**Applied Speech and Audio Processing 第二章读书报告**

第二章的内容是一些对声音的基本处理，主要包括录音、回放、读取、打开、加载等操作，以及图形相关、波形生成相关的知识。

**2.1 在matlab上处理声音**

介绍录音回放和相关的函数。

**2.1.1 录制声音**

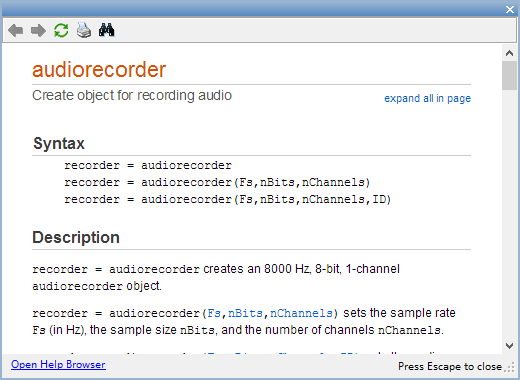
首先介绍的是waverecorder()函数，但由于matlab即将移除了这个函数，所以直接进入了audiorecorder()函数的练习。

>> speech=wavrecord(16000,8000,1,'double');

Error using wavrecord (line 8)

WAVRECORD has been removed. Use AUDIORECORDER instead.

软件对audiorecorder函数的格式要求如下：



照原文，这两个函数中，前者自行指定采样长度，后者则需要通过其他指令来终止录音从而得到一个确定的采样。但在实际应用当中后者应该更为实用，因为录音的过程主观性和随机性很强，强行指定的一段时间里可能在录音时因为突发事件而影响录音质量（如麦克风的线材突然被绊到了需要整理等）。

以上录音用的函数需要指定精度、采样率、通道数。Matlab不在录音操作前指定具体格式，而是在音频确定需要保存时使用用save函数指定格式和保存目录等。

stop()和play()用来停止录音和回放，getaudiodata()将音频转换成向量。

先后输入：

>> aro=audiorecorder(16000,16,1);

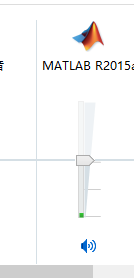
record(aro);

>> stop(aro);

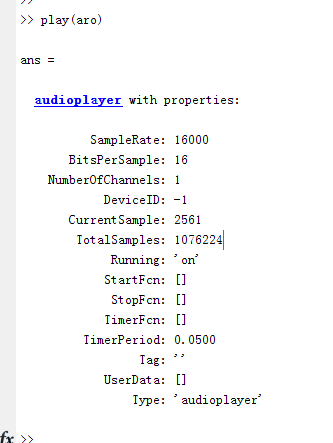
play(aro);

>>

输入play()时在windows混音台显示了matlab的回放信号：



已录制声音的所有参数：



**2.1.2 保存和重放声音**

上一步，getaudiodata()函数（仅限本案例）会将整型16bit数转换成双精度数，然后可用sound()函数播放它们。这两个函数也是用户指定采样率的，可通过设置采样率改变音调。matlab中也可以不指定特定参数，如果没有要求使用默认参数即可：



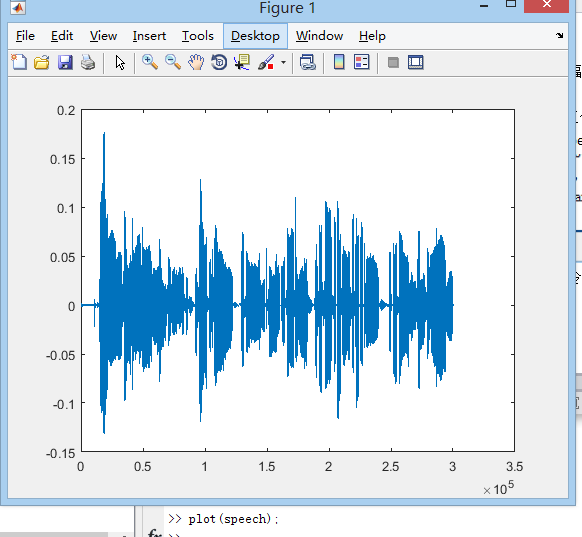
soundsc()函数用于播放时避免削波和音量过小的情况，其结果能明显看到之前偏小的音量放大了：



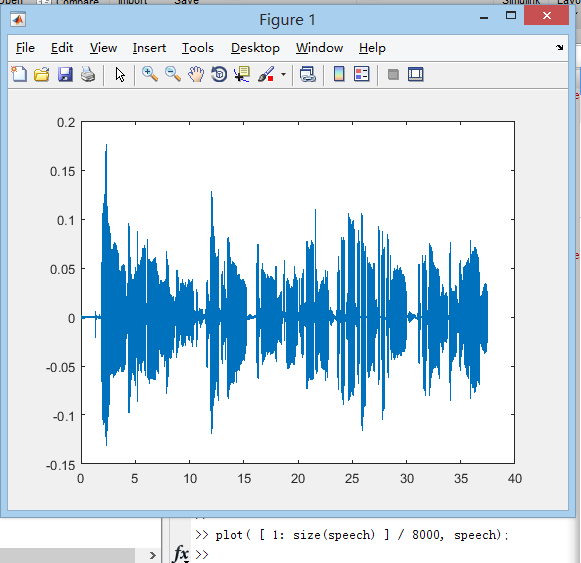
double事实上是指整型的16bit数变成了值处于+1和-1之间的浮点数（**归一化**），由于声音振幅的特性，这些数是有正有负的。浮点数和整型数只是表达形式上不同而已，从系统角度来说，虽然使用的播放函数不同，播放的声音是一样的。

用于绘制图像的函数是：

>>plot(speech);



或者：plot( [ 1: size(speech) ] / 8000, speech);



由图像可知这两张图形状是一样的，但**坐标单位**不一样。

**2.1.3 声音存取**

fopen()是通过文件名打开的文件的指令，fread()则是读取该文件数据。

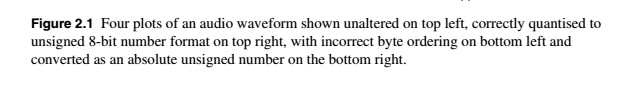
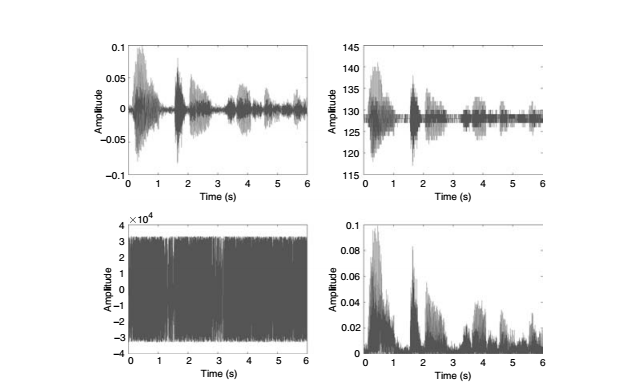
fclose(fid)用于关闭已打开的文件，节约空间。

原文中信息框内是另一段关于音频格式的介绍。上回介绍的是无损格式，本次则是有损的压缩格式。值得注意的是有损格式便于传输，但不便于研究，本书的研究都是无损格式。

save myspeech.mat speech speech2指令用来存储向量；load myspeech.mat 用来读取。

**2.1.4 音频转换**

根据原文，不匹配的精度/位数、符号数和无符号数混用和非原始采样率会导致不同程度的失真、噪音和变调。如下图：



值得注意的是，尽管有失真，它们在试听时还是能表现出很多内容，没有完全损坏（有点像传输失真的电话音）。

**2.2 常规化**

前文提到max(abs(speech)指令是为了将输入音量规范到±1.0之间，如果speech是一个用绝对尺度（最大值为32768的16bit符号数，或原文提到的“线性不动点格式”）表示的向量集，该指令仍然可能导致削波；也就是说在这里只有当speech本身为相对尺度（处于±1.0之间，其中+1.0和-1.0对应双极性码的极大值和极小值）时回放最为真实。

这里原文提到相对尺度（Relative scaling）和绝对尺度（Absolute scaling）。这两个东西在应用上来说，绝对尺度还原度更高，相对尺度绘图更直观。

以上两段是没看懂存疑的。

另外此处还有信息框提到的字节序概念。大字节序（Big endian ）指将高序字节存储在起始地址，相反小字节序（Little endian ）指将低序字节存储在起始地址。

小字节序系统的检测比大字节序更简单。在使用wav格式、8位或A-Law采样宽字节格式时，系统已经处理好了字节序问题，但16位的PCM文件要求用户自己注意即将出现的一些问题。

**2.3 音频处理**

尽管本节标题如此，从目录来看，声音处理的内容实际上是分布在全书的各个章节中的。

在本章节着重讨论了声音处理过程中切片/分割（Segmentation）的问题。一般来说，需要切片的情况不外乎如下几种：

1. 声音连续，但想要试听的效果是实时的；
2. 声音的特质不断改变，或及时的分析很重要（如一个不断音调升高的采样，较短的fft间隔可以分析出它的特点，但完整一次性做频率分析会导致整张图为白噪音）；
3. 一次性处理将导致内存受限（常见）；
4. 想要的是较长时间里的分布规律，而不是一个最终的分析结果；
5. 延迟时间（第一个采样时钟和分析输出的延迟）最小化——这是语音交互系统的常见要求。

很多时域操作（如数字滤波器）与输入声音的长度无关，也与其分割与否无关。书中介绍了两种常用的数字滤波器：FIR (finite impulse response有限脉冲响应)和IIR（Infinite Impulse Response零极点/递归/无限脉冲响应）。

FIR 滤波器的工作模式如下：y=filter(b, 1, x)。

其中b是滤波器施加的对象，x、y之类是与输入输出采样相关的等长度向量，y的计算公式为：y(n) = b(1) × x(n) + b(2) × x(n - 1) + b(3) × x(n - 2) + …+ b(m + 1) × x(n - m).(2.1)

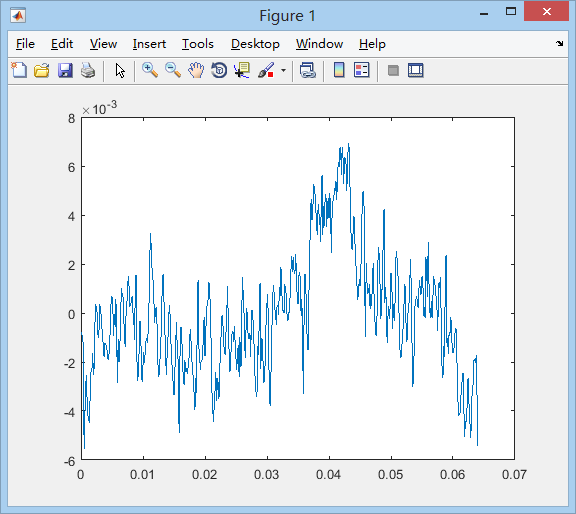
IIR的工作模式如下：y=filter(b, a, x)。

b向量都是滤波器的系数，x、y同上一个公式，y的计算公式：y(n) = b(1) × x(n) + b(2) × x(n - 1) + b(3) × x(n - 2) + …+b(m + 1) × x(n - m)-a(2) × y(n - 1) - a(3) × y(n - 2) … -a(m + 1) × y(n - m). (2.2)

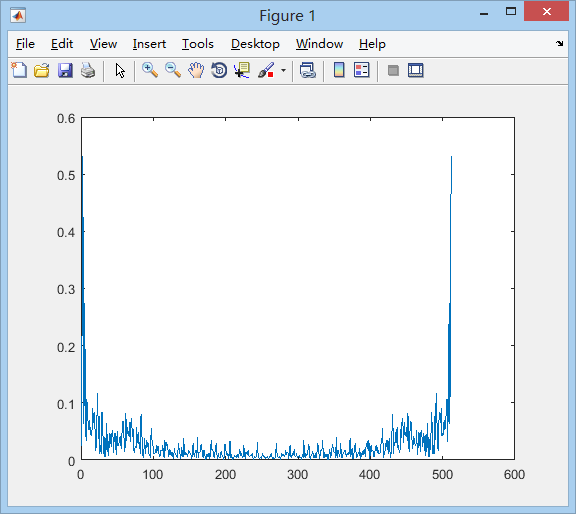
在频域操作方面，首先要求声音被fft转变为频域信号：a\_spec=fft(a\_vector); 若声音向量的长度二的若干次幂，可以尝试截断他们来满足fft的要求：a\_spec=fft(a\_vector, 256);

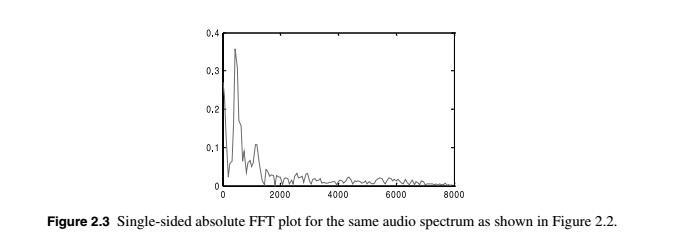
如要考虑更合适的频率分辨率则（此举2.5.2中将详细讨论）：a\_spec=fft(a\_vector(1:256));

以下为一段采样：



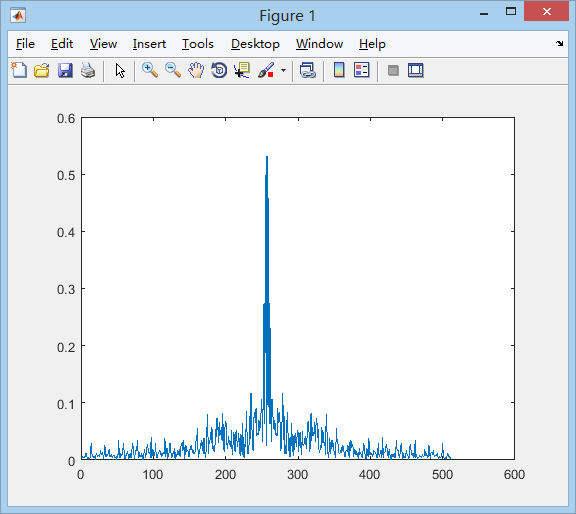
可由上述转换绘制如下频谱：



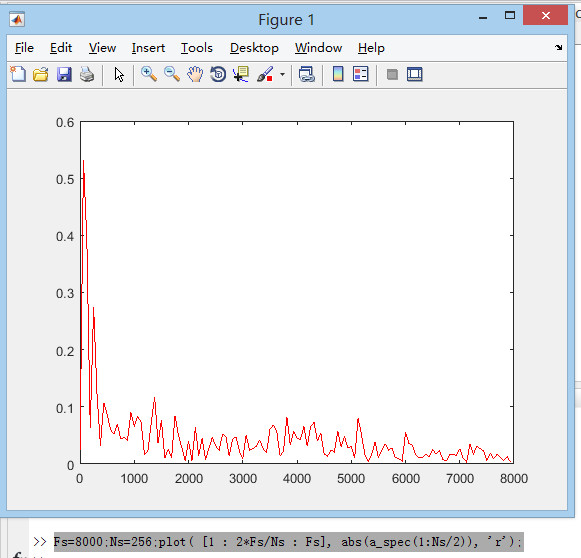


在绘制频谱图过程中还有些问题需要注意——上图频率坐标轴从零开始，正中间是奈奎斯特频率，幅度在频率最远端衰减至0。这种绘制方式把能量集中和不集中的频率都画出来了，显然效率不高。

此时采用以下指令能使低频更靠中间：plot(abs(fftshift(a\_spec)));



指定更详细的参数可以（需指定FS和NS具体数值，“r”表示所绘图像为红色）：plot( [1 : 2\*Fs/Ns : Fs], abs(a\_spec(1:Ns/2)), ’r’);



**2.4分割**

如上节所提到，2.3并没有完全说明声音处理的细节。上节提到实时处理是声音的分割问题，这节有了一些讨论。

首先是声音的“特质”，或者说说一段采样的特征。当分析一段采样时，可能它的特征被分成了两半且分别出现在不同的两帧里，这时窗口上将找不到任何完整的特征。处于采样帧正中央的特征幸免于难，而被拦腰砍断的那些就难免被淹没。考虑到后文将提到的某个问题（2.4.2），分析帧最后的部分将被忽略的更多，那么刚才提到的问题就更严重了。为了解决丢失特征问题，可以交叠采样（to overlap frames）。

**2.4.1 交叠**

原书定义交叠为：“并不直接将声音分割成按顺序的帧，而是每一帧包含一些前一帧的一部分和下一帧的一部分。交叠使得音频的特征以非连续的形式在接连交叠的帧中出现。”

交叠的程度用百分比表示，常见的如25%和50%。

下图为一50%交叠的示意图：顶端图为原波形，下面是分割交叠的帧，分别标记为0-6。帧间的重复部分可以找到原始音频的所有特征。

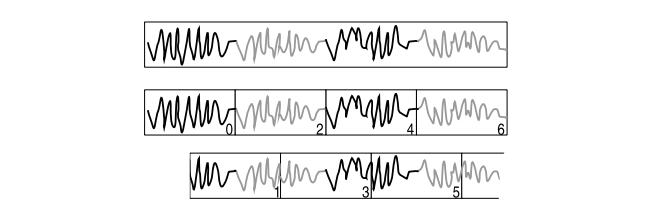


Figure 2.4 Illustration of an original audio recording (the upper waveform) divided into two  
offset sequences of analysis windows (two lower waveforms) with 50% overlapping frames so  
that no short-term auditory feature is obscured by straddling the boundary between analysis  
windows: it will appear unbroken in at least one of the two analysis streams.

这种分割方式有一个地方不好——运算量加倍了。还有一大问题是：分析器（按上下文理解也就是处理算法之类）的工作模式为一进一出。输入时声音将损失一次，输出时再损失一次——比如每一帧进行处理时，如果处理要求是输出音量按平均值缩放（有时称为自动归零），所有帧会一起输出平均音量。那么如何交叠输出的这些帧呢？首先它们不能一般直接相连或相加，不然许多采样会直接放大两倍。

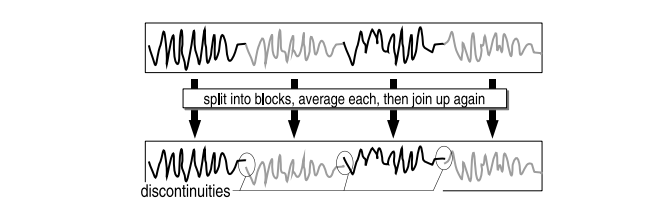


Figure 2.5 Illustration of an original audio recording (upper waveform) being split into equal  
length analysis windows, each of which is normalised, and then rejoined (lower waveform). The  
rejoined audio exhibits discontinuities between frames which would result in significant audio  
distortion.

而根据上图，如果进行无交叠处理，相邻帧可能会连不起来造成“咔哒”声——而且许多常用的处理都会出现这种情况。

**2.4.2 加窗**

许多分割与交叠的问题都可用加窗变通解决，合适的窗函数可以减少交叠可能产生的问题（(see, for example, Chapter 18 of [1]），同时也能有效避免吉布斯现象。

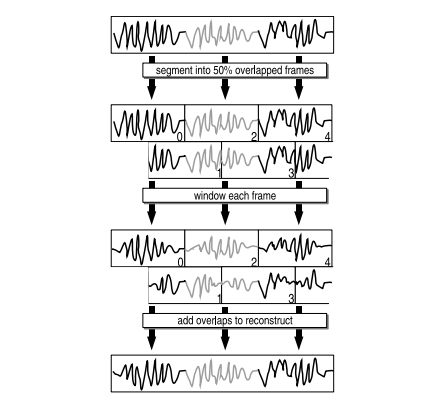
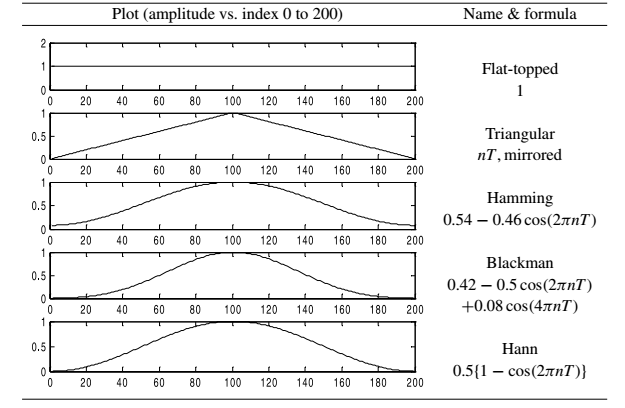


Figure 2.6 Illustration of an original audio recording (upper waveform) being split into 50%  
overlapped analysis frames, each of which are windowed before being summed together to  
reconstruct output audio which does not exhibit discontinuities (lower waveform).

图2.6显示了50%交叠、分割、加窗重建后的结果。

以下为常用的预定义窗类型，他们的特征不同（许多信号处理书籍上有详细介绍）。如有疑问使用汉明窗即可（虽然它两边都不趋近于零）。表2.1给了几个常用窗函数。

Table 2.1. Common window functions.



还有许多未列出的窗函数，以及，每个窗函数可以被各种变形放入一个分析帧，或分成两个并削平顶部——此举在25%交叠中很管用（图2.7）。

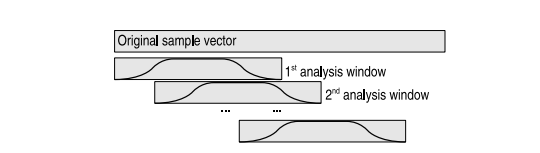


Figure 2.7 Illustration of overlap analysis employing a custom window shape.

由于此前没有接触过窗函数和加窗，特此查阅了一些资料：首先加窗此举是为了减少频谱泄露，泄露产生的原因与数字信号处理过程和特征有关。不同的窗对信号的保留成都产生不同的影响。如果仅要求精确读出主瓣频率，而不考虑幅值精度，则可选用主瓣宽度比较窄而便于分辨的矩形窗，例如测量物体的自振频率等；如果分析窄带信号，且有较强的干扰噪声，则应选用旁瓣幅度小的窗函数，如汉宁窗、三角窗等；对于随时间按指数衰减的函数，可采用指数窗来提高信噪比。

加窗是一个数字信号处理范畴的概念。

加窗这部分不是太明白。

**2.4.3 连续滤波器**

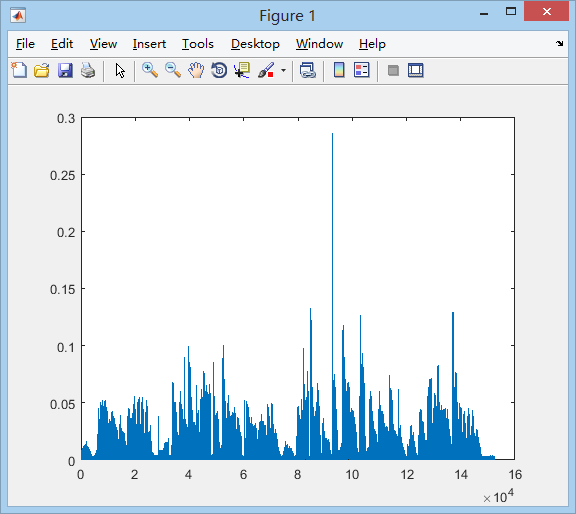
根据经验，相邻帧的分析方式不管相同与否，都可能遇到连续性问题。运行一个案例如下：

1. 首先，录下几秒的声音，采样率8khz，命名为序列s，用soundsc(s)试听它（具体方式见前文），其效果为：正常录音，音量有放大。

2.定义一个数字滤波器h = (1 - 0.9375z-1) （第五章将出现的语音预增强函数）：

h=[1, -0.9375];  
y=filter(h, 1, s);  
soundsc(y);

plot(abs(y));

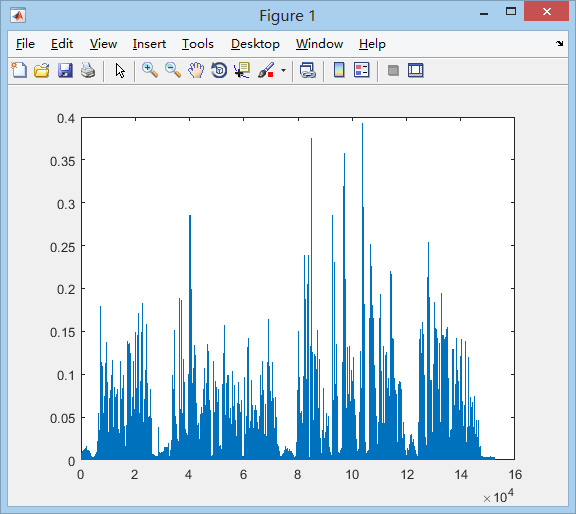


效果：“紧绷”或“鼻音重”，与直接录音相比听起来非常清晰。

3.加窗/切片，每帧单独滤波，再叠加起来。

w=240;  
n=floor(length(s)/w);  
for k=1:n  
 seg=s(1+(k-1)\*w:k\*w);  
 segf=filter(h, 1, seg);  
 outsp(1+(k-1)\*w:k\*w)=segf;  
end  
soundsc(outsp);

plot(abs(outsp));



效果：与前一个相似，但有咔哒声。频谱图上看不太出来二者的区别，但以后用到的光谱图几乎能很清楚地看到。

以上不连续性产生的原因为：在本例中，有递推公式：

y[n] = s[n] - 0.9375 × s[n - 1] (2.3)

在正常整体滤波的情况下程序从头到尾按以上公式运行，没有丝毫问题；而当切片/加窗处理后，在除了第一个采样帧外所有采样帧的的第一个采样点上，公式里运行的s[n - 1]都被程序默认为0，而正确的值为加窗前应有的采样点值。这就是直接加窗的后果。

这还只是一阶滤波而已，如果阶数高达十阶或以上，音质会受到严重损害。

首先为了便于解决问题定义一个概念：历史状态/内部状态。

对一个序列滤波时，历史状态自动在每个采样点间更新；然而在序列初始处滤波时，历史状态一定要声明，否则将默认为零。在切片后，历史状态在每帧的起始处置零，所以输出不连续。

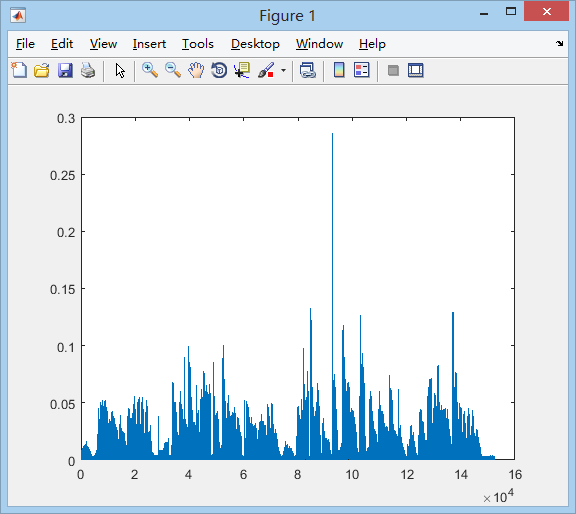
[Matlab](https://cn.bing.com/dict/clientsearch?mkt=zh-CN&setLang=zh&form=BDVEHC&ClientVer=BDDTV3.5.1.4320&q=Matlab%20actually%20provides%20a%20convenient%20way%20to%20set%20and%20store%20internal%20history%20using%20the%20filter()%20command." \t "https://cn.bing.com/dict/_blank)[提供了](https://cn.bing.com/dict/clientsearch?mkt=zh-CN&setLang=zh&form=BDVEHC&ClientVer=BDDTV3.5.1.4320&q=Matlab%20actually%20provides%20a%20convenient%20way%20to%20set%20and%20store%20internal%20history%20using%20the%20filter()%20command." \t "https://cn.bing.com/dict/_blank)[一种](https://cn.bing.com/dict/clientsearch?mkt=zh-CN&setLang=zh&form=BDVEHC&ClientVer=BDDTV3.5.1.4320&q=Matlab%20actually%20provides%20a%20convenient%20way%20to%20set%20and%20store%20internal%20history%20using%20the%20filter()%20command." \t "https://cn.bing.com/dict/_blank)[使用](https://cn.bing.com/dict/clientsearch?mkt=zh-CN&setLang=zh&form=BDVEHC&ClientVer=BDDTV3.5.1.4320&q=Matlab%20actually%20provides%20a%20convenient%20way%20to%20set%20and%20store%20internal%20history%20using%20the%20filter()%20command." \t "https://cn.bing.com/dict/_blank)[filter](https://cn.bing.com/dict/clientsearch?mkt=zh-CN&setLang=zh&form=BDVEHC&ClientVer=BDDTV3.5.1.4320&q=Matlab%20actually%20provides%20a%20convenient%20way%20to%20set%20and%20store%20internal%20history%20using%20the%20filter()%20command." \t "https://cn.bing.com/dict/_blank)[(](https://cn.bing.com/dict/clientsearch?mkt=zh-CN&setLang=zh&form=BDVEHC&ClientVer=BDDTV3.5.1.4320&q=Matlab%20actually%20provides%20a%20convenient%20way%20to%20set%20and%20store%20internal%20history%20using%20the%20filter()%20command." \t "https://cn.bing.com/dict/_blank)[)](https://cn.bing.com/dict/clientsearch?mkt=zh-CN&setLang=zh&form=BDVEHC&ClientVer=BDDTV3.5.1.4320&q=Matlab%20actually%20provides%20a%20convenient%20way%20to%20set%20and%20store%20internal%20history%20using%20the%20filter()%20command." \t "https://cn.bing.com/dict/_blank)[命令](https://cn.bing.com/dict/clientsearch?mkt=zh-CN&setLang=zh&form=BDVEHC&ClientVer=BDDTV3.5.1.4320&q=Matlab%20actually%20provides%20a%20convenient%20way%20to%20set%20and%20store%20internal%20history%20using%20the%20filter()%20command." \t "https://cn.bing.com/dict/_blank)时[设置](https://cn.bing.com/dict/clientsearch?mkt=zh-CN&setLang=zh&form=BDVEHC&ClientVer=BDDTV3.5.1.4320&q=Matlab%20actually%20provides%20a%20convenient%20way%20to%20set%20and%20store%20internal%20history%20using%20the%20filter()%20command." \t "https://cn.bing.com/dict/_blank)[和](https://cn.bing.com/dict/clientsearch?mkt=zh-CN&setLang=zh&form=BDVEHC&ClientVer=BDDTV3.5.1.4320&q=Matlab%20actually%20provides%20a%20convenient%20way%20to%20set%20and%20store%20internal%20history%20using%20the%20filter()%20command." \t "https://cn.bing.com/dict/_blank)[存储](https://cn.bing.com/dict/clientsearch?mkt=zh-CN&setLang=zh&form=BDVEHC&ClientVer=BDDTV3.5.1.4320&q=Matlab%20actually%20provides%20a%20convenient%20way%20to%20set%20and%20store%20internal%20history%20using%20the%20filter()%20command." \t "https://cn.bing.com/dict/_blank)的[内部](https://cn.bing.com/dict/clientsearch?mkt=zh-CN&setLang=zh&form=BDVEHC&ClientVer=BDDTV3.5.1.4320&q=Matlab%20actually%20provides%20a%20convenient%20way%20to%20set%20and%20store%20internal%20history%20using%20the%20filter()%20command." \t "https://cn.bing.com/dict/_blank)/[历史记录](https://cn.bing.com/dict/clientsearch?mkt=zh-CN&setLang=zh&form=BDVEHC&ClientVer=BDDTV3.5.1.4320&q=Matlab%20actually%20provides%20a%20convenient%20way%20to%20set%20and%20store%20internal%20history%20using%20the%20filter()%20command." \t "https://cn.bing.com/dict/_blank)的[方法](https://cn.bing.com/dict/clientsearch?mkt=zh-CN&setLang=zh&form=BDVEHC&ClientVer=BDDTV3.5.1.4320&q=Matlab%20actually%20provides%20a%20convenient%20way%20to%20set%20and%20store%20internal%20history%20using%20the%20filter()%20command." \t "https://cn.bing.com/dict/_blank)：

w=240;  
hst=[];%历史状态序列

n=floor(length(s)/w);

for k=1:n  
 seg=s(1+(k-1)\*w:k\*w);  
 [segf, hst]=filter(h, 1, seg, hst);  
outsp2(1+(k-1)\*w:k\*w)=segf;  
end  
soundsc(outsp2);

plot(abs(outsp2));



以上代码在filter() 指令中加入了历史状态序列，每次运行滤波器都存储了内部历史状态——通过在每帧的矢量输入时声明历史状态，在整个处理过程里将它们置位以确保帧之间的平滑滤波。此代码产生的输出音频**避免了明显的咔哒声和不连续**。

**2.5 分析窗大小的决策**

由于matlab基于矩阵运算的性质，其在处理大量数据时相比其他语言更有优势，工程师倾向于用它进行数据量大的运算。但在诸如电话音等数据量很多的系统中，延迟却是很大的问题，它将限制分析窗的大小。

另外，声音特质在分析窗内部变化的时候，FFT的切片过大时将隐藏一些声音细节。

由以上两个因素，接下来将讨论：信号稳定性和时频域分辨率。

**2.5.1 信号稳定性**

举例：一般认为乐器演奏的一个长音符是稳定的，而当转音到另一个音符时，声音变化得很明显（频率、音量、音调、音色等）。

通常只有在分析窗的小于等于音符转变的部分时它才能被正确分析出来——每个窗上分别运行FFT，然后找到频率峰值点；但如果一个窗大到跨越了两个音符，FFT的分析就会出问题。所以实际应用中窗足够小程序才能正常运行。

一般在语音分析中，说话者的肌肉运动会导致音色缓慢变化。通常认为20–30 ms 内语音是近似稳定的，每帧小于20ms这个长度就可以。

以上过程就是所谓稳定性分析，它也常见于线性预测（5.2.1）等情况，使用时必须仔细分析要处理的音频信号的特征。

**2.5.2 时频分辨率**

FFT运行过程中，对采样频率Fs赫兹的音频做N点FFT，得到包含N/2+1个正值的频率点。每个点从原始信号中的一个小范围的频率中计算频率能量，量化的频率点间距离与采样率和分析采样的数量相关（Fs/N ），意即其宽度等于分析窗口所涵盖的时间跨度的倒数。为了实现更高的频率分辨率，我们需要分析采样率的时间更长。

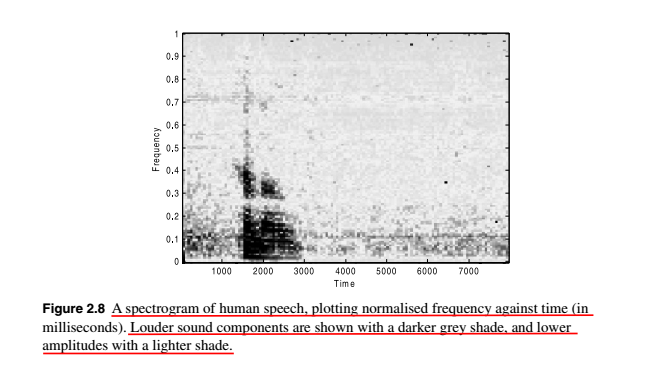
个人理解：一个窗/帧内的采样点越少，越难获得完整的较低频信号周期，那么对应的最低可分辨的频率就更大，分辨率也就更低。然而，根据前文所述对于快速变化的信号，更多采样意味着错过一些时域特征。

一个FFT，要么获得更高的频率分辨率（样本多），要么得到更高的时间分辨率(样本少)，需要工程师多做权衡。

**2.6 可视化**

波形特性的观察主要有以下方式：

1. 波形绘图（时域幅度变化）：最容易和基础的波形可视化方式，较长录音的最方便快捷的浏览方式。但它无法表达声音全部的信息；
2. 频谱图：一个工具，不恰当使用容易观丢失音频特征。是整段长音频的可视化工具；
3. 短时傅里叶变换（Short-time Fourier transform (STFT)，实为光谱图）：滑动窗窄边傅里叶变换，在很长采样间依次重复工作的时-频分解或分析。一个函数可以先乘上仅在一段时间不为零的[窗函数](http://baike.sogou.com/lemma/ShowInnerLink.htm?lemmaId=8906031" \t "http://baike.sogou.com/_blank)(window function)再进行一维的傅利叶转换。再将这个窗函数沿著[时间轴](http://baike.sogou.com/lemma/ShowInnerLink.htm?lemmaId=592270" \t "http://baike.sogou.com/_blank)[挪移](http://baike.sogou.com/lemma/ShowInnerLink.htm?lemmaId=7552323" \t "http://baike.sogou.com/_blank)，所得到一系列的傅利叶转换结果排开则成为结果。



FFT绘制的图像为频谱图，STFT则为光谱图。前者为二位图像，后者是三维，它的x、y轴分别为时间和频率，第三维是特定时间频率点（段）的幅度（能量），这一维通过色度（2.8图中为灰度）表示。

**2.6.1 关于坐标轴**

如前文所说，光谱图其实是一个三维图像。

而频谱中，纵轴也不是振幅，而是振幅的绝对值。有时使用20 × log10的功率谱 ：semilogy(spectrum);

横轴频率是线性的，纵轴则是fft输出的指数分布。为了更方便视图，不使用hz的频率单位，而是缩放到0-1（直流到奈奎斯特频率）或0-π的角频率（角频率也叫自然频率）。

res=pi/size(spectrum);  
semilogy(res:res:pi, spectrum);

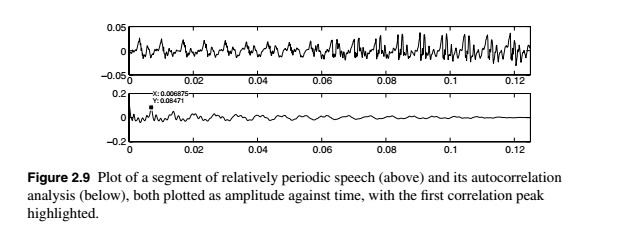
**2.6.2 其它可视化手段**

**2.6.2.1 相关函数**

这是描述一个信号的自相关程度的量，工作方式是比较一个信号在不同时刻的相似程度。其公式如下：

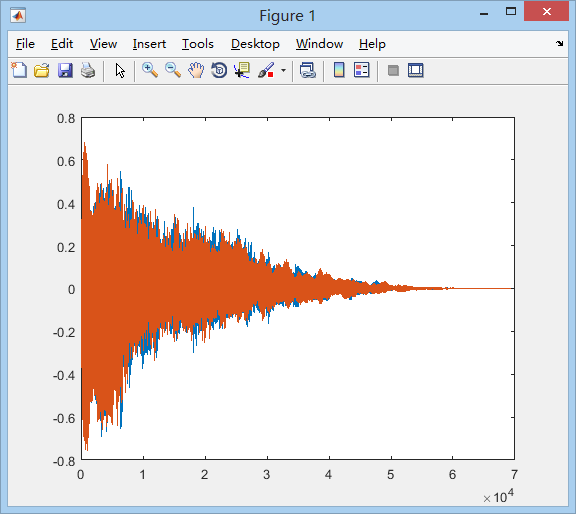
代码：[c,lags]=xcorr(x,y);

自相关是指信号在一个时刻的瞬时值与另一个时刻的瞬时值之间的依赖关系，是对一个随机信号的时域描述。如下图，下半图所对应函数的最高值点即自相关函数求得的延迟时间。



只有一个参数输入xcorr时，它将对这个量做自相关分析。可观察其周期。

自相关只能用来判定相关程度最大的点，也就是周期性，而设有一段短音频speech，有一段声音在绘图中看起来很规则。假设它处于第9000和10000个采样之间，先将它提取出来：

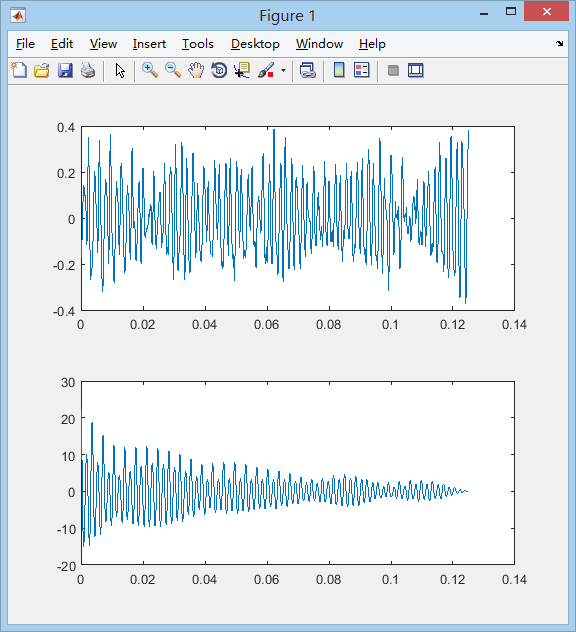


segment=speech(9000:10000);

接着做自相关分析，绘制输入输出图像：

[c,lags]=xcorr(segment);  
subplot(2,1,1);  
plot([0:1:1000]/8000,segment);  
axis([0, 0.125. -0.05, 0.05]);

subplot(2,1,2);  
plot(lags(1001:2001)/8000,c(1001:2001));  
axis([0, 0.125. -0.2, 0.2]);



Axis指定y的范围（-0.05到0.05等）由输入的采样幅值和成分决定，正确使用时图像会铺满所有绘图区域。见图2.9。第一个峰出现在x值为0.006875即6.875ms位置。测量两峰值的距离后与第一个自相关峰值出现的位置，发现了其一大用途：测量周期。

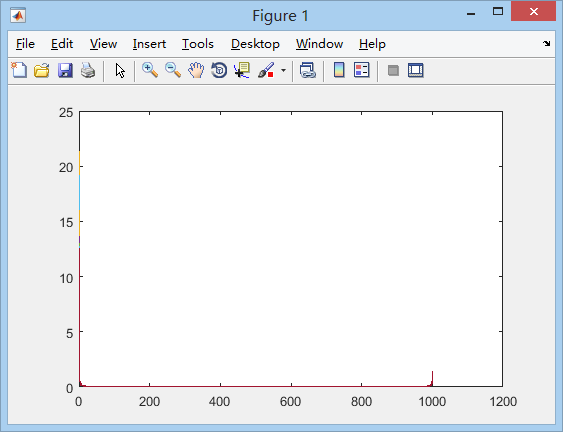
衡量相关程度的具体指标则可以用相关系数。

**2.6.2.2 倒谱**

倒谱（cepstrum)一种信号的傅里叶变换谱经对数运算后再进行的傅里叶反变换。由于一般傅里叶谱是复数谱，因而又称复倒谱。

运行程序：

ps=log(abs(fft(hamming(length(segment)).\*segment)));  
plot(abs(ifft( ps )));



倒谱常常采用非线性频率波段的映射，由此产生的Mel频率倒谱系数(MFCC)用于说话人识别，语音识别和其他几个语音分析任务。6.1.5处将用到这个分析。

**2.7 声音生成**

**2.7.1 纯音**

保存一个函数如下：

function [s]=tonegen(Ft, Fs, Td)  
s=sin([1:Fs\*Td]\*2\*pi\*Ft/Fs);

然后在指令窗口运行：note=tonegen(440, 16000, 2);soundsc(note, 16000); 可以生成固定音高正弦波，似乎没有什么特别的。

2.7.2 白噪

noise=rand(1,Fs\*Td);

以上代码生成的白噪音是最简单的一种，所有值都是随机数。

2.7.3 可变调音高

函数：

function [snd]=freqgen(frc, Fs)  
th=0;  
fr=frc\*2\*pi/Fs;  
for si=1:length(fr)  
th=th+fr(si);  
snd(si)=sin(th);  
th=unwrap(th);  
end

以下运用了函数的代码可以用来生成可变的音调，试听效果是连续两次下降的一个音：

freq=[440\*(1+zeros(1,1000)), 415.2\*(1+zeros(1,1000)),392\*(1+zeros(1,1000))];  
music=freqgen(freq, 8000);  
soundsc(music, 8000);

**2.7.4 混合声音**

使用几个tongen函数后简单相加即可：

C=tonegen(261.63, 8000, 2);

E=tonegen(329.63, 8000, 2);

G=tonegen(783.99, 8000, 2);

B=tonegen(987.77, 8000, 2);

soundsc(C+E+G+B, 8000);

一个和弦。

**总结**

本章主要是基础的一些函数用法，用以熟悉matlab环境和语音处理相关背景。相关函数的内容在开始读的时候没学过，读完的时候已经学完了。窗函数部分始终不是很明白。