Tutorial 2 - Learning about the Discrete Fourier Transform

This tutorial will be about the Discrete Fourier Transform basis, or the DFT basis in short.

What is a basis?

If we google define 'basis', we get: "the underlying support or foundation for an idea, argument, or process". In mathematics, a basis is similar. It is an underlying structure of how we look at something. It is similar to a coordinate system, where we can choose to describe a sphere in either the Cartesian system, the cylindrical system, or the spherical system. They will all describe the same thing, but in different ways. And the reason why we have different systems is because doing things in specific basis is easier than in others. For example, calculating the volume of a sphere is very hard in the Cartesian system, but easy in the spherical system.

When working with discrete signals, we can treat each consecutive element of the vector of values as a consecutive measurement. This is the most obvious basis to look at a signal. Where if we have the vector [1, 2, 3, 4], then at time 0 the value was 1, at the next sampling time the value was 2, and so on, giving us a ramp signal. This is called a time series vector. However, there are also other basis for representing discrete signals, and one of the most useful of these is to use the DFT of the original vector, and to express our data not by the individual values of the data, but by the summation of different frequencies of sinusoids, which make up the data. It's like a Fourier transform, but discrete.

A more 'mathsy' definition of a basis may be: "A basis is any set of linearly independent vectors. It is useful because for a basis of R^n space, any n-dimensional vector can be exactly reproduced by a linear combination of the basis." Also, there are some boring mathsy terms which I should bring up, which may be used later: an 'Orthogonal' basis is a basis where all vectors in the basis are orthogonal to each other. An easy example of this is the matrix:

$$\begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{1}$$

In this matrix, the three vectors are going in the x, y, and z directions respectively, and therefore it is an orthogonal basis. Another term is 'Orthonormal'. Any matrix which is orthonormal is orthogonal, and each vector in it has unit length.[1] The useful properties of orthonormal matrices is that for matrix A, $A^{-1} = A^T$. That is, its inverse is equal to its transpose. Also extremely useful is the fact that when you transform between two orthogonal basis, the length and angles are maintained.

[1] Read: Length = 1, so that when we change basis everything doesn't get bigger.

Exercise: Are the following matrices orthonormal, orthogonal, or neither?

$$\begin{bmatrix} 1 & 1 & 0 \\ 1 & -1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{2}$$

$$\begin{bmatrix} 0 & 1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{3}$$

$$\begin{bmatrix} 1 & 1 & 0 \\ 0 & 0 & 0 \\ 0 & 1 & 1 \end{bmatrix} \tag{4}$$

$$\begin{bmatrix} \frac{1}{\sqrt{2}} & 0 & \frac{1}{\sqrt{2}} \\ \frac{1}{\sqrt{6}} & \sqrt{\frac{2}{3}} & -\frac{1}{\sqrt{6}} \\ -\frac{1}{\sqrt{3}} & \frac{1}{\sqrt{3}} & \frac{1}{\sqrt{3}} \end{bmatrix} \tag{5}$$

You don't really need to check that last one. It **is** orthonormal, it's just more of an exercise to show that complicated matrices can also be orthonormal.

Basis Transforms

Basis transforms are when you take a vector, and describe that same vector in another basis. The way you do this, and you probably remember it from MATH1051 or 1052 or something, is you multiply your vector by the transform matrix.

Take for example the rotation of a 2 dimensional matrix by some angle θ. This is done like so:

$$\begin{bmatrix} x\prime \\ y\prime \end{bmatrix} = \begin{bmatrix} cos(\theta) & sin(\theta) \\ -sin(\theta) & cos(\theta) \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} \tag{6}$$

where the transformation matrix changes depending on the angle of rotation to the new basis.

The DFT Basis Transform

Because of the way imaginary numbers work, and the way they are represented on the unit plane, we can show that: $f(t) = \cos(\omega t) + isin(\omega t)$ which is equal to the complex exponential $f(t) = e^{-2\pi i\omega t}$. We will be using the exponential form from now on.

So now we want to invent the vectors for our DFT transform matrix. The way we do this is to say: "Hey, we want each vector to be a sinusoid, so lets make each vector the sampled values of a sinusoid, and each vector varies by the value of ω we use!"

So for a 4x4 matrix, our vectors become:

$$\begin{bmatrix} e^{-2\pi i\omega_0 t_0} & e^{-2\pi i\omega_0 t_1} & e^{-2\pi i\omega_0 t_2} & e^{-2\pi i\omega_0 t_3} \\ e^{-2\pi i\omega_1 t_0} & e^{-2\pi i\omega_1 t_1} & e^{-2\pi i\omega_1 t_2} & e^{-2\pi i\omega_1 t_3} \\ e^{-2\pi i\omega_2 t_0} & e^{-2\pi i\omega_2 t_1} & e^{-2\pi i\omega_2 t_2} & e^{-2\pi i\omega_2 t_3} \\ e^{-2\pi i\omega_3 t_0} & e^{-2\pi i\omega_3 t_1} & e^{-2\pi i\omega_3 t_2} & e^{-2\pi i\omega_3 t_3} \end{bmatrix} \tag{7}$$

Where here, t = [0, 1, 2, 3]. And ω = [0, $\frac{1}{4}$, $\frac{2}{4}$, $\frac{3}{4}$].[2] Substituting these in gives you the transformation matrix:

$$\begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & -i & -1 & i \\ 1 & -1 & 1 & -1 \\ 1 & i & -1 & -i \end{bmatrix} \tag{8}$$

Note that this isn't orthonormal, just orthogonal. It needs to be scaled by $\frac{1}{\sqrt{N}} = \frac{1}{2}$ to be orthonormal, but we will do this later because it's easier.

An Example

So to test this out, lets find the DFT of the relatively simply vector: [0, 1, 0, -1]. This is a sampled sine wave with a frequency of $\sin(0.5\pi t)$. If we push this through Matlab we should get the vector:

$$\begin{bmatrix} 0 \\ -2i \\ 0 \\ 2i \end{bmatrix} \tag{9}$$

And now if we add our scaling factor of $\frac{1}{2}$ our vector is:

$$\begin{bmatrix} 0 \\ -i \\ 0 \\ i \end{bmatrix} \tag{10}$$

So if we substitute this into the form: $f(t) = \cos(\omega t) + i\sin(\omega t)$, we find that our signal consists of:

$$-i\cos(\frac{1}{2}\pi t) + \sin(\frac{1}{2}\pi t) + i\cos(\frac{3}{2}\pi t) - \sin(\frac{3}{2}\pi t) \tag{11}$$

Now, if we only plot the real parts of the equation, we get the red graph below:

[2] These are the example numbers for a matrix with $N = 4$. If we generalise the matrix, then t ranges from 0 to $N-1$, and ω ranges from $\frac{0}{N}$ to $\frac{N-1}{N}$

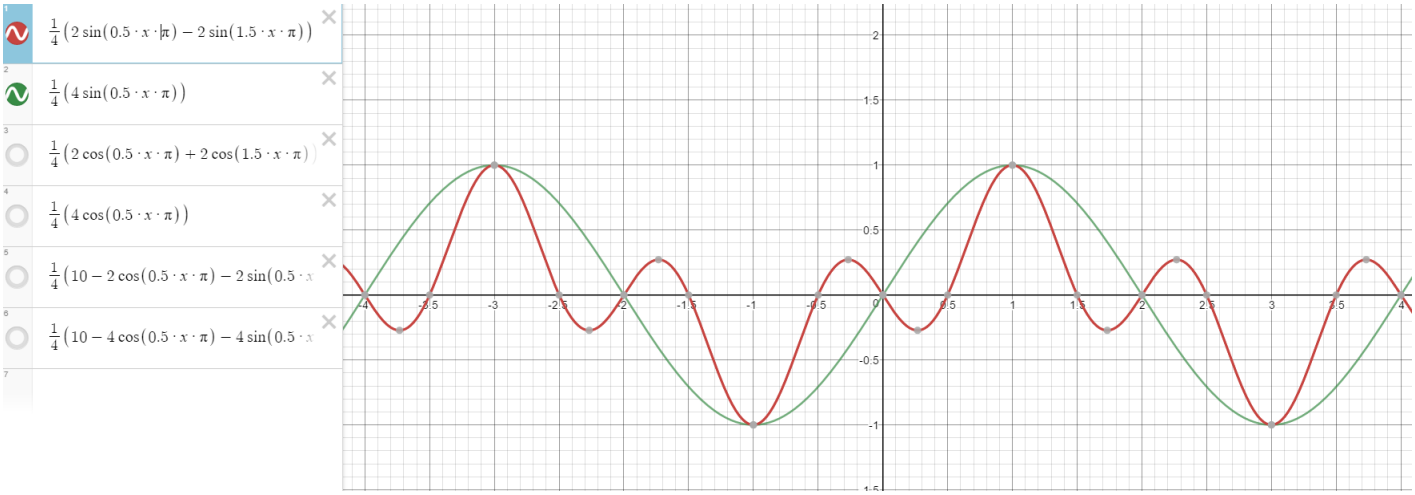

Figure 1: A sinusoidal reconstruction of the time series vector: [0, 1, 0, -1].

Note that the red line is quite wavy, and more complicated than we originally assumed it would be, since it is only meant to reconstruct a single sin wave. However, despite this, it still touches the four specified coordinates perfectly, at (0, 0), (1, 1), (2, 0), and (3, -1).

If we look carefully at the frequencies of the sin waves, we see that it is a combination of two sine waves at frequencies of: $\omega = \frac{1}{2}\pi$ rads per t and $\omega = \frac{3}{2}\pi$ rads per t respectively. Since our sample rate is 1 per t, then the Nyquist frequency is exactly π rads per t. So the DFT has split our reconstruction into two equal parts, one below the Nyquist frequency, and one above the Nyquist frequency. If we wish to only use the frequency component below Nyquist frequency, then we simply remove the higher frequency section, and double the strength of the lower one. Essentially 'flipping' the high frequency part of the spectrum back over the lower part. Doing this results in the green function also plotted in the above Figure. This is the original sine wave we hoped to recover!

Some More Examples

So that last example was rather simple, it was just recovering a simple sine wave. One question you might have from that example is: "Why were the coefficients of the vector imaginary? That doesn't even make sense!". Well lets use another example to explain this.

Lets do the exact same calculation with a cosine wave. We will use the time series set of points: [1, 0, -1, 0]. Using the following code in Matlab, you can find the transformed vector after we multiple by the DFT matrix:

```
a = [1; 0; -1; 0]
b = dtfmtx(4);
b * a
```

This will reveal the result: [0, 2, 0, 2]. So now when we multiply: $cos(\omega t) + i\sin(\omega t)$ by 2, rather than 2i, we recover a real cosine portion, and the sin remains imaginary. So when we plot the addition of these sinusoids, we get the following graph:

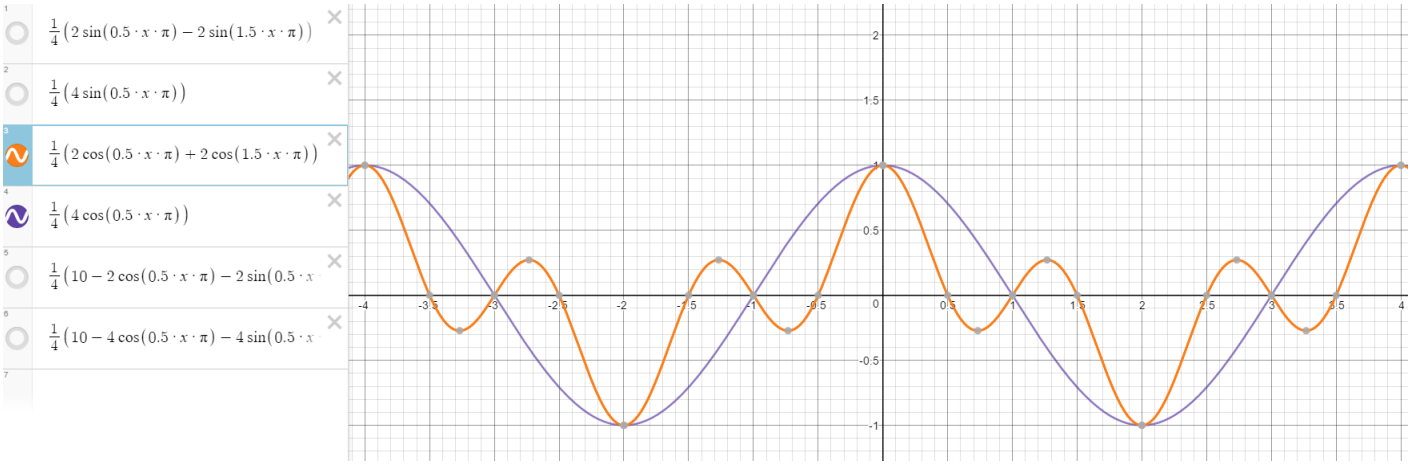

Figure 2: A sinusoidal reconstruction of the time series vector: [1, 0, -1, 0].

So you can see, that these imaginary numbers which come out as coefficients help tune the combinations of sin and cosine waves, to control the phase of the output. After all, this cosine is essentially the same set of points as last example, but shifted by 90 degrees. So in summary, these weird imaginary numbers control the phase of the output. In Figure 2 also, the orange function contains a frequency higher than Nyquist, so I've folded that back over to the lower frequency spectrum to give the blue sinusoid. Note that both still recreate the original time series vector.

"But both of those examples were extremely boring!" you might now say, so never fear, I've also prepared an interesting one! How about the time series vector: [1, 2, 3, 4]. That ramp can't be neatly recreated with sin waves, will the matrix transformation still work? We can type the following into Matlab and check!

```
a = [1; 2; 3; 4]
b = dtfmtx(4);
b * a
```

And the answer we will get returned is: [10, -2+2i, -2, -2-2i]. Lets plot these coefficients!

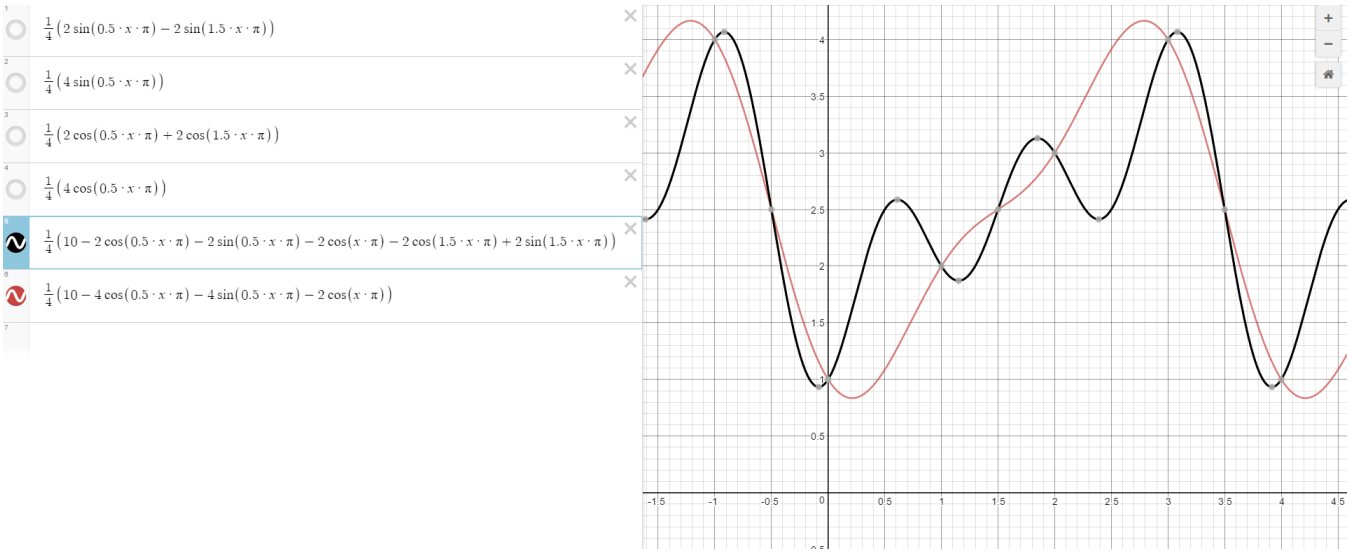

Figure 3: A sinusoidal reconstruction of the time series vector: [1, 2, 3, 4].

So here you can see that again there are two reconstructions which match the points perfectly, The black, which features high frequency components, and the red, where I have removed the high frequency ones, and doubled the low frequency ones.

The FFT Algorythm

So far we've gone over how to calculate the DFT's from the original time series data points. And we've interpreted these to recreate our origional set of points. But we've only done this with the 4x4 DFT transformation matrix, and already it looks like a large amount of effort to calculate. I mean, a computer can obviously do it in a few seconds, but as the matrix gets larger, that means many more calculations. As it stands, the current method of solving the DFT is an $O(n^2)$ algorythm. So if you want to calculate this with one million coefficients, you're looking at one trillion calculations, which will take ≈250 seconds to calculate with a quad core at 1GHz assuming low overhead.[3] The FFT algorythm is a method of reducing the complexity of the same calculation we just performed, to reduce the problem to an $O(nln(n))$ problem. This means with a million coefficients, you're looking at closer to 20 million operations which, on the same computer, will execute in ≈5ms rather than the 250 seconds of before. So as you can see, it's a rather large improvement.

For an easy[4] explanation of how the FFT works with maths to actually back it up, check out this link: https://jakevdp.github.io/blog/2013/08/28/understanding-the-fft/, for I am going to give a more hand wavey explanation so I don't have to type out all the equations.

If you assume the dimensions of your matrix are a power of two, like in our 4x4 matrix, you can see that a lot of the elements of the DFT transformation matrix are the same. In order to see this further, this is the 8x8 transformation matrix:[5]

$$\begin{pmatrix} 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & \sqrt{2}\left(\frac{1}{2}-\frac{\mathrm{i}}{2}\right) & -\mathrm{i} & \sqrt{2}\left(-\frac{1}{2}-\frac{\mathrm{i}}{2}\right) & -1 & \sqrt{2}\left(-\frac{1}{2}+\frac{\mathrm{i}}{2}\right) & \mathrm{i} & \sqrt{2}\left(\frac{1}{2}+\frac{\mathrm{i}}{2}\right) \\ 1 & -\mathrm{i} & -1 & \mathrm{i} & 1 & -\mathrm{i} & -1 & \mathrm{i} \\ 1 & \sqrt{2}\left(-\frac{1}{2}-\frac{\mathrm{i}}{2}\right) & \mathrm{i} & \sqrt{2}\left(\frac{1}{2}-\frac{\mathrm{i}}{2}\right) & -1 & \sqrt{2}\left(\frac{1}{2}+\frac{\mathrm{i}}{2}\right) & -\mathrm{i} & \sqrt{2}\left(-\frac{1}{2}+\frac{\mathrm{i}}{2}\right) \\ 1 & -1 & 1 & -1 & 1 & -1 & 1 & -1 \\ 1 & \sqrt{2}\left(-\frac{1}{2}+\frac{\mathrm{i}}{2}\right) & -\mathrm{i} & \sqrt{2}\left(\frac{1}{2}+\frac{\mathrm{i}}{2}\right) & -1 & \sqrt{2}\left(\frac{1}{2}-\frac{\mathrm{i}}{2}\right) & \mathrm{i} & \sqrt{2}\left(-\frac{1}{2}-\frac{\mathrm{i}}{2}\right) \\ 1 & \mathrm{i} & -1 & -\mathrm{i} & 1 & \mathrm{i} & -1 & -\mathrm{i} \\ 1 & \sqrt{2}\left(\frac{1}{2}+\frac{\mathrm{i}}{2}\right) & \mathrm{i} & \sqrt{2}\left(-\frac{1}{2}+\frac{\mathrm{i}}{2}\right) & -1 & \sqrt{2}\left(-\frac{1}{2}-\frac{\mathrm{i}}{2}\right) & -\mathrm{i} & \sqrt{2}\left(\frac{1}{2}-\frac{\mathrm{i}}{2}\right) \end{pmatrix} \tag{12}$$

The reason so many of these values are the same, is because the way they are calculated is by traversing the unit circle.

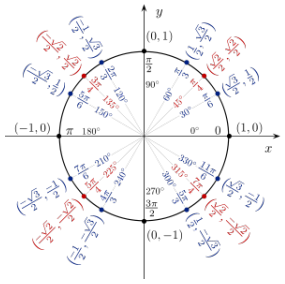

Figure 4: The unit circle.

If a matrix has 2^x sides, then the points on the unit circle are evenly distributed, and take on largely the same values, so if we can somehow find a pattern with these, then we can calculate them only once, rather than multiple times, and save a lot of time. The Cooley-Tukey FFT algorithm splits large DFT's recursively into many many much smaller DFT's which can be solved much faster.

[3] This is a very rough estimate.

[4] It's still really difficult, just easier than anything else I've read.

[5] Check out the 'latex' function in Matlab, it lets you create the latex equations for huge matrices like this in one line of code.

Practical Applications of DFTs

What are some real life practical applications of this? We'll talk about that in another tutorial.