# 6 Oct 23 - Notes: Fast Fourier Transform

It might not have been obvious to you, but our work so far has expected that our be a continuous function of time, and that we can sample it any time we want. In reality, you are limited to the response of your equipment, which leads to a phenomenon knows as the [Nyquist-Shannon sampling theorem](https://en.wikipedia.org/wiki/Nyquist%E2%80%93Shannon_sampling_theorem). This theorem states that if you want to study a signal with a frequency , you need to sample it at least times per second. This is known as the [Nyquist frequency](https://en.wikipedia.org/wiki/Nyquist_frequency). If you don’t sample at least this often, you will get aliasing, which is when a signal at a higher frequency is indistinguishable from a signal at a lower frequency. As you can tell, there’s a lot of issues with sampling when it comes to studying signals.

## The Fast Fourier Transform

This video from Veritasium really explains the importance of this algorithm. It’s a great watch.

[](https://inv.tux.pizza/watch?v=nmgFG7PUHfo)

* Non-Commercial Link: <https://inv.tux.pizza/watch?v=nmgFG7PUHfo>
* Commercial Link: <https://youtube.com/watch?v=nmgFG7PUHfo>

## The Discrete Fourier Transform

The foundation of this algorithm is the [Discrete Fourier Transform](https://en.wikipedia.org/wiki/Discrete_Fourier_transform). This is a mathematical operation that takes a discrete signal and transforms it into a continuous function of frequency. The equation for this is:

We start with complex form of the approximation of the Fourier Series:

Where is the complex coefficient of the th harmonic, and is the fundamental frequency. As we have seen before, we can rewrite this as:

Where is the period of the signal. Sometimes that integral is analytical and sometimes it isn’t. If it’s real data, it is definitely questionable if it’s analytical. So, we can approximate this integral using the trapezoidal rule:

Where is the number of samples, is the time of the th sample, and is the time between samples. We then use this to compute the approximate signal:

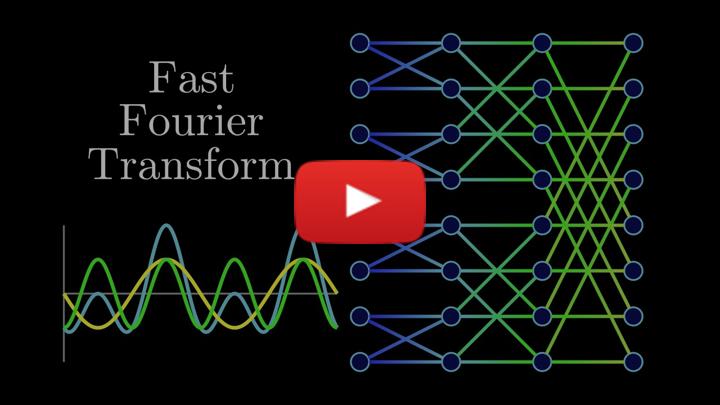
It’s great that the numerical technique is well known and canned tools are available like scipy.integrate.trapz, but here we will start to use scipy.fft to do the heavy lifting for us.

## Additional Resources

We do not expect you to understand the details of an FFT, how it is derived and how it works when written into a computer program. However, if you are interested, here are some resources that you can use to learn more about the FFT.

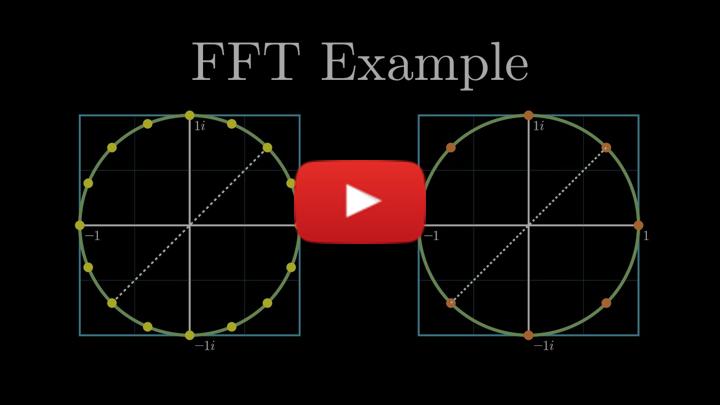
### Videos

This is an excellent discussion of the underlying mathematics and the setup of the algorithm.

[](https://inv.tux.pizza/watch?v=h7apO7q16V0)

* Non-Commercial Link: <https://inv.tux.pizza/watch?v=h7apO7q16V0>
* Commercial Link: <https://youtube.com/watch?v=h7apO7q16V0>

This is a presentation of how to implement the FFT using Python code.

[](https://inv.tux.pizza/watch?v=Ty0JcR6Dvis)

* Non-Commercial Link: <https://inv.tux.pizza/watch?v=Ty0JcR6Dvis>
* Commercial Link: <https://youtube.com/watch?v=Ty0JcR6Dvis>

### Handwritten notes

* [Introduction to FFT](../assets/notes/Notes-Introduction_to_FFT.pdf)