Matrices

A matrix, like a vector, is also a collection of numbers. The difference is that a matrix is a table of numbers rather than a list. Many of the same rules we just outlined for vectors above apply equally well to matrices. (In fact, you can think of vectors as matrices that happen to only have one column or one row.) First, let's consider matrix addition and subtraction. This part is uncomplicated. You can add and subtract matrices the same way you add vectors – element by element:

$$A + B = C$$

$$\begin{pmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{pmatrix} + \begin{pmatrix} b_{11} & b_{12} \\ b_{21} & b_{22} \end{pmatrix} = \begin{pmatrix} a_{11} + b_{11} & a_{12} + b_{12} \\ a_{21} + b_{21} & a_{22} + b_{22} \end{pmatrix}$$

Matrix multiplication gets a bit more complicated, since multiple elements in the first matrix interact with multiple elements in the second to generate each element in the product matrix. This means that matrix multiplication can be a tedious task to carry out by hand, and can be time consuming on a computer for very large matrices. But as we shall see, this also means that, depending on the matrices we are multiplying, we will be able to perform a vast array of computations. Shown here is a simple example of multiplying a pair of 2×2 matrices, but this same procedure can generalize to matrices of arbitrarily large size, as will be discussed below:

$$AB = D$$

$$\begin{pmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{pmatrix} \cdot \begin{pmatrix} b_{11} & b_{12} \\ b_{21} & b_{22} \end{pmatrix} = \begin{pmatrix} a_{11}b_{11} + a_{12}b_{21} & a_{11}b_{12} + a_{12}b_{22} \\ a_{21}b_{11} + a_{22}b_{21} & a_{21}b_{12} + a_{22}b_{22} \end{pmatrix}$$

If you look at what's going on here, you may notice that what's going on it that we're taking dot products between rows of the ***A*** matrix with columns of the ***B*** matrix. for example to find the entry in our new matrix that's in the first-row-second-column position, we take the first row of ***A*** and compute it's dot product with the second column of ***B***. Matrix multiplication and dot products are intimately linked.

When multiplying two matrices together, the two matrices do not need to be the same size. For example, the vector $\vec{a} = (4,3)$ from earlier in this chapter represents a very simple matrix. If we multiply this matrix by another simple matrix the result is:

$$\begin{pmatrix} 0 & 1 \\ -1 & 0 \end{pmatrix} \begin{pmatrix} 4 \\ 3 \end{pmatrix} = \begin{pmatrix} 3 \\ -4 \end{pmatrix}$$

Linear Transformations

Linear transformations are functions (functions that take vectors as inputs), but their a special subset of functions. Linear transformations are the set of all functions that can be written as a matrix multiplication:

$$\vec{y} = A\vec{x}$$

This sees like a pretty restrictive class of function, and indeed it is. But as it turns out, these simple functions can accomplish some surprisingly interesting things.

Consider the matrix we used for matrix multiplication in our last example.

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

When multiplied the vector [4,3] by this matrix, we got the vector [-3,4]. If you draw those to vectors you'll notice that the two vectors have the same length, but the second vector is rotated 90° counter clockwise from the first vector. Coincidence? Pick another vector, say [100, 0]; that vector gets transformed to the vector [0, 100], rotating 90° from the 3 o'clock position to the 12 o'clock position. As it turns out this is true for any vector $\vec{x}$ you pick: $\vec{y}$ will always be rotated 90° counter clockwise.

As it turns out our matrix $\boldsymbol{A}$ is a very special matrix. It's a *rotation matrix* – it never changes the length of a vector, but changes every vector's direction by a certain fixed amount – in this case 90°. This is just one special case of a rotation matrix. We can generate a rotation matrix that will allow us to rotate a vector by any angle we want. The general rotation matrix to rotate a vector by an angle θ looks like this:

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

First you plug in the desired value for θ to calculate the different elements of the matrix, then you multiply the matrix by the vector and you have a rotated vector.

This may sound like an awful lot to go through when a protractor would do just as well. The reason this is worth all the trouble is that it doesn't stop at rotations, it doesn't even stop at two dimensions. Above we defined a rotation matrix $\boldsymbol{R}$ for two dimensions. We can just as well define a rotation matrix in three dimensions, it would just be a 3×3 matrix rather than the 2×2 one above. In the case above, we think of the rotation matrix as "operating on" the vector to perform a rotation. Following this terminology, we call the rotation matrix the *operator*. We can make all sorts of other linear operators (matrixes) to do all sorts of other transformations. A few other useful linear transformations include identity $(\boldsymbol{I})$, Stretch and squash $(\boldsymbol{S})$, Skew $(\boldsymbol{W})$ and Flip $(\boldsymbol{F})$.

will also be a linear operator. For example if we want to rotate a vector and then stretch it we could, premultiply those to matrices together, and the resulting matrix operator will still be linear and will still follow all the same rules. So if $\boldsymbol{S} \cdot \boldsymbol{R} = \boldsymbol{X}$, then we can apply $\boldsymbol{X}$ as its own operator. Extending this, any set of linear transformations, no mater how long can be multiplied together and condensed into a single operator. For instance, if we wanted to stretch-skew-rotate-skew-stretch-flip-rotate $(\boldsymbol{S} \cdot \boldsymbol{W} \cdot \boldsymbol{R} \cdot \boldsymbol{W} \cdot \boldsymbol{S} \cdot \boldsymbol{F} \cdot \boldsymbol{R})$, we can define all of that into a single new operator $\boldsymbol{Y}$.