哈尔滨工业大学（深圳）

通信系统仿真

实验报告

题　　目　蒙特卡罗系统仿真方法

专　　业　通信工程

学　　号　190210303

姓　　名　陈兴基

日　　期　2022/4/1

通信系统仿真实验任务书

姓　　名：陈兴基　　院　（系）：电子与信息工程学院

专　　业：通信工程　　班　　号：1902103

任务起至日期：　　2021　年 3 月 30 日至　2021　年 4 月 22 日

实验题目：蒙特卡罗系统仿真方法

实验目的：通过对蒙特卡罗系统仿真方法的学习和仿真实验，深入了解进行参数估计的蒙特卡罗方法的基础。掌握基本的参数估计方法，掌握多种通信系统的蒙特卡罗仿真。

实验内容：

实验 1-1：绘制对如下星座图的二进制 PSK 系统进行仿真，绘制 BER 曲线。

$$d[n]=0: x_d[n]=\cos(\frac{\pi}{6}), x_q[n]=0$$

$$d[n]=1: x_d[n]=0, x_q[n]=\cos(\frac{\pi}{6})$$

实验 1-2：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/符号或更高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。

（参考 c208、c209、c210 程序）

实验 1-3：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。

可以根据 c212 给出的方法，上述要求可以用两 种不同的方法满足：一种是将每个符号的多次采样 一起通过滤波器得到结果，即名义上的“逐个符号的方式”；另一种则是采用自己设定的移位寄存器 确定滤波器传递函数所需要的分子和分母多项式，不使用现成的滤波器函数，进行逐个采样样本方式的仿真。

实验 1-4：例 7 考察了差分 QPSK 系统的蒙特卡罗仿真，重新编写 QPSK 系统而不是差分 QPSK 系统的仿真程序。

（1）与差分 QPSK 系统从多个方面（例如 BER，符号同步误差，相位同步

误差和抖动等方面）进行对比

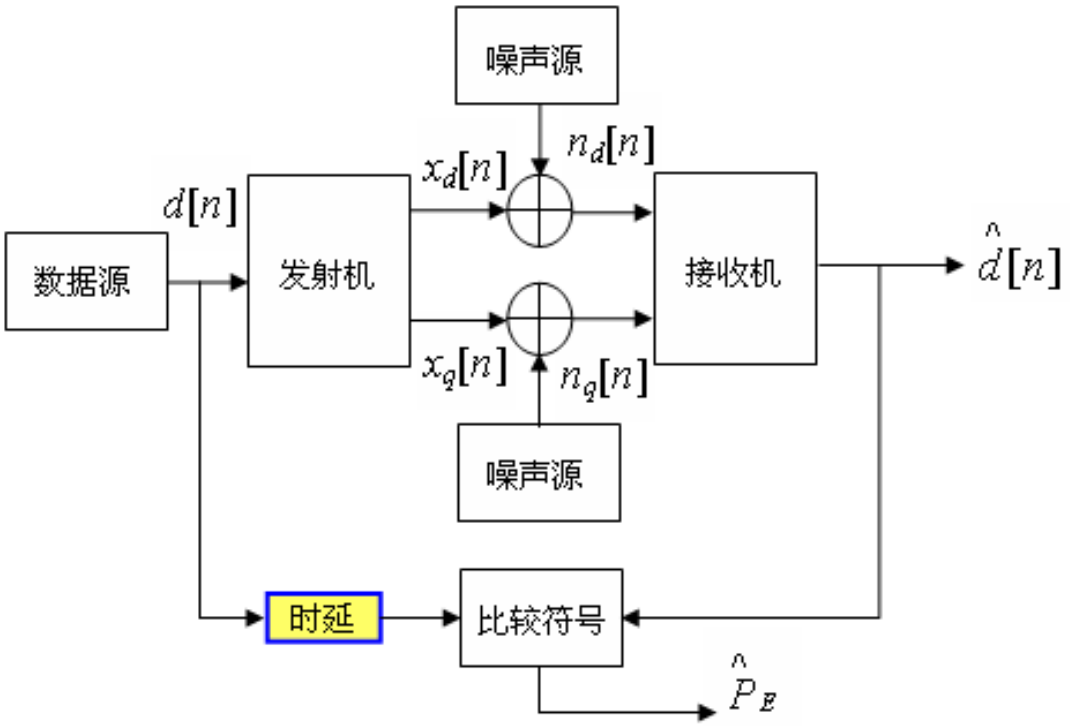
（2）比较有无 ISI 的情况

（3）比较相移在 0°到 360°的范围内，两种系统的抗相位同步误差性能

实验 1-1

1. 基本原理

简单通信系统的仿真模型：



调制器输出端的带通信号

$$x(t,n) = A_c \cos[2\pi f_c t + k_m d[n] + \theta]$$

其中 Ac 表示载波副值，K_m 是跟调制有关的常数，的 d[n]是第 n 个数据符号(d[n] =0 或 1)，可知 x(t,n)的复包络只是符号序数 n 的函数，表示为

$$\tilde{x}[n] = A_c \exp[k_m d[n] + \theta]$$

同相和正交分量（θ=0）

$$x_d[n] = A_c \cos(k_m d[n])$$

$$x_q[n] = A_c \sin(k_m d[n])$$

确定以 Eb/N0 为函数的 BER

保持 Eb 恒定，让噪声功率在感兴趣的范围内递增，噪声方差和噪声功率谱密度 PSD 的关系是

$$\sigma_n^2 = \frac{N_0 f_s}{2}$$

$$N_0 = \sigma_n^2 \frac{f_s}{2}$$

fs 为采样频率，且能量 Eb 和采样频率都归一化为 1。

$$SNR = \frac{E_b}{N_0} = \frac{f_s}{2}\frac{E_b}{\sigma_n^2}$$

$$\sigma_n = \sqrt{\frac{1}{2SNR}}$$

确定任意的功率谱密度或自相关函数

要确定一个随机数序列，使之具有给定自相关函数或等价地具有给定的功率谱密度，一个通用的方法是，对一组不相关的样本进行适当的滤波，从而使之具有目标功率谱密度。根据定义，不相关样本的功率谱密度在仿真带宽<fs/2 上是常数，而方差就是 $S_n(f)$曲线下的面积

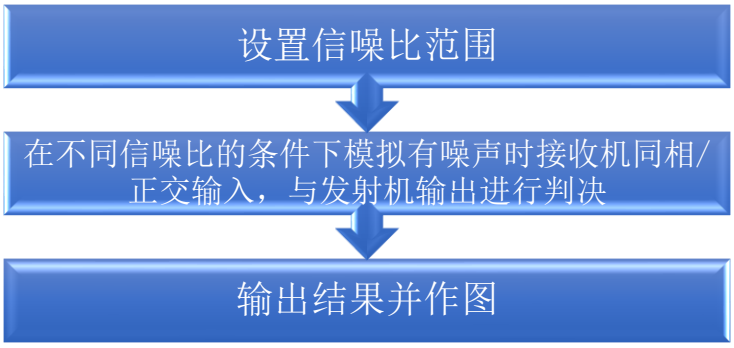
因此，为了产生功率谱密度为 N_0 的噪声，随机数(噪声)发生器的方差和采样频率必须满足

$$f_s = \frac{2\sigma_n^{\ 2}}{N_0}$$

可见，给定功率谱密度所要求的噪声发生器的方差是采样频率的函数。

2. 仿真实验设计

总体设计流程：



3. 仿真实验实现

1、设置信噪比范围

```
clear all
snrdB_min = -3; snrdB_max = 8;       % SNR (in dB) limits
snrdB = snrdB_min:1:snrdB_max;
Nsymbols = input('Enter number of symbols > ');
```

```
snr = 10.^(snrdB/10);                % convert from dB
h = waitbar(0,'SNR Iteration');
len_snr = length(snrdB);
```

2、在不同信噪比条件下模拟有噪声条件下的接收机同相/正交输入，将其与发射机输出进行对比判决

```
for j=1:len_snr                      % increment SNR
   waitbar(j/len_snr)
   sigma = sqrt(1/(2*snr(j)));       % noise standard deviation
   error_count = 0;
   for k=1:Nsymbols                  % simulation loop begins
      d = round(rand(1));            % data
      if d == 0
          x_d = sqrt(3)/2;           % direct transmitter output
          x_q = 0;                   % quadrature transmitter output
      else
          x_d = 0;                   % direct transmitter output
          x_q = 1/2;                 % quadrature transmitter output
      end
      n_d = sigma*randn(1);          % direct noise
      n_q = sigma*randn(1);          % quadrature noise
      y_d = x_d + n_d;               % direct receiver input
      y_q = x_q + n_q;               % quadrature receiver input
      if y_q < sqrt(3)*y_d-0.5       % test condition
         d_est = 0;                  % conditional data estimate
      else
         d_est = 1;                  % conditional data estimate
      end
      if (d_est ~= d)
         error_count = error_count + 1; % error counter
      end
   end                                   % simulation loop ends
   errors(j) = error_count;              % store error count for plot
end
```

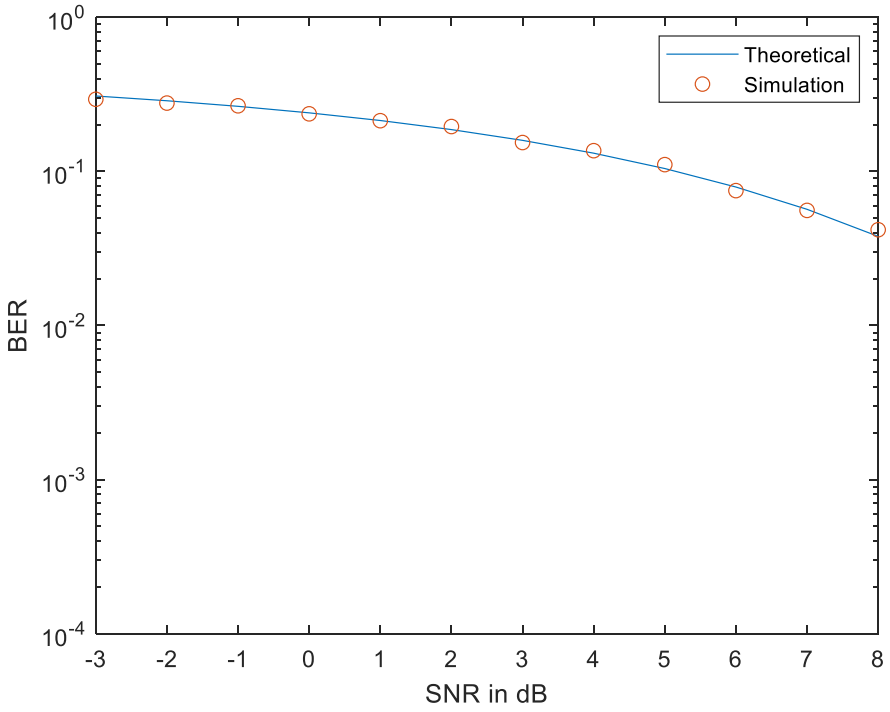
3、输出结果并作图

```
close(h)
ber_sim = errors/Nsymbols;           % BER estimate
```

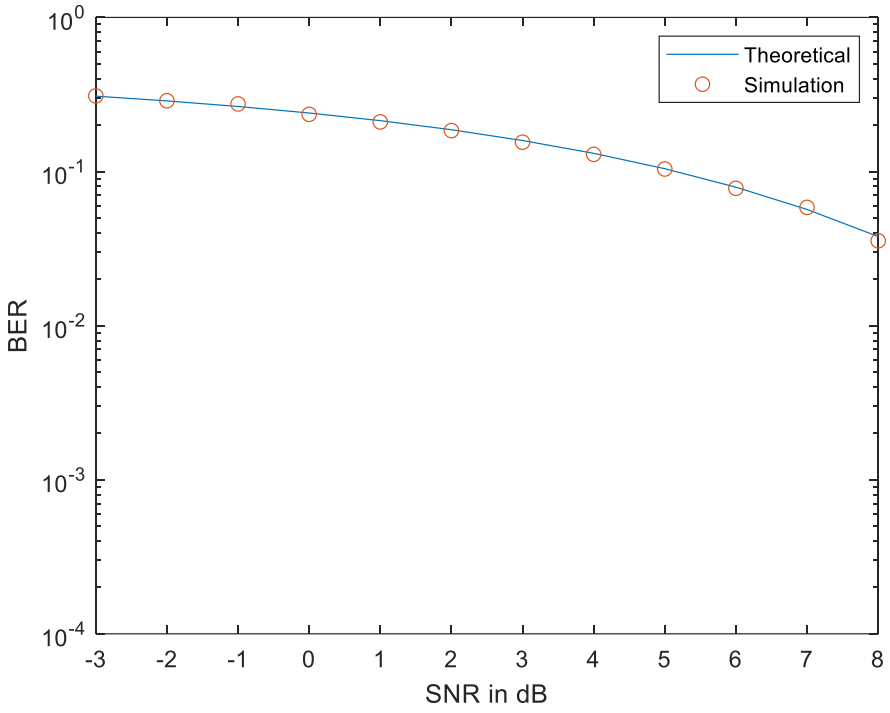
```
ber_theor = qfunc(sqrt(snr/2));          % theoretical BER
semilogy(snrdB,ber_theor,snrdB,ber_sim,'o')
axis([snrdB_min snrdB_max 0.0001 1])
xlabel('SNR in dB')
ylabel('BER')
legend('Theoretical','Simulation')
```

4. 仿真实验结论

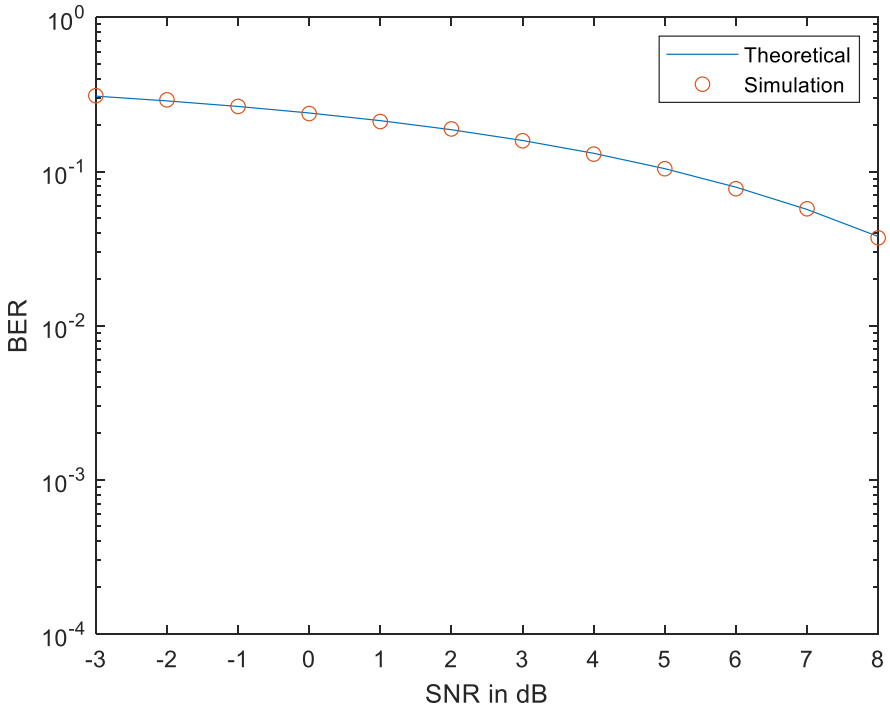
- 5000 symbols



- 10000 symbols



- 50000 symbols

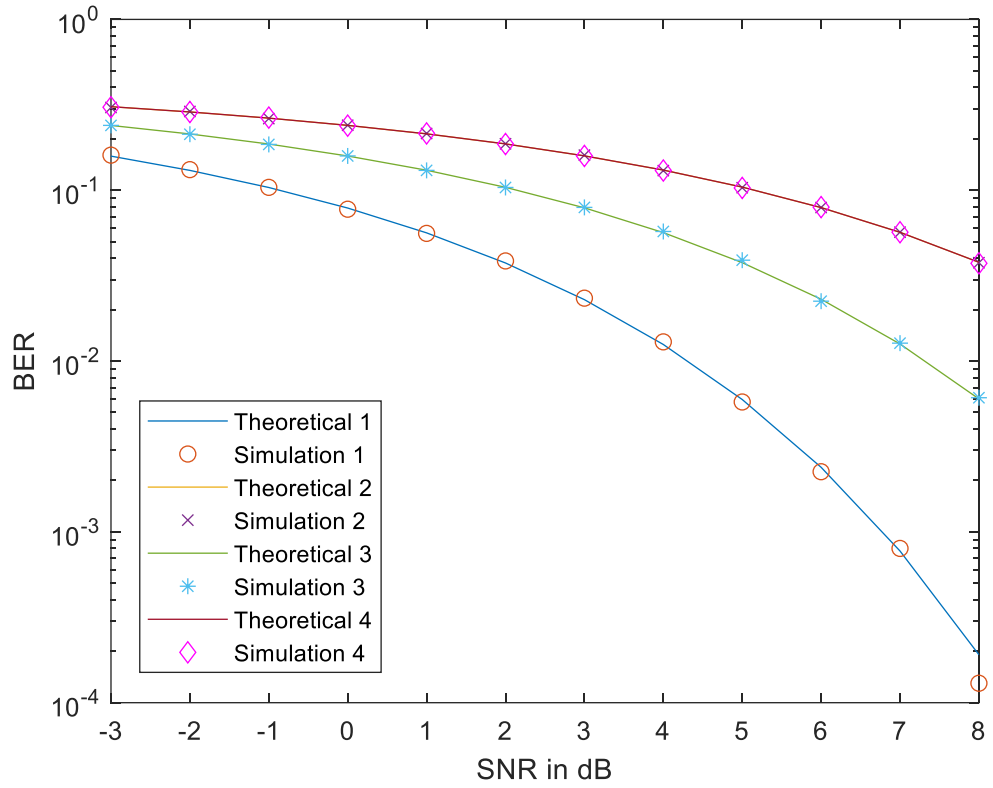


从图中可以看出，仿真结果与理论结果比较吻合。

同时，增大样本数可以增加估计的可靠性，即反映出蒙特卡罗方法的一致性与无偏性。

5. 拓展

- 四种调制方法仿真



- 代码

```
clear all

snrdB_min = -3; snrdB_max = 8;          % SNR (in dB) limits
snrdB = snrdB_min:1:snrdB_max;
Nsymbols = input('Enter number of symbols > ');
snr = 10.^(snrdB/10);                   % convert from dB
len_snr = length(snrdB);

for j1=1:len_snr                        % increment SNR
   sigma1 = sqrt(1/(2*snr(j1)));        % noise standard deviation
   error_count1 = 0;
   for k1=1:Nsymbols                    % simulation loop begins
      d1 = round(rand(1));              % data
      x_d1 = 2*d1 - 1;                  % transmitter output
      n_d1 = sigma1*randn(1);           % noise
```

```
      y_d1 = x_d1 + n_d1;                % receiver input
      if y_d1 > 0                        % test condition
         d_est1 = 1;                     % conditional data estimate
      else
         d_est1 = 0;                     % conditional data estimate
      end
      if (d_est1 ~= d1)
         error_count1 = error_count1 + 1;   % error counter
      end
   end                                      % simulation loop ends
   errors1(j1) = error_count1;              % store error count for
plot
end
ber_sim1 = errors1/Nsymbols;                % BER estimate
ber_theor1 = qfunc(sqrt(2*snr));            % theoretical BER
semilogy(snrdB,ber_theor1,snrdB,ber_sim1,'o')
hold on
axis([snrdB_min snrdB_max 0.0001 1])
xlabel('SNR in dB')
ylabel('BER')

for j2=1:len_snr                            % increment SNR
   sigma2 = sqrt(1/(2*snr(j2)));            % noise standard deviation
   error_count2 = 0;
   for k2=1:Nsymbols                        % simulation loop begins
      d2 = round(rand(1));                  % data
      x_d2 = d2;                            % transmitter output
      n_d2 = sigma2*randn(1);               % noise
      y_d2 = x_d2 + n_d2;                   % receiver input
      if y_d2 > 0.5                         % test condition
         d_est2 = 1;                        % conditional data estimate
      else
         d_est2 = 0;                        % conditional data estimate
      end
      if (d_est2 ~= d2)
         error_count2 = error_count2 + 1;   % error counter
      end
   end                                      % simulation loop ends
   errors2(j2) = error_count2;              % store error count for
plot
end
```

```
ber_sim2 = errors2/Nsymbols;                % BER estimate
ber_theor2 = qfunc(sqrt(snr/2));            % theoretical BER
semilogy(snrdB,ber_theor2,snrdB,ber_sim2,'x')
hold on

for j3=1:len_snr                            % increment SNR
   sigma3 = sqrt(1/(2*snr(j3)));            % noise standard
deviation
   error_count3 = 0;
   for k3=1:Nsymbols                        % simulation loop begins
      d3 = round(rand(1));                  % data
      if d3 ==0
         x_d3 = 1;                          % direct transmitter
output
         x_q3 = 0;                          % quadrature transmitter
output
      else
         x_d3 = 0;                          % direct transmitter
output
         x_q3 = 1;                          % quadrature transmitter
output
      end
      n_d3 = sigma3*randn(1);               % direct noise component
      n_q3 = sigma3*randn(1);               % quadrature noise
component
      y_d3 = x_d3 + n_d3;                   % direct receiver input
      y_q3 = x_q3 + n_q3;                   % quadrature receiver
input
      if y_d3 > y_q3                        % test condition
         d_est3 = 0;                        % conditional data
estimate
      else
         d_est3 = 1;                        % conditional data
estimate
      end
      if (d_est3 ~= d3)
         error_count3 = error_count3 + 1;   % error counter
      end
   end                                      % simulation loop ends
   errors3(j3) = error_count3;              % store error count for
plot
```

```
end
ber_sim3 = errors3/Nsymbols;                % BER estimate
ber_theor3 = qfunc(sqrt(snr));              % theoretical BER
semilogy(snrdB,ber_theor3,snrdB,ber_sim3,'*')
hold on

for j4=1:len_snr                            % increment SNR
   sigma4 = sqrt(1/(2*snr(j4)));            % noise standard deviation
   error_count4 = 0;
   for k4=1:Nsymbols                        % simulation loop begins
      d4 = round(rand(1));                  % data
      if d4 == 0
          x_d4 = sqrt(3)/2;                 % direct transmitter output
          x_q4 = 0;                         % quadrature transmitter
output
      else
          x_d4 = 0;                         % direct transmitter output
          x_q4 = 1/2;                       % quadrature transmitter
output
      end
      n_d4 = sigma4*randn(1);               % direct noise
      n_q4 = sigma4*randn(1);               % quadrature noise
      y_d4 = x_d4 + n_d4;                   % direct receiver input
      y_q4 = x_q4 + n_q4;                   % quadrature receiver input
      if y_q4 < sqrt(3)*y_d4-0.5            % test condition
         d_est4 = 0;                        % conditional data estimate
      else
         d_est4 = 1;                        % conditional data estimate
      end
      if (d_est4 ~= d4)
         error_count4 = error_count4 + 1;   % error counter
      end
   end                                      % simulation loop ends
   errors4(j4) = error_count4;              % store error count for
plot
end

ber_sim4 = errors4/Nsymbols;                % BER estimate
ber_theor4 = qfunc(sqrt(snr/2));            % theoretical BER
semilogy(snrdB,ber_theor4,snrdB,ber_sim4,'md')
legend('Theoretical 1','Simulation 1','Theoretical 2','Simulation
```

```
2','Theoretical 3','Simulation 3','Theoretical 4','Simulation 4')
```

实验 1-2

1. 基本原理

对于 AWGN 信道，通过采样积分-清除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]=1 条件下的条件差错概率为：

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

系统的差错概率估计：

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

假设我们想要求一个积分：

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

其中 g(x)为一个在积分区间内有界的函数，其总体均值为：

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

随机变量 X 的概率密度在区间 $(0, 1)$ 上为 1，其余地方为 0，U 是区间 $(0, 1)$ 上均匀分布的随机变量，有：

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

利用相对频率的观点，可得

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

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。

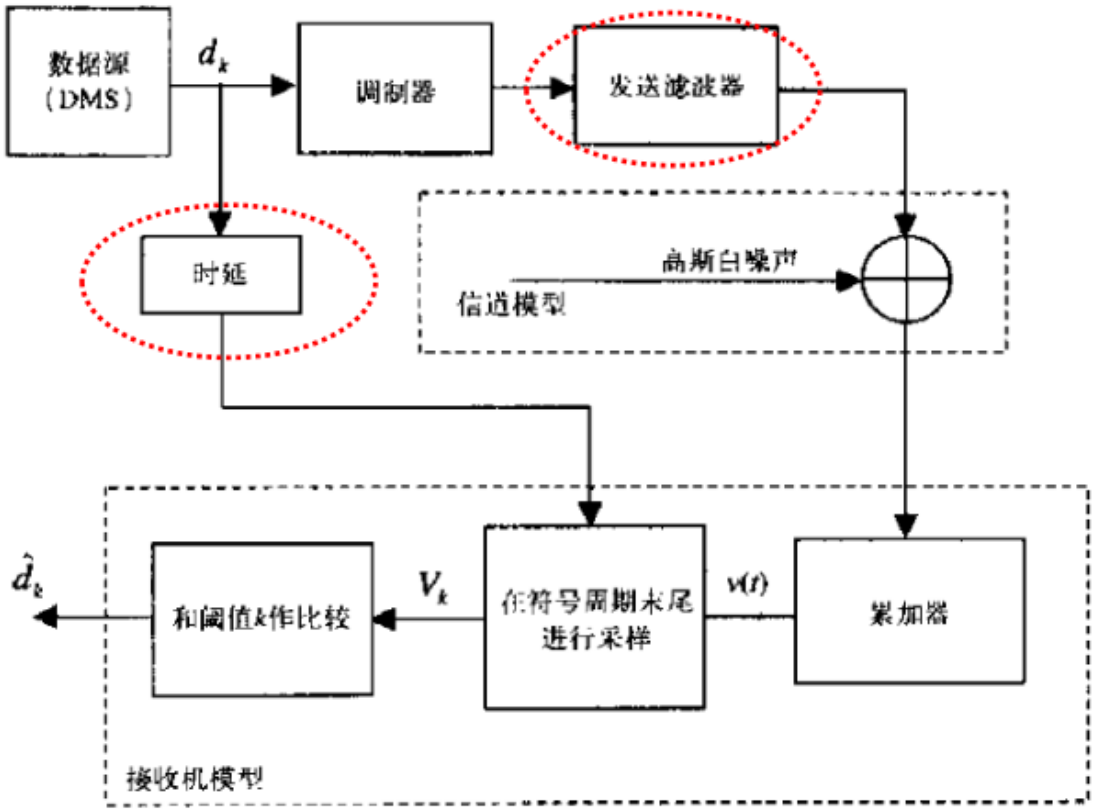
系统蒙特卡罗仿真基本上也是这个思路，因为通常无法获得充分统计量在差错区域内的解析表达式，所以采用系统仿真来产生统计量的采样。

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。

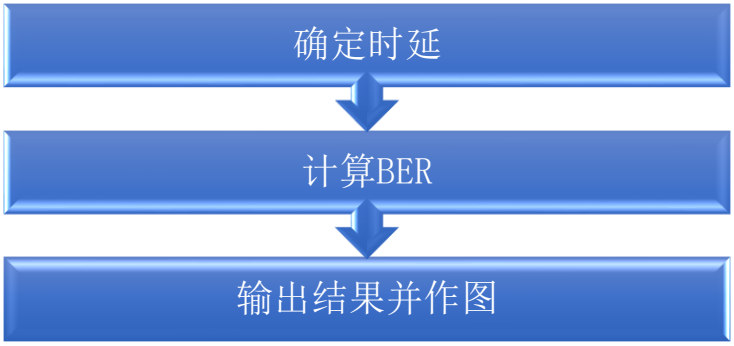
这种方法适合于任何常义积分。通过简单的变量代换，可用蒙特卡罗方法来估算任意区间上的积分。



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，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。

2. 仿真实验设计

总体设计流程：



- 确定时延

第一个问题就是要确定 delay 的值。

选定一个 Eb/N0 值，用不同的 delay 的值对系统进行仿真，并观察结果。

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 0.1Ts，其中 Ts 为符号周期。

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_T 是 AWGN 情况下的理论差错概率。

对于一个给定的 Eb/N0，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 Eb/N0=6dB 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。

- 计算 BER

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 Block Size 作为输入参数带入子函数 random_binary。

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。

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零，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：y=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。

3. 仿真实验实现

1、函数：确定时延

```
EbNodB = 6;                                    % Eb/No (dB) value
z = 10.^(EbNodB/10);                           % convert to linear
scale
delay = 0:19;                                  % 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');
```

2、函数：计算 BER

```
function [BER,Errors]=MCBPSKrun(N,EbNo,delay,FilterSwitch)
SamplesPerSymbol = 20;                         % 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
[BTx,ATx] = butter(5,2/SamplesPerSymbol);      % compute filter
```

```
parameters
[TxOutput,TxFilterState] = filter(BTx,ATx,0);    % initialize state
vector
BRx = ones(1,SamplesPerSymbol); ARx=1;           % matched filter
parameters
Errors = 0;                                      % initialize error
counter
%
% Simulation loop begine here.
%
for Block=1:NumberOfBlocks
     %
     % Generate transmitted symbols.
     %
     [SymbolSamples,TxSymbols] =
random_binary(BlockSize,SamplesPerSymbol);
     %
     % Transmitter filter if desired.
     %
     if FilterSwitch==0
        TxOutput = SymbolSamples;
     else
        [TxOutput,TxFilterState] =
filter(BTx,ATx,SymbolSamples,TxFilterState);
     end
     %
     % Generate channel noise.
     %
     NoiseSamples = NoiseSigma*randn(size(TxOutput));
     %
     % Add signal and noise.
     %
     RxInput = TxOutput + NoiseSamples;
     %
     % Pass Received signal through matched filter.
     %
     IntegratorOutput = filter(BRx,ARx,RxInput);
     %
     % Sample matched filter output every SamplesPerSymbol samples,
     % compare to transmitted bit, and count errors.
     %
```

```
    for k=1:BlockSize,
        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
BER = Errors/(BlockSize*NumberOfBlocks);    % calculate BER
```

3、主函数

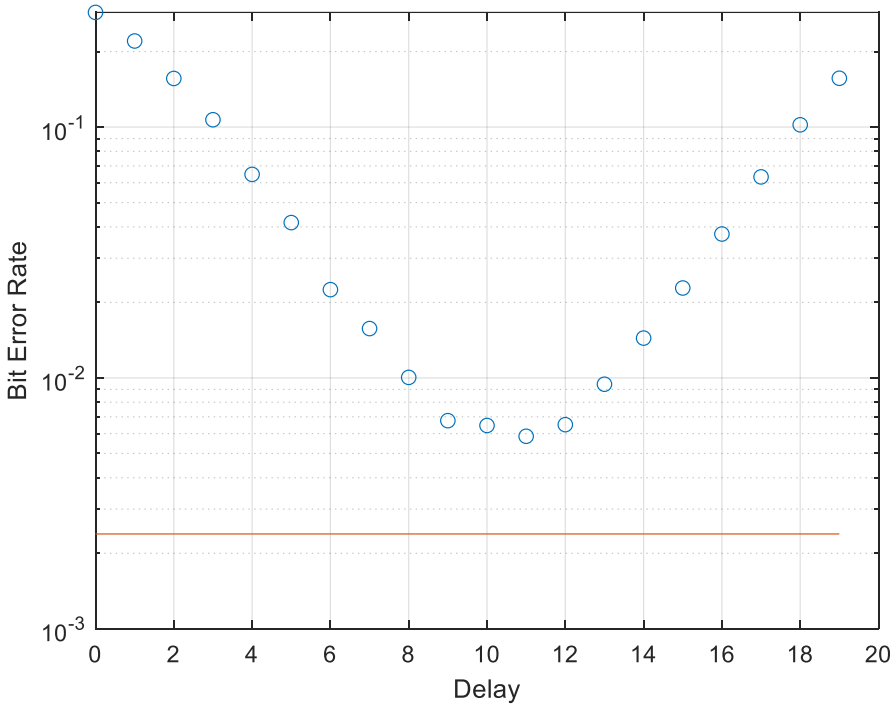
```
EbNodB = 0:8;                       % vector of Eb/No (dB) values
z = 10.^(EbNodB/10);                % convert to linear scale
delay = 11;                         % enter delay value (samples)
BER = zeros(1,length(z));           % initialize BER vector
Errors = zeros(1,length(z));        % initialize Errors vector
BER_T = q(sqrt(2*z));               % theoretical (AWGN) BER vector
N = round(20./BER_T);               % 20 errors for ideal (zero ISI)
system
FilterSwitch = 1;                   % Tx 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)] = c209_MCBPSKrun(N(k),z(k),delay,FilterSwitch)
end
semilogy(EbNodB,BER,'o',EbNodB,BER_T)
xlabel('E_b/N_0 - dB'); ylabel('Bit Error Rate'); grid
legend('System Under Study','AWGN Reference')
```

4. 仿真实验结论

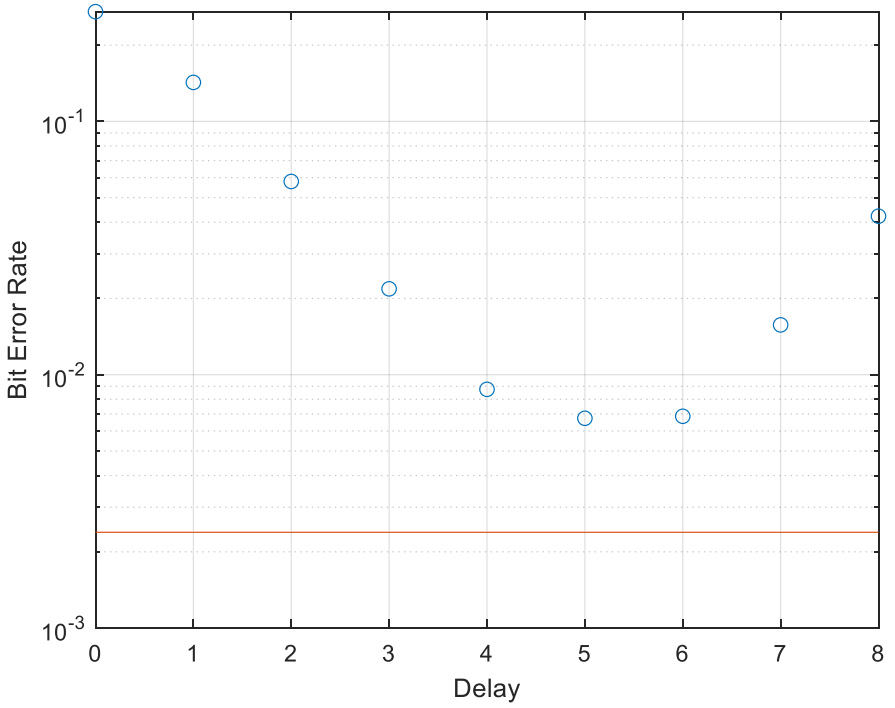
将采样频率提高至每个符号 20 次采样，发现并不能改进时延估计。而且比较 20 次采样/符号与 10 次采样/符号的误比特率，二者结果相差不大。

- 时延估计

20 次采样/符号：

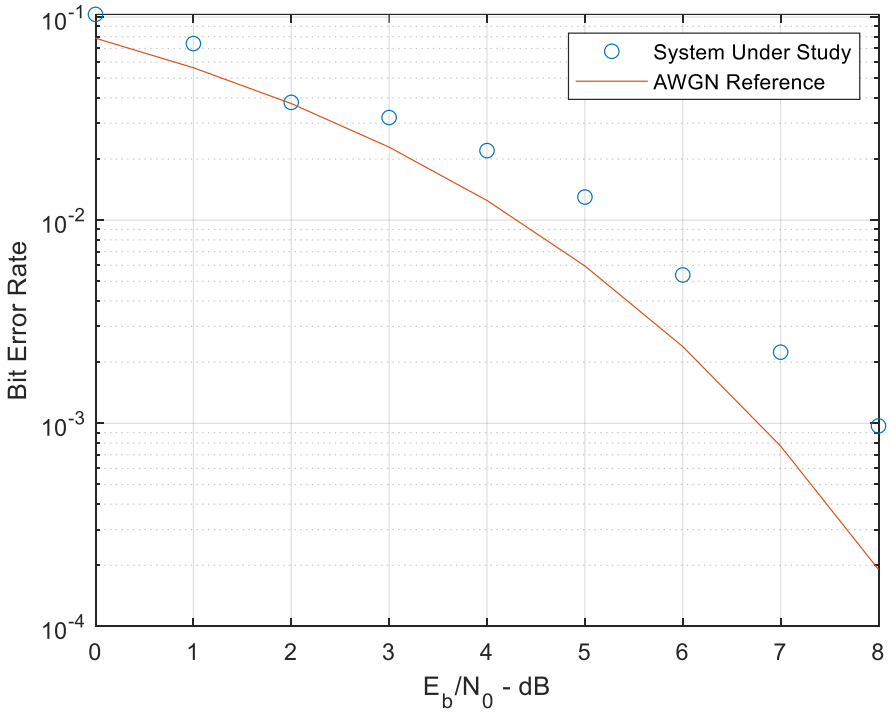


10 次采样/符号：

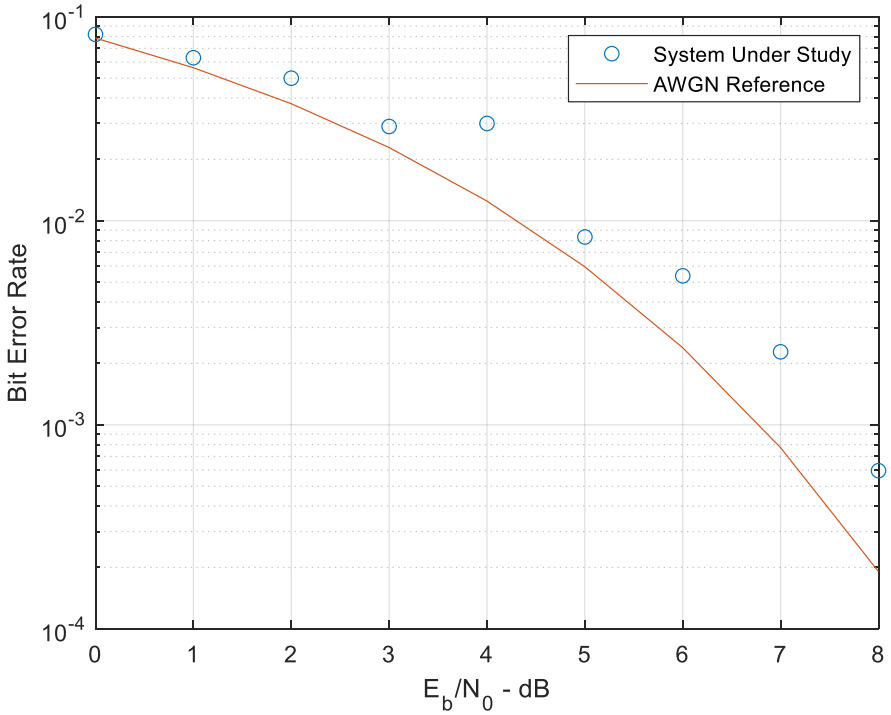


- 误码率

20 次采样/符号：



10 次采样/符号：



如图所示，不正确的选择时延会导致过大的误比特率，更高的采样频率能改进时延的估计。可以用更高的采样频率来确定更精确的 delay 的值。

同时，采用 20 次采样/符号确定时延，所得的结果较为准确。

实验 1-3

1. 基本原理

对于 AWGN 信道，通过采样积分-清除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]=1 条件下的条件差错概率为：

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

系统的差错概率估计：

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

假设我们想要求一个积分：

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

其中 g(x)为一个在积分区间内有界的函数，其总体均值为：

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

随机变量 X 的概率密度在区间 $(0, 1)$ 上为 1，其余地方为 0，U 是区间 $(0, 1)$ 上均匀分布的随机变量，有：

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

利用相对频率的观点，可得

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

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。

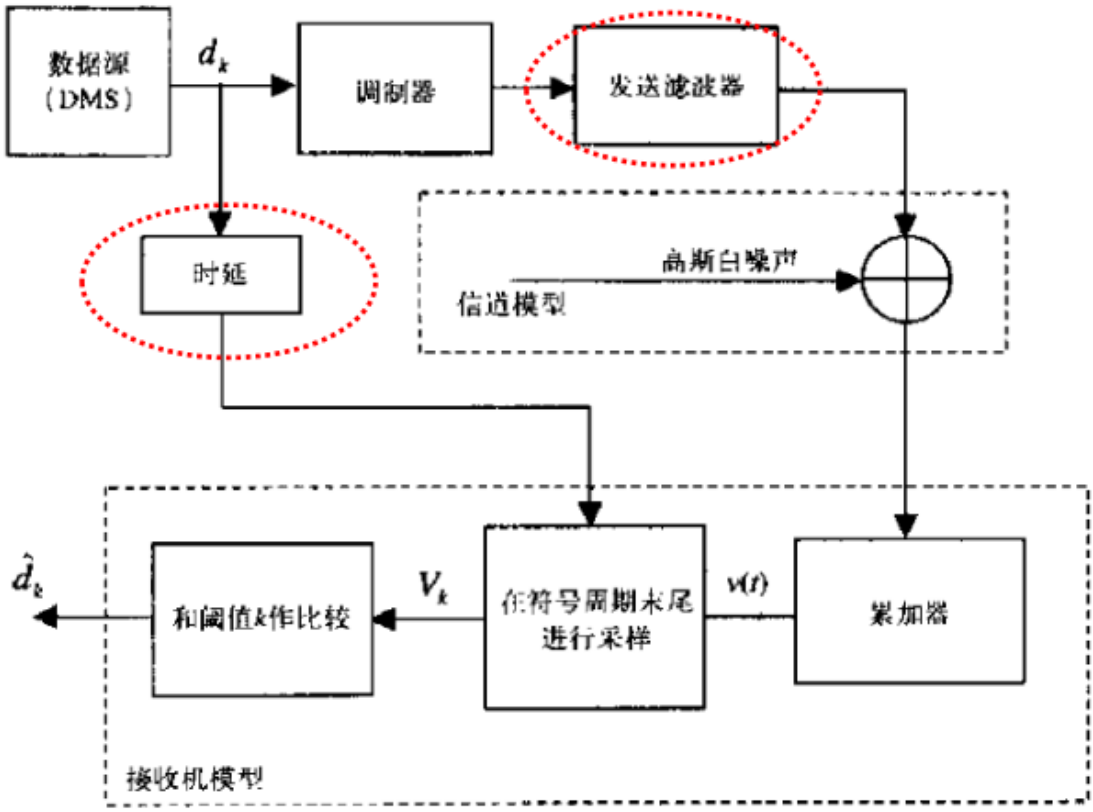
系统蒙特卡罗仿真基本上也是这个思路，因为通常无法获得充分统计量在差错区域内的解析表达式，所以采用系统仿真来产生统计量的采样。

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。

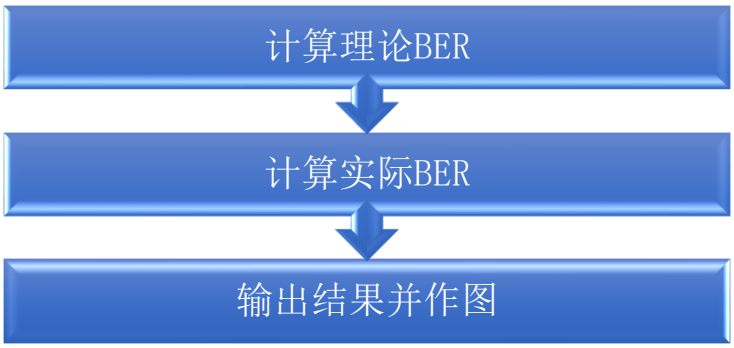
这种方法适合于任何常义积分。通过简单的变量代换，可用蒙特卡罗方法来估算任意区间上的积分。



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，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。

2. 仿真实验设计

总体设计流程：



- 计算 BER

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 Block Size 作为输入参数带入子函数 random_binary。

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。

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零，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：y=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。

3. 仿真实验实现

1、函数：计算实际 BER

```
function [BER,Errors]=MCBPSKrun(N,EbNo,delay,FilterSwitch)
SamplesPerSymbol = 10;
BlockSize = 1000;
NoiseSigma = sqrt(SamplesPerSymbol/(2*EbNo));
DetectedSymbols = zeros(1,BlockSize);
NumberOfBlocks = floor(N/BlockSize);
[BTx,ATx] = butter(5,2/SamplesPerSymbol);
[TxOutput,TxFilterState] = filter(BTx,ATx,0);
BRx = ones(1,SamplesPerSymbol); ARx=1;
Errors = 0;
for Block=1:NumberOfBlocks
    [SymbolSamples,TxSymbols]
```

```
=random_binary(BlockSize,SamplesPerSymbol);
    if FilterSwitch==0
        TxOutput = SymbolSamples;
    else
        [TxOutput,TxFilterState]
=filter(BTx,ATx,SymbolSamples,TxFilterState);
    end
    NoiseSamples = NoiseSigma*randn(size(TxOutput));
    RxInput = TxOutput + NoiseSamples;
    IntegratorOutput = filter(BRx,ARx,RxInput);
    for k=1:BlockSize,
        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
BER = Errors/(BlockSize*NumberOfBlocks);
```

2、函数：计算理论 BER

```
function y=q(x)
y = 0.5*erfc(x/sqrt(2));
```

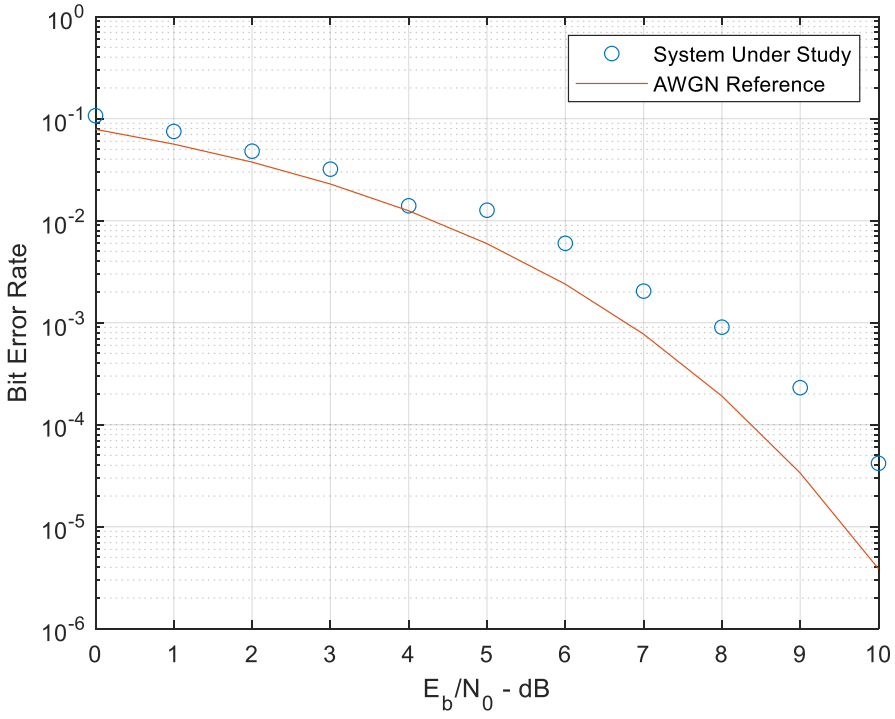
3、函数：随机生成信号

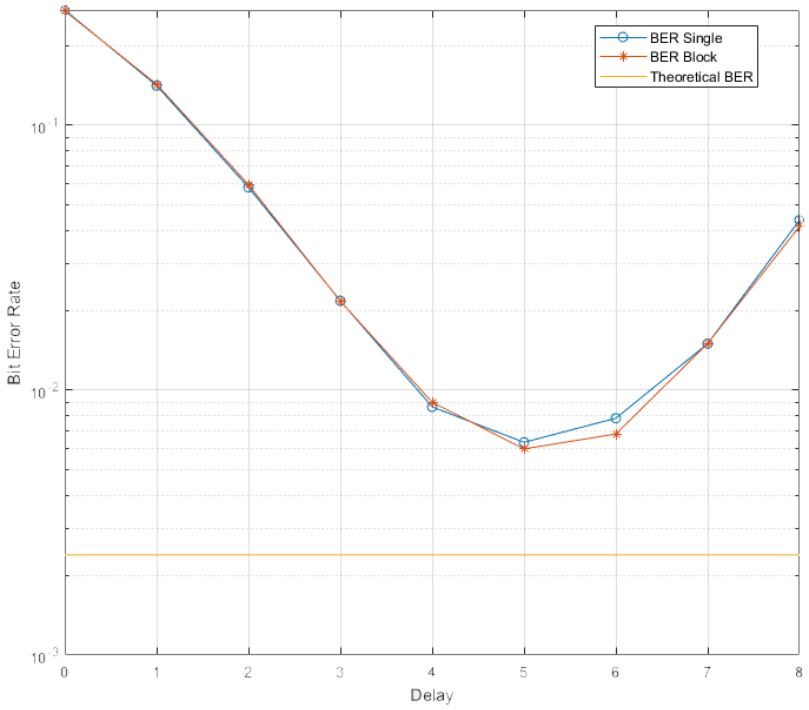
```
function [x, bits] = random_binary(nbits,nsamples)
x = zeros(1,nbits*nsamples);
bits = round(rand(1,nbits));
for m=1:nbits
    for n=1:nsamples
        index = (m-1)*nsamples + n;
        x(1,index) = (-1)^bits(m);
    end
end
```

4、主函数

```
EbNodB = 0:10;
z = 10.^(EbNodB/10);
delay = 5;
BER = zeros(1,length(z));
Errors = zeros(1,length(z));
BER_T = q(sqrt(2*z));
N = round(20./BER_T);
FilterSwitch = 1;
for k=1:length(z)
    N(k) = max(1000,N(k));
    [BER(k),Errors(k)] = MCBPSKrun(N(k),z(k),delay,FilterSwitch) ;
end
semilogy(EbNodB,BER,'o',EbNodB,BER_T) ;
xlabel('E_b/N_0 - dB'); ylabel('Bit Error Rate');
grid
legend('System Under Study','AWGN Reference') ;
```

4. 仿真实验结论




BER Single
BER Block
Theoretical BER
Bit Error Rate
Delay
10^-1
10^-2
10^-3
0
1
2
3
4
5
6
7
8

实验 1-4

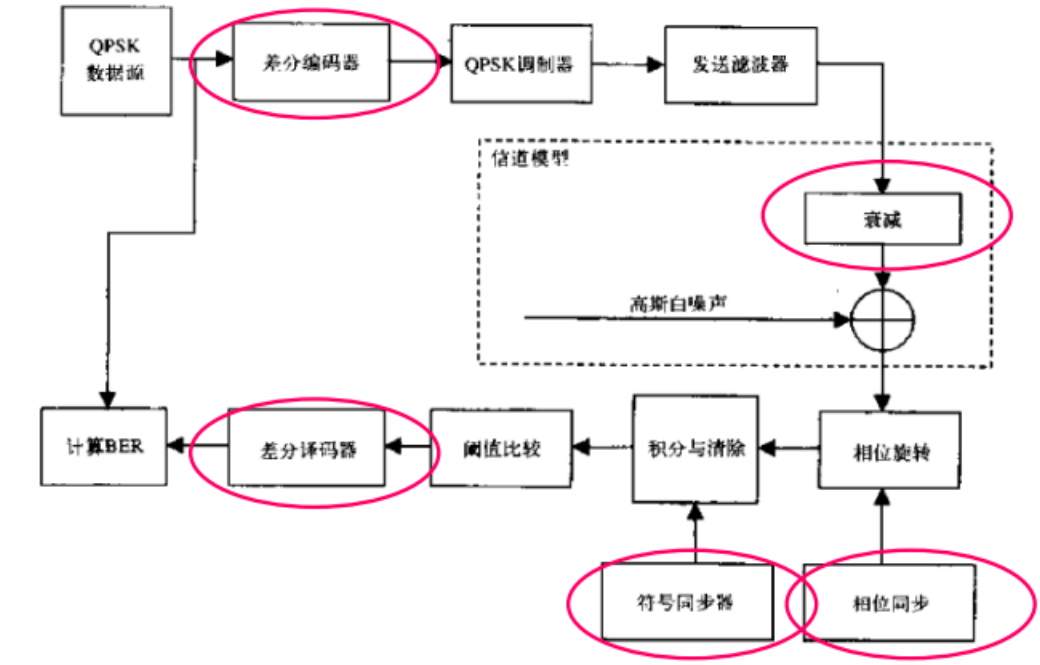
1. 基本原理

对于相干射频系统，接收机必须具备提供载波和符号同步的功能，但噪声和信道失真使接收机不可能完美地实现载波和符号同步。

不正确的载波同步将会导致发送信号相对于接收信号产生相位误差或相位旋转；不正确的符号同步会导致积分—清楚检测器在不正确的时间区间上处理接收信号。

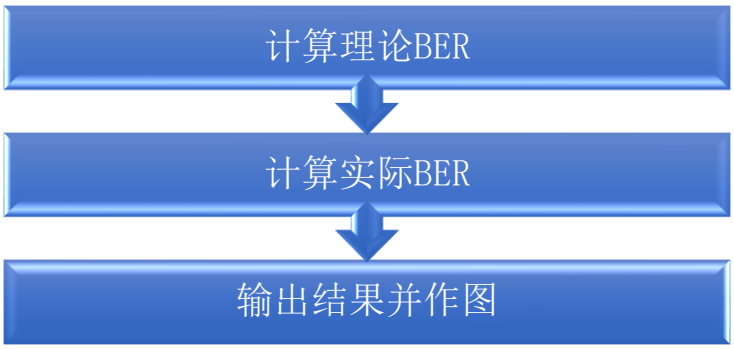
BPSK 和 QPSK 系统在解调时都存在相位模糊的问题。由于信道造成位置的信号时延，所以接收机不可能确定发送信号的绝对相位，从而发生相位模糊问题。

传统的解决这个问题的办法是不把信息位的编码包含在绝对相位中，而是包含在符号间的相位差中，实现差分编码 DQPSK。



2. 仿真实验设计

总体设计流程：



3. 仿真实验实现

1、函数：计算实际 BER

```
function BER_MC=MCQPSKrun(N,Eb,No,ChanAtt,...
    TimingBias,TimingJitter,PhaseBias,PhaseJitter)
fs = 1e+6;                                  % sampling Rate
(samples/second)
SymRate = 1e+5;                             % symbol rate
(symbols/second)
Ts = 1/fs;                                  % sampling period
TSym = 1/SymRate;                           % symbol period
SymToSend = N;                              % symbols to be transmitted
ChanBW = 4.99e+5;                           % bandwidth of channel (Hz)
MeanCarrierPhaseError = PhaseBias;          % mean of carrier phase
StdCarrierPhaseError = PhaseJitter;         % stdev of phese error
MeanSymbolSyncError = TimingBias;           % mean of symbol sync error
StdSymbolSyncError = TimingJitter;          % stdev of symbol sync error
ChanGain = 10^(-ChanAtt/20);                % channel gain (linear
units)
TxBitClock = Ts/2;                          % transmitter bit clock
RxBitClock = Ts/2;                          % reciever bit clock
%
%  Standard deviation of noise and signal amplitude at receiver
input.
%
RxNoiseStd = sqrt((10^((No-30)/10))*(fs/2));        % stdev of noise
TxSigAmp = sqrt(10^((Eb-30)/10)*SymRate);           % signal
amplitude
%
```

```
% Allocate some memory for probes.
%
SampPerSym = fs/SymRate;
probe1 = zeros((SymToSend+1)*SampPerSym,1);
probe1counter = 1;
probe2 = zeros((SymToSend+1)*SampPerSym,1);
probe2counter = 1;
%
% Counters to keep track of how many symbols have have been sent.
%
TxSymSent = 1;
RxSymDemod = 0;
%
% Buffers that contain the transmitted and received data.
%
[unused,SourceBitsI] = random_binary(SymToSend,1);
[unused,SourceBitsQ] = random_binary(SymToSend,1);
%
% Differentially encode the transmitted data.
%
TxBitsI = SourceBitsI;
TxBitsQ = SourceBitsQ;
% TxBitsI = SourceBitsI*0;
% TxBitsQ = SourceBitsQ*0;
% for k=2:length(TxBitsI)
%     TxBitsI(k) =
or(and(not(xor(SourceBitsI(k),SourceBitsQ(k))),...
%                   xor(SourceBitsI(k),TxBitsI(k-1))), ...
%                   and(xor(SourceBitsI(k),SourceBitsQ(k)),...
%                   xor(SourceBitsQ(k),TxBitsQ(k-1))));
%     TxBitsQ(k) =
or(and(not(xor(SourceBitsI(k),SourceBitsQ(k))),...
%                   xor(SourceBitsQ(k),TxBitsQ(k-1))), ...
%                   and(xor(SourceBitsI(k),SourceBitsQ(k)),...
%                   xor(SourceBitsI(k),TxBitsI(k-1))));
% end
%
% Make a complex data stream of the I and Q bits.
%
TxBits = ((TxBitsI*2)-1)+(sqrt(-1)*((TxBitsQ*2)-1));
%
```

```
RxIntegrator = 0;                    % initialize receiver integrator
TxBitClock = 2*TSym;                 % initialize transmitter
%
% Design the channel filter, and create the filter state array.
%
[b,a] = butter(2,ChanBW/(fs/2));
b=[1]; a=[1];                                % filter bypassed
[junk,FilterState]=filter(b,a,0);
%
% Begin simulation loop.
%
while TxSymSent < SymToSend
   % Update the transmitter's clock, and see
   % if it is time to get new data bits
   TxBitClock=TxBitClock+Ts;
   if TxBitClock > TSym
      % Time to get new bits
      TxSymSent=TxSymSent+1;
      % We don't want the clock to increase off
      % to infinity, so subtract off an integer number
      % of Tb seconds
      TxBitClock=mod(TxBitClock,TSym);
      % Get the new bit, and scale it up appropriately.
      TxOutput=TxBits(TxSymSent)*TxSigAmp;
   end
   %
   % Pass the transmitted signal through the channel filter.
   %
   [Rx,FilterState]=filter(b,a,TxOutput,FilterState);
   %
   % Add white Gaussian noise to the signal.
   %
   Rx=(ChanGain*Rx)+(RxNoiseStd*(randn(1,1)+sqrt(-1)*randn(1,1)));
   %
   % Phase rotation due to receiver carrier synchronization error.
   %
   PhaseRotation = exp(sqrt(-1)*2*pi*...

(MeanCarrierPhaseError+(randn(1,1)*StdCarrierPhaseError))/360);
   Rx=Rx*PhaseRotation;
   probe1(probe1counter)=Rx; probe1counter=probe1counter+1;
```

```
    %
    % Update the Integrate and Dump Filter at the receiver.
    %
    RxIntegrator = RxIntegrator+Rx;
    probe2(probe2counter) = RxIntegrator;
    probe2counter = probe2counter+1;
    %
    % Update the receiver clock, to see if it is time to
    % sample and dump the integrator.
    %
    RxBitClock = RxBitClock+Ts;
    RxTSym =
TSym*(1+MeanSymbolSyncError+(StdSymbolSyncError*randn(1,1)));
    if RxBitClock > RxTSym                    % time to demodulate
symbol
        RxSymDemod = RxSymDemod+1;
        RxBitsI(RxSymDemod) = round(sign(real(RxIntegrator))+1)/2;
        RxBitsQ(RxSymDemod) = round(sign(imag(RxIntegrator))+1)/2;
        RxBitClock = RxBitClock - TSym;       % reset receive clock
        RxIntegrator = 0;                     % reset integrator
    end
end
%
% Differential decoder.
%
SinkBitsI = RxBitsI;
SinkBitsQ = RxBitsQ;
% SinkBitsI = SourceBitsI*0;
% SinkBitsQ = SourceBitsQ*0;
% %
% for k=2:RxSymDemod
%     SinkBitsI(k) = or(and(not(xor(RxBitsI(k),RxBitsQ(k))),...
%                       xor(RxBitsI(k),RxBitsI(k-1))),...
%                       and(xor(RxBitsI(k),RxBitsQ(k)),...
%                       xor(RxBitsQ(k),RxBitsQ(k-1))));
%     SinkBitsQ(k) = or(and(not(xor(RxBitsI(k),RxBitsQ(k))),...
%                       xor(RxBitsQ(k),RxBitsQ(k-1))),...
%                       and(xor(RxBitsI(k),RxBitsQ(k)),...
%                       xor(RxBitsI(k),RxBitsI(k-1))));
% end
%
```

```
% Look for best time delay between input and output for 100 bits.
%
[C,Lags] = vxcorr(SourceBitsI(10:110),SinkBitsI(10:110));
[MaxC,LocMaxC] = max(C);
BestLag = Lags(LocMaxC);
%
% Adjust time delay to match best lag
%
if BestLag > 0
    SourceBitsI = SourceBitsI(BestLag+1:length(SourceBitsI));
    SourceBitsQ = SourceBitsQ(BestLag+1:length(SourceBitsQ));
elseif BestLag < 0
    SinkBitsI = SinkBitsI(-BestLag+1:length(SinkBitsI));
    SinkBitsQ = SinkBitsQ(-BestLag+1:length(SinkBitsQ));
end
%
% Make all arrays the same length.
%
TotalBits = min(length(SourceBitsI),length(SinkBitsI));
TotalBits = TotalBits-20;
SourceBitsI = SourceBitsI(10:TotalBits);
SourceBitsQ = SourceBitsQ(10:TotalBits);
SinkBitsI = SinkBitsI(10:TotalBits);
SinkBitsQ = SinkBitsQ(10:TotalBits);
%
% Find the number of errors and the BER.
%
Errors = sum(SourceBitsI ~= SinkBitsI) + sum(SourceBitsQ ~=
SinkBitsQ);
BER_MC = Errors/(2*length(SourceBitsI));
% End of function file.
```

2、主函数：QPSK 系统仿真

```
Eb = 22:0.5:26; No = -50;                    % Eb (dBm) and No
(dBm/Hz)
ChannelAttenuation = 70;                     % Channel attenuation in
dB
EbNodB = (Eb-ChannelAttenuation)-No;         % Eb/No in dB
EbNo = 10.^(EbNodB./10);                     % Eb/No in linear units
BER_T = 0.5*erfc(sqrt(EbNo));                % BER (theoretical)
```

```
%BER_T = erfc(sqrt(EbNo));
N = round(100./BER_T);                          % Symbols to transmit
BER_MC = zeros(size(Eb));                       % Initialize BER vector
for k=1:length(Eb)                              % Main Loop
  BER_MC(k) =
c214_MCQPSKrun01(N(k),Eb(k),No,ChannelAttenuation,0,0,0,4);
  disp(['Simulation ',num2str(k*100/length(Eb)),'% Complete']);
end
%semilogy(EbNodB,BER_MC,'o',EbNodB,2*BER_T,'-')
semilogy(EbNodB,BER_MC,'o',EbNodB,BER_T,'-')
xlabel('Eb/No (dB)'); ylabel('Bit Error Rate');
%legend('MC BER Estimate','Theoretical BER'); grid;
legend('BER Estimate','Theoretical BER'); grid;
% End of script file.
```

3、主函数：相位抖动

```
PhaseBias = 0; PhaseJitter = 0:2:30;
Eb = 24; No = -50;                                  % Eb (dBm) and No
(dBm/Hz)
ChannelAttenuation = 70;                            % dB
EbNo = 10.^((Eb-ChannelAttenuation-No)/10);
BER_T = 0.25*erfc(sqrt(EbNo)*ones(size(PhaseJitter)));
N = round(100./BER_T);
BER_MC = zeros(size(PhaseJitter));
for k=1:length(PhaseJitter)
  BER_MC(k) = c214_MCQPSKrun01(N(k),Eb,No,ChannelAttenuation,0,0,...
    PhaseBias,PhaseJitter(k));
  disp(['Simulation ',num2str(k*100/length(PhaseJitter)),'%
Complete']);
end
semilogy(PhaseJitter,BER_MC,'o',PhaseJitter,2*BER_T,'-')
xlabel('Phase Error Std. Dev. (Degrees)');
ylabel('Bit Error Rate');
legend('MC BER Estimate','Theoretical BER'); grid;
% End of script file.
```

4、主函数：相位同步误差

```
PhaseError = 0:10:360;                  % Phase Error at Receiver
```

```
Eb = 24; No = -50;                    % Eb (dBm) and No (dBm/Hz)
ChannelAttenuation = 70;              % dB
EbNo = 10.^((Eb-ChannelAttenuation-No)/10);
BER_T = 0.25*erfc(sqrt(EbNo)*ones(size(PhaseError)));
N = round(100./BER_T);
BER_MC = zeros(size(PhaseError));
for k=1:length(PhaseError)
  BER_MC(k) = c214_MCQPSKrun01(N(k),Eb,No,ChannelAttenuation,0,0,...
    PhaseError(k),0);
  disp(['Simulation ',num2str(k*100/length(PhaseError)),'%
Complete']);
end
semilogy(PhaseError,BER_MC,'o',PhaseError,2*BER_T,'-')
xlabel('Phase Error (Degrees)');
ylabel('Bit Error Rate');
legend('MC BER Estimate','Theoretical BER'); grid;
 % End of script file.
```

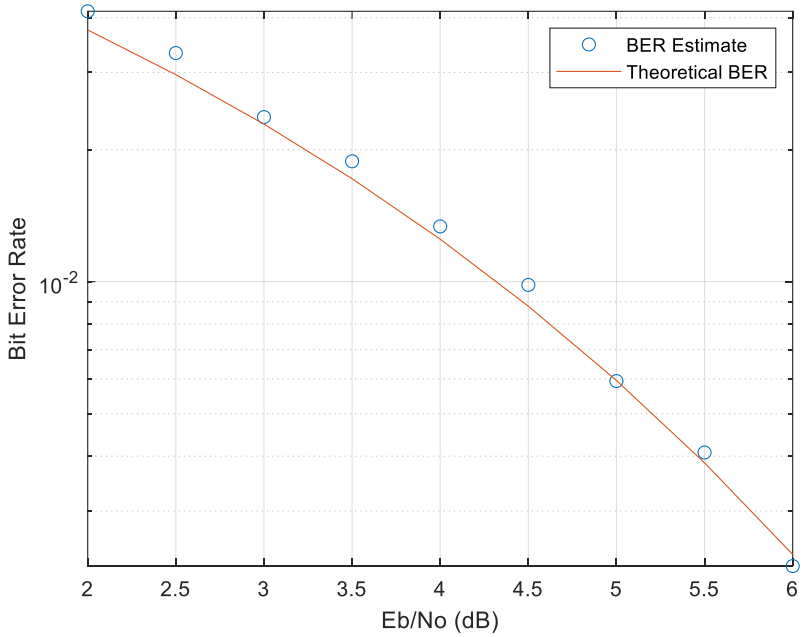
5、主函数：系统时延

```
Eb = 23; No = -50;                % Eb (dBm) and No (dBm/Hz)
ChannelAttenuation = 70;          % channel attenuation in dB
N = 1000;
delay = -0.1:0.1:0.5;
EbNo = 10.^(((Eb-ChannelAttenuation)-No)/10);
BER_MC = zeros(size(delay));
for k=1:length(delay)
  BER_MC(k) = c214_MCQPSKrun01(N,Eb,...
       No,ChannelAttenuation,delay(k),0,0,0);
  disp(['Simulation ',...
       num2str(k*100/length(delay)),'% Complete']);
end
BER_T = 0.25*erfc(sqrt(EbNo))*ones(size(delay)); % Theoretical BER
semilogy(delay,BER_MC,'o',delay,BER_T,'-')    %Plot BER vs Delay
xlabel('Delay (symbols)'); ylabel('Bit Error Rate');
legend('MC BER Estimate','Theoretical BER');grid;
% End of script file.
```

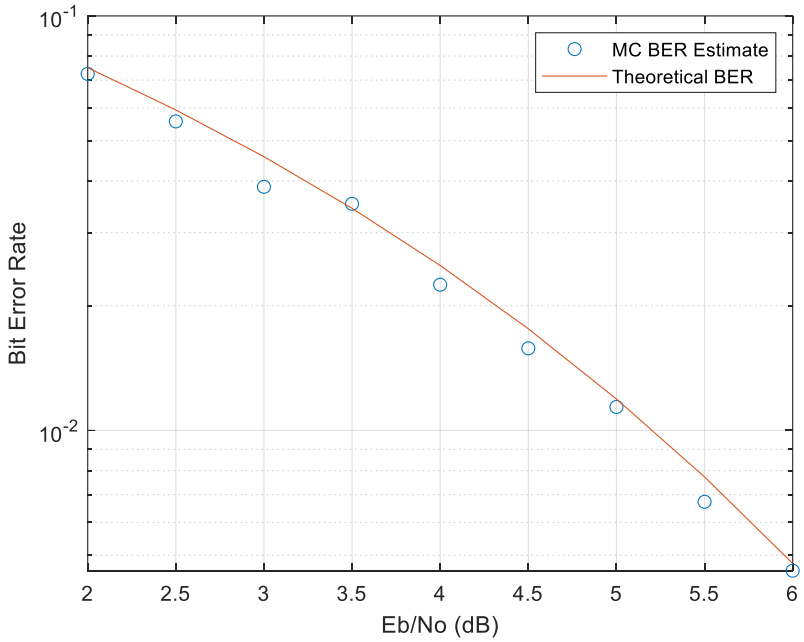
4. 仿真实验结论

4.1 BER

QPSK：



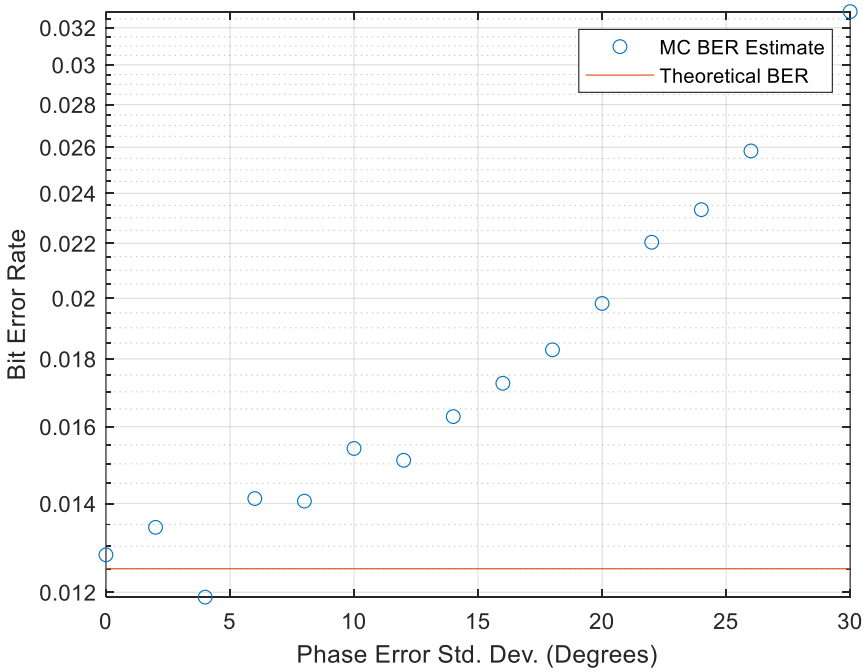
DQPSK：



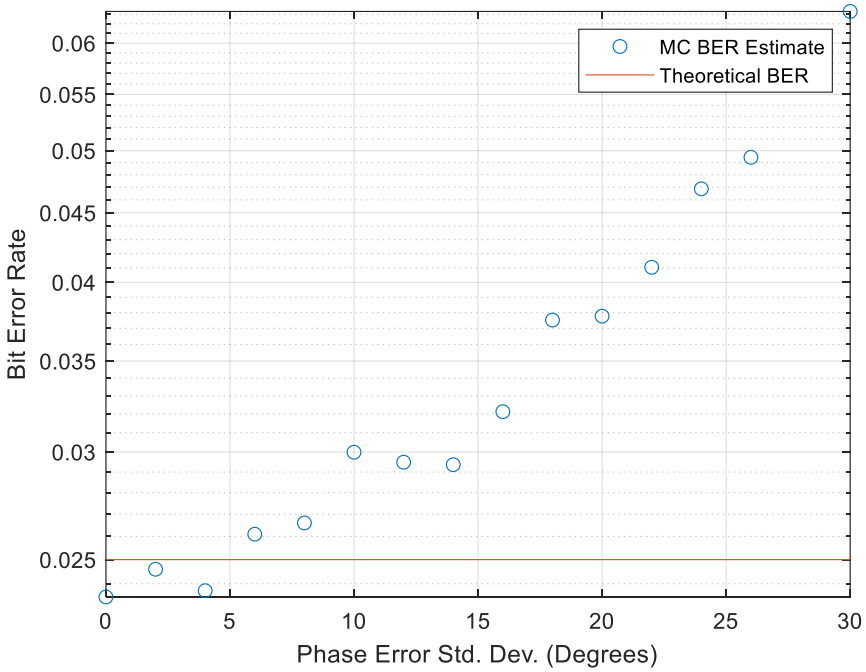
由图可知，QPSK 和 DQPSK 仿真系统的 BER 相差不大。

4.2 相位抖动

QPSK：



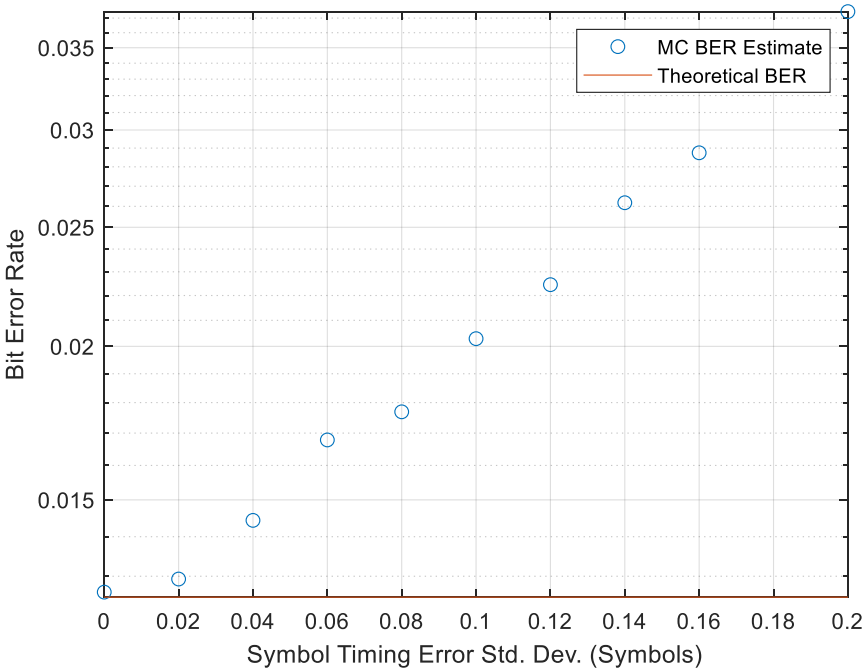
DQPSK：



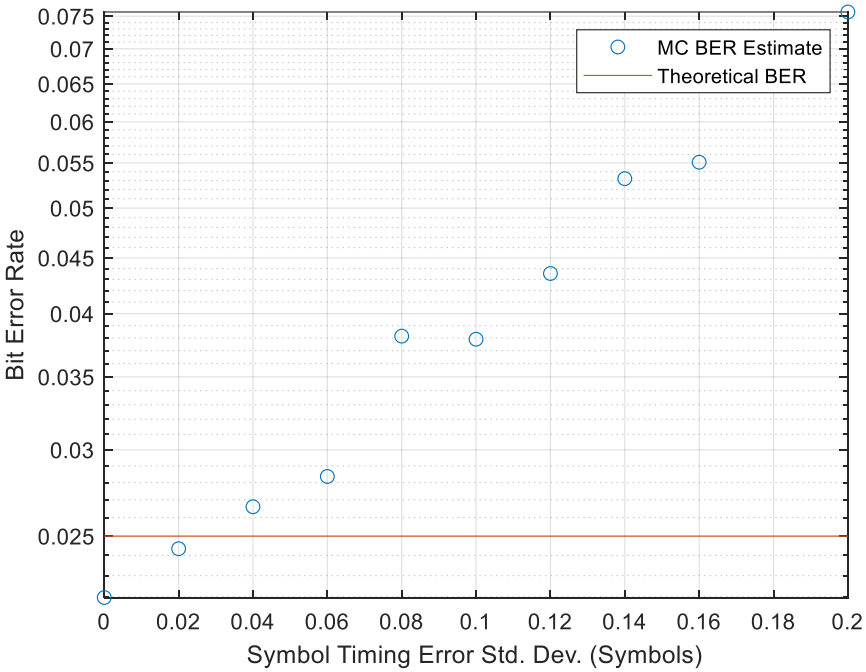
在相同的相位抖动情况下，QPSK 的误码率小于 DQPSK

4.3 符号同步抖动

QPSK：



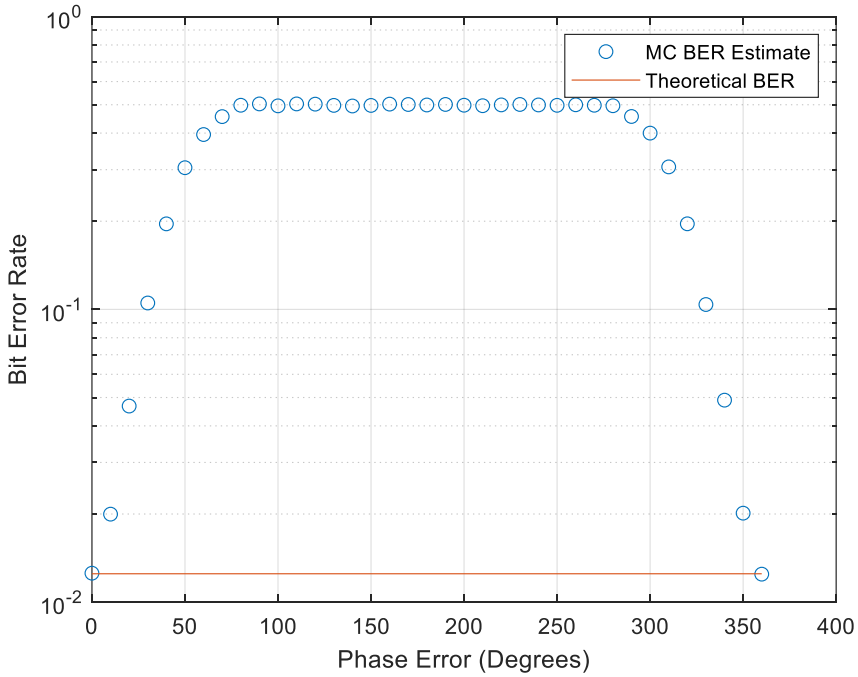
DQPSK：



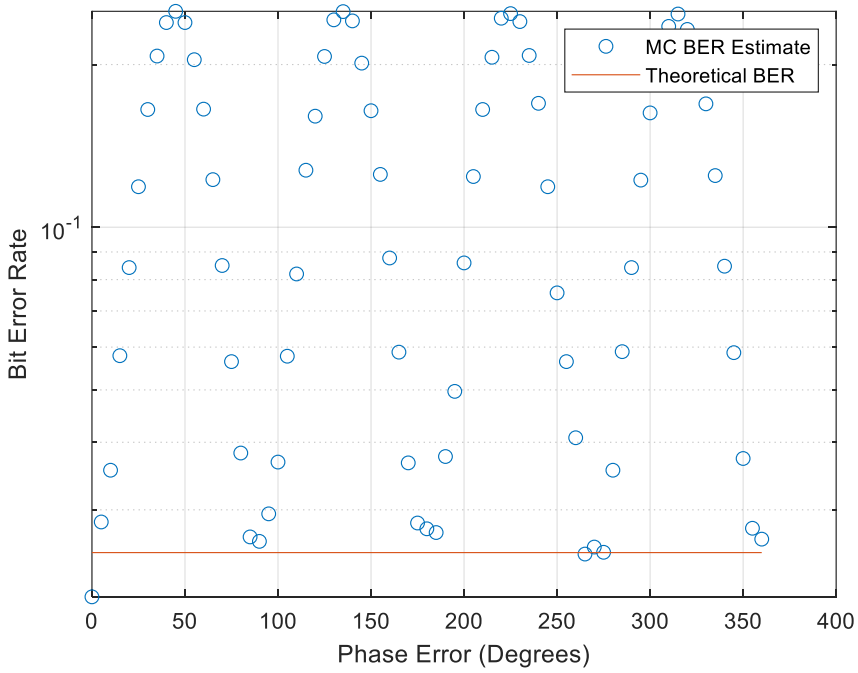
相同符号同步抖动的情况下，DQPSK 的 BER 大于 QPSK。

4.4 相位同步误差

QPSK：



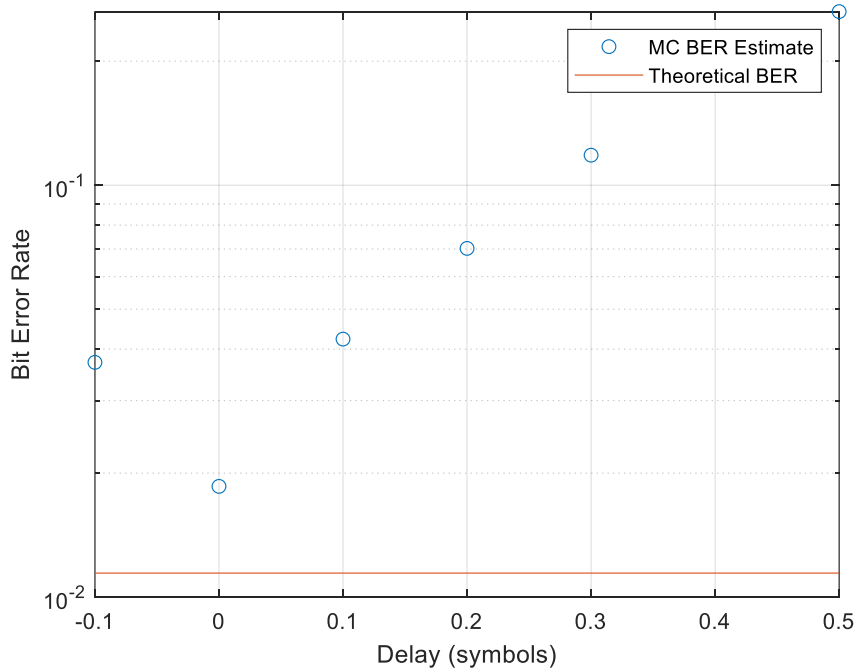
DQPSK：



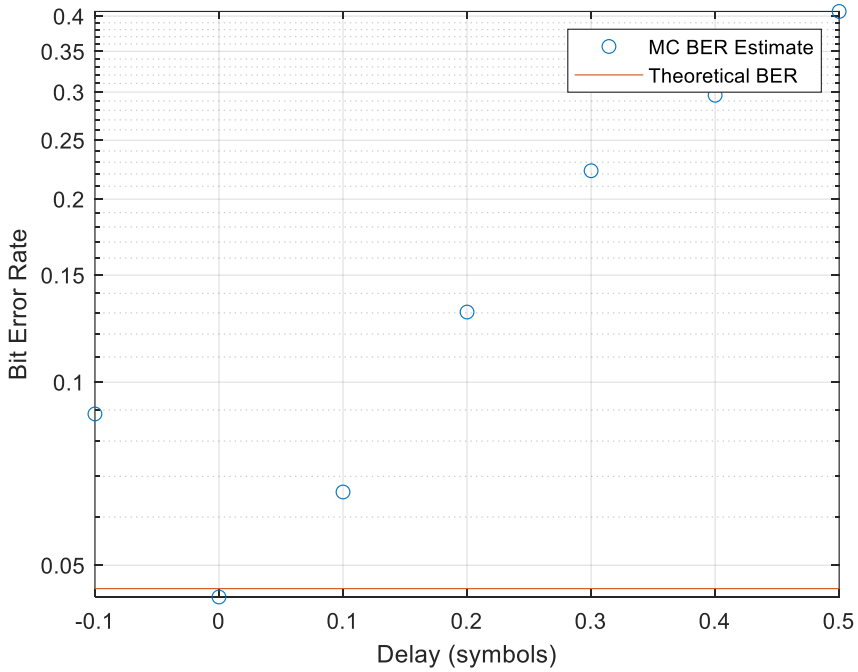
在相同相位同步误差时，QPSK 的误码率大于 DQPSK。

4.5 系统时延

QPSK：

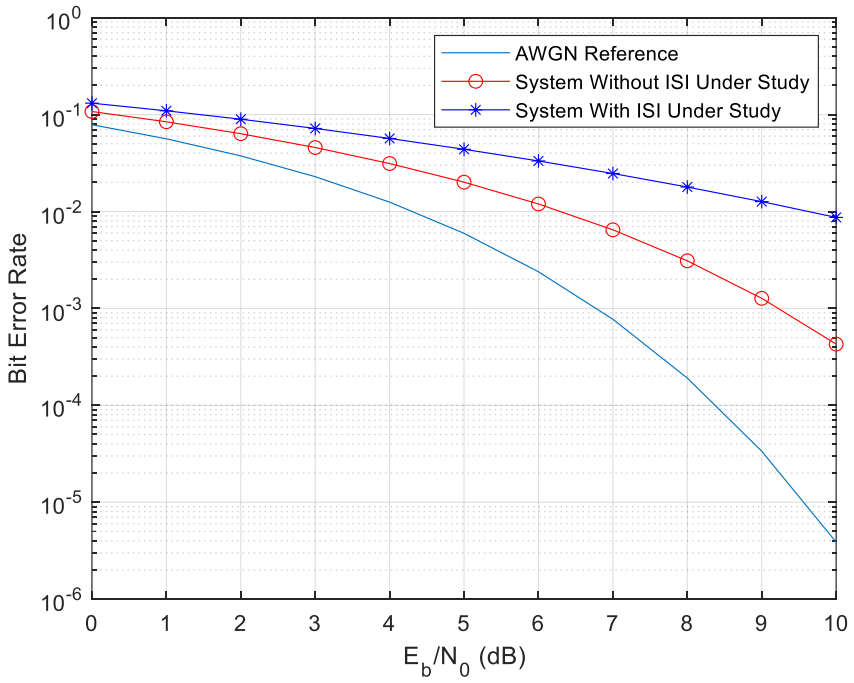


DQPSK：



在相同时延条件下，QPSK 的误码率小于 DQSK。

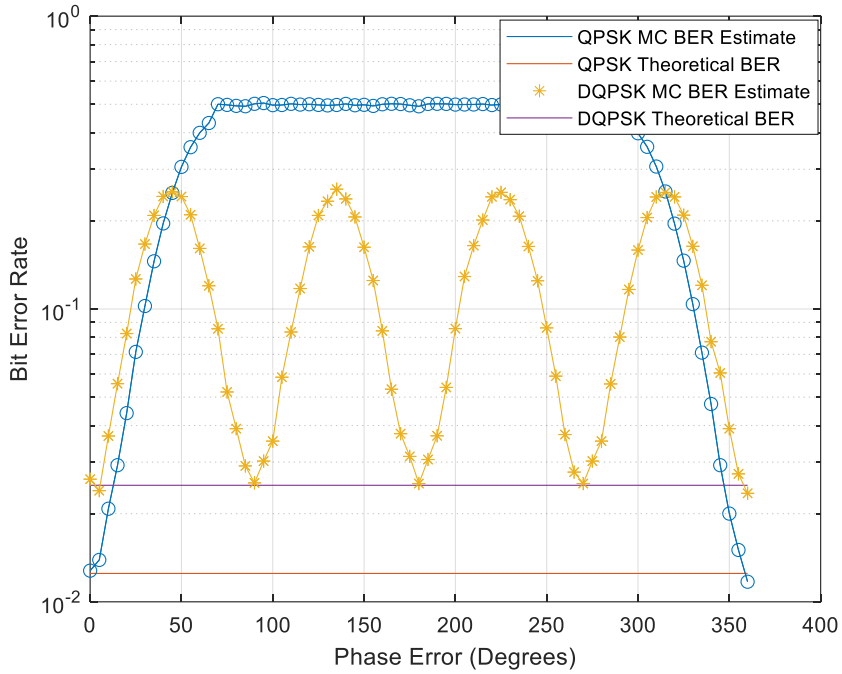
4.6 有无 ISI 对比




```
NN = 256;                                   % number of symbols
tb = 0.5;                                   % bit time
p0 = 1;                                     % power
fs = 16;                                    % samples/symbol
ebn0db = [0:1:10];                          % Eb/N0 vector
[b,a] = butter(5,1/20);                     % transmitter filter
parameters
%
% Establish QPSK signals
%
x = random_binary(NN,fs)+i*random_binary(NN,fs);     % QPSK signal
y1 = x;                                              % save signal
y2a = y1*sqrt(p0);                                   % scale
amplitude
%
% Transmitter filter
%
y2 = filter(b,a,y2a);                       % filtered signal
%
% Matched filter
%
b = ones(1,fs); b = b/fs; a = 1;            % matched filter parameters
```

```
y = filter(b,a,y2);                          % matched filter output
%
% End of simulation
%
% Use the semianalytic BER estimator. The following sets
% up the semi analytic estimator. Find the maximum magnitude
% of the cross correlation and the corresponding lag.
%
[cor lags] = vxcorr(x,y);
cmax = max(abs(cor));
nmax = find(abs(cor)==cmax);
timelag = lags(nmax);
theta = angle(cor(nmax));
y = y*exp(-i*theta);                         % derotate
%
% Noise BW calibration
%
hh = impz(b,a);                              % receiver impulse response
nbw = (fs/2)*sum(hh.^2);                     % noise bandwidth
%
% Delay the input, and do BER estimation on the last 128 bits.
% Use middle sample. Make sure the index does not exceed number
% of input points. Eb should be computed at the receiver input.
%
index = (10*fs+8:fs:(NN-10)*fs+8);
xx = x(index);
yy = y(index-timelag+1);
[n1 n2] = size(y2); ny2=n1*n2;
eb = tb*sum(sum(abs(y2).^2))/ny2;
eb = eb/2;
[peideal,pesystem] = qpsk_berest(xx,yy,ebn0db,eb,tb,nbw);
% subplot(1,2,1)
% yscale = 1.5*max(real(yy));
% plot(yy,'+')
% xlabel('Direct Sample'); ylabel('Quadrature Sample'); grid;
% axis([-yscale yscale -yscale yscale])
% subplot(1,2,2)
semilogy(ebn0db,peideal,ebn0db,pesystem,'ro-'); grid;
xlabel('E_b/N_0 (dB)'); ylabel('Bit Error Rate')
legend('AWGN Reference','System Under Study')
% End of script file.
```

4.7 QPSK 和 DDPSK 相位误差对比




```
PhaseError = 0:5:360;                    % Phase Error at Receiver
Eb = 24; No = -50;                       % Eb (dBm) and No (dBm/Hz)
ChannelAttenuation = 70;                 % dB
EbNo = 10.^((Eb-ChannelAttenuation-No)/10);
BER_T1 = 0.25*erfc(sqrt(EbNo)*ones(size(PhaseError)));
N1 = round(100./BER_T1);
BER_T2 = 0.5*erfc(sqrt(EbNo)*ones(size(PhaseError)));
N2 = round(100./BER_T2);
BER_MC1 = zeros(size(PhaseError));
BER_MC2 = zeros(size(PhaseError));

for k1=1:length(PhaseError)
  BER_MC1(k1) =
c214_MCQPSKrun01(N1(k1),Eb,No,ChannelAttenuation,0,0,...
    PhaseError(k1),0);
  disp(['Simulation ',num2str(k1*100/length(PhaseError)),'%
Complete']);
end

for k2=1:length(PhaseError)
  BER_MC2(k2) =
c214_MCQPSKrun(N2(k2),Eb,No,ChannelAttenuation,0,0,...
```

```
    PhaseError(k2),0);
  disp(['Simulation ',num2str(k2*100/length(PhaseError)),'%
Complete']);
end

semilogy(PhaseError,BER_MC1,PhaseError,2*BER_T1,'-',...
    PhaseError,BER_MC2,'*',PhaseError,2*BER_T2,'-')
line(PhaseError,BER_MC1,'color','[0 0.4470 0.7410]','Marker','o');
line(PhaseError,BER_MC2,'color','[0.9290 0.6940
0.1250]','Marker','*');
xlabel('Phase Error (Degrees)');
ylabel('Bit Error Rate');
legend('QPSK MC BER Estimate','QPSK Theoretical BER',...
    'DQPSK MC BER Estimate','DQPSK Theoretical BER');
grid on;
 % End of script file.
```