ENG5027: Digital Signal Processing - Assignment 1

Zeynep Oner - 2693256O
Vishwa Poswa - 2699522P
Zainab Meerza - 2361575M
Evelyn Anyebe - 2615331A

University of Glasgow

1. QUESTION 1

In the first question, the aim was to plot the recorded voice in time and frequency domains, respectively. To initiate the process, voice was recorded at the laboratory using Audacity, a voice recording software, at a sampling rate of 44.1 kHz and with ".wav" extension.

The recorded audio was processed in python, using commands from numpy and scipy libraries, while matplotlib was used for the respective plots. The audio file was read into the code and sampling rate and audio data were extracted. As the extracted data was a 2 dimensional array, it was identified that the recorded audio was a 2 channel stereo sound. To convert to mono, row-wise mean of the data was calculated and stored. This conversion changed the data type from 16-bit integer to 64 bit floating point number. The mean data vector was further normalized by dividing each element of the mean data by its maximum absolute value. To plot the variation of amplitude in the time domain, the duration and time vector were calculated. The resulting plot of the normalized amplitude in the time domain is shown in figure 1.

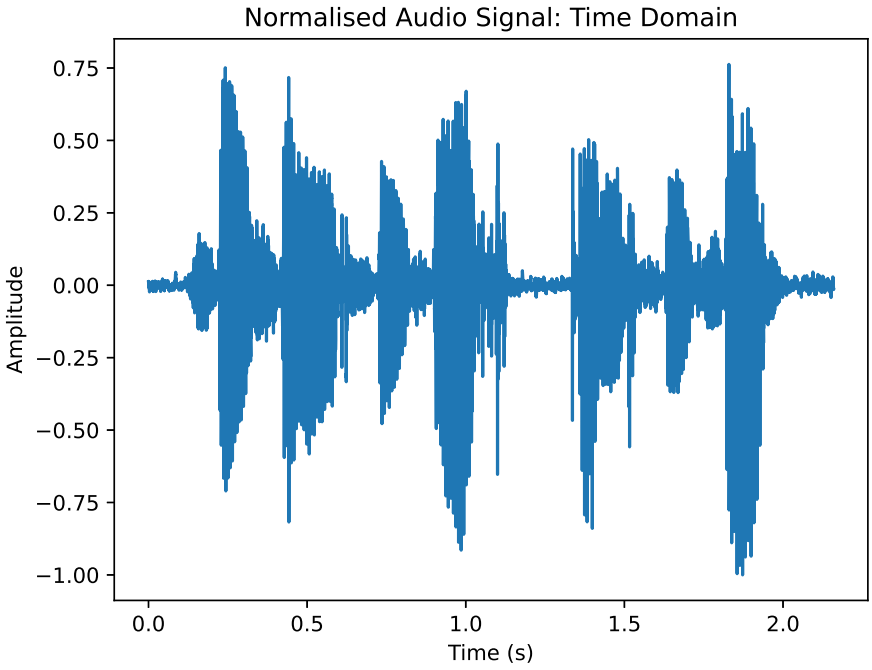

Figure 1. Normalized Audio Signal in the time domain

In order to achieve the variation of amplitudes in the frequency domain, normalized data was transformed using fast fourier transform and its absolute values were calculated. Further, the x axis of the plot was achieved by calculating the value of frequencies corresponding to the transformed data. The data was then plotted in the log scale, for the frequency vector and the absolute values of transformed amplitude data in dB. This plot is shown in figure 2.

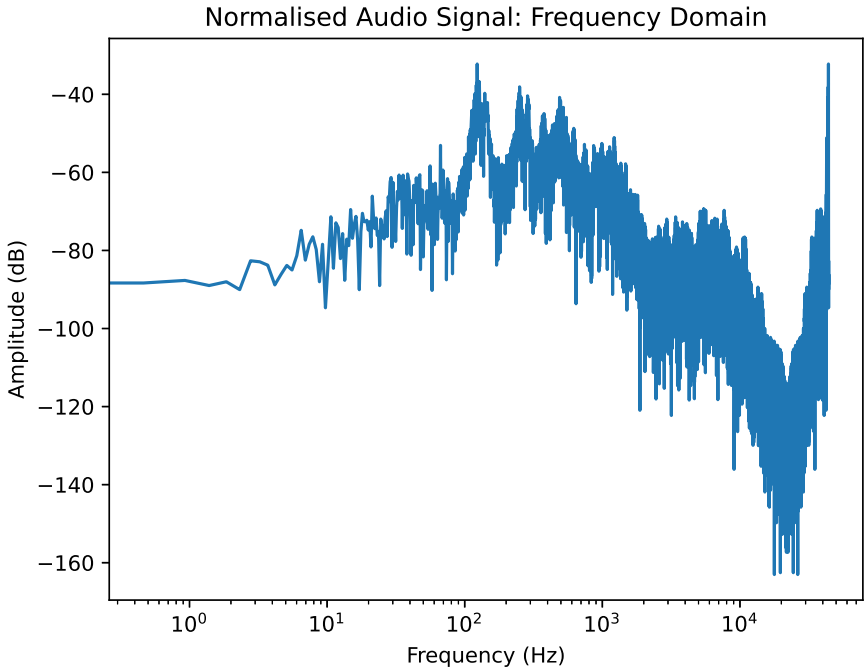

Figure 2. Normalized Audio Signal in the frequency domain

Figure 2 shows various peaks at different frequencies. These are the harmonics of the fundamental frequency. They correspond to different vowels and consonants. In the next section the fundamental frequencies, consonants, and vowels have been detected.

2. QUESTION 2

In this section, it was aimed to identify the various components of the recorded audio, which included fundamental frequency, vowels, consonants and high frequency harmonics. These speech signal components were identified and marked on the Amplitude (in decibels) vs frequency curve in the frequency domain, as shown in figure 3.

Voice recordings are complex signals which comprise of multiple frequencies. In complex signals, fundamental frequency can be identified as the lowest frequency of the signal. Fundamental frequency of the recorded voice, as shown in figure 3, was identified as the first peak, at 123Hz.

All greater frequencies in a signal can be defined as multiples of the fundamental frequency. In the graph, these frequencies have been marked at 246Hz, 369Hz, and 492Hz, respectively. These can distinctly be identified as the peaks following that of the fundamental frequency.

Consonants have higher frequency compared to vowels. As marked in the figure, consonants range from 250Hz - 8000Hz, and the entire speech spectrum ranged from 123Hz - 8000Hz.

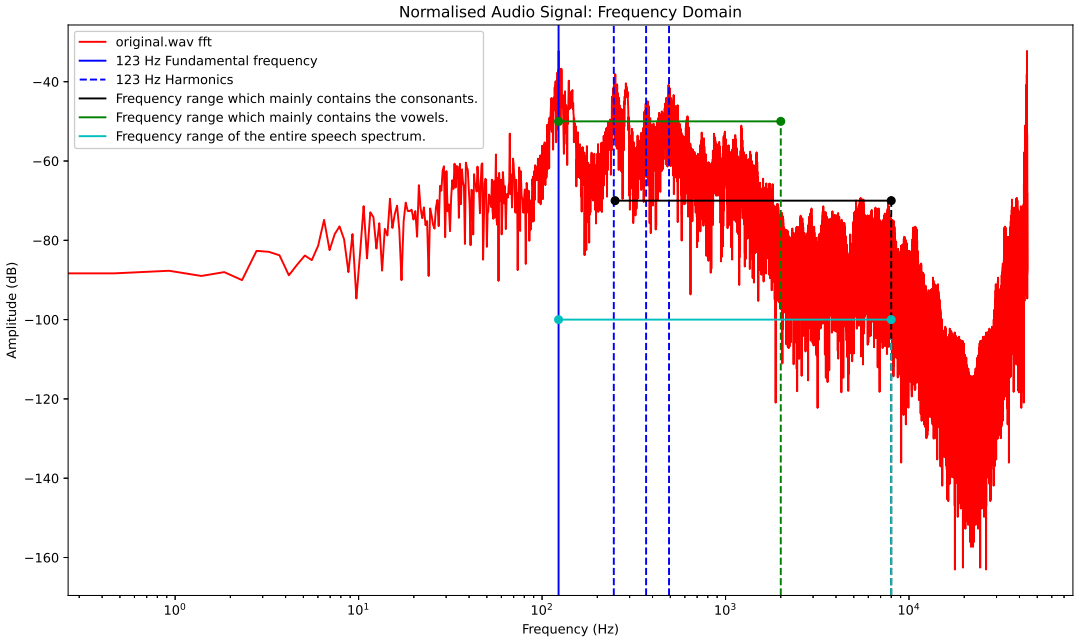

Figure 3. The frequency spectrum of the signal with the marked fundamental, harmonics, consonants and vowels.

3. QUESTION 3

For question three, an intensity against frequency plot of the audio signal shown below in figure 4(A) was analysed in order to evaluate which frequencies in the spectrum carried the most voice information. From this, it was found that the region between 70 Hz to 250 Hz corresponds to the base frequencies of the recorded signal, it contained the highest harmonic voice frequencies in the spectrum. The FFT output for the corresponding frequencies in the range mentioned were then multiplied by a scale factor of 2.25 in order to increase the frequency amplitudes within that region and improve the overall quality of the recorded voice.

The voice enhancer program was created, and applied in the `voiceenhancer.py` Python file. The program reads the audio recording file, carries out the necessary signal processing in order to extract parameters and achieve a normalized signal, followed by conversion into the frequency domain, as discussed in question 1. The achieved fft spectrum was modified and enhanced via a for loop.

The for loop was created to access each data point through indexing, starting from 70 Hz to 250 Hz. Each data point within this range was multiplied by an enhancer factor of 2.25 and this increased the frequency amplitude. The frequency spectrum achieved after enhancing the voice is shown on figure 5(B). Comparing this against the original signal on figure 5(A) (before the signal was enhanced) we can see a slight increase in the dB amplitude, however this is more clear when we look at the intensity plots.

Comparing figure 4(A) and figure 4(B), we can clearly observe that the region of the highest harmonic voice frequencies has increased in intensity amplitude by nearly twice as much. The frequency spectrum that was achieved in such a was used to generate a new time signal with the improved user voice perception.

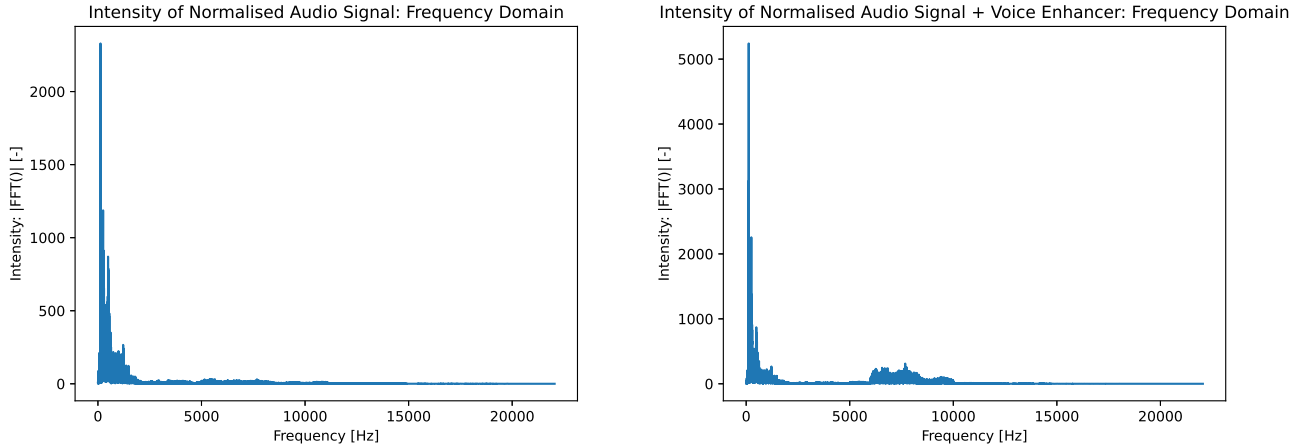

Figure 4. A (Figure on the left): The intensity of the original audio signal in the frequency domain - before enhancing the signal; B (Figure on the right): The intensity of the improved audio signal in the frequency domain - after enhancing the signal

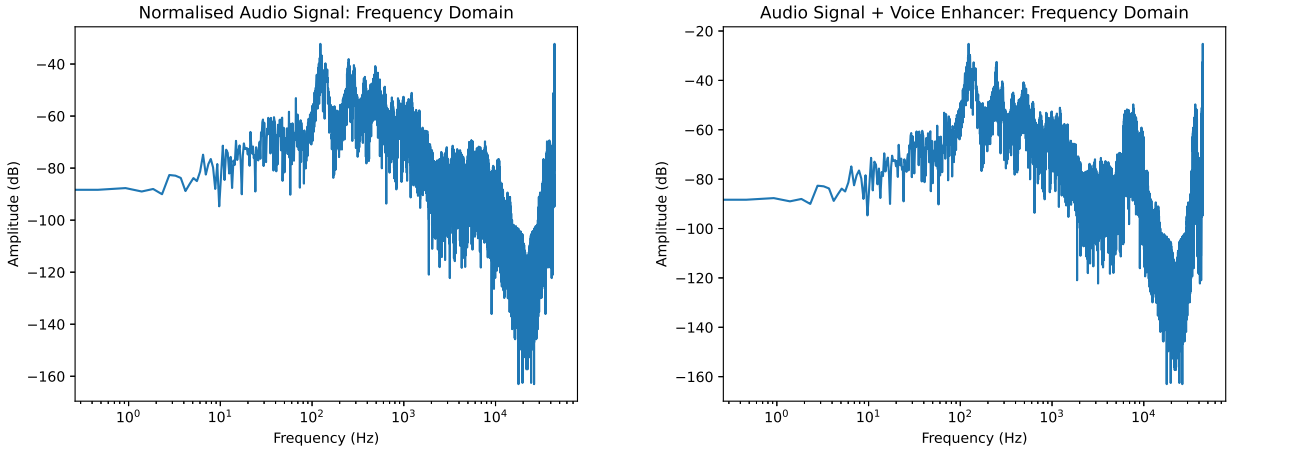

Figure 5. A (Figure on the left): Normalized Audio Signal in the frequency domain; B (Figure on the right): The audio signal with the added voice enhancer in the frequency domain

The enhanced audio data was then written onto a WAV file improved.wav, where we were able to hear an evident improvement in the the voice quality.

4. QUESTION 4

The vowel detector function was created, and applied in the `voweldetector.py` Python file. The python file comprises of a function that accepts an audio file. The audio file is processed to get normalized mono data in the time and frequency domain. This processing is similar to that done in previous parts. The frequency spectrum of the signals was inspected, and it was detected that to be able to differentiate between vowels a and e, we can use the frequencies: 1220 Hz and 460 Hz. At these frequencies, there are significant differences in amplitude between the spectra. By checking the amplitude of those signals at the given frequencies, the vowel was detected. The program does the detection, and prints the vowel if it is detected. Another function, the main function uses detects 2 vowels, vowels a and e, and plots the input file in time and frequency domains.

APPENDIX A. CODES

A.1 voiceenhancer.py

```
from scipy.io import wavfile
from scipy.fft import ifft
import matplotlib.pyplot as plt
import numpy as np


def main():
    # Read the audio files
    audio_file = r'original.wav'
    sample_rate, audio_data = wavfile.read(audio_file)

    # DATA PROCESSING

    # Data is 2 dimensional. Averaging data to get in Mono.
    avg_data = (audio_data[:, 0] + audio_data[:, 1]) / 2
    # Normalizing Data
    normalized_audio_data = avg_data / (np.max(np.abs(avg_data)))
    N = len(normalized_audio_data)
    # Duration and time vector of the audio sample
    duration = N / sample_rate
    time = np.linspace(0, duration, N)

    # Spectrum in Frequency Domain and Frequency Vector
    fft_spectrum = np.fft.fft(normalized_audio_data)
    fft_spectrum_abs = np.abs(fft_spectrum)
    frequency = np.linspace(0, sample_rate, len(avg_data))

    # VOICE ENHANCER QUESTION 3
    enhanced_fft_spectrum = np.fft.fft(avg_data)
    enhanced_fft_spectrum_plot = np.fft.fft(normalized_audio_data)

    for lower, upper, enhancer in ((70, 250, 2.25), (6000, 10000, 10.0)):
        x_lower = abs((frequency - lower)).argmin()
        x_upper = abs((frequency - upper)).argmin()
        enhanced_fft_spectrum[x_lower:x_upper + 1] *= enhancer
        enhanced_fft_spectrum[N - x_upper:N - x_lower + 1] *= enhancer
        enhanced_fft_spectrum_plot[x_lower:x_upper + 1] *= enhancer
        enhanced_fft_spectrum_plot[N - x_upper:N - x_lower + 1] *= enhancer

    improved_audio = ifft(enhanced_fft_spectrum).real
    improved_audio = improved_audio.astype(np.int16)

    wavfile.write("improved.wav", sample_rate, improved_audio)

    # PLOTS

    # Normalized Amplitude vs Time Plot
    plt.figure(1)
    plt.plot(time, normalized_audio_data)
    plt.title("Normalised Audio Signal: Time Domain")
    plt.xlabel("Time (s)")
    plt.ylabel("Amplitude")
    plt.savefig('normalized_time.eps', format='eps')
```

```
    # Normalized Amplitude vs Frequency Plot
    plt.figure(2)
    plt.semilogx(frequency, 20 * np.log10(fft_spectrum_abs / len(fft_spectrum)), label='original')
    plt.title("Normalised Audio Signal: Frequency Domain")
    plt.xlabel("Frequency (Hz)")
    plt.ylabel("Amplitude (dB)")
    plt.savefig('normalized_freq.eps', format='eps')

    # Intensity vs Frequency Plot
    plt.figure(3)
    plt.plot(frequency[:len(frequency) // 2], abs(fft_spectrum[:len(frequency) // 2]))
    plt.title("Intensity of Normalised Audio Signal: Frequency Domain")
    plt.xlabel('Frequency [Hz]')
    plt.ylabel('Intensity: |FFT()| [-]')
    plt.savefig('intensity_original.eps', format='eps')

    # Intensity vs Frequency Plot with Voice Enhancer Added
    plt.figure(4)
    plt.plot(frequency[:len(frequency) // 2], abs(enhanced_fft_spectrum_plot[:len(frequency) // 2]))
    plt.xlabel('Frequency [Hz]')
    plt.ylabel('Intensity: |FFT()| [-]')
    plt.title('Intensity of Normalised Audio Signal + Voice Enhancer: Frequency Domain')
    plt.savefig('intensity_enhanced.eps', format='eps')

    # Normalized Amplitude vs Frequency Plot with Voice Enhancer Added
    plt.figure(5)
    plt.semilogx(frequency, 20 * np.log10((np.abs(enhanced_fft_spectrum_plot)) /
        len(enhanced_fft_spectrum_plot)))
    plt.xlabel('Frequency (Hz)')
    plt.ylabel('Amplitude (dB)')
    plt.title('Audio Signal + Voice Enhancer: Frequency Domain')
    plt.savefig('enhanced_signal_frequency_domain.eps', format='eps')

    # Normalized Amplitude vs Time Plot with Voice Enhancer Added
    plt.figure(6)
    plt.plot(time, improved_audio)
    plt.xlabel('Time (s)')
    plt.ylabel('Amplitude')
    plt.title('Audio Signal + Voice Enhancer: Time Domain')
    plt.savefig('enhanced_signal_time_domain.eps', format='eps')

    plt.show()
    pass


if __name__ == "__main__":
    main()
```

A.2 voweldetector.py

```
import numpy as np
import matplotlib.pyplot as plt
from scipy.io import wavfile


def vowel_detector(audio_file, freq_constant, label):
    # VOWEL SIGNAL PROCESSING
    sample_rate, audio_data = wavfile.read(audio_file)
    avg_data = (audio_data[:, 0] + audio_data[:, 1]) / 2
    normalized_audio_data = avg_data / (np.max(np.abs(avg_data)))
    duration = len(normalized_audio_data) / sample_rate
    time = np.linspace(0, duration, len(normalized_audio_data))
    # PLOTTING THE AUDIO FILE IN THE TIME DOMAIN
    plt.figure()
    plt.plot(time, normalized_audio_data)
    plt.xlabel("Time (s)")
    plt.ylabel("Amplitude")
    plt.title('Normalized Audio Signal, Time Domain: Vowel {}'.format(label))
    plt.savefig('vowel_{}_time.eps'.format(label), format='eps')

    # CALCULATING THE FFT & FREQUENCY OF THE SIGNAL
    fft_spectrum = np.fft.fft(normalized_audio_data)
    fft_spectrum = fft_spectrum / normalized_audio_data.shape[0]
    frequency_axis = np.linspace(0, sample_rate, len(avg_data))
    # FFT CONVERTED TO dB
    fft_spectrum_in_dB = 20 * np.log10(abs(fft_spectrum))

    # PLOTTING THE VOWELS TOGETHER IN THE FREQUENCY DOMAIN
    plt.figure()
    plt.semilogx(frequency_axis, fft_spectrum_in_dB)
    plt.title('Frequency Spectrum of Vowel {}'.format(label))
    plt.xlabel("Frequency")
    plt.ylabel("Amplitude")
    plt.savefig('vowel_{}_frequency.eps'.format(label), format='eps')
    plt.show()
    # PERFORMING LINEAR INTERPOLATION TO EXTRACT THE AMPLITUDE OF THE SIGNALS AT A GIVEN FREQUENCY
    # THE FREQUENCY WAS DETECTED EARLIER BY INTERPRETING THE FREQUENCY SPECTRUM
    amplitude = np.interp(freq_constant, frequency_axis, fft_spectrum_in_dB)
    # With np.interp, we find the corresponding amplitude to the frequency we detected by inspection
    print('amplitude:', amplitude)
    vowel_detected = ''
    if amplitude > -35: # At amplitudes greater than -35, we can safely say we found the corresponding
        vowel
        vowel_detected = label + ' detected'
    # FUNCTION RETURNS A STRING
    return vowel_detected


def main():
    audio_file_a = r'vowel1.wav'
    audio_file_e = r'vowel2.wav'
    frequency_constant_a = 1220
    frequency_constant_e = 460
    # We find the frequency value where we can differentiate between different vowels
    print('Analysing vowel...')
```

```
    vowel = vowel_detector(audio_file_a, frequency_constant_a, 'a')
    print(vowel)
    print('Analysing next vowel...')
    vowel = vowel_detector(audio_file_e, frequency_constant_e, 'e')
    print(vowel)

    pass


if __name__ == "__main__":
    main()
```