PHY324 Computational Project:
Fast Fourier Transforms (FFT) using Python
January 2023

1 Project Description

This document gives you some theory, and then has some exercises. You must write your own Python code to answer the exercises. The report that you submit should include some text that you write which describes what you have done. It should also include all necessary graphs. Make sure your graphs are labelled well and easy to read! **Your report should NOT include your Python code!**

This is a paper, not a homework assignment. Do not write "The answer to Exercise 1 is ..." Instead, describe what you are doing. Imagine you are writing this report for your fellow students so they can study it for a test, where their project is on a completely different topic which you will have to study for the same test. That means that along with your results and calculations, you need some brief theoretical introductions to give your results some context. Otherwise your fellow students will be completely lost trying to study from your paper and they will hate you.

Finally, before you read any further, make sure you download the available Python codes, and the data file. You will need the data file to do one of the exercises. The Python code will be very useful for you to reuse/recycle when you go to do the exercises.

If it takes you more than an hour or two to do any given exercise then I **strongly suggest** you stop and ask for help!

The figures here, and in the available Python code, are not well labelled. **You should improve that if you want a good mark.** Even the units are not always consistent. You need to figure this out for yourself as part of understanding how do use a Fast Fourier transform.

This is an individual assignment. While you can get help from anyone you wish, you must create your own graphs and write your own explanations.

2 Project Requirements

There are 4 exercises. Do all 4.

For exercise 1 you will need to submit 2 graphs: a position-time graph and an amplitude-frequency graph (the absolute value of the FFT of your first graph). Your description must describe how they contain the same information. Sample calculations showing how your two graphs agree on the frequencies and amplitudes is one good way to clearly demonstrate this information.

For exercise 2 you will need to submit 6 graphs: you basically need to reproduce Figures 2 and 3 from this document. Since you have access to the code which produced these graphs,

it should be easy to reproduce them using your own data. Your discussion should include how you chose the parameters for the Gaussian filter function. Also discuss how similar your filtered function (your fifth graph) resembles the ideal graph (your sixth graph).

For exercise 3 you will need to submit the same 6 graphs as exercise 2. Since you have been given a data file with no other information (unlike in exercise 2) there will be much more trial-and-error in this exercise. Your report should include your logic for the parameters you chose for your filter function. It must include your best guess value for how many waves, of what frequencies and relative amplitudes, are contained in the provided data file.

For exercise 4 you will need 2 graphs: a position-time graph and the FFT of that graph. Discuss the relationship between these two graphs and try to explain how the FFT graph contains the information in the position-time graph.

Before you begin, make sure you have downloaded these three files: "pickle-example.py", "PHY324-FFT.py" and "noisy_sine_wave".

3 Fourier Series

Fourier transform methods (spectral methods) are widely used in research as efficient data manipulating tools: signal processing, digital frequency filters, data analysis, etc. **An accessible introduction to Fourier analysis and its applications can be found** at `http://www.upscale.utoronto.ca/PVB/Harrison/FourierTransform.pdf` . **Check it out before proceeding any further!**

It is useful to remember that the Fourier transformation does not contain any new information. It is simply a different method of visualizing the original data. For example: if you shine light from hot helium through a prism, you would get a spectrum which is unique to helium. If we consider the original light to be a series of sinusoidal waves of an electro-magnetic field (which it basically is), then the light that comes out of the prism is basically the Fourier transform of that electro-magnetic wave. Again, there is no new information, but the information is presented in a completely different manner.

Enough theory! Let's see an example. We will start with an easy example. Let our function be $y(t) = A\sin(\omega t)$. The FFT (fast Fourier transform which, if you did not read the link above, is an efficient method of finding the Fourier transform of finite/discrete data) should look like two spikes in an otherwise-boring graph. For technical reasons, those spikes are the same (there is a mirror symmetry at work, so really only the first half of the FFT is relevant). This is because our function is a mono-chromatic (single-frequency) wave, so its "spectrum" should have a single non-zero data point. The top-half of Figure 1 shows the wave and its FFT. On another technical note, the FFT is a complex function, so we will always be plotting the absolute value of the FFT.

Of course no real data is ever so nice as a mono-chromatic wave. One of the nice features of the FFT is that even real, noisy data can be analyzed very easily with the FFT algorithm. The bottom-half of Figure 1 shows the same original wave with a lot of noise added to the signal. You can see that the noise is about the same amplitude as the signal, however the FFT graph still shows the same spikes at the same locations, and all the noise just looks like

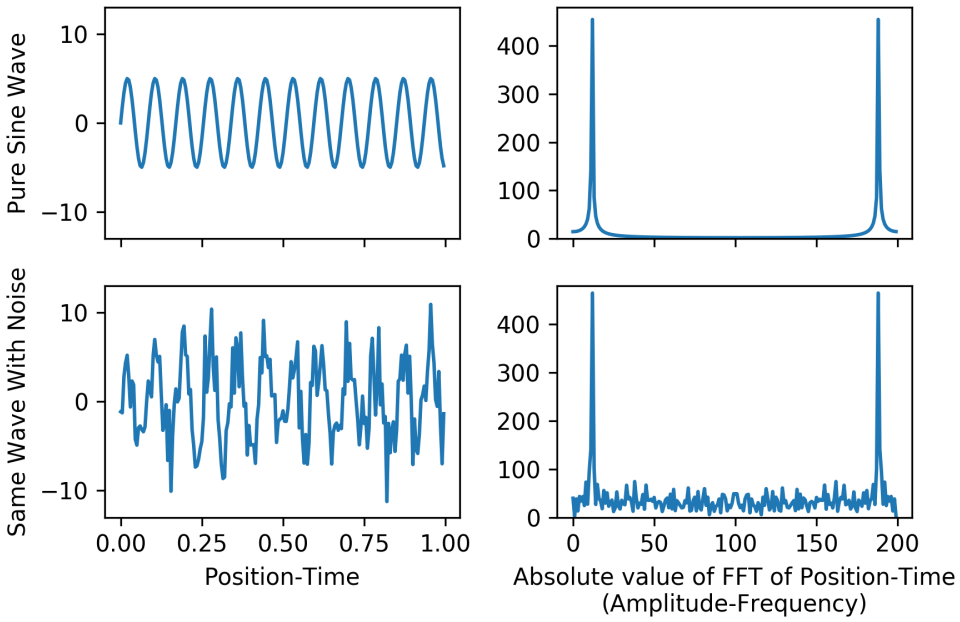

Figure 1: A pure sine wave (top-left) with its associated FFT (top-right). Also, the same wave with Guassian noise added (bottom-left) with its associated FFT (bottom-right). If the time on the left is in seconds then the frequency on the right is in hertz.

noise.

(Another technical note: the noise is Gaussian noise with a mean value of zero and a standard deviation of half the amplitude of the sine wave.)

Exercise 1 *Note:* Before you start this exercise, I suggest that you look at the Python file you will need for exercise 2. It contains a fair bit of code which could prove useful when doing this exercise and might save you some time. This is optional, but recommended.

Write a Python code which adds two waves of different amplitudes and different frequencies to get one function. Plot the function, and the (absolute value of the) FFT of your function. Do not add noise. Make sure your report includes these two graphs. Ideally, your function should oscillate around 10-20 times, as in the top-left graph of Figure 1.

Now explain how to use the FFT to find the 2 frequency values you used, and how to find the relative amplitudes. That is, by looking at the locations (x-axis) and heights (y-axis) of your FFT graph, show the calculations that convert those values into the values you used to create the data. You will need to understand this process in order to finish Exercise 3.

4 Signal Filtering

We now want to "clean up" our noisy signal in an attempt to reproduce the original signal. Obviously, this is a silly exercise as written since we have the original data. However, once we figure out how to clean up our noisy data we can use the same algorithm on new data, data for which we do not know the "real" signal.

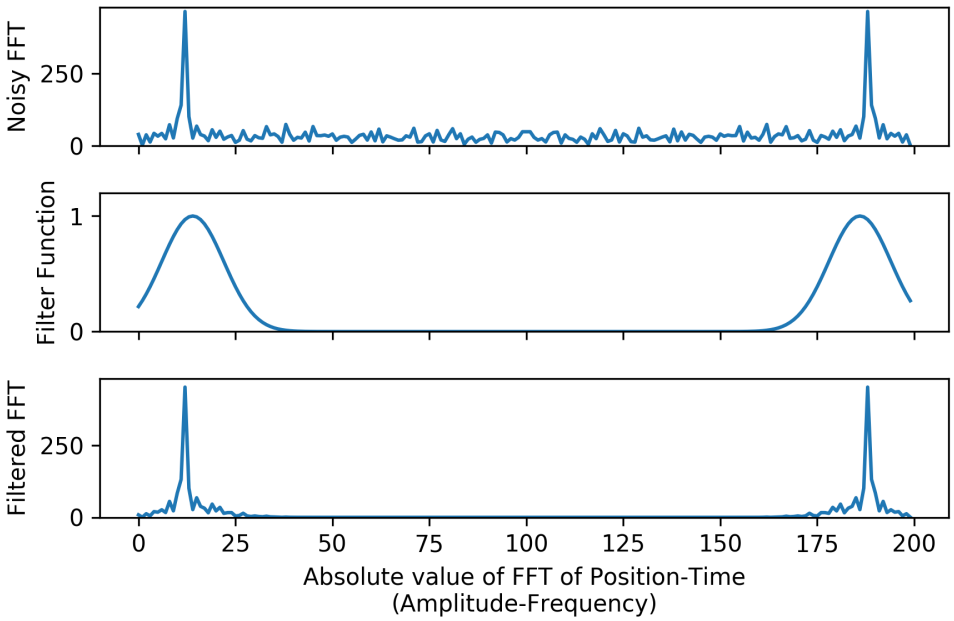

Figure 2: The process of filtering our data. We start with the FFT of our data (top). We then create our filter function (middle). We multiply these two graphs to get our filtered data (bottom).

We demonstrate this graphically. In Figure 2 we have the same FFT from the bottom-right corner of Figure 1. We then create our filter function. Filter functions can have any form we want, but a fairly reliable function is a Gaussian function.

It is important that the filter function have a magnitude which is as close to 1.0 as possible (without going over) at all spike locations in the FFT, and be as small as possible everywhere else. In general, a wider filter function keeps more noise but also keeps more signal. If you make the filter function too narrow then you risk losing some of the signal, especially if your alignment is not perfect (as our alignment is deliberately poor in Figure 2).

If you multiply your filter function by your FFT, your new graph is a smoothed-out FFT, as seen in the bottom part of Figure 2.

As well as the FFT, there is an inverse FFT function in Python which converts an FFT graph into a position-time graph. If you take a graph, take the FFT of the graph, then take the inverse FFT you get exactly the same graph back (up to some rounding errors). However, if you filter your FFT first, and then take the inverse FFT of your filtered function, your

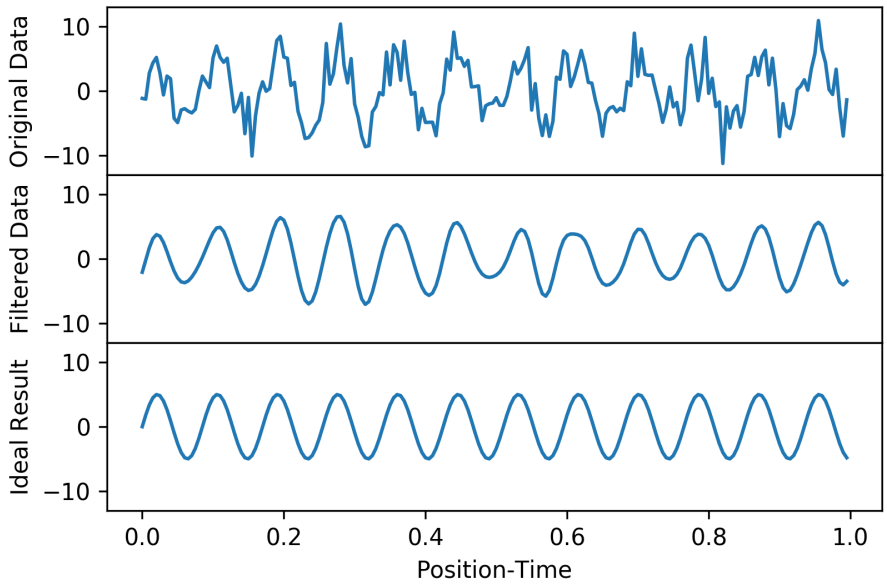

Figure 3: The original data (top), the result of our filtering as depicted in Figure 2 (middle); the ideal result (bottom). The ideal result is, in fact, the original data before any noise had been added.

resulting graph should be much smoother. See the middle graph in Figure 3. For reference, the bottom graph in Figure 3 is the original, zero-noise signal which we are trying to recover. As you can see, our filter function did a pretty good job of eliminating most of the noise, but the signal has clearly-visible differences.

Exercise 2 You should have access to the code which produced Figures 1 and 2 (named PHY324-FFT.py). Adjust the parameters of the filter function to improve the result of the inverse FFT. Specifically, shift the peak locations to align perfectly with the peak and then reduce the width of the filter function until you get a result which is very similar to the original graph. You cannot make it perfect, but you can make it hard to distinguish between the two.

Your report should include your version of Figure 2, and it should include your version of Figure 3. Make sure you include some text describing what you did in order to demonstrate your understanding of what the filter function is supposed to do. For example, you should be able to figure out how to align the filter function with the peak exactly without using trial and error. (*Hint:* you have access to the original data function so you know its period and frequency.)

Exercise 3 Put the files 'noisy_sine_wave' and 'pickle-example.py' in the same directory. Then run the 'pickle-example.py' code. It should plot a noisy position-time graph. This graph has a fairly large amount of noise and multiple waves of

different amplitudes and frequencies. Your task is to try to reconstruct, as best as you can, the original data before the noise was added. This process is basically the same as Exercise 2, except you do not know what the correct answer is. Feel free to reuse whatever code you want.

Hint: Although you can get away with using a single Gaussian filter for this, your results will not be very good. For best results make your filter function be the sum of multiple Gaussian filters, each as narrow as possible.

Finally, make your best guess as to how many waves the original data had, what their frequencies were, and what their relative amplitudes were.

Make sure your report includes the same graphs as you had in Exercise 2. While you do not have access to the original data to make the bottom graph of Figure 3, you can plot a theoretical curve given the frequencies and relative amplitudes which you guessed at the end of this exercise in its place.

5 FFT of other functions

While the FFT is primarily designed to analyse functions with a fixed periodicity, we can use it on any function. To see what it does, consider just one case.

Exercise 4 Take the FFT of a sinusoidal function with constant amplitude but a time-dependent frequency, i.e. $(y) = A\sin(\omega(t)t)$. For convenience, do a simple function for the frequency: make $\omega(t)$ a linear function of time, perhaps a frequency which changes value by about 25% over the course of 100 oscillations. Plot your sine function and the FFT of your sine function. Discuss any observations you make with respect to the FFT you produced.

Hint: make sure you have enough data points that your sine function repeats multiple times. More than 10 complete oscillations is good. You want less than 100 oscillations or else your graph will just be a blur.