# PHY324 Computational Assignment

Chaolin Wang
Student number: 1005751740

January 2023

## 1 Abstract

The purpose of this report is to explore the fast Fourier transform (FFT) algorithm in Python. I performed FFT on various functions, some ideal, some noisy. I then used the outputs of the FFT to recover the primary frequency components of the original functions. For the noisy inputs, I used filter functions on the FFT to obtain a smoother version of the original signal.

## 2 Introduction

### 2.1 Fourier Series and Fourier Transforms

By **Fourier's theorem**, any reasonably continuous periodic function $f(x)$ with period $L$ is equivalent to an infinite sum of sinusoidal or complex exponential functions (i.e. a **Fourier series**).[1] The Fourier series for $f(x)$ in exponential form is as below:

$$f(x) = \sum_{k=-\infty}^{\infty} \gamma_k \exp\left(\mathrm{i}\frac{2\pi kx}{L}\right), \tag{1}$$

where for each exponential at index $k$: $\dfrac{k}{L}$ is the **frequency** and $|\gamma_k|$ is the **amplitude**. In addition, if $f(x)$ is a real valued function, $\gamma_k = \overline{\gamma_{-k}}$ for every $k$.[2]

The coefficients $\gamma_k$ are computed using the following formula:

$$\gamma_k = \frac{1}{L}\int_0^L f(x)\exp\left(-\mathrm{i}\frac{2\pi kx}{L}\right)dx. \tag{2}$$

The **Fourier transform (FT)** is a process to turn the function $f(x)$ into its frequency domain. In particular, the Fourier transform takes $f(x)$ and returns the amplitude of each exponential of the Fourier series as a function of its corresponding frequency.

[1] Complex exponential and sinusoidal functions are related by Euler's Formula $e^{\mathrm{i}\theta} = \cos(\theta) + \mathrm{i}\sin(\theta)$.
[2] $\overline{z}$ means the complex conjugate of $z$.

Ideally, the function to transform is periodic. However, if we are only interested in a finite interval from 0 to $L$ of a general function $f(x)$, we can take $L$ to be the period for the Fourier series and only consider the results in this interval.

## 2.2 Discrete Fourier Transforms and Fast Fourier Transforms

A **discrete Fourier Transform (DFT)** is a discrete version of the FT highlighted above. Suppose the are $N$ samples of $f(x)$ at evenly-spaced points $x_n = \frac{n}{N}L$, for $\quad n = 0, 1, ..., N-1$. Define $y_n = f(x_n)$. With some numerical integration on equation (2) and simplification, we obtain the following:

$$\gamma_k = \frac{1}{N}\sum_{n=0}^{N-1} y_n \exp\left(-\mathrm{i}\frac{2\pi kn}{N}\right). \tag{3}$$

However, by convention the sum in equation (3) is defined as the DFT of the samples $y_n$. Thus, we define:

$$c_k = \sum_{n=0}^{N-1} y_n \exp\left(-\mathrm{i}\frac{2\pi kn}{N}\right). \tag{4}$$

This means that our original sample of $y_k$ can be obtained by the following:

$$y_n = \frac{1}{N}\sum_{k=0}^{N-1} c_k \exp\left(\mathrm{i}\frac{2\pi kn}{N}\right). \tag{5}$$

Equation (5) describes the process called the **inverse discrete Fourier transform (inverse DFT)**.

Recall that $x_n = \frac{n}{N}L$. So if we want to recover the original function depending on $x_n$, $f(x_n)$, we can substitute $n = \frac{N}{L}x_n$ into equation (5), to get

$$f(x_n) = \sum_{k=0}^{N-1} \frac{c_k}{N} \exp\left(\mathrm{i}\frac{2\pi k x_n}{L}\right). \tag{6}$$

Let us define $R := \frac{N}{L}$ as the sampling rate. Thus, for each exponential at index $k$: the **frequency** $f_k$ and the **amplitude** $A_k$ are the following:

$$f_k = \frac{k}{L} = \frac{k}{N}R, \qquad A_k = |\gamma_k| = \frac{|c_k|}{N}. \tag{7}$$

In addition, if $f(x_n)$ is a real valued function, $c_k = \overline{c_{N-k}}$ for every $k$.

The **fast Fourier transform (FFT)** is an optimized version of the DFT algorithm. The FFT and DFT produce the same output and the optimization is not relevant in this report. The equations (3-7) still apply to the FFT. As a result, I will be using FFT and DFT interchangeably henceforth.

## 3 Methods and Procedures

### Materials

- A computer capable of running Python.

### 3.1 Sum of Two Sines

1. First, I picked amplitudes and angular frequencies for two sine waves. I chose the values $A_1 = 3, \omega_1 = 30, A_2 = 10, \omega_2 = 120$, where $A_i$ and $\omega_i$ denote the amplitude and angular frequency of the $i^{\text{th}}$ sine wave respectively. Note that I set the signal's independent variable as time and its dependant variable as position.[3] In addition, the $A_i$'s are in units of meters and the $\omega_i$'s are in units of radians per second.

2. Then, I set $N = 400$ and created a time array from 0.0 to 1.0 seconds with a step size of $d = 1.0/N$ seconds. Note that the sampling rate is $R = 1/d$.

3. Next, I created the sample $y_n$ as the sum of two pure sine waves with the chosen frequencies $A_1, A_2$ and amplitudes $\omega_1, \omega_2$ (both having a phase of 0). I plotted $y_n$ on a position vs. time graph.

4. Finally, I called `numpy.fft.fft` on the sample $y_n$ and plotted the absolute value of each output from the FFT vs. its index.

### 3.2 Filtering a Sum of Two Sines

1. First, I added some Gaussian distributed noise with a spread of $(A_1 + A_2)/4$ (half of the average of the amplitudes) to the previous $y_n$.

2. Then, I performed FFT on the new noisy sample to obtain a noisy FFT graph.

3. Next, I located the indices $k_{1,2}$ of the two leftmost peaks on the noisy FFT by eye and with the help of `scipy.signal.find_peaks`. I converted the values to spacial frequencies $f_{1,2}$ and amplitudes $A_{1,2}$ as per equation (7).[4] These frequencies should match the ones of the original function. Also, the amplitudes should be around half of the original as the peaks are mirrored across $k = N/2$.

4. Next, I created a filter function as the sum of Gaussian distributions centered at $k_{1,2}$. The variance/spread of each Gaussian was determined by experimentation.

5. Next, I multiplied the filter function to the noisy FFT to obtain a filtered FFT. I called `numpy.fft.ifft` on the filtered FFT to obtain a filtered signal.

6. Finally, I plotted the noisy FFT, filter function, and filtered FFT in one figure; and I plotted the noisy, filtered, and original(ideal) signal in another figure.

[3]This is a natural choice as this exercise serves an example. A plot of position vs. time in units of meters vs. seconds is a clear illustration.

[4]Note that one can go in the reverse direction. Since we know the exact values of $\omega_{1,2}$, we can use equation (7) to get the indices for the peaks of the FFT.

## 3.3 Filtering an Unknown Data Set

1. First, I imported the noisy data using the `pickle` module from the file `noisy_sine_wave`.

2. Then, I performed FFT on the data to obtain a noisy FFT graph.

3. Next, I located the indices $k_{1,2,3}$ of the three leftmost peaks on the noisy FFT by eye and with the help of `scipy.signal.find_peaks`. I converted the values to spacial frequencies $f_{1,2,3}$ and amplitudes $A_{1,2,3}$ as per equation (7). These frequencies should match the ones of the original function. Also, the amplitudes should be around half of the original as the peaks are mirrored across $k = N/2$.

4. Next, I created a filter function as the sum of Gaussian distributions centered at $k_{1,2,3}$. The variance/spread of each Gaussian was determined by experimentation.

5. Next, I multiplied the filter function to the noisy FFT to obtain a filtered FFT. I called `numpy.fft.ifft` on the filtered FFT to obtain a filtered signal.

6. Next, I created a theoretical signal as the sum of three sines with the frequencies $f_{1,2,3}$ and amplitudes $2A_{1,2,3}$.

7. Finally, I plotted the noisy FFT, filter function, and filtered FFT in one figure; and I plotted the noisy, filtered, and theoretical signal in another figure.

## 3.4 A Sine with a Varying Frequency

1. First, I created a sine function where the frequency increases linearly with respect to time.

2. Then, I applied FFT to the function and plotted the position vs. time of the function and the FFT.

## 4 Results and Analysis

### 4.1 Sum of Two Sines

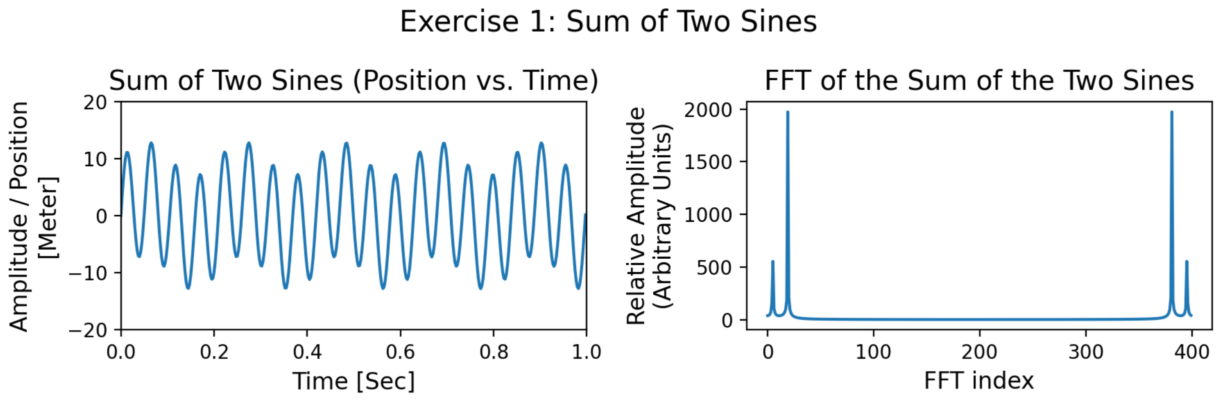

Figure 1: The sum of 2 sines along with its FFT. One sine has $A_1 = 3, \omega_1 = 30$ and the other has $A_2 = 10, \omega_2 = 120$.

The FFT shows two pairs of peaks, mirrored across $k = N/2 = 200$. This is reasonable as the original function is real valued so $|c_k| = |\overline{c_{N-k}}|$. Using `scipy.signal.find_peaks`, I got the indices $k_1 = 5$ and $k_2 = 19$. By equation (7), we get:

$$\omega_1 = 2\pi f_{k_1} = 2\pi \frac{k_1}{L} = 2\pi \frac{5}{1.0} = 31.416 \text{ rad s}^{-1}$$

$$\omega_2 = 2\pi f_{k_2} = 2\pi \frac{k_2}{L} = 2\pi \frac{19}{1.0} = 119.381 \text{ rad s}^{-1}$$

These values do in fact match closely with the original values of $\omega_1 = 30$ and $\omega_2 = 120$.

For the amplitudes, we can recover half of the original amplitudes by using equation (7) again. I got the magnitude of the peaks from the FFT as $|c_{k_1}| = 554.229$ and $|c_{k_2}| = 1973.048$.

$$A_1 = 2|\gamma_{k_1}| = 2\frac{|c_{k_1}|}{N} = 2\frac{554.229}{400} = 2.771 \text{ m}$$

$$A_2 = 2|\gamma_{k_2}| = 2\frac{|c_{k_2}|}{N} = 2\frac{1973.048}{400} = 9.866 \text{ m}$$

The values here are close to the original values of $A_1 = 3$ and $A_2 = 10$. Also the ratio of $\frac{|c_{k_1}|}{|c_{k_2}|} = 0.281$ is roughly equal to $\frac{A_1}{A_2} = 0.25$.

## 4.2 Filtering a Sum of Two Sines

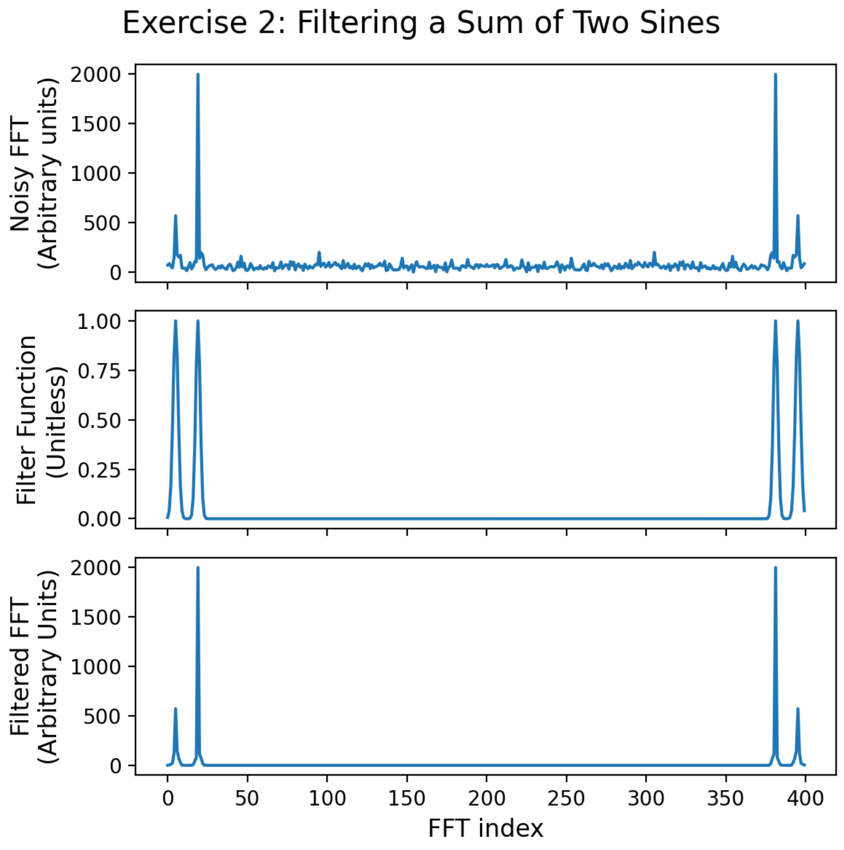

Figure 2: The filtering process for the previous sum of two sines with added noise.

The filter function was chosen so that the mean of each Gaussian distribution lies at the index of each corresponding peak of the FFT. Knowing that the underlying signal here is the exact same as the signal from part 4.1, I just reused the indices $k_1 = 5$ and $k_2 = 19$. After some experimentation, a width $(2\sigma^2)$ of 5 for the $k_1$ peak and 4 for the $k_2$ peak seemed to work well. Taller peaks tend to have less spread so I chose the second Gaussian to be slightly narrower.

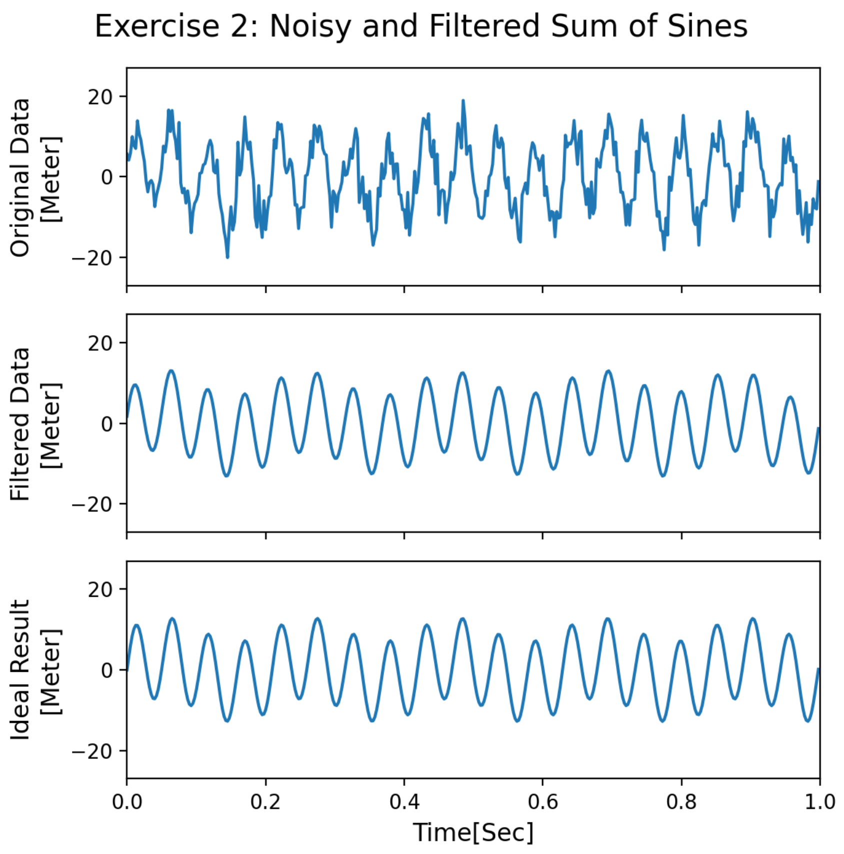

Figure 3: A noisy version of the sum of sines from 4.1 compared to a filtered signal and the ideal result (signal before adding noise).

As can be seen in Figure 3, I successfully filtered the the signal as the filtered signal matches up with both the noisy and the ideal clean signal. This is to be expected as even with noise, the signal consists of two primary sine waves. This means that the FFT should work well here.

## 4.3 Filtering an Unknown Data Set

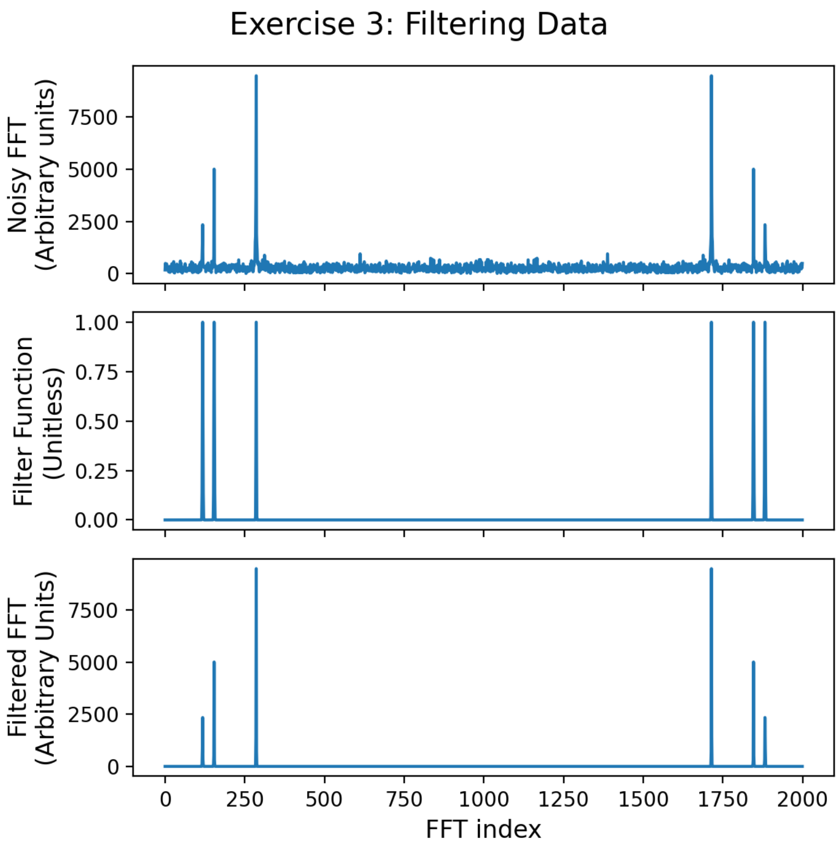

Figure 4: The filtering process for the signal from the provided unknown data set

In Figure 4, we can see 6 main peaks(half mirrored across $N/2 = 1000$). Then, it would be reasonable to hypothesize that the original signal without noise consists of 3 sinusoidal frequencies.

For that reason, I created the filter function for this data set using 3 sets of 2 Gaussians to isolate the 3 main frequencies.

Using the methods from 4.1 based on equation (7), we obtain the angular frequencies and the amplitudes from each peak.

$$\omega_1 = 0.371, \qquad A_1 = 2.344$$
$$\omega_2 = 0.484, \qquad A_2 = 5.008$$
$$\omega_3 = 0.898, \qquad A_3 = 9.485$$

I used these values to create a theoretical signal consisting of 3 pure sine waves.

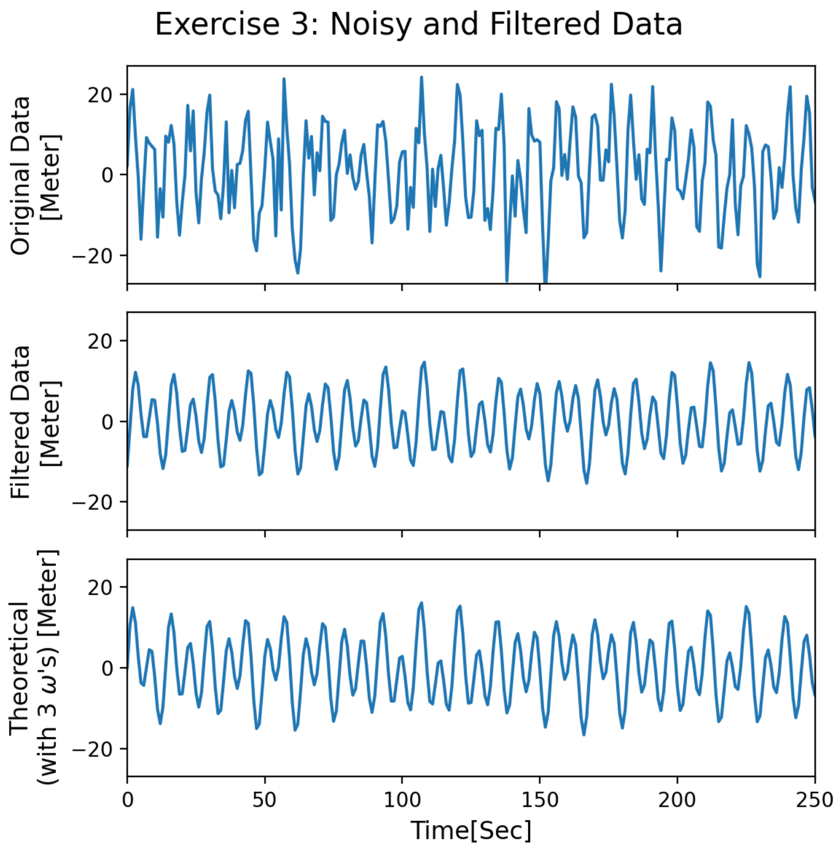

Figure 5: Original noisy signal compared to a filtered signal and the theoretical signal consisting of only 3 sine waves

In Figure 5, the phases of all three signals line up exactly. The amplitudes are also similar. However, the amplitude of the theoretical signal seems to be slighter greater than that of the filtered data. This could be due to the greater amount of noise in the original signal. Also, the nature of picking 3 distinct frequencies versus filtering gradually with Gaussian curves could lead to different results.

Note that I did not have information about the units or sampling rate of the signal. For simplicity, I chose a sampling rate $R$ of 1. For clarity, I took the signal as positions as a function of time. I assigned seconds to be the unit of time and meters to be the unit of position. This is not necessary but it does help make the plots easier to interpret.

Also I chose to only plot the first 250 point data points because plotting the whole data set would make the details too fine to see properly.

## 4.4 A Sine with a Varying Frequency

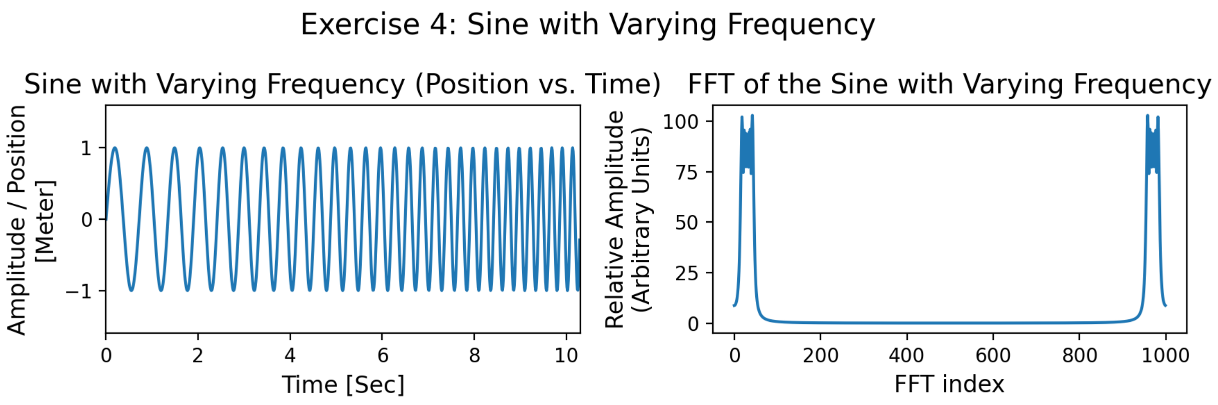

Figure 6: Sine with a constant amplitude and a frequency linearly increasing in time along with its FFT

In this section, I defined the signal as:

$$y(t) = A\sin(\omega(t)t),$$

where $A = 1$ and

$$\omega(t) = \omega_0 + st,$$

with $s = 1$ and $\omega_0 = 8$.

From Figure 6, we can see that the FFT contains a dense range of peaks at an interval of frequencies. This makes sense, because as we include more and more data points, every single angular frequency greater than $\omega_0$ will be included in the signal $y(t)$. Thus, the FT would theoretically yield a constant positive amplitude for all frequencies $f > \omega_0/(2\pi)$. However, since the domain is finite, the FFT can only produce a finite range of non-zero amplitudes.

## 5 Conclusion

In this report, I explored the FFT algorithm in Python. I learned how to convert between the output of the FFT and the actual (angular) frequency and amplitude of the original signal. The main conversion is documented in equation (7). I have also shown that the FFT can be useful for analyzing and filtering functions, as in the case with the noisy signals.

## 6 References

Newman, Mark, director. *Where Is Frequency in the Output of the FFT?*, YouTube, 15 May 2022, https://youtu.be/3aOaUv3s8RY.

Newman, Mark. "Ch. 7 Fourier Transforms." *Computational Physics*, Mark Newman, 2012, pp. 289–304.