MC Practical Codes (Exp 1-6) in Octave

**Experiment 1** –

clc;

clear all;

close all;

pkg load statistics;

x = -5:0.001:5;

% ---------- Normal Distribution ----------

figure;

%Normal Distribution PDF for Varing st. dev.

subplot(2, 2, 1);

y1 = normpdf(x, 0, 0.1);

plot(x, y1, "linewidth", 2);

xlabel("x");

ylabel("pdf");

title("Normal Distribution PDF st. dev = 0.1");

subplot(2, 2, 2);

y2 = normpdf(x, 0, 0.5);

plot(x, y2, "linewidth", 2);

xlabel("x");

ylabel("pdf");

title("Normal Distribution PDF st. dev = 0.5");

subplot(2, 2, 3);

y3 = normpdf(x, 0, 1);

plot(x, y3, "linewidth", 2);

xlabel("x");

ylabel("pdf");

title("Normal Distribution PDF st. dev = 1.0");

subplot(2, 2, 4);

y4 = normpdf(x, 0, 2);

plot(x, y4, "linewidth", 2);

xlabel("x");

ylabel("pdf");

title("Normal Distribution PDF st. dev = 2.0");

figure;

%Normal Distribution PDF for Varing mean

subplot(2, 2, 1);

y1 = normpdf(x, -1, 0.5);

plot(x, y1, "linewidth", 2);

xlabel('x');

ylabel('pdf');

title('Normal Distribution PDF mean = -1')

subplot(2, 2, 2);

y2 = normpdf(x, 0, 0.5);

plot(x, y2, "linewidth", 2);

xlabel('x');

ylabel('pdf');

title('Normal Distribution PDF mean = 0')

subplot(2, 2, 3);

y3 = normpdf(x, 1, 0.5);

plot(x, y3, "linewidth", 2);

xlabel('x');

ylabel('pdf');

title('Normal Distribution PDF mean = 1')

subplot(2, 2, 4);

y4 = normpdf(x, 2, 0.5);

plot(x, y4, "linewidth", 2);

xlabel('x');

ylabel('pdf');

title('Normal Distribution PDF mean = 2')

% ---------- Uniform Distribution ----------

figure;

%Uniform Distribution PDF for Varing Mean

subplot(2,2,1);

y1 = unifpdf(x,1,2);

plot(x,y1,"linewidth",2);

xlabel('x');

ylabel('pdf');

title('Uniform Distribution PDF mean = 1')

subplot(2,2,2);

y2 = unifpdf(x,0,2);

plot(x,y2,"linewidth",2);

xlabel('x');

ylabel('pdf');

title('Uniform Distribution PDF mean = 0')

subplot(2,2,3);

y3 = unifpdf(x,-1,2);

plot(x,y3,"linewidth",2);

xlabel('x');

ylabel('pdf');

title('Uniform Distribution PDF mean = -1')

subplot(2,2,4);

y4 = unifpdf(x,-2,2);

plot(x,y4,"linewidth",2);

xlabel('x');

ylabel('pdf');

title('Uniform Distribution PDF mean = -2')

% ---------- Poisen Distribution ----------

figure;

%Poisen Distribution PDF for Varing Mean

subplot(2,2,1);

y1 = poisspdf(x,1);

plot(x,y1,"linewidth",2);

xlabel('x');

ylabel('pdf');

title('Poisen Distribution PDF mean = 1')

subplot(2,2,2);

y2 = poisspdf(x,2);

plot(x,y2,"linewidth",2);

xlabel('x');

ylabel('pdf');

title('Poisen Distribution PDF mean = 2')

subplot(2,2,3);

y3 = poisspdf(x,3);

plot(x,y3,"linewidth",2);

xlabel('x');

ylabel('pdf');

title('Poisen Distribution PDF mean = 3')

subplot(2,2,4);

y4 = poisspdf(x,10);

plot(x,y4,"linewidth",2);

xlabel('x');

ylabel('pdf');

title('Poisen Distribution PDF mean = 10')

% ---------- Exponential Distribution ----------

figure;

%Exponential Distribution PDF for Varing Mean

subplot(2,2,1);

y1 = exppdf(x,1);

plot(x,y1,"linewidth",2);

xlabel('x');

ylabel('pdf');

title('Exponential Distribution PDF mean = 1')

subplot(2,2,2);

y2 = exppdf(x,2);

plot(x,y2,"linewidth",2);

xlabel('x');

ylabel('pdf');

title('Exponential Distribution PDF mean = 2')

subplot(2,2,3);

y3 = exppdf(x,3);

plot(x,y3,"linewidth",2);

xlabel('x');

ylabel('pdf');

title('Exponential Distribution PDF mean = 3')

subplot(2,2,4);

y4 = exppdf(x,10);

plot(x,y4,"linewidth",2);

xlabel('x');

ylabel('pdf');

title('Exponential Distribution PDF mean = 10')

% ---------- Experiment 1 - Part B ----------

clc;

clear all;

close all;

%Auto Correlation function and Power Spectral Density

pkg load statistics;

y=normrnd(0,1,1,200);

Gy=periodogram(y);

Ry=abs(ifft(Gy,256));

Ry=[Ry(130:256)' Ry(1:129)'];

t=-127:1:128;

figure;

subplot(1,2,1);

plot(Gy);

xlabel('frequencysamples');

title('PSD');

subplot(1,2,2);

stem(t,Ry);

xlabel('time shift');

title('autocorrection');

Experiment 2 –

clc;

clear;

close all;

n = input("Enter the no. of Source Elements : ");

q = input("Enter the channel matrix P(Y/X) : ");

%matrix p(Y/X)

disp(q);

disp(' ');

N=1:n;

p = input("Enter the source Probability : ");

%Probabilitys for x

px= diag(p,n,n);

%matrix P(X)

disp("P(X) :");

disp(px);

disp(' ');

pxy=px\*q;

% P(X,Y)=P(X)\*P(Y|X)

disp("P(X,Y) : ");

disp(pxy);

disp(' ');

% P(Y)

py=p\*q;

disp('P(Y):');

disp(py);

disp(' ');

%Entropy of source h(x)

Hx = 0 ;

for i = 1:n

Hx=Hx+(-(p(i)\*log2(p(i))));

end

disp('H(x): ');

disp(Hx);

disp(' ');

% Entropy of destination H(y)

Hy = 0;

for i = 1:n

Hy=Hy+(-(py(i)\*log2(py(i))));

end

disp('Entropy of destination H(y): ');

disp(Hy);

disp(' ');

% Mutual Entropy H(x,y)

hxy=0

for i = 1:n

for j = 1:n

hxy=hxy+(-pxy(i,j) \* log2(pxy(i,j)) );

end

end

disp('Mutual Entropy H(x,y): ');

disp(hxy);

disp(' ');

% Conditional Entropy H(y/x)

h1 = hxy - Hx ;

disp('Conditional Entropy H(x/y): ');

disp(h1);

disp(' ');

% Conditional Entropy H(x/y)

h2= hxy - Hy;

disp('Conditional Entropy H(y/x): ');

disp(h2);

disp(' ');

% Mutual Information I(x,y)

lxy= Hx - h2;

disp('Mutual Information l(x,y): ');

disp(lxy);

disp(' ');

if h2==0

disp("This channel is a lossless channel ");

else

disp("This channel is not a lossless channel ");

end

if lxy==0

disp ("This channel is a useless channel ");

else

disp("This channel is not a useless channel ");

end

if Hx==Hy

if h1==0

disp("This channel is a noiseless channel ");

else

disp("This channel is not a noiseless channel ");

end

endif

**Experiment 3** –

%Halitman code using MATLAB/Octave. Determine Efficiency and redundancy for the given Source Coding technique. (D1)

clc;

close all;

clear all;

pkg load communications;

symbols = 1:4;

prob = [0.5 0.25 0.15 0.10];

disp("Symbols : ");

disp(symbols);

disp(" ");

disp("Probalities : ");

disp(prob);

disp(" ");

dict = huffmandict(symbols,prob);

disp("Huffman Dict. : ");

disp(dict);

disp(" ");

inputSig = randsrc(1,10,[symbols;prob]);

disp("Input Signal : ");

disp(inputSig);

disp(" ");

code = huffmanenco(inputSig,dict);

disp("Coded signal : ");

disp(code);

disp(" ");

decoded = huffmandeco(code,dict);

disp("Decoded signal : ");

disp(decoded);

disp(" ");

avg\_code\_len = 0;

for i = 1:length(symbols)

%disp(length(dict{i}));

%disp(Prob(i));

avg\_code\_len = avg\_code\_len + (prob(i)\*length(dict{i}));

end

disp("Avg code len (L) : ");

disp(avg\_code\_len);

disp(" ");

H = 0 ;

for i = 1:length(symbols)

H = H - sum(prob(i)\*log2(prob(i))) ;

end

disp("Entropy (H) : ");

disp(H);

disp(" ");

efficency = (H/avg\_code\_len)\*100;

disp("Efficiency : ");

disp(efficency);

disp("");

**Experiment 4** –

clc;

close all;

clear all;

n = input('Enter no of code bits: ');

k = input('Enter no of message bits: ');

p = input('Enter parity matrix: ');

disp('Parity matrix: ');

disp(p);

I = eye(k);

p =[1 0 1; 1 1 1;1 1 0; 0 1 1];

G =[I,p];

disp('Generator matrix');

disp(' G = [Ik : P]');

disp(G);

m = dec2bin(0:1:2^k-1);

disp('message words')

disp(m);

C = m \* G;

for i = 1:2^k

for j =1:n

C(i,j) = mod(C(i,j),2);

end

end

disp('Codewords are:');

disp(' C = mG');

disp(C);

weight = sum(C');

disp('Hamming weight of codes');

disp(weight');

weight(1,1) = weight(1,2);

d = min(weight);

disp('Minimum Hamming weight(dmin):')

disp(d);

Td = d - 1;

disp('td =');

disp('dmin - 1');

disp(Td);Tc = (d-1)/2;

disp('tc=');

disp('(dmin-1)/2');

disp(Tc);

H = [p',eye(n-k)];

disp(' H = [transpose(P):I(n-k)');

disp(H);

disp( 'H transpose')

disp(H');

E = eye([n,n]);

r = input('Enter recieved codeword(r):');

s = r\*H';

for i=1:n-k

s(1,i) = mod(s(1,i),2);

end

h = H';

disp('Syndrome is (S):');

disp('S = r \* transpose(H)');

disp(s);

if(s==[0 0 0])

disp ('valid code word');

else

for i=1:n

if(s ==h(i,:))

error = i;

disp('error is in')

disp(i);

break;

end

end

end

disp('Error pattern(e)= ');

disp(E(error,:));

c = mod(r + E(error,:),2);

disp('error corrected code word');

disp('C = r + e');

disp(c);

**Experiment 5-A** –

% Write a program to plot signal constellation diagram of received 8-PSK signal in the presence of AWGN.

clc;

close all;

clear all;

pkg load communications

N = 3000; % Number of bits to be transmitted using \*-PSK

% Too large value may slow down the program

x = randi([0,1],1,N); % Random input bits generation

M = 8; % Number of Symbols in 8-PSK

% Symbol Generation

yy = [];

for i=1:3:length(x)

if x(i)==0 && x(i+1)==0 & x(i+2)==0

y = cosd(0)+1j\*sind(0);

elseif x(i)==0 && x(i+1)==0 & x(i+2)==1

y = cosd(45)+1j\*sind(45);

elseif x(i)==0 && x(i+1)==1 & x(i+2)==1

y = cosd(90)+1j\*sind(90);

elseif x(i)==0 && x(i+1)==1 & x(i+2)==0

y = cosd(135)+1j\*sind(135);

elseif x(i)==1 && x(i+1)==1 & x(i+2)==0

y = cosd(180)+1j\*sind(180);

elseif x(i)==1 && x(i+1)==1 & x(i+2)==1

y = cosd(225)+1j\*sind(225);

elseif x(i)==1 && x(i+1)==0 & x(i+2)==1

y = cosd(270)+1j\*sind(270);

elseif x(i)==1 && x(i+1)==0 & x(i+2)==0

y = cosd(315)+1j\*sind(315);

endif

% Transmitted Symbols

yy = [yy y];

endfor

scatterplot(yy); % Constellation Diagram without Noise

EbN0db = 20; % Change this value & run program to see the noisy constellation.

EbN0 = 10^(EbN0db/10);

% AWGN Channel

n = (1/sqrt(2))\*[randn(1,length(yy)) + 1j\*randn(1,length(yy))];

sigma = sqrt(1/((log2(M))\*EbN0));

% Received Symbols

r = yy + sigma\*n;

scatterplot(r); % Constellation Diagram with Noise

**Experiment 5-B** –

% Write a program to plot Practical and Theoretical BER vs SNR graph of received 8-PSK signal in the presence of AWGN for ML receiver.

clc;

clear all;

close all;

pkg load communications

N = 3000; % Number of bits to be transmitted using \*-PSK

% Too large value may slow down the program

x = randi([0,1],1,N); % Random input bits generation

M = 8; % Number of Symbols in 8-PSK

% Symbol Generation

yy = [];

for i=1:3:length(x)

if x(i)==0 && x(i+1)==0 & x(i+2)==0

y = cosd(0)+1j\*sind(0);

elseif x(i)==0 && x(i+1)==0 & x(i+2)==1

y = cosd(45)+1j\*sind(45);

elseif x(i)==0 && x(i+1)==1 & x(i+2)==1

y = cosd(90)+1j\*sind(90);

elseif x(i)==0 && x(i+1)==1 & x(i+2)==0

y = cosd(135)+1j\*sind(135);

elseif x(i)==1 && x(i+1)==1 & x(i+2)==0

y = cosd(180)+1j\*sind(180);

elseif x(i)==1 && x(i+1)==1 & x(i+2)==1

y = cosd(225)+1j\*sind(225);

elseif x(i)==1 && x(i+1)==0 & x(i+2)==1

y = cosd(270)+1j\*sind(270);

elseif x(i)==1 && x(i+1)==0 & x(i+2)==0

y = cosd(315)+1j\*sind(315);

endif

% Transmitted Symbols

yy = [yy y];

endfor

% Detection based on euclidean distance

ber\_simulated = [];

ber\_theoretical = [];

ref\_symbols = [cosd(0)+1j\*sind(0), cosd(45)+1j\*sind(45), cosd(90)+1j\*sind(90), cosd(135)+1j\*sind(135), cosd(180)+1j\*sind(180) cosd(225)+1j\*sind(225) cosd(270)+1j\*sind(270) cosd(315)+1j\*sind(315)];

for EbN0db = 0:15

EbN0 = 10^(EbN0db/10);

n = (1/sqrt(2))\*[randn(1,length(yy)) + 1j\*randn(1,length(yy))];

sigma = sqrt(1/((log2(M))\*EbN0)); % Symbol energy is normalized to Unity

r = yy + sigma\*n;

% Calculation of Euclidian Distances of received symbols from reference symobols

min\_dist\_index = [];

for i=1:length(r)

Dist = [];

for k=1:length(ref\_symbols)

dist = sqrt((real(r(i))-real(ref\_symbols(k)))^2 + (imag(r(i))-imag(ref\_symbols(k)))^2);

Dist = [Dist dist];

endfor

min\_dist\_index = [min\_dist\_index find(Dist==min(Dist))];

endfor

% Estimation of Bits

x\_estimated = [];

for i=1:length(r)

if ref\_symbols(min\_dist\_index(i))== cosd(0)+1j\*sind(0);

x\_estimated = [x\_estimated 0 0 0];

elseif ref\_symbols(min\_dist\_index(i))== cosd(45)+1j\*sind(45);

x\_estimated = [x\_estimated 0 0 1];

elseif ref\_symbols(min\_dist\_index(i))== cosd(90)+1j\*sind(90);

x\_estimated = [x\_estimated 0 1 1];

elseif ref\_symbols(min\_dist\_index(i))== cosd(135)+1j\*sind(135);

x\_estimated = [x\_estimated 0 1 0];

elseif ref\_symbols(min\_dist\_index(i))== cosd(180)+1j\*sind(180);

x\_estimated = [x\_estimated 1 1 0];

elseif ref\_symbols(min\_dist\_index(i))== cosd(225)+1j\*sind(225);

x\_estimated = [x\_estimated 1 1 1];

elseif ref\_symbols(min\_dist\_index(i))== cosd(270)+1j\*sind(270);

x\_estimated = [x\_estimated 1 0 1];

elseif ref\_symbols(min\_dist\_index(i))== cosd(315)+1j\*sind(315);

x\_estimated = [x\_estimated 1 0 0];

endif

endfor

% BER Computation

ber\_simulated =[ber\_simulated sum(x~=x\_estimated)/N];

ber\_theoretical = [ber\_theoretical (1/log2(M))\*erfc(sqrt(3\*EbN0)\*sind(180/M))];

endfor

EbN0db = 0:15;

% BER Plotting

semilogy(EbN0db, ber\_simulated, 'ro-', EbN0db, ber\_theoretical, 'k>-');

title('BER vs Eb/N0 Plot for 8-PSK');

xlabel('Eb/N0 (dB)');

ylabel('BER');

grid on;

legend('Simulated', 'Theoretical');

axis([0 15 10^-3 10^0]);

**Experiment 6-A** –

% Write a program to plot signal constellation diagram of received 16-Square QAM signal in the presence of AWGN.

clc;

clear all;

close all;

pkg load communications

N = 16000; % Number of bits to be transmitted using 16-Square QAM

% Too large value may slow down the program

x = randi([0,1],1,N); % Random input bits generation

M = 16; % Number of Symbols in 16-Square QAM

d = sqrt(2/5); % Average symbol energy is normalised to unity

% Symbol Generation

yy = [];

for i=1:4:length(x)

if x(i)==0 && x(i+1)==0 & x(i+2)==0 & x(i+3)==0

y = -3\*d/2+j\*(-3\*d/2);

elseif x(i)==0 && x(i+1)==0 & x(i+2)==0 & x(i+3)==1

y = -3\*d/2+j\*(-d/2);

elseif x(i)==0 && x(i+1)==0 & x(i+2)==1 & x(i+3)==1

y = -3\*d/2+j\*(d/2);

elseif x(i)==0 && x(i+1)==0 & x(i+2)==1 & x(i+3)==0

y = -3\*d/2+j\*(3\*d/2);

elseif x(i)==0 && x(i+1)==1 & x(i+2)==0 & x(i+3)==0

y = -d/2+j\*(-3\*d/2);

elseif x(i)==0 && x(i+1)==1 & x(i+2)==0 & x(i+3)==1

y = -d/2+j\*(-d/2);

elseif x(i)==0 && x(i+1)==1 & x(i+2)==1 & x(i+3)==1

y = -d/2+j\*(d/2);

elseif x(i)==0 && x(i+1)==1 & x(i+2)==1 & x(i+3)==0

y = -d/2+j\*(3\*d/2);

elseif x(i)==1 && x(i+1)==1 & x(i+2)==0 & x(i+3)==0

y = d/2+j\*(-3\*d/2);

elseif x(i)==1 && x(i+1)==1 & x(i+2)==0 & x(i+3)==1

y = d/2+j\*(-d/2);

elseif x(i)==1 && x(i+1)==1 & x(i+2)==1 & x(i+3)==1

y = d/2+j\*(d/2);

elseif x(i)==1 && x(i+1)==1 & x(i+2)==1 & x(i+3)==0

y = d/2+j\*(3\*d/2);

elseif x(i)==1 && x(i+1)==0 & x(i+2)==0 & x(i+3)==0

y = 3\*d/2+j\*(-3\*d/2);

elseif x(i)==1 && x(i+1)==0 & x(i+2)==0 & x(i+3)==1

y = 3\*d/2+j\*(-d/2);

elseif x(i)==1 && x(i+1)==0 & x(i+2)==1 & x(i+3)==1

y = 3\*d/2+j\*(d/2);

elseif x(i)==1 && x(i+1)==0 & x(i+2)==1 & x(i+3)==0

y = 3\*d/2+j\*(3\*d/2);

endif

% Transmitted Symbols

yy = [yy y];

endfor

scatterplot(yy); % Constellation Diagram without Noise

EbN0db = 20; % Change this value & run program to see the noisy constellation

EbN0 = 10^(EbN0db/10);

% AWGN Channel

n = (1/sqrt(2))\*[randn(1,length(yy)) + 1j\*randn(1,length(yy))];

sigma = sqrt(1/((log2(M))\*EbN0));

% Received Symbols

r = yy + sigma\*n;

scatterplot(r); % Constellation Diagram with Noise

**Experiment 6-B** –

% Write a program to plot Practical and Theoretical BER vs SNR graph of received 16-Square QAM signal in the presence of AWGN for ML receiver.

clc;

clear all;

close all;

pkg load communications

N = 4000; % Number of bits to be transmitted using \*-PSK

% Too large value may slow down the program

x = randi([0,1],1,N); % Random input bits generation

M = 16; % Number of Symbols in 16-Square QAM

d = sqrt(2/5); % Average symbol energy is normalised to unity

% Symbol Generation

yy = [];

for i=1:4:length(x)

if x(i)==0 && x(i+1)==0 & x(i+2)==0 & x(i+3)==0

y = -3\*d/2+j\*(-3\*d/2);

elseif x(i)==0 && x(i+1)==0 & x(i+2)==0 & x(i+3)==1

y = -3\*d/2+j\*(-d/2);

elseif x(i)==0 && x(i+1)==0 & x(i+2)==1 & x(i+3)==1

y = -3\*d/2+j\*(d/2);

elseif x(i)==0 && x(i+1)==0 & x(i+2)==1 & x(i+3)==0

y = -3\*d/2+j\*(3\*d/2);

elseif x(i)==0 && x(i+1)==1 & x(i+2)==0 & x(i+3)==0

y = -d/2+j\*(-3\*d/2);

elseif x(i)==0 && x(i+1)==1 & x(i+2)==0 & x(i+3)==1

y = -d/2+j\*(-d/2);

elseif x(i)==0 && x(i+1)==1 & x(i+2)==1 & x(i+3)==1

y = -d/2+j\*(d/2);

elseif x(i)==0 && x(i+1)==1 & x(i+2)==1 & x(i+3)==0

y = -d/2+j\*(3\*d/2);

elseif x(i)==1 && x(i+1)==1 & x(i+2)==0 & x(i+3)==0

y = d/2+j\*(-3\*d/2);

elseif x(i)==1 && x(i+1)==1 & x(i+2)==0 & x(i+3)==1

y = d/2+j\*(-d/2);

elseif x(i)==1 && x(i+1)==1 & x(i+2)==1 & x(i+3)==1

y = d/2+j\*(d/2);

elseif x(i)==1 && x(i+1)==1 & x(i+2)==1 & x(i+3)==0

y = d/2+j\*(3\*d/2);

elseif x(i)==1 && x(i+1)==0 & x(i+2)==0 & x(i+3)==0

y = 3\*d/2+j\*(-3\*d/2);

elseif x(i)==1 && x(i+1)==0 & x(i+2)==0 & x(i+3)==1

y = 3\*d/2+j\*(-d/2);

elseif x(i)==1 && x(i+1)==0 & x(i+2)==1 & x(i+3)==1

y = 3\*d/2+j\*(d/2);

elseif x(i)==1 && x(i+1)==0 & x(i+2)==1 & x(i+3)==0

y = 3\*d/2+j\*(3\*d/2);

endif

% Transmitted Symbols

yy = [yy y];

endfor

% Detection based on euclidean distance

ber\_simulated = [];

ber\_theoretical = [];

ref\_symbols = [-3\*d/2+j\*(-3\*d/2) -3\*d/2+j\*(-d/2) -3\*d/2+j\*d/2 -3\*d/2+j\*(3\*d/2) -d/2+j\*(-3\*d/2) -d/2+j\*(-d/2) -d/2+j\*d/2 -d/2+j\*(3\*d/2) d/2+j\*(-3\*d/2) d/2+j\*(-d/2) d/2+j\*d/2 d/2+j\*(3\*d/2) 3\*d/2+j\*(-3\*d/2) 3\*d/2+j\*(-d/2) 3\*d/2+j\*d/2 3\*d/2+j\*(3\*d/2)];

for EbN0db = 0:15

EbN0 = 10^(EbN0db/10);

n = (1/sqrt(2))\*[randn(1,length(yy)) + 1j\*randn(1,length(yy))];

sigma = sqrt(1/((log2(M))\*EbN0)); % Symbol energy is normalized to Unity

r = yy + sigma\*n;

% Calculation of Euclidian Distances of received symbols from reference symobols

min\_dist\_index = [];

for i=1:length(r)

Dist = [];

for k=1:length(ref\_symbols)

dist = sqrt((real(r(i))-real(ref\_symbols(k)))^2 + (imag(r(i))-imag(ref\_symbols(k)))^2);

Dist = [Dist dist];

endfor

min\_dist\_index = [min\_dist\_index find(Dist==min(Dist))];

endfor

% Estimation of Bits

x\_estimated = [];

for i=1:length(r)

if ref\_symbols(min\_dist\_index(i))== -3\*d/2+j\*(-3\*d/2);

x\_estimated = [x\_estimated 0 0 0 0];

elseif ref\_symbols(min\_dist\_index(i))== -3\*d/2+j\*(-d/2);

x\_estimated = [x\_estimated 0 0 0 1];

elseif ref\_symbols(min\_dist\_index(i))== -3\*d/2+j\*(d/2);

x\_estimated = [x\_estimated 0 0 1 1];

elseif ref\_symbols(min\_dist\_index(i))== -3\*d/2+j\*(3\*d/2);

x\_estimated = [x\_estimated 0 0 1 0];

elseif ref\_symbols(min\_dist\_index(i))== -d/2+j\*(-3\*d/2);

x\_estimated = [x\_estimated 0 1 0 0];

elseif ref\_symbols(min\_dist\_index(i))== -d/2+j\*(-d/2);

x\_estimated = [x\_estimated 0 1 0 1];

elseif ref\_symbols(min\_dist\_index(i))== -d/2+j\*(d/2);

x\_estimated = [x\_estimated 0 1 1 1];

elseif ref\_symbols(min\_dist\_index(i))== -d/2+j\*(3\*d/2);

x\_estimated = [x\_estimated 0 1 1 0];

elseif ref\_symbols(min\_dist\_index(i))== d/2+j\*(-3\*d/2);

x\_estimated = [x\_estimated 1 1 0 0];

elseif ref\_symbols(min\_dist\_index(i))== d/2+j\*(-d/2);

x\_estimated = [x\_estimated 1 1 0 1];

elseif ref\_symbols(min\_dist\_index(i))== d/2+j\*(d/2);

x\_estimated = [x\_estimated 1 1 1 1];

elseif ref\_symbols(min\_dist\_index(i))== d/2+j\*(3\*d/2);

x\_estimated = [x\_estimated 1 1 1 0];

elseif ref\_symbols(min\_dist\_index(i))== 3\*d/2+j\*(-3\*d/2);

x\_estimated = [x\_estimated 1 0 0 0];

elseif ref\_symbols(min\_dist\_index(i))== 3\*d/2+j\*(-d/2);

x\_estimated = [x\_estimated 1 0 0 1];

elseif ref\_symbols(min\_dist\_index(i))== 3\*d/2+j\*(d/2);

x\_estimated = [x\_estimated 1 0 1 1];

elseif ref\_symbols(min\_dist\_index(i))== 3\*d/2+j\*(3\*d/2);

x\_estimated = [x\_estimated 1 0 1 0];

endif

endfor

% BER Computation

ber\_simulated =[ber\_simulated sum(x~=x\_estimated)/N];

ber\_theoretical = [ber\_theoretical (3/(2\*log2(M)))\*erfc(sqrt(2\*EbN0/5))];

endfor

EbN0db = 0:15;

% BER Plotting

semilogy(EbN0db, ber\_simulated, 'ro-', EbN0db, ber\_theoretical, 'k>-');

title('BER vs Eb/N0 Plot for 16-Square QAM');

xlabel('Eb/N0 (dB)');

ylabel('BER');

grid on;

legend('Simulated', 'Theoretical');

axis([0 15 10^-3 10^0]);