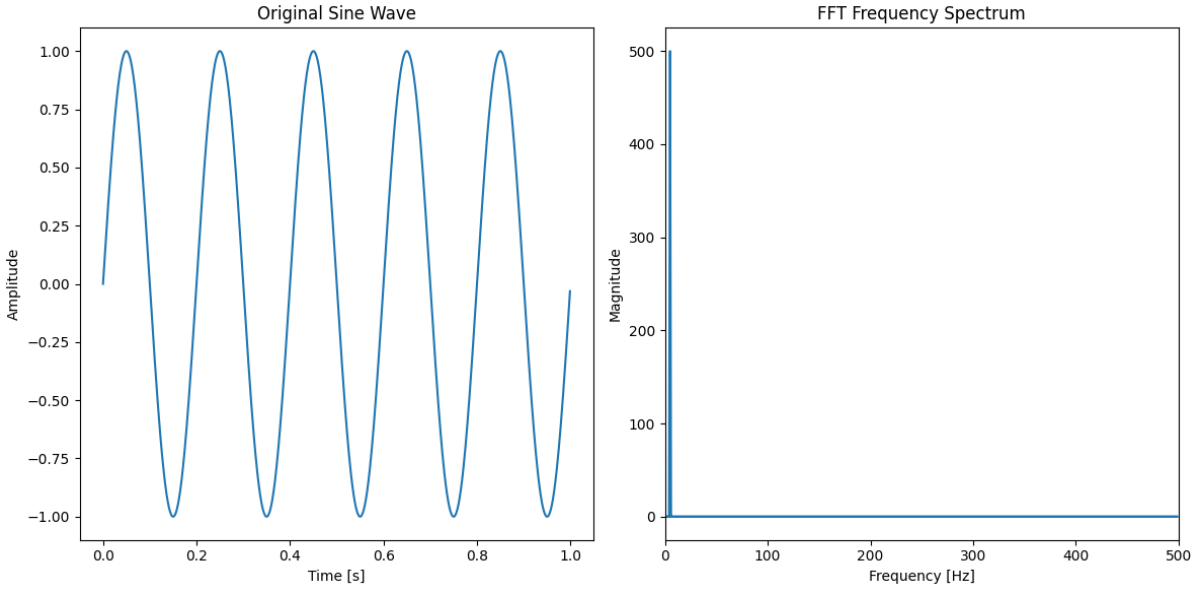
Python Assignment 2

Q1. Perform a Fast Fourier Transform (FFT) on a sine wave signal and visualize both the original signal and its frequency spectrum in python

Here's how you can perform a Fast Fourier Transform (FFT) on a sine wave signal and visualize both the original signal and its frequency spectrum using Python:



Code to generate the sine wave signal:

```
import numpy as np
import matplotlib.pyplot as plt

# Parameters
fs = 1000   # Sampling frequency
f = 5      # Frequency of the sine wave
T = 1      # Duration in seconds

# Time vector
t = np.linspace(0, T, int(T * fs), endpoint=False)
# Generate sine wave
signal = np.sin(2 * np.pi * f * t)
```

This code generates a sine wave signal with a frequency of 5 Hz, sampled at 1000 Hz for 1 second.

To plot the original sine wave:

```
plt.figure(figsize=(8, 4))
plt.plot(t, signal)
plt.xlabel('Time [s]')
plt.ylabel('Amplitude')
plt.title('Original Sine Wave')
```

```
plt.tight_layout()
plt.show()
```

This will display the original sine wave signal in the time domain.

To compute and Plot the FFT Frequency Spectrum:

```
# Compute FFT
fft_signal = np.fft.fft(signal)
fft_freq = np.fft.fftfreq(len(signal), 1/fs)

# Compute magnitude spectrum
magnitude_spectrum = np.abs(fft_signal)

# Plot the FFT frequency spectrum
plt.figure(figsize=(8, 4))
plt.plot(fft_freq, magnitude_spectrum)
plt.xlabel('Frequency [Hz]')
plt.ylabel('Magnitude')
plt.title('FFT Frequency Spectrum')
plt.tight_layout()
plt.show()
```

This code computes the Fast Fourier Transform (FFT) of the sine wave signal, calculates the frequency vector using np.fft.fftfreq(), and plots the magnitude spectrum. The resulting plot will show the frequency spectrum of the sine wave signal, with a peak at the original frequency of 5 Hz. By combining these two plots, you can visualize both the original sine wave signal in the time domain and its frequency spectrum obtained through the Fast Fourier Transform.

Q2. Use the scipy library to numerically integrate the function f(x) = x^2 over the range [0,5].

```
from scipy.integrate import quad
def f(x):
    return x**2
result, error = quad(f, 0, 5)
result, error
```

Result:
(41.66666666666666, 4.625929269271485e-13)

The result of numerically integrating f(x) = x^2 over the range [0,5] is approximately 41.67 with an error of 4.63×10^(-13).

Q3. Solve a simple optimization problem where you need to minimize the function f(x) = (x-3)^2+2 using scipy.optimize

```
from scipy.optimize import minimize
def f(x):
    return (x - 3)**2 + 2
```

```
result = minimize(f, x0=0)
result.fun, result.x
```

Result:
(2.000000000000001, array([3.00000003]))

The function f(x) = (x - 3)^2 + 2 is minimized at x≈3.000, and the minimum value of the function is approximately 2.

Q4. Solve a system of linear equations using numpy. Given the system:

2x+3y = 5

4x+y = 6

Solve for x & y.

```
import numpy as np
x = np.array([[2, 3], [4, 1]])
y = np.array([5, 6])
solution = np.linalg.solve(A, b)
solution
```

Result:
array([1.3, 0.8])

The solution to the system of equations is x=1.3 and y=0.8 .