数字通信第二次作业-调制

姓名：王浩

学号：MG20230081

专业：通信与信息系统

## 一．基本概念

1. 正交相移键控（Quadrature Phase Shift Keying，QPSK）：是一种四进制相位调制，具有良好的抗噪特性和频带利用率。

2．正交幅度调制（Quadrature Amplitude Modulation，QAM）：16QAM是指包含16种符号的QAM调制方式。16QAM 是用两路独立的正交 4ASK 信号叠加而成，4ASK 是用多电平信号去键控载波而得到的信号。它是 2ASK 调制的推广，和 2ASK 相比，这种调制的优点在于信息传输速率高。

## 二．运行结果

### 2.1误比特率-信噪比（Eb/N0）



图1.通过AWGN信道，QPSK和16QAM误码率性能曲线

**结果分析：**从上图可以看出，无论是QPSK还是16QAM，信噪比越高，被误判的概率就越小，系统的误比特率就越低；对于相同信噪比情况下，QPSK的误比特率要比16QAM的误比特率要低，可以得出QPSK调制系统的可靠性要优于16QAM，原因是16QAM的相邻码元的最小欧式距离小于QPSK。但在相同的码元传输速率下，16QAM的比特率是QPSK的两倍，可见16QAM是牺牲系统的可靠性换取有效性，而且16QAM系统在信噪比为14dB左右时，误码率也达到了10-6的数量级，这对于大部分通信系统来说完全是可以接受的。

### 2.2误码率-信噪比（Es/N0）



**结果分析：**从上图可以看出，在相同信噪比的情况下，QPSK的抗噪声性能仍然优于16QAM，计算误码率的过程与计算误比特率稍有区别。在计算误码率时，在一个码元内只要有一个比特接收错误，就认为整个码元接收错误。因此，一般情况下，接收到同一组信息序列，计算它的误码率要高于误比特率。

## 三．附录代码

### 3.1误比特率-信噪比（Eb/N0）

clc;clear all;close all;

N = 10000000;

s = source(N); %信源产生，序列个数为N

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%16QAM

Eb = 2.5;%16QAM每个比特能量

mu = 0;

SNR = 0 :1 : 16;

BER = zeros(1,length(SNR));

N0 = Eb./(power(10,SNR/10));

sigma = sqrt(N0/2); %计算噪声的标准差

for i = 1:length(sigma)

n = normrnd(mu,sigma(i),[2,N/4]); %产生服从高斯分布的双路噪声

n\_c = n(1,:);n\_s = n(2,:);

s1 = zeros(4,N/4);

for c = 1:N/4

s1(1,c) = s(4\*c-3);

s1(2,c) = s(4\*c-2);

s1(3,c) = s(4\*c-1);

s1(4,c) = s(4\*c);

end %将信源分解成四路信号

[s\_c,s\_s] = QAM(s1); %进行16QAM编码

r\_c = s\_c + n\_c;r\_s = s\_s + n\_s;

y = judgement\_16QAM(r\_c,r\_s); %%16QAM解码，判决输出

BER(i) = error\_rate(s,y); %%求误比特率

end

semilogy(SNR,BER,'b\*');

axis([0 16 10^-6 0.15]);

hold on;

grid on;

xlabel('SNR/dB');ylabel('BER');

title('BER-SNR,AWGN');

BER\_true = 1/4\*(3\*qfunc(sqrt(4/5\*Eb./N0))+2\*qfunc(3\*sqrt(4/5\*Eb./N0))+qfunc(5\*sqrt(4/5\*Eb./N0))); %16QAM理想误比特率

semilogy(SNR,BER\_true,'-m');

hold on

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%QPSK

Eb = 1/2;%QPSK每个比特能量为1/2

N0 = Eb./(power(10,SNR/10));

sigma = sqrt(N0/2); %计算噪声的标准差

for i = 1:length(sigma)

n = normrnd(mu,sigma(i),[2,N/2]); %产生服从高斯分布的双路噪声

n\_c = n(1,:);n\_s = n(2,:);

s1\_c = zeros(1,N/2);s1\_s = zeros(1,N/2);

for c = 1:N/2

s1\_c(c) = s(2\*c-1);

s1\_s(c) = s(2\*c);

end %将信源分解成双路信号

[s\_c,s\_s] = QPSK(s1\_c,s1\_s); %进行QPSK编码

r\_c = s\_c + n\_c;r\_s = s\_s + n\_s;

y = judgement\_QPSK(r\_c,r\_s); %%QPSK解码，判决输出

BER(i) = error\_rate(s,y); %%求误比特率

end

semilogy(SNR,BER,'rs');

hold on;

BER\_true = erfc(sqrt(Eb./N0))/2; %QPSK理想误比特率

semilogy(SNR,BER\_true,'-y');

hold on

legend('16QAM simulated','16QAM theoretical','QPSK simulated','QPSK theoretical');

### 3.2误码率-信噪比（Es/N0）

clc;clear all;

N = 10000000;

s = source(N); %信源产生，序列个数为N

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%16QAM

Es = 10;%16QAM每个符号能量

mu = 0;

SNR = -5 :1 :20;

SER = zeros(1,length(SNR));

N0 = Es./(power(10,SNR/10));

sigma = sqrt(N0/2); %计算噪声的标准差

for i = 1:length(sigma)

n = normrnd(mu,sigma(i),[2,N/4]); %产生服从高斯分布的双路噪声

n\_c = n(1,:);n\_s = n(2,:);

s1 = zeros(4,N/4);

for c = 1:N/4

s1(1,c) = s(4\*c-3);

s1(2,c) = s(4\*c-2);

s1(3,c) = s(4\*c-1);

s1(4,c) = s(4\*c);

end %将信源分解成四路信号

[s\_c,s\_s] = QAM(s1); %进行16QAM编码

r\_c = s\_c + n\_c;r\_s = s\_s + n\_s;

y = judgement\_16QAM(r\_c,r\_s); %%16QAM解码，判决输出

SER(i) = symbol\_error\_16QAM(s,y); %%求误符号率

end

semilogy(SNR,SER,'b\*');

axis([0 20 10^-6 1]);

hold on;

grid on;

xlabel('SNR/dB');ylabel('BER');

title('BER-SNR,AWGN');

SER\_true = 3\*qfunc(sqrt(Es/5./N0)) - 9/4\*(qfunc(sqrt(Es/5./N0)).^2); %16QAM理想误符号率

semilogy(SNR,SER\_true,'-m');

hold on

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%QPSK

Es = 1;%QPSK每个符号能量为1

N0 = Es./(power(10,SNR/10));

sigma = sqrt(N0/2); %计算噪声的标准差

for i = 1:length(sigma)

n = normrnd(mu,sigma(i),[2,N/2]); %产生服从高斯分布的双路噪声

n\_c = n(1,:);n\_s = n(2,:);

s1\_c = zeros(1,N/2);s1\_s = zeros(1,N/2);

for c = 1:N/2

s1\_c(c) = s(2\*c-1);

s1\_s(c) = s(2\*c);

end %将信源分解成双路信号

[s\_c,s\_s] = QPSK(s1\_c,s1\_s); %进行QPSK编码

r\_c = s\_c + n\_c;r\_s = s\_s + n\_s;

y = judgement\_QPSK(r\_c,r\_s); %%QPSK解码，判决输出

SER(i) = symbol\_error\_QPSK(s,y); %%求误符号率

end

semilogy(SNR,SER,'rs');

hold on;

SER\_true = erfc(sqrt(Es./N0/2)); %QPSK理想误符号率

semilogy(SNR,SER\_true,'-y');

hold on

legend('16QAM simulated','16QAM theoretical','QPSK simulated','QPSK theoretical');

### 3.3功能函数

#### 3.4.1 信源产生

function [S]=source(L)

S=rand(1,L);

for i=1:L

if S(i)<0.5

S(i)=0;

else

S(i)=1;

end

end

#### 3.4.2 QPSK调制

function [s\_c,s\_s] = QPSK(s1\_c,s1\_s)

N = length(s1\_c);

s\_c = zeros(1,N);s\_s = s\_c;

for i = 1:N

if s1\_c(i) == 0 && s1\_s(i) == 0

s\_c(i) = sqrt(2)/2;s\_s(i) = sqrt(2)/2;

elseif s1\_c(i) == 0 && s1\_s(i) == 1

s\_c(i) = -sqrt(2)/2;s\_s(i) = sqrt(2)/2;

elseif s1\_c(i) == 1 && s1\_s(i) == 1

s\_c(i) = -sqrt(2)/2;s\_s(i) = -sqrt(2)/2;

elseif s1\_c(i) == 1 && s1\_s(i) == 0

s\_c(i) = sqrt(2)/2;s\_s(i) = -sqrt(2)/2;

end

end

#### 3.4.3 QPSK解调

function [y]=judgement\_QPSK(r\_c,r\_s)

N=length(r\_c);M=2\*N;

y\_c=zeros(1,N);y\_s=zeros(1,N);

for i=1:N

a=(r\_c(i)-sqrt(2)/2)^2+(r\_s(i)-sqrt(2)/2)^2;

b=(r\_c(i)+sqrt(2)/2)^2+(r\_s(i)-sqrt(2)/2)^2;

c=(r\_c(i)+sqrt(2)/2)^2+(r\_s(i)+sqrt(2)/2)^2;

d=(r\_c(i)-sqrt(2)/2)^2+(r\_s(i)+sqrt(2)/2)^2;

e=min([a,b,c,d]);

if a==e

y\_c(i)=0;y\_s(i)=0;

else if b==e

y\_c(i)=0;y\_s(i)=1;

else if c==e

y\_c(i)=1;y\_s(i)=1;

else

y\_c(i)=1;y\_s(i)=0;

end

end

end

y(2\*i-1)=y\_c(i);y(2\*i)=y\_s(i);

end

#### 3.4.4 16QAM调制

function [s\_c,s\_s] = QAM(s)

N = length(s(1,:));

s\_c = zeros(1,N);s\_s = s\_c;

for i = 1:N

%同向分量的映射

if s(1,i) == 0

if s(2,i) == 0

s\_c(i) = 1;

else

s\_c(i) = 3;

end

else

if s(2,i) == 0

s\_c(i) = -1;

else

s\_c(i) = -3;

end

end

%正交分量的映射

if s(3,i) == 0

if s(4,i) == 0

s\_s(i) = -3;

else

s\_s(i) = -1;

end

else

if s(4,i) == 0

s\_s(i) = 3;

else

s\_s(i) = 1;

end

end

end

#### 3.4.4 16QAM解调

function [y]=judgement\_16QAM(r\_c,r\_s)

N=length(r\_c);

y = zeros(1,4\*N);

for i=1:N

x = zeros(1,16);

x(1) = Distance([r\_c(i), r\_s(i)], [-3,-3]);

x(2) = Distance([r\_c(i), r\_s(i)], [-1,-3]);

x(3) = Distance([r\_c(i), r\_s(i)], [1,-3]);

x(4) = Distance([r\_c(i), r\_s(i)], [3,-3]);

x(5) = Distance([r\_c(i), r\_s(i)], [-3,-1]);

x(6) = Distance([r\_c(i), r\_s(i)], [-1,-1]);

x(7) = Distance([r\_c(i), r\_s(i)], [1,-1]);

x(8) = Distance([r\_c(i), r\_s(i)], [3,-1]);

x(9) = Distance([r\_c(i), r\_s(i)], [-3,1]);

x(10) = Distance([r\_c(i), r\_s(i)], [-1,1]);

x(11) = Distance([r\_c(i), r\_s(i)], [1,1]);

x(12) = Distance([r\_c(i), r\_s(i)], [3,1]);

x(13) = Distance([r\_c(i), r\_s(i)], [-3,3]);

x(14) = Distance([r\_c(i), r\_s(i)], [-1,3]);

x(15) = Distance([r\_c(i), r\_s(i)], [1,3]);

x(16) = Distance([r\_c(i), r\_s(i)], [3,3]);

m = min(x);

for j = 1:16

if x(j) == m

break;

end

end

switch j

case 1

y(4\*i-3) = 1;

y(4\*i-2) = 1;

y(4\*i-1) = 0;

y(4\*i) = 0;

case 2

y(4\*i-3) = 1;

y(4\*i-2) = 0;

y(4\*i-1) = 0;

y(4\*i) = 0;

case 3

y(4\*i-3) = 0;

y(4\*i-2) = 0;

y(4\*i-1) = 0;

y(4\*i) = 0;

case 4

y(4\*i-3) = 0;

y(4\*i-2) = 1;

y(4\*i-1) = 0;

y(4\*i) = 0;

case 5

y(4\*i-3) = 1;

y(4\*i-2) = 1;

y(4\*i-1) = 0;

y(4\*i) = 1;

case 6

y(4\*i-3) = 1;

y(4\*i-2) = 0;

y(4\*i-1) = 0;

y(4\*i) = 1;

case 7

y(4\*i-3) = 0;

y(4\*i-2) = 0;

y(4\*i-1) = 0;

y(4\*i) = 1;

case 8

y(4\*i-3) = 0;

y(4\*i-2) = 1;

y(4\*i-1) = 0;

y(4\*i) = 1;

case 9

y(4\*i-3) = 1;

y(4\*i-2) = 1;

y(4\*i-1) = 1;

y(4\*i) = 1;

case 10

y(4\*i-3) = 1;

y(4\*i-2) = 0;

y(4\*i-1) = 1;

y(4\*i) = 1;

case 11

y(4\*i-3) = 0;

y(4\*i-2) = 0;

y(4\*i-1) = 1;

y(4\*i) = 1;

case 12

y(4\*i-3) = 0;

y(4\*i-2) = 1;

y(4\*i-1) = 1;

y(4\*i) = 1;

case 13

y(4\*i-3) = 1;

y(4\*i-2) = 1;

y(4\*i-1) = 1;

y(4\*i) = 0;

case 14

y(4\*i-3) = 1;

y(4\*i-2) = 0;

y(4\*i-1) = 1;

y(4\*i) = 0;

case 15

y(4\*i-3) = 0;

y(4\*i-2) = 0;

y(4\*i-1) = 1;

y(4\*i) = 0;

case 16

y(4\*i-3) = 0;

y(4\*i-2) = 1;

y(4\*i-1) = 1;

y(4\*i) = 0;

end

end