Fast Fourier Transform Computational Assignment

(Alex) Xi Zeng
1007099373
2023/01/20

Introduction

The objective of this lab is to inform and show us how to use Fast Fourier Transform (FFT) in Python and how powerful it is by allowing us to go through multiple scenarios where we need to filter out sets of data from different kinds of noise. FFT is important in today's digital world for its wide use across multiple fields, such as signals or audio processing, and its fast computational speed.

Methods & Analysis

Section 1:

The diagram below shows what would happen if you add two wave equations with different amplitudes and frequencies together. The ones on the left are the time domain graphs, whereas the ones on the right are the (absolute value of the) FFT of the functions on their left, also known as the frequency domain graphs. For both wave equations, we are taking 200 data points. The formulas of the two wave equations are:

$$y_1 = 5\,sin(2/20\pi t) \quad and \quad y_2 = 3\,sin(2/13\pi t)$$

When two waves are added together, you can see that on the left side, the amplitudes of the waves are adding on top of each other to form constructive and destructive interference. On the right side, since the two graphs have two different but similar frequencies, when added together, the two spikes of different frequencies are placed near each other on the same graph without much interference.

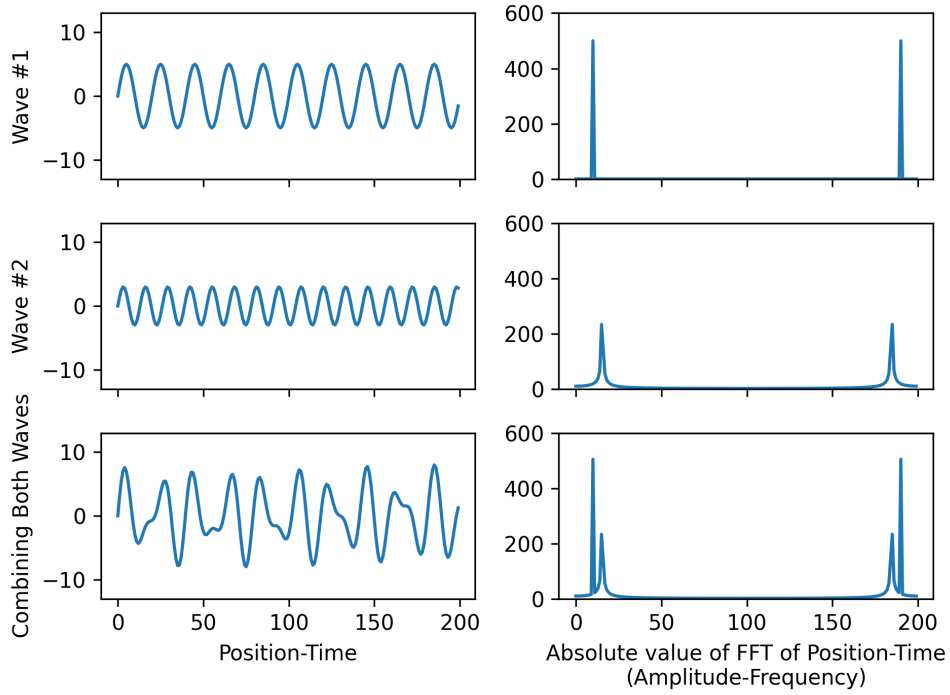

Figure 1: Adding two different waves together

FFT can be very useful in finding the two frequency values we are using. Since the numpy FFT function provides frequency information about the signal by converting a signal into individual spectral components, if we can find which index of those spectral components is the longest, then by substituting that index into the numpy.fft function fftfreq(), we can find the frequencies we need since the fftfreq() function outputs an array the same size as the FFT that tells us the correlated frequency of each index from the FFT.

To find the amplitude, we can use the equation state in the given instruction that $amplitude = (2/N) * np.abs(data\_fft[index])$, where data_fft is the FFT function of the signal and the same index we used in the last paragraph to find the frequency.

Section 2:

This section aims to reduce the noise of this function by adjusting the parameters of the filter function to improve the result of the inverse FFT. More specifically, we are adjusting the filter function's width and peak. As observed from the FFT graphs at the bottom-right of Figure 1, there are two long spikes indicating the frequencies that we should keep by using it as the peak

parameter. Using a simple code $np.argmax(waveFFT(z2[0:100])$, we can locate the x-value of the first spike that's closer to 0. Once we can find the exact spot of the peak, we can make the width as small as possible before it starts disturbing the graph.

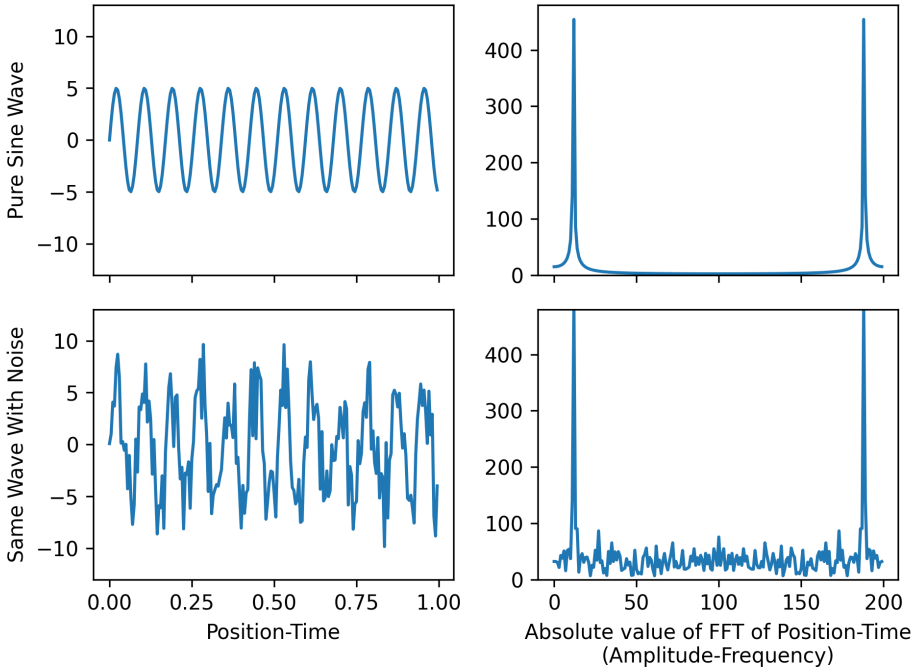

Figure 2: Section 2 - Single Wave And Noise With FFT

Using the peak and width parameters, we can create a Filter Function to isolate the noise of the original wave equation. To do that, all we need to do is to multiply the original Noisy FFT wave function with the Filter function. By doing so, all the important parts we want to keep using the parameter will remain in the graph, and everything else that's unimportant will become much smaller or even 0.

Since we had access to the original data function with critical information such as period and frequency, it saved us a lot of time that would've been spent on trials and errors to find out the critical information that we could use to find any peak parameter.

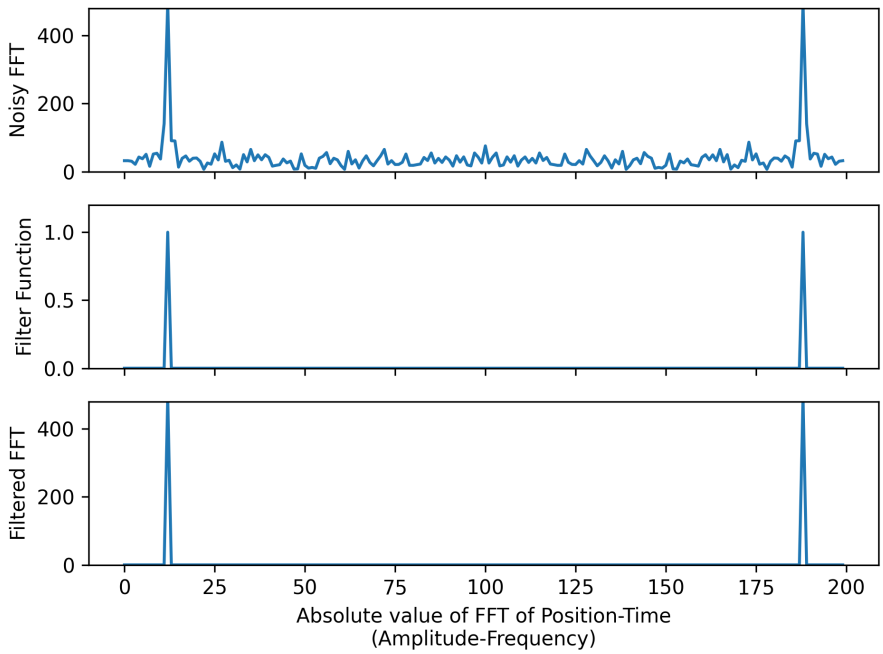

Figure 3: Section 2 - Filtering Process

Figure 4 shows the result of applying the Filter Function to the Noise FFT in the time domain. To get this amazing final Filtered Data, we filtered the FFT of the original noisy function first and then took the inverse FFT of that filtered function. As you can see, the now filtered data is almost identical to the Ideal result, especially compared to the Original Data. The Filtered Data is much smoother and can show all the clear ups and downs of the Ideal Functions while not having a noticeable phase shift.

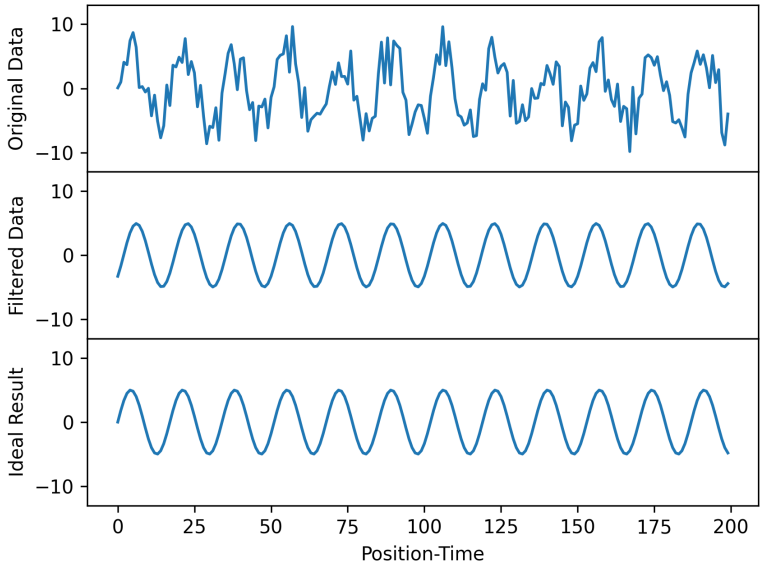

Figure 4: Section 3 - Original, Filtered and Ideal Waves Comparison

Section 3:

Section 3 differs from Section 2 because here, we do not have any information about the Ideal Result, so we will need to make educated guesses on whether our Filtered Functions are good or not.

Figure 5 and Figure 9 are the Position-Time graphs of the original data provided to us. Figure 5 is the shorter version with only 300 data points, and we will be using it to compare if the Filter Function is working since it is easier to see the patterns and trends with fewer data points. Figure 6 is the full version with 2000 data points, and this is the one that we will be using to do all the calculations.

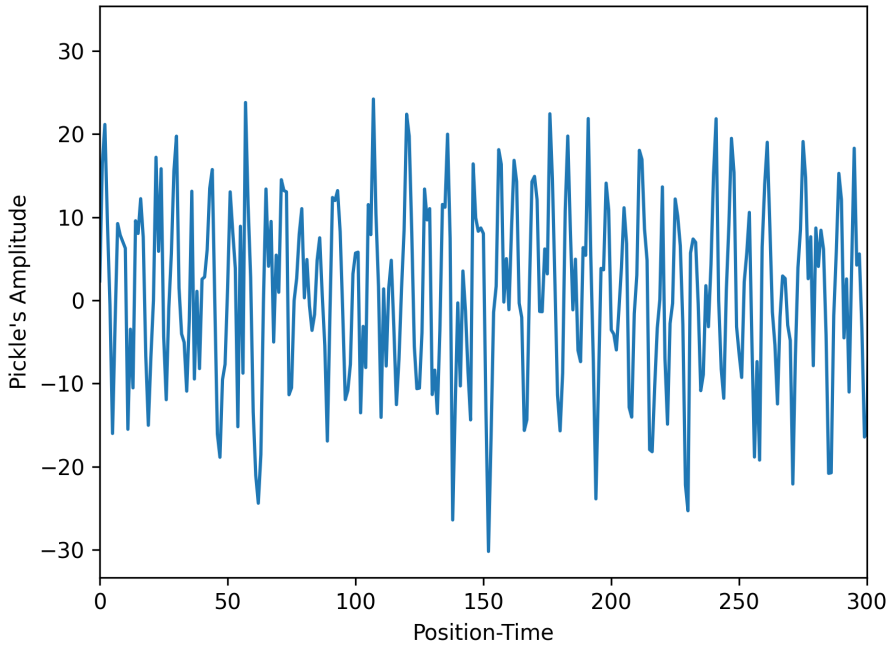

Figure 5: Section 3 - Provided Data Short

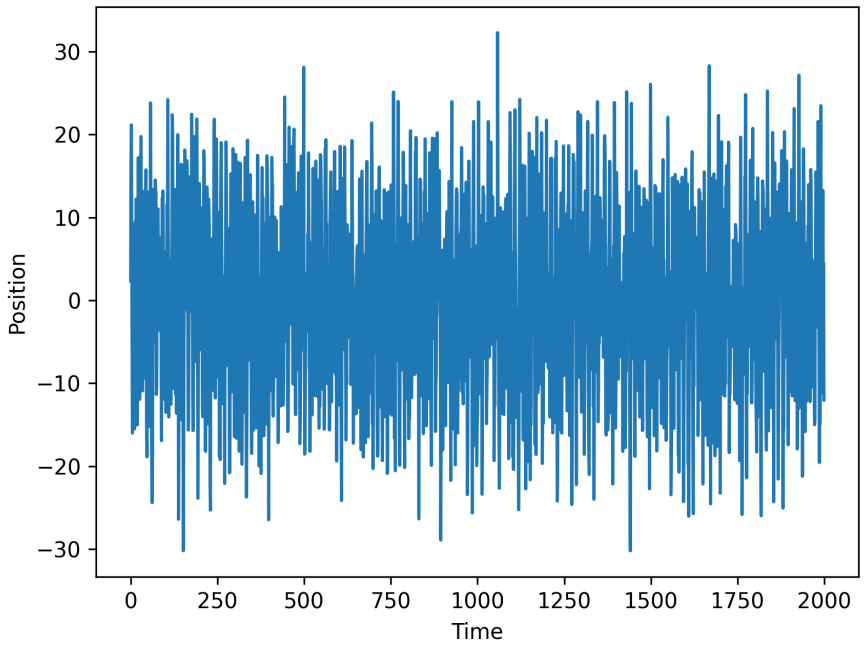

Figure 6: Section 3 - Provided Data Full

Figure 7 shows the FFT of the original function in the frequency domain. As we can see in the diagram below, there are three sets of different frequencies that stand out from the rest. Thus we will be using three Gaussian filters with these three different peaks to filter the data. The next step would be finding the peak and width for each Gaussian Filter Function so that we can all three together to generate the final Filter Function.

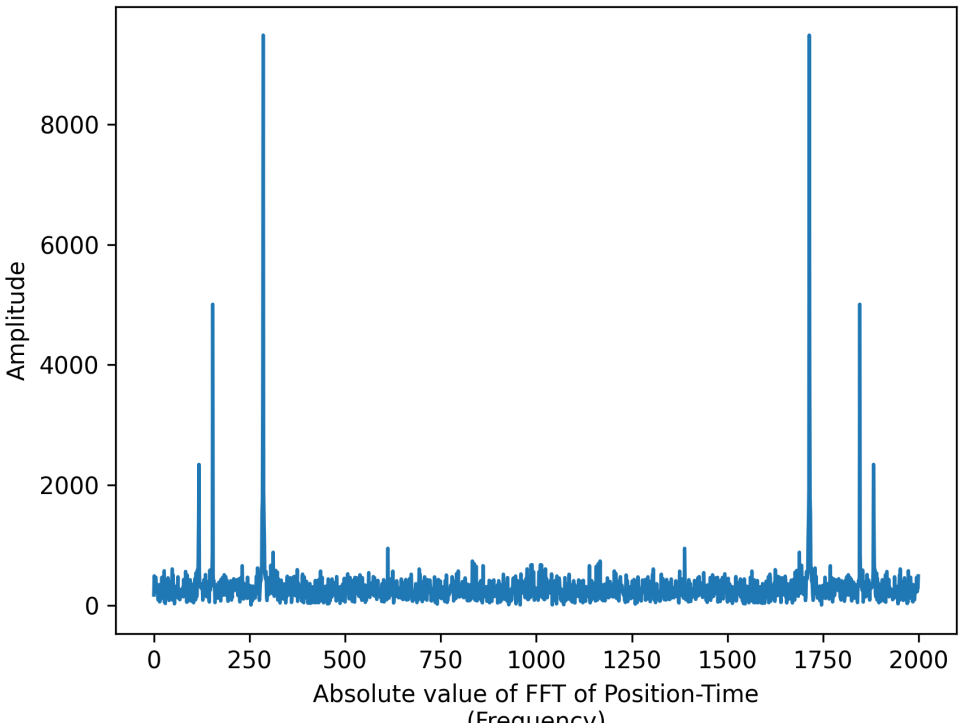

Figure 7: Section 3 - Single Wave And Noise Full Length With FFT

Repeating the same steps mentioned earlier in Section 1 and Section 2, we will first need to find the x-axis peak for each of the three frequencies we want to use. Once we calculate the accurate peak for each frequency, we can make the width as small as we can until we are satisfied before it distorts the wave functions. Now substitute this into the filter function equation, and we will get the middle graph from Figure 8. Then multiplying the middle Filtered Function by the top Noise FFT, we will get the bottom Filtered FFT graph.

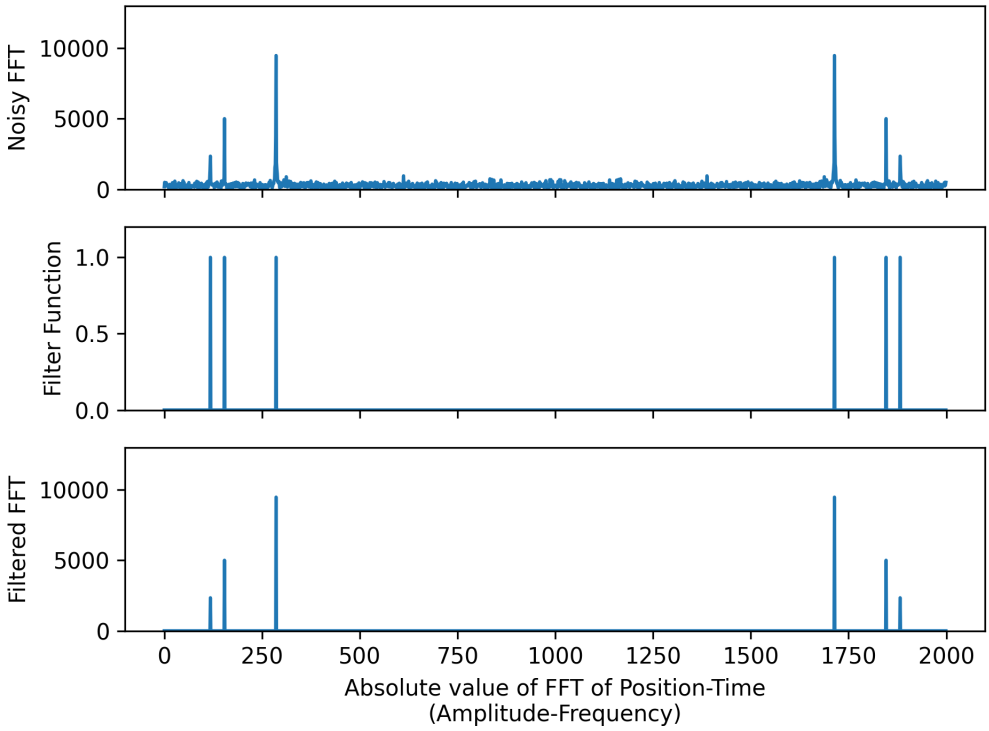

Figure 8: Section 3 - Filtering Process

Just like Section 2, Figure 8 only shows the filtered data in the frequency domain. To see the result in the time domain, we will need to repeat the same process as Section 2, where we first filter the FFT of the original noisy function and then take the inverse FFT of that filtered function to get the middle Filter Data graph in the middle of Figure 9.

The bottom of figure 9 is the Theoretical Data, where we estimated how the graph would look by estimating some important parameters, such as the relative amplitudes and frequencies and reconstructing it with all the Gaussian filter functions. The amplitude and frequency calculations are the same as in Section 1. As you can see, the Filtered Data is very similar to the Theoretical Data. Thus we can make the educated guess that our approach should be on the right track to remove the noise completely.

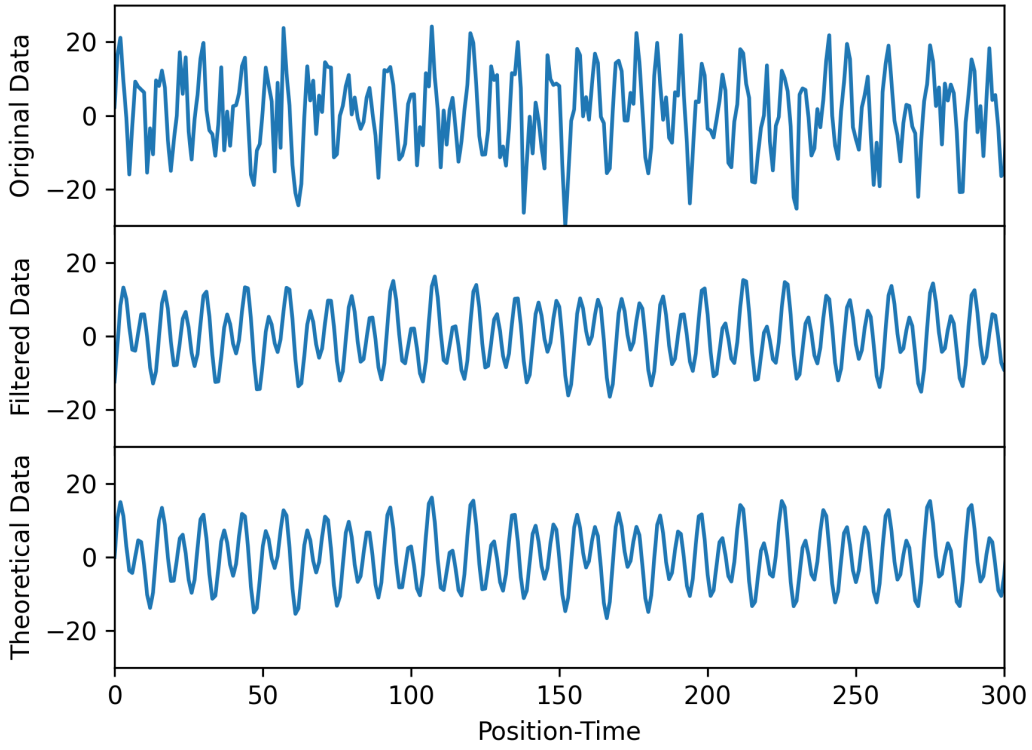

Figure 9: Section 3 - Original, Filtered, and Theoretical Data

Section 4:

This section was told to construct a random linear function of time for amplitude and combine it with a sine wave to form a position time graph. Thus, I made the amplitude function to be $\omega(t) = t/400$ since the question also specified they want the value to change by about 25% throughout 100 oscillations. Because of my choice of the amplitude function, the function with a time-dependent frequency is going to be $y = Asin(\omega(t)t)$, where A can be any constant I choose to be the amplitude. Let A=1, and take 100 data points, we will get the following graph shown in Figure 10. Converting it into its FFT graph, we will get what's shown in Figure 11.

What's shown in Figure 11 is interesting. Since the amplitude is proportional to the time, unlike any of the previous graphs we've made, this frequency domain graph no longer has the pointy spikes to represent the hand full of frequencies that the

time domain graph contains. Instead, you can see that all frequency is covered, but some have a higher density than the rest, and no frequencies have an amplitude of 0.

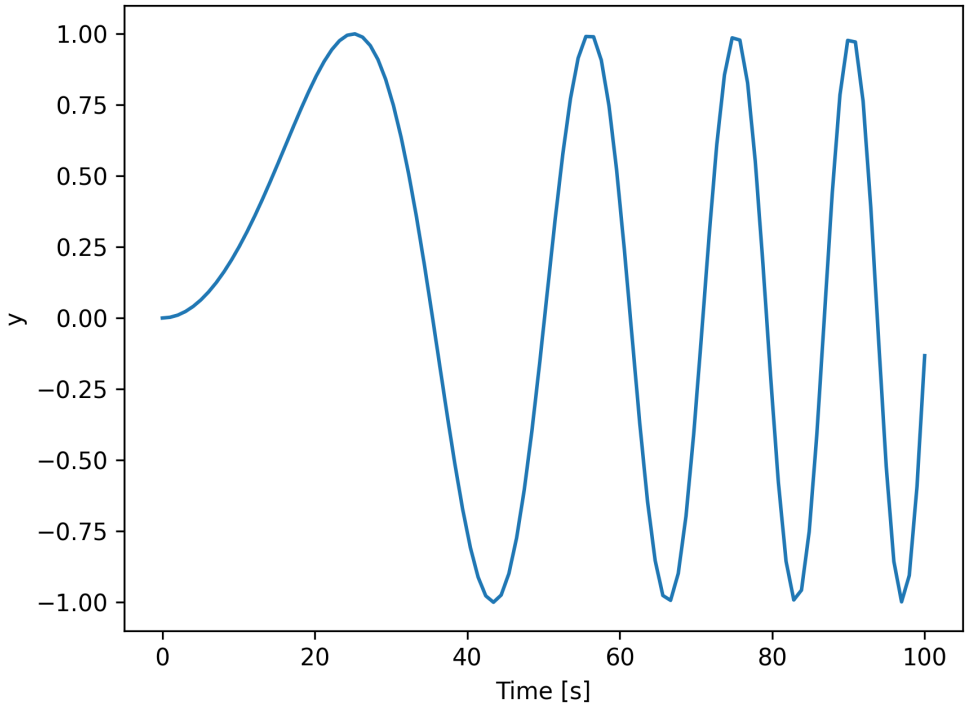

Figure 10: Section 4 - Position-Time Graph of a Time-Dependent Frequency

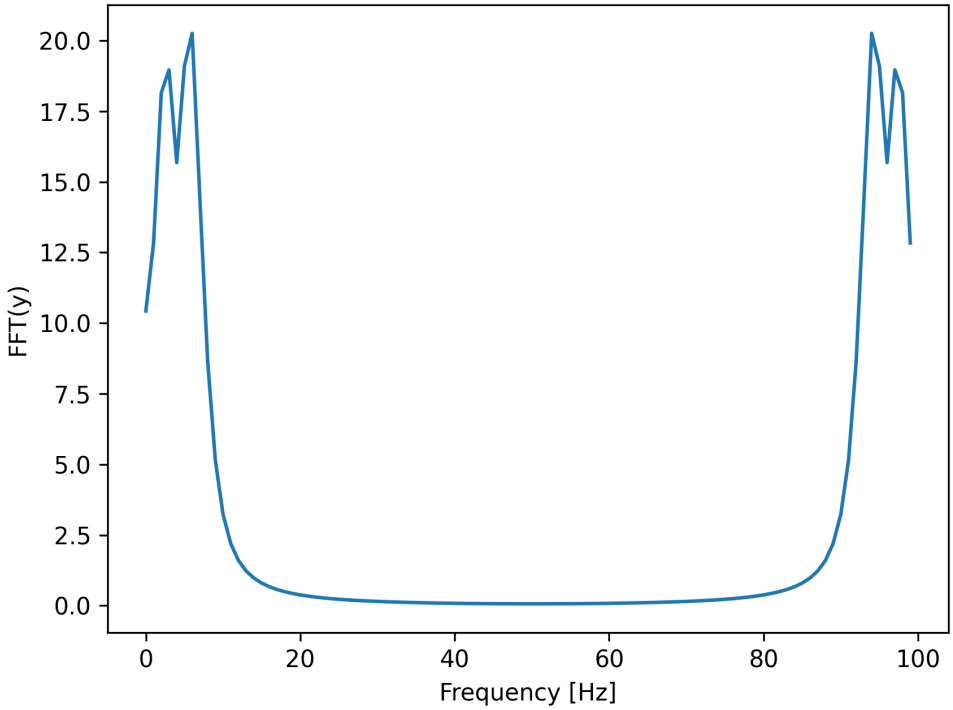

Figure 11: Section 4 - Amplitude-Frequency Graph of a Time-Dependent Frequency

Conclusion

After learning and applying FFT in multiple scenarios and seeing how it can transform a noisy signal into something much cleaner, we can conclude that FFT works and is a useful skill set for today's age.

Appendix

- All of my code can be found at: https://github.com/AlexZengXi/PHY324_Physics_Practical_2
- All math/formulas are given/inspired from the given PDF instruction: http://www.upscale.utoronto.ca/PVB/Harrison/FourierTransform.pdf