Assignment 1: Fourier Transform

Haocheng Li (2825174L)
Yue Hu (2817566H)
Xukai Li (2817896L)

Introduction

In assignment 1, we need to record a WAV file, plot the signal in both time domain and frequency domain and find the vowels' fundamental frequencies and the range of consonants. We also need to use the Fourier Transform to increase the amplitudes of the signal and filtering the noise. Besides, we need to design a python program to detect the vowels.

Task 1

1. Read the audio samples into python. Firstly, we recorded a female's voice. We defined the `wavread` function which uses the open function from the `wave` package to enable the reading of specific audio file.

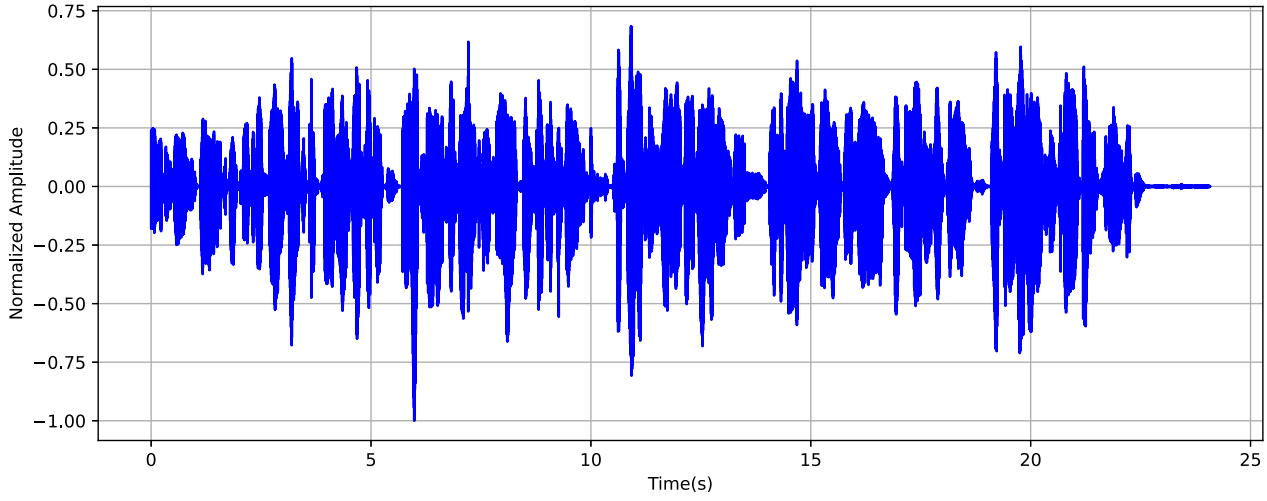
   We used the `getparams` function to get the header files of the audio, including the sampling frequency and the amplitude of the signal at each sampling point. We used the `readframes` function to get the total number of frames of the audio.

   To convert string to integer, we used the `frombuffer` function from the `numpy` package. We divided the list of the number of the frame by the framerate to get a list of time.

   At the end of this part, we returned the amplitudes at each signal points, the list of time (from 0 to the time of the recording), sampling rate and the total number of frames for further use.

2. (a) To plot the audio signal in the time domain, we defined a `namp_vs_time` function. By using the function we have already defined, we can get the amplitudes at each signal point and the corresponding time.

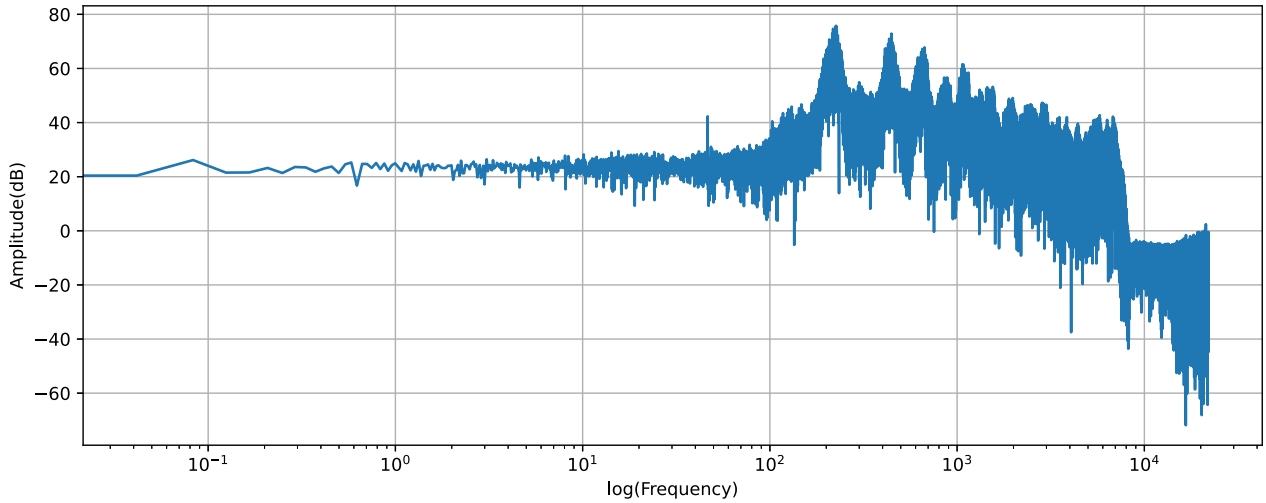
   In order to normalize the amplitude of the signal, we divided each original amplitude by the maximum value of the amplitude. At the end of this part, we saved a figure of the signal in the time domain.



(b) Plot the audio signal in the frequency domain. Use the same method in (a) to normalize the amplitude. After that, perform a Fourier transform on the data to get the signal points in the frequency domain.

According to the Nyquist Frequency, we only need to focus on half of the FFT list.

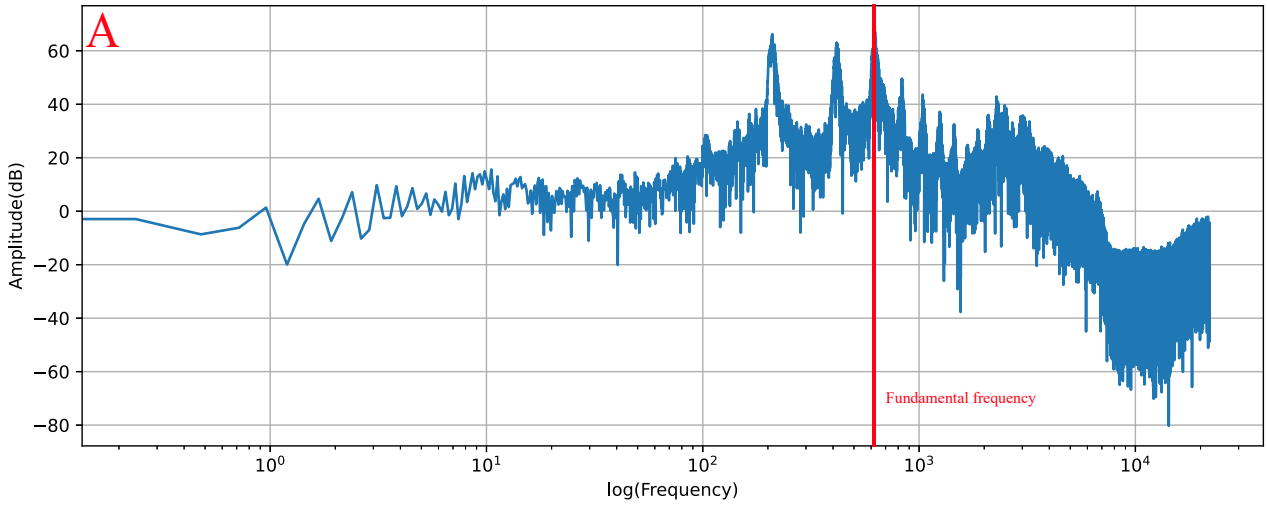
At the end of this part, we saved a figure of the signal in the frequency domain.



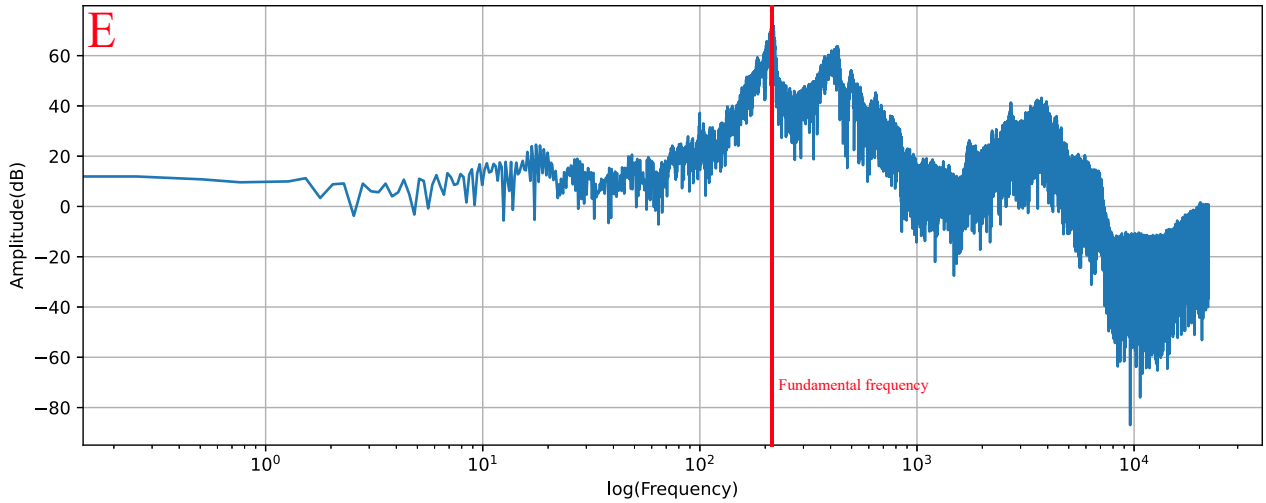
Task 2

1. We plotted spectrograms for each of the five vowels and obtained the fundamental frequencies of the corresponding vowels by determining where the maximum amplitude was located. The fundamental frequencies of `a` and `o` are about 660Hz, `e` and `i` are about 220Hz and `u` is about 430Hz. Here are the spectrograms of these five vowels.

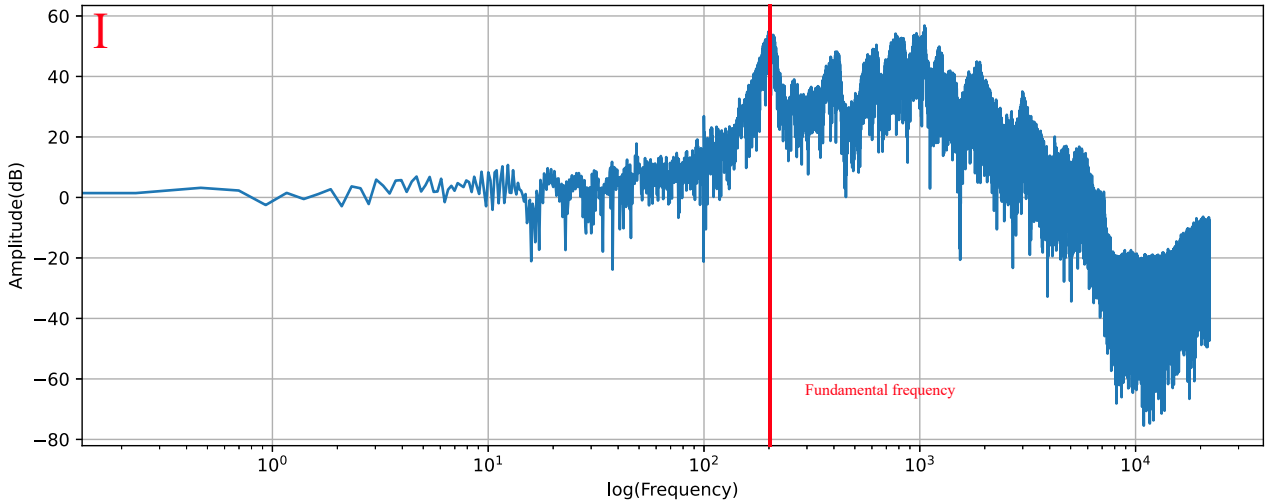
   A:



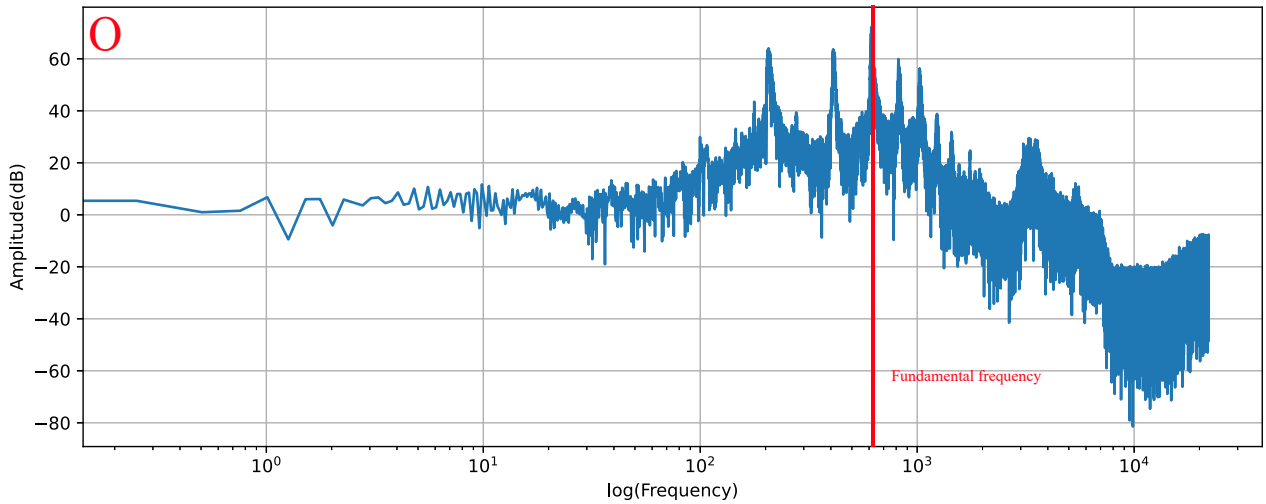
   E:


E
Amplitude(dB)
log(Frequency)
Fundamental frequency

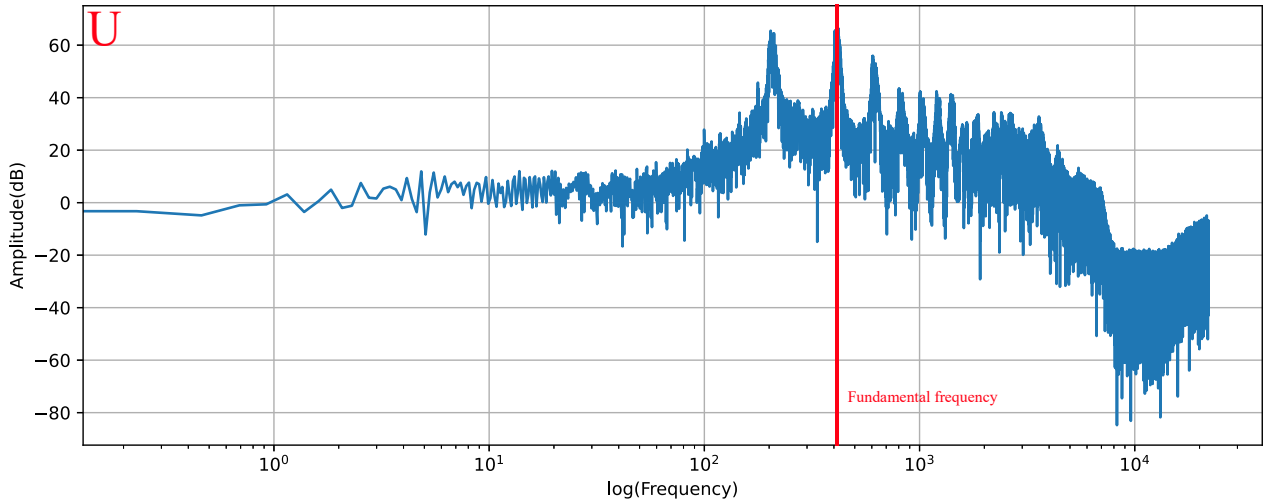
I:


I
Amplitude(dB)
log(Frequency)
Fundamental frequency

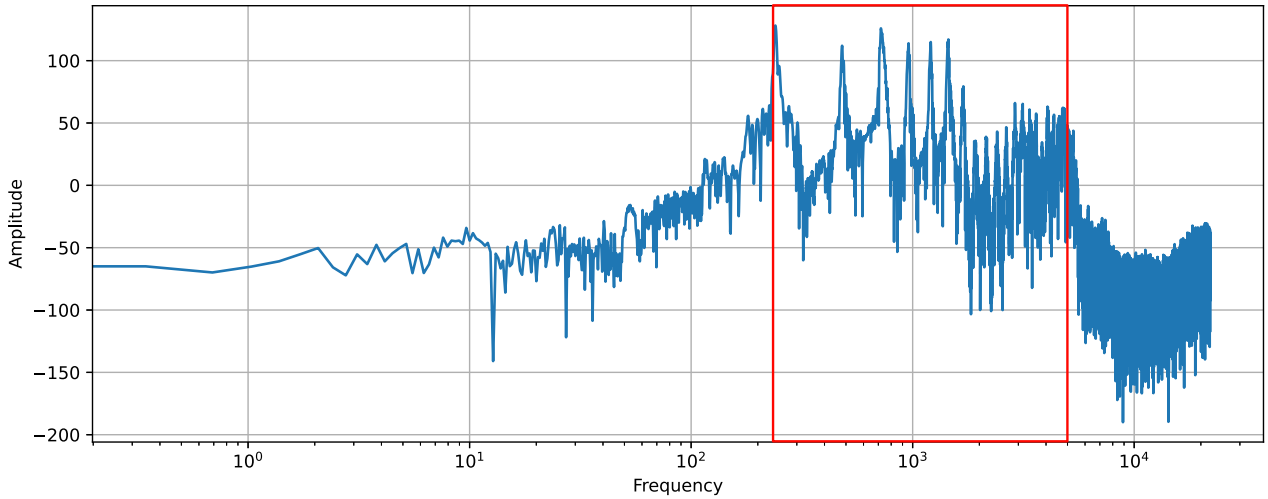
O:


O
Amplitude(dB)
log(Frequency)
Fundamental frequency

U:



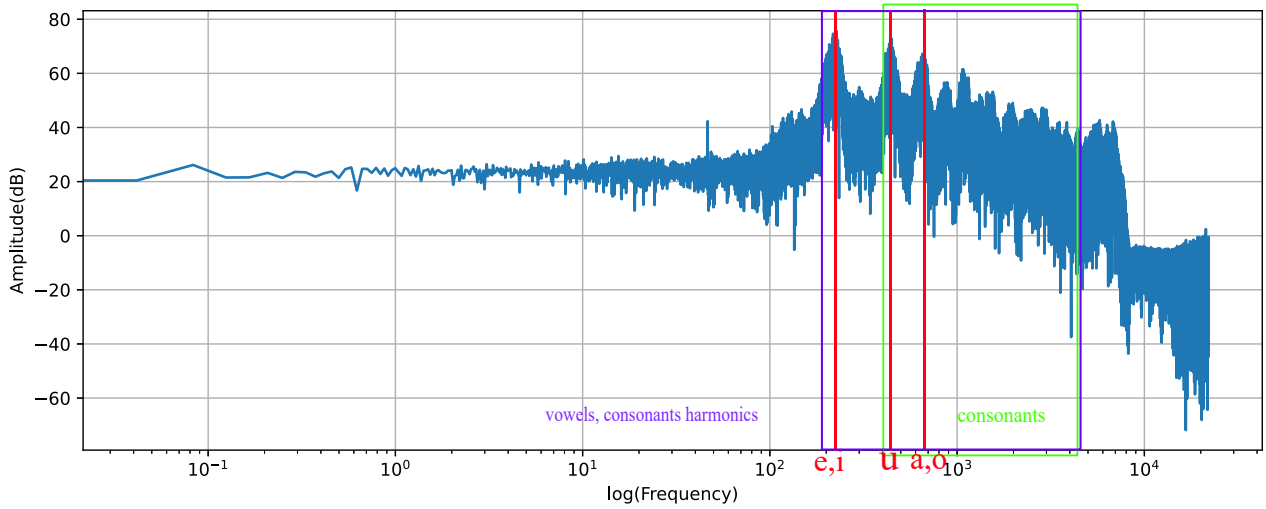
2. We recorded a further portion of the consonants and drew the corresponding spectrograms. For example, here is the spectrogram of consonant b.



By analysing the spectrogram of consonants, we can obtain a frequency range of 500-4800Hz for consonants.

3. To determine the whole speech spectrum, we tried to cut off the first peak from the original audio and the recording is distorted. So we use the first peak as the start of the speech spectrum. From the above analysis we know that the maximum consonants harmonics frequencies is 4800Hz. We set the amplitude of the higher signal to 0 and found that human's voice can still be fully recorded. So we can

determine the end of the whole speech spectrum.



Task 3

We defined a `voice_enhancer` function. Firstly, we set the depth of the Python interpreter stack to the maximum by using `sys.setrecursionlimit(1000000)` to avoid overflow of the C stack. We used the same method in Task 1 to get the params. Then we do the fourior transform of these data by using `fft` function from `scipy.fftpack` package. By using a `for` loop to remove the low and high frequency noise which has frequencies below 130Hz or above 7500Hz.

There would be some white noise if we remove the low decibels noise. And there is no noticeable high decibel noise.

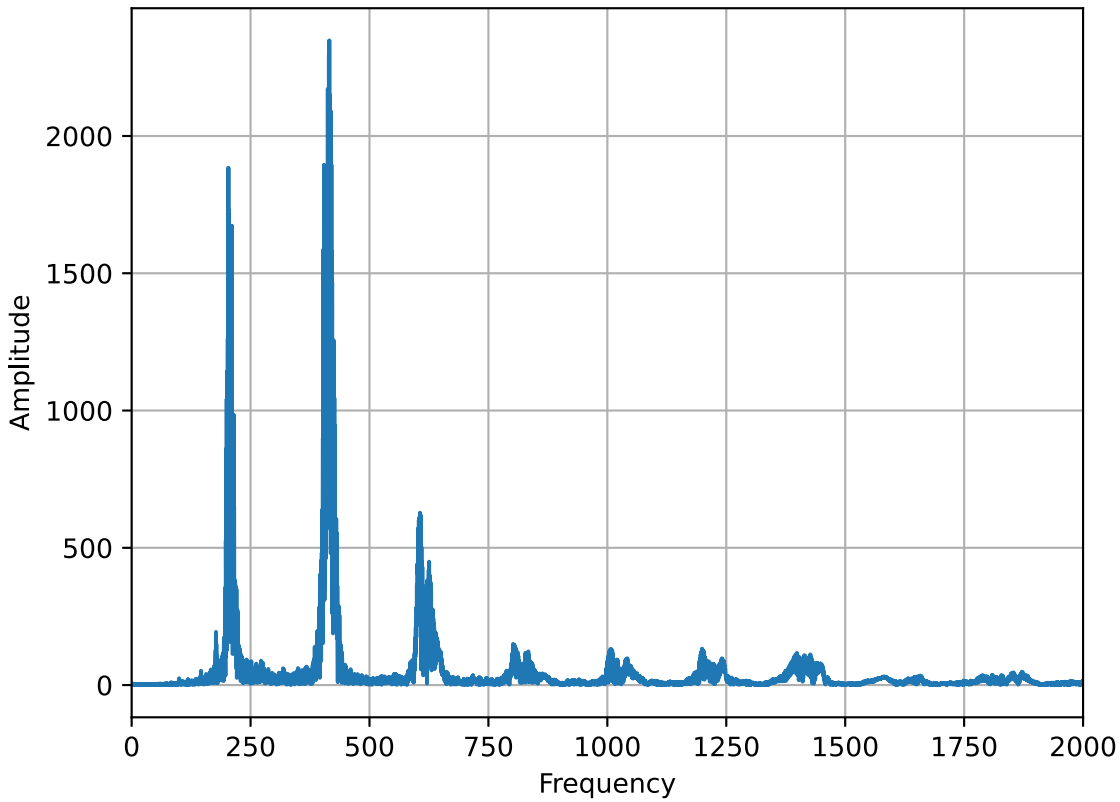
At last, we increased the filtered voice by around eight decibels and generate a new audio file named `improved.wav`.

Task 4

In task 4, we implemented a vowel detector to determine the vowels of `a, e ,i ,o ,u` in the wave file.

Based on task 2 and 3, we had already got the spectrograms and marked the frequency peaks of the five vowels. The vowels have the similar frequency peaks, and the first peak value appeared in the range of 150-250Hz. By comparing with some consonants we recorded, we found that the first peak's amplitude value of the vowels are much bigger than that of consonants. Based on this characteristic we found, we can determine if a voice contain vowels. There are 6 peaks at maximum of the vowels, the second frequency in 300-450Hz, the third peak appears in 550-650Hz, the fourth peak appears in 750-850Hz. The fifth and sixth peaks we figured out but not used, and they are about 950-1050Hz and 1200-1300Hz correspondingly.

For example, the spectrogram of `u` is shown below. We can see the range of the peaks.



In the `vowels.wav` files, we recorded a single vowel only for detecting. By plotting the frequency domain plots, we can get all the frequency ranges we are going to use.

Please note that the voices in the audio are recorded from the voices of one girl in our group and the frequency range may vary due to a number of factors. For example, the fundamental frequencies of male vowels are generally lower than those of females.

We can judge whether the voice contains one vowel (or more vowels) using the value of the six peaks we found. In our code, we took recordings which only contain a vowel each for detecting, and generated the FFT plots. If the amplitude of the first peak is larger than a range of the maximum of the amplitude, we can consider it a vowel. For different audios, parameters need to be adjusted.

Combined with the frequency range we specified and the amplitude value the the peaks, we can get the maximum amplitude peak values of the peaks(i.e. the amplitude of the fundamental frequency), and then by comparing them, we can detect which one the vowel is.

At the end of this part, we returned the recognized vowel string for printing.

Conclusion

In this lab, we learned how to draw the audio signal in time domain and frequency domain. We also learned how the Fourier Transform works in specific cases. We used the knowledge that we learned from the lecture to accomplish this lab. This will help us to get better understanding of Digital Signal Processing.

Appendix

Python code

voice_enhancer.py

```
import sys
import wave

import matplotlib.pyplot as plt
import numpy as np
from scipy.fftpack import fft, ifft
from scipy.io.wavfile import write


# function to read the audio samples into a python application
def wavread(wavpath):
    wavfile = wave.open(wavpath, 'rb')
    params = wavfile.getparams()
    framerate, nframes = params[2], params[3]  # get framerate and nframes
    # Reads and returns at most n frames of audio, as a bytes object.
    wavdata = wavfile.readframes(nframes)
    wavfile.close()
    datause = np.frombuffer(wavdata, dtype=np.short)  # convert string to int
    time = np.arange(nframes) * (1.0 / framerate)  # return one list of time
    return datause, time, framerate, nframes


# function to plot normalised amplitudes vs time using a linear axis in the time
domain and save the image in the current path
def namp_vs_time(wavpath):
    data = wavread(wavpath)
    wavdata, time = data[0], data[1]
    nwavdata = wavdata * 1.0 / (max(abs(wavdata)))  # normalized amplitude
    plt.figure(figsize=(12, 4.5))
    plt.plot(time, nwavdata, color='b')
    plt.xlabel("Time(s)")
    plt.ylabel("Normalized Amplitude")
    plt.grid(True)
    plt.savefig("NormalizedAmplitude_vs_Time.svg", dpi=600,
                format="svg", bbox_inches='tight', pad_inches=0.1)
    return None


# function to plot amplitude (dB) vs frequency using logarithmic axis in the
frequency domain and save the image in the current path
def dB_vs_freq(wavpath):
    data = wavread(wavpath)
    wavdata, framerate, nframes = data[0], data[2], data[3]
    nwavdata = wavdata * 1.0 / max(abs(wavdata))  # normalized amplitude
    ft_signal = fft(nwavdata)  # fft
```

```
    n = nframes // 2  # only need half of the fft list
    T = nframes / framerate
    # the real frequency of the signal at the sampling point
    freq = np.arange(nframes) / T
    plt.figure(figsize=(12, 4.5))
    plt.plot(freq[:n], 20 * np.log((abs(ft_signal[:n]))))
    plt.xscale('log')
    plt.xlabel('Frequency')
    plt.ylabel('Amplitude')
    plt.grid(True)
    plt.savefig("Amplitude(dB)_vs_Log(Frequency).svg", dpi=600,
                format="svg", bbox_inches='tight', pad_inches=0.1)
    return None


# function to increase the voice quality and save in the current path
def voice_enhancer(wavpath):
    sys.setrecursionlimit(1000000)
    # Set the maximum depth of the Python interpreter stack to limit.
    # This limit prevents infinite recursion from causing an overflow of the C
stack and crashing Python.
    data = wavread(wavpath)
    wavdata, time, framerate, nframes = data[:4]
    nwavdata = wavdata * 1.0 / max(abs(wavdata))  # normalized amplitude
    ft_signal = fft(nwavdata)  # fft
    # frequency of signal, the same as dB_vs_freq
    freq = np.arange(nframes) / (nframes / framerate)
    # Removal of low and high frequency noise
    for i in range(nframes):
        if freq[i] < 130 or freq[i] > 4800:
            ft_signal[i] = 0
    # Removal of noise below 40dB, but after this there will be white noise as a
background sound
    # dB = 20 * np.log10(abs(ft_signal))
    # for i in range(nframes):
    #     if dB[i] < 20:
    #         ft_signal[i] = 0
    ft_signal = ft_signal * 6  # increased of around 15dB
    filtered = (ifft(ft_signal)).real * max(abs(wavdata))  # ifft
    # creat 'improved.wav' and save the voice
    write('improved.wav', framerate, filtered.astype(np.int16))
    return None

# function to input and output
def main():
    wavpath = 'original.wav'
    namp_vs_time(wavpath)
    dB_vs_freq(wavpath)
    plt.show()
    voice_enhancer(wavpath)


if __name__ == '__main__':
```

```
    main()
```

voweldetector.py

```
import sys
import wave

import numpy as np
from scipy.fftpack import fft


# function to read the audio samples into a python application
def wavread(wavpath):
    wavfile = wave.open(wavpath, 'rb')
    params = wavfile.getparams()
    framerate, nframes = params[2], params[3]  # get framerate and nframes
    # Reads and returns at most n frames of audio, as a bytes object.
    wavdata = wavfile.readframes(nframes)
    wavfile.close()
    datause = np.frombuffer(wavdata, dtype=np.short)  # convert string to int
    time = np.arange(nframes) * (1.0 / framerate)  # return one list of time
    return datause, time, framerate, nframes


# function to indentify vowels
def voweldector(wavpath):
    sys.setrecursionlimit(1000000)
    # Set the maximum depth of the Python interpreter stack to limit.
    # This limit prevents infinite recursion from causing an overflow of the C
stack and crashing Python.
    data = wavread(wavpath)
    wavdata, time, framerate, nframes = data[:4]
    nwavdata = wavdata * 1.0 / max(abs(wavdata))  # normalized amplitude
    ft_signal = fft(nwavdata)  # fft
    # ft_siganl_copy = ft_signal
    # frequency of signal, the same as dB_vs_freq
    freq = np.arange(nframes) / (nframes / framerate)
    amp1 = []
    amp2 = []
    amp3 = []
    amp4 = []
    # assign new arrays to put all ft_signal[i] value
    # then calculate the np.max()
    unit_max = max(abs(ft_signal))
    for i in range(nframes):
        if 100 <= freq[i] <= 320:
            amp1.append(abs(ft_signal[i]))
        if 280 <= freq[i] <= 480:
            amp2.append(abs(ft_signal[i]))
        if 500 <= freq[i] <= 650:
            amp3.append(abs(ft_signal[i]))
```

```
            if 750 <= freq[i] <= 850:
                amp4.append(abs(ft_signal[i]))
        ft_signal_max1 = np.max(amp1)
        ft_signal_max2 = np.max(amp2)
        ft_signal_max3 = np.max(amp3)
        ft_signal_max4 = np.max(amp4)
        # initial a value to be returned
        vowel = "Not vowel"
        # detection
        if ft_signal_max1 >= 0.45*unit_max:
            vowel = "Is vowel"
            if ft_signal_max2 >= 0.35 * ft_signal_max1:
                if ft_signal_max3 >= 0.3 * ft_signal_max1:
                    if ft_signal_max3 <= 0.5 * ft_signal_max1:
                        vowel = "u"
                    else:
                        vowel = "a"
                else:
                    vowel = "e"
            elif ft_signal_max4 >= 0.8 * ft_signal_max1:
                vowel = "o"
            else:
                vowel = "i"
        return vowel

    # input and output
    def main():
        for i in range(1,6):
            wavpath = 'vowel'+str(i)+'.wav'
            print("Reading file: ",wavpath)
            print("Detected vowel: ",voweldector(wavpath))

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