Technical University of Munich

Neuroprosthetics Exercise 1 Report

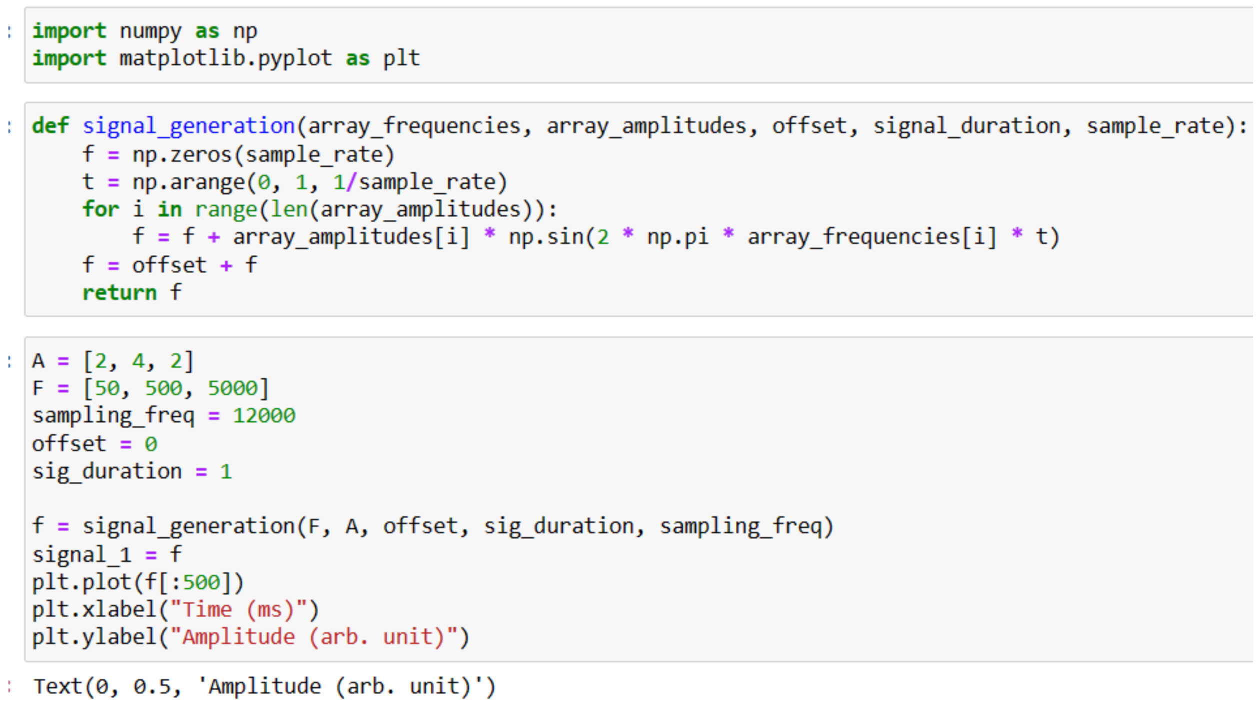
Student Name: Abdelrahman Elsissy

Student Number: 03780605

1. Signal Generation and plotting

1.1. 0 DC Offset Signal

The first task was to create a function that generates a signal. This function sums three sinusoidal signals with three arbitrary frequency components and three corresponding array of amplitudes. In the first task, the DC offset is 0 as none of the given frequencies have a 0 Hz value. As the signal length is 1 second and the sampling frequency is 12 kHz, then the corresponding sampling rate will be 1/12 kHz, and so we will obtain 12000 plotting points in the graph. A function called “signal_generation” was defined in the Jupyter Notebook code to generate and plot this signal. The chart shall demonstrate the amplitude in the time domain. Figure 1 below shows the Python code for signal generation and plotting the signal, and Figure 2 shows the obtained time domain plotted signal in Jupyter Notebook.



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

```
def signal_generation(array_frequencies, array_amplitudes, offset, signal_duration, sample_rate):
    f = np.zeros(sample_rate)
    t = np.arange(0, 1, 1/sample_rate)
    for i in range(len(array_amplitudes)):
        f = f + array_amplitudes[i] * np.sin(2 * np.pi * array_frequencies[i] * t)
    f = offset + f
    return f
```

```
A = [2, 4, 2]
F = [50, 500, 5000]
sampling_freq = 12000
offset = 0
sig_duration = 1

f = signal_generation(F, A, offset, sig_duration, sampling_freq)
signal_1 = f
plt.plot(f[:500])
plt.xlabel("Time (ms)")
plt.ylabel("Amplitude (arb. unit)")
```

```
Text(0, 0.5, 'Amplitude (arb. unit)')
```

Figure 1. Task 1.1 Python Code in Jupyter Notebook

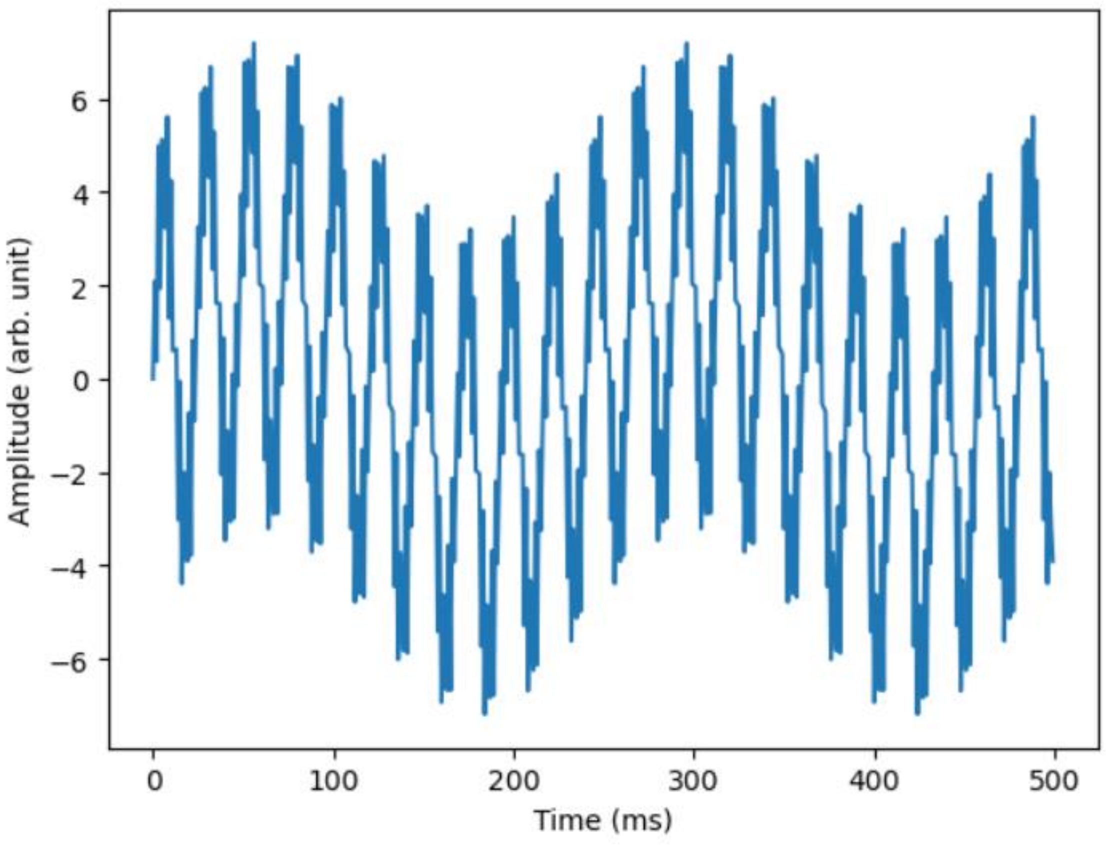
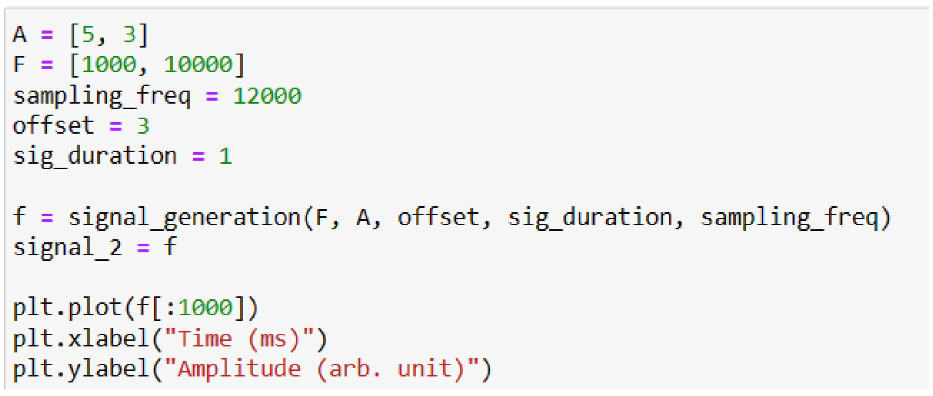


Figure 2. Signal in the time domain

1.2. 3 DC OFFSET SIGNAL

The second task is almost the same as task one. The only difference is that the second signal has a DC offset, as one of the frequency values is 0 Hertz. The corresponding amplitude of this frequency is three, which means that the signal has a DC offset with the value 3. Figure 1 below shows the Python code for signal generation and plotting the signal, and Figure 2 shows the obtained time domain plotted signal in Jupyter Notebook.



```
A = [5, 3]
F = [1000, 10000]
sampling_freq = 12000
offset = 3
sig_duration = 1

f = signal_generation(F, A, offset, sig_duration, sampling_freq)
signal_2 = f

plt.plot(f[:1000])
plt.xlabel("Time (ms)")
plt.ylabel("Amplitude (arb. unit)")
```

Figure 3. Signal with DC offset Python code in Jupyter Notebook

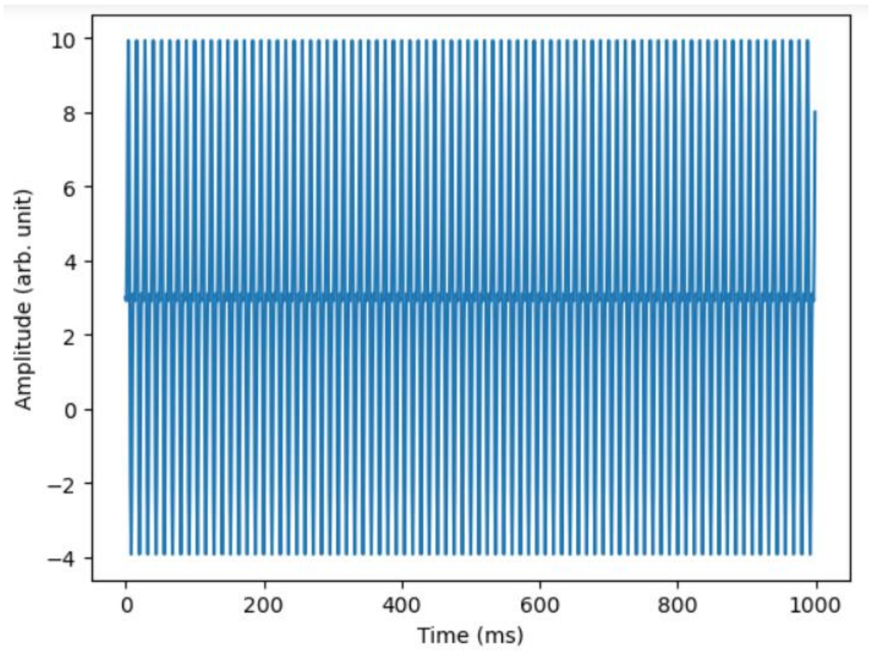
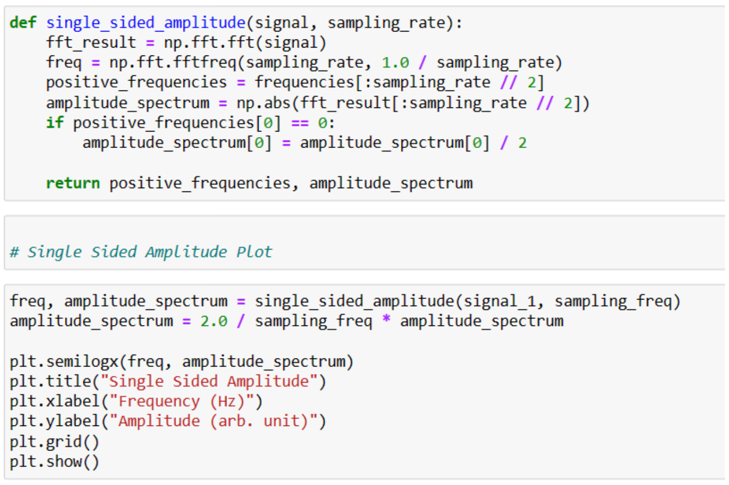


Figure 4. Signal in the time domain

2. Spectrum

2.1. Spectrum Plotting

A. The third task was to plot the single-sided spectrum of the signals obtained above. This could be done using the Fast Fourier Transform as it converts the signal from the time domain to the frequency domain. A function called “single_sided_amplitude” was defined to calculate the single-sided spectrum of the signal generated in section 1.1. Figure 5 illustrates the Python code to obtain and plot the single-sided spectrum, while Figure 6 shows the plotted spectrum. A logarithmic scale was required on the x-axis to plot this graph.



```
def single_sided_amplitude(signal, sampling_rate):
    fft_result = np.fft.fft(signal)
    freq = np.fft.fftfreq(sampling_rate, 1.0 / sampling_rate)
    positive_frequencies = frequencies[:sampling_rate // 2]
    amplitude_spectrum = np.abs(fft_result[:sampling_rate // 2])
    if positive_frequencies[0] == 0:
        amplitude_spectrum[0] = amplitude_spectrum[0] / 2

    return positive_frequencies, amplitude_spectrum
```

```
# Single Sided Amplitude Plot
```

```
freq, amplitude_spectrum = single_sided_amplitude(signal_1, sampling_freq)
amplitude_spectrum = 2.0 / sampling_freq * amplitude_spectrum

plt.semilogx(freq, amplitude_spectrum)
plt.title("Single Sided Amplitude")
plt.xlabel("Frequency (Hz)")
plt.ylabel("Amplitude (arb. unit)")
plt.grid()
plt.show()
```

Figure 5. Single Sided Spectrum Calculation and Plotting code (1.1 section signal)

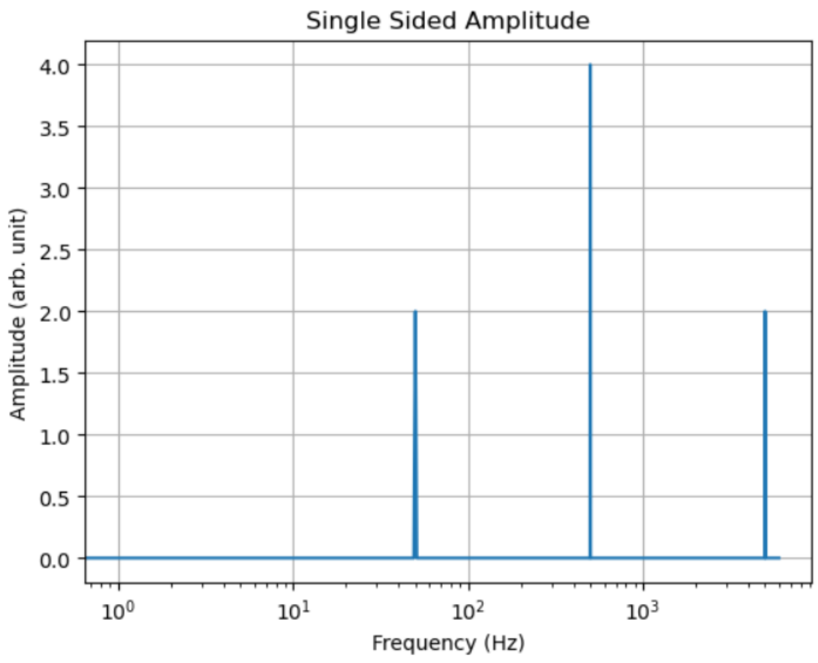
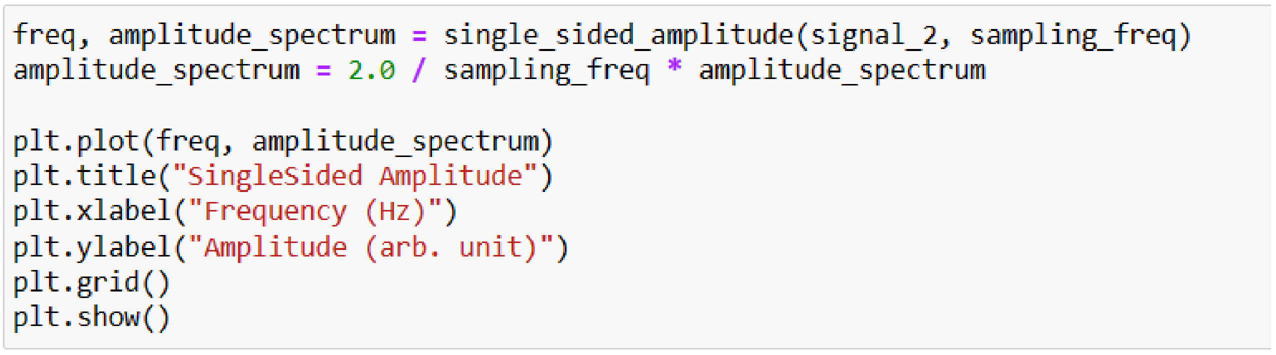


Figure 6. Spectrum of Signal generated in task 1.1

B. Similarly to task 2.1a, the "single_sided_spectrum" function was required to calculate the single-sided spectrum of the signal generated in section 1.2. Figure 7 illustrates the Python code to obtain and plot the single-sided spectrum, while Figure 8 shows the plotted spectrum.



```
freq, amplitude_spectrum = single_sided_amplitude(signal_2, sampling_freq)
amplitude_spectrum = 2.0 / sampling_freq * amplitude_spectrum

plt.plot(freq, amplitude_spectrum)
plt.title("SingleSided Amplitude")
plt.xlabel("Frequency (Hz)")
plt.ylabel("Amplitude (arb. unit)")
plt.grid()
plt.show()
```

Figure 7. Figure 5. Single Sided Spectrum Calculation and Plotting code (1.2 section signal)

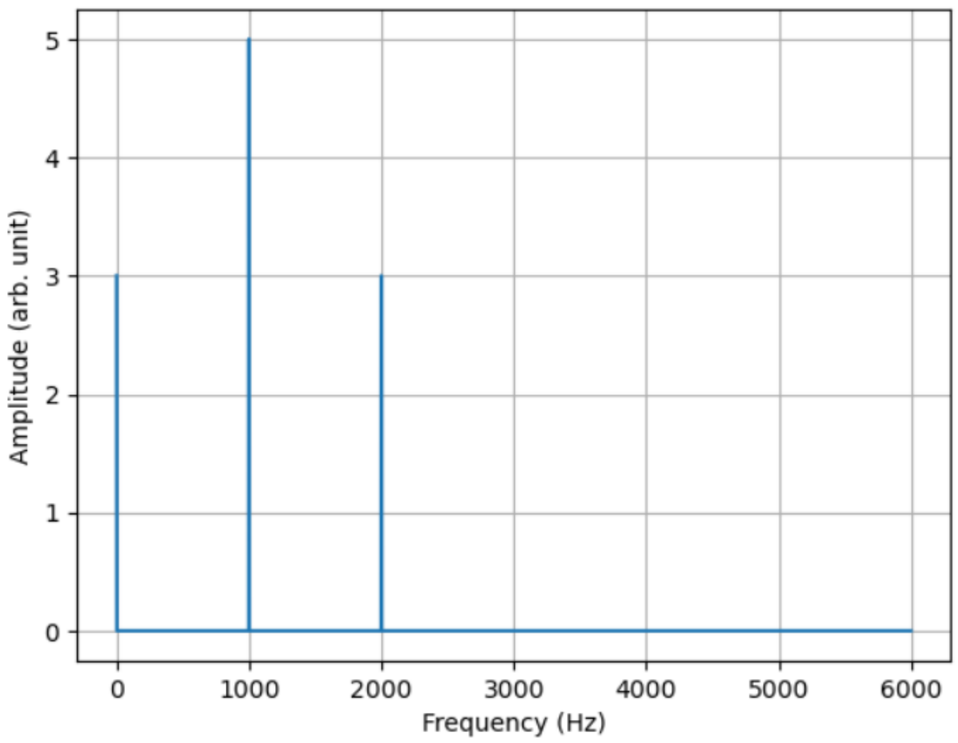


Figure 8. Figure 6. Spectrum of Signal generated in task 1.2

2.2. SPECTRUM INTERPRETATION

a) For the first generated signal, we the following array of frequencies respectively:
A = [2, 4, 2]

F = [50 Hz, 500 Hz, 5000 Hz]

We can see that the amplitude spectrum matches the frequency and amplitude components given at signal generation, which means that for the first signal generated, this was the expected amplitude spectrum.

For the first generated signal, we have the following array of frequencies respectively:
A = [3, 5, 3]

F = [0 kHz, 1 kHz, 10 kHz]

We can see that the amplitude spectrum does not fully match the frequency and amplitude components given at signal generation, as the expected frequency responses should be at 0 kHz, 1 kHz, and 10 kHz; however, they are obtained at 0 kHz, 1 kHz, and 2 kHz, which means that for the first signal generated, this was not the expected amplitude spectrum.

b) For the first generated signal, we choose a logarithmic scale as it allows spectrum analysis over a broader range of frequencies as we have a wider bandwidth between 50 Hz to 5000 Hz, so it would be ideal to use the logarithmic scale. This is because we have both low and high-frequency responses, and the frequency response should be analyzed over a more extensive range of frequencies. For the second generated signal, the range of frequencies is between 0 and 10 kHz. The bandwidth is narrower, so it would be applicable to choose a linear scale.

c) **The DC component,** as the doubling of the DC component is one of the most common artifacts observed in FFT analysis for signal generation. To avoid this, we initially adjusted the offset parameter to the required DC level to automatically add a DC element to the signal rather than specifying a DC component at 0 Hertz in the frequency components array.