

# 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>Code</b>	<b>9</b>

# 1 Task1

The record can be found in `"/Resources/recording1.wav"`

# 2 Task2

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

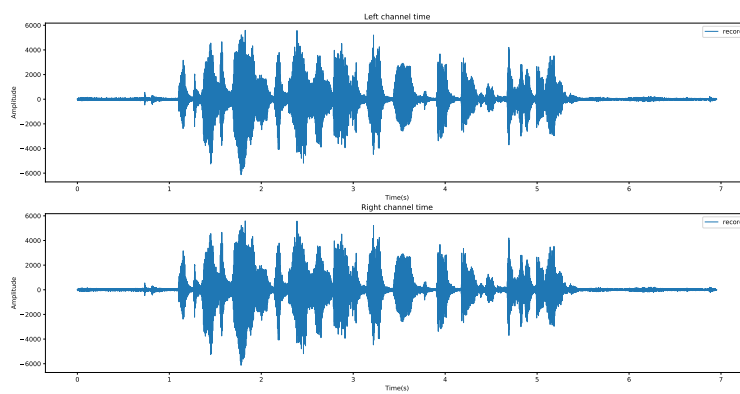


Figure 1: time domain

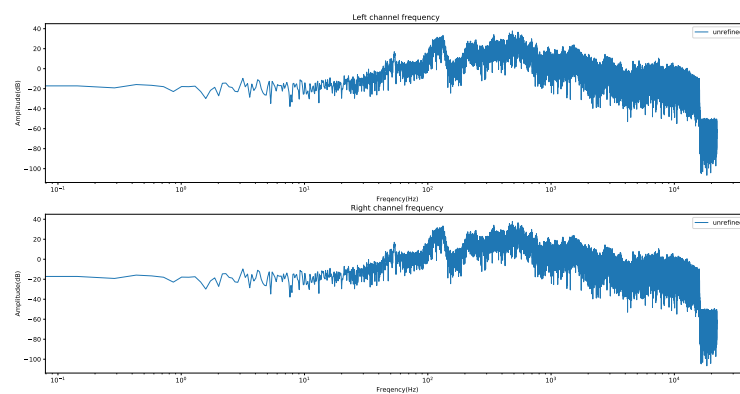


Figure 2: frequency domain

### 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 can find difference in both time domain and frequency domain with Figure4 and Figure5

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

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

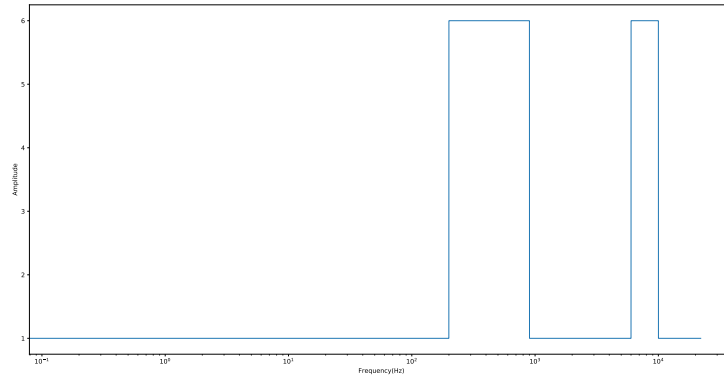


Figure 3: window

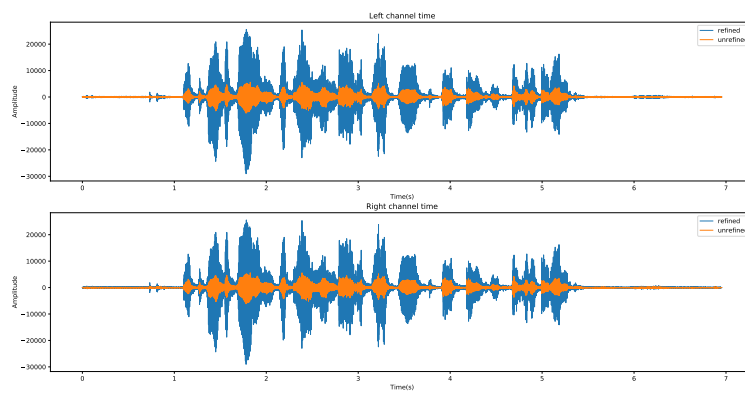


Figure 4: recordTR

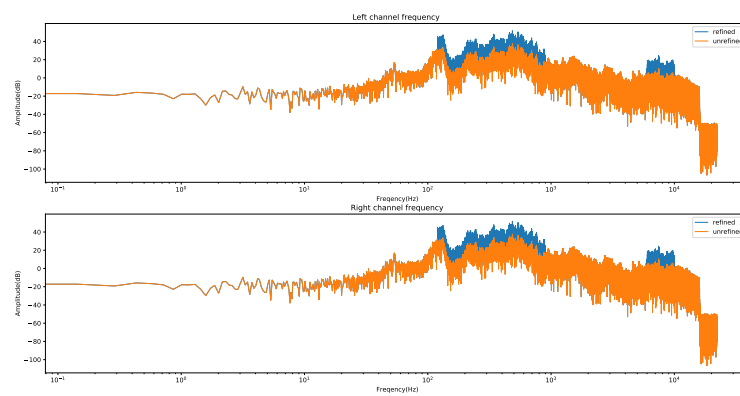


Figure 5: recordFR

## 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
```

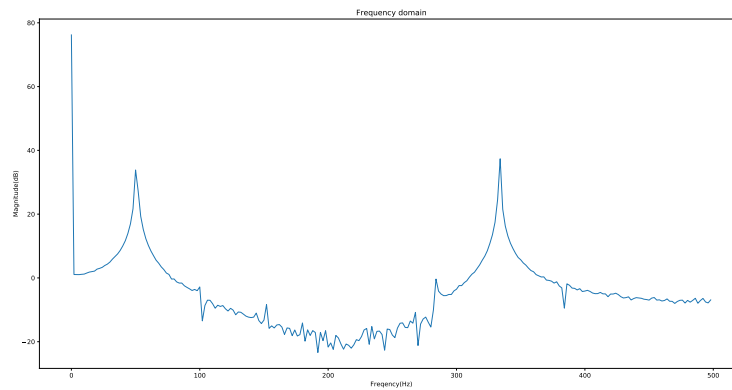


Figure 6: frequency spectrum example of single chunk

```

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
```

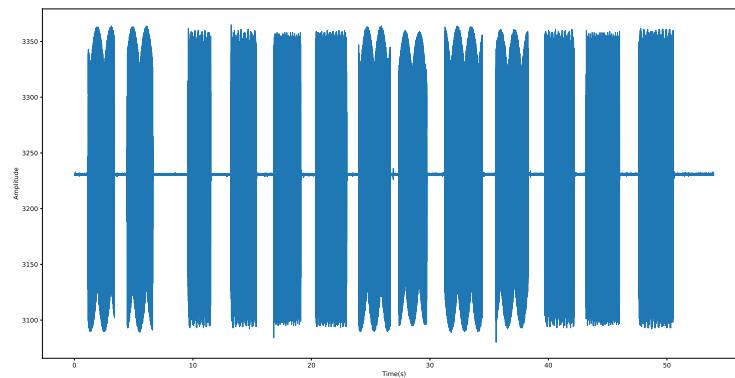


Figure 7: file msc matric 4



## 6 Code

```
1  #!/usr/bin/env python3
2  # -*- coding: utf-8 -*-
3  """
4  Created on Sat Oct 10 23:57:41 2020
5
6  @author: wayenvan
7  """
8
9
10 """import essential modules"""
11 import os
12 import numpy as np
13 import matplotlib.pyplot as plt
14 import matplotlib
15 from scipy.io import wavfile
16 from enum import Enum
17
18 """change font to fit Latex"""
19 matplotlib.rcParams['mathtext.fontset'] = 'custom'
20 matplotlib.rcParams['mathtext.rm'] = 'Times New Roman'
21 matplotlib.rcParams['mathtext.it'] = 'Times New Roman'
22 matplotlib.rcParams['mathtext.bf'] = 'Times New Roman'
23
24 """define globael number"""
25 dtmfHighFrequency = (1209, 1336, 1477, 1633)
26 dtmfLowFrequency = (697, 770, 852, 941)
27 dtmfLetter = [['1', '2', '3', 'A'],
28               ['4', '5', '6', 'B'],
29               ['7', '8', '9', 'C'],
30               ['*', '0', '#', 'D']]
31
32 class ToneFlag(Enum):
33     low = 0
34     high = 1
35     NoFind = 2
36
37 """define functions"""
38 def readWavefile(address):
39     """read wave file and devide into two channel"""
40     # check if the file is a wave
41     assert os.path.splitext(address)[-1]==".wav"
42     sampleRate, wav = wavfile.read(address)
43     # make sure the wave is 2 channel
44     assert len(wav.shape) == 2
45
46
```

```

47     lchannel = wav[:,0]
48     rchannel = wav[:,1]
49
50     return (sampleRate, lchannel, rchannel)
51
52 def writeWavefile(address, sampleRate, lchannel, rchannel):
53     """read wave file from folder"""
54     #merge left and right channel
55     data = np.hstack((lchannel[...,np.newaxis], rchannel[...,np.
newaxis]))
56     wavfile.write(address, sampleRate, data)
57
58
59 def subPlot(x, y, xlabel, ylabel, legend, title, xscale="linear"
, yscale="linear"):
60     """plot each subplot in time domain """
61     plt.title(title)
62     plt.plot(x, y, label=legend)
63     plt.xlabel(xlabel)
64     plt.ylabel(ylabel)
65     plt.xscale(xscale)
66     plt.yscale(yscale)
67     plt.legend()
68
69 def wavePlotT(figure, x, lchannel, rchannel, legend="waveform"):
70     """plot all channels of wave in time domain once"""
71     xlabel="Time(s)"
72     ylabel="Amplitude"
73
74     plt.figure(figure,figsize=(20,10))
75     plt.subplot(2,1,1)
76     subPlot(x, lchannel, xlabel, ylabel, legend, title="Left
channel time")
77     plt.subplot(2,1,2)
78     subPlot(x, rchannel, xlabel, ylabel, legend, title="Right
channel time")
79     plt.show()
80
81
82 def wavePlotF(figure, xf, lchannelf, rchannelf, legend="waveform
"):
83     """plot all channels of signal in frequency dowmain once"""
84     xlabel="Frequency(Hz)"
85     ylabel="Amplitude(dB)"
86
87     plt.figure(figure, figsize=(20,10))
88     plt.subplot(2,1,1)
89     subPlot(xf, lchannelf, xlabel, ylabel, legend, title="Left
channel frequency ", xscale = "log")

```

```

90     plt.subplot(2,1,2)
91     subPlot(xf, rchannelf, xlabel, ylabel, legend, title="Right
    channel frequency", xscale = "log")
92     plt.show()
93
94     def generateXf(sampleRate, N):
95         """generateXf for frequency domain"""
96         return np.linspace(0.0, (N-1)*sampleRate/N, N)
97
98     def generateXt(sampleRate, N):
99         """generateXt for time domain"""
100        return np.linspace(0.0, (N-1)*1/sampleRate, N)
101
102    def mag2dB(yf):
103        """ change magnitude into dB form """
104        return 20*np.log10(yf)
105
106    def modifyWindow(w, startFrequency, endFrequency, sampleRate,
    value):
107        """modify the window function into rectangular form"""
108        beginPoint = int(startFrequency//(sampleRate/N))
109        endPoint = int(endFrequency//(sampleRate/N))
110
111        w[beginPoint:endPoint] = value
112        w[-endPoint:-beginPoint] = value
113
114    def peakFinding(data):
115        """finding the max value of an array"""
116        maxIndex = -1
117        maxValue = 0
118
119        for i in range(len(data)):
120            if(data[i]>maxValue):
121                maxIndex = i
122                maxValue = data[i]
123
124        return maxIndex
125
126    def peakFindingDouble(data):
127        """finding the first 2 greatest value of an array"""
128        indexMax = peakFinding(data[1:])+1
129
130        indexTemp1 = peakFinding(data[1:indexMax-1])+1
131        indexTemp2 = peakFinding(data[indexMax+1:])+indexMax+1
132
133        if(data[indexTemp1]>=data[indexTemp2]):
134            indexMaxSec = indexTemp1
135        elif(data[indexTemp2]>data[indexTemp1]):
136            indexMaxSec = indexTemp2

```

```

137
138     ret = [indexMaxSec, indexMax]
139
140     return ret
141
142 def aliasingFrequency(fs, sampleRate):
143     """convert the signal frequency into (0, N/2)"""
144     N = int(fs/sampleRate+0.5)
145     return abs(fs-N*sampleRate)
146
147 def findFrequencyBelong(f, dtmfMin, dtmfMax, sampleRate):
148     """
149     Parameters
150     -----
151     f:
152         input frequency of the chunk signal
153     dtmfMin:
154         acceptable lower bound
155     dtmfMax:
156         acceptable high bound
157     sampleRate:
158         sampling frequency
159
160     Returns
161     -----
162     flag:
163         define which tone of the frequency belongs (high or low)
164     index:
165         return the index of dtmfFrequency array for easy finding
166         of letter
167
168     """
169
170     for indexLow in range(4):
171         for indexHigh in range(4):
172             if(f-dtmfMin<aliasingFrequency(dtmfHighFrequency[
173 indexHigh], sampleRate)<f+dtmfMax):
174                 flag = ToneFlag.high
175                 return flag, indexHigh
176             if(f-dtmfMin<aliasingFrequency(dtmfLowFrequency[
177 indexLow], sampleRate)<f+dtmfMax):
178                 flag = ToneFlag.low
179                 return flag, indexLow
180     return ToneFlag.NoFind, -1
181
182 def detectOneDigitFromChunk(data, sampleRate):
183     """
184     detect each chunk

```

```

183 Parameters
184 -----
185 data: ndarray
186     series of chunk data in time domain
187 sampleRate: float
188     the sampling frequency of signal
189
190 Returns
191 -----
192 letter: string
193     goal letter of this chunk 'N' means no letter found
194 ""
195 #prepare the data
196 dataf = np.fft.fft(data)
197 N=len(data)
198 minMagnitude = 30
199 #cut the data half
200 rdataf = dataf[0:N//2]
201
202 dtmfMin = 9
203 dtmfMax = 9
204
205 #calculate the peak point
206 #ind = np.argmaxpartition(abs(rdataf), -3)[-3:]
207 ind = peakFindingDouble(abs(rdataf))
208
209 #cut out small signal
210 if((2/N*(abs(rdataf)[ind[0]])<minMagnitude) | (2/N*(abs(
211 rdataf)[ind[1]])<minMagnitude)):
212     return 'N'
213
214 f1 = ind[0]*(sampleRate/N)
215 f2 = ind[1]*(sampleRate/N)
216
217 #print(f1, f2)
218 #start the for loop to check if the frequency meet any of
219 high or low frequency of dtmf
220 (flag1, index1) = findFrequencyBelong(f1, dtmfMin, dtmfMax,
221 sampleRate)
222 (flag2, index2) = findFrequencyBelong(f2, dtmfMin, dtmfMax,
223 sampleRate)
224
225 #find out corresponding point of this 2 frequency
226 if((flag1==ToneFlag.high) & (flag2==ToneFlag.low)):
227     return dtmfLetter[index2][index1]
228 elif((flag1==ToneFlag.low) & (flag2==ToneFlag.high)):
229     return dtmfLetter[index1][index2]
230 elif((flag1==ToneFlag.NoFind)|(flag2==ToneFlag.NoFind)):
231     #print("index1:", index1, flag1, "index2", index2, flag2

```

```

228         return 'N'
229     else:
230         return 'N'
231
232 def autoDetectNumbers(data, sampleRate):
233     """
234
235     Parameters
236     -----
237     data : ndarray
238         touch tone data
239     sampleRate : int or float
240         sampling frequency
241
242     Returns
243     -----
244     seriesNumber : String
245         the number detected
246
247     """
248     K = 0
249     N = len(data)
250     gap = 300          #the length of eah chunk
251     T = 1/sampleRate
252
253     preResult = 'N'
254     seriesNumber = ''
255
256     #start checking numbers
257     print("start finding raising edge chunk")
258     while gap-1+K*gap < N:
259         result = detectOneDigitFromChunk(data[K*gap: gap-1+K*gap
260 ], sampleRate)
261         if((preResult=='N') & (result != 'N')):
262             print(K*gap*T,'s', "-", (gap-1+K*gap)*T,'s:',result)
263             seriesNumber = seriesNumber + result
264             preResult = result
265             K = K + 1
266
267     return seriesNumber
268
269 """main function """
270 inputWaveAddress = "./resources/recordding1.wav"
271 outputWaveAddress = "./Output/refinedVoice.wav"
272 figurePath = "./Output/Figures/"
273
274 (rate, lchannel, rchannel) = readWavefile(inputWaveAddress)

```

```

275
276 N = np.size(lchannel)
277 T = 1.0/rate
278 xt = generateXt(rate, N)
279 xf = generateXf(rate, N)
280
281 #plot time domain wave
282 #wavePlotT(xt, lchannel, rchannel)
283
284 """start fft"""
285 #caculate fft
286 lchannelf = np.fft.fft(lchannel)
287 rchannelf = np.fft.fft(rchannel)
288
289 #calculate PSD
290 PSDlchannelf = np.abs(lchannelf)**2 / N
291 PSDrchannelf = np.abs(rchannelf)**2 / N
292
293 """task4 refine the record"""
294 #generate window
295 w = np.ones(N)
296 modifyWindow(w, 120, 900, rate, 5)
297 modifyWindow(w, 6000, 10000, rate, 5)
298
299 lchannelfRefine = lchannelf*w
300 rchannelfRefine = rchannelf*w
301
302 lchannelRefine = np.fft.ifft(lchannelfRefine)
303 rchannelRefine = np.fft.ifft(rchannelfRefine)
304
305 """task5 result"""
306 #load .dat file, if change i, it can load all files
307 for i in range(4,5):
308     dataAddress = './Resources/TouchToneData/msc_matric_'+str(i)
309     +'.dat'
310     dataI = np.loadtxt(dataAddress, usecols=(1), dtype=np.int16)
311
312     data = dataI
313     Fs2 = 1000
314     N2 = len(data)
315     x2 = range(N2)
316     xt2 = generateXt(Fs2, N2)
317     xf2 = generateXf(Fs2, N2)
318     dataf = np.fft.fft(data)
319
320     series = autoDetectNumbers(data, Fs2)
321     print(dataAddress+":")
322     print("final result: ", series)

```

```

323 """plot and save all figures"""
324 #wavePlotT("timedomainRecord", xt, lchannel, rchannel, legend="
    record")
325 #plt.savefig("./Output/Figures/recordT.pdf")
326
327 #wavePlotF("frequencydomainRecord", xf[0:N//2], mag2dB(2/N*np.
    abs(lchannelf[0:N//2])), mag2dB(2/N*np.abs(rchannelf[0:N//2])
    ), legend="unrefined")
328 #plt.savefig("./Output/Figures/recordF.pdf")
329
330 #plot time domain wave form
331 wavePlotT("timedomainReference", xt, lchannelRefine.astype(np.
    int16), rchannelRefine.astype(np.int16), legend="refined")
332 wavePlotT("timedomainReference", xt, lchannel, rchannel, legend=
    "unrefined")
333 #plt.savefig(filePath+"recordTR.pdf")
334
335 #plot frequency
336 wavePlotF("frequencydomainReference", xf[0:N//2], mag2dB(2/N*np.
    abs(lchannelfRefine[0:N//2])), mag2dB(2/N*np.abs(
    rchannelfRefine[0:N//2])), legend="refined")
337 wavePlotF("frequencydomainReference", xf[0:N//2], mag2dB(2/N*np.
    abs(lchannelf[0:N//2])), mag2dB(2/N*np.abs(rchannelf[0:N//2])
    ), legend="unrefined")
338 #plt.savefig(filePath+"recordFR.pdf")
339
340
341 #plot the window
342 #plt.figure(figsize=(20,10))
343 #plt.plot(xf[0:N//2], w[0:N//2])
344 #plt.xlabel("Frequency(Hz)")
345 #plt.xscale("log")
346 #plt.ylabel("Amplitude")
347 #plt.show()
348 #plt.savefig(filePath+"window.pdf")
349
350 #plot task5 wave
351 plt.figure(figsize=(20,10))
352 plt.plot(xt2, data)
353 plt.xlabel("Time(s)")
354 plt.ylabel("Amplitude")
355 plt.show()
356 #plt.savefig(filePath+"task5ExampleT.pdf")
357 plt.savefig(filePath+"DTMFtime.pdf")
358
359 plt.figure(figsize=(20,10))
360 plt.plot(xf2[0:N2//2], mag2dB(abs(2/N2*dataf[0:N2//2])))
361 plt.title("Frequency domain")
362 plt.xlabel("Frequency(Hz)")

```



```
363 plt.ylabel("Magnitude(dB)")
364 plt.show()
365 #plt.savefig(figurePath+"task5ExampleF.pdf")
366
367 """export the .wav file"""
368 writeWavefile(outputWaveAddress, rate, lchannelRefine.astype(np.
    int16), rchannelRefine.astype(np.int16))
```