使用 MATLAB 处理声音的基本操作

你可以在这里下载 文件。这个一个 400Hz 的基础频率加上噪音构成的复杂音频。

```
>> [snd, sampFreq, nBits] = wavread('440_sine.wav');
```

这个音频文件还有 2 个声道，5060 个采样点

```
>> size(snd)

ans =

        5060      2
```

素材音频(采样频率为 sampFreq = 44110)对应的时长大约是 114ms（Zt：应该只是为了下面绘制图象确定范围使用的）

```
>> 5060 / sampFreq

ans =

   0.1147
```

用下面的方法可以播放这个音频

```
>> sound(snd, 44100, 16)
```

我们只处理音频中的一个声道

```
>> s1 = snd(:,1);
```

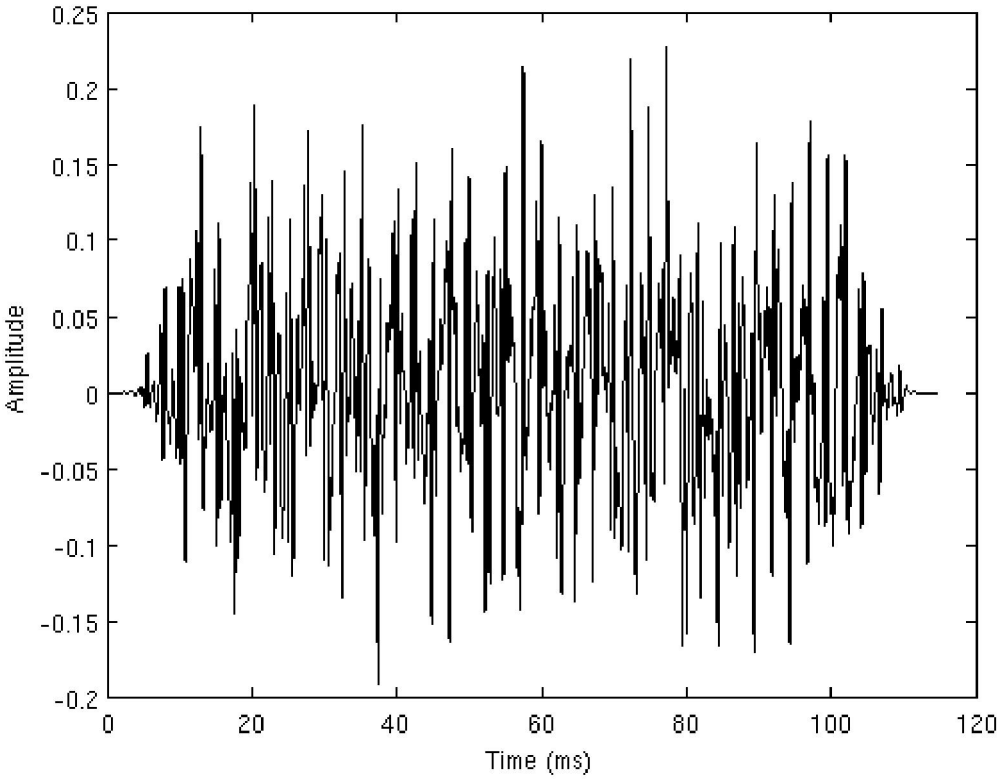
绘制音频

一种表示声音的方法是以声压为纵轴，时间为横轴。首先，我们需要建立一个包含时间点的数组：

```
>> timeArray = (0:5060-1) / sampFreq;
>> timeArray = timeArray * 1000; %放大到毫秒级
```

之后，即可绘制音频：

```
plot(timeArray, s1, 'k')
```



绘制频率信息

另外一种表示音频的方法是绘制其中的频率信息。我们可以用过 FFT 函数取得音频中的频率信息，FFT 是“快速傅立叶变换”的缩写。我们通过下面文档 http://www.mathworks.com/support/tech-notes/1700/1702.html 介绍的技术来获得声音中的功率谱（横坐标是频率，纵坐标是功率）。

```
 n = length(s1);
 p = fft(s1); % 计算傅立叶变换
```

注意：和上面的那份技术文档中提到的不同，我们没有指定参加 FFT 的点的数量，默认下 FFT 函数会使用信号全部采样值（上面表达式中的 n）。上面的 n 并不是 2 的幂次，在计算上会稍微慢一些，但因为我们数值总量不多，这样的影响完全可以忽略不计。

```
  nUniquePts = ceil((n+1)/2);
  p = p(1:nUniquePts); % 选择前半部，因为后半部是前半部的一个镜像
  p = abs(p); % 取绝对值，或者称之为幅度
```

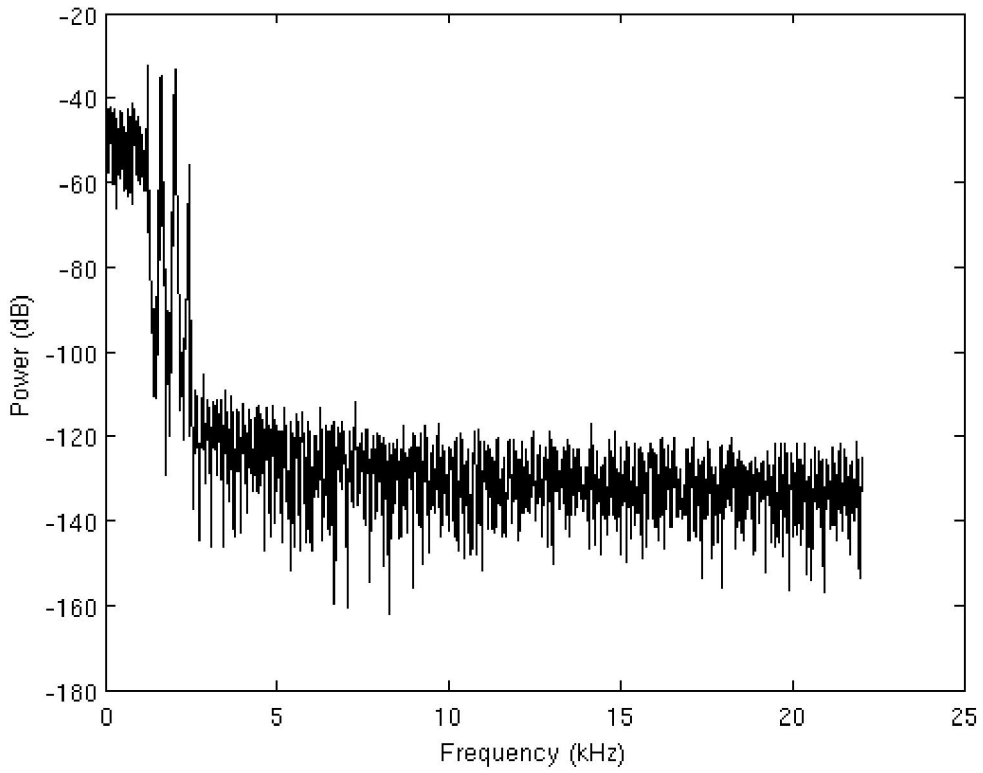
FFT 函数处理音频返回值包括幅度和相位信息，是以复数的形式给出的（返回复数）。对傅立叶变换后的结果取绝对值后，我们就可以取得频率分量的幅度信息。

```
  p = p/n; % 使用点数按比例缩放，这样幅度和信号长度或者它自身
           % 的频率无关
  p = p.^2;  % 平方得到功率
```

```
% 乘以 2（原因请参考上面的文档）
  if rem(n, 2) % 奇数，nfft 需要排除奈奎斯特点
    p(2:end) = p(2:end)*2;
  else
    p(2:end -1) = p(2:end -1)*2;
  end

freqArray = (0:nUniquePts-1) * (sampFreq / n); % 创建频率数组
plot(freqArray/1000, 10*log10(p), 'k')
xlabel('Frequency (kHz)')
ylabel('Power (dB)')
```

运行结果如下，需要注意的是我们在绘制功率时使用 10*log10(p)将其单位换算为分贝，我们也将频率除以 1000 单位换算为 KHz.



为了验证计算结果是信号的能量。我们可以计算信号的均方根。宽泛的说，rms 可以看作是波形的幅度值的测量（Zt:面积吧？）。如果你只是简单的取正弦信号的平均值，结果将会是 0，原因是正数部分会和负数部分相互抵消。为了避免这样的情况，在求平均值之前先求平方然后再开方（单纯的平方会放大一些极值）：

```
>> rms_val = sqrt(mean(s1.^2))
rms_val =
```

0.0615

因为 rms 等于全部信号求平方后再开方的值，将 fft 后的每个频率的功率相加结果应该相同。

```
>> sqrt(sum(p))

ans =

0.0615
```

References

- Hartman, W. M. (1997), Signals, Sound, and Sensation. New York: AIP Press
- Plack, C.J. (2005). The Sense of Hearing. New Jersey: Lawrence Erlbaum Associates.
- The MathWorks support. Technical notes 1702, available: http://www.mathworks.com/support/tech-notes/1700/1702.html

Basic Sound Processing with MATLAB

原文如下：
来自：http://xoomer.virgilio.it/sam_psy/psych/sound_proc/sound_proc_matlab.html

This page describes some basic sound processing functions in MATLAB. We'll begin by reading in a wav file. You can download it here 440_sine.wav it contains a complex tone with a 440 Hz fundamental frequency (F0) plus noise.

```
>> [snd, sampFreq, nBits] = wavread('440_sine.wav');
```

the wav file has two channels and 5060 sample points

```
>> size(snd)

ans =

        5060      2
```

considering the sample rate (sampFreq = 44110) this corresponds to about 114 ms duration

```
>> 5060 / sampFreq
```

```
  ans =

    0.1147
```

we can listen to the tone with the sound function

```
>> sound(snd, 44100, 16)
```

we'll select and work only with one of the channels from now onwards

```
>> s1 = snd(:,1);
```

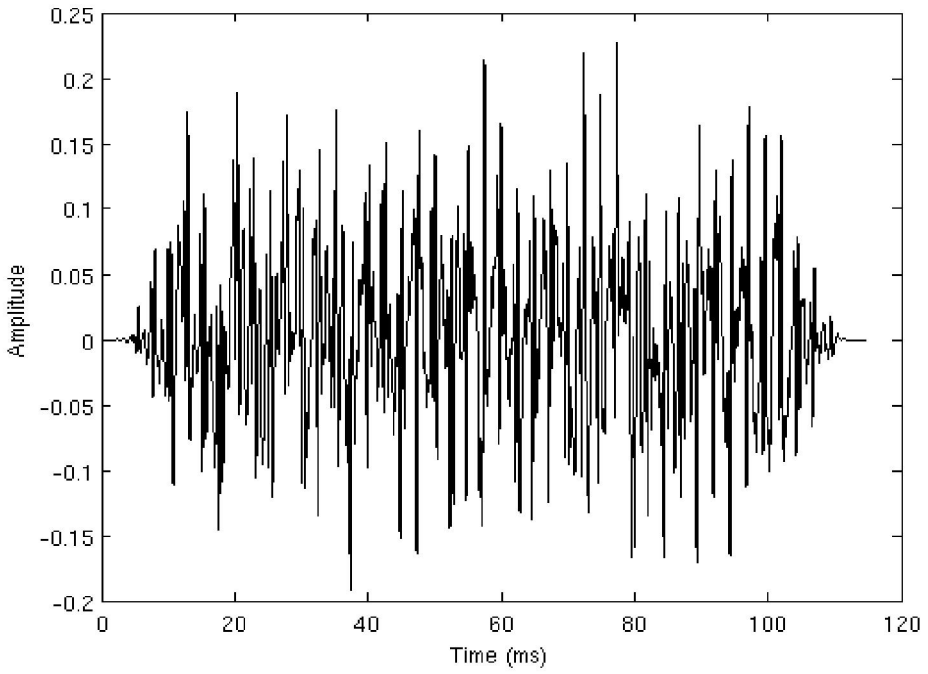
Plotting the Tone

A time representation of the sound can be obtained by plotting the pressure values against the time axis. However we need to create an array containing the time points first:

```
>> timeArray = (0:5060-1) / sampFreq;
>> timeArray = timeArray * 1000; %scale to milliseconds
```

now we can plot the tone

```
plot(timeArray, s1, 'k')
```



Plotting the Frequency Content

Another useful graphical representation is that of the frequency content of the tone. We can obtain the frequency content of the sound using the fft function, that implements a Fast Fourier Transform algorithm. We'll follow closely the following technical document http://www.mathworks.com/support/tech-notes/1700/1702.html to obtain the power spectrum of our sound.

```
 n = length(s1);
 p = fft(s1); % take the fourier transform
```

notice that compared to the technical document, we didn't specify the number of points on which to take the fft, by default then the fft is computed on the number of points of the signal (n). Since we're not using a power of two the computation will be a bit slower, but for signals of this duration this is negligible.

```
  nUniquePts = ceil((n+1)/2);
  p = p(1:nUniquePts); % select just the first half since the second half
                       % is a mirror image of the first
  p = abs(p); % take the absolute value, or the magnitude
```

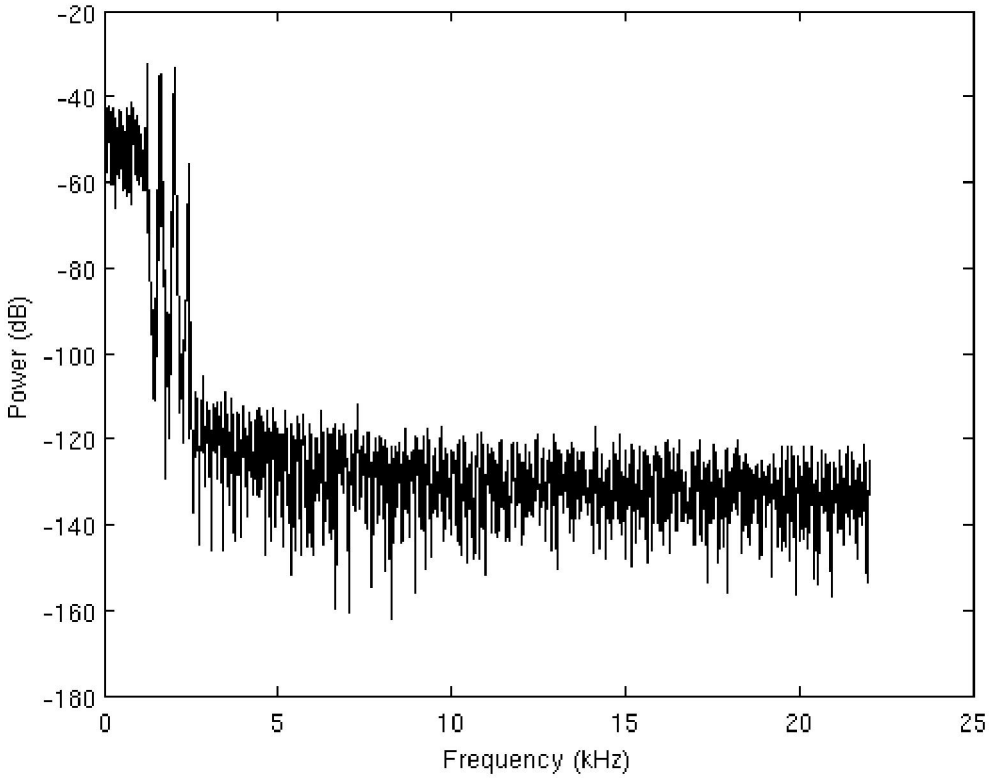
the fourier transform of the tone returned by the fft function contains both magnitude and phase information and is given in a complex representation (i.e. returns complex numbers). By taking the absolute value of the fourier transform we get the information about the magnitude of the frequency components.

```
  p = p/n; % scale by the number of points so that
           % the magnitude does not depend on the length
           % of the signal or on its sampling frequency
  p = p.^2;  % square it to get the power

 % multiply by two (see technical document for details)
  if rem(n, 2) % odd nfft excludes Nyquist point
    p(2:end) = p(2:end)*2;
  else
    p(2:end -1) = p(2:end -1)*2;
  end

freqArray = (0:nUniquePts-1) * (sampFreq / n); % create the frequency
array
plot(freqArray/1000, 10*log10(p), 'k')
xlabel('Frequency (kHz)')
ylabel('Power (dB)')
```

The resulting plot can bee seen below, notice that we're plotting the power in decibels by taking 10*log10(p), we're also scaling the frequency array to kilohertz by dividing it by 1000



To confirm that the value we have computed is indeed the power of the signal, we'll also compute the root mean square (rms) of the signal. Loosely speaking the rms can be seen as a measure of the amplitude of a waveform. If you just took the average amplitude of a sinusoidal signal oscillating around zero, it would be zero since the negative parts would cancel out the positive parts. To get around this problem you can square the amplitude values before averaging, and then take the square root (notice that squaring also gives more weight to the extreme amplitude values):

```
>> rms_val = sqrt(mean(s1.^2))
rms_val =

    0.0615
```

since the rms is equal to the square root of the overall power of the signal, summing the power values calculated previously with the fft over all frequencies and taking the square root of this sum should give a very similar value

```
>> sqrt(sum(p))

ans =
```

0.0615

References

- Hartman, W. M. (1997), Signals, Sound, and Sensation. New York: AIP Press
- Plack, C.J. (2005). The Sense of Hearing. New Jersey: Lawrence Erlbaum Associates.
- The MathWorks support. Technical notes 1702, available: http://www.mathworks.com/support/tech-notes/1700/1702.html

2010-3-7

Zoologist@www.lab-z.com