PHY324 Computational Assignment

Fast Fourier Transforms of Signal Data

Jace Alloway - 1006940802

Introduction

Rigorously defined by Joseph Fourier in 1822, the Fourier transform (FT) is an analytical computation which, when applied to any function, extracts the frequency spectrum of the respective function in frequency space. This is most evidently seen when the input function is periodic, such as a sinusoid. These functions may or may not be complex. Upon creation of the computer, discrete-time Fourier transforms (DTFT's) and fast Fourier transforms (FFT's) were created to apply the Fourier transform algorithm on a finite set of discrete points, such as an array of data. Refining the data by the use of sampling produces better approximations for the true frequencies of the input signal. FFT's are now widely used in radio telecommunication, image compression, circuit analysis, and more.

Theory

For any real-valued continuous function f, the FT of f, denoted by $\overline{f}$, is given by

$$\overline{f}(k) = \frac{1}{\sqrt{2\pi}} \int_{-\infty}^{\infty} f(x) e^{-ikx}\, dx. \tag{1}$$

Similarly, the inverse Fourier transform can be used to extract an original signal from a function of frequency:

$$f(x) = \frac{1}{\sqrt{2\pi}} \int_{-\infty}^{\infty} \overline{f}(k) e^{ikx}\, dk. \tag{2}$$

The above expressions are utilized only for continuous functions f and $\overline{f}$. By converting the integral into a discrete sum, we define the DFT (or FFT) of a discrete set of N data points, which may be real or complex, as

$$y_k = \frac{1}{\sqrt{2\pi}} \sum_{n=0}^{N-1} x_n e^{-ikn/N}, \tag{3}$$

where y_k are points located in frequency space, x_k in position space, for $k = 0, \ldots, N-1$. The inverse DFT is similarly defined by exchanging the y_k and x_k, and inverting the minus sign in the exponential to a positive.

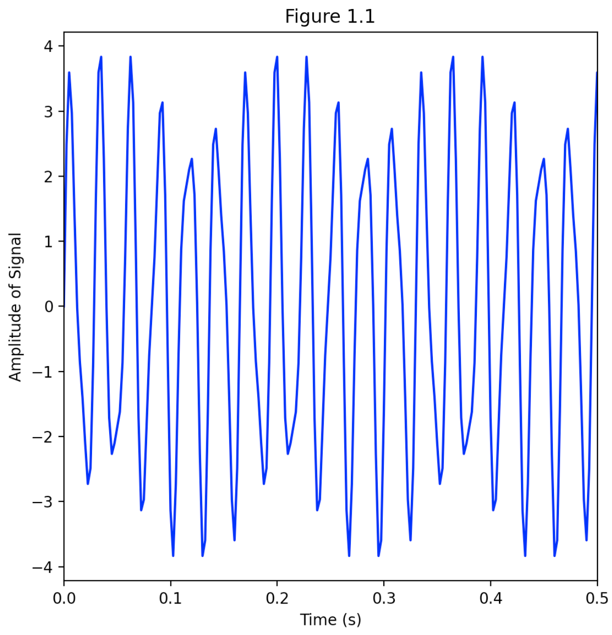
In signal filtering, FFT's are applied to a set a data points which retrieve the primitive frequencies of the signal. Using Gaussian functions $g(x) = e^{-a(x-b)^2}$, the noisy frequencies can be filtered, which by applying an inverse FFT yields a clean signal.

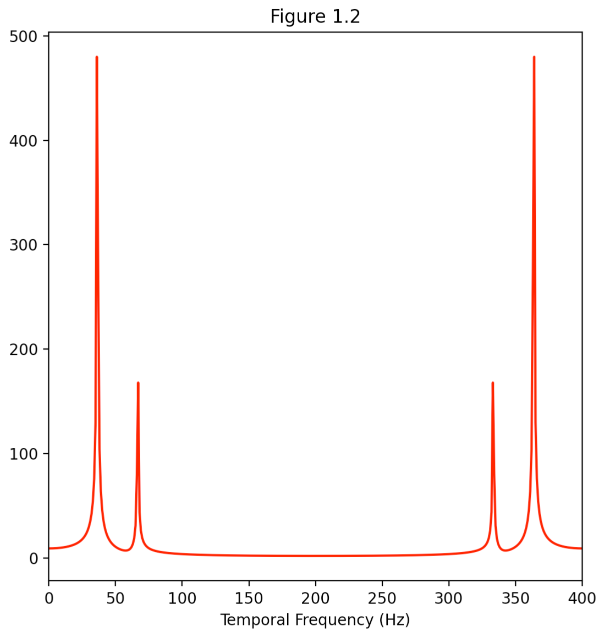
Exercise 1

The goal of this exercise is to superpose two sine waves of two different amplitudes and frequencies, then invoke the Numpy FFT algorithm to extract from the input signal the two frequencies utilized in the superposition. In this instance, I have selected the period values of the two sine waves to be $T_1 = 6$ and $T_2 = 11$, and I defined the superposition of the two waves in the function

$$f(t) = \sin\left(\frac{2\pi t}{6}\right) + 3\sin\left(\frac{2\pi t}{11}\right).$$

Here, the variable t represents time, which as defined by the 'arange' function, establishes an array of integers from 0 to the specified number of samples, which I chose as 400 for this exercise. Hence f is the array of numbers whose values are $f(t)$, for t in the time array. Using the 'numpy.fft.fft' Fourier transform algorithm on the array f, a new array g is obtained which contains the transformed data. Both sets of data can be plotted in terms of magnitude - time (s) and magnitude - (temporal) frequency (Hz):

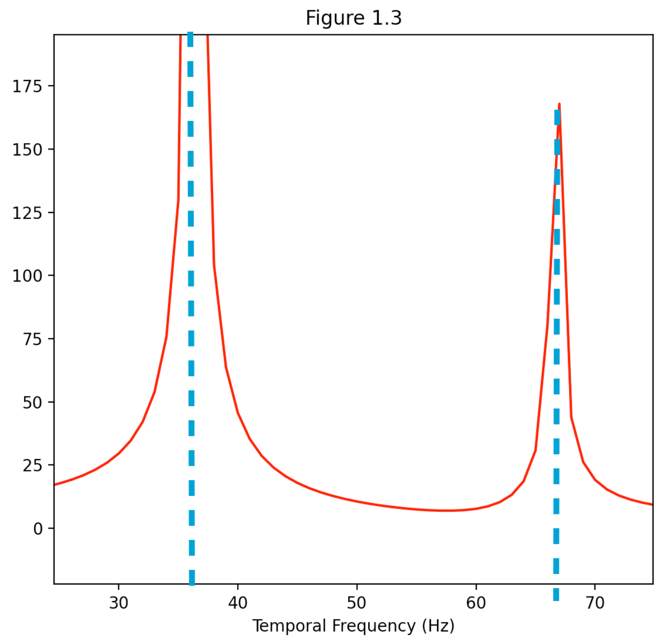




[Figure 1.1 (Left)] The graphical representation of the superposition of two sine waves of different amplitudes ($A_1 = 1, A_2 = 3$) and respective frequencies ($T_1 = 6, T_2 = 11$).

[Figure 1.2 (Right)] The graphical representation of the Fourier Transform of the superposed sinusoidal signals, of various amplitudes and frequencies.

Consider the plot of the transformed signal (Figure 1.2). The relative amplitudes of each of the signals are easy to extract. Suppose the lowest peak has an amplitude A_1, in which case is located around the $160-$magnitude value. The highest peak has a magnitude value of around $A_2 = 490$, hence $\dfrac{A_2}{A_1} = \dfrac{490}{160} = 3.06$, thus it can be concluded that the relative amplitudes are roughly on a ratio of $1:3$. This is what we had determined from the amplitudes of the superpositions of our two sine signals. Furthermore, by zooming in on the frequency spectrum, one can interpolate the values of the frequency (Figure 1.3).



[Figure 1.3] The graphical representation of the Fourier Transform of the superposed sinusoidal signals, zoomed so that the reader may interpolate the values of the chosen frequencies.

These frequency values are approximately read as 36 and 66. By equation (3), one may note that the FFT yields frequency values on the order of n/N, that is, the value divided by the number of samples. Hence $1/T = n/N$, so $n = N/T$, for the period of the wave T. Therefore if our value of n is 36, and $N = 400$ (I took 400 samples), then our period is $T_1 = \dfrac{400}{36} \approx 11.1$, and likewise $T_2 = \dfrac{400}{66} \approx 6.06$, which yield the values of our inputted frequencies: 6 and 11.

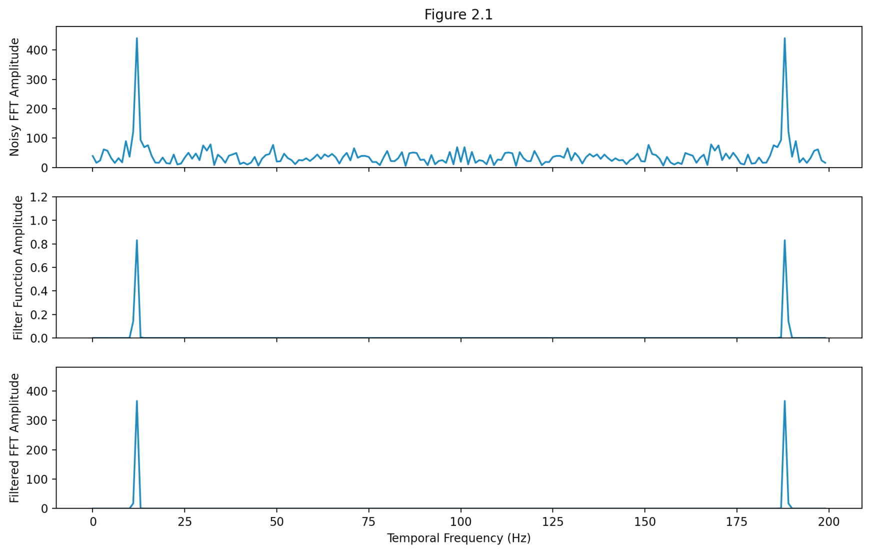
This completes exercise 1.

Exercise 2

The purpose of this exercise is to portray the effect of the filter function on noisy data, and attempt to reproduce any initial signals.

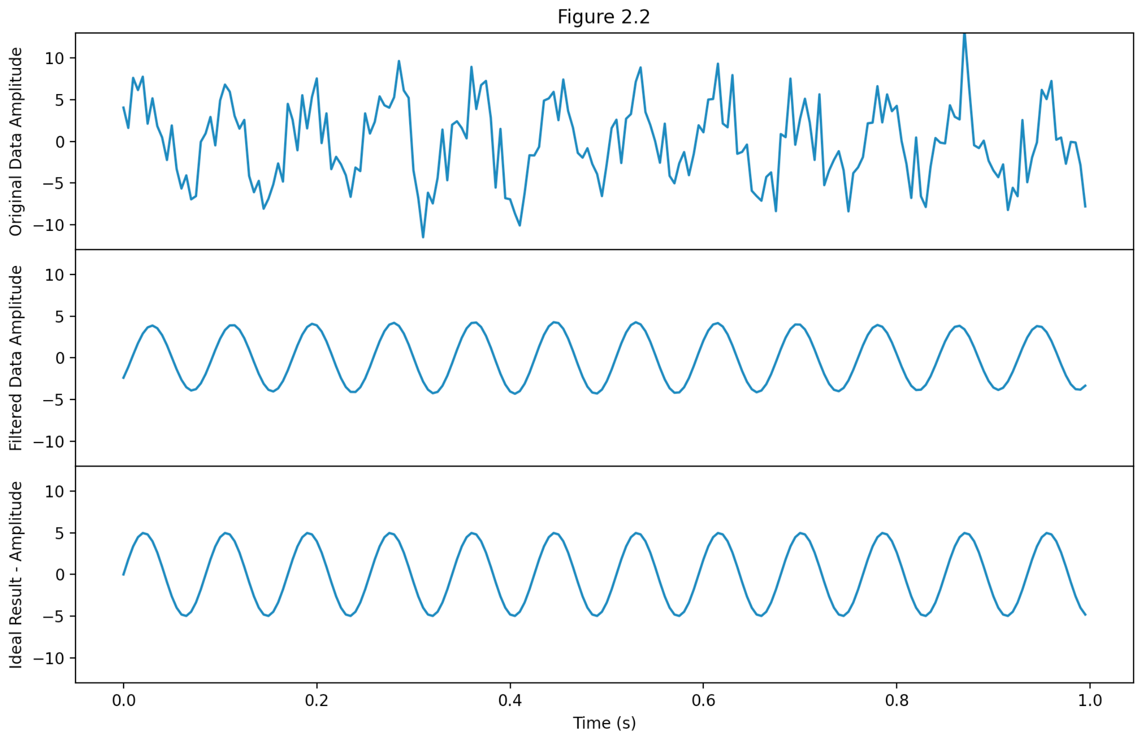
If our initial signal is a sine wave, then we may add random noise to the sine wave with the 'random.gauss' function (or just 'gauss'), whose primary inputs are the mean value and the standard deviation. In this exercise, the standard deviation was set as $A/2$, where A is the amplitude of the sine wave. It may suffice to explain what exactly this 'gauss()' function does, which appends an array of random numbers centered around the mean value, which conform to the Gaussian distribution with standard deviation $A/2$. Adding this noise to the sinusoid yields a noisy signal.

In the python file, the initial sine wave amplitude was set to $A = 5$ with period $T = 17$, so the noise amplitude is $\sigma = 5/2$. To filter the noise, we apply Gaussian filter functions centered at the peaks of the frequency spectrum, given by N/T (as described in exercise 1). Applying 200 samples to the data yields the peak location to be $200/17 \approx 11.8$, which is what I set the mean location to in the filter function. The ideal width of the filter function should be significantly less than that of the standard deviation noise amplitude, because this is what will filter out the noise values which clutter the surrounding peak. If the standard deviation is approximately $\sigma = 2.5$, I chose the width of the filter function to be $w = 0.3$. These selections produced the following results:



[Figure 2.1] Graphical depiction of the filtering process of Fourier-transformed signals. (Top) A noisy signal which has been Fourier transformed into amplitude-frequency space. (Middle) The depiction of the filter function which will be applied to the noisy data. (Bottom) The result of the product between the noisy FFT data and the Gaussian filter function. Note the location of the peaks: approximately 11.8 Hz , $200 - 11.8 = 188.2$ Hz.

Similarly, one may graphically compare the difference between the result which one gets from filtering the data, with the ideal result, which should be the initial input of the signal prior to the addition of noise:



[Figure 2.2] The visual representation of the effect of filtering a signal. (Top) The primitive signal input with noise, prior to any Fourier transform. (Middle) The result of the inverse Fourier transform being applied to the filtered signal, as depicted in Figure 2.1. (Bottom) The ideal result, or the initial sinusoid signal prior to the addition of noise.

I may draw the readers attention to the importance of the middle and lower plots in Figure 2.2. In fact, to within a small amplitude variation, they are virtually identical. This achieves the goal of exercise 2, since the filtered signal and the initial signal are equivalent.

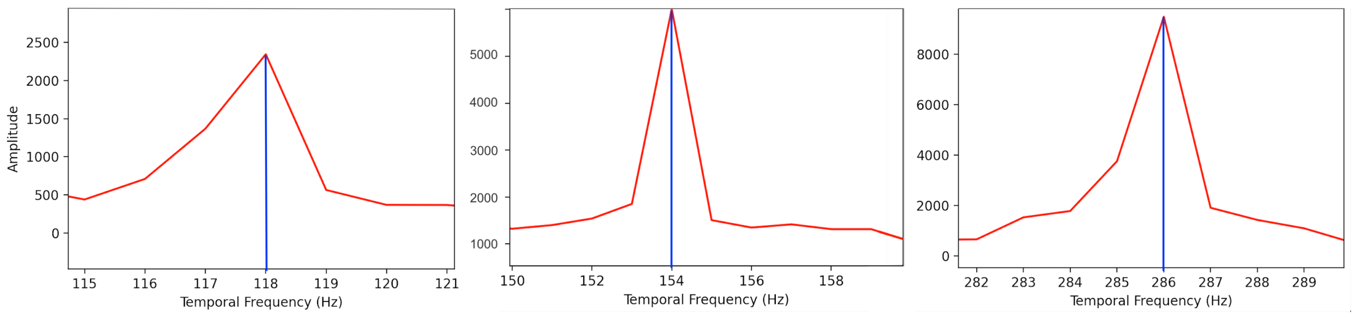
This completes exercise 2.

Exercise 3

The goal of exercise 3 is to reconstruct a noisy data file via the process of Fourier-transforming and filtering the signal. From this we may invoke the process from exercises 1 and 2 to determine the relative amplitudes and frequencies of the signal.

The first step in the analysis is loading the data and determining the number of samples. In the 'noisy-sine-wave' file, $N = 2000$ so there are 2000 total samples. We now determine the peaks of the frequency-amplitude plot by graphing the Fourier transform data, given by 'np.fft.fft'.

By zooming in on the plot, one may visually determine the frequency values: $118, 154, 286$.

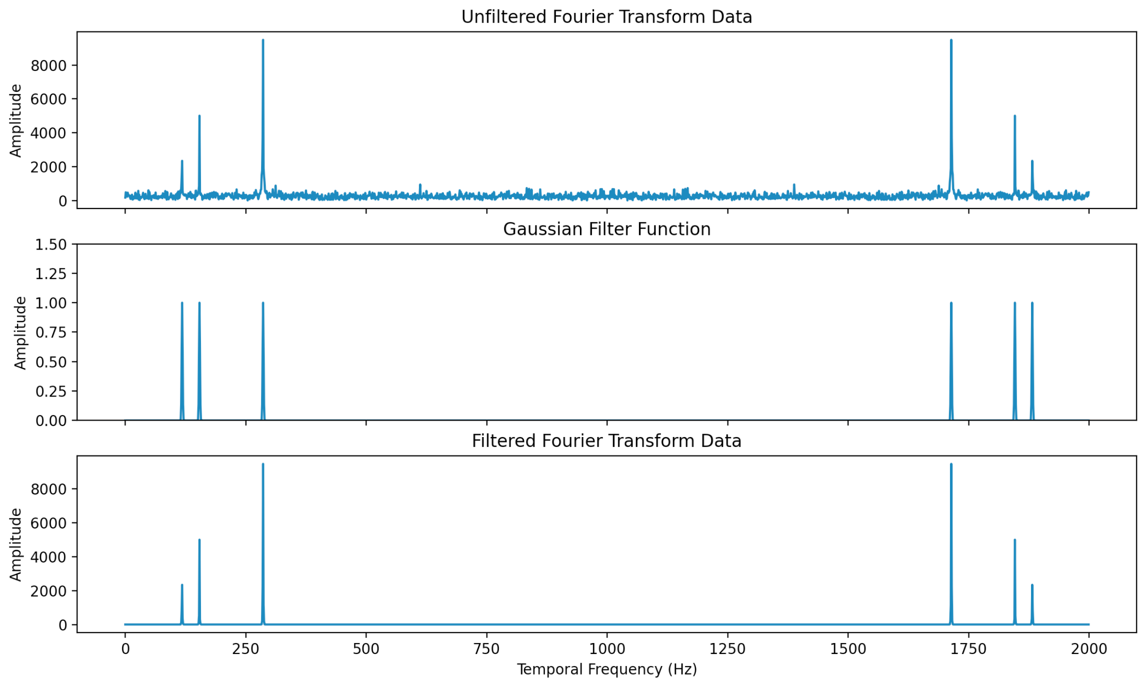


[Figure 3.1] The graphical depiction of prominent frequencies in the Fourier transform data of the 'pickle' file. (Left) The peak centered around 118 Hz. (Center) The peak centered around 154 Hz. (Right) The frequency peak centered around 286 Hz.

From this information, I proceeded by defining the following Gaussian filter function:

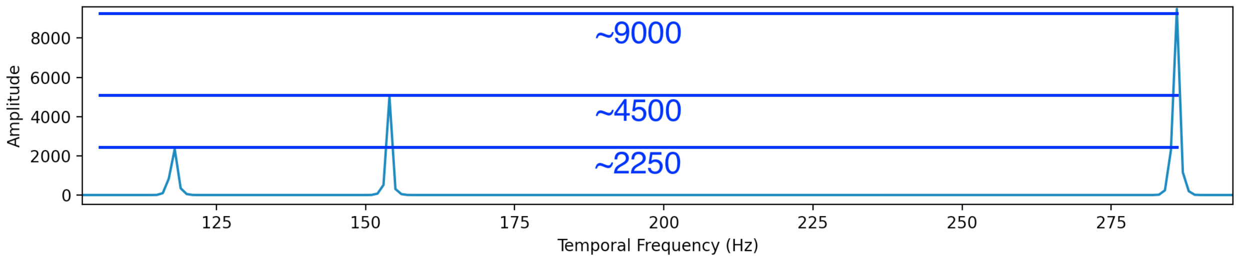
$$\begin{aligned} g(k) = e^{-(k-118)^2/2} + e^{-(k-154)^2/2} + e^{-(k-286)^2/2} \\ + e^{-(k+118-2000)^2/2} + e^{-(k+154-2000)^2/2} + e^{-(k+286-2000)^2/2}. \end{aligned} \tag{3.1}$$

From trial and error, I had discovered that a common width of 2 was most appropriate. The first three terms account for the real components of the FFT data (left hand side of plot), while the last three terms account for the complex Fourier transform terms, which appear when taking the absolute value of the data ('np.abs()'). Then, multiplying the filter function with the noisy transformed data, the filtered data is obtained:



[Figure 3.2] The application of a Gaussian filter function on noisy Fourier-transform data. (Top) Noisy input data, taking by the FFT of the input signal. (Middle) The Gaussian filter function, centered around each of the three peaks of the noisy FFT data. (Bottom) The product of the noisy FFT data with the filter function. Results in filtered data.

Furthermore, it can clearly be seen the relative amplitudes of the peaks of the frequency spikes: 1:2:4.



[Figure 3.3] The relative amplitudes of the filtered FFT data. The highest peak can be seen to have an approximate amplitude of 9000; the second peak approximately 4500, and the smallest peak approximately 2250. These ratios yield the relative amplitudes of the initial input signal to be about $1 : 2 : 4$.

Once the noisy data has been filtered, it just suffices to apply the inverse fast Fourier transform to the filtered data to acquire a cleaned signal. One may compare the filtered signal with the ideal result we would expect to obtain, which can be determined by our frequency and amplitude data.

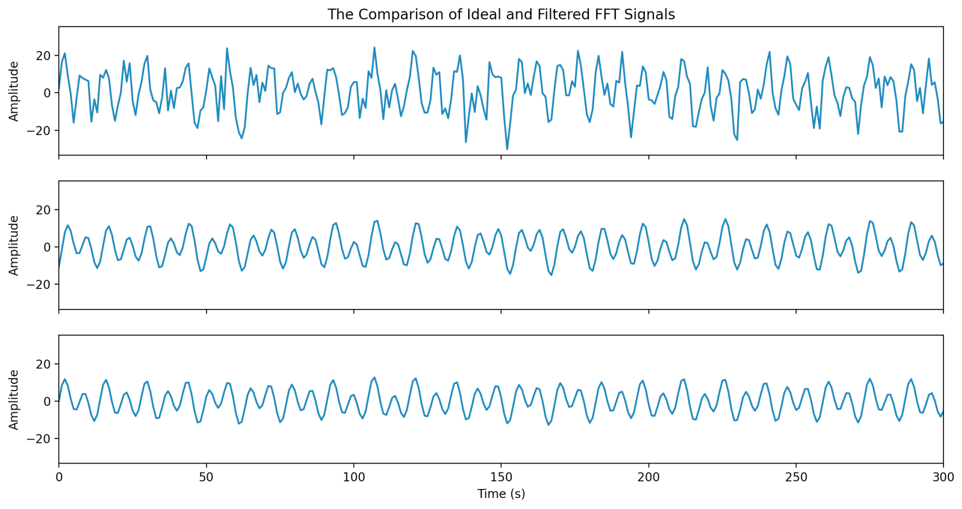
With 2000 samples, and peaks located at $118, 154$, and 286, then taking $N/n = T$ for each of the three peak frequency values yields

$$\frac{2000}{118} = 16.949 \approx 17$$
$$\frac{2000}{154} = 12.987 \approx 13$$
$$\frac{2000}{286} = 6.993 \approx 7.$$

Taking the respective amplitudes to be $1, 4$ and 8, I propose the ideal signal to be given by the function

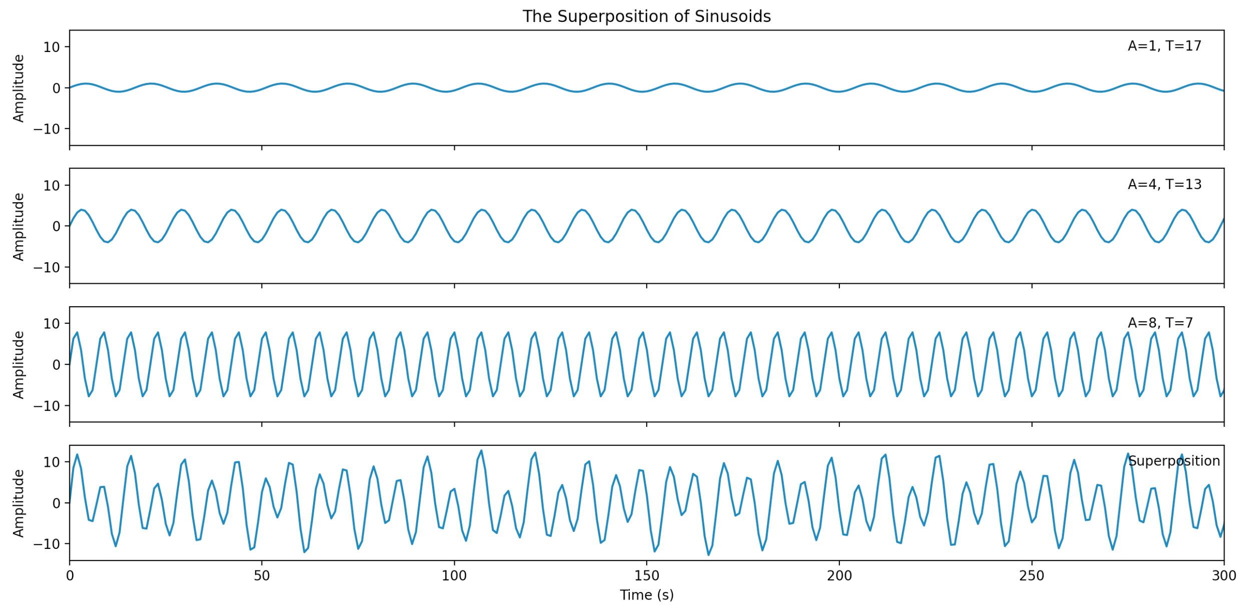
$$h(t) = \sin\left(\frac{2\pi t}{17}\right) + 4\sin\left(\frac{2\pi t}{13}\right) + 8\sin\left(\frac{2\pi t}{17}\right). \tag{3.2}$$

Plotting this function in Python and comparing it with the cleaned signal, we obtain the plots



[Figure 3.4 (Above)] The comparison of FFT filtered and noisy signal data with an ideal result, taken from the FFT data. (Top) The noisy, unfiltered input signal. (Middle) The FFT filtered output signal. (Bottom) The ideal output signal, as plotted from equation (3.2).

It is quite apparent in the previous figure that the ideal output signal and the filtered signal are almost indistinguishable, up to a relative phase difference. From this data and the Fourier transform plots, one may see that the original data had three waves superposed with each other, each with different frequencies and amplitudes.



[Figure 3.5] The superposition of sinusoids with various amplitudes and frequencies. The lowermost plot depicts the result. The frequencies and amplitudes were determine by the FFT data given by the cleaned wave signal.

This completes exercise 3.

Exercise 4

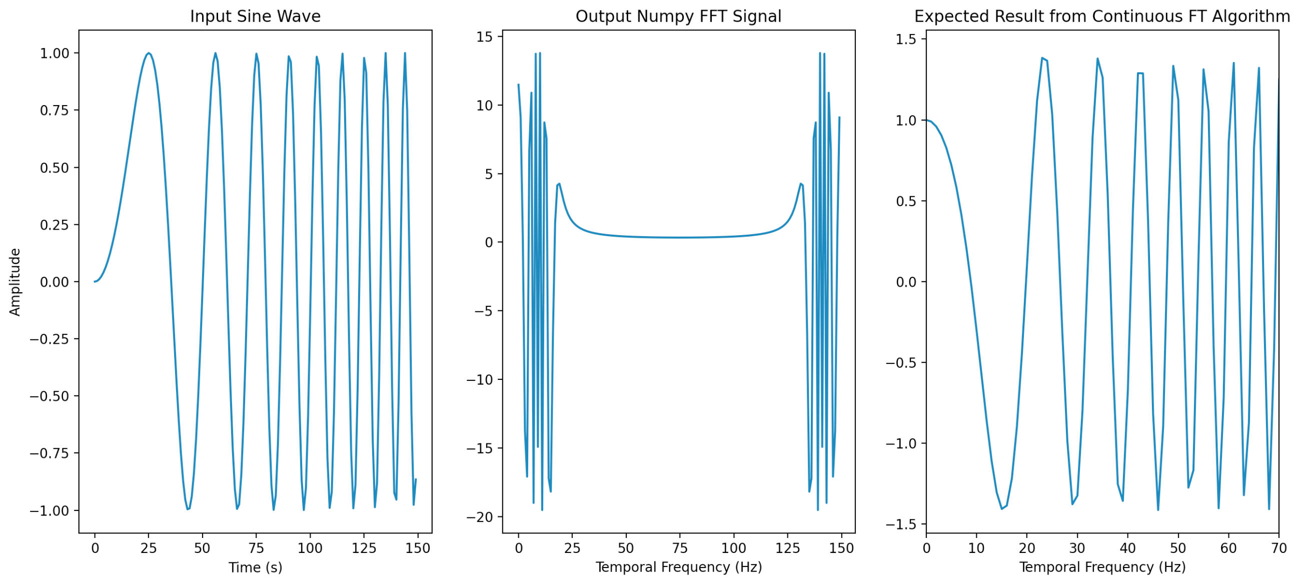
We now turn to the idea of a sinusoidal signal containing a time-varying angular frequency $\omega(t) = \dfrac{2\pi}{T(t)}$. In this case, I have chosen the angular frequency to be of a simple analyzable form $\omega(t) = \dfrac{1}{4}t$, so that the signal is

$$f(t) = \sin\left(\frac{t^2}{4}\right). \tag{4.1}$$

I have found that taking 150 samples produces a relatively smooth curve which oscillates about 10 times. Upon taking the fast Fourier transform, the number of non-zero data points are now limited due to the integration of complex-valued entries in the array due to the nature of the FFT algorithm. From this, I expect the plot depicting the FFT to be somewhat ‘blocky’. Moreover, allow me to first show the theoretical prediction of which function we would expect to get when applying the continuous transform operation:

$$
\begin{aligned}
\overline{f}(k) &= \frac{1}{\sqrt{2\pi}} \int_{-\infty}^{\infty} \sin\left(\frac{x^2}{4}\right) e^{-ikx}\, dx \\
&= \frac{1}{\sqrt{2\pi}} \left[\int_{-\infty}^{\infty} \sin\left(\frac{x^2}{4}\right) \cos(kx)\, dx - i \int_{-\infty}^{\infty} \sin\left(\frac{x^2}{4}\right) \sin(kx)\, dx\right] \\
&= \frac{1}{\sqrt{2\pi}} \left[\sqrt{\pi} i \sqrt{i}\, \mathrm{e}^{-ik^2} - \sqrt{i}\sqrt{\pi}\, i\mathrm{e}^{ik^2}\right] + \frac{i}{\sqrt{2\pi}}[0] && \text{(integral calculator)} \\
&= \cos(k^2) - \sin(k^2). && (4.1)
\end{aligned}
$$

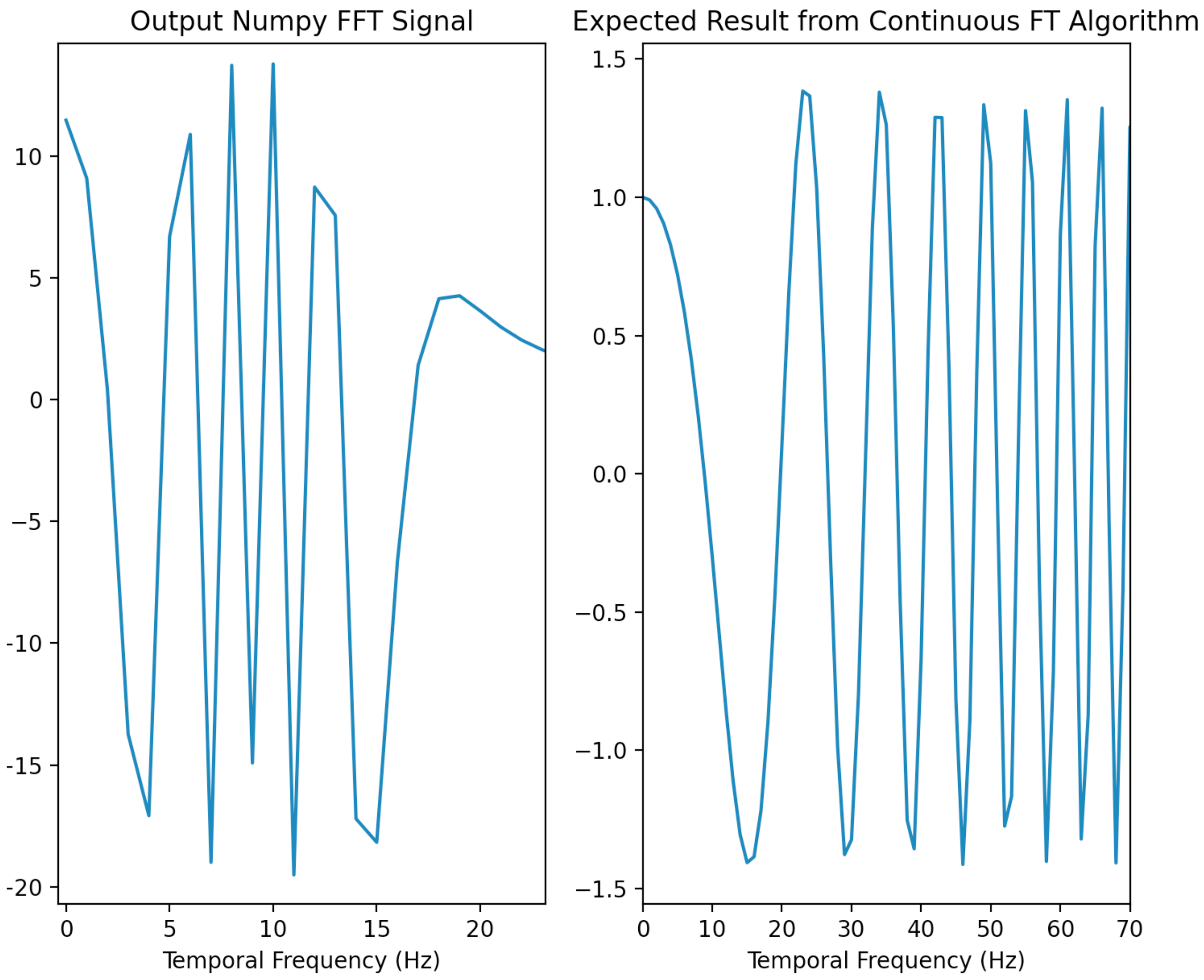
Comparing with the plotted fast Fourier transform from python with plots, we obtain the following information:



[Figure 4.1] The comparison of fast Fourier transform outputs for a nonlinear frequency input. (Left) The input signal. (Center) The data produced by the Numpy FFT algorithm, plotted along the whole real and complex frequency spectrum. (Right) The expected output signal, as depicted by the continuous Fourier transform algorithm in equation (4.1).

Figure 4.1 unveils a discrepancy between the algorithm and the continuous operator, since the FFT and continuous plots differ significantly in amplitude and frequency variation. This reveals information regarding how the FFT algorithm works.

Although the continuous operator acting on the frequency-varying sinusoid returns another continuous sinusoid, the FFT algorithm only outputs an equal number of input entries, explaining the line of zero amplitude in the middle of the graph. Furthermore, the FFT algorithm outputs an equivalent spectrum of frequencies to which the original sample covers in its lifetime of 150 data points. The peak and frequency magnitudes are determined by the magnitudes of the neighbourhood points of the initial data array, and are rescaled according to the number of samples used. Due to this, we wish to observe similar shapes between the expected and FFT curves. Consider a closer comparison of the FFT and continuous plots:



[Figure 4.2] A closer comparison of the output FFT signal and the expected result determined by equation (4.1).

One may quickly interpolate to see that the two curves above indeed display identical information, relative to a certain scaling and vertical shift. In the left plot, the smoothness of the curve is limited due to the number of samples (data points) taken from 0 to 150 (about 1 sample per axis number), which is why the curves do not look identical. Once the frequency range has been exhausted, the FFT plot dies off to zero, which is the main difference between the FFT and continuous operator, which does not vanish.

This completes exercise 4.