Report

of

Digital Signal Processing Assignment 1

Team 106

Shaobo Yang (2692028Y)

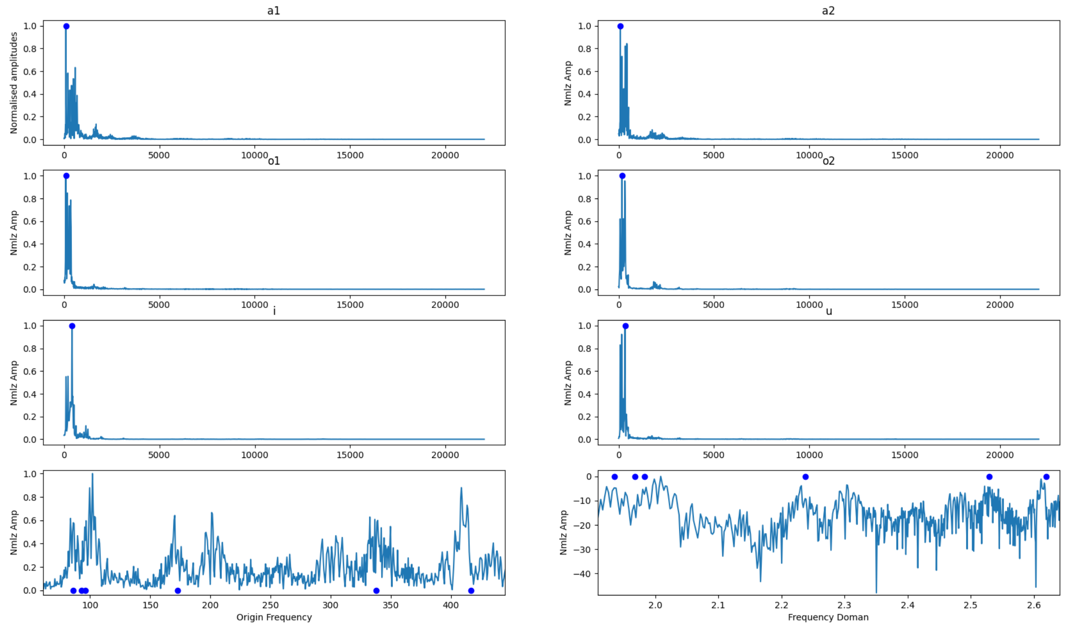
Heting Ying (2647735Y)

Chongzhi Gao(2680340G)

17/10/2021

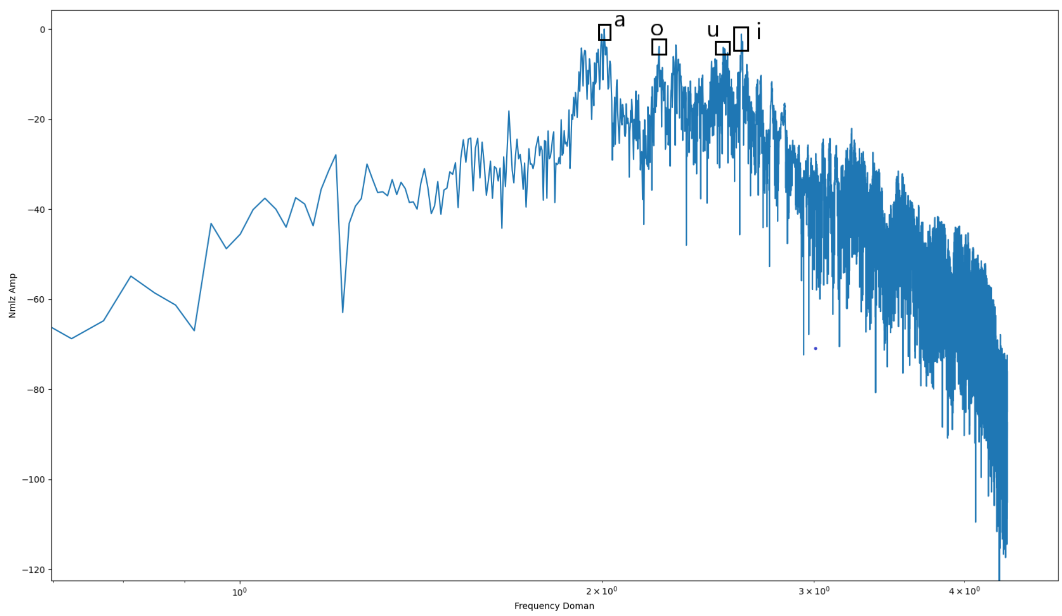
Q2.1 Mark the peaks in the spectrum which correspond to the fundamental frequencies of the vowels spoken.

To get the final answer mark of these series questions, we need to split the source wav file with every single voice, such as split happy(h-a-pp-y) into multiple wav files.



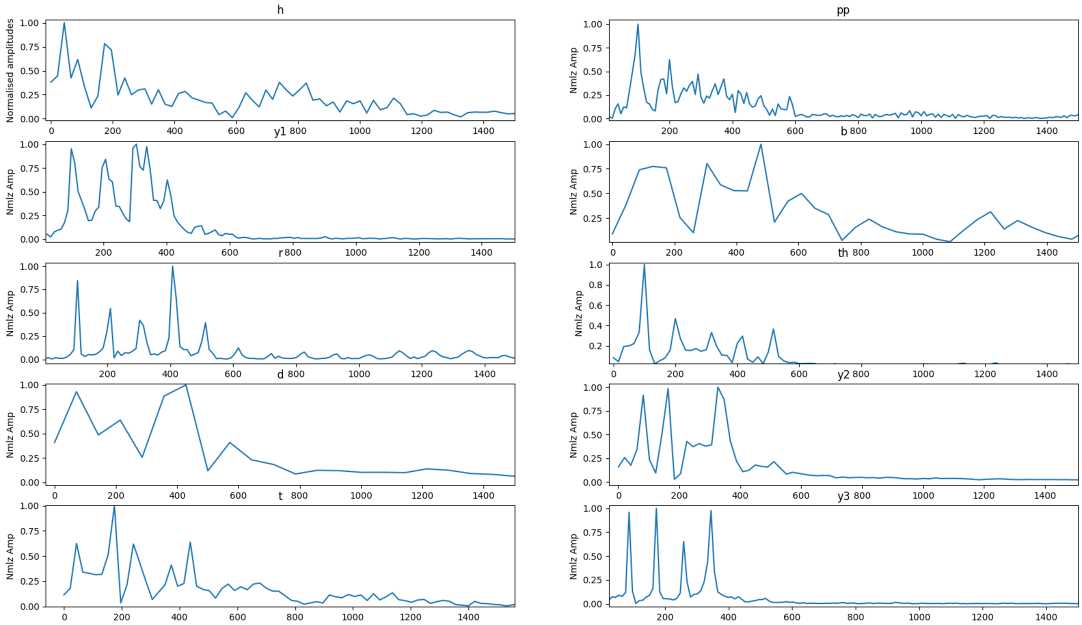
After splitting files, I marked up the peak of every single vowel in their unlogged frequency domain. Calculating the value log10(x) which corresponds with y equals 1 because the fundamental frequency should be the highest one, then I can mark up in the frequency domain.

Answer of Q2.1

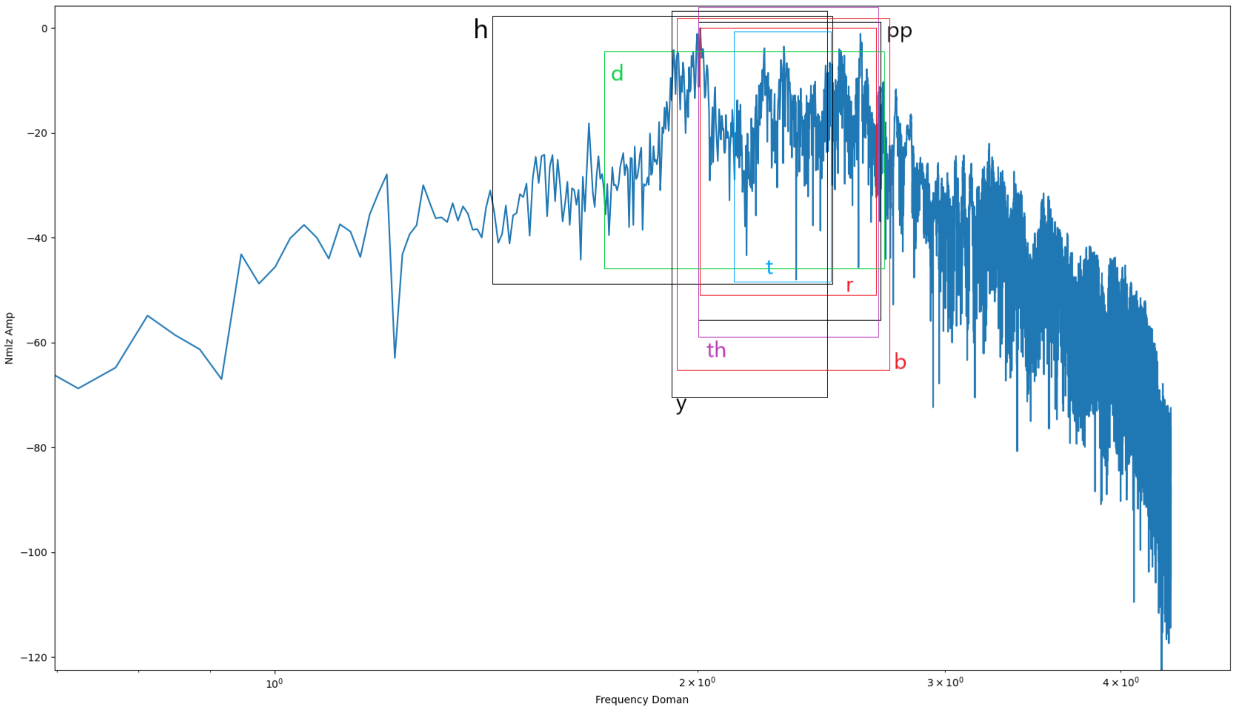


Q2.2 Mark up the frequency range which mainly contains the consonants up to the highest frequencies containing them.

Using the same strategy in the previous question, quote all the consonant hamonic range separately, and then mark those ranges in the whole frequency domain.

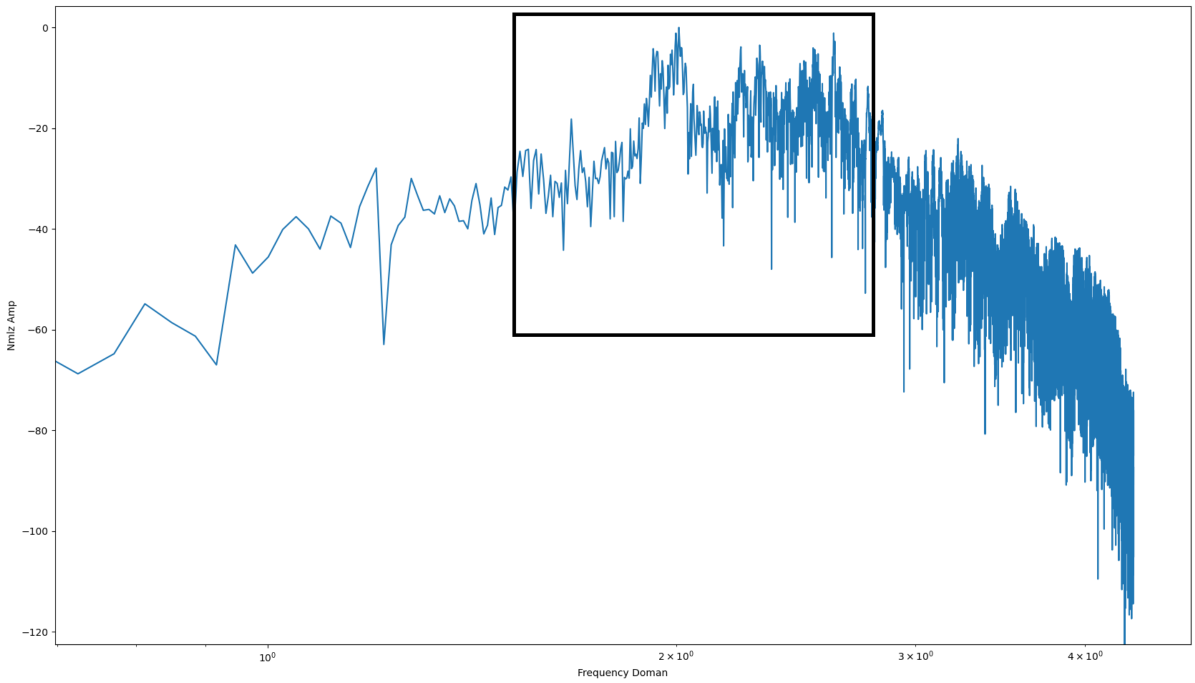


Answer of Q2.2



Q2.3 Mark up the whole speech spectrum containing the vowels, consonants and harmonics.

Answer of Q2.3



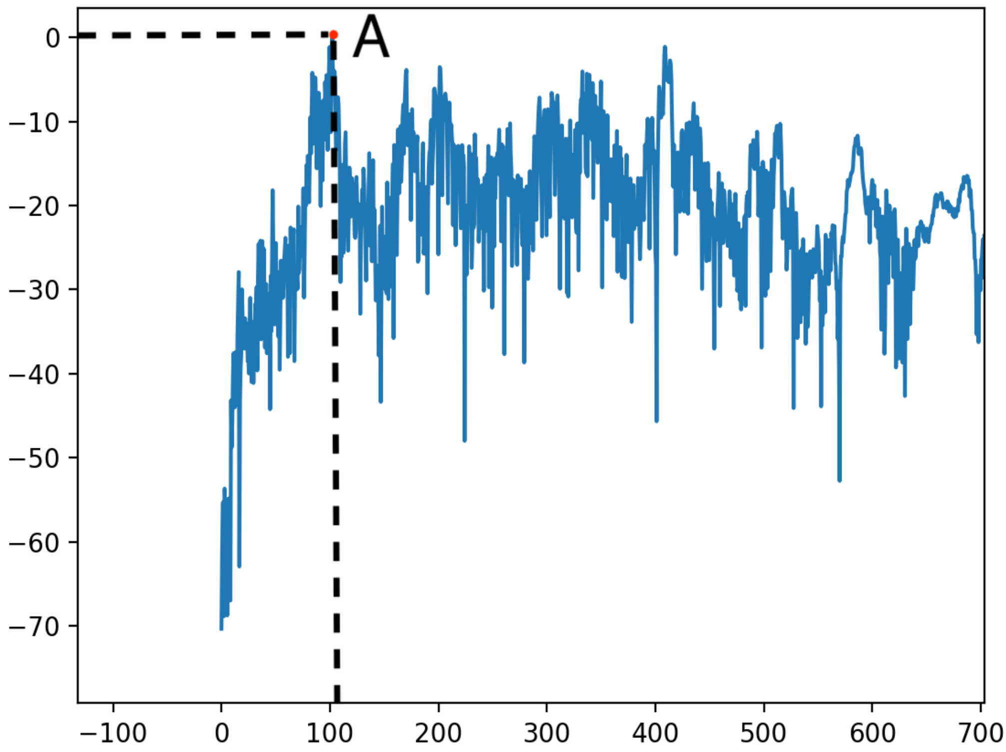
Sum up all the edge values of vowels, consonants and harmonics, then select the minimum and the maximum value to mark up the whole speech.

Q3:

1. Speech quality is improved by using the Fast Fourier Transform to increase the amplitude of speech harmonics.
2. Converting the signal in the time domain to the frequency domain, processing the spectrum, and then converting it back into the time domain.
3. Save the resulting time-domain signal as WAV (16 bits) and include it in the commit.

Q4:

1. Split the audio file into several small audio files (0.1s as unit) to further Fourier Transform.
2. Store the frequency value of peak in every figure after Fourier Transform processing. For example, A is the peak of this figure, so we store the horizontal coordinates of point A, i.e., the frequency value of point A.



3. Define the similarity of sentences' peak frequency and vowels' frequency as $Score = \frac{Freq_{sentence}}{Freq_{vowel}}$, and for those $Score > 1.3$, we use decimals. Thus, we get the dictionary of $Score$ which looks like $\{Vowel_1: Score_1, Vowel_2: Score_2, ..., Vowel_n: Score_n\}$ The more the frequency in the sentence corresponds to the frequency of the vowel, the closer the $Score$ is to 1.
4. Define the distance to 1 as $Distance = 1 - Score$.
5. Define the threshold as 0.04, if $Distance < 0.04$, it represents this vowel in this sentence.

Code is shown in Appendix A, B and C

Appendix A: file_process.py (read audio file and plot figures)

```
from matplotlib import pyplot as plt
import numpy as np
import wave
import struct


def read_file(file_path):
    wave_file = wave.open(file_path, 'rb')

    nchannels = wave_file.getnchannels()
    sample_width = wave_file.getsampwidth()
    framerate = wave_file.getframerate()
    numframes = wave_file.getnframes()

    wave_data = np.zeros(numframes)

    for i in range(numframes):
        val = wave_file.readframes(1)
        if len(val) >2:
            # left = val[0:2]
            right = val[2:4]
            v = struct.unpack('h', right)[0]
            wave_data[i] = v
        else:
            v = struct.unpack('h', val)[0]
            wave_data[i] = v
    return wave_data, nchannels, sample_width, framerate, numframes


def show_time_domain(file_path):
    wave_data, nchannels, sample_width, framerate, numframes =
read_file(file_path)

    time = np.linspace(0, numframes / framerate, numframes)
    max_wave = np.max(wave_data)
    normalized_wave_data = wave_data / max_wave
    return time, normalized_wave_data
```

```
def show_freq_domain(file_path):

    wave_data, nchannels, sample_width, framerate, numframes =
read_file(file_path)
    # print(wave_data)
    abs_fft = np.abs(np.fft.fft(wave_data))
    normalized_abs_fft = abs_fft / len(wave_data)
    half_fft = 2 * normalized_abs_fft[range(int(len(wave_data) / 2))]
    freqs = np.linspace(0, framerate, numframes)
    return freqs[:int(len(freqs) / 2)], 20 * np.log10(half_fft / np.max(half_fft))

def show_log_freq_domain(file_path):
    wave_data, nchannels, sample_width, framerate, numframes =
read_file(file_path)
    # print(wave_data)
    abs_fft = np.abs(np.fft.fft(wave_data))
    normalized_abs_fft = abs_fft / len(wave_data)
    half_fft = 2 * normalized_abs_fft[range(int(len(wave_data) / 2))]
    freqs = np.linspace(0, framerate, numframes)
    return np.log10(freqs[:int(len(freqs) / 2)]), 20 * np.log10(half_fft /
np.max(half_fft))

def show_org_freq_domain(file_path):
    epsilon = 1e-30
    wave_data, nchannels, sample_width, framerate, numframes =
read_file(file_path)
    # print(wave_data)
    abs_fft = np.abs(np.fft.fft(wave_data))
    normalized_abs_fft = abs_fft / len(wave_data)
    half_fft = 2 * normalized_abs_fft[range(int(len(wave_data) / 2))]
    freqs = np.linspace(0, framerate, numframes)
    return freqs[:int(len(freqs) / 2)], half_fft / np.max(half_fft)

if __name__ == '__main__':


######################################################################
##############################
```

```
#2.1
#import splitted wav files
#vowels
x_a1, y_a1 = show_org_freq_domain('asset/seperate_hb/0.196.wav')
x_a2, y_a2 = show_org_freq_domain('asset/seperate_hb/0.875.wav')
x_o1, y_o1 = show_org_freq_domain('asset/seperate_hb/1.183.wav')
x_o2, y_o2 = show_org_freq_domain('asset/seperate_hb/1.373.wav')
x_i, y_i = show_org_freq_domain('asset/seperate_hb/0.593.wav')
x_u, y_u = show_org_freq_domain('asset/seperate_hb/1.425.wav')
x_org, y_org = show_org_freq_domain('original.wav')
x_log, y_log = show_log_freq_domain('original.wav')

#plot Frequency domain
plt.figure(figsize=(40, 20))

plt.subplot(4, 2, 1)
plt.title("a1")
plt.plot(x_a1, y_a1)
plt.plot(96.2, 1, 'bo')
#a1 96.2
# plt.xlabel('A1-Frequency')
plt.ylabel('Normalised amplitudes')

plt.subplot(4, 2, 2)
plt.title("a2")
plt.plot(x_a2 , y_a2 )
plt.plot(86.2, 1, 'bo')
#a2 86.2
# plt.xlabel('A2-Frequency')
plt.ylabel('Nmlz Amp')

plt.subplot(4, 2, 3)
plt.title("o1")
plt.plot(x_o1 , y_o1 )
plt.plot(92.8, 1, 'bo')
#o1 92.8
# plt.xlabel('O1-Frequency')
plt.ylabel('Nmlz Amp')

plt.subplot(4, 2, 4)
```

```
    plt.title("o2")
    plt.plot(x_o2 , y_o2 )
    plt.plot(173, 1, 'bo')
    #o2 173
    # plt.xlabel('O2-Frequency')
    plt.ylabel('Nmlz Amp')

    plt.subplot(4, 2, 5)
    plt.title("i")
    plt.plot(x_i , y_i )
    plt.plot(416.8, 1, 'bo')
    #i 416.8
    # plt.xlabel('I-Frequency')
    plt.ylabel('Nmlz Amp')

    plt.subplot(4, 2, 6)
    plt.title("u")
    plt.plot(x_u , y_u )
    plt.plot(338, 1, 'bo')
    #u 338
    # plt.xlabel('U-Frequency')
    plt.ylabel('Nmlz Amp')

    plt.subplot(4, 2, 7)
    plt.plot(x_org, y_org)
    #ORG-- #a1 [96.2,0.243] #a2 [86.2,0.577] #o1 [92.8,] #o2 [173,] #i [416.8,]
#u [338,]
    plt.plot([96.2, 86.2, 92.8, 173, 416.8, 338],[0,0,0,0,0,0], 'bo')
    #plt.plot([np.log10(96.2)],[20*np.log10(0.243)], 'bo')
    plt.xlabel('Unlogged Frequency')
    plt.ylabel('Nmlz Amp')

    plt.subplot(4, 2, 8)
    plt.plot(x_log, y_log)
    #ORG-- #a1 [96.2,0.243] #a2 [86.2,0.577] #o1 [92.8,] #o2 [173,] #i [416.8,]
#u [338,]
    #plt.plot([np.log10(96.2), np.log10(86.2), np.log10(92.8), np.log10(173),
np.log10(416.8), np.log10(338)],[0,0,0,0,0,0], 'bo')
    #plt.plot([np.log10(96.2)],[20*np.log10(0.243)], 'bo')
    plt.xlabel('Frequency Doman')
```

```
plt.ylabel('Nmlz Amp')
plt.show()


########################################################################
###########################################
#2.2
#import split wav file
#consonant
x_h, y_h = show_org_freq_domain('asset/seperate_hb/0.15.wav')
x_pp, y_pp = show_org_freq_domain('asset/seperate_hb/0.30.wav')
x_y1, y_y1 = show_org_freq_domain('asset/seperate_hb/0.41.wav')
x_b, y_b = show_org_freq_domain('asset/seperate_hb/0.57.wav')
x_r, y_r = show_org_freq_domain('asset/seperate_hb/0.641.wav')
x_th, y_th = show_org_freq_domain('asset/seperate_hb/0.727.wav')
x_d, y_d = show_org_freq_domain('asset/seperate_hb/0.861.wav')
x_y2, y_y2 = show_org_freq_domain('asset/seperate_hb/0.991.wav')
x_t, y_t = show_org_freq_domain('asset/seperate_hb/t.wav')
x_y3, y_y3 = show_org_freq_domain('asset/seperate_hb/1.28.wav')

plt.figure(figsize=(15, 15))
plt.subplot(5, 2, 1)
plt.title("h")
plt.plot(x_h, y_h)
# plt.plot(96.2, 1, 'bo')
#h
# plt.xlabel('H-Frequency')
plt.ylabel('Normalised amplitudes')

plt.subplot(5, 2, 2)
plt.title("pp")
plt.plot(x_pp , y_pp)
# plt.plot(92.8, 1, 'bo')
# plt.xlabel('PP-Frequency')
plt.ylabel('Nmlz Amp')

plt.subplot(5, 2, 3)
plt.title("y1")
plt.plot(x_y1 , y_y1 )
# plt.plot(92.8, 1, 'bo')
```

```
# plt.xlabel('Y1-Frequency')
plt.ylabel('Nmlz Amp')

plt.subplot(5, 2, 4)
plt.title("b")
plt.plot(x_b, y_b )
# plt.plot(173, 1, 'bo')
# plt.xlabel('B-Frequency')
plt.ylabel('Nmlz Amp')

plt.subplot(5, 2, 5)
plt.title("r")
plt.plot(x_r, y_r)
# plt.plot(416.8, 1, 'bo')
# plt.xlabel('R-Frequency')
plt.ylabel('Nmlz Amp')

plt.subplot(5, 2, 6)
plt.title("th")
plt.plot(x_th , y_th )
# plt.plot(338, 1, 'bo')
# plt.xlabel('TH-Frequency')
plt.ylabel('Nmlz Amp')

plt.subplot(5, 2, 7)
plt.title("d")
plt.plot(x_d, y_d)
# plt.plot(416.8, 1, 'bo')
# plt.xlabel('D-Frequency')
plt.ylabel('Nmlz Amp')

plt.subplot(5, 2, 8)
plt.title("y2")
plt.plot(x_y2 , y_y2 )
# plt.plot(338, 1, 'bo')
# plt.xlabel('Y2-Frequency')
plt.ylabel('Nmlz Amp')

plt.subplot(5, 2, 9)
plt.title("t")
```

```
    plt.plot(x_t, y_t)
    # plt.plot(416.8, 1, 'bo')
    # plt.xlabel('T-Frequency')
    plt.ylabel('Nmlz Amp')

    plt.subplot(5, 2, 10)
    plt.title("y3")
    plt.plot(x_y3 , y_y3 )
    # plt.plot(338, 1, 'bo')
    # plt.xlabel('R-Frequency')
    plt.ylabel('Nmlz Amp')
    plt.show()

    plt.plot(x_log[:1500], y_log[:1500])
    #ORG-- #a1 [96.2,0.243] #a2 [86.2,0.577] #o1 [92.8,] #o2 [173,] #i [416.8,]
#u [338,]
    #plt.plot([np.log10(44), np.log10(86.2), np.log10(92.8), np.log10(173),
np.log10(416.8), np.log10(338)],[0,0,0,0,0,0], 'bo')
    #plt.plot([np.log10(96.2)],[20*np.log10(0.243)], 'bo')
    plt.xlabel('Frequency Doman')
    plt.ylabel('Nmlz Amp')
    plt.show()
```

Appendix B:voice_enhancer.py (improve audio ability via FFT)

```
from file_process import *
import scipy.io.wavfile as wavfile

def find_highst_freq():
    """
    Find the highest harmonic voice frequencies
    :return: The value of Hertz corresponding to the highest frequency in
freq_domain
     """
    x_freq, y_freq = show_freq_domain('original.wav')
    maxAmp = np.argmax(y_freq)
    return maxAmp

def enhance(x_freq, start_freq, end_freq):
    """
    Improve sound quality and reduce noise, and write the audio file after
enhanced named 'enhance.wav'
    :param x_freq: X axis of frequency domain diagram
    :param start_freq: The beginning of the highest harmonic voice frequencies
    :param end_freq: The end of the highest harmonic voice frequencies
     """

    # Delimit human voice area and noise area
    bounds = list()

    for i in range(len(x_freq)):
        # 85 is the lowest Hertz value of human sound
        if x_freq[i] > 20:
            bounds.append(i)
            break

    for i in range(len(x_freq)):
        if x_freq[i] > start_freq:
            bounds.append(i)
            break

    for i in range(len(x_freq)):
        if x_freq[i] > end_freq:
```

```
            bounds.append(i)
            break

    for i in range(len(x_freq)):
        # 2000 is the highest Hertz value of human sound
        if x_freq[i] > 2000:
            bounds.append(i)
            break

    # Read wavfile and fft operation
    wave_data, nchannels, sample_width, framerate, numframes =
read_file('original.wav')
    wave_data_fre = np.fft.fft(wave_data)

    start = bounds[0]
    start_voice = bounds[1]
    end_voice = bounds[2]
    end = bounds[3]

    # Increase the region of the highest harmonic voice frequency amplitudes
    wave_data_fre[start_voice:end_voice] =
wave_data_fre[start_voice:end_voice] * 10

wave_data_fre[int(len(wave_data_fre)-end_voice):int(len(wave_data_fre)-start
_voice)] =
wave_data_fre[int(len(wave_data_fre)-end_voice):int(len(wave_data_fre)-start
_voice)] * 10

    # Lower the frequency of other parts
    wave_data_fre[start:start_voice] = wave_data_fre[start:start_voice] / 2

wave_data_fre[int(len(wave_data_fre)-start_voice):int(len(wave_data_fre)-star
t)] =
wave_data_fre[int(len(wave_data_fre)-start_voice):int(len(wave_data_fre)-star
t)] / 2

    wave_data_fre[end:-1] = wave_data_fre[end:-1] / 2
    wave_data_fre[1:int(len(wave_data_fre)-end)] =
wave_data_fre[1:int(len(wave_data_fre)-end)] / 2
```

```
    # Ifft operation and write file
    after_enhance = np.fft.ifft(wave_data_fre)
    clr = np.real(after_enhance)
    enhanced_audio = clr.astype(np.int16)
    wavfile.write('improved.wav', framerate, enhanced_audio)

if __name__ == '__main__':

    x_freq, y_freq = show_freq_domain('original.wav')
    enhance(x_freq, 85, find_highst_freq())
```

Appendix C:voweldetector.py (detect vowels in sentences)

```
import math
import os
import shutil

from file_process import *


def divide_wav_file(file_path, time_slot):
    """
    Splitting audio files into time_slot spaced files and write into 'temp' folder to
further FFT operation
    :param file_path: Original wav file path
    :param time_slot: Time period to be intercepted, second as unit
    :return: the relative path of the audio file
    """
    # open original wav file
    f = wave.open(file_path, 'rb')
    params = f.getparams()
    nchannels, sampwidth, framerate, nframes = params[:4]
    str_data = f.readframes(nframes)
    f.close()

    wave_data = np.frombuffer(str_data, dtype=np.short)
    # data process depending on the number of channels
    if nchannels > 1:
        wave_data.shape = -1, 2
        wave_data = wave_data.T
        temp_data = wave_data.T
    else:
        wave_data = wave_data.T
        temp_data = wave_data.T

    # Number of frames in one time slot
    frames_num = framerate * time_slot
    # Number of slot
    slot_num = nframes / frames_num
    frames_num_int = int(frames_num)
```

```
    # Determine if 'temp' folder exists
    if not os.path.exists('temp/'):
        # Create 'temp' folder if not exist
        os.makedirs('temp/')

    for j in range(int(math.ceil(slot_num))):
        current_file_name = "temp/slot" + "-" + str(j) + ".wav"
        current_slot_data = temp_data[int(frames_num_int *
j):int(frames_num_int * j + frames_num)]
        current_slot_data.shape = 1, -1
        current_slot_data = current_slot_data.astype(np.short)
        f = wave.open(current_file_name, 'wb')
        f.setnchannels(nchannels)
        f.setsampwidth(sampwidth)
        f.setframerate(framerate)
        f.writeframes(current_slot_data.tobytes())
        f.close()
    return file_path


def fft_operation(file_path):
    """
    FFT operation
    :param file_path: Original wav file path
    :return: freqs, 20 * log10(fft/max_fft), name
    """
    wave_data, nchannels, sample_width, framerate, numframes =
read_file(file_path)

    abs_fft = np.abs(np.fft.fft(wave_data))
    normalized_abs_fft = abs_fft / len(wave_data)
    half_fft = 2 * normalized_abs_fft[range(int(len(wave_data) / 2))]
    freqs = np.linspace(0, framerate, numframes)

    return freqs[:int(len(freqs) / 2)], 20 * np.log10(half_fft / np.max(half_fft)),
file_path[file_path.rfind(
        '/') + 1:file_path.find('.')]


def compare_freqs(file_path):
```

```
"""
Compare frequency between sentences and vowels to generate the score
:return:
"""
# read files under the temp folder
path = r"temp"
files = os.listdir(path)
files = [path + "/" + f for f in files if f.endswith('.wav')]

# Store the highest frequency of current file
sen_freqs = list()
for i in range(len(files)):
    freq_i, slot_fft_i, _ = fft_operation(files[i])
    sen_freqs.append(freq_i[int(np.argmax(slot_fft_i))])
sen_freqs.sort()

# print(sen_freqs)

# Store frequency of different vowels
vowel_freqs = dict()

# FFT operation of different vowel audio files
freq_ei, slot_fft_ei, name_ei = fft_operation('asset/vowel/ei.wav')
freq_i, slot_fft_i, name_i = fft_operation('asset/vowel/i.wav')
freq_er, slot_fft_er, name_er = fft_operation('asset/vowel/er.wav')
freq_wu, slot_fft_wu, name_wu = fft_operation('asset/vowel/wu.wav')
freq_u, slot_fft_u, name_u = fft_operation('asset/vowel/u.wav')
freq_o, slot_fft_o, name_o = fft_operation('asset/vowel/o.wav')
freq_ai, slot_fft_ai, name_ai = fft_operation('asset/vowel/ai.wav')
freq_uh, slot_fft_uh, name_uh = fft_operation('asset/vowel/uh.wav')
freq_e, slot_fft_e, name_e = fft_operation('asset/vowel/e.wav')

vowel_freqs['ei'] = (freq_ei[int(np.argmax(slot_fft_ei))])
vowel_freqs['i'] = (freq_i[int(np.argmax(slot_fft_i))])
vowel_freqs['er'] = (freq_er[int(np.argmax(slot_fft_er))])
vowel_freqs['wu'] = (freq_wu[int(np.argmax(slot_fft_wu))])
vowel_freqs['u'] = (freq_u[int(np.argmax(slot_fft_u))])
vowel_freqs['o'] = (freq_o[int(np.argmax(slot_fft_o))])
vowel_freqs['ai'] = (freq_ai[int(np.argmax(slot_fft_ai))])
vowel_freqs['uh'] = (freq_uh[int(np.argmax(slot_fft_uh))])
```

```
    vowel_freqs['e'] = (freq_e[int(np.argmax(slot_fft_e))])

    # print(vowel_freqs)

    '''
    Store the score which defines as 1 - distance between different sentences'
highest frequency and vowels' frequency
    e.g.
        A: sentence_1's highest frequency
        B: vowel /æ/'s frequency
        score = 1 - A / B
    '''
    scores = dict()
    for k, v, in vowel_freqs.items():
        temp_list = list()
        for idx in range(len(sen_freqs)):
            value = sen_freqs[idx] / v
            '''
            Due to the fact that some frequency is higher than 1e2 even more, so
we define that if frequency is higher
            than 1.3e2, we use the decimals
            '''
            if value > 1.3:
                value = math.modf(value)[0]
            temp_list.append(value)

        # Get the score
        for i in range(len(temp_list)):
            if temp_list[i] < 1:
                temp_list[i] = 1 - temp_list[i]
            else:
                temp_list[i] = temp_list[i] - 1
        temp_list.sort(reverse=False)
        scores[k] = temp_list[0]

    # print('score(distance to 100%): ', sorted(scores.items(), key=lambda x:
x[1], reverse=False))

    # We set threshold as 4%, if distance to 100% is lower than threshold, it
means the vowel is in the sentence
```

```
    threshold = 0.04
    output_vowels = dict()
    score_list = sorted(scores.items(), key=lambda x: x[1], reverse=False)
    for i in range(len(score_list)):
        if float(score_list[i][1]) < threshold:
            output_vowels[score_list[i][0]] = score_list[i][1]
    output_vowels_list = list(output_vowels.keys())

    # Phonetic transcription of vowels
    vowel_tostring = dict()
    vowel_tostring['ei'] = '/e/'
    vowel_tostring['i'] = '/ɪ/'
    vowel_tostring['er'] = '/ɜː/'
    vowel_tostring['wu'] = '/u:/'
    vowel_tostring['u'] = '/juː/'
    vowel_tostring['o'] = '/ɔ:/'
    vowel_tostring['ai'] = '/aɪ/'
    vowel_tostring['uh'] = '/ʌ/'
    vowel_tostring['e'] = '/iː/'
    # print(list(output_vowels.keys()))
    print('\nThese vowels are in sentences (' + file_path + '): ', end='')
    for i in range(len(output_vowels_list)):
        print(vowel_tostring[output_vowels_list[i]], end=' ')

    return [(freq_ei, slot_fft_ei, name_ei), (freq_i, slot_fft_i, name_i), (freq_er,
slot_fft_er, name_er),
            (freq_wu, slot_fft_wu, name_wu), (freq_u, slot_fft_u, name_u), (freq_o,
slot_fft_o, name_o),
            (freq_ai, slot_fft_ai, name_ai), (freq_uh, slot_fft_uh, name_uh),
(freq_e, slot_fft_e, name_e)]


def show_figures(time_and_frequency):
    """
    Show figures of vowels' frequency
    :param time_and_frequency: list, contains (frequency, fft_data, name)
    :return: None
    """
    length = len(time_and_frequency)
    row = math.floor(math.sqrt(length))
```

```
    col = math.ceil(math.sqrt(length))
    for i in range(length):
        # Plot vowels' frequency
        plt.subplot(row, col, i + 1)
        plt.plot(time_and_frequency[i][0], time_and_frequency[i][1])
        plt.title(time_and_frequency[i][-1])
        plt.xlim(0, 600)

    plt.show()


if __name__ == '__main__':
    # 2 arguments: audio file path, time interval
    # Sentence 1 detection
    freq_list_1 = compare_freqs(divide_wav_file('vowel1.wav', 0.1))
    # Remove temp file
    shutil.rmtree('temp/')

    # Sentence 2 detection
    freq_list_2 = compare_freqs(divide_wav_file('vowel2.wav', 0.1))
    # Remove temp file
    shutil.rmtree('temp/')
```