第 5 章 语音识别基础

语言是人类最重要的交流工具，也是人类区别于其他动物的本质特征。语音信号中除了含有语义信息外，通常还含有说话人特征、性别、年龄、情感等信息。

语音识别，简单来讲就是让机器明白人在说什么，而要做到这一点，需要从语音信号的产生和感知说起。

5.1 语音的产生和感知

5.1.1 语音信号的产生

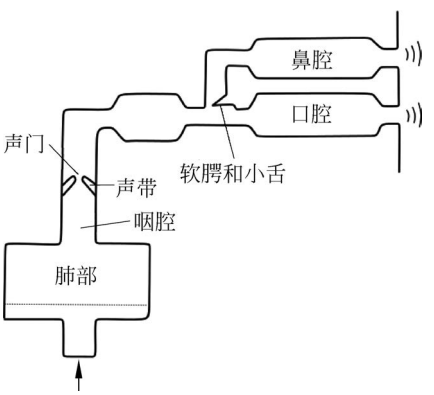

图 5-1 人类的发音器官

人类的发音器官自下而上包括肺部、气管、喉部、咽部、鼻腔、口腔和唇部，它们连成一体，从而形成一个连续的管道，如图 5-1 所示。发音时肺部收缩形成气流，经气管至喉头声门处。发声之初，声门处的声带肌肉收缩、声带并拢，这股气流冲过细小的缝隙使声带产生振动，从而发出声音。

在发音过程中，肺部与相连的肌肉相当于激励源。当声带处于收紧状态时，气流使声带发生振动，此时产生的声音称为浊音(Voiced Sound)，而发音时声带不振动的音称为清音(Unvoiced Sound)。

5.1.2 语音信号的感知

1. 听觉的形成

耳是人类的听觉器官。人耳由外耳、中耳、内耳三部分组成，而外耳、中耳、内耳的耳蜗部分才是真正的听觉器官，如图 5-2 所示。

声音经过外耳、中耳和内耳的传导系统，引起耳蜗内淋巴液的振动，这样的刺激使耳蜗内的听觉细胞产生兴奋，将声音刺激转换为神经冲动，沿听觉神经传到大脑皮层的听觉中枢，从而产生听觉。

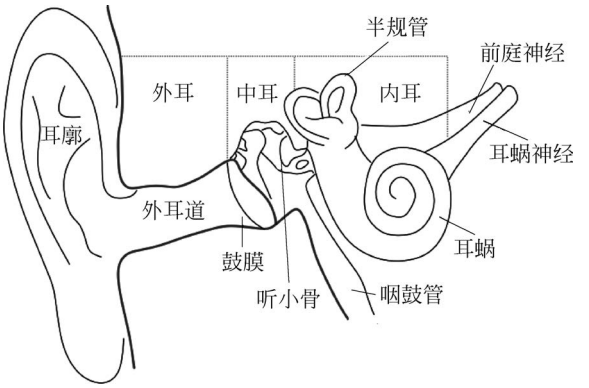

图 5-2　人类的听觉器官

2．听阈和痛阈

人耳可以听到的声音的频率范围为 20Hz～20kHz，但是声音必须达到一定强度才能引起听觉。刚能引起人耳听觉反应的最小声音刺激量称为听阈，而刚能引起人耳不适或疼痛的最小刺激量称为痛阈。声强超过 140dB 时会在耳膜引起疼痛感觉。实验表明，听阈随频率变化相当剧烈，但痛阈受频率的影响不大，如图 5-3 所示。

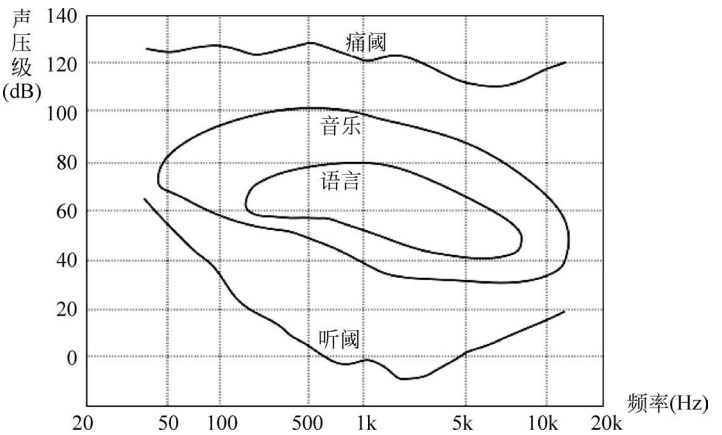

图 5-3　听阈和痛阈

在听觉范围内，人耳对声音的敏感程度随频率变化，听觉最敏感的频率段是 2～4kHz。

3．掩蔽效应

掩蔽效应是指在一个较强的声音附近，一个较弱的声音将不被人耳感觉到的现象，其中较强的声音称为掩蔽声，较弱的声音称为被掩蔽声。例如，飞机发动机的轰鸣声会淹没人的说话声，此时发动机的声音是掩蔽声，人的说话声则是被掩蔽声。

被掩蔽掉的不可闻信号的最大声压级称为掩蔽门限或掩蔽阈值，这个掩蔽阈值以下的声音都会被掩蔽掉。掩蔽声的存在会使听阈曲线发生变化，如图 5-4 所示。图中最下方的曲线表示在安静环境下人耳可以听到的各种频率声音的最低声压级，由于 1kHz 频率的掩

蔽声的存在，听阈曲线发生了变化，本来可以听到的声音变得听不到了，或者说由于掩蔽声的存在而产生了掩蔽效应。

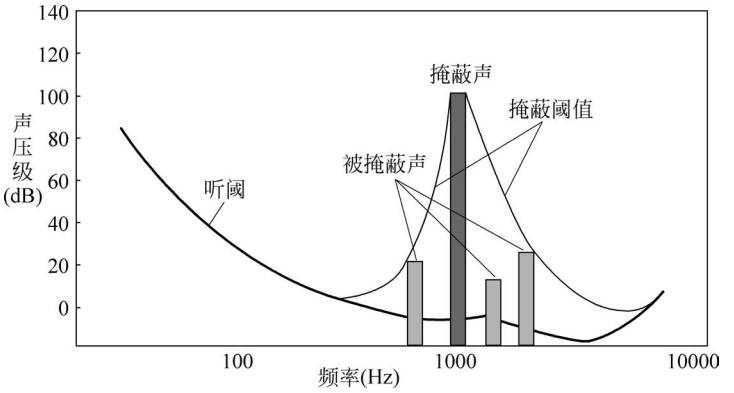

图 5-4　掩蔽效应原理图

掩蔽效应可根据掩蔽声和被掩蔽声是否同时出现分为同时掩蔽和时域掩蔽。

同时掩蔽又称为频域掩蔽，是指在同一时间内一个声音对另一个声音产生掩蔽现象。一般来讲，对于同时掩蔽，掩蔽声愈强、掩蔽声与被掩蔽声的频率越接近，掩蔽效果也越明显。

当两个声音不同时出现时发生的掩蔽效应称为时域掩蔽。时域掩蔽又分为后向掩蔽和前向掩蔽。当掩蔽声和被掩蔽声同时存在时，掩蔽声突然消失后，其掩蔽作用会持续很短一段时间，这种情况称为后向掩蔽。如果被掩蔽声先出现，接着在很短时间内出现了掩蔽声，则这种情况称为前向掩蔽。一般来讲，后向掩蔽可持续 100ms 左右，前向掩蔽则仅可持续 20ms 左右。

5.1.3　语音信号的数字模型

为了定量描述语音处理所涉及的各种因素，人们一直在寻找一个理想的模型，但是，鉴于语音信号的复杂性，目前尚未找到一个可以详细描述所有特征的理想模型。传统的基于声道的语音产生模型包括三部分：激励模型、声道模型和辐射模型，如图 5-5 所示。

1. 激励模型

在发浊音时，气流对声带产生冲击而产生振动，形成间歇的脉冲波。这个脉冲波类似于斜三角形的脉冲，如图 5-6 所示。在发清音时，声道处于松弛状态，此时的声道被阻碍，从而形成湍流，此时的激励信号相当于一个随机的白噪声。

当然，简单地把激励分为清音和浊音这两种情况并不严谨，也有人提出了一些其他的模拟方法，不过也并不完美。

2. 声道模型

对于声道的数学模型有以下两种观点：

(1) 将声道视为多个不同截面积的声管串联而成的系统，称为声道模型，如图 5-7 所示。

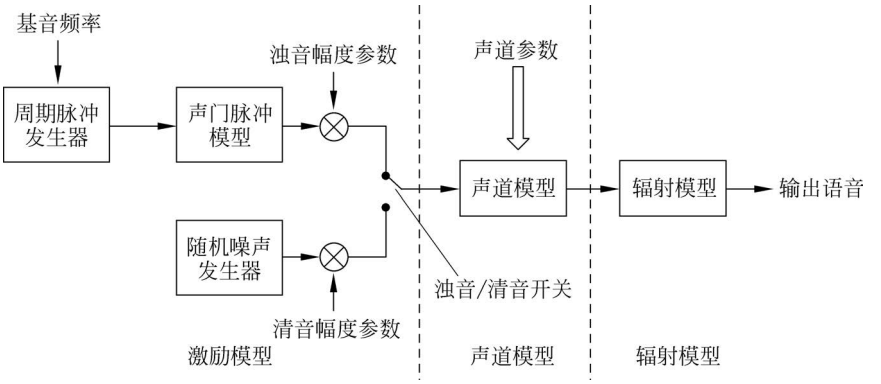

图 5-5　语音信号的数字模型

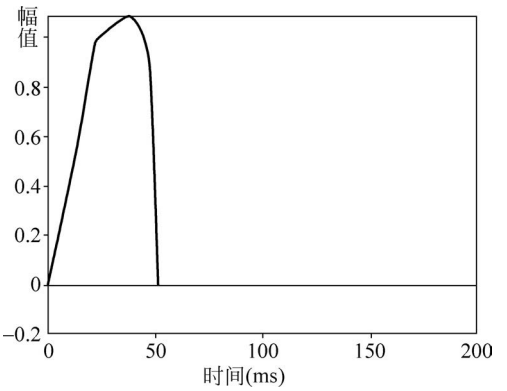

图 5-6　浊音产生的脉冲波

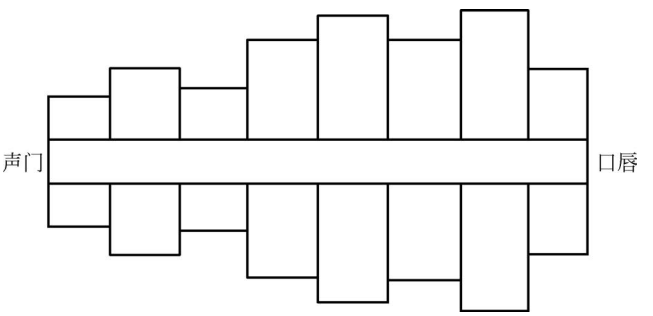

图 5-7　声道的声管模型剖面图

(2) 将声道视为一个谐振腔,而共振峰就是这个腔体的谐振频率,这种模型称为共振峰模型。

基于声学的共振峰理论,可以建立起 3 种共振峰模型:级联型、并联型和混合型。

级联型共振峰模型把声道看作一组串联的二阶谐振器,如图 5-8 所示(图中 G 为幅值因子,下同)。对于一般元音来讲,用级联型模型就可以了。

从共振峰理论来看,整个声道具有多个谐振频率和多个反谐振频率,所以它可被模拟为

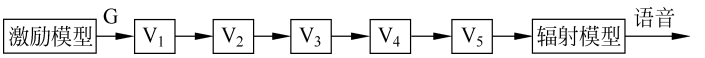

图 5-8　级联型共振峰模型

一个零极点的数学模型。对于鼻化元音或阻塞音、摩擦音等辅音，级联模型就不能胜任了，此时必须采用零极点模型，这就是并联型共振峰模型，如图 5-9 所示，并联型共振峰模型适用于非一般的元音和大部分辅音。

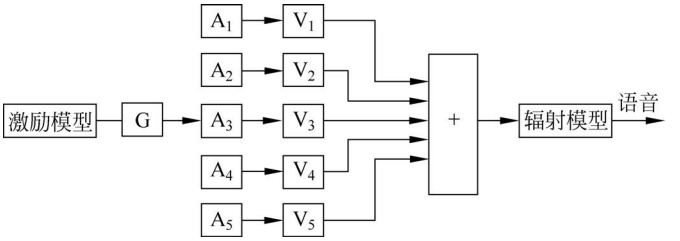

图 5-9　并联型共振峰模型

级联型和并联型各有侧重，如果需要一个较为完备的共振峰模型，则需要将两者结合起来，这就是混合型共振峰模型，如图 5-10 所示。图中并联部分还添加了一条直通路径，其幅度控制因子为 AB，这是专为频谱特性较为平坦的音素准备的。

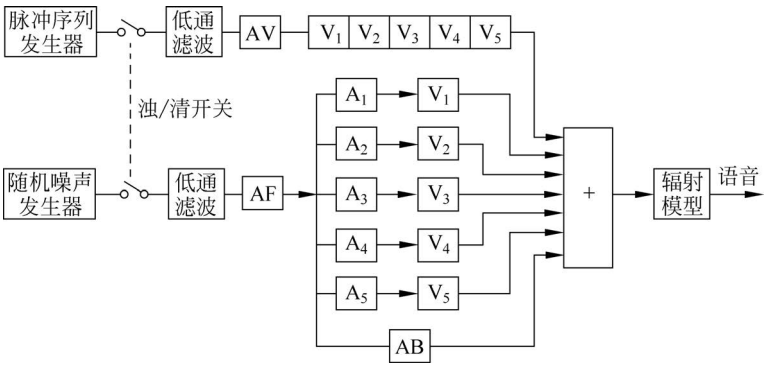

图 5-10　混合型共振峰模型

3. 辐射模型

从声道输出的是速度波，而语音信号是声压波，二者的倒比称为辐射阻抗，它表征口唇的辐射效应。研究表明，口唇端辐射在高频端较为明显，在低频段则影响较小，所以可用一个高通滤波器来表示辐射模型。

5.2　汉语的语音特征

汉语语音的基础是汉语拼音，按照元音和辅音分，可分成 10 个元音和 22 个辅音；如果按声母和韵母分类，则可以分为 21 个声母和 38 个韵母。

5.2.1　元音和辅音

汉语中共有 10 个单元音，又可细分为舌面元音(7 个)、舌尖元音(2 个)和卷舌元音(1 个)3 类，具体如下：

(1) 舌面元音：a，o，e，i，u，ü，ê。

(2) 舌尖元音：i[ɿ]，i[ʅ]。

(3) 卷舌元音：er。

不同的元音是由不同的口腔形状(唇舌状态)造成的。舌位按高低分一般可分为高、半高、半低、低 4 种，按前后分可分为前、中、后 3 种，元音与舌位的关系如图 5-11 所示。

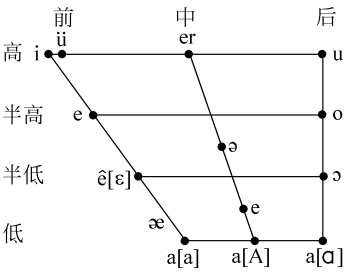

图 5-11　元音与舌位的关系

汉语语音中的 22 个辅音及其分类见表 5-1。辅音按照发音方法可分为鼻音、塞音、擦音、塞擦音、边音等；按照发音部位又可分为双唇音、唇齿音、舌尖音、卷舌音、舌面音、舌根音等。

表 5-1　汉语辅音表

发音方法			双唇音	唇齿音	舌尖音	舌面音	舌根音
塞音	清音	不送气	b		d		g
		送气	p		t		k
塞擦音	清音	不送气			z，zh	j	
		送气			c，ch	q	
擦音	清音			f	s，sh	x	h
	浊音				r		
鼻音	浊音		m		n		ng
边音	浊音				l		

5.2.2　声母和韵母

按照我国传统音素分类方法，汉语音节由声母和韵母拼合而成。声母一般仅包含一个辅音，而韵母则由一个/多个元音或元音和辅音组合而成。

声母共 21 个(不含零声母)，见表 5-2。

表 5-2　汉语声母表

声母	读音	声母	读音	声母	读音	声母	读音	声母	读音	声母	读音
b	波	p	坡	m	摸	f	佛	d	得	t	特
n	讷	l	勒	g	哥	k	科	h	喝	j	基
q	期	x	希	zh	知	ch	吃	sh	诗	r	日
z	资	c	雌	s	思						

韵母共 39 个，又可分为单韵母、复韵母和鼻韵母，见表 5-3。

表 5-3 汉语韵母表

	−i(前)，−i(后)	i	u	ü
单韵母	a	ia	ua	
	o		uo	
	e			
	ê	ie		üe
	er			
复韵母	ai		uai	
	ei		uei	
	ao	iao		
	ou	iou		
鼻韵母	an	ian	uan	üan
	en	in	uen	ün
	ang	iang	uang	
	eng	ing	ueng	
	ong	iong		

元音和辅音、声母和韵母是两种不同的分类方法。元音和辅音是国际上通行的一种分类法，不只汉语有，其他语言也有，但声母和韵母却是汉语独有的。另外，元音和辅音是按发音方式划分的，而声母和韵母则是按音节中的位置区分的。不过，两者之间又有着一定的联系，例如声母一般是辅音，而元音都是韵母。

5.2.3 音素

在汉语里，音素是最小的语音单位，而音节则是说话时的发音单位，可以从听觉上把它们分开。音节由一个或多个音素组成，单个元音音素也可自成音节。

汉语一般是一字一音节，仅有两种例外情况：

(1) 一字两音节，如砑(音“千瓦”)，但这种用法已很罕见。

(2) 儿化音，如“花儿”虽是两字，儿化后只有一个音节。

连续发音时音素可能产生变形，主要有以下几种情况。

(1) 协同发音：连续语音中的音素会受到前后音响的影响而发生变形，称为协同发音。例如复合元音 uai，韵腹 a 发音时，声道形状是由韵头 u 的形状转换而来的，而且还要为转换为韵尾 i 的形状做准备。

(2) 轻声：汉语拼音里面只有阴平、阳平、上声和去声 4 个声调，但在有些情况下某个音节会失去原有的声调，而读成一个又轻又短的调子，这就是“轻声”。例如，“姐”字是三声，可是在“姐姐”这个词中，后一个“姐”字失去了原来的声调，读得比第 1 个“姐”轻得多，成为一个轻声音节。不过，轻声并不是第 5 种声调，而是四声的一种特殊音变，具体表现为“音长变短、音强变弱”。

(3) 变调：两个字连续使声调发生变化称为变调。例如两个三声字连在一起读时，前一个三声字受到后一个字的影响会变成二声字，如“你好”“理想”等。

5.2.4　音调

汉语中每个音节都对应一定的音调，除轻音外，音调有 4 种变化：阴平、阳平、上声、去声。同一个声母和韵母构成的发音，如音调不同，则对应的字也不同，意思也不一样。例如“妈、麻、马、骂”4 个字的声母和韵母均相同，但音调不同，意思也截然不同。

声调可以用“五度标调法”来标注，如图 5-12 所示。在“五度标调法”中，声调用数字 1～5 分成“低、次低、中、次高、高”5 个等级，如果是直线型，则只需记起点与终点的度数，如果是曲线型，则需要加记曲折起落的度数。

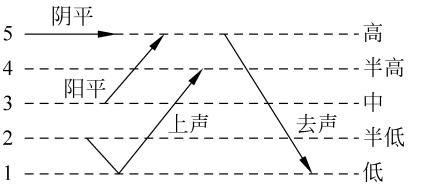

图 5-12　汉语四声位置图

普通话中的四声用五度标调法标注如下：

(1) 一声(阴平)：55。

(2) 二声(阳平)：35。

(3) 三声(上声)：214。

(4) 四声(去声)：51。

5.3　元音与共振峰

共振峰是指在声音的频谱中能量相对集中的一些区域。在频谱图上，共振峰是包络线的极大值，如图 5-13 所示。

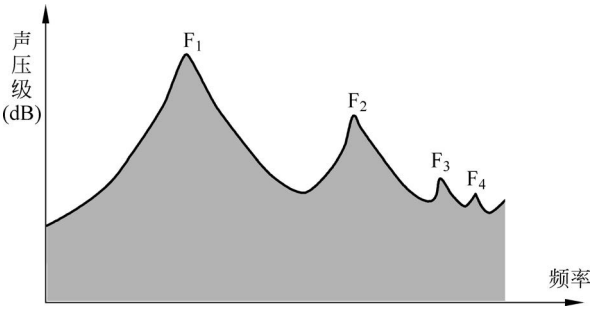

图 5-13　共振峰示意图

元音在发音时会同时受到舌头位置和嘴唇形状的影响，产生多个共振频率，因此一个元音会有 3～5 个共振峰。研究表明，一个元音通常用 3 个共振峰就可以表示出来，而复杂的辅音或鼻音，则需要用 5 个共振峰来表示。

共振峰是频谱图上包络线的极大值，但是频谱图的局部峰值往往很多且不容易观察。语音学软件 Praat 对共振峰的功能进行了强化，极大地简化了共振峰的观察，如图 5-14 所示，Praat 中专设了 Formant 的菜单栏，不但可以在图形上显示共振峰，还能列出共振峰的值。

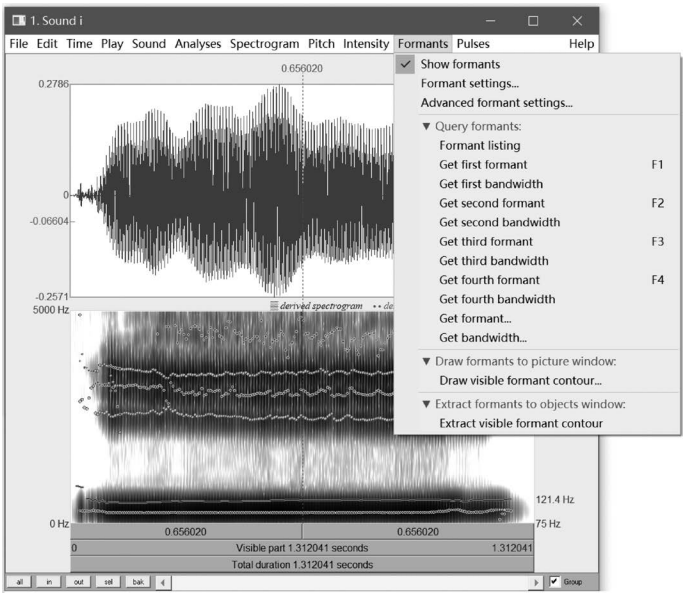

图 5-14　Praat 中的共振峰菜单

Praat 中元音 a、o、i 的共振峰如图 5-15～图 5-17 所示，每幅图的上半部分为波形图，下半部分为语谱图，语谱图中的红点组成了共振峰。为了精确地了解共振峰的频率数据，每幅图的右下角还显示了 Praat 识别的前 4 个共振峰的数据（F_1 代表第 1 共振峰，F_2 代表第 2 共振峰，F_3 代表第 3 共振峰，第 4 共振峰一般用不到）。

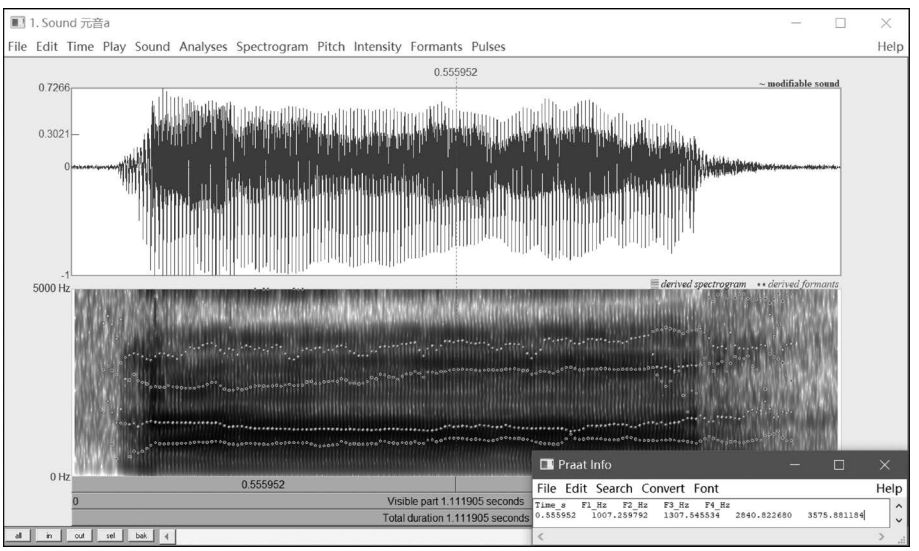

图 5-15　元音 a 的共振峰

从图 5-15～图 5-17 中可以看出，元音 a 的 F_1、F_2 约为 1007Hz、1308Hz，元音 o 的 F_1、

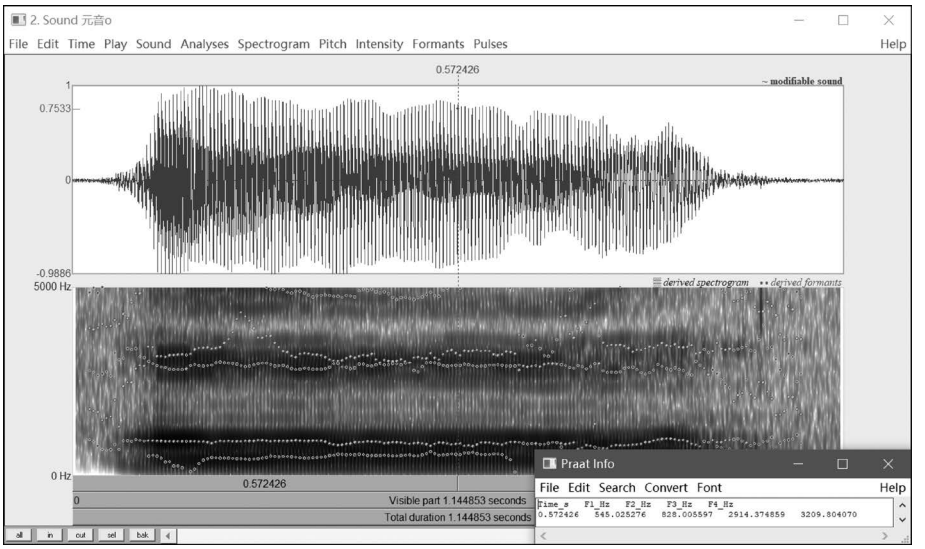

图 5-16　元音 o 的共振峰

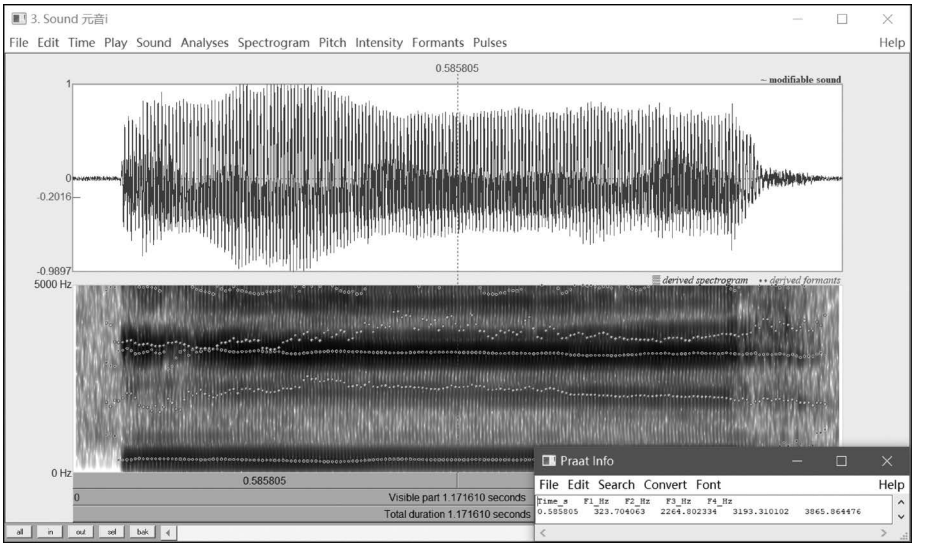

图 5-17　元音 i 的共振峰

F_2 约为 545Hz、828Hz；元音 i 的 F_1、F_2 约为 324Hz、2265Hz。当然，这只是某个人的共振峰数据。那么，元音的第 1、第 2 共振峰是否有什么共性呢？大量实验表明，每个元音的第 1、第 2 共振峰大致在一个区间内，它们的位置关系如图 5-18 所示。由于女性的声音频率高于男性的频率，因此总体而言，女声的共振峰也处于区间内频率较高的位置，男声则处于频率较低的位置。

以上是单元音的共振峰图，复元音（复韵母）中的共振峰要复杂一些。复韵母 ai 的语谱图如图 5-19 所示，其中语音部分大致可以分成 3 部分：左边的 a 段、右边的 i 段和中间的过

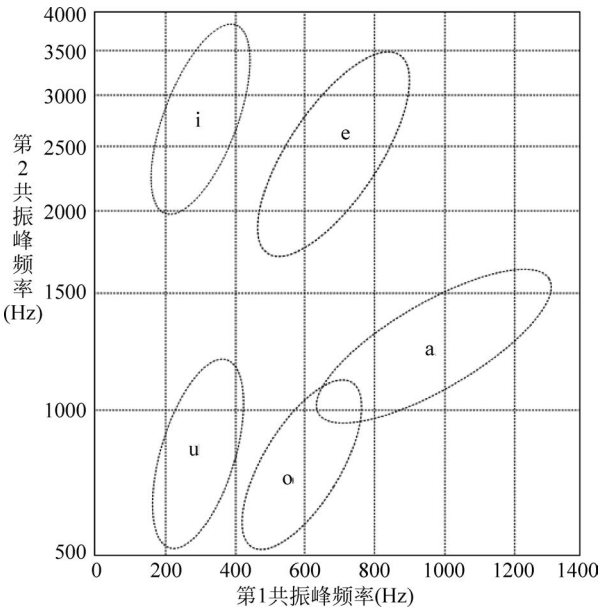

图 5-18　元音的共振峰分布图

渡段。该图下方有两个箭头，分别指向 a 段和 i 段某处，这两处的共振峰频率见下方标注。左方箭头处的 F_1 和 F_2 约为 754Hz 和 1316Hz，右方箭头处的 F_1 和 F_2 约为 251Hz 和 2418Hz，均位于元音 a 和 i 的频率范围内。不难看出，第 1 共振峰其实是逐渐降低的，因为 i 的 F_1 比 a 的 F_1 要低，而第 2 共振峰则是上升的，因为 i 的 F_2 比 a 的 F_2 要高不少，在 a 段结束后有一段甚至是急速上升的，之后再慢慢过渡到 i 的 F_2。

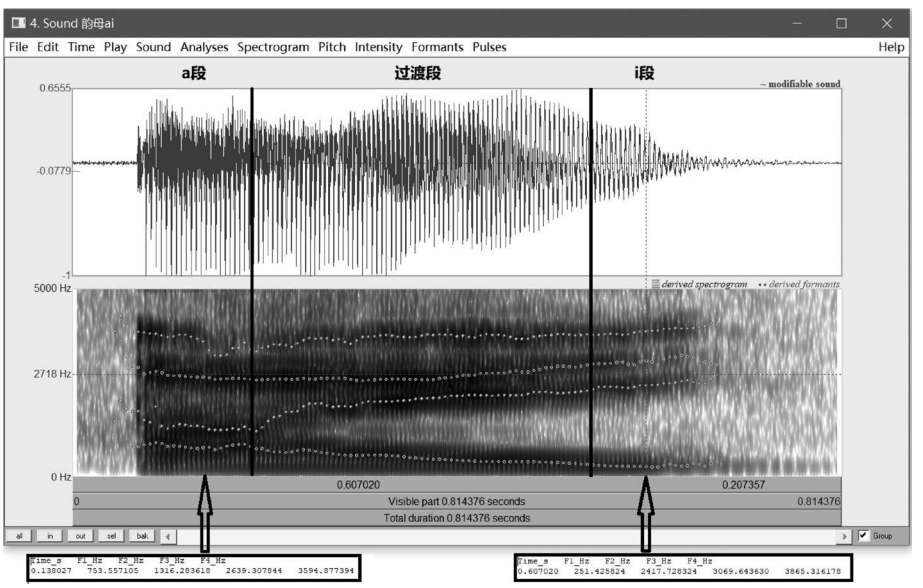

图 5-19　复韵母 ai 的语谱图

用同样的方法可以观察其他韵母的共振峰，有的韵母甚至更为复杂，如复韵母中的 uai 和鼻韵母中的 uang，有兴趣的读者可以自行研究。

5.4　语音端点检测

在对语音信号进行处理时，语音的端点检测是非常重要的一环。语音端点检测（End-point Detection，EPD）是指从包含语音的信号中确定语音的起点和终点。本节将介绍几种常用的端点检测方法。

5.4.1　音量法

基于音量进行端点检测是最简单高效的方法，Librosa 中的 trim()函数就是基于音量（分贝数）来判断是否是静音段，从而对音频头尾进行修剪，该函数原型如下：

```
librosa.effects. trim(y, *, top_db=60, ref=np.max, frame_length=2048, hop_
length =512, aggregate =np.max) -> Tuple[np.ndarray, np.ndarray]

【参数说明】
(1) y：待处理音频信号。
(2) top_db：最高的分贝数，将此阈值以下视为静音。
(3) ref：参考振幅。
(4) frame_length：每个分析帧的样本数。
(5) hop_length：帧移。
(6) aggregate：总计，用于通道间总计。

【返回值】
(1) y_trimmed：修剪后的信号。
(2) index：修剪位置的索引值，每个通道有两个值。
```

下面用一个简单的例子演示用 trim()函数进行静音检测的方法，代码如下：

```
#第 5 章/EPD_trim.py

import librosa
import librosa.display
import matplotlib.pyplot as plt

#读取音频文件并修剪
y, sr = librosa.load('wav/withblank.wav', sr=None)
y2, index = librosa.effects.trim(y, top_db=25)
print(len(y), len(y2))
print(index)

#绘制修剪前后波形图
fig, ax = plt.subplots(nrows=2, ncols=1)
librosa.display.waveshow(y, sr=sr, ax=ax[0])
```

```
ax[0].vlines(index[0]/sr, -0.5, 0.5, colors='r') #左分割线
ax[0].vlines(index[1]/sr, -0.5, 0.5, colors='r') #右分割线
librosa.display.waveshow(y2, sr=sr, ax=ax[1])
plt.show()
```

程序的运行结果如图 5-20 所示，图中上半部分是原波形图，下半部分是修剪后语音的波形图。程序中将阈值设为 25dB，凡是低于此值的都视作静音，返回值中的 index 显示了切割位置，索引位置为 28672 和 101376，如图 5-21 所示，其间有 72704 个样本。原音频信号共 121200 个样本，处理后的信号有 72704 个样本，与索引位置间的距离一致。为了便于观察，上方图中标出了两个索引的位置。

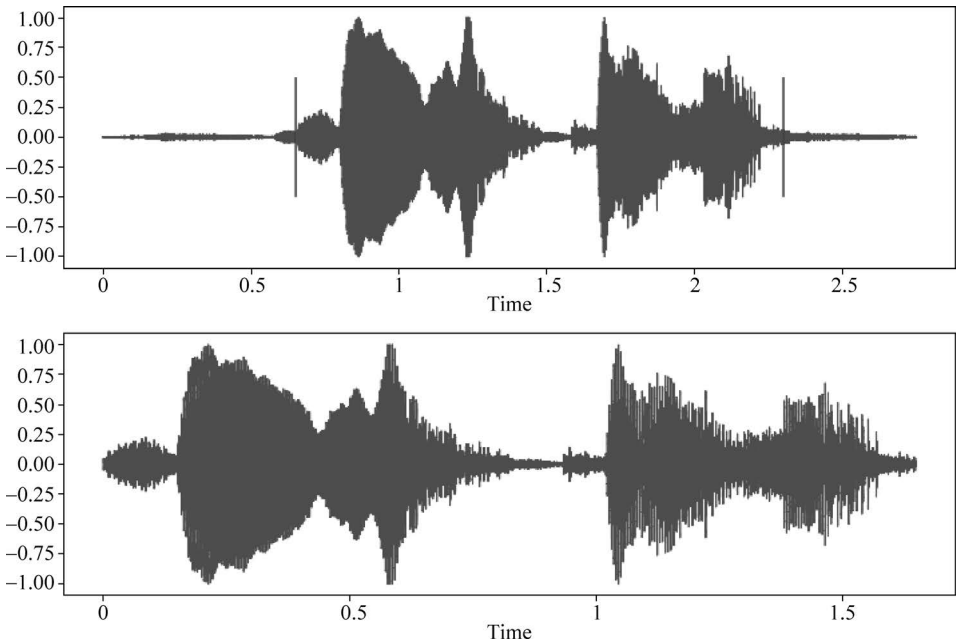

图 5-20　EPD_trim.py 运行结果

```
121200 72704
[ 28672 101376]
```

图 5-21　切割位置索引

不过，trim()函数只能对两端进行切割，而在语音序列中音节与音节之间也会有静音段，对这些位置进行切割需要用 librosa.effects.split()函数，该函数原型如下：

```
librosa.effects.split(y, *, top_db=60, ref=np.max, frame_length=2048, hop_
length=512, aggregate=np.max) -> np.ndarray
```

【参数说明】
(1) y：待处理音频信号。
(2) top_db：最高的分贝数，将此阈值以下视为静音。

```
(3) ref：参考振幅。
(4) frame_length：每个分析帧的样本数。
(5) hop_length：帧移。
(6) aggregate：总计，用于通道间总计。

【返回值】
intervals：切割点的数组，intervals[i]==(start_i,end_i)是切割点的起始和终点。
```

上述 split()函数只是标出切割点的位置，如果需要将静音段去除的音频重新组成一个文件，则需要用 librosa.effects.remix()函数进行处理，该函数原型如下：

```
librosa.effects.remix(y, intervals, *, align_zeros=True) -> np.ndarray

【参数说明】
(1) y：音频时间序列。
(2) intervals：指示起始和终点的数组。
(3) align_zeros：如果为 True，则 interval 的分界处将匹配到最近的过零点。

【返回值】
y_remix：重混后的音频信号。
```

下面的例子将通过 split()和 remix()两个函数对音频进行检测并重新组合成一个新文件，代码如下：

```
#第 5 章/EPD_split.py

import librosa
import librosa.display
import matplotlib.pyplot as plt

#读取音频文件并分割、重混
y, sr = librosa.load('wav/withblank.wav', sr=None)
intervals = librosa.effects.split(y, top_db=25)   #分割
y2 = librosa.effects.remix(y, intervals)          #重混
print(len(y), len(y2))
print(intervals.shape)
print(intervals)

#绘制处理前后波形图
fig, ax = plt.subplots(nrows=2, ncols=1)
librosa.display.waveshow(y, sr=sr, ax=ax[0])
for i in intervals:
   ax[0].vlines(i[0]/sr, -0.5, 0.5, colors='r')
   ax[0].vlines(i[1]/sr, -0.5, 0.5, colors='r')

librosa.display.waveshow(y2, sr=sr, ax=ax[1])
plt.show()
```

程序的运行结果如图 5-22 所示，split()函数从原始语音中分离出两个非静音段，经过

重混后形成新的音频信号，样本数从原来的 121200 个减少到了 68578 个，如图 5-23 所示。

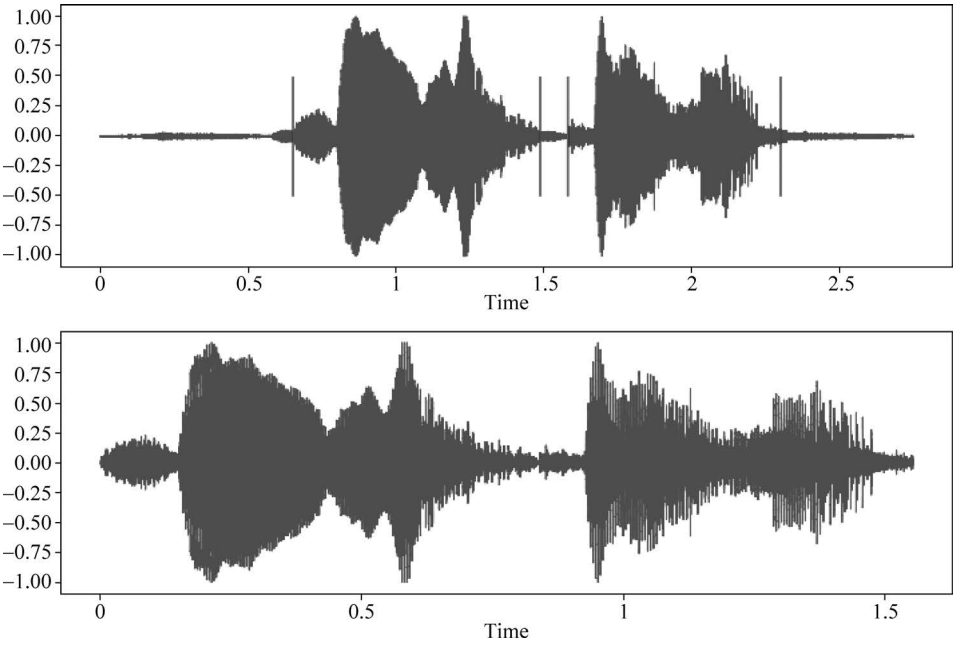

图 5-22　EPD_split.py 运行结果

```
121200 68578
(2, 2)
[[ 28672  65536]
 [ 69632 101376]]
```

图 5-23　程序输出的样本数等数据

基于音量的端点检测非常简单，但也有着相当的局限性。由于音量是判断端点的唯一标准，因而阈值的选择显得非常重要。如果语音较为干净且没有什么噪声，则此方法的效果也会不错，但是如果噪声较大或者说话时音量忽高忽低，则效果就差强人意了。

5.4.2　平均能量法

基于短时平均能量进行端点检测也是一种较为简单的办法。在 4.3.2 节曾介绍过平均能量的计算方法，并据此绘制了平均能量图，在此基础上设定阈值也能进行端点检测，如图 5-24 所示。根据图中的端点可以将静音段去除，重混后就是去静音后的语音信号，代码从略。

5.4.3　双门限法

双门限法是基于短时平均能量和过零率进行端点检测的一种方法。汉语中的韵母能量较高，可以通过平均能量来判断，而声母频率较高，可通过过零率来判断，两者的结果就能找出汉语的声母和韵母，从而进行端点识别。

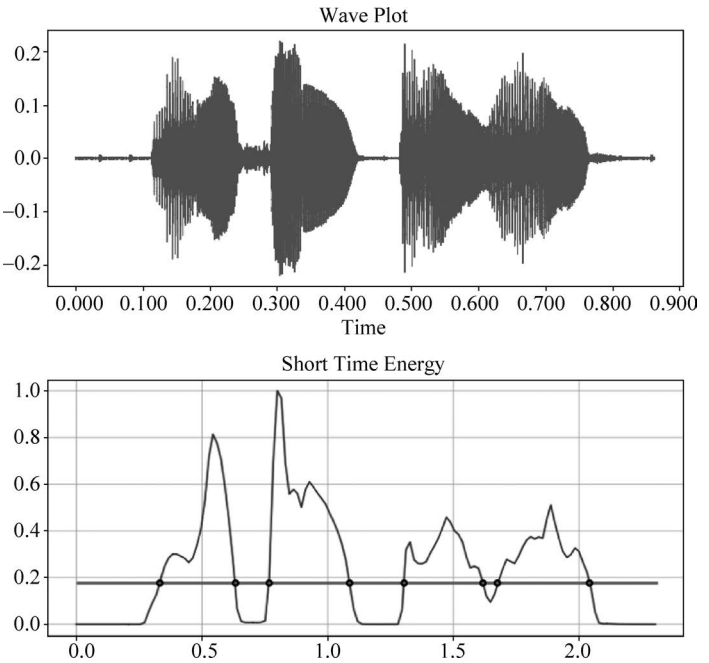

图 5-24　平均能量法进行端点检测

双门限法进行语音端点检测的过程如图 5-25 所示，具体如下：

(1) 在短时能量线上设定一个较高的阈值 T_1 并据此找到端点 A 与 B。

(2) 在短时能量线上设定一个较低的阈值 T_2 并将(1)中的端点向外延伸找到 T_2 与能量线的交点 C 与 D。

(3) 在过零率曲线上设置阈值 T_3，并将(2)中获得的端点向外延申至 T_3 与过零率曲线的交点 E 与 F，这就是这段语音的起点和终点。

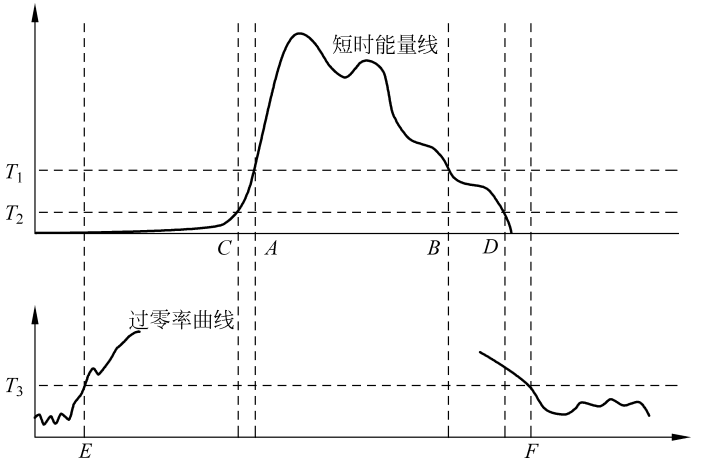

图 5-25　双门限法进行语音端点检测的过程

语音端点检测的方法还有很多，限于篇幅就不一一介绍了。

5.5 基音估计

声音可以分成纯音和复合音，而大多数声音属于复合音。通过傅里叶变换可以把复合音分解为一定数目的纯音，称为分音，其中振幅最大、频率最低的分音就是基音，其他分音的振幅一般比基音小，而频率则是基音的整数倍，称为陪音，在音乐中称为泛音。基音的振动频率被称为基音频率，简称基频，它的倒数称为基音周期。窄带语谱图中一条条水平条纹自下而上依次表示元音的各个谐波，其中最下面的一条通常就是基音，如图 5-26 所示。

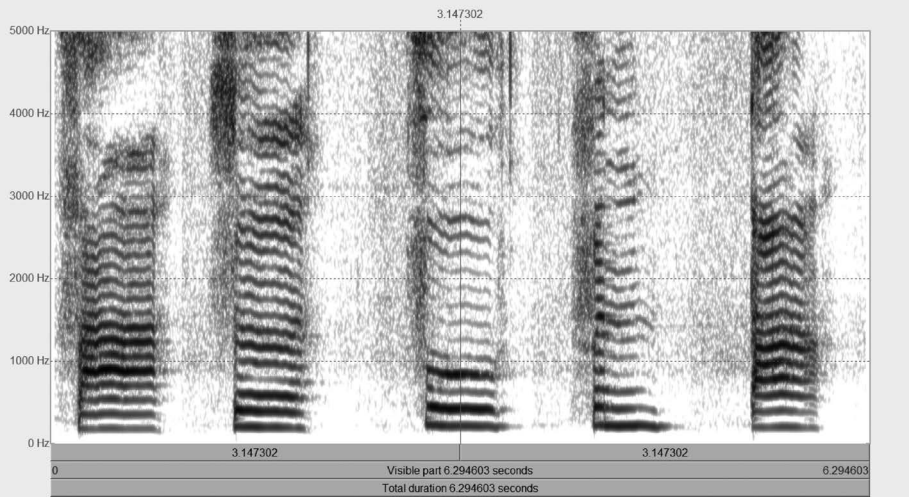

图 5-26　五个汉字的窄带语谱图

基音的检测和估计是语音处理中一个十分重要的问题，但同时也是一个相当棘手的问题，因为基音频率的精确估计实际上相当困难。首先，不同的人的基音频率往往是不一样的。一般来讲，男声频率较低，女声和童声频率较高，其次，一个人的基音频率受到多种音素的影响。声带结构和发音习惯构成了个人声音的特质，但是随着年龄变化，这些特质也会发生变化，而由于环境的影响或者说话人情绪的变化，即使是同一个字的发音，其基频也会有所不同。

尽管基音估计存在着诸多困难，但是鉴于其重要性，对基音估计的研究始终在进行。

由于一段语音的基频往往是变化的，所以基音估计的第 1 步是对音频信号进行分帧，然后逐帧提取基频。

基频提取的方法大致可分为时域法和频域法两大类。时域法是以波形图为基础，寻找波形的最小正周期。频域法则先通过傅里叶变换得到频谱，频谱上基频的整数倍处会有尖峰，频域法就是求出这些尖峰频率的最大公约数，如图 5-27 所示。

需要注意的是，并非每帧都有基频，提取基频时还需要判断基频的有无。此外，逐帧提

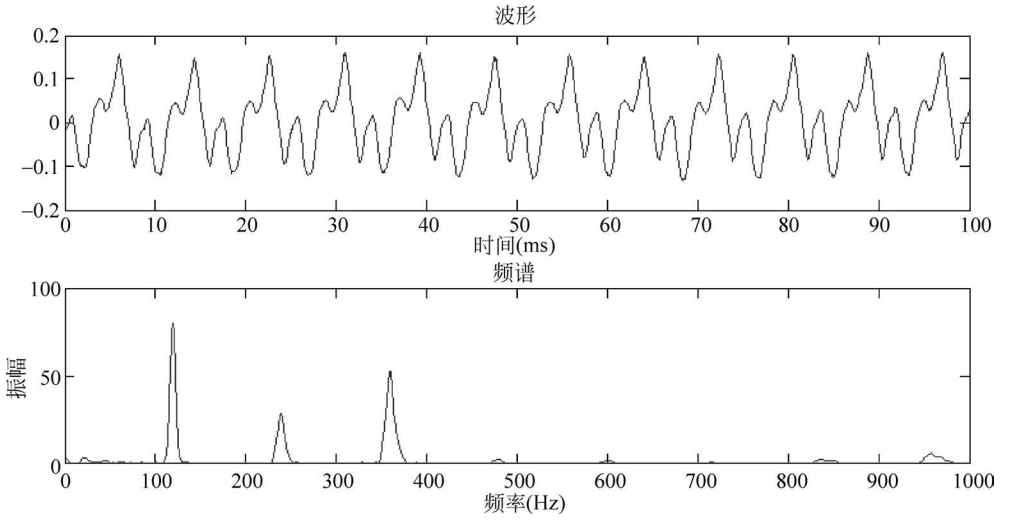

图 5-27　频域法基频提取示意图

取的基频常常含有错误，其中最常见的错误是倍频错误和半频错误，即提取出的基频是实际基频的数倍或者一半。

基音检测的算法很多，常见的有自相关法、倒谱法、YIN 算法、pYIN 算法等。Librosa 中提供了用 YIN 算法和 pYIN 算法进行基音检测的函数。

YIN 算法的名称取自东方哲学中阴阳的阴，算法的核心思想是在差函数上寻找谷值，而不是在自相关函数上寻找峰值。该算法出自一篇名为 *YIN，A Fundamental Frequency Estimator for Speech and Music* 的论文，其在 Librosa 中的函数原型如下：

```
librosa.yin(y, *, fmin, fmax, sr=22050, frame_length =2048, win_length=None, hop
_length =None, trough _threshold=0.1, center =True, pad _mode="constant") ->
np.ndarray

【参数说明】
(1) y：音频时间序列。
(2) fmin：最小频率，单位为 Hz，推荐值为 librosa.note_to_hz('C2')，约 65Hz。
(3) fmax：最大频率，单位为 Hz，推荐值为 librosa.note_to_hz('C7')，约 2093Hz。
(4) sr：y 的采样率。
(5) frame_length：帧长。
(6) win_length：窗长。
(7) hop_length：帧移。
(8) trough_threshold：峰值估计的绝对阈值。
(9) center：是否中心对齐。
(10) pad_mode：填充模式，仅在 center=True 时有效。

【返回值】
f0：基频的时间序列，单位为 Hz。
```

下面举一个简单的例子说明其用法，代码如下：

```
#第 5 章/pitch_yin.py

import librosa
import matplotlib.pyplot as plt

#生成啁啾信号并提取基频
fmin = 440
fmax = 880
y = librosa.chirp(fmin=fmin, fmax=fmax, duration=1.0)
f0 = librosa.yin(y, fmin=fmin, fmax=fmax)
print(f0)

#绘制基频图
t = librosa.times_like(f0)
plt.plot(t, f0, '_', linewidth=1)
plt.xlabel('Time')
plt.ylabel('F0')
```

程序运行后将产生如图 5-28 所示的基频序列，绘制出的基频图则如图 5-29 所示。

```
[750.1475413  444.47709273 447.62799609 454.81924124 462.2711191
 469.77874612 477.39861342 485.11984747 492.97368664 501.04255072
 509.1062392  517.328343   525.77761824 534.29107266 542.92827901
 551.80305899 560.6965497  569.77855225 579.07684054 588.40314894
 598.06362901 607.73298916 617.58467429 627.63081002 637.69327889
 648.17272341 658.47529799 669.40385367 680.07206845 691.32944244
 702.37915636 713.95022932 725.38541689 737.30226567 749.06453681
 761.41802069 773.427765   786.41036729 798.84629944 812.11180987
 825.14711758 838.55610283 852.32846141 882.        ]
```

图 5-28　程序输出的基频序列

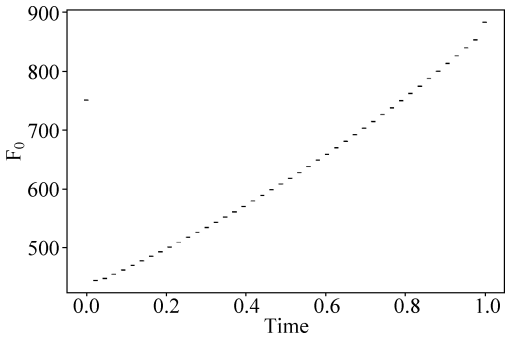

图 5-29　pitch_yin. py 生成的基频序列图

Librosa 中提供的另一个基音估计算法是 pYIN 算法，出自 Matthias Mauch 和 Simon Dixon 于 2014 年发表的一篇名为 *pYIN：A Fundamental Frequency Estimator Using Probabilistic Threshold Distributions* 的论文。该算法是 YIN 算法的改进版，它先采用 YIN 算法计算 F_0 的候选值及概率，然后用 Viterbi 算法估计出最有可能的 F_0 序列。

该函数在 Librosa 中的原型如下：

```
librosa.pyin(y, *, fmin, fmax, sr=22050, frame_length=2048, win_length=None, hop_
length=None, n_thresholds=100, beta_parameters=(2, 18), boltzmann_parameter=2,
resolution=0.1, max_transition_rate=35.92, switch_prob=0.01, no_trough_prob=
0.01, fill_na=np.nan, center=True, pad_mode="constant") -> Tuple[np.ndarray, np.
ndarray, np.ndarray]
```

【参数说明】

(1) y：音频时间序列。
(2) fmin：最小频率，单位为 Hz，推荐值为 librosa.note_to_hz('C2')，约 65Hz。
(3) fmax：最大频率，单位为 Hz，推荐值为 librosa.note_to_hz('C7')，约 2093Hz。
(4) sr：y 的采样率。
(5) frame_length：帧长。
(6) win_length：窗长。
(7) hop_length：帧移。
(8) n_thresholds：峰值估计阈值数。
(9) beta_parameters：Beta 分布的 shape 参数。
(10) boltzmann_parameter：玻耳兹曼分布的 shape 参数。
(11) resolution：音高 bins 的分辨率。
(12) max_transition_rate：最大音高跃迁率。
(13) switch_prob：从清音转浊音或从浊音转清音的转换概率。
(14) no_trough_prob：当无波谷低于阈值时添加到全局最小值的最大概率。
(15) fill_na：清音帧的 F0 的默认值。
(16) center：是否中心对齐。
(17) pad_mode：填充模式，仅在 center=True 时有效。

【返回值】

(1) f0：基频的时间序列，单位为 Hz。
(2) voiced_flag:是否是浊音帧的布尔值标志的时间序列。
(3) voiced_prob：某帧是浊音帧的概率的时间序列。

下面的例子将用 pYIN 算法对一段音乐进行基音估计并在语谱图上进行标注，代码如下：

```
#第 5 章/pitch_pyin.py

import librosa
import numpy as np
import matplotlib.pyplot as plt

#读取音频文件
y, sr = librosa.load('wav/trumpet.wav')
mag = np.abs(librosa.stft(y))
db = librosa.amplitude_to_db(mag, ref=np.max)

#用 pYIN 算法进行基音估计
fmin=librosa.note_to_hz('C2')
fmax=librosa.note_to_hz('C7')
f0, flag, prob = librosa.pyin(y, fmin=fmin, fmax=fmax)
```

```
#绘制语谱图并标出基音轨迹
t = librosa.times_like(f0)
fig, ax = plt.subplots()
graph = librosa.display.specshow(db, x_axis='time', y_axis='log', ax=ax)
ax.set(title='F0 by pYIN')
fig.colorbar(graph, ax=ax, format="%+2.f dB")
ax.plot(t, f0, label='F0', color='blue', linewidth=5)
ax.legend(loc='upper right')
```

程序的运行结果如图 5-30 所示，图中用蓝色线条标出了基音序列，这段线条与语谱图中最下面的条纹重合。

彩图

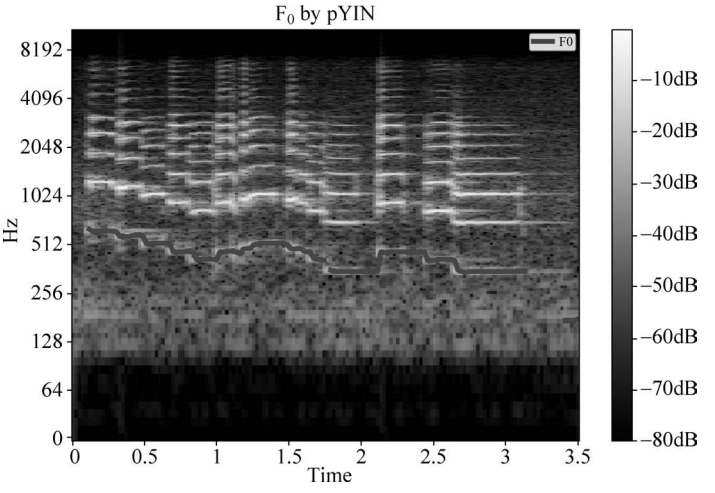

图 5-30　pitch_pyin. py 运行结果

5.6　梅尔倒谱系数

梅尔倒谱系数（Mel-Frequency Cepstral Coefficients，MFCC）是一种音频特征提取方法，常用于语音识别等领域，其流程如图 5-31 所示。

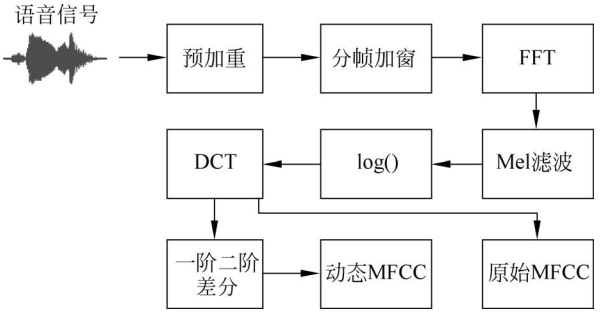

图 5-31　梅尔倒谱系数计算流程图

5.6.1　MFCC 特征提取步骤

MFCC 特征提取分为以下几步。

1. 预加重

预加重的目的是对信号的高频部分进行强化处理。预加重处理其实是将语音信号通过一个高通滤波器，其函数表达式如下：

$$y(n)=x(n)-\alpha \cdot x(n-1) \tag{5-1}$$

其中，α 为预加重系数，一般取 0.97。

预加重前后的效果如图 5-32 所示，图中上半部分为预加重前的原始信号，下半部分为预加重后的信号。显而易见，经过预加重后高频部分得到了加强。

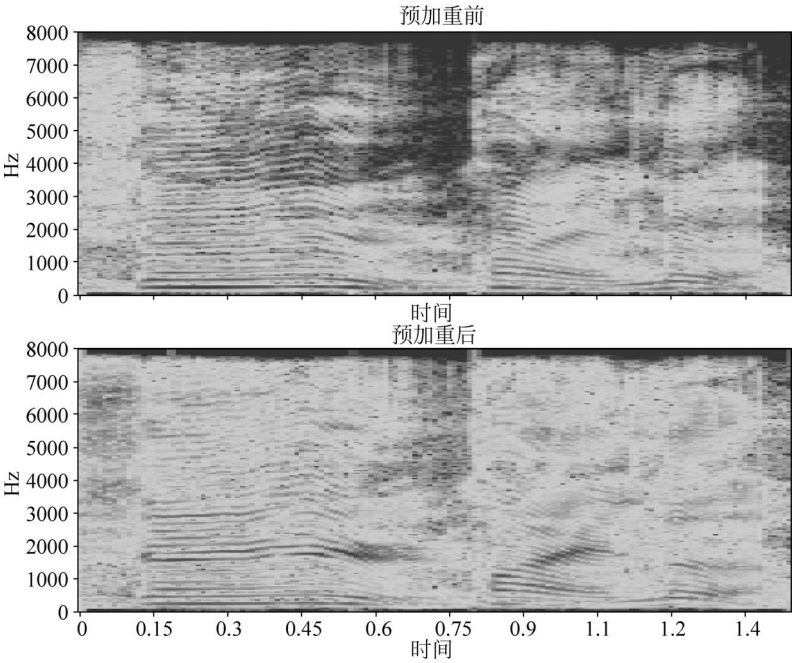

图 5-32　预加重前后的效果

2. 分帧与加窗

分帧就是将较长的信号切分成较短的小段，其中每段信号都称为一帧；相邻的两帧一般会有重叠的部分，每帧窗口都会沿时间轴作一次平移，称为帧移，如图 5-33 所示。假设帧长为 400，帧移为 160，则第 1 帧是第 1～400 个样本点，第 2 帧是第 161～560 个样本点，以此类推。如果最后一帧不足 400 个样本点，则一般在后面用 0 填充，填充的部分称为 Padding。与短时傅里叶变换一样，MFCC 提取时也需要加窗。

3. 傅里叶变换

经过上述预处理后的信号用傅里叶变换转换为频谱，此步和上一步合并其实就是短时

图 5-33　分帧示意图

傅里叶变换，MFCC 在实现时一般会调用此函数，Librosa 中的 mfcc()函数就是如此。傅里叶变换后可以得到信号的功率谱。

4. 梅尔滤波器组

接下来用梅尔滤波器组对功率谱进行滤波，计算每个滤波器里的能量。

梅尔滤波器组是由若干个带通滤波器组成的，每个滤波器都具有三角滤波特性。在梅尔频率范围内，这些滤波器是等带宽的，因此梅尔滤波器组表现为低频端密集、高频端稀疏的特性，如图 5-34 所示。将梅尔滤波器组和傅里叶变换后计算得到的功率谱相乘即可得到梅尔频谱。

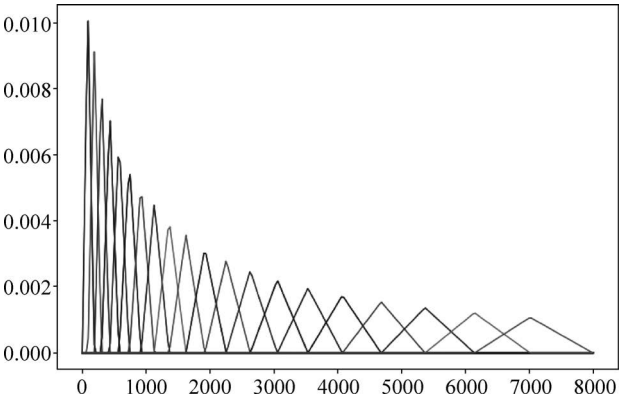

图 5-34　梅尔滤波器组

为了方便调用，Librosa 中提供了梅尔频谱函数，其原型如下：

```
librosa.feature.melspectrogram(*, y=None, sr=22050, S=None, n_fft=2048, hop_
length =512, win_length=None, window="hann", center=True, pad_mode ="constant",
power=2.0, n_mels=128, fmin=0.0, fmax=None, htk=False, norm="slaney", dtype=np.
float32) -> np.ndarray
```

【参数说明】
(1) y：音频时间序列。
(2) sr：y 的采样率。

(3) S：语谱图，如果提供该参数，则将直接用此计算。
(4) n_fft：快速傅里叶变换的序列长度。
(5) hop_length：帧移。
(6) win_length：窗长；如果未指定，则 win_length=n_fft，当然也可不同。
(7) window：指定的窗函数，默认为汉宁窗。
(8) center：是否中心对齐。
(9) pad_mode：填充模式，默认用 0 填充。
(10) power：梅尔频谱幅值的幂指数，默认值为 2.0。
(11) n_mels：梅尔滤波器的数量。
(12) fmin：最小频率。
(13) fmax：最大频率。
(14) htk：如果为 True，则采用 HTK 公式，否则采用 Slaney 公式。
(15) norm：归一化方式，具体可参照 librosa.util.normalize。
(16) dtype：输出时的数据类型，默认采用 32 位(单精度)浮点数。

【返回值】
S：梅尔频谱。

下面用一个例子说明梅尔频谱的计算及绘制方法，代码如下：

```
#第 5 章/mel_spectrogram.py

import numpy as np
import librosa
import librosa.display
import matplotlib.pyplot as plt

#参数设置
sr = 16000 #采样率
n_fft = 512
win_length = 512
hop_length = 256
n_mels = 128

#绘制梅尔滤波器组
melfilters = librosa.filters.mel(sr=sr, n_fft=n_fft, n_mels=n_mels, htk=True)
x = np.arange(melfilters.shape[1])*sr/n_fft
fig = plt.figure()
plt.plot(x, melfilters.T)
plt.title('Mel filters')
plt.show()

#绘制梅尔频谱图
y, fs = librosa.load('wav/shengrikuaile.wav', sr=sr)
fig = plt.figure()
mel_spec = librosa.feature.melspectrogram(y=y,
                                          sr=fs,
                                          n_fft=n_fft,
                                          win_length=win_length,
```

```
                                        hop_length=hop_length,
                                        n_mels=n_mels)
mel_db = librosa.power_to_db(mel_spec, ref=np.max)
img = librosa.display.specshow(mel_db, x_axis='time', y_axis='mel', sr=fs)
fig.colorbar(img, format='%+2.0f dB')
plt.title('Mel spectrogram')
plt.show()
```

程序运行后将输出相应的梅尔滤波器组和梅尔频谱图,如图 5-35 所示。

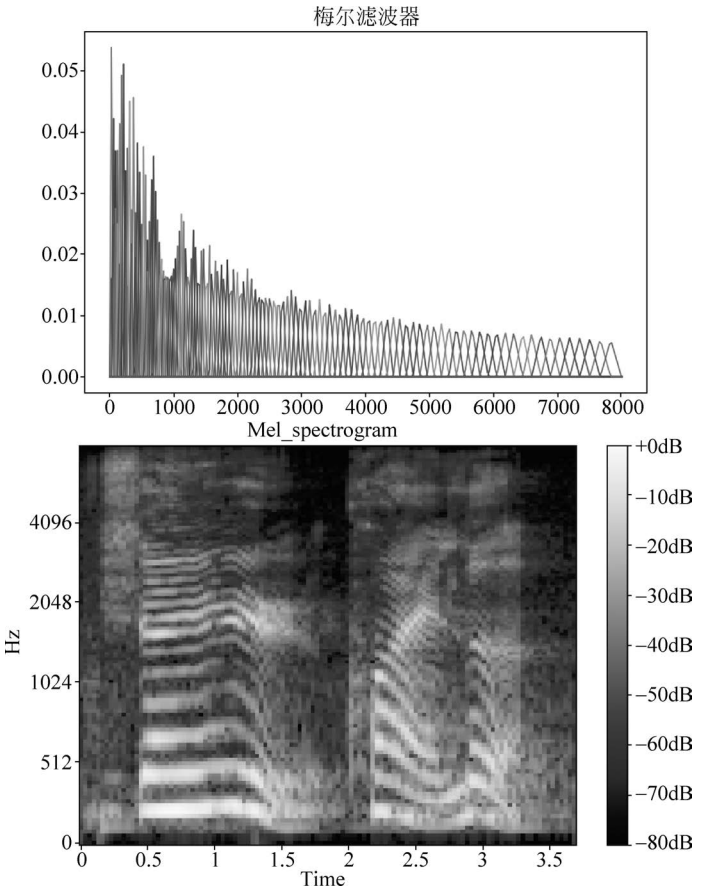

图 5-35 mel_spectrogram.py 运行结果

5. 离散余弦变换

接下来对通过梅尔滤波器的能量取对数,然后进行离散余弦变换(DCT),这样就得到了 MFCC 系数,离散余弦变换的目的是提取信号的包络。由于大部分信号数据集中在变换后的低频区,因此一般取每帧的前 13 个数字即可,这些数字就是 MFCC 特征。

6. Deltas 和 Delta-Deltas 特征

MFCC 特征描述了一帧语音信号的功率谱的包络信息,在识别元音时可以直接使用,

但是在识别辅音时还需要帧与帧之间的动态变换关系。对当前帧和前后两帧的 MFCC 进行差分计算的结果称为 ΔMFCC；同理，对 ΔMFCC 可以再次进行差分计算，其结果为 ΔΔMFCC。最后，将 MFCC、ΔMFCC 和 ΔΔMFCC 拼接起来，就得到了完整的 MFCC 特征。

5.6.2　MFCC 特征

MFCC 经常被用来进行语音识别，Librosa 中设有相应的函数，其函数原型如下：

```
librosa.feature.mfcc(*, y=None, sr=22050, S=None, n_mfcc=20, dct_type=2, norm=
"ortho", lifter=0, n_fft=2048, hop_length=512, win_length=None, window ="hann",
center=True, pad_mode="constant", power=2.0, n_mels=128, fmin=0.0, fmax=None,
htk=False, dtype=np.float32) -> np.ndarray

【参数说明】
(1) y：音频时间序列。
(2) sr：y 的采样率。
(3) S：log-power 梅尔频谱。
(4) n_mfcc：MFCC 系数的个数。
(5) dct_type：离散余弦变换(DCT)的类型，可选参数为 1,2,3。
(6) n_fft：快速傅里叶变换的序列长度。
(7) hop_length：帧移。
(8) win_length：窗长；如果未指定，则 win_length=n_fft。
(9) window：指定的窗函数，默认为汉宁窗。
(10) center：是否中心对齐。
(11) pad_mode：填充模式，默认用 0 填充。
(12) power：梅尔频谱幅值的幂指数，默认值为 2.0。
(13) n_mels：梅尔滤波器的数量。
(14) fmin：最小频率。
(15) fmax：最大频率。
(16) htk：如果为 True，则采用 HTK 公式而不是 Slaney 公式。
(17) dtype：输出时的数据类型，默认采用 32 位(单精度)浮点数。

【返回值】
M：MFCC 序列。
```

上述参数中的 S 是在计算离散余弦变换前的值，如果调用时有此参数，则将直接进行后续计算，从而大大减少计算时间。在 mfcc()函数的源代码中会对 S 是否存在进行判断，相应的代码如下：

```
def mfcc(
    *,
    y: Optional[np.ndarray] = None,
    sr: float = 22050,
    S: Optional[np.ndarray] = None,
    n_mfcc: int = 20,
    dct_type: int = 2,
    norm: Optional[str] = "ortho",
```

```
    lifter: float = 0,
    **kwargs: Any,
) -> np.ndarray:

    if S is None:
        # multichannel behavior may be different due to relative noise floor
differences between channels
       S = power_to_db(melspectrogram(y=y, sr=sr, **kwargs))

    M: np.ndarray = scipy.fftpack.dct(S, axis=-2, type=dct_type, norm=norm)[
       ..., :n_mfcc, :
    ]

    if lifter > 0:
        #shape lifter for broadcasting
        LI = np.sin(np.pi *np.arange(1, 1 + n_mfcc, dtype=M.dtype) / lifter)
        LI = util.expand_to(LI, ndim=S.ndim, axes=-2)

        M *= 1 + (lifter / 2) *LI
        return M
    elif lifter == 0:
        return M
    else:
         raise ParameterError(f"MFCC lifter={lifter} must be a non-negative
number")
```

调用 MFCC 函数进行特征提取非常方便，下面是一个简单的例子，代码如下：

```
#第 5 章/mfcc_all.py

import numpy as np
import librosa
import librosa.display
import matplotlib.pyplot as plt

#读取音频文件并计算 mfcc
y, fs = librosa.load('shengrikuaile.wav', sr=16000)
win_length = 512
hop_length = 256
n_fft = 512
n_mels = 128
n_mfcc = 13
mfcc = librosa.feature.mfcc(y=y,
                            sr=fs,
                            n_mfcc=n_mfcc,
                            win_length=win_length,
                            hop_length=hop_length,
                            n_fft=n_fft,
                            n_mels=n_mels,
```

```
                            dct_type=1
                            )

#计算 Δmfcc 和 ΔΔmfcc 并拼接
mfcc_d1 = librosa.feature.delta(mfcc)
mfcc_d2 = librosa.feature.delta(mfcc, order=2)
mfcc_all = np.concatenate([mfcc, mfcc_d1, mfcc_d2], axis=0)

#绘图并输出维度
fig = plt.figure()
img = librosa.display.specshow(mfcc_all, x_axis='time',
                               hop_length=hop_length, sr=fs)
fig.colorbar(img)
plt.show()

print(mfcc.shape)
print(mfcc_d1.shape)
print(mfcc_d2.shape)
print(mfcc_all.shape)
```

程序的运行结果如图 5-36 所示。此外，程序还输出了 MFCC、ΔMFCC、ΔΔMFCC 及最后拼接后的完整特征的维度值，如图 5-37 所示。

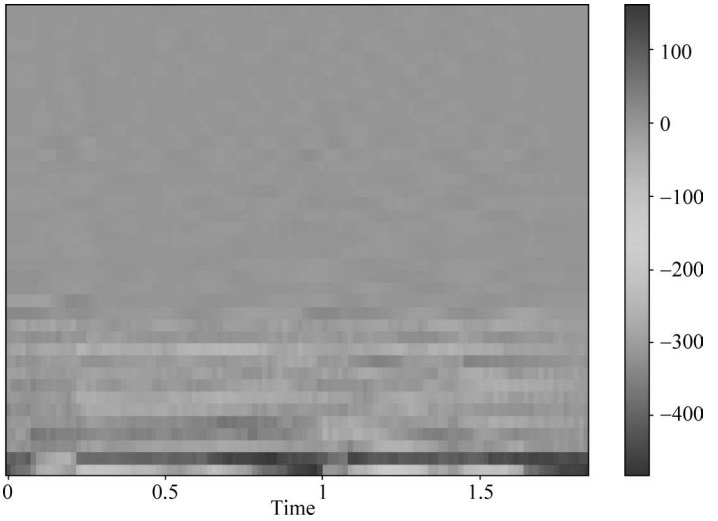

图 5-36　mfcc_all. py 运行结果

```
(13, 116)
(13, 116)
(13, 116)
(39, 116)
```

图 5-37　输出的维度信息

如果选取每帧的 MFCC 系数的第 1 个数字组成一个数组 MFCC0，则将在一定程度上体现出语音信号的特点和走势，如图 5-38 所示。图中上半部分是 shengrikuaile. wav 的波

形图，下半部分是 MFCC0 组成的折线图。音频文件有 29477 个采样点（采样率＝16kHz），而 MFCC0 仅用 116 个数字（分帧后共 116 帧）就描绘出了波形的轮廓，数据量大幅缩减，因而 MFCC0 可以看作波形的一个缩略图。

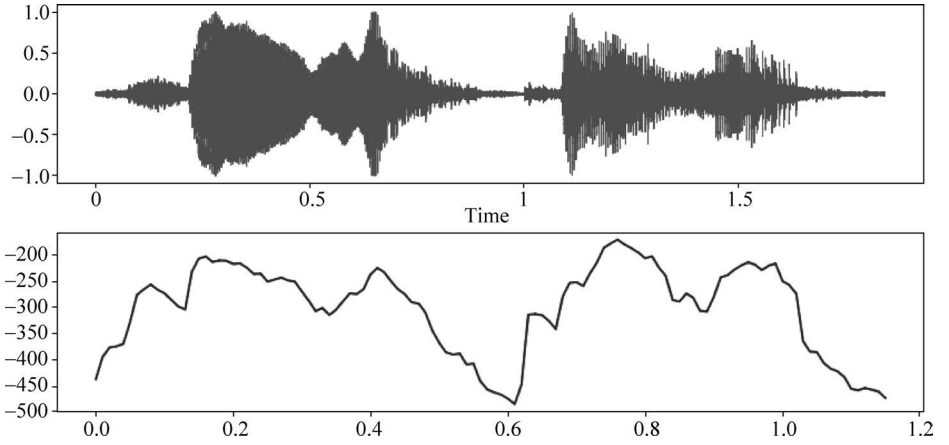

图 5-38　MFCC0 与原波形对比图

MFCC0 包含了语音信号的时域能量信息，因而也可以用作语音信号的端点检测。

5.6.3　Fbank 特征

值得一提的是，在计算 MFCC 的过程中，如果将最后一步离散余弦变换去掉，则得到的是 Fbank 特征（Filter bank 的简称）。在深度学习出现之前，MFCC 与 GMM-HMM 配合是语音识别的主流技术，然而，随着深度学习的飞速发展，人们逐渐发现 Fbank 在深度神经网络中的表现要大大优于 MFCC，因而 Fbank 大有取代 MFCC 之势。