重庆交通大学

CHONGQING JIAOTONG UNIVERSITY

《声音及语音信号处理》
实验报告

学　　院：信息科学与工程学院

专　　业：人工智能

学 生 姓 名：李财华

学　　号：632107100216

评 阅 教 师：杨祥立

完 成 时 间：2023 年 12 月 2 日

实验一 声音采集与语音读写实验

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 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。

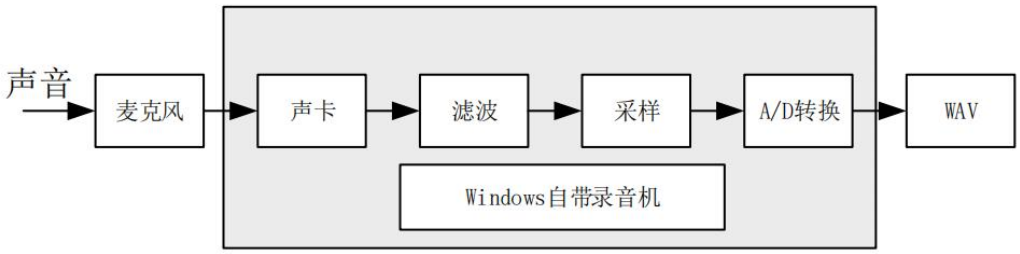


图 2-1 PC 机语音信号采集原理图

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

3 实验结果

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

```python
frames = [] # 创建一个新列表，用于存储采集到的的数据\
pa = pyaudio.PyAudio() # 实例化一个 Pyaudio 对象
def record(Rate,filename):
    frames.clear()
    stream = pa.open( format=FORMAT, channels=CHANNELS, rate=Rate, input=True,
    frames_per_buffer=CHUNK)
    print("开始第录制"+filename)
    #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(filename+"录制完成")

    # 保存录音数据为WAV文件
def save(filename):
    wf = wave.open(filename, 'wb')
    wf.setnchannels(CHANNELS)
    wf.setsampwidth(pa.get_sample_size(FORMAT))
    wf.setframerate(RATE)
    wf.writeframes(b''.join(frames))
    wf.close()
    print(f"录音已保存为 {filename}")
```

图 1

2）使用 wavfile.read 函数读取 sweep.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。

```python

# 读取 WAV 文件
sample_rate, data = wavfile.read("01.wav")
# 获取信号的时长（秒）
duration = len(data) / sample_rate
# 创建时间轴（横轴）数据
time_axis = np.arange(data.shape[0]) / sample_rate
# 归一化音频数据
normalized_data = data / np.max(np.abs(data),axis=0)
# 绘制图形
plt.figure(figsize=(10, 4))
plt.plot(time_axis, normalized_data)
plt.xlabel("时间/s")
plt.ylabel("幅度")
plt.title("音频:1.wav")
plt.grid(True)
plt.show()

```

图 2

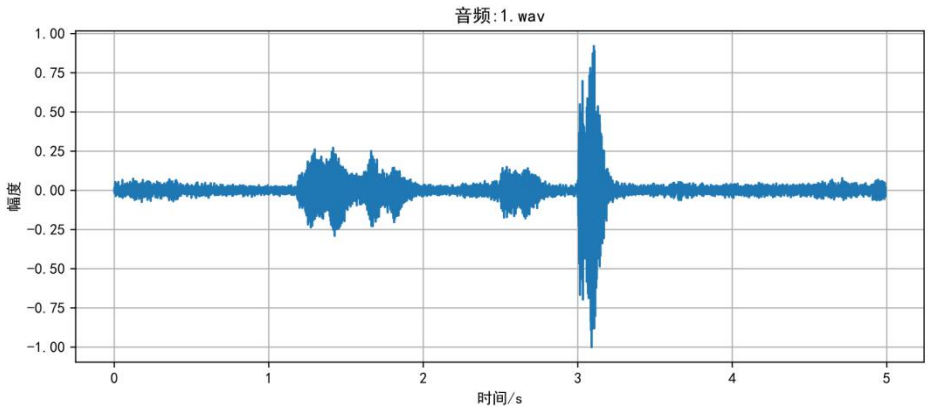


图 3

4 思考与总结

通过这个实验，我深入了解了 Python 中采集语音信号的原理和常用命令，掌握了基于 Python 的语音文件的创建、读写等基本操作。使用 matplotlib 库绘制语音信号波形并进行标注，提高了图的可读性。这次实验让我更加熟悉了 Python 在语音信号处理领域的应用，为今后的学习和实践奠定了基础。

实验二 声音及语音编辑实验

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 实验原理

1.信号的叠加

两个信号 x1 和 x2 ，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

2.信号的卷积

两序列 x_1 和 x_2 的卷积 y 定义为：

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

卷积运算满足交换律，即：

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

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

3 实验结果

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

```
frames = [] # 创建一个新列表，用于存储采集到的的数据\
pa = pyaudio.PyAudio() # 实例化一个 Pyaudio 对象
def record(Rate,filename):
    frames.clear()
    stream = pa.open( format=FORMAT, channels=CHANNELS, rate=Rate, input=True,
    frames_per_buffer=CHUNK)
    print("开始第录制"+filename)
    #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(filename+"录制完成")
    stream.stop_stream() # 关闭数据流
    stream.close() # 关闭声卡
    pa.terminate()
    # 保存录音数据为WAV文件
def save(filename,frames,rate):
    wf = wave.open(filename, 'wb')
    wf.setnchannels(CHANNELS)
    wf.setsampwidth(pa.get_sample_size(FORMAT))
    wf.setframerate(rate)
    wf.writeframes(b''.join(frames))
    wf.close()
    print(f"录音已保存为 {filename}")
```

```
filename='1.wav'
record(Rate=RATE,filename=filename)
save(filename=filename,frames=frames,rate=RATE)
```

图 4 录制语音代码

```python
# 读取语音信号并归一化
sample_rate, audio_signal = wavfile.read('../01.wav')
audio_signal = audio_signal.astype(float) / np.max(np.abs(audio_signal))
# 生成随机信号并归一化
random_signal = np.random.randn(len(audio_signal))
random_signal = random_signal.astype(float) / np.max(np.abs(random_signal))
random_signal_p = random_signal
# 两段信号线性叠加
convolved_signal = audio_signal+random_signal_p*0.01
# 时间轴
time_axis = np.arange(0, len(audio_signal)) / sample_rate
# 绘制语音信号、随机信号和叠加信号
plt.figure(figsize=(10, 6))
plt.subplot(3, 1, 1)
plt.plot(time_axis, audio_signal, label='语音信号')
plt.title("原始信号")
plt.ylabel('归一化值')
plt.grid(True)

plt.subplot(3, 1, 2)
plt.plot(time_axis, random_signal_p)
plt.title("随机序列")
plt.ylabel('归一化值')
plt.grid(True)

plt.subplot(3, 1, 3)
plt.plot(time_axis, convolved_signal)
plt.title("卷积信号")
plt.xlabel('时间 (秒)')
plt.ylabel('归一化值')
plt.grid(True)

plt.tight_layout()
plt.show()
```

图 5

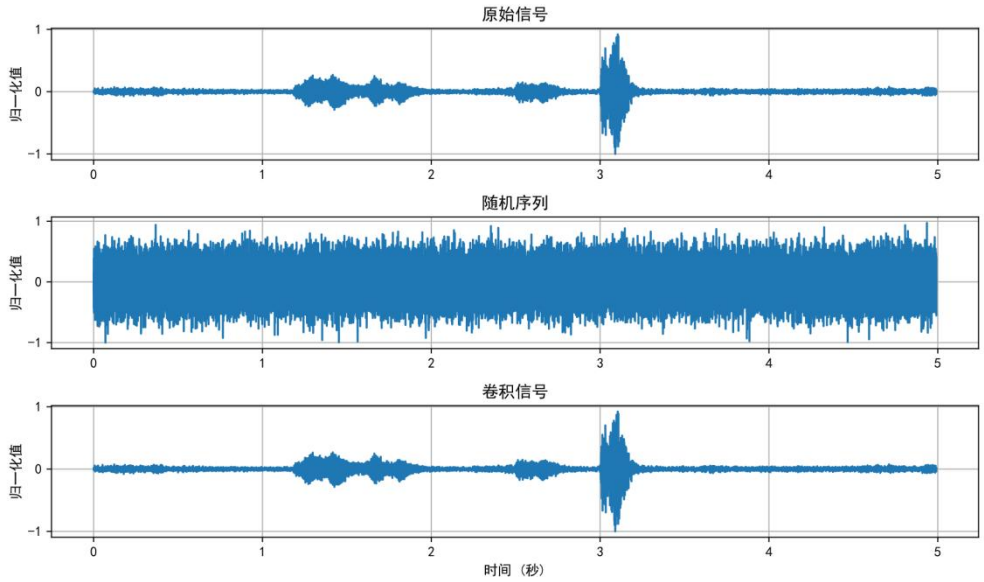


图 6

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

```python
# 1. 读取语音信号并归一化
sample_rate, audio_signal = wavfile.read('../01.wav')
audio_signal = audio_signal.astype(float) / np.max(np.abs(audio_signal))
# 生成随机信号
random_signal = random_signal.astype(float) / np.max(np.abs(random_signal))
# 两段信号卷积
convolved_signal = np.convolve(audio_signal,random_signal , mode='same')
# 时间轴
time_axis = np.arange(0, len(audio_signal)) / sample_rate
# 绘制语音信号、随机信号和叠加信号
plt.figure(figsize=(12, 6))
plt.subplot(3, 1, 1)
plt.plot(time_axis, audio_signal, label='语音信号')
plt.title("原始信号")
plt.ylabel('归一化值')
plt.grid(True)

plt.subplot(3, 1, 2)
plt.plot(time_axis, random_signal_p)
plt.title("随机序列")
plt.ylabel('归一化值')
plt.grid(True)

plt.subplot(3, 1, 3)
plt.plot(time_axis, convolved_signal)
plt.title("卷积信号")
plt.xlabel('时间 (秒)')
plt.ylabel('归一化值')
plt.grid(True)

plt.tight_layout()
plt.show()
```

图 7

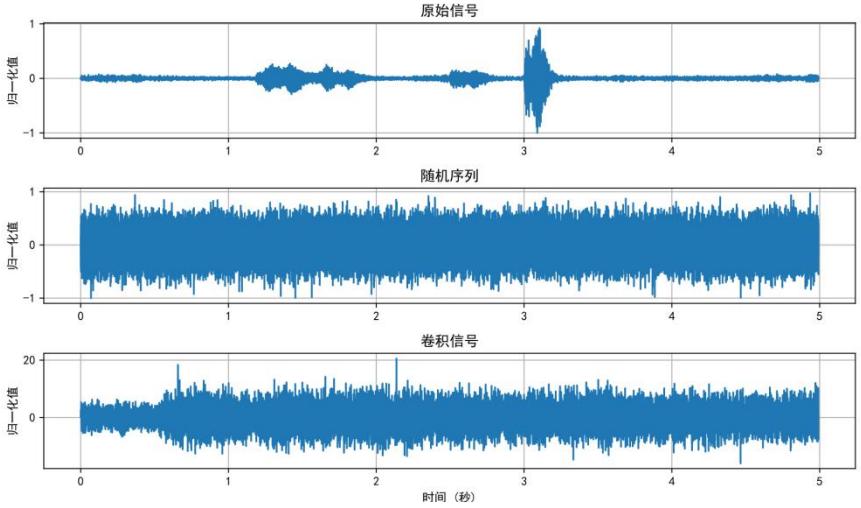


图 8

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，然后播放，比较采样频率改变对语音信号的影响。

```python
# 读取 WAV 文件
fs, sound = wavfile.read("../01.wav")
# 获取信号的时长（秒）
duration = len(sound) / fs
# 创建时间轴（横轴）数据
time_axis = np.linspace(0, duration, len(sound))
# 归一化音频数据
normalized_data = sound / np.max(np.abs(sound))
# 绘制图形
plt.figure(figsize=(10, 8))
plt.subplot(311)
plt.plot(time_axis, normalized_data, linewidth=1)
# plt.xlabel("时间/s")
plt.ylabel("归一化幅度")
plt.title("原始信号")
plt.grid(True)


plt.subplot(312)
sound2=signal.resample(sound, int(len(sound))*2)
Sound2=sound2
sound2_max=np.absolute(sound2).max()
sound2=sound2/sound2_max
# 构建时间轴数据
f2=2*fs
t2=np.array([i/f2 for i in range(len(sound2))])
plt.subplot(312)
plt.plot(t2, sound2, linewidth=1)
# plt.xlabel("时间/s")
plt.ylabel("归一化幅度")
plt.title("2倍采样率信号")
plt.grid(True)

# 将采样点变为原来的1/2
plt.subplot(313)
sound3=signal.resample(sound, int(sound.size/2))
Sound3=sound3
sound3_max=np.absolute(sound3).max()
sound3=sound3/sound3_max
# 构建时间轴数据
f3=fs//2
t3=np.array([i/f3 for i in range(len(sound3))])
plt.plot(t3, sound3, linewidth=1)
plt.xlabel("时间/s")
plt.ylabel("归一化幅度")
plt.title("1/2倍采样率信号")
plt.grid(True)
plt.show()
```

图 9

```python
# 保存2倍采样率的音频
wavfile.write("2x_sample_rate.wav", f2, Sound2.astype(np.int16))
# 保存1/2倍采样率的音频
wavfile.write("0.5x_sample_rate.wav", f3, Sound3.astype(np.int16))
```

图 10

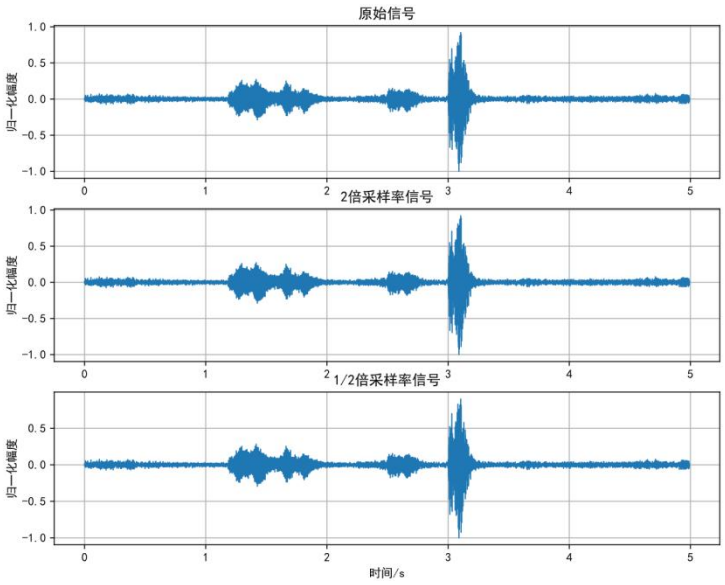


图 11

4 思考与总结

通过这次实验，我深入了解了语音信号处理中线性叠加、卷积、升采样和降采样的原理，并使用 Python 编写了相应的程序。这些操作在语音信号处理中具有重要的应用，例如混音、滤波、信号重建等。通过实践，我更好地理解了这些概念，并学会了如何使用 Python 进行实际的语音信号处理任务。这为我今后深入学习和应用语音信号处理领域奠定了基础。

实验三 声音及语音预处理实验

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。

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。

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。

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。

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。

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。

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。

3 实验结果

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)

```python
def enframe(x, win, inc):
    frame_length = len(win)
    num_frames = (len(x) - frame_length) // inc + 1

    frameout = np.zeros((num_frames, frame_length))
    for i in range(num_frames):
        start = i * inc
        end = start + frame_length
        frameout[i, :] = x[start:end] * win

    return frameout
```

图 12 enframe 函数

```python
# 分帧处理
frame_length = 0.06 * sample_rate  # 帧长为20毫秒
frame_shift = 0.03 * sample_rate  # 帧移为10毫秒
framesout = enframe(data, np.ones(int(frame_length)), int(frame_shift))
```

executed in 15ms, finished 15:10:12 2023-11-20

```python
# 绘制连续四帧语音信号
num_frames_to_plot = 4
plt.figure(figsize=(10, 6))
for i in range(num_frames_to_plot):
    plt.subplot(num_frames_to_plot,1, i + 1)
    x_coordinates = np.arange(i * (len(framesout[i]) // 2), i * len(framesout[i])//2 + len(framesout[i]))
    plt.plot(x_coordinates, framesout[i])
    plt.title(f'Frame {i + 1}')
    plt.xlabel('Sample Index')
    plt.ylabel('Amplitude')
            # 手动设置刻度标签
    x_ticks = np.arange(0,2800, 200)  # 生成刻度标签
    x_tick_labels = [str(x) for x in x_ticks]  # 将刻度标签转为字符串
    plt.xticks(x_ticks, x_tick_labels)  # 设置x轴刻度标签
plt.tight_layout()
plt.show()
```

图 13

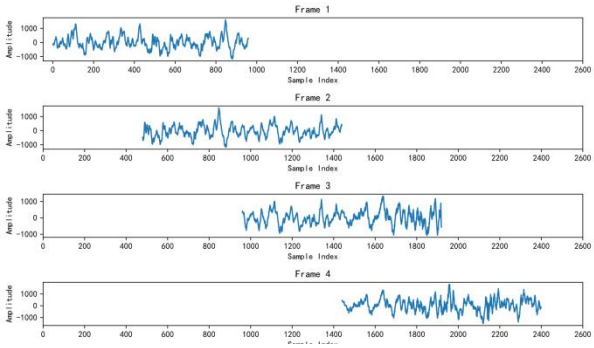


图 14 分帧后结果可视化

2）编程实现矩形窗、汉明窗和汉宁窗

```python
# 矩形窗（Rectangular Window）
def rectangular_window(N):
    return np.ones(N)

# 汉明窗（Hamming Window）
def hamming_window(N):
    return 0.54 - 0.46 * np.cos(2 * np.pi * np.arange(N) / (N - 1))

# 汉宁窗（Hanning Window）
def hanning_window(N):
    return 0.5 * (1 - np.cos(2 * np.pi * np.arange(N) / (N - 1)))
```

图 15 定义函数

```python
# 窗口长度
N = 30
t = np.linspace(0, N, N, endpoint=False)  # 生成时间序列
# 生成窗函数
rect_window = rectangular_window(N)
hamming_win = hamming_window(N)
hanning_win = hanning_window(N)

# 绘制窗函数
plt.figure(figsize=(8, 6))
plt.subplot(3,1, 1)
for i in range(len(rect_window)):
    plt.plot([t[i],t[i]],[0,rect_window[i]],'b-.')
plt.scatter(t,rect_window)
plt.title('矩形窗')

plt.subplot(3, 1, 2)
plt.scatter(t,hamming_win)
for i in range(len(hamming_win)):
    plt.plot([t[i],t[i]],[0,hamming_win[i]],'b-.')
plt.title('汉明窗')

plt.subplot(3, 1, 3)
plt.scatter(t,hanning_win)
plt.title('汉宁窗')
for i in range(len(hanning_win)):
    plt.plot([t[i],t[i]],[0,hanning_win[i]],'b-.')
plt.tight_layout()
plt.show()
```

图 16

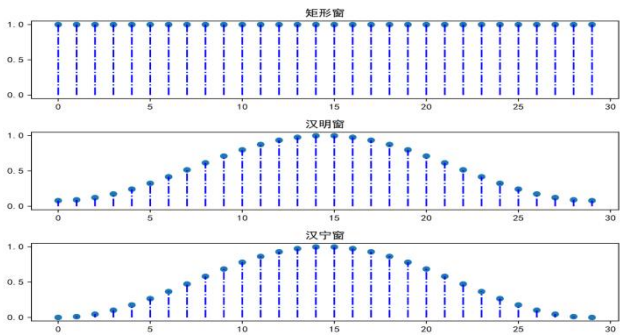


图 17

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）

```
# 合成信号
reconstructed_signal = np.zeros(len(data))
for i, frame in enumerate(framesout):
    frame_length = len(frame)
    frame_shift = int(0.5 * frame_length)  # 重叠部分为帧长度的一半
    start = i * frame_shift
    end = start + frame_length
    reconstructed_signal[start:end] += frame

# 归一化
reconstructed_signal /= np.max(np.abs(reconstructed_signal), axis=0)

# 绘制原始信号和重建信号
plt.figure(figsize=(10, 6))
plt.subplot(2, 1, 1)
plt.plot(data)
plt.title("(a)原始信号")
plt.xlabel("采样点")
plt.ylabel("幅度")

plt.subplot(2, 1, 2)
plt.plot(reconstructed_signal)
plt.title(" (b)还原信号")
plt.xlabel("采样点")
plt.ylabel("幅度")

plt.tight_layout()
plt.show()
```

图 18

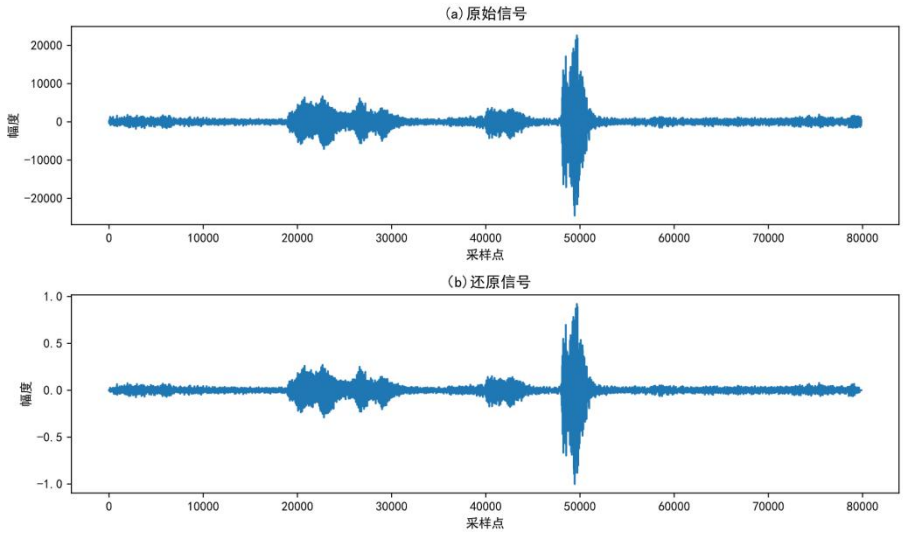


图 19

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。

```python
# 去线性趋势项函数
def detrend_linear(signal):
    t = np.arange(len(signal))
    p = np.polyfit(t, signal, 1)  # 用线性多项式拟合信号
    detrended_signal = signal - np.polyval(p, t)  # 减去线性趋势项
    return detrended_signal

# 去线性趋势
reconstructed_signal_detrended = detrend_linear(reconstructed_signal)

# 绘制去趋势后的信号
plt.figure(figsize=(10, 4))
plt.plot(reconstructed_signal_detrended)
plt.title("Reconstructed Signal (Detrended)")
plt.xlabel("Sample Index")
plt.ylabel("Amplitude")
plt.grid(True)
plt.show()

```

图 20

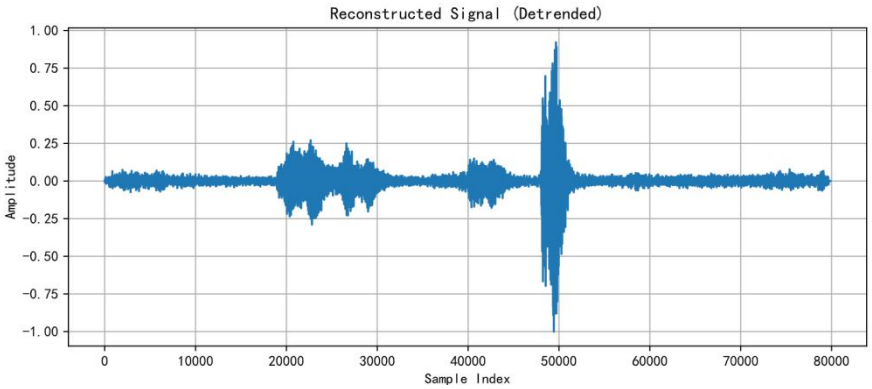


图 21

5）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

```python
from scipy import signal
from scipy.io import wavfile
# 设计低通滤波器
fs = 8000  # 采样频率
Wp = 60.0  # 通带截止频率
Ws = 50.0  # 阻带截止频率
Rp = 3.0   # 通带最大衰减
Rs = 80.0  # 阻带最小衰减
```

xecuted in 2.17s, finished 15:11:20 2023-11-20

```python
# 计算滤波器阶数
n, Wn = signal.buttord(Wp, Ws, Rp, Rs, fs=fs)
# 设计Butterworth滤波器
b, a = signal.butter(n, Wn, btype='low', analog=False, fs=fs)

```

xecuted in 25ms, finished 15:11:25 2023-11-20

```python
# 绘制幅频响应曲线
w, h = signal.freqz(b, a, worN=8000)
plt.figure()
plt.plot(0.5 * fs * w / np.pi, np.abs(h), 'b')
plt.title("Lowpass Filter Frequency Response")
plt.xlabel('Frequency [Hz]')
plt.ylabel('Gain')
plt.grid()
plt.show()
```

图 22

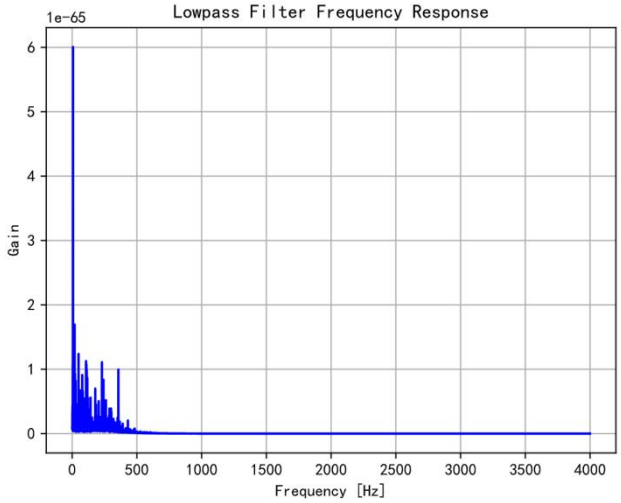


图 23

```
# 绘制相频响应曲线
plt.figure()
plt.plot(0.5 * fs * w / np.pi, np.angle(h), 'b')
plt.title("Lowpass Filter Phase Response")
plt.xlabel('Frequency [Hz]')
plt.ylabel('Phase [radians]')
plt.grid()
plt.show()
```

executed in 579ms, finished 15:11:31 2023-11-20

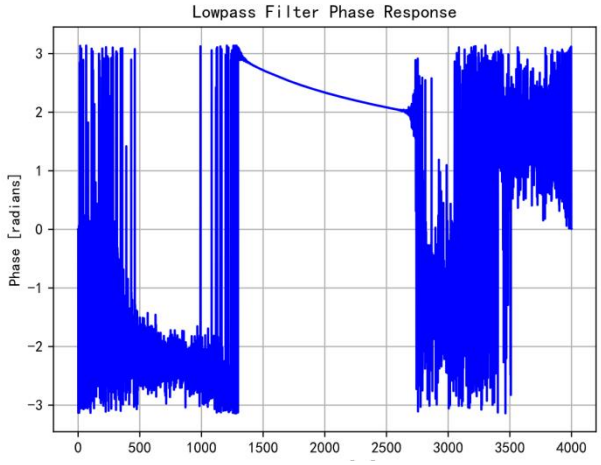


图 24

6）编写预加重函数

```
def preemphasis(signal, coefficient=0.97):
    return np.append(signal[0], signal[1:] - coefficient * signal[:-1])

# 读取语音信号
sample_rate, data = wavfile.read('../01.wav')

# 应用预加重
preemphasized_data = preemphasis(signal=framesout[0][0:100])

# 绘制原始语音信号和预加重后的语音信号
plt.figure(figsize=(12.,10))
plt.subplot(2, 1, 1)
plt.plot(framesout[0][0:100])
plt.scatter(x=range(0,len(framesout[0][0:100])),y=framesout[0][0:100],label="原始语音",marker="*")
plt.plot(preemphasized_data)
plt.scatter(x=range(0,len(framesout[0][0:100])),y=preemphasized_data,label="预加重语音",marker="o")
plt.ylabel("幅度")
plt.xlabel("采样点")
plt.legend()
plt.show()
```

图 25

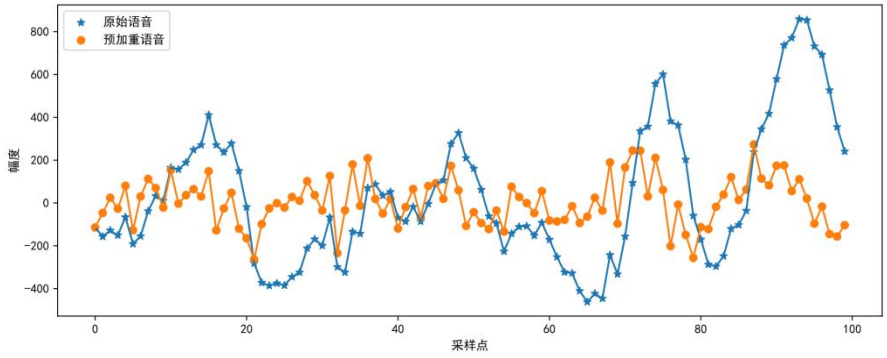


图 26

4 思考与总结

通过这次实验，我深入了解了语音信号处理中分帧、加窗、趋势项消除、预加重和预滤波的原理和方法，并通过编程实现了这些操作。这些技术对于语音信号处理中的特征提取和分析具有重要的作用，为后续的语音识别等任务提供了基础。

实验四 语音短时时域分析实验

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。

2 实验原理

1、短时能量与短时平均幅度

设第 n 帧语音信号 xn(m) 的短时能量用 En 表示，则其计算公式如下： $En=\sum_{m=0}^{N-1}x_n^2(m)$

En 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 Mn，它定义为：$\mathrm{Mn}=\sum_{m=0}^{N-1}|x_n(m)|$

Mn 也是一帧语音信号能量大小的表征，它与 En 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 xn (m) 的短时过零率 Zn 为： $Zn=\frac{1}{2}\sum_{m=0}^{N-1}|\mathrm{sgn}[x_n(m)]-\mathrm{sgn}[x_n(m-1)]|$

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 xn (m) 的短时自相关函数 Rn (k) 的计算式如下： $\mathrm{R}n(k)=\sum_{m=0}^{N-1-k}x_n(m)x_n(m+k)\qquad(0\le k\le K)$ 这里 K 是最大的延迟点数

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 Np) ，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： $d(n)=x(n)-x(n+k)=0\qquad(k=0,\pm N_p,\pm 2N_p,\cdots)$ 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： $Fn(k)=\sum_{m=0}^{N-1-k}|x_n(m)-x_n(m+k)|$， 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: $\mathrm{F}n(k)=\sqrt{2}\beta(k)[R_n(0)-R_n(k)]^{1/2}$

3 实验结果

1）编程实现短时能量、短时平均幅度和短时过零率,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。

```
def STEn(x, win, inc):
    """
    计算短时能量函数
    :param x:
    :param win:
    :param inc:
    :return:
    """
    X = enframe(x, win, inc)
    s = np.multiply(X, X)
    return np.sum(s, axis=1)
```

图 27 短时能量函数

```
def STZcr(x, win, inc):
    """
    计算短时过零率
    :param x:
    :param win:
    :param inc:
    :return:
    """
    X = enframe(x, win, inc)
    X1 = X[:, :-1]
    X2 = X[:, 1:]
    s = np.multiply(X1, X2)
    sgn = np.where(s < 0, 1, 0)
    return np.sum(sgn, axis=1)
```

图 28 短时过零率

```
def STMn(x, win, inc):
    """
    计算短时平均幅度计算函数
    :param x:
    :param win:
    :param inc:
    :return:
    """
    X = enframe(x, win, inc)
    s = np.abs(X)
    return np.mean(s, axis=1)
```

图 29 短时平均幅度

```python
# 读取语音信号
sample_rate, data = wavfile.read('../01.wav')
# 设置参数
framelen = 256
inc = 128
frameNum = 1 + int((len(data) - framelen) / inc)
fs = sample_rate
# 计算分帧后每帧对应的时间
frametime = FrameTimeC(frameNum, framelen, inc, fs)
```

executed in 19ms, finished 22:06:31 2023-12-12

```python
wlen =framelen
win = hamming_window(wlen)
EN = STEn(data, win, inc)  # 短时能量
Mn = STMn(data, win, inc)  # 短时平均幅度
Zcr = STZcr(data, win, inc)  # 短时过零率
```

executed in 22ms, finished 22:06:34 2023-12-12

```python
X = enframe(data, win, inc)
X = X.T
Ac = STAc(X)
Ac = Ac.T
Ac = Ac.flatten()
Amdf = STAmdf(X)
Amdf = Amdf.flatten()
```

executed in 1.37s, finished 22:10:28 2023-12-12

```python

fig = plt.figure(figsize=(12, 10))
plt.subplot(3, 1, 1)
plt.plot( data)
plt.title('语音波形')
plt.subplot(3, 1, 2)
frameTime = FrameTimeC(len(EN), wlen, inc, fs)
plt.plot(Mn)
plt.title('短时幅值')
plt.subplot(3, 1, 3)
plt.plot(EN)
plt.title('短时能量')
plt.show()
```

图 30

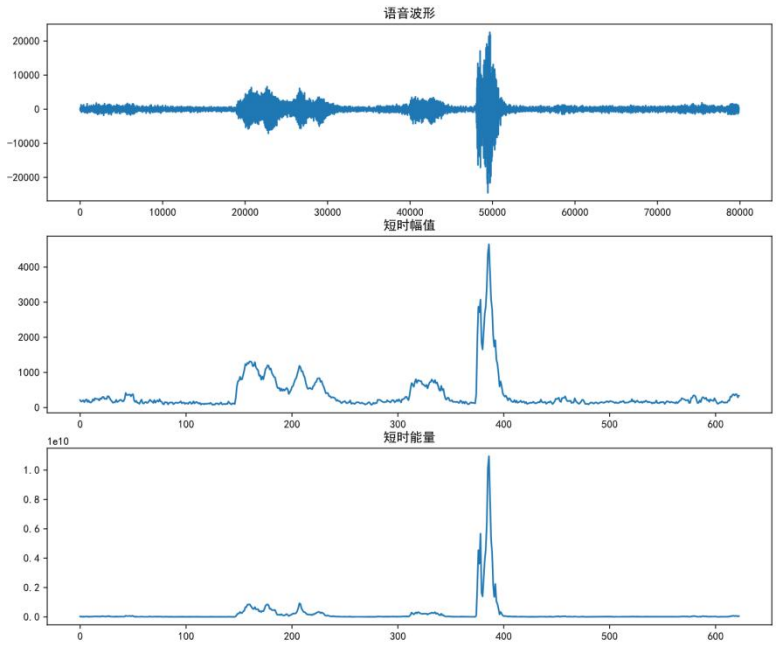


图 31

```
fig = plt.figure(figsize=(12, 7))
plt.subplot(2, 1, 1)
plt.plot(data)
plt.title('语音波形')
plt.subplot(2, 1, 2)
plt.plot( Zcr)
plt.title('短时过零率')
plt.show()
```

executed in 373ms, finished 22:10:31 2023-12-12

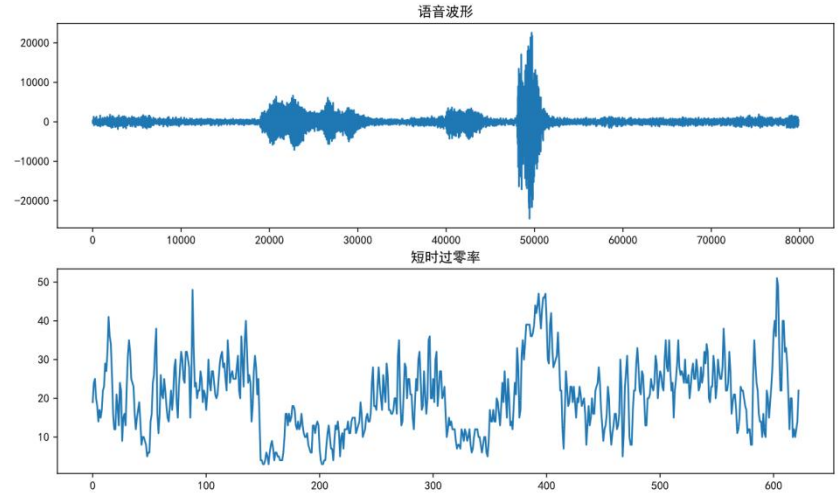


图 32

2）编程实现短时自相关和短时平均幅度差

```
def STAc(x):
    """
    计算短时相关函数
    :param x:
    :return:
    """
    para = np.zeros(x.shape)
    fn = x.shape[1]
    for i in range(fn):
        R = np.correlate(x[:, i], x[:, i], 'valid')
        para[:, i] = R
    return para
```

图 33 短时相关函数

```
fig = plt.figure(figsize=(10, 6))
plt.subplot(2, 1, 1)
plt.plot( data)
plt.title('语音波形')
plt.subplot(2, 1, 2)
plt.plot(Ac)
plt.title('短时自相关')
```

executed in 569ms, finished 22:23:51 2023-12-12

Text(0.5, 1.0, '短时自相关')

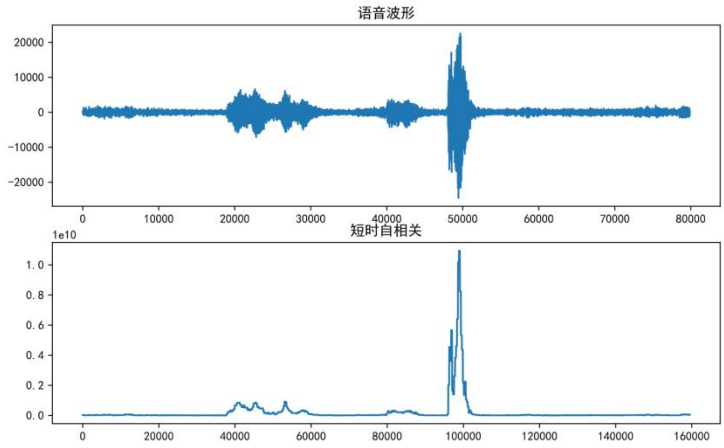


图 34

```
def STAmdf(X):
    """
    :param X:
    :return:
    """
    # para = np.zeros(X.shape)
    fn = X.shape[1]
    wlen = X.shape[0]
    para = np.zeros((wlen, wlen))
    for i in range(fn):
        u = X[:, i]
        for k in range(wlen):
            en = len(u)
            para[k, :] = np.sum(np.abs(u[k:] - u[:en - k]))
    return para
```

图 35 短时幅度偏差

```
fig = plt.figure(figsize=(10, 5))
plt.subplot(2, 1, 1)
plt.plot( data)
plt.title('(a)语音波形')
plt.subplot(2, 1, 2)
plt.plot(Amdf)
plt.title('(b)短时幅度差')
```

executed in 474ms, finished 22:19:59 2023-12-12

Text(0.5, 1.0, '(b)短时幅度差')

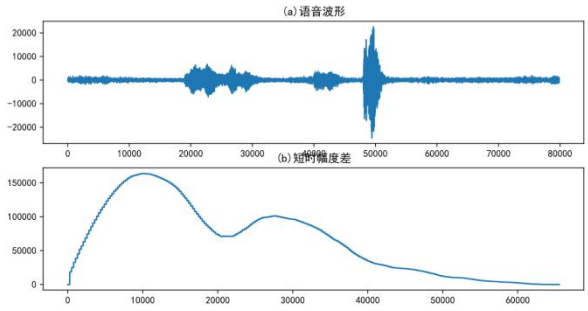


图 36

4 思考与总结

1.重要性与应用： 短时时域分析在信号处理领域有着广泛的应用，特别是在语音处理、音频处理和振动分析等方面。通过短时时域分析，我们可以了解信号在时间和频率上的变化，对于诸如声音的特征提取、音频处理、语音识别等任务具有重要价值。

2.窗口函数的选择： 实验中发现窗口函数的选择对分析结果有显著的影响。不同窗口函数在时域和频域的主瓣宽度、副瓣衰减等方面表现不同。根据实际需求，选择合适的窗口函数是关键的一步。

3.窗口大小与分辨率： 在实验过程中，观察到窗口大小对时域分辨率和频域分辨率的影响。较小的窗口可以提供更好的时域分辨率，但可能导致频域分辨率不足。需要在实际应用中平衡这两个因素。

4.参数调整与实时性： 不同应用场景对短时时域分析的要求可能有所不同。在实时应用中，需要考虑到计算复杂度，可能需要通过调整参数来达到平衡实时性和准确性。

实验五 语音短时频域分析实验

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；

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：$x_n(m)=w(m)x(n+m) \quad 0\le m\le N-1$

设离散时域采样信号为x(n)，n=1,2,3…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=0,1,2…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：$X_n(e^{jw})=\sum_{m=0}^{N-1}x_n(m)e^{-jwm}$

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)= $|X_n(k)|$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

3 实验结果

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。

```
def enframe(x, win, inc):
    frame_length = len(win)
    num_frames = (len(x) - frame_length) // inc + 1

    frameout = np.zeros((num_frames, frame_length))
    for i in range(num_frames):
        start = i * inc
        end = start + frame_length
        frameout[i, :] = x[start:end] * win

    return frameout
# 汉明窗（Hamming Window）
def hamming_window(N):
    return 0.54 - 0.46 * np.cos(2 * np.pi * np.arange(N) / (N - 1))
def FrameTimeC(frameNum, frameLen, inc, fs):
    ll = np.array([i for i in range(frameNum)])
    return ((ll - 1) * inc + frameLen / 2) / fs
def STFFT(x, win, nfft, inc):
    xn = enframe(x, win, inc)
    xn = xn.T
    y = np.fft.fft(xn, nfft, axis=0)
    return y[:nfft // 2, :]

```

图 37

2）根据语谱图的显示原理，编程实现语谱图的计算和显示

```
# 读取语音信号
sample_rate, data = wavfile.read('../01.wav')
# 设置参数
framelen = 256
inc = 128
frameNum = 1 + int((len(data) - framelen) / inc)
```

executed in 17ms, finished 22:31:49 2023-12-12

```
wlen = 256
nfft = wlen
win = hamming_window(wlen)
inc = 128
fs=sample_rate
y = STFFT(data, win, nfft, inc)
freq = [i * fs / wlen for i in range(wlen // 2)]
frame = FrameTimeC(y.shape[1], wlen, inc, fs)

plt.matshow(np.log10(np.flip(np.abs(y)*np.abs(y), 0)))
plt.colorbar()
plt.close()

plt.specgram(data, NFFT=256, Fs=fs, window=np.hamming(256),cmap='jet')
plt.ylabel('频率')
plt.xlabel('Time(s)')
plt.show()
```

图 38

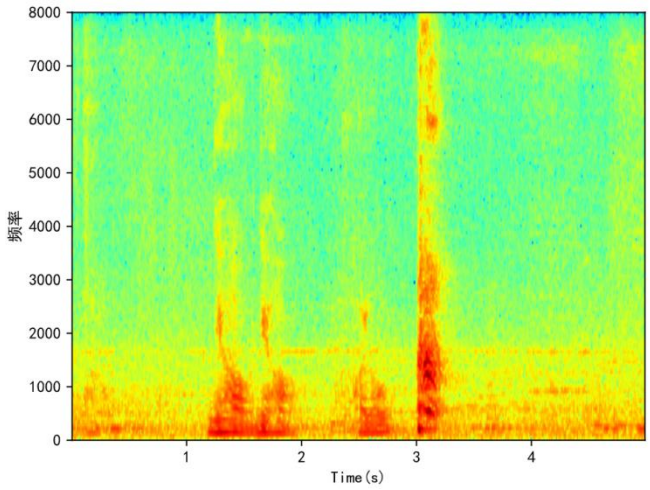


图 39

4 思考与总结

1.频谱信息的可视化： 短时傅里叶变换和语谱图的结合使得频谱信息在时间上的变化可以直观地观察，有助于理解信号的动态特性。

2.参数调整与实验效果： 调整短时傅里叶变换的参数，如窗口大小和重叠，可以影响语谱图的清晰度。在实验中需要根据信号特性进行调优。

实验六 语音端点检测实验

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。

2 实验原理

1、基于双门限法的端点检测原理

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 xn(m) 的短时能量用 En 表示，则其计算公式如下：$\mathrm{E}_n = \sum_{m=0}^{N-1} x_n^2(m)$ En 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

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 xn (m) 的短时过零率 Zn 为：$Z_n=\frac{1}{2}\sum_{m=0}^{N-1}|\mathrm{sgn}[x_n(m)]-\mathrm{sgn}[x_n(m-1)]|$

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。

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 xn (m)的短时自相关函

数 Rn (k)的计算式如下：$R_n(k)=\sum_{m=0}^{N-1-k}x_n(m)x_n(m+k)\quad(0\le k\le K)$ 这里 K 是最大的延迟点数

2.2 自相关函数最大值法

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：

（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 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

4、比例法端点检测

4.1 能零比的端点检测

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。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。

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：$EEF_{\mathrm{n}}=\sqrt{1+|LE_n/H_n|}$

3 实验结果

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。

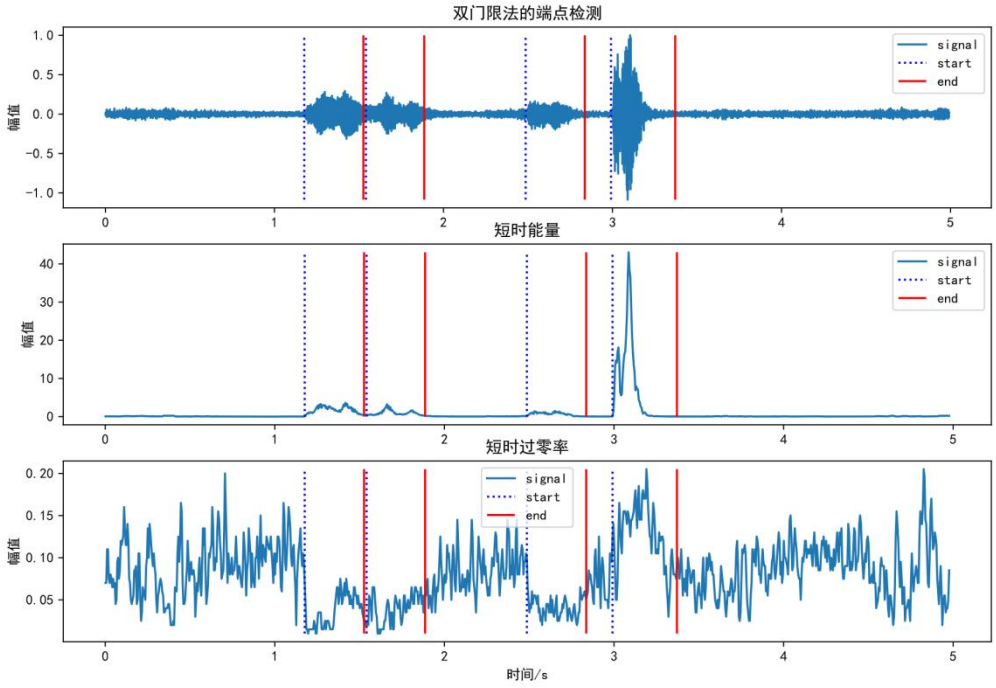


图 40

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

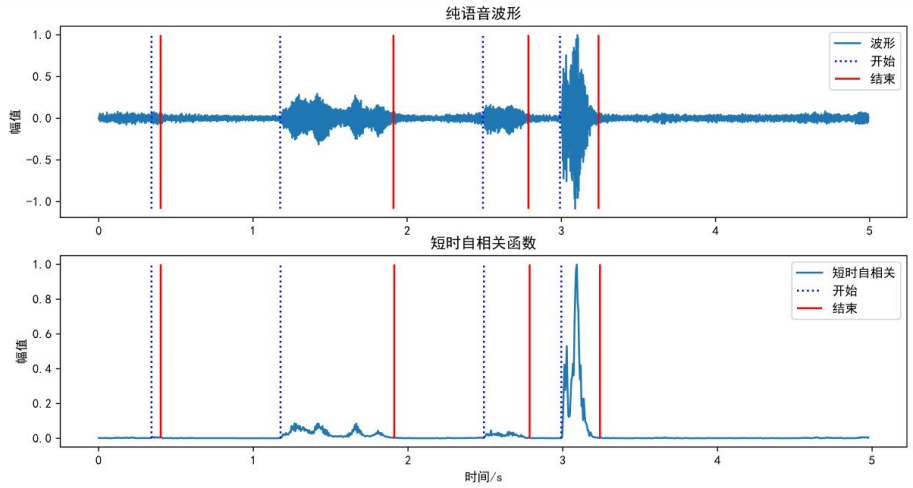

纯语音波形
波形
开始
结束
幅值
1.0
0.5
0.0
−0.5
−1.0
0
1
2
3
4
5
短时自相关函数
短时自相关
开始
结束
幅值
1.0
0.8
0.6
0.4
0.2
0.0
0
1
2
3
4
5
时间/s

图 41

```
    maxsilence = 8
    minlen = 5
    status = 0
    count = np.zeros(fn)
    silence = np.zeros(fn)
    xn = 0
    x1 = np.zeros(fn)
    x2 = np.zeros(fn)
    for n in range(1, fn):
        if status == 0 or status == 1:
            if dst1[n] > T2:
                x1[xn] = max(1, n - count[xn] - 1)
                status = 2
                silence[xn] = 0
                count[xn] += 1
            elif dst1[n] > T1:
                status = 1
                count[xn] += 1
            else:
                status = 0
                count[xn] = 0
                x1[xn] = 0
                x2[xn] = 0
        if status == 2:
            if dst1[n] > T1:
                count[xn] += 1
            else:
                silence[xn] += 1
                if silence[xn] < maxsilence:
                    count[xn] += 1
                elif count[xn] < minlen:
                    status = 0
                    silence[xn] = 0
                    count[xn] = 0
                else:
                    status = 3
                    x2[xn] = x1[xn] + count[xn]
        if status == 3:
            status = 0
            xn += 1
            count[xn] = 0
            silence[xn] = 0
            x1[xn] = 0
            x2[xn] = 0
    el = len(x1[:xn])
    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
        NF[int(x1[i]):int(x2[i])] = 0
    voiceseg = findSegment(np.where(SF == 1)[0])
    vsl = len(voiceseg.keys())
    return voiceseg, vsl, SF, NF


def findSegment(express):
```

(variable) minlen: Literal[5]

4 思考与总结

1.效果评估： 通过对实验音频的端点检测，可以通过与真实标注比较来评估每种方法的性能。可以使用一些评价指标如准确率、召回率、F1 分数等来进行评估。

2.参数调整： 不同的语音信号可能需要不同的参数设置，因此在实际应用中，需要根据具体情况进行调整，以达到最佳的端点检测效果。

3.实时性能： 一些端点检测方法可能需要较大的计算开销，对于实时应用可能不太适用。因此，在实际应用中需要考虑算法的实时性能。

4.噪声处理： 上述方法对于噪声敏感，对于噪声较大的情况可能需要采用噪声抑制技术或其他先进的端点检测方法

实验七 语音增强实验

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。

2 实验原理

1、语音降噪的意义

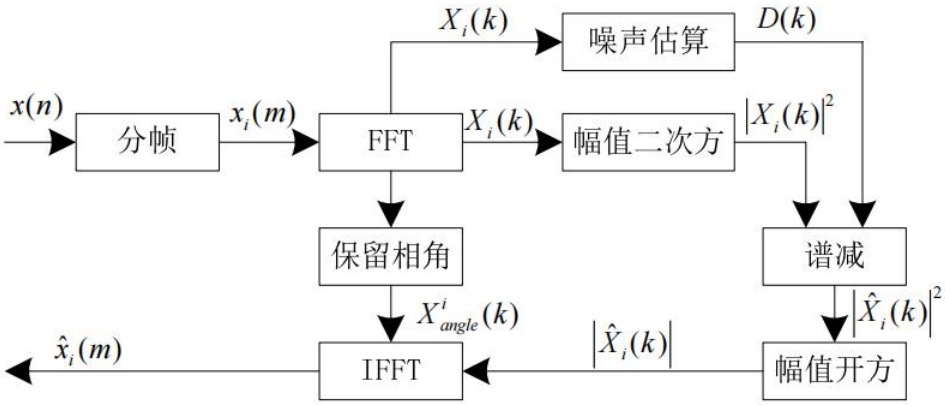
语音降噪主要研究如何利用信号处理技术消除信号中的强噪声干扰，从而提高输出信噪比以提取出有用信号的技术。消除信号中噪声污染的通常方法是让受污染的信号通过一个能抑制噪声而让信号相对不变的滤波器，此滤波器从信号不可检测的噪声场中取得输入，将此输入加以滤波，抵消其中的原始噪声，从而达到提高信噪比的目的。

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： $y(n) = s(n) + d(n)$ 其中 s(n)表示纯净语音，d(n)表示噪声，y(n)表示带噪语音。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



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

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

噪声段的平均谱值为

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

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：

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

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”。

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

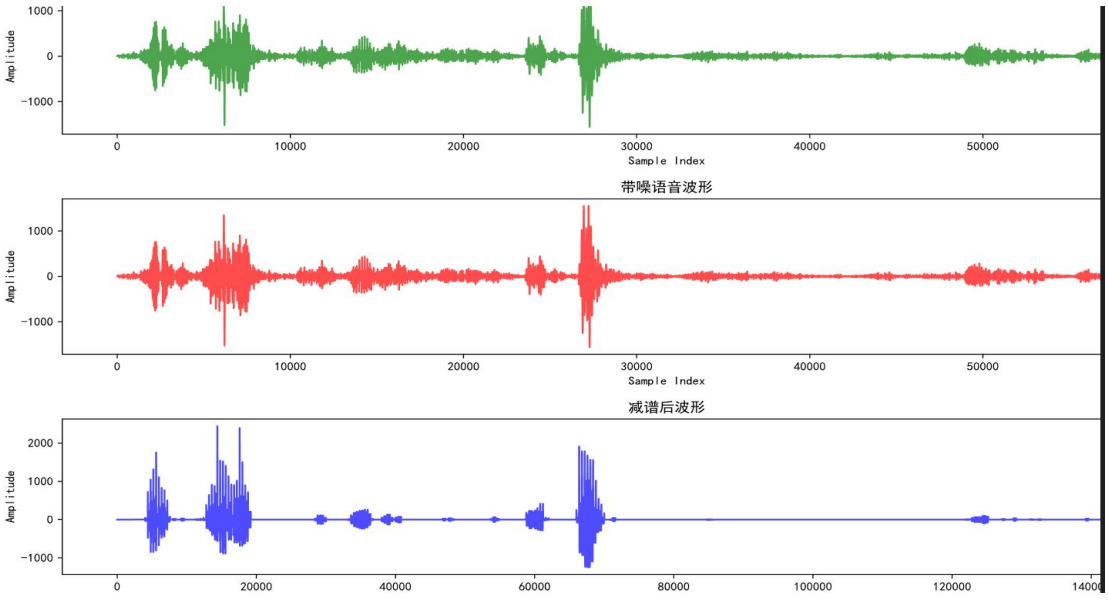
此处，$\max|N_R(k)|$代表最大的噪声残余。

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。

3 实验结果

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



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

```python
def normalize(signal):
    return signal / np.max(np.abs(signal))

def OverlapAdd2(XNEW, yphase, ShiftLen):
    Y = np.multiply(XNEW, np.exp(1j * yphase))
    y = np.real(ifft(Y))

    output = np.zeros((Y.shape[0] - 1) * ShiftLen + windowLen)
    for i in range(Y.shape[0]):
        start = i * ShiftLen
        output[start:start + windowLen] += y[i]

    return output

def SpectralSub(signal, wlen, inc, NIS, a, b):
    frames = np.arange(0, len(signal) - wlen, inc, dtype=int)
    output = np.zeros(len(signal))

    for start in frames:
        end = start + wlen

        frame = signal[start:end]

        # 计算傅里叶变换
        X = fft(frame)
        magnitude = np.abs(X)

        # 谱减法处理
        D = np.mean(magnitude[:NIS])
        alphaD = a * D
        betaD = b * D

        mask = np.where(magnitude > alphaD, 1, magnitude / betaD)
        enhanced_frame = mask * X

        # 反变换回时域
        enhanced_signal = np.real(ifft(enhanced_frame))

        # 重叠相加
        output[start:end] += enhanced_signal

    return normalize(output)

def SpectralSubIm(signal, wind, inc, NIS, Gamma, Beta):
    frames = np.arange(0, len(signal) - len(wind) + 1, inc, dtype=int)
    output = np.zeros(len(signal))

    for start in frames:
        end = start + len(wind)

        frame = signal[start:end]

        # 计算傅里叶变换
        X = fft(frame)
        magnitude = np.abs(X)

        # Boll 改进谱减法处理
        avg_magnitude = np.convolve(magnitude, wind, mode='valid')
        avg_magnitude = np.pad(avg_magnitude, (start, len(signal) - end), 'constant', constant_values=(0, 0))

        # 使用 np.resize 代替 np.broadcast_to
        Gamma_avg_magnitude = np.resize(Gamma * avg_magnitude, magnitude.shape)
        Beta_broadcasted = np.resize(Beta, magnitude.shape)

        mask = np.where(magnitude > Gamma_avg_magnitude, 1, magnitude / Beta_broadcasted)
        enhanced_frame = mask * X

        # 反变换回时域
        enhanced_signal = np.real(ifft(enhanced_frame))

        # 重叠相加
        output[start:end] += enhanced_signal

    return normalize(output)
```

xecuted in 21ms, finished 23:05:45 2023-12-12

图 42

```python
import numpy as np
import scipy.io.wavfile as wav
import matplotlib.pyplot as plt
from scipy.fft import fft, ifft
# 读取测试语音文件
fs, signal = wav.read("../01.wav")

# 生成不同信噪比的噪声并添加到语音信号
np.random.seed(0)
noise = np.random.normal(0, 400, len(signal))
noisy_signal = signal + noise

# 定义参数
windowLen = 256  # 帧长
shiftLen = 128   # 帧移
NIS = 10         # 前导无话段帧数
alpha = 2.0      # 过减因子（可调整）
beta = 3.0       # 增益补偿因子（可调整）
gamma = 1.5      # Boll 改进谱减法参数（可调整）
beta_boll = 2.0  # Boll 改进谱减法参数（可调整）

# 调用基本谱减法进行降噪
enhanced_signal_basic = SpectralSub(noisy_signal, windowLen, shiftLen, NIS, alpha, beta)

# 合成语音信号
output_basic = OverlapAdd2(fft(enhanced_signal_basic), np.angle(fft(noisy_signal)), shiftLen)

# 调用 Boll 改进谱减法进行降噪
wind = np.hanning(windowLen)
enhanced_signal_boll = SpectralSubIm(noisy_signal, wind, shiftLen, NIS, gamma, beta_boll)

# 合成语音信号
output_boll = OverlapAdd2(fft(enhanced_signal_boll), np.angle(fft(noisy_signal)), shiftLen)

# 绘制波形图
plt.figure(figsize=(10, 8))

time = np.arange(len(signal)) / fs  # 时间序列

plt.subplot(2, 2, 1)
plt.plot(time, normalize(signal), label='原始信号')
plt.title('原始信号')

plt.subplot(2, 2, 2)
plt.plot(time, normalize(noisy_signal), label='带噪音的原始信号')
plt.title('带噪音的原始信号')

plt.subplot(2, 2, 3)
plt.plot(time, enhanced_signal_basic, label='增强后的信号（基本谱减法）')
plt.title('增强后的信号（基本谱减法）')

plt.subplot(2, 2, 4)
plt.plot(time, enhanced_signal_boll, label='增强后的信号（Boll改进谱减法）')
plt.title('增强后的信号（Boll改进谱减法）')

plt.tight_layout()
plt.show()
```

图 43

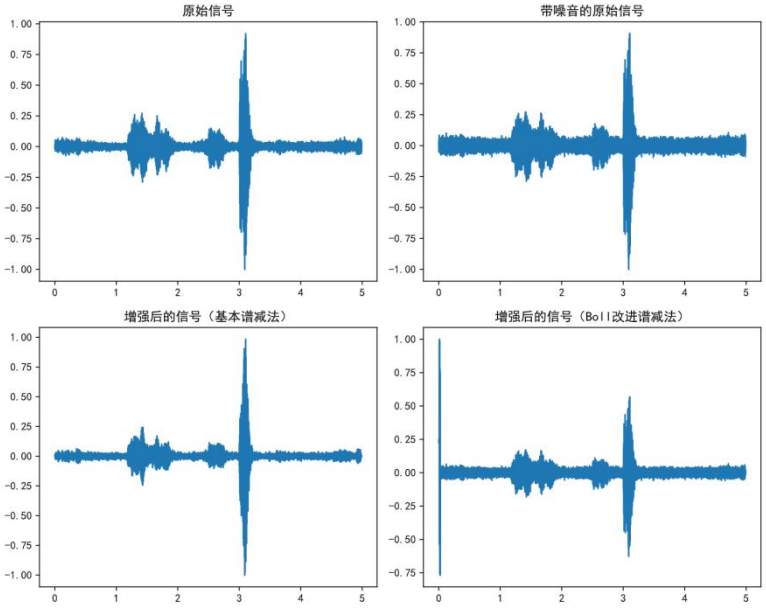


图 44

4 思考与总结

1.效果评估： 通过听觉感受和比较降噪前后的音频，评估降噪算法的效果。

2.参数调整： 调整噪声阈值等参数，观察对降噪效果的影响。

3.噪声环境： 在实验中可以尝试不同的噪声环境，了解降噪算法对不同噪声类型的适应性。

4.实时性能： 考虑算法在实时应用中的性能，尤其是对于大规模语音数据的处理。

5.其他降噪算法： 谱减法是一种基本的降噪方法，实际应用中可能需要考虑其他高级算法，如小波变换、深度学习等。