2. 语音信号处理基础实验

2.1 语音采集与读写实验

2.1.1 实验目的

1）了解 Python 采集语音信号的原理及常用命令；

2）熟练掌握基于 Python 的语音文件的创建、读写等基本操作；

3）学会使用 plt.plot 命令来显示语音信号波形，并掌握基本的标注方法。

2.1.2 实验原理

1.语音信号特点

20 世纪 90 年代以来，语音信号采集与分析在实用化方面取得了许多实质性的研究进展。其中，语音识别逐渐由实验室走向实用化。一方面，对声学语音学统计模型的研究逐渐深入，鲁棒的语音识别、给予语音段的建模方法及隐马尔可夫模型与人工神经网络的结合成为研究的热点。另一方面，为了语音识别实用化的需要，讲者自适应、听觉模型、快速搜索识别算法以及进一步的语音模型研究等课题备受关注。

通过对大量语音信号的观察和分析发现，语音信号主要有下面两个特点:

①在频域内，语音信号的频谱分量主要集中在 300-3400Hz 的范围内。利用这个特点，可以用一个防混叠的带通滤波器将此范围内的语音信号频率分出，然后按 8kHz 的采样率对语音信号进行采样，就可以得到离散的语音信号。

②在时域内，语音信号具有“短时性”的特点，即在总体上，语音信号的特征是随着时间而变化的，但在一段较短的时间间隔内，语音信号保持平稳。在浊音段表现出周期信号的特征，在清音段表现出随机噪声的特征。

2.语音信号采集的基本原理

为了将原始模拟语音信号变为数字信号，必须经过采样和量化两个步骤，从而得到时间和幅度上均为离散的数字信号语音。采样是信号在时间上的离散化，即按照一定时间间隔△t 在模拟信号 x（t）上逐点采取其瞬时值。采样时必须要注意满足奈奎斯特定理，即采样频率 fs 必须以高于受测信号的最高频率两倍以上的速度进行取样，才能正确的重建信号。

在 windows 环境下，学生可以使用 windows 自带的录音机录制语音文件，图 2-1 是基于 PC 机的语音信号采集过程，声卡可以完成语音波形的 A/D 转换，获得 WAV 文件。通过 windows 录制的语音信号，一方面可以为后续实验储备原始语音，另一方面可以与通过其它方式录制的语音进行比对，比如使用 Python 的 Pyaudio 库进行录制。

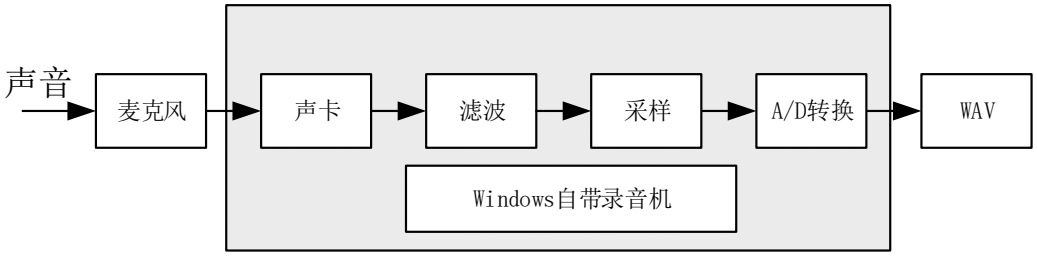

图 2-1 PC 机语音信号采集原理图

3.基于 Python 的语音信号采集与读写方法

Python 将声卡作为对象处理，其后的一切操作都不与硬件直接相关，而是通过对该对象的操作来作用于硬件设备（声卡）。操作时首先要对声卡产生一个模拟输入对象，并给模拟输入对象添加一个通道设置采样频率后，就可以启动设备对象，开始采集数据，采集完成后停止对象并删除对象。

常用的相关 Python 函数包括实例化 Pyaudio 对象、打开声卡、读取数据流以及写入数据流、读取音频文件等函数，下面分别介绍其具体形式及用法：

1）实例化 Pyaudio 对象

功能说明：

在导入 Pyaudio 库后，再创建一个 Pyaudio 对象，以便于后续对声卡进行操作；

函数语法：

pa = pyaudio.PyAudio()

参数解析：

此函数无需参数，对象名可自行设置。

参考例程：

```
import pyaudio                  # 导入 Pyaudio 库
pa = pyaudio.PyAudio()          # 实例化一个 Pyaudio 对象
```

2）打开音频设备函数

功能说明：

打开音频设备，根据所设定的参数创建数据流，以进行录制或播放音频操作。

函数语法：

stream = pyaudio.PyAudio.open(rate, channels, format, input=False, output=False, input_device_index=None, output_device_index=None, frames_per_buffer=1024, start=True, input_host_api_specific_stream_info=None, output_host_api_specific_stream_info=None, stream_callback=None)

参数解析：

rate--采样率。录制音频时需自行给出；播放音频时利用函数 wf.getframerate()将其设定为所播放文件 wf 的采样率。

channels--声道数量。录制音频时需自行给出；播放音频时利用函数 wf.getnchannels()将其设定为所播放文件 wf 的声道数量。

format--采样大小和格式。录制音频时常设置为 pyaudio.paInt16，即将采样深度设置为 16 位；播放音频时利用函数 wf.getsampwidth()从文件 wf 中获取采样深度。

input--指定是否为输入流。录制音频时将此参数设置为 True，若不设置则默认为 False。

output--指定是否为输出流。播放音频时将此参数设置为 True，若不设置则默认为 False。

input_device_index--指定要使用的输入设备的索引。未指定(或设置为 None)使用默认设备。如果参数 input 设置为 False，则忽略该项。

output_device_index--指定要使用的输出设备的索引。未指定(或设置为 None)使用默认设备。如果参数 output 设置为 False，则忽略该项。

frames_per_buffe--指定每一个缓冲区的帧数。若不指定则默认为 1024。

start--指定是否立即启动数据流运行。默认为 True。一般来说，此项通常不会设置为 False。

input_host_api_specific_stream_inf--是否指定主机 API 以输入的特定流信息数据结构。默认为 False。

output_host_api_specific_stream_info--是否指定主机 API 以输出的特定流信息数据结构。默认为 False。

stream_callback--为回调操作指定一个回调函数。默认为 None，即不指定回调函数

参考例程：

```
import pyaudio                                                    # 导入 Pyaudio 库
pa = pyaudio.PyAudio()                                            # 实例化一个 Pyaudio 对象
stream = pa.open( format=pyaudio.paInt16, channels=2, rate=16000, input=True, frames_per_buffer=2048)    # 双通道，16kHz，16 位精度，缓存为 2048 的输入流
```

3）读取数据流函数

功能说明：

从数据流中读取样本，进行流式传输数据以录制音频。

函数语法：

pyaudio.Stream.read(num_frames, exception_on_overflow=True)

参数解析：

num_frame--指定需要读取的帧数。

exception_on_overflow--指定是否需要在输入缓冲区满溢时给出 IOError 的异常信号。默认为 True。

参考例程：

```
import pyaudio                                  # 导入 Pyaudio 库
import wave                                     # 导入 wave 库
CHUNK = 1024                                    # 设定缓存区帧数为 1024
FORMAT = pyaudio.paInt16                        # 设定数据流采样深度为 16 位
CHANNELS = 2                                    # 设置声卡通道为 2
RATE = 44100                                    # 设置采样率
RECORD_SECONDS = 5                              # 设置记录秒数
pa = pyaudio.PyAudio()                          # 实例化一个 Pyaudio 对象
stream = pa.open( format=FORMAT, channels=CHANNELS, rate=RATE, input=True, frames_per_buffer=CHUNK)
print("* recording")                            # 打印开始“录音”标志
frames = []                                     # 创建一个新列表，用于存储采集到的的数据
#开启循环采样直至采集到所需的样本数量
for i in range(0, int(RATE / CHUNK * RECORD_SECONDS)):
    data = stream.read(CHUNK)                   # 从数据流中读取样本
    frames.append(data)                         # 将该样本记录至列表中
print("* done recording")                       # 打印“完成录音”标志
stream.stop_stream()                            # 关闭数据流
stream.close()                                  # 关闭声卡
p.terminate()                                   # 终止 PyAudio
```

4）写入数据流函数

功能说明：

向数据流中写入样本数据，进行流式数据传输以播放音频

函数语法：

pyaudio.Stream.write(frames, num_frames=None, exception_on_underflow=False)

参数解析：

frames--指定数据帧，即需写入的数据。

num_frames--指定需要写入的帧数。默认为 None，此值为 None 时进行自动计算。

exception_on_underflow --指定是否需要在缓冲区下溢时给出 IOError 的异常信号。默认为 False。

参考例程：

```
import pyaudio                                          # 导入 Pyaudio 库
import wave                                             # 导入 wave 库
CHUNK = 1024                                            # 定义数据流块
wf = wave.open(”01.wav”, ‘rb’)                          # 以只读方式打开需要播放的文件
pa = pyaudio.PyAudio()                                  # 实例化一个 Pyaudio 对象
# 使用该对象打开声卡，并从 wf 中获取采样深度
stream = pa.open( format= pa.get_format_from_width(wf.getsampwidth()), channels= wf.getnchannels(), rate= wf.getframerate(), output=True)
data = wf.readframes(CHUNK)                             # 读取数据
while data != ' ':                                      # 开始循环以播放音频
        stream.write(data)                              # 向数据流中写入数据
        data = wf.readframes(CHUNK)                     # 再次读取数据
        print('while 循环中！ ')                        # 设定循环标语
  stream.stop_stream()                                  # 停止数据流
stream.close()                                          # 关闭声卡
pa.terminate()                                          # 终止 PyAudio
```

5）读取音频文件函数

功能说明：

打开一个 WAV 文件，返回采样率(以采样/秒为单位)和来自 WAV 文件的数据。

函数语法：

wavfile.read(filename, mmap=False)

参数解析：

filename --需要打开的 WAV 文件的文件名。

mmap --是可选项，是否以内存映射的方式读取数据，默认为 False。

参考例程：

```
import scipy.io.wavfile as wavfile                    # 导入 wavfile 库
import matplotlib.pyplot as plt                       # 导入 plt 库
(fs , sound) = wavfile.read("01.wav")                 # 读取音频文件的信息
plt.plot(sound)                                       # 画出音频图
plt.show()                                            # 显示图像
```

2.1.3 实验要求

1）编写 Python 程序实现录制语音信号“你好，欢迎”，并保存为 C2_1_y_1.wav 文件，要求采样频率为 16000Hz，采样精度 16bit；

2）使用 wavfile.read 函数读取 C2_1_y_1.wav 文件，并使用 plt.plot 函数显示出来。要求：横轴和纵轴带有标注。横轴的单位为秒，纵轴显示的为归一化后的数值。图 2-2 为参考图例。

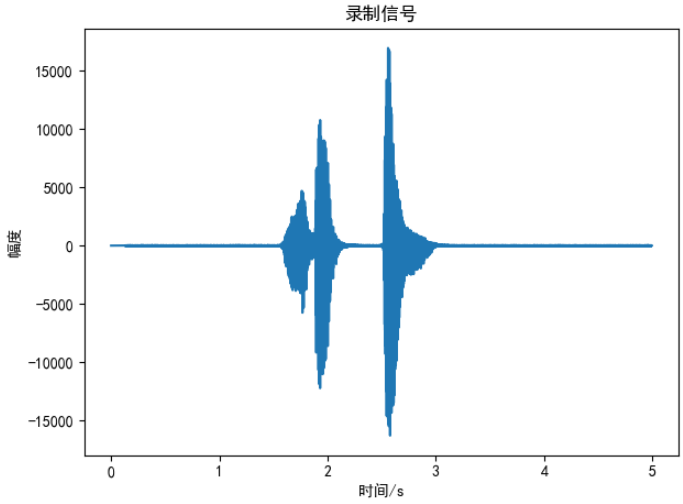

图 2-2 ‘你好，欢迎’时域图

3）使用读取数据流函数播放录制的语音信号，并改变播放的采样频率为原始采样频率的倍数，体验效果。

2.1.4 思考题

1、分析并解释实验要求 3）的现象原理。

2、自行录制一段语音，并存储为 wav 文件。要求：存储为 wav 文件时，分别以采样频率、2 倍采样频率和 1/2 采样频率存为三个 wav 文件，并使用 plt 库中的 plot 函数结合 subplot 函数在一副图上，显示 3 个波形。横轴和纵轴带有标注。横轴的单位为秒，纵轴显示的为归一化后的数值。

2.1.5 参考例程

```
# -*- coding: utf-8 -*
import pyaudio
import wave
import time
from speechlib import *
#  定义数据流参数信息
CHUNK = 1024
FORMAT = pyaudio.paInt16
CHANNELS = 2
RATE = 44100
RECORD_SECONDS = 5
nframes = int(RATE / CHUNK * RECORD_SECONDS)   #计算出所需采集帧的数量
WAVE_OUTPUT_FILENAME = "C2_1_y_1.wav"
#  实例化一个 Pyaudio 对象
p = pyaudio.PyAudio()
#  使用该对象打开声卡，并用上述参数信息对数据流进行赋值
stream = p.open(format=FORMAT,
                channels=CHANNELS,
                rate=RATE,
                input=True,
                frames_per_buffer=CHUNK)
#  开始录音
print("* recording")

frames = []

for i in range(0, nframes):
    data = stream.read(CHUNK)
    frames.append(data)

print("* done recording")
#  关闭数据流，声卡；终止 Pyaudio
```

```
stream.stop_stream()
stream.close()
p.terminate()
# 设定存储录音的 WAV 文件的基本信息
wf = wave.open(WAVE_OUTPUT_FILENAME, 'wb')
wf.setnchannels(CHANNELS)
wf.setsampwidth(p.get_sample_size(FORMAT))
wf.setframerate(RATE)
wf.writeframes(b''.join(frames))
wf.close()

# 只读模式打开需要播放的文件
wf = wave.open(WAVE_OUTPUT_FILENAME, 'rb')

# 实例化一个 pyaudio 对象
p = pyaudio.PyAudio()

# 定义回调函数
def callback(in_data, frame_count, time_info, status):
    data = wf.readframes(frame_count)
    return (data, pyaudio.paContinue)

# 以回调函数的形式打开声卡，创建数据流
stream = p.open(format=p.get_format_from_width(wf.getsampwidth()),
                channels=wf.getnchannels(),
                rate=wf.getframerate(),
                output=True,
                stream_callback=callback)

# 打开数据流
stream.start_stream()

# 等待数据流停止
while stream.is_active():
    time.sleep(0.1)

# 停止数据流，关闭声卡、音频文件；终止 Pyaudio
stream.stop_stream()
stream.close()
wf.close()
p.terminate()

(fs , sound) = wavfile.read(WAVE_OUTPUT_FILENAME)
t = np.array([i/fs for i in range(sound.size)])
```

```
plt.plot(t,sound)
plt.title('录制信号')
plt.xlabel('时间/s')
plt.ylabel('幅度')
plt.show()
```

2.2 语音编辑实验

2.2.1 实验目的

1）掌握语音信号线性叠加的方法，编写 Python 程序实现非等长语音信号的叠加；
2）熟悉语音信号卷积原理，编写 Python 程序实现两语音卷积；
3）熟悉语音信号升采样/降采样方法，并编写 Python 程序实现。

2.2.2 实验原理

1.信号的叠加

两个信号 x_1 和 x_2，通过信号的补零操作使两语音信号有相同的长度，叠加信号为 $x_{new} = x_1 + x_2$。

实验中常通过生成随机信号的方法来叠加白噪声，Python 中的 random 模块和 numpy 模块都可以用于生成随机数，下面介绍一些常用的函数。

功能说明：

用于产生正态分布的随机信号

函数语法：

Y = random.random()

Y = random.uniform(a, b)

Y = random.randint(a, b)

Y = random.randrange(start, stop, step)

Y = np.random.rand(d0,d1,…,dn)

Y = np.random.randn(d0,d1,…,dn)

Y = np.random.randint(low[, high, size])

参数解析：

Y = random.random()--返回一个伪随机数，其值在0.0至1.0之间。

Y = random.uniform(a, b)--用于生成指定范围内的随机浮点数。参数a，b分别表示下界和上界。

Y = random.randint(a, b)--用于生成指定范围内的整数。生成的伪随机数满足 $a \le Y \le b$。

Y = random.randrange(start, stop, step)--从指定的范围内，按指定步长递增的集合中获取一个随机数。

Y = np.random.rand(d0,d1,…,dn)--返回随机n+1维矩阵，元素是从[0,1)的随机采样。

Y = np.random.randn(d0,d1,…,dn)--返回随机n+1维矩阵，元素是从标准正态分布中取的样本值。

Y = np.random.randint(low[, high, size])—返回随机的整数。元素位于左闭右开区间[low, high)中，参数size表示返回值形状，如size=10返回10个元素的向量，size=(2,4)返回2行4列的矩阵。

2.信号的卷积

两序列 x_1 和 x_2 的卷积 y 定义为：

$$y(n)=\sum_{k=-\infty}^{\infty}x_1(k)x_2(n-k)=x_1(n)*x_2(n) \tag{2-1}$$

卷积运算满足交换律，即：

$$y(n)=\sum_{k=-\infty}^{\infty}x_2(k)x_1(n-k)=x_2(n)*x_1(n) \tag{2-2}$$

注意：产生的序列 y（n）长度为 x_1 和 x_2 长度之和减 1。

Python 中有多个模块都可以实现卷积运算，下面给出了 numpy 模块和 scipy 的 signal 模块进行卷积的示例

功能说明：

表示卷积和多项式乘法

函数语法：

w = numpy.convolve(a, v, mode='full')

w = scipy.signal.convolve(x,h)

参数解析：

w = numpy.convolve(a, v, mode='full')--实现两一维信号的卷积。

a--表示长度为N的一维数组；

v--表示长度为M的一维数组；

mode--有三项可选'full','valid','same'。其中mode='full'是默认值，表示返回每一个卷积值，数组长度为N+M-1；mode='valid'返回的数组长度为max(M,N)-min(M,N)+1；mode='same'返回的数组长度为max(M, N)。

w = scipy.signal.convolve(x,h)--实现两一维信号的卷积，x和h分别表示数组类型的待卷积的一维信号。

3.信号采样频率的变换

采样率变换是多采样率信号处理的基础，主要由两个主要操作组成：抽取和内插。下面将分别介绍两种操作。

抽取就是把原采样序列 $x(n)$ 每隔 $D-1$ 点取一个值，形成一个新的序列，有：

$$x_D(m)=x(mD) \tag{2-3}$$

其中，D为正整数。为了避免抽取序列频谱的混叠，通常需要在抽取前将信号通过一个抗混叠滤波器。

内插器和抽取器作用相反，它在两个原始序列的样点之间插入 $I-1$ 个值。设原始序列为 $x(\mathrm{n})$，则内插后的序列 $x_I(m)$ 为：

$$\mathrm{X}(\mathrm{m})=\begin{cases}\mathrm{X}\left(\dfrac{\mathrm{m}}{\mathrm{I}}\right), \mathrm{m}=0,\pm\mathrm{I},\pm2\mathrm{I}\cdots \\ 0 \quad , \quad others\end{cases} \tag{2-4}$$

内插之后还要通过低通滤波器，抑制混叠信号。

Python中scipy库的signal模块resample函数能实现采样率的变换。

功能说明：

沿给定轴使用傅里叶方法对信号x进行重采样。

函数语法：

y = signal.resample(x, num, t=None, axis=0, window=None, domain='time')

y = signal.resample_poly(x,up,down)

参数解析：

y = signal.resample(x, num, t=None, axis=0, window=None, domain='time')

- x--需重新采样的数据。
- num--重采样信号中的采样点数。
- t--可选项。如果给出t，则被假设为与x中的信号数据相关联的等间隔样本位置，默认为None。
- axis--可选项。重新采样的x轴，默认为0。
- window--可选项。指定在傅里叶变换中应用于信号的窗口形状，默认为None。
- domain--可选项。指示输入x的字符串是哪个域的。默认为’time’，即输入的x是时域的；若domain=’freq’则将输入的x看作是频域的。

y = signal.resample_poly(x,up,down)--类似于MATLAB中的resample函数。x表示需重新采样的数据，参数up表示目标采样率，down表示当前采样率。

参考例程：

%对简单的线性序列进行为原采样率3/2倍的重采样

```
from scipy import signal                    # 从scipy库中导入signal模块
x = [i for i in range(1,11)]                # 构建[1,10]的线性序列
y = signal.resample_poly(x,3,2)             # 采样率为原来3/2倍的重采样
```

2.2.3 实验要求

1）录制或从 wav 文件中读取一段语音，并归一化。然后生成一段随机信号（长度与语音信号相同），归一化后幅度乘以 0.01。最后线性叠加两段语音，并用 plt.plot 函数显示三种

信号。要求：横轴和纵轴带有标注。横轴的单位为秒，纵轴显示的为归一化后的数值。图 2-3 为参考图例。

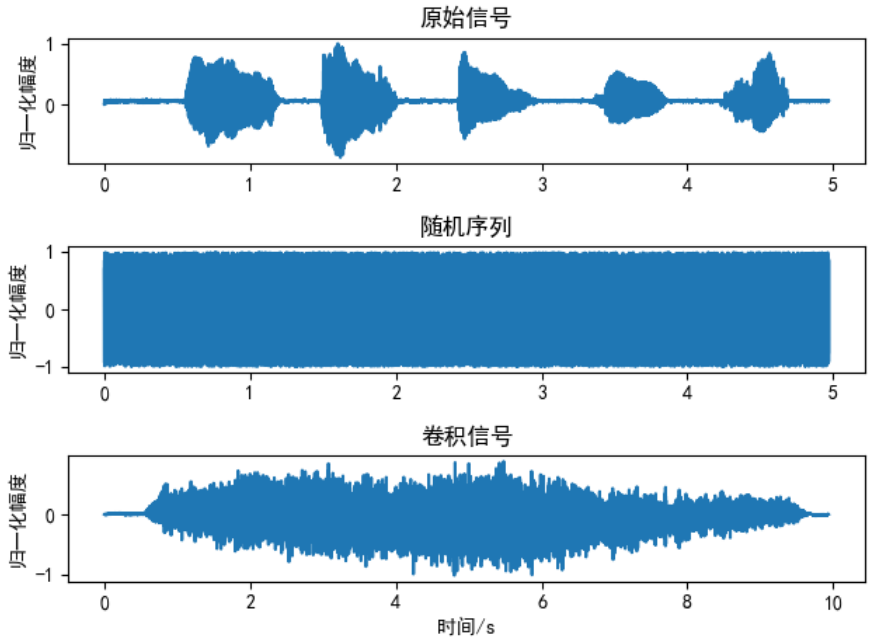

图 2-3 语音叠加示例

2）将录制或读取的语音信号与随机信号进行卷积，并用 plt.plot 函数显示该信号，并对比线性叠加信号的区别。然后播放两种信号，并比较区别。图 2-4 为参考图例。

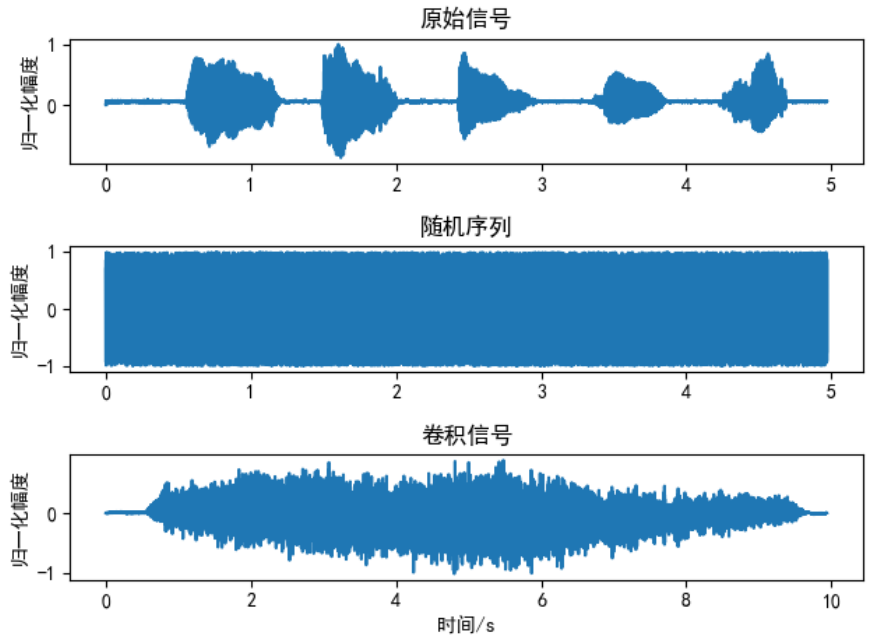

图 2-4 信号卷积示例

3）改变录制或读取的语音信号的采样频率，使用 plt.plot 函数进行显示，如图 2-5 所示。然后播放，比较采样频率改变对语音信号的影响。

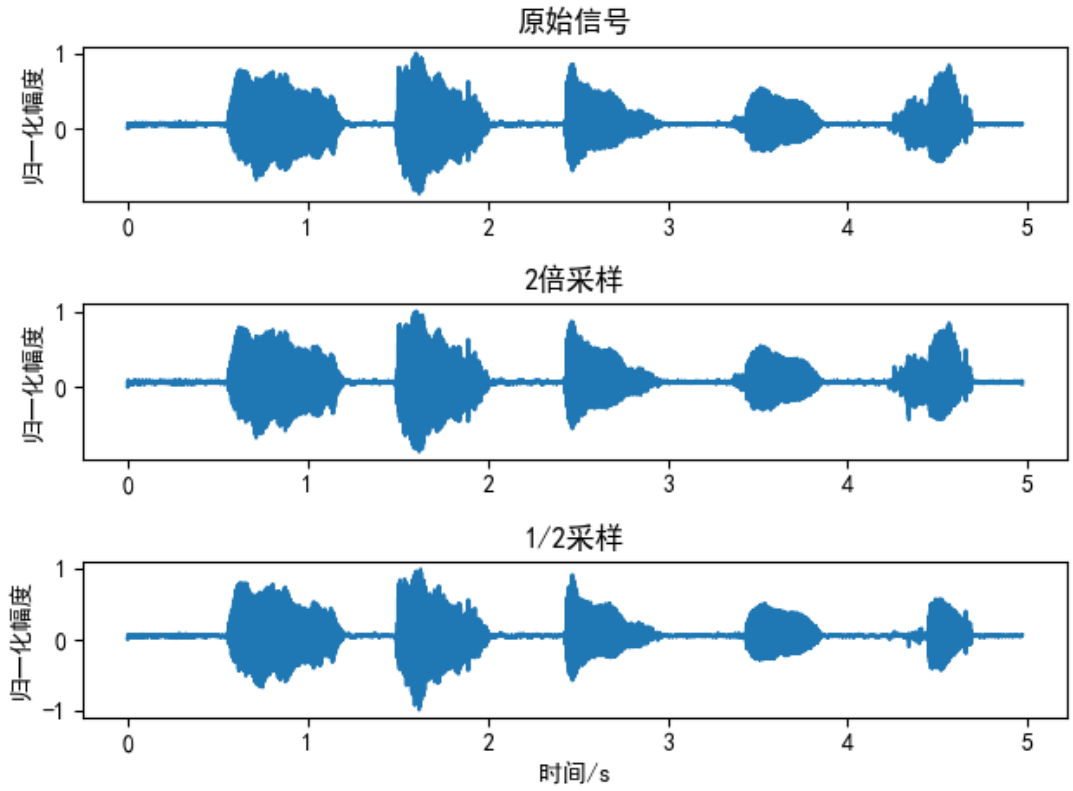

图 2-5 采样频率改变示例

2.2.4 思考题

1、编写 Python 函数实现任意长度的两个信号的线性叠加，生成信号的长度为两个叠加信号的最大长度。

2、改变实验要求 1）的随机信号幅度，重复实验步骤 1）和实验步骤 2）并观察叠加信号的波形，感知语音质量，总结规律。

3、自行编写卷积函数，并编程同 Python 函数进行比较。

2.2.5 参考例程

```
from speechlib import *
#读取之前录制好的语音
(fs, sound) = wavfile.read("C2_2_y.wav")
#  构建横坐标时间 t
t = np.array([i/fs for i in range(sound.size)])
#  语音信号归一化
sound_max = np.absolute(sound).max()
sound = sound / sound_max
#  画原始语音信号的图
plt.figure(1)
```

```
plt.subplot(311)
plt.plot(t,sound)
plt.title('原始信号')
plt.ylabel('归一化幅度')
# 将采样点变为原来的两倍
sound2 = signal.resample(sound,2*sound.size)
# 两倍采样率信号的归一化
sound2_max = np.absolute(sound2).max()
sound2 = sound2 / sound2_max
# 构建横坐标时间 t2
f2 = 2 * fs
t2 = np.array([i/f2 for i in range(sound2.size)])
# 画 2 倍采样率信号的图
plt.subplot(312)
plt.plot(t2,sound2)
plt.title('2 倍采样')
plt.ylabel('归一化幅度')
# 将采样点变为原来的 1/2
sound3 = signal.resample(sound,int(sound.size/2))
# 1/2 被采样率信号的归一化
sound3_max = np.absolute(sound3).max()
sound3 = sound3 / sound3_max
# 构建横坐标时间 t3
f3 = fs / 2
t3 = np.array([i/f3 for i in range(sound3.size)])
# 画 1/2 倍采样率信号的图
plt.subplot(313)
plt.plot(t3,sound3)
plt.title('1/2 采样')
plt.ylabel('归一化幅度')
plt.xlabel('时间/s')
plt.show()
```

2.3 语音信号生成的数学模型

2.3.1 实验目的

1）了解语音信号产生的基本原理；
2）熟悉常见的语音信号产生模型；
3）能编程实现语音信号产生模型，并分析其时频特性。

2.3.2 实验原理

从人类的发音器官的机理来看，发不同性质的声音时，声道的情况是不同的。此外，声门和声道的相互耦合会形成语音信号的非线性特性。但语音信号特性随着时间变化是很缓慢的，所以可以作出一些合理的假设，将语音信号分为一些相继的短段进行处理，在这些短段中可以认为语音信号特性是不随着时间变化的平稳随机过程。通过对发音器官和语音产生机理的分析，语音生成系统理论上分成三个部分，在声门（声带）以下，称为“声门子系统”，它负责产生激励振动，是“激励系统”；从声门到嘴唇的呼气通道是声道，是“声道系统”；语音从嘴唇辐射出去，因此嘴唇以外部分称为“辐射系统”。

1 激励模型

激励模型一般分成浊音激励和清音激励来讨论。发浊音时，由于声带不断张开和关闭，将产生间歇的脉冲波。这个脉冲波的波形类似于斜三角形的脉冲，如图 2-6(a)所示。它的数学表达式如下：

$$g(n)=\begin{cases}(1/2)[1-\cos(\pi n/N_1)], & 0\le n\le N_1\\ \cos[\pi(n-N_1)/2N_2], & N_1\le n\le N_1+N_2\\ 0, & \text{其他}\end{cases} \tag{2-5}$$

式中，N_1为斜三角波上升部分的时间，N_2为其下降部分的时间。单个斜三角波波形的频谱$G(e^{jw})$的图形如图 2-6(b)所示。由图可见，这是一个低通滤波器，其z变换的全极模型为：

$$G(z)=\frac{1}{(1-e^{-cT}z^{-1})^2} \tag{2-6}$$

这里，c是一个频率常数。由于$G(z)$是一个二极点的模型，因此斜三角波形可视为加权的单位脉冲串激励上述单个斜三角波模型的结果。而单位脉冲串及幅值因子则可表示成下面的z变换形式：

$$E(z)=\frac{A_v}{1-z^{-1}} \tag{2-7}$$

所以，整个浊音激励模型可表示为：

$$U(z)=G(z)E(z)=\frac{A_v}{1-z^{-1}}\cdot\frac{1}{(1-e^{-cT}z^{-1})^2} \tag{2-8}$$

即浊音激励波是一个以基音为周期的斜三角脉冲串。

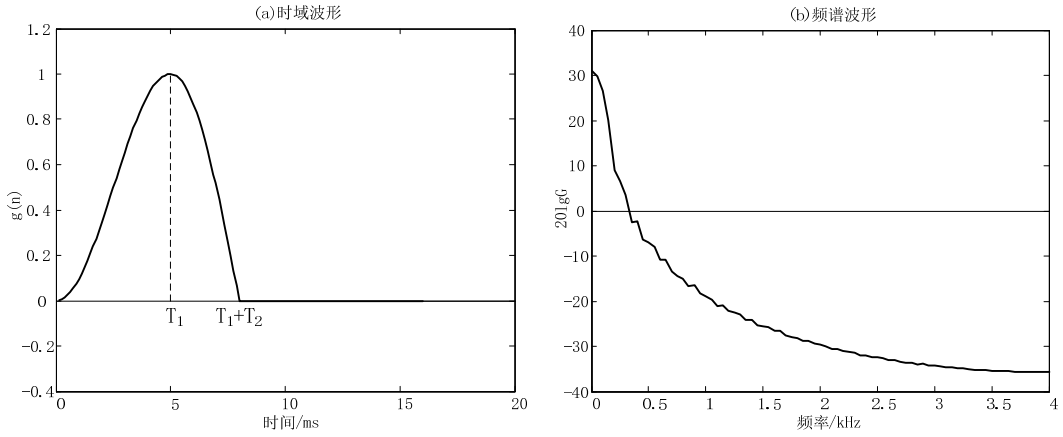

图 2-6 激励模型响应

发清音时，无论是发阻塞音或摩擦音，声道都被阻碍形成湍流。所以，可把清音激励模拟成随机白噪声。实际情况一般使用均值为 0 的、方差为 1 的，并在时间或/和幅值上为白色分布的序列。

2 声道模型

共振峰模型把声道视为一个谐振腔，共振峰就是这个腔体的谐振频率。由于人耳听觉的柯替氏器官的纤毛细胞是按频率感受而排列其位置的，所以这种共振峰的声道模型方法是非常有效的。一般来说，一个元音用前三个共振峰来表示就足够了；而对于较复杂的辅音或鼻音，大概要用到前五个以上的共振峰才行。

从物理声学观点，可以很容易推导出均匀断面的声管的共振频率。一般成人的声道约为 17cm 长，因此算出其开口时的共振频率为：

$$F_i=\frac{(2i-1)c}{4L} \tag{2-9}$$

这里，$i=1,2,\cdots$为正整数，表示共振峰的序号，c为声速，L为声管长度。由此可算的元音 e[∂]的前三共振峰为 $F_1=500Hz$， $F_2=1500Hz$， $F_3=2500Hz$。另外，除了共振峰频率之外，共振峰模型还包括共振峰带宽和幅度等参数。

对于级联型的共振峰模型来说，声道可以视为一组串联的二阶谐振器。从共振峰理论来看，整个声道具有多个谐振频率和反谐振频率，所以可被模拟为一个包含零极点的数学模型；但对于一般元音，则用全极点模型即可，其传输函数可表示为：

$$V(z)=\frac{G}{1-\sum_{k=1}^{N}a_k z^{-k}} \tag{2-10}$$

式中，N 是极点个数，G 是幅值因子，a_k 是常系数。此时可将它分解为多个二阶极点的网络的串联，即：

$$V(z)=\prod_{i=1}^{M}\frac{a_i}{1-b_i z^{-1}-c_i z^{-2}} \tag{2-11}$$

式中，

$$\begin{cases} c_i=-\exp(-2\pi B_i T) \\ b_i=2\exp(-\pi B_i T)\cos(2\pi F_i T) \\ a_i=1-b_i-c_i, \\ G=a_1\cdot a_2\cdot a_3\cdots\cdots a_M \end{cases} \tag{2-12}$$

并且 M 是小于 $(N+1)/2$ 的整数。若 z_k 是第 k 个极点，则有 $z_k=e^{-B_k T}e^{-2\pi F_k T}$，$T$ 是取样周期。根据公式（2-11），则第一共振峰的二阶谐振器的幅频特性如图 2-7 所示。

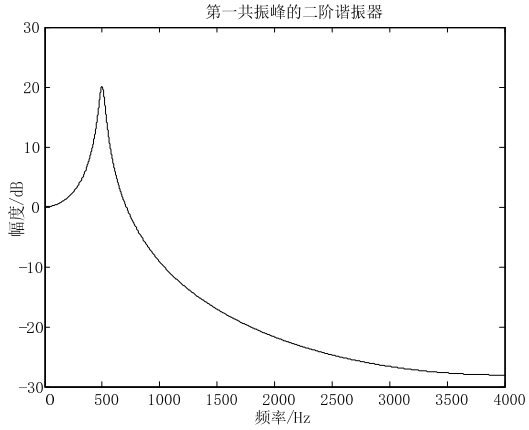

图 2-7 二阶谐振器

3 辐射模型

从声道模型输出的是速度波 $u_L(n)$，而语音信号是声压波 $p_L(n)$，二者之倒比称为辐射阻抗 Z_L。其即表征口唇的辐射效应，又包括圆形的头部的绕射效应等。从理论上推导这个阻抗是有困难的，但是如果认为口唇张开的面积远小于头部的表面积，则可近似地看成平板开槽辐射的情况。此时，辐射阻抗的公式如下：

$$Z_L(\Omega)=\frac{j\Omega L_r R_r}{R_r+j\Omega L_r} \tag{2-13}$$

式中，$R_r=\frac{128}{9\pi^2}$，$L_r=\frac{8a}{3\pi c}$。这里，a 是口唇张开时的开口半径，c 是声波传播速度。

由辐射引起的能量损耗正比于辐射阻抗的实部，所以辐射模型是一阶类高通滤波器。不考虑冲激脉冲串模型 $E(z)$，斜三角波模型是二阶低通，而辐射模型是一阶高通。因此，在实际信号分析时，常用所谓“预加重技术”，即在取样之后，插入一个一阶的高通滤波器，从而只剩下声道部分，便于声道参数分析。在语音合成时再进行“去加重”处理，就可以恢复原来的语音。常用的预加重因子为 $[1-(R(1)z^{-1}/R(0))]$。这里，$R(n)$ 是信号 $s(n)$ 的自相关函数。通常对于浊音来说，$R(1)/R(0)\approx 1$；而对于清音来说，该值可取得很小。

4 语音信号的数字模型

综上所述，完整的语音信号数字模型可用激励模型、声道模型和辐射模型的串联来表示，如图 2-8 所示。传输函数 $H(z)$ 可表示为：

$$H(z)=A\cdot U(z)V(z)R(z) \tag{2-14}$$

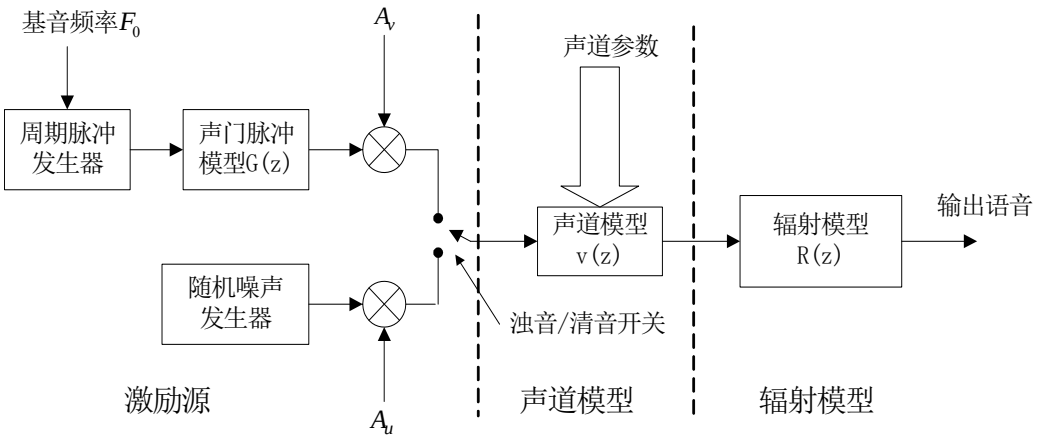

图 2-8 语音信号产生的数字模型

这里，$U(z)$ 是激励信号，浊音时 $U(z)$ 是声门脉冲即斜三角形脉冲序列的 z 变换；在清音的情况下，$U(z)$ 是一个随机噪声的 z 变换。$V(z)$ 是声道传输函数，既可用声管模型，也可以共振峰模型等来描述。实际上就是全极点模型：

$$V(z)=\frac{1}{1-\sum_{k=1}^{N}a_k z^{-k}} \tag{2-15}$$

而 $R(z)$ 可表示一阶高通的形式：

$$R(z)=R_0(1-z^{-1}) \tag{2-16}$$

应该指出，式（2-10）所示模型的内部结构并不和语音产生的物理过程相一致，但这种模型和真实模型在输出处是等效的。1）这种模型是“短时”的模型，因为一些语音信号的变化是缓慢的，例如元音在 10-20ms 内其参数可假定不变。而声道转移函数 $V(z)$ 是一个参数随时间缓慢变化的模型；2）这一模型认为语音是声门激励源激励线性系统---声道所产生的；

实际上，声带---声道相互作用的非线性特征还有待研究；3）模型中用浊音和清音进行简单划分的方法是有缺陷的，对于某些音是不适用的，例如浊音当中的摩擦音。这种音要有发浊音和发清音的两种激励，而且两者不是简单的叠加关系。对于这些音可用一些修正模型或更精确的模型来模拟。

2.3.3 实验要求

1）编写激励模型函数。参考公式（2-5）至公式（2-8），编写激励模型函数，实现类似图 2-6 的时域和频域曲线。

2）编写共振峰模型函数。根据实验原理介绍的共振峰频率组（$F_1 = 500Hz$，$F_2 = 1500Hz$，$F_3 = 2500Hz$），参考级联型共振峰模型，编写类似图 2-7 的二阶谐振器曲线。

2.3.4 思考题

已知英文字母'a'的前三共振峰频率分别为 730Hz，1090Hz，2440Hz，试根据图 2-8 编写程序实现字母'a'的发音。

2.3.5 参考例程

```
from speechlib import *
import math

f1 = 500
f2 = 1500
f3 = 2500
rate = 8000
pitch = 100

#冲激函数
yt = np.zeros(8000)
yt[0] = 1

#  根据指定的共振峰频率和带宽(50Hz)建模语音信号中的共振峰
#  第一共振峰
if(f1 > 0):
    cft = f1 / rate
    bw = 50                                  #  指定带宽为 50Hz
    q = f1 / bw
    rho = math.exp(-math.pi * cft / q)       #  表示式（2-12）中的 exp(−πB_iT)
```

```
        theta = 2 * math.pi * cft * math.sqrt(1 - 1/(4*q*q))           # 表示式（2-12）中的2πF_iT
        a2 = -2 * rho * math.cos(theta)          # 表示式（2-12）中的-b_i
        a3 = rho * rho                           # 表示式（2-12）中的-c_i
        y = signal.lfilter([1+a2+a3],[1,a2,a3],yt)          # 以式（2-11）的滤波器形式对冲激函数进行滤波
plt.figure(1)
N = y.size
fn = np.array([i*rate/N for i in range(N)])
fftg = np.fft.fft(y)
disg = 20 * np.log10(np.absolute(fftg))
plt.plot(fn[:int(N/2)+1],disg[:int(N/2)+1])
plt.xlabel('频率/Hz')
plt.ylabel('幅度/dB')
plt.title('二阶谐振器的第一共振峰')
plt.show()
# 第二共振峰
if(f2 > 0):
        cft = f2 / rate
        bw = 50
        q = f2 / bw
        rho = math.exp(-math.pi * cft / q)
        theta = 2 * math.pi * cft * math.sqrt(1 - 1/(4*q*q))
        a2 = -2 * rho * math.cos(theta)
        a3 = rho * rho
        y = signal.lfilter([1+a2+a3],[1,a2,a3],y)

plt.figure(2)
N = y.size
fn = np.array([i*rate/N for i in range(N)])
fftg = np.fft.fft(y)
disg = 20 * np.log10(np.absolute(fftg))
plt.plot(fn[:int(N/2)+1],disg[:int(N/2)+1])
plt.xlabel('频率/Hz')
plt.ylabel('幅度/dB')
plt.title('二阶谐振器的第二共振峰')
plt.show()
# 第三共振峰
if(f3 > 0):
        cft = f3 / rate
        bw = 50
        q = f3 / bw
```

```
        rho = math.exp(-math.pi * cft / q)
        theta = 2 * math.pi * cft * math.sqrt(1 - 1/(4*q*q))
        a2 = -2 * rho * math.cos(theta)
        a3 = rho * rho
        y = signal.lfilter([1+a2+a3],[1,a2,a3],y)

plt.figure(3)
N = y.size
fn = np.array([i*rate/N for i in range(N)])
fftg = np.fft.fft(y)
disg = 20 * np.log10(np.absolute(fftg))
plt.plot(fn[:int(N/2)+1],disg[:int(N/2)+1])
plt.xlabel('频率/Hz')
plt.ylabel('幅度/dB')
plt.title('二阶谐振器的第三共振峰')
plt.show()
```

3. 语音信号分析实验

3.1 语音信号的预处理

3.1.1 实验目的

1）了解语音信号分帧与加窗的重要性和基本原理；
2）能编程实现分帧函数，并恢复。
3）掌握消除趋势项、直流项的方法；
4）掌握语音信号预加重的原理及方法；
5）掌握语音信号预滤波的目的和方法。

3.1.2 实验原理

1、语音分帧

贯穿于语音分析全过程的是“短时分析技术”。因为，语音信号从整体来看其特性及表征其本质特征的参数均是随时间而变化的，所以它是一个非平稳态过程，不能用处理平稳信号的数字信号处理技术对其进行分析处理。但是，由于不同的语音是由人的口腔肌肉运动构成声道某种形状而产生的响应，而这种口腔肌肉运动相对于语音频率来说是非常缓慢的，所以从另一方面看，虽然语音信号具有时变特性，但是在一个短时间范围内（一般认为在10ms～30ms 的短时间内），其特性基本保持不变即相对稳定，因而可以将其看作是一个准稳态过程，即语音信号具有短时平稳性。所以任何语音信号的分析和处理必须建立在“短时”的基础上，即进行“短时分析”，将语音信号分为一段一段来分析其特征参数，其中每一段称为一“帧”，帧长一般即取为 10~30ms。这样，对于整体的语音信号来讲，分析出的是由每一帧特征参数组成的特征参数时间序列。

对于语音信号处理来说，一般每秒约取 33~100 帧，视实际情况而定。分帧虽然可以采用连续分段的方法，但一般采用如图 3-1 所示的交叠分段的方法，这是为了保证帧与帧之间平滑过渡，保持其连续性。前一帧和后一帧的交叠部分称为帧叠。相邻两帧的起始位置差称为帧移。帧移与帧长的比值一般取为 0～1/2。分帧是用可移动的有限长度窗口进行加权的方法来实现的，即用一定的窗函数来乘以语音信号。

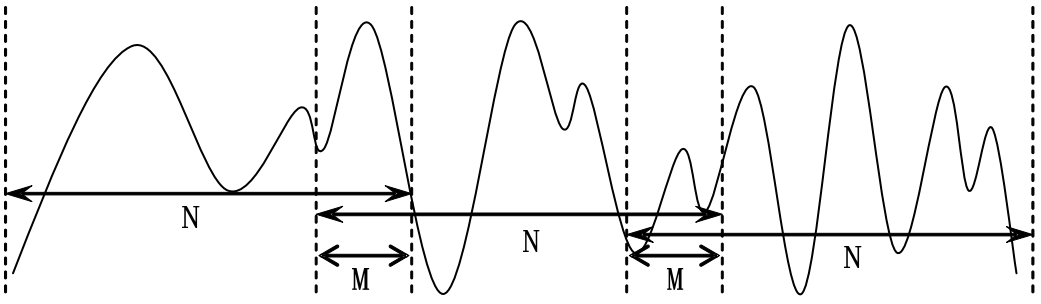

图 3-1 分帧示意图（N 为帧长，M 为帧间重叠长度）

2、窗函数

分帧是用可移动的有限长度窗口进行加权的方法来实现的，这就是用一定的窗函数 $w(n)$ 来乘 $s(n)$，从而形成加窗语音信号 $s_w(n)=s(n)*w(n)$。窗函数 $w(n)$ 的选择（形状和长度），对于短时分析参数的特性影响很大。为此应选择合适的窗口，使其短时参数更好地反映语音信号的特性变化。选择的依据有两类：

1）窗口的形状：一个好的窗函数的标准是：在时域因为是语音波形乘以窗函数，所以要减小时间窗两端的坡度，使窗口边缘两端不引起急剧变化而平滑过渡到零，这样可以使截取出的语音波形缓慢降为零，减小语音帧的截断效应；在频域要有较宽的 3dB 带宽以及较小的边带最大值。

2）窗口的长度：如果长度很大，则它等效于很窄的低通滤波器，语音信号通过时，反映波形细节的高频部分被阻碍，短时能量随时间变化很小，不能真实的反映语音信号的幅度变化；反之，长度太小时，滤波器的通带变宽，短时能量随时间有急剧的变化，不能得到平滑的能量函数。通常认为在一个语音帧内应包含 1~7 个基音周期。然而不同人的基音周期变化很大，从女性和儿童的 2ms 到老年男子的 14ms（即基音频率的变化范围为 500Hz~70Hz），所以 N 的选择比较困难。通常在 10kHz 取样频率下，N 折中选择为 100~200 点为宜(即 10~20ms 持续时间)。

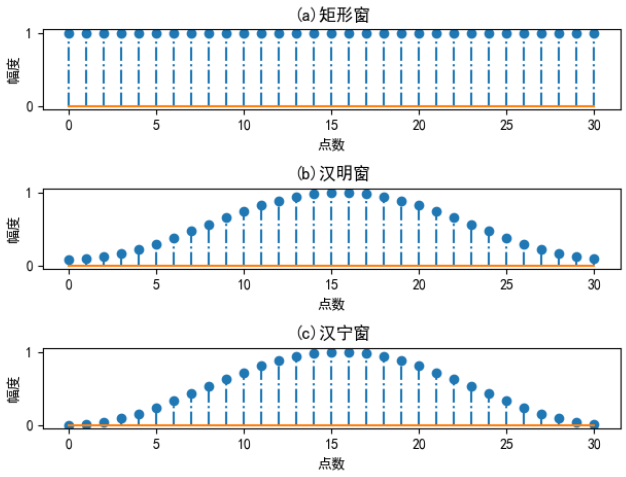

图 3-2 窗函数时域波形

在语音信号数字处理中常用的窗函数有三种：

1）矩形窗：

$$w(n)=\begin{cases}1 & 0\le n\le N-1\\ 0 & \text{其他}\end{cases} \qquad (3-1)$$

2）汉明窗：

$$w(n)=\begin{cases}0.54-0.46\cos[2\pi n/(N-1)] & 0\le n\le N-1\\ 0 & \text{其他}\end{cases} \qquad (3-2)$$

3）海宁窗：

$$\omega(n)=\begin{cases}0.5(1-\cos(2\pi n/(N-1))) & 0\le \mathrm{n} << N-1\\ 0 & \text{其他}\end{cases} \quad (3\text{-}3)$$

Pyhton 的 numpy 库给出了汉明窗和海宁窗的函数 numpy.hamming(N)，numpy.hanning(N)。其中，N 代表窗口长度。三种窗口的时域波形如图 3-2 所示。

3．加权交叠相加法

语音信号处理在实时频域计算时通常采用 WOLA 技术进行分帧处理。加权重叠相加法(Weighted OLA，WOLA)在 IFFT 变换之后还要对数据进行加权，加权使用窗函数实现，然后按照 FFT 对应段再相加，这个窗被称为综合窗或输出窗，重叠相加结果是最终结果。加窗的目的是减小截断效应带来的不利影响，以抑制帧边缘主观听感上的不连续性，常用于语音信号处理场景。常用的窗函数有均方根汉宁窗(root-Hanninng)和均方根布莱克曼窗(root-Blackman)，要求不高的场景通常只在 FFT 变换时加窗。

图 3-3 WOLA 基本原理

WOLA 计算过程如图 3-3 所示。具体步骤为：

1）分块：对于实时语音信号处理来说，数据的处理是以帧为单位来进行的，帧的大小可以依据应用来定。比如 10ms 一帧，那么对于 16kHz 的数据来说，一帧就是 160 点。

2）拼帧：拼帧的主要目的有两个：1）利用语音的短时平稳性；2）为了使用快速 FFT 来实现计算效率。比如将 160 点的一帧数据与上一帧的后 96 点数据一起拼成 256 点的实际一帧算法处理的数据。

3）加窗：对第 n 帧数据进行加窗截断，得到分段加窗数据：

$$x_n(m)=w(m)x(m-nR) \qquad m=nR,\cdots,nR+N-1 \quad (3\text{-}4)$$

其中，R 是平移长度，N 是分段后帧长。在上例中，R 为 160，N 为 256。

4）FFT：对每段加窗数据进行 FFT，则

$$X_n(w_k)=FFT(x_n(m)),k=0,\cdots,N-1 \quad (3\text{-}5)$$

5）信号处理：对频域信号 $X_n(w_k)$ 进行处理可到 $Y_n(w_k)$。

6）IFFT：对处理后的频域信号 $Y_n(w_k)$ 进行 IFFT，得

$$y_n(m),m=0,\cdots,N-1 \quad (3\text{-}6)$$

7）加窗：对该信号乘以窗函数，此处的窗函数通常和第 3）步保持一致，得

$$\hat{y}_n(m)=w(m)y_n(m) \quad (3\text{-}7)$$

8）去交叠：取出第 n 帧信号 $\hat{y}_n(m)$ 前 R 个点，并将其中前 N-R 个点与前一帧输出的后 N-R 个点相加，得到最终输出信号。如果第 5）步的信号不做任何处理，即 $Y_n(w_k)=X_n(w_k)$。那么，WOLA 得到的最终时域信号即为最初的原始输入信号。

如果不进行加窗处理的话，该方法会产生时域混叠，这会在逐渐减弱的时域信号段拼接处引入峰值，从而在听感上会有“啪啪声”。而 WOLA 的方法可以避免这种问题，为了保证幅度平稳性（听感上响度一致)，其窗函数的均方根需满足式（3-8）：

$$\sum_{n} w^2(m-nR)=C, n\in Z \tag{3-8}$$

满足上式的窗函数都可以用在 WOLA 里。但是，不同的窗具有的最优重叠长度不一样，如布莱克曼哈里斯窗（Blackman-Harris）的最优重叠长度为窗长的 66.1%，而汉明窗和汉宁窗的最优重叠长度为窗长的 50%。

4、消除趋势项和直流分量

在采集语音信号数据的过程中，由于测试系统的某些原因在时间序列中会产生的一个线性的或者慢变的趋势误差，例如放大器随温度变化产生的零漂移，传声器低频性能的不稳定或传声器周围的环境干扰，总之使语音信号的零线偏离基线，甚至偏离基线的大小还会随时间变化。零线随时间偏离基线被称为信号的趋势项。趋势项误差的存在，会使相关函数、功率谱函数在处理计算中出现变形，甚至可能使低频段的谱估计完全失去真实性和正确性，所以应该将其去除。数据采集仪中信号放大器受环境温度、湿度、电磁场、噪声、振动冲击等外部环境的干扰，采集到的振动信号往往与真实信号发生了偏离，使信号信噪比降低、甚至信号完全失真。一般情况下测量被测物体的加速度比测量位移和速度方便得多。但由于信号中含有长周期趋势项，在对数据进行二次积分时得到的结果可能完全失真，因此消除长周期趋势项是振动信号预处理的一项重要任务。直流分量的消除比较简单，即减去语音信号的平均项即可。而对于线性趋势项或多项式趋势项，常用的消除趋势项的方法是用多项式最小二乘法。

设实测语音信号的采样数据为 $\{x_k\}(k=1,2,3,\cdots,n)$， n 为样本总数，由于采样数据是等时间间隔的，为简化起见，令采样时间间隔 $\Delta t=1$。用一个多项式函数 $\hat{x}_k$ 表示语音信号中的趋势项：

$$\hat{x}_k=a_0+a_1k+a_2k^2+\cdots+a_mk^m=\sum_{j=0}^{m}a_jk^j\ \ (k\in[1,n]) \tag{3-9}$$

为了确定各待定系数 a_j，令函数 $\hat{x}_k$ 与离散数据 x_k 的误差二次方和为最小，即

$$E=\sum_{k=1}^{n}(\hat{x}_k-x_k)^2=\sum_{k=1}^{n}(\sum_{j=0}^{m}a_jk^j\ -x_k)^2 \tag{3-10}$$

对 E 求偏导，得

$$\frac{\partial E}{\partial a_i}=2\sum_{k=1}^{n}k^i(\sum_{j=0}^{m}a_jk^j\ -x_k)=0\ \mathrm{i}\in[0,\mathrm{m}] \tag{3-11}$$

依次对 a_i 求偏导，可得 $m+1$ 元线性方程组

$$\sum_{k=1}^{n}\sum_{j=0}^{m}a_j k^{j+i}-\sum_{k=1}^{n}x_k k^i=0 \quad i\in[0,m] \tag{3-12}$$

通过解方程组求出 $m+1$ 个待定系数 a_j。各式中，m 为设定的多项式阶次。

当 $m=0$ 时求得的趋势项为常数，有

$$\sum_{k=1}^{n}a_0 k^0-\sum_{k=1}^{n}x_k k^0=0 \tag{3-13}$$

解方程得

$$a_0=\frac{1}{n}\sum_{k=1}^{n}x_k \tag{3-14}$$

由此可知，当 $m=0$ 时的趋势项为信号采样数据的算术平均值，即是直流分量。消除常数趋势项的计算公式为

$$y_k=x_k-\hat{x}_k=x_k-a_0 \tag{3-15}$$

当 $m=1$ 时为线性趋势项，有

$$\begin{cases}\sum_{k=1}^{n}a_0 k^0+\sum_{k=1}^{n}a_1 k-\sum_{k=1}^{n}x_k k^0=0\\ \sum_{k=1}^{n}a_0 k+\sum_{k=1}^{n}a_1 k^2-\sum_{k=1}^{n}x_k k=0\end{cases} \tag{3-16}$$

解方程组得

$$\begin{cases}a_0=\dfrac{2(2n+1)\sum_{k=1}^{n}x_k-6\sum_{k=1}^{n}x_k k}{n(n-1)}\\ a_1=\dfrac{12\sum_{k=1}^{n}x_k k-6(n-1)\sum_{k=1}^{n}x_k}{n(n-1)(n+1)}\end{cases} \tag{3-17}$$

消除线性趋势项的计算公式为

$$y_k=x_k-\hat{x}_k=x_k-(a_0+a_1 k) \tag{3-18}$$

当 $m\geq 2$ 时为曲线趋势项。在实际语音信号数据处理中，通常取 $m=1\sim 3$ 来对采样数据进行多项式趋势项消除的处理。

在 Python 的 scipy 库里带有消除线性趋势项的函数 detrend。函数定义为：

函数格式：y = scipy.signal.detrend(data)

输入参数：data 是带有线性趋势项的信号序列。

输出参数：y 是消除趋势项的序列。

5、数字滤波器

在采集语音信号时，交流隔离不好常会将工频 50Hz 的交流声混入到语音信号中，因此需要采用高通滤波器滤除工频干扰；此外，由于基音的频率较低，通常位于 60-450Hz 之间。因此，在基音提取算法中，为了抗干扰，常设计低通滤波器来提取低频段信号。

常用的经典 IIR 数字滤波器包含巴特沃斯滤波器、切比雪夫 I 型滤波器、切比雪夫 II 型滤波器和椭圆滤波器四类。基于 Python 的数字滤波器设计步骤为：

1）根据设计指标确定滤波器参数

滤波器的设计指标包括：通带截止频率 Wp 和阻带截止频率 Ws，其取值范围为 0 至 1 之间，当其值为 1 时代表采样频率的一半。通带和阻带区的波纹系数分别是 Rp 和 Rs。

不同类型（高通、低通、带通和带阻）滤波器对应的 Wp 和 Ws 值遵循以下规则：

a．高通滤波器：Wp 和 Ws 为一元矢量且 Wp>Ws；

b．低通滤波器：Wp 和 Ws 为一元矢量且 Wp<Ws；

c．带通滤波器：Wp 和 Ws 为二元矢量且 Wp<Ws，如 Wp=[0.2,0.7],Ws=[0.1,0.8];

d．带阻滤波器：Wp 和 Ws 为二元矢量且 Wp>Ws，如 Wp=[0.1,0.8],Ws=[0.2,0.7]。

2）采用 Python 函数设计数字滤波器

数字滤波器设计函数包括：

a．巴特沃斯滤波器：b,a= scipy.signal.butter(n,Wn,'btype');

b．切比雪夫 I 型滤波器：b,a= scipy.signal.cheby1(n,rp,Wn,'btype');（通带等波纹）

c．切比雪夫 II 型滤波器：b,a= scipy.signal. cheby2(n,rs,Wn,'btype');（阻带等波纹）

d．椭圆滤波器：b,a= scipy.signal. ellip(n,rp,rs,Wn,'ftype');

这里 btype 的取值包括'lowpass',’highpass’,’bandpass’,'bandstop'，分别指代低通、高通、带通和带阻（通）。其中，n 代表滤波器阶数，Wn 为滤波器的截止频率（无论高通、带通、带阻滤波器在设计中最终都等效于一个低通滤波器）。

参考例程：设计一个椭圆带通滤波器，通带衰减 3dB，阻带衰减 80dB，通带频率为[45 55]Hz，阻带频率为[40 60]Hz，并画出滤波器的幅频曲线和相频曲线。

参考程序：

```
from speechlib import *                                    //自定义的库

fs = 200;
fs2=fs/2;
# 把截止频率转成弧度表示
wp = np.divide([45, 55], fs2);
ws = np.divide([40, 60], fs2);
rp = 3;
rs = 80;
Nn = 512;
# 根据需求求出椭圆带通滤波器的系数
[n,wn] = signal.ellipord(wp, ws, rp, rs);
[b,a] = signal.ellip(n,rp,rs,wn,btype='bandpass');
[w, h] = signal.freqz(b, a, Nn, fs);
```

```
am = np.absolute(h[int(h.size/2):][::-1])
ph = np.unwrap(np.angle(h[:int(h.size/2)]))
f = np.linspace(0,fs/2,int(h.size/2))
# 画图
plt.figure(1)
plt.subplot(211)
plt.title('（a）幅频曲线')
plt.plot(f,20*np.log10(am))
plt.ylabel('幅度/dB')
plt.subplot(212)
plt.title('（b）相频曲线')
plt.plot(f,ph*180/np.pi)
plt.ylabel('相位')
plt.xlabel('频率/Hz')
plt.show()
```

幅频曲线和相频曲线如图 3-4所示。

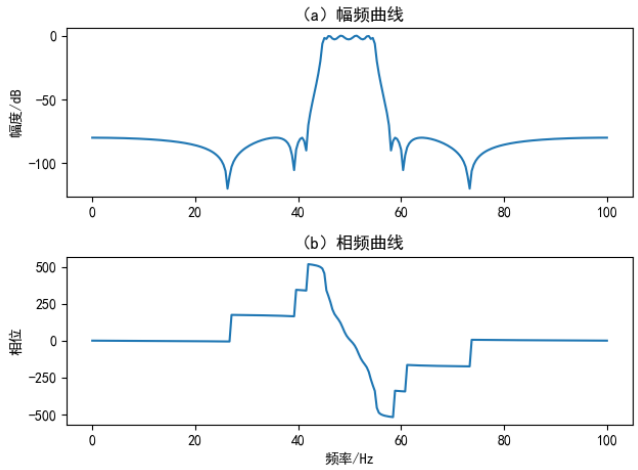

图 3-4 滤波器的幅频曲线和相频曲线

6、预加重与去加重

语音和图像信号低频段能量大，高频段信号能量明显小；而鉴频器输出噪声的功率谱密度随频率的平方而增加（低频噪声小，高频噪声大），造成信号的低频信噪比很大，而高频信噪比明显不足，使高频传输困难。调频收发技术中，通常采用预加重和去加重技术来解决这一问题。预加重：发送端对输入信号高频分量的提升；去加重：解调后对高频分量的压低。

对于语言和音乐来说，其功率谱随频率的增加而减小，其大部分能量集中在低频范围内，这就造成消息信号高频端的信噪比可能降到不能容许的程度。但是由于消息信号中较高频率分量的能量小，很少有足以产生最大频偏的幅度，因此产生最大频偏的信号幅度多数是由信号的低频分量引起。平均来说，幅度较小的高频分量产生的频偏小得多。因为调频系统的传输带宽是由需要传送的消息信号

（调制信号）的最高有效频率和最大频偏决定的，所以调频信号并没有充分占用给予它的带宽。然而，接收端输入的噪声频谱却占据了整个调频带宽，也就是说，鉴频器输出端的噪声功率谱在较高频率上已被加重了。

为了抵消这种不希望有的现象，在调频系统中普遍采用一种叫做预加重和去加重措施，其中心思想是利用信号特性和噪声特性的差别来有效地对信号进行处理。在噪声引入之前采用适当的网络（预加重网络），人为地加重（提升）发射机输入调制信号的高频分量。然后在接收机鉴频器的输出端，再进行相反的处理，即采用去加重网络把高频分量去加重，恢复原来的信号功率分布。在去加重过程中，同时也减小了噪声的高频分量，但是预加重对噪声并没有影响，因此有效地提高了输出信噪比。很多信号处理都使用这个方法，对高频分量电平提升（预加重）然后记录（调制、传输），播放（解调）时对高频分量衰减（去加重）。录音带系统中的杜比系统是个典型的例子。假设信号高频分量为 10，经记录后，再播放时，引入的磁带本底噪声为 1，那么还原出来信号高频段信躁比为 10：1；如果在记录前对信号的高频分量提升，假设提升为 20，经记录后再播放时，引入的磁带本底噪声为 1，此时依然是 10：1 的信躁比，但是此时的高频分量是被提升了的，在对高频分量进行衰减的同时，磁带本底噪声也被衰减，如果将信号高频分量衰减还原到原来的 10，则本底噪声就会被降低到 0.5。

常用所谓“预加重技术”是在取样之后，插入一个一阶的高通滤波器，具体效果如图 3-5 所示。常用的预加重因子为 $[1-R(1)z^{-1}/R(0))]$。这里，$R(n)$ 是信号 $s(n)$ 的自相关函数。通常对于浊音，$R(1)/R(0)\approx 1$；而对于清音，则该值可取得很小。在语音播放时再进行“去加重”处理，即预加重的反处理。

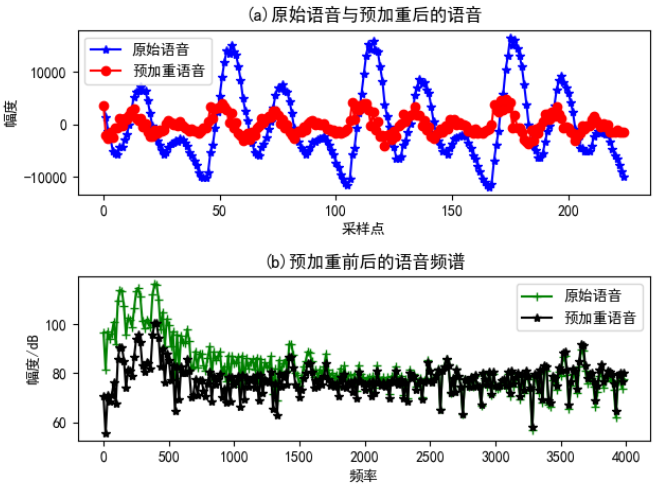

图 3-5 语音信号处理预加重效果图

3.1.3 实验要求

1）根据语音分帧的思想，编写分帧函数。函数定义如下：

函数格式：frameout=enframe(x,win,inc)。

输入参数：x 是语音信号；win 是帧长或窗函数，若为窗函数，帧长便取窗函数长；inc 是帧移。

输出参数：frameout 是分帧后的数组，长度为帧长和帧数的乘积。

根据分帧后的语音，绘制连续四帧语音信号（不用窗函数），效果如图 3-6 所示。

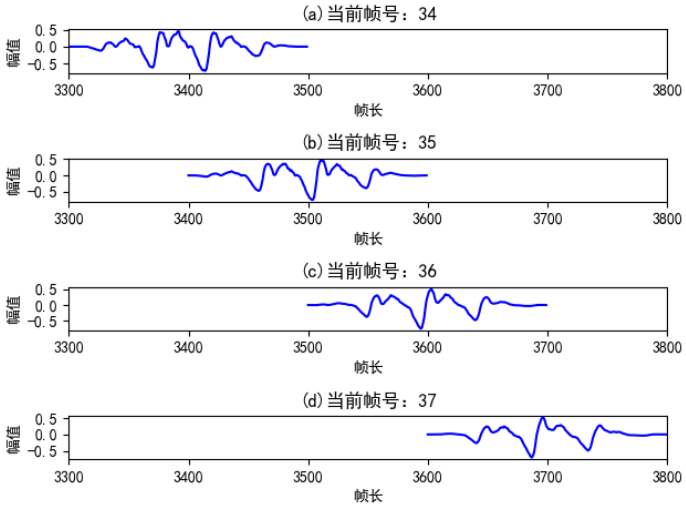

图 3-6 连续四帧语音信号

2）编程实现矩形窗、汉明窗和汉宁窗，效果如图 3-2 所示。

3）编程实现 WOLA 的全过程（中间信号处理不做），观察信号的复原程度（参考语音信号 sweep.wav），效果如图 3-7 所示。

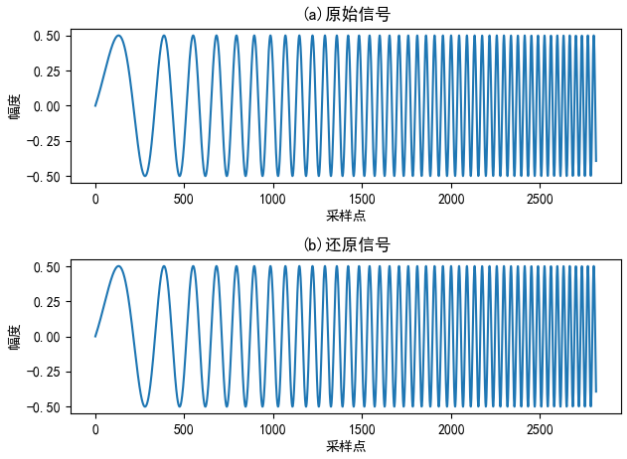

图 3-7 WOLA 信号的复原显示

4）测试 Python 的去线性趋势项函数 detrend 的作用。根据消除多项式趋势项的原理，编写消除多项式趋势项的函数 detrendN，并仿真测试，参考语音信号 C3_1_y_2.wav。

函数定义如下：

函数格式：y=detrendN(x,fs,m)

输入参数：x 是带有趋势项的信号序列；fs 是采样频率；m 是多项式的阶次。

输出参数：y 是消除趋势项的序列。

仿真效果如图 3-8 所示。

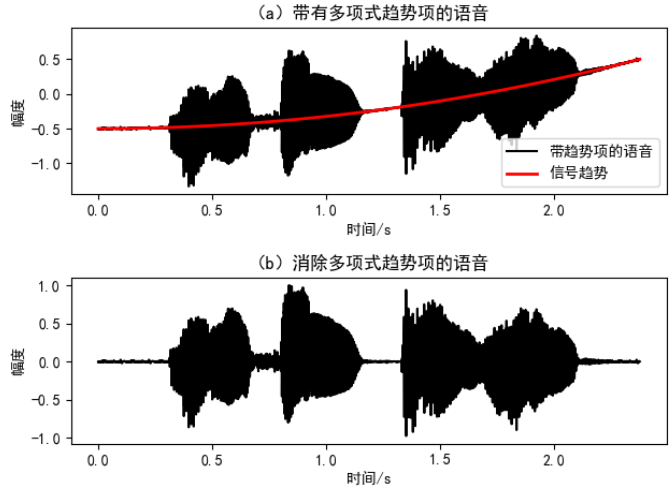

图 3-8 消除趋势项效果图

5)参考例程，设计一个低通滤波器，并绘出其幅频和相频曲线。设计指标为：fs=8000Hz，Wp=60Hz，Ws=50Hz， Rp=3dB，Rs=80dB。

6）编写预加重函数实现类似图 3-5 的效果，参考语音信号 C3_1_y_1.wav。

3.1.4 思考题

1）编程实现将分帧加窗后的语音信号恢复成原始分帧前的信号。

2）使用实验要求 5）设计的低通滤波器来滤去语音信号中的低频成分，并用 FFT 分析其频谱变化情况；

3）编写去加重函数，并绘制如图 3-8 所示的图例来说明其效果。

3.1.5 参考例程

```
WOLA 算法解析
from speechlib import *
import soundfile

s,fs   = soundfile.read('sweep.wav')
Slen=len(s)
FFTLen=128              #FFT 长度
FrameLen=64             #帧长
win = np.sin(np.linspace(0.5,FFTLen-0.5,FFTLen)/FFTLen*np.pi).T
ss=np.zeros(FFTLen)
```

```
sst=np.zeros(FFTLen)
frameoverlap=np.zeros(FFTLen-FrameLen)
framenum=int(Slen/FrameLen)-1                # 求帧数
out = np.zeros(Slen)

for kk in range(framenum):
    pos = kk * FrameLen
    In =s[pos: pos + FrameLen]
    ss[FFTLen - FrameLen:] = In
    ss[0: FFTLen - FrameLen]=sst[FrameLen:]
    sst = ss.copy()

    ss1 = ss*win
    fftss = np.fft.fft(ss1)
    # # # %%%%%%%%%%%添加必要的算法%%%%%%%%%%%
    ifftss = np.real(np.fft.ifft(fftss))
    ifftss = ifftss*win
    ifftss[0: FFTLen - FrameLen]=ifftss[0: FFTLen - FrameLen]+frameoverlap
    frameoverlap = ifftss[FrameLen:]
    out[pos: pos + FrameLen]=ifftss[0: FrameLen]

out=out[(FFTLen-FrameLen):]
lenout=len(out)
re = np.zeros([2,lenout])
re[0,:]=s[0:lenout]
re[1,:]=out

plt.subplot(2,1,1)
plt.plot(re[0,:])
plt.xlabel('采样点')
plt.ylabel('幅度')
plt.title('(a)原始信号')

plt.subplot(2,1,2)
plt.plot(re[1,:])
plt.xlabel('采样点')
plt.ylabel('幅度')
plt.title('(b)还原信号')
plt.show()

soundfile.write('sweepN.wav', out, fs)
```

3.2 短时时域分析

3.2.1 实验目的

1）了解语音信号短时时域分析的原理；
2）掌握短时时域分析的一些参数计算方法；
3）根据原理能编程实现短时时域分析的参数计算。

3.2.2 实验原理

语音信号的时域分析就是分析和提取语音信号的时域参数。语音信号本身就是时域信号，因而时域分析是最早使用，也是应用最广泛的一种分析方法，这种方法直接利用语音信号的时域波形。时域分析通常用于最基本的参数分析及应用，如语音的分割、预处理、分类等。语音信号的时域参数有短时能量、短时过零率、短时自相关函数和短时平均幅度差函数等。这些最基本的短时参数在各种语音信号数字处理技术中都有重要的应用。

1、短时能量与短时平均幅度

设第n帧语音信号$x_n(m)$的短时能量用E_n表示，则其计算公式如下：

$$E_n = \sum_{m=0}^{N-1} x_n^2(m) \qquad (3\text{-}19)$$

E_n是一个度量语音信号幅度值变化的函数，但它有一个缺陷，即它对高电平非常敏感（因为它计算时用的是信号的平方）。为此，可采用另一个度量语音信号幅度值变化的函数，即短时平均幅度函数M_n，它定义为：

$$M_n = \sum_{m=0}^{N-1} |\, x_n(m)\,| \qquad (3\text{-}20)$$

M_n也是一帧语音信号能量大小的表征，它与E_n的区别在于计算时小取样值和大取样值不会因取平方而造成较大差异，在某些应用领域中会带来一些好处。

2、短时过零率

短时过零率表示一帧语音中语音信号波形穿过横轴（零电平）的次数。对于连续语音信号，过零即意味着时域波形通过时间轴；而对于离散信号，如果相邻的取样值改变符号则称为过零。因此，过零率就是样本改变符号的次数。

定义语音信号$x_n(m)$的短时过零率Z_n为：

$$Z_n = \frac{1}{2}\sum_{m=0}^{N-1} |\mathrm{sgn}[x_n(m)] - \mathrm{sgn}[x_n(m-1)]| \qquad (3\text{-}21)$$

式中，sgn[·]是符号函数，即：

$$\mathrm{sgn}[x]=\begin{cases}1 & ,\quad (x\geq 0)\\ -1, & (x<0)\end{cases} \tag{3-22}$$

在实际中求过零率参数时，需要十分注意的一个问题是如果输入信号中包含有 50Hz 的工频干扰或者 A/D 变换器的工作点有偏移(这等效于输入信号有直流偏移)，往往会使计算的过零率参数很不准确。为了解决前一个问题，A/D 变换器前的防混叠带通滤波器的低端截频应高于 50Hz，以有效地抑制电源干扰。对于后一个问题除了可以采用低直流漂移器件外，也可以在软件上加以解决，这就是算出每一帧的直流分量并予以滤除。

这里应注意，在 MATALAB 编程中，实际上求短时平均过零率并不是按照上述公式计算，而是使用了另外一种方法。按上述过零的描述，即离散信号相邻的取样值改变符号，那它们的乘积一定为负数，即

$$\chi_{\mathrm{i}}(\mathrm{m})*\chi_{\mathrm{i}}(\mathrm{m}+1)<0 \tag{3-23}$$

3、短时自相关

自相关函数具有一些性质，如它是偶函数；假设序列具有周期性，则其自相关函数也是同周期的周期函数等。对于浊音语音可以用自相关函数求出语音波形序列的基音周期。此外，在进行语音信号的线性预测分析时，也要用到自相关函数。

语音信号 $x_n(m)$ 的短时自相关函数 $R_n(k)$ 的计算式如下：

$$R_n(k)=\sum_{m=0}^{N-1-k}x_n(m)x_n(m+k)\qquad (0\leq k\leq K) \tag{3-24}$$

这里 K 是最大的延迟点数。

短时自相关函数具有以下性质：

（1）如果 $x_n(m)$ 是周期的（设周期为 N_p），则自相关函数是同周期的周期函数，即 $R_n(k)=R_n(k+N_p)$。

（2）$R_n(k)$ 是偶函数，即 $R_n(k)=R_n(-k)$。

（3）当 $k=0$ 时，自相关函数具有最大值，即 $R_n(0)\geq|R_n(k)|$，并且 $R_n(0)$ 等于确定性信号序列的能量或随机性序列的平均功率。

4、短时平均幅度差

短时自相关函数是语音信号时域分析的重要参量。但是，计算自相关函数的运算量很大，其原因是乘法运算所需要的时间较长。利用快速傅里叶变换（FFT）等简化计算方法都无法避免乘法运算。为了避免乘法，一个简单的方法就是利用差值。为此常常采用另一种与自相关函数有类似作用的参量，即短时平均幅度差函数。

平均幅度差函数能够代替自相关函数进行语音分析的原因在于：如果信号是完全的周期信号(设周期为N_p)，则相距为周期的整数倍的样点上的幅值是相等的，差值为零。即：

$$d(n) = x(n) - x(n+k) = 0 \qquad (k = 0,\ \pm N_p,\ \pm 2N_p, ...) \tag{3-25}$$

对于实际的语音信号，$d(n)$虽不为零，但其值很小。这些极小值将出现在整数倍周期的位置上。为此，可定义短时平均幅度差函数：

$$F_n(k) = \sum_{m=0}^{N-1-k} \left| x_n(m) - x_n(m+k) \right| \tag{3-26}$$

显然，如果$x(n)$在窗口取值范围内具有周期性，则$F_n(k)$在$k = N_p, 2N_p, \cdots$时将出现极小值。

平均幅度差函数和自相关函数有密切的关系，两者之间的关系可由下式表达：

$$F_n(k) = \sqrt{2}\beta(k)[R_n(0) - R_n(k)]^{1/2} \tag{3-27}$$

式中$\beta(k)$对不同的语音段在0.6~1.0之间变化，但是对一个特定的语音段，它随k值变化并不明显。

显然，计算$F_n(k)$只需加、减法和取绝对值的运算，与自相关函数的加法与乘法相比，其运算量大大减少，尤其在用硬件实现语音信号分析时有很大好处。为此，平均幅度差已被用在许多实时语音处理系统中。

3.2.3 实验要求

1）为了显示方便，编程实现FrameTimeC函数，函数功能为计算分帧后每帧语音中点处对应的时间。函数定义如下：

函数格式：frametime=FrameTimeC(frameNum,framelen,inc,fs)。

输入参数：frameNum是帧的个数；framelen是帧长；inc是帧移；fs是采样频率。

输出参数：frametime是分帧后每帧对应的时间。

2）编程实现短时能量、短时平均幅度和短时过零率，显示例图如图 3-9 所示。参考测试语音为C3_2_y.wav。

每个参数的函数定义格式为：funcvalue=funcname(x,win,inc),其中x为语音信号,win为窗函数或帧长，inc为帧移,funcvalue为[1,帧数]的向量。

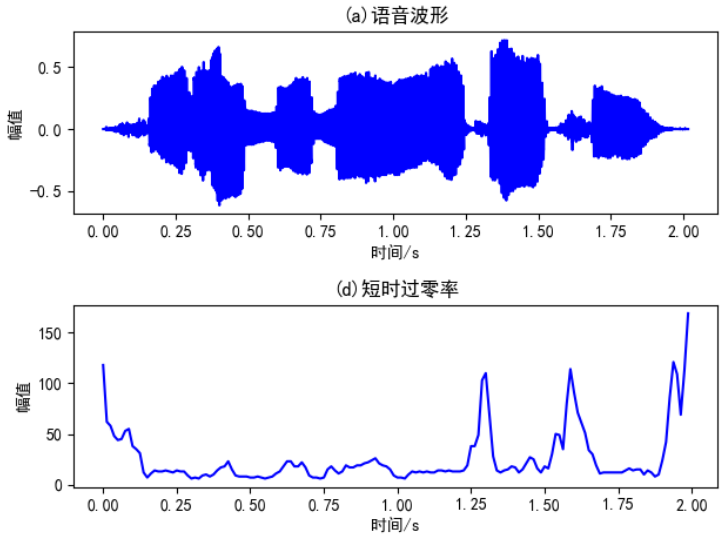

图 3-9 短时过零率的显示例图

3）编程实现短时自相关和短时平均幅度差，显示例图如图 3-10 所示。参考测试语音为 C3_2_y. wav。

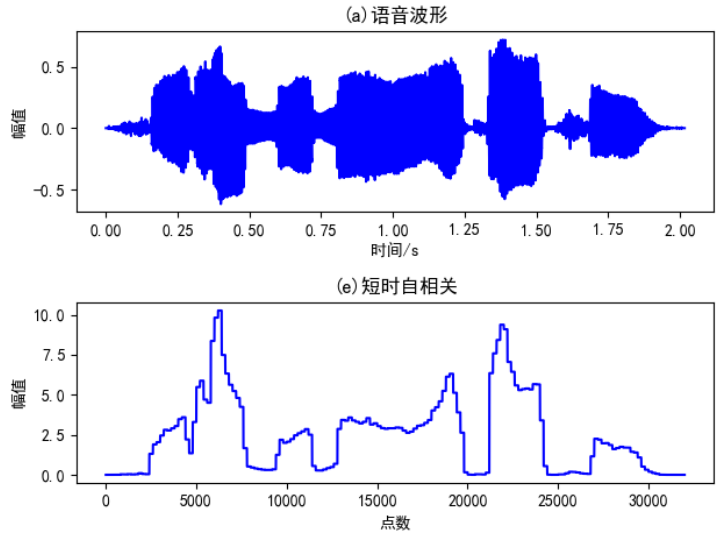

图 3-10 短时自相关的显示例图

3.2.4 思考题

1）编程比较不同的窗函数对短时时域参数估计的影响。

3.2.5 参考例程

【C3_2_y 短时时域分析参数计算并显示】

```
# -*- coding: utf-8 -*-
from speechlib import *
#读取波形数据
(framerate, wave_data) = wavfile.read("C3_2_y.wav")
```

```
wave_data = wave_data / 32768
time = np.array([i/framerate for i in range(wave_data.size)])
N = len(wave_data)

wlen = 200
inc = 100
win = np.hanning(wlen)

En = STEn(wave_data,win,inc)              #短时能量
Mn = STMn(wave_data,win,inc)              #短时平均幅度
Zcr = STZcr(wave_data,win,inc)           #短时过零率

X = enframe(wave_data,win,inc)
X = X.T
Ac = STAc(X)                                    #短时自相关
Ac = Ac.T
Ac = Ac.flatten()
X = X.T
Amdf = STAmdf(X)                              #短时平均幅度差

fn = len(En)

plt.figure(1)                                    #此处只显示短时能量
plt.subplot(311)
plt.plot(time,wave_data,'b')
plt.axis("tight")
plt.title(u"(a)语音波形")
plt.ylabel(u"幅值")
plt.xlabel(u"时间/s")

frameTime = FrameTimeC(fn,wlen,inc,framerate)
plt.subplot(312)
plt.plot(frameTime,Mn,'b')
plt.title(u"(b)短时幅度")
plt.ylabel(u"幅值")
plt.xlabel(u"时间/s")

plt.subplot(313)
plt.plot(frameTime,En,'b')
plt.title(u"(c)短时能量")
plt.ylabel(u"幅值")
plt.xlabel(u"时间/s")
```

3.3 短时频域分析

3.3.1 实验目的

1）了解短时傅里叶变换的原理，并编程实现短时傅里叶函数；
2）了解语谱图的意义和表现方法，并编程实现；

3.3.2 实验原理

1、短时傅里叶变换

语音信号是一种典型的非平稳信号，但是其非平稳性是由发音器官的物理运动过程而产生的，此过程与声波振动的速度相比较缓慢，可以假定在10～30ms 这样的短时间内是平稳的。傅立叶分析是分析线性系统和平稳信号稳态特性的强有力的手段，而短时傅里叶分析，也叫时间依赖傅立叶变换，就是在短时平稳的假定下，用稳态分析方法处理非平稳信号的一种方法。

设语音波形时域信号为 $x(l)$、加窗分帧处理后得到的第 n 帧语音信号为 $x_n(m)$，则 $x_n(m)$ 满足下式：

$$x_n(m) = w(m)x(n+m) \qquad 0 \le m \le N-1 \qquad (3\text{-}28)$$

设离散时域采样信号为 $x(n)$，$n=0,1,...,N-1$，其中 n 为时域采样点序号，$N-1$ 是信号长度。然后对信号进行分帧处理，则 $x(n)$ 表示为 $x_n(m)$，$m=0,1,...,N-1$，其中 n 是帧序号，m 是帧同步的时间序号。信号 $x(n)$ 的短时傅里叶变换为：

$$X_n(e^{jw}) = \sum_{m=0}^{N-1} x_n(m)e^{-jwm} \qquad (3\text{-}29)$$

定义角频率 $w = 2\pi k / N$，则得离散的短时傅里叶变换(DFT)，它实际上是 $X_n(e^{jw})$ 在频域的取样，如下所示：

$$X_n(e^{j\frac{2\pi k}{N}}) = X_n(k) = \sum_{m=0}^{N-1} x_n(m)e^{-j\frac{2\pi km}{N}} \qquad (0 \le k \le N-1) \qquad (3\text{-}30)$$

在语音信号数字处理中，都是采用 $x_n(m)$ 的离散傅里叶变换 $X_n(k)$ 来替代 $X_n(e^{jw})$，并且可以用高效的快速傅里叶变换(FFT)算法完成由 $x_n(m)$ 至 $X_n(k)$ 的转换。当然，这时窗长 N 必须是 2 的倍数 2^L（L 是整数）。

2、语谱图表示与实现方法

一般定义$|X_n(k)|$为$x(n)$的短时幅度谱估计，而时间处频谱能量密度函数（或功率谱函数）$P(n,k)$为：

$$P(n,k)=|X_n(k)|^2 \qquad (3\text{-}31)$$

则$P(n,k)$是二维的非负实值函数，并且不难证明它是信号$x(n)$的短时自相关函数的傅里叶变换。用时间n作为横坐标，k作纵坐标，将$P(n,k)$的值表示为灰度级所构成的二维图像就是语谱图。如果通过变换$10\log_{10}P(n,k)$后，得到语谱图就是采用dB进行表示的。将经过变换后的矩阵精细图像和色彩的映射后，就可得到彩色的语谱图，如图 3-11所示。

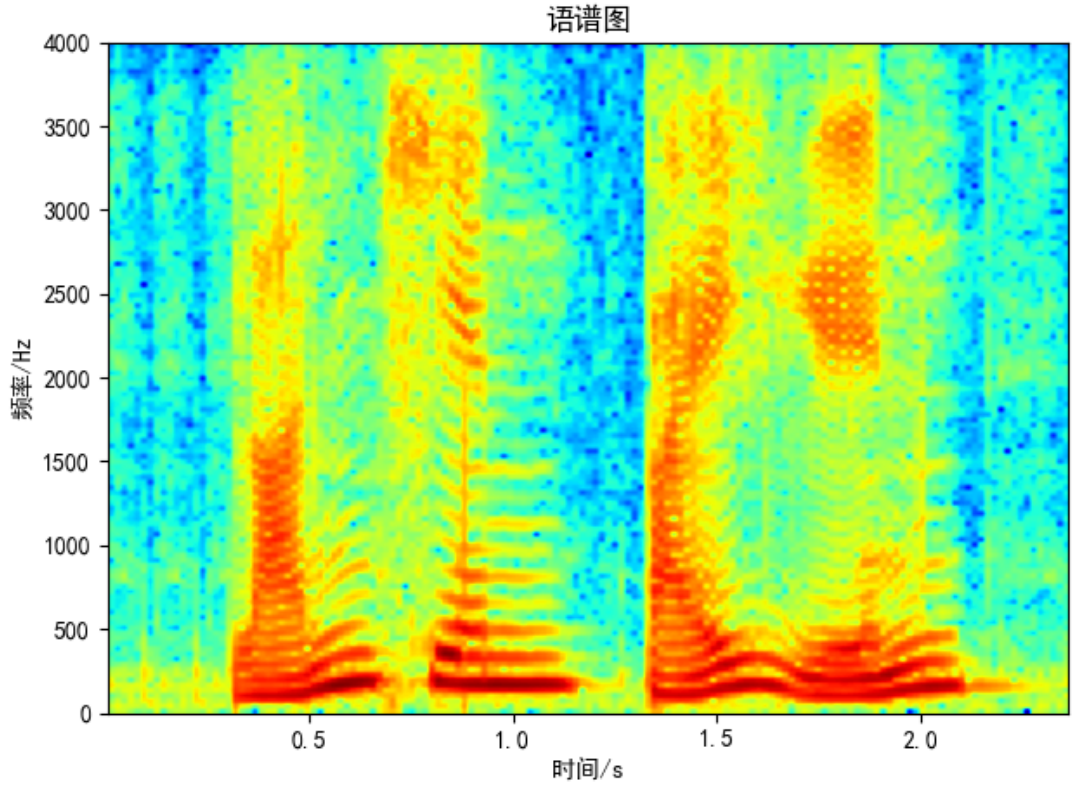

图 3-11 语谱图显示

需要的用到的Python函数如下：

matplotlib.pyplot.specgram(x, NFFT=None, Fs=None, window=None, noverlap=None)

此函数计算并绘制数据x的语谱图。它将数据分割成NFFT个片段，计算每个片段的频谱，然后将窗函数window应用于每个片段，每个片段的重叠数由noverlap指定。最终呈现的图像以时间为横轴，以频率为纵轴，以颜色表示幅值，在一幅图中表示信号的频率、幅度随时间的变化情况。

语谱图中的花纹有横杠、乱纹和竖直条等。横杠是与时间轴平行的几条深黑色带纹，它们是共振峰。从横杠对应的频率和宽度可以确定相应的共振峰频率和带宽。在一个语音段的语谱图中，有没有横杠出现是判断它是否是浊音的重要标志。竖直条（又叫冲直条）是语谱图中出现与时间轴垂直的一条窄黑条。每个竖直条相当于一个基音，条纹的起点相当于声门脉冲的起点，条纹之间的距离表示基音周期。条纹越密表示基音频率越高。

3.3.3 实验要求

1）根据短时傅里叶变换的原理，编写其函数。函数定义如下：

函数格式：d=STFFT(x,win,nfft,inc)。

输入参数：x 是语音信号；win 是帧长或窗函数，若为窗函数，帧长便取窗函数长；inc 是帧移；nfft 是快速傅里叶的点数。

输出参数：d 是得到的语谱图矩阵。

2）根据语谱图的显示原理，编程实现语谱图的计算和显示，显示效果如图 3-11 所示。

3.3.4 思考题

1）编程实现将语谱图矩阵还原为原始的语音信号。

3.3.5 参考例程

【C3_3_y 语谱图显示】

```
# -*- coding: utf-8 -*-
from speechlib import *

(framerate, wave_data) = wavfile.read("C3_3_y.wav")        #读取数据

#参数设置
wlen = 256
nfft = wlen
# win = np.hanning(wlen)
inc = 128
#画语谱图
plt.specgram(wave_data,nfft,framerate,noverlap=inc,cmap='jet')
plt.xlabel("时间/s")
plt.ylabel("频率/Hz")
plt.title("语谱图")
plt.show()
```

3.4 倒谱分析与 MFCC 系数

3.4.1 实验目的

1）了解语音信号倒谱分析的意义。
2）掌握语音信号倒谱和复倒谱分析的原理。
3）编程实现倒谱和复倒谱计算函数。

3.4.2 实验原理

1、倒谱分析

根据对语音产生的机理的研究可知，语音信号 $x(n)$ 可看作是声门激励信号 $x_1(n)$ 和声道冲激响应信号 $x_2(n)$ 的卷积，即：

$$x(n)=x_1(n)*x_2(n) \tag{3-32}$$

为了将参与卷积的各个信号分开，便于处理，同态处理是常用的方法之一，是一种将卷积关系变为求和关系的一种分离技术。一般同态系统可分解为三个部分，如图 3-4-1 所示。

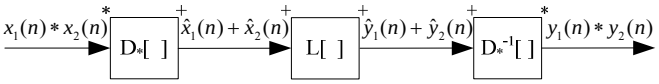

图 3-4-1 同态系统的组成

如图 3-4-1 所示，系统保护两个特征子系统（取决于信号的组合规则）和一个线性子系统（取决于处理的要求）。图中，符号 * 、+和·分别表示卷积、加法和乘法运算。

第一个子系统 $D_*[\ ]$ 完成将卷积性信号转化为加性信号的运算，即对于信号 $x(n)=x_1(n)*x_2(n)$ 进行如下运算处理：

$$\begin{cases}(1)\ Z[x(n)]=X(z)=X_1(z)\cdot X_2(z)\\(2)\ \ln X(z)=\ln X_1(z)+\ln X_2(z)=\hat{X}_1(z)+\hat{X}_2(z)=\hat{X}(z)\\(3)\ Z^{-1}[\hat{X}(z)]=Z^{-1}[\hat{X}_1(z)+\hat{X}_2(z)]=\hat{x}_1(n)+\hat{x}_2(n)=\hat{x}(n)\end{cases} \tag{3-33}$$

第二个子系统是一个普通线性系统，满足线性叠加原理，用于对加性信号进行线性变换。由于 $\hat{x}(n)$ 为加性信号，所以第二个子系统可对其进行需要的线性处理得到 $\hat{y}(n)$。

第三个子系统是逆特征系统 $D_*^{-1}[\ ]$，它对 $\hat{y}\ (n)=\hat{y}_1(n)+\hat{y}_2(n)$ 进行逆变换，使其恢复为卷积性信号，处理如下：

$$\begin{cases}(1)\ \ Z[\hat{y}(n)]=\hat{Y}(z)=\hat{Y}_1(z)+\hat{Y}_2(z)\\(2)\ \ \exp\hat{Y}(z)=Y(z)=Y_1(z)\cdot Y_2(z)\\(3)\ \ y(n)=Z^{-1}[Y_1(z)\cdot Y_2(z)]=y_1(n)*y_2(n)\end{cases} \tag{3-34}$$

由此可知，通过第一个子系统 $D_*[\ ]$，可以将 $x(n)=x_1(n)*x_2(n)$ 变换为 $\hat{x}(n)=\hat{x}_1(n)+\hat{x}_2(n)$。此时，如果 $\hat{x}_1(n)$ 与 $\hat{x}_2(n)$ 处于不同的位置并且互不交替，那么适当的设计线性系统，便可将 $x_1(n)$ 与 $x_2(n)$ 分离开来。

在 $D_*[\ ]$ 和 $D_*^{-1}[\ ]$ 系统中，$\hat{x}(n)$ 和 $\hat{y}(n)$ 信号也均是时域序列，但是它们与 $x(n)$ 和 $y(n)$ 所处的离散时域不同，称为复倒频谱域。$\hat{x}(n)$ 是 $x(n)$ 的复倒频谱域，简称复倒谱。其表达式如下：

$$\hat{x}(n)=Z^{-1}[\ln Z[x(n)]] \tag{3-35}$$

在绝大多数数字信号处理中，$X(z)$，$\hat{X}(z)$，$Y(z)$，$\hat{Y}(z)$ 的收敛域均包含单位圆，因而 $D_*[\ ]$ 和 $D_*^{-1}[\ ]$ 系统有如下形式：

$$D_*[\ ]:\begin{cases}F[x(n)]=X(e^{j\omega})\\ \hat{X}(e^{j\omega})=\ln[X(e^{j\omega})]\\ \hat{x}(n)=F^{-1}[\hat{X}(e^{j\omega})]\end{cases} \tag{3-36}$$

$$D_*^{-1}[\ ]:\begin{cases}\hat{Y}(e^{j\omega})=F[\hat{y}(n)]\\ Y(e^{j\omega})=\exp[\hat{Y}(e^{j\omega})]\\ y(n)=F^{-1}[Y(e^{j\omega})]\end{cases} \tag{3-37}$$

设 $X(e^{j\omega})=\left|X(e^{j\omega})\right|e^{j\arg[X(e^{j\omega})]}$，则对其取对数得：

$$\hat{X}(e^{j\omega})=\ln\left|X(e^{j\omega})\right|+j\arg[X(e^{j\omega})] \tag{3-38}$$

如果只考虑 $\hat{X}(e^{j\omega})$ 的实部，得：

$$c(n)=F^{-1}[\ln\left|X(e^{j\omega})\right|] \tag{3-39}$$

式中，$c(n)$ 是 $x(n)$ 对数幅值谱的逆傅里叶变换，称为倒频谱，简称倒谱。

由于浊音信号的倒谱中存在着峰值，出现位置等于该语音段的基音周期，而清音的倒谱中则不存在峰值。这个特性可以进行清浊音的判断，并且可以估计浊音的基音周期。

可利用 Python 的 numpy 库来实现倒谱的计算，定义如下三个函数分别来计算信号的复倒谱、实倒谱和复倒谱的逆变换。

1）cceps 函数——计算复倒谱

函数定义：def cceps(x):

```
    y = np.fft.fft(x)
    return np.fft.ifft(np.log(y))
```

2）rceps 函数——计算实倒谱

函数定义：def rcceps(x):

y = np.fft.fft(x)

return np.fft.ifft(np.log(np.abs(y)))

3）icceps 函数——计算逆复倒谱

函数定义：def icceps(y):

x = np.fft.fft(y)

return np.fft.ifft(np. abs(x))

2、离散余弦变换

离散余弦变换（Discrete Cosine Transform，DCT）具有信号谱分量丰富、能量集中，且不需要对语音相位进行估算等优点，能在较低的运算复杂度下取得较好的语音增强效果。

设 $x(n)$ 是 N 个有限值的一维实数信号序列，$n=0,1,\cdots,N-1$，DCT 的完备正交归一函数是：

$$\begin{cases} X(k)=a(k)\sum_{n=0}^{N-1}x(n)\cos\left(\dfrac{(2n+1)k\pi}{2N}\right) \\ x(n)=\sum_{k=0}^{N-1}a(k)\,X(k)\cos\left(\dfrac{(2n+1)k\pi}{2N}\right) \end{cases} \tag{3-40}$$

式中，$a(k)$ 的定义为：

$$a(k)=\begin{cases}\sqrt{1/N}, & \text{k=0} \\ \sqrt{2/N}, & \text{k}\in[1,\text{N-1}]\end{cases} \tag{3-41}$$

式中，$n=0,1,\cdots,N-1$；$k=0,1,\cdots,N-1$。

将式（3-4-9）略作变形，可得到 DCT 的另一表示形式：

$$X(k)=\sqrt{\frac{2}{N}}\sum_{n=0}^{N-1}C(k)x(n)\cos\left(\frac{(2n+1)k\pi}{2N}\right) \quad \text{k=0,1,}\cdots\text{,N-1} \tag{3-42}$$

式中，$C(k)$ 是正交因子。

$$C(k)=\begin{cases}\sqrt{2}/2, & \text{k=0} \\ 1, & \text{k}\in[1,\text{N-1}]\end{cases} \tag{3-43}$$

则 DCT 的逆变换为：

$$x(n)=\sqrt{\frac{2}{N}}\sum_{k=0}^{N-1}C(k)X(k)\cos\left(\frac{(2n+1)k\pi}{2N}\right) \quad \text{n=0,1,}\cdots\text{,N-1} \tag{3-44}$$

根据上述公式，利用 Python 实现离散余弦变换：

1）dct 函数——离散余弦变换

函数定义：def dct(x):

N = len(x)

X = np.zeros(N)

```
    ts = np.array([i for i in range(N)])
    C = np.ones(N)
    C[0] = np.sqrt(2) / 2
    for k in range(N):
        X[k] = np.sqrt(2 / N) * np.sum(C[k] * np.multiply(x, np.cos((2 * ts + 1) * k * np.pi / 2 / N)))
    return X
```

说明：输入 x 是原始信号；输出 X 是离散余弦变换后的序列。定义此函数需调用 numpy 库 import numpy as np

2）idct 函数——离散余弦逆变换

函数定义：def idct(X):

```
    N = len(X)
    x = np.zeros(N)
    ts = np.array([i for i in range(N)])
    C = np.ones(N)
    C[0] = np.sqrt(2) / 2
    for n in range(N):
        x[n] = np.sqrt(2 / N) * np.sum(np.multiply(np.multiply(C[ts], X[ts]), np.cos((2 * n + 1) * np.pi * ts / 2 / N)))
    return x
```

说明：x 是原始信号；X 是离散余弦变换后的序列。定义此函数需调用 numpy 库 import numpy as np。

3、Mel 频率倒谱系数

Mel 频率倒谱系数（Mel-Frequency Cepstral Coefficients，MFCC）的分析是基于人的听觉特性机理，即根据人的听觉实验结果来分析语音的频谱。因为人耳所听到的声音的高低与声音的频率并不成线性正比关系，所以用 Mel 频率尺度则更符合人耳的听觉特性。Mel 频率尺度的值大体上对应于实际频率的对数分布关系，其与实际频率的具体关系可用下式表示：

$$F_{Mel}(f)=1125\ln(1+f/700) \qquad (3\text{-}45)$$

式中，F_{Mel} 是以美尔（Mel）为单位的感知频率；f 是以 Hz 为单位的实际频率。临界频率带宽随着频率的变化而变化，并与 Mel 频率的增长一致，在 1000Hz 以下，大致呈线性分布，带宽为 100Hz 左右；在 1000Hz 以上呈对数增长。类似于临界频带的划分，可以将语音频率划分成一系列三角形的滤波器序列，即 Mel 滤波器组，如图 3-12 所示。

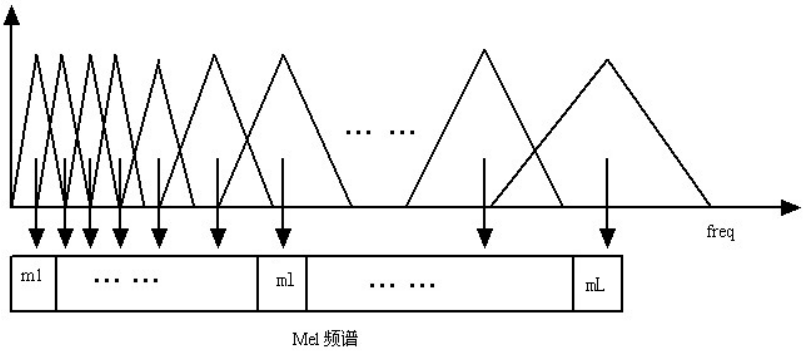

图 3-12 Mel 频率尺度滤波器组

在语音的频谱范围内设置若干带通滤波器$H_m(k)$，$0 \le m \le M$，M 为滤波器的个数。每个滤波器具有三角形滤波特性，其中心频率为 $f(m)$，在 Mel 频率范围内，这些滤波器是等带宽的。每个带通滤波器的传递函数为：

$$H_m(k)=\begin{cases}0 & k<f(m-1)\\ \dfrac{k-f(m-1)}{f(m)-f(m-1)} & f(m-1)\le k\le f(m)\\ \dfrac{f(m+1)-k}{f(m+1)-f(m)} & f(m)\le k\le f(m+1)\\ 0 & k>f(m+1)\end{cases} \tag{3-46}$$

其中$\sum_{m}^{M-1}H_m(k)=1$。

Mel 滤波器的中心频率 $f(m)$ 定义为：

$$f(m)=\frac{N}{f_s}F_{Mel}^{-1}(F_{Mel}(f_l)+m\frac{F_{Mel}(f_h)-F_{Mel}(f_l)}{M+1}) \tag{3-47}$$

其中，f_h 和 f_l 分别为滤波器组的最高频率和最低频率，f_s 为采样频率，单位为 Hz。 M 是滤波器组的数目，N 为 FFT 变换的点数，式中 $F_{Mel}^{-1}(b)=700(e^{\frac{b}{1125}}-1)$。

在 MATLAB 中，melbankm 函数可用于计算 Mel 滤波器组。函数定义如下：

调用格式：h=melbankm(p,n,fs,fl,fh,w)

输入参数：fs 是采样频率；fl 是设计的滤波器的最低频率；fh 是设计的滤波器的最高频率（fl 和 fh 都需要用 fs 进行归一化）；p 是设计的 Mel 滤波器的个数；n 是一帧 FFT 后数据的长度；w 是窗函数（‘t’代表三角窗；‘n’代表汉宁窗；‘m’代表汉明窗）。

输出参数：h 是滤波器的频域响应，是一个 $p\times(n/2+1)$ 的数组，p 为滤波器个数，每个滤波器的响应曲线长 n/2+1，相当于取正频率的部分。

4、MFCC 系数的计算

MFCC 特征参数提取原理框图如图 3-13 所示。

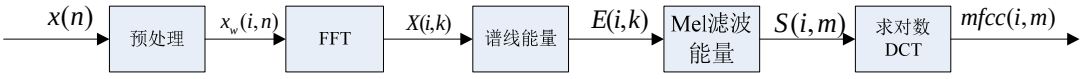

图 3-13 MFCC 特征参数提取原理框图

（1）预处理

预处理包括预加重、分帧、加窗函数。

预加重：声门脉冲的频率响应曲线接近于一个二阶低通滤波器，而口腔的辐射响应也接近于一个一阶高通滤波器。预加重的目的是为了补偿高频分量的损失，提升高频分量。预加重的滤波器常设为

$$H(z)=1-az^{-1} \tag{3-48}$$

式中，a 为一个常数。

分帧处理：由于一个语音信号是一个准稳态的信号，把它分成较短的帧，在每帧信号中可将其看作稳态信号，可用处理稳态信号的方法来处理。同时，为了使一帧与另一帧之间的参数能较平稳地过渡，在相邻两帧之间互相有部分重叠。

加窗函数：加窗函数的目的是减少频域中的泄漏，将对每一帧语音乘以汉明窗或海宁窗。

语音信号 $x(n)$ 经预处理后为 $x_i(m)$，其中下标 i 表示分帧后的第 i 帧。

（2）快速傅立叶变换

对每一帧信号进行 FFT 变换，从时域数据转变为频域数据：

$$X(i,k)=FFT\left[x_i(m)\right] \tag{3-49}$$

（3）计算谱线能量

对每一帧 FFT 后的数据计算谱线的能量：

$$E(i,k)=\left[X_i(k)\right]^2 \tag{3-50}$$

（4）计算通过 Mel 滤波器的能量

把求出的每帧谱线能量谱通过 Mel 滤波器，并计算在该 Mel 滤波器中的能量。在频域中相当于把每帧的能量谱 $E(i,k)$（其中 i 表示第 i 帧，k 表示频域中的第 k 条谱线）与 Mel 滤波器的频域响应 $H_m(k)$ 相乘并相加：

$$S_i(m)=\sum_{k=0}^{N-1}E(i,k)H_m(k),\ 0\le m<M \tag{3-51}$$

（5）计算 DCT 倒谱

序列 $x(n)$ 的 FFT 倒谱 $\hat{x}(n)$ 为

$$\hat{x}(n)=FT^{-1}\left[\hat{X}(k)\right] \tag{3-52}$$

式中，$\hat{X}(k)=\ln\left\{FT\left[x(n)\right]\right\}=\ln\left\{X(k)\right\}$，$FT$ 和 FT^{-1} 表示傅立叶变换和傅立叶逆变换。序列 $x(n)$ 的 DCT 为

$$X(k)=\sqrt{\frac{2}{N}}\sum_{n=0}^{N-1}C(k)x(n)\cos\left[\frac{\pi(2n+1)k}{2N}\right],\ k=0,1,\cdots,N-1 \qquad (3\text{-}53)$$

式中，参数 N 是序列 $x(n)$ 的长度；$C(k)$ 是正交因子，可表示为

$$C(k)=\begin{cases}\sqrt{2}/2 & k=0\\ 1 & k=1,2,\cdots,N-1\end{cases} \qquad (3\text{-}54)$$

在式（3-4-21）中求取 FFT 的倒谱是把 $X(k)$ 取对数后计算 FFT 的逆变换。而这里求 DCT 的倒谱和求 FFT 的倒谱相类似，把 Mel 滤波器的能量取对数后计算 DCT:

$$mfcc(i,n)=\sqrt{\frac{2}{M}}\sum_{m=0}^{M-1}\log\left[S(i,m)\right]\cos\left[\frac{\pi n(2m-1)}{2M}\right] \qquad (3\text{-}55)$$

式中，$S(i,m)$ 是由式（3-4-20）求出的 Mel 滤波器能量；m 是指第 m 个 Mel 滤波器（共有 M 个）；i 是指第 i 帧；n 是 DCT 后的谱线。

3.4.3 实验要求

1. 实验步骤

运行 Python——>新建 py 文件——>编写 py 程序——>编译并调试。

2、实验要求

1）根据倒谱计算的原理，Python 编程实现 rceps 函数，函数定位为 xh=Nrceps(x)。然后利用 Python 的 cceps 函数和 icceps 函数以及设计的 rceps 函数对一段较短语音进行复倒谱的计算与恢复，图例如图 3-14 所示。

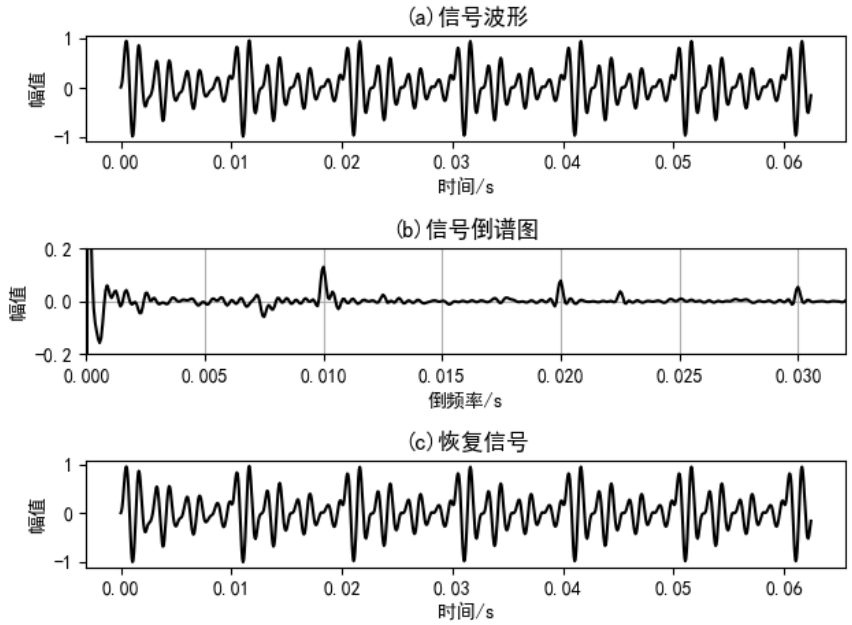

图 3-14 倒谱程序例图

2）根据离散余弦变换的原理，Python 编程实现 DCT 和 IDCT 函数，并求序列 $x(n)=\cos(2\pi fn/f_s)$ 的 DCT 系数（其中，$n\in[0,1000)$，$f_s=1000Hz$，$f=50Hz$），然后仅用幅值大于 5 的系数进行信号重建，并比较重建前后信号的差异。实验结果如图 3-15 所示。

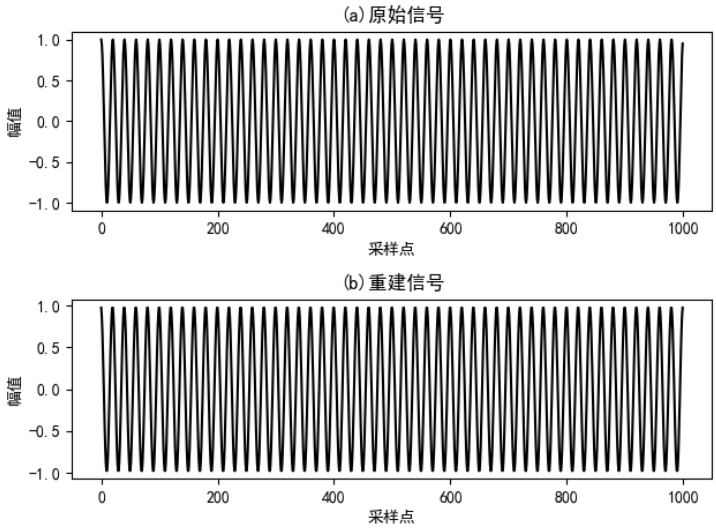

图 3-15 DCT 重建信号对比

3）使用 Python 设计 24 个 Mel 滤波器，其中采样频率为 8000，最低频率为 0，最高频率为 0.5，使用三角函数，图例如图 3-16 所示。

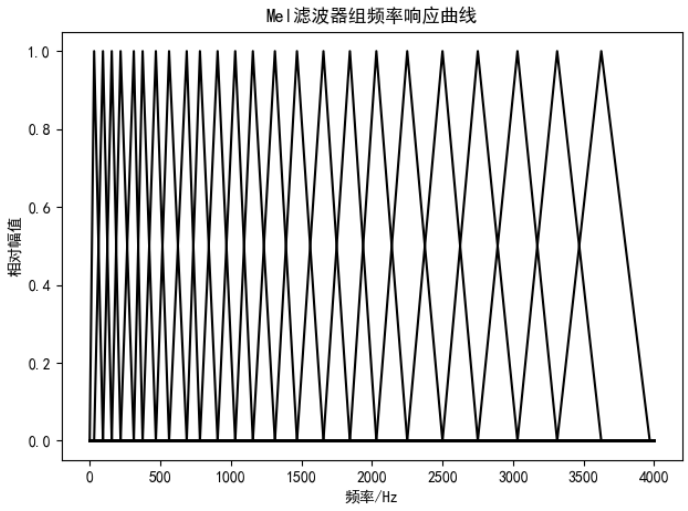

图 3-16 Mel 滤波器组频率响应曲线

4）调用 librosa.feature.mfcc 函数计算一段语音（C3_4_y_4.wav）的 MFCC 系数。主要参数为：fft 点数为 256，mfcc 个数为 24，窗长 256，帧移 128。显示结果如图 3-17 所示。

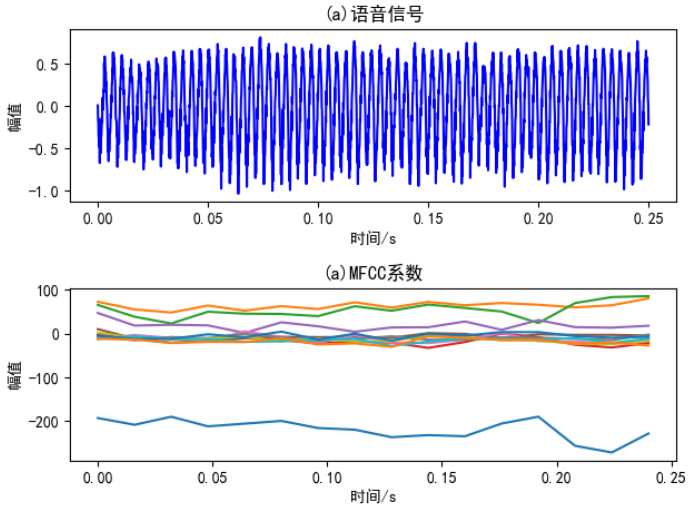

图 3-17 MFCC 计算结果

3.4.4 思考题

1）根据 MFCC 系数计算流程，编写 MFCC 计算函数，并用来计算一段语音的 MFCC 系数。函数定义如下：

ccc=Nmfcc(x,fs,p,frameSize,inc)；

其中，x 是输入语音序列，Mel 滤波器的个数为 p，采样频率为 fs，frameSize 为帧长和 FFT 点数，inc 为帧移；ccc 为 MFCC 参数。

试分析与 librosa.feature.mfcc 的差异。

3.4.5 参考例程

【C3_4_y_4MFCC系数显示】

```
# -*- coding: utf-8 -*-
from speechlib import *
import librosa

(framerate, wave_data) = wavfile.read("C3_4_y_4.wav")
wlen=256
inc=128
num=8
nfft=256
n_dct=24
x = wave_data / max(np.abs(wave_data))
time = np.arange(0, len(wave_data))/ framerate

plt.figure(1)
plt.subplot(211)
plt.plot(time,x,'b')
```

```
plt.title("(a)语音信号")
plt.ylabel("幅值")
plt.xlabel("时间/s")

ccc1 = librosa.feature.mfcc(y=x,
                            n_fft=wlen,
                            sr=framerate,
                            n_mfcc=24,
                            fmax=4000,
                            dct_type=2,
                            hop_length=inc,
                            win_length=wlen)
ccc1=np.transpose(ccc1)
fn=ccc1.shape[0]
cn=ccc1.shape[1]
frameTime=FrameTimeC(fn,wlen,inc,framerate)
plt.subplot(212)
plt.plot(frameTime,ccc1[:,0:int(cn/2)])
plt.title("(a)MFCC系数")
plt.ylabel("幅值")
plt.xlabel("时间/s")
plt.show()
```

3.5 线性预测分析

3.5.1 实验目的

1）了解线性预测分析在语音信号处理中的重要性和必要性；
2）掌握线性预测分析的基本思想；
3）掌握 MATLAB 进行线性预测分析的流程。

3.5.2 实验原理

1、语音信号线性预测分析

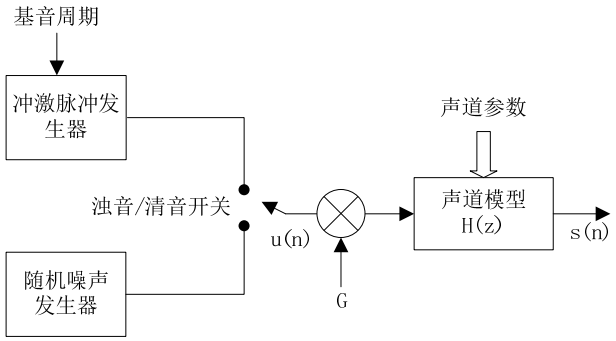

图 3-18 简化的语音产生模型

图 3-18 是简化的语音产生模型，将辐射、声道以及声门激励的全部效应简化为一个时变的数字滤波器来等效，其传递函数为

$$H(z)=\frac{S(z)}{U(z)}=\frac{G}{1-\sum_{i=1}^{p}a_i z^{-i}} \tag{3-56}$$

这种表现形式称为 p 阶线性预测模型，这是一个全极点模型。

此时，$s(n)$ 和 $u(n)$ 间的关系可以用差分方程

$$s(n)=\sum_{i=1}^{p}a_i s(n-i)+Gu(n) \tag{3-57}$$

表示，称系统

$$\hat{s}(n)=\sum_{i=1}^{p}a_i s(n-i) \tag{3-58}$$

为线性预测器。$\hat{s}(n)$ 是 $s(n)$ 的估计值，它由过去 p 个值线性组合得到的，即由 $s(n)$ 过去的值来预测或估计当前值 $s(n)$。式中 $a_i\,(i=1,2,\cdots,p)$ 是线性预测系数。线性预测系数可以通过在某个准则下使预测误差 $e(n)$ 达到最小值的方法来决定，预测误差的表示形式如下：

$$e(n)=s(n)-\hat{s}(n)=s(n)-\sum_{i=1}^{p}a_i s(n-i) \tag{3-59}$$

预测的二次方误差为

$$E=\sum_n e^2(n)=\sum_n[s(n)-\hat{s}(n)]^2=\sum_n[s(n)-\sum_{i=1}^{p}a_i s(n-i)]^2 \qquad (3\text{-}60)$$

为使 E 最小，求 E 对 a_i 的偏导为 0，即

$$\frac{\partial E}{\partial a_j}=0 \quad (1\le j\le p) \qquad (3\text{-}61)$$

则有，

$$\frac{\partial E}{\partial a_j}=2\sum_n s(n)s(n-j)-2\sum_{i=1}^{p}a_i\sum_n s(n-i)s(n-j)=0 \quad (1\le j\le p) \qquad (3\text{-}62)$$

定义 $\phi(j,i)=\sum_n s(n-i)s(n-j)$，则式（3-5-7）可简化为

$$\phi(j,0)=\sum_{i=1}^{p}a_i\phi(j,i) \quad (1\le j\le p) \qquad (3\text{-}63)$$

联立式（3-5-5）、式（3-5-7）和式（3-5-8），可得最小均方误差表示为

$$E=\phi(0,0)-\sum_{i=1}^{p}a_i\phi(0,i) \qquad (3\text{-}64)$$

因此，最小误差有一个固定分量 $\phi(0,0)$ 和一个依赖于预测系数的分量 $\sum_{i=1}^{p}a_i\phi(0,i)$ 构成。为求解最佳预测器系数，必须首先求出 $\phi(j,i)(i,j\in[1,p])$，然后可按照式（3-5-8）进行求解。很显然，$\phi(j,i)$ 的计算及方程组的求解都是十分复杂的。

2、线性预测分析的自相关解法

为了有效地进行线性预测分析，求得线性预测系数有必要用一种高效的方法来求解线性方程组。虽然可以用各种各样的方法来解包含 p 个未知数的 p 个线性方程，但是系数矩阵的特殊性质使得解方程的效率比普通解法的效率要高得多。自相关法是经典解法之一，其原理是在整个时间范围内使误差最小，即设 $s(n)$ 在 $0\le n\le N-1$ 以外等于 0，等同于假设 $s(n)$ 经过有限长度的窗（如矩形窗、海宁窗或汉明窗）的处理。

通常，$s(n)$ 的加窗自相关函数定义为

$$r(j)=\sum_{n=0}^{N-1}s(n)s(n-j) \quad 1\le j\le p \qquad (3\text{-}65)$$

同式（3-5-8）比较可知，$\phi(j,i)$ 等效为 $r(j-i)$。但是由于 $r(j)$ 为偶函数，因此 $\phi(j,i)$ 可表示为

$$\phi(j,i)=r(|j-i|) \qquad (3\text{-}66)$$

此时式（3-5-8）可表示为

$$\sum_{i=1}^{p}a_i r(|j-i|)=r(j) \qquad 1\le j\le p \qquad (3\text{-}67)$$

则最小均方误差改写为

$$E=r(0)-\sum_{i=1}^{p}a_i r(i) \qquad (3\text{-}68)$$

展开式（3-5-11），可得方程组为

$$\begin{bmatrix} r(0) & r(1) & r(2) & \cdots & r(p-1) \\ r(1) & r(0) & r(1) & \cdots & r(p-2) \\ r(2) & r(1) & r(0) & \cdots & r(p-3) \\ \vdots & \vdots & \vdots & \cdots & \vdots \\ r(p-1) & r(p-2) & r(p-3) & \cdots & r(0) \end{bmatrix} \begin{bmatrix} a_1 \\ a_2 \\ a_3 \\ \vdots \\ a_p \end{bmatrix} = \begin{bmatrix} r(1) \\ r(2) \\ r(3) \\ \vdots \\ r(p) \end{bmatrix} \tag{3-69}$$

式（3-5-14）左边为相关函数的矩阵，以对角线为对称，其主对角线以及和主对角线平行的任何一条斜线上所有的元素相等。这种矩阵称为托普利兹（Toeplitz）矩阵，而这种方程称为 Yule-Walker 方程。对于式（3-5-14）的矩阵方程无需像求解一般矩阵方程那样进行大量的计算，利用托普利兹矩阵的性质可以得到求解这种方程的一种高效方法。

这种矩阵方程组可以采用递归方法求解，其基本思想是递归解法分布进行。在递推算法中，最常用的是莱文逊-杜宾（Levinson-Durbin）算法（如图 3-19 所示）。

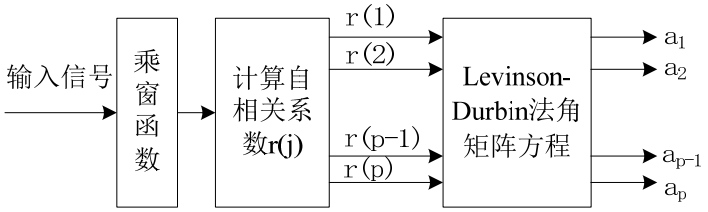

图 3-19 自相关解法

算法的过程和步骤为：

$$\text{当 } i=0 \text{ 时，} E_0 = r(0)\text{，} a_0 = 1\text{；} \tag{3-70}$$

② 对于第 i 次递归 $(i = 1, 2, \cdots, p)$：

i $$k_i = \frac{1}{E_{i-1}} \left[r(i) - \sum_{j=1}^{i-1} a_j^{i-1} r(j-i) \right] \tag{3-71}$$

ii $$a_i^{(i)} = k_i \tag{3-72}$$

iii 对于 j=1 到 i-1

$$a_j^{(i)} = a_j^{(i-1)} - k_i a_{i-j}^{(i-1)} \tag{3-73}$$

iv $$E_i = (1 - k_i^2) E_{i-1} \tag{3-74}$$

③ 增益 G 为

$$G = \sqrt{E_p} \tag{3-75}$$

通过对式（3-5-16）~式（3-5-18）进行递推求解，可获得最终解为

$$a_i = a_j^{(p)} \quad 1 \le j \le p \tag{3-76}$$

由式（3-5-19）可得

$$E_p = r(0) \prod_{i=1}^{p} (1 - k_i^2) \tag{3-77}$$

由式(3-5-22)可知，最小均方误差 E_p 一定要大于 0，且随着预测器阶数的增加而减小。因此每一步算出的预测误差总是小于前一步的预测误差。这就表明，虽然预测器的精度会随着阶数的增加而提高，但误差永远不会消除。由式(3-5-22)还可知，参数 k_i 一定满足

$$|k_i|<1, \quad 1\le i\le p \tag{3-78}$$

由递归算法可知，每一步计算都与 k_i 有关，说明这个系数具有特殊的意义，通常称之为反射系数或偏相关系数。可以证明，它就是多项式 $A(z)$ 的根在单位圆内的充分必要条件，因此它可以保证系统 $H(k)$ 的稳定性。

3、线性预测的其它参数

用线性预测分析法求得的是一个全极点模型的传递函数。在语音产生模型中，这一全极点模型与声道滤波器的假设相符合，而形式上是一自回归滤波器。用全极点模型所表征的声道滤波器，除预测系数 $\{a_i\}$ 外，还有其他不同形式的滤波器参数。这些参数一般可由线性预测系数推导得到，但各有不同的物理意义和特性。在对语音信号做进一步处理时，为了达到不同的应用目的时，往往按照这些特性来选择某种合适的参数来描述语音信号。

3.1 预测误差及其自相关函数

由式（3-5-4）可知，预测误差为

$$e(n)=s(n)-\sum_{i=1}^{p}a_i s(n-i) \tag{3-79}$$

而预测误差的自相关函数为

$$R_e(m)=\sum_{n=0}^{N-1-m}e(n)e(n+m) \tag{3-80}$$

3.2 反射系数和声道面积

反射系数 $\{k_i\}$ 在低速率语音编码、语音合成、语音识别和说话人识别等许多领域都是非常重要的特征参数。由式（3-5-18）可得：

$$\begin{cases} a_j^{(i)}=a_j^{(i-1)}-k_i a_{i-j}^{(i-1)} \\ a_{i-j}^{(i)}=a_{i-j}^{(i-1)}-k_i a_j^{(i-1)} \end{cases} \qquad j=1,\cdots,i-1 \tag{3-81}$$

进一步推导，可得：

$$a_j^{(i-1)}=(a_j^{(i)}+a_j^{(i)}a_{i-j}^{(i)})\big/(1-k_i^2) \qquad j=1,\cdots,i-1 \tag{3-82}$$

由线性预测系数 $\{a_i\}$ 可递推出反射系数 $\{k_i\}$，即

$$\begin{cases} a_j^{(p)}=a_j & j=1,2,\cdots,p \\ k_i=a_i^{(i)} & \\ a_j^{(i-1)}=(a_j^{(i)}+a_j^{(i)}a_{i-j}^{(i)})\big/(1-k_i^2) & j=1,\cdots,i-1 \end{cases} \tag{3-83}$$

反射系数的取值范围为$[-1,1]$，这是保证相应的系统函数稳定的充分必要条件。从声学理论可知，声道可以被模拟成一系列截面积不等的无损声道的级联。反射系数$\{k_i\}$反映了声波在各管道边界处的反射量，有：

$$k_i = \frac{A_{i+1} - A_i}{A_{i+1} + A_i} \tag{3-84}$$

式中，A_i是第i节声管的面积函数。式（3-5-30）经变换后，可得声管模型各节的面积比为：

$$\frac{A_i}{A_{i+1}} = \frac{(1-k_i)}{(1+k_i)} \tag{3-85}$$

3.3 线性预测的频谱

由式（3-5-1）可知，一帧语音信号$x(n)$模型可化为一个p阶的线性预测模型。当$z = e^{j\omega}$时，能得到线性预测系数的频谱（令$G=1$）：

$$H(e^{j\omega}) = \frac{1}{1 - \sum_{n=1}^{p} a_n z^{-j\omega n}} \tag{3-86}$$

根据语音信号的数字模型，在不考虑激励和辐射时，$H(e^{j\omega})$即$X(e^{j\omega})$的频谱的包络谱。线性预测系数的频谱勾画出了 FFT 频谱的包络，反映了声道的共振峰的结构。

3.4 线性预测倒谱

语音信号的倒谱可以通过对信号做傅里叶变换，取模的对数，再求傅里叶逆变换得到。由于频率响应$H(e^{j\omega})$反映声道的频率响应和被分析信号的谱包络，因此用$\log|H(e^{j\omega})|$做傅里叶逆变换求出的线性预测倒谱系数（Linear Prediction Cepstrum Coefficient, LPCC），其可看做是原始信号短时倒谱的一种近似。

通过线性预测分析得到的合成滤波器的系统函数为$H(z) = 1\Big/(1 - \sum_{i=1}^{p} a_i z^{-i})$，其冲激响应为$h(n)$。下面求$h(n)$的倒谱$\hat{h}(n)$，首先根据同态处理法，有

$$\hat{H}(z) = \log H(z) \tag{3-87}$$

因为$H(z)$是最小相位的，即在单位圆内是解析的，所以$\hat{H}(z)$可以展开成级数形式，即

$$\hat{H}(z) = \sum_{n=1}^{+\infty} \hat{h}(n) z^{-n} \tag{3-88}$$

也就是说，$\hat{H}(z)$ 的逆变换 $\hat{h}(n)$ 是存在的。设 $\hat{h}(0)=0$，将式（3-5-34）两边同时对 z^{-1} 求导，得

$$\frac{\partial}{\partial z^{-1}}\log\frac{1}{1-\sum_{i=1}^{p}a_i z^{-i}}=\frac{\partial}{\partial z^{-1}}\sum_{n=1}^{+\infty}\hat{h}(n)z^{-n} \tag{3-89}$$

得到

$$\sum_{n=1}^{+\infty}n\hat{h}(n)z^{-n+1}=\frac{\sum_{i=1}^{p}ia_i z^{-i+1}}{1-\sum_{i=1}^{p}a_i z^{-i}} \tag{3-90}$$

有

$$(1-\sum_{i=1}^{p}a_i z^{-i})\sum_{n=1}^{+\infty}n\hat{h}(n)z^{-n+1}=\sum_{i=1}^{+\infty}ia_i z^{-i+1} \tag{3-91}$$

令式（3-5-37）等号两边 z 的各次幂前系数分别相等，得到 $\hat{h}(n)$ 和 a_i 间的递推关系：

$$\hat{h}(1)=a_1 \tag{3-92}$$

$$\hat{h}(n)=a_n+\sum_{i=1}^{n-1}(1-\frac{i}{n})a_i\hat{h}(n-i)\qquad 1<n\le p \tag{3-93}$$

$$\hat{h}(n)=\sum_{i=1}^{p}(1-\frac{i}{n})a_i\hat{h}(n-i)\qquad n>p \tag{3-94}$$

按式（3-5-38）~式（3-5-40）可直接从预测系数 $\{a_i\}$ 求得倒谱 $\hat{h}(n)$。这个倒谱系数是根据线性预测模型得到的，又利用线性预测中声道系统函数 H(z)的最小相位特性，因此避免了一般同态处理中求复对数的麻烦。

3.5.3 实验要求

1）根据莱文逊-杜宾自相关法求线性预测系数的原理，编写函数，进行测试，测试语音文件为 C3_5_y.wav。

函数定义如下：

名称：lpc_coeff

功能：用莱文逊-杜宾自相关法计算线性预测系数。

调用格式：[ar,G]=lpc_coeff(s,p)

说明：输入参数 s 是一帧数据；p 是线性预测阶数。输出参数 ar 是按式（3-5-19）计算得到的预测系数 $\{a_i\}$ $(i=1,2,\cdots,p)$，共得 p 个预测系数；G 是按式（3-5-20）计算得到的增益系数。

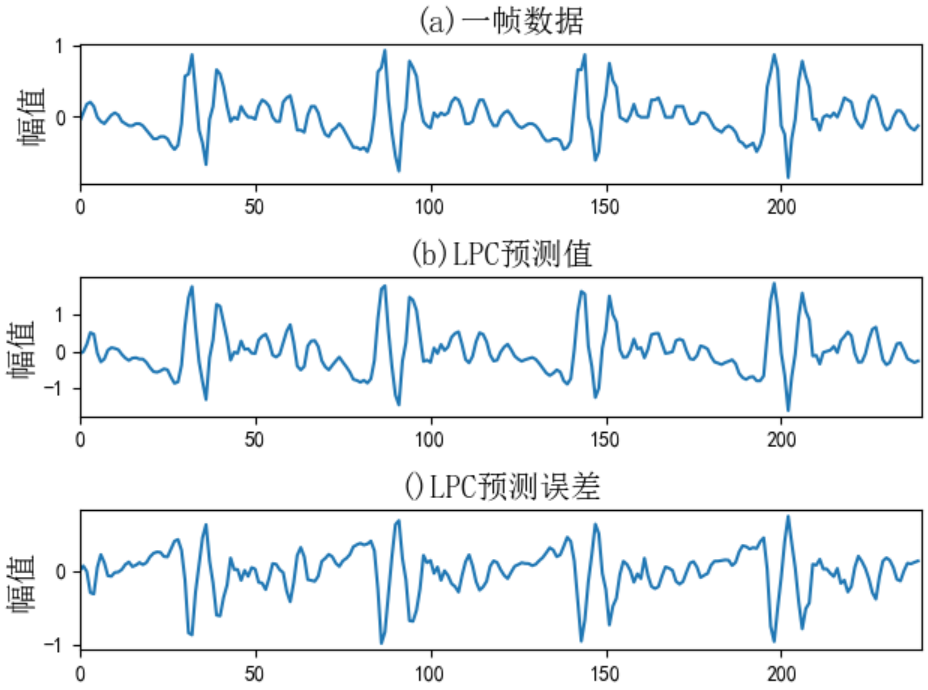

图 3-20 线性预测系数对比

2）编写求取 LPC 预测系数的复频谱函数，并与 FFT 频谱进行对比。测试效果如图 3-21 所示，测试语音文件为 C3_5_y.wav。LPC 预测系数的复频谱函数的定义如下：

名称：lpcff

功能：计算线性预测系数的复频谱。

调用格式：ff=lpcff(ar,np)

说明：ar 是线性预测系数，np 是 FFT 阶数；输出 ff 是线性预测系数的复频谱。

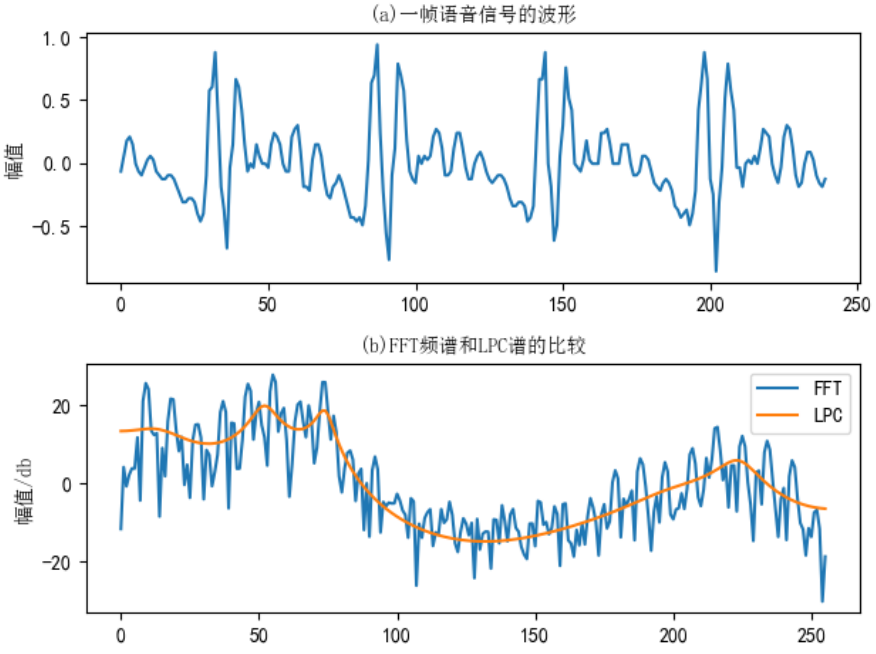

图 3-21 LPC 预测系数的复频谱与 FFT 频谱

3）编程实现由预测系数求取线性预测倒谱系数，并比较与 rceps 函数的区别，测试效果如图 3-22 所示，测试语音文件为 C3_5_y.wav。

名称：lpc_lpccm

功能：由预测系数 ai 求 LPCC。

调用格式：lpcc=lpc_lpccm(ar,n_lpc,n_lpcc)

说明：输入参数 ar 是线性预测系数；n_lpc 是预测系数的长度；n_lpcc 是 LPC 倒谱的长度；输出参数 lpcc 是线性预测倒谱系数。

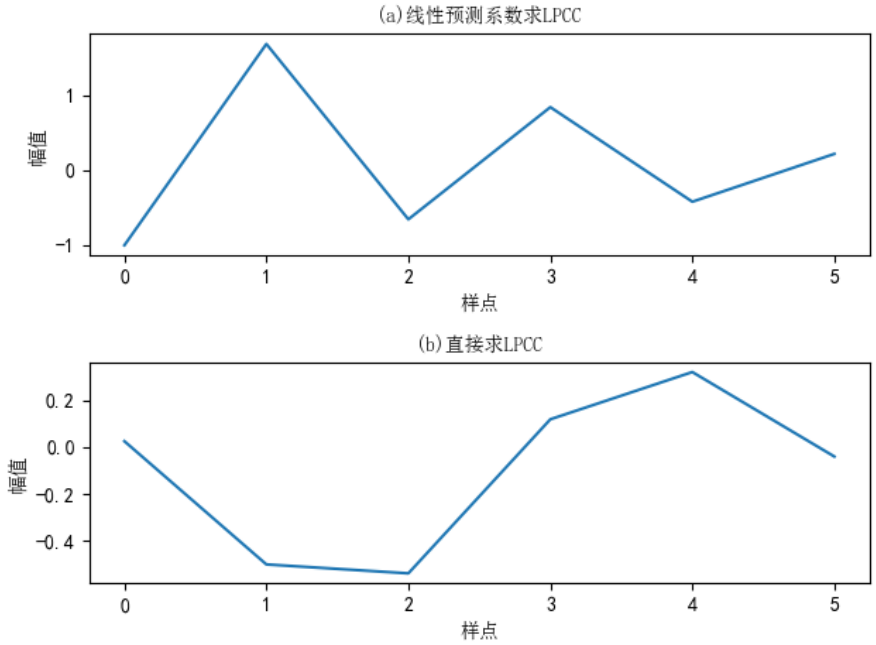

图 3-22 LPCC 比较

3.5.4 思考题

编程实现预测系数与反射系数间的转换和反射系数与声管面积间的转换。函数定义如下：

（1） 由预测系数求反射系数

名称：lpcar2rf

功能：已知预测系数计算出反射系数

调用格式：rf=lpcar2rf（ar）

说明：ar 是预测系数；rf 是反射系数。

（2） 由反射系数求出预测系数

名称：lpcrf2ar

功能：已知反射系数计算出预测系数

调用格式：[ar,arp,aru,g]=lpcrf2ar(rf)

说明：rf 是反射系数；ar 是预测系数；arp 是压力传递函数；aru 是体积速度的传递函数；g 是增益。

（3） 由反射系数求声管面积比

名称：lpcrf2ao

功能：已知反射系数计算出归正化的声管面积。

调用格式：ao=lpcrf2ao(rf)

说明：rf 是反射系数；ao 是声管面积比。

（4） 由声管面积求反射系数

名称：lpcao2rf

功能：已知归正化的声管面积比计算出反射系数。

调用格式：rf=lpcao2rf（ao）

说明：ao 是声管面积比；rf 是反射系数。

3.5.5 参考例程

【C3_5_y_2 LPC预测系数的复频谱函数】

```
# -*-coding:utf-8-*-
from speechlib import *

(framerate, wave_data) = wavfile.read("C3_5_y.wav")

wave_data=wave_data/max(abs(wave_data))
L = 240                                    # 设定帧长
x = wave_data[8000:8000 + L]       # 给定一帧数据
p = 12                                     # 设定线性预测阶数
ar, G = lpc_coeff(x, p)               # 进行线性预测
nfft = 512                                 # 设定FFT点数
W2 = nfft // 2                           # FFT阶数
m = np.arange(W2)
Y = np.fft.fft(x, nfft)               # FFT频谱
Y1 = lpcff(ar, W2)                      # 线性预测系数的复频谱

plt.figure(1)
plt.subplot(2, 1, 1)
plt.plot(x)
plt.title('(a)一帧语音信号的波形')
plt.ylabel('幅值')

plt.subplot(2, 1, 2)
plt.plot(m, 20 * np.log10(np.abs(Y[m])))
plt.plot(m, 20 * np.log10(np.abs(Y1[m])))
plt.title('(b)FFT频谱和LPC谱的比较')
plt.legend(labels=('FFT','LPC'))
plt.ylabel('幅值/db')
plt.show()
```

3.6 线谱对转换实验

3.6.1 实验目的

1）了解线谱对的定义和特点；
2）掌握线性预测与线谱对转换的基本思想；
3）能编程实现线性预测系数与线谱对的相互转换。

3.6.2 实验原理

1、线谱对的定义与特点

根据线性预测的原理可知，$A(z)=1-\sum_{i=1}^{p}a_i z^{-i}$ 为线性预测误差滤波器，其倒数 $H(z)=1/A(z)$ 为线性预测合成滤波器。该滤波器常被用于重建语音，但是当直接对线性预测系数 a_i 进行编码时，$H(z)$ 的稳定性就不能得到保证。由此引出了许多与线性预测等价的表示方法，以提高线性预测的鲁棒性，如线谱对（Line Spectrum Pair，LSP）就是线性预测的一种等价表示形式。LSP 最早由 Itakura 引入的，但是直到人们发现利用 LSP 在频域对语音进行编码，比其它变换技术更能改善编码效率时，LSP 才被重视。由于 LSP 能够保证线性预测滤波器的稳定性，其小的系数偏差带来的谱误差也只是局部的，且 LSP 具有良好的量化特性和内插特性，因而已经在许多编码系统中得到成功的应用。LSP 分析的主要缺点是运算量较大。

LSP 作为线性预测参数的一种表示形式，可通过求解 p+1 阶对称和反对称多项式的共轭复根得到。其中，p+1 阶对称和反对称多项式表示如下：

$$P(z)=A(z)+z^{-(p+1)}A(z^{-1}) \tag{3-95}$$

$$Q(z)=A(z)-z^{-(p+1)}A(z^{-1}) \tag{3-96}$$

其中，

$$z^{-(p+1)}A(z^{-1})=z^{-(p+1)}-a_1z^{-p}-a_2z^{-p+1}-\ldots-a_pz^{-1} \tag{3-97}$$

可以推出：

$$P(z)=1-(a_1+a_p)z^{-1}-(a_2+a_{p-1})z^{-2}-\ldots-(a_p+a_1)z^{-p}+z^{-(p+1)} \tag{3-98}$$

$$Q(z)=1-(a_1-a_p)z^{-1}-(a_2-a_{p-1})z^{-2}-\ldots-(a_p-a_1)z^{-p}-z^{-(p+1)} \tag{3-99}$$

P(z)、Q(z)分别为对称和反对称的实系数多项式，它们都有共轭复根。可以证明，当 A(z)的根位于单位圆内时，P(z)和 Q(z)的根都位于单位圆上，而且相互交替出现。如果阶数 P 是偶数，则 P(z)和 Q(z)各有一个实根，其中 P(z)有一个根 z=-1，Q(z)有一个根 z=1.如果阶数 p 是奇数，则 P(z)有两个根 z=-1，z=1，Q(z)没有实根。此处假定 p 是偶数，这样 P(z)和 Q(z)

各有 p/2 个共轭复根位于单位圆上，共轭复根的形式为 $z_i = e^{\pm jw_i}$，设 p(z)的零点为 $e^{\pm jw_i}$，Q(z)的零点为 $e^{\pm j\theta_i}$，则满足：

$$0 < \omega_1 < \theta_1 < ... < \omega_{p/2} < \theta_{p/2} < \pi \qquad (3\text{-}100)$$

其中，ω_i 和 θ_i 分别为 P(z)和 Q(z)的第 i 个根。

$$P(z) = (1+z^{-1})\prod_{i=1}^{p/2}(1-z^{-1}e^{j\omega_i})(1-z^{-1}e^{-j\omega_i}) = (1+z^{-1})\prod_{i=1}^{p/2}(1-2\cos\omega_i z^{-1}+z^{-2}) \qquad (3\text{-}101)$$

$$Q(z) = (1-z^{-1})\prod_{i=1}^{p/2}(1-z^{-1}e^{j\theta_i})(1-z^{-1}e^{-j\theta_i}) = (1-z^{-1})\prod_{i=1}^{p/2}(1-2\cos\theta_i z^{-1}+z^{-2}) \qquad (3\text{-}102)$$

式中，$\cos\omega_i$ 和 $\cos\theta_i$（$i = 1,2,\cdots,p/2$）是 LSP 系数在余弦域的表示；ω_i 和 θ_i 则是与 LSP 系数对应的线谱频率（Line Spectrum Frequency，LSF）。

由于 LSP 参数成对出现，且反应信号的频谱特性，因此称为线谱对。LSF 就是线谱对分析所要求解的参数。

LSP 参数的特性包括：

1．LSP 参数都在单位圆上且降序排列。

2．与 LSP 参数对应的 LSF 升序排列，且 $P(z)$和 $Q(z)$的根相互交替出现，这可使与 LSP 参数对应的 LPC 滤波器的稳定性得到保证。上述特性保证了在单位圆上，任何时候 $P(z)$和 $Q(z)$不可能同时为零。

3．LSP 参数具有相对独立的性质。如果某个特定的 LSP 参数中只移动其中任意一个线谱频率的位置，那么它所对应的频谱只在附近与原始语音频谱有差异，而在其它 LSP 频率上则变化很小。这样有利于 LSP 参数的量化和内插。

4．LSP 参数能够反映声道幅度谱的特点，在幅度大的地方分布较密，反之较疏。这样就相当于反映出了幅度谱中的共振峰特性。

按照线性预测分析的原理，语音信号的谱特性可以由 LPC 模型谱来估计，将式（3-6-1）和式（3-6-2）相加，可得：

$$A(z) = \frac{1}{2}[P(z)+Q(z)] \qquad (3\text{-}103)$$

此时，功率谱可以表示为：

$$\begin{aligned}\left|H(e^{j\omega})\right|^2 &= \frac{1}{\left|A(e^{j\omega})\right|^2} = 4\left|P(e^{j\omega})+Q(e^{j\omega})\right|^{-2} \\ &= 2^{-p}[\sin^2(\omega/2)\prod_{i=1}^{p/2}(\cos\omega-\cos\theta_i)^2+\cos^2(\omega/2)\prod_{i=1}^{p/2}(\cos\omega-\cos\omega_i)^2]^{-1}\end{aligned} \qquad (3\text{-}104)$$

由此可见，LSP 分析是用 p 个离散频率的分布密度来表示语音信号谱特性的一种方法，即在语音信号幅度谱较大的地方 LSP 分布较密，反之较疏。

5．相邻帧 LSP 参数之间都具有较强的相关性，便于语音编码时帧间参数的内插。

2、LPC 到 LSP 参数的转换

在进行语音编码时，要对 LPC 进行量化和内插，就需要将 LPC 转换为 LSP 参数，为计算方便，可将 LSP 参数无关的两个实根去掉，得到如下多项式：

$$P'(z) = \frac{P(z)}{(1+z^{-1})} = \prod_{i=1}^{p/2}(1-z^{-1}e^{j\omega})(1-z^{-1}e^{-j\omega}) = \prod_{i=1}^{p/2}(1-2\cos\omega_i z^{-1}+z^{-2}) \qquad (3\text{-}105)$$

$$Q'(z)=\frac{Q(z)}{(1-z^{-1})}=\prod_{i=1}^{p/2}(1-z^{-1}e^{j\theta_i})(1-z^{-1}e^{-j\theta_i})=\prod_{i=1}^{p/2}(1-2\cos\theta_i z^{-1}+z^{-2}) \quad (3\text{-}106)$$

从 LPC 到 LSP 参数的转换过程，其实就是求解式（3-6-1）和式（3-6-2）等于零时的 $\cos\omega_i$ 和 $\cos\theta_i$ 的值，可采用下述几种方法求解：

1）代数方程式求解

由式（3-6-11）可知，等式右边可进一步表示为

$$\begin{aligned}1-2\cos\omega_i z^{-1}+z^{-2}&=2z^{-1}(0.5z-\cos\omega_i+0.5z^{-1})\\&=2z^{-1}[0.5(z+z^{-1})-\cos\omega_i]\end{aligned} \quad (3\text{-}107)$$

令 $z=e^{j\omega}$，则由 $e^{j\omega}=\cos\omega+j\sin\omega$，可得 $z+z^{-1}=2\cos\omega=2x$。因此，式（3-6-1）和式（3-6-2）就是关于 x 的一对 p/2 次代数方程式，其系数决定于 $a_i(i=1,2,\cdots,p)$，且 a_i 是已知的，可以用牛顿迭代法来求解。

2）离散傅立叶变换方法

对 $P'(z)$ 和 $Q'(z)$ 的系数求离散傅立叶变换，得到 $z_k=\exp(-\frac{jk\pi}{N})$，$(k=0,1,\cdots,N-1)$ 各点的值，搜索最小值的位置，即是零点所在。由于除了 0 和 π 之外，总共有 p 个零点，而且 $P'(z)$ 和 $Q'(z)$ 的根是相互交替出现的，因此只要很少的计算量即可解得，其中 N 的取值取 64～128 就可以。

3）切比雪夫多项式求解

用切比雪夫多项式估计 LSP 系数，可直接在余弦域得到。$z=e^{j\omega}$ 时，$P'(z)$ 和 $Q'(z)$ 可写为：

$$P'(z)=2e^{-j\frac{p}{2}\omega}C(x) \quad (3\text{-}108)$$

$$Q'(z)=2e^{-j\frac{p}{2}\theta}C(x) \quad (3\text{-}109)$$

此处，

$$C(x)=T_{\frac{p}{2}}(x)+f(1)T_{\frac{p}{2}-1}(x)+f(2)T_{\frac{p}{2}-2}(x)+\cdots+f(\frac{p}{2}-1)T_1(x)+f(\frac{p}{2})\Big/2 \quad (3\text{-}110)$$

其中，$T_m(x)=\cos mx$ 是 m 阶的切比雪夫多项式；$f(i)$ 是由递推关系计算得到的 $P'(z)$ 和 $Q'(z)$ 的每个系数。由于，$P'(z)$ 和 $Q'(z)$ 是对称和反对称的，所以每个多项式只计算前 5 个系数即可。用下面的递推关系可得：

$$\begin{cases}f_1(i+1)=a_{i+1}+a_{p-i}-f_1(i)\\f_2(i+1)=a_{i+1}-a_{p-i}+f_2(i)\end{cases}\quad i=0,1,\cdots,p/2 \quad (3\text{-}111)$$

其中，$f_1(0)=f_2(0)=1.0$。

多项式 $C(x)$ 在 $x=\cos\omega$ 时的递推关系是：

$$\text{for} \quad k=\frac{p}{2}-1 \quad to \quad 1$$

$$\lambda_k = 2x\lambda_{k+1} - \lambda_{k+2} + f(\frac{p}{2}-k)$$

end

$$C(x) = x\lambda_1 - \lambda_2 + f(\frac{p}{2}) \Big/ 2$$

其中初始值 $\lambda_{p/2}=1$， $\lambda_{p/2+1}=0$ 。

4）其它方法

将 0～π 之间均分为 60 个点，以这 60 个点的频率值代入式（3-6-1）和式（3-6-2），检查它们的符号变化，在符号变化的两点之间均分为 4 份，再将这三个点频率值代入式（3-6-1）和式（3-6-2），符号变化的点即为所求的解。这种方法误差略大，计算量较大，但程序实现容易。

3.LSP 参数到 LPC 的转换

LSP 系数被量化和内插后，应再转换为预测系数 $a_i(i=1,2,...,p)$ 。已知量化和内插的 LSP 参数 $q_i(i=1,2,...,p)$ ，可用式（3-6-1）和式（3-6-2）来计算 $P'(z)$ 和 $Q'(z)$ 的系数 $p'(i)$ 和 $q'(i)$ 。其中， $p'(i)$ 可通过以下的递推关系获得：

for $i=1$ to $p/2$

$p'(i) = -2q_{2i-1}p'(i-1) + 2p'(i-2)$

for $j=i-1$ to 1

$p'(j) = p'(j) - 2q_{2i-1}p'(j-1) + p'(j-2)$

end

end

其中 $q_{2i-1}=\cos\omega_{2i-1}$，初始值 $p'(0)=1$，$p'(-1)=0$。把上面递推关系中的 q_{2i-1} 替换为 q_{2i}，就可以得到 $q'(i)$ 。

一旦得出系数 $p'(i)$ 和 $q'(i)$，就可以得到 $P'(z)$ 和 $Q'(z)$， $P'(z)$ 乘以 $(1+z^{-1})$ 得到 $P(z)$，$Q'(z)$ 乘以 $(1-z^{-1})$ 得到 $Q(z)$，即

$$\begin{cases} p_1(i) = p'(i) + p'(i-1), i=1,2,...,p/2 \\ q_1(i) = q'(i) + q'(i-1), i=1,2,...,p/2 \end{cases} \qquad (3\text{-}112)$$

最后得到预测系数为

$$a_i = \begin{cases} 0.5p_i(i) + 0.5q_1(i) & i=1,2,...,p/2 \\ 0.5p_i(p+1-i) - 0.5q_1(p+1-i) & i=p/2+1,...,p \end{cases} \qquad (3\text{-}113)$$

3.6.3 实验要求

1.实验步骤

运行 Python——>新建 py 文件——>编写 py 程序——>编译并调试。

2、实验要求

1）根据 LPC 到 LSP 参数和 LSP 参数到 LPC 的转换原理，编写函数，并基于测试语音 C3_6_y.wav 比较转换前后的线性预测谱。函数定义如下：

（a）名称：lpctolsf

功能：把 LPC 系数转换成线谱频率 LSF。

调用格式：lsf=lpctolsf (a)

说明：a 是预测系数，当预测为 p 阶时，要输入 p+1 个值。lsf 是 LSP 的参数，将得到 p 个数值，奇数项是 $P(z)$的根，偶数项式是 $Q(z)$的根，所得 lsf 是角频率。

（b）名称：lsftolpc

功能：将线谱频率 LSF 转换为 LPC 系数。

调用格式：a=lsftolpc(lsf)

说明：lsf 是 LSP 的参数，预测为 p 阶时要输入 p 个值，奇数项是 $P(z)$的根，偶数项式是 $Q(z)$的根，lsf 单位是角频率。a 是预测系数，将输出 p+1 个 a 值。

仿真效果如图 3-6-1 所示。

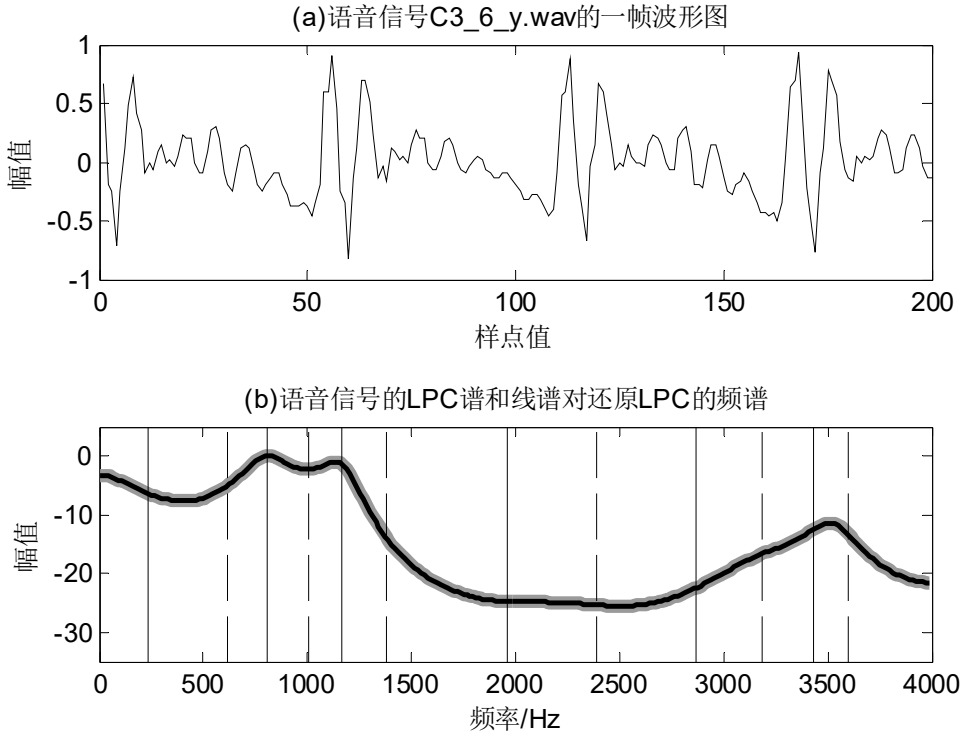

图 3-23 仿真效果图

3.6.4 思考题

1）添加适当噪声，观察参数转换的效果。

3.6.5 参考例程

```
# 将线谱频率LSF转换为LPC系数
import numpy as np
```

```
def lsftolpc(lsf):
    """
        线谱频率LSF转换为LPC系数函数
        :param lfs:LFS参数
        :return lpc:LPC参数
        """
    # 如果线谱频率lsf是复数，则返回错误信息
    if isinstance(lsf,complex):
        print('Line spectral frequencies must be real.')
    # 如果线谱频率lsf不在O～pi范围，则返回错误信息
    if np.max(lsf)>np.pi or np.min(lsf)<0:
        print('Line spectral frequencies must be between 0 and pi.')

    lsf = lsf[:]            # 将lsf转换为列向量
    p = np.size(lsf)        # lsf阶次为p
    z = np.exp(complex(0, lsf))   #用lsf形成零点
    rP = z[1:2:]                  # 把奇次z(1)、z(3)到z(p-1)赋给rP
    rQ = z[2:2:]                  # 把偶次z(2)、z(4)到z(p)赋给rQ
    # 考虑共轭复根
    rQ = np.array([rQ], [np.conj(rQ)])          # 把rQ的共轭复根赋上
    rP = np.array([rP], [np.conj(rP)])          # 把rP的共轭复根赋上
    # 构成多项式P和Q，注意必须是实系数
    Q = np.poly(rQ)
    P = np.poly(rP)
    # 考虑z=1和z=-1以形成对称和反对称多项式
    if divmod(p,2):
        # 如果是奇数阶次，则z=+l和z=-1都是Q1(z)的根
        Q1 = np.convolve(Q,np.array([1, 0, -1]))
        P1 = P
    else:
        # 如果是偶数阶次，z=-1是对称多项式P1(z)的根，z=1是反对称多项式Ql(z)的
根
        Q1 = np.convolve(Q, np.array([1, -1]))
        P1 = np.convolve(P, np.array([1, 1]))
    # 按式(4-5-8)由P1和Q1求解LPC系数
    a = .5 * (P1+Q1)
    a[-1] = []                    # 最后一个系数是O，不返回
    return a
```

4. 语音信号特征提取实验

4.1 语音端点检测实验

4.1.1 实验目的

1）了解语音端点检测的重要性和必要性；
2）掌握基于双门限法、相关法、谱熵法、比例法的语音端点检测原理；
3）编程实现基于双门限法、相关法、谱熵法、比例法的语音端点检测函数。

4.1.2 实验原理

1、基于双门限法的端点检测原理

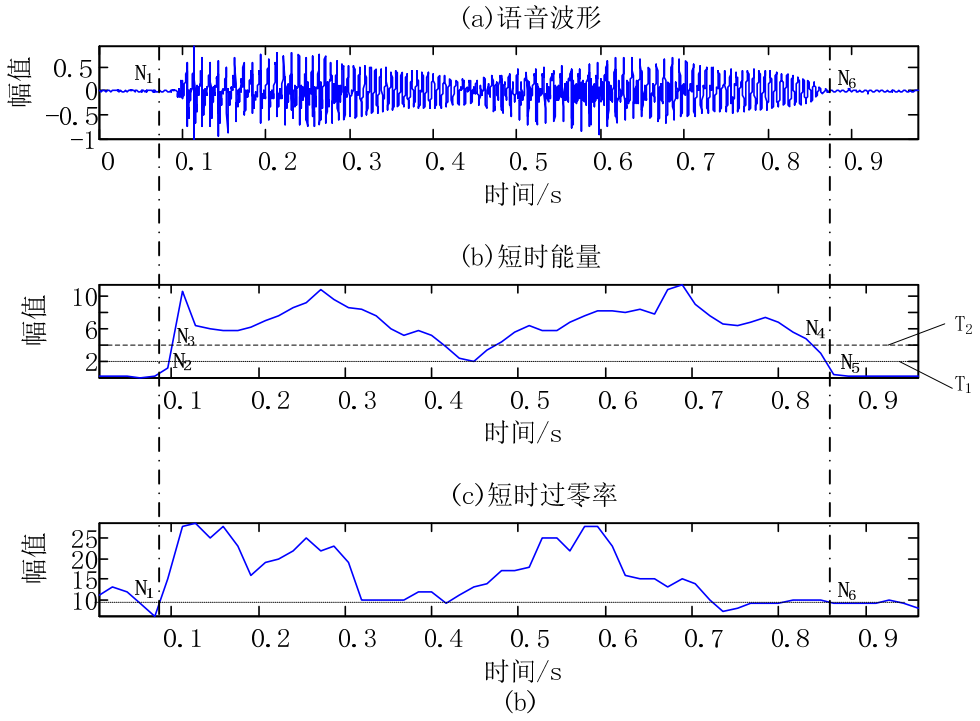

图 4-1 双门限法端点检测的二级判决示意图

语音端点检测本质上是根据语音和噪声的相同参数所表现出的不同特征来进行区分。传统的短时能量和过零率相结合的语音端点检测算法利用短时过零率来检测清音，用短时能量来检测浊音，两者相配合便实现了信号信噪比较大情况下的端点检测。算法以短时能量检测为主，短时过零率检测为辅。根据语音的统计特性，可以把语音段分为清音、浊音以及静音（包括背景噪声）三种。

1.1 短时能量

设第 n 帧语音信号 $x_n(m)$ 的短时能量用 E_n 表示，则其计算公式如下：

$$E_n = \sum_{m=0}^{N-1} x_n^2(m) \qquad (4\text{-}1)$$

E_n 是一个度量语音信号幅度值变化的函数，但它有一个缺陷，即它对高电平非常敏感（因为它计算时用的是信号的平方）。

1.2 短时过零率

短时过零率表示一帧语音中语音信号波形穿过横轴（零电平）的次数。对于连续语音信号，过零即意味着时域波形通过时间轴；而对于离散信号，如果相邻的取样值改变符号则称为过零。因此，过零率就是样本改变符号的次数。

定义语音信号 $x_n(m)$ 的短时过零率 Z_n 为：

$$Z_n = \frac{1}{2}\sum_{m=0}^{N-1} |\mathrm{sgn}[x_n(m)] - \mathrm{sgn}[x_n(m-1)]| \qquad (4\text{-}2)$$

式中，sgn[·] 是符号函数，即：

$$\mathrm{sgn}[x] = \begin{cases} 1, & (x \geq 0) \\ -1, & (x < 0) \end{cases} \qquad (4\text{-}3)$$

1.3 双门限法

在双门限算法中，短时能量检测可以较好地区分出浊音和静音。对于清音，由于其能量较小，在短时能量检测中会因为低于能量门限而被误判为静音；短时过零率则可以从语音中区分出静音和清音。将两种检测结合起来，就可以检测出语音段（清音和浊音）及静音段。在基于短时能量和过零率的双门限端点检测算法中首先为短时能量和过零率分别确定两个门限，一个为较低的门限，对信号的变化比较敏感，另一个是较高的门限。当低门限被超过时，很有可能是由于很小的噪声所引起的，未必是语音的开始，当高门限被超过并且在接下来的时间段内一直超过低门限时，则意味着语音信号的开始。

如图 4-1 所示为双门限法的示例，其主要步骤如下：

（1）计算信号的短时能量和短时平均过零率；

（2）根据语音能量的轮廓选取一个较高的门限 T_2，语音信号的能量包络大部分都在此门限之上，这样可以进行一次初判。语音起止点位于该门限与短时能量包络交点 N_3 和 N_4 所对应的时间间隔之外；

（3）根据背景噪声的能量确定一个较低的门限 T_1，并从初判起点往左，从初判终点往右搜索，分别找到能零比曲线第一次与门限 T_1，相交的 2 个点 N_2 和 N_5，于是 N_2N_5 段就是用双门限方法所判定的语音段；

（4）以短时平均过零率为准，从 N_2 点往左和 N_5 往右搜索，找到短时平均过零率低于某阈值 T_3 的两点 N_1 和 N_6，这便是语音段的起止点。

注意：门限值要通过多次实验来确定，门限都是由背景噪声特性确定的。语音起始段的复杂度特征与结束时的有差异，起始时幅度变化比较大，结束时，幅度变化比较缓慢。在进行起止点判决前，通常都要采集若干帧背景噪声并计算其短时能量和短时平均过零率，作为选择 M1 和 M2 的依据。

2、基于相关法的端点检测原理

2.1 短时自相关

自相关函数具有一些性质，如它是偶函数；假设序列具有周期性，则其自相关函数也是同周期的周期函数等。对于浊音语音可以用自相关函数求出语音波形序列的基音周期。此外，在进行语音信号的线性预测分析时，也要用到自相关函数。

语音信号 $x_n(m)$ 的短时自相关函数 $R_n(k)$ 的计算式如下：

$$R_n(k)=\sum_{m=0}^{N-1-k}x_n(m)x_n(m+k)\qquad(0\le k\le K)\tag{4-4}$$

这里 K 是最大的延迟点数。

为了避免语音端点检测过程中受到绝对能量带来的影响，把自相关函数进行归一化处理，即用 $R_n(0)$ 进行归一化，得到

$$R_n(k)=R_n(k)/R_n(0)\qquad(0\le k\le K)\tag{4-5}$$

2.2 自相关函数最大值法

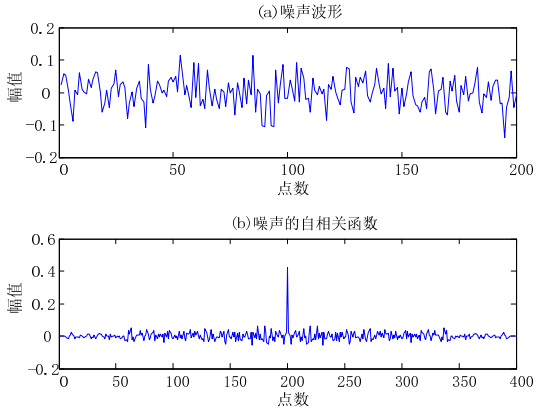

图 4-2 噪声信号的自相关函数

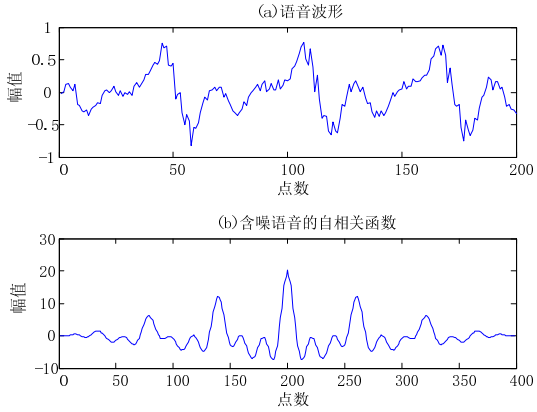

图 4-3 含噪语音的自相关函数

图 4-2 和图 4-3 分别是噪声信号和含噪语音的自相关函数。从图可知，两种信号的自相关函数存在极大的差异，因此可利用这种差别来提取语音端点。根据噪声的情况，设置两个阈值 T1 和 T2，当相关函数最大值大于 T2 时，便判定是语音；当相关函数最大值大于或小于 T1 时，则判定为语音信号的端点。

3、基于谱熵的语音端点检测

3.1 谱熵特征

所谓熵就是表示信息的有序程度。在信息论中，熵描述了随机事件结局的不确定性，即一个信息源发出的信号以信息熵来作为信息选择和不确定性的度量，是由Shannon引用到信息理论中来的。1998年，Shne JL首次提出基于熵的语音端点检测方法，Shne在实验中发现语音的熵和噪声的熵存在较大的差异，谱熵这一特征具有一定的可选性，它体现了语音和噪声在整个信号段中的分布概率。

谱熵语音端点检测方法是通过检测谱的平坦程度，从而达到语音端点检测的目的，经实验研究可知谱熵具有如下特征：

（1）语音信号的谱熵不同于噪声信号的谱熵。

（2）理论上,如果谱的分布保持不变,语音信号幅值的大小不会影响归一化。但实际上,语音谱熵随语音随机性而变化，与能量特征相比,谱熵的变化是很小的。

（3）在某种程度上讲，谱熵对噪声具有一定的稳健性，在相同的语音信号当信噪比降低时，语音信号的谱熵值的形状大体保持不变，这说明谱熵是一个比较稳健性的特征参数。

（4）语音谱熵只与语音信号的随机性有关，而与语音信号的幅度无关，理论上认为只要语音信号的分布不发生变化，那么语音谱熵不会受到语音幅度的影响。另外，由于每个频率分量在求其概率密度函数的时侯都经过了归一化处理,所以从这一方面也证明了语音信号的谱熵只会与语音分布有关，而不会与幅度大小有关。

3.2 谱熵定义

设语音信号时域波形为 $x(i)$，加窗分帧处理后得到的第 n 帧语音信号为 $x_n(m)$，其 FFT 变换表示为 $X_n(k)$，其中下标 n 表示为第 n 帧，而 k 表示为第 k 条谱线。该语音帧在频域中的短时能量为

$$E_n = \sum_{k=0}^{N/2} X_n(k) X_n^*(k) \tag{4-6}$$

式中， N 为 FFT 的长度，只取正频率部分。

而对于某一谱线 k 的能量谱为 $Y_n(k) = X_n(k)X_n^*(k)$，则每个频率分量的归一化谱概率密度函数定义为

$$p_n(k) = \frac{Y_n(k)}{\sum_{l=0}^{N/2} Y_n(l)} = \frac{Y_n(k)}{E_n} \tag{4-7}$$

该语音帧的短时谱熵定义为

$$H_n = -\sum_{l=0}^{N/2} p_n(k) \log p_n(k) \tag{4-8}$$

3.3 基于谱熵的端点检测

由于谱熵语音端点检测方法是通过检测谱的平坦程度，来进行语音端点检测的，为了更好的进行语音端点检测，本文采用语音信号的短时功率谱构造语音信息谱熵，从而更好的对语音段和噪声段进行区分。

其大概检测思路如下：

（1）首先对语音信号进行分帧加窗、取 FFT 变换的点数；

（2）计算出每一帧的谱的能量；

（3）计算出每一帧中每个样本点的概率密度函数；

（4）计算出每一帧的谱熵值；

（5）设置判决门限；

（6）根据各帧的谱熵值进行端点检测。

计算每一帧的谱熵值采用以下公式进行计算的：

$$H(i)=\sum_{i=0}^{N/2-1}P(n,i)*\log[1/P(n,i)] \qquad (4\text{-}9)$$

$H(i)$是第i帧的谱熵，$H(i)$计算是基于谱的能量变化而不是谱的能量，所以在不同水平噪声环境下谱熵参数具有一定的稳健性，但每一谱点的幅值易受噪声的污染进而影响端点检测的稳健性。

4、比例法端点检测

4.1 能零比的端点检测

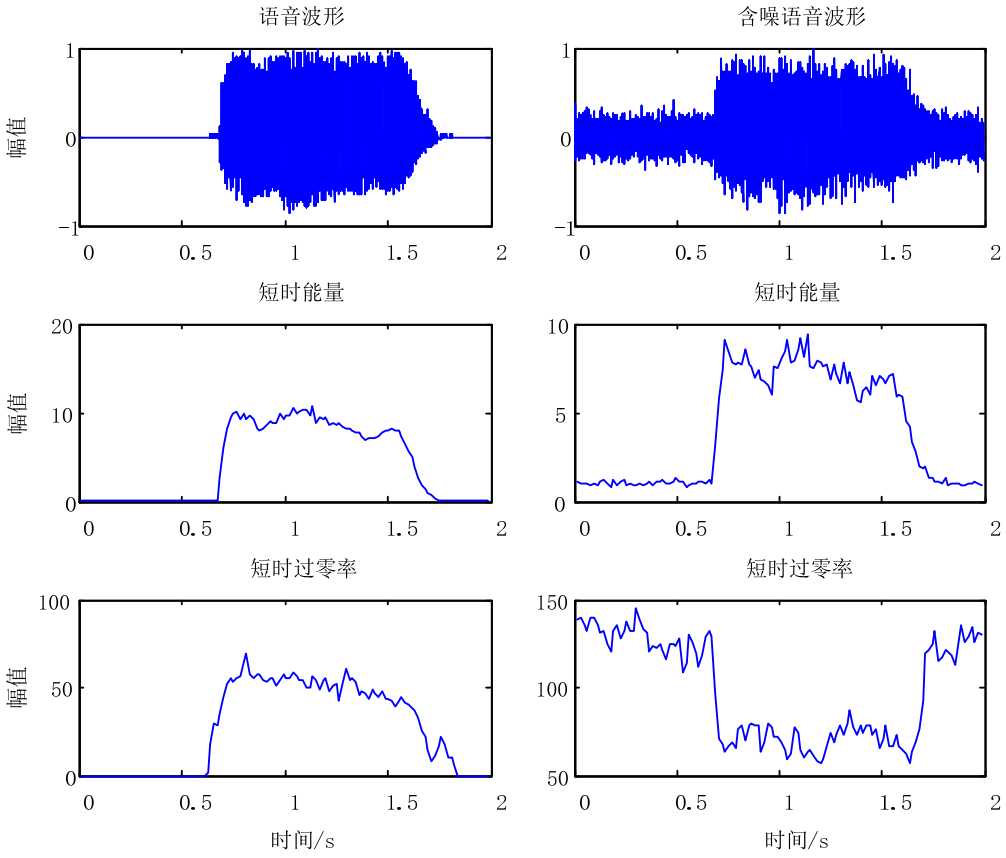

图 4-4 含噪信号的短时能量和短时过零率

在噪声情况下，信号的短时能量和短时过零率会发生一定变化，严重时会影响端点检测性能。图 4-4 是含噪情况下的短时能量和短时过零率显示图。从图中可知，在语音中的说话

区间能量是向上凸起的，而过零率相反，在说话区间向下凹陷。这表明，说话区间能量的数值大，而过零率数值小；在噪声区间能量的数值小，而过零率数值大，所以把能量值除以过零率的值，则可以更突出说话区间，从而更容易检测出语音端点。

改进式（4-1）的能量表示为

$$LE_n = \log_{10}(1 + E_n / a) \tag{4-10}$$

这里，a 为常数，适当的数值有助于区分噪音和清音。

过零率的计算基本同式（4-2）和式（4-3）。不过，这里 $x_n(m)$ 需要先进行限幅处理，即

$$\tilde{x}_n(m) = \begin{cases} x_n(m) & |x_n(m)| > \sigma \\ 0 & |x_n(m)| < \sigma \end{cases} \tag{4-11}$$

此时，能零比可表示为：

$$EZR_n = LE_n / (ZCR_n + b) \tag{4-12}$$

此处，b 为一较小的常数，防止 ZCR_n 为零时溢出。

4.2 能熵比的端点检测

谱熵值很类似于过零率值，在说话区间内的谱熵值小于噪声段的谱熵值，所以同能零比，能熵比的表示为：

$$EEF_n = \sqrt{1 + |LE_n / H_n|} \tag{4-13}$$

4.1.3 实验步骤及要求

1.实验步骤

运行 Python——>新建 py 文件——>编写 py 程序——>编译并调试。

2、实验要求

注：实验中的用到的噪声添加函数 awgn 说明。

名称：awgn

功能：在信号中加入高斯白噪声。

调用格式：y = awgn(x,SNR)

说明：在信号 x 中加入高斯白噪声。信噪比 SNR 以 dB 为单位；y 为叠加噪声后的信号。

1）根据**双门限法**的原理，编写 Python 函数，并基于测试语音 C4_1_y.wav 实现端点检测效果图 4-5。函数定义如下：

名称：vad_TwoThr

功能：用双门限法进行端点检测。

调用格式：[voiceseg,vsl,SF,NF,amp,zcr]=vad_TwoThr(x,wlen,inc,NIS)

说明：输入参数 x 是输入的语音数据；wlen 是帧长；inc 是帧移；NIS 是无声段的帧数，用来计算阈值。输出参数 voiceseg 是一个数据结构，记录了语音端点的信息；vsl 是 voiceseg 的长度；SF 是语音帧标志（SF=1 表示该帧是语音段）；NF 是噪声/无声帧标志（NF=1 表示该帧是噪声/无声段）；amp 是返回的短时能量，zcr 是返回的短时过零率。

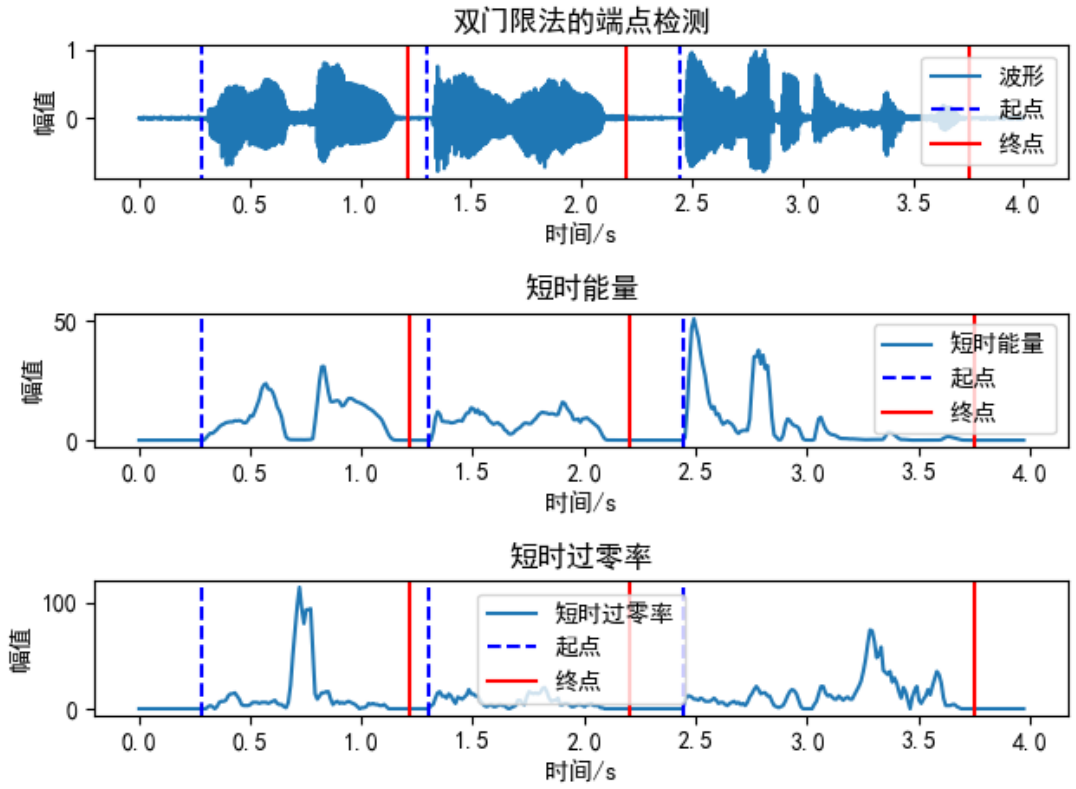

图 4-5 双门限法端点检测效果图

2）根据**相关法**的原理，编写 Python 函数，并基于测试语音 C4_1_y.wav 实现端点检测效果图 4-6。函数定义如下：

名称：vad_corr

功能：用自相关函数最大值法进行端点检测。

调用格式：[voiceseg,vsl,SF,NF,Rum]=vad_corr(x,wnd,inc,NIS,th1,th2)

说明：输入参数 x 是输入的语音数据；wnd 是窗函数或窗长；inc 是帧移；th1 是端点检测阈值；th2 是语音检测阈值；NIS 是无声段的帧数，用来计算阈值。输出参数 voiceseg 是一个数据结构，记录了语音端点的信息；vsl 是 voiceseg 的长度；SF 是语音帧标志（SF=1 表示该帧是语音段）；NF 是噪声/无声帧标志（NF=1 表示该帧是噪声/无声段）；Rum 是返回的短时自相关序列的最大值。

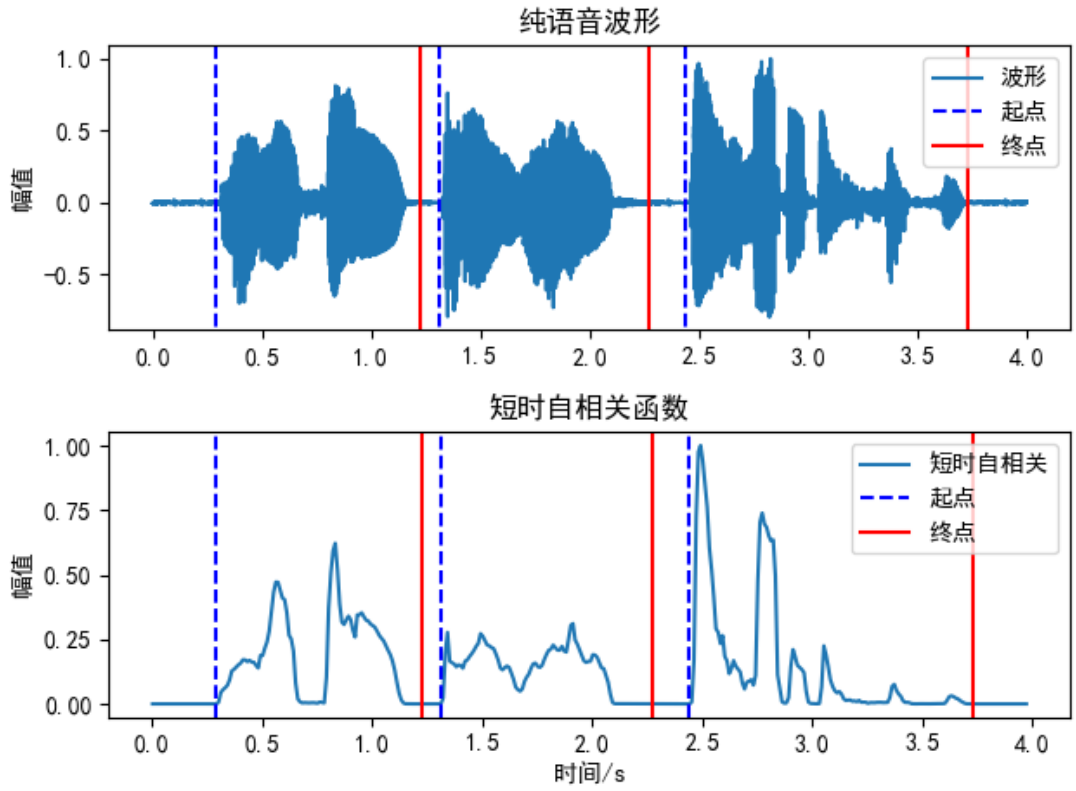

图 4-6 自相关法端点检测效果图

3）根据**谱熵法**的原理，编写 Python 函数，并基于测试语音 C4_1_y.wav 实现端点检测效果图 4-7。函数定义如下：

名称：vad_specEn

功能：用谱熵法进行端点检测。

调用格式：[voiceseg,vsl,SF,NF,Enm]=vad_specEn(x,wnd,inc,NIS,th1,th2,fs)

说明：输入参数 x 是输入的语音数据；wnd 是窗函数或窗长；inc 是帧移；th1 是端点检测阈值；th2 是语音检测阈值；NIS 是无声段的帧数，用来计算阈值；fs 是采样频率。输出参数 voiceseg 是一个数据结构，记录了语音端点的信息；vsl 是 voiceseg 的长度；SF 是语音帧标志（SF=1 表示该帧是语音段）；NF 是噪声/无声帧标志（NF=1 表示该帧是噪声/无声段）；Enm 是计算的谱熵。

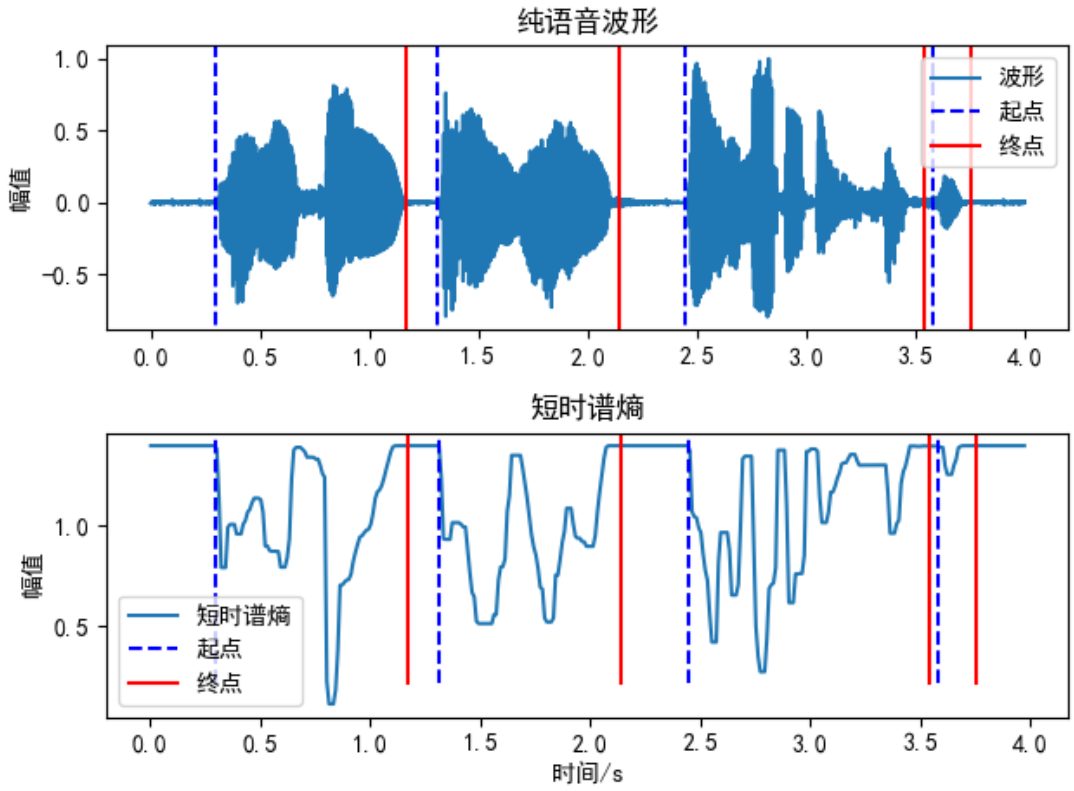

图 4-7 谱熵法端点检测效果图

4）根据比例法的原理，编写 Python 函数，并基于测试语音 C4_1_y.wav 实现端点检测效果图 4-8。函数定义如下：

名称：vad_pro

功能：用比例法进行端点检测。

调用格式：[voiceseg,vsl,SF,NF,Epara]=vad_pro(x,wnd,inc,NIS,th1,th2,mode);

说明：输入参数 x 是输入的语音数据；wnd 是窗函数或窗长；inc 是帧移；th1 是端点检测阈值；th2 是语音检测阈值；NIS 是无声段的帧数，用来计算阈值；mode 是算法模式，1 代表能零比，2 代表能熵比。输出参数 voiceseg 是一个数据结构，记录了语音端点的信息；vsl 是 voiceseg 的长度；SF 是语音帧标志（SF=1 表示该帧是语音段）；NF 是噪声/无声帧标志（NF=1 表示该帧是噪声/无声段）；Epara 是计算的能零比或能熵比。

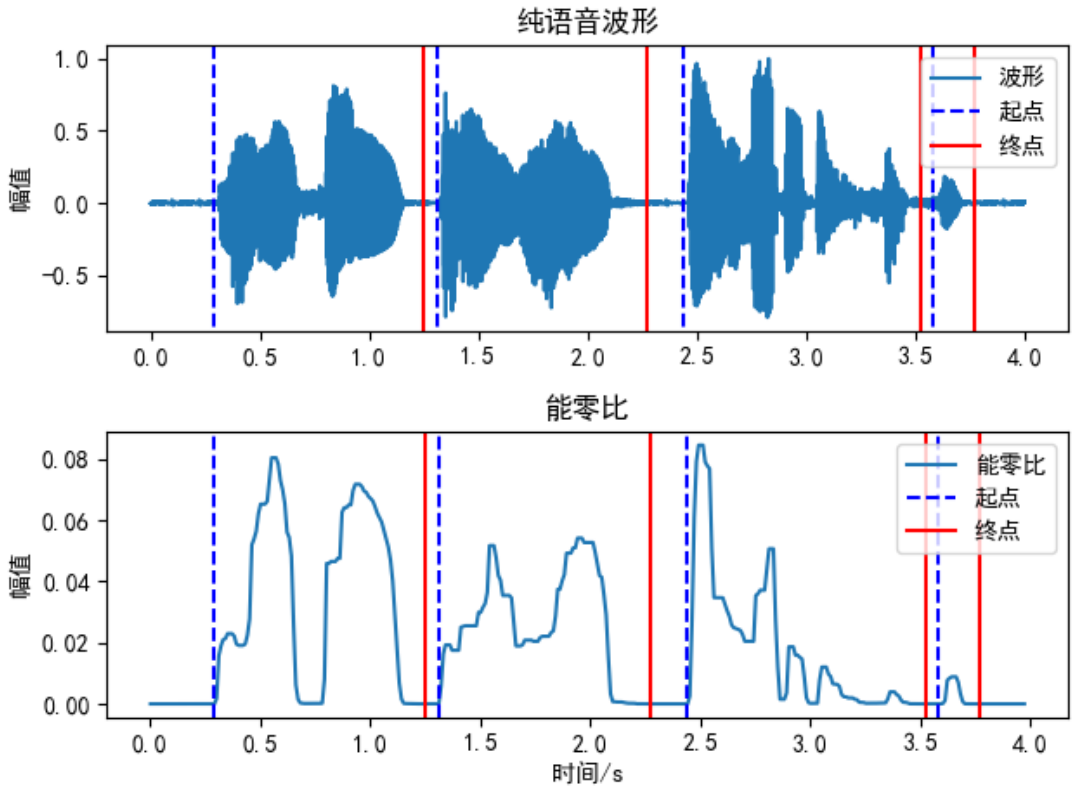

图 4-8 比例法端点检测效果图

4.1.4 思考题

1）当使用能零比进行端点检测时，参数起伏会比较大，如图所示。此时，算法的精度就会影响，此时就需要平滑处理进行滤波。试编写函数对能零比进行平滑滤波，实现能零比端点检测的改善，效果如图 4-9 所示。提示：参考 Python 的中值滤波函数。

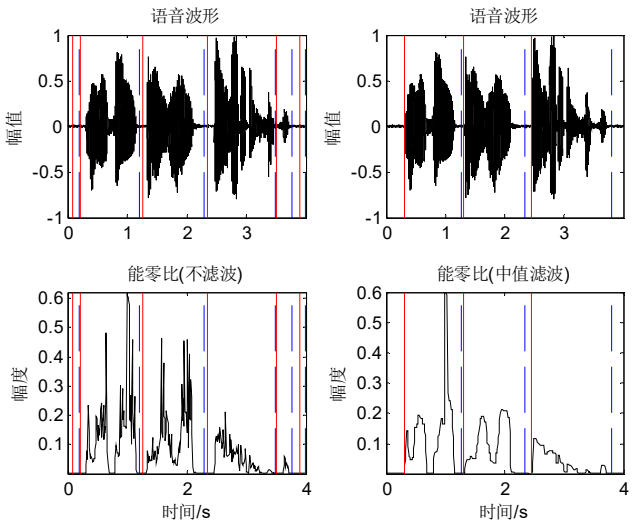

图 4-9 滤波前后的能零比法端点检测效果图

2）使用 awgn 函数，往语音中添加不同信噪比的白噪声，观察端点检测效果，并考虑如何改进算法性能。

4.1.5 参考例程

【双门限法端点检测函数】

```
def vad_TwoThr(x, wlen, inc, NIS):
    """
    使用双门限法检测语音段
    :param x: 语音信号
    :param wlen: 分帧长度
    :param inc: 帧移
    :param NIS: 无声段的帧数
    :return:
    """
    maxsilence = 15 # 设置最大无声
    minlen = 5  # 设置最小段长
    status = 0
    y = enframe(x, wlen, inc)    # 语音分帧
    fn = y.shape[0] # 取出列数
    amp = STEn(x, wlen, inc) # 计算短时能量
    zcr = STZcr(x, wlen, inc, delta=0.01)    # 计算短时过零率
    ampth = np.mean(amp[:NIS])  # 有声段短时能量的平均值
    zcrth = np.mean(zcr[:NIS])  # 有声段短时过零率平均值
    amp2 = 2 * ampth      # 门限 T1
    amp1 = 4 * ampth      # 门限 T2
    zcr2 = 2 * zcrth      # 门限 T3
    xn = 0  # 计数器
    count = np.zeros(fn)
    silence = np.zeros(fn)
    x1 = np.zeros(fn)
    x2 = np.zeros(fn)
    for n in range(fn):
        if status == 0 or status == 1:
            if amp[n] > amp1:    # 若该点短时能量在 T2 之上，不是端点，将其标为情况 2，进行下一点的检测
                x1[xn] = max(1, n - count[xn] - 1)
                status = 2
                silence[xn] = 0
                count[xn] += 1  # 语音段长加一
            elif amp[n] > amp2 or zcr[n] > zcr2:      # 若该点短时能量在 T2 之下，T1 之上，或短时过零率在 T3 之上，不是端点，将其标为情况 1，进行下一点的检测
                status = 1
                count[xn] += 1  # 语音段长加一
```

```
            else:    # 若该点的短时能量在所有门限之下，标为情况 0，设置为无声段
                status = 0
                count[xn] = 0    # 语音段长清零
                x1[xn] = 0
                x2[xn] = 0

        elif status == 2:    # 如果是情况 2
            if amp[n] > amp2 and zcr[n] > zcr2:    # 继续检测,如果同时在 T1、T3 之上
                count[xn] += 1    # 语音段长加一
            else:    # 否则，标为无声段
                silence[xn] += 1
                if silence[xn] < maxsilence:       # 如果无声段段长没超出限制
                    count[xn] += 1    # 语音段长加一
                elif count[xn] < minlen:       # 如果语音段长达不到最小段长限制
                    status = 0    # 回归到情况 0
                    silence[xn] = 0
                    count[xn] = 0
                else:    # 语音段长超过最小段长且无声段超过限制
                    status = 3    # 标为情况 3
                    x2[xn] = x1[xn] + count[xn]
        elif status == 3:    # 只有在情况 3 下，计数器加一
            status = 0    # 情况重新归位 0
            xn += 1
            count[xn] = 0    # 设置新计数各项参数的初值
            silence[xn] = 0
            x1[xn] = 0
            x2[xn] = 0
    el = len(x1[:xn])
    # 报错提示
    if x1[el - 1] == 0:
        el -= 1
    if x2[el - 1] == 0:
        print('Error: Not find endding point!\n')
        x2[el] = fn
    SF = np.zeros(fn)
    NF = np.ones(fn)
    for i in range(el):
        SF[int(x1[i]):int(x2[i])] = 1    # 标出语音段
        NF[int(x1[i]):int(x2[i])] = 0    # 标出无声段
    voiceseg = findSegment(np.where(SF == 1)[0])       # 分割成语音段
    vsl = len(voiceseg.keys())    # 计算语音段的长度
    return voiceseg, vsl, SF, NF, amp, zcr
```

4.2 基音周期检测实验

4.2.1 实验目的

1）了解语音基音周期检测的意义；

2）了解基音周期检测预处理的意义；

3）掌握基于倒谱法、短时自相关法和线性预测法的语音基音周期检测原理；

4）编写基于倒谱法、短时自相关法和线性预测法的语音基音周期检测函数，并仿真验证。

4.2.2 实验原理

人在发音时，根据声带是否震动可以将语音信号分为清音跟浊音两种。浊音又称有声语言，携带者语言中大部分的能量，浊音在时域上呈现出明显的周期性；而清音类似于白噪声，没有明显的周期性。发浊音时，气流通过声门使声带产生张弛震荡式振动，产生准周期的激励脉冲串。这种声带振动的频率称为基音频率，相应的周期就成为基音周期。

通常，基音频率与个人声带的长短、薄厚、韧性、劲度和发音习惯等有关系，在很大程度上反应了个人的特征。此外，基音频率还跟随着人的性别、年龄不同而有所不同。一般来说，男性说话者的基音频率较低，大部分在 70～200Hz 的范围内，而女性说话者和小孩的基音频率相对较高，在 200～450Hz 之间。

基音周期的估计称谓基音检测，基音检测的最终目的是为了找出和声带振动频率完全一致或尽可能相吻合的轨迹曲线。

基音周期作为语音信号处理中描述激励源的重要参数之一，在语音合成、语音压缩编码、语音识别和说话人确认等领域都有着广泛而重要的问题，尤其对汉语更是如此。汉语是一种有调语言，而基音周期的变化称为声调，声调对于汉语语音的理解极为重要。因为在汉语的相互交谈中，不但要凭借不同的元音、辅音来辨别这些字词的意义，还需要从不同的声调来区别它，也就是说声调具有辨义作用；另外，汉语中存在着多音字现象，同一个字的不同的语气或不同的词义下具有不同的声调。因此准确可靠地进行基音检测对汉语语音信号的处理显得尤为重要。

自进行语音信号分析研究以来，基音检测一直是一个重点研究的课题。尽管目前基音检测的方法有很多种，然而这些方法都有其局限性。迄今为止仍然没有一种检测方法能够适用不同的说话人、不同的要求和环境、究其原因，可归纳为如下几个方面：

1．语音信号变化十分复杂，声门激励的波形并不是完全的周期脉冲串，在语音的头、尾部并不具有声带振动那样的周期性，对于有些清浊音的过度帧很难判定其应属于周期性或非周期性，从而也就无法估计出基音周期。

2．声道共振峰有时会严重影响激励信号的谐波结构，使得想要从语音信号中去除声道影响，直接取出仅和声带振动有关的声源信息并不容易。

3．在浊音语音段很难对每个基音周期的开始和结束位置进行精确的判断，一方面因为语音信号本身是准周期的。另一方面因为语音信号的波形受共振峰、噪音等因素的影响。

4．在实际应用中，语音信号常常混有噪声，而噪声的存在对于基音检测算法的性能产生强烈影响。

5．基音频率变化范围大，从低音男声的 70Hz 到儿童女性的 450Hz，接近 3 个倍频程，给基音检测带来了一定的困难。

尽管语音检测面临着很多困难，然而由于基音周期在语音信号处理领域的重要性，使得语音基音周期检测一直是不断研究改进的重要课题之一。数十年来，国内外众多学者对如何准确地从语音波形中提取出基音周期作出了不懈的努力，提出了多种有效的基音周期检测方法。我国基音检测方面的研究起步要比国外发达国家晚一点，但是进步很大，特别是对汉语的基音检测取得成果尤为突出。目前的基音检测算法大致可分为两大类：非基于事件检测方法和基于事件检测方法,这里的事件是指声门闭合。

非基于事件的检测方法主要有：自相关函数法、平均幅度差函数法，倒谱法，以及在以上算法基础上的一些改进算法。语音信号是一种典型的时变、非平稳信号，但是，由于语音的形成过程是发音器官的运动密切相关的，而这种物理运动比起声音振动速度来讲要缓慢得多，因此语音信号常常可假定为短时平稳的，即在短时间内，其频谱特性和某些物理特征参量可近似地看作是不变的，非基于事件的检测方法正是利用语音信号短时平稳性这一特点，先将语音信号分为长度一定的语音帧，然而对每一帧语音求基音周期。相比基于事件的基音周期检测方法来说，它的优点是算法简单，运算量小，然而从本质上说这些方法无法检测帧内基音周期的非平稳变化，检测精度不高。

基于事件的检测方法是通过定位声门闭合时刻来对基音周期进行估计，而不需要对语音信号进行短时平稳假设，主要有小波变换方法和 Hilbert-Huang 变换方法两种。在时域和频域上这两种方法又具有良好的局部特性，能够跟踪基音周期的变化，并可以将微小的基音周期变化检测出来，因此检测精度较高，但是计算量较大。

1、基音检测预处理

由于语音的头部和尾部并不具有声带振动那样的周期性，因此为了提高检测基音检测的准确性，基音检测也需要进行端点检测，但是基音检测中的端点检测更严格。本节主要采用谱熵比法进行端点检测。

如 4.1 节所述，能熵比的定义为

$$EEF_n = \sqrt{1 + \left| LE_n / H_n \right|} \qquad (4\text{-}14)$$

不同的是，这里只用一个门限 $T1$ 作判断，判断能熵比值是否大于 $T1$，把大于 $T1$ 的部分作为有话段的候选值，再进一步判断该段的长度是否大于最小值 minL，只有大于最小值的才作为有话段。此处，minL 一般设定为 10 帧。

此外，为了减少共振峰的干扰，基音检测的预滤波器选择带宽一般为 60~500Hz。这里，选择 60Hz 是为了减少工频和低频噪声的干扰；选择 500Hz 是因为基频区间的高端在这个区域中。当采样频率为 f_s 时，在 60Hz 处对应的基音周期（样本点值）为 $P_{\max} = f_s / 60$，而 500Hz 对应的基音周期（样本点值）为 $P_{\min} = f_s / 500$。

考虑到语音信号对相位不敏感，因此选择运算量少的椭圆 IIR 滤波器。因为在相同过渡带和带宽条件下，椭圆滤波器需要的阶数较小。当采样频率为 8000Hz 时，通带是 60~500Hz，通带波纹为 1dB；阻带分别为 30Hz 和 2000Hz，阻带衰减为 40dB。此时的滤波器频响如图 4-10 所示。

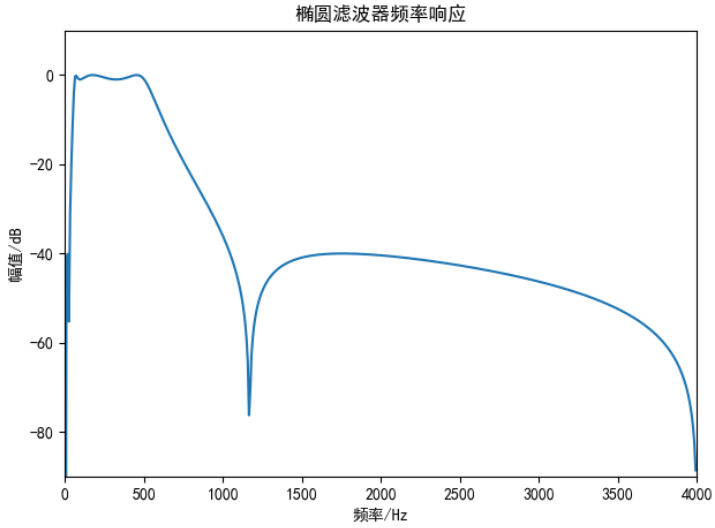

图 4-10 带通椭圆滤波器频响曲线

2、倒谱法基音检测

由于语音 $x(i)$ 是由声门脉冲激励 $u(i)$ 经声道响应 $v(i)$ 滤波而得，即

$$x(i)=u(i)*v(i) \tag{4-15}$$

设这三个量的倒谱分别为 $\hat{x}(i)$ 、 $\hat{u}(i)$ 、 $\hat{v}(i)$ ，则有

$$\hat{x}(i)=\hat{u}(i)+\hat{v}(i) \tag{4-16}$$

由于在倒谱域中 $\hat{u}(i)$ 和 $\hat{v}(i)$ 是相对分离的，说明包含有基音信息的声脉冲倒谱可与声道响应倒谱分离，因此从倒频谱域分离 $\hat{u}(i)$ 后恢复出 $u(i)$ ，可从中求出基音周期。在计算出倒谱后，就在倒频率为 $P_{\min}\sim P_{\max}$ 之间寻找倒谱函数的最大值，倒谱函数最大值对应的样本点数就是当前帧语音信号的基音周期 $T_0(n)$ ，基音频率为 $F_0(n)=f_s/T_0(n)$ 。

图 4-11 为倒谱法检测的基音周期图。

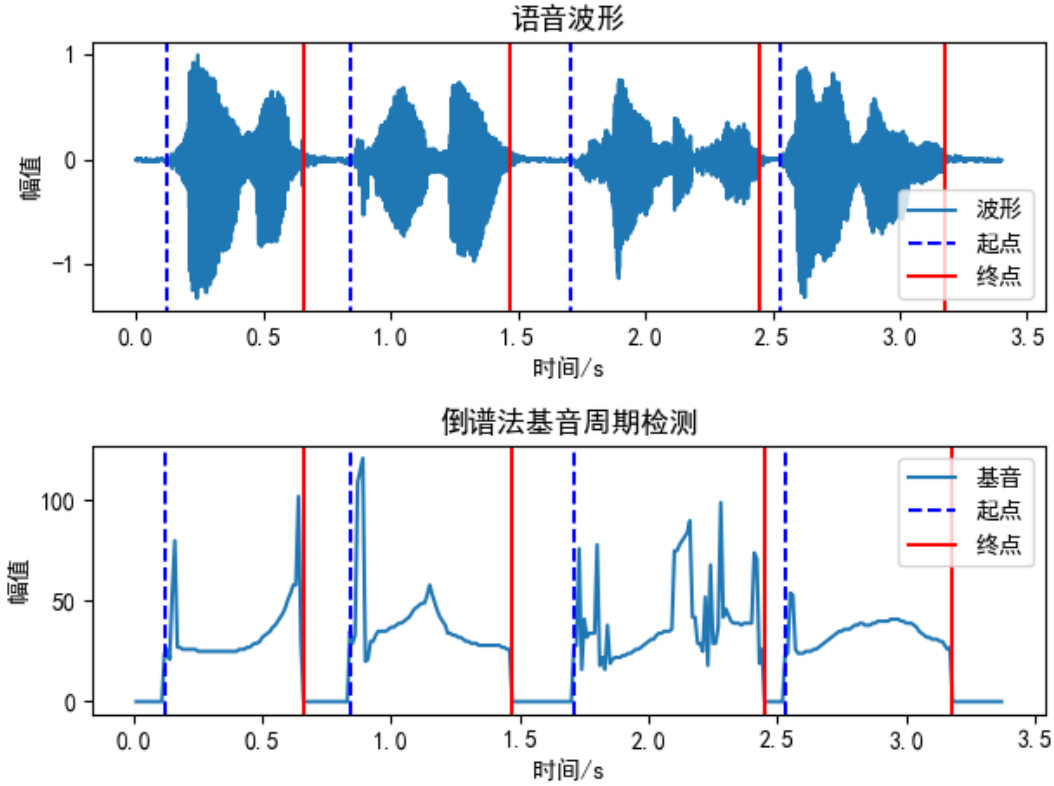

图 4-11 倒谱法检测的基音周期图

3、短时自相关法基音检测

短时自相关法基音检测主要是利用短时自相关函数的性质，通过比较原始信号及其延迟后信号间的类似性来确定基音周期。归一化自相关函数的最大幅值是 1，其它延迟量时，幅值都小于 1。如果延迟量等于基音周期，那两个信号具有最大类似性；或直接找出短时自相关函数的两个最大值间的距离，即作为基音周期的初估值。

和倒谱法寻找最大值一样，用相关函数法时也在 $P_{\min} \sim P_{\max}$ 间寻找归一化相关函数的最大值，最大值对应的延迟量就是基音周期。图 4-12 为自相关法检测的基音周期图。

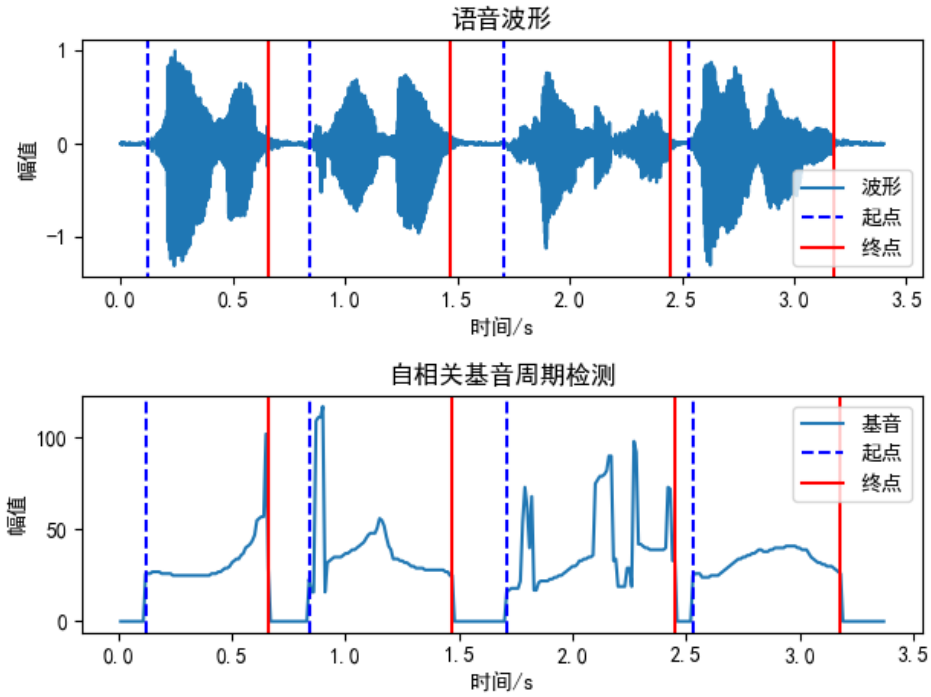

图 4-12 自相关法检测的基音周期图

4.2.3 实验步骤及要求

1.实验步骤

运行 Python——>新建 py 文件——>编写 py 程序——>编译并调试。

2、实验要求

1）根据用于基音周期检测的端点检测算法的原理，编写端点检测函数，并基于测试语音 C4_2_y.wav 进行测试，显示效果如图 4-13 所示。

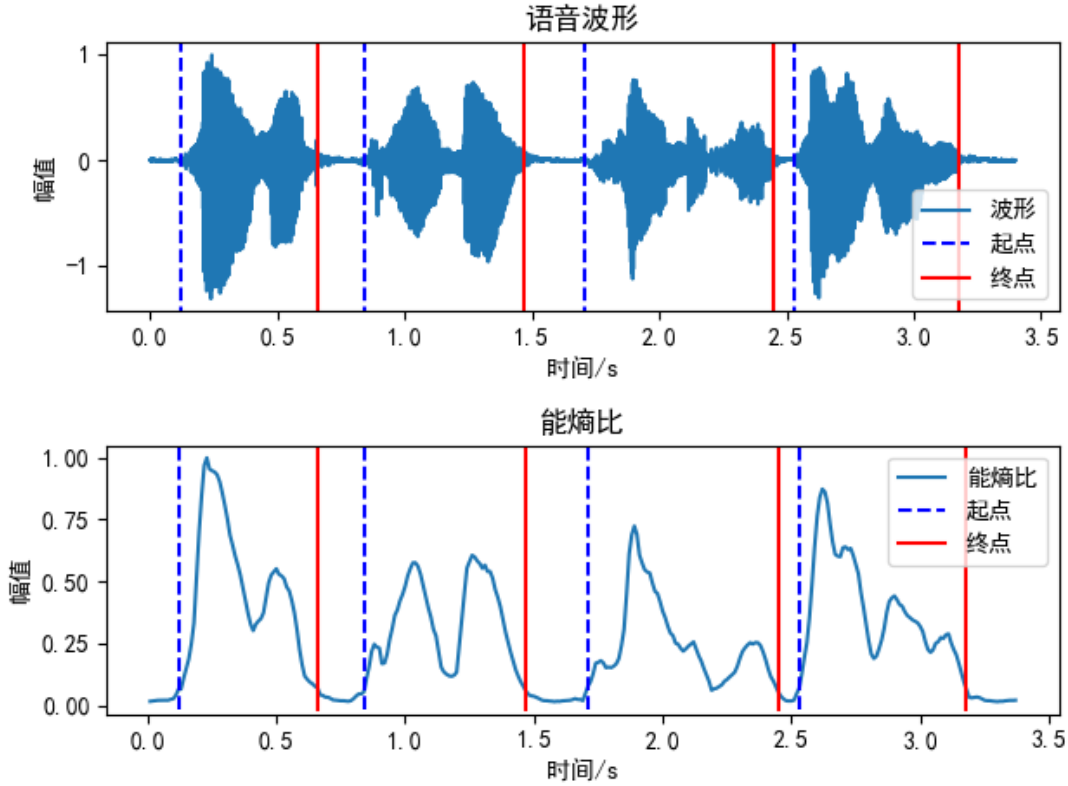

图 4-13 端点检测效果图

函数定义为：

名称：pitch_vad

功能：用谱熵比法进行端点检测。

调用格式：[voiceseg,vosl,SF,Ef]=pitch_vad(x,wnd,inc,T1,miniL)

说明：输入参数 x 是输入的语音数据；wnd 是窗函数或窗长；inc 是帧移；T1 是端点检测阈值；miniL 是语音段的最小帧数。输出参数 voiceseg 是一个数据结构，记录了语音端点的信息；vsl 是 voiceseg 的长度；SF 是语音帧标志（SF=1 表示该帧是语音段）；EF 是噪声/无声帧标志（EF=1 表示该帧是噪声/无声段）。

2）设计根据实验原理中提出的带通滤波器参数，编写程序实现该带通滤波器，并求出如图 4-10 的频响曲线。

3）根据倒谱法的原理，编写 Python 函数，并基于测试语音 C4_2_y.wav 实现基音周期检测，效果如图 4-11。函数定义如下：

名称：pitch_Ceps

功能：用倒谱法进行基音周期检测。

调用格式：[voiceseg,vsl,SF,Ef,period]=pitch_Ceps(x,wnd,inc,T1,fs,miniL)

说明：输入参数 x 是输入的语音数据；wlen 是帧长；inc 是帧移；miniL 是语音段的最小帧数；fs 是采样频率。输出参数 voiceseg 是一个数据结构，记录了语音端点的信息；vsl 是 voiceseg 的长度；SF 是语音帧标志（SF=1 表示该帧是语音段）；EF 是噪声/无声帧标志（EF=1 表示该帧是噪声/无声段）；period 是返回的基音周期。

4）根据短时自相关法的原理，编写 Python 函数，并基于测试语音 C4_2_y.wav 实现基音周期检测，效果如图 4-12。函数定义如下：

名称：pitch_Corr

功能：用短时自相关法进行基音周期检测。

调用格式：[voiceseg,vsl,SF,Ef,period]=pitch_Corr(x,wnd,inc,T1,fs,miniL)

说明：输入参数 x 是输入的语音数据；wlen 是帧长；inc 是帧移；miniL 是语音段的最小帧数；fs 是采样频率。输出参数 voiceseg 是一个数据结构，记录了语音端点的信息；vsl 是 voiceseg 的长度；SF 是语音帧标志（SF=1 表示该帧是语音段）；EF 是噪声/无声帧标志（EF=1 表示该帧是噪声/无声段）；period 是返回的基音周期。

4.2.4 思考题

当基音周期检测中，常会产生基音检测错误，使求得的基音周期轨迹中有一个或几个基音周期的估算值出现偏差（通常是偏离到实际值的整数倍），如图 4-11 所示。试结合中值滤波和线性平滑法对基音周期进行滤波处理。

其中，线性平滑是用滑动窗进行线性滤波处理，其原理为

$$y(m)=\sum_{l=-L}^{L}x(m-l)w(l) \qquad (4\text{-}17)$$

式中，$w(l)$ 为 $2L+1$ 点平滑窗，满足 $\sum_{l=-L}^{L}w(l)=1$。

试编写函数对检测的基音周期进行平滑滤波，改善效果如图 4-14 所示。

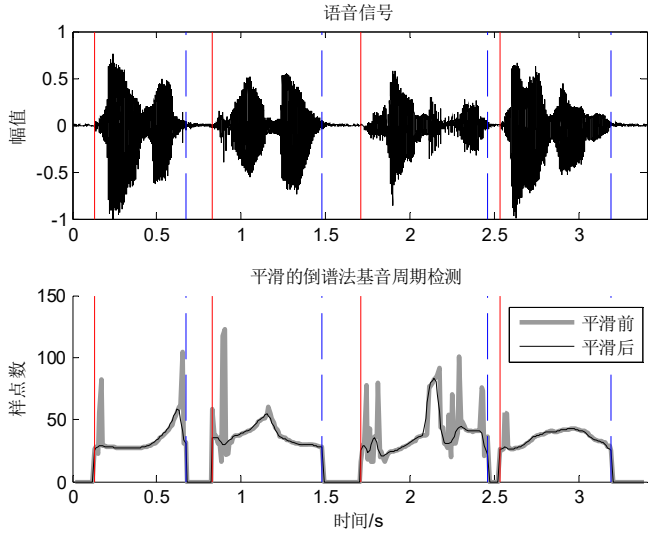

图 4-14 平滑前后的倒谱法基音周期检测效果图

4.2.5 参考例程

【倒谱法基音检测函数】

```
def pitch_Ceps(x, wnd, inc, T1, fs, miniL=10):
    """
    倒谱法基音周期检测函数
    :param x:  语音信号
    :param wnd:  窗函数或窗长
```

```
    :param inc: 帧移
    :param T1: 门限
    :param fs: 采样率
    :param miniL: 语音段的最小帧数
    :return:
    """
    y = enframe(x, wnd, inc)
    fn = y.shape[0]
    if isinstance(wnd, int):
        wlen = wnd
    else:
        wlen = len(wnd)
    voiceseg, vsl, SF, Ef = pitch_vad(x, wnd, inc, T1, miniL)  # 语音分段
    lmin = fs // 500  # 基音周期的最小值
    lmax = fs // 60  # 基音周期的最大值
    period = np.zeros(fn)
    y1 = y[np.where(SF == 1)[0], :]  # 找到语音段
    y1 = np.multiply(y1, np.hamming(wlen))  # 语音段加汉明窗
    xx = np.fft.fft(y1, axis=1)        # 进行FFT变换
    b = np.fft.ifft(2 * np.log(np.abs(xx) + 1e-10))    # 计算倒谱
    Lc = np.argmax(b[:, lmin:lmax], axis=1) + lmin - 1  # 在设定的倒频率间找到倒谱的最大值对应的样本点
    period[np.where(SF == 1)[0]] = Lc    # 标记基音周期
    return voiceseg, vsl, SF, Ef, period
```

4.3 共振峰估计实验

4.3.1 实验目的

1）了解共振峰估计的意义；
2）了解共振峰估计预处理的方法与意义；
3）掌握基于倒谱法和线性预测法的共振峰检测原理；
4）编写基于倒谱法和线性预测法的共振峰检测函数，并仿真验证。

4.3.2 实验原理

声道可以被看成一根具有非均匀截面的声管，在发音时将起共鸣器的作用。当声门处准周期脉冲激励进入声道时会引起共振特性，产生一组共振频率，这一组共振频率称为共振峰频率或简称为共振峰。共振峰参数包括共振峰频率、频带宽度和幅值，共振峰信息包含在语音频谱的包络中。因此共振峰参数提取的关键是估计语音频谱包络，并认为谱包络中的最大值就是共振峰。利用语音频谱傅里叶变换相应的低频部分进行逆变换，就可以得到语音频谱的包络曲线。依据频谱包络线各峰值能量的大小确定出第 1～第 4 共振峰，如图 4-15 所示。

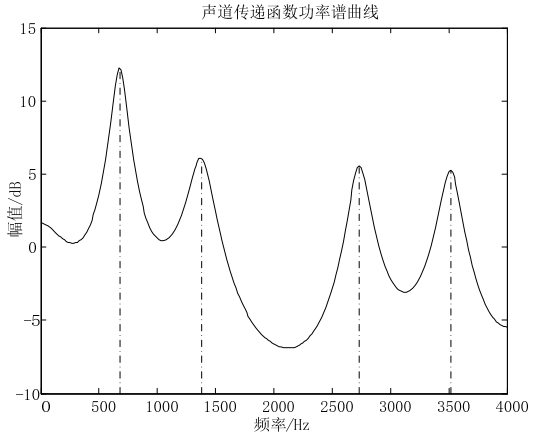

图 4-15 声道传递函数功率谱曲线

在经典的语音信号模型中,共振峰等效为声道传输函数的复数极点对。对平均长度约为 17cm 声道(男性),在 3kHz 范围内大致包含三个或四个共振峰,而在 5kHz 范围内包含四个或五个共振峰。高于 5kHz 的语音信号,能量很小。根据语音信号合成的研究表明,表示浊音信号最主要的是前三个共振峰。一个语音信号的共振峰模型,只用前三个时变共振峰频率就可以得到可懂度很好的合成浊音。共振峰信息包含在语音信号的频谱包络中，谱包络的峰值基本上对应与共振峰频率。因此一切共振峰估计都是直接或间接地对频谱包络进行考察，关键是估计语音频谱包络，并认为谱包络中的最大值就是共振峰。与基音提取，共振峰估计也是表面看很容易但实际上又为许多问题所困扰。包括：虚假峰值、共振峰合并、高基音语音。语音信号共振峰估计,在语音信号合成、语音信号自动识别和低比特律语音信号传输等方面都起着重要作用。

1、共振峰估计预处理

声门的激励源发出斜三角脉冲串，其 Z 变换相当于一个二阶低通低通的模型，而辐射模型是一个一阶高通。所以，实际信号分析中，常采用预加重技术使声门脉冲的影响减到最小，只剩下声道部分，以便对声道参数进行分析。

预加重是一个一阶高通滤波器，用来对语音信号进行高频提升，用一阶 FIR 滤波器表示为：

$$s'(n) = s(n) - a \cdot s(n-1) \tag{4-18}$$

预加重有两个作用：

1）增加一个零点，抵消声门脉冲引起的高端频谱幅度下跌，使信号频谱变得平坦及各共振峰幅度相接近；语音中只剩下声道部分的影响，所提取的特征更加符合原声道的模型；

2）会削减低频信息，使有些基频幅值变大时，通过预加重后降低基频对共振峰检测的干扰，有利于共振峰的检测；同时减少频谱的动态范围。

此外，由于共振峰检测一般是分析韵母部分，所以需要进行端点检测。此处，可采用同 4-2 节基音周期检测相同的端点检测算法。

2、倒谱法共振峰估计

由于语音 $x(n)$ 是由声门脉冲激励 $u(n)$ 经声道响应 $v(n)$ 滤波而得，即

$$x(n) = u(n) * v(n) \tag{4-19}$$

设这三个量的倒谱分别为 $\hat{x}(n)$ 、 $\hat{u}(n)$ 、 $\hat{v}(n)$ ，则有

$$\hat{x}(n) = \hat{u}(n) + \hat{v}(n) \tag{4-20}$$

由于在倒谱域中 $\hat{u}(n)$ 和 $\hat{v}(n)$ 是相对分离的，说明包含有基音信息的声脉冲倒谱可与声道响应倒谱分离，因此从倒频谱域分离 $\hat{u}(n)$ 后恢复出 $u(n)$，可从中求出基音周期。因此，求取共振峰时，则是把由倒谱域 $\hat{v}(n)$ 中分离出的恢复出 $v(n)$ 。具体步骤如下：

1）对语音信号 $x(i)$ 进行预加重，并进行加窗和分帧，然后做傅里叶变换；

$$X_i(k) = \sum_{n=0}^{N-1} x_i(n) e^{-j2\pi kn/N} \tag{4-21}$$

这里， i 代表第 i 帧。

2）求取 $X_i(k)$ 的倒谱；

$$\hat{x}_i(n) = \frac{1}{N} \sum_{k=0}^{N-1} \log\left|\hat{X}_i(k)\right| e^{j2\pi kn/N} \tag{4-22}$$

3）给倒谱信号 $\hat{x}_i(n)$ 加窗 $h(n)$，得

$$h_i(n) = \hat{x}_i(n) \times h(n) \tag{4-23}$$

此处的窗函数和倒频率的分辨率有关，即和采样频率及 FFT 长度有关。其定义为：

$$h(n) = \begin{cases} 1 & \mathrm{n} \le \mathrm{n}_0\text{-}1 \ \&\ \mathrm{n} \ge \mathrm{N}\text{-}\mathrm{n}_0\text{+}1 \\ 0 & \mathrm{n}_0\text{-}1 < \mathrm{n} < \mathrm{N}\text{-}\mathrm{n}_0\text{+}1 \end{cases}, n \in [0, N-1] \tag{4-24}$$

4）求取 $h_i(n)$ 的包络线

$$H_i(k) = \sum_{n=0}^{N-1} h_i(n) e^{-j2\pi kn/N} \tag{4-25}$$

5）在包络线上寻找极大值，获得相应的共振峰参数。

3、LPC 法共振峰估计

简化的语音产生模型是将辐射、声道以及声门激励的全部效应简化为一个时变的数字滤波器来等效，其传递函数为

$$H(z)=\frac{S(z)}{U(z)}=\frac{G}{1-\sum_{i=1}^{p}a_i z^{-i}} \tag{4-26}$$

这种表现形式称为 p 阶线性预测模型，这是一个全极点模型。

令 $z^{-1}=\exp(-j2\pi f/f_s)$，则功率谱 $P(f)$ 可表示为：

$$P(f)=\left|H(f)\right|^2=\frac{G^2}{\left|1-\sum_{i=1}^{p}a_i\exp(-j2\pi if/f_s)\right|^2} \tag{4-27}$$

利用 FFT 方法可对任意频率求得其功率谱幅值响应，并从幅值响应中找到共振峰，相应的求解方法有两种：1）抛物线内插法；2）线性预测系数求复数根法。

3.1 LPC 内插法

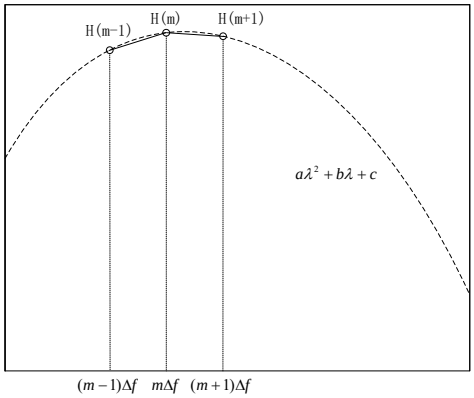

图 4-16 共振峰频率的抛物线内插图

任何一个共振峰频率都可以用抛物线内插法更精确的计算共振峰频率及其带宽。如图 4-16 所示，任一共振峰频率 F_i 的局部峰值频率为 $m\Delta f$ （Δf 为谱图的频率间隔），其邻近的两个频率点分别为 $(m-1)\Delta f$ 和 $(m+1)\Delta f$，这三个点在功率谱中的幅值分别为 $H(m-1)$，$H(m)$，$H(m+1)$。此时，可用二次方程组 $a\lambda^2+b\lambda+c$ 来拟合，以求出更精确的中心频率 F_i 和带宽 B_i。

令局部峰值频率 $m\Delta f$ 处为零，则对应于 $-\Delta f$，0，$+\Delta f$ 处的功率谱分别为 $H(m-1)$，$H(m)$，$H(m+1)$，按表示式 $H=a\lambda^2+b\lambda+c$，可得方程组：

$$\begin{cases}H(m-1)=a\Delta f^2-b\Delta f+c\\ H(m)=c\\ H(m+1)=a\Delta f^2+b\Delta f+c\end{cases} \tag{4-28}$$

假设 $\Delta f=1$，则计算的系数为：

$$\begin{cases} a = \dfrac{H(m-1)+H(m+1)}{2} - H(m) \\ b = \dfrac{H(m+1)-H(m-1)}{2} \\ c = H(m) \end{cases} \tag{4-29}$$

求导数 $\partial H / \partial \lambda = \partial(a\lambda^2 + b\lambda + c) / \partial \lambda = 0$，得极大值为：

$$\lambda_{\max} = -b / 2a \tag{4-30}$$

考虑到实际频率间隔，则共振峰的中心频率为

$$F_i = \lambda_{\max} \Delta f + m\Delta f \tag{4-31}$$

中心频率对应的功率谱 H_p 为

$$H_p = a\lambda_p^2 + b\lambda_p + c = -\frac{b^2}{4a} + c \tag{4-32}$$

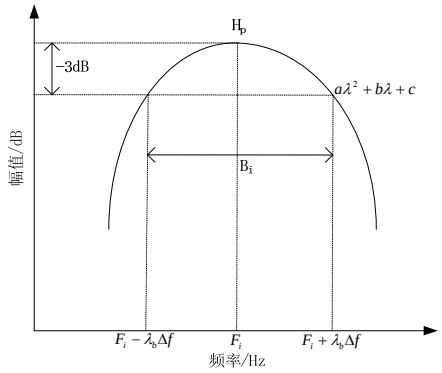

图 4-17 带宽求法的示意图

带宽 B_i 的求法如图 4-17 所示。在某一个 λ 处，其谱值为 H_p 值的一半，即有

$$\frac{a\lambda^2 + b\lambda + c}{H_p} = \frac{1}{2} \tag{4-33}$$

可以导出

$$a\lambda^2 + b\lambda + c - 0.5H_p = 0 \tag{4-34}$$

其根为

$$\lambda_{root} = \frac{-b \pm \sqrt{b^2 - 4a(c - 0.5H_p)}}{2a} \tag{4-35}$$

而半带宽 $B_i / 2$ 是根值与峰值位置的差值：

$$\lambda_b = \lambda_{root} - \lambda_p \tag{4-36}$$

可得

$$\lambda_b = -\frac{\sqrt{b^2 - 4a(c - 0.5H_p)}}{2a} \tag{4-37}$$

因为抛物线是下凹的，所以 λ_b 取正值。考虑到实际频率间隔 Δf ，则带宽 B_i 为

$$B_i = 2\lambda_b \Delta f \tag{4-38}$$

3.2 LPC 求根法

预测误差滤波器 $A(z)$ 的表示为：

$$A(z) = 1 - \sum_{i=1}^{p} a_i z^{-i} \tag{4-39}$$

求其多项式复根可精确的确定共振峰的中心频率和带宽。

设 $z_i = r_i e^{j\theta_i}$ 为任意复根值，则其共轭值 $z_i^* = r_i e^{-j\theta_i}$ 也是一个根。设与 z_i 对应的共振峰频率为 F_i ，3dB 带宽为 B_i ，则 F_i 及 B_i 与 z_i 之间的关系为

$$\begin{cases} 2\pi F_i / f_s = \theta_i \\ e^{-B_i \pi / f_s} = r_i \end{cases} \tag{4-40}$$

其中 f_s 为采样频率，所以

$$\begin{cases} F_i = \theta_i f_s / 2\pi \\ B_i = -\ln r_i \cdot f_s / \pi \end{cases} \tag{4-41}$$

因为预测误差滤波器阶数 p 是预先设定的，所以复共轭对的数量最多是 $p/2$ 。因为不属于共振峰的额外极点的带宽远大于共振峰带宽，所以比较容易剔除非共振峰极点。

4.3.3 实验步骤及要求

1.实验步骤

运行 Python——>新建 py 文件——>编写 py 程序——>编译并调试。

2、实验要求

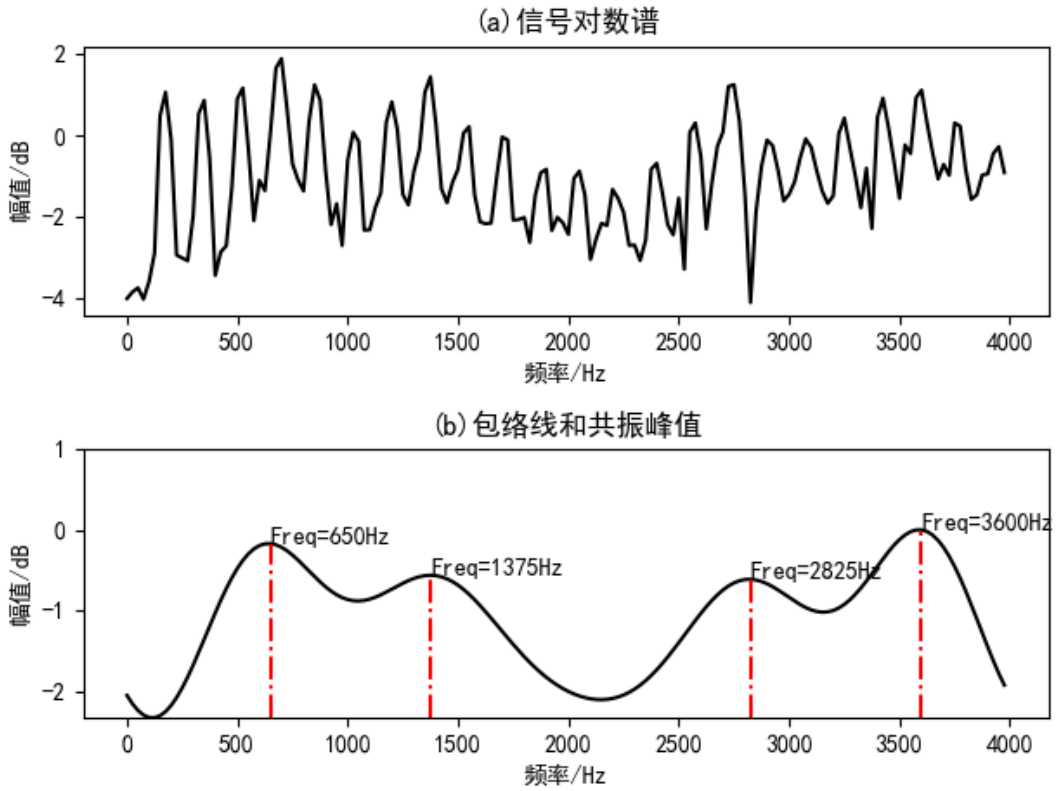

图 4-18 基于倒谱法的共振峰估计

1）根据基于倒谱法的共振峰估计的原理，编写基于倒谱法的共振峰估计函数，并基于测试语音 C4_3_y.wav 进行测试，显示效果如图 4-18 所示。

函数定义为：

名称：Formant_Cepst

功能：用倒谱法估计共振峰。

调用格式：[Val,Loc,spect]=Formant_Cepst(u,cepstL)

说明：输入参数 u 是一帧语音输入数据；cepstL 是倒频率上的窗函数宽度。输出参数 Val 是共振峰的幅值；Loc 是共振峰的位置；spect 是求得包络线。

2）根据基于 LPC 内插法的共振峰估计的原理，编写基于 LPC 内插法的共振峰估计函数，并基于测试语音 C4_3_y.wav 进行测试，显示效果如图 4-19 所示。

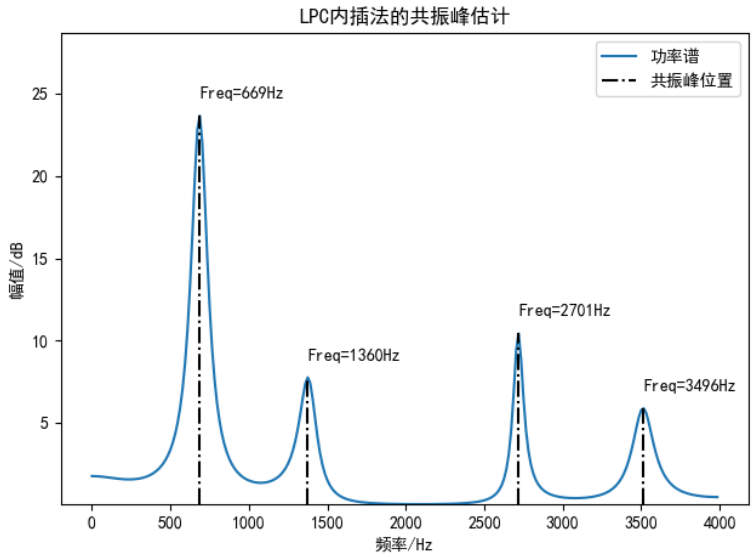

图 4-19 基于 LPC 内插法的共振峰估计

函数定义如下：

名称：Formant_Interpolation

功能：用 LPC 内插法进行共振峰估计。

调用格式：[F,Bw,pp,U]=Formant_Interpolation(u,p,fs)

说明：输入参数 u 是一帧语音输入数据；p 是 LPC 阶数；fs 是采样频率。输出参数 F 是共振峰频率；Bw 是共振峰带宽；pp 是共振峰的幅值；U 是功率谱包络。

3）根据基于 LPC 求根法的共振峰估计的原理，编写基于 LPC 求根法的共振峰估计函数，并基于测试语音 C4_3_y.wav 进行测试，显示效果如图 4-20 所示。

函数定义如下：

名称：Formant_Root

功能：用 LPC 求根法进行共振峰估计。

调用格式：[F,Bw,pp,U]=Formant_Root(u,p,fs,n_frmnt)

说明：输入参数 u 是一帧语音输入数据；p 是 LPC 阶数；fs 是采样频率；n_frmnt 是待求的共振峰个数。输出参数 F 是共振峰频率；Bw 是共振峰带宽；pp 是共振峰的幅值；U 是功率谱包络。

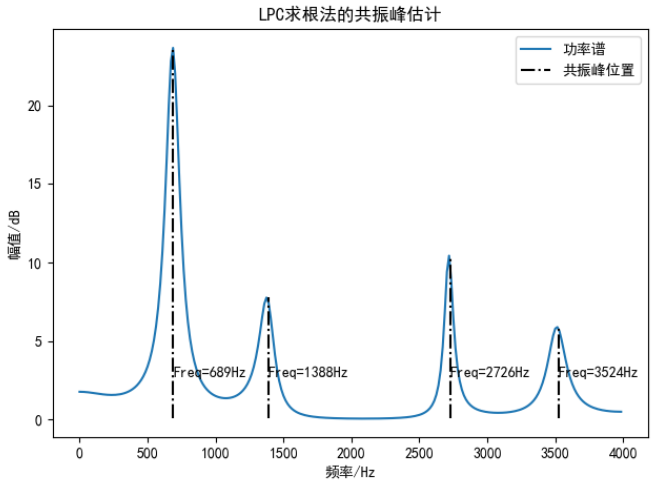

图 4-20 基于 LPC 求根法的共振峰估计

4.3.4 思考题

1）思考连续语音的共振峰求取问题，并基于 LPC 求根法编程实现连续语音的共振峰估计。测试语音为 C4_3_s.wav，效果如图 4-21 所示。

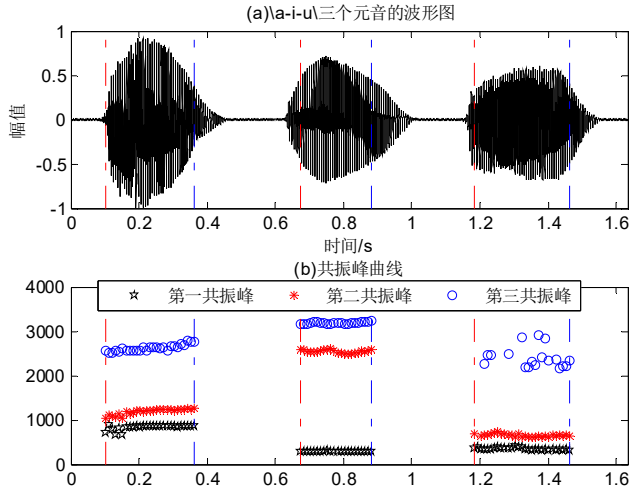

图 4-21 基于 LPC 求根法的连续语音共振峰估计

4.3.5 参考例程

【插值法求共振峰函数】

```
def Formant_Interpolation(u, p, fs):
    """
    插值法估计共振峰函数
    :param u:  一帧语音输入数据
    :param p: LPC阶数
    :param fs:  采样率
    :return:
    """
    ar, _ = lpc_coeff(u, p)       # 计算p阶线性预测器的系数
    U = np.power(np.abs(np.fft.rfft(ar, 2 * 255)), -2)   # 计算其离散傅里叶变换
    df = fs / 512      # 谱图的频率间隔
    val, loc = local_maxium(U)   # loc存放共振峰的位置
    ll = len(loc)
    pp = np.zeros(ll)      # 存储共振峰幅值
    F = np.zeros(ll)       # 存储共振峰频率
    Bw = np.zeros(ll)      # 存储共振峰带宽
    for k in range(ll):
        m = loc[k]   # 定位第k个共振峰
        m1, m2 = m - 1, m + 1      # 找到其左右两边的相邻点
        p = val[k]   # 计算此共振峰幅值
        p1, p2 = U[m1], U[m2]      # 给出左右相邻点的幅值
```

```
        # 根据式4-3-12来计算拟合函数的系数
        aa = (p1 + p2) / 2 - p
        bb = (p2 - p1) / 2
        cc = p
        dm = -bb / 2 / aa     # 极大值
        pp[k] = -bb * bb / 4 / aa + cc    # 中心频率对应的功率谱
        m_new = m + dm    # 共振峰的中心频率
        bf = -np.sqrt(bb * bb - 4 * aa * (cc - pp[k] / 2)) / aa         # 带宽的计算
        F[k] = (m_new - 1) * df
        Bw[k] = bf * df
    return F, Bw, pp, U, loc
```

6. 语音增强实验

6.1　基于谱减法的语音降噪实验

6.1.1 实验目的

1）了解语音降噪的重要性和必要性；
2）熟练掌握语音降噪的仿真方法；
3）了解一般谱减法的基本原理；
4）掌握基于谱减法的语音降噪原理；
5）编程实现基于谱减法的语音降噪函数，并仿真验证。

6.1.2 实验原理

1、语音降噪的意义

语音降噪主要研究如何利用信号处理技术消除信号中的强噪声干扰，从而提高输出信噪比以提取出有用信号的技术。消除信号中噪声污染的通常方法是让受污染的信号通过一个能抑制噪声而让信号相对不变的滤波器，此滤波器从信号不可检测的噪声场中取得输入，将此输入加以滤波，抵消其中的原始噪声，从而达到提高信噪比的目的。

然而，由于干扰通常都是随机的，从带噪语音中提取完全纯净的语音几乎不可能。在这种情况下，语音增强的目的主要有两个：一是改进语音质量，消除背景噪音，使听者乐于接受，不感觉疲劳，这是一种主观度量；二是提高语音可懂度，这是一种客观度量。这两个目的往往不能兼得，所以实际应用中总是视具体情况而有所侧重。

根据语音和噪声的特点，有多种不同的语音增强算法。比较常用的有谱减法、维纳滤波法、卡尔曼滤波法、自适应滤波法等。此外，随着科学技术的发展又出现了一些新的增强技术，如基于神经网络的语音增强、基于 HMM 的语音增强、基于听觉感知的语音增强、基于多分辨率分析的语音增强、基于语音产生模型的线性滤波法、基于小波变换的语音增强方法、梳状滤波法、自相关法、基于语音模型的语音增强方法等。

2、带噪语音模型

一般噪声都假设其是加性的、局部平稳的、噪声与语音统计独立或不相关。因此，带噪语音模型表达式如下：

$$y(n)=s(n)+d(n) \qquad (6\text{-}1)$$

其中 $s(n)$ 表示纯净语音，$d(n)$ 表示噪声，$y(n)$ 表示带噪语音。带噪语音模型如图 6-1 ~~图 6-1~~ 所示：

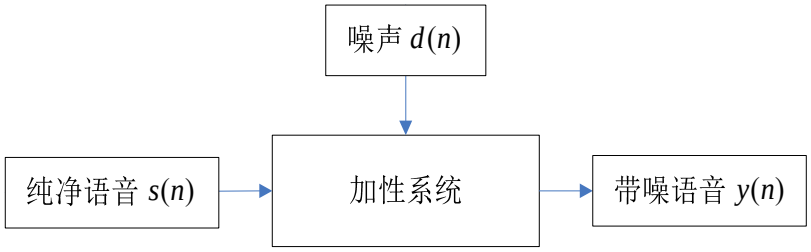

图 6-1 带噪语音模型

噪声的局部平稳，是指一段带噪语音中的噪声，具有和语音段开始前那段噪声相同的统计特性，且在整个语音段中保持不变。也就是说，可以根据语音开始前那段噪声来估计语音中所叠加的噪声统计特性。仿真实验中采用白噪声作为测试噪声源。

3、基本谱减法

设语音信号的时间序列为 $x(n)$，加窗分帧处理后得到第 i 帧语音信号为 $x_i(m)$，帧长为 N。任何一帧语音信号 $x_i(m)$ 做 FFT 后为

$$X_i(k)=\sum_{m=0}^{N-1}x_i(m)\exp(j\frac{2\pi mk}{N})\quad \text{k=0,1,...,N-1} \tag{6-2}$$

对 $X_i(k)$ 求出每个分量的幅值和相角，幅值是 $|X_i(k)|$，相角为

$$X_{angle}^{i}(k)=\arctan[\frac{\mathrm{Im}(X_i(k))}{\mathrm{Re}(X_i(k))}] \tag{6-3}$$

已知前导无话段（噪声段）时长为 IS，对应的帧数为 NIS，可以求出该噪声段的平均能量为

$$D(k)=\frac{1}{NIS}\sum_{i=1}^{NIS}|X_i(k)|^2 \tag{6-4}$$

谱减公式为

$$|\hat{X}_i(k)|^2=\begin{cases}|X_i(k)|^2-a\times D(k) & |X_i(k)|^2\geq a\times D(k)\\ b\times D(k) & |X_i(k)|^2<a\times D(k)\end{cases} \tag{6-5}$$

式中，a 和 b 是两个常数，a 称为过减因子；b 称为增益补偿因子。

谱减后的幅值为 $|\hat{X}_i(k)|$，结合原先的相角 $X_{angle}^{i}(k)$，利用快速傅里叶逆变换求出增强后的语音序列 $\hat{x}_i(m)$。

整个算法的原理如图 6-2 ~~图 6-2~~ 所示。

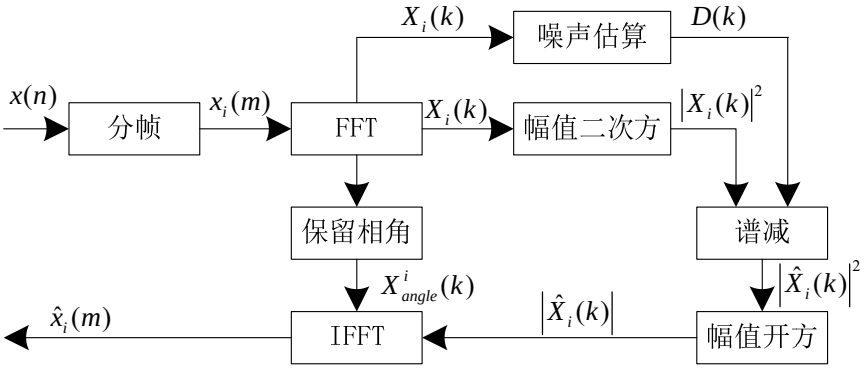

图 6-2 基本谱减法原理图

4、Boll 的改进谱减法

1979 年，S. F. Boll 提出一种改进的谱减法。主要的改进点为：

1）在谱减法中使用信号的频谱幅值或功率谱

改进的谱减公式为

$$\left|\hat{X}_i(k)\right|^\gamma = \begin{cases} \left|X_i(k)\right|^\gamma - \alpha \times D(k) & \left|X_i(k)\right|^\gamma \geq \alpha \times D(k) \\ \beta \times D(k) & \left|X_i(k)\right|^\gamma < \alpha \times D(k) \end{cases} \tag{6-6}$$

噪声段的平均谱值为

$$D(k) = \frac{1}{NIS}\sum_{i=1}^{NIS}\left|X_i(k)\right|^\gamma \tag{6-7}$$

式中，当 γ 为 1 时，算法相当于用谱幅值做谱减法；当 γ 为 2 时，算法相当于用功率谱做谱减法。式中，α 为过减因子；β 为增益补偿因子。

2）计算平均谱值

在相邻帧之间计算平均值：

$$Y_i(k) = \frac{1}{2M+1}\sum_{j=-M}^{M} X_{i+j}(k) \tag{6-8}$$

利用 $Y_i(k)$ 取代 $X_i(k)$，可以得到较小的谱估算方差。

3）减少噪声残留

在减噪过程中保留噪声的最大值，从而在谱减法中尽可能地减少噪声残留，从而削弱“音乐噪声”。

$$D_i(k) = \begin{cases} D_i(k) & D_i(k) \geq \max\left|N_R(k)\right| \\ \min\{D_j(k)\left|\mathrm{j} \in [\mathrm{i\text{-}1,i,i+1}]\right.\} & D_i(k) < \max\left|N_R(k)\right| \end{cases} \tag{6-9}$$

此处，$\max\left|N_R(k)\right|$ 代表最大的噪声残余。

5、语音质量性能指标

语音质量包括两方面内容：可懂度和自然度。前者对应语音的辨识水平。而后者则是衡量语音中字、单词和句的自然流畅程度。总体上可以将语音质量评价分为两大类:主观评价和客观评价。

5.1 主观指标

主观评价以人为主体来评价语音的质量。主观评价方法的优点是可以真实反映人对语音质量的实际感觉，目前应用广泛。常用方法有平均意见得分(Mean Opinion Score，MOS 得分)、可懂度评价(Diagnostic Rhyme Test，DRT 得分)、判断满意度测量(Diagnostic Acceptability Measure，DAM 得分)等。语音质量的主观评价需要大量的测试者、大量次数的测听实验，以得到普适结果。但是由于语音质量的主观评价耗时耗力，因此不容易实现。

为了克服主观评价的缺点，人们寻求一种能够方便快捷地给出语音质量评价的客观评价方法。不过值得注意的是，研究语音客观评价的目的不是要用客观评价来完全替代主观评价，而是使客观评价成为一种既方便快捷又能准确预测出主观评价价值的语音质量评价手段。尽管客观评价具有省时省力等优点，但它还不能反映人对语音质量的全部感觉，而且当前的大多客观评价方法都是以语音信号的时域、频域及变换域等特征参量作为评价依据，没有涉及到语义、语法、语调等影响语音质量主观评价的重要因素。

5.2 客观指标

语音质量客观评价方法采用某个特定的参数去表征语音通过增强或编码系统后的失真程度，并以此来评估处理系统的性能优劣。

1）信噪比(Signal-to-Noise Ratio，SNR)

SNR 一直是衡量针对宽带噪声失真的语音增强算的常规方法，其定义如下：

$$SNR = 10\log_{10}\frac{\sum_{n=0}^{N-1}s^2(n)}{\sum_{n=0}^{N-1}d^2(n)} \qquad (6\text{-}10)$$

式中，$\sum_{n=0}^{N-1}s^2(n)$ 表示信号的能量；$\sum_{n=0}^{N-1}d^2(n)$ 表示噪声的能量。

但要计算信噪比必须知道纯净语音信号，在实际应用中这是不可能的。因此，SNR 主要用于纯净语音信号和噪声信号都已知的算法仿真中。计算整个时间轴上的语音信号与噪声信号的平均功率之比得到信噪比。由于语音信号是一种缓慢变化的短时平稳信号，在不同时间段上的信噪比也应不一样，可以采用分段信噪比改善此问题。

2）PESQ(Perceptual Evaluation of Speech Quality)

2001 年 2 月，ITU-T 推出了 P.862 标准《窄带电话网络端到端语音质量和话音编解码器质量的客观评价方法》，推荐使用语音质量感知评价 PESQ 算法，该方法是基于输入-输出方式的典型算法，效果良好。

PESQ 算法需要一个原始参考信号和一个带噪的衰减信号。首先将两个待比较的语音信号依次调整电平、输入滤波器滤波、时间对准和补偿、听觉变换，再分别提取两路信号的参数，综合其时频特性，得到 PESQ 分数，最终将这个分数映射到主观平均意见分(MOS)。PESQ 得分范围在-0.5~4.5 之间，得分越高表示语音质量越好。

6.1.3 实验步骤及要求

1.实验步骤

运行 Python——>新建 py 文件——>编写 py 程序——>编译并调试。

2、实验要求

注：由实验内容可知，谱减法是以帧为单位进行计算的。为此，如果从增强后的语音序列 $\hat{x}_i(m)$ 获得增强后的语音信号 $\hat{x}(m)$，需要进行合成操作。此处，调用 E.Zavarehei 编写的 OverlapAdd2 函数进行处理。函数定义如下：

名称：OverlapAdd2

功能：把频域中的每帧信号的频谱幅值参数和相位参数合成为连续的语音信号。

调用格式：ReconstructedSignal=OverlapAdd2(XNEW,yphase,windowLen,ShiftLen);

说明：输入参数 XNEW 是频谱幅值（只包含正频率部分）；yphase 是频谱相角（只包含正频率部分）；windowLen 为帧长；ShiftLen 是帧移。output 是合成后的连续语音信号。

1）编写基本谱减法函数：根据基本谱减法的原理，编写函数，并基于测试语音 C6_1_y.wav 叠加不同信噪比的噪声，进行降噪测试，效果如图 6-3 ~~图 6-3~~。

函数定义如下：

名称：SpectralSub

功能：谱减法语音降噪

调用格式：output=SpectralSub(signal,wlen,inc,NIS,a,b)

输入参数：signal 是输入的含噪语音信号；wlen 为窗函数或窗长；inc 是帧移；NIS 是前导无话段帧数；a 为过减因子；b 为增益补偿因子

输出参数：output 是降噪后的信号

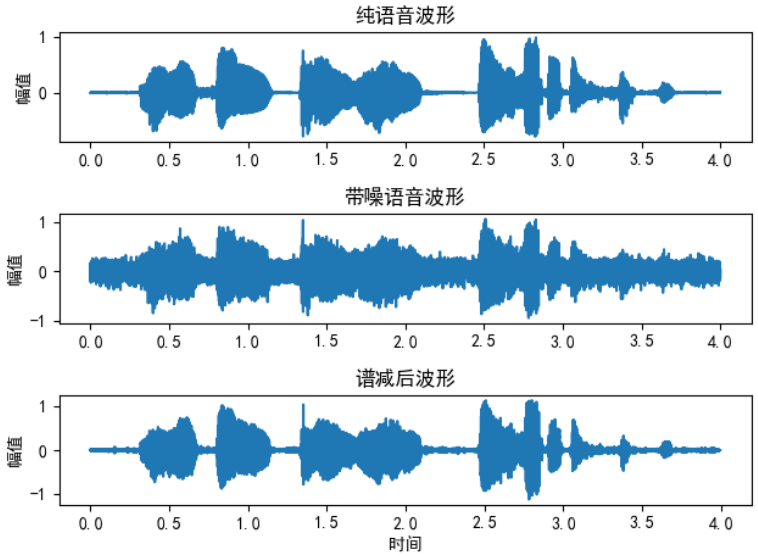

图 6-3 基本谱减法降噪效果图

2）编写 Boll 改进谱减法函数：根据 Boll 的改进谱减法的原理，编写函数，并基于测试语音 C6_1_y.wav 叠加不同信噪比的噪声，进行降噪测试，效果如图 6-4 ~~图 6-4~~。

函数定义如下：

名称：SpectralSubIm

功能：基于 Boll 的改进谱减法语音降噪

调用格式：output=SpectralSubIm(signal,wind,inc,NIS,Gamma,Beta)

输入参数：signal 是输入的含噪语音信号；wind 为窗函数或窗长；inc 是帧移；NIS 是前导无话段帧数；Gamma 和 Beta 是算法参数

输出参数：output 是降噪后的信号

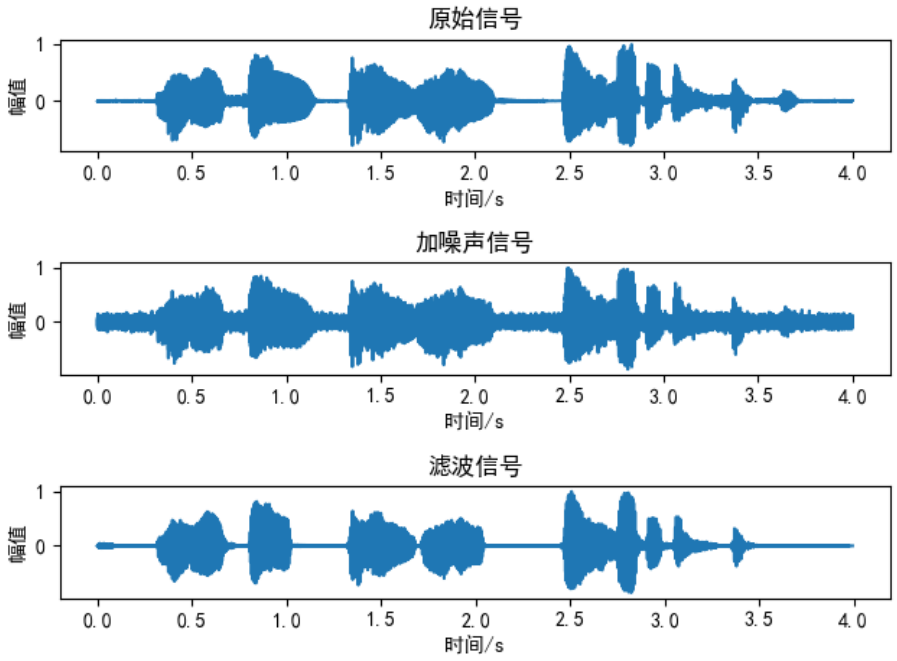

图 6-4 改进谱减法降噪效果图

提示：可以借鉴基于对数谱的语音帧和噪声帧判别函数进行语音帧和噪声帧的判断，从而更新噪声谱。

6.6.1.4 参考例程

【Boll 改进谱减法函数】

```
def SpectralSubIm(signal, wlen, inc, NIS, Gamma, Beta):
    wnd = np.hamming(wlen)
    y = enframe(signal, wnd, inc)
    y_fft=np.fft.fft(y, n=wlen, axis=1)
    y_angle = np.angle(y_fft[:,:int(y_fft.shape[1]/2)+1])
    Y = np.power(np.abs(y_fft[:,:int(y_fft.shape[1]/2)+1]),Gamma)
    fn, flen = Y.shape
    N = np.mean(Y[0:NIS,:],axis=0)
    NRM = np.zeros(N.shape)
    X = np.zeros((fn,flen))
    NoiseCounter = 0
```

```
NoiseLength = 9

YS = Y
for i in range(1,fn-1):
    YS[i, :]=(Y[i-1, :] + Y[i, :]+Y[i+1, :]) / 3

for i in range(fn):
    NoiseFlag, SpeechFlag, NoiseCounter, Dist = vad_LogSpec(np.power(Y[i, :],(1 / Gamma)), np.power(N, (1 / Gamma)), NoiseCounter)    # 基于频谱距离的 VAD 检测
    if SpeechFlag == 0:
        N = (NoiseLength * N + Y[i, :]) / (NoiseLength + 1)# 更新并平滑噪声
        NRM = np.maximum(NRM,YS[i, :]-N)    # 更新最大的噪声残余
        X[i, :]=Beta * Y[i, :]
    else:
        D = YS[i, :]-N    # 谱减
        if i > 1 and i < fn:        # 减少噪声残留项
            for j in range(len(D)):
                if D[j] < NRM[j]:
                    D[j] = np.min([D[j],YS[i-1, j] - N[j],YS[i + 1, j] - N[j]])
        X[i, :]=np.maximum(D, 0)
output = OverlapAdd2(np.power(X, (1 / Gamma)), y_angle, wlen, inc)
return output
```

6.2 基于维纳滤波的语音降噪实验

6.2.1 实验目的

1）了解维纳滤波的意义；
2）掌握基于维纳滤波的语音降噪原理；
3）编程实现基于维纳滤波的语音降噪函数，并仿真验证。

6.2.2 实验原理

1、维纳滤波的基本原理

基本维纳滤波就是用来解决从噪声中提取信号问题的一种过滤(或滤波)方法。它基于平稳随机过程模型，且假设退化模型为线性空间不变系统的。实际上这种线性滤波问题，可以看成是一种估计问题或一种线性估计问题。基本的维纳滤波是根据过去和当前全部的观察数据来估计信号的当前值，它的解是以均方误差最小条件下所得到的系统的传递函数 $H(z)$ 或单位样本响应 $h(n)$ 的形式给出的，因此更常称这种系统为最佳线性过滤器或滤波器。设计维纳滤波器的过程就是寻求在最小均方误差下滤波器的单位样本响应 $h(n)$ 或传递函数 $H(z)$ 的表达式，其实质是解维纳-霍夫(Wiener-Hopf)方程。

设带噪语音信号为

$$x(n)=s(n)+v(n) \tag{6-11}$$

其中，s(n)表示带噪信号，$v(n)$ v(n表示噪声，则经过维纳滤波器 $h(n)$ 的输出响应y(n)为

$$y(n)=x(n)*h(n)=\sum_m h(m)x(n-m) \tag{6-12}$$

理论上，$x(n)$ 通过线性系统 $h(n)$ 后得到的 $y(n)$ 应尽量接近于 $s(n)$，因此 $y(n)$ 为 $s(n)$ 的估计值，可用 $\hat{s}(n)$ 表示，即

$$y(n)=\hat{s}(n) \tag{6-13}$$

从式(6-12)~~(6-12)~~可知，卷积形式可以理解为从当前和过去的观察值 $x(n)$，$x(n-1)$，$x(n-2)\cdots x(n-m)$，…来估计信号的当前值 $\hat{s}(n)$。因此，用 $h(n)$ 进行滤波实际上是一种统计估计问题。

$\hat{s}(n)$ 按最小均方误差准则使 $\hat{s}(n)$ 和 $s(n)$ 的均方误差 $\xi=E[e^2(n)]=E[\{s(n)-\hat{s}(n)\}^2]$ 达到最小。使 ξ 最小的充要条件是 ξ 对于 $h(n)$ 的偏导数为零，即

$$\frac{\partial\xi}{\partial h(n)}=\frac{\partial E\{e^2(n)\}}{\partial h(n)}=E[2e(n)\frac{\partial e(n)}{\partial w(n)}]=-E[2e(n)x(n-m)]=0 \tag{6-14}$$

上式整理可得

$$E[\{s(n)-\hat{s}(n)\}x(n-m)]=0 \tag{6-15}$$

这就是正交性原理或投影原理。将式（6-12）~~（6-12）~~代入式（6-15）~~（6-15）~~可得

$$E[s(n)x(n-m)-\sum_{l}h(l)E\{x(n-l)x(n-m)\}]=0 \tag{6-16}$$

已知，$s(n)$ 和 $d(n)$ 是联合宽平稳的，因此令 $x(n)$ 的自相关函数 $R_x(m-l)=E\{x(n-m)x(n-l)\}$，$s(n)$ 与 $x(n)$ 的互相关函数 $R_{sx}(m)=E\{s(n)x(n-m)\}$ 则式（6-16）~~（6-16）~~可变为

$$\sum_{l}h(l)R_x(m-l)=R_{sx}(m) \tag{6-17}$$

式（6-17）~~（6-17）~~称为维纳滤波器的标准方程或维纳-霍夫（Wiener-Hopf）方程。如果已知 $R_{sx}(m)$ 和 $R_x(m-l)$，那么解此方程即可求得维纳滤波器的冲激响应。

当 l 从 0 到 $N-1$ 取有限个整数值时，设滤波器冲激响应序列的长度为 N，冲激响应矢量为

$$h=[h(0)h(1)....h(N-1)]^T \tag{6-18}$$

滤波器输入数据矢量为

$$x(n)=[x(n)x(n-1)...x(n-N+1)]^T \tag{6-19}$$

则滤波器的输出为

$$y(n)=\hat{s}(n)=x^T(n)h=h^Tx(n) \tag{6-20}$$

这样，式（6-17）~~（6-17）~~所示的维纳-霍夫方程可写成

$$Q=Rh \tag{6-21}$$

其中，$Q=E[x(n)s(n)]$ 是 $s(n)$ 与 $x(n)$ 的互相关函数，它是一个 N 维列矢量；R 是 $x(n)$ 的自相关函数，是 N 阶方阵 $R=E[x(n)x^T(n)]$，则最优的维纳滤波器的冲激响应为

$$h_{opt}=R^{-1}Q \tag{6-22}$$

如果对式（6-17）~~（6-17）~~进行傅里叶变换可得，

$$H(k)=\frac{P_{sx}(k)}{P_x(k)} \tag{6-23}$$

式中，$P_x(k)$ 为 $x(n)$ 的功率谱密度；$P_{sx}(k)$ 为 $x(n)$ 与 $s(n)$ 的互功率谱密度。

由于 $v(n)$ 与 $s(n)$ 互不相关，即 $R_{sv}(k)=0$，则可得

$$P_{sx}(k)=P_s(k) \tag{6-24}$$

$$P_x(k)=P_s(k)+P_v(k) \tag{6-25}$$

此时，式（6-23）~~（6-23）~~可变为

$$H(k)=\frac{P_s(k)}{P_s(k)+P_v(k)} \tag{6-26}$$

该式为维纳滤波器的谱估计器。此时，$\hat{s}(n)$ 的频谱估计值为

$$\hat{S}(k)=H(k)\cdot X(k) \tag{6-27}$$

此外，$H(k)$ 还可以写为

$$H(k)=\frac{\lambda_s(k)}{\lambda_s(k)+\lambda_v(k)} \tag{6-28}$$

式中，$\lambda_s(k)$ 和 $\lambda_s(k)$ 分别为第 k 个频点上的信号和噪声的功率谱。

传统的维纳滤波法需要估计出纯净语音信号的功率谱，一般用类似谱减法的方法得到，即用带噪语音功率谱减去估计到的噪声功率谱，这种方法会存在残留噪声大的问题。

2、基于先验信噪比的维纳滤波基本原理

改进的维纳滤波器为基于先验信噪比的维纳滤波器，实现框图如图 6-5 ~~图 6-5~~所示。

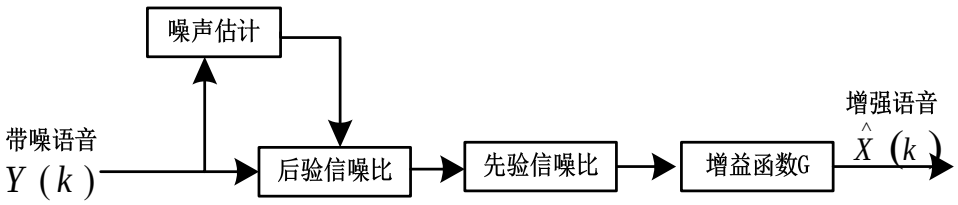

图 6-5 维纳滤波原理框图

对于第 m 帧带噪语音信号：

$$y_m(n)=s_m(n)+n_m(n) \tag{6-29}$$

式中 $s_m(n)$ 是第 m 帧纯净语音信号，$n_m(n)$ 为第 m 帧噪声信号，维纳滤波器就是在最小均方误差准则（MSE）下实现对语音信号 $s_m(n)$ 的估计。在 $s_m(n)$ 与 $n_m(n)$ 不相关且均为平稳随机过程条件下，对（6-29）~~（6-29）~~式进行离散傅里叶变换，得：

$$Y(m,k)=S(m,k)+N(m,k) \tag{6-30}$$

谱增益函数为

$$G(m,k)=\frac{\zeta(m,k)}{1+\zeta(m,k)}=\frac{SNR_{prio}(m,k)}{1+SNR_{prio}(m,k)} \tag{6-31}$$

式中 $\zeta(m,k)$（SNR_{prio}）为先验信噪比，m 为帧号，k 为频点。

则第 m 帧增强语音可表示为：

$$\hat{S}(m,k)=G(m,k)\cdot Y(m,k) \tag{6-32}$$

采用直接判决（Decision-Directed）法来估计先验信噪比 SNR_{prio}：

$$SNR_{prio}(m,k)=\alpha\cdot SNR_{prio}(m-1,k)+(1-\alpha)\cdot\max(SNR_{post}(m,k)-1,0) \quad (6\text{-}33)$$

$$SNR_{post}(m,k)=\frac{|Y(m,k)|^2}{|\hat{N}(m,k)|} \quad (6\text{-}34)$$

式中，SNR_{post} 表示后验信噪比，$Y(m,k)$ 表示估计的第 m 帧信号的功率谱，$\hat{N}(m,k)$ 表示估计的第 m 帧噪声功率谱。

6.2.3 实验步骤及要求

1.实验步骤

运行 Python——>新建 py 文件——>编写 py 程序——>编译并调试。

2、实验要求

1）编写基本维纳滤波函数：根据基本维纳滤波法的原理，编写函数，并基于测试语音 C6_2_y.wav 叠加不同信噪比的噪声，进行降噪测试，效果如图 6-6 ~~图 6-6~~。

函数定义如下：

名称：weina_Norm

功能：基本维纳滤波算法

调用格式：enhanced=weina_Norm(x,wind,inc,NIS,alpha,beta)

输入参数：x 是输入的含噪语音信号；wlen 为窗函数或窗长；inc 是帧移；NIS 是前导无话段帧数；alpha 和 beta 是谱减法的参数

输出参数：enhanced 是降噪后的信号

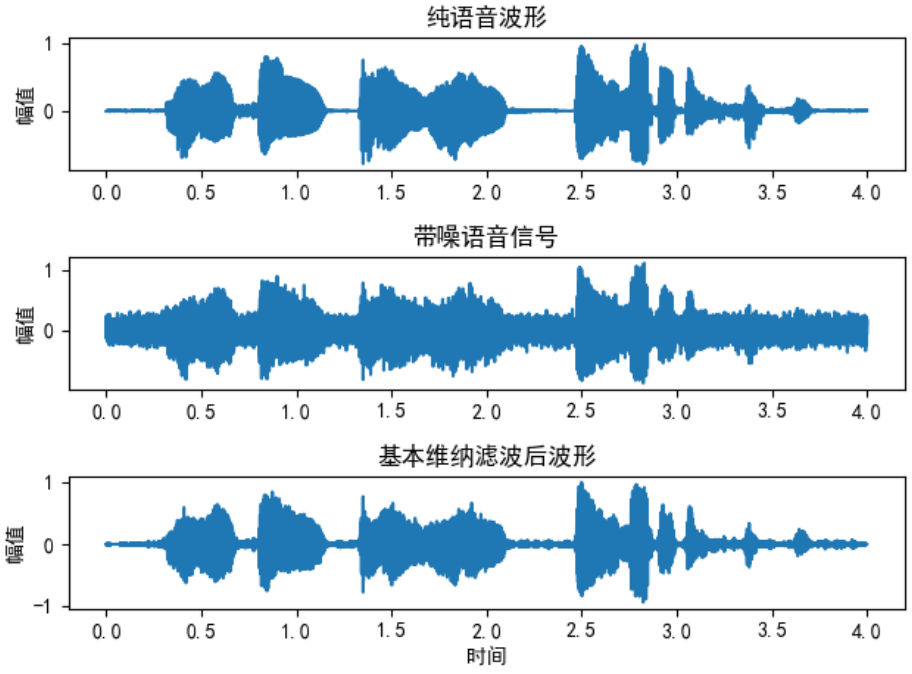

图 6-6 基本维纳滤波降噪效果图

2）编写基于先验信噪比的维纳滤波函数：根据基于先验信噪比的维纳滤波法的原理，编写函数，并基于测试语音 C6_2_y.wav 叠加不同信噪比的噪声，进行降噪测试，效果如图 6-7 ~~图 6-7~~。函数定义如下：

名称：weina_Im

功能：基于先验信噪比的维纳滤波算法

调用格式：enhanced=weina_Im(x,wind,inc,NIS,alpha)

输入参数：x 是输入的含噪语音信号；wlen 为窗函数或窗长；inc 是帧移；NIS 是前导无话段帧数；alpha 是信噪比平滑参数

输出参数：enhanced 是降噪后的信号

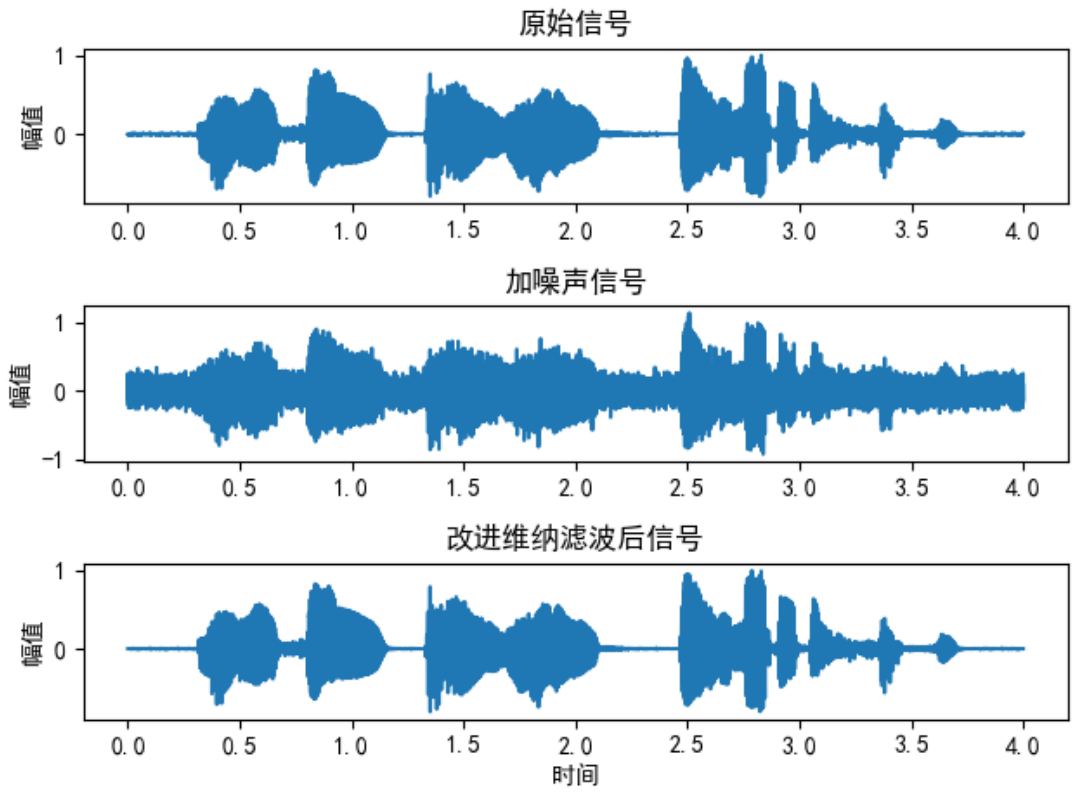

图 6-7 改进维纳滤波降噪效果图

6.2.4 思考题

1）结合上节的端点检测算法，尝试改善上述函数，只在语音段进行维纳滤波。

函数定义如下：

名称：wiener_Vad

调用格式：output=wiener_Vad(signal,wind,inc,NIS,alpha)

输入参数：x 是输入的含噪语音信号；wlen 为窗函数或窗长；inc 是帧移；NIS 是前导无话段帧数；alpha 是信噪比平滑参数

输出参数：enhanced 是降噪后的信号

6.2.5 参考例程

【基本维纳滤波算法函数】

```
def weina_Norm(x, wlen, inc, NIS, alpha, beta):
    """
    使用维纳滤波
    :param x:输入语音信号
    :param wind:窗的类型
    :param framesize:帧长
    :param overlap:帧重叠长度
    :param NIS:无声帧帧数
    :param alpha,beta:抑制参数
    :return:
    """
    wnd = np.hamming(wlen)
    y = enframe(x, wnd, inc)                    #分帧
    fn, flen = y.shape                          #fn 为帧数，flen 为频点数
    y_a = np.abs(np.fft.fft(y,axis=1))
    y_a2 = np.power(y_a, 2)
    Nt = np.mean(y_a2[0:NIS,:], axis=0)
    signal = np.zeros((flen))
    yw = np.zeros((fn, flen),dtype=complex)
    yt = np.zeros((fn,flen))
    for i in range(fn):                         #对每一帧循环
        frame = y[i,:]
        y_fft = np.fft.fft(frame)
        y_fft2 = np.power(abs(y_fft),2)
        for k in range(flen):                               #每一帧的每一个频点循环
            if abs(y_fft2[k]) > alpha*Nt[k]:
                signal[k] =y_fft2[k] - alpha*Nt[k]
                if signal[k] < 0:
                    signal[k] = 0
            else:
                signal[k] = beta*Nt[k]
```

```
        Hw = signal/(signal+Nt)
        yw[i,:] = Hw*y_fft
        yt[i,:] = np.fft.ifft(yw[i,:])
sig = np.zeros(int((fn-0.5) * inc + wlen))
for i in range(fn):
        start = i * inc
        sig[start:start + flen] += yt[i, :]
        return sig
```