

# DSP assignment1 report

Jingyan Wang, 2533494w  
Qianqian Wang , 2595993w

## Contents

|          |  |           |
|----------|--|-----------|
| <b>1</b> | <b>Task1</b>   | <b>2</b>  |
| <b>2</b> | <b>Task2</b>   | <b>2</b>  |
| <b>3</b> | <b>Task3</b>   | <b>3</b>  |
| <b>4</b> | <b>Task4</b>   | <b>3</b>  |
| <b>5</b> | <b>Task5</b>   | <b>5</b>  |
| 5.1      | 5.a . . . . .  | 5         |
| 5.2      | 5.b . . . . .  | 7         |
| <b>6</b> | <b>Declaration of Originality and Submission Information</b> | <b>9</b>  |
| <b>7</b> | <b>Appendix</b>  | <b>10</b> |
| 7.1      | voiceEnhancer.py . . . . .                                   | 10        |
| 7.2      | touchToneDecoder.py . . . . .                                | 13        |

# 1 Task1

The record can be found in `"/Resources/original.wav"`

# 2 Task2

The time domain is as Figure1, the frequency domain is as Figure2

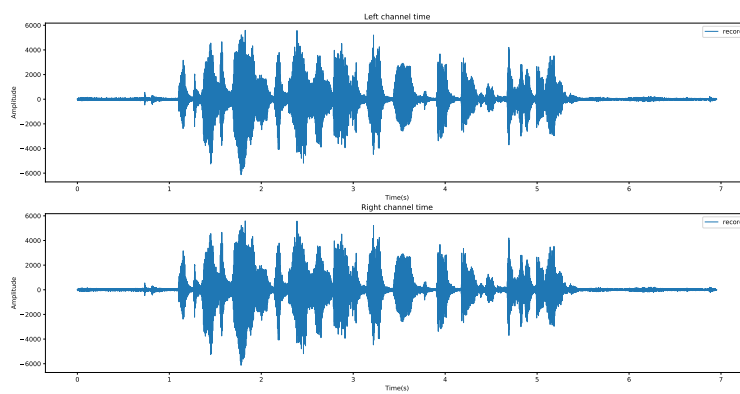


Figure 1: time domain

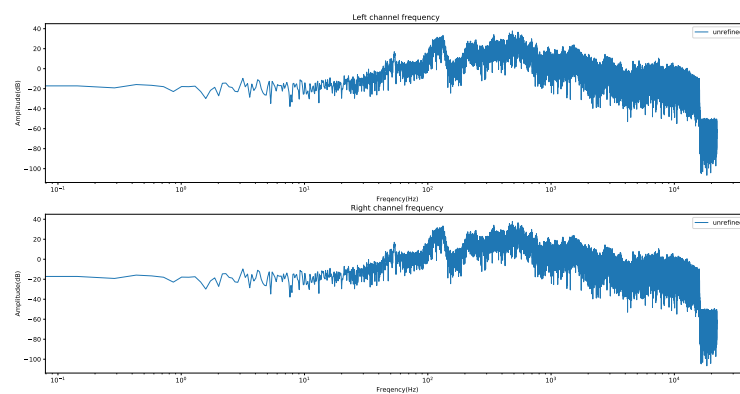


Figure 2: frequency domain

### 3 Task3

As shown by Figure2, the vowel fundamental frequency is around 127Hz, and the containing the consonants is around 170-4200Hz

### 4 Task4

To amplify the amplitude of the waveform, we created a window function(Figure3) whose amplitude is 5 in 120-900Hz and 6K-10KHz. Then, multiplied it with the frequency domain of the original signal (amplify the original signal 5 times in both base and higher frequency range):

$$f_{out} = f * f_{window}$$

Finally, we turn it back to a time series by ifft:

```
1 w = np.ones(N)
2 modifyWindow(w, 120, 900, rate, 5)
3 modifyWindow(w, 6000, 10000, rate, 5)
4
5 lchannelRefine = lchannel*f*w
6 rchannelRefine = rchannel*f*w
7
8 lchannelRefine = np.fft.ifft(lchannelRefine)
9 rchannelRefine = np.fft.ifft(rchannelRefine)
10
11 def modifyWindow(w, startFrequency, endFrequency, sampleRate,
12     value):
13     """modify the window function into rectangular form"""
14     beginPoint = int(startFrequency//(sampleRate/N))
15     endPoint = int(endFrequency//(sampleRate/N))
16
17     w[beginPoint:endPoint] = value
18     w[-endPoint:-beginPoint] = value
```

we could find the variation in both time domain and frequency domain with Figure4 and Figure5

Finally, Output the wavfile in "./Output/improved.wav":

```
1 writeWavefile(outputWaveAddress, rate, lchannelRefine.astype(np.
    int16), rchannelRefine.astype(np.int16))
```

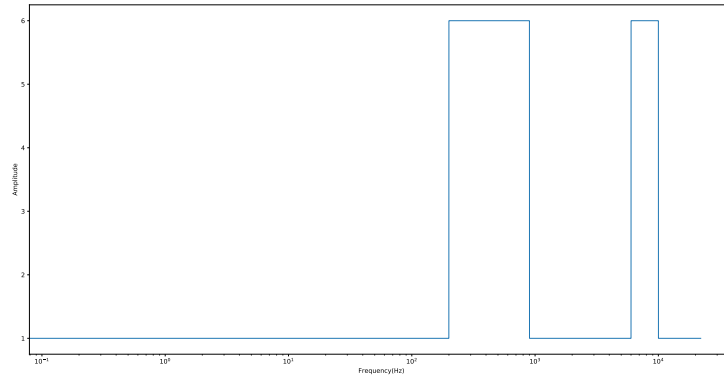


Figure 3: window

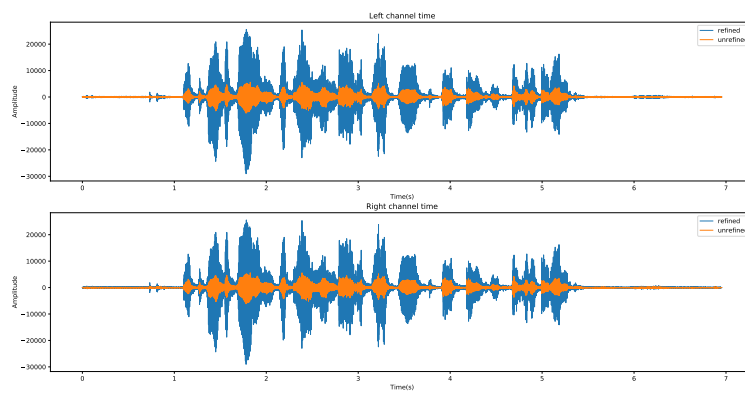


Figure 4: difference in time domain

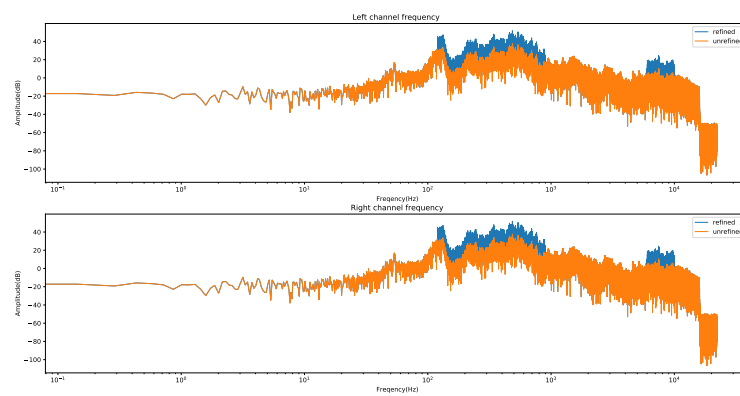


Figure 5: difference in frequency domain

## 5 Task5

### 5.1 5.a

The DTMF telephone keypad is laid out as a matrix of push buttons in which each row represents a low frequency component and each column represent a high frequency component of DTMF signal. So, in specific, we define the DTMF keypad as 3 data structures:

```
1 dtmfHighFrequency = (1209, 1336, 1477, 1633)
2 dtmfLowFrequency = (697, 770, 852, 941)
3 dtmfLetter = [['1', '2', '3', 'A'],
4               ['4', '5', '6', 'B'],
5               ['7', '8', '9', 'C'],
6               ['*', '0', '#', 'D']]
```

Because frequencies of both higher and lower components are all greater than Nyquist frequency (500Hz), so we needed to convert them into 0-500Hz using periodicity and symmetry property.

Secifically, we could convert the higher components by periodicity:

$$f_{convert} = f_{signal} - f_{sampling}$$

And the lower component by symmetry:

$$f_{convert} = f_{sampling} - f_{signal}$$

In code:

```
1 def aliasingFrequency(fs, sampleRate):
2     """convert the signal frequency into (0, N/2)"""
```

The frequency spectrum of a DTMF sinal has a distinct fearture. In the range of  $[0, f_{Nyquist}]$ , it have 3 peaks, one is the DC component of the signal at 0Hz and the other two are belong to DTMF frequency (Figure6).

Thus, we need firstly find the 2 peaks:

```
1 def peakFinding(data):
2     """find the max value of an array"""
3
4 def peakFindingDouble(data):
5     """find the first 2 greatest value of an array, except the 0
   point"""
```

And figured out which DTMF component these peaks belonged:

```
1 def findFrequencyBelong(f, dtmfMin, dtmfMax, sampleRate):
2     """
3     Parameters
```

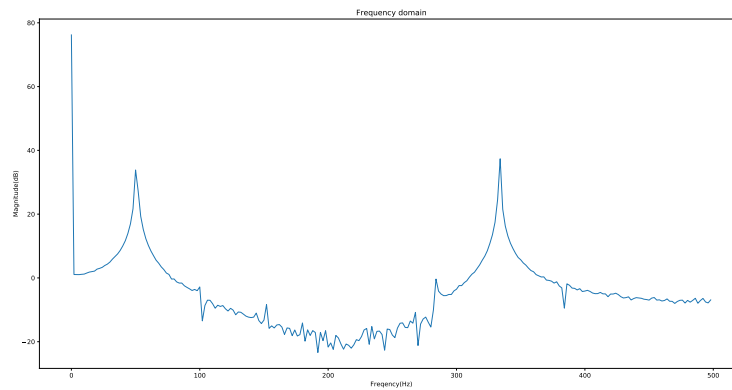


Figure 6: frequency spectrum example of single chunk

```

4  -----
5  f:
6  input frequency of the chunk signal
7  dtmfMin:
8  acceptable lower bound
9  dtmfMax:
10 acceptable high bound
11 sampleRate:
12 sampling frequency
13
14 Returns
15 -----
16 flag:
17 define which tone of the frequency belongs (high or low)
18 index:
19 return the index of dtmfFrequency array for easy finding of
    letter
20 """

```

After we found the index corresponding to the position in DTMF frequency list, we could easily find the number by:

```
1 dtmfLetter[index1][index2]
```

Consequently, we could finish the function

```

1 def detectOneDigitFromChunk(data, sampleRate):
2     """
3     detect each chunk
4
5     Parameters
6     -----
7     data: ndarray

```

```

8 series of chunk data in time domain
9 sampleRate: float
10 the sampling frequency of signal
11
12 Returns
13 -----
14 letter: string
15 goal letter of this chunk 'N' means no letter found
16 """

```

## 5.2 5.b

In the time domain, the signal `"/Resources/TouchToneData/msc_matrix_4"` could be plotted as Figure7. We detected each chunk by rising edge. Specifically, we defined the length of each chunk as 300(300ms). A rising-edge chunk should be:

- the previous chunk should contains no DTMF number
- this chunk should contains a legal DTMF number

In code:

```

1 (preResult=='N') & (result != 'N')

```

Thus we could finish the function by while loop:

```

1 def autoDetectNumbers(data, sampleRate):
2     """
3     Parameters
4     -----
5     data : ndarray
6     touch tone data
7     sampleRate : int or float
8     sampling frequency
9
10    Returns
11    -----
12    seriesNumber : String
13    the number detected
14    """
15    while gap-1+K*gap < N:
16        result = detectOneDigitFromChunk(data[K*gap: gap-1+K*gap],
17                                           sampleRate)
18        if((preResult=='N') & (result != 'N')):
19            #print(K*gap*T,'s', "-", (gap-1+K*gap)*T,'s')
20            seriesNumber = seriesNumber + result
21            preResult = result
22            K = K + 1
23    return seriesNumber

```

Finally, the output is:

```
1 start finding raising edge chunk
2 1.2 s - 1.499 s: 0
3 4.5 s - 4.799 s: 0
4 9.6 s - 9.899000000000001 s: 3
5 13.200000000000001 s - 13.499 s: 3
6 16.8 s - 17.099 s: 1
7 20.400000000000002 s - 20.699 s: 4
8 24.0 s - 24.299 s: 0
9 27.3 s - 27.599 s: 2
10 31.2 s - 31.499000000000002 s: 0
11 35.7 s - 35.999 s: 5
12 39.6 s - 39.899 s: 3
13 43.2 s - 43.499 s: 1
14 47.7 s - 47.999 s: 7
15 ./Resources/TouchToneData/msc_matric_4.dat:
16 final result: 0033140205317
```

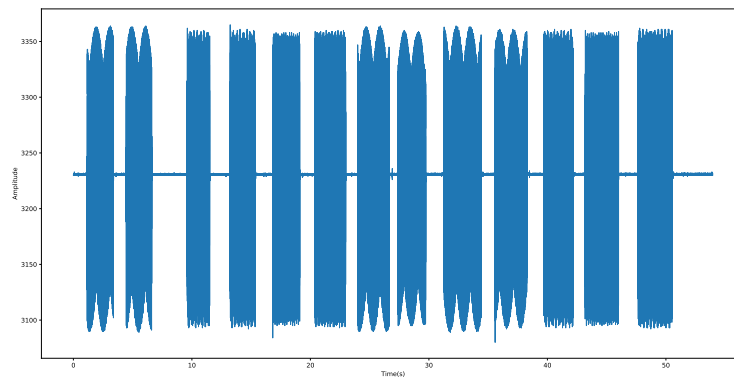


Figure 7: file msc matric 4



## **6 Declaration of Originality and Submission Information**

I affirm that this submission is my own / the groups original work in accordance with the University of Glasgow Regulations and the School of Engineering Requirements.

- Student Number: 2533494w Student Name: Jingyan Wang
- Student Number: 2595993w Student Name: Qianqian Wang

## 7 Appendix

### 7.1 voiceEnhancer.py

```
1  #!/usr/bin/env python3
2  # -*- coding: utf-8 -*-
3  """
4  Created on Mon Oct 19 12:20:00 2020
5
6  @author: wayenvan
7  """
8
9  """import essential modules"""
10 import os
11 import numpy as np
12 import matplotlib.pyplot as plt
13 from scipy.io import wavfile
14
15 """define functions"""
16 def readWavefile(address):
17     """read wave file and devide into two channel"""
18     # check if the file is a wave
19     assert os.path.splitext(address)[-1]==".wav"
20     sampleRate, wav = wavfile.read(address)
21     # make sure the wave is 2 channel
22     assert len(wav.shape) == 2
23
24
25     lchannel = wav[:,0]
26     rchannel = wav[:,1]
27
28     return (sampleRate, lchannel, rchannel)
29
30 def writeWavefile(address, sampleRate, lchannel, rchannel):
31     """read wave file from folder"""
32     #merge left and right channel
33     data = np.hstack((lchannel[...,np.newaxis], rchannel[...,np.newaxis]))
34     wavfile.write(address, sampleRate, data)
35
36
37 def subPlot(x, y, xlabel, ylabel, legend, title, xscale="linear",
38             , yscale="linear"):
39     """plot each subplot in time domain """
40     plt.title(title)
41     plt.plot(x, y, label=legend)
42     plt.xlabel(xlabel)
43     plt.ylabel(ylabel)
```

```

43     plt.xscale(xscale)
44     plt.yscale(yscale)
45     plt.legend()
46
47 def wavePlotT(figure, x, lchannel, rchannel, legend="waveform"):
48     """plot all channels of wave in time domain once"""
49     xlabel="Time(s)"
50     ylabel="Amplitude"
51
52     plt.figure(figure,figsize=(20,10))
53     plt.subplot(2,1,1)
54     subPlot(x, lchannel, xlabel, ylabel, legend, title="Left
channel time")
55     plt.subplot(2,1,2)
56     subPlot(x, rchannel, xlabel, ylabel, legend, title="Right
channel time")
57
58
59 def wavePlotF(figure, xf, lchannelf, rchannelf, legend="waveform
"):
60     """plot all channels of signal in frequency dowmain once"""
61     xlabel="Frequency(Hz)"
62     ylabel="Amplitude(dB)"
63
64     plt.figure(figure, figsize=(20,10))
65     plt.subplot(2,1,1)
66     subPlot(xf, lchannelf, xlabel, ylabel, legend, title="Left
channel frequency ", xscale = "log")
67     plt.subplot(2,1,2)
68     subPlot(xf, rchannelf, xlabel, ylabel, legend, title="Right
channel frequency", xscale = "log")
69
70 def generateXf(sampleRate, N):
71     """generateXf for frequeny domain"""
72     return np.linspace(0.0, (N-1)*sampleRate/N, N)
73
74 def generateXt(sampleRate, N):
75     """generateXt for time domain"""
76     return np.linspace(0.0, (N-1)*1/sampleRate, N)
77
78 def mag2dB(yf):
79     """ change magnitude into dB form """
80     return 20*np.log10(yf)
81
82 def modifyWindow(w, startFrequency, endFrequency, sampleRate,
value):
83     """modify the window function into rectangular form"""
84     N = len(w)
85

```

```

86     beginPoint = int(startFrequency//(sampleRate/N))
87     endPoint = int(endFrequency//(sampleRate/N))
88
89     w[beginPoint:endPoint] = value
90     w[-endPoint:-beginPoint] = value
91
92     """main function """
93
94     inputWaveAddress = "./resources/original.wav"
95     outputWaveAddress = "./Output/improved.wav"
96     figurePath = "./Output/Figures/"
97
98     (rate, lchannel, rchannel) = readWavefile(inputWaveAddress)
99
100    N = np.size(lchannel)
101    T = 1.0/rate
102    xt = generateXt(rate, N)
103    xf = generateXf(rate, N)
104
105    #plot time domain wave
106    #wavePlotT(xt, lchannel, rchannel)
107
108    """start fft"""
109    #caculate fft
110    lchannelf = np.fft.fft(lchannel)
111    rchannelf = np.fft.fft(rchannel)
112
113    #calculate PSD
114    PSDlchannelf = np.abs(lchannelf)**2 / N
115    PSDrchannelf = np.abs(rchannelf)**2 / N
116
117    """task4 refine the record"""
118    #generate window
119    w = np.ones(N)
120    modifyWindow(w, 120, 900, rate, 5)
121    modifyWindow(w, 6000, 10000, rate, 5)
122
123    lchannelfRefine = lchannelf*w
124    rchannelfRefine = rchannelf*w
125
126    lchannelRefine = np.fft.ifft(lchannelfRefine)
127    rchannelRefine = np.fft.ifft(rchannelfRefine)
128
129    """plot and save all figures"""
130    wavePlotT("time domain Record", xt, lchannel, rchannel, legend="
        record")
131    #plt.savefig("./Output/Figures/recordT.pdf")
132
133    wavePlotF("frequency domain Record", xf[0:N//2], mag2dB(2/N*np.

```

```

134     abs(lchannelf[0:N//2])), mag2dB(2/N*np.abs(rchannelf[0:N//2])
135     ), legend="unrefined")
136 #plt.savefig("./Output/Figures/recordF.pdf")
137
138 #plot time domain wave form
139 wavePlotT("timedomainReference", xt, lchannelRefine.astype(np.
140     int16), rchannelRefine.astype(np.int16), legend="refined")
141 wavePlotT("timedomainReference", xt, lchannel, rchannel, legend=
142     "unrefined")
143 #plt.savefig(filePath+"recordTR.pdf")
144
145 #plot frequency
146 wavePlotF("frequencydomainReference", xf[0:N//2], mag2dB(2/N*np.
147     abs(lchannelfRefine[0:N//2])), mag2dB(2/N*np.abs(
148     rchannelfRefine[0:N//2])), legend="refined")
149 wavePlotF("frequencydomainReference", xf[0:N//2], mag2dB(2/N*np.
150     abs(lchannelf[0:N//2])), mag2dB(2/N*np.abs(rchannelf[0:N//2])
151     ), legend="unrefined")
152 #plt.savefig(filePath+"recordFR.pdf")
153
154 #plot the window
155 #plt.figure(figsize=(20,10))
156 #plt.plot(xf[0:N//2], w[0:N//2])
157 #plt.xlabel("Frequency(Hz)")
158 #plt.xscale("log")
159 #plt.ylabel("Amplitude")
160 #plt.savefig(filePath+"window.pdf")
161
162 plt.show()
163
164 """export the .wav file"""
165 writeWavefile(outputWaveAddress, rate, lchannelRefine.astype(np.
166     int16), rchannelRefine.astype(np.int16))

```

## 7.2 touchToneDecoder.py

```

1 #!/usr/bin/env python3
2 # -*- coding: utf-8 -*-
3 """
4 Created on Mon Oct 19 12:15:25 2020
5
6 @author: wayenvan
7 """
8 """import essential modules"""
9 import numpy as np
10 import matplotlib.pyplot as plt
11 from enum import Enum
12

```

```

13 """define globael number"""
14 dtmfHighFrequency = (1209, 1336, 1477, 1633)
15 dtmfLowFrequency = (697, 770, 852, 941)
16 dtmfLetter = [['1', '2', '3', 'A'],
17               ['4', '5', '6', 'B'],
18               ['7', '8', '9', 'C'],
19               ['*', '0', '#', 'D']]
20
21 class ToneFlag(Enum):
22     low = 0
23     high = 1
24     NoFind = 2
25
26 """define functions"""
27 def generateXf(sampleRate, N):
28     """generateXf for frequeny domain"""
29     return np.linspace(0.0, (N-1)*sampleRate/N, N)
30
31 def generateXt(sampleRate, N):
32     """generateXt for time domain"""
33     return np.linspace(0.0, (N-1)*1/sampleRate, N)
34
35 def mag2dB(yf):
36     """ change magnitude into dB form """
37     return 20*np.log10(yf)
38
39 def peakFinding(data):
40     """finding the max value of an array"""
41     maxIndex = -1
42     maxValue = 0
43
44     for i in range(len(data)):
45         if(data[i]>maxValue):
46             maxIndex = i
47             maxValue = data[i]
48
49     return maxIndex
50
51 def peakFindingDouble(data):
52     """finding the first 2 greatest value of an array"""
53     indexMax = peakFinding(data[1:])+1
54
55     indexTemp1 = peakFinding(data[1:indexMax-1])+1
56     indexTemp2 = peakFinding(data[indexMax+1:])+indexMax+1
57
58     if(data[indexTemp1]>=data[indexTemp2]):
59         indexMaxSec = indexTemp1
60     elif(data[indexTemp2]>data[indexTemp1]):
61         indexMaxSec = indexTemp2

```

```

62
63     ret = [indexMaxSec, indexMax]
64
65     return ret
66
67 def aliasingFrequency(fs, sampleRate):
68     """convert the signal frequency into (0, N/2)"""
69     N = int(fs/sampleRate+0.5)
70     return abs(fs-N*sampleRate)
71
72 def findFrequencyBelong(f, dtmfMin, dtmfMax, sampleRate):
73     """
74     Parameters
75     -----
76     f:
77         input frequency of the chunk signal
78     dtmfMin:
79         acceptable lower bound
80     dtmfMax:
81         acceptable high bound
82     sampleRate:
83         sampling frequency
84
85     Returns
86     -----
87     flag:
88         define which tone of the frequency belongs (high or low)
89     index:
90         return the index of dtmfFrequency array for easy finding
          of letter
91
92     """
93
94     for indexLow in range(4):
95         for indexHigh in range(4):
96             if(f-dtmfMin<aliasingFrequency(dtmfHighFrequency[
indexHigh], sampleRate)<f+dtmfMax):
97                 flag = ToneFlag.high
98                 return flag, indexHigh
99             if(f-dtmfMin<aliasingFrequency(dtmfLowFrequency[
indexLow], sampleRate)<f+dtmfMax):
100                 flag = ToneFlag.low
101                 return flag, indexLow
102     return ToneFlag.NoFind, -1
103
104 def detectOneDigitFromChunk(data, sampleRate):
105     """
106     detect each chunk
107

```

```

108 Parameters
109 -----
110 data: ndarray
111     series of chunk data in time domain
112 sampleRate: float
113     the sampling frequency of signal
114
115 Returns
116 -----
117 letter: string
118     goal letter of this chunk 'N' means no letter found
119 ""
120 #prepare the data
121 dataf = np.fft.fft(data)
122 N=len(data)
123 minMagnitude = 30
124 #cut the data half
125 rdataf = dataf[0:N//2]
126
127 dtmfMin = 9
128 dtmfMax = 9
129
130 #calculate the peak point
131 #ind = np.argmaxpartition(abs(rdataf), -3)[-3:]
132 ind = peakFindingDouble(abs(rdataf))
133
134 #cut out small signal
135 if((2/N*(abs(rdataf)[ind[0]])<minMagnitude) | (2/N*(abs(
136 rdataf)[ind[1]])<minMagnitude)):
137     return 'N'
138
139 f1 = ind[0]*(sampleRate/N)
140 f2 = ind[1]*(sampleRate/N)
141
142 #print(f1, f2)
143 #start the for loop to check if the frequency meet any of
144 high or low frequency of dtmf
145 (flag1, index1) = findFrequencyBelong(f1, dtmfMin, dtmfMax,
146 sampleRate)
147 (flag2, index2) = findFrequencyBelong(f2, dtmfMin, dtmfMax,
148 sampleRate)
149
150 #find out corresponding point of this 2 frequency
151 if((flag1==ToneFlag.high) & (flag2==ToneFlag.low)):
152     return dtmfLetter[index2][index1]
153 elif((flag1==ToneFlag.low) & (flag2==ToneFlag.high)):
154     return dtmfLetter[index1][index2]
155 elif((flag1==ToneFlag.NoFind)|(flag2==ToneFlag.NoFind)):
156     #print("index1:", index1, flag1, "index2", index2, flag2

```



```

153         return 'N'
154     else:
155         return 'N'
156
157 def autoDetectNumbers(data, sampleRate):
158     """
159
160     Parameters
161     -----
162     data : ndarray
163         touch tone data
164     sampleRate : int or float
165         sampling frequency
166
167     Returns
168     -----
169     seriesNumber : String
170         the number detected
171
172     """
173     K = 0
174     N = len(data)
175     gap = 300          #the length of eah chunk
176     T = 1/sampleRate
177
178     preResult = 'N'
179     seriesNumber = ''
180
181     #start checking numbers
182     print("start finding raising edge chunk")
183     while gap-1+K*gap < N:
184         result = detectOneDigitFromChunk(data[K*gap: gap-1+K*gap
185 ], sampleRate)
186         if((preResult=='N') & (result != 'N')):
187             print(K*gap*T,'s', "-", (gap-1+K*gap)*T,'s:',result)
188             seriesNumber = seriesNumber + result
189             preResult = result
190             K = K + 1
191
192     return seriesNumber
193
194 """main function"""
195 #load .dat file, if change i, it can load all files
196 figurePath = "./Output/Figures/"
197 for i in range(4,5):
198     dataAddress = './Resources/TouchToneData/msc_matric_'+str(i)
199     +'.dat'

```

```

199     dataI = np.loadtxt(dataAddress, usecols=(1), dtype=np.int16)
200
201     data = dataI
202     Fs2 = 1000
203     N2 = len(data)
204     x2 = range(N2)
205     xt2 = generateXt(Fs2, N2)
206     xf2 = generateXf(Fs2, N2)
207     dataf = np.fft.fft(data)
208
209     series = autoDetectNumbers(data, Fs2)
210     print(dataAddress+":")
211     print("final result: ", series)
212
213     """plot and save all figures"""
214     #plot task5 wave
215     # plt.figure(figsize=(20,10))
216     # plt.plot(xt2, data)
217     # plt.xlabel("Time(s)")
218     # plt.ylabel("Amplitude")
219     # plt.savefig(figurePath+"DTMFtime.pdf")
220
221     # plt.figure(figsize=(20,10))
222     # plt.plot(xf2[0:N2//2], mag2dB(abs(2/N2*dataf[0:N2//2])))
223     # plt.title("Frequency domain")
224     # plt.xlabel("Frequency(Hz)")
225     # plt.ylabel("Magnitude(dB)")
226     # plt.savefig(figurePath+"task5ExampleF.pdf")
227
228     #plt.show()

```