

《多媒体技术》第 4 次作业

姓名：张半佛

完成日期：2019 年 11 月

题目：

- 1.录一段声音，内容为“间隔读出数字 0-4”，转为单声道、8khz、wav 格
- 2.每帧帧长 20ms160 个样点，使用汉明窗 hamming 加窗；
- 3.设定能量阈值，判断语音中无声、有声区间；
- 4.计算短时自相关 $R(k)$ ， k 取 20-100，计算基音频率；
- 5.信息预测：数字“0”的预测增益 EO/Ep （原始信号能量/残差信号能量），阶数 p 取 10。

解：

(1) 技术思路，主要计算公式

- ① 使用函数 `wavread()` 读取音频文件 01234.wav（单声道、采样率 8khz，时长 6s 整，共计 48000 个采样点）。并画图。
- ② 使用函数 `enframe()` 进行汉明窗分帧，窗口大小为 160，窗口偏移为 40。
- ③ 使用函数 `calEnergy()` 进行计算每一帧的能量，并画图。
- ④ 使用函数 `calzhuo()` 计算浊音对应的帧，这里的阈值选择 0.1，如果能量大于 0.1，则记为浊音。
- ⑤ 使用函数 `calxiangguan()` 计算浊音每一帧的基音频率，并画图。
- ⑥ 选取数字‘0’对应的帧，计算预测帧，两帧相减，求得残差帧，计算原始信号能量/残差信号能量。并画图表示数字‘0’的预测增益。
- ⑦ 选取数字‘0’中的一个帧第 180 帧，画图表示真实帧信号与预测信号。

(2) 核心代码（如果小于 100 行，贴出所有代码，如果大于 100 行，分段贴出核心代码）

主函数：

```
1. filename = '01234.wav'
2. data = wavread(filename)
3. nw = 160 #窗大小
4. inc = 40 #窗偏移
```

```

5. winfunc = signal.hamming(nw) #汉明窗
6. Frame,nf = enframe(data[0], nw, inc, winfunc) #分帧加窗
7. Energy=calEnergy(Frame,nf,nw) #计算每一帧的能量
8.
9. #画出每一帧的能量图
10. x = np.arange(1,len(Energy)+1)
11. plt.figure()
12. plt.title(u'每一帧的能量')
13. plt.xlabel(u'帧数')
14. plt.ylabel(u'能量')
15. plt.plot(x,Energy)
16. plt.show()
17.
18. #计算浊音对应的帧
19. zhuo=calzhuo(Energy,nf)
20. print(zhuo)
21.
22. #计算浊音每一帧的基音频率
23. xiangguanindex=[]
24. for i in range(5):
25.     for j in zhuo[i]:
26.         xiangguanindexi=calxiangguan(Frame,j,nw)
27.         xiangguanindex.append(xiangguanindexi)
28.     xiangguanindex.append(0 )
29. #画图
30. x = np.arange(1,len(xiangguanindex)+1)
31. plt.figure()
32. plt.title(u'浊音每一帧的基音频率')
33. plt.ylabel(u'基音频率')
34. plt.xlabel(u'每一部分的帧')
35. plt.plot(x,xiangguanindex)
36. plt.show()
37.
38.
39. #求预测增益
40. ans=[]
41. for i in zhuo[0]:
42.     A,E,K=lpc.levinson_1d(Frame[i],10)
43.     pre= signal.lfilter(A,[1],Frame[i])
44.     err=Frame[i]-pre
45.     erreng=calEnergy(err,1,160)
46.     result=Energy[i]/erreng
47.     ans.append(result)
48.     if i==180 :

```

```

49.     plt.figure()
50.     plt.subplot(2,1,1)
51.     x = np.arange(1, len(Frame[180]) + 1)
52.     plt.plot(x, Frame[180])
53.     plt.title(u'原始 180 帧')
54.     plt.subplot(2, 1, 2)
55.     x = np.arange(1, len(pre) + 1)
56.     plt.plot(x, pre)
57.     plt.title(u'预测 180 帧')
58. #画图
59. x = np.arange(1,len(ans)+1)
60. plt.figure()
61. plt.title(u'数字 0 的预测增益')
62. plt.plot(x,ans)
63. plt.show()

```

读取文件函数：

```

1. def wavread(filename):
2.     f = wave.open(filename, 'rb')
3.     params = f.getparams()
4.     nchannels, sampwidth, framerate, nframes = params[:4]
5.     strData = f.readframes(nframes) # 读取音频，字符串格式
6.     waveData = np.fromstring(strData, dtype=np.int16) # 将字符串转化为 int
7.     f.close()
8.     waveData = waveData * 1.0 / (max(abs(waveData))) # wave 幅值归一化
9.     #画图
10.    time = np.arange(0, nframes) * (1.0 / framerate)
11.    plt.figure()
12.    plt.xlabel("Time(s)")
13.    plt.ylabel("Amplitude")
14.    plt.title("Single channel wavedata")
15.    plt.plot(time, waveData)
16.    plt.show()
17.    waveData = np.reshape(waveData, [nframes, nchannels]).T
18.
19.    return waveData

```

加窗分帧函数：

```

1. def enframe(signal, nw, inc, winfunc):
2.     '''将音频信号转化为帧。
3.     参数含义：
4.     signal:原始音频型号

```

```

5.     nw:每一帧的长度(这里指采样点的长度, 即采样频率乘以时间间隔)
6.     inc:相邻帧的间隔 (同上定义)
7.     '''
8.     signal_length=len(signal) #信号总长度
9.     if signal_length<=nw: #若信号长度小于一个帧的长度, 则帧数定义为 1
10.         nf=1
11.     else: #否则, 计算帧的总长度
12.         nf=int(np.ceil((1.0*signal_length-nw+inc)/inc))
13.     pad_length=int((nf-1)*inc+nw) #所有帧加起来总的铺平后的长度
14.     zeros=np.zeros((pad_length-signal_length,)) #不够的长度使用 0 填补, 类似于
        FFT 中的扩充数组操作
15.     pad_signal=np.concatenate((signal,zeros)) #填补后的信号记为 pad_signal
16.     indices=np.tile(np.arange(0,nw),(nf,1))+np.tile(np.arange(0,nf*inc,inc),
        (nw,1)).T #相当于对所有帧的时间点进行抽取, 得到 nf*nw 长度的矩阵[0-160],[40-
        200]...
17.     indices=np.array(indices,dtype=np.int32) #将 indices 转化为矩阵
18.     frames=pad_signal[indices] #得到帧信号
19.     win=np.tile(winfunc,(nf,1)) #window 窗函数, 这里默认取 1
20.     return frames*win,nf #返回帧信号矩阵

```

计算帧能量函数:

```

1. def calEnergy(Frame,nf,nw):
2.     '''
3.     计算每一帧的能量
4.     :param Frame: 要计算的帧数组
5.     :param nf: 数组行, 多少个帧
6.     :param nw: 数组列, 每个帧多少个采样点
7.     :return: 返回所有帧的能量, 一个列表
8.     '''
9.     result=[]
10.    if nf==1:
11.        sum = 0
12.        for j in range(nw):
13.            sum = sum + Frame[j] * Frame[j]
14.        result.append(sum)
15.    else:
16.        for i in range(nf):
17.            sum=0
18.            for j in range(nw):
19.                sum=sum+Frame[i][j]*Frame[i][j]
20.            result.append(sum)
21.    return result

```

计算浊音对应的帧索引:

```

1. def calzhuo(Energy,nf):
2.     """
3.     计算出浊音的索引
4.     :param Energy: 能量数组
5.     :param nf: 多少帧
6.     :return: 返回每一个浊音段对应的帧的索引
7.     """
8.     result=[]
9.     cache=[]
10.    for i in range(nf):
11.        if Energy[i]>0.1:
12.            cache.append(i)
13.            if Energy[i+1]<0.1 and Energy[i+2]<0.1:
14.                result.append(cache)
15.                cache=[]
16.    return result

```

计算自相关函数，返回该帧的基音频率：

```

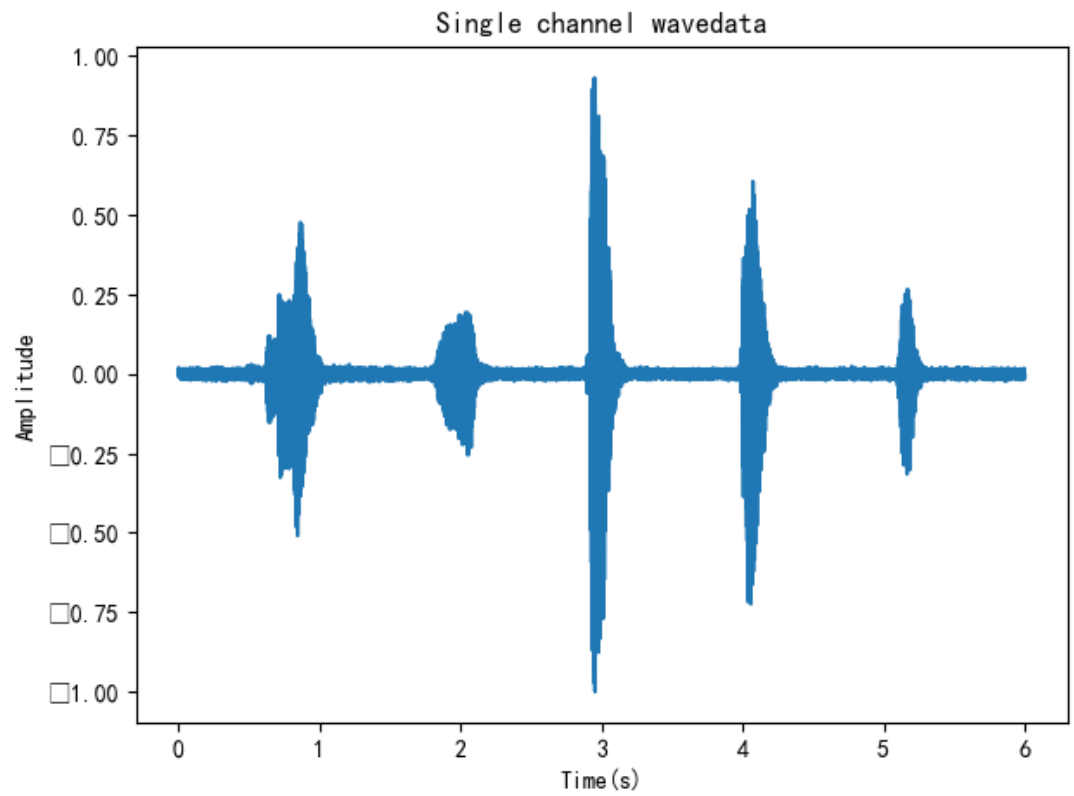
1. def calxiangguan(Frame,i,nw):
2.     """
3.     计算一帧自相关
4.     :param Frame: 加窗后的帧
5.     :param i: 对应帧的索引
6.     :param nw: 该帧有多少
7.     :return: 返回该帧的基音频率
8.     """
9.     sum=0
10.    result=[]
11.    result2=[]
12.    for k in range(0,100):
13.        for j in range(nw-k):
14.            sum=sum+Frame[i][j]*Frame[i][j+k]
15.        result.append(sum)
16.    #归一化
17.    for l in range(len(result)):
18.        result2.append(result[l]/result[0])
19.    return 8000/result2.index(max(result2[-80:]))

```

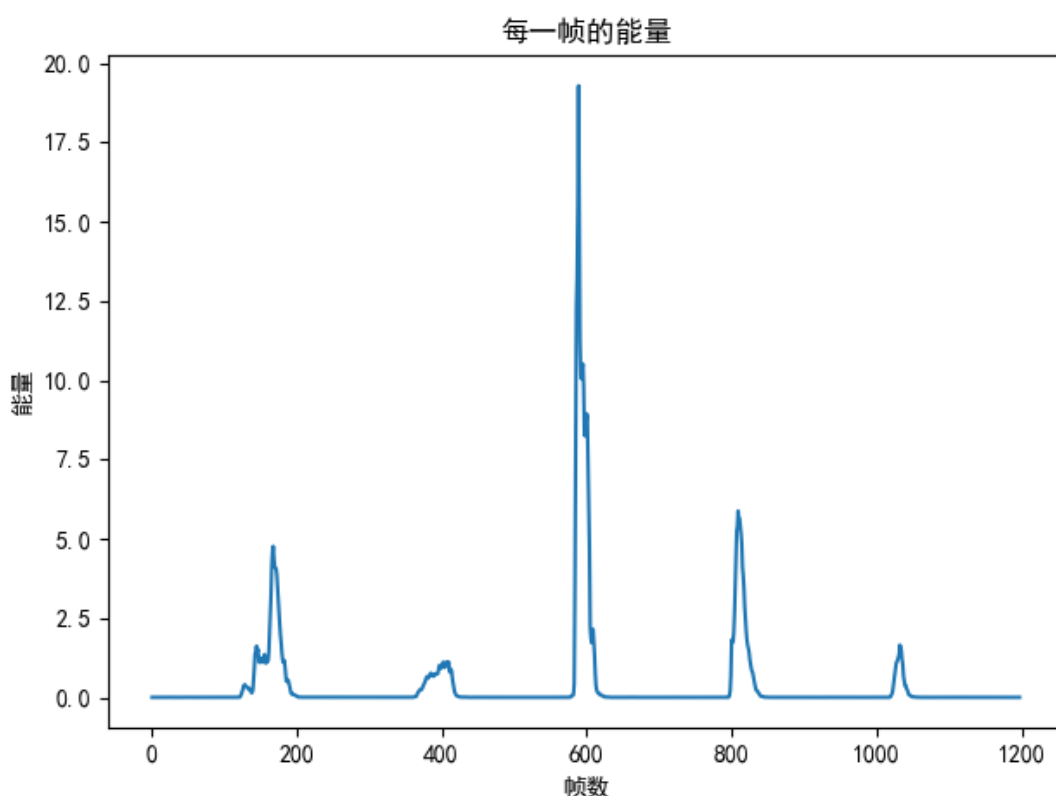
- (3) 运行结果：（包含主机配置、运行环境、开发环境、执行时间、输入文件名、输出文件名。可以给出简单的分析）

主机配置: ASUSTek Computer Inc., CPU:Inter i7-7500U RAM:12GB
运行环境: WIN10
开发环境: pycharm
执行时间: 2019.11.14
效果展示:

原始的音频信号:



每一帧的能量: (由于共有 48000 个采样点, 窗口大小为 160, 偏移为 40, 可算出共有 1197 帧)



根据能量阈值 0.1，可算出每一个声音段对应的帧索引：

```
[123, 124, 125, 126, 127, 128, 129, 130, 131, 132, 133, 134, 135, 136, 137, 138, 139, 140, 141, 142, 143, 144, 145, 146, 147, 148, 149, 150, 151, 152, 153, 154, 155, 156, 157, 158, 159, 160, 161, 162, 163, 164, 165, 166, 167, 168, 169, 170, 171, 172, 173, 174, 175, 176, 177, 178, 179, 180, 181, 182, 183, 184, 185, 186, 187, 188, 189, 190, 191, 192, 193, 194], [366, 367, 368, 369, 370, 371, 372, 373, 374, 375, 376, 377, 378, 379, 380, 381, 382, 383, 384, 385, 386, 387, 388, 389, 390, 391, 392, 393, 394, 395, 396, 397, 398, 399, 400, 401, 402, 403, 404, 405, 406, 407, 408, 409, 410, 411, 412, 413, 414, 415, 416, 417, 418], [582, 583, 584, 585, 586, 587, 588, 589, 590, 591, 592, 593, 594, 595, 596, 597, 598, 599, 600, 601, 602, 603, 604, 605, 606, 607, 608, 609, 610, 611, 612, 613, 614, 615, 616, 617], [797, 798, 799, 800, 801, 802, 803, 804, 805, 806, 807, 808, 809, 810, 811, 812, 813, 814, 815, 816, 817, 818, 819, 820, 821, 822, 823, 824], [1038, 1039, 1040, 1041, 1042, 1043, 1044, 1045, 1046, 1047, 1048, 1049, 1050, 1051, 1052, 1053, 1054, 1055, 1056, 1057, 1058, 1059, 1060, 1061, 1062, 1063, 1064, 1065, 1066, 1067, 1068, 1069, 1070, 1071, 1072, 1073, 1074, 1075, 1076, 1077, 1078, 1079, 1080, 1081, 1082, 1083, 1084, 1085, 1086, 1087, 1088, 1089, 1090, 1091, 1092, 1093, 1094, 1095, 1096, 1097, 1098, 1099, 1100, 1101, 1102, 1103, 1104, 1105, 1106, 1107, 1108, 1109, 1110, 1111, 1112, 1113, 1114, 1115, 1116, 1117, 1118, 1119, 1120, 1121, 1122, 1123, 1124, 1125, 1126, 1127, 1128, 1129, 1130, 1131, 1132, 1133, 1134, 1135, 1136, 1137, 1138, 1139, 1140, 1141, 1142, 1143, 1144, 1145, 1146, 1147, 1148, 1149, 1150, 1151, 1152, 1153, 1154, 1155, 1156, 1157, 1158, 1159, 1160, 1161, 1162, 1163, 1164, 1165, 1166, 1167, 1168, 1169, 1170, 1171, 1172, 1173, 1174, 1175, 1176, 1177, 1178, 1179, 1180, 1181, 1182, 1183, 1184, 1185, 1186, 1187, 1188, 1189, 1190, 1191, 1192, 1193, 1194, 1195, 1196, 1197, 1198, 1199, 1200]
```

其中数字 0:

[123, 124, 125, 126, 127, 128, 129, 130, 131, 132, 133, 134, 135, 136, 137, 138, 139, 140, 141, 142, 143, 144, 145, 146, 147, 148, 149, 150, 151, 152, 153, 154, 155, 156, 157, 158, 159, 160, 161, 162, 163, 164, 165, 166, 167, 168, 169, 170, 171, 172, 173, 174, 175, 176, 177, 178, 179, 180, 181, 182, 183, 184, 185, 186, 187, 188, 189, 190, 191, 192, 193, 194]

数字 1:

[366, 367, 368, 369, 370, 371, 372, 373, 374, 375, 376, 377, 378, 379, 380, 381, 382, 383, 384, 385, 386, 387, 388, 389, 390, 391, 392, 393, 394, 395, 396, 397, 398, 399, 400, 401, 402, 403, 404, 405, 406, 407, 408, 409, 410, 411, 412, 413, 414, 415, 416, 417, 418]

数字 2:

[582, 583, 584, 585, 586, 587, 588, 589, 590, 591, 592, 593, 594, 595, 596, 597, 598, 599, 600, 601, 602, 603, 604, 605, 606, 607, 608, 609, 610, 611, 612, 613, 614, 615, 616, 617]

数字 3:

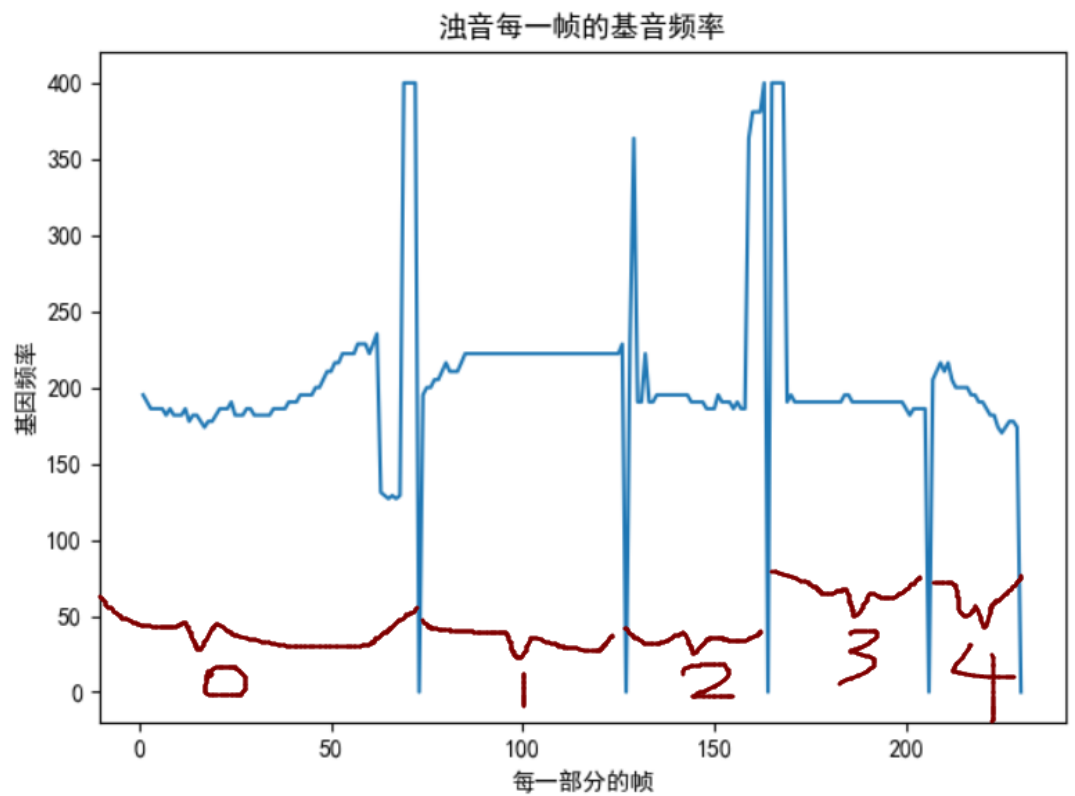
[797, 798, 799, 800, 801, 802, 803, 804, 805, 806, 807, 808, 809, 810, 811, 812, 813, 814, 815, 816, 817, 818, 819, 820, 821, 822, 823, 824]

825, 826, 827, 828, 829, 830, 831, 832, 833, 834, 835, 836, 837]

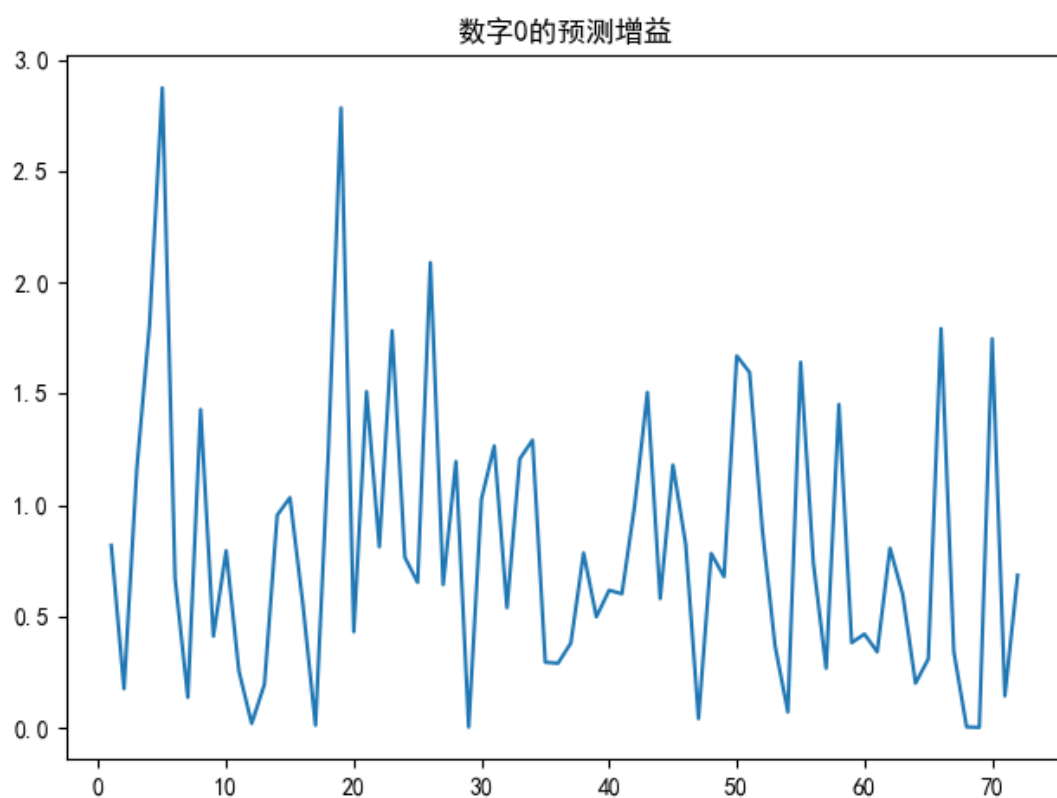
数字 4:

[1020, 1021, 1022, 1023, 1024, 1025, 1026, 1027, 1028, 1029, 1030,
1031, 1032, 1033, 1034, 1035, 1036, 1037, 1038, 1039, 1040, 1041,
1042]

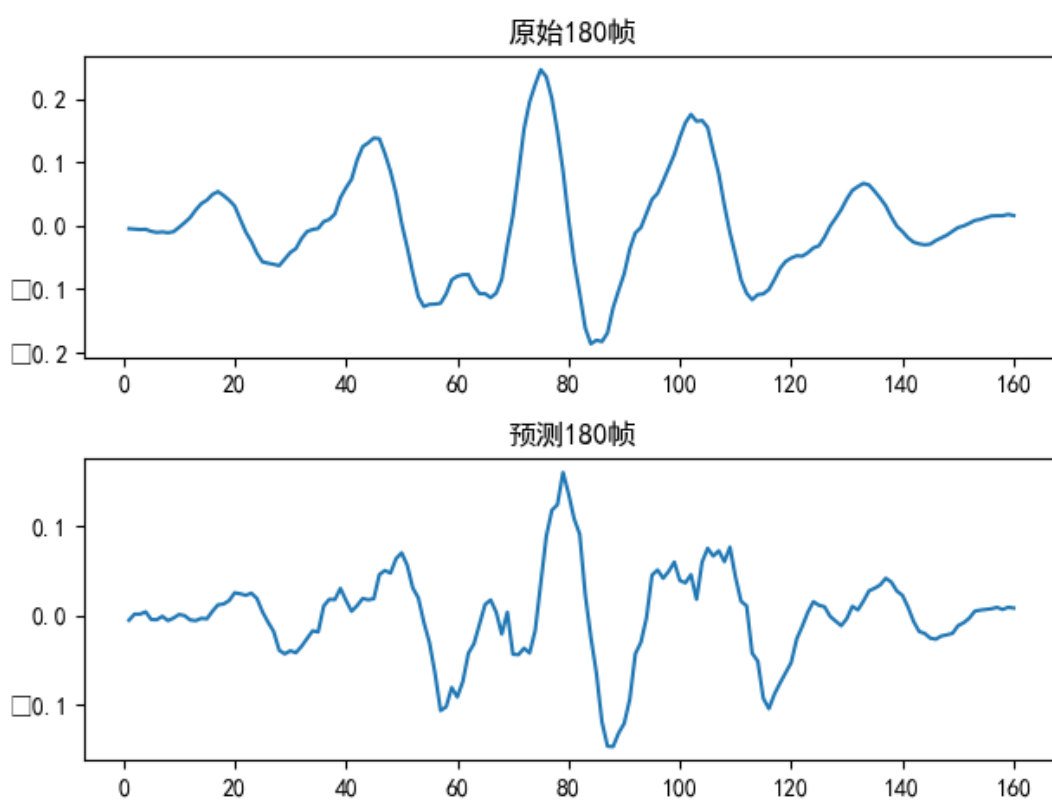
浊音部分对应的基音频率:



数字 0 对应的帧的预测增益:



其中第 180 帧如下图所示：



预测还是比较准确的。

