The Fast Fourier Transform

In this document:

'Complex' means having many parts

Complex means complex numbers

The fast fourier transform (FFT) is a specific implementation of a discrete fourier transform (DFT) which is a special case of fourier tranforms (FT). The name FFT is usually used to refer to a popular method, called Cooley-Tukey, for calculating a DFT on a computer very fast. The number of sources that have called this method the most important (excuse the overused term for a lack of an equally specific one) algorithm in computing or in the last century is too large for me to track.

As mentioned in the 'Time and Frequency Domains' chapter, fourier transforms enable EEG and ERP data to be broken down into their component frequencies. Power spectral densities are also estimated by taking the square of fourier transforms. Power spectral densities are calculated by the '.plot_psd()' method in python, which is explained in the 'Filtering EEG Data' chapter.

Fourier transform is a method for breaking up complex 'shapes' into a series of sine waves. These shapes can take many different forms. Some examples include EEG signals and audio files.

https://youtu.be/r6sGWTCMz2k?t=34

Sine waves are naturally occurring patterns. The tip of an oscillating string creates a sine wave in time such as in this video.

https://www.youtube.com/watch?v=PhvJcVDuJsY

Albeit the oscillation may be damped by the force of drag. Sine waves are found in domains other than a spring. The height of a point traced along a circle also creates a sine wave.

https://www.youtube.com/watch?v=Ohp6Okk_tww

This is an important point. In the previous example the shape of the graph linearly increases in time (i.e. 5m for 5s, 10m for 10s). However, in this example, the tracing can go on forever and the point would still be on the circle. This creates a periodicity/symmetry such as going through the same point once every period of the oscillation. Symmetries in math are… let this video tell you what they are:

https://youtu.be/8yMJI918ub4?t=15

I will visit this point later in the document.

Sine waves come in different frequencies, with more or fewer 'hills' per time, and in different amplitudes. Many 'complex' shapes including recorded sound and EEG waves and even outlines of objects can be drawn by adding different frequencies, amplitudes, or starting points (all the way up, down, or any point in between) of sine waves. Just like how one sine wave can be represented by a

single circle, many sine waves can be represented by many different circles drawn by pendulums with many arms.

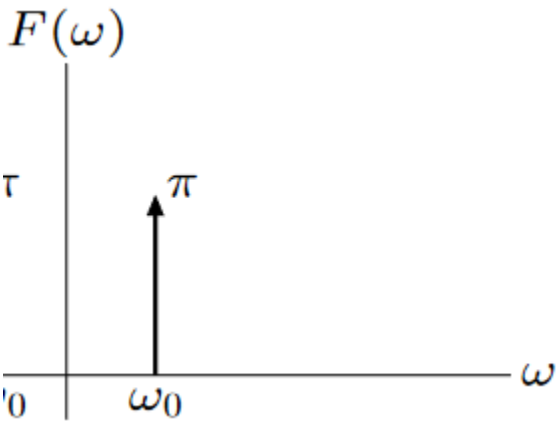
https://youtu.be/ds0cmAV-Yek?t=225

https://www.youtube.com/watch?v=-qgreAUpPwM

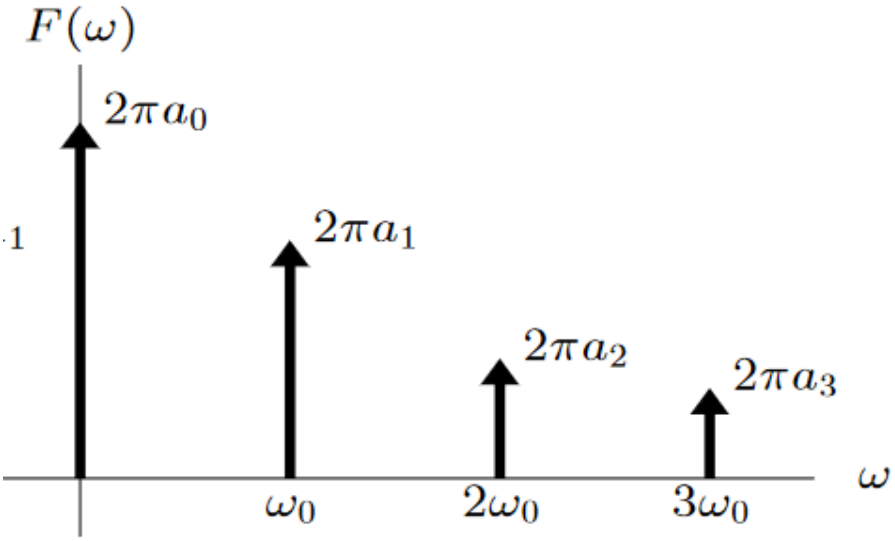
The fourier transform takes a single wave made up of series of sine waves and breaks it down into its constituent frequencies and amplitudes.

https://youtu.be/spUNpyF58BY?t=520

For a pure sine wave, it looks like this ('w'/omega is frequency):



For many frequencies:



https://web.stanford.edu/class/ee102/lectures/fourtran

The fourier transform produces these outputs when a predefined series of sine waves is passed to it. This is not very useful in a real-life scenario like signal analysis where the constituent frequencies and amplitudes of a 'complex' signal are not mathematically defined unlike the above examples. The examples are also different from a real-life scenario in that in real life, raw recorded data does not come in elegant mathematical equations but as discrete time series data.

This is where the discrete fourier transform comes in. As the name suggest, it takes discrete time series data as input and outputs a frequency vs. amplitude plot, albeit with more noise than the elegant delta functions (arrows) above.

It works by separating the time series into points on a circle. Assume a constant sampling rate for the rest of this document. Each 'rotation' of the circle can contain any number of data points placed in equally spaced pie slices. These slices can have different lengths, which represents the value of the data point.

Remember values of sine waves are on a circle as time goes to infinity so all values of a time series plotted on a circle over time can be represented by sine and cosine waves. Cosine waves are sine waves shifted to the left by a quarter period and they follow the same principles as sine waves.

Thanks to 18th century Mathematician Leonhart Euler, points on a circle can be represented with a single number instead of two. This number is a complex number and it can be separated into two components that can represent the two dimensional surface that a two dimensional circle is drawn in.

https://www.youtube.com/watch?v=v0YEaeIClKY

Going back to the pie slices, the DFT plots all values in a time series on a circle as pie slices (think of a spiral staircase) and finds the center of gravity/mass of all the slices and represents it as another slice. The DFT repeats this process for different frequencies, i.e. different number of data points that take to complete one rotation around the circle. Repetitions yield different averaged slices of different lengths.

Each of those slices have a frequency value associated with them. When the lengths and associated frequencies of these slices are plotted, they create a graph in the 'frequency domain'. This frequency domain graph shows the constituent frequencies of a discrete time series.

This is the end goal of a DFT. It can be calculated with for loops or with 'matrix multiplication', but they are both very computationally expensive. This method involves multiplying the time series data with 'n' elements represented as a column of values of shape 'n x 1' by a precomputed table of size 'n x n'. This multiplication is called a matrix multiplication and it involves multiplying the time series column with each row of the 'n x n' table and summing the values of each row x column multiplication and storing each value as an entry in a new 'n x 1' column. This series data is the DFT. When plotted it gives the graph in the frequency domain mentioned above.

Note that this approach uses n^2 calculations. Every value in the 'n x n' table is multiplied with a value in the time series for $ n x n = n^2 $ calculations. I will visit this point later.

The values in the 'n x n' table are special values. They are complex numbers that correspond to different locations on a circle and they represent the time point of a data point in a time series.

https://www.youtube.com/watch?v=M0Sa8fLOajA

This approach is still very computationally expensive. The FFT can calculate DFT's faster by a factor of 2n/log_2 (n), which is

= 2*n/np.log(n)*np.log(2)

in python, where 'n' is the number of data points in a time series. A 5 minute = 300 second song at a common 44.1 kHz has 44100*300 data points. The FFT calculates the FT of such a 5 minute song 1,120,000 times faster. That is 13 days long if it takes 1 second to calculate an FFT.

The FFT is a recursive loop. Turns out, the 'n x n' table is equivalent to the matrix multiplication of three tables.

(1) The most important one is a 'n x n' table divided into four equal quadrants, where top right and bottom left quadrants all have zero values and the top left and bottom right quadrants are both equal to the top left quadrant, [0 : n/2, 0 : n/2], of the original 'n x n' table. This equality can be applied recursively to the non-zero top left and bottom right quadrants until n=1.

(2) The second table separates the constituent sine waves of the time series into two types of symmetry across zero. One where the waves are symmetric (e.g. sin(5) = sin(5)), the other where the waves are symmetric to their negatives (e.g. sin(5 + pi/2) = -sin(- (5 + pi/2)))). Based on the values of this table, computers take the negative of some values which can be done instantaneously.

(3) The third table has special entries and adds n/2 calculations to the total number of calculations a 'n' count time series requires.

After one iteration of this process n^2 calculations go down to $ 2*((n/2)^2) + (n/2) $ calculations. The first term comes from (1) and the second term comes from (3). When this process is applied recursively, the first term becomes 2*2*((n/4)^2 + n/4). Applying this until n=1, the sum becomes n*log_2 (n).

Now that I explained what FFT is, its time to see it in action in the jupyter notebook in the repository.