西北工业大学计算机学院

《多媒体技术》实验报告

学　　号：　2019302863

姓　　名：　李奇

实验时间：　2021.11.18

实验地点：　实验大楼 209-3

实验题目：　Audio feature n coding

西北工业大学

2021 年 11 月

一、 实验目的及要求。

1.了解学习语音处理的一般流程。

2.学会如何计算短时能量、短时过零率和短时自相关函数等语音特征。

2.理解并学会使用 mu-law 变换，能够比较转换前后音频变化。

二、 实验内容与操作步骤。

1.了解语音处理的一般流程，计算一段语音的短时能量、短时过零率和短时自相关函数，将这些特征取值绘图，并观察不同语音单元在这些特征取值上的差异。

2.编写一段程序，利用 mu-law 变换的公式，将 16 位 Linear PCM 格式存储的音频文件转换为 8 位 non-linear PCM 格式，并比较转换前后音频声音质量和文件大小。

三、 实验结果分析

1.了解语音处理的一般流程：

1)录制或者下载一段 wav 格式的语音（最好小一点，大了多硬件要求较高）。

2)预处理，首先找出语音起始点，即端点检测，然后对信号适当的放大和增益控制，消除工频信号干扰。(这里关于预处理和数字化在不同著作上稍有不同，比如某些著作中方其流程是先数字化，放大及增益控制，反混叠滤波，采样、A/D 转换、编码。然后是预处理，提升高频部分，加窗操作，即变换为一帧一帧的语音数据)。

3)数字化，将模拟信号数字化，PCM 编码方式储存。

4)对数字化的信息进行分析，提取特征参数 MCFF。

5)根据不同的目的:

a.语音识别，分为识别和训练阶段。

b.语音编码，将语音进行压缩编码，解压。

c.语音合成，对编码后的信号进行储存。

2.计算语音的短时能量、短时过零率、短时自相关函数并绘图，观察不同语音单元在这些特征取值上的差异。

由于要求使用的音频文件需要是 assignment1 中的录音，且需要保证这段录音采样率 8k、以 16 位 Linear PCM 格式存储。首先使用 sox 工具查看原始音频文件是否满足要求：

```
C:\Users\61060\Desktop\NPU-LQ\大三上\多媒体\LiQi_Audio\work2\音频>sox --i 清音.wav
sox WARN wav: wave header missing extended part of fmt chunk

Input File     : '清音.wav'
Channels       : 2
Sample Rate    : 44100
Precision      : 25-bit
Duration       : 00:00:05.78 = 254880 samples = 433.469 CDDA sectors
File Size      : 2.05M
Bit Rate       : 2.83M
Sample Encoding: 32-bit Floating Point PCM
```

```
C:\Users\61060\Desktop\NPU-LQ\大三上\多媒体\LiQi_Audio\work2\音频>sox --i 元音.wav
sox WARN wav: wave header missing extended part of fmt chunk

Input File     : '元音.wav'
Channels       : 2
Sample Rate    : 44100
Precision      : 25-bit
Duration       : 00:00:08.71 = 384208 samples = 653.415 CDDA sectors
File Size      : 3.08M
Bit Rate       : 2.83M
Sample Encoding: 32-bit Floating Point PCM
```

发现两段语音的采样率为 44100Hz，存储格式为 32-bit Floating Point PCM，不合要求，故对音频格式做出相应修改，修改后的音频信息如下：

```
C:\Users\61060\Desktop\NPU-LQ\大三上\多媒体\LiQi_Audio\work2\音频>sox --i 清音1.wav

Input File     : '清音1.wav'
Channels       : 2
Sample Rate    : 8000
Precision      : 16-bit
Duration       : 00:00:05.78 = 46237 samples ~ 433.472 CDDA sectors
File Size      : 185k
Bit Rate       : 256k
Sample Encoding: 16-bit Signed Integer PCM
```

```
C:\Users\61060\Desktop\NPU-LQ\大三上\多媒体\LiQi_Audio\work2\音频>sox --i 元音1.wav

Input File     : '元音1.wav'
Channels       : 2
Sample Rate    : 8000
Precision      : 16-bit
Duration       : 00:00:08.71 = 69698 samples ~ 653.419 CDDA sectors
File Size      : 279k
Bit Rate       : 256k
Sample Encoding: 16-bit Signed Integer PCM
```

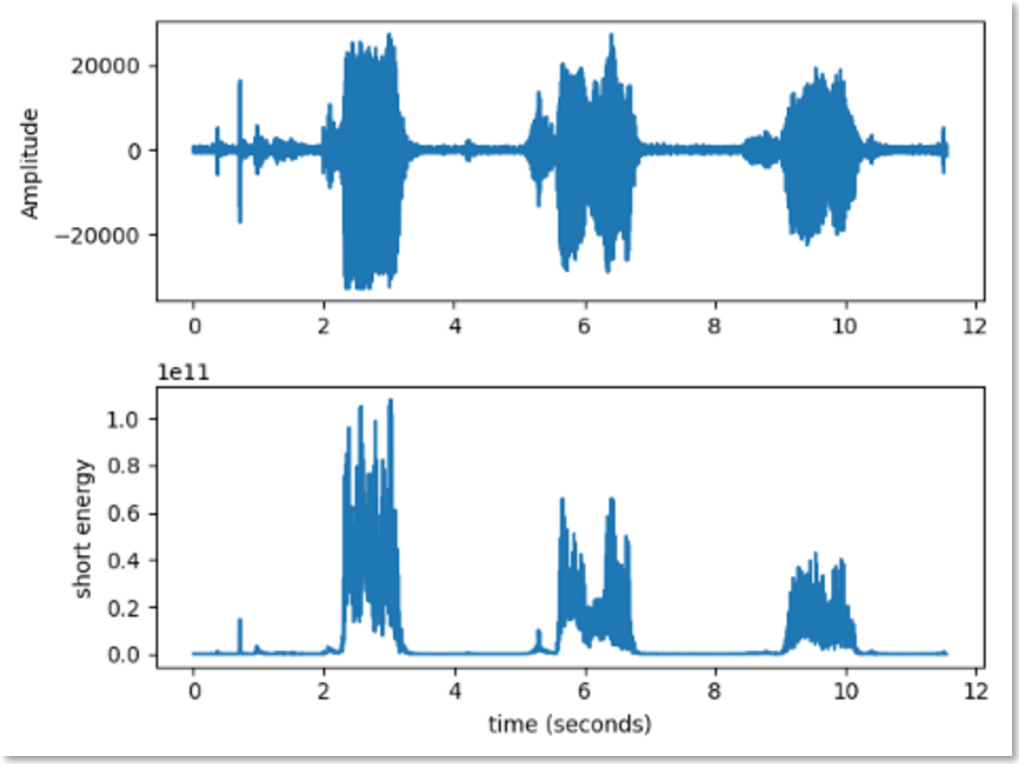
可以看到修改后采样率为 8000Hz，存储格式为 16-bit Signed Integer PCM，都已符合实验要求，下面开始进行相应的实验内容。

短时能量部分的代码为 calEnergy.py，具体如下：

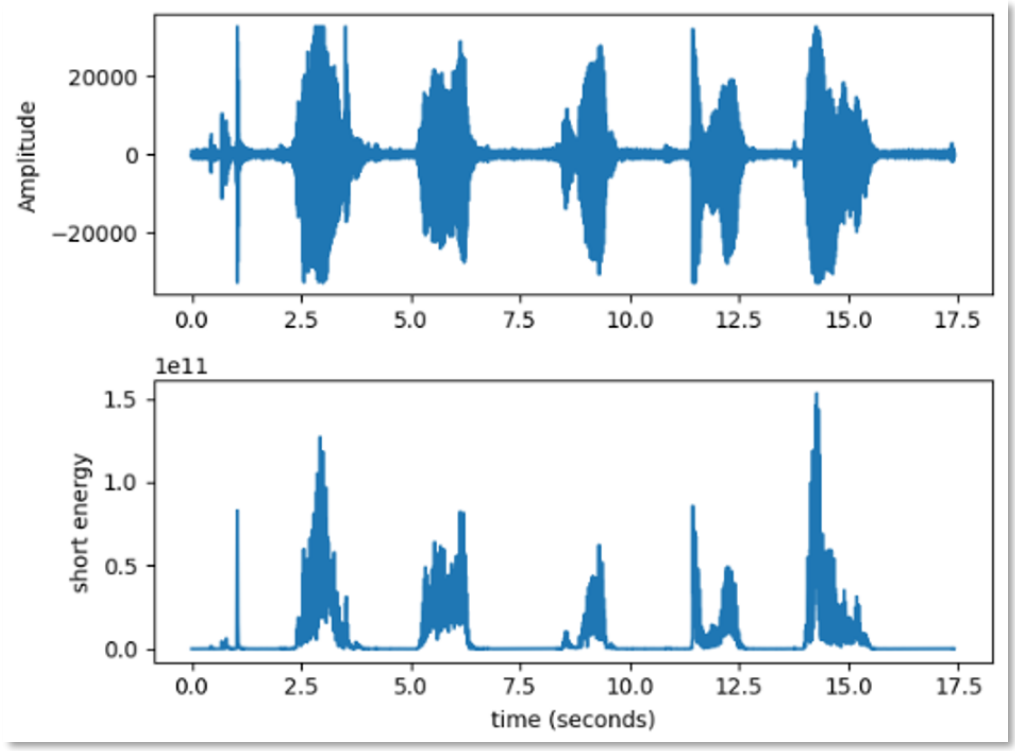
```python
import wave
import numpy as np
import numpy as np
import pylab
import pylab as pl

def calEnergy(wave_data) :
    energy = []
    sum = 0
    for i in range(len(wave_data)) :
        sum = sum + (int(wave_data[i]) * int(wave_data[i]))
        if (i + 1) % 256 == 0 :
            energy.append(sum)
            sum = 0
        elif i == len(wave_data) - 1 :
```

```
            energy.append(sum)
    return energy
f = wave.open(r"C:\\Users\\61060\\Desktop\\NPU-LQ\\大三上\\多媒体
\\LiQi_Audio\\work2\\音频\\清音 1.wav" , "rb")
# getparams() 一次性返回所有的 WAV 文件的格式信息
params = f.getparams()
# nframes 采样点数目
nchannels, sampwidth, framerate, nframes = params[:4]
# readframes() 按照采样点读取数据
str_data = f.readframes(nframes)            # str_data 是二进制字符串

# 以上可以直接写成 str_data = f.readframes(f.getnframes())

# 转成二字节数组形式（每个采样点占两个字节）
wave_data = np.frombuffer(str_data, dtype = np.short)
print( "采样点数目：" + str(len(wave_data)))          #输出应为采样点数目
f.close()
# 计算每一帧的能量 256 个采样点为一帧
energy = calEnergy(wave_data)
print(energy)

time = np.arange(0, len(wave_data)) * (1.0 / framerate)
time2 = np.arange(0, len(energy)) * (len(wave_data) / len(energy) /
framerate)
pl.subplot(211)
pl.plot(time, wave_data)
pl.ylabel("Amplitude")
pl.subplot(212)
pl.plot(time2, energy)
pl.ylabel("short energy")
pl.xlabel("time (seconds)")
pl.show()
# print("短时能量：",energy)
```

对应的生成的清音和元音的图像如下图所示：



清音短时能量图



元音短时能量图

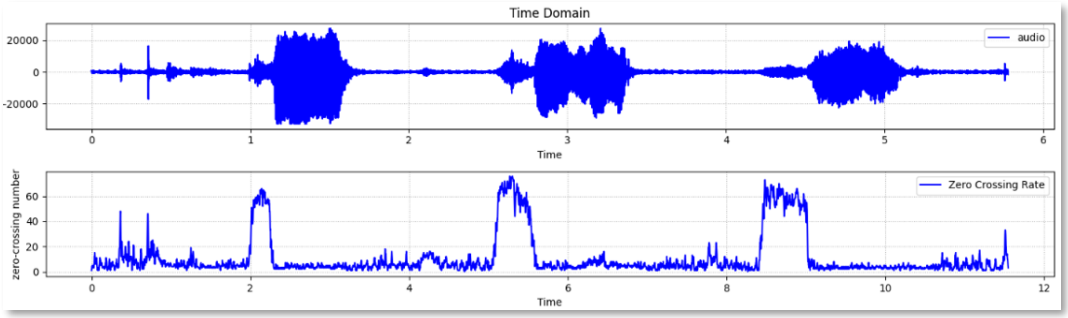
可以看出清音的短时能量最高在 1 左右，而浊音的短时能量最高在 1.5 左右，所以浊音相对清音的短时能量更高。由此可以看出，短时能量能够作为区分清音和浊音的特征参数。并且在信噪比较高时，可以作为区分有声和无声的根据。

短时过零率部分的代码为 zero.py，具体如下：

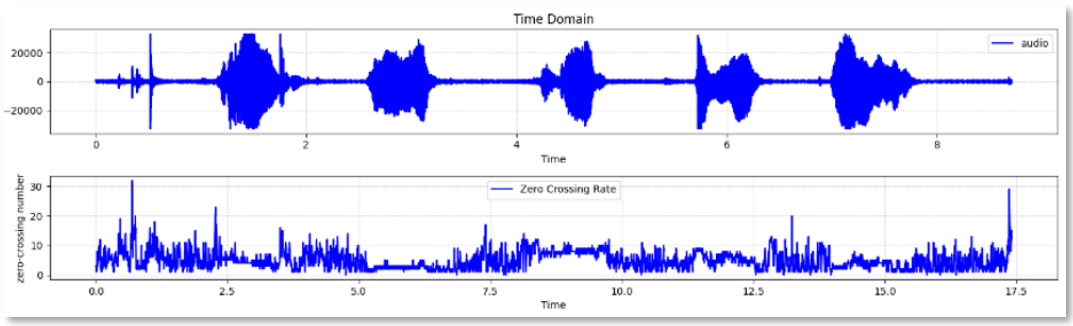
```
import numpy as np
import wave as we
import matplotlib.pyplot as plt
import scipy.io.wavfile as wf

path = 'C:\\Users\\61060\\Desktop\\NPU-LQ\\大三上\\多媒体
\\LiQi_Audio\\work2\\音频'
wlen=480
inc=120
sample_rate, sigs = wf.read(path)
times = np.arange(len(sigs)) / sample_rate
f = we.open(r'元音 1.wav', "rb")
params = f.getparams()
nchannels, sampwidth, framerate, nframes = params[:4]
print(params)
str_data = f.readframes(nframes)
wave_data = np.frombuffer(str_data, dtype=np.short)
wave_data = wave_data*1.0/(max(abs(wave_data)))
#print(wave_data[:10])
signal_length=len(wave_data)
if signal_length<=wlen:
       nf=1
else:
       nf=int(np.ceil((1.0*signal_length-wlen+inc)/inc))
#print(nf)
pad_length=int((nf-1)*inc+wlen)
zeros=np.zeros((pad_length-signal_length,))
pad_signal=np.concatenate((wave_data,zeros))
indices=np.tile(np.arange(0,wlen),(nf,1))+np.tile(np.arange(0,nf*inc,inc),(
wlen,1)).T
#print(indices[:2])
indices=np.array(indices,dtype=np.int32)
frames=pad_signal[indices]
windown=np.hamming(wlen)
d=np.zeros(nf)
x=np.zeros(nf)
c=np.zeros(nf)
e=np.zeros(nf)
for i in range(nf):
    a=frames[i:i+1]
    b=windown*a[0]
    for j in range(wlen-1):
        if b[j]*b[j+1]<0:
            c[i]=c[i]+1
```

```
time = np.arange(0,nf) * (inc*1.0/framerate)
for i in range(0,nf):
        a=frames[i:i+1]
        b = a[0] * windown
        e[i] =np.sum(abs(b))
        c1=np.square(b)
        d[i]=np.sum(c1)
e = e*1.0/(max(abs(e)))
d = d*1.0/(max(abs(d)))
#print(d)

#信号波形
plt.figure(figsize=(15,8))
plt.subplot(4,1,1)
plt.title('Time Domain')
plt.xlabel('Time')
plt.ylabel('Signal')
plt.tick_params(labelsize=10)
plt.grid(linestyle=':')
plt.plot(times, sigs, c='blue', label='Signal')
plt.legend(['audio'])
#短时过零率
plt.subplot(4,1,2)
plt.xlabel('Time')
plt.ylabel('zero-crossing number')
plt.plot(time,c,c='blue')
plt.grid(linestyle=':')
plt.legend(['Zero Crossing Rate'])

plt.rcParams['savefig.dpi'] = 300 #图片像素
plt.rcParams['figure.dpi'] = 300 #分辨率
plt.savefig('image.png')
plt.show()
```

对应的生成的清音和元音的图像如下图所示：



清音短时过零率图



元音短时过零率图

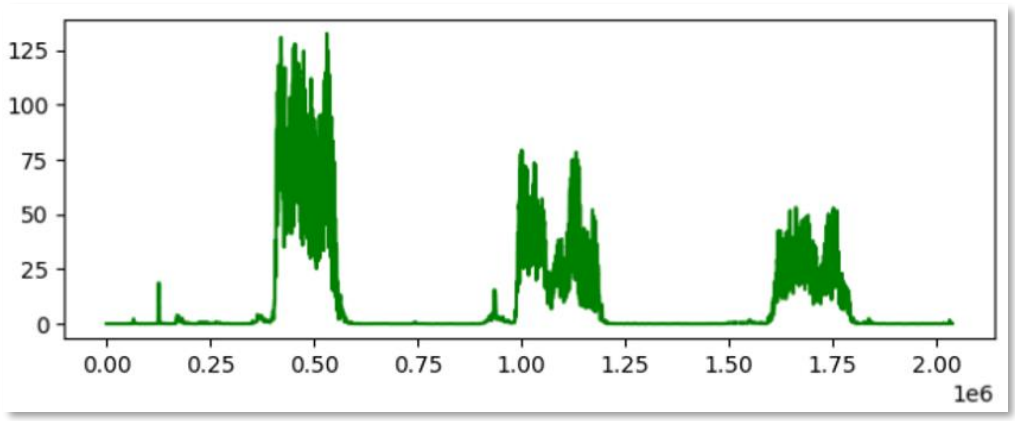
看图分析可知，清音的过零率较高，浊音的过零率较低。这是因为发清音时，声带不震动，能量集中在较高的频率段内，而发浊音时，声带振动，能量集中在较低频率段内。利用短时过零率可以初步判断清音和浊音。

短时自相关函数部分的代码为 self.py，具体如下：

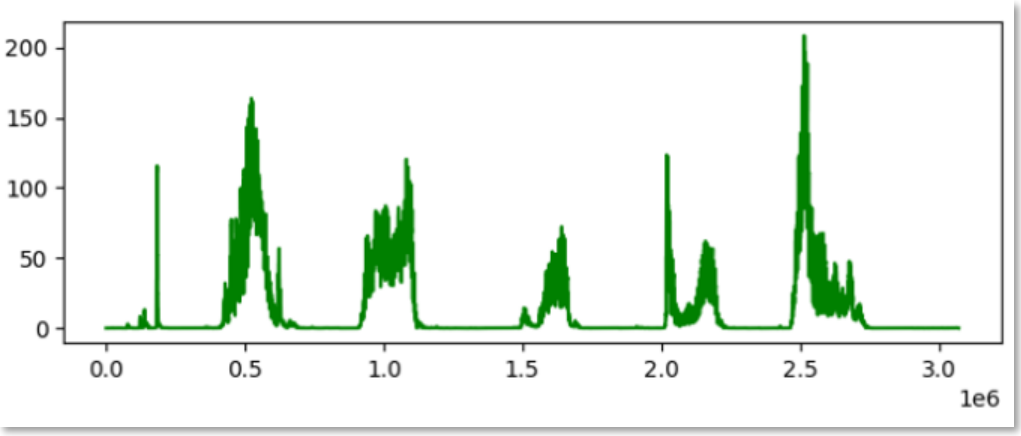
```
import numpy as np
import wave
import matplotlib.pyplot as plt
wlen=512
inc=128
f = wave.open(r"C:\\Users\\61060\\Desktop\\NPU-LQ\\大三上\\多媒体
\\LiQi_Audio\\work2\\音频\\元音 1.wav" , "rb")
params = f.getparams()
nchannels, sampwidth, framerate, nframes = params[:4]
str_data = f.readframes(nframes)
wave_data = np.frombuffer(str_data, dtype=np.short)
wave_data = wave_data*1.0/(max(abs(wave_data)))
print(wave_data[:10])
#信号总长度
signal_length=len(wave_data)
#若信号长度小于一个帧的长度，则帧数定义为 1。否则，计算帧的总长度。
if signal_length<=wlen:
    nf=1
else:
    nf=int(np.ceil((1.0*signal_length-wlen+inc)/inc))
print(nf)
#所有帧加起来总的铺平后的长度
pad_length=int((nf-1)*inc+wlen)
#不够的长度使用 0 填补，类似于 FFT 中的扩充数组操作
zeros=np.zeros((pad_length-signal_length,))
#填补后的信号记为 pad_signal
pad_signal=np.concatenate((wave_data,zeros))
#相当于对所有帧的时间点进行抽取，得到 nf*nw 长度的矩阵
indices=np.tile(np.arange(0,wlen),(nf,1))+np.tile(np.arange(0,nf*inc,inc),(
wlen,1)).T
```

```
#print(indices[:2])
#将 indices 转化为矩阵
indices=np.array(indices,dtype=np.int32)
#得到帧信号
frames=pad_signal[indices]
windown=np.hanning(wlen)
d=np.zeros(nf)
x=np.zeros(nf)
time = np.arange(0,nf) * (inc*1.0/framerate)
frames=frames.T
para = np.zeros(frames.shape)
for i in range(nf):
        R = np.correlate(frames[:, i], frames[:, i], 'valid')
        para[:, i] = R
plt.subplot(2, 1, 2)
para=para.T
para = para.flatten()
plt.plot(para,c="g")
plt.show()
```

对应的生成的清音和元音的图像如下图所示：



清音短时自相关函数图



元音短时自相关函数图

短时自相关函数可以很明显的反映出浊音信号的周期性。分析上图可以看出，清音的短时自相关函数没有周期性，也不具有明显突出的峰值，其性质类似于噪声；浊音信号的自相关函数在基音周期的整数倍位置上出现峰值。由此可以发现，检测自相关函数是否有峰值就可以判断是清音或浊音，峰-峰值之间对应的就是基音周期。

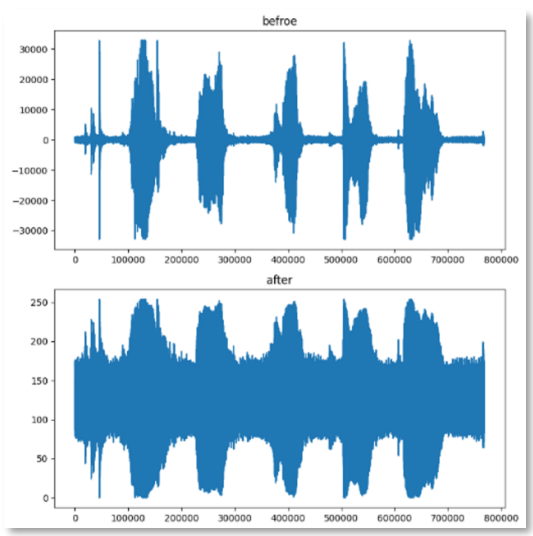
3. 编写一段程序，利用 mu-law 变换的公式，将 16 位 Linear PCM 格式存储的音频文件转换为 8 位 non-linear PCM 格式，并比较转换前后音频声音质量和文件大小。

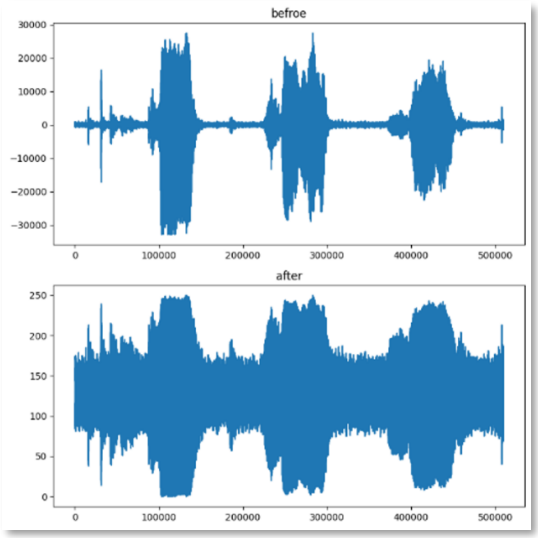
编写的程序命名为 mu_law.py，生成的图像命名为 p2mu_law.png，具体如下所示：

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

def mu_law(x, mu=255):
    x = np.clip(x, -1, 1)
    x_mu = np.sign(x) * np.log(1 + mu*np.abs(x))/np.log(1 + mu)
    return ((x_mu + 1)/2 * mu).astype('int16')
def get_wave_data():
    # 采用只读的方式读取 wav 文件，并将其保存到 f 中
    f = wave.open("C:\\Users\\61060\\Desktop\\NPU-LQ\\大三上\\多媒体
\\LiQi_Audio\\work2\\音频\\元音 1.wav", "rb")
    # 获取所有参数
    params = f.getparams()
    # 以原频率进行采样采样，获取 bytes
    str_data = f.readframes(f.getnframes())
    # 转成二字节数组形式（每个采样点占两个字节）
    wave_data = np.frombuffer(str_data, dtype=np.short)
    f.close()
    ret = [mu_law(x / 2 ** 15) for x in wave_data]
    ret = np.array(ret,dtype=np.uint8)
    write_back(ret)
    plt.figure(figsize=(8,8))
    plt.subplot(211)
    plt.title("befroe")
    plt.plot(wave_data)  #转化前
    plt.subplot(212)
    plt.title("after")
    plt.plot(ret)  #转化后
    plt.savefig("p2mu_law.png")
    plt.show()

def write_back(wave_data):
    f = wave.open("元音 out.wav", "wb")
```

```
    # 设置通道数
    f.setnchannels(1)
    #  设置采样宽度为 8bit
    f.setsampwidth(1)
    # 采样率
    f.setframerate(8000)
    f.writeframes(wave_data)
    f.close()

if __name__ == '__main__':
   get_wave_data()
```

生成的对应的 p2mu_law.png 图像如下图所示：






```
C:\Users\61060\Desktop\NPU-LQ\大三上\多媒体\LiQi_Audio\work2\音频>sox --i 清音1.wav

Input File     : '清音1.wav'
Channels       : 2
Sample Rate    : 8000
Precision      : 16-bit
Duration       : 00:00:05.78 = 46237 samples ~ 433.472 CDDA sectors
File Size      : 185k
Bit Rate       : 256k
Sample Encoding: 16-bit Signed Integer PCM
```

```
C:\Users\61060\Desktop\NPU-LQ\大三上\多媒体\LiQi_Audio\work2\音频>sox --i 清音out.wav

Input File     : '清音out.wav'
Channels       : 1
Sample Rate    : 8000
Precision      : 8-bit
Duration       : 00:01:03.72 = 509760 samples ~ 4779 CDDA sectors
File Size      : 510k
Bit Rate       : 64.0k
Sample Encoding: 8-bit Unsigned Integer PCM
```

清音转换前后对比

```
C:\Users\61060\Desktop\NPU-LQ\大三上\多媒体\LiQi_Audio\work2\音频>sox --i 元音1.wav

Input File     : '元音1.wav'
Channels       : 2
Sample Rate    : 8000
Precision      : 16-bit
Duration       : 00:00:08.71 = 69698 samples ~ 653.419 CDDA sectors
File Size      : 279k
Bit Rate       : 256k
Sample Encoding: 16-bit Signed Integer PCM
```

```
C:\Users\61060\Desktop\NPU-LQ\大三上\多媒体\LiQi_Audio\work2\音频>sox --i 元音out.wav

Input File     : '元音out.wav'
Channels       : 1
Sample Rate    : 8000
Precision      : 8-bit
Duration       : 00:01:36.05 = 768416 samples ~ 7203.9 CDDA sectors
File Size      : 768k
Bit Rate       : 64.0k
Sample Encoding: 8-bit Unsigned Integer PCM
```

元音转换前后对比

4.使用 ffmpeg 获取高质量的 8 位 non-linear PCM 音频：

以清音为例，使用如下 ffmpeg 命令：

ffmpeg -i 清音 1.wav -acodec pcm_mulaw 清音 out_f.wav

```
Input #0, wav, from '清音1.wav':
  Duration: 00:00:05.78, bitrate: 256 kb/s
  Stream #0:0: Audio: pcm_s16le ([1][0][0][0] / 0x0001), 8000 Hz, stereo, s16, 256 kb/s
Stream mapping:
  Stream #0:0 -> #0:0 (pcm_s16le (native) -> pcm_mulaw (native))
Press [q] to stop, [?] for help
Output #0, wav, to '清音out_f.wav':
  Metadata:
    ISFT            : Lavf59.9.102
  Stream #0:0: Audio: pcm_mulaw ([7][0][0][0] / 0x0007), 8000 Hz, stereo, s16, 128 kb/s
    Metadata:
      encoder         : Lavc59.13.101 pcm_mulaw
size=      90kB time=00:00:05.77 bitrate= 128.1kbits/s speed= 155x
video:0kB audio:90kB subtitle:0kB other streams:0kB global headers:0kB muxing overhead: 0.099
487%
```

四、实验中存在问题及改进。