clc;

clear all;

close all;

printf("Name: Jeevesh Wagh\n");

printf("Roll No: 32375\n");

printf("Batch: L8\n");

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

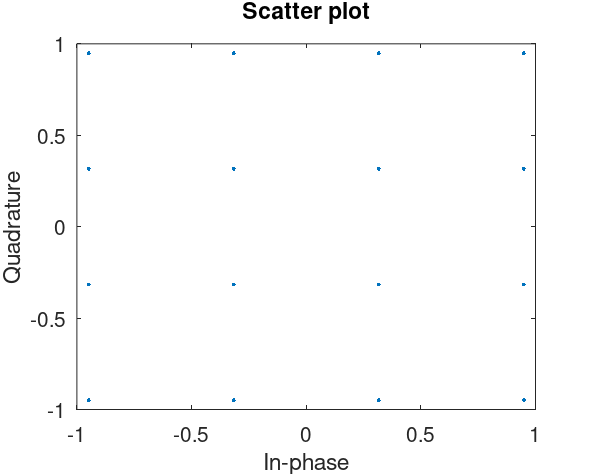
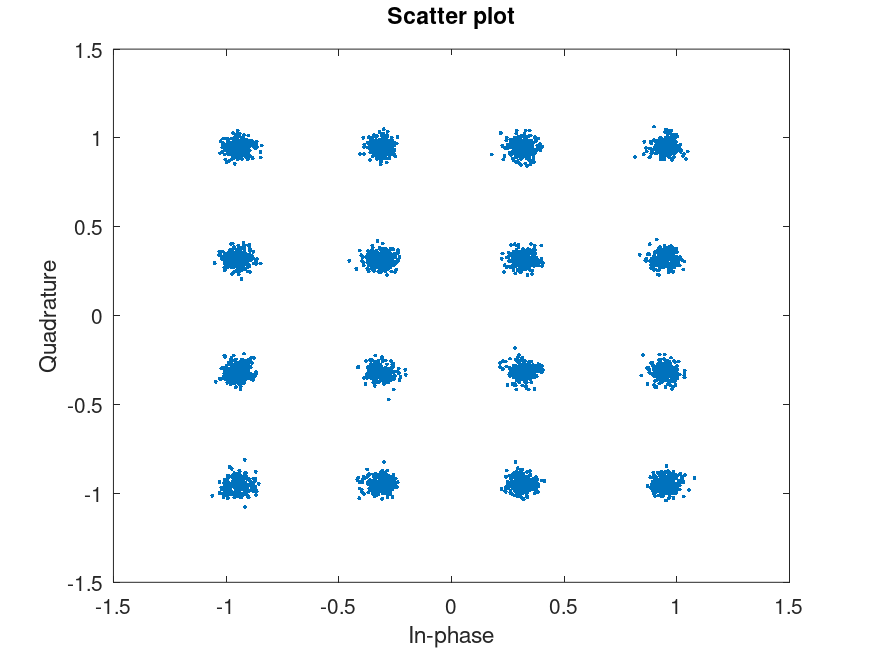
n = (1/sqrt(2))\*[randn(1,length(yy)) + 1j\*randn(1,length(yy))];

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

% Received Symbols

r = yy + sigma\*n;

scatterplot(r); % Constellation Diagram with Noise



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]);

