PHYS 381 – Computational Physics I (Winter 2025)

Assignment #4: Fourier analysis using Python

Due date: March 31, 2025

Group members:

Member #1: Jared Crebo (30085839)

Authors' contributions:

This assignment was completed solo. All work presented in this document and the related code was completed by Jared Crebo.

Abstract (0.5 points):

This report investigates Fourier analysis techniques, including the Fourier series, Fourier transform, and Discrete Fourier Transform (DFT) to analyze functions and signals. The Fourier series decomposes periodic functions into a series of harmonic functions, and the Fourier transform extends this to non-periodic functions by integrating over time for each frequency. The DFT provides a method for computing the Fourier transform for discrete, non-periodic data making it the most applicable for practical usage. Each of these three methods are implemented in Python using various functions and data to test the effectiveness of reconstructing these signals. Certain parameters such as the sampling step size, dominant frequencies, and number of Fourier coefficients are extracted to investigate their effects on the accuracy of these methods.

Introduction (0.5 points):

The purpose of this assignment is to implement several signal processing methods to understand the underlying mathematics behind these useful tools. A Fourier series is able to convert any periodic signal, such as a square wave, into a continuous signal via a weighted sum of harmonic functions. In front of each harmonic function is a Fourier coefficient that determines the dominance of that harmonic function within the overall signal. A large number of Fourier coefficients increases the number of harmonic functions applied to the Fourier series, which increases the accuracy of the Fourier series.

$$f(t) = a_0 + \sum_{n=1}^{\infty}[a_n \cos(n\omega t) + b_n \sin(n\omega t)]$$

The Fourier transform is based upon the Fourier series, but is used for non-periodic, continuous signals. It integrates the function over time for a continuous series of frequencies, effectively creating a Fourier series as a function of frequency rather than time.

$$F(\omega) = \frac{1}{\sqrt{2\pi}}\int_{-\infty}^{\infty} dt\, f(t)e^{-i\omega t}$$

A Discrete Fourier Transform (DFT) uses the Fourier transform mathematics, but computes it numerically for discrete, finite signals that are sampled at *N* points along the signal.

$$F_n = \sum_{m=0}^{N-1} f_m e^{-i\frac{2\pi mn}{N}}$$

Both $F(\omega)$ and F_n can be used to reconstruct the original signal, $f(t)$, using the original Fourier series equation. All three methods can be used for analyzing audio signals, EMR signals, sensor signals, and more. They can decompose the raw data in these signals into a continuous function that can be filtered and analyzed for its period, amplitude, phase, and rate of change.

The power spectrum in Fourier analysis is a spectrum of which frequencies dominate the signal and is based off of the calculated Fourier coefficients, or the magnitude of the Fourier transform.

$$P(\omega) = |F(\omega)| = |F_n|$$

This can be used to determine the dominant frequency in the signal, which for a periodic signal is the fundamental frequency. These signal processing methods and tools will be implemented in Python and discussed in detail in the following report.

Methods (1 point):

The Fourier analysis is very relevant for signal processing. It can decompose both period and non-periodic signals into constituent harmonic vibrations. First looking at periodic functions, any periodic function can be represented as a Fourier series:

$$f(t) = a_0 + \sum_{n=1}^{\infty}[a_n \cos(n\omega t) + b_n \sin(n\omega t)]$$

The frequency $\omega_1 = \omega$ is the fundamental frequency, and every other ω_n is harmonic. The Fourier coefficients a_n and b_n amplify certain harmonic functions depending on their values. Functions can be reconstructed in this format as an infinite sum of this Fourier series. However, in practical use it is summed over *N* Fourier coefficients.

The Fourier coefficients a_0, a_n, and b_n are calculated using the following formulas based on the orthogonality properties of sines and cosines:

$$a_o = \frac{1}{T}\int_0^T f(t)dt$$

$$a_k = \frac{2}{T}\int_0^T f(t)\cos(k\omega t)dt$$

$$b_k = \frac{2}{T}\int_0^T f(t)\sin(k\omega t)dt$$

Numerically, these integrals are evaluated via Simpson's Rule:

$$\int_0^T f(t)dt \approx \frac{h}{3}\left[f(0) + 2\sum_{j=1}^{\frac{n}{2}-1} f(t_{2j}) + 4\sum_{j=1}^{\frac{n}{2}} f(t_{2j-1}) + f(T)\right]$$

The value of *n* represents the number of intervals in which the integral is split, therefore a large value of *n* will result in higher accuracy.

For non-period functions, a Fourier transform is carried out to integrate the series over a continuous range of frequencies.

$$F(\omega) = \frac{1}{\sqrt{2\pi}}\int_{-\infty}^{\infty} dt\, f(t)e^{-i\omega t}$$

The magnitude $|F(\omega)|$ can be viewed as the total summation of the signal over time at a frequency, ω. The Discrete Fourier Transform (DFT) is a numerical solution of $F(\omega)$ that samples the signal *N* times at intervals *h*, where the time at each sample point is $t_m = mh$ with $m = 0, 1, \dots, N-1$. $f_m = f(t_m)$ is therefore defined as the signal evaluated at each time, t_m, and F_n is the Fourier transform of this sampled signal f_m. The real and imaginary components of F_n are the Fourier coefficients of the Fourier series of f_m and can be calculated as such:

$$Re[F_n] = \sum_{m=0}^{N-1} f_m \cos\,(\frac{2\pi mn}{N})$$

$$Im[F_n] = \sum_{m=0}^{N-1} f_m \sin\,(\frac{2\pi mn}{N})$$

The original signal can therefore be reconstructed as a Fourier series, regardless of if its periodic or non-periodic:

$$f_m = \frac{1}{N}\sum_{n=0}^{N-1}\left\{Re[F_n]\cos\left(\frac{2\pi mn}{N}\right) + Im[F_n]\sin\left(\frac{2\pi mn}{N}\right)\right\}$$

Code workflow (1 point):

In problem 2.2b, the *pitch.txt* data is loaded into an array *pitch* through the NumPy.loadtxt() function. The number of sampling points is evaluated as the number of total data points in *pitch*. The noise filtration threshold, e, is set to $e = 50$. An array of time is created from $t = 0$ to $t = N - 1$.

The functions *re_data* and *im_data* take in the parameters of the function, *fm*, as well as the iterative indices *m* and *n* and the total sampling points *N* and returns the real and imaginary components of $Re[F_n]$ and $Im[F_n]$.

$$Re[F_n] = \sum_{m=0}^{N-1} f_m \cos\,(\frac{2\pi mn}{N})$$

$$Im[F_n] = \sum_{m=0}^{N-1} f_m \sin\,(\frac{2\pi mn}{N})$$

The function *real_imag* takes the function, *func*, and the total sampling points, *N*, and will return the array of real components, imaginary components, and power of F_n corresponding to *func*. When called, new NumPy arrays of *real* and *imag* are initialized to store their respective components. The function iterates through both *n* and *m* from $0 - N$ to calculate the real and imaginary components of F_n for each *n*. The function takes the power, $P_n = \sqrt{Re[F_n] + Im[F_n]}$, and checks whether $P_n < e$ in which case it will set $Re[F_n] = Im[F_n] = 0$ to filter out the noise. The power spectrum, real components, and imaginary components are returned from the function.

The *real_imag* function is called for the *pitch* data, in which its power spectrum, real components, and imaginary components are calculated and returned. These are then inputted into the *reconstruct* function to reconstruct the original function based on the Fourier transform. This function will simply iterate though *m* and *n* and calculate the Fourier transform at each step *m*.

$$f_m = \frac{1}{N}\sum_{n=0}^{N-1}\left\{Re[F_n]\cos\left(\frac{2\pi mn}{N}\right) + Im[F_n]\sin\left(\frac{2\pi mn}{N}\right)\right\}$$

The real components, imaginary components, and the power spectrum of the DFT are all plotted against time. f_m is plotted next to its original function to show how it has been reconstructed with the noise filtered from the data.

Results and analysis (1 point):

Problem 1.2b: Compute and plot the Fourier coefficients of the functions:

1) $f(t) = \sin(\omega t)$
2) $f(t) = \cos(\omega t) + 3\cos(2\omega t) - 4\cos(3\omega t)$
3) $f(t) = \sin(\omega t) + 3\sin(3\omega t) + 5\sin(5\omega t)$
4) $f(t) = \sin(\omega t) + 2\cos(3\omega t) + 3\sin(5\omega t)$

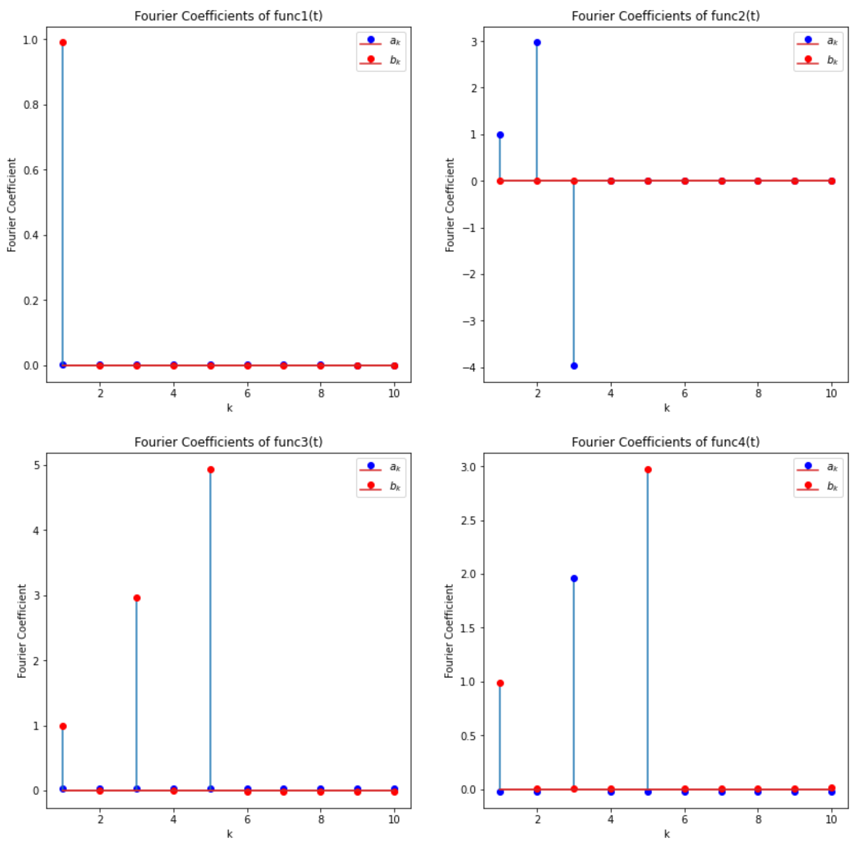

Figure 1: Fourier Coefficients of Various Functions

Figure 1 shows the plots of the Fourier coefficients of all four functions listed above with respect to their frequency number, *k*. A notable pattern between these plots is that the real (a_k) components are added when instances of cosine exist in *f(t)*, and imaginary (b_k) components are added for instances of sine. Moreover, these coefficients are corresponding to specific frequencies that are denoted within the sinusoidal functions denoted by *k*.

Problem 1c: Analyze the square wave with the Fourier series

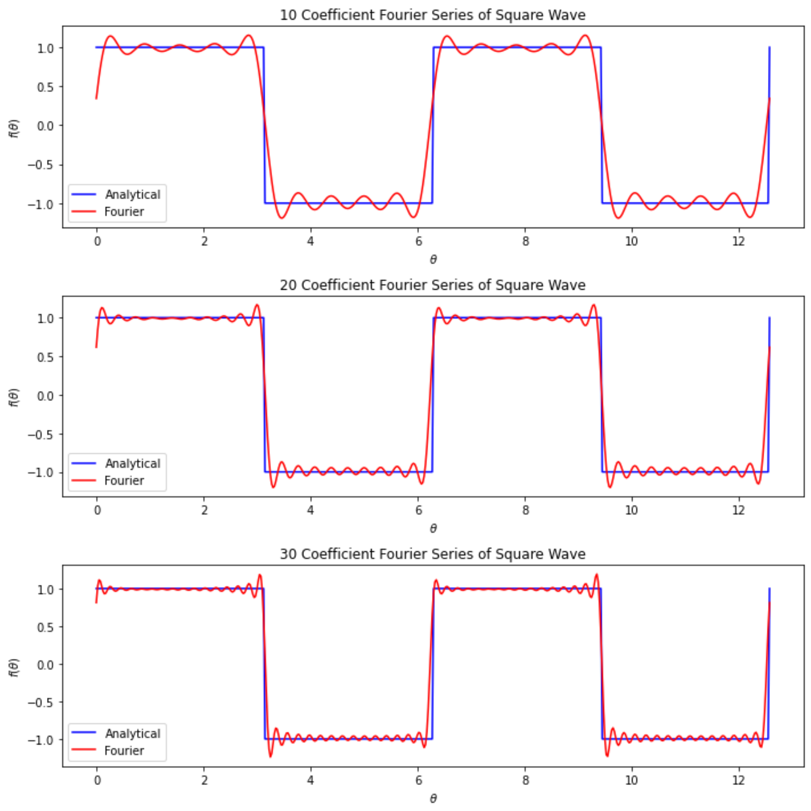

Figure 2: Fourier Series Reconstructed Over Square Wave

The square wave is an example of a good use of the Fourier series for analysis, since a perfect square wave is discontinuous at each period and half period. A highly resolute Fourier series over a square wave would be a continuous function and its instantaneous rate of change can therefore be calculated. Figure 2 demonstrates how the number of Fourier coefficients increases the resolution of the Fourier series reconstruction. With a greater number of coefficients, the series is more accurate and with a theoretically infinite number of coefficients the square wave will be perfectly replicated as a continuous function.

The Fourier coefficients calculated can also be compared to their analytical counterparts for comparison. The methodology used to calculate these coefficients involves numerically solving the integral using Simpson's Rule, which will be subject to numerical errors depending on its resolution.

Average Coefficient	$\boldsymbol{a_k\ (k = 1, 3, 5)}$	$\boldsymbol{b_k\ (k = 1, 3, 5)}$	$\boldsymbol{b_k\ (k = 2, 4, 6)}$
Simpson's Rule	0.02631	0.64143	-0.00460
Analytical	0	0.65076	0
Absolute Error	0.02631	-0.00933	-0.00460

Table 1: Numerical and Analytical Fourier Coefficients, Averaged Over k

Table 1 shows the Fourier coefficients calculated by each method and averaged for three iterations of k. Results show low error between coefficients with the Simpson's Rule at $n = 100$ steps.

Problem 1d: Analyze pulse with Fourier series

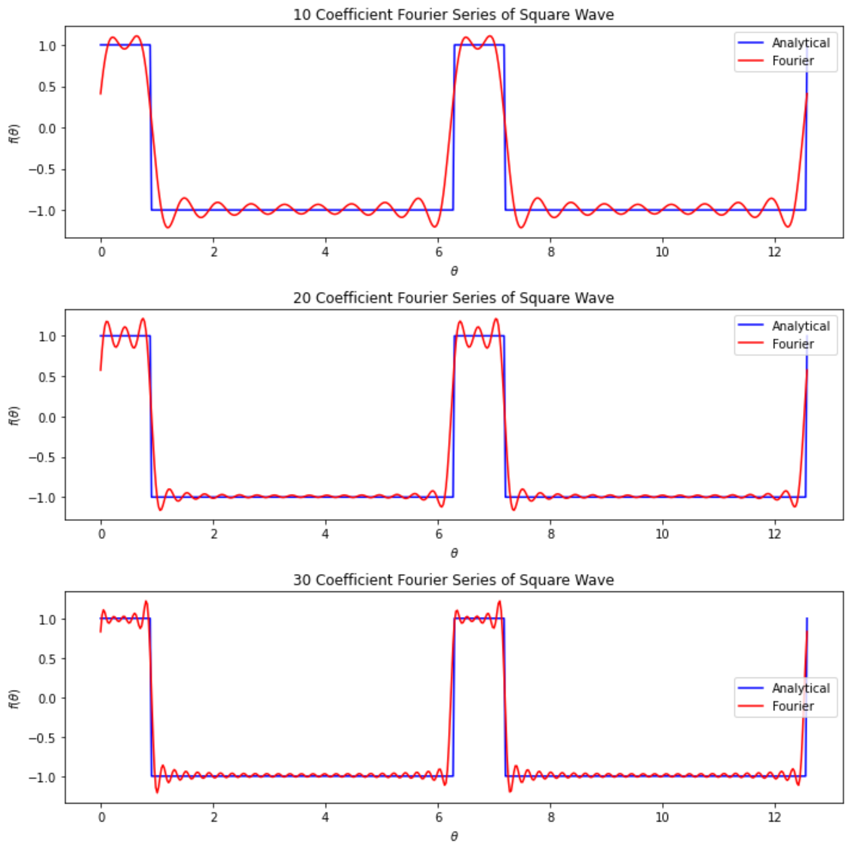

Figure 3: Fourier Series Reconstructed Over Pulse, $\alpha = 7$

For an asymmetric square wave, offset by $\omega\tau = 2\pi/\alpha$, the reconstructed Fourier series shows equivalent results in Figure 3 as it does in Figure 2. As the number of Fourier coefficients increases, so does the accuracy of the reconstruction.

Average Coefficient	$\boldsymbol{a_0}$	$\boldsymbol{a_k\ (k = 1, 2, 3)}$	$\boldsymbol{b_k\ (k = 1, 2, 3)}$
Simpson's Rule	-0.21857	0.32911	0.34417
Analytical	-0.71428	0.30004	0.34407
Absolute Error	0.49571	0.02906	0.00092

Table 2: Numerical and Analytical Fourier Coefficients, Averaged Over k

Table 2 shows good agreement with the average numerical and analytical Fourier coefficients. a_0 does show a higher error than the other coefficients, which is possibly because it cannot be averaged over different k values so the numerical errors are not averaged out of the data.

Problem 2.2a: DFT of $f(t) = \sin(0.45\pi t)$

This problem goes through the entire process of a Discrete Fourier Transform for an already continuous function.

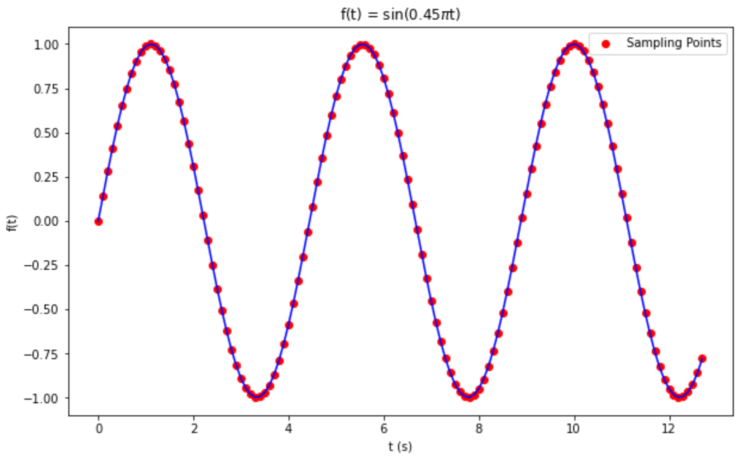

Figure 4: Original Function and Sample Points

Figure 4 shows the original function and the points along the function where it is being sampled. It gets sampled by $N = 128$ points and a timestep of $h = 0.1$, resulting in a total runtime of $\tau = 12.8s$.

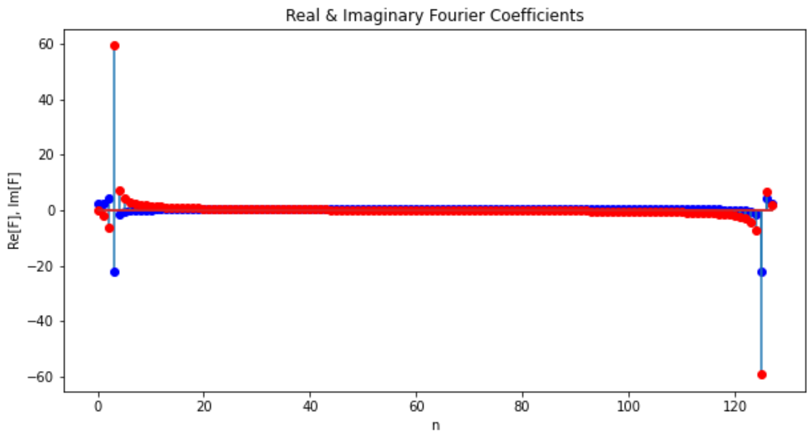

Figure 5: Real and Imaginary Fourier Coefficients

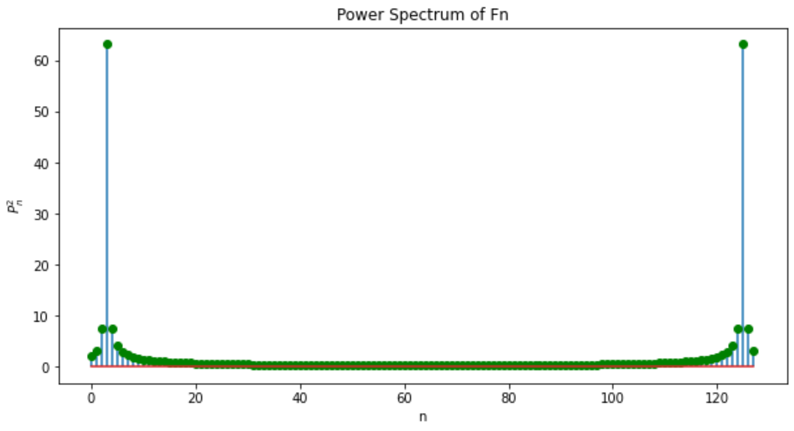

Figure 6: Power Spectrum of Fn

In Figure 5, the real and imaginary Fourier coefficients are calculated and plotted as a function of the frequency number, n. Figure 6 shows the power spectrum for the same Fourier coefficients. The dominant frequency is shown to be $0.234375Hz$ at $n = 4$. This is determined by the location of the highest power in the power spectrum.

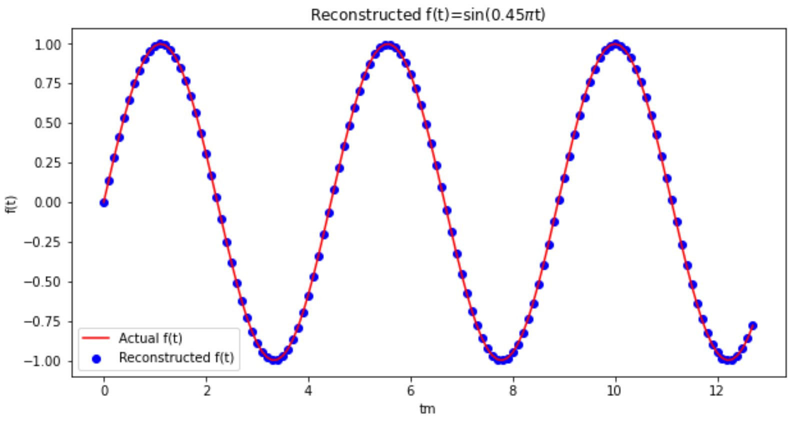

Figure 7: Reconstructed Signal

Lastly, the signal is reconstructed using the DFT methodology and plotted next to the original function. Given the fact that the original function was already continuous, and the sampling step size was tight, these functions line up perfectly. If the sampling step size was not adequate to capture the resolution of the oscillations, then the reconstructed signal would not line up to the original.

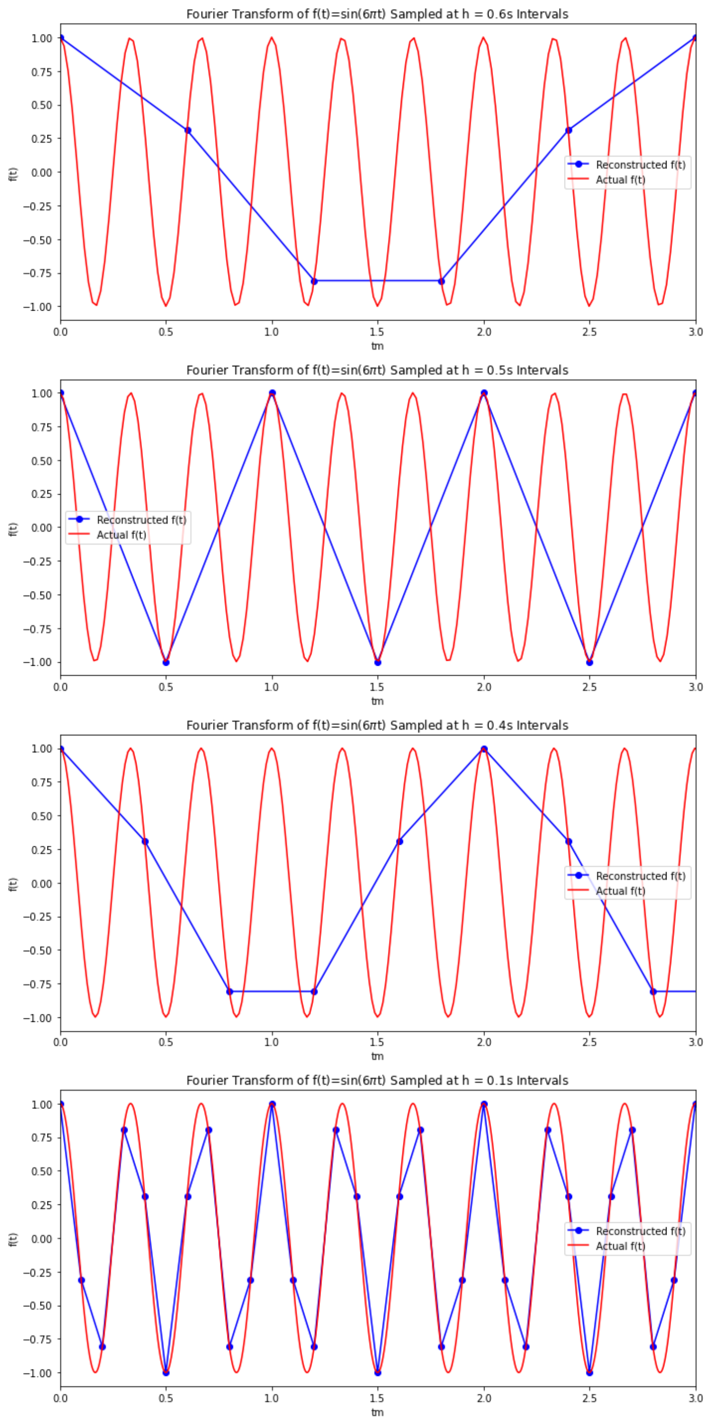

Figure 8: DFT of $f(t) = \sin(6\pi t)$ with varying step sizes

Figure 8 shows the effect of varying step sizes. The first three plots show that the reconstructed signal is missing several actual oscillations between its own datapoints due to their larger timesteps. The final plot shows that at $h = 0.1s$ all the oscillations are being captured, but at a very low resolution. A proper DFT would require even tighter sample sizing to properly reconstruct the signal. The plots show that a smaller timestep between sample points is correlated to a more accurately reconstructed signal.

Problem 2.2b: DFT of *pitch.txt* data

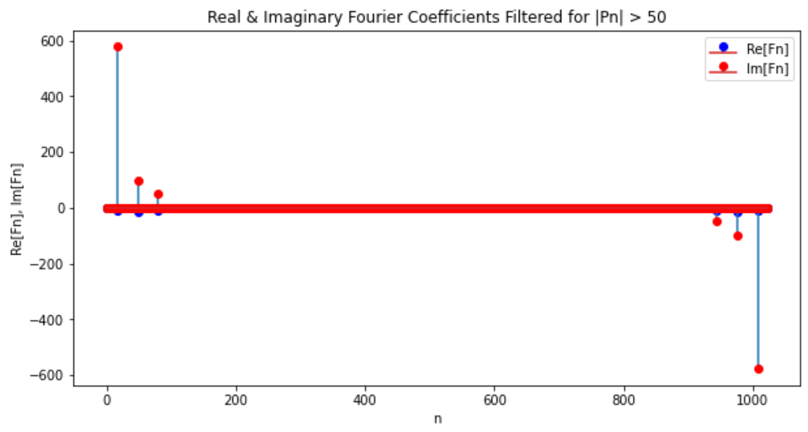

Figure 9: Real and Imaginary Fourier Coefficients, Filtered for $|P_n| > 50$

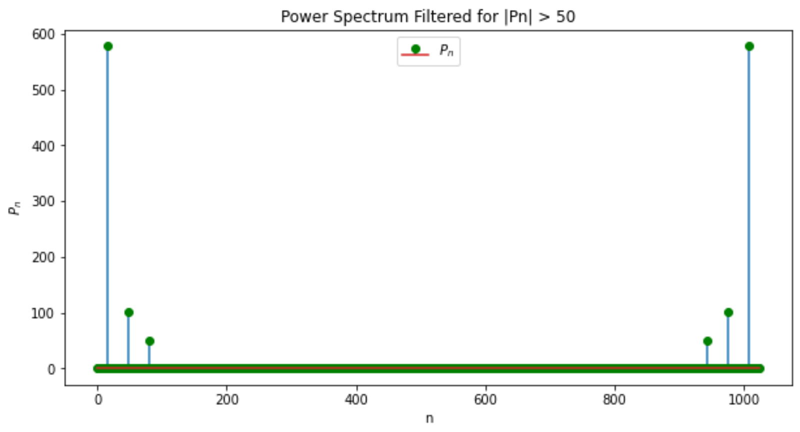

Figure 10: Power Spectrum Filtered for $|P_n| > 50$

In Figures 9 and 10, the Fourier coefficients are filtered for noise such that the power is no greater than a specified threshold. The effect of this is shown in the final reconstructed signal. Only the dominant frequencies are kept in the function. The peak power in the spectrum occurs at $0.015625Hz$ at $n = 17$.

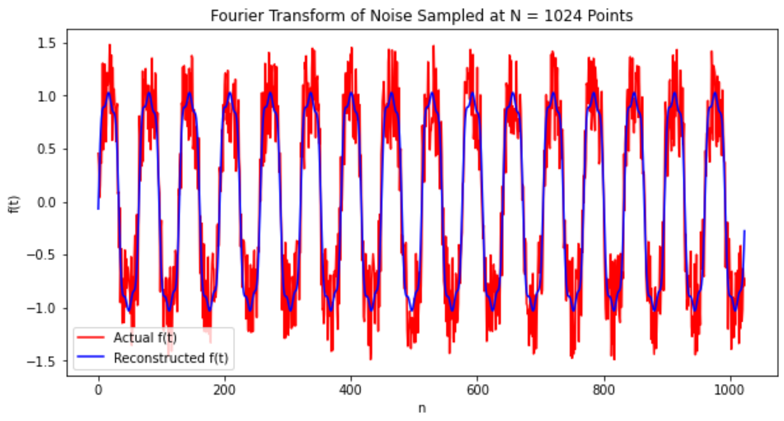

Figure 11: Reconstructed Signal

Due to the noise filtration of the Fourier coefficients, the reconstructed signal is a smooth, continuous function that can be analyzed. Discrete and noisy forms of data can be converted to smooth functions with this method. This highlights the usefulness of the DFT and its applications in audio, sensor, and electromagnetic signal processing.

Conclusions (0.5 points):

This assignment provided an in-depth exploration into Fourier analysis, including Fourier series, Fourier transforms, and Discrete Fourier Transforms. By implementing these mathematical tools in Python, it was demonstrated how signals can be decomposed into their constituent frequencies, analyzed for their Fourier coefficients, and reconstructed as a series of harmonic functions with varying degrees of accuracy. Several use cases for each of these were examined. The Fourier series is able to reconstruct periodic signals, while the Fourier transform can extend this to non-periodic signals, with both being originally continuous signals. The DFT is the most widely useable method since it allows the reconstruction of all signals whether they are discrete, continuous, periodic, and non-periodic. Additionally, the Fourier coefficients and power spectrum was plotted for each example case to highlight the dominant frequency components. The final example demonstrated a practical use of DFT with noise filtration by removing the non-dominant frequencies in the power spectrum. Overall, this assignment applied the methods and usefulness of Fourier analysis in computational physics and its broad applications across many fields of science and engineering.

References:

[1] PHYS 381 Assignment 4: Fourier analysis in Python, *PHYS 381: Computational Physics I,* Department of Physics and Astronomy, University of Calgary, Winter 2025

Other (0.5 points):

This page is to be filled by the instructor or TAs ONLY.

Remaining 0.5 points are granted for following the template and overall quality of the report.

Was the assignment submitted before the due date (mark one option)?   YES   NO