

通信系统仿真

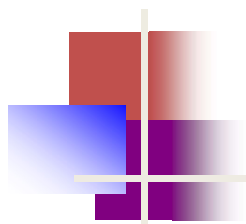
第2章 蒙特卡罗方法02

何晨光

哈尔滨工业大学

电子与信息工程学院

Communication Research Center

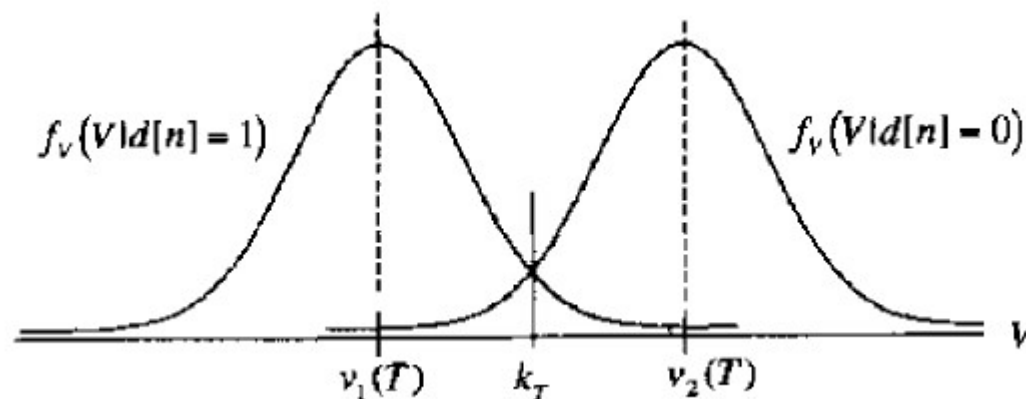


- 1 蒙特卡洛方法简史
- 2 基本概念
- 3 在通信系统中的应用
- 4 蒙特卡罗积分
- 5 繁琐通信系统中的应用
- 6 半解析方法

4 蒙特卡罗积分

对于AWGN信道，通过采样积分-清除（integrate-and-dump）检测器的输出得到所有的充分统计量 V ，是一个高斯随机变量，其均值由数据符号决定，方差由信道噪声决定。

在 $d[n]=0$ 和 $d[n]=1$ 的条件下，调价概率密度函数如下图所示，其中 K_T 是接收机阈值。



高斯噪声条件下二进制信号的条件概率密度函数



4 蒙特卡罗积分

在 $d[n]=1$ 的条件下的条件差错概率为

$$P_r(E|d[n]=1) = \int_{K_T}^{\infty} \frac{1}{\sqrt{2\pi}\sigma_n} \exp\left[-\frac{1}{2\sigma_n^2}(x - v_1(T))^2\right] dx$$

对于 $P_r(E|d[n]=0)$ 也可以推导出类似的表达式。
从而，系统的差错概率估计

$$P_E = \frac{1}{2} P_r(E|d[n]=1) + \frac{1}{2} P_r(E|d[n]=0)$$

所以，结果牵涉到积分值的估计问题



4 蒙特卡罗积分

假设，我们想求一下积分

$$I = \int_0^1 g(x) dx$$

其中 $g(x)$ 是一个在积分区域内有界的函数。由基本的概率论，可知函数 $g(x)$ 的总体均值为

$$E[g(x)] = \int_{-\infty}^{\infty} g(x) f_X(x) dx$$

其中 $f_X(x)$ 是随机变量 X 的概率密度函数。如果随机变量 X 的概率密度函数在区间 $(0,1)$ 上满足

$f_X(x) = 1$ ，而在其他地方为零，则 $E[g(x)] = I$ 。

因此，如果 U 是在区间 $(0,1)$ 上均匀分布的随机变量，那么有

$$I = E[g(U)]$$



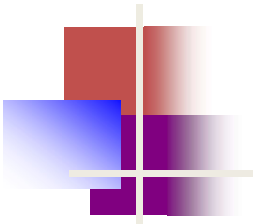
4 蒙特卡罗积分

利用相对频率的观点，可得

$$\lim_{N \rightarrow \infty} \left[\frac{1}{N} \sum_{i=0}^N g(U_i) \right] = E[g(U)] = I$$

因此，我们对被积函数进行仿真，再在 $(0,1)$ 区间上对它进行 N 次采样，采样点的平均值可以用来估计积分值。

系统蒙特卡罗仿真基本上也是这个思路，因为通常无法获得充分统计量在差错区域内的解析表达式，所以采用系统仿真来产生统计量的采样。


$$\lim_{N \rightarrow \infty} \left[\frac{1}{N} \sum_{i=0}^N g(U_i) \right] = E[g(U)] = I$$

可以得到积分的近似 $\frac{1}{N} \sum_{i=0}^N g(U_i) = \hat{I}$

通过对函数 $g(x)$ 在 N 个均匀分布的采样点上求值再取平均，就可以实现积分的估计器。

这种方法适合于任何常义积分。通过简单的变量代换，可用蒙特卡罗方法来估算任意区间上的积分。

例如：积分 $I = \int_a^b f_X(x) dx$

通过变量代换 $y = (x - a)/(b - a)$ 变成标准形式并求得：

$$I = (b - a) \int_0^1 f(a + (b - a)y) dy$$



4 蒙特卡罗积分

例3：为了用蒙特卡罗估计 π 值，只需要找到一个定积分值为 π 的函数。很容易就可以得到如下积分：

$$I = \int_0^1 \frac{1}{1+x^2} dx = \frac{\pi}{4}$$

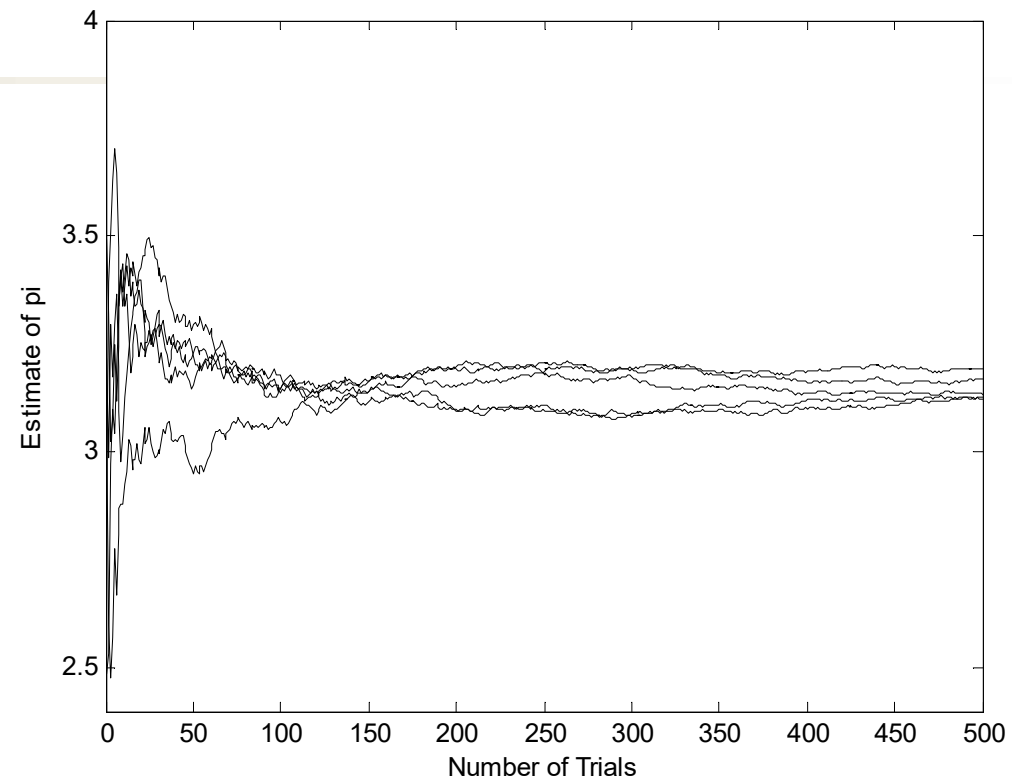
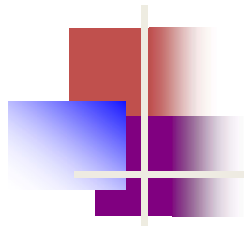
然后就可以利用 $\frac{1}{N} \sum_{i=0}^N g(U_i) = \hat{I}$ 定义的算法，求解

积分值 I ，并将所得结果乘以4。

(代码见c207.m)

4 蒙特卡罗积分

```
M=5; % Number of experiments
N=500; % Trials per experiment
u = rand(N,M); % Generate random numbers
uu = 1./(1+u.*u); % Define function
data = zeros(N,M); % Initialize array
% The following four lines of code determine
% M estimates as a function of j, 0<j<=N.
data(1,:) = 4*uu(1,:);
for j=2:N
    data(j,:)=4*sum(uu(1:j,:))/j;
end
est = data(N,:); % M estimates of pi
est1 = sum(est)/M; % Average estimate
plot(data,'k') % Plot results
xlabel('Number of Trials')
ylabel('Estimate of pi')
```



| 变量 - est | | | | | | |
|------------|--------|--------|--------|--------|--------|--|
| est | | | | | | |
| 1x5 double | | | | | | |
| | 1 | 2 | 3 | 4 | 5 | |
| 1 | 3.1269 | 3.1695 | 3.1932 | 3.1337 | 3.1227 | |
| 2 | | | | | | |

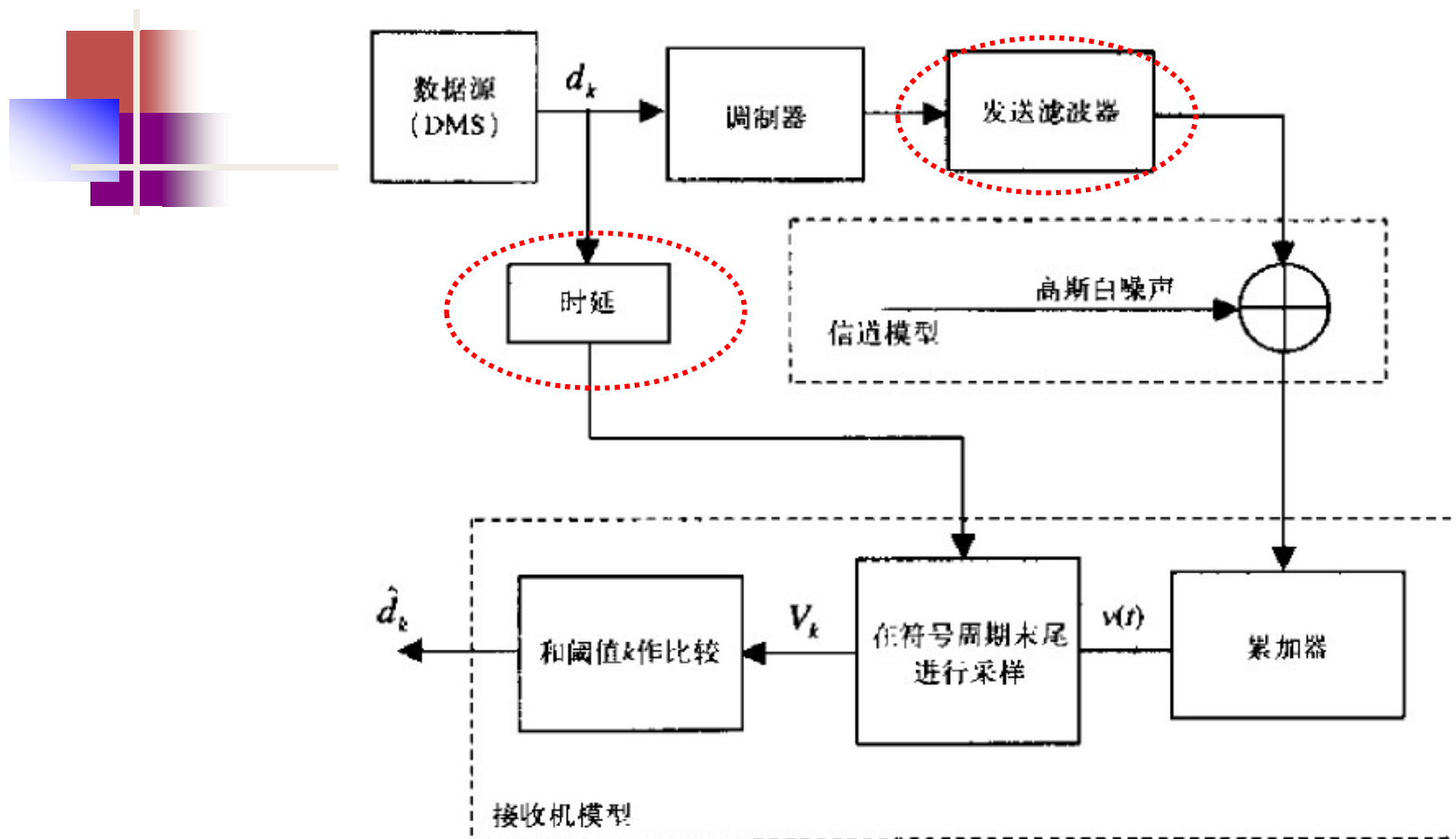
| est1 | | |
|------------|--------|---|
| 1x1 double | | |
| | 1 | 2 |
| 1 | 3.1492 | |
| 2 | | |

5 繁琐通信系统中的应用

将蒙特卡罗方法用于估计数字通信系统的误比特率时，是让N个采样符号通过系统的仿真模型，并计算产生差错的个数来实现的。假设通过系统仿真模型的N个符号导致了 N_e 个错误，那么误比特率的估计值为

$$\hat{P}_E = \frac{N_e}{N}$$

从前面学习可知 \hat{P}_E 是一个随机变量，要获得误比特率的准确估计，估计器必须是无偏的，并且具有最小的方差。小的方差要求有大的N，而这又会导致较长的计算时间。



例4：基本框图如图所示。假设为BPSK调制，调制器输出端的滤波器是三阶巴特沃思滤波器，其带宽等于比特率($BW = r_b$)，该滤波器会产生码间干扰 (ISI)。仿真的目的是确定由滤波器带来的ISI所增加的误比特率。

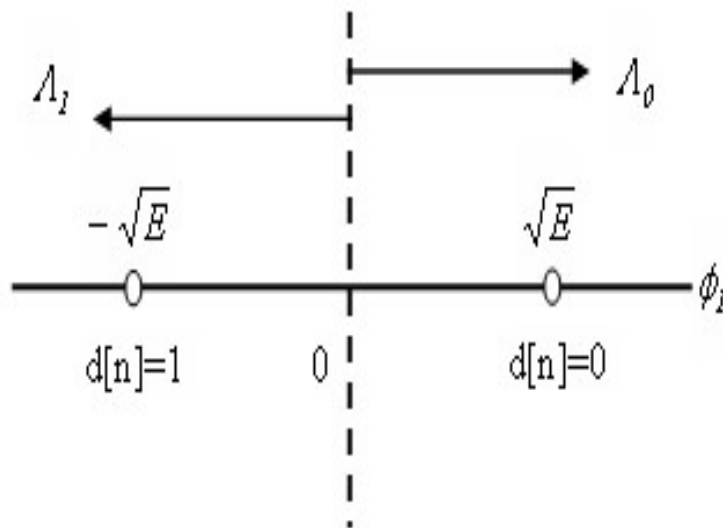
5 繁琐通信系统中的应用

BPSK 令 $A_c=1$ 和 $k_m=\pi$ ，由此可得

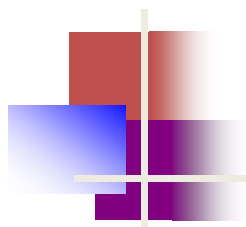
$$x_d[n] = \cos(\pi d[n]) = \begin{cases} 1, & d[n] = 0 \\ -1, & d[n] = 1 \end{cases}$$

$$x_q[n] = \sin(\pi d[n]) = 0$$

BPSK信号的空间表示



信号星座点中的两个信号点都位于桐乡信道上，在仿真中我们可以不考虑正交信道



- 步骤一：理论推导
- 步骤二：确定时延值
- 步骤三：编写BER子函数
- 步骤四：编写主函数

步骤二：确定时延

第一个问题就是要确定delay的值。

选定一个 E_b/N_0 值，用不同的delay的值对系统进行仿真，并观察结果。
(代码见c208.m)

```
EbNodB = 6; % Eb/No (dB) value
z = 10.^(EbNodB/10); % convert to linear scale
delay = 0:8; % delay vector
BER = zeros(1,length(delay)); % initialize BER vector
Errors = zeros(1,length(delay)); % initialize Errors vector
BER_T = q(sqrt(2*z))*ones(1,length(delay)); % theoretical BER vector
N = round(100./BER_T); % 100 errors for ideal (zero ISI) system
FilterSwitch = 1; % set filter switch (in=1 or out=0)
for k=1:length(delay)
    [BER(k),Errors(k)] = c209_MCBPSKrun(N(k),z,delay(k),FilterSwitch)
end
semilogy(delay,BER,'o',delay,BER_T,'-'); grid;
xlabel('Delay'); ylabel('Bit Error Rate');
% End of script file.
```

步骤二：确定时延

第一个问题就是要确定delay的值。

选定一个 E_b/N_0 值，用不同的delay的值对系统进行仿真，并观察结果。

```
EbNodB = 6;
z = 10.^(EbNodB/10);
delay = 0:8;
BER = zeros(1,length(delay));
Errors = zeros(1,length(delay));
BER_I = q(sqrt(2*z))*ones(1,length(delay));
N = round(100./BER_I);
FilterSwitch = 1;
for k=1:length(delay)
    [BER(k),Errors(k)] = c10_MCBPSKrun(N(k),z,de
end
semilogy(delay,BER,'o',delay,BER_I,'-'); grid;
xlabel('Delay'); ylabel('Bit Error Rate');
% End of script file.
```

时延从0到8个采样周期依次迭代。因为采样频率为每个符号10次采样，delay的步长为 $0.1T_s$ ，其中 T_s 为符号周期。

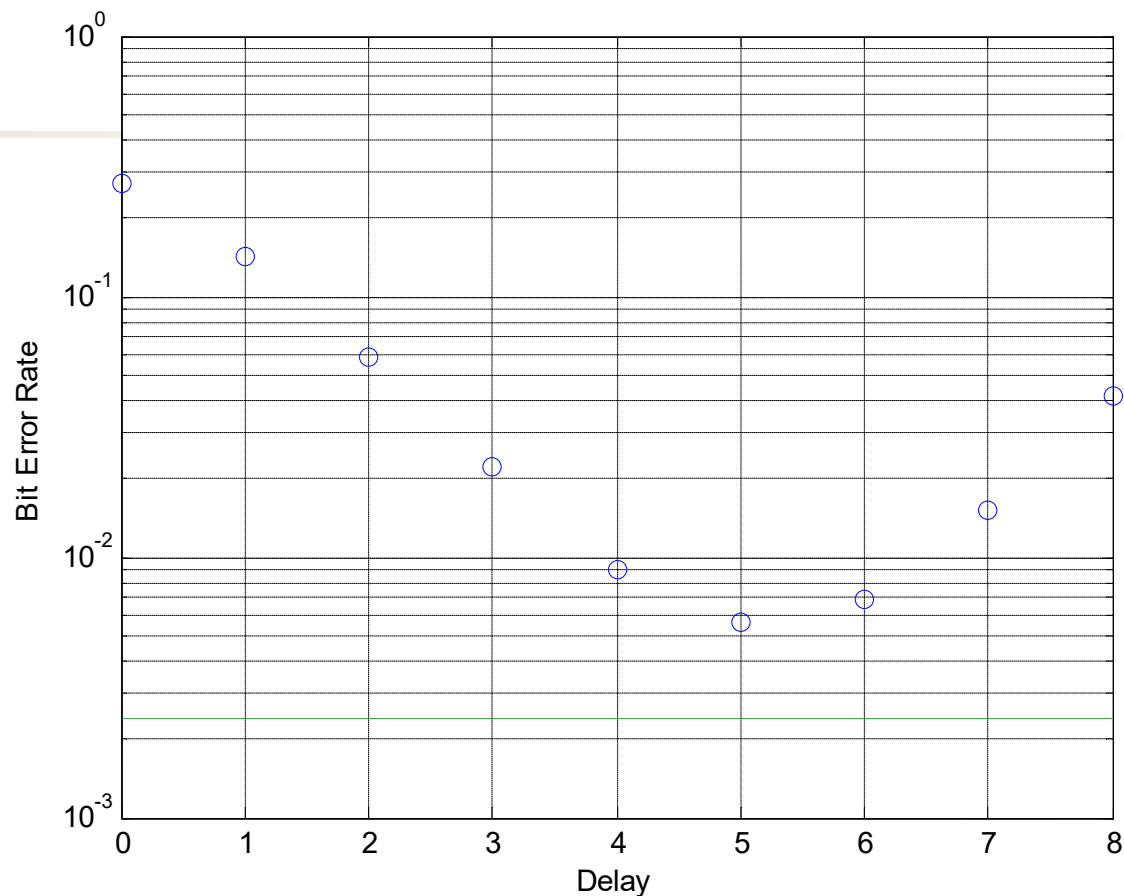
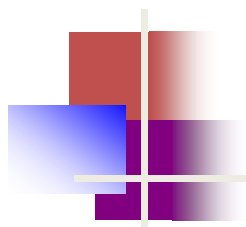
步骤二：确定时延

选择合适的N值使得有足够多的差错发生，从而保证适当小的估计器误差。

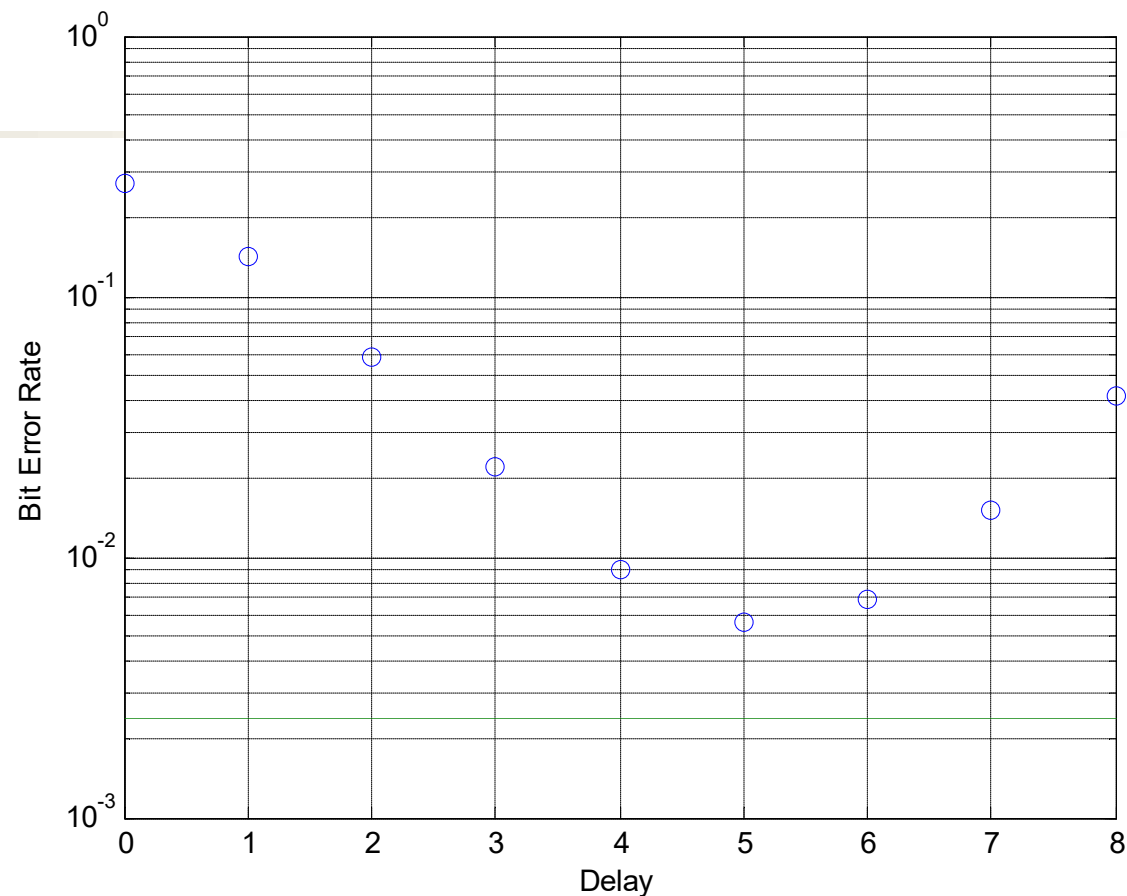
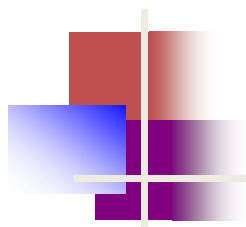
```
BER_I = q(sqrt(2*z))*ones(1,length(delay));  
N = round(100./BER_I);
```

BER_T是AWGN情况下的理论差错概率。

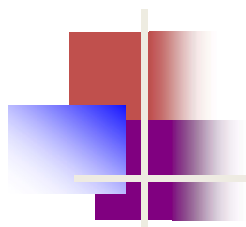
对于一个给定的 E_b/N_0 ，由于存在ISI和其它干扰，差错发生的次数会增加并超过平均值100。注意对每一个delay值，同时给出了误比特率的数值和用于计算误比特率的差错次数。这样就可以得出结论：在差错次数足够多的情况下可得到可靠的误比特率估计。



不同的仿真结果用小圆圈表示，作为参考，用实线表示在AWGN信道下无ISI时系统在 $E_b/N_0=6\text{dB}$ 时的性能。可以发现不正确的选择时延会导致过大的误比特率。



寻找时某延值时的系统有最小的误比特率。如图发现正确的时延值极有可能在5和6个采样周期之间。由于时延必须量化到采样周期的整数倍，所以选择时延为5个采样周期。可以用更高的采样频率来确定更精确的delay的值。



- 步骤一：理论推导
- 步骤二：确定时延值
- 步骤三：编写BER子函数
- 步骤四：编写主函数

```

function [BER, Errors]=MCBPSKrun(N, EbNo, delay, FilterSwitch)
    SamplesPerSymbol = 10; % samples per symbol
    BlockSize = 1000; % block size
    NoiseSigma = sqrt(SamplesPerSymbol/(2*EbNo)); % scale noise level
    DetectedSymbols = zeros(1,BlockSize); % initialize vector
    NumberOfBlocks = floor(N/BlockSize); % number of blocks processed
    [B1x,A1x] = butter(5, 2/SamplesPerSymbol); % compute filter parameters
    [IxOutput,IxFilterState] = filter(B1x,A1x,0); % initialize state vector
    BRx = ones(1, SamplesPerSymbol); ARx=1; % matched filter parameters
    Errors = 0; % initialize error counter
    %
    % Simulation loop begin here.
    %
    for Block=1:NumberOfBlocks
        %
        % Generate transmitted symbols.
        %
        [SymbolSamples,IxSymbols] = random_binary(BlockSize, SamplesPerSymbol);
        %
        % Transmitter filter if desired.
        %
        if FilterSwitch==0
            IxOutput = SymbolSamples;
        else
            [IxOutput,IxFilterState] = filter(B1x,A1x, SymbolSamples, IxFilterState);
        end
    end
end

```



步骤三：编写BER子函数

```
function [BER, Errors]=MCBPSKrun(N, EbNo, delay, FilterSwitch)
SamplesPerSymbol = 10;           % samples per symbol
BlockSize = 1000;                % block size
NoiseSigma = sqrt(SamplesPerSymbol/(2*EbNo)); % scale noise level
DetectedSymbols = zeros(1, BlockSize); % initialize vector
NumberOfBlocks = floor(N/BlockSize); % number of blocks processed
[B1x, A1x] = butter(5, 2/SamplesPerSymbol); % compute filter parameters
[IxOutput, IxFilterState] = filter(B1x, A1x, 0); % initialize state vector
BRx = ones(1, SamplesPerSymbol); ARx=1; % matched filter parameters
Errors = 0;                       % initialize error counter
```

(代码见c209.m)



步骤三：编写BER子函数

```
function [BER,Errors]=MCBPSKrun(N,EbNo,delay,FilterSwitch)
SamplesPerSymbol = 10;           % samples per symbol
BlockSize = 1000;                 % block size
```

本程序采用块级联（block-serial）的方法：迭代处理由1000个符号组成的块。

SamplesPerSymbol = 10 每符号采样的个数（采样频率）
BlockSize = 1000 块数

将SamplesPerSymbol 和BlockSize作为输入参数带入子函数random_binary

步骤三：编写BER子函数

数字调制器中一个常用模块：random_binary

```
% File: function random_binary
```

```
function [x,bits]=random_binary(nbits,nsamples)
```

```
x=zeros(1,nbits*nsamples)
```

```
bits=round(rand(1,nbits));
```

```
for m=1:nbits
```

```
    for n=1:nsamples
```

```
        index=(m-1)*nsamples+n;
```

```
        x(1,index)=(-1)^bits(m);
```

```
    end
```

```
end
```

```
% End of function
```

例：QPSK调制

```
x= random_binary(nbits,nsamples)+j*random_binary(nbits,nsamples)
```

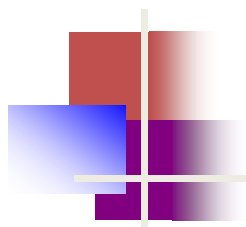
调制器的基本构建模块是函数random_binary，它产生电平值为+1和-1的二进制波形，产生的比特数以及每比特的采样数是该函数的参数。

步骤三：编写BER子函数

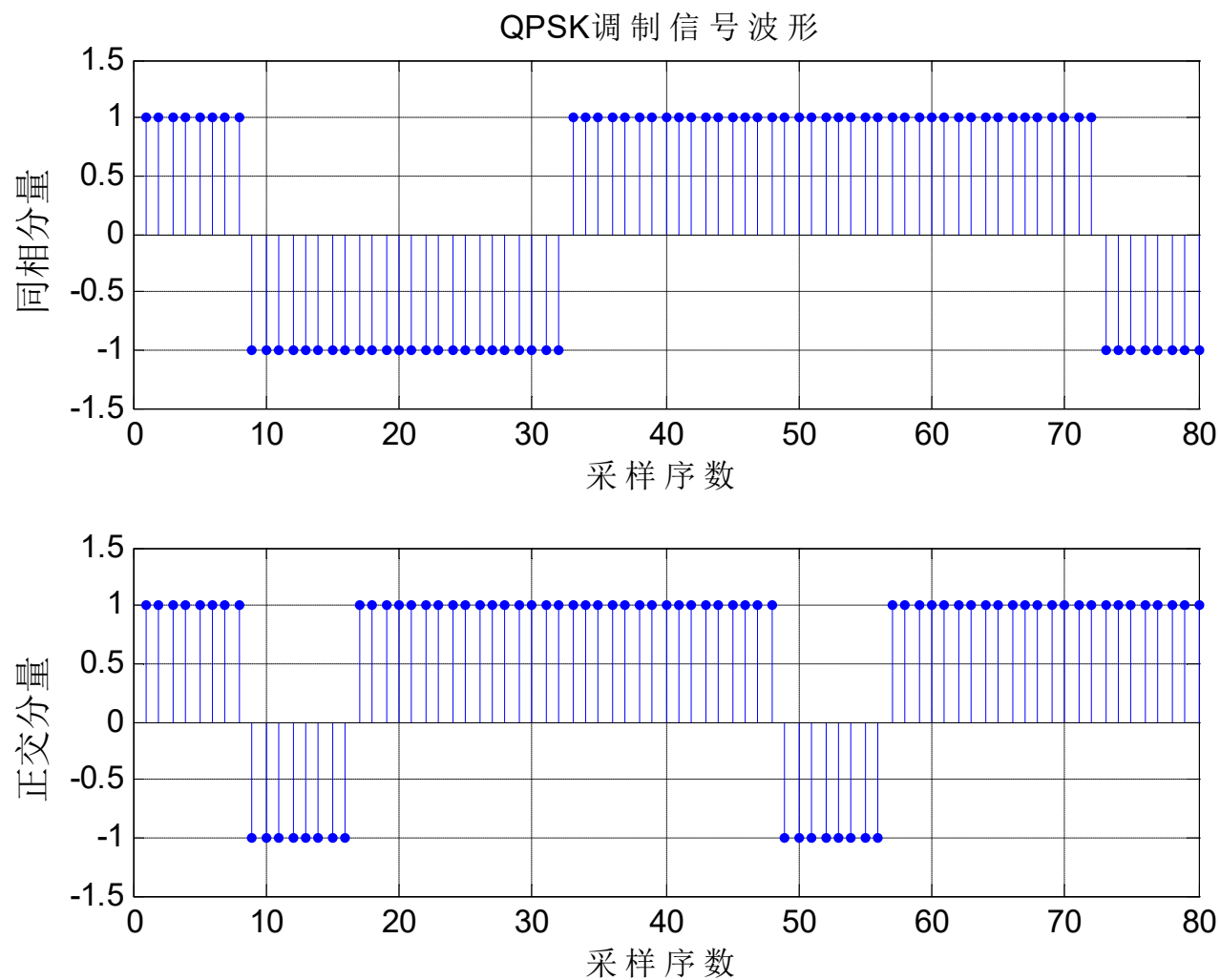
应用实例：产生一个10比特的QPSK信号，采样频率为每比特8个采样点

```
nbits = 10; nsamples = 8;  
x = random_binary(nbits,nsamples)+i*random_binary(nbits,nsamples);  
xd = real(x); xq = imag(x);  
subplot(2,1,1)  
stem(xd, '.'); grid; axis([0 80 -1.5 1.5]);  
xlabel('Sample Index'); ylabel('xd')  
subplot(2,1,2)  
stem(xq, '.'); grid; axis([0 80 -1.5 1.5]);  
xlabel('Sample Index'); ylabel('xq')
```

(代码见c206.m)



步骤三：编写BER子函数





步骤三：编写BER子函数

```
function [BER,Errors]=MCBPSKrun(N,EbNo,delay,FilterSwitch)
SamplesPerSymbol = 10;           % samples per symbol
BlockSize = 1000;                 % block size
NoiseSigma = sqrt(SamplesPerSymbol/(2*EbNo)); % scale noise level
DetectedSymbols = zeros(1,BlockSize); % initialize vector
NumberOfBlocks = floor(N/BlockSize); % number of blocks processed
```

NoiseSigma 计算噪声的能量

DetectedSymbols 初始化统计量数组

NumberOfBlocks 实际处理的包个数



步骤三：编写BER子函数

Matlab取整函数有: fix, floor, ceil, round.

具体应用方法如下:

fix朝零方向取整

如: $\text{fix}(-1.3)=-1$;

$\text{fix}(1.3)=1$;

floor, 顾名思义, 就是地板, 所以是取比它小的整数, 即朝负无穷方向取整

如: $\text{floor}(-1.3)=-2$;

$\text{floor}(1.3)=1$;

$\text{floor}(-1.8)=-2$;

$\text{floor}(1.8)=1$



步骤三：编写BER子函数

Matlab取整函数有: fix, floor, ceil, round.

具体应用方法如下:

ceil, 与floor相反, 它的意思是天花板, 也就是取比它大的最小整数, 即朝正无穷方向取整

如 $\text{ceil}(-1.3)=-1$;

$\text{ceil}(1.3)=2$;

$\text{ceil}(-1.8)=-1$;

$\text{ceil}(1.8)=2$

round四舍五入到最近的整数

如 $\text{round}(-1.3)=-1$; $\text{round}(-1.52)=2$;

$\text{round}(1.3)=1$; $\text{round}(1.52)=2$

步骤三：编写BER子函数

```
function [BER, Errors]=MCBPSKrun(N, EbNo, delay, FilterSwitch)
SamplesPerSymbol = 10;           % samples per symbol
BlockSize = 1000;                % block size
NoiseSigma = sqrt(SamplesPerSymbol/(2*EbNo)); % scale noise level
DetectedSymbols = zeros(1, BlockSize); % initialize vector
NumberOfBlocks = floor(N/BlockSize); % number of blocks processed
[B1x, A1x] = butter(5, 2/SamplesPerSymbol); % compute filter parameters
[IxOutput, IxFilterState] = filter(B1x, A1x, 0); % initialize state vector
BRx = ones(1, SamplesPerSymbol); ARx=1; % matched filter parameters
Errors = 0; % initialize error counter
```

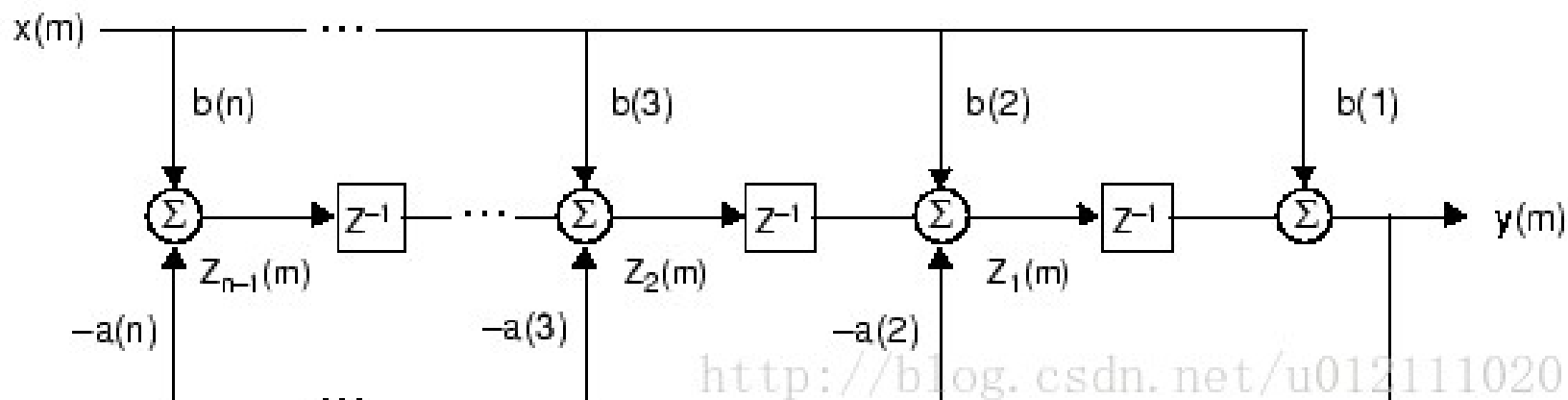
butter 产生滤波器系数

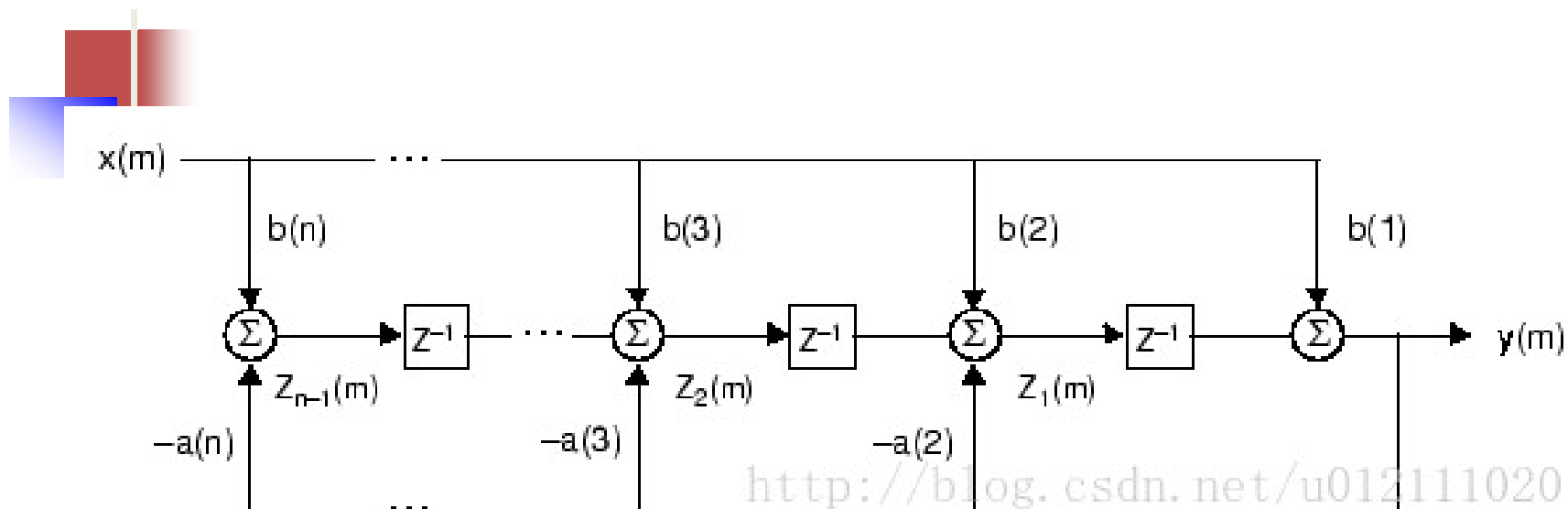
filter 数据过滤波器后的输出

BRx, ARx 匹配滤波器参数

步骤三：编写BER子函数

滤波器是组成通信系统的许多子系统中的重要部分。这些滤波器中的许多是模拟的，为了便于仿真，必须将他们映射为合适的等价数字滤波器。





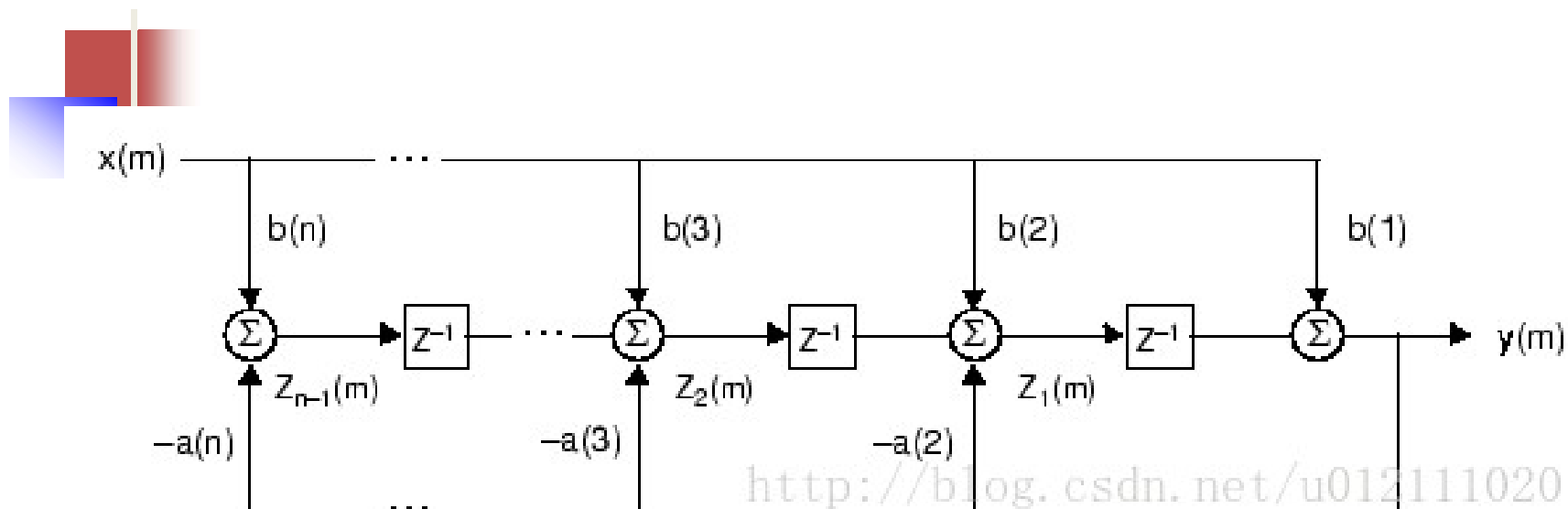
滤波器的系数向量 a （分母系数向量）和 b （分子系数向量）可以使用Matlab滤波器子程序，如：

butter 巴特沃思滤波器

cheby1 切比雪夫1型滤波器

elliptic 椭圆滤波器

确保存放系数 a 和 b 的向量保持相同长度，如果这两个向量长度不相等，较短的向量应该进行补领，使之与较长向量的长度相等



使用Matlab的一个方便之处在与：对于大量不同的模拟滤波器原型，都可以轻易的算出滤波器系数 a 和 b ，生成的这些滤波器系数通常会用于Matlab子程序filter（一个为分块处理开发的子程序）

$y = \text{filter}(b,a,x)$

作用：使用由分子和分母系数 b 和 a 定义的有理传递函数对输入数据 x 进行滤波。



步骤三：编写BER子函数

```
y = filter(b,a,x)
y = filter(b,a,x,zi)
y = filter(b,a,x,zi,dim)
[y,zf] = filter(____)
```

`y = filter(b,a,x)` 使用由分子和分母系数 `b` 和 `a` 定义的有理传递函数对输入数据 `x` 进行滤波。

如果 `a(1)` 不等于 1，则 `filter` 按 `a(1)` 对滤波器系数进行归一化。因此，`a(1)` 必须是非零值。

- 如果 `x` 为向量，则 `filter` 将滤波后数据以大小与 `x` 相同的向量形式返回。
- 如果 `x` 为矩阵，则 `filter` 沿着第一维度操作并返回每列的滤波后的数据。
- 如果 `x` 为多维数组，则 `filter` 沿大小不等于 1 的第一个数组维度进行计算。

`y = filter(b,a,x,zi)` 将初始条件 `zi` 用于滤波器延迟。`zi` 的长度必须等于 `max(length(a),length(b))-1`。

`y = filter(b,a,x,zi,dim)` 沿维度 `dim` 进行计算。例如，如果 `x` 为矩阵，则 `filter(b,a,x,zi,2)` 返回每行滤波后的数据。

`[y,zf] = filter(____)` 还使用任一上述语法返回滤波器延迟的最终条件 `zf`。



步骤三：编写BER子函数

例5：移动平均滤波器

移动平均滤波器是用于对噪声数据进行平滑处理的常用方法。此示例使用 `filter` 函数计算沿数据向量的平均值。

创建一个由正弦曲线数据组成的 1×100 行向量，其中的正弦曲线被随机干扰所损坏。

```
t = linspace(-pi,pi,100);  
rng default %initialize random number generator  
x = sin(t) + 0.25*rand(size(t));
```

(代码见c211.m)



步骤三：编写BER子函数

例5：移动平均滤波器

移动平均值滤波器沿数据移动长度为windowSize的窗口，并计算每个窗口中包含的数据的平均值。以下差分方程定义向量 x 的移动平均值滤波器：

$$y(n) = \frac{1}{\text{windowSize}} (x(n) + x(n-1) + \dots + x(n - (\text{windowSize} - 1))).$$

窗口大小为 5 时，计算有理传递函数的分子和分母系数。

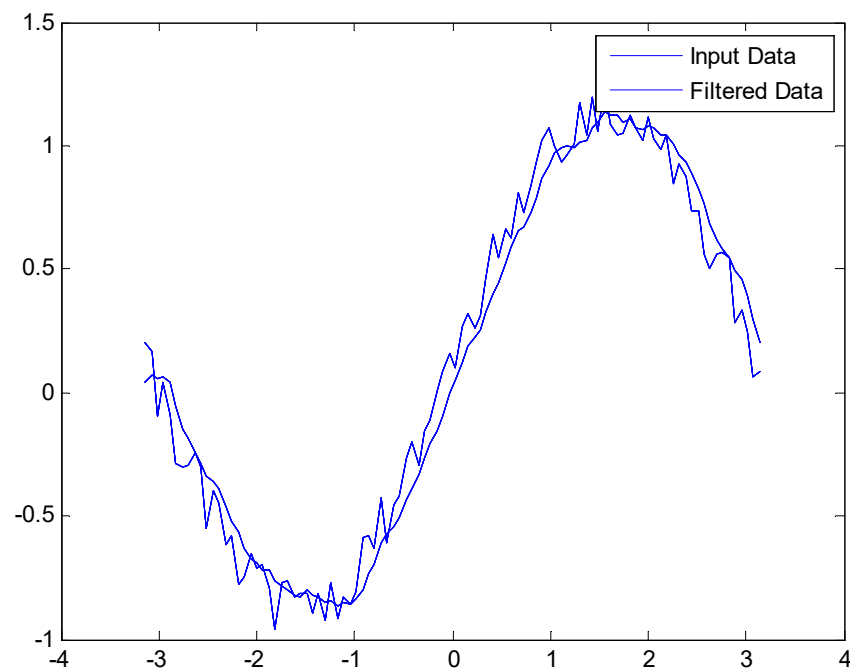
```
windowSize = 5;  
b = (1/windowSize)*ones(1,windowSize);  
a = 1;
```

步骤三：编写BER子函数

例5：移动平均滤波器

求数据的移动平均值，并绘制其对原始数据的图。

```
y = filter(b, a, x);  
plot(t, x)  
hold on  
plot(t, y)  
legend('Input Data', 'Filtered Data')
```





步骤三：编写BER子函数

例6：采用快处理和串行（逐个采样点）处理方法，来确定一个四阶巴特沃思滤波器的冲激响应。

(代码见c212.m)

```

n = 40;                % number of samples
order = 4;             % filter order
[b, a] = butter(order, 0.1); % prototype
%
% The following program segment is the block processing implementation.
%

```

```

in1 = [1, zeros(1, n-1)]; % input vector
out1 = filter(b, a, in1); % output vector
%

```

```

% The following program segment is the sample-by-sample implementation.
%

```

```

sreg = zeros(1, order+1); % initialize shift register

```

```

for k=1:n
    if k==1
        in=1; % impulse input
    else
        in=0;
    end
    out = b(1)*in + sreg(1,1); % determine output
    sreg = in*b - out*a + sreg; % update register
    sreg = [sreg(1, 2:(order+1)), 0]; % shift
    out2(k) = out; % create output vector

```

Sreg代表长度为order+1的移位寄存器，
order为滤波器阶数

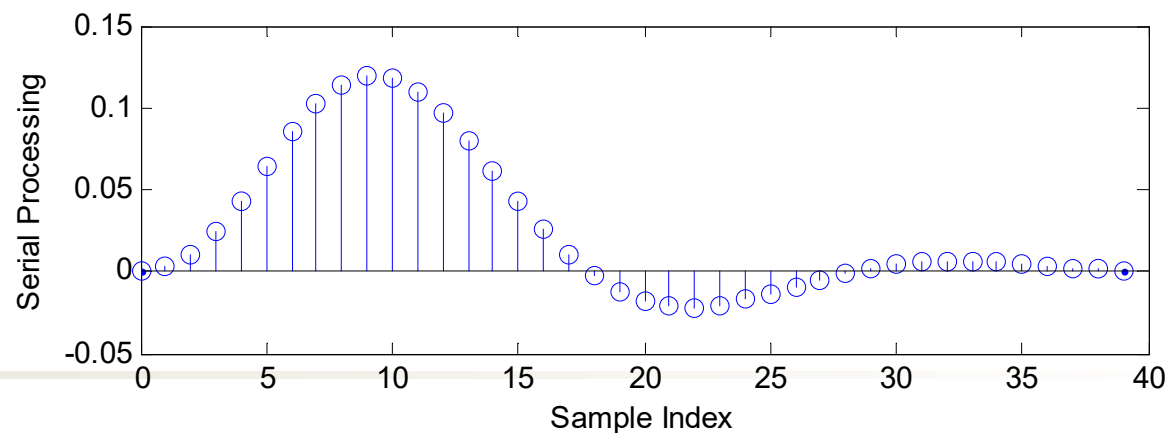
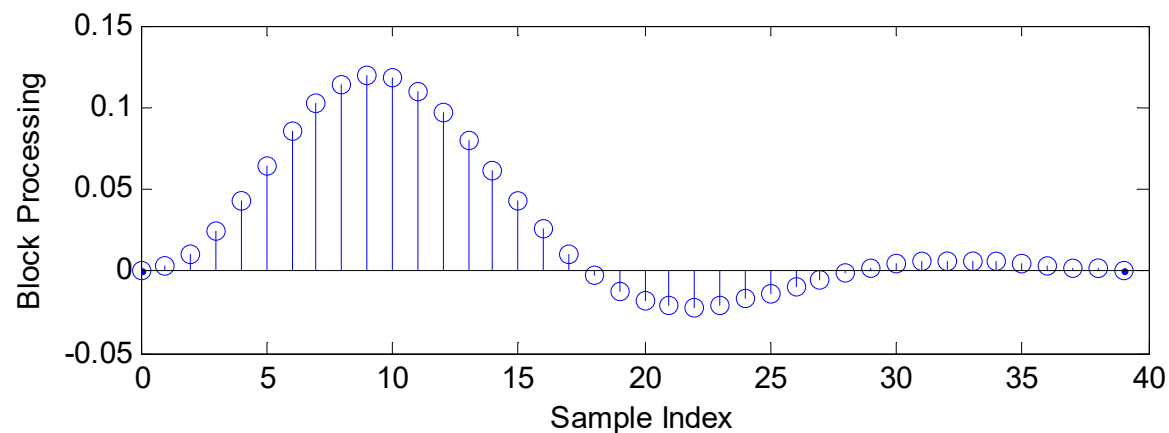
```

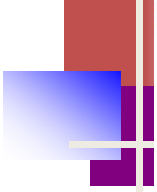
end

```

步骤三：编写BER子函数

例：采用快处理和串行（逐个采样点）处理方法，来确定一个四阶巴特沃思滤波器的冲激响应。





```
for Block=1:NumberOfBlocks
```

```
%
```

```
% Generate transmitted symbols.
```

```
%
```

```
[SymbolSamples, IxSymbols] = random_binary(BlockSize, SamplesPerSymbol);
```

```
%
```

```
% Transmitter filter if desired.
```

```
%
```

```
if FilterSwitch==0
```

```
    IxOutput = SymbolSamples;
```

```
else
```

```
    [IxOutput, IxFilterState] = filter(BIx, AIx, SymbolSamples, IxFilterState);
```

```
end
```

```
%
```

```
% Generate channel noise.
```

```
%
```

```
NoiseSamples = NoiseSigma*randn(size(IxOutput));
```

```
%
```

```
% Add signal and noise.
```

```
%
```

```
RxInput = IxOutput + NoiseSamples;
```

```
%
```

```
% Pass Received signal through matched filter.
```

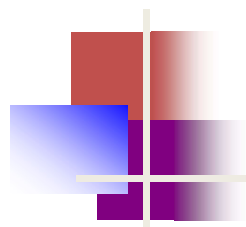
```
%
```

```
IntegratorOutput = filter(BRx, ARx, RxInput);
```

SamplesPerSymbol = 10

BlockSize = 1000

发送滤波器关闭，则可以
达到零ISI



```
%
% Sample matched filter output every SamplesPerSymbol samples,
% compare to transmitted bit, and count errors.
%
] for k=1:BlockSize,                               SamplesPerSymbol = 10
                                           BlockSize = 1000
    m = k*SamplesPerSymbol+delay;
    if (m < length(IntegratorOutput))
        DetectedSymbols(k) = (1-sign(IntegratorOutput(m)))/2;
        if (DetectedSymbols(k) ~= TxSymbols(k))
            Errors = Errors + 1;
        end
    end
end
- end
- end
- BER = Errors/(BlockSize*NumberOfBlocks);    % calculate BER
% End of function file.
```



步骤三：编写BER子函数

sign函数

判断数值的正负

```
>> sign(2)

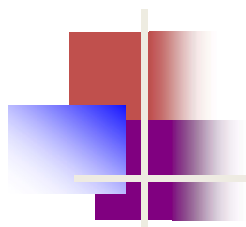
ans =

     1

>> sign(-2)

ans =

    -1
```



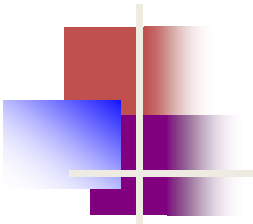
- 步骤一：理论推导
- 步骤二：确定时延值
- 步骤三：编写BER子函数
- 步骤四：编写主函数

步骤四：编写主函数

确定了delay值，完成子函数的编写，可以变下最后的主程序。

```
EbNodB = 0:8; % vector of Eb/No (dB) values
z = 10.^(EbNodB/10); % convert to linear scale
delay = 5; % enter delay value (samples)
BER = zeros(1,length(z)); % initialize BER vector
Errors = zeros(1,length(z)); % initialize Errors vector
BER_I = q(sqrt(2*z)); % theoretical (AWGN) BER vector
N = round(20./BER_I); % 20 errors for ideal (zero ISI) system
FilterSwitch = 1; % 1x filter out (0) or in (1)
for k=1:length(z)
    N(k) = max(1000,N(k)); % ensure at least one block processed
    [BER(k),Errors(k)] = c10_MCBPSKrun(N(k),z(k),delay,FilterSwitch)
end
semilogy(EbNodB,BER,'o',EbNodB,BER_I)
xlabel('E_b/N_0 - dB'); ylabel('Bit Error Rate'); grid
legend('System Under Study','AWGN Reference',0)
% End of script file.
```

(代码见c210.m)



当对一定取值范围内的多个 E_b/N_0 进行蒙特卡罗仿真时，如果对每一个 E_b/N_0 值都用同样的 N ，则基于误比特率估计的差错率会随着 E_b/N_0 的增加而减少，所以 E_b/N_0 越大时估计得误比特率越不可靠。

```
BER_I = q(sqrt(2*z));
```

```
N = round(20./BER_I);
```

把要处理的采样次数设为 K/P_T ，其中 P_T 是AWGN下的差错概率，可以部分的解决这个问题。



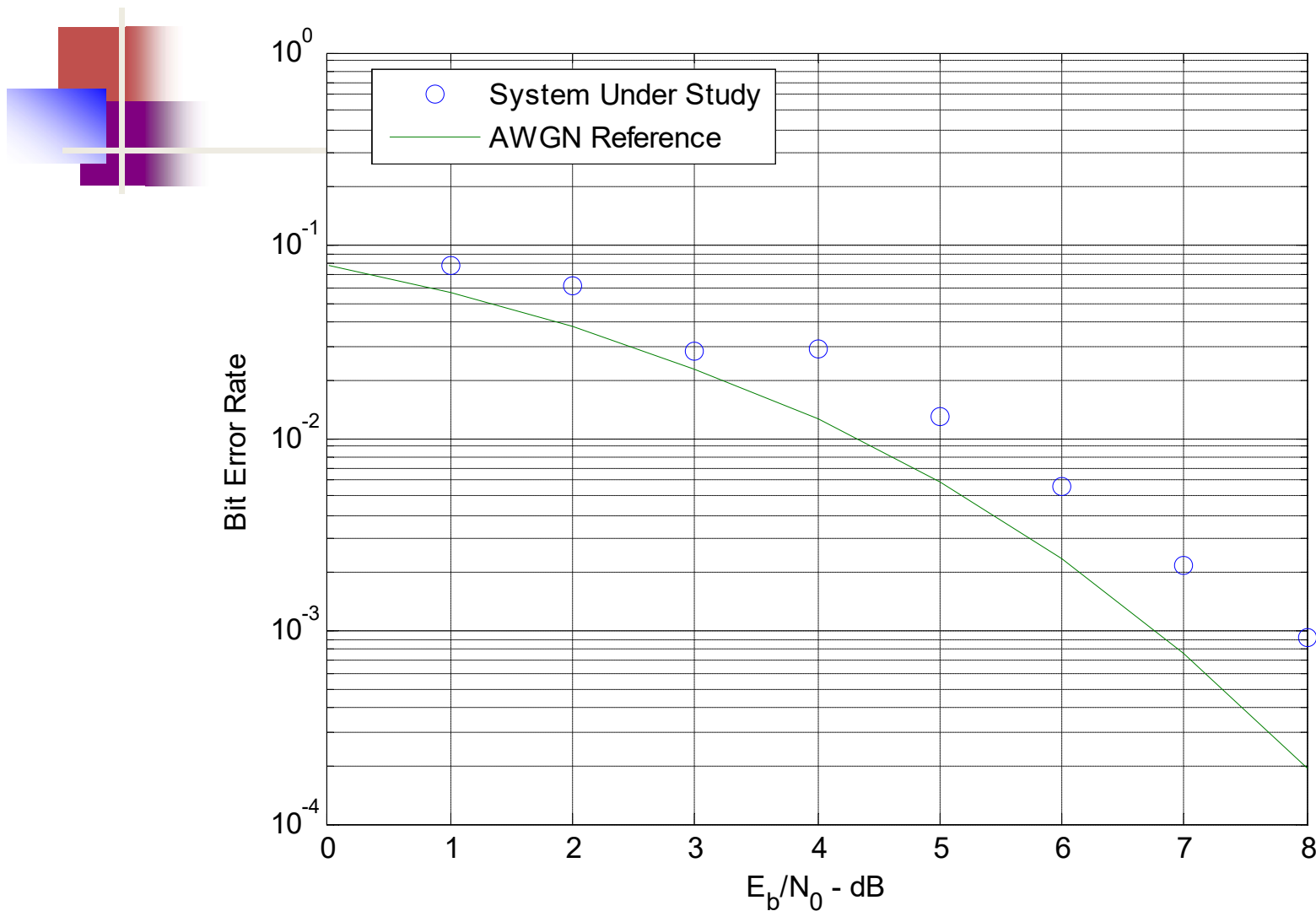
系统中存在一些损伤，如ISI和同步误差等，回到值仿真的估计值超过理论 P_T ，再一次仿真运行中光查到的差错次数通常会超过 K 。

```
BER_I = q(sqrt(2*z));
```

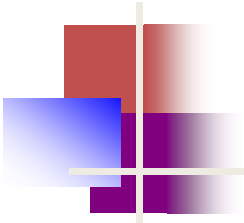
```
N = round(20./BER_I);
```

如果以目前的方式 $K=20$ 来确定 N ，则对充分小的 E_b/N_0 ， N 可能小于1000。由于仿真基于级联采样块的处理，每块由1000个符号组成（10000个采样），我们必须保证 $N > 1000$ ，从而保证仿真至少能处理一个完整的块。如果 $N < 1000$ ，会出现错误的结果。

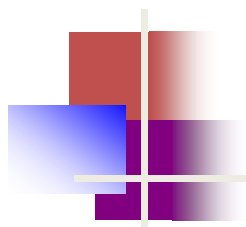
```
N(k) = max(1000, N(k));           % ensure at least one block processed
```



仿真结果用小圆圈表示，理想情况下（零ISI）的误比特率用实线表示。显然，滤波器引起的ISI增大了误比特率。



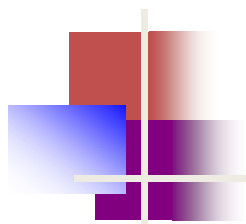
习题1: 使用每个符号20次采样的采样频率，重新运行上面BPSK调制的仿真，这样能否改进时延的估计？试对你的答案进行解释。用20次采样/符号的采样频率估计误比特率，并和上面例题的10次采样/符号的结果进行比较，作出结果曲线图并简要说明。



习题2: 假设 $x_d[n] = \cos(\frac{\pi}{6}), \quad d[n] = 0$

以及 $x_q[n] = \sin(\frac{\pi}{6}), \quad d[n] = 1$

仿真二进制PSK系统，并与例题的结果进行对比



习题3: 修改本章中BPSK调制的仿真，使PSK系统的仿真是逐个符号的方式而不是块级联的方式。也就是说，随机二进制数据源产生二进制比特（0或1），重复对应于这些二进制符号的波形样本，以满足给定的采样数/符号指标所需的次数。