DSP assignment1 report

Jingyan Wang, 2533494w
Qianqian Wang , 2595993w

Contents

1 Task1 **2**

2 Task2 **2**

3 Task3 **3**

4 Task4 **3**

5 Task5 **5**
5.1 5.a 5
5.2 5.b 7

6 Declaration of Originality and Submission Information **9**

7 Appendix **10**
7.1 voice_enhancer.py 10
7.2 touchtonedecoder.py 13

1 Task1

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

2 Task2

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

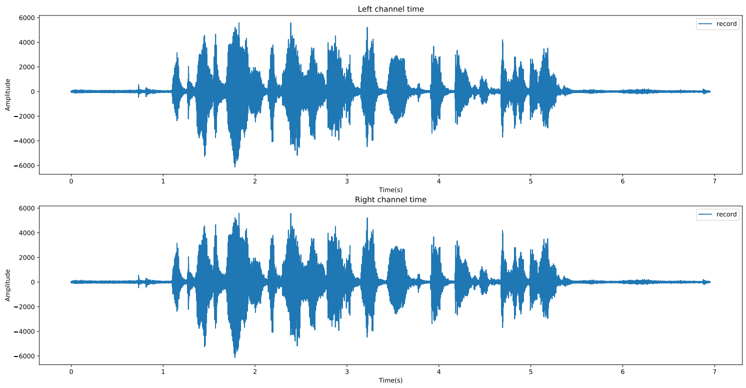

Figure 1: time domain

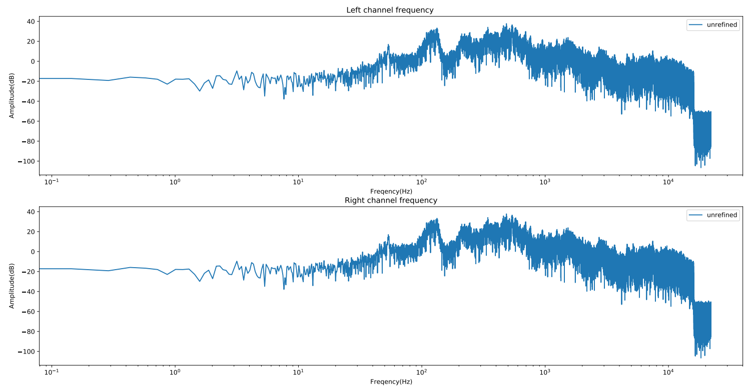

Figure 2: frequency domain

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:

```
w = np.ones(N)
modifyWindow(w, 120, 900, rate, 5)
modifyWindow(w, 6000, 10000, rate, 5)

lchannelfRefine = lchannelf*w
rchannelfRefine = rchannelf*w

lchannelRefine = np.fft.ifft(lchannelfRefine)
rchannelRefine = np.fft.ifft(rchannelfRefine)

def modifyWindow(w, startFreqency, endFreqency, sampleRate,
    value):
  """modify the window function into rectangular form"""
  beginPoint = int(startFreqency//(sampleRate/N))
  endPoint = int(endFreqency//(sampleRate/N))

  w[beginPoint:endPoint] = value
  w[-endPoint:-beginPoint] = value
```

we could find the variation in both time domain and frequency domain with Figure4 and Figure5
Finally, Ouput the wavefile in `"./improved.wav"`:

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

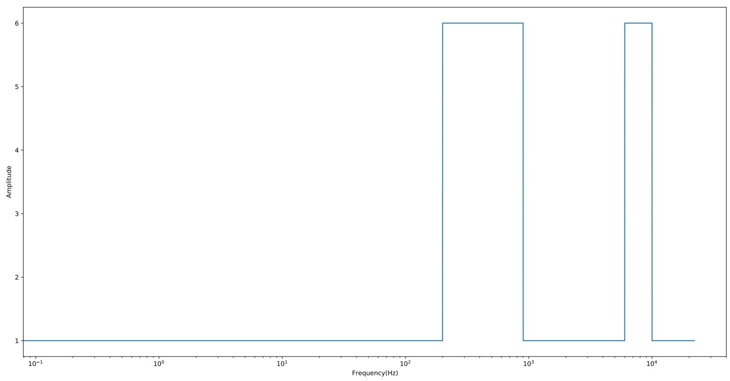

Figure 3: window

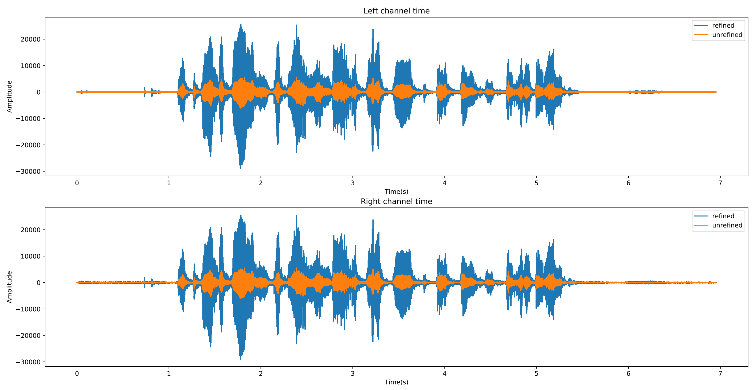

Figure 4: difference in time domain

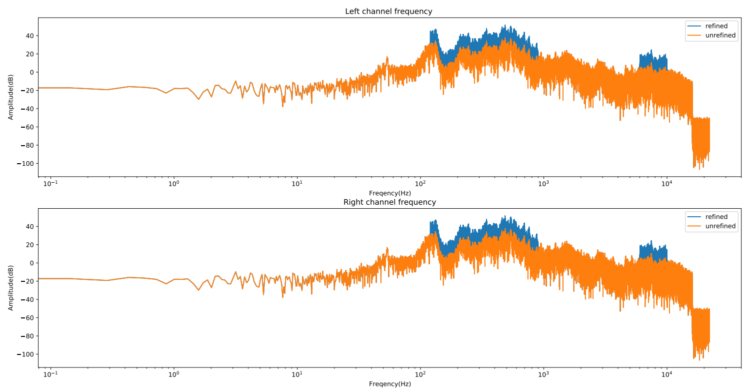

Figure 5: difference in frequency domain

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:

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

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

And the lower component by symmetry:

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

In code:

```
def aliasingFrequency(fs, sampleRate):
  """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:

```
def peakFinding(data):
  """find the max value of an array"""

def peakFindingDouble(data):
  """find the first 2 greatest value of an array, except the 0
    point"""
```

And figured out which DTMF component these peaks belonged:

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

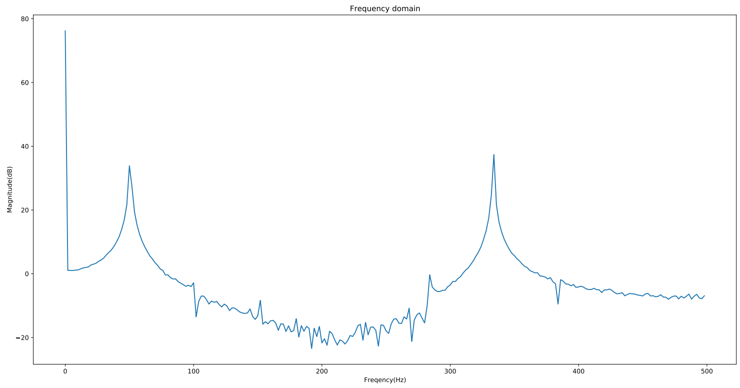

Figure 6: frequency spectrum example of single chunk

```
    ---------
    f:
    input frequency of the chunk signal
    dtmfMin:
    acceptable lower bound
    dtmfMax:
    acceptable high bound
    sampleRate:
    sampling frequency

    Returns
    ------
    flag:
    define which tone of the frequency belongs (high or low)
    index:
    return the index of dtmfFrequency array for easy finding of
     letter
    """
```

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

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

Consequently, we could finish the function

```
def detectOneDigitFromChunk(data, sampleRate):
"""
detect each chunk

Parameters
----------
data: ndarray
```

```
series of chunk data in time domain
sampleRate: float
the sampling frequency of signal

Returns
-------
letter: string
goal letter of this chunk 'N' means no letter found
"""
```

5.2 5.b

In the time domain, the signal `"./Resources/TouchToneData/msc_matric_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:

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

Thus we could finish the function by while loop:

```
def autoDetectNumbers(data, sampleRate):
  """
  Parameters
  ----------
  data : ndarray
  touch tone data
  sampleRate : int or float
  sampling frequency

  Returns
  -------
  seriesNumber : String
  the number detected
  """
  while gap-1+K*gap < N:
  result = detectOneDigitFromChunk(data[K*gap: gap-1+K*gap],
    sampleRate)
  if((preResult=='N') & (result != 'N')):
  #print(K*gap*T,'s', "-", (gap-1+K*gap)*T,'s')
  seriesNumber = seriesNumber + result
  preResult = result
  K = K + 1
  return seriesNumber
```

Finally, the output is:

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

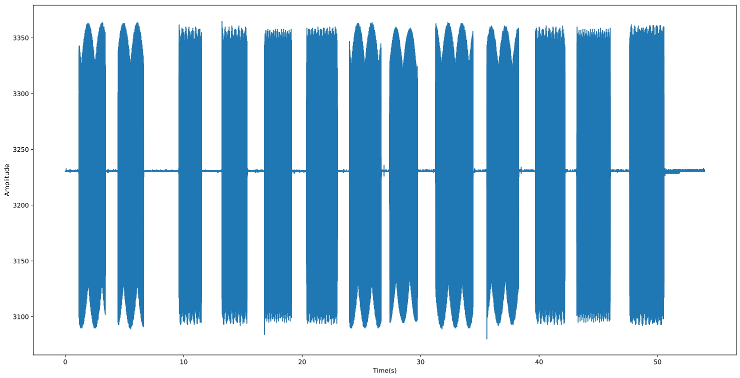

Figure 7: file msc matric 4

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 voice_enhancer.py

```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
"""
Created on Mon Oct 19 12:20:00 2020

@author: wayenvan
"""

"""import essential modules"""
import os
import numpy as np
import matplotlib.pyplot as plt
from scipy.io import wavfile

"""define functions"""
def readWavefile(address):
    """read wave file and devide into two channel"""
    # check if the file is a wave
    assert os.path.splitext(address)[-1]==".wav"
    sampleRate, wav = wavfile.read(address)
    # make sure the wave is 2 channel
    assert len(wav.shape) == 2


    lchannel = wav[:,0]
    rchannel = wav[:,1]

    return (sampleRate, lchannel, rchannel)

def writeWavefile(address, sampleRate, lchannel, rchannel):
    """read wave file from folder"""
    #merge left and right channel
    data = np.hstack((lchannel[...,np.newaxis], rchannel[...,np.newaxis]))
    wavfile.write(address, sampleRate, data)


def subPlot(x, y, xlabel, ylabel, legend, title, xscale="linear", yscale="linear"):
    """plot each subplot in time domain """
    plt.title(title)
    plt.plot(x, y, label=legend)
    plt.xlabel(xlabel)
    plt.ylabel(ylabel)
```

```python
    plt.xscale(xscale)
    plt.yscale(yscale)
    plt.legend()

def wavePlotT(figure, x, lchannel, rchannel, legend="waveform"):
    """plot all channels of wave in time domain once"""
    xlabel="Time(s)"
    ylabel="Amplitude"

    plt.figure(figure,figsize=(20,10))
    plt.subplot(2,1,1)
    subPlot(x, lchannel, xlabel, ylabel, legend, title="Left channel time")
    plt.subplot(2,1,2)
    subPlot(x, rchannel, xlabel, ylabel, legend, title="Right channel time")


def wavePlotF(figure, xf, lchannelf, rchannelf, legend="waveform"):
    """plot all channels of signal in frequency dowmain once"""
    xlabel="Freqency(Hz)"
    ylabel="Amplitude(dB)"

    plt.figure(figure, figsize=(20,10))
    plt.subplot(2,1,1)
    subPlot(xf, lchannelf, xlabel, ylabel, legend, title="Left channel frequency ", xscale = "log")
    plt.subplot(2,1,2)
    subPlot(xf, rchannelf, xlabel, ylabel, legend,  title="Right channel frequency", xscale = "log")

def generateXf(sampleRate, N):
    """generateXf for frequeny domain"""
    return np.linspace(0.0, (N-1)*sampleRate/N, N)

def generateXt(sampleRate, N):
    """generateXt for time domain"""
    return np.linspace(0.0, (N-1)*1/sampleRate, N)

def mag2dB(yf):
    """ change magnitude into dB form """
    return 20*np.log10(yf)

def modifyWindow(w, startFreqency, endFreqency, sampleRate, value):
    """modify the window function into rectangular form"""
    N = len(w)

```

```
    beginPoint = int(startFreqency//(sampleRate/N))
    endPoint = int(endFreqency//(sampleRate/N))

    w[beginPoint:endPoint] = value
    w[-endPoint:-beginPoint] = value

"""main function """

inputWaveAddress = "original.wav"
outputWaveAddress = "improved.wav"
figurePath = "./Output/Figures/"

(rate, lchannel, rchannel) = readWavefile(inputWaveAddress)

N = np.size(lchannel)
T = 1.0/rate
xt = generateXt(rate, N)
xf = generateXf(rate, N)

#plot time domain wave
#wavePlotT(xt, lchannel, rchannel)

"""start fft"""
#caculate fft
lchannelf = np.fft.fft(lchannel)
rchannelf = np.fft.fft(rchannel)

#calculate PSD
PSDlchannelf = np.abs(lchannelf)**2 / N
PSDrchannelf = np.abs(rchannelf)**2 / N

"""task4 refine the record"""
#generate window
w = np.ones(N)
modifyWindow(w, 120, 900, rate, 5)
modifyWindow(w, 6000, 10000, rate, 5)

lchannelfRefine = lchannelf*w
rchannelfRefine = rchannelf*w

lchannelRefine = np.fft.ifft(lchannelfRefine)
rchannelRefine = np.fft.ifft(rchannelfRefine)

"""plot and save all figures"""
wavePlotT("time domain Record", xt, lchannel, rchannel, legend="
    record")
#plt.savefig("./Output/Figures/recordT.pdf")

wavePlotF("frequency domain Record", xf[0:N//2], mag2dB(2/N*np.
```

```
    abs(lchannelf[0:N//2])), mag2dB(2/N*np.abs(rchannelf[0:N//2])
    ), legend="unrefined")
#plt.savefig("./Output/Figures/recordF.pdf")

#plot time domain wave form
wavePlotT("timedomainReference", xt, lchannelRefine.astype(np.
    int16), rchannelRefine.astype(np.int16), legend="refined")
wavePlotT("timedomainReference", xt, lchannel, rchannel, legend=
    "unrefined")
#plt.savefig(figurePath+"recordTR.pdf")

#plot frequency
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")
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")
#plt.savefig(figurePath+"recordFR.pdf")


#plot the window
#plt.figure(figsize=(20,10))
#plt.plot(xf[0:N//2], w[0:N//2])
#plt.xlabel("Frequency(Hz)")
#plt.xscale("log")
#plt.ylabel("Amplitude")
#plt.savefig(figurePath+"window.pdf")

plt.show()

"""export the .wav file"""
writeWavefile(outputWaveAddress, rate, lchannelRefine.astype(np.
    int16), rchannelRefine.astype(np.int16))
```

7.2 touchtonedecoder.py

```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
"""
Created on Mon Oct 19 12:15:25 2020

@author: wayenvan
"""
"""import essential modules"""
import numpy as np
import matplotlib.pyplot as plt
from enum import Enum

```

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

class ToneFlag(Enum):
    low = 0
    high = 1
    NoFind = 2

"""define functions"""
def generateXf(sampleRate, N):
    """generateXf for frequeny domain"""
    return np.linspace(0.0, (N-1)*sampleRate/N, N)

def generateXt(sampleRate, N):
    """generateXt for time domain"""
    return np.linspace(0.0, (N-1)*1/sampleRate, N)

def mag2dB(yf):
    """ change magnitude into dB form """
    return 20*np.log10(yf)

def peakFinding(data):
    """finding the max value of an array"""
    maxIndex = -1
    maxValue = 0

    for i in range(len(data)):
        if(data[i]>maxValue):
            maxIndex = i
            maxValue = data[i]

    return maxIndex

def peakFindingDouble(data):
    """finding the first 2 greatest value of an array"""
    indexMax = peakFinding(data[1:])+1

    indexTemp1 = peakFinding(data[1:indexMax-1])+1
    indexTemp2 = peakFinding(data[indexMax+1:])+indexMax+1

    if(data[indexTemp1]>=data[indexTemp2]):
        indexMaxSec = indexTemp1
    elif(data[indexTemp2]>data[indexTemp1]):
        indexMaxSec = indexTemp2
```

```python

    ret = [indexMaxSec, indexMax]

    return ret

def aliasingFrequency(fs, sampleRate):
    """convert the signal frequency into (0, N/2)"""
    N = int(fs/sampleRate+0.5)
    return abs(fs-N*sampleRate)

def findFrequencyBelong(f, dtmfMin, dtmfMax, sampleRate):
    """
    Parameters
    ---------
    f:
        input frequency of the chunk signal
    dtmfMin:
        acceptable lower bound
    dtmfMax:
        acceptable high bound
    sampleRate:
        sampling frequency

    Returns
    ------
    flag:
        define which tone of the frequency belongs (high or low)
    index:
        return the index of dtmfFrequency array for easy finding of letter

    """

    for indexLow in range(4):
        for indexHigh in range(4):
            if(f-dtmfMin<aliasingFrequency(dtmfHighFrequency[indexHigh], sampleRate)<f+dtmfMax):
                flag = ToneFlag.high
                return flag, indexHigh
            if(f-dtmfMin<aliasingFrequency(dtmfLowFrequency[indexLow], sampleRate)<f+dtmfMax):
                flag = ToneFlag.low
                return flag, indexLow
    return ToneFlag.NoFind, -1

def detectOneDigitFromChunk(data, sampleRate):
    """
    detect each chunk

```

```
    Parameters
    ----------
    data: ndarray
        series of chunk data in time domain
    sampleRate: float
        the sampling frequency of signal

    Returns
    -------
    letter: string
        goal letter of this chunk 'N' means no letter found
    """
    #prepare the data
    dataf = np.fft.fft(data)
    N=len(data)
    minMagnitude = 30
    #cut the data half
    rdataf = dataf[0:N//2]

    dtmfMin = 9
    dtmfMax = 9

    #calculate the peak point
    #ind = np.argpartition(abs(rdataf), -3)[-3:]
    ind = peakFindingDouble(abs(rdataf))

    #cut out small signal
    if((2/N*(abs(rdataf)[ind[0]])<minMagnitude) | (2/N*(abs(rdataf)[ind[1]])<minMagnitude)):
        return 'N'

    f1 = ind[0]*(sampleRate/N)
    f2 = ind[1]*(sampleRate/N)

    #print(f1, f2)
    #start the for loop to check if the frequency meet any of high or low frequency of dtmf
    (flag1, index1) = findFrequencyBelong(f1, dtmfMin, dtmfMax, sampleRate)
    (flag2, index2) = findFrequencyBelong(f2, dtmfMin, dtmfMax, sampleRate)

    #find out corresponding point of this 2 frequency
    if((flag1==ToneFlag.high) & (flag2==ToneFlag.low)):
        return dtmfLetter[index2][index1]
    elif((flag1==ToneFlag.low) & (flag2==ToneFlag.high)):
        return dtmfLetter[index1][index2]
    elif((flag1==ToneFlag.NoFind)|(flag2==ToneFlag.NoFind)):
        #print("index1:", index1, flag1, "index2", index2, flag2
```

```
    )
            return 'N'
    else:
        return 'N'

def autoDetectNumbers(data, sampleRate):
    """

    Parameters
    ----------
    data : ndarray
        touch tone data
    sampleRate : int or float
        sampling frequency

    Returns
    -------
    seriesNumber : String
       the number detected

    """
    K = 0
    N = len(data)
    gap = 300          #the length of eah chunk
    T = 1/sampleRate

    preResult = 'N'
    seriesNumber = ''

    #start checking numbers
    print("start finding raising edge chunk")
    while gap-1+K*gap < N:
        result = detectOneDigitFromChunk(data[K*gap: gap-1+K*gap
    ], sampleRate)
        if((preResult=='N') & (result != 'N')):
            print(K*gap*T,'s', "-", (gap-1+K*gap)*T,'s:',result)
            seriesNumber = seriesNumber + result
        preResult = result
        K = K + 1

    return seriesNumber

"""main function"""
#load .dat file, if change i, it can load all files
figurePath = "./Output/Figures/"

dataAddress = 'touchToneData.dat'
dataI = np.loadtxt(dataAddress, usecols=(1), dtype=np.int16)

```

```
data = dataI
Fs2 = 1000
N2 = len(data)
x2 = range(N2)
xt2 = generateXt(Fs2, N2)
xf2 = generateXf(Fs2, N2)
dataf = np.fft.fft(data)

series = autoDetectNumbers(data, Fs2)
print(dataAddress+":")
print("final result: ", series)

"""plot and save all figures"""
#plot task5 wave
# plt.figure(figsize=(20,10))
# plt.plot(xt2, data)
# plt.xlabel("Time(s)")
# plt.ylabel("Amplitude")
# plt.savefig(figurePath+"DTMFtime.pdf")

# plt.figure(figsize=(20,10))
# plt.plot(xf2[0:N2//2], mag2dB(abs(2/N2*dataf[0:N2//2])))
# plt.title("Frequency domain")
# plt.xlabel("Freqency(Hz)")
# plt.ylabel("Magnitude(dB)")
# plt.savefig(figurePath+"task5ExampleF.pdf")

#plt.show()
```