Assignment 1, Digital Signal Processing: Fourier Transform

Weng, Wei-Ming(2305670)

Q1. Record a sound sample: Students record their voice with one of the supplied headsets, microphones or apps. Make sure that you record at least at 10kHz or higher sampling rates.

A1. I recorded my sound with my phone and converted online (https://audio.online-convert.com/convert-to-wav), set the sample rate to 44100Hz and mono channel. The converted sound track(Q3.wav) is in the Zip file which was uploaded to moodle.

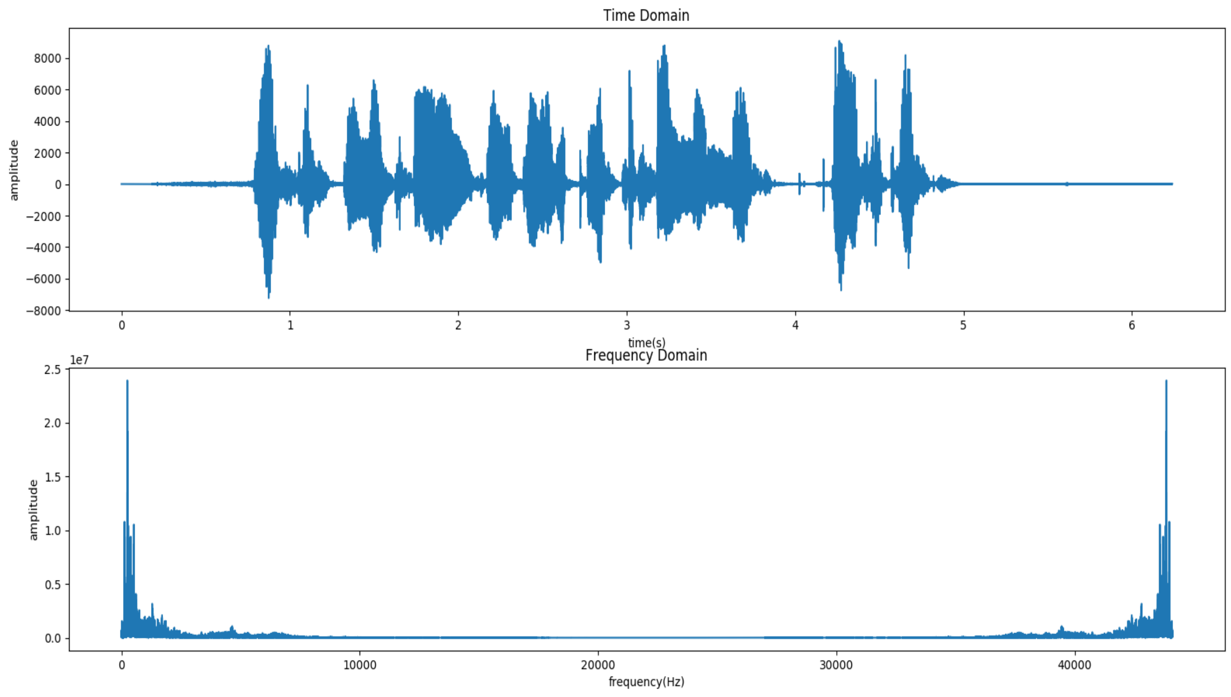
Q2. Load the audio sample into python and plot the audio signal in the time domain and in the frequency domain with proper axis lables (time,frequency,amplitude).

A2. With the code:

```
rate,data = wavfile.read("Thirdattempt.wav")

taxis=np.linspace(0,len(data)/rate,len(data))

plt.subplot(211)

matplt.title('Time Domain')

matplt.xlabel('time(s)')

matplt.ylabel('amplitude')

plt.plot(taxis,data)
```

By using the libraries matplotlib.pyplot and matplotlib.pylab, we can basically add the labels and the datas in vecter fomat to produce clear pictures. Additionally, axis can use linspace to reorganize the range we want it to be displayed on the pictures.

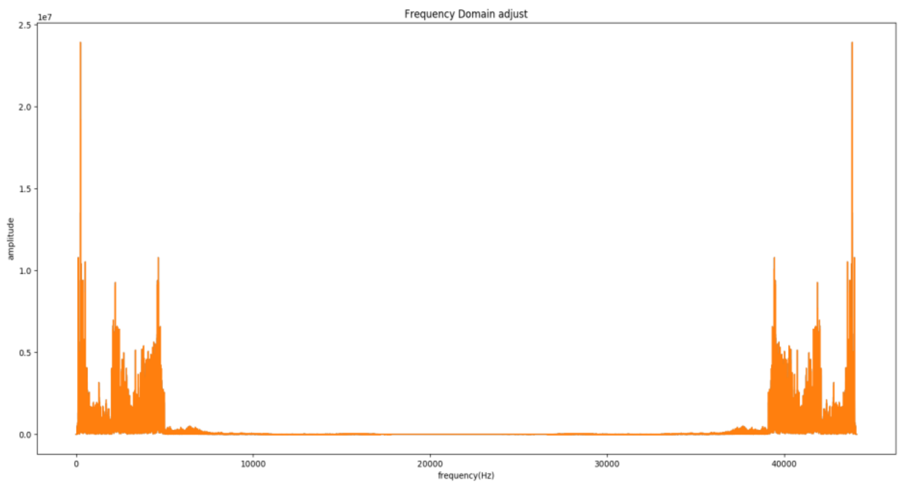
Results:



Q3. Enhancing the sound by removing low frequencies…Justify your cutoff frequency by referring to the original timedomain and frequency plots…Enhance the perception of the sounds by raising the amplitudesaround 2-5kHz. ..Justify your choice of centre frequency and bandwidth.

A3. The adjustments including removing DC and enhancing the 2~5kHz are written in the code:

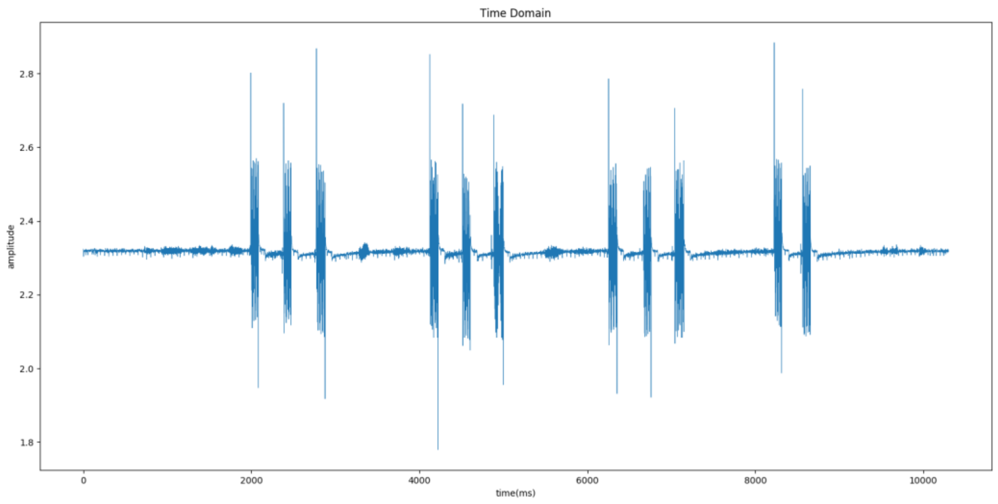
```
#Get rid of DC components
k0=int(len(dataf)/rate*50)
dataf[0:k0+1]=0#0~50Hz removed
dataf[len(dataf)-k0:len(dataf)]=0#mirror
#enhance the sound from 2kHz~5Hz
amplify=10
k1=int(len(dataf)/rate*2000)
k2=int(len(dataf)/rate*5000)
dataf[k1:k2+1]=amplify*dataf[k1:k2+1]#amplifing the amplitudes
dataf[len(dataf)-k2:len(dataf)-k1]=amplify*dataf[len(dataf)-k2:len(dataf)-k1]#mirror
```



In order to remove the low frequencies of the data, I chose 50Hz as my cutoff frequency to filter the data. By doing any adjustment, always mirror the same adjustment to the Fs/2~Fs domain to keep the iFFT time domain result real. To justify the center frequency, it lays between the lower cutoff frequency and the upper cutoff frequency, which is 22050Hz. The bandwidth is 44100-100=44000Hz. In addition, the adjusted sound track(answer_to_Q3.wav) is also in the Zip file on moodle.

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.

A4. Since my ending number (2305670) was 0, I took a look into the time domain of the data file given on moodle, it is obvious that there are eleven numbers being dialed.



Since the DTMF signals are contributed by two sin waves with different frequencies, I can do this either by convolutions between different sin filters(FIR concepts) or look into the FFT results of different numbers. I chose the FFT method and divided the time domain into 11 parts. After that, I applied the FFT to the 11 time domain parts to examine the peaks of their response. Since the DTMF signals have a peak frequency of 1477, the lack of sample rate(1000Hz) will cause fold down. In this case, we can calculate the fold down peaks of all the DTMF signals and compare it to the FFT results.

Frequencies	1209 Hz	1336 Hz	1477 Hz
697 Hz	1	2	3
770 Hz	4	5	6
852 Hz	7	8	9
941 Hz	*	0	#

By the given chart on the left, I was able to find the "fold down corresponding DTMF frequencies" due to the sampling frequency. In this situation, I only focus on the fold down frequency smaller than 500Hz since it is mirrored. The function I wrote is to simplify the work of viewing every FFT results and get the analog frequencies that lies in the number.

```
def analogf_detector(soundf):
    samplef=1000#due to the assighment request
    threshold=2#Threshold to examine the peak
    resolution=0.001#Every time I move the threshold
    goal=np.array([0,0,0,0,0,0,0,0])#remembers what and how many times of the f
    for i in range(int(threshold/resolution)):
        if(threshold<soundf[int(303/samplef*len(soundf))] and goal[1]==0):
            goal[0]=goal[0]+1
            goal[1]=goal[1]+1
        if(threshold<soundf[int(230/samplef*len(soundf))] and goal[2]==0):
            goal[0]=goal[0]+1
            goal[2]=goal[2]+1
        if(threshold<soundf[int(148/samplef*len(soundf))] and goal[3]==0):
```

```
        goal[0]=goal[0]+1
        goal[3]=goal[3]+1
if(threshold<soundf[int(59/samplef*len(soundf))] and goal[4]==0):
        goal[0]=goal[0]+1
        goal[4]=goal[4]+1
if(threshold<soundf[int(220/samplef*len(soundf))] and goal[5]==0):######adjust
        goal[0]=goal[0]+1
        goal[5]=goal[5]+1
if(threshold<soundf[int(336/samplef*len(soundf))] and goal[6]==0):
        goal[0]=goal[0]+1
        goal[6]=goal[6]+1
if(threshold<soundf[int(477/samplef*len(soundf))] and goal[7]==0):
        goal[0]=goal[0]+1
        goal[7]=goal[7]+1
if(goal[0] != 2):
        threshold=threshold-resolution
elif(goal[0]==2):
        if(goal[6]==1 and goal[4]==1):
                print('0')
        if(goal[5]==1 and goal[1]==1):
                print('1')
        if(goal[6]==1 and goal[1]==1):
                print('2')
        if(goal[7]==1 and goal[1]==1):
                print('3')
        if(goal[5]==1 and goal[2]==1):
                print('4')
        if(goal[6]==1 and goal[2]==1):
                print('5')
        if(goal[7]==1 and goal[2]==1):
                print('6')
        if(goal[5]==1 and goal[3]==1):
                print('7')
        if(goal[6]==1 and goal[3]==1):
                print('8')
        if(goal[7]==1 and goal[3]==1):
                print('9')
        break
```

It is a quite straight forward and simple approach just to move the threshold little by little. However, the DTMF signal of 1209Hz is not as same as the chart which I posted but much closer to 1220Hz, so I did a small adjustment to the code. It is also fine to run 11 times of FFT and find the peaks manually. The “fold down corresponding DTMF frequencies” are 303, 230, 148, 59, 220, 336 and 477 Hz. By running the 11 parts of the manually cut time domain data, doing FFT 11 times and throwing them into the “analogf_detector”function, it will print out the data for me. The telephone numbers that showed up was **02070420700,** which is the number for Victoria Beckham London.

Q5. Enhancing the sound by adding non-linearities to the audio in Q2

A5. First, I tried to look into the .cpp file and the pdf on moodle and tried to understand the fundamentals and work it out in a python style way. For the atan adjustment, I pulled out the 800Hz to 43300Hz of the FFT data and did a arctan function on it. Subsequently, I added the new atan part back to the initial FFT of the data and do iFFT to get the new audio(answer_to_Q5_atan.wav). Since atan function is bounded in value and does not result in the imaginary part, it can be used in nonlinear adjustment of aural exciter. The code is shown below:

```
import numpy as np
import scipy.io.wavfile as wavfile
#import the data
rate,data = wavfile.read("Q3.wav")
dataf=np.fft.fft(data)
#setup variable
high_pass_f1=800
#follow the given .cpp file and adjust it to python style
dataf_part1=dataf[0:int(len(dataf)*(high_pass_f1/rate))]
dataf_part2=dataf[int(len(dataf)*(high_pass_f1/rate)):len(dataf)-1-int(len(dataf)*(high_pass_f1/rate))]#mirror
dataf_part3=dataf[len(dataf)-1-int(len(dataf)*(high_pass_f1/rate)):len(dataf)]
dataf_arctan=np.arctan(dataf_part2)
dataf_combine=np.hstack((dataf_part1,dataf_part2+dataf_arctan,dataf_part3))
data_result=np.fft.ifft(dataf_combine)
data_result=np.real(data_result)
data_clean=np.float32(data_result/np.max(np.abs(data_result)))
wavfile.write('answer_to_Q5_atan.wav',44100,data_clean)
#arctan succeed
```

Now I moved on to the cube function that is abstract than the atan function. Since that cube function(y=x^3) is not bounded, there are several ways to implement it. First I tried to pull out the 800~43300Hz FFT and normalize the value to 1 to -1, then do the cube. In this case the cube function adjustment will be bounded in 1 to -1, then add these data back to the initial FFT to obtain the new FFT. Finally do the iFFT to get the new time domain. The code is shown below, it is basically same as atan function but differs from the code with bold and italic. The result audio is called" answer_to_Q5_cube.wav":

```
import numpy as np
import scipy.io.wavfile as wavfile
#import data
rate,data = wavfile.read("Q3.wav")
dataf=np.fft.fft(data)
#setup variable
high_pass_f1=800
#follow the given .cpp file and adjust it to python style
dataf_part1=dataf[0:int(len(dataf)*(high_pass_f1/rate))]
dataf_part2=dataf[int(len(dataf)*(high_pass_f1/rate)):len(dataf)-1-int(len(dataf)*(high_pass_f1/rate))]
dataf_part3=dataf[len(dataf)-1-int(len(dataf)*(high_pass_f1/rate)):len(dataf)]
coefficient=np.max(np.abs(dataf_part2))
#normalized
dataf_part2_normalized=dataf_part2/coefficient
dataf_cube=np.power(dataf_part2_normalized,3)
dataf_combine=np.hstack((dataf_part1,dataf_part2+dataf_cube,dataf_part3))
data_result=np.fft.ifft(dataf_combine)
data_result=np.real(data_result)
data_clean=np.float32(data_result/np.max(np.abs(data_result)))
wavfile.write('answer_to_Q5_cube.wav',44100,data_clean)
#cube succeed
```

There is another possible way to implement the cube function by combining it with atan function. Since cube function is not bounded, the amplitude of the cube can bring more adjustment to the arctan function with the formula: y=arctan(x^3). The code is similar to the atan function (result audio: answer_to_Q5_cube2.wav).

```
import numpy as np
import scipy.io.wavfile as wavfile
#import the data
rate,data = wavfile.read("Q3.wav")
dataf=np.fft.fft(data)
#setup variable
high_pass_f1=800
```

```
#follow the given .cpp file and adjust it to python style
dataf_part1=dataf[0:int(len(dataf)*(high_pass_f1/rate))]
dataf_part2=dataf[int(len(dataf)*(high_pass_f1/rate)):len(dataf)-1-int(len(dataf)*(high_pass_f1/rate))]#mirror
dataf_part3=dataf[len(dataf)-1-int(len(dataf)*(high_pass_f1/rate)):len(dataf)]
dataf_arctan=np.arctan(dataf_part2*dataf_part2*dataf_part2)
dataf_combine=np.hstack((dataf_part1,dataf_part2+dataf_arctan,dataf_part3))
data_result=np.fft.ifft(dataf_combine)
data_result=np.real(data_result)
data_clean=np.float32(data_result/np.max(np.abs(data_result)))
wavfile.write('answer_to_Q5_cube2.wav',44100,data_clean)
#cube2 succeed
```

Appendix:

Complete Python code for Q1,Q2 and Q3:

```
"""
Created on Wed Oct  4 09:39:00 2017
@author: Wei-Ming Weng (2305670)
"""
import numpy as np
import scipy.io.wavfile as wavfile
import matplotlib.pyplot as plt
import matplotlib.pylab as matplt

#Part 1
#https://stackoverflow.com/questions/2060628/reading-wav-files-in-python
#https://stackoverflow.com/questions/18644166/how-to-manipulate-wav-file-data-in-python
rate,data = wavfile.read("Q3.wav")

#Part 2
taxis=np.linspace(0,len(data)/rate,len(data))
plt.subplot(211)
matplt.title('Time Domain')
matplt.xlabel('time(s)')
matplt.ylabel('amplitude')
plt.plot(taxis,data)
#Fourier transform
dataf=np.fft.fft(data)
faxis=np.linspace(0,rate,len(dataf))
plt.subplot(212)
```

```
matplt.title('Frequency Domain')
matplt.xlabel('frequency(Hz)')
matplt.ylabel('amplitude')
plt.plot(faxis,abs(dataf))

#Part 3
#Get rid of DC components
k0=int(len(dataf)/rate*50)
dataf[0:k0+1]=0#0~50Hz removed
dataf[len(dataf)-k0:len(dataf)]=0#mirror
#enhance the sound from 2kHz~5Hz
amplify=10
k1=int(len(dataf)/rate*2000)
k2=int(len(dataf)/rate*5000)
dataf[k1:k2+1]=amplify*dataf[k1:k2+1]#amplifing the amplitudes
dataf[len(dataf)-k2:len(dataf)-k1]=amplify*dataf[len(dataf)-k2:len(dataf)-k1]#mirror
#reversing the fft
data_clean=np.fft.ifft(dataf)
data_clean=np.real(data_clean)
#Normalize the data_clean
data_clean=np.float32(data_clean/np.max(np.abs(data_clean)))
wavfile.write('answer_to_Q3.wav',44100,data_clean)
```

Complete Python code for Q4:

```
import numpy as np
import matplotlib.pyplot as plt
import matplotlib.pylab as matplt
#checking the datas in the moodle
#http://www.eecs.umich.edu/courses/eecs206/public/lab/lab7/lab7.pdf
#We know that DTMF frequencies are 697, 770, 852, 941, 1209, 1336, 1477
def analogf_detector(soundf):
    samplef=1000#due to the assighment request
    threshold=2#Threshold to examine the peak
    resolution=0.001#Every time I move the threshold
    goal=np.array([0,0,0,0,0,0,0,0])#remembers what and how many times of the f
    for i in range(int(threshold/resolution)):
        if(threshold<soundf[int(303/samplef*len(soundf))] and goal[1]==0):
            goal[0]=goal[0]+1
            goal[1]=goal[1]+1
```

```
if(threshold<soundf[int(230/samplef*len(soundf))] and goal[2]==0):
    goal[0]=goal[0]+1
    goal[2]=goal[2]+1
if(threshold<soundf[int(148/samplef*len(soundf))] and goal[3]==0):
    goal[0]=goal[0]+1
    goal[3]=goal[3]+1
if(threshold<soundf[int(59/samplef*len(soundf))] and goal[4]==0):
    goal[0]=goal[0]+1
    goal[4]=goal[4]+1
if(threshold<soundf[int(220/samplef*len(soundf))] and goal[5]==0):######adjust
    goal[0]=goal[0]+1
    goal[5]=goal[5]+1
if(threshold<soundf[int(336/samplef*len(soundf))] and goal[6]==0):
    goal[0]=goal[0]+1
    goal[6]=goal[6]+1
if(threshold<soundf[int(477/samplef*len(soundf))] and goal[7]==0):
    goal[0]=goal[0]+1
    goal[7]=goal[7]+1
if(goal[0] != 2):
    threshold=threshold-resolution
elif(goal[0]==2):
    if(goal[6]==1 and goal[4]==1):
        print('0')
    if(goal[5]==1 and goal[1]==1):
        print('1')
    if(goal[6]==1 and goal[1]==1):
        print('2')
    if(goal[7]==1 and goal[1]==1):
        print('3')
    if(goal[5]==1 and goal[2]==1):
        print('4')
    if(goal[6]==1 and goal[2]==1):
        print('5')
    if(goal[7]==1 and goal[2]==1):
        print('6')
    if(goal[5]==1 and goal[3]==1):
        print('7')
    if(goal[6]==1 and goal[3]==1):
        print('8')
```

```
            if(goal[7]==1 and goal[3]==1):
                print('9')
            break
#importing the data from moodle
sound_track_0 = np.loadtxt('ending_0.dat')
time=sound_track_0[:,0]
sound=sound_track_0[:,1]
matplt.title('Time Domain')
matplt.xlabel('time(ms)')
matplt.ylabel('amplitude')
plt.plot(sound,linewidth=0.5)
#do FFT to all the imported data
#manually cutting the data
soundf_0=np.fft.fft(sound[1900:2200])
soundf_1=np.fft.fft(sound[2300:2500])
soundf_2=np.fft.fft(sound[2700:2900])
soundf_3=np.fft.fft(sound[4100:4300])
soundf_4=np.fft.fft(sound[4500:4700])
soundf_5=np.fft.fft(sound[4800:5100])
soundf_6=np.fft.fft(sound[6200:6400])
soundf_7=np.fft.fft(sound[6600:6800])
soundf_8=np.fft.fft(sound[7000:7250])
soundf_9=np.fft.fft(sound[8200:8400])
soundf_10=np.fft.fft(sound[8500:8700])

analogf_detector(abs(soundf_0))
analogf_detector(abs(soundf_1))
analogf_detector(abs(soundf_2))
analogf_detector(abs(soundf_3))
analogf_detector(abs(soundf_4))
analogf_detector(abs(soundf_5))
analogf_detector(abs(soundf_6))
analogf_detector(abs(soundf_7))
analogf_detector(abs(soundf_8))
analogf_detector(abs(soundf_9))
analogf_detector(abs(soundf_10))
```

Complete Python code for Q5(atan):

```
import numpy as np

import scipy.io.wavfile as wavfile

#import the data

rate,data = wavfile.read("Q3.wav")

dataf=np.fft.fft(data)

#setup variable

high_pass_f1=800

#follow the given .cpp file and adjust it to python style

dataf_part1=dataf[0:int(len(dataf)*(high_pass_f1/rate))]

dataf_part2=dataf[int(len(dataf)*(high_pass_f1/rate)):len(dataf)-1-int(len(dataf)*(high_pass_f1/rate))]#mirror

dataf_part3=dataf[len(dataf)-1-int(len(dataf)*(high_pass_f1/rate)):len(dataf)]

dataf_arctan=np.arctan(dataf_part2)

dataf_combine=np.hstack((dataf_part1,dataf_part2+dataf_arctan,dataf_part3))

data_result=np.fft.ifft(dataf_combine)

data_result=np.real(data_result)

data_clean=np.float32(data_result/np.max(np.abs(data_result)))

wavfile.write('answer_to_Q5_atan.wav',44100,data_clean)

#arctan succeed
```

Complete Python code for Q5(cube1):

```
import numpy as np

import scipy.io.wavfile as wavfile

#import data

rate,data = wavfile.read("Q3.wav")

dataf=np.fft.fft(data)

#setup variable

high_pass_f1=800

#follow the given .cpp file and adjust it to python style

dataf_part1=dataf[0:int(len(dataf)*(high_pass_f1/rate))]

dataf_part2=dataf[int(len(dataf)*(high_pass_f1/rate)):len(dataf)-1-int(len(dataf)*(high_pass_f1/rate))]

dataf_part3=dataf[len(dataf)-1-int(len(dataf)*(high_pass_f1/rate)):len(dataf)]

coefficient=np.max(np.abs(dataf_part2))

#normalized

dataf_part2_normalized=dataf_part2/coefficient

dataf_cube=np.power(dataf_part2_normalized,3)

dataf_combine=np.hstack((dataf_part1,dataf_part2+dataf_cube,dataf_part3))

data_result=np.fft.ifft(dataf_combine)

data_result=np.real(data_result)
```

```
data_clean=np.float32(data_result/np.max(np.abs(data_result)))
wavfile.write('answer_to_Q5_cube.wav',44100,data_clean)
#cube succeed
```

Complete Python code for Q5(cube2):

```
import numpy as np
import scipy.io.wavfile as wavfile
#import the data
rate,data = wavfile.read("Q3.wav")
dataf=np.fft.fft(data)
#setup variable
high_pass_f1=800
#follow the given .cpp file and adjust it to python style
dataf_part1=dataf[0:int(len(dataf)*(high_pass_f1/rate))]
dataf_part2=dataf[int(len(dataf)*(high_pass_f1/rate)):len(dataf)-1-int(len(dataf)*(high_pass_f1/rate))]#mirror
dataf_part3=dataf[len(dataf)-1-int(len(dataf)*(high_pass_f1/rate)):len(dataf)]
dataf_arctan=np.arctan(dataf_part2*dataf_part2*dataf_part2)
dataf_combine=np.hstack((dataf_part1,dataf_part2+dataf_arctan,dataf_part3))
data_result=np.fft.ifft(dataf_combine)
data_result=np.real(data_result)
data_clean=np.float32(data_result/np.max(np.abs(data_result)))
wavfile.write('answer_to_Q5_cube2.wav',44100,data_clean)
#cube2 succeed
```