仿真试验数据采集处理系统

方案设计报告

滤波处理

V0.7 2012/12/27

系统提供滤波器进行滤波处理。

可以实现六种滤波分析方式的设计、引用、管理。包括：

FIR 有限冲击响应滤波分析：矩形窗，汉宁窗，海明窗；

IIR 无限冲击响应滤波分析：贝塞尔，巴特沃斯，切比雪夫(1)。

提供基于时域分析、频域分析的多种算法。包含：

FFT、频谱、互功率谱、自功率谱。

一. FIR 有限冲击响应滤波分析：矩形窗,汉宁窗,海明窗

1.有限脉冲响应滤波器定义：

有限脉冲响应滤波器是数字滤波器的一种，简称 FIR 数字滤波器（finite impulse response filter)。这类滤波器对于脉冲输入信号的响应最终趋向于 0，因此是有限的，而得名。它是相对于无限脉冲响应滤波器(IIR)而言。由于无限脉冲响应滤波器中存在反馈回路，因此对于脉冲输入信号的响应是无限延续的。

有限脉冲响应滤波器是一线性系统，输入信号， $x(0),x(1),...,x(n)$，经过该系统后的输出信号， $y(n)$ 可表示为： $y(n)=h_0x(n)+h_1x(n-1)+...+h_Nx(n-N)$

其中，$h_0,h_1,...,h_N$ 是滤波器的脉冲响应，通常称为滤波器的系数。N 是滤波器的阶数。

上式也可表示为： $y(n)=\sum_{k=0}^{N}h_kx(n-k).$

如果输入信号为脉冲信号 $\delta(n)$， $\delta(n)=\begin{cases}1 & \text{if } n=0\\0 & \text{if } n\neq 0\end{cases}$

输出信号则为： $y(n)=\sum_{k=0}^{N}h_k\delta(n-k)=h_n.$

这也是脉冲响应 h_n 得名的原因，即，它是滤波器脉冲输入的响应。 有限脉冲响应滤波器的传递函数可由其脉冲响应的 z 变换获得：

$$H(z)=Z\{h(n)\}=\sum_{n=-\infty}^{\infty}h(n)z^{-n}=\sum_{n=0}^{N}h(n)z^{-n}.$$

因此，有限脉冲响应滤波器的频率响应为：

$$H(e^{j\omega})=\sum_{n=0}^{N}h(n)e^{-j\omega n}.$$

2.FIR 滤波器的窗函数设计原理：

窗函数法的设计思想是按照所要求的理想滤波器频率响应 $H_d(e^{j\omega})$，设计一个 FIR 滤波器，使之频率响应 $H(e^{j\omega})=\sum_{n=0}^{N}h(n)e^{-j\omega n}$ 来逼近 $H_d(e^{j\omega})$。先由 $H_d(e^{j\omega})$ 的傅里叶反变换导出理想滤波器的冲激响应序列 $H_d(n)$，即：$H_d(n)=\frac{1}{2\pi}\int_{-\pi}^{\pi}H_d(e^{jwn})e^{jwn}dw$

由于 $H_d(e^{jwn})$ 是矩形频率特性，所以 $H_d(n)$ 是一无限长的序列，且是非因果的，而要计的 FIR 滤波器的冲激响应序列是有限长的，所以要用有限长的序列 $H(n)$ 来逼近无限长的序列 $H_d(n)$，最有效的方法是截断，或者说用一个有限长度的窗口函数 $W(n)$ 序列来截取 $H_d(n)$，即：$H(n)=W(n)H_d(n)$

按照复卷积公式，在时域中的乘积关系可表示成在频域中的周期性卷积关系，即可得所设计的 FIR 滤波器的频率响应：$H(e^{jw})=\frac{1}{2\pi}\int_{-\pi}^{\pi}H_d(e^{j\theta})W(e^{j(w-\theta)})d\theta$

其中，$W(e^{jw})$ 为截断窗函数的频率特性。由此可见，实际的 FIR 数字滤波器的频率响应 $H(e^{jw})$ 逼近理想滤波器频率响应 $H_d(e^{jw})$ 的好坏，完全取决于窗函数的频率特性 $W(e^{jw})$。

如果 $W(n)$ 具有下列形式：$w(n)=\begin{cases}0, & n<0,n>=N\\ 1, & 0<=n<N\end{cases}$，$W(n)$ 相当于一个矩形，我们称之为矩形窗。即我们可采用矩形窗函数 $W(n)$ 将无限脉冲响应 $H_d(n)$ 截取一段 $H(n)$ 来近似为 $H_d(n)$。

上边只考虑了矩形窗，如果我们使窗的主瓣宽度尽可能地窄，旁瓣尽可能地小，可以获得性能更好的滤波器，通过改变窗的形状来达到这个目的。在数字信号处理的发展过程中形成了不同于矩形窗的很多窗函数，这些窗函数在主瓣和旁瓣特性方面各有特点，可满足不同的要求。为此，用窗函数法设计 FIR 数字滤波器时，要根据给定的滤波器性能指标选择窗口宽度 N 和窗函数 $W(n)$。下面具体介绍几类类窗函数及其特性。

3.矩形窗：

矩形窗函数的时域特征形式可以表示为：

$$w(n)=R_N(n)=\begin{cases}1, & 0<=n<=N-1\\ 0, & \text{其他}\end{cases}$$

它的频域特征为：

$$W(e^{jw})=e^{-j(\frac{N-1}{2})w}\frac{\sin(\frac{wN}{2})}{\sin(\frac{w}{2})}$$

4.汉宁窗：

汉宁窗函数的时域特征可以表示为：

$$W(k)=0.5(1-\cos(2\pi\frac{k}{n+1})),k=1,2,...,N$$

它的频域特性为：

$$W(w)=\{0.5W_R(w)+0.25[W_R(w-\frac{2\pi}{N-1})+W_R(w+\frac{2\pi}{N-1})]\}e^{-jw(\frac{N-1}{2})}$$

其中，$W_R(w)$ 为矩形窗函数的幅度频率特性函数。

汉宁窗函数的最大旁瓣值比主瓣值低 31dB，但是主瓣宽度比矩形窗函数的主瓣宽度增加了 1 倍，为 $\frac{8\pi}{N}$ 。

5.海明窗：

海明窗函数的时域形式可以表示为：

$$W(k)=0.54-0.46\cos(2\pi\frac{k}{N-1})),k=1,2,...,N$$

它的频域特性为：

$$W(w)=0.54W_R(w)+0.23[W_R(w-\frac{2\pi}{N-1})+W_R(w+\frac{2\pi}{N-1})]$$

其中，$W_R(w)$ 为矩形窗函数的幅度频率特性函数。

海明窗函数的最大旁瓣值比主瓣值低 41dB，但是它和汉宁窗函数的主瓣宽度是一样大的。

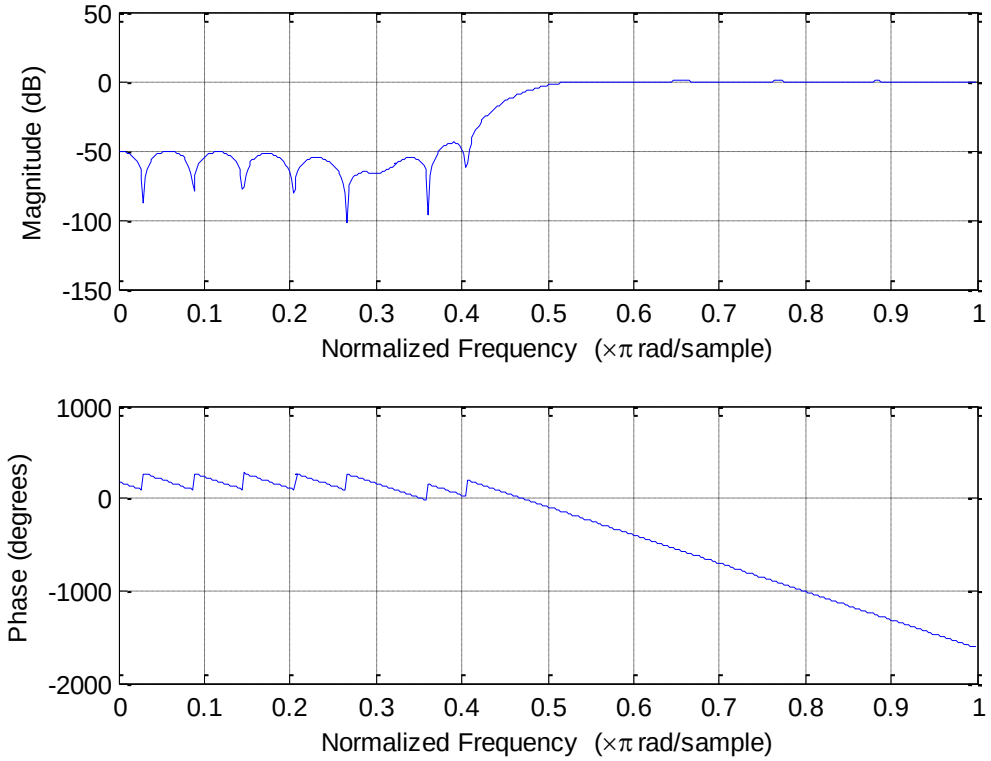
6.Matlab 函数：

FIR 有限冲击响应滤波分析：fir1

代码示例：

```
load chirp % Load y and fs.
b = fir1(34,0.48,'high',chebwin(35,30));
freqz(b,1,512)
```

代码输出：

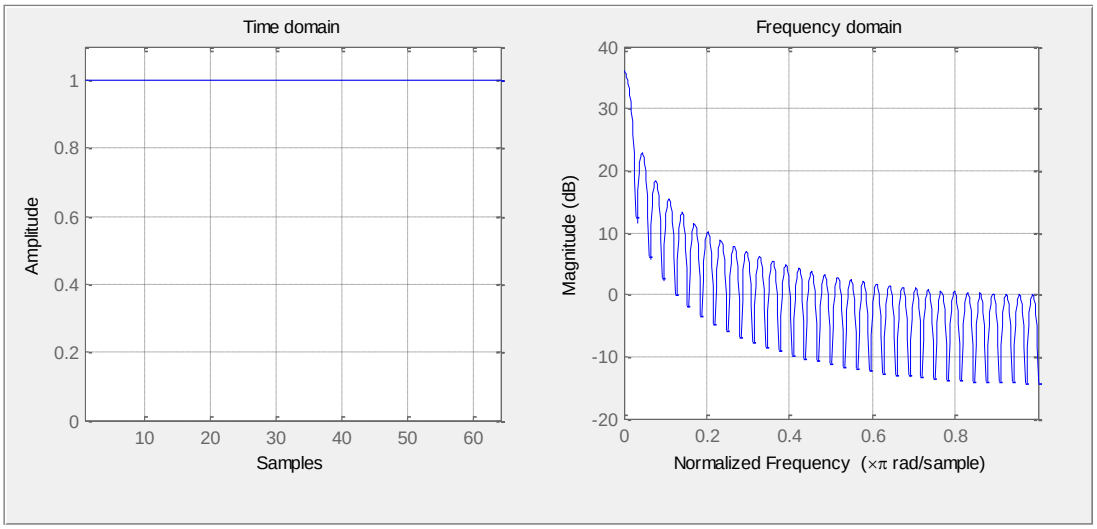


矩形窗：rectwin

代码示例：

```
L=64;
wvtool(rectwin(L))
```

代码输出：

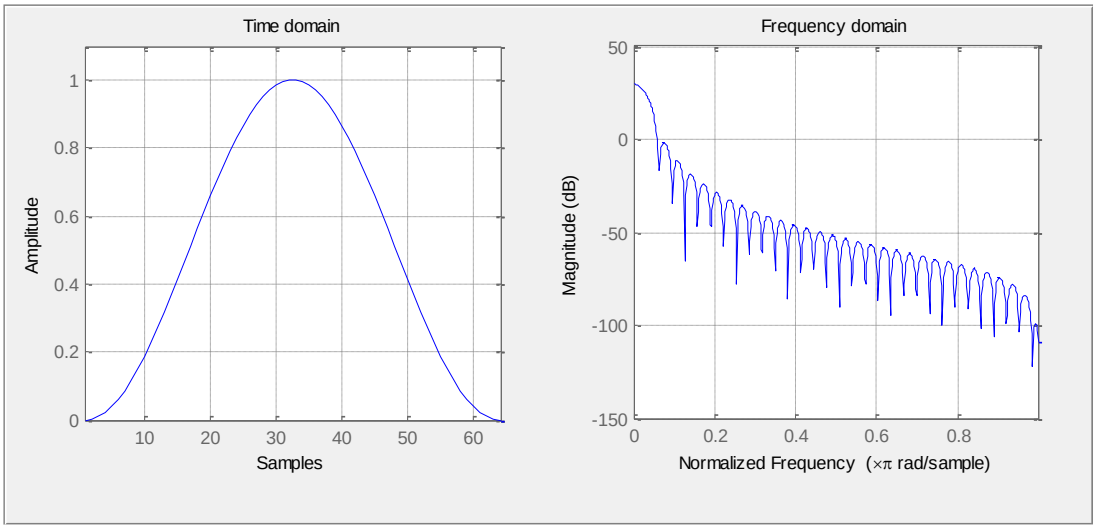


汉宁窗：hann

代码示例：

```
L=64;
wvtool(hann(L))
```

代码输出：

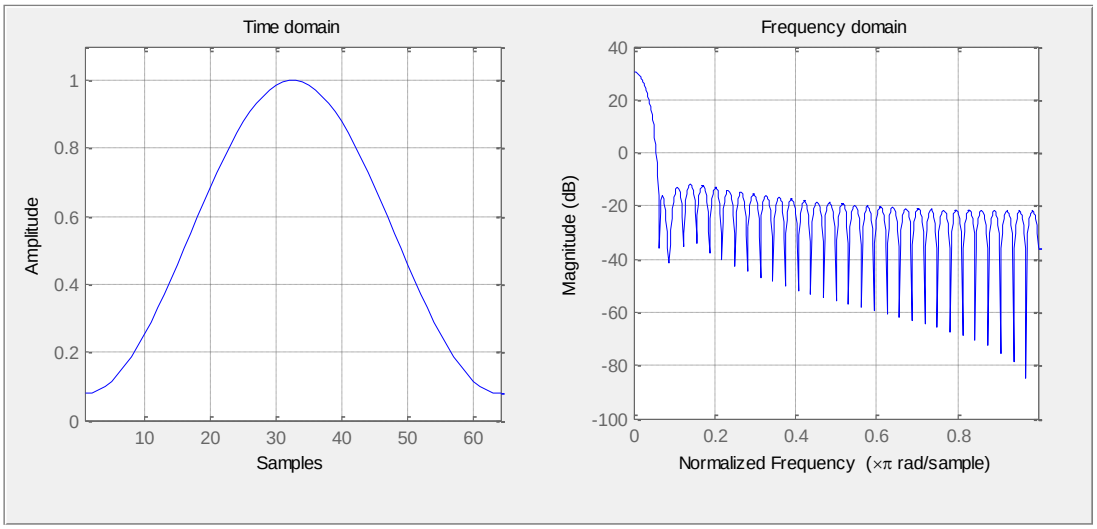


海明窗：hamming

代码示例：

```
L=64;
wvtool(hamming(L))
```

代码输出：



二．IIR 无限冲击响应滤波分析：贝塞尔,巴特沃斯,切比雪夫（1）

1.无限脉冲响应滤波器定义：

无限脉冲响应滤波器，简称 IIR 数字滤波器（英语：infinite impulse response filter），是数字滤波器的一种。由于无限脉冲响应滤波器中存在反馈回路，因此对于脉冲输入信号的响应是无限延续的。

无限脉冲响应滤波器的脉冲响应无限长，不像有限脉冲响应滤波器，所以有以下问题：

如何使能量集中在 $n=0$ 附近。

如何正向 Z 转换(forward Z transform)以及逆向 Z 转换(inverse Z transform)为稳定。

能使用最小相位滤波器(minimum phase filter)来解决以上问题，使一无限脉冲响应滤波器为稳定且能量集中在 $n=0$ 附近。

最小相位滤波器(minimum phase filter)：所有的极点(poles)以及零点(zeros)都在单位圆以内。

最小相位滤波器为稳定(stable)且因果(causal)，且最小相位滤波器的逆向(inverse)转换也为稳定(stable)且因果(causal)。

$$H(z)=C\frac{(z-z_1)(z-z_2)(z-z_3)...(z-z_R)}{(z-p_1)(z-p_2)(z-p_3)...(z-p_S)}$$

$$H^{-1}(z)=C^{-1}z^{S-R}\frac{(1-p_1z^{-1})(1-p_2z^{-1})(1-p_3z^{-1})...(1-p_Sz^{-1})}{(1-z_1z^{-1})(1-z_2z^{-1})(1-z_3z^{-1})...(1-z_Rz^{-1})}$$

其中 $z_1,z_2,...z_R$ 为 R 个零点，$p_1,p_2,...p_S$ 为 S 个极点，R 个零点和 S 个极点都在单位圆内。

如何将无限脉冲响应滤波器转换成最小相位滤波器：

$$H(z)=C\frac{(z-z_1)(z-z_2)(z-z_3)...(z-z_R)}{(z-p_1)(z-p_2)(z-p_3)...(z-p_S)}$$

假设 z_2 在单位圆内。

$$H_1(z)=C\frac{(z-z_1)(z-z_2)(z-z_3)...(z-z_R)}{(z-p_1)(z-p_2)(z-p_3)...(z-p_S)}\times z_2\frac{z-\overline{(z_2^{-1})}}{(z-z_2)}$$

$$=z_2C\frac{(z-z_1)(z-\overline{(z_2^{-1})})(z-z_3)...(z-z_R)}{(z-p_1)(z-p_2)(z-p_3)...(z-p_S)}$$

$H_1(z)$ 为最小相位滤波器(minimum phase filter)，符合$|H_1(z)|=|H(z)|$且$H_1(z)$和$H(z)$只有相位不同。

另外，$z_2\frac{z-\overline{(z_2^{-1})}}{(z-z_2)}$称为全通滤波器(all pass filter)。

无限脉冲响应滤波器为一个最小相位滤波器和一个全通滤波器串接(cascade)：

$$H(z)=H_{mp}(z)H_{ap}(z)$$

其中 $H(z)$：无限脉冲响应滤波器，$H_{mp}(z)$：最小相位滤波器，$H_{ap}(z)$：全通滤波器。

2.贝塞尔：

在电子学和信号处理领域，贝塞尔滤波器（Bessel filter）是具有最大平坦的群延迟（线性相位响应）的线性滤波器。贝塞尔滤波器常用在音频天桥系统中。模拟贝赛尔滤波器在几乎整个通频带都具有恒定的群延迟，因而在通频带上保持了被过滤的信号波形。滤波器的得名德国数学家弗雷德里希·贝塞尔，他发展了滤波器的数学理论基础。

描述贝塞耳滤波器低通滤波器的传递函数如下：

$$H(s)=\frac{\theta_n(0)}{\theta_n(s/\omega_0)}$$

这里 θn(s)是一个反向贝塞耳多项式，ω0 是选定的期望截止频率。

下面是一个三阶贝塞尔低通滤波

$$H(s)=\frac{15}{s^3+6s^2+15s+15}.$$

gain 值为

$$G(\omega)=|H(j\omega)|=\frac{15}{\sqrt{\omega^6+6\omega^4+45\omega^2+225}}.$$

相位为

$$\phi(\omega) = -\arg(H(j\omega)) = -\arctan\left(\frac{15\omega - \omega^3}{15 - 6\omega^2}\right).$$

群延迟为

$$D(\omega) = -\frac{d\phi}{d\omega} = \frac{6\omega^4 + 45\omega^2 + 225}{\omega^6 + 6\omega^4 + 45\omega^2 + 225}.$$

群延迟的泰勒级数展开为

$$D(\omega) = 1 - \frac{\omega^6}{225} + \frac{\omega^8}{1125} + \cdots.$$

注意在 ω2 和 ω4 的二个 term 是零，在 ω=0 造成非常平坦的群延迟。这是可以调整到零 term 的最大数量，因为在三阶贝赛尔多项式中总共有四个系数，要求定义四个等式。一个等式是为了在 ω = 0 时 the gain be unity，第二个等式指定 ω =无穷时 gain 是零，剩下二个等式指定二个 terms 的级数展开是零。这是 n 秩贝赛尔滤波的群延迟的一般特性：在群延迟的前 n-1 级数展开的 term 为零，因而 ω = 0 时群延迟的扁平得以最大化。

3.巴特沃斯：

巴特沃斯滤波器是电子滤波器的一种。巴特沃斯滤波器的特点是通频带的频率响应曲线最平滑。这种滤波器最先由英国工程师斯替芬·巴特沃斯（Stephen Butterworth）在 1930 年发表在英国《无线电工程》期刊的一篇论文中提出的。

巴特沃斯低通滤波器可用如下振幅的平方对频率的公式表示:

$$\left|H(\omega)\right|^2 = \frac{1}{1+\left(\frac{\omega}{\omega_c}\right)^{2n}} = \frac{1}{1+\text{ò}^2\left(\frac{\omega}{\omega_p}\right)^{2n}}$$

其中, n = 滤波器的阶数

ωc =截止频率 =振幅下降为 -3 分贝时的 频率

ωp = 通频带边缘频率

1/(1 + ε2) = |H(ω)|2 在通频带边缘的数值.

在二维复平面上$\left|H(\omega)\right|^2 = H(s)H^*(s) = H(s)H(-s)$ 在 s = jω 点的数值= |H(ω)|2,

因此通过解析延拓:

$$H(s)H(-s) = \frac{1}{1+\left(\frac{-s^2}{\omega_c^2}\right)^n}$$

上述函数的极点等距离地分布在半径为 ωc 的圆上

$$\frac{-s^2}{\omega_c^2} = (-1)^{\frac{1}{n}} = e^{\frac{j(2k+1)\pi}{n}}$$

k = 0, 1, 2, ..., n-1

因此,

$$s_k = \omega_c e^{\frac{j\pi}{2}} e^{\frac{j(2k+1)\pi}{2n}}$$

k = 0, 1, 2, ..., n-1

n 阶巴特沃斯低通滤波器的振幅和频率关系可用如下的公式表示：

$$G_n(\omega) = |H_n(j\omega)| = \frac{1}{\sqrt{1+(\omega/\omega_c)^{2n}}}$$

其中：

G 表示滤波器的放大率，

H 表示 传递函数，

j 是 虚数单位，

n 表示滤波器的级数，

ω 是信号的 角频率，以弧度/秒 为单位，

ω_c 是振幅下降 3 分贝时的截止频率。

令截止频率$\omega_c = 1$)，将上列公式规定一化成为：

$$G_n(\omega) = |H_n(j\omega)| = \frac{1}{\sqrt{1+\omega^{2n}}}$$

4.切比雪夫（1）：

切比雪夫滤波器（又译车比雪夫滤波器）是在通带或阻带上频率响应幅度等波纹波动的滤波器。在通带波动的为“I 型切比雪夫滤波器”，在阻带波动的为“II 型切比雪夫滤波器”。切比雪夫滤波器在过渡带比巴特沃斯滤波器的衰减快,但频率响应的幅频特性不如后者平坦。切比雪夫滤波器和理想滤波器的频率响应曲线之间的误差最小,但是在通频带内存在幅度波动。

I 型切比雪夫滤波器

I 型切比雪夫滤波器最为常见。

n 阶第一类切比雪夫滤波器的幅度与频率的关系可用下列公式表示：

$$G_n(\omega) = |H_n(j\omega)| = \frac{1}{\sqrt{1+\text{ò}^2 T_n^2\left(\frac{\omega}{\omega_0}\right)}}$$

其中：

$|ò|<1$

而$|H(\omega_0)|=\frac{1}{\sqrt{1+ò^2}}$ 是滤波器在截止频率 ω_0 的放大率（注意：常用的以幅度下降3分贝的频率点作为截止频率的定义不适用于切比雪夫滤波器!）

$T_n\left(\frac{\omega}{\omega_0}\right)$ 是 n 阶切比雪夫多项式：

$$T_n\left(\frac{\omega}{\omega_0}\right)=\cos\left(n*\arccos\frac{\omega}{\omega_0}\right);0\le\omega\le\omega_0$$

$$T_n\left(\frac{\omega}{\omega_0}\right)=\cosh\left(n*\operatorname{arccosh}\frac{\omega}{\omega_0}\right);\omega>\omega_0$$

或:

$$T_n\left(\frac{\omega}{\omega_0}\right)=a_0+a_1\frac{\omega}{\omega_0}+a_2\left(\frac{\omega}{\omega_0}\right)^2+\cdots+a_n\left(\frac{\omega}{\omega_0}\right)^n;0\le\omega\le\omega_0$$

$$T_n\left(\frac{\omega}{\omega_0}\right)=\frac{\left(\frac{\omega}{\omega_0}\sqrt{\left(\frac{\omega}{\omega_0}\right)^2-1}\right)^n+\left(\frac{\omega}{\omega_0}\sqrt{\left(\frac{\omega}{\omega_0}\right)^2-1}\right)^{-n}}{2};\omega>\omega_0$$

切比雪夫滤波器的阶数等于此滤波器的电子线路内的电抗元件数。

切比雪夫滤波器的幅度波动 = $20\log_{10}\sqrt{1+ò^2}$ 分贝

当 $ò=1$, 切比雪夫滤波器的幅度波动= 3 分贝。
如果需要幅度在在阻频带边上衰减得更陡峭，可允许在复平面的 $j\omega$ 轴上存在零点。但结果会使通频带内振幅波动较大，而在阻频带内对信号抑制较弱。 这种滤波器叫椭圆函数滤波器或考尔滤波器。

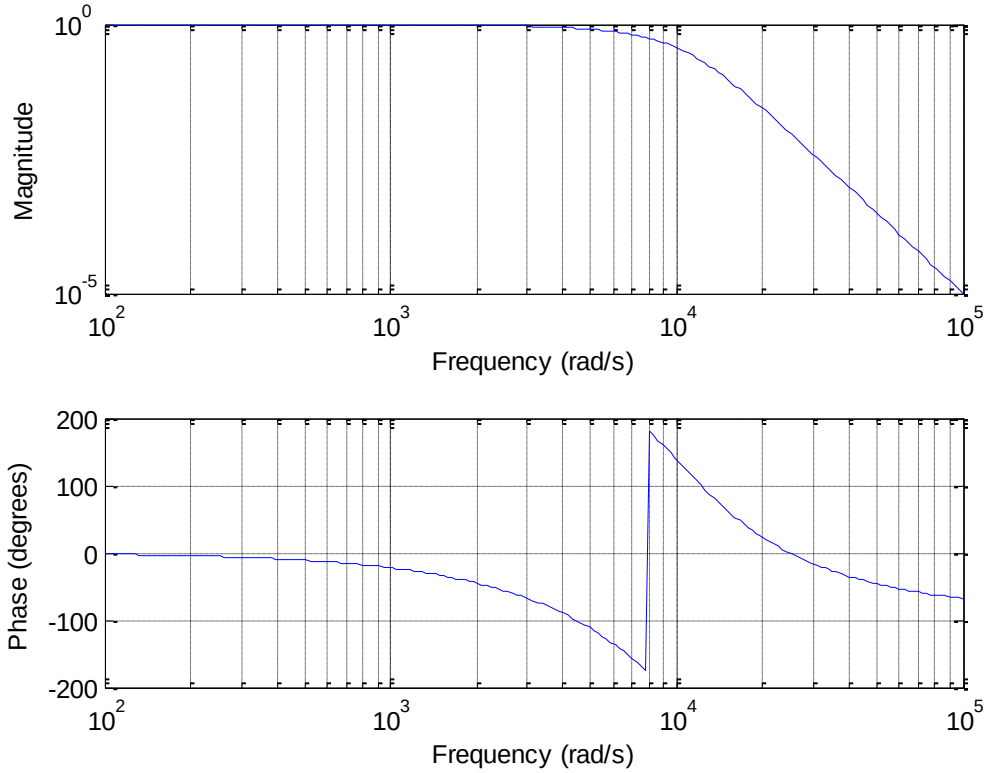
4.Matlab 函数：

贝塞尔：besself

代码示例：

```
[b,a] = besself(5,10000);
freqs(b,a)
```

代码输出：

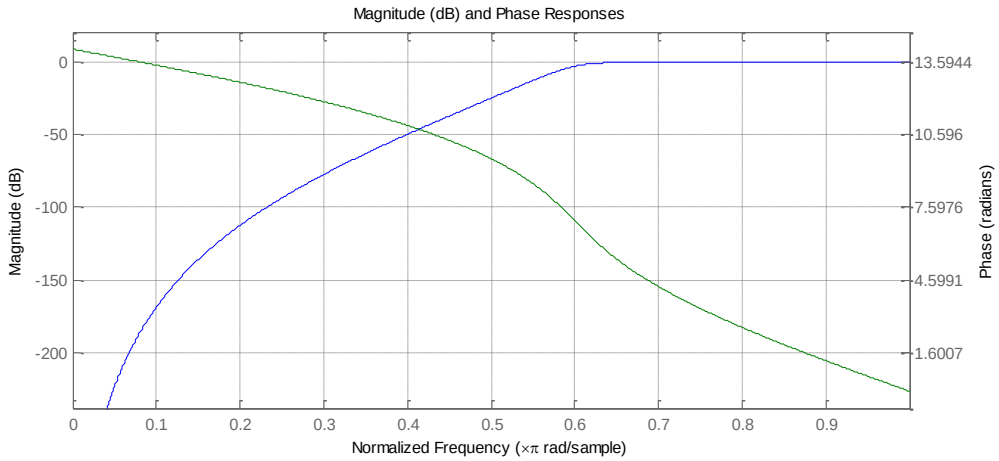


巴特沃斯：butter

代码示例：

```
[z,p,k] = butter(9,300/500,'high');
[sos,g] = zp2sos(z,p,k); % Convert to SOS form
Hd = dfilt.df2tsos(sos,g); % Create a dfilt object
h = fvtool(Hd); % Plot magnitude response
set(h,'Analysis','freq')
```

代码输出：

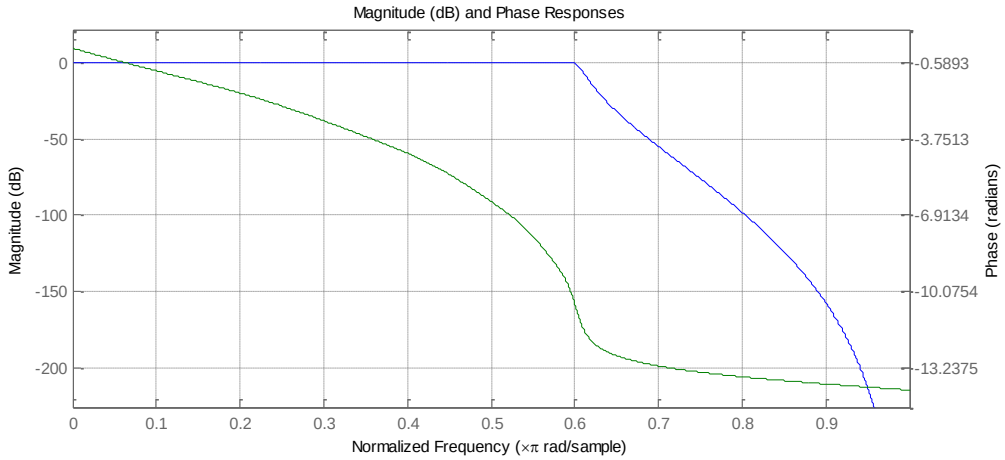


切比雪夫（1）：cheby1

代码示例：

```
[z,p,k] = cheby1(9,0.5,300/500);
[sos,g] = zp2sos(z,p,k); % Convert to SOS form
Hd = dfilt.df2tsos(sos,g); % Create a dfilt object
h = fvtool(Hd) % Plot magnitude r
set(h,'Analysis','freq')
```

代码输出：



三．时域/频域分析算法 FFT，频谱，互功率谱,自功率谱

1.FFT:

快速傅里叶变换（英语：Fast Fourier Transform，FFT），是离散傅里叶变换的快速算法，也可用于计算离散傅里叶变换的逆变换。快速傅里叶变换有广泛的应用，如数字信号处理、计算大整数乘法、求解偏微分方程等等。本条目只描述各种快速算法。

对于复数序列 $x_0, x_1, ..., x_{n-1}$，离散傅里叶变换公式为：

$$y_j = \sum_{k=0}^{n-1} e^{-\frac{2\pi\imath}{n}jk} x_k \qquad j = 0,1,\ldots,n-1.$$

直接变换的计算复杂度是 $\mathrm{O}(n^2)$（参见大 O 符号）。快速傅里叶变换可以计算出与直接计算相同的结果，但只需要 $\mathrm{O}(n \log n)$ 的计算复杂度。通常，快速算法要求 n 能被因数分解，但不是所有的快速傅里叶变换都要求 n 是合数，对于所有的整数 n，都存在复杂度为 $\mathrm{O}(n \log n)$ 的快速算法。

除了指数的符号相反、并多了一个 1/n 的因子，离散傅里叶变换的正变换与逆变换具有相同的形式。因此所有的离散傅里叶变换的快速算法同时适用于正逆变换。

2.频谱：

频谱是指一个时域的信号在频域下表示方式，可以针对信号进行傅立叶变换而得，所得的结果会是以分别以振幅及相位为纵轴，频率为横轴的两张图，不过有时也会省略相位的资讯，只有不同频率下对应振幅的资料。有时也以“振幅频谱”表示振幅随频率变化的情形，“相位频谱”表示相位随频率变化的情形。

频谱分析是一种将复噪声号分解为较简单讯号的技术。许多物理讯号均可以表示为许多不同频率简单讯号的和。找出一个讯号在不同频率下的资讯（可能是振幅、功率、强度或相位等）的作法就是频谱分析。

频谱分析可以对整个讯号进行。不过有时也会将讯号分割成几段，再针对各段的讯号进行频谱分析。周期函数（例如 $sin(t)$ ）最适合只考虑一个周期的讯号来进行频谱分析。傅立叶分析中有许多分析非周期函数时需要的数学工具。

一个函数的傅立叶变换包括了原始讯号中的所有资讯，只是表示的型式不同。因此可以用反傅立叶变换重组原始的讯号。若要完整的重组原始讯号，需要有每个频率下的振幅及其相位，这些资讯可以用二维向量、复数、或是极座标下的大小及角度来表示。在讯号处理中常常考虑振幅的平方，也就是功率，所得的就是功率谱密度。

实际上，大部份的仪器及软件都用快速傅立叶变换（FFT）来产生频谱的讯号。快速傅立叶变换是一种针对取样讯号计算离散傅里叶变换的数学工具，可以近似傅立叶变换的结果。

随机性讯号（或噪声）的傅立叶变换也是随机性的。需要利用一些取平均值的方式来得到其频率分布（frequency distribution）。一般来说会将资料依一定的时间分段，将各段资料进行傅立叶变换，再将转换后的振幅或振幅平方（振幅平方较常用）平均，以得到傅立叶变换的平均值。在处理取样的时域资料时，常用上述的作法，配合离散傅立叶变换来处理，这种处理方式称为 Welch 法（Welch's method）。若所得的频谱是平的，此讯号会视为“白噪声”，不过许多讯号在时域下看似噪声，却可以借由这样的处理方式得到一些频域的资讯。

3.互功率谱：

互功率谱简称互谱，它反映了两个信号中共同的频率成分。互谱为复频谱，包括模和相角两部分，模的大小等于两个信号中共同频率分量幅值乘积的 1/2，相角等于它们的相位差。与定义自谱类似，对互相关函数进行傅立叶变换，得到互功率谱密度函数，用符号 $S_{xy}(\omega)$ 表示。

$$r_{xy}(m) \Leftrightarrow R_{xy}(\omega) = S_{xy}(\omega)$$

互相关序列 $r_{xy}(m) = E\left[x(n)\, y^{*}(n+m) \right]$

互功率谱具有如下对称性质：

a、$S_{xy}(\omega) = S_{xy}^{*}(-\omega)$ 推导过程如下

$$S_{xy}(\omega) = \sum_{m=-\infty}^{\infty} E\left[x(n)\, y^{*}(n+m) \right] e^{-j\omega m}$$

$$= \left[\sum_{m=-\infty}^{\infty} E\left(x^{*}(n) y(n+m) \right) e^{j\omega n} \right]^{*} = \left[\sum_{m=-\infty}^{\infty} r_{xy} e^{j\omega n} \right]^{*}$$

$$= S_{xy}^{*}(-\omega)$$

b、$S_{xy}(\omega) = S_{yx}^{*}(\omega)$ 推导过程如下

$$S_{xy}(\omega) = \sum_{m=-\infty}^{\infty} E\left[x(n)\, y^{*}(n+m) \right] e^{-j\omega m} \overset{k=n+m}{=} \sum_{m=-\infty}^{\infty} E\left[x(k-m)\, y^{*}(k) \right] e^{-j\omega m}$$

$$= \sum_{m=-\infty}^{\infty} E\left[x(k+m)\, y^{*}(k) \right] e^{j\omega m} = \left[\sum_{m=-\infty}^{\infty} E\left[y(k) x^{*}(k+m) \right] e^{-j\omega m} \right]^{*}$$

$$= S_{yx}^{*}(\omega)$$

c、$\left| S_{xy}(\omega) \right| * \left| S_{xy}(\omega) \right| \le S_{x}(\omega) S_{y}(\omega)$

4. 自功率谱：

a、能量有限随机信号（确定性信号）

$$x(n) \Leftrightarrow X(e^{j\omega})$$

根据帕斯维尔定理得

$$\sum_{n=-\infty}^{\infty} x^{2}(n) = \frac{1}{2\pi} \int_{-\pi}^{\pi} \left| X(e^{j\omega}) \right|^{2} d\omega$$

自相关序列 $r_{xx}(m) = \sum_{n=-\infty}^{\infty} x(n) x(n+m)$

$$r_{xx}(m) \Leftrightarrow R_{xx}(e^{j\omega})$$

$$R_{xx}(e^{j\omega}) = X(e^{j\omega}) X(e^{-j\omega}) = \left| X(e^{j\omega}) \right|^{2}$$

b、无限能量随机信号 x(n)

（1)、表示式

x(n)的傅立叶变换不存在，令 $\chi_N(n)=\begin{cases}x(n) & |n|\le N\\ 0 & |n|>N\end{cases}$

$$r_{xx}(0)=E\left[\chi^2(n)\right]=\langle x(n)x(n)\rangle=\lim_{N\to\infty}\frac{1}{2N+1}\sum_{n=-N}^{N}x(n)x(n)$$

$$=\lim_{N\to\infty}E\left[\frac{1}{2N+1}\sum_{n=-N}^{N}\chi_N(n)\chi_N(n)\right]$$

根据帕斯维尔定理，上式又= $\lim_{N\to\infty}E\left[\frac{1}{2N+1}\frac{1}{2\pi}\int_{-\pi}^{\pi}\left|\chi_N(e^{j\omega})\right|^2d\omega\right]$

又 $r_{xx}(0)=\Phi_x^2$ 为平均功率

$\Phi_x^2=\frac{1}{2\pi}\int_{-\pi}^{\pi}S_x(\omega)d\omega$

可得 $S_x(\omega)=\lim_{N\to\infty}\frac{1}{2N+1}E\left[\left|\chi_N(e^{j\omega})\right|^2\right]$

(2)、性质

$S_x(x)\ge 0$

$$S_x(x)=\lim_{N\to+\infty}\frac{1}{2N+1}E\left[\left|X_N(e^{j\omega})\right|^2\right]\ge 0$$

$S_x(\omega)=S_x(-\omega)$

由自相管序列性质 $r_{xx}(m)=r_{xx}(-m)$

得 $S_x(-\omega)=R_{xx}(-\omega)=R_{xx}(\omega)=S_x(\omega)$

5.Matlab 函数：

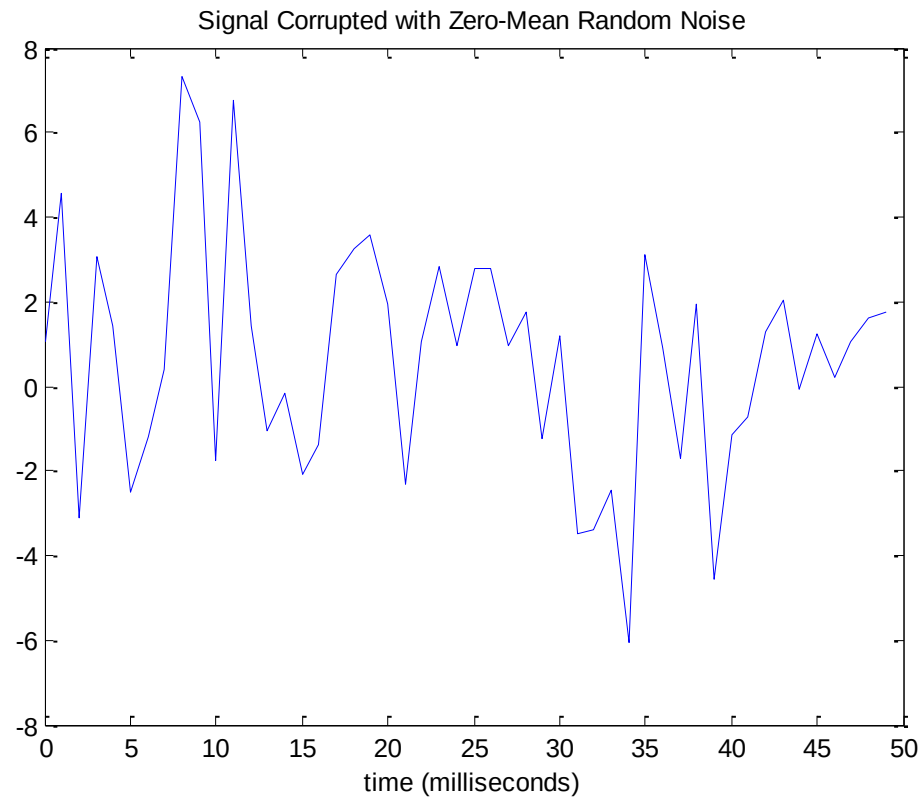
FFT: fft

代码示例：

```
Fs = 1000; % Sampling frequency
T = 1/Fs; % Sample time
L = 1000; % Length of signal
t = (0:L-1)*T; % Time vector
```

```
% Sum of a 50 Hz sinusoid and a 120 Hz sinusoid
x = 0.7*sin(2*pi*50*t) + sin(2*pi*120*t);
y = x + 2*randn(size(t)); % Sinusoids plus noise
plot(Fs*t(1:50),y(1:50))
title('Signal Corrupted with Zero-Mean Random Noise')
xlabel('time (milliseconds)')
```

代码输出：

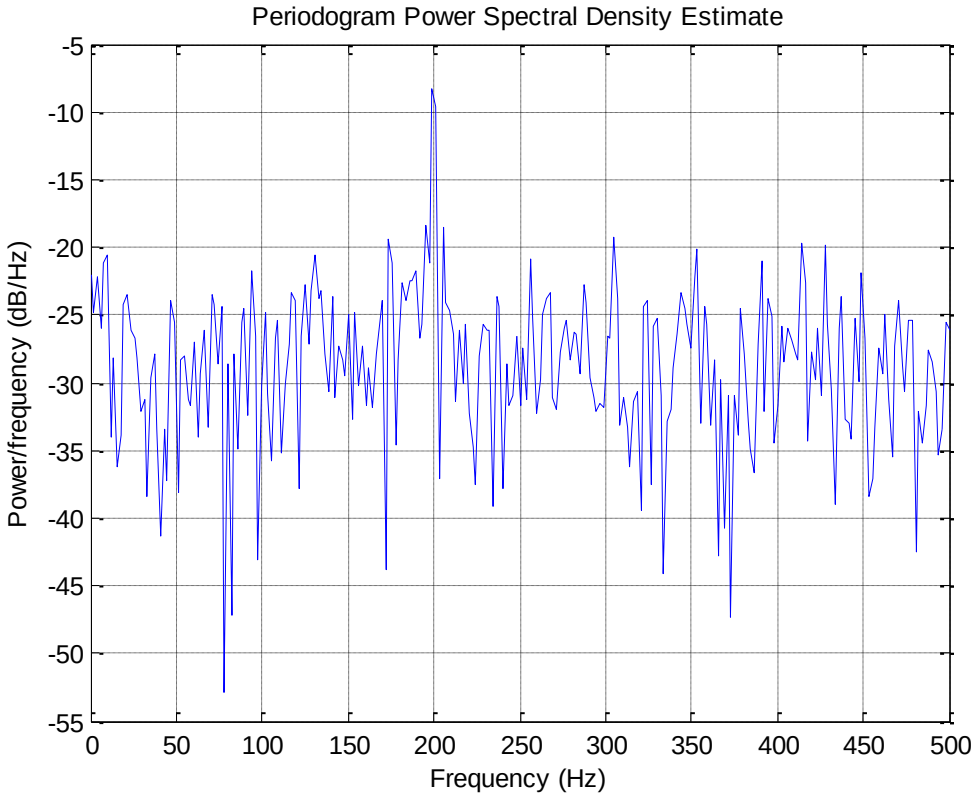


频谱：spectrum

代码示例：

```
Fs = 1000;
t = 0:1/Fs:.3;
x=cos(2*pi*t*200)+randn(size(t));
Hs=spectrum.periodogram;
psd(Hs,x,'Fs',Fs)
```

代码输出：

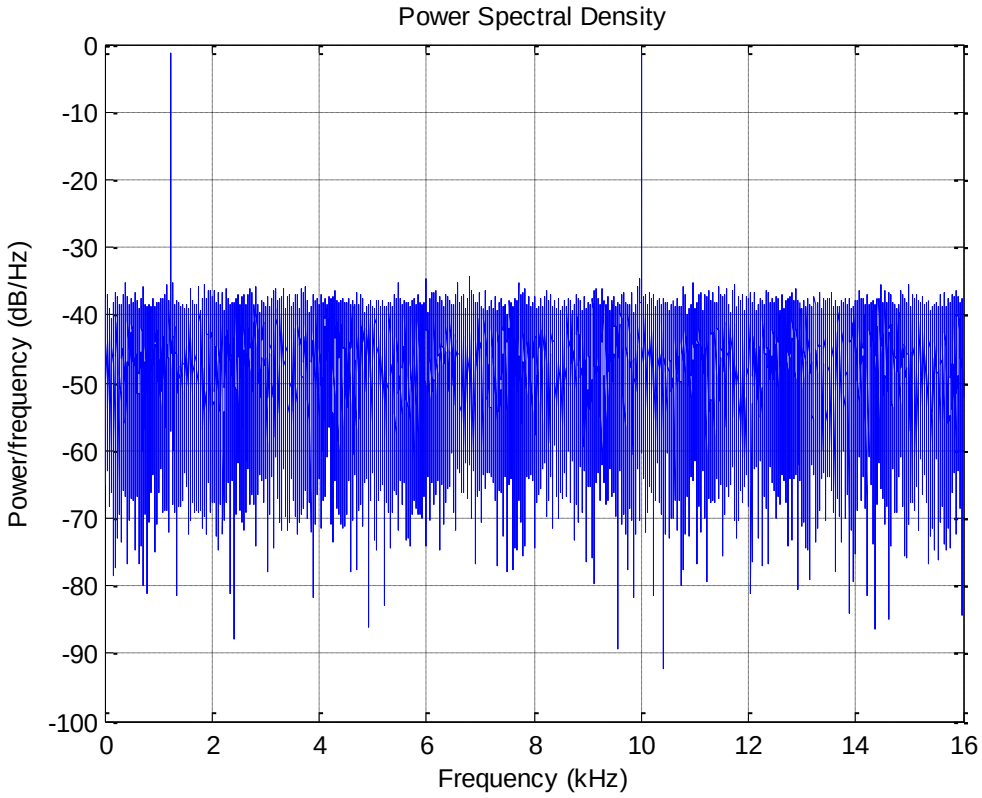


互功率谱： psd

代码示例：

```
Fs = 32e3;
t = 0:1/Fs:2.96;
x = cos(2*pi*t*1.24e3)+ cos(2*pi*t*10e3)+ randn(size(t));
nfft = 2^nextpow2(length(x));
Pxx = abs(fft(x,nfft)).^2/length(x)/Fs;
% Create a single-sided spectrum
Hpsd = dspdata.psd(Pxx(1:length(Pxx)/2),'Fs',Fs);
plot(Hpsd);
```

代码输出：

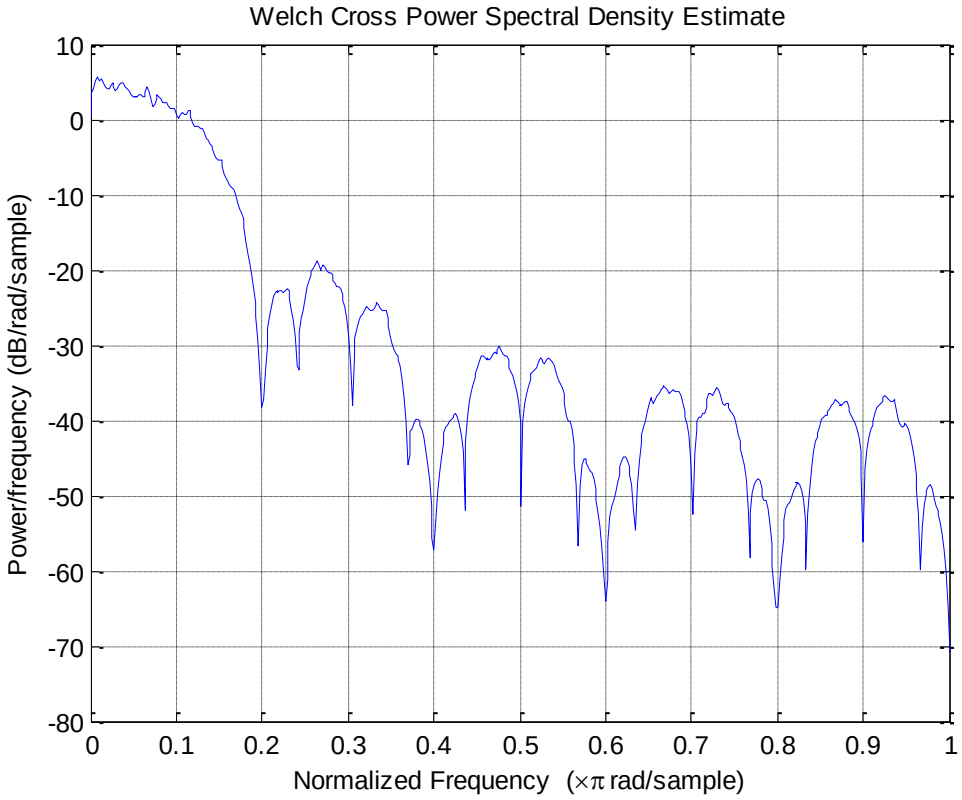


自功率谱： cpsd

代码示例：

```
randn('state',0);
h = fir1(30,0.2,rectwin(31));
h1 = ones(1,10)/sqrt(10);
r = randn(16384,1);
x = filter(h1,1,r);
y = filter(h,1,x);
cpsd(x,y,triang(500),250,1024)
```

代码输出：



四. C/C++调用 Matlab 函数

Visual C++是当前主流的应用程序开发环境之一，开发环境强大，开发的程序执行速度快。但在科学计算方面函数库显得不够丰富、读取、显示数据图形不方便。Matlab 是一款将数值分析、矩阵计算、信号处理和图形显示结合在一起，包含大量高度集成的函数可供调用，适合科学研究、工程设计等众多学科领域使用的一种简洁、高效的编程工具。不过由于 Matlab 使用的是解释性语言，大大限制了它的执行速度和应用场合。基于 VC 和 Matlab 混合编程是很多熟悉 VC++编程而又需要进行科学计算、数据仿真的科研人员常用的一种方式，其中最简单也最直接的方法就是调用 Matlab 引擎。以下部分将详细介绍通过 VC++6.0 调用 Matlab6.5 引擎来达到 VC++与 Matlab 数据共享编程的方法。

1.什么是 Matlab 引擎

所谓 Matlab 引擎（engine），是指一组 Matlab 提供的接口函数，支持 C/C++、Fortran 等语言，通过这些接口函数，用户可以在其它编程环境中实现对 Matlab 的控制。可以主要功能有：

★ 打开/关闭一个 Matlab 对话；

★ 向 Matlab 环境发送命令字符串；

★ 从 Matlab 环境中读取数据；

★ 向 Matlab 环境中写入数据。

与其它各种接口相比，引擎所提供的 Matlab 功能支持是最全面的。通过引擎方式，应用程序会打开一个新的 Matlab 进程，可以控制它完成任何计算和绘图操作。对所有的数据结构提供 100%的支持。同时，引擎方式打开的 Matlab 进程会在任务栏显示自己的图标，打开该窗口，可以观察主程序通过 engine 方式控制 Matlab 运行的流程，并可在其中输入任何 Matlab 命令。

实际上，通过引擎方式建立的对话，是将 Matlab 以 ActiveX 控件方式启动的。在 Matlab 初次安装时，会自动执行一次：

matlab /regserver

将自己在系统的控件库中注册。如果因为特殊原因，无法打开 Matlab 引擎，可以在 Dos 命令提示符后执行上述命令，重新注册。

2.配置编译器

要在 VC 中成功编译 Matlab 引擎程序，必须包含引擎头文件 engine.h 并引入 Matlab 对应的库文件 libmx.lib、libmat.lib、libeng.lib。具体的说，打开一个工程后，做如下设置（以 VC6 为例）：

1) 通过菜单工程/选项，打开设置属性页，进入 Directories 页面，在目录下拉列表框中选择 Include files，添加路径："C:/matlab/extern/include"（假定 matlab 安装在 C:/matlab 目录）。

2) 选择 Library files,添加路径:C:/matlab/extern/lib/win32/microsoft/msvc60。

3) 通过菜单工程/设置，打开工程设置属性页，进入 Link 页面，在 Object/library modules 编辑框中，添加文件名 libmx.lib libmat.lib libeng.lib。

以上步骤 1)、2)只需设置一次，而步骤 3)对每个工程都要单独设定，对于其它 C++编译器如 Borland C++ Builder，设置大体相同，不再赘述。

3.引擎 API 详解

在调用 Matlab 引擎之前，首先应在相关文件中加入一行：#include "enging.h"，该文件包含了引擎 API 函数的说明和所需数据结构的定义。可以在 VC 中调用的引擎函数分别如下：

3.1 引擎的打开和关闭

engOpen—打开 Matlab engine

函数声明：

```
Engine *engOpen(const char *startcmd);
```

参数 startcmd 是用来启动 Matlab 引擎的字符串参数，在 Windows 操作系统中只能为 NULL。

函数返回值是一个 Engine 类型的指针，它是在 engine.h 中定义的 engine 数据结构。

EngClose－关闭 Matlab 引擎

函数声明：

```
int engClose(Engine *ep);
```

参数 ep 代表要被关闭的引擎指针。

函数返回值为 0 表示关闭成功，返回 1 表示发生错误。

例如，通常用来打开/关闭 Matlab 引擎的代码如下：

```
Engine *ep; //定义 Matlab 引擎指针。
if (!(ep=engOpen(NULL))) //测试是否启动 Matlab 引擎成功。
{
MessageBox("Can't start Matlab engine!" );
exit(1);
}
. ……
engClose(ep); //关闭 Matlab 引擎。
```

3.2 向 Matlab 发送命令字符串

engEvalString－发送命令让 Matlab 执行。

函数声明：

```
int engEvalString(Engine *ep, Const char *string);
```

参数 ep 为函数 engOpen 返回的引擎指针，字符串 string 为要 matlab 执行的命令。

函数返回值为 0 表示成功执行，返回 1 说明执行失败（如命令不能被 Matlab 正确解释或 Matlab 引擎已经关闭了）。

3.3 获取 Matlab 命令窗口的输出

要在 VC 中获得函数 engEvalString 发送的命令字符串被 Matlab 执行后在 matlab 窗口中的输出，可以调用 engOUtputBuffer 函数。

函数声明：

```
int engOutputBuffer(Engine *ep, char *p, int n);
```

参数 ep 为 Matlab 引擎指针，p 为用来保存输出结构的缓冲区，n 为最大保存的字符个数，通常就是缓冲区 p 的大小。该函数执行后，接下来的 engEvalString 函数所引起的命令行输出结果会在缓冲区 p 中保存。如果要停止保存，只需调用代码：engOutputBuffer(ep, NULL, 0)。

3.4 读写 Matlab 数据

3.4.1 从 Matlab 引擎工作空间中获取变量。

```
mxArray *engGetVariable(Engine *ep, const char *name);
```

参数 ep 为打开的 Matlab 引擎指针，name 为以字符串形式指定的数组名。

函数返回值是指向 name 数组的指针，类型为 mxArray*（mxArray 数据类型在本文第 4 节详细简介）。

3.4.2 向 Matlab 引擎工作空间写入变量。

```
int engPutVariable(Engine *ep, const char *name, const mxArray *mp);
```

参数 ep 为打开的 Matlab 引擎指针，mp 为指向被写入变量的指针，name 为变量写入后在 Matlab 引擎工作空间中的变量名。

函数返回值为 0 表示写入变量成功，返回值为 1 表示发生错误。

3.5 调用引擎时显示/隐藏 Matlab 主窗口

默认情况下，以 engine 方式调用 Matlab 的时候，会打开 Matlab 主窗口，可在其中随意操作。但有时也会干扰应用程序的运行，可用以下设置是否显示该窗口。

```
int engSetVisible(Engine *ep, bool value);
```

参数 ep 为打开的 Matlab 引擎指针，value 为是否显示的标志，取值 true（或 1）表示显示 Matlab 窗口，取值 false（或 0）表示隐藏 Matlab 窗口。

函数返回值为 0 表示设置成功，为 1 表示有错误发生。

要获得当前 Matlab 窗口的显示/隐藏情况，可以调用函数：

```
int engGetVisible(Engine *ep, bool *value);
```

参数 ep 为打开的 Matlab 引擎指针，Value 为用来保存显示/隐藏情况的变量（采用指针方式传递）。

函数返回值为 0 表示获取成功，为 1 表示有错误发生。

4.数据类型 mxArray 的操作

在上节的 Matlab 引擎函数中，所有与变量有关的数据类型都是 mxArray 类型。数据结构 mxArray 以及大量的 mx 开头的函数，广泛用于 Matlab 引擎程序和 Matlab C 数学库中。mxArray 是一种很复杂的数据结构，与 Matlab 中的 array 相对应，我们只需熟悉 Matlab 的 array 类型和几个常用的 mxArray 函数即可。

在 VC 中，所有和 Matlab 的数据交互都是通过 mxArray 来实现的，在使用 mxArray 类型的程序中，应包含头文件 matrix.h，不过在引擎程序中，一般会包含头文件 engine.h，该文件里面已经包含了 matrix.h，因此无需重复包含。

4.1 创建和清除 mxArray 型数据

Matlab 有很多种变量类型，对应于每种类型，基本上都有一个函数用于创建，但它们都有相同的数据结构，就是 mxArray。

数组的建立采用 mxCreatexxx 形式的函数，例如新建一个 double 类型数组，可用函数 mxCreateDoubleMatrix，函数形式如下：

```
mxArray *mxCreateDoubleMatrix(int m, int n, mxComplexity ComplexFlag);
```

参数 m 和 n 为矩阵的函数和列数。ComplexFlag 为常数，用来区分矩阵中元素是实数还是复数，取值分别为 mxREAL 和 mxCOMPLEX。

例如，创建一个 3 行 5 列的二维实数数组，可用如下语句：

```
mxArray *T = mxCreateDoubleMatrix(3, 5, mxREAL);
```

对应的，要删除一个数组 mxDestroyArray，该函数声明如下：

```
void mxDestroyArray(mxArray *array_ptr);
```

参数 array_ptr 为要删除的数组指针。

例如，要删除上面创建的数组 T，可用如下语句：

```
mxDestroyArray(T);
```

类似的创建函数还有：

```
mxArray *mxCreateString(const char *str);
```

创建一个字符串类型并初始化为 str 字符串。

一般的在 VC 与 Matlab 交互中，以上两种类型就够了，其它类型数组的创建这里不再介绍。

4.2 管理 mxArray 数据类型

4.2.1 管理 mxArray 数据大小

要获得 mxArray 数组每一维上元素的个数，可以用 mxGetM 和 mxGetN 函数。其中 mxGetM 用来获得数组第一维的元素个数，对于矩阵来说就是行数。

```
int mxGetM(const mxArray *array_ptr); //返回 array_ptr 对应数组第一维的元素个数(行数)
int mxGetN(const mxArray *array_ptr); //返回 array_ptr 对应数组其它维的元素个数，对于矩阵来说是列数。对于多维数组来说是从第 2 维到最后一维的各维元素个数的乘积。
```

要获得某一特定维的元素个数，则要用函数：

```
const int *mxGetDimensions(const mxArray *array_ptr);
```

该函数返回 array_ptr 各维的元素个数保存在一个 int 数组中返回。对于常用的矩阵来说，用 mxGetM 和 mxGetN 两个函数就可以了。

另外还可以通过 mxGetNumberOfDimensions 来获得数组的总的维数，用 mxSetM、mxSetN 设置矩阵的行数和列数，函数说明如下：

```
mxGetNumberOfDimensions(const mxArray *array_ptr); //返回数组的维数
void mxSetM(mxArray *array_ptr, int m); //设置数组为 m 行
void mxSetN(mxArray *array_ptr, int n); //设置数组为 n 列
```

4.2.2 判断 mxArray 数组类型

在对 mxArray 类型的变量进行操作之前，可以验证以下其中的数组的数据类型，比如是否为 double 数组、整数、字符串、逻辑值等，以及是否为某种结构、类、或者是特殊类型，比如是否为空数组，是否为 inf、NaN 等。常见的判断函数有：

```
bool mxIsDouble(const mxArray *array_ptr);
bool mxIsComplex(const mxArray *array_ptr);
bool mxIsChar(const mxArray *array_ptr);
bool mxIsEmpty(const mxArray *array_ptr);
bool mxIsInf(double value);
…… ……
```

这些函数比较简单，意义自明，不再解释。

4.2.3 管理 mxArray 数组的数据

对于常用的 double 类型的数组，可以用 mxGetPr 和 mxGetPi 两个函数分别获得其实部和虚部的数据指针，这两个函数的声明如下：

```
double *mxGetPr(const mxArray *array_ptr); //返回数组 array_ptr 的实部指针
double *mxGetPi(const mxArray *array_ptr); //返回数组 array_ptr 的虚部指针
```

这样，就可以通过获得的指针对 mxArray 类型的数组中的数据进行读写操作。例如可以用函数 engGetVariable 从 Matlab 工作空间读入 mxArray 类型的数组，然后用 mxGetPr 和 mxGetPi 获得数据指针，对并其中的数据进行处理，最后调用 engPutVariable 函数将修改后的数组重新写入到 Matlab 工作空间。具体实现见第 5 节程序实例。

5.程序实例

对大部分软件研发人员来说利用 VC 编程方便、高效，但是要显示数据图形就不那么容易了，这时候不防借助 Matlab 引擎辅助画图做数据分析。下面通过实例演示如何利用 VC 调用 Matlab 绘图，程序的主要功能是在 VC 中对数组 x 计算函数值 y＝sin(x) ±log(x)，然后调用 Matlab 绘制 y 对 x 的图形。

在 VC 中新建工程，编写代码如下：

```
#include <iostream>
#include <math.h>
#include "engine.h"
using namespace std;
void main()
{
  const int N = 50;
```

```
    double x[N],y[N];
    int j = 1;
    for (int i=0; i<N; i++) //计算数组 x 和 y
    {
        x[i] = (i+1);
        y[i] = sin(x[i]) + j * log(x[i]); //产生一之间的随机数赋给 xx[i];
        j *= -1;
    }
    Engine *ep; //定义 Matlab 引擎指针。
    if (!(ep=engOpen(NULL))) //测试是否启动 Matlab 引擎成功。
    {
        cout <<"Can't start Matlab engine!" <<endl;
        exit(1);
    }
    //定义 mxArray，为行，N 列的实数数组。
    mxArray *xx = mxCreateDoubleMatrix(1,N, mxREAL);
    mxArray *yy = mxCreateDoubleMatrix(1,N, mxREAL); //同上。
    memcpy(mxGetPr(xx), x, N*sizeof(double)); //将数组 x 复制到 mxarray 数组 xx 中。
    memcpy(mxGetPr(yy), y, N*sizeof(double)); //将数组 x 复制到 mxarray 数组 yy 中。
    engPutVariable(ep, "xx",xx); //将 mxArray 数组 xx 写入到 Matlab 工作空间，命名为 xx。
    engPutVariable(ep, "yy",yy); //将 mxArray 数组 yy 写入到 Matlab 工作空间，命名为 yy。
    //向 Matlab 引擎发送画图命令。plot 为 Matlab 的画图函数，参见 Matlab 相关文档。
    engEvalString(ep, "plot(xx, yy); ");
    mxDestroyArray(xx); //销毁 mxArray 数组 xx 和 yy。
    mxDestroyArray(yy);
    cout <<"Press any key to exit!" <<endl;
    cin.get();
    engClose(ep); //关闭 Matlab 引擎。
}
```

6.小结

上面详细的介绍了 Matlab 引擎使用方法并演示了一个简单的利用 VC 调用 Matlab 画图

的程序实例。大多数时候，程序员可以利用 Matlab 强大的数据读写、显示能力和 VC 编程的高效率。例如，在 Matlab 中要读入一幅任意格式的图像均只需一条命令 i=imread('test.jp');图像数据矩阵便存放在了二维数组 i 中，可以通过 VC 读入该数组进行相关处理再调用 Matlab 显示，这种混合编程方式能大大提高工作效率。

当然，利用 VC 编译的 Matlab 引擎程序，运行环境中还必须 Matlab 的支持，如果要编译完全脱离 Matlab 的程序，可采用其它方式，如利用第三方 Matcom 程序编译独立的可执行程序等