Assignment 1, Digital Signal Processing: Fourier Transform

Chengzhi Fu (2168628)

Introduction

This assignment helps us familiar with the Fourier Transform. By using Fourier Transform with Python code for audio processing, the original sound sample will be improved. Also, by identifying a touch tone telephone audio, we can get the dialed numbers. The dialed numbers of ending8 is 02072840263.

Q1: Record a sound sample

Q2: Load the audio sample into python and plot the audio signal in the time domain and in the frequency domain with proper axis lables (time,frequency,amplitude).

Methodology:

Firstly, import the module

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

Then, use the code below to read wavefile, where fs is sampling rate read from file

```
fs,data=wavfile.read("G:/Python/voice.wav")
print("sampling rate is", fs)
```

Also, time domain, which is defined the length of the data divided by the sample rate

```
time=np.linspace(0, len(data)/fs, num=len(data))
print("length signal is",len(data))
```

Plot the graph that amplitude versus time (in time domain)

```
plt.figure(1)
plt.title('my voice')
plt.plot(time,data)
plt.xlabel('time(s)')
plt.ylabel('amplitude')
plt.show()
plt.savefig('time domain',format='eps',dpi=1000)
```

Doing FFT for the data

a=data[:,1]
xf=np.fft.fft(a)
faxis=np.linspace(0,fs,len(xf))

Plot the graph that amplitude versus frequency (in frequency domain)

plt.plot(faxis,abs(xf))
plt.xlabel('frequency(Hz)')
plt.ylabel('amplitude')
plt.savefig('frequency domain',format='eps',dpi=1000)

Results:

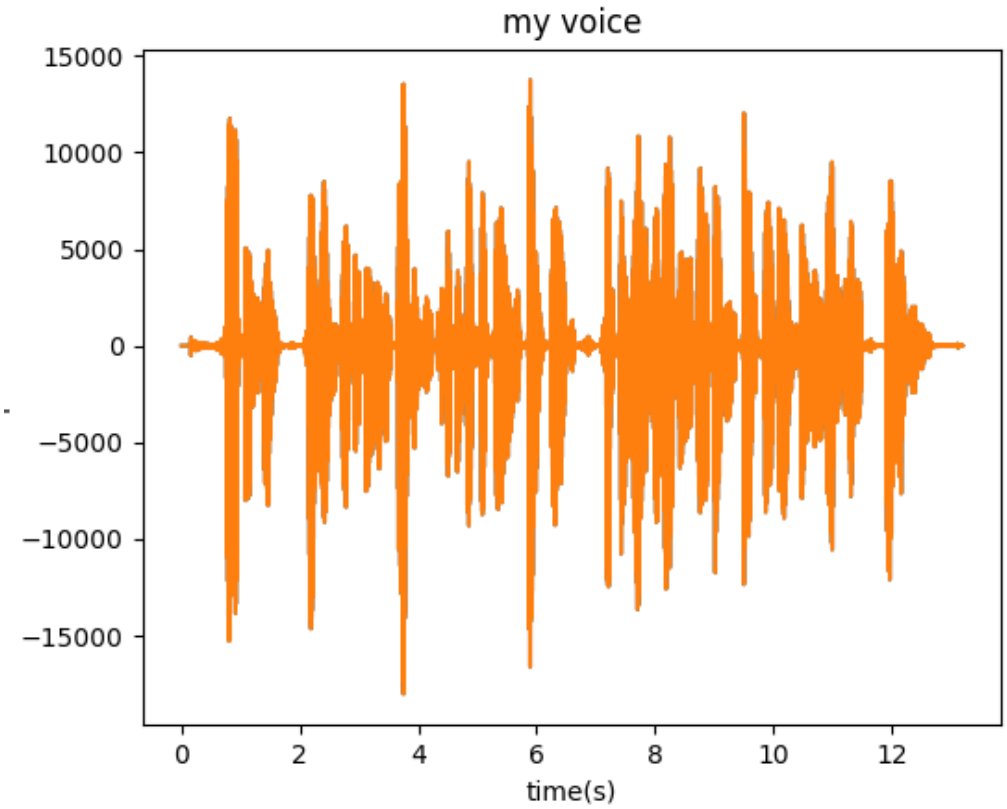

Figure 1. My voice (amplitude versus time) in time domain

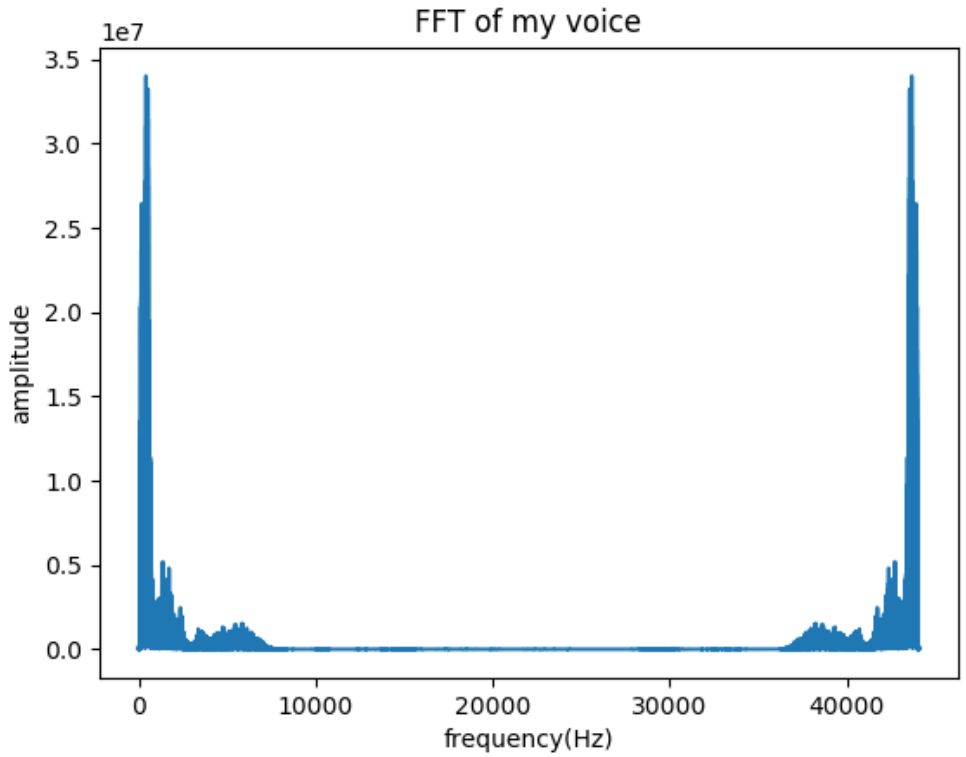

Figure 2. FFT of my voice (amplitude versus frequency) in frequency domain

Q3: Enhancing the sound by removing low frequencies from the sound recordings which cause just baseline shifts and/or pop-sounds. Justify your cutoff frequency by referring to the original time domain and frequency plots. Enhance the perception of the sounds by raising the amplitudes around 2-5kHz. Again, justify your choice of center frequency and bandwidth. Remember that this needs to be done with the help of the Fourier Transform.

Methodology:
Continuous with Q2, use the code below to remove the mirror sides in FFT

```
k1=int(len(xf)/44100*35000)
k2=int(len(xf))
xf[k1:k2+1]=0
```

Then justify my cutoff frequency from actual audio and frequency plots, and remove it. Also, plot the graph that amplitude versus frequency (in frequency domain)

```
k3=int(len(xf)/44100*0)      #choose frequency in corresponding samples
k4=int(len(xf)/44100*100)

plt.figure(3)
plt.plot(faxis,abs(xf))
```

```
plt.xlabel('frequency(Hz)')
plt.ylabel('amplitude')
plt.title('Remove low frequency of FFT')
plt.savefig('Remove low frequency of FFT',format='eps',dpi=1000)
```

Enhance the perception of the sounds by raising the amplitudes from 2-5kHz

```
xf[k3:k4+1]=0
k5=int(len(xf)/44100*2000)
k6=int(len(xf)/44100*5000)
xf[k5:k6+1]=xf[k5:k6+1]*2
plt.plot(faxis,abs(xf))
plt.xlabel('frequency(Hz)')
plt.ylabel('amplitude')
plt.title('Enhancement of FFT')
```

Then, do the inverse FFT, get the normalized signal in time domain and write it in the wavefile again.

```
idata=np.fft.ifft(xf)
idata=np.real(idata)
k7=max(abs(idata))
idata=idata/k7                    #divided let the range of the audio from -1 to 1
plt.plot(time,idata)

wavfile.write("G:/Python/abc1.wav",fs,idata)
```

By normalized the signal, the audio can be exported.

Results:

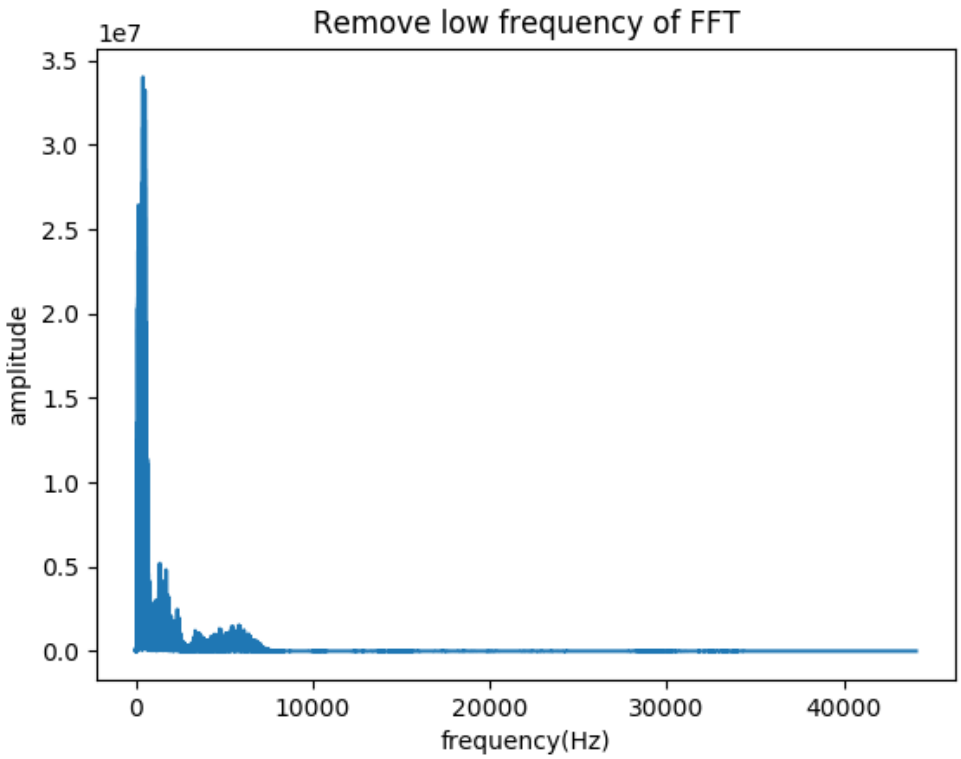

Figure 3. Remove low frequency of FFT (amplitude versus frequency)

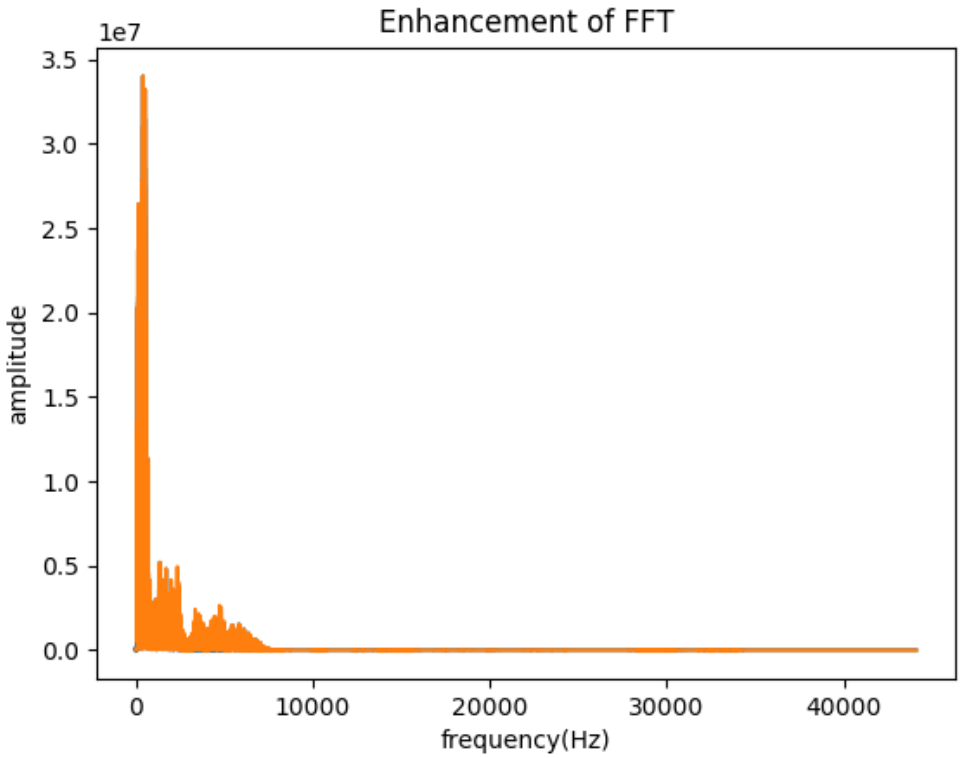

Figure 4. Enhancement in frequency domain(amplitude versus frequency)

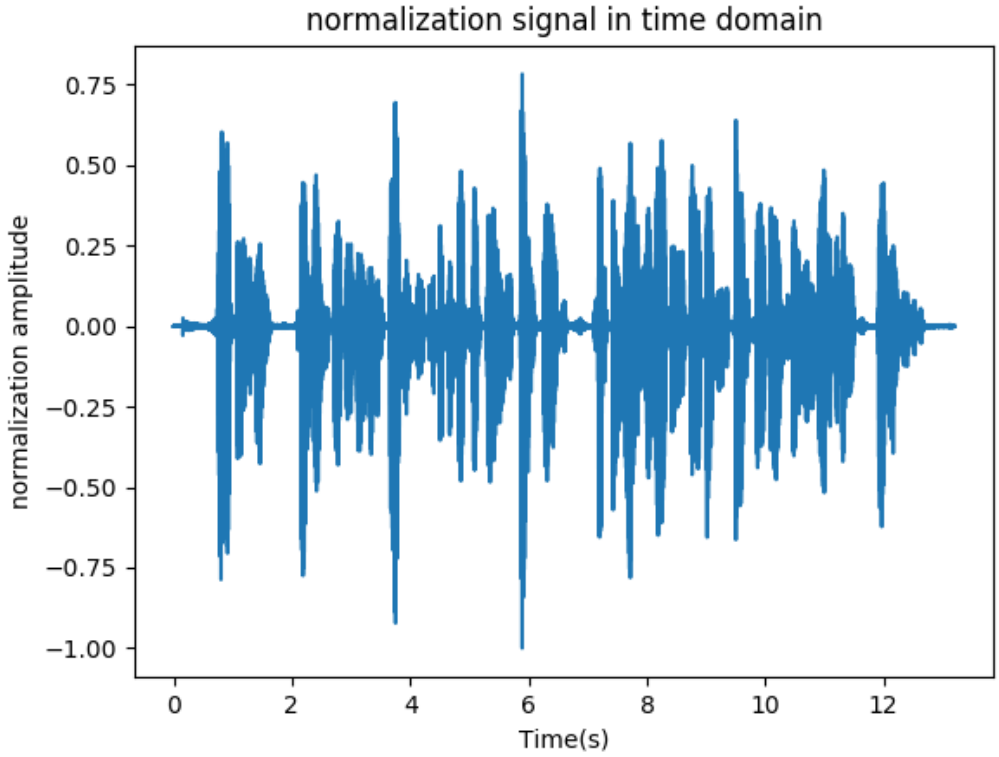

Figure 5. Normalization signal in time domain (normalization amplitude versus time)

Q4: We have dialled numbers with a touch tone telephone and sampled them at a sampling rate of 1kHz (which causes fold down). Find out which number have been dialled. We have recorded 10 different numbers. Use the file on moodle according to the last digit of your matric number. Find out who we've dialled.

Load the file on moodle and plot the graph in time domain

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


x = np.loadtxt("H:/Python/ending_8.dat")
ecg = x[:,1]

plt.figure(1)
plt.plot(ecg)
plt.title('dialled numbers at a sampling rate of 1kHz')
plt.xlabel('Samples')
plt.ylabel('amplitude')
```

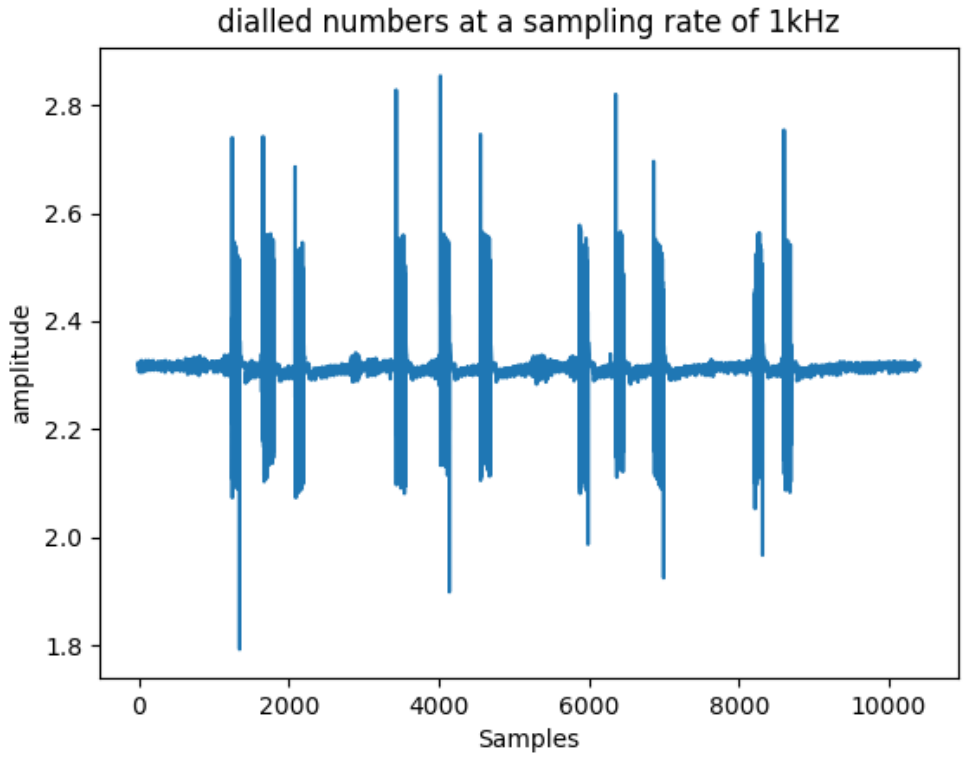

Figure 6. dialed numbers at 1kHz sampling rate (amplitude versus samples)

From the graph, we can see clearly that 11 dialed numbers in samples. In order to analysis the specific number, it is necessary to divide different numbers in specific range and do the FFT.

The sample ranges are shown below

1st number 1200-1600
2nd number 1600-2000
3rd number 2000-2400
4th number 3000-3800
5th number 3800-4300
6th number 4300-4800
7th number 5600-6200
8th number 6200-6600
9th number 6600-7200
10th number 8100-8400
11st number 8400-8800

For the first number, roughly divide in a range from samples 1200 to 1600

```
dial = ecg[1200:1600]
plt.figure(2)
```

plt.plot(dial)
plt.title('specific number range in samples')

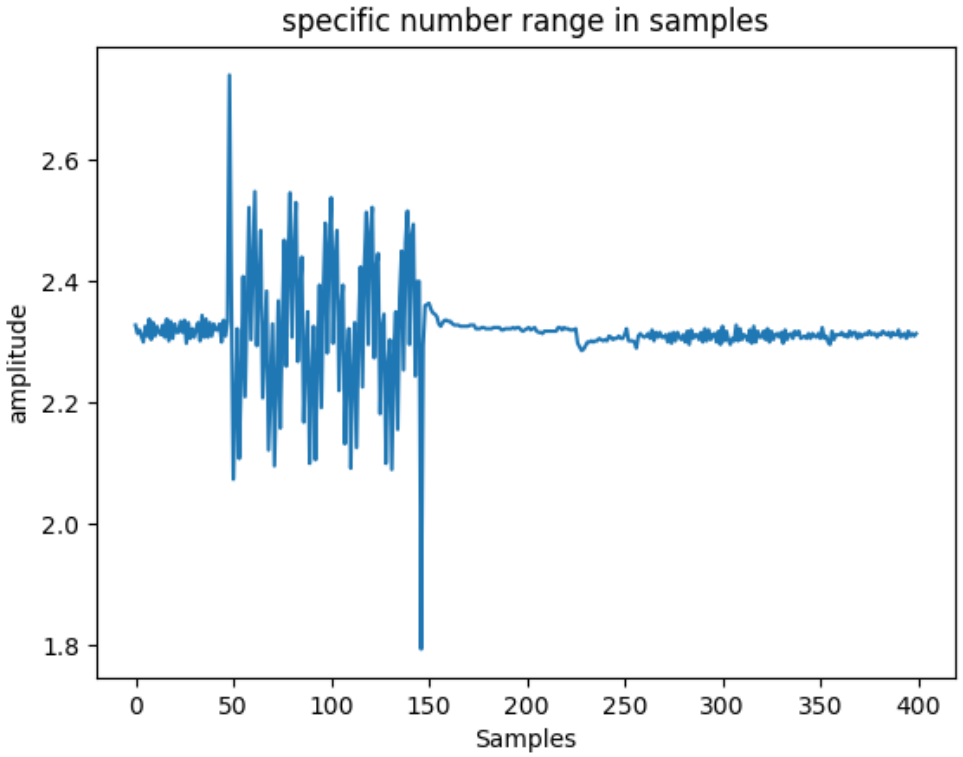

Figure 7. 1st number in specific sample range (amplitude versus samples)

After dividing this, we can now do the Fourier Transform. Note: The 0-2Hz's signal is DC, which have large amplitude but trivial, it will affect the observation of the peak of the dialed number, so we remove it.

plt.figure(3)
dial_fft=np.fft.fft(dial)
dial_fft[0:2]=0

faxis=np.linspace(0,500,int(len(dial)/2+1))
print("length",len(dial))
plt.plot(faxis,abs(dial_fft[0:int(len(dial)/2+1)]))
plt.xlabel('frequency(Hz)')
plt.ylabel('amplitude')
plt.title('1st phone number')

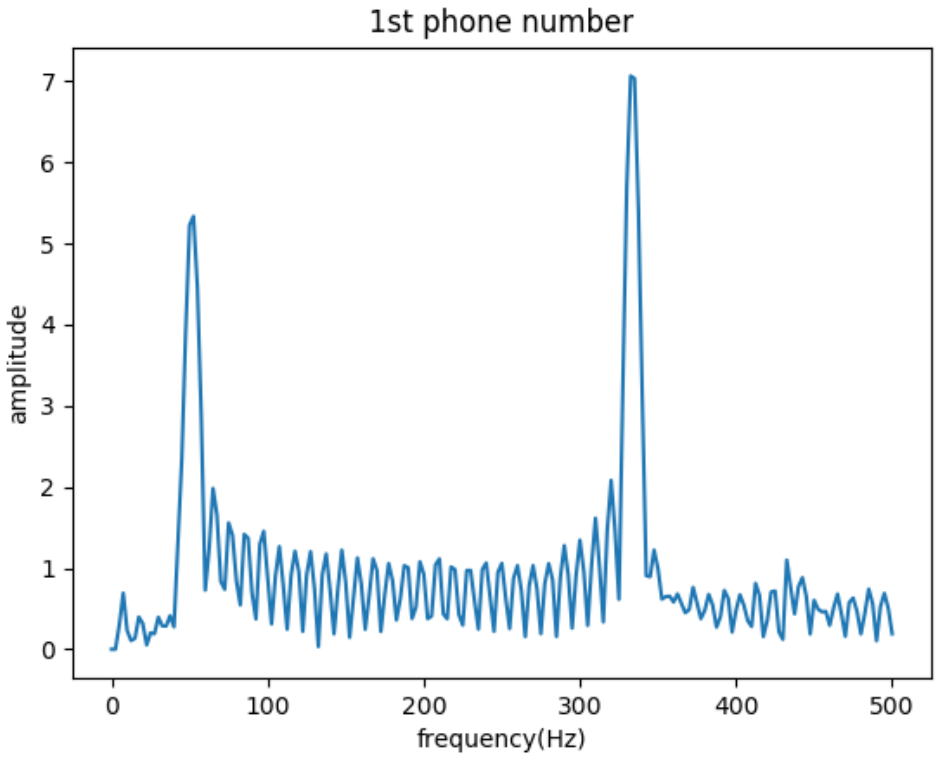

Figure 8. 1st number in frequency domain (amplitude versus frequency)

From this graph, we can see two obvious peaks. In DTMF definition, the DTMF telephone keypad is laid out in a 4×4 matrix of push buttons in which each row represents the low frequency component and each column represents the high frequency component of the DTMF signal. Pressing a key sends a combination of the row and column frequencies.[1]

So, the exact dialed number can be read from the graph below

DTMF keypad frequencies (with sound clips)

	1209 Hz	1336 Hz	1477 Hz	1633 Hz
697 Hz	1	2	3	A
770 Hz	4	5	6	B
852 Hz	7	8	9	C
941 Hz	*	0	#	D

Figure 9. DTMF keypad frequencies[1]

For Fig.8, using 1000Hz minus lower peak's frequency to get row frequency

$$1000 - 50 \approx 941\text{Hz}$$

Using 1000Hz plus higher peak's frequency to get column frequency

$$1000 + 330 \approx 1336\text{Hz}$$

From the graph, we can identify the 1st number is 0.

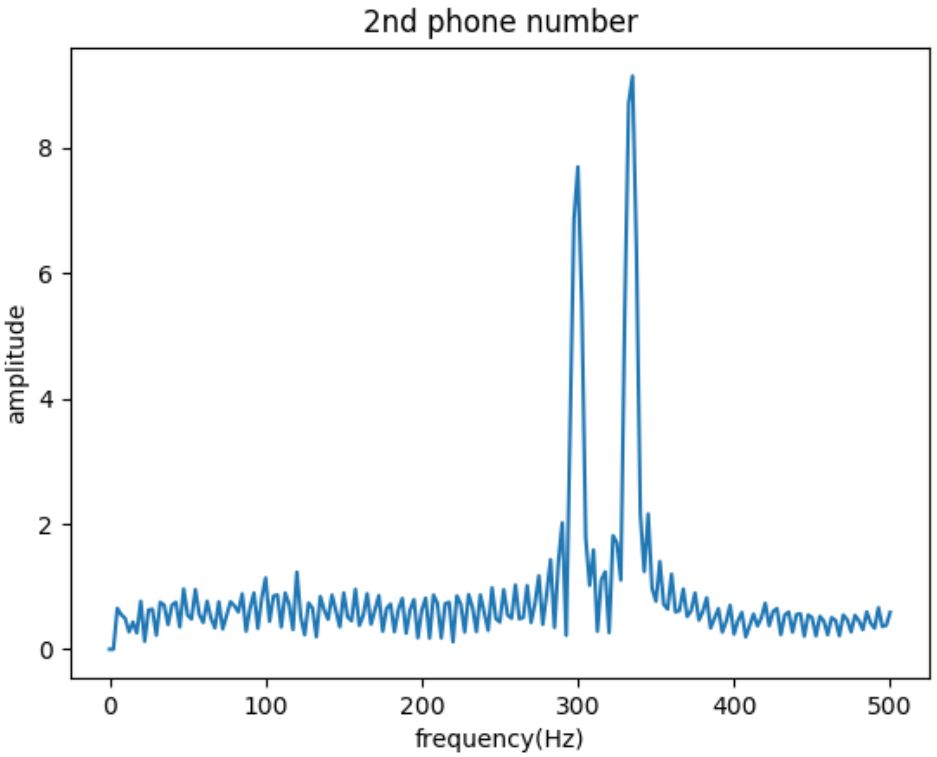

Figure 10. 2nd phone number graph in frequency

Using 1000Hz minus lower peak's frequency to get row frequency

$$1000 - 300 \approx 697\text{Hz}$$

Using 1000Hz plus higher peak's frequency to get column frequency

$$1000 + 330 \approx 1336\text{Hz}$$

From the graph, we can identify the 2nd number is 2.

Here, just show 1st and 2nd number as example. Totally, 11 number are identified by this method.

In ending_8 file, the phone number is 02072840263.

Results:

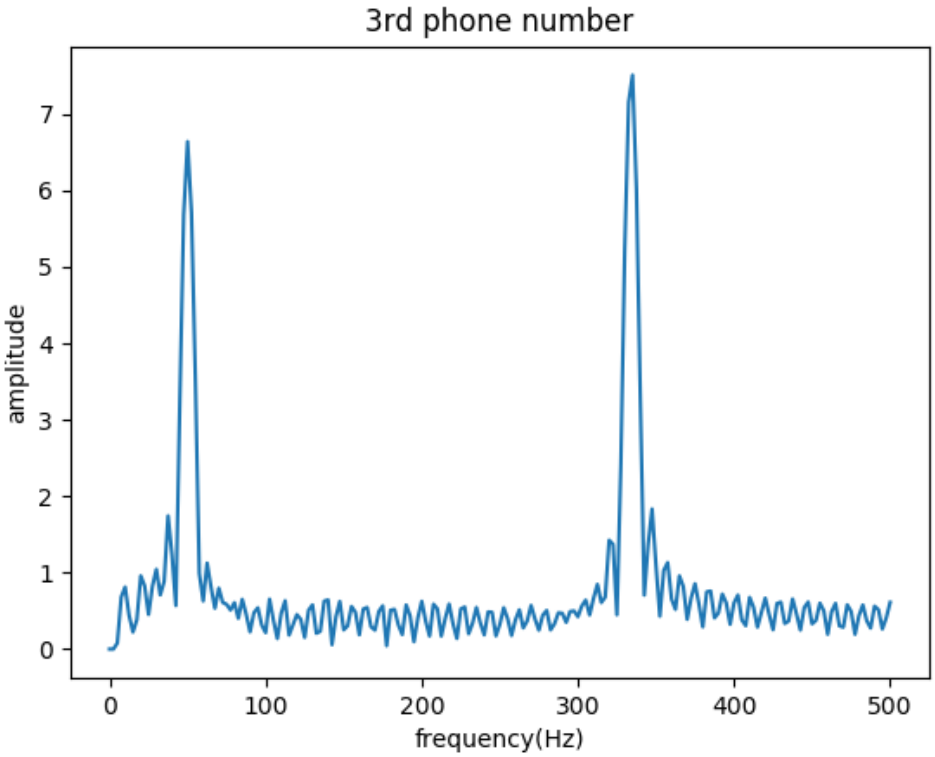

Figure 11. 3rd phone number graph in frequency

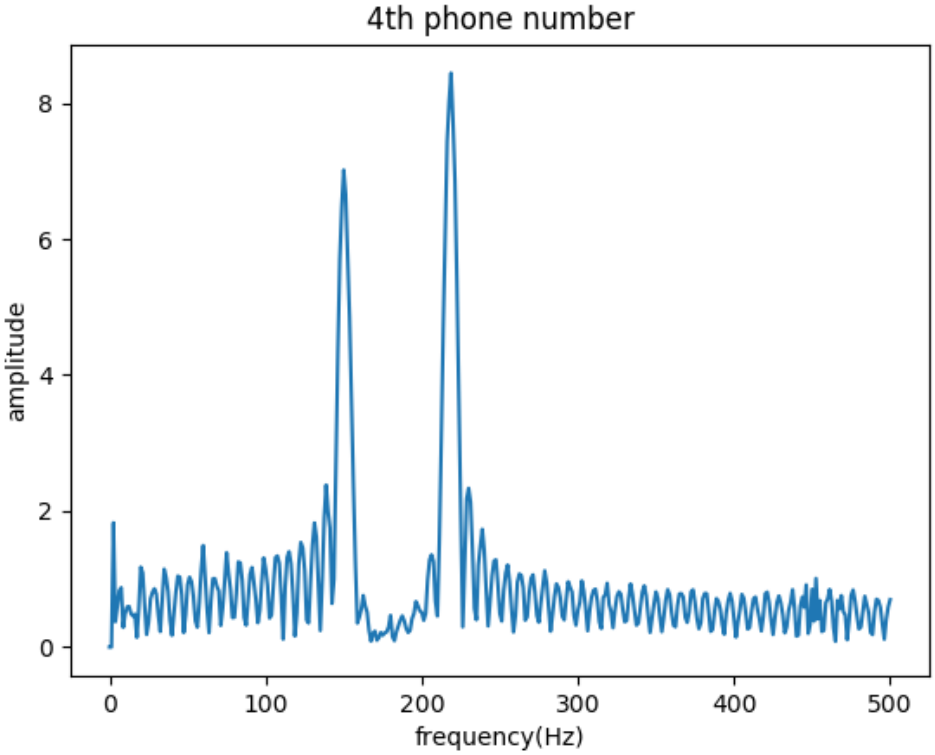

Figure 12. 4th phone number graph in frequency

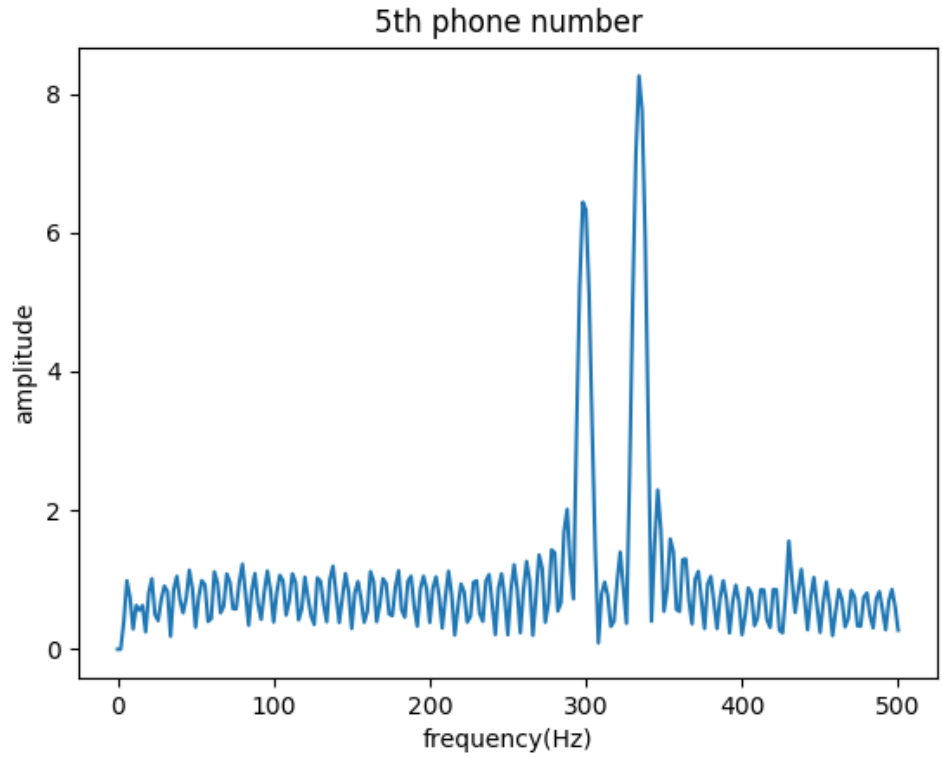

Figure 13. 5th phone number graph in frequency

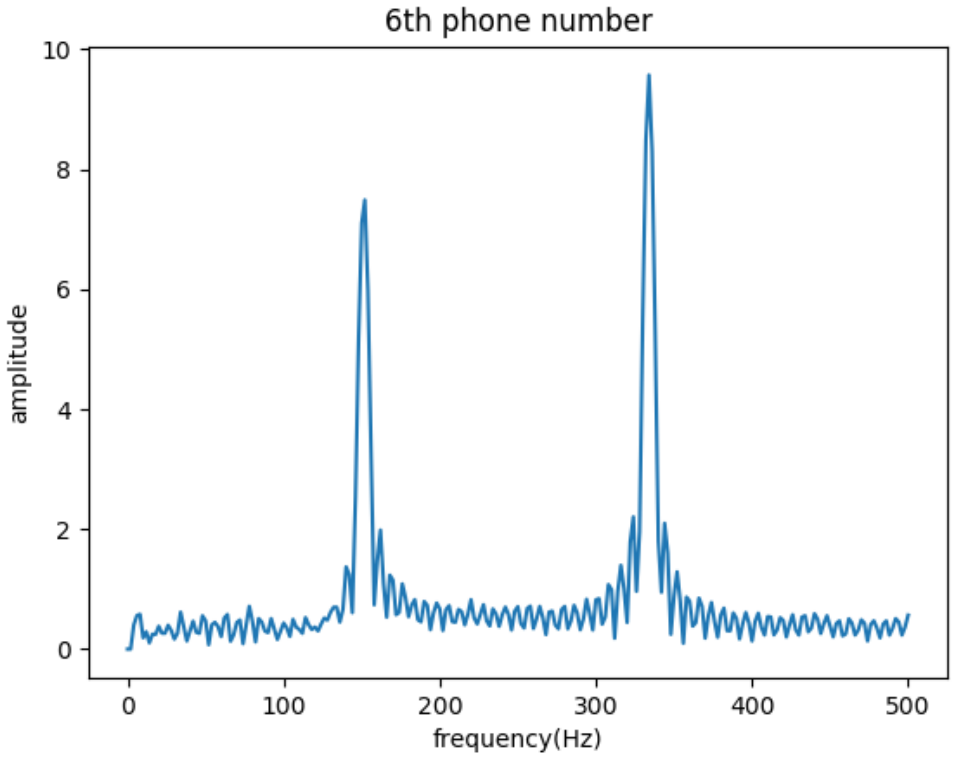

Figure 14. 6th phone number graph in frequency

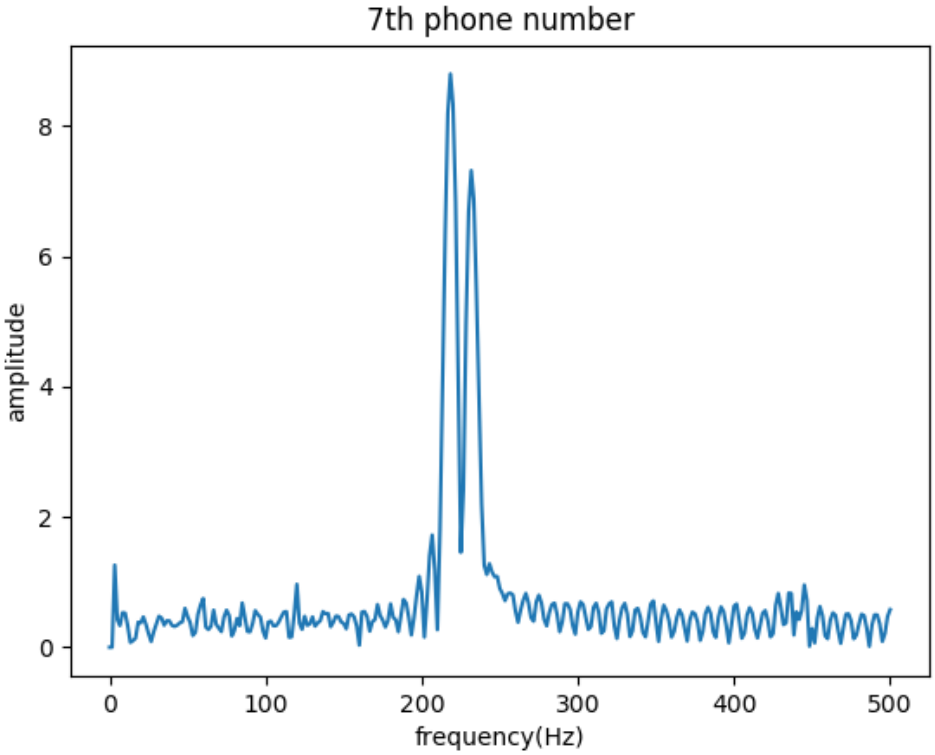

Figure 15. 7th phone number graph in frequency

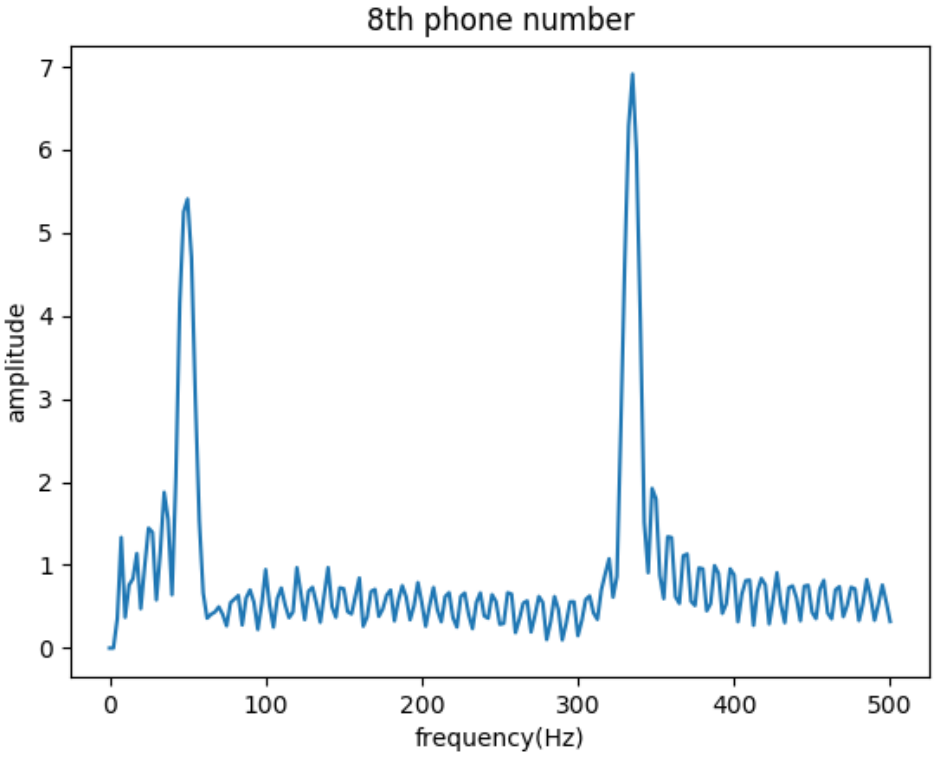

Figure 16. 8th phone number graph in frequency

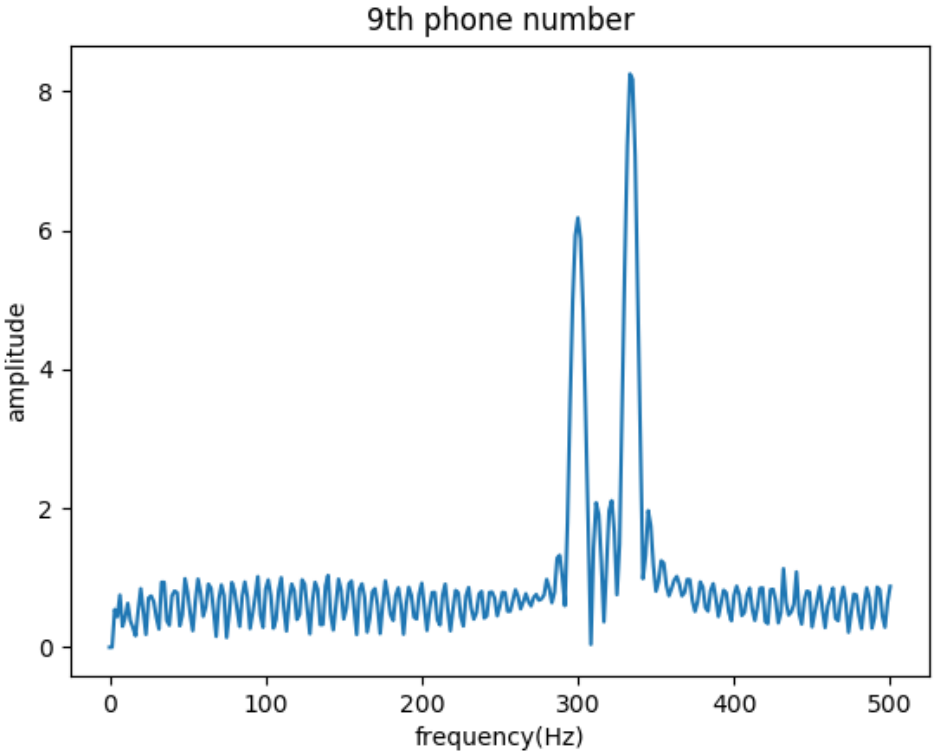

Figure 17. 9th phone number graph in frequency

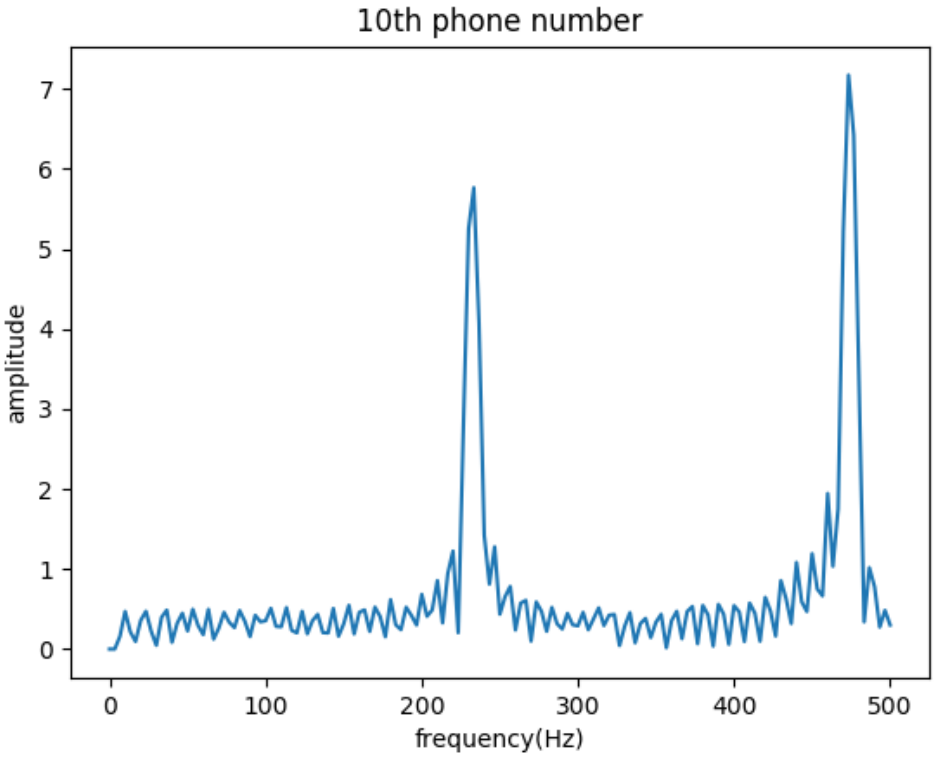

Figure 18. 10th phone number graph in frequency

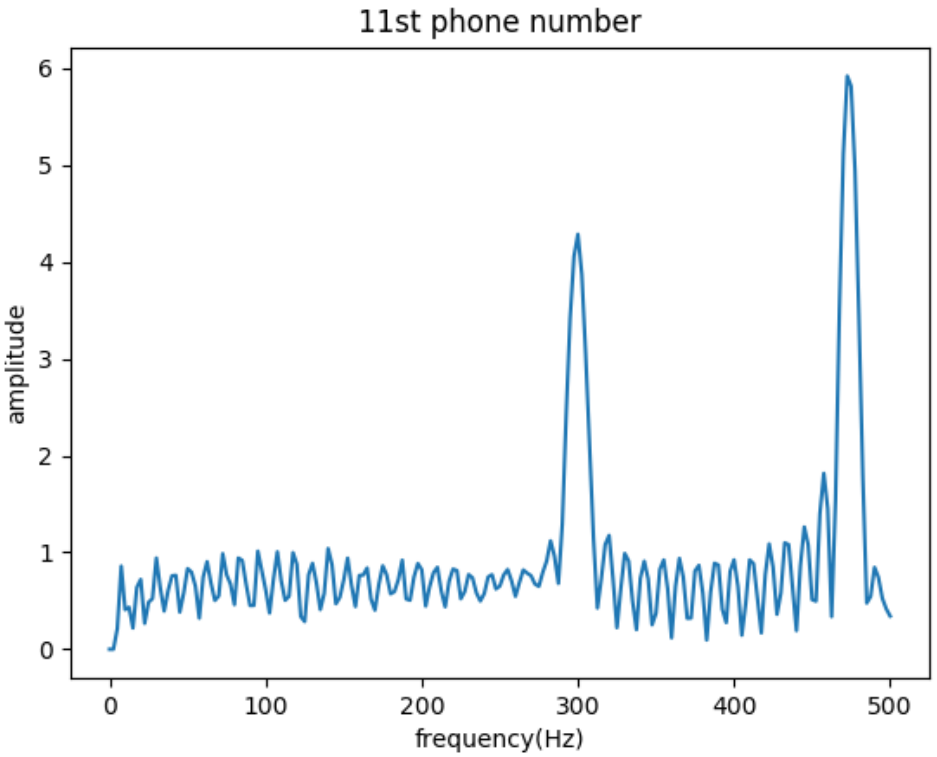

Figure 19. 11st phone number graph in frequency

Q5: Enhancing the sound by adding non-linearities: The so called aural exciter sends the sound through non-linearities and then adds these to the original audio to make it sound more exciting.
Try out different non linearlities (atan, cube, etc) and then add small amounts to the original audio to see which work best. Some of them add DC to the signal so that you need to remove the DC. In general sending the sound through a non-linearity will create harmonics which will be perceived as louder (see, for example a guitar amp in overdrive).

In this part, we enhance the sound by adding non-linearities instead of raising the amplitudes directly. Previous code is the same with Q3, but not raise the amplitude.

Firstly, do a high-pass filter and make those high-frequency signal adds some small amounts non-linearities operation, then add back to the original audio in time domain and improve the performance.

```
idata=np.fft.ifft(xf)
idata=np.real(idata)

k5=int(len(xf)/44100*0)
k6=int(len(xf)/44100*2000)
xf[k5:k6+1]=0
```

This part, I modify it by test the audio and try to find the best non-linearities function for my audio

idata1=np.fft.ifft(xf)
idata1=np.real(idata1)
*idata1=np.arctan(idata1*10)*
k7=max(abs(idata))
idata=idata/k7
*idata2=idata+idata1*0.001*

In order to output the audio properly, we have to normalize the amplitude

k8=max(abs(idata2))
idata2=idata2/k8
plt.plot(time,idata2)
wavfile.write("H:/Python/abc53.wav",fs,idata2)

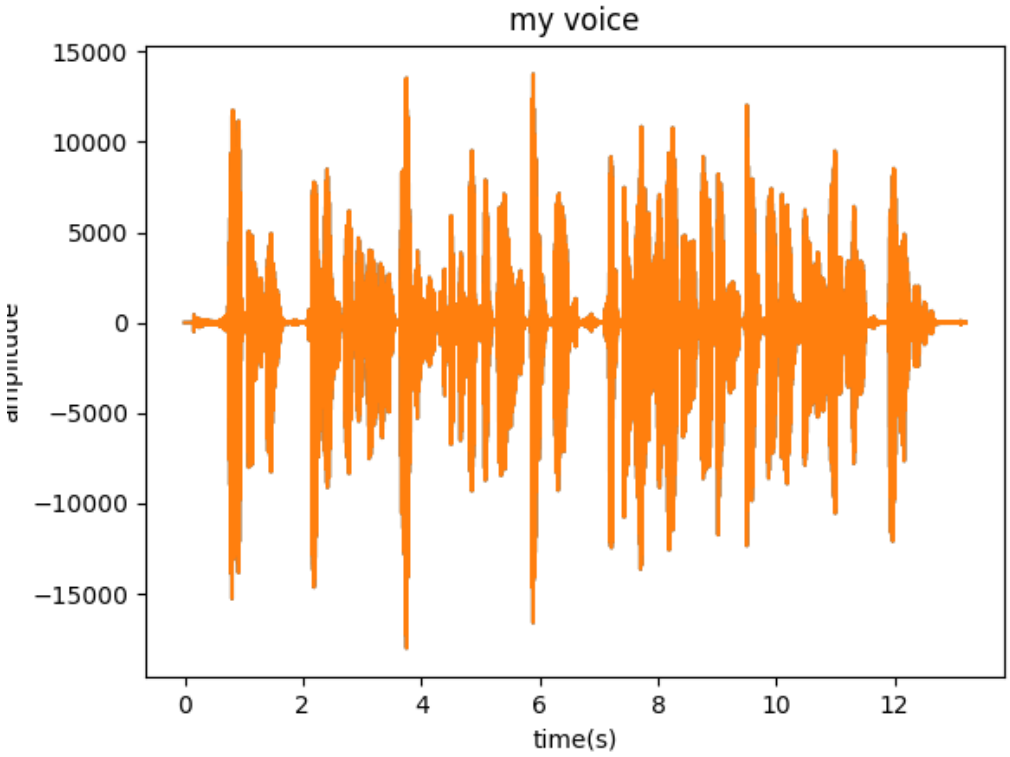

Figure 20. My original voice

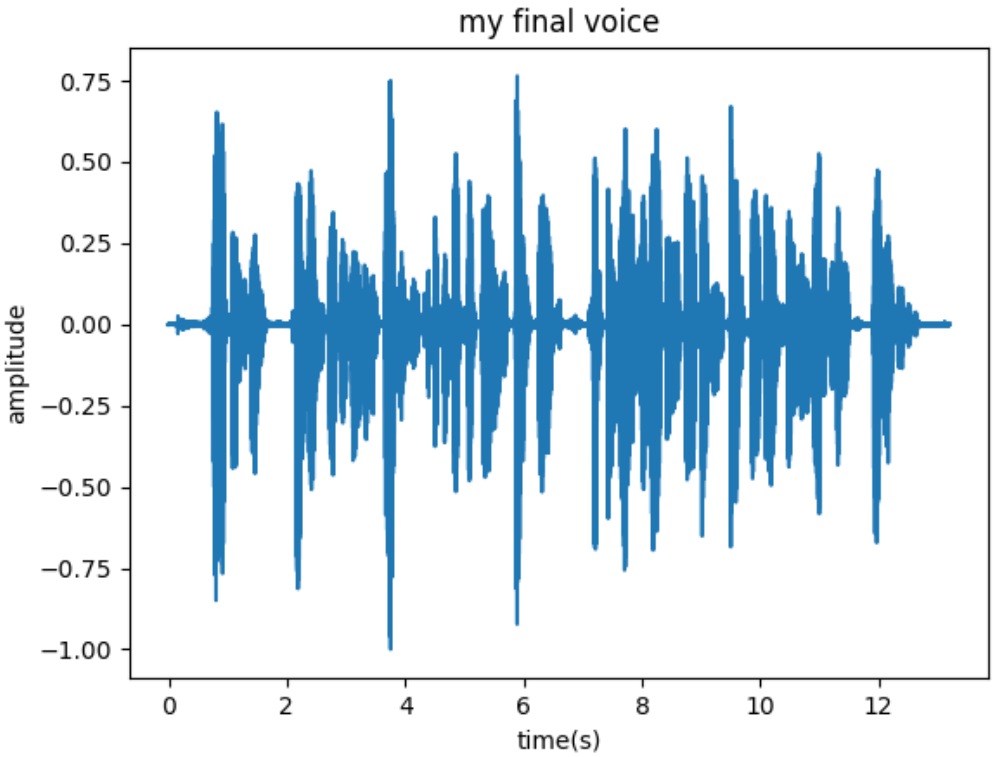

Figure 21. My Final voice

From the comparation of Fig20 and Fig21, there are some difference can be seen. The amplitude has been increased. Equally, the sound is louder, which have similar effect as raising amplitude directly. And the audio will sound more exciting.

Conclusion:

In this assignment, we use python to do the FFT and use it for audio processing. Also, by this assignment, I successfully identify the dialed numbers of phone based on DTMF and enhance the audio sample. By FFT, the signal in frequency domain are shown in the python. Then, by adjusting the frequency domain and enhance the signal we need, we can finally get a better audio.

Reference

[1] Dual-tone multi-frequency signaling,From Wikipedia
https://en.wikipedia.org/wiki/Dual-tone_multi-frequency_signaling

Appendices

Appendix A: Codes for Q2 and Q3

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



fs,data=wavfile.read("H:/Python/voice.wav")
print("sampling rate is", fs)

time=np.linspace(0, len(data)/fs, num=len(data))
print("length signal is",len(data))

plt.figure(1)
plt.title('my voice')
plt.plot(time,data)
plt.xlabel('time(s)')
plt.ylabel('amplitude')
plt.show()

a=data[:,1]
xf=np.fft.fft(a)
faxis=np.linspace(0,fs,len(xf))

plt.figure(2)
plt.title('FFT of my voice')
plt.plot(faxis,abs(xf))
plt.xlabel('frequency(Hz)')
plt.ylabel('amplitude')



k1=int(len(xf)/44100*35000)
k2=int(len(xf))
xf[k1:k2+1]=0

k3=int(len(xf)/44100*0)    #choose frequency in corresponding samples
k4=int(len(xf)/44100*100)

plt.figure(3)
```

```
plt.plot(faxis,abs(xf))
plt.xlabel('frequency(Hz)')
plt.ylabel('amplitude')
plt.title('Remove low frequency of FFT')
plt.savefig('Remove low frequency of FFT',format='eps',dpi=1000)

plt.figure(4)
a=np.fft.ifft(xf)
a=np.real(a)
plt.plot(time,a)
plt.xlabel('Time(s)')
plt.ylabel('normalization amplitude')
plt.title('normalization signal in time domain')


xf[k3:k4+1]=0
k5=int(len(xf)/44100*2000)
k6=int(len(xf)/44100*5000)
xf[k5:k6+1]=xf[k5:k6+1]*2
plt.plot(faxis,abs(xf))
plt.xlabel('frequency(Hz)')
plt.ylabel('amplitude')
plt.title('Enhancement of FFT')

plt.figure(5)
idata=np.fft.ifft(xf)
idata=np.real(idata)
k7=max(abs(idata))
idata=idata/k7    #divided let the range of the audio from -1 to 1
plt.plot(time,idata)
plt.xlabel('Time(s)')
plt.ylabel('normalization amplitude')
plt.title('normalization signal in time domain')

wavfile.write("G:/Python/abc1.wav",fs,idata)
```

Appendix B: Codes for Q4

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


x = np.loadtxt("H:/Python/ending_8.dat")
ecg = x[:,1]
```

```
plt.figure(1)
plt.plot(ecg)
plt.title('dialled numbers at a sampling rate of 1kHz')
plt.xlabel('Samples')
plt.ylabel('amplitude')

'''
1st    number 1200-1600
2nd    number 1600-2000
3rd    number 2000-2400
4th    number 3000-3800
5th    number 3800-4300
6th    number 4300-4800
7th    number 5600-6200
8th    number 6200-6600
9th    number 6600-7200
10th number 8100-8400
11st number 8400-8800
'''
dial = ecg[1200:1600]

plt.figure(2)

plt.plot(dial)
plt.title('specific number range in samples')
plt.xlabel('Samples')
plt.ylabel('amplitude')

print("length",len(dial))
plt.figure(3)
dial_fft=np.fft.fft(dial)
dial_fft[0:2]=0
faxis=np.linspace(0,500,int(len(dial)/2+1))
print("length",len(dial))
plt.plot(faxis,abs(dial_fft[0:int(len(dial)/2+1)]))
plt.xlabel('frequency(Hz)')
plt.ylabel('amplitude')
plt.title('1st phone number')
```

Appendix C: Codes for Q5

```
import scipy.io.wavfile as wavfile
import numpy as np
```

```
import matplotlib.pyplot as plt


fs,data=wavfile.read("H:/Python/voice.wav")
print("sampling rate is", fs)

time=np.linspace(0, len(data)/fs, num=len(data))
print("length signal is",len(data))


plt.figure(1)
plt.title('my voice')
plt.plot(time,data)
plt.xlabel('time(s)')
plt.ylabel('amplitude')
plt.show()


a=data[:,1]
xf=np.fft.fft(a)
faxis=np.linspace(0,fs,len(xf))

plt.figure(2)
plt.plot(faxis,abs(xf))
plt.xlabel('frequency(Hz)')
plt.ylabel('amplitude')



k1=int(len(xf)/44100*35000)    #eliminate the mirror signal
k2=int(len(xf))
xf[k1:k2+1]=0

plt.figure(3)
plt.plot(faxis,abs(xf))
plt.xlabel('frequency(Hz)')
plt.ylabel('amplitude')
plt.title('FFT of my voice')


plt.figure(4)
k3=int(len(xf)/44100*0) #remove DC
k4=int(len(xf)/44100*100)
xf[k3:k4+1]=0
```

```
plt.plot(faxis,abs(xf))


idata=np.fft.ifft(xf)
idata=np.real(idata)

k5=int(len(xf)/44100*0)
k6=int(len(xf)/44100*2000)
xf[k5:k6+1]=0

idata1=np.fft.ifft(xf)
idata1=np.real(idata1)
idata1=np.arctan(idata1*10)
k7=max(abs(idata))
idata=idata/k7
idata2=idata+idata1*0.001

k8=max(abs(idata2))
idata2=idata2/k8

plt.figure(5)
plt.plot(time,idata2)
wavfile.write("H:/Python/abc53.wav",fs,idata2)
```