three matrices:

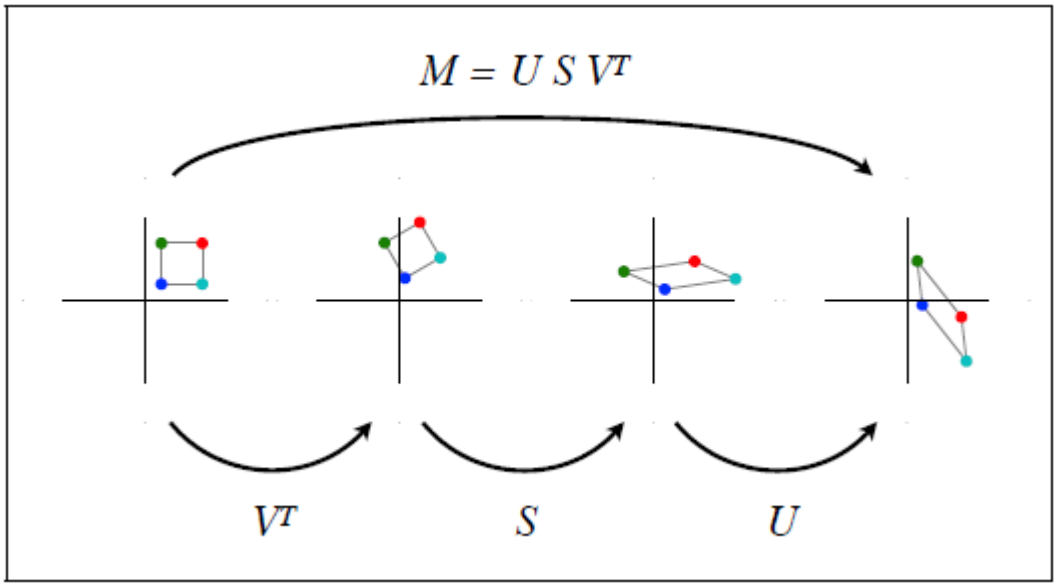
$$A = USV^T$$

such that U and V are orthogonal and S is diagonal with positive entries. The matrix S always has the same dimensions as M, and the diagonal elements of S are called the singular values.

The above result can be achieved with any matrix A, and the advantage of this decomposition is that it describes the action of A in terms of easily understood pieces:

- A (generalized) rotation (V^T)
- Scaling of the axes (S)

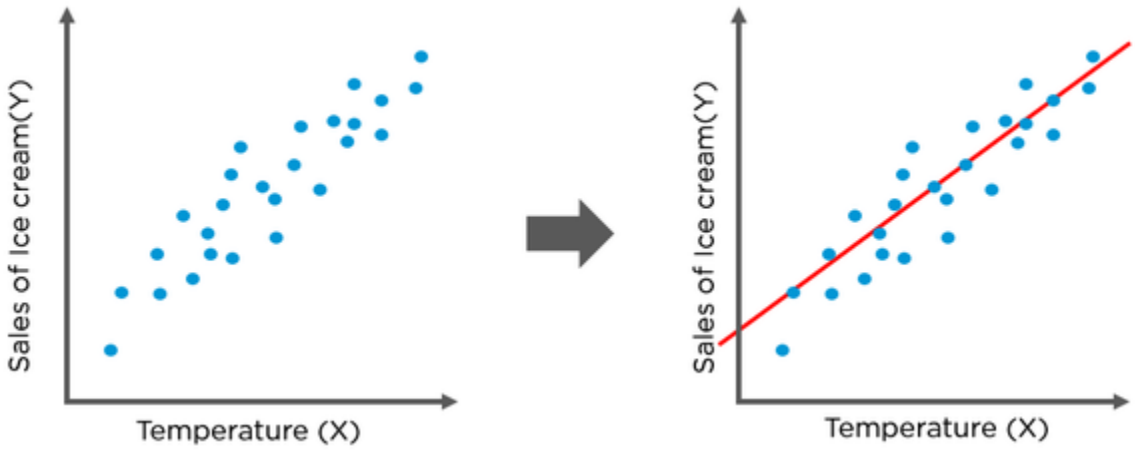
- And another rotation (U)



Given this decomposition, we can directly answer many important questions regarding the matrix and use it to compress our transformations / data.

Least Squares / Linear Regression

Suppose you want to predict the amount of air conditioner sales you would make based on the temperature of the day. You can plot a regression line that passes through all the data points:



This regression line in red is the best fit line that predict the sale of ice creams to best possible accuracy. One of the methods to draw this line is using the least squares method, and we can use linear algebra to find this line using the methodology shown below.

More formally: let the data consist of a set of n points given by (x_1, y_1), (x_2, y_2),... , (x_n, y_n). Here, we assume that the x values are exact, and the y values are noisy. We further assume that the best fit line to the data takes the form $y = B_0 + B_1x$. Although we know that the line will not go through all of the data points, we can still write down the equations as if it does. We have:

$$y_1 = B_0 + B_1x_1 \ , \ \ y_2 = B_0 + B_1x_1 \ , \ \dots \ , \ \ y_n = B_0 + B_1x_1$$

These equations constitute a system of n equations in the two unknowns B_0 and B_1. The corresponding matrix equation is given by:

$$\begin{pmatrix} 1 & x_1 \\ 1 & x_2 \\ \vdots & \vdots \\ 1 & x_n \end{pmatrix} \begin{pmatrix} \beta_0 \\ \beta_1 \end{pmatrix} = \begin{pmatrix} y_1 \\ y_2 \\ \vdots \\ y_n \end{pmatrix}.$$

This is an over-determined system of equations with no solution. The problem of least squares is to find the best solution, as we already discussed when showing our air conditioner sales example.

How do we do this?

We can generalize the problem as follows:

Suppose we are given a matrix equation, Ax = b, that has no solution because b is not in the column space of A. The least-squares solutions of Ax = b are the solutions of the matrix equation:

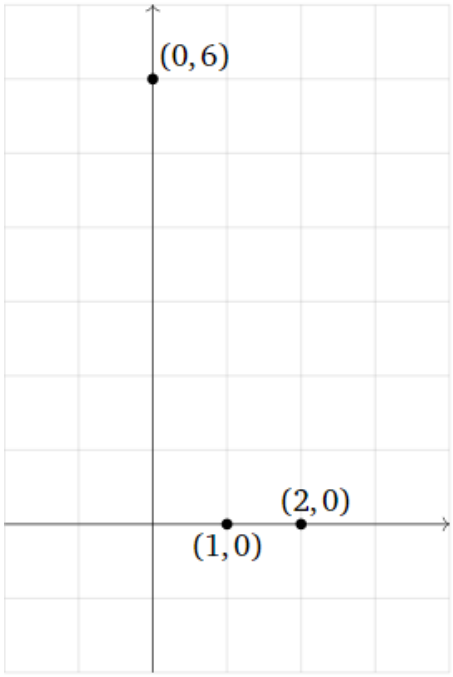
$$A^TAx = A^Tb$$

Example:

Suppose that we have measured three data points:

$$(0, 6), \quad (1, 0), \quad (2, 0)$$

and that our model for these data asserts that the points should lie on a line. Of course, these three points do not actually lie on a single line, but this could be due to errors in our measurement.



How do we predict which line they are supposed to lie on?

We first model our problem using the methodology we discussed in the first section, which is, we try to model in terms of a linear line / equation using the formula shown below:

$$y = Mx + B$$

If our three data points actually fit on a single line, then the following equations would hold:

$$6 = 0M + B$$
$$0 = 1M + B$$
$$0 = 2M + B$$

We can easily see from the above that the equations are not consistent, and we thus cannot find a line / points which satisfy all of them.

To get past this, we need to find the least squares solution of Ax = b where:

$$A = \begin{pmatrix} 0 & 1 \\ 1 & 1 \\ 2 & 1 \end{pmatrix} \qquad b = \begin{pmatrix} 6 \\ 0 \\ 0 \end{pmatrix}.$$

This is where our least square formula $A^TAx = A^Tb$ comes in handy. Let's plug it into our matrix modeling our equations to find our least squares solution.

First, lets find A^TA:

$$A^TA = \begin{pmatrix} 0 & 1 & 2 \\ 1 & 1 & 1 \end{pmatrix} \begin{pmatrix} 0 & 1 \\ 1 & 1 \\ 2 & 1 \end{pmatrix} = \begin{pmatrix} 5 & 3 \\ 3 & 3 \end{pmatrix}$$

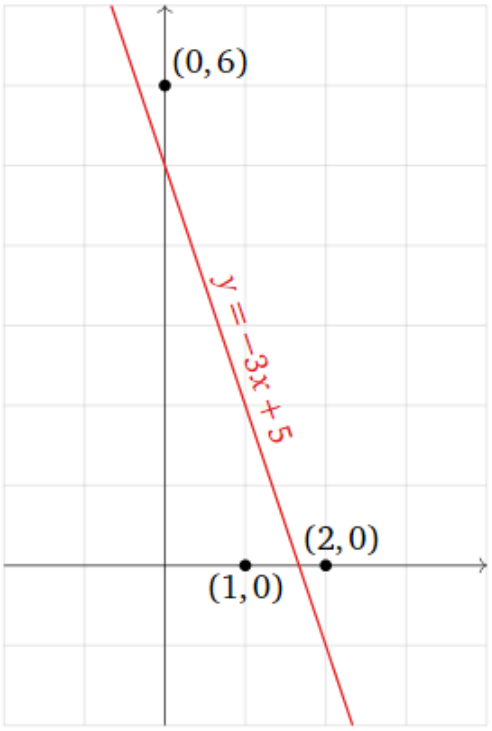
and A^Tb:

$$A^Tb = \begin{pmatrix} 0 & 1 & 2 \\ 1 & 1 & 1 \end{pmatrix} \begin{pmatrix} 6 \\ 0 \\ 0 \end{pmatrix} = \begin{pmatrix} 0 \\ 6 \end{pmatrix}.$$

We then use an augmented matrix and row reduce to find out solution:

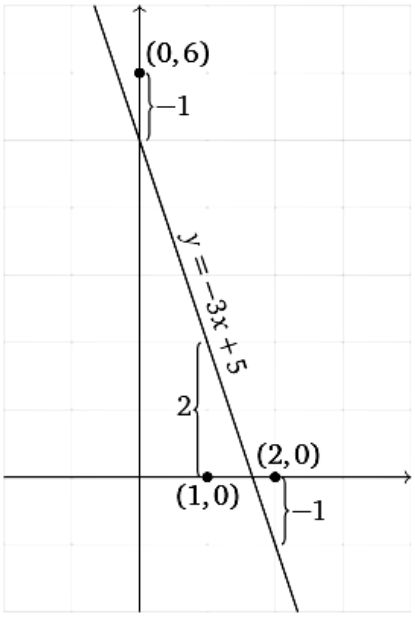
$$\left(\begin{array}{cc|c} 5 & 3 & 0 \\ 3 & 3 & 6 \end{array}\right) \xrightarrow{\text{RREF}} \left(\begin{array}{cc|c} 1 & 0 & -3 \\ 0 & 1 & 5 \end{array}\right).$$

Our least-squares solution is $[-3, 5]^T$ and the line modeling our closes fit solution is represented by $y = -3x + 5$.



This solution essentially minimizes the distance from Ax to b, and although we won't go into the full details as to why our formula above works, we can show what our error using the above solution is the square root of 6:

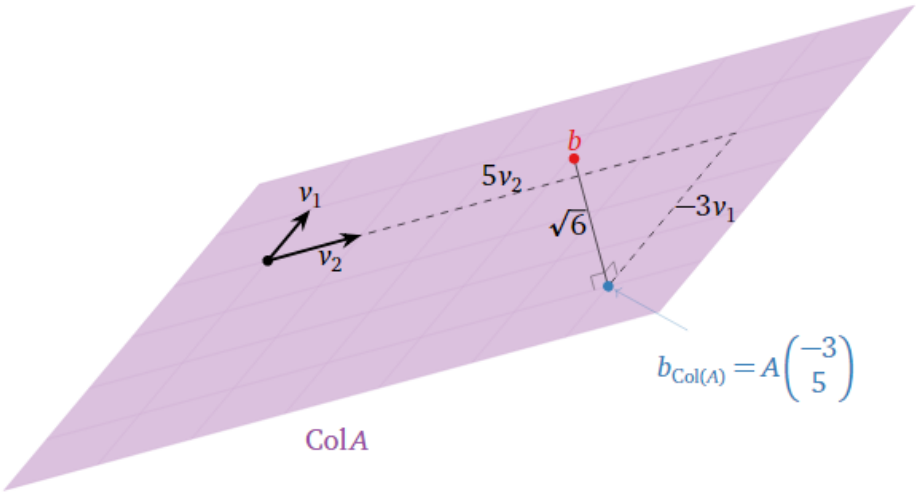
$$\|b - A\hat{x}\| = \left\| \begin{pmatrix} 6 \\ 0 \\ 0 \end{pmatrix} - \begin{pmatrix} 5 \\ 2 \\ -1 \end{pmatrix} \right\| = \left\| \begin{pmatrix} 1 \\ -2 \\ 1 \end{pmatrix} \right\| = \sqrt{1^2 + (-2)^2 + 1^2} = \sqrt{6}.$$



$$b - A\hat{x} = \begin{pmatrix} 6 \\ 0 \\ 0 \end{pmatrix} - A\begin{pmatrix} -3 \\ 5 \end{pmatrix} = \begin{pmatrix} -1 \\ 2 \\ -1 \end{pmatrix}$$

All we're doing in our example above is projecting our linear system onto a column space of A which allows us to find a projection of b which minimizes our square error.

We can try to visualize this using the figure below:



Once again, to find the above, all we have to do is find the solution to $A^TAx = A^Tb$, and the magic of linear algebra and the rules of linear transformation take care of everything for us.

Principal Component Analysis (PCA)

PCA aims to fit straight lines to the data points and we call these straight lines "principal components". There are as many principal components as there are variables. The first principal component is the best straight line you can fit to the data. The second principal component is the best straight line you can fit to the errors from the first principal component. The third principal component is the best straight line you can fit to the errors from the first and second principal components, and we can keep finding more if we choose to do so.

Let's take an example and say that we have high dimension data - say 50 measurements made on ciders and we want to find a way of allocating them onto shelves grouped by their similarity. The ciders have different qualities and slightly different characteristics based on some of these dimensions. With such high dimension data, it's hard to tell which ciders belong to which group. We can essentially use PCA to reduce our dimensions by taking linear combinations of the original variables and separating out which combinations have the most variance in our data set.

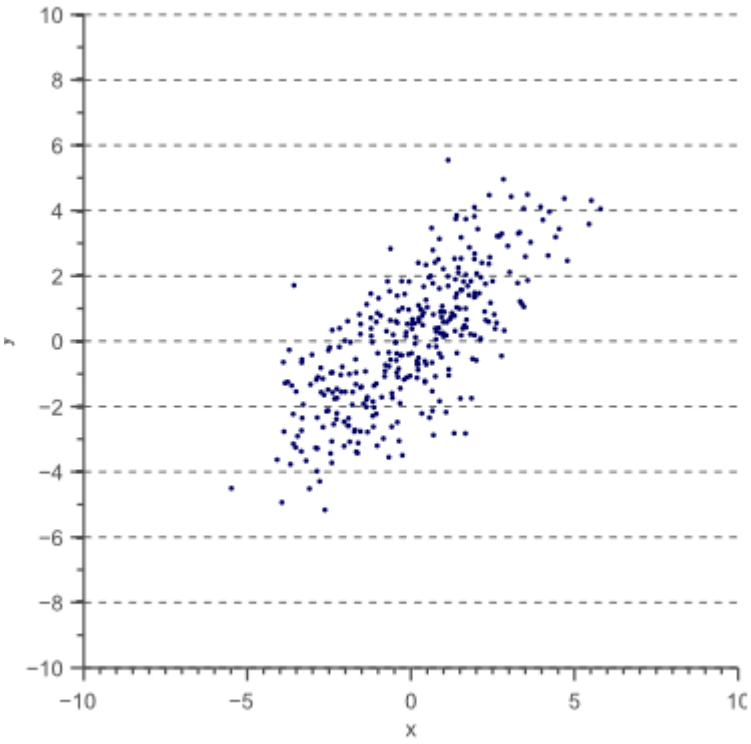
We can try to form a picture of this through trying to visualize shadows and projecting 3-D space onto a 2-D sheet. Suppose you have a Calder mobile that is very complex. Some points in 3-d space close to each other, others aren't. If we hung this mobile from the ceiling and shined light on it from one angle, we get a projection onto a lower dimension plane (a 2-d wall). Now, if this mobile is mainly wide in one direction, but skinny in the other direction, we can rotate it to get projections that differ in usefulness. Intuitively, a skinny shape in one dimension projected on a wall is less useful - all the shadows overlap and don't give us much information. However, if we rotate it so the light shines on the wide side, we get a better picture of the reduced dimension data. The points are more spread out.

So, now that we've established what PCA does, the main question is: how do we get the principle components / components with the most variation?

We do this by taking a covariance matrix and finding the eigenvalues and eigenvectors of this matrix. The eigenvectors and eigenvalues are not needed concepts per se, rather they happened to be mathematical concepts that already existed. When you solve the mathematical problem of PCA, it ends up being equivalent to finding the eigenvalues and eigenvectors of the covariance matrix, and we'll go through the intuition behind by going through an example and showing why this is true.

Covariance Matrix

Let’s take an example data set represented by our graph below:

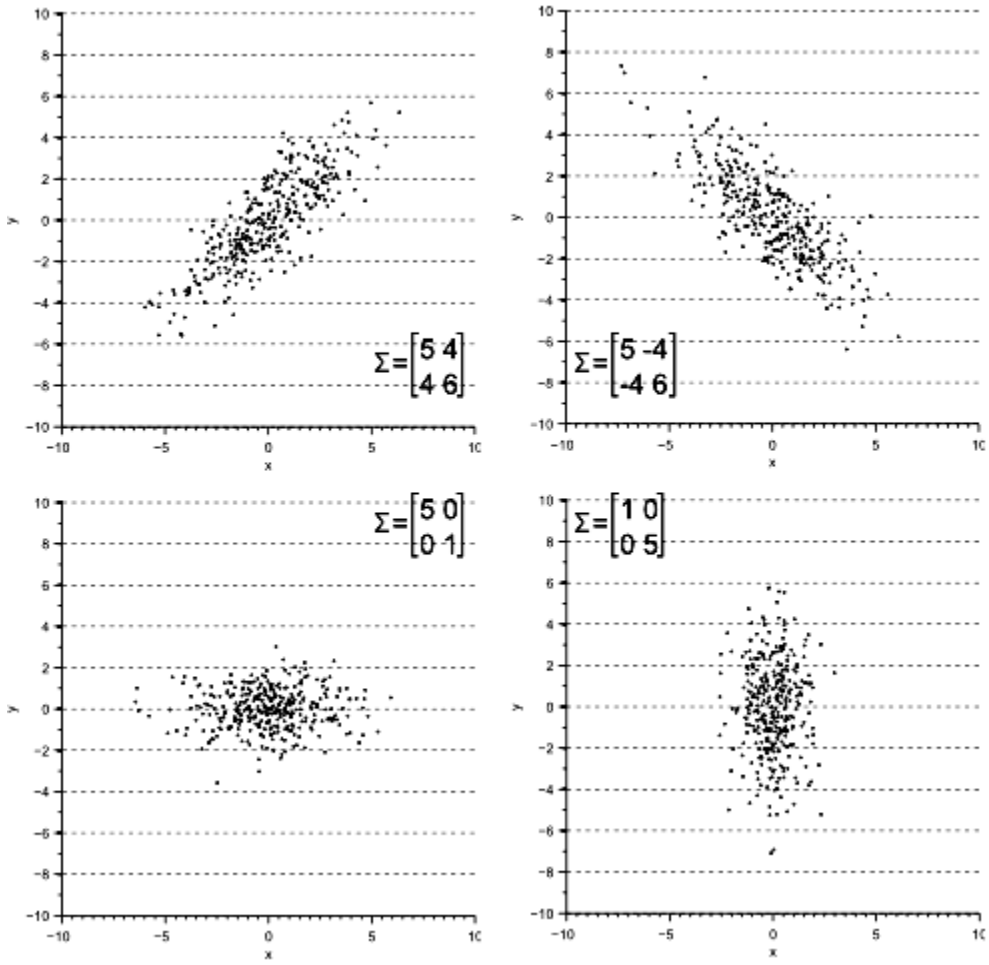


For this data, we could calculate the variance σ(x,x) in the x-direction and the σ(y,y) in the y-direction. However, the horizontal spread and the vertical spread of the data does not explain the clear diagonal correlation. The diagonal spread of the data is captured by the covariance.

In 2-D, we can use the below values to obtain our covariance matrix:

$$\Sigma = \begin{bmatrix} \sigma(x,x) & \sigma(x,y) \\ \sigma(y,x) & \sigma(y,y) \end{bmatrix}$$

The below figure illustrates the shape of our example data by displaying our covariance matrices:

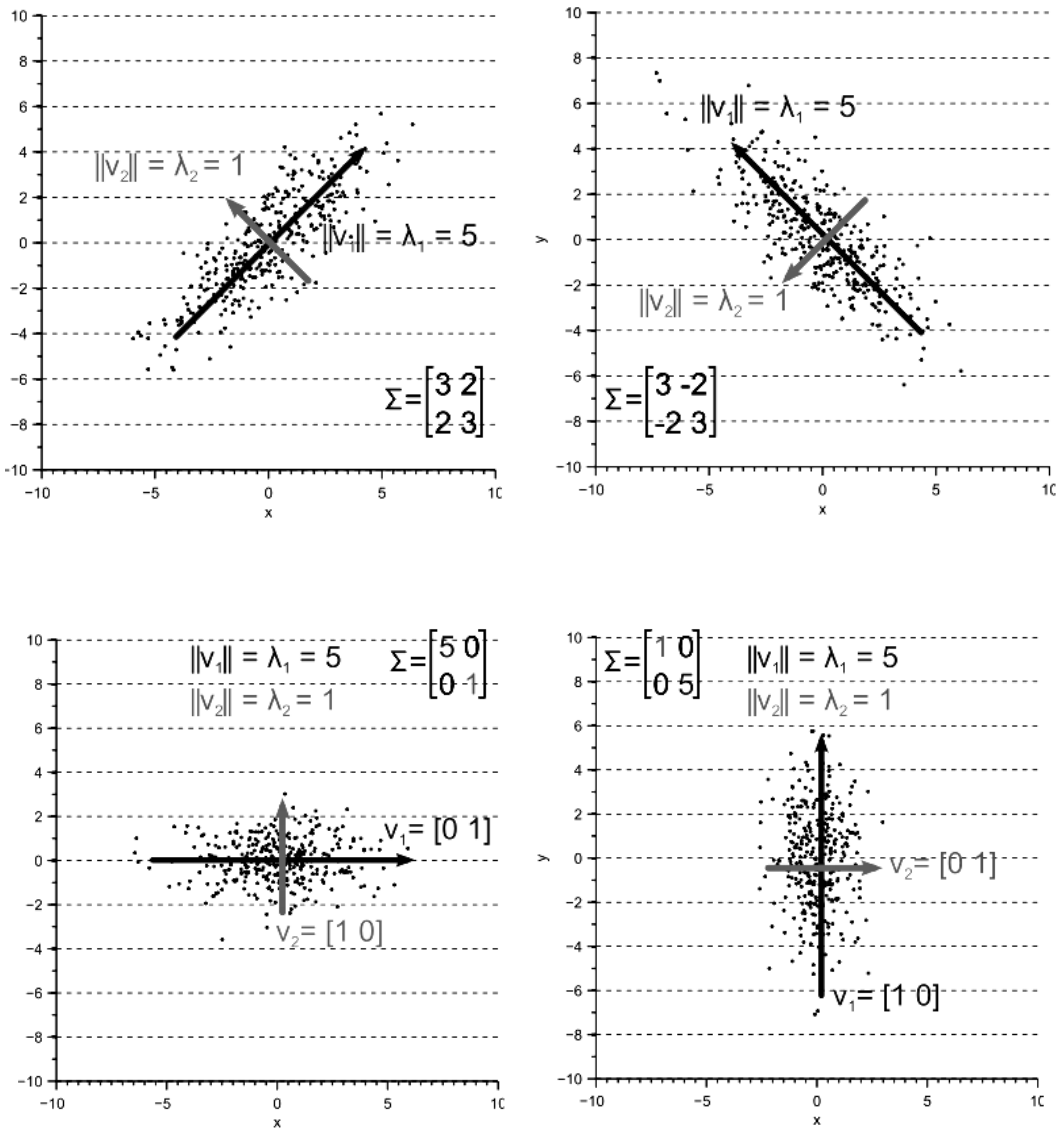


We can see from the above that the diagonal spread is captured by the covariance, while axis-aligned spread is captured by the variance.

To get our principle components, we need to get the eigenvectors and eigenvalues of our matrices.

Why?

Let’s display the eigenvectors and eigenvalues of our matrices:



We can see by looking at the above that the largest eigenvector of the covariance matrix always points into the direction of the largest variance of the data. The magnitude of this vector equals the corresponding eigenvalue. The second largest eigenvector is always orthogonal to the largest eigenvector, and points into the direction of the second largest spread of the data.

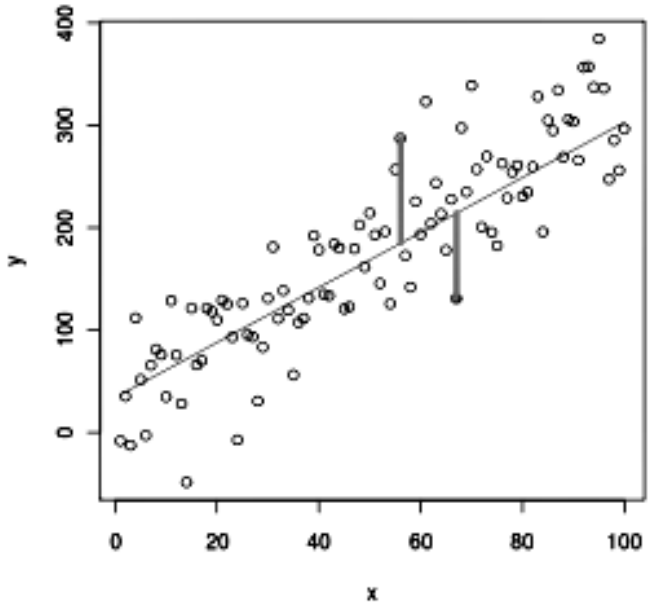
In other words, the eigenvector with the largest eigenvalue is the direction along which the data set has the maximum variance, and we use this fact to obtain our principal components.

PCA vs Ordinary Least Squares

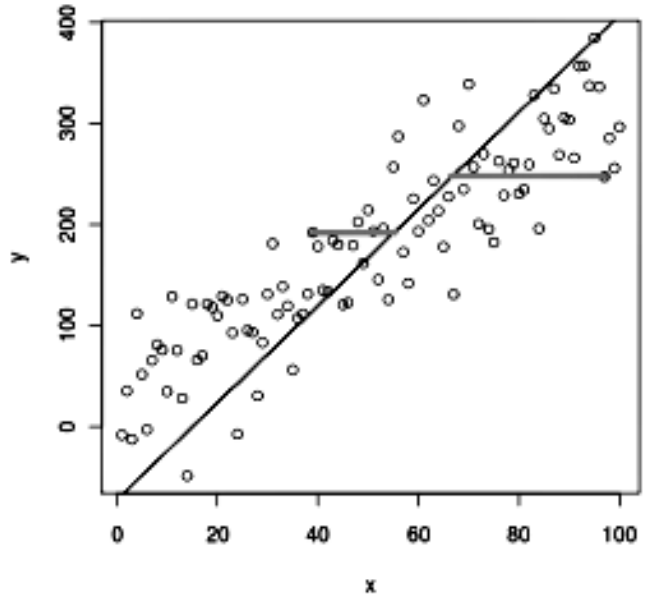
We can also visualize PCA in terms of ordinary least squares to get an intuitive feel for what it is that we're doing.

So, what's the main difference between ordinary least squares regression and PCA?

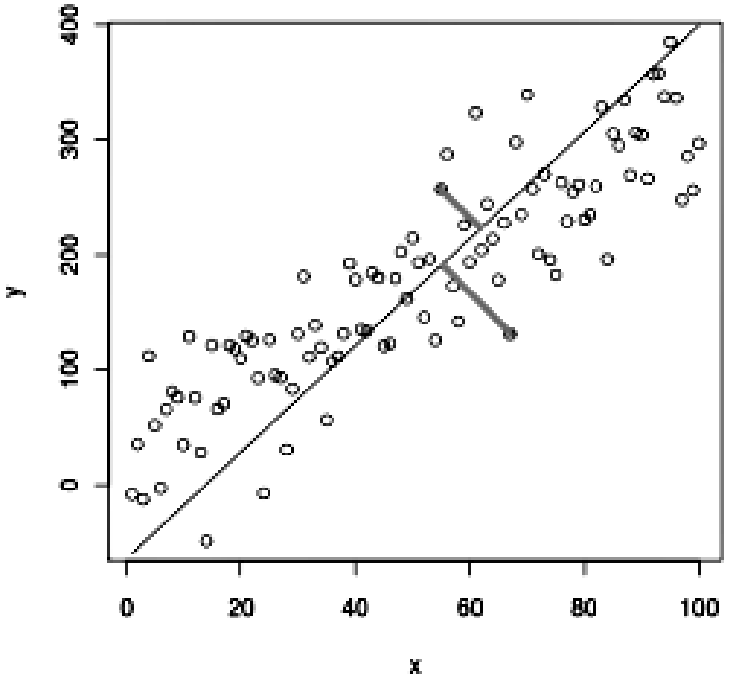
The very short version is ordinary least squares of y ~ x minimizes error perpendicular to the independent axis like this (the lines shown are examples of two errors):



If you were to regress x ~ y (as opposed to y ~ x in the first example) it would minimize error like this:



PCA effectively minimizes error orthogonal to the model itself, like so:



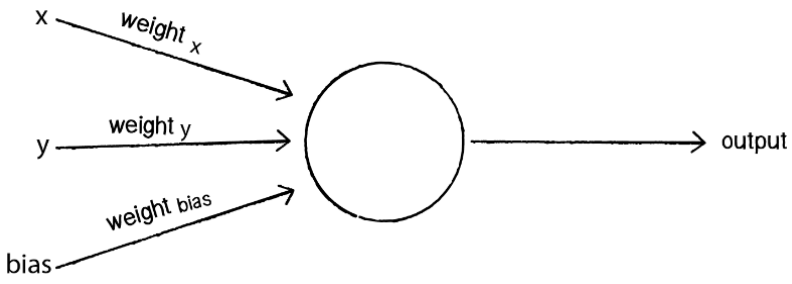
The above picture is a very simplified one with only 2 variables. We have to remember that in everyday situations, we'll have a lot more dimensions to model / choose from. Walking through a simplified case though lets us get a good intuition of what we are doing whenever we use PCA in terms of ordinary least squares regression without having to resort to using our covariance matrix and eigenvalues / eigenvectors.

Deep Learning / Neural Networks

A neural network is actually just a mathematical function. You enter a vector of values, those values get multiplied by other values, and a value or vector of values is output. That is all it is.

They are very useful in problem domains where there is no known function for approximating the given features (or inputs) to their outputs (classification or regression). One example would be the weather, where we might have lots of features (temperature, heat degree days, cool degree days, wind speed, etc…) but nobody can say exactly how to calculate what the weather will be a day from now. A neural network is a function that is structured in a way that makes it easy to alter its parameters to approximate weather predication based on features.

Normally, neural networks are modeled after ‘neurons’ we call perceptrons which are functions which take inputs and provide outputs. A simple perceptron which takes 2 variables (x and y) and has a bias term is shown below:



We can train our perceptron to take inputs and ‘guess’ and output by feeding it training data and adjusting our input weights. This is what we call supervised learning, and using this method, the network is provided with inputs for which there is a known answer and for which it uses the answer to find out if it has made a correct guess. If it’s incorrect, the network can learn from its mistake and adjust its weights.

The process is as follows:

1. Provide the perceptron with inputs for which there is a known answer.

2. Ask the perceptron to guess an answer.

3. Compute the error. (Did it get the answer right or wrong?)

4. Adjust all the weights according to the error.

5. Return to Step 1 and repeat!

The perceptron’s error can be defined as the difference between the desired answer and its guess.

ERROR = DESIRED OUTPUT - GUESS OUTPUT

The error is the determining factor in how the perceptron’s weights should be adjusted. For any given weight, what we are looking to calculate is the change in weight, often called Δweight (or “delta” weight, delta being the Greek letter Δ).

NEW WEIGHT = WEIGHT + ΔWEIGHT

Δweight is calculated as the error multiplied by the input.

ΔWEIGHT = ERROR * INPUT

Therefore:

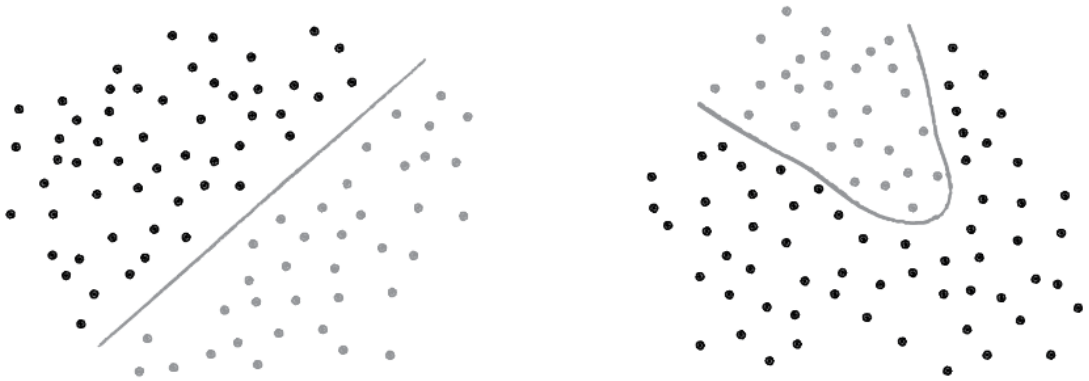
NEW WEIGHT = WEIGHT + ERROR * INPUT

The neural network will also employ another variable called the “learning constant.” We’ll add in the learning constant as follows:

NEW WEIGHT = WEIGHT + ERROR * INPUT * LEARNING CONSTANT

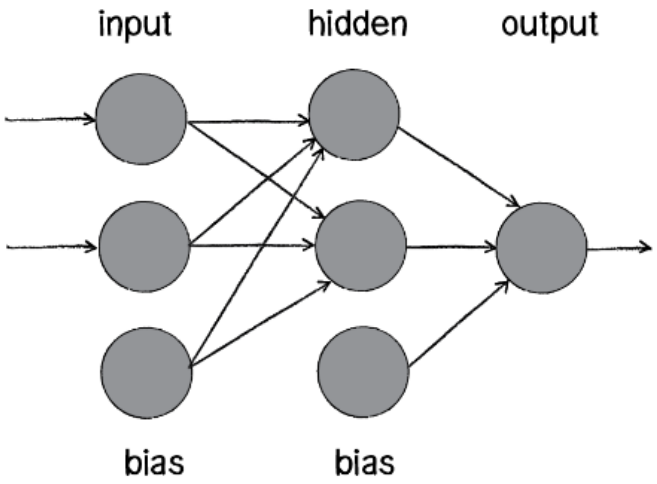
Extending the Perceptron Network into a Neural Network

A perceptron can have multiple inputs, but it’s still only one neuron and it can only be used to solve linearly separable problems:



If we can classify the data with a straight line, then it is linearly separable (left). Our diagram on the right however contains non-linearly separable data. This means that we can’t draw a straight line to separate the black dots from the gray ones.

So how can we extend our perceptrons to be able to classify more complex / non-linearly separable data? We layer our perceptrons:



The above diagram is known as a multi-layered perceptron, a network of many neurons. Some are input neurons and receive the inputs, some are part of what's called a "hidden" layer (as they are connected to neither the inputs nor the outputs of the network directly), and then there are the output neurons, from which we read the results.

Training these networks is much more complicated. With the simple perceptron, we could easily evaluate how to change the weights according to the error. But here there are so many different connections, each in a different layer of the network. How does one know how much each neuron or connection contributed to the overall error of the network?

The solution to optimizing weights of a multi-layered network is known as backpropagation. The output of the network is generated in the same manner as a perceptron. The inputs multiplied by the weights are summed and fed forward through the network. The difference here is that they pass through additional layers of neurons before reaching the output. Training the network (i.e. adjusting the weights) also involves taking the error (desired result - guess). The error, however, must be fed backwards through the network. The final error ultimately adjusts the weights of all the connections, and this is where linear algebra comes to the rescue.

Before that though, we need to review some calculus.

Gradient Descent

Consider the first year calculus derivative. It tells you whether the function is increasing or decreasing to the right.

Consider a function of several variables to the real numbers. It's gradient is a kind of derivative. It is a vector that points in the direction of greatest increase of the function. To increase the value that the function takes fastest, move in the direction of the gradient. To decrease the gradient the fastest, move in the opposite direction.

Take a step, look at the direction the function is decreasing fastest, take another step, look at the direction …

If you are in a space of functions, the gradient is called a functional derivative by some. It's still a vector that points in the direction of greatest increase. To decrease the value of the functional move in the opposite direction.

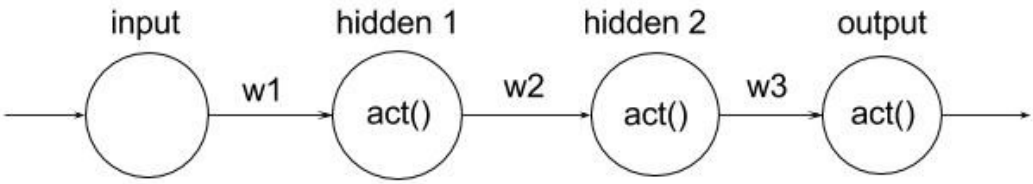
In other words, gradient descent is a first-order optimization algorithm which is used to find a local minimum of a function. Using gradient descent, one takes steps proportional to the negative of the gradient (or of the approximate gradient) of the function at the current point. Depending on where you start, the descent converges to the local optimum.

Backpropagation

In order to understand how backpropagation for these networks is done, we need to familiarize ourselves with a few concepts from vector calculus which we present below.

For any given supervised machine learning problem, we (aim to) select weights that provide the most optimal estimation of a function that models our training data. In other words, we want to find a set of weights W that minimizes on the output of J(W). For the gradient descent algorithm— we would update each weight by some negative, scalar reduction of the error derivative with respect to that weight. If we do choose to use gradient descent (or almost any other convex optimization algorithm), we need to find said derivatives in numerical form.

For other machine learning algorithms like logistic regression or linear regression, computing the derivatives is an elementary application of differentiation. This is because the outputs of these models are just the inputs multiplied by some chosen weights, and at most fed through a single activation function (the sigmoid function in logistic regression). The same, however, cannot be said for neural networks. To demonstrate this, here is a diagram of a single-layered, shallow neural network:



As you can see, each neuron is a function of the previous one connected to it. In other words, if one were to change the value of w1, both “hidden 1” and “hidden 2” (and ultimately the output) neurons would change. Because of this notion of functional dependencies, we can mathematically formulate the output as an extensive composite function:

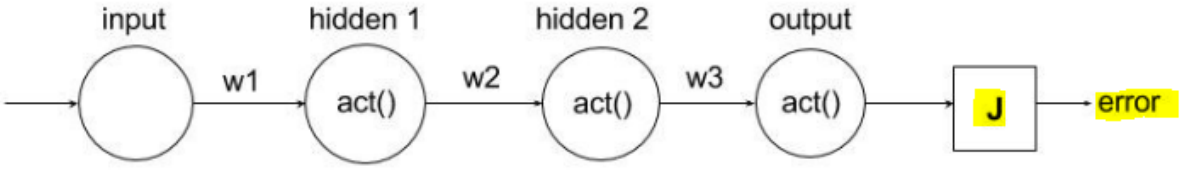
$$output = act(w3 * hidden2)$$
$$hidden2 = act(w2 * hidden1)$$
$$hidden1 = act(w1 * input)$$

And thus:

$$output = act(w3 * act(w2 * act(w1 * input)))$$

Here, the output is a composite function of the weights, inputs, and activation function(s). It is important to realize that the hidden units/nodes are simply intermediary computations that, in actuality, can be reduced down to computations of the input layer. If we were to then take the derivative of said function with respect to some arbitrary weight (for example w1), we would iteratively apply the chain rule (which I’m sure you all remember from your calculus classes).

Now, let’s attach a black box to the tail of our neural network. This black box will compute and return the error—using the cost function—from our output:



All we've done is add another functional dependency; our error is now a function of the output and hence a function of the input, weights, and activation function. If we were to compute the derivative of the error with any arbitrary weight (again, we'll choose w1), the result would be:

$$\frac{\partial error}{\partial w1} = \frac{\partial error}{\partial output} * \frac{\partial output}{\partial hidden2} * \frac{\partial hidden2}{\partial hidden1} * \frac{\partial hidden1}{\partial w1}$$

Each of these derivatives can be simplified once we choose an activation and error function, such that the entire result would represent a numerical value. At that point, any abstraction has been removed, and the error derivative can be used in gradient descent (as discussed earlier) to iteratively improve upon the weight. We compute the error derivatives with regards to every other weight in the network and apply gradient descent in the same way. This is backpropagation. It's simply the computation of derivatives that are fed to a convex optimization algorithm. We call it "backpropagation" because it almost seems as if we are traversing from the output error to the weights, taking iterative steps using chain the rule until we "reach" our weight.

To perform the above, we use linear algebra and the concepts from matrix calculus which we'll overview below.

Vector Calculus and Partial Derivatives

Neural network layers are not single functions of a single parameter, f(x). So, let's move on to functions of multiple parameters such as f(x,y). For example, what is the derivative of x * y?

Well, it depends on whether we are changing x or y. We compute derivatives with respect to one variable (parameter) at a time, giving us two different partial derivatives for this two-parameter function (one for x and one for y). Instead of using operator d/dx, the partial derivative operator is $\partial / \partial x$ (a stylized d and not the Greek letter δ). So $\partial(xy)/\partial x$ and $\partial(xy)/\partial y$ are the partial derivatives of xy; often, these are just called the partials.

The partial derivative with respect to x is just the usual scalar derivative, simply treating any other variable in the equation as a constant.

Consider function $f(x,y) = 3x^2y$. The gradient of this function showing its partial derivatives is provided below:

$$\nabla f(x,y) = [\frac{\partial f(x,y)}{\partial x}, \frac{\partial f(x,y)}{\partial y}] = [6yx, 3x^2]$$

So the gradient of f(x,y) is simply a vector of its partial.

Matrix Calculus

When we move from derivatives of one function to derivatives of many functions, we move from the world of vector calculus to matrix calculus. Let us bring one more function $g(x,y) = 2x + y^8$. So gradient of g(x,y) is:

$$\nabla g(x,y) = [2, 8y^7]$$

If we have two functions, we can organize their gradients into a matrix by stacking the gradients. When we do so, we get the Jacobian matrix where the gradients are rows:

$$J = \begin{bmatrix} \nabla f(x,y) \\ \nabla g(x,y) \end{bmatrix} = \begin{bmatrix} \frac{\partial f(x,y)}{\partial x} & \frac{\partial f(x,y)}{\partial y} \\ \frac{\partial g(x,y)}{\partial x} & \frac{\partial g(x,y)}{\partial y} \end{bmatrix} = \begin{bmatrix} 6yx & 3x^2 \\ 2 & 8y^7 \end{bmatrix}$$

Generally speaking, the Jacobian matrix is the collection of all m by n possible partial derivatives (m rows and n columns), modeled by the stack of m gradients with respect to x.

We can’t compute partial derivatives of very complicated functions using just the basic matrix calculus rules. Part of our goal here is to clearly define and name three different chain rules and indicate in which situation they are appropriate.

The chain rule is conceptually a divide and conquer strategy that breaks complicated expressions into sub-expressions whose derivatives are easier to compute. Its power derives from the fact that we can process each simple sub-expression in isolation yet still combine the intermediate results to get the correct overall result.

The chain rule comes into play when we need the derivative of an expression composed of nested sub-expressions. Chain rules are typically defined in terms of nested functions, such as y= f(u) where u= g(x) so y= f(g(x)) for single-variable chain rules:

$$\frac{dy}{dx} = \frac{dy}{du}\frac{du}{dx}$$

In a similar way, we can apply chain rules to Jacobian matrices as well. A Jacobian can be expressed as 2 other Jacobians:

$$\frac{\partial}{\partial \mathbf{x}}\mathbf{f}(\mathbf{g}(\mathbf{x})) = \begin{bmatrix} \frac{\partial f_1}{\partial g_1} & \frac{\partial f_1}{\partial g_2} & \cdots & \frac{\partial f_1}{\partial g_k} \\ \frac{\partial f_2}{\partial g_1} & \frac{\partial f_2}{\partial g_2} & \cdots & \frac{\partial f_2}{\partial g_k} \\ & & \cdots & \\ \frac{\partial f_m}{\partial g_1} & \frac{\partial f_m}{\partial g_2} & \cdots & \frac{\partial f_m}{\partial g_k} \end{bmatrix} \begin{bmatrix} \frac{\partial g_1}{\partial x_1} & \frac{\partial g_1}{\partial x_2} & \cdots & \frac{\partial g_1}{\partial x_n} \\ \frac{\partial g_2}{\partial x_1} & \frac{\partial g_2}{\partial x_2} & \cdots & \frac{\partial g_2}{\partial x_n} \\ & & \cdots & \\ \frac{\partial g_k}{\partial x_1} & \frac{\partial g_k}{\partial x_2} & \cdots & \frac{\partial g_k}{\partial x_n} \end{bmatrix}$$

Even within this formula, we can simplify further because, for many applications, the Jacobians are square and the off-diagonal entries are zero.

We can use our above rules to derive the gradient with respect to our neural network weights and bias, and use this to perform gradient descent which lets us train our network to output the correct results.

As you can see, linear algebra plays a major role in neural networks. We use matrix / vector calculus in order to train our networks such that our input variables map to our outputs. Through backpropagation, we’re able to come up with ways of tuning our networks such that they’re capable of reaching their goals.

Page Rank

The PageRank algorithm was invented by Page and Brin around 1998 and used in the prototype of Google's search engine. The objective was to estimate the popularity, or the importance, of a web page, based on the interconnection of the web. The rationale behind it is:

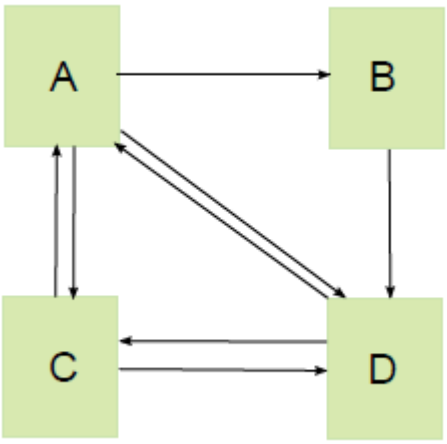
- A page with more incoming links is more important than a page with less incoming links
- A page with a link from a page which is known to be of high importance is also important.

Let's study a simplified example of how PageRank works, as well as how we can relate it to linear algebra and finding eigenvalues / eigenvectors in order to obtain our page rankings.

Simplified PageRank Algorithm

Let's assume that we have 4 pages. Page A contains a link to page B, a link to page C, and a link to page D. Page B contains one single link to page D. Page C points to pages A and D, and page D points to pages A and C.

We can represent the example above through the figure shown below:



Let N be the total number of pages. We create an N by N matrix A by defining the (i, j) - entry as:

$$a_{ij} = \begin{cases} \frac{1}{L(j)} & \text{if there is a link from } j \text{ to } i, \\ 0 & \text{otherwise.} \end{cases}$$

In Example 1, our matrix is a 4 by 4 matrix shown below:

$$\begin{bmatrix} 0 & 0 & 1/2 & 1/2 \\ 1/3 & 0 & 0 & 0 \\ 1/3 & 0 & 0 & 1/2 \\ 1/3 & 1 & 1/2 & 0 \end{bmatrix}$$

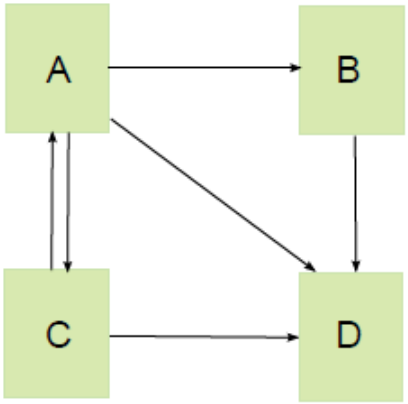
The simplified PageRank algorithm is:

- Initialize x to an N by 1 column vector with non-negative components, and then repeatedly replace x by the product Ax until it converges.

We call the vector x the PageRank **vector.**

The issue with using the above simplified algorithm is the fact that it doesn't handle nodes without any outgoing links very well.

Let's take another example shown below:



The associated matrix is:

$$\begin{bmatrix} 0 & 0 & 1/2 & 0 \\ 1/3 & 0 & 0 & 0 \\ 1/3 & 0 & 0 & 0 \\ 1/3 & 1 & 1/2 & 0 \end{bmatrix}$$

We can see that the entries in the last column are all zero, and that this matrix is not column-stochastic. This leads to the simplified PageRank algorithm collapsing. In this instance, the PageRank vector would ultimately converge to 0.

One remedy is to modify the simplified PageRank algorithm by replacing any all-zero column by:

$$\begin{bmatrix} 1/N \\ 1/N \\ \vdots \\ 1/N \end{bmatrix}$$

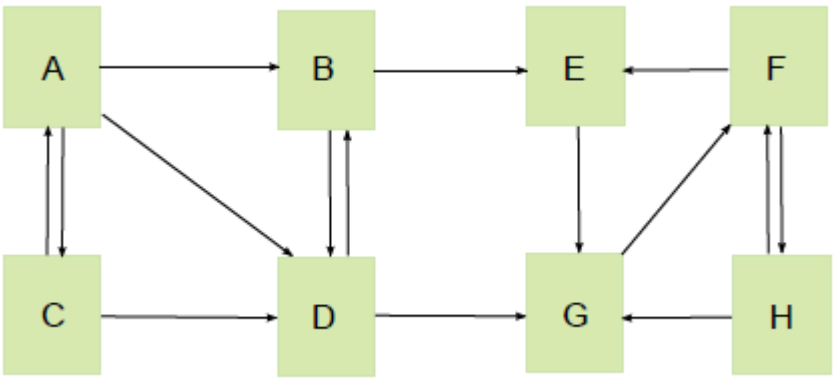
This adjustment assumes that a web surfer who reads a page with no outgoing links jumps to a random page after he/she is done reading. This model may not be realistic, but it simplifies the computation of the algorithm.

In Example 2, the matrix A is now:

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

So, are we done?

Not quite yet. Consider the following web graph consisting of eight web pages:



There is no dangling node. But, once a surfer arrives at pages E, F, G or H, he/she will get stuck in these 4 pages. If we construct matrix A of this graph and try to make it converge, we will find that the ranking of pages A to D eventually drop to 0. This is clearly wrong, since page D has 3 incoming links, and should have some non-zero importance. The PageRank algorithm once again does not work.

Our example web graph belongs to the category of being a reducible graph. In general, a graph is called irreducible if for any pair of distinct nodes, we can start from one of them, follow the links in the web graph and arrive at the other node, and vice versa. In our example, there are no paths from E to B or G to D and the graph is therefore reducible.

In order to calculate page ranks properly for a reducible web graph, Page and Brin proposed that take a weighted average of the matrix with an all-one N by N matrix. Their formulation of this new matrix M is shown below:

$$\mathbf{M} = d\bar{\mathbf{A}} + \frac{1-d}{N}\begin{bmatrix} 1 & 1 & 1 & \dots & 1 \\ 1 & 1 & 1 & \dots & 1 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & 1 & 1 & \dots & 1 \end{bmatrix}$$

In the above figure, we represent A is the modified matrix in the previous section, and d as a number between 0 and 1. The constant d is called the damping factor or the damping constant. The default Google value of d is 0.85. If we set d to 0.85, the matrix M for the web graph in our above example becomes:

$$\mathbf{M} = \begin{bmatrix} 0.0187 & 0.0187 & 0.4437 & 0.0187 & 0.0187 & 0.0187 & 0.0187 & 0.0187 \\ 0.3021 & 0.0187 & 0.0187 & 0.4437 & 0.0187 & 0.0187 & 0.0187 & 0.0187 \\ 0.3021 & 0.0187 & 0.0187 & 0.0187 & 0.0187 & 0.0187 & 0.0187 & 0.0187 \\ 0.3021 & 0.4437 & 0.4437 & 0.0187 & 0.0187 & 0.0187 & 0.0187 & 0.0187 \\ 0.0187 & 0.4437 & 0.0187 & 0.0187 & 0.0187 & 0.4437 & 0.0187 & 0.0187 \\ 0.0187 & 0.0187 & 0.0187 & 0.0187 & 0.0187 & 0.0187 & 0.8688 & 0.4437 \\ 0.0187 & 0.0187 & 0.0187 & 0.4437 & 0.8688 & 0.0187 & 0.0187 & 0.4437 \\ 0.0187 & 0.0187 & 0.0187 & 0.0187 & 0.0187 & 0.4437 & 0.0187 & 0.0187 \end{bmatrix}$$

Using the above matrix, we can say that the PageRank algorithm now converges. The matrix M is column-stochastic by the design of our algorithm. Also, all entries are positive.

We can invoke the theorems listed below to show that the PageRank algorithm now converges.

Perron-Frobenius Theorem: If M is a positive, column stochastic matrix, then:

- 1 is an eigenvalue of multiplicity one.

- 1 is the largest eigenvalue: all the other eigenvalues have absolute value smaller than 1.
- The eigenvectors corresponding to the eigenvalue 1 have either only positive entries or only negative entries. In particular, for the eigenvalue 1 there exists a unique eigenvector with the sum of its entries equal to 1.

Power Method Convergence Theorem: Let M be a positive, column stochastic N by N matrix. We denote v as its probabilistic eigenvector corresponding to the eigenvalue 1. Let z be the column vector with all entries equal to 1 / n . Then the sequence z, Mz, ..., $M^k z$ converges to the vector v.

Essentially, we can see from the above that PageRank creates a matrix representing our page linkage and transforms this mapping in order to ensure that we can obtain a solution showing the importance of each page ranked according to a page's incoming links.

The final PageRank vector is a unique probabilistic eigenvector of our matrix and has a corresponding eigenvalue of 1.

Note that we just used linear algebra to derive an algorithm which easily generates billions of dollars in annual revenue for Google.

Physics

Linear algebra plays an important role in many areas of physics.

Some examples are provided below:

- Thinking about a particle traveling through space, we imagine that its speed and direction of travel can be represented by a vector v in 3-dimensional Euclidean space R^3. Its path in time t might be given by a continuously varying line at each point of which we have the velocity vector v(t).
- In the theory of electromagnetism, Maxwell's equations deal with vector fields in 3-dimensional space which can change with time. At each point of space and time, two vectors are specified, giving the electrical and the magnetic fields at that point.
- Given two different frames of reference in the theory of relativity, the transformation of the distances and times from one to the other is given by a linear mapping of vector spaces.

- In quantum mechanics, a given experiment is characterized by an abstract space of complex functions. Each function is thought of as being a kind of vector. We therefore have a vector space of functions.

The real major importance of linear algebra comes in when we talk about quantum mechanics. Quantum mechanics is important because it plays a fundamental role in explaining how the world works. We can say that quantum mechanics governs the behavior of microscopic systems which in turn govern the behavior of all physical systems, regardless of their size.

Quantum mechanics 'lives' in a Hilbert space, and a Hilbert space is an infinite-dimensional vector space whereby the vectors are actually functions. The mathematics of quantum mechanics is modeled through linear operators in the Hilbert space. We can make a more full analogy / equivalence between quantum mechanics and linear algebra using the mapping shown below:

Quantum mechanics ↔ Linear algebra

wave function	↔	vector
linear operator	↔	matrix
eigenstates	↔	eigenvectors
physical system	↔	Hilbert space
physical observable	↔	Hermitian matrix

The use and importance of linear algebra though doesn't just seemingly 'model' physical behavior. Let's take the example of a wave function and show how Heisenberg derived his famous law using some of the principles that can be taken from linear algebra.

We first start off with a wave function which is used to model particle behavior. It is a function which takes a coordinate vector position = (x, y, z) as an input, which models a point in 3-dimensional space and a time, and outputs (position, time) (normally represented as Ψ (x, t)). Ψ (x, t) is a complex number, called a probability amplitude.

As an example, if we fix time = 3, then the function Ψ (position, time=3) provides a complete description of the state of the particle 3 seconds after the clock was started and the probability of finding it at that point. A probability amplitude is like a probability density function, except that probability density functions are always real valued functions, while Ψ is complex.

If we take the modulus of the probability amplitude and square it, we get an ordinary probability density function $|\Psi(\text{position, time} = 3)|^2$ which gives the probability of finding the particle at that point after 3 seconds. The total probability of finding the

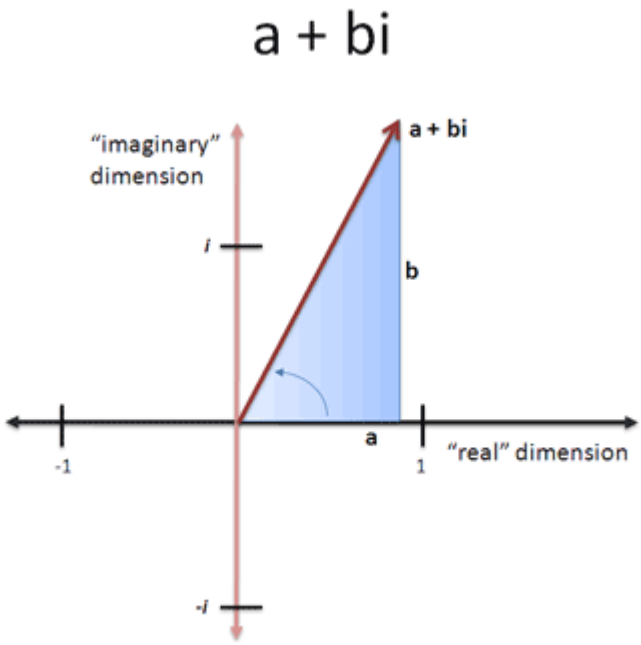
particle somewhere must be equal to 1, so a defining characteristic of a wave function Ψ is that it is normalized, meaning that $\int |\Psi(\text{position, time})|^2 \, dt = 1$ for every value of variable ‘time’.

Once again though, although we may now have a better grasp of continuous probability distributions, in the case of modeling the wave function, we have to note that we don’t obtain a real valued output. We instead obtain complex numbers.

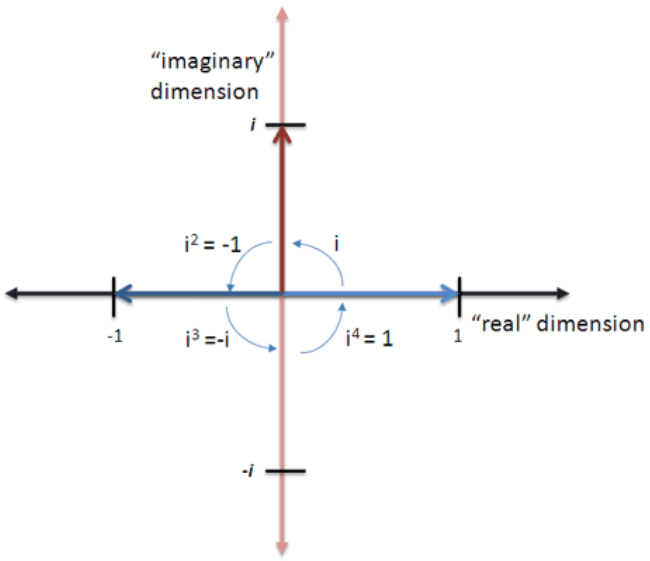
How can a probability of an event be modeled by a complex number? It isn’t immediately obvious what statements such as Ψ(position, time=3) = i / 4 + 1 / 3 say about the particle at the specified position at 3 seconds.

We can get an intuitive grasp of how complex numbers model amplitudes as imagining that the probability distribution is shifted from being a simple one-dimensional value to a 2-dimensional one, where we now have one value representing the ‘real’ dimension and another one representing an ‘imaginary’ one.

We can visualize this intuitively by taking a look at the diagram below, where a is the real part while b is the imaginary one:



We also note that using our new system of dimensions, multiplication by i simply represents a 90 degree rotation:



Like negative numbers modeling flipping, imaginary numbers can model anything that rotates between two dimensions "X" and "Y". Or anything with a cyclic, circular relationship.

To put it more simply, the above means that we're transitioning from modeling a regular scalar probability to modeling it in terms of vectors we locate in our 'complex' space.

Noting the above, we must also note also that the function doesn't simply encode one property of our point / particle. It also encodes the familiar properties such as position, momentum, and energy, even if they take on radically new flavors in terms of 2-dimensional complex probability distribution values rather than real valued ones. We call these properties observables.

The simplest observable is position. This has a straightforward formula, given by the coordinates at point p. Accompanying this is a probability distribution which gives the particle's probability of having the position at time t, which is once again given by the real number $|\Psi(\text{point, time})|^2$

Using the wave function also allows us to extract information about its momentum, another important observable. Momentum is a vector quantity, with components in the x, y, z dimensions / directions.

Focusing on the x direction, the defining formula of momentum is:

$$p_x = -ih(\partial/\partial x)$$

This looks extremely strange! We expect momentum to be a number, but this defines it as something else, a differential operator. Similar formulas hold for the y and z components of momentum.

Using vector calculus, we can write these three formulas as one, to give the definition of the momentum operator:

$$p = -ih\nabla$$

Just as for the position, we may also extract a corresponding probability distribution which gives the probability of the particle having momentum of a particular value.

There are two ways of viewing a wave function. One is for the function Ψ to take position as the primary consideration, and deduce information about momentum from it. It is equally possible to take the opposite perspective, which is to take the momentum as the primary consideration, and deduce the positional information from it. However, the order in which these are applied makes a difference.

Let's focus in on the x-direction and note that we use x to model the position operator, and we use our momentum equation to model the x-coordinate momentum: -(i)(h) (∂/∂x).

A simple application of the product rule says that:

$$(\partial/\partial x)(x)(\Psi) = (\Psi) + (x)(\partial/\partial x)(\Psi)$$

Re-arranging this, we get:

$$(\partial/\partial x)(x)(\Psi) - (x)(\partial/\partial x)(\Psi) = \Psi$$

We can actually note that the above works for any wave function, so we can simplify further by eliminating Ψ from our equation:

$$(\partial/\partial x)(x) - (x)(\partial/\partial x) = 1$$

Of course, momentum in the x-direction is not given by (∂/∂x), but by -(i)(h) (∂/∂x), so we multiply both sides by -(i)(h) to get:

$$\mathbf{-(i)(h)(}\partial/\partial\mathbf{x)}(x) - (x)\mathbf{(-(i)(h)(}\partial/\partial\mathbf{x))} = -(i)(h)$$

Or, substituting p_x to represent the momentum operator, we get:

$$p_x x - x p_x = -ih$$

As we can see from the above, $p_x x \neq x p_x$,which shows that the two operators do not commute. Similar formulas hold in the y and z directions.

If two operators don't commute, it means that they cannot have simultaneous eigenvector bases, and that the two physical quantities are never well defined simultaneously.

The non-commuting of the position and momentum operators was first noticed by Warner Heisenberg in 1925. At first, it may seemed no more than a minor technical inconvenience, but within a year, Heisenberg had realized its extraordinary repercussions, with major consequences for experimental physics and experimental science.

Heisenberg's uncertainty relation says that:

$$\Delta x \Delta p \geq (½)\ \hbar$$

where ħ is the reduced plank's constant.

Heisenberg's uncertainty relations have a profound implication. If the position x is extremely localized in space, it follows that Δx is very small. If so, then for the Heisenberg inequality we outlined above to hold, Δp_x must a large number, and therefore, the momentum is widely spread out. The converse is also true: if the momentum is narrowed down to a small number, then it must be true that the position is widely spread. In other words, it is **impossible** to pin down both the position and momentum simultaneously.

As you can see, what Heisenberg thought was an algebraic hindrance turned out to be a very important law within our universe, which states that a 'particle' cannot have a unique position and momentum at the same instance in time.

Resources / Credits

Full credit on all of the content included in this linear algebra review goes to the authors of the linked content below:

https://www.youtube.com/playlist?list=PLZHQObOWTQDPD3MizzM2xVFitgF8hE_ab

http://immersivemath.com/ila/ch03_dotproduct/ch03.html

https://uwaterloo.ca/institute-for-quantum-computing/sites/ca.institute-for-quantum-computing/files/uploads/files/mathematics_qm_v21.pdf

https://mathinsight.org/determinant_linear_transformation

https://mathinsight.org/cross_product

https://ocw.mit.edu/courses/mathematics/18-06sc-linear-algebra-fall-2011/index.htm

www.cns.nyu.edu/~eero/NOTES/geomLinAlg.pdf

http://www.visiondummy.com/2014/03/eigenvalues-eigenvectors/

https://textbooks.math.gatech.edu/ila/least-squares.html

http://www.visiondummy.com/2014/04/geometric-interpretation-covariance-matrix/

https://stats.stackexchange.com/questions/2691/making-sense-of-principal-component-analysis-eigenvectors-eigenvalues/140579#140579

http://pi.math.cornell.edu/~mec/Winter2009/RalucaRemus/Lecture3/lecture3.html

http://home.ie.cuhk.edu.hk/~wkshum/papers/pagerank.pdf

https://natureofcode.com/book/chapter-10-neural-networks/

https://medium.com/@rohitrpatil/the-matrix-calculus-you-need-for-deep-learning-notes-from-a-paper-by-terence-parr-and-jeremy-4f4263b7bb8