

课程名称： 无线通信系统仿真实验

学　　期： 2016-2017年第三学期

任课教师： 徐位凯

题 目： 课程实验报告（物理层部分）

姓 名： 赵晗

学 号： 22920142203759

学 院： 信息科学与技术学院

**一、实验目的**

1. 加深对通信原理的理解；

2. 掌握使用计算机仿真对简单通信系统进行性能评估的方法；

1. 了解通信系统物理层算法的研究方法。

**二、实验原理**

通信系统的性能评价有以下几种：

信噪比（SNR）（模拟系统）

误比特率（BER）（数字系统）

1.通信系统的组成

通信系统的组成如下图所示：

2.信道容量与带宽效率

信道容量：表示一个信道的最大数据传输速率 ，可归一化表示为bit/s/Hz

AWGN信道：，W为信号带宽，S/N为信噪比。

带宽效率：传输速率与信道带宽的比



3.带宽-能量效率关系

数字通信系统中两个最可贵的资源：带宽W和信号功率（能量Eb）

衡量数字通信系统的两个主要方面：有效性和可靠性，有效性->传输速率，可靠性->BER

设计的矛盾：提高传速率->牺牲BER性能，提高BER性能->牺牲传输速率R，因此，它们有一个折中（tradeoff）

目标：在给定的设计约束下，以可接受的复杂度，实现传输速率的最大化

衡量数字通信系统的两个主要方面：有效性和可靠性，有效性（传输速率），可靠性（BER）

4.带通信号的低通等效（复包络）

带通信号一般用复包络进行表示：，可以用欧拉公式进行展开。

复包络在直角坐标中可以表示为。

其中为同相分量，为正交分量。

**实验工具**

利用matlab进行仿真，通过仿真结果与理想值进行比较，从而得出结论。

**三、实验内容：**

**实验一：随机信号的产生**

1.实验原理：

通信系统的各种信号以及噪声、信道衰落都可以看做相应的随机过程，通信系统仿真就是通过计算机产生各种随机信号，并依据通信原理对这些信号进行相应的处理，获取期望的结果，或者验证算法的正确性。因此，随机信号的发生在通信系统仿真中具有重要的意义。该实验的目的是掌握如何使用计算机产生各种通信系统仿真中所需的随机信号。

基本原理：计算机本身是一个确定性设备，要求计算机程序产生完全随机的数据是不可能的，因此，在仿真中通过计算机产生的数据只能算是伪随机数。从预测的角度看，周期数据是完全可以预测的，也就是确定的，但周期趋于无穷大时，可以认为该数据具有伪随机特性，因此，这就为我们产生伪随机数据提供了思路，即产生周期尽量大的数据，并作为仿真中的随机数。

产生伪随机数的方法通常基于数论的线性同余法。

2.实验目的：

掌握伪随机数的产生方法。

3.实验设计：根据实验原理部分，完成以下实验任务。

仿真程序流程图

Figure Wichmann-Hill算法 Figure 逆变换法Rayleigh Figure 通过Rayleigh分布随

均匀分布随机序列产生流程图 随机变量序列产生流程图 机变量产生Gussian分布随机变量流程图

任务一：采用Wichmann-Hill算法产生10000个均匀分布的随机变量，

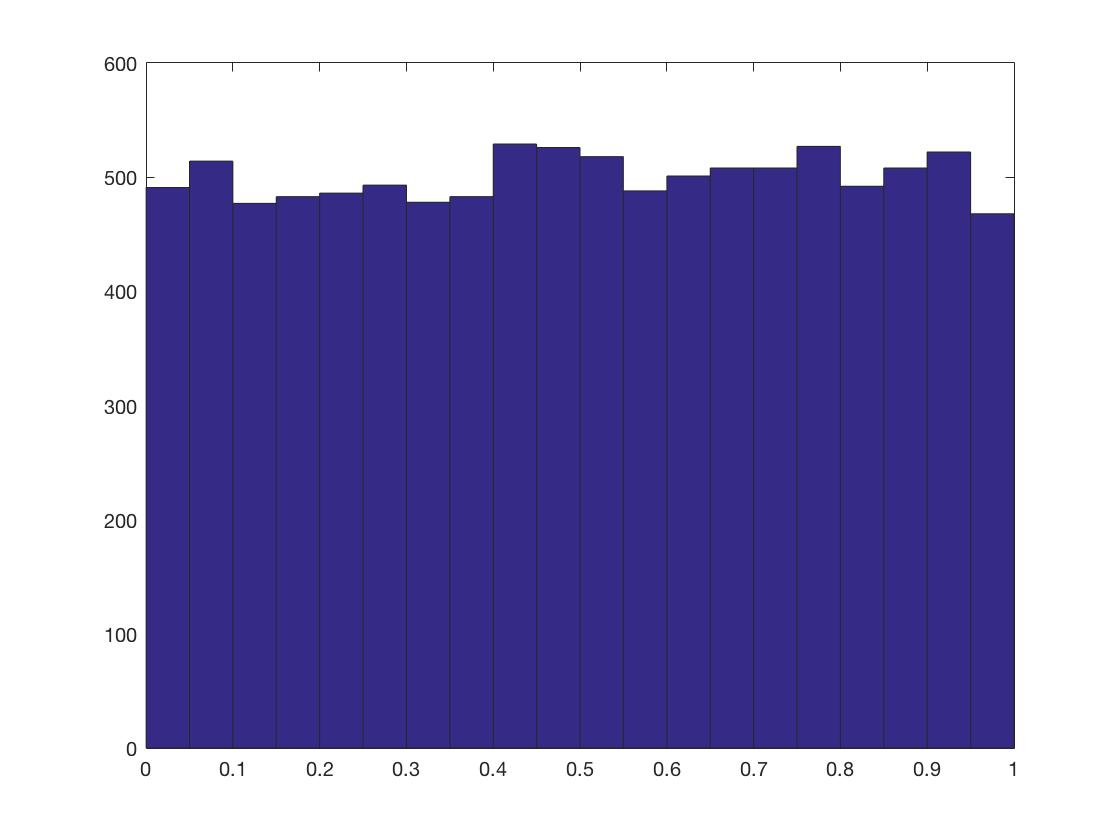


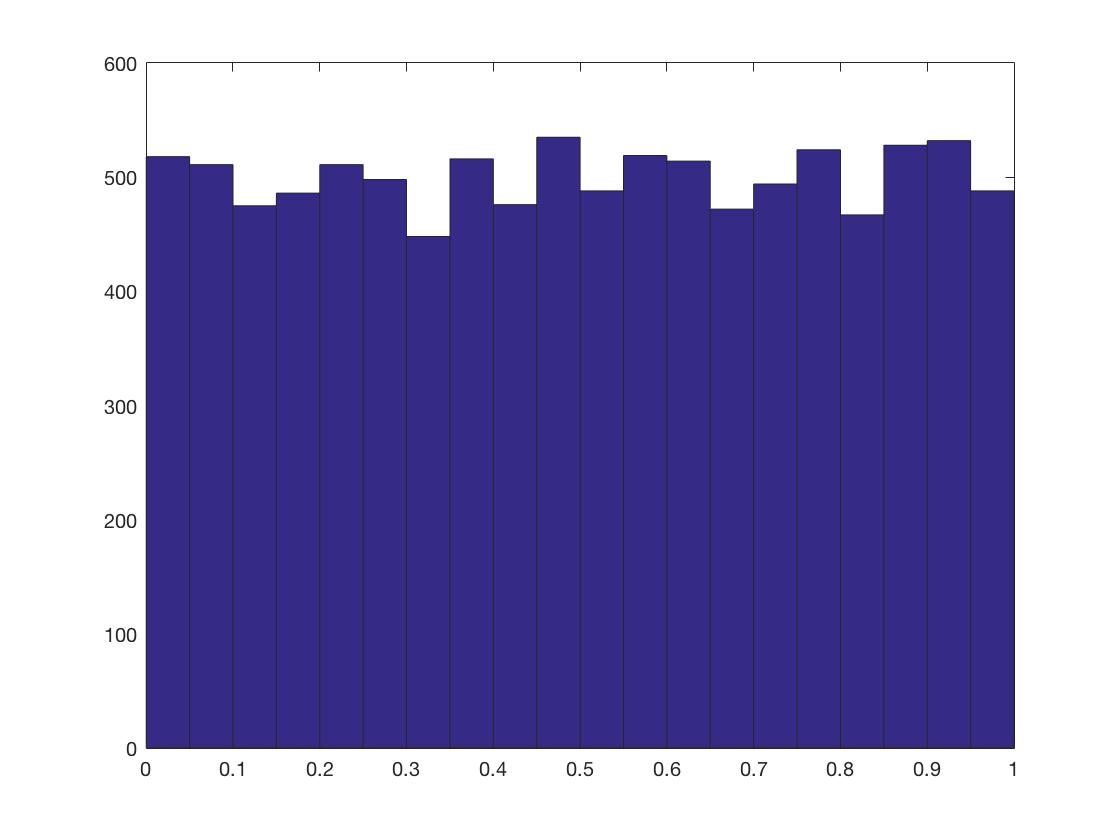


根据两组种子做出的随机序列的直方图，两组种子自行设定。

编写的wichmannhill函数内部以randi（n）（产生随机均匀分布的n以内正整数）作为随机种子。

Wichmann-hill算法产生均匀随机序列直方图



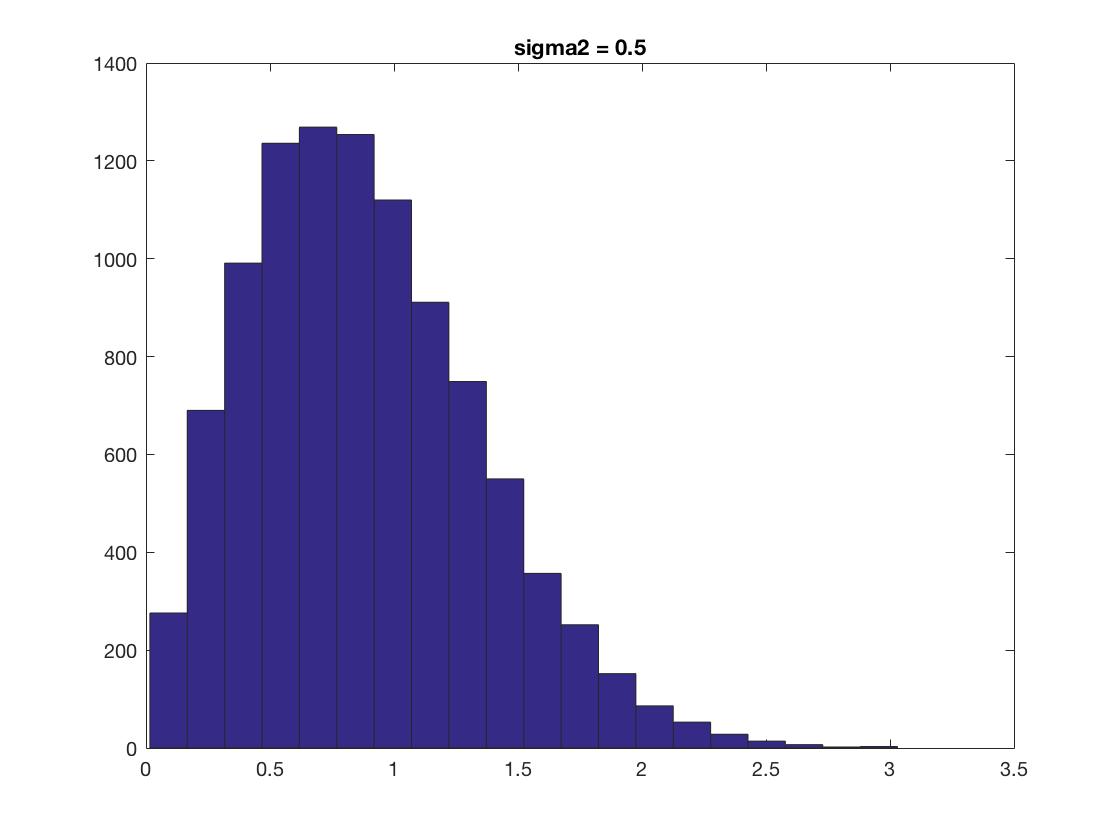


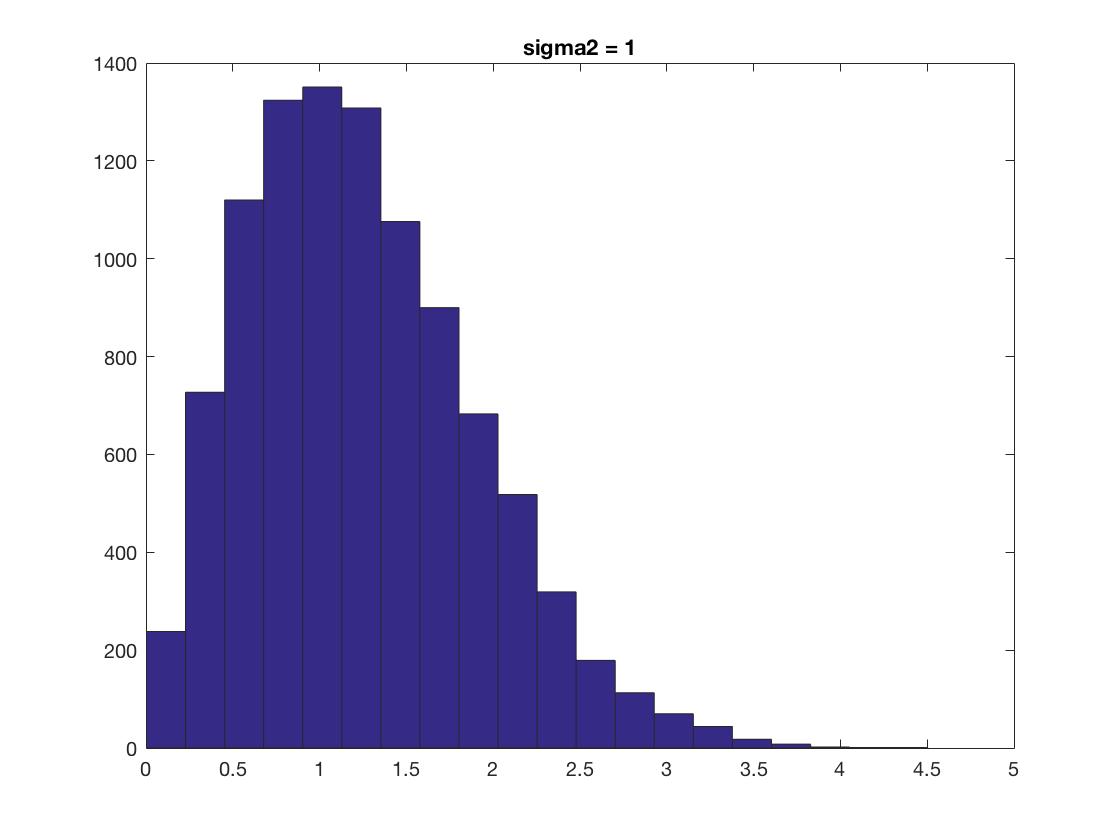
近似可以看作［0，1］均匀分布随机序列。

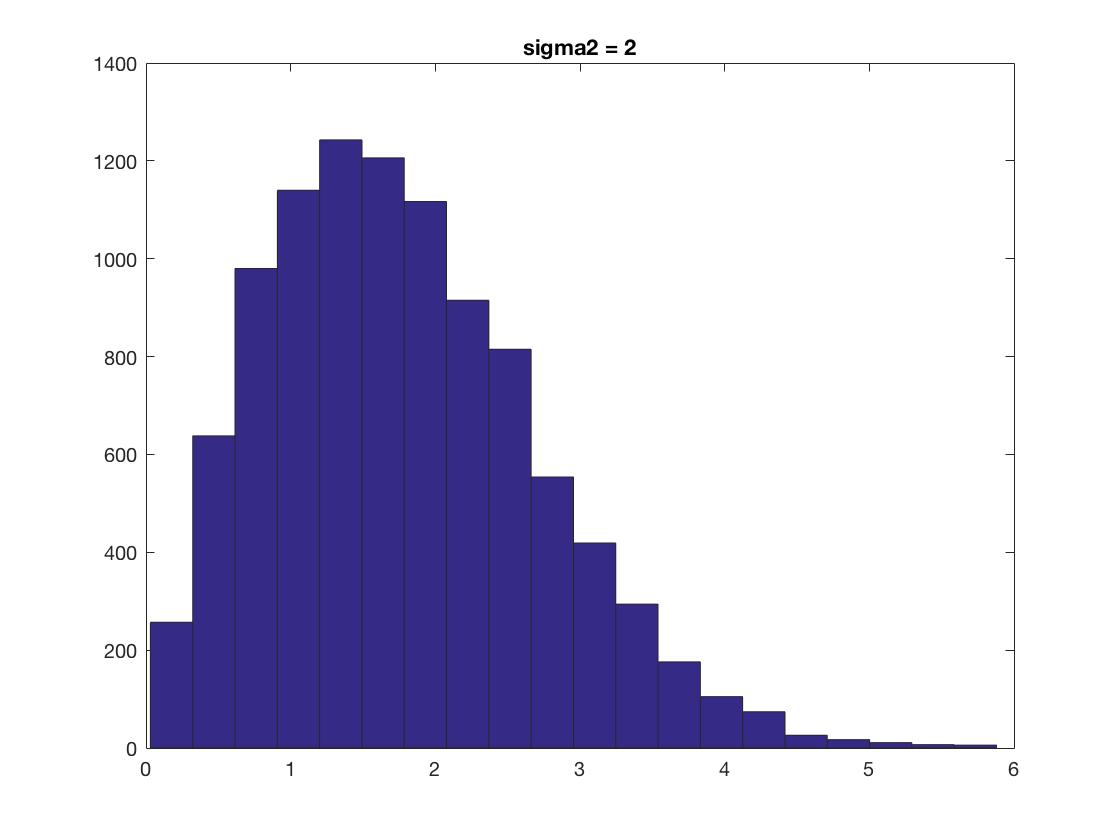
任务二：根据逆变换产生Rayleigh随机变量，设定σ2为0.5、1和2，分别做出Rayleigh随机变量序列的直方图，观察图形。

通过公式可以逆推出

，从而得到瑞利分布的随机变量。







Rayleigh随机变量幅度服从瑞利分布，相位服从[0, 2pi] 均匀分布。

幅度直方图与理论分布较为接近。

sigma^2增大则均值与方差都增大。

任务三：通过Rayleigh分布随机变量产生Gussian分布随机变量，设定方差为1产生标准正态分布的随机变量，计算其方差并作直方图。改变σ值观察直方图的变化。

1. 通过Rayleigh分布随机变量产生Gussian分布随机变量，设定sigma^2为1产生标准正态分布的随机变量，计算其方差并做直方图。

高斯随机变量与Rayleigh随机变量之间有关系：

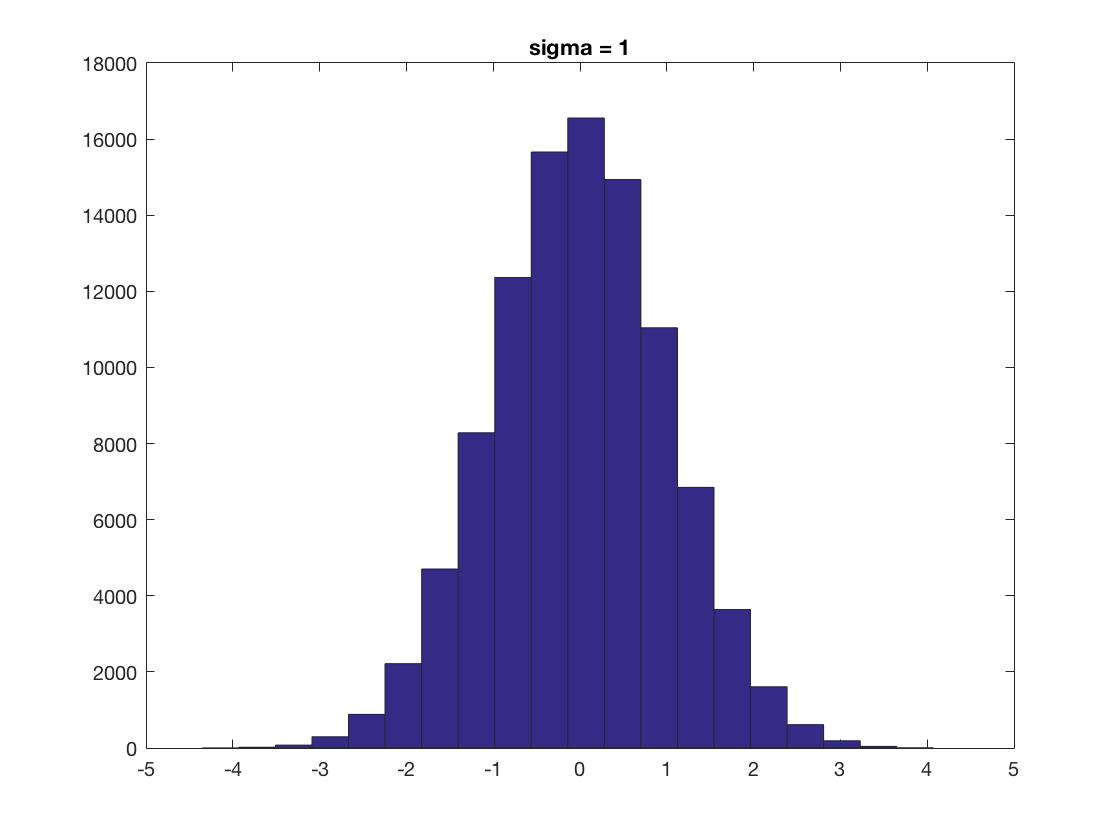


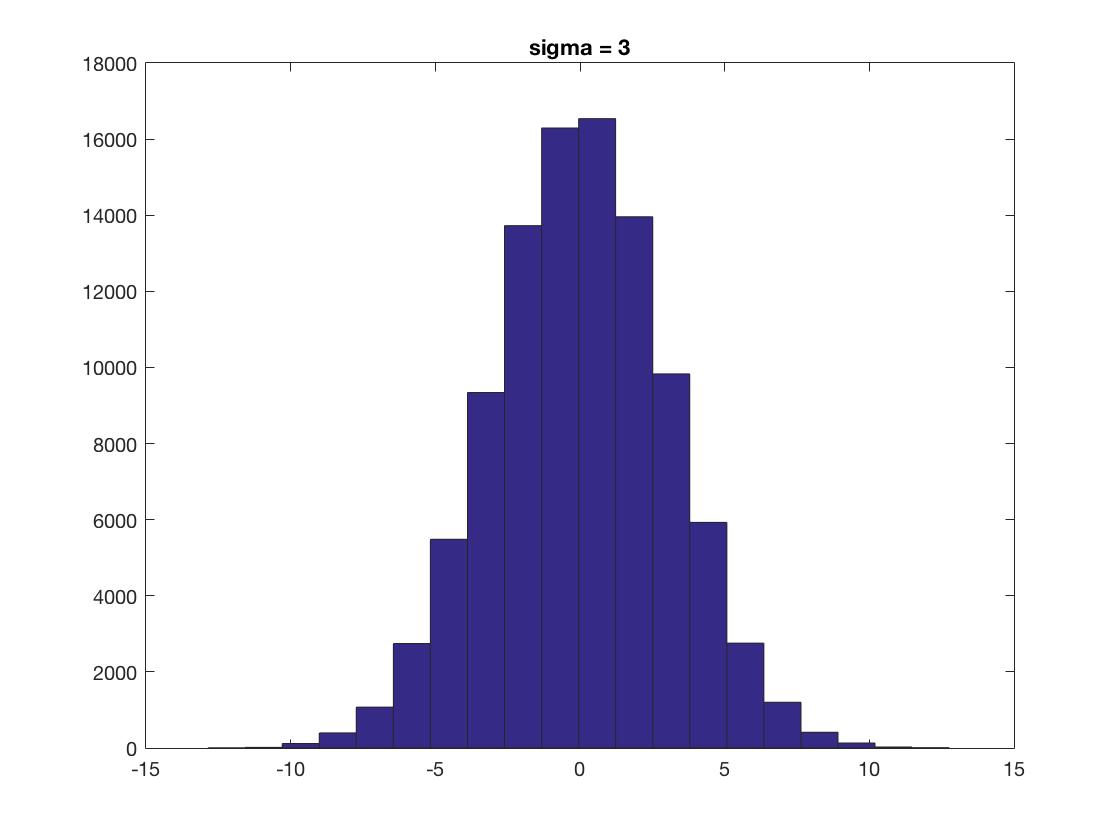
故可以得出：



其中u1,u2分别是两个（0~1）之间均匀分布的随机变量，产生的X 和Y均为高斯随机变量。

得到高斯分布直方图如下所示:





高斯随机变量 均值为0，标准差分别为1，3。分布呈钟型。标准差越大，则随机变量分布越分散。大部分观测值落在3sigma内。

1. **QPSK(4QAM)系统的仿真**

1.实验原理

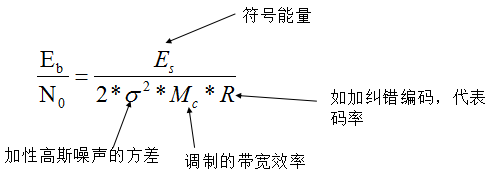
调制的过程主要是将比特映射到复平面上的信号星座点。

高斯信道和瑞利信道的信道模型如下所示：

AWGN信道模型 瑞利衰落信道模型

信噪比Eb/No的统一公式：



2.实验目的

掌握简单调制方法的基带仿真实现，以及AWGN信道和瑞利平坦衰落信道的建模，并完成在这两种信道下的误码率分析。

3.实验任务

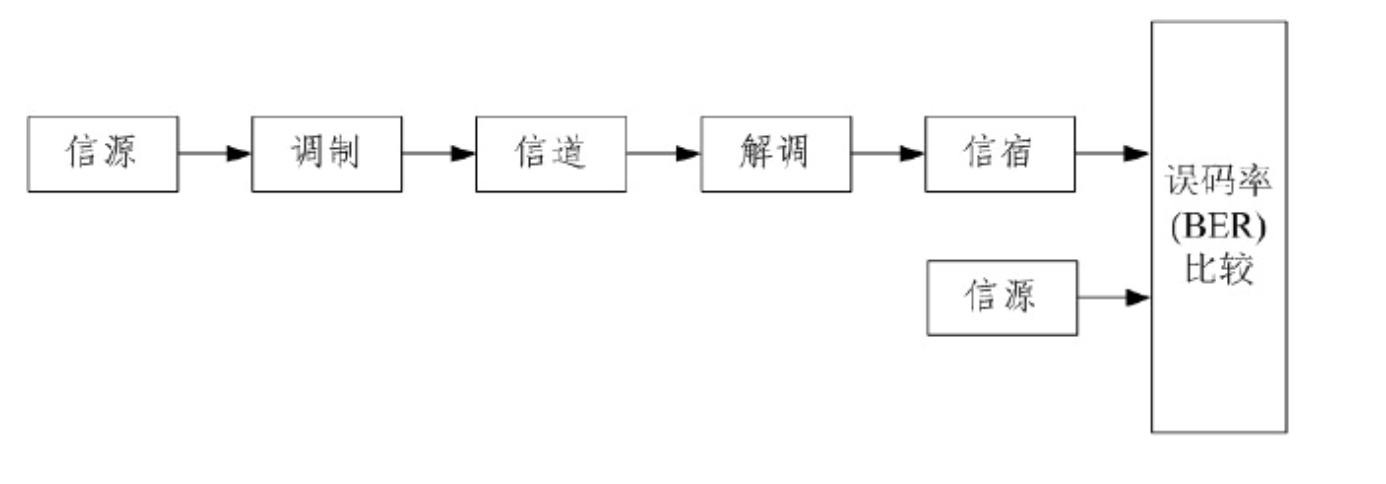
根据QPSK,8PSK,16QAM的原理搭建相应的仿真系统，并完成以下任务。

程序思路如下所示:  
(1)产生高斯分布的随机序列;

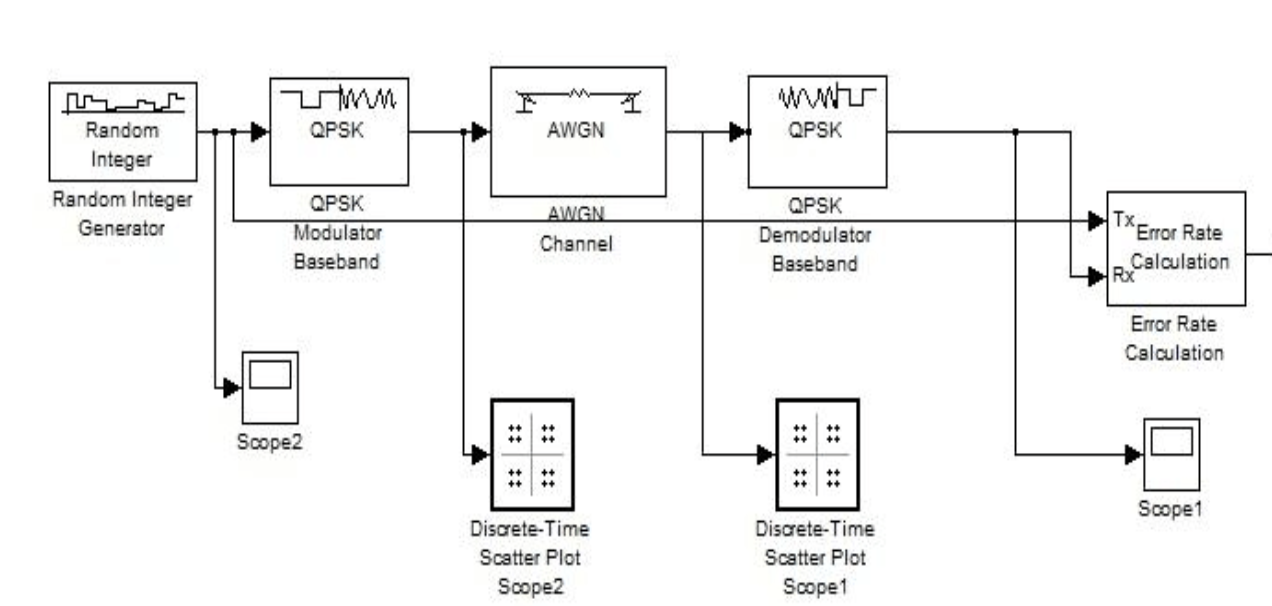
(2)产生对应信噪比的噪声;

1. 对输入序列进行调制;
2. 对调制信号加入高斯噪声,用来模拟高斯噪声信道;
3. 对信号进行解调并计算误码率;
4. 计算理论误码率;
5. 绘制误码率曲线.

基本流程图：



AWGN仿真模型：



Rayleigh信道具有相似的模型。

仿真程序流程图

Figure1 AWGN下 Figure 2 AWGN下 Figure3 AWGN下

QPSK编码仿真 8PSK编码仿真 16QAM编码仿真

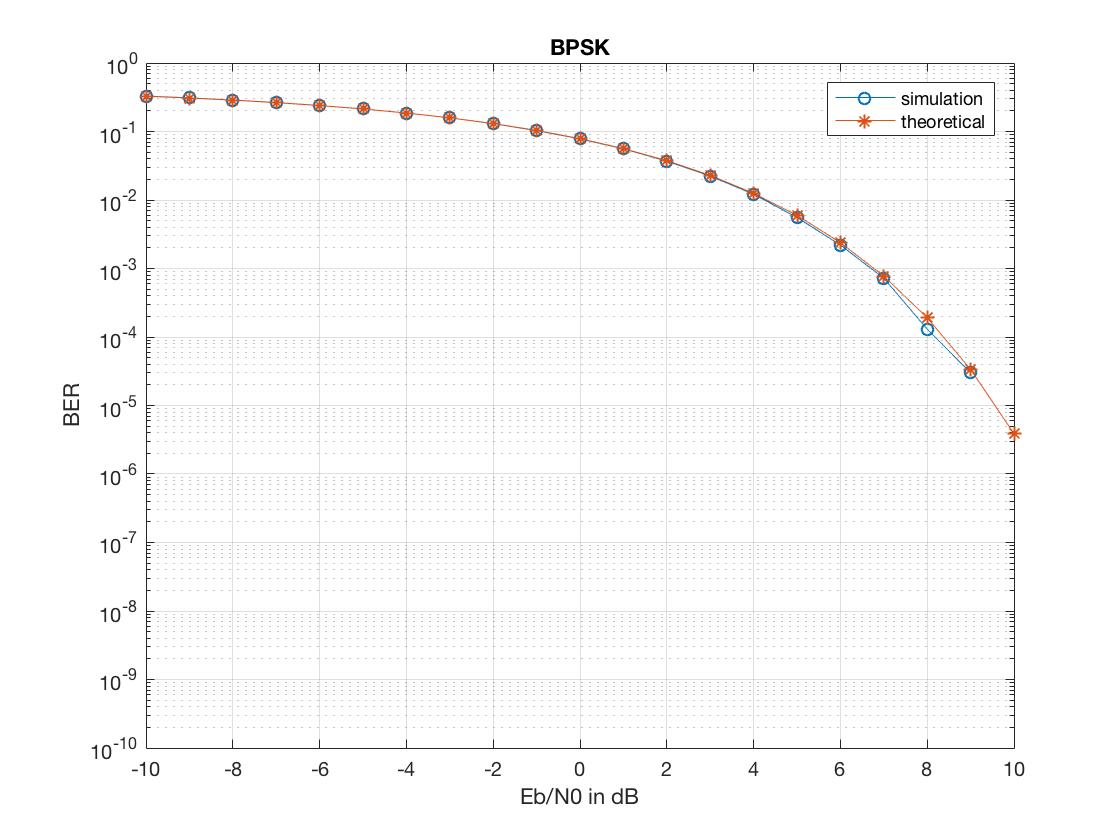


Figure4 Rayleigh衰落信道下 Figure 5 Rayleigh衰落信道下 Figure 6 Rayleigh衰落信道下

QPSK编码仿真 8PSK编码仿真 16QAM编码仿真

任务一：采用实验一产生随机数的方法重新改写程序，并仿真BER结果。

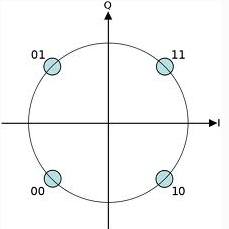
得到AWGN信道下的仿真曲线与理论曲线如下所示:



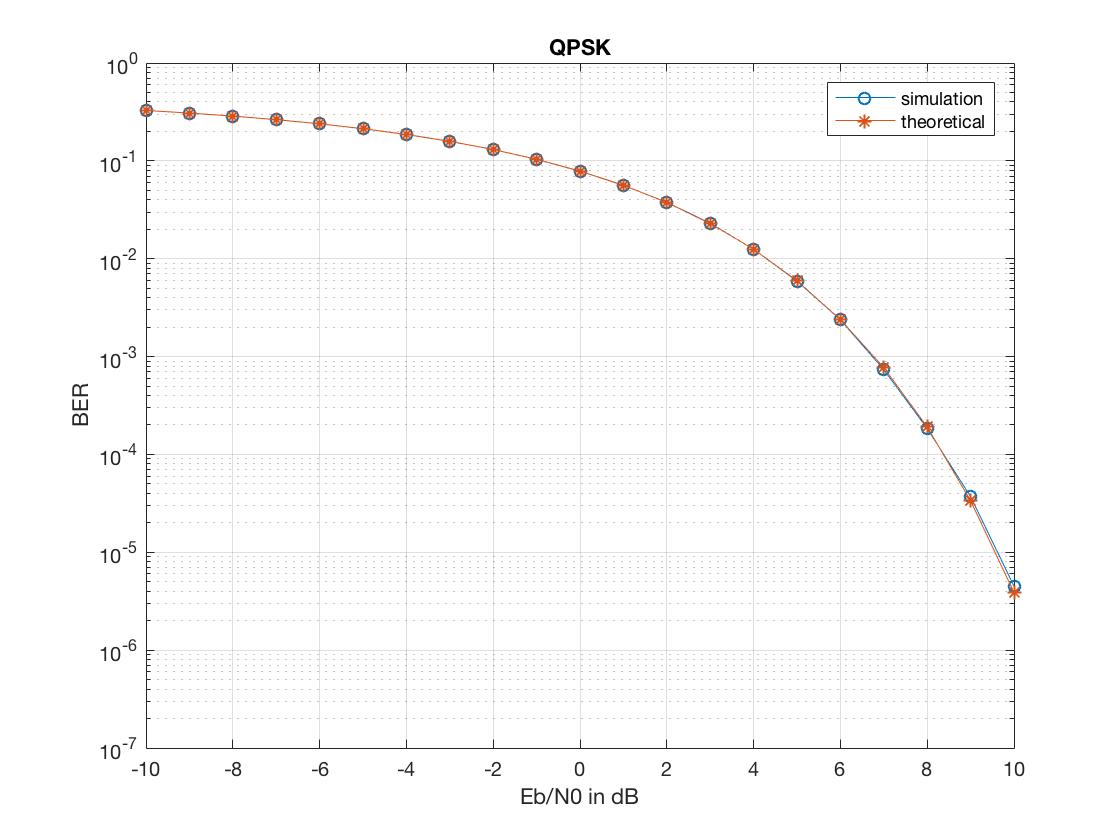
BPSK仿真结果与理论结果几乎没有差别。信噪比越大则误比特率越小。

任务二：参考BPSK程序，搭建QPSK,8PSK,16QAM的基带仿真程序，仿真在AWGN信道和瑞利平坦衰减信道下的误码率性能，比较它们的误码率和带宽效率，解释它们误码率性能差别的原因。  
(1)QPSK调制

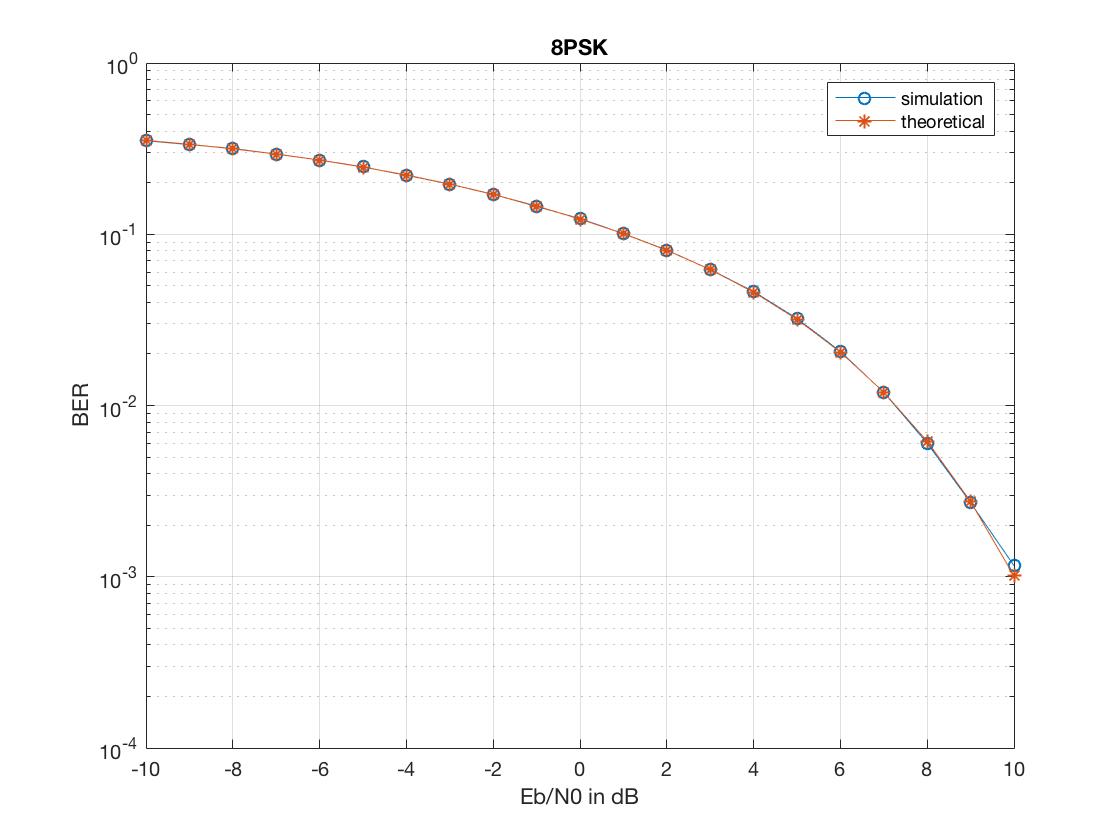
QPSK调制的星座图如下所示：



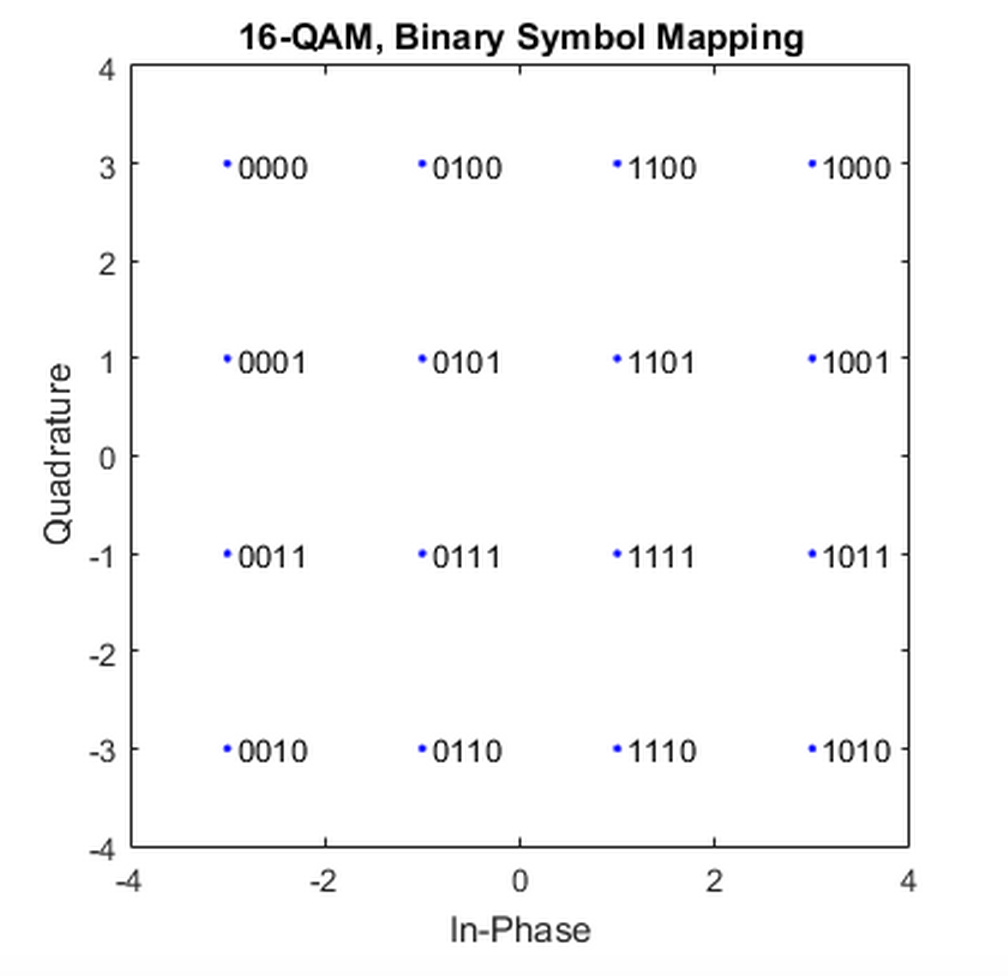
得到AWGN信道的误比特率曲线：



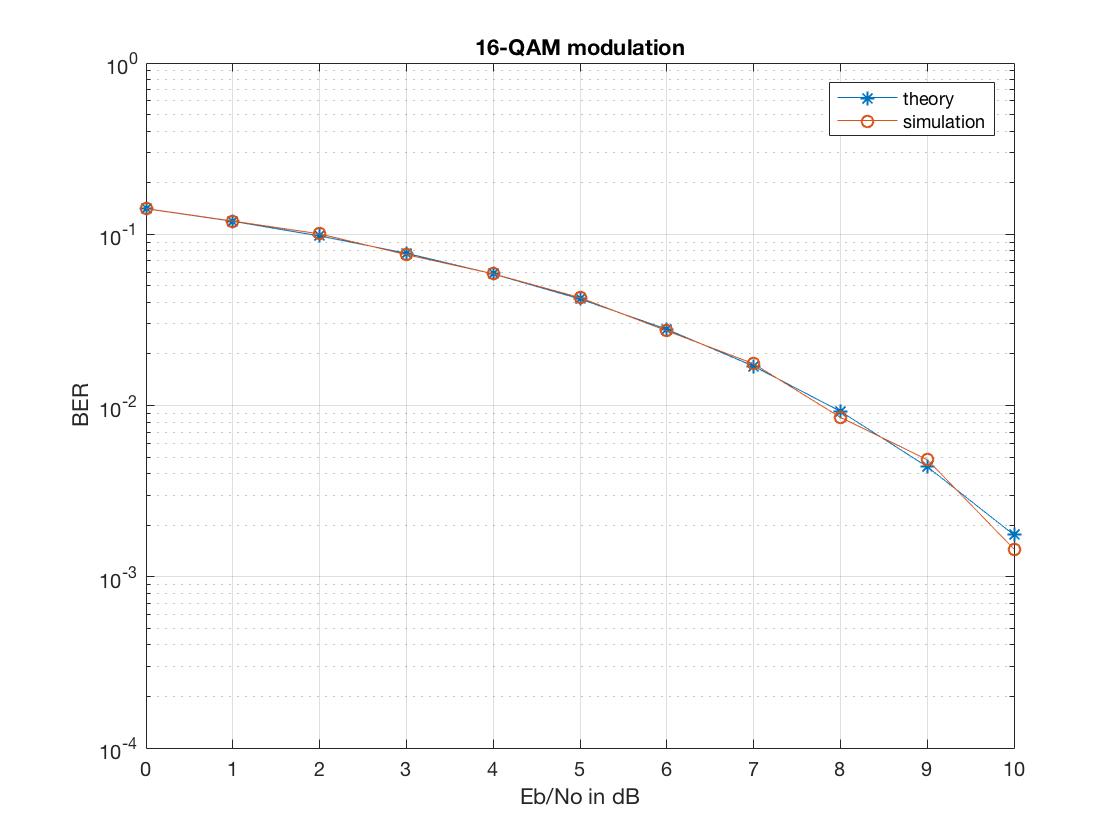
（2）8PSK调制在AWGN信道下的误码率曲线：



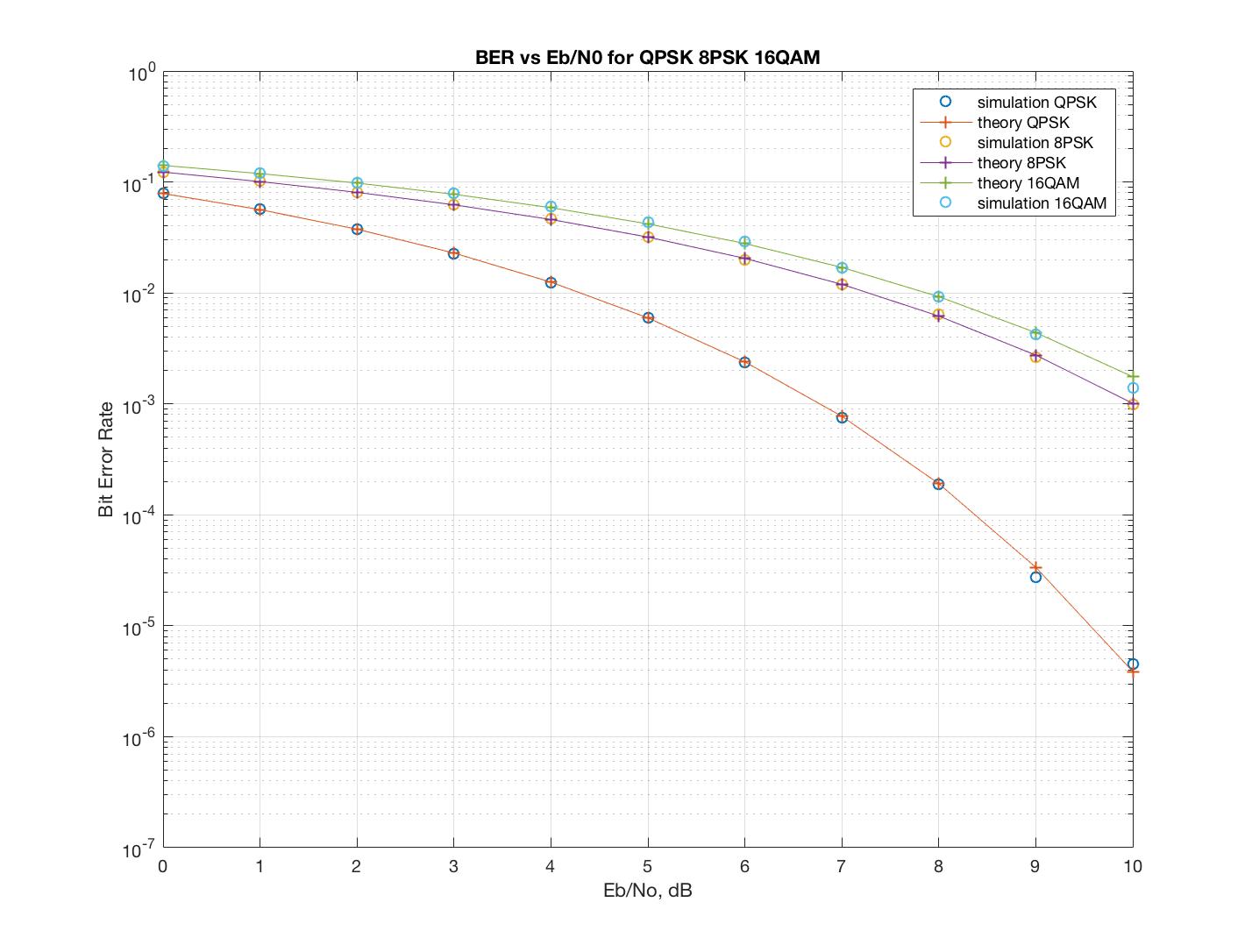
仿真时应注意berawgn函数16QAM调制采用格雷码，而不是书上采用的编码方式。



16QAM仿真：



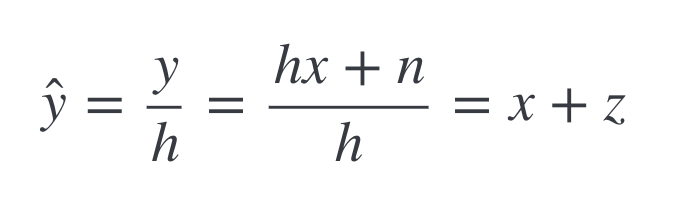
比较：



可以看出，随着信噪比的上升，系统的误比特率减小。对比不同调制方式可以看出，误码率的性能：BPSK>QPSK>8PSK>16QAM。但是从带宽效率看，16QAM的带宽效率最高为4bps/Hz,8PSK的带宽效率为3bps/Hz，QPSK为2bps/Hz。可靠性要求越高，则需要的带宽资源越大，即信道的有效性和可靠性需求是矛盾的。

**瑞利衰落信道**

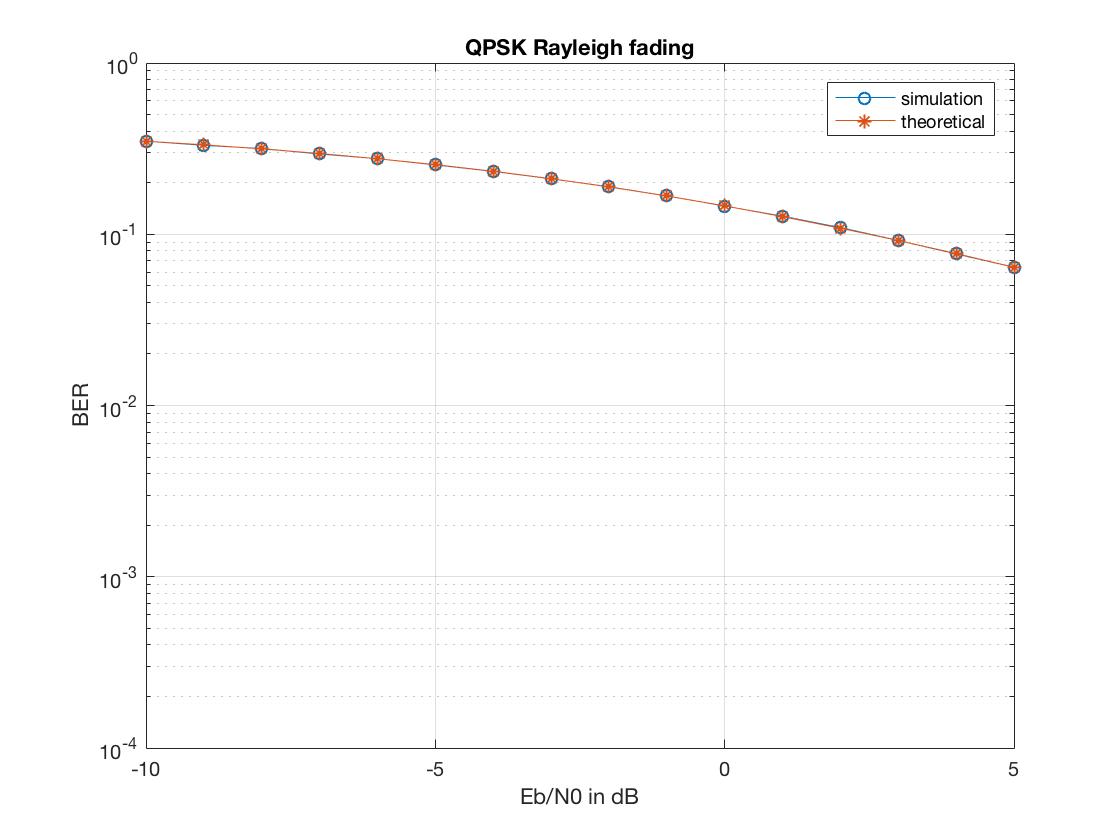
对于Rayleigh衰落信道，非相干解调不适用。因为接收信号y = hx +n中h相位为［0，2pi］均匀分布。因此采用相干解调。假设采用导频信号等方法使得接收端对于信道特性h充分了解（对h的估计无偏，一致）。



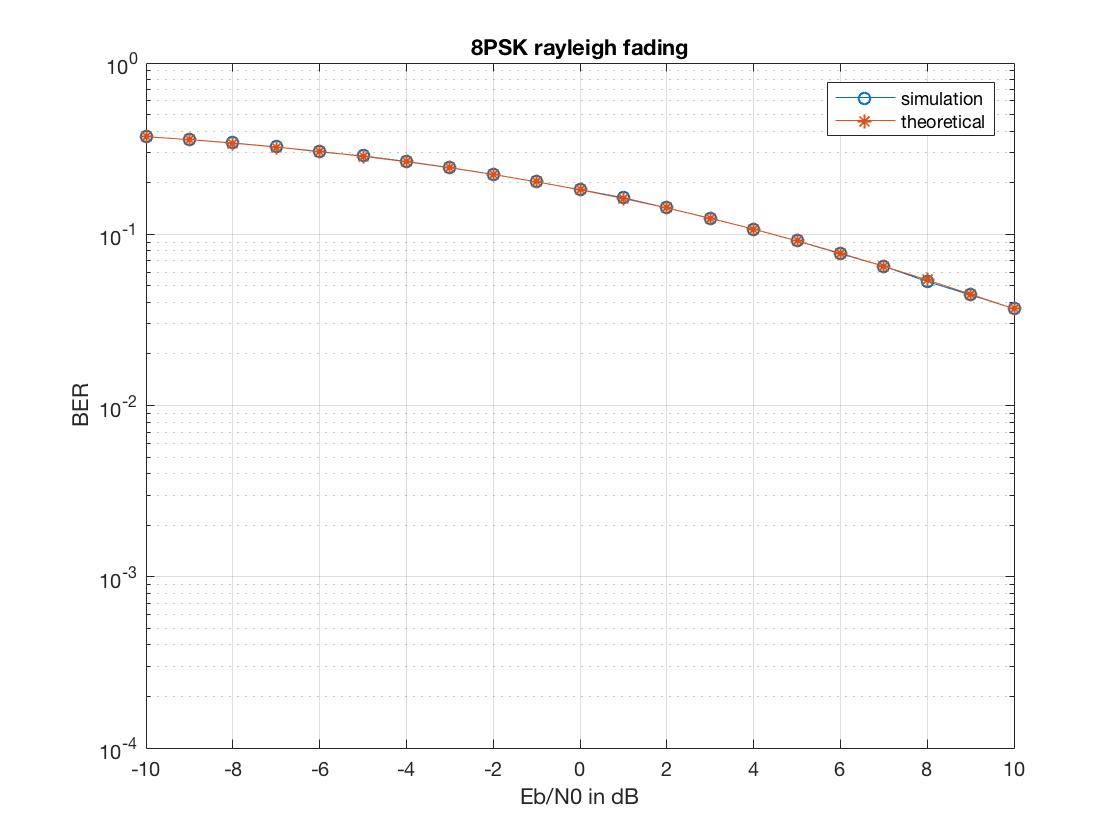
z依旧为高斯随机变量。

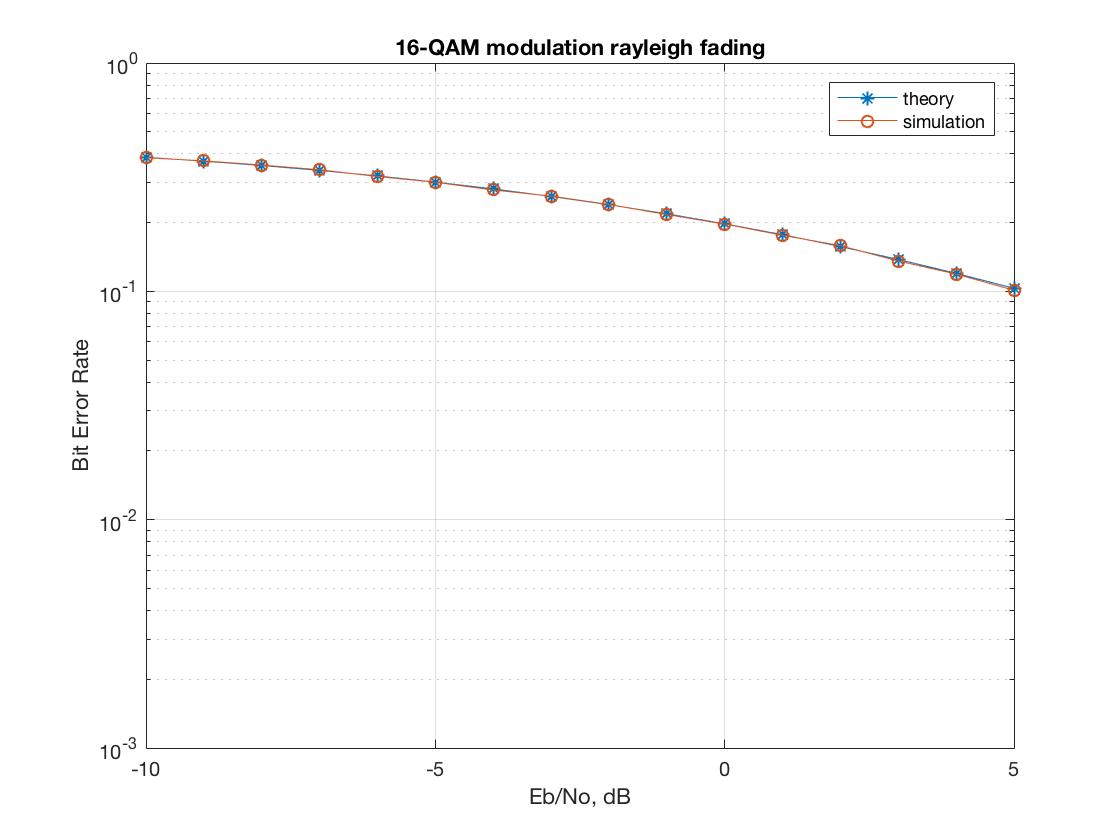
仿真中分集阶数皆为1.采用如上式所示方法进行仿真。

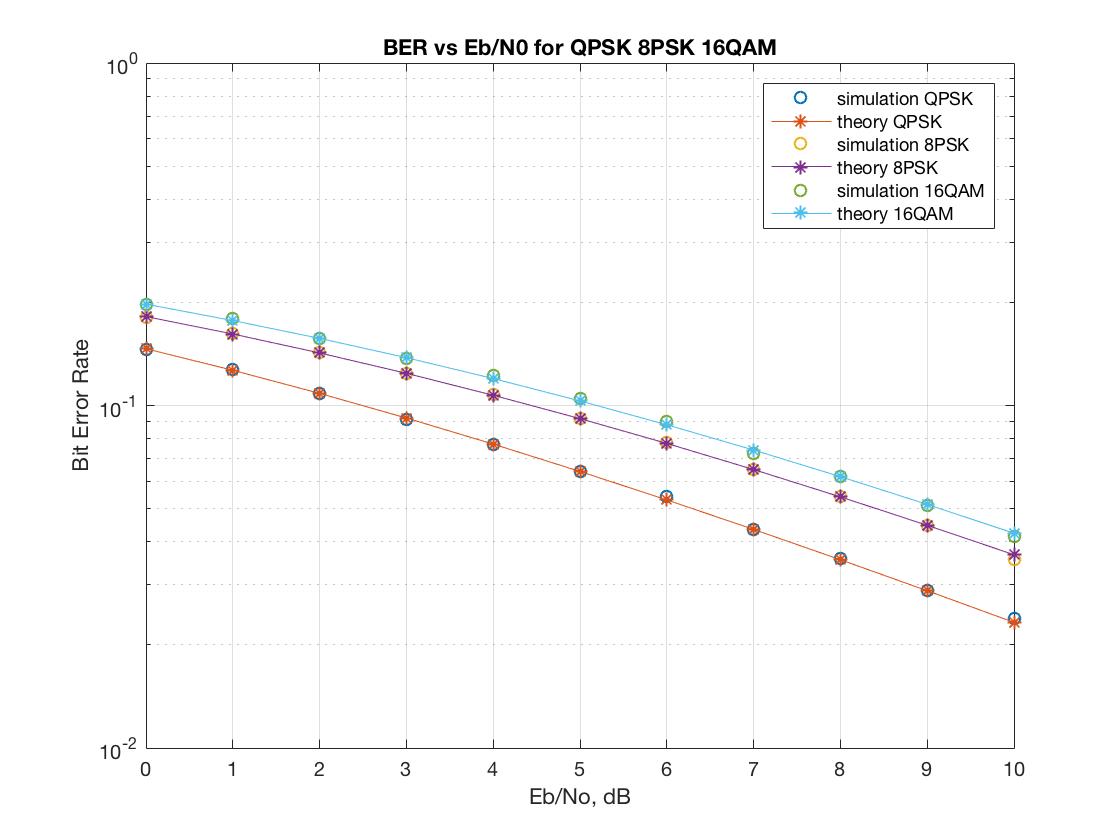
QPSK仿真结果：



结果接近理论值。与AWGN信道相比误比特率大大增加。







说明：相同信噪比时，三种调制方式在瑞利信道下依然具有和AWGN信道相同的误码率顺序。与AWGN信道相比误比特率高了很多。由于没有采用分集，加性高斯噪声使得解调后相位和幅度发生改变。

**实验五 接收分集性能仿真——仿真SIMO系统性能**

延续实验二，但在接收端有若干路独立信号，通过最大比合并，等增益合并和选择性合并，对瑞利信道进行仿真。

系统模型

输入－输出关系，M为接收天线数，s为发送符号，Es为符号能量，



假定噪声矢量的均值为0，协方差矩阵为N0/2\*Im.即各路噪声～iid复高斯。

h中的元素是0均值、圆对称的复高斯随机变量。

berfading函数采用理论上最优的最大比合并方式进行合并。所以首先比较理论与仿真中天线增加（分集阶数增大），BER vs EbN0曲线的变化。



最大比合并：^2

等增益合并：

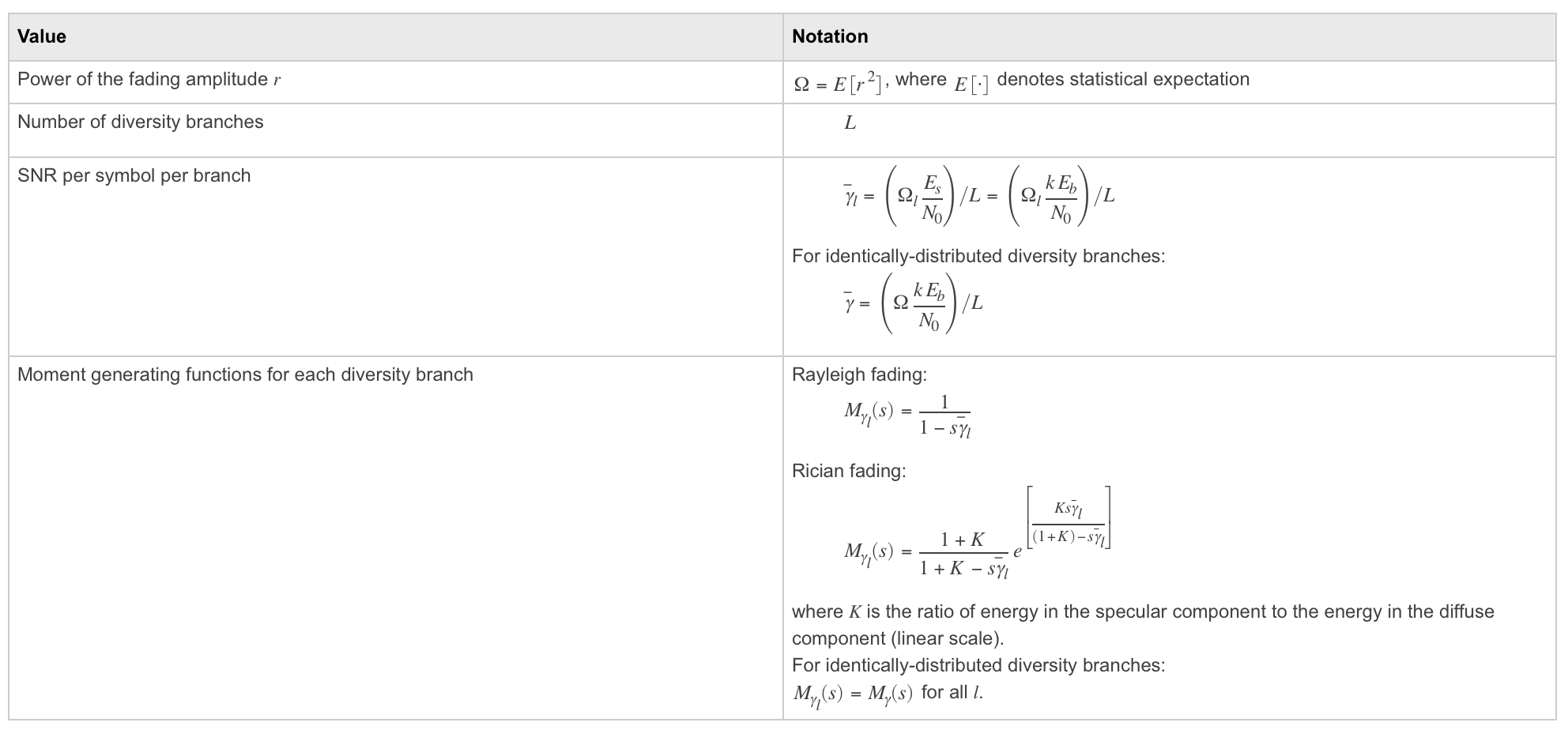
选择性合并：选择信噪比最大的支路

仿真中应注意：

MRC maximal-ratio combining（最大比合并）

EGC equal-gain combining（等增益合并）

在matlab文档中以显式表达式计算BER方法：



需对Es／N0的计算进行修正。

对于M-PSK M-QAM，berfading采用MRC（最大比合并）方式进行分集。

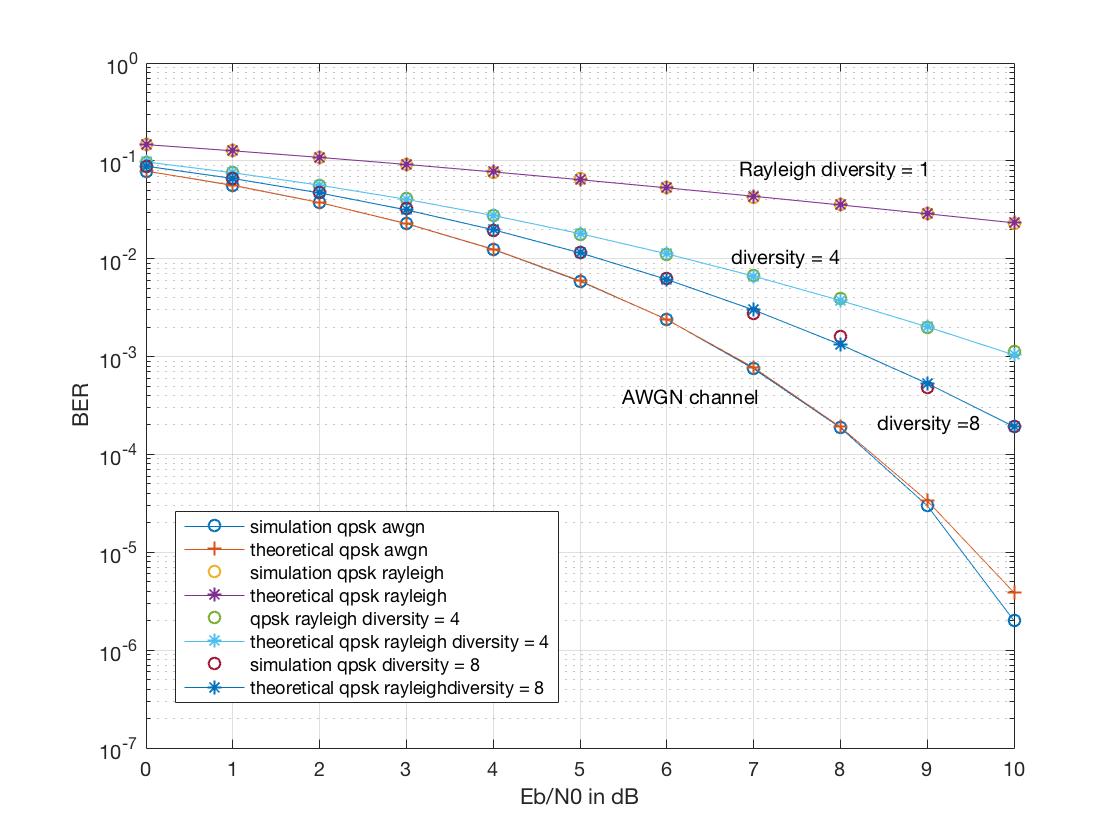
因此每一路信号加权值为abs(hi).^2

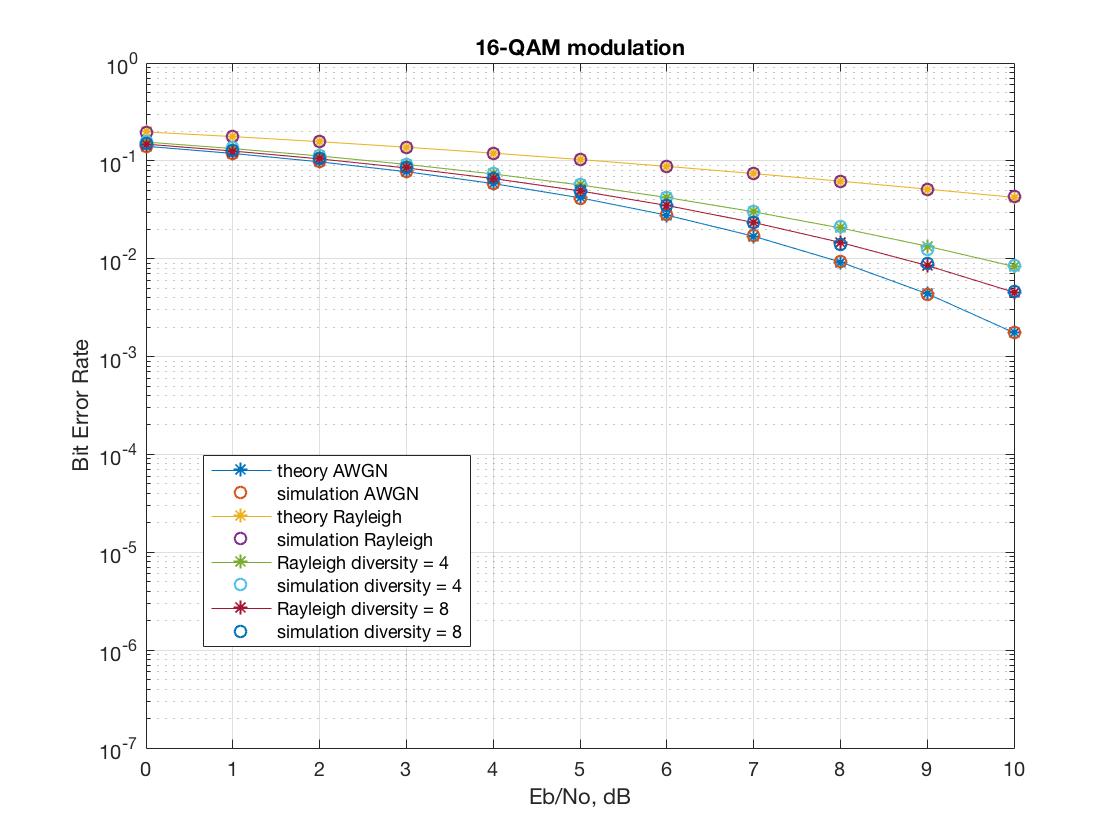
由于使用分集，仿真中每一路信号的Eb／N0应相应减小为Eb／(N0\*分集阶数)。

最大比合并中由于仿真产生的高斯噪声能量并不一定精确为1。因此加权系数应为abs(hi)^2/sigma^2

仿真结果：

QPSK调制：



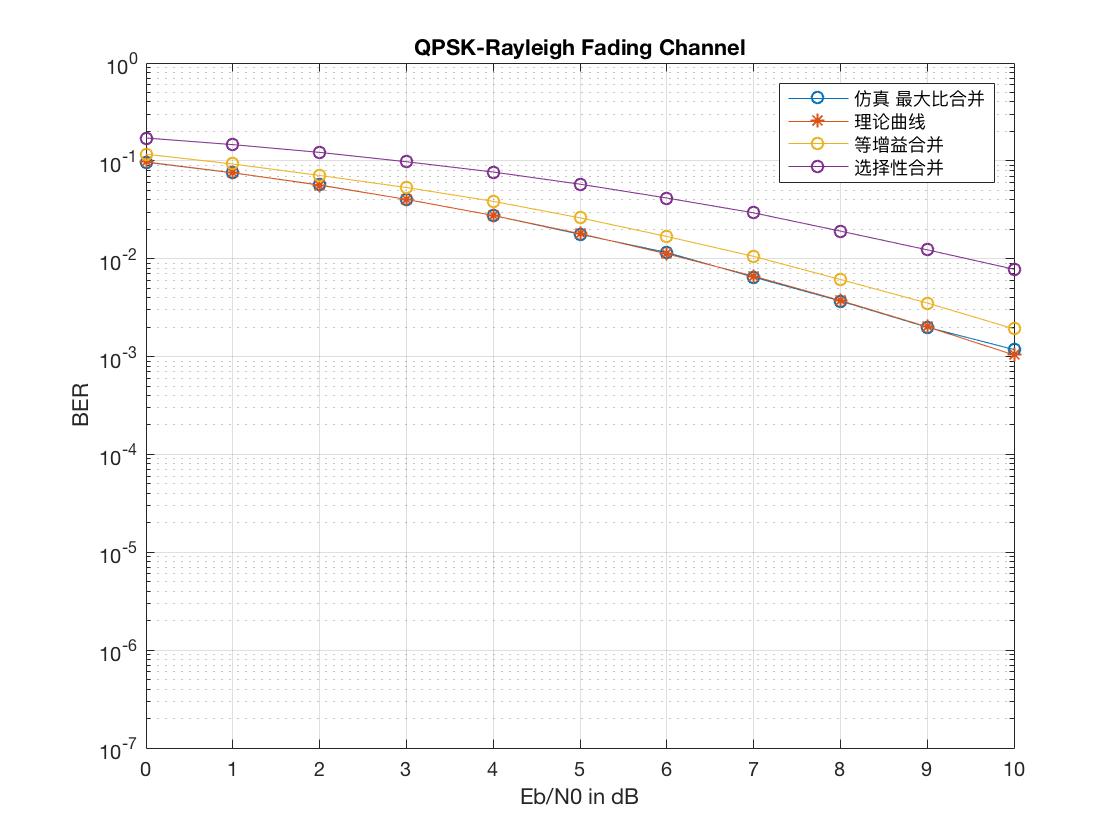


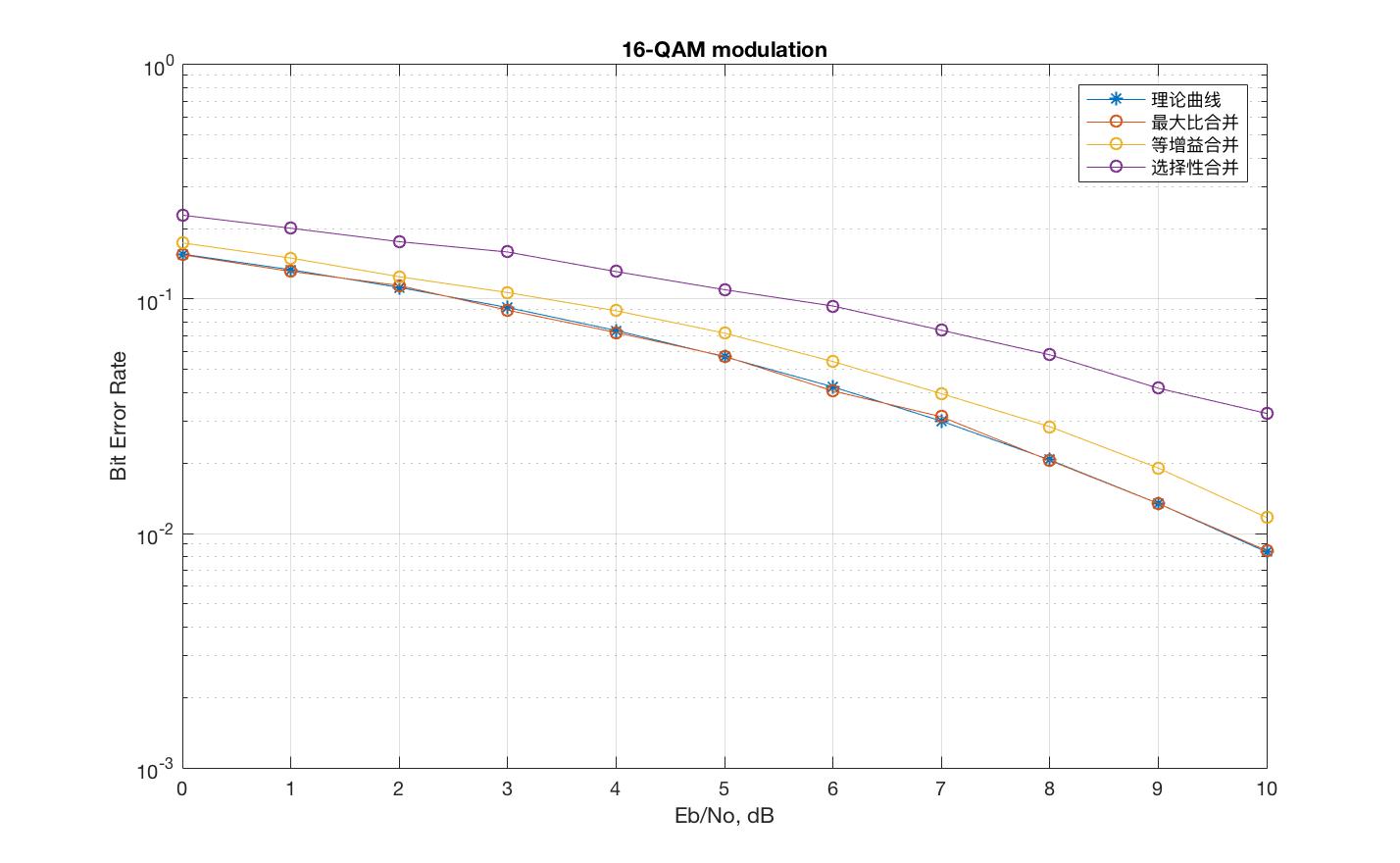
不同调制方式以及不同分集阶数都与理论曲线相符合。分集阶数越大则误比特率越小。阶数趋于无穷时曲线接近AWGN误比特率。



不同调制在三种合并方式下的仿真。由于方法相近，取QPSK和16QAM进行仿真。

分集阶数取4.





上图16QAM仿真依然采用4重分集。纵轴有改变。可以看出高信噪比下瑞利衰落信道依然具有较大的误比特率，大于10^-3 且大于QPSK调制误比特率。

三种合并方式中最大比合并具有最好的特性曲线。理论上也可以证明这一点。

而其它两种合并方式抛弃了部分接收端的信息以及关于信道的信息，导致误比特率大于最大比合并方式。但随着分集阶数增加，三种方式之间差距变小。

五、实验总结

本次物理层的仿真实验主要是通过matlab进行仿真并与理论进行比较。

本次实验首先采用给定的算法产生均匀分布随机变量。通过逆变换法产生具有特定概率密度分布函数的随机序列。并以此作为仿真中的噪声和信号源。

第二个实验对BPSK、QPSK、8PSK和16QAM等通信系统进行了仿真，并且得到了仿真曲线和理论值的仿真曲线。

选做实验五延续实验二对瑞利信道进行仿真，加入了不同的分集方式。并与理论结果进行比较。

总体上本次实验较为满意。因为仿真得到了十分满意的结果。在误差允许范围内结果即为理论上的BER vs Eb/N0曲线。有时理论与仿真结果已经十分接近，但一步步排查错误，最终得到了更加逼近、十分漂亮的误比特率曲线。

通过这次实验，对通信系统整体上又有了更深的理解。仿真中用到了复信号与复高斯分布的噪声，对仿真有着很大的简化作用。巩固了随机信号分析课程中关于瑞利分布，复高斯分布等的知识。也真正理解了什么是相位均衡以及无线通信课程上所学的各种分集方式。

Matlab部分则需注意要想取得与理论值十分接近的仿真结果，必须使用与帮助文档中相同的编码方式和译码规则。如berfading函数中采用的是最大比合并，需要给定分集阶数，16QAM采用的是格雷码编码，分集时默认不同信道具有不同衰减系数，分集各支路码元噪声比应成比例减小等。同时函数输出值为误比特率而不是误码率，输入参数为Eb／N0.Eb为每一比特对应能量。N0为复高斯噪声功率谱密度。

为提高仿真效率，除了不同信噪比时使用了for循环，其余部分均使用矩阵运算进行处理。for循环中数组应提前开辟空间。

误比特率产生平台则一定是解调中出现问题。如8PSK解调时相位判断中有一个码元逻辑判断应与其它7个码元不同。

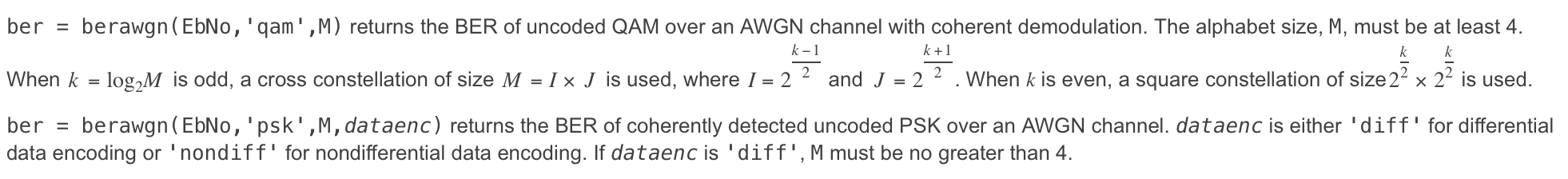
16QAM解调后应与原信号具有相同的幅度值范围（分集系数应进行归一化）。

而PSK根据相位进行判断，不需要进行幅度值的调整。

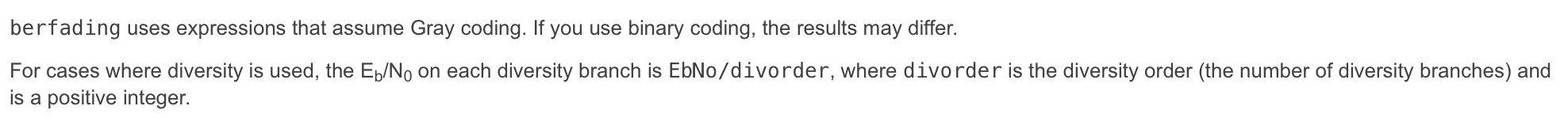
**附录：**

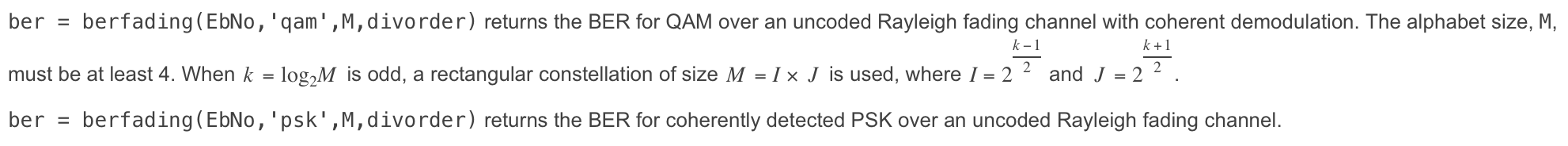
**函数说明**

实验中反复用到的理论曲线计算函数为berawgn和berfading分别给出AWGN信道和瑞利衰落信道的误比特率。



**berfading:**

****

****

**berfading采用格雷码且若采用分集需相应调整每一路的信噪比。**

其余函数详见matlab documentation。

**部分代码**

QPSK 8重分集

clear all;

EbN0dB = 0:10;%设定信噪比

N = 10^5;

M = 4;

k = log2(M);

divorder = 8; %分集阶数

sigma2 = 1./( 2 \*10.^(EbN0dB/10) );

ber = zeros(size(EbN0dB));

EsN0dB = EbN0dB + 10\*log10(k);

%EsN0dB = EsN0dB - 10\*log10(divorder);

for i = 1:length(EbN0dB)

errCount = 0;

totalN = 0;

% rand1 = wichhillrand(N);

% rand2 = wichhillrand(N);

rand1 = rand(1,N);

rand2 = rand(1,N);

h = sqrt(0.5) \* (randn(divorder,N) + 1j\*randn(divorder,N) );

E = mean(abs(h).^2);

bitsx = zeros([1,N]);

bitsx(rand1>0.5) = 1;

bitsy = zeros([1,N]);

bitsy(rand2>0.5) = 1;

symsx = 1 - 2 \* bitsx;

symsy = 1 - 2 \* bitsy;

s = symsx + 1j \* symsy;

s = s./sqrt(2); % normalization

%modulation

%pass

n = 1/sqrt(2)\*[randn(divorder,N) + 1j\*randn(divorder,N)]; % white guassian noise, 0dB variance

y = ones(divorder,1)\* s .\* h + 10^(-EsN0dB(i)/20)\*sqrt(divorder).\*n; % additive white gaussian noise

y = y./(h);

y = y.\*abs(h).^2;

y = sum(y);

%demode

y\_re = real(y).\*sqrt(2);

y\_im = imag(y).\*sqrt(2);

demodx = zeros(1,N);

demodx(y\_re < 0) = 1;

demody = zeros([1,N]);

demody(y\_im < 0) = 1;

temp = sum(demodx ~= bitsx ) + sum(demody ~= bitsy);%错误比特数

errCount = errCount + temp;

totalN = totalN + N;

%end

ber(i) = errCount/(totalN \* k);

end

semilogy(EbN0dB, ber, 'o');%纵轴采用对数坐标

hold on;

grid on;

bertheory = berfading(EbN0dB,'psk',4,divorder);%最后一个参数为分集阶数

semilogy(EbN0dB, bertheory, '-\*');

axis([EbN0dB(1), EbN0dB(end), 10^-7 1])

xlabel('Eb/N0 in dB')

ylabel('BER')

legend('simulation','theoretical')

16QAM 8重分集：

clear

N = 10^4; % number of symbols

M = 16; % constellation size

k = log2(M); % bits per symbol

divorder = 8;%diversity order

% defining the real and imaginary PAM constellation

% for 16-QAM

alphaRe = [-3 -1 1 3];

alphaIm = [-3 -1 1 3];

k\_16QAM = 1/sqrt(10);

Eb\_N0\_dB = 0:10;

EsN0dB = Eb\_N0\_dB + 10\*log10(k);

% Mapping for binary <--> Gray code conversion

ref = [0:k-1];

%map = bitxor(ref,floor(ref/2));

%[tt, ind] = sort(map);

ind = [1 2 4 3];

for i = 1:length(Eb\_N0\_dB)

% symbol generation

% ------------------

h = sqrt(0.5) \* (randn(divorder,N) + 1j\*randn(divorder,N) );

E = mean(abs(h).^2);

ipBit = rand(1,N\*k,1)>0.5; % random 1's and 0's

ipBitReshape = reshape(ipBit,k,N).';%转化为4列

% conversion from binary to decimal每行为2 1的列向量

bin2DecMatrix = ones(N,1)\*(2.^[(k/2-1):-1:0]) ;

% real

ipBitRe = ipBitReshape(:,[1:k/2]);%取出第1、2列

ipDecRe = sum(ipBitRe.\*bin2DecMatrix,2);

ipGrayDecRe = bitxor(ipDecRe,floor(ipDecRe/2));%将0123变为0132

% imaginary

ipBitIm = ipBitReshape(:,[k/2+1:k]);

ipDecIm = sum(ipBitIm.\*bin2DecMatrix,2);

ipGrayDecIm = bitxor(ipDecIm,floor(ipDecIm/2));

% mapping the Gray coded symbols into constellation

modRe = alphaRe(ipGrayDecRe+1);%映射为 -3 -1 1 3

modIm = alphaIm(ipGrayDecIm+1);

% complex constellation

mod = modRe + 1i\*modIm;

s = k\_16QAM\*mod; % normalization of transmit power to one

% noise

% -----

n = 1/sqrt(2)\*[randn(divorder,N) + 1j\*randn(divorder,N)]; % white guassian noise, 0dB variance

y = ones(divorder,1)\* s .\* h + 10^(-EsN0dB(i)/20)\*sqrt(divorder).\*n; % additive white gaussian noise

y = y./(h);

y = y.\*abs(h).^2;

y = sum(y);

y = y./sum(abs(h).^2) ;

y\_re = real(y)/k\_16QAM; % real part

y\_im = imag(y)/k\_16QAM; % imaginary part

% rounding to the nearest alphabet 将收端信号规整到[-3,3]

ipHatRe = 2\*floor(y\_re/2)+1;

ipHatRe(ipHatRe>max(alphaRe)) = max(alphaRe);

ipHatRe(ipHatRe<min(alphaRe)) = min(alphaRe);

ipHatIm = 2\*floor(y\_im/2)+1;

ipHatIm(ipHatIm>max(alphaIm)) = max(alphaIm);

ipHatIm(ipHatIm<min(alphaIm)) = min(alphaIm);

% Constellation to Decimal conversion

ipDecHatRe = ind(floor((ipHatRe+4)/2+1))-1; % LUT based 映射到[1.5,4.5] 取floor

ipDecHatIm = ind(floor((ipHatIm+4)/2+1))-1; % LUT based

% converting to binary string 十进制转化为二进制

ipBinHatRe = dec2bin(ipDecHatRe,k/2);

ipBinHatIm = dec2bin(ipDecHatIm,k/2);

% converting binary string to number

ipBinHatRe = ipBinHatRe.';

ipBinHatRe = ipBinHatRe(1:end).';

ipBinHatRe = reshape(str2num(ipBinHatRe).',k/2,N).' ;

ipBinHatIm = ipBinHatIm.';

ipBinHatIm = ipBinHatIm(1:end).';

ipBinHatIm = reshape(str2num(ipBinHatIm).',k/2,N).' ;

% counting errors for real and imaginary

nBitErr(i) = size(find([ipBitRe- ipBinHatRe]),1) + size(find([ipBitIm - ipBinHatIm]),1) ;

end

simBer = nBitErr/(N\*k);

%theoryBer = (1/k)\*3/2\*erfc(sqrt(k\*0.1\*(10.^(Eb\_N0\_dB/10))));

theoryBer = berfading(Eb\_N0\_dB,'qam',M,divorder);

semilogy(Eb\_N0\_dB,theoryBer,'-\*');

hold on

semilogy(Eb\_N0\_dB,simBer,'o');

axis([Eb\_N0\_dB(1), Eb\_N0\_dB(end), 10^-7 1])

grid on

legend('theory', 'simulation');

xlabel('Eb/No, dB')

ylabel('Bit Error Rate')

title('16-QAM modulation')

瑞利衰落信道 8PSK

clear all;

EbN0dB = 0:10;%设定信噪比

N = 10^5;

sigma2 = 1./( 3 \* 2 \* 10.^(EbN0dB/10) );

ber = zeros(size(EbN0dB));

decibound = 1/8 : 1/4 : 7/8;

decibound = [decibound, -7/8:1/4:-1/8];

decibound = decibound \* pi;

for i = 1:length(EbN0dB)

errCount = 0;

totalN = 0;

% rand1 = wichhillrand(N);

% rand2 = wichhillrand(N);

rand1 = rand(1,N);

rand2 = rand(1,N);

rand3 = rand(1,N);

h = sqrt(0.5) \* (randn(1,N) + 1j\*randn(1,N) );

bitsx = zeros(1,N);

bitsx(rand1>0.5) = 1;

bitsy = zeros([1,N]);

bitsy(rand2>0.5) = 1;

bitsz = zeros([1,N]);

bitsz(rand3>0.5) = 1;

phaseIQ = zeros(1,N);

phaseIQ(bitsx == 0 & bitsy == 0 & bitsz == 0) = 0;

phaseIQ(bitsx == 0 & bitsy == 0 & bitsz == 1) = 1;

phaseIQ(bitsx == 0 & bitsy == 1 & bitsz == 1) = 2;

phaseIQ(bitsx == 0 & bitsy == 1 & bitsz == 0) = 3;

phaseIQ(bitsx == 1 & bitsy == 1 & bitsz == 0) = 4;

phaseIQ(bitsx == 1 & bitsy == 1 & bitsz == 1) = 5;

phaseIQ(bitsx == 1 & bitsy == 0 & bitsz == 1) = 6;

phaseIQ(bitsx == 1 & bitsy == 0 & bitsz == 0) = 7;

phaseIQ = phaseIQ/4 \* pi;

complexsignal = exp(1j \* phaseIQ );

complexsignal = complexsignal .\* h;

I = real(complexsignal);

Q = imag(complexsignal);

%modulation

%pass

receiverI = I +sqrt(sigma2(i)) \* randn(1,N) ;

receiverQ = Q +sqrt(sigma2(i)) \* randn(1,N) ;

receiverIQ = (receiverI + 1j \* receiverQ)./h;

receiverI = real(receiverIQ);

receiverQ = imag(receiverIQ);

%demode

dephase = angle(receiverI + 1j \* receiverQ);

demodxyz = zeros(1,N);

demodxyz(dephase >= decibound(8) & dephase < decibound(1)) = 0;

demodxyz(dephase >= decibound(1) & dephase < decibound(2)) = 1;

demodxyz(dephase >= decibound(2) & dephase < decibound(3)) = 2;

demodxyz(dephase >= decibound(3) & dephase < decibound(4)) = 3;

demodxyz(dephase >= decibound(4) | dephase < decibound(5)) = 4;

demodxyz(dephase >= decibound(5) & dephase < decibound(6)) = 5;

demodxyz(dephase >= decibound(6) & dephase < decibound(7)) = 6;

demodxyz(dephase >= decibound(7) & dephase < decibound(8)) = 7;

xyz = zeros(3, length(demodxyz));

xyz(:, demodxyz == 0) = [0; 0; 0] \* ones(1,sum(demodxyz == 0));

xyz(:, demodxyz == 1) = [0; 0; 1] \* ones(1,sum(demodxyz == 1));

xyz(:, demodxyz == 2) = [0; 1; 1] \* ones(1,sum(demodxyz == 2));

xyz(:, demodxyz == 3) = [0; 1; 0] \* ones(1,sum(demodxyz == 3));

xyz(:, demodxyz == 4) = [1; 1; 0] \* ones(1,sum(demodxyz == 4));

xyz(:, demodxyz == 5) = [1; 1; 1] \* ones(1,sum(demodxyz == 5));

xyz(:, demodxyz == 6) = [1; 0; 1] \* ones(1,sum(demodxyz == 6));

xyz(:, demodxyz == 7) = [1; 0; 0] \* ones(1,sum(demodxyz == 7));

demodx = xyz(1,:);

demody = xyz(2,:);

demodz = xyz(3,:);

temp = sum(demodx ~= bitsx ) + sum(demody ~= bitsy) + sum((demodz ~= bitsz));

errCount = errCount + temp;

totalN = totalN + N \* 3;

%end

ber(i) = errCount/totalN;

end

semilogy(EbN0dB, ber , 'o');%纵轴采用对数坐标

hold on;

grid on;

bertheory = berfading(EbN0dB,'psk',8,1);

semilogy(EbN0dB, bertheory, '-\*');

axis([EbN0dB(1), EbN0dB(end), 10^-2 1])

xlabel('Eb/N0 in dB')

ylabel('BER')

legend('simulation','theoretical')