Faculty of Engineering Alexandria University
Communication and Electronics Department

Digital Communication
PROJECT

Team Members

Engy Ashraf 66

Michel Emad Waqim 261

Mina Nabil Wahib 264

Youstina Abd El Massih 286

Experiment 1

objective:

Compare the performance of the different modulation schemes(ASK – FSK – PSK).

Code implementation:

```
close all
clc
clear all
M = 16;                  % Size of signal constellation
k = log2(M);              % Number of bits per symbol
NumSNR= 10 ^(6);
SNR=0:2:30; %range of SNR
RandomSignal=randi([0,1],NumSNR,1); % generate stream of random bits
numSamplesPerSymbol = 1;
```

first we initialize bits , snr and random signal to be modulated later

```
%% modulations

OOK_mod = RandomSignal;% OOK modulation
OOK_mod_fn = (pammod(RandomSignal,2)+1)./2; % OOK modulation using fn

PRK_mod = 2.*RandomSignal - 1;%PRK modulation
PRK_mod_fn = pammod(RandomSignal,2);%PRK modulation using fn

FSk_H_i = find(RandomSignal==1);
FSk_L_i = find(RandomSignal==0);
FSK_modv(FSk_H_i)=1j;%FSK modulation
FSK_modv(FSk_L_i)=1;%FSK modulation
FSK_mod=FSK_modv.';%transpose
FSK_mod_fn = sqrt(pskmod(RandomSignal,2));

% qam modulation
b_data_block = reshape(RandomSignal,length(RandomSignal)/k,k);   % Reshape data into
binary k-tuples, k = log2(M)
block_data_symbol = bi2de(b_data_block);                % Convert to integers

QAM_mod_fn = qammod(block_data_symbol,M,'bin');         % Binary coding, phase offset = 0
```

we modulate our random signal using 4 modulations:

1- OOK → - leaving zeros as zeros and one as one no change
- using pammod by modifying it by adding 1 and dividing the whole result by 2

2- PSK → - by multiplying the random signal by 2 then subtracting 1
- using pammod as it satisfy the required formula

3- FSK → - expressing one as j and zero as 1
- using psk as it satisfy the required formula after taking square root to it

4- 16QAM → - using qammod function

```
%% calculating power
PTX_OOK= mean(OOK_mod.^(2)); %ook power
PTX_OOK_fn= mean(OOK_mod_fn.^(2)); %ook power

PTX_PRK= mean(PRK_mod.^(2)); %prk power
PTX_PRK_fn= mean(PRK_mod_fn.^(2)); %prk power

PTX_FSK= mean(abs(FSK_mod.^(2))); %fsk power
PTX_FSK_fn= mean(abs(FSK_mod_fn.^(2))); %fsk power
```

calculating the power for each modulation

```
%% adding noise , demodulation and calculating bit error
for n=1:length(SNR)

    snr_i=10^(SNR(n)/10);% converting from db to linear
    %calculating noise
    noise_OOK=sqrt(PTX_OOK/(2 * snr_i) *( randn(size(OOK_mod)) + 1j * randn(size(OOK_mod)) ) ) ; %OOK noise
    noise_OOK_fn=sqrt(PTX_OOK_fn/(2 * snr_i) *( randn(size(OOK_mod_fn)) + 1j * randn(size(OOK_mod_fn)) ) ) ; %OOK
noise


    noise_PRK=sqrt(PTX_PRK/(2 * snr_i) *( randn(size(PRK_mod)) + 1j * randn(size(PRK_mod)) ) ) ; %PRK noise
    noise_PRK_fn=sqrt(PTX_PRK_fn/(2 * snr_i) *( randn(size(PRK_mod_fn)) + 1j * randn(size(PRK_mod_fn)) ) ) ; %PRK
noise


    noise_FSK=sqrt(PTX_FSK/(2 * snr_i) *( randn(size(FSK_mod)) + 1j * randn(size(FSK_mod)) ) ) ; %FSK noise
    noise_FSK_fn=sqrt(PTX_FSK_fn/(2 * snr_i) *( randn(size(FSK_mod_fn)) + 1j * randn(size(FSK_mod_fn)) ) ) ; %FSK
noise

    %recieved signals
    RX_OOK = OOK_mod + noise_OOK ; % OOK recieved signal
    RX_OOK_fn = OOK_mod_fn + noise_OOK_fn ; % OOK recieved signal


    RX_PRK = PRK_mod +noise_PRK ;  % PRK recieved signal
    RX_PRK_fn = PRK_mod_fn + noise_PRK_fn;

    RX_FSK = FSK_mod + noise_FSK ; % FSK recieved signal
    RX_FSK_fn = FSK_mod_fn + noise_FSK_fn;

    snr_i_qam = SNR(n) + 10*log10(k) - 10*log10(numSamplesPerSymbol);
    RX_QAM_fn = awgn(QAM_mod_fn,snr_i_qam,'measured');% qam  recieved signal

    %demodulation
    %OOK_demodulation
    RX_OOK_abs=abs(RX_OOK);
    RX_OOK_H_i = find(RX_OOK_abs>0.5);
    RX_OOK(RX_OOK_H_i) = 1;
    RX_OOK_L_i = find(RX_OOK_abs<0.5);
    RX_OOK(RX_OOK_L_i) = 0;

    RX_OOK_fn=pamdemod((2.*RX_OOK_fn)-1,2);

    %PRK demodulation
    RX_PRK_H_i = find(real(RX_PRK)>0);
    RX_PRK(RX_PRK_H_i) = 1;
    RX_PRK_L_i = find(real(RX_PRK)<0);
    RX_PRK(RX_PRK_L_i) = 0;

    RX_PRK_fn=pamdemod(RX_PRK_fn,2);

    %FSK demodulation
    RX_FSK_R_abs=abs(real(RX_FSK));
    RX_FSK_I_abs=abs(imag(RX_FSK));
    RX_FSK_L_i = find (RX_FSK_R_abs>=RX_FSK_I_abs);
    RX_FSK(RX_FSK_L_i)=0;
    RX_FSK_H_i = find (RX_FSK_R_abs<RX_FSK_I_abs);
    RX_FSK(RX_FSK_H_i)=1;
    RX_FSK_fn = pskdemod(RX_FSK_fn.^2,2);

     if SNR(n)==0
        RX_QAM_fn_0=RX_QAM_fn;
```

```
    elseif SNR(n)==16
        RX_QAM_fn_15=RX_QAM_fn;
    elseif SNR(n)==30
        RX_QAM_fn_30=RX_QAM_fn;
    end
    %QAM demodulation
    symbol_data_block = qamdemod(RX_QAM_fn,M,'bin');
    block_data_binary = de2bi(symbol_data_block,k);
    RX_QAM_fn = block_data_binary(:);% Return data in column vector

    %error detection
   [number_OOK,ratio_OOK]=biterr(OOK_mod,RX_OOK); %OOK_error detection
   [number_OOK_fn,ratio_OOK_fn]=biterr(OOK_mod_fn,RX_OOK_fn); %OOK_error detection

   [number_PRK,ratio_PRK]=biterr(OOK_mod,RX_PRK); %PRK_error detection
   [number_PRK_fn,ratio_PRK_fn]=biterr(OOK_mod,RX_PRK_fn); %PRK_error detection

   [number_FSK,ratio_FSK]=biterr(OOK_mod,RX_FSK); %FSK_error detection
   [number_FSK_fn,ratio_FSK_fn]=biterr(OOK_mod,RX_FSK_fn); %FSK_error detection

   [number_QAM_fn,ratio_QAM_fn]=biterr(OOK_mod,RX_QAM_fn); %QAM_error detection


    Error_OOK(n)=ratio_OOK;
    Error_OOK_fn(n)=ratio_OOK_fn;

    Error_PRK(n)=ratio_PRK;
    Error_PRK_fn(n)=ratio_PRK_fn;

    Error_FSK(n)=ratio_FSK;
    Error_FSK_fn(n)=ratio_FSK_fn;

    Error_QAM_fn(n)=ratio_QAM_fn;


end
```

here we start by adding noise to each modulation according to its power , so we get the received signal then we modulate each one according to the appropriate detection condition

as for OOK the detection condition is the v th so if the value more than 0.5 it is one else it is zero

for PRK we check real part if greater than vth (zero) it is one else it is zero

for FSK we compare the real part with the imaginary one and see which is bigger if the real part is bigger then it is zero else it is one

for QAM we use the built in function

then we calculate the bit error rate using biterr

```
%% plotting
figure
semilogy(SNR,Error_OOK_fn,'r','linewidth',3,'DisplayName','OOK') %plotting BER VS SNR
hold on
semilogy(SNR,Error_FSK_fn,'b','linewidth',3,'DisplayName','FSK') %plotting BER vs SNR
hold on
semilogy(SNR,Error_PRK_fn,'g','linewidth',3,'DisplayName','PRK') %plotting BER vs SNR
hold on
semilogy(SNR,Error_QAM_fn,'k','linewidth',3,'DisplayName','QAM') %plotting BER vs SNR
ylabel('BER'), xlabel('SNR') ,grid on;
hold off
legend;

figure
semilogy(SNR,Error_OOK,'r','linewidth',3,'DisplayName','OOK') %plotting BER VS SNR
hold on
```

```
semilogy(SNR,Error_FSK,'b','linewidth',3,'DisplayName','FSK') %plotting BER vs SNR
hold on
semilogy(SNR,Error_PRK,'g','linewidth',3,'DisplayName','PRK') %plotting BER vs SNR
hold on
semilogy(SNR,Error_QAM_fn,'k','linewidth',3,'DisplayName','QAM') %plotting BER vs SNR
ylabel('BER'), xlabel('SNR') ,grid on;
hold off
legend;


figure
subplot(4,2,1);
semilogy(SNR,Error_OOK);
xlabel('Range(dB)'), ylabel('Error'),grid on;
title('Error against SNR OOK','FontSize',12);
subplot(4,2,2);
semilogy(SNR,Error_OOK_fn);
xlabel('Range(dB)'), ylabel('Error'),grid on;
title('Error against SNR OOK-fn','FontSize',12);
subplot(4,2,3);
semilogy(SNR,Error_PRK);
xlabel('Range(dB)'), ylabel('Error'),grid on;
title('Error against SNR PRK','FontSize',12);
subplot(4,2,4);
semilogy(SNR,Error_PRK_fn);
xlabel('Range(dB)'), ylabel('Error'),grid on;
title('Error against SNR PRK-fn','FontSize',12);
subplot(4,2,5)
semilogy(SNR,Error_FSK);
xlabel('Range(dB)'), ylabel('Error'),grid on;
title('Error against SNR FSK','FontSize',12);
subplot(4,2,6)
semilogy(SNR,Error_FSK_fn);
subplot(4,2,7:8)
semilogy(SNR,Error_QAM_fn);
xlabel('Range(dB)'), ylabel('Error'),grid on;
title('Error against SNR FSK-fn','FontSize',12);

% constillation diagram
sPlotFig = scatterplot(RX_QAM_fn_15,1,0,'g.');
hold on
scatterplot(QAM_mod_fn,1,0,'k*',sPlotFig)

sPlotFig2 = scatterplot(RX_QAM_fn_30,1,0,'g.');
hold on
scatterplot(QAM_mod_fn,1,0,'k*',sPlotFig2)

sPlotFig3 = scatterplot(RX_QAM_fn_0,1,0,'g.');
hold on
scatterplot(QAM_mod_fn,1,0,'k*',sPlotFig3)
hold off
```

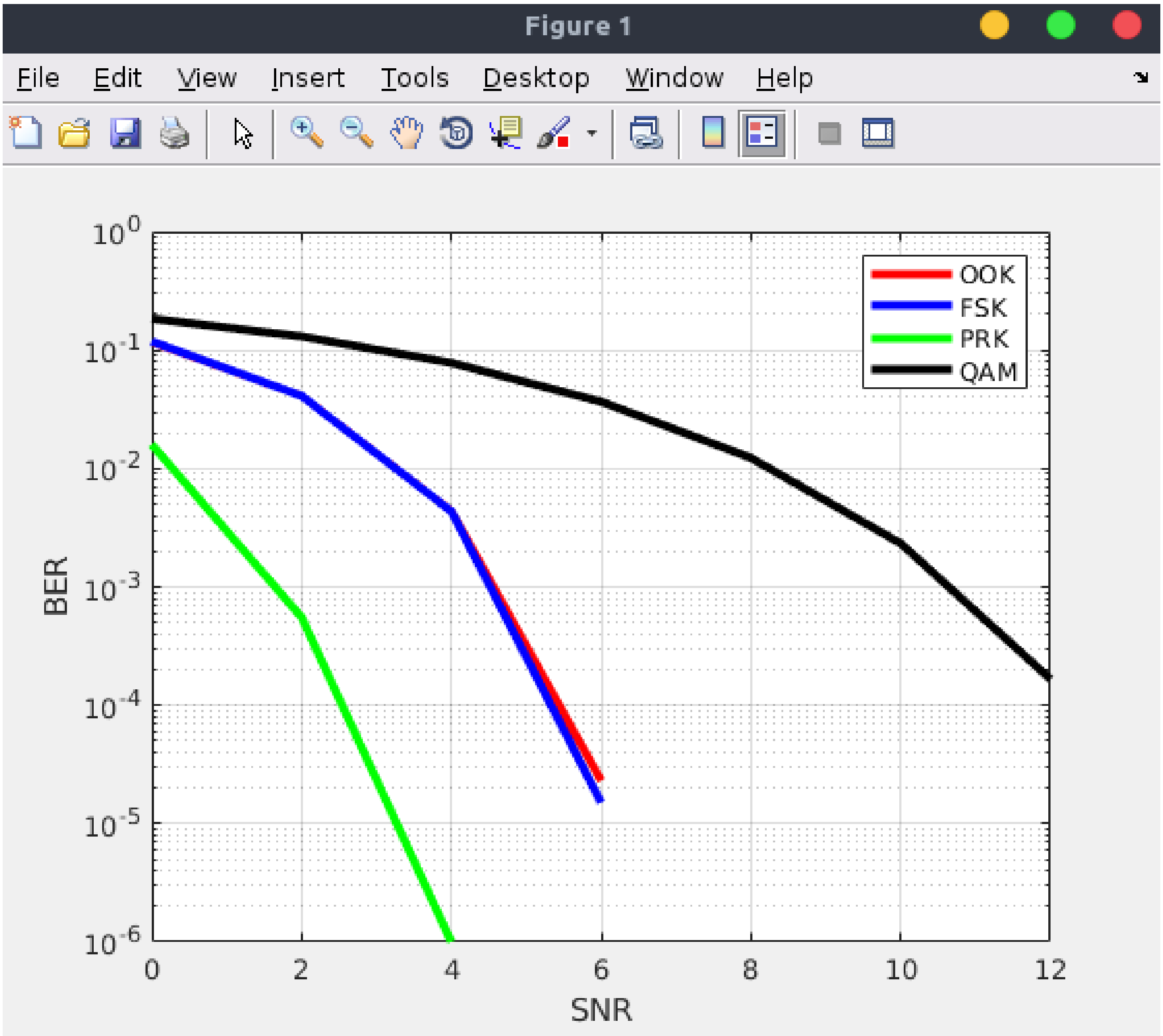
this part we plot the results and constellation diagram for 16QAM at 0 , 15 and 30 db

results and graphs

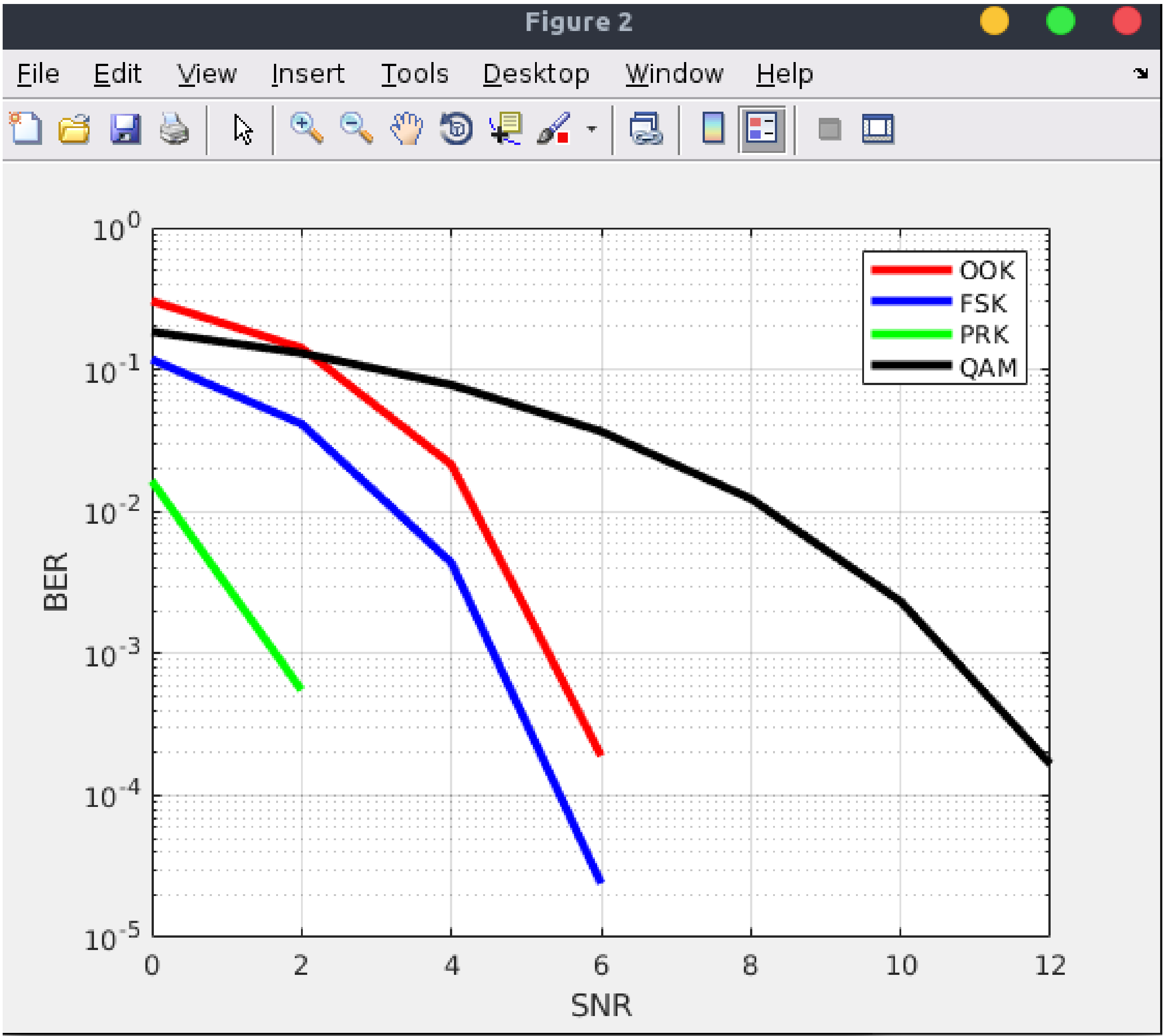
1- power

PTX_FSK	1
PTX_FSK_fn	1
PTX_OOK	0.5005
PTX_OOK_fn	0.5005
PTX_PRK	1
PTX_PRK_fn	1

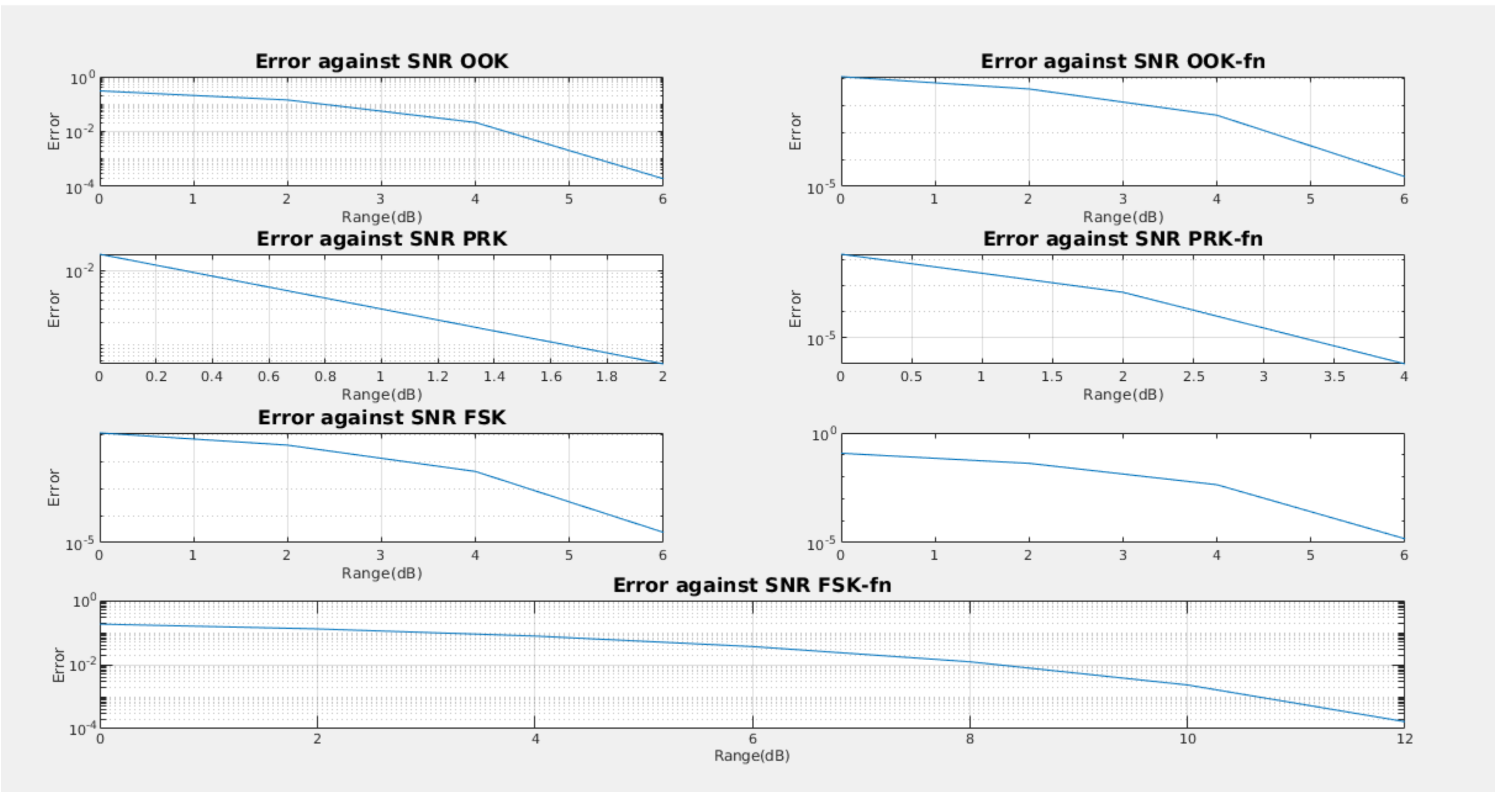
2- error vs snr

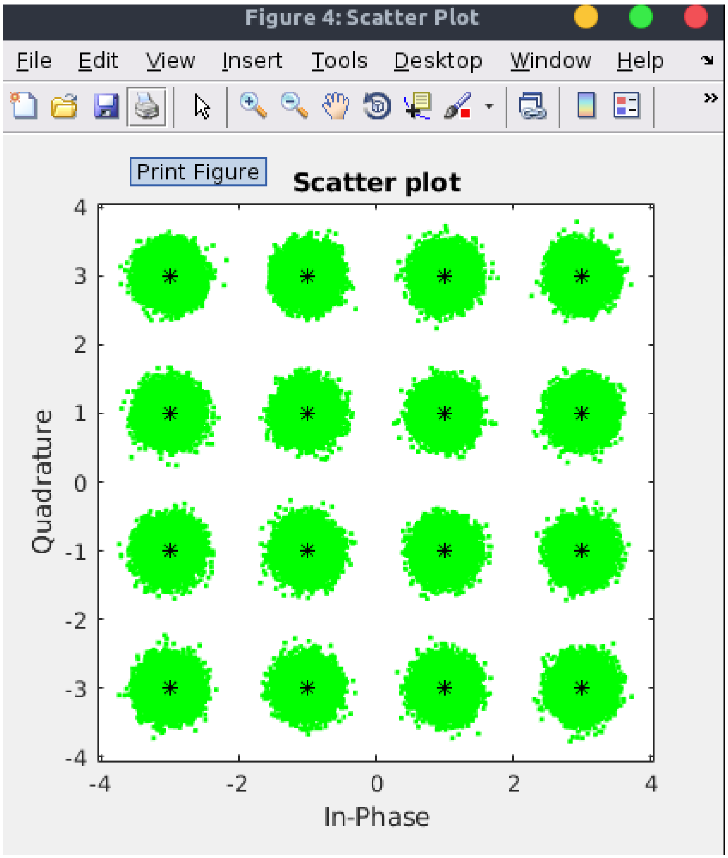


using functions

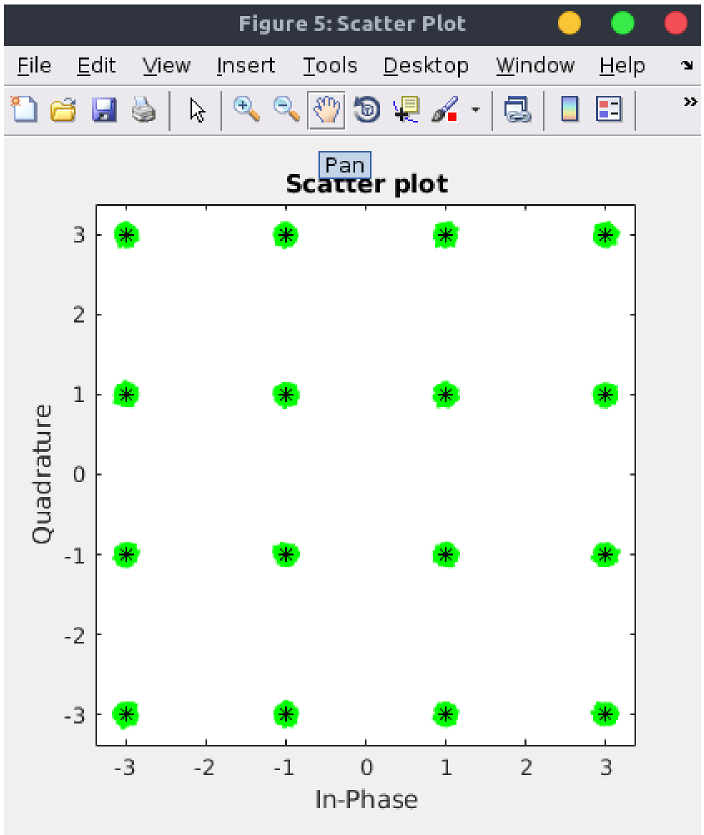


using equations

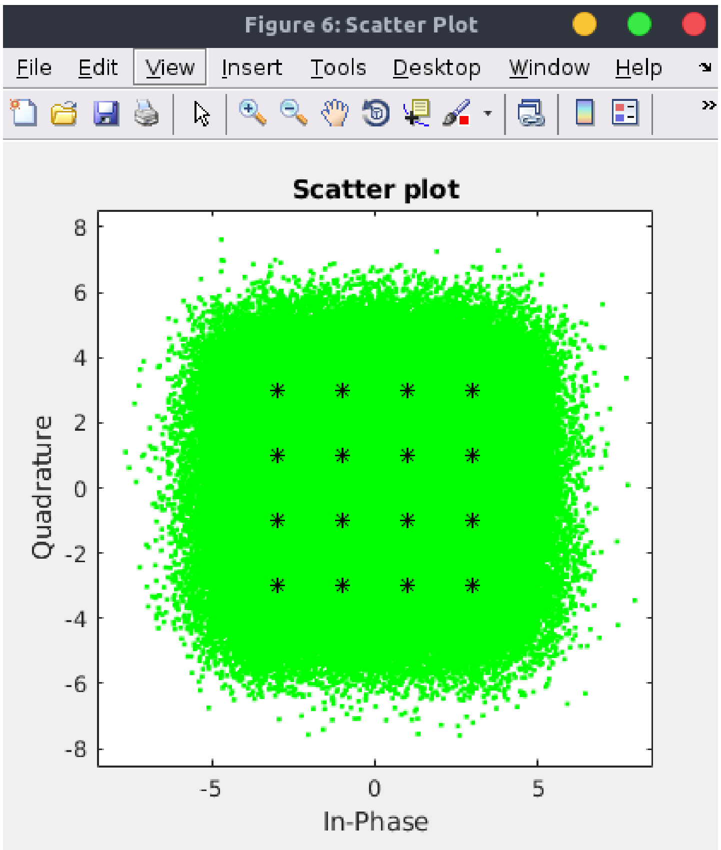




15 db



30db



0db

Experiment 2

Objective

(1) Investigate the importance of channel coding.
(2) Investigate the repetition codes.
(3) Investigate the linear block codes.
(4) Investigate the convolutional codes.
(5) Investigate the main parameters of the convolutional codes.
(6) Calculation of BER in case of coded transmission.

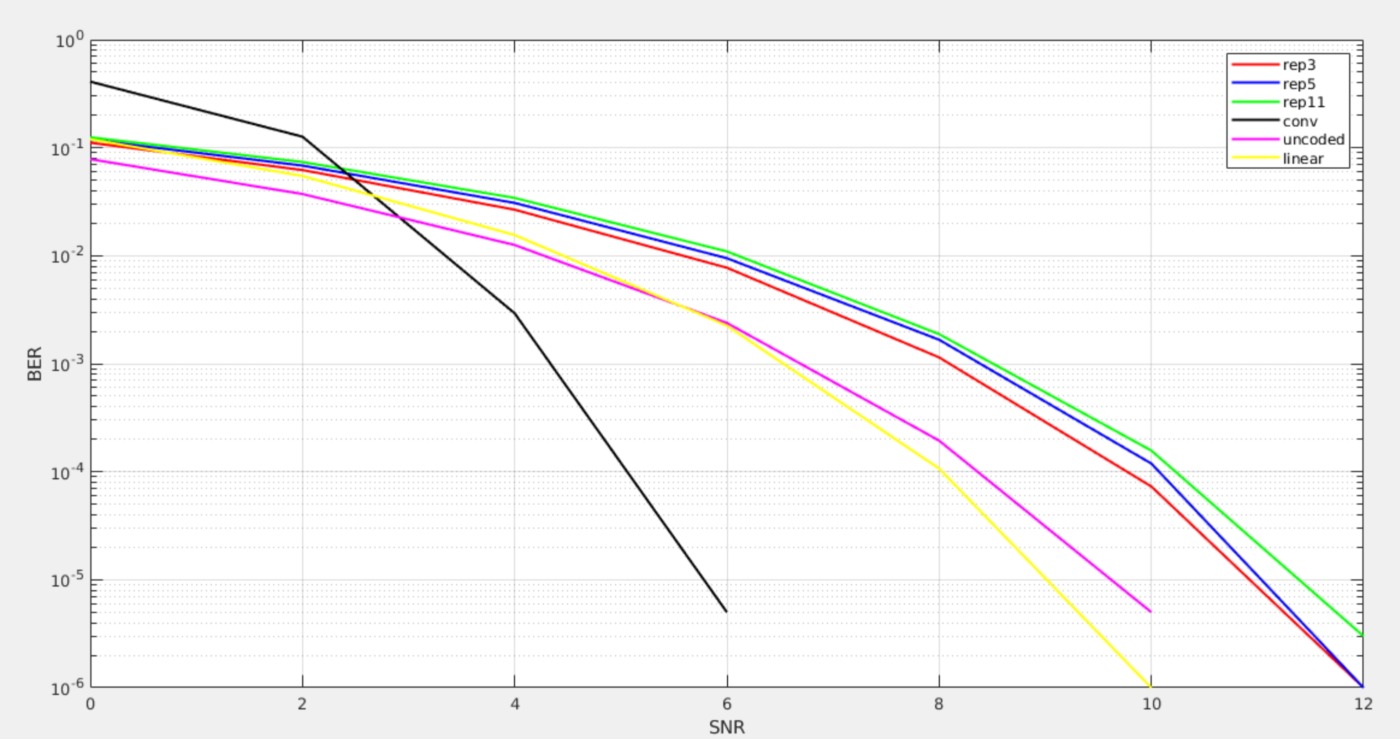
Code implementation

```
clear all
clc
close all
NumSNR= 10 ^(6);
SNR=0:2:30; %range of SNR
RandomSignal=randi([0,1],1,NumSNR); % generate stream of random bits

%% Encoding :
    %first repetiton
    %second linear block code
    %third convolution block code

%% first repetition
% using repmat to repeat bit then reshaping it using reshape to make it in
% one row
encoded_data_3=reshape(repmat(RandomSignal,3,1),[],1);% take data repeat every
bit 3 times
encoded_data_5=reshape(repmat(RandomSignal,5,1),[],1);% take data repeat every
bit 5 times
encoded_data_11=reshape(repmat(RandomSignal,11,1),[],1);% take data repeat every
bit 11 times

%% second Linear block code
%{
k = 4;        % Data length
m = 7;        % Code length
% first create polynomial using cyclpoly then generate parity matrix
% then generate G matrix
% then get the decoding table syndrome

Polynomial = cyclpoly(m,k);
P_M = cyclgen(m,Polynomial);% parity matrix
G_M = gen2par(P_M);%generator matrix
d_t = syndtable(P_M);%decoding table using syndrome to detect error

% Encode the message sequence by using the generator matrix.
data_mat = vec2mat (RandomSignal,k); %convert data from vector form to matrix
form of column length k
[G,U] = size(data_mat);
encoded_data_LBC = [];

for i  = 1 : G
    output = encode(data_mat(i,:),m,k,'linear/binary',G_M);

    encoded_data_LBC = [encoded_data_LBC , output] ;
end;
%}
```

```
encoded_data_LBC = encode(RandomSignal,7,4,'hamming/binary');
%% third Convolutional Codes
constLength = 9;
traceBack = 5 * constLength;
polynomial = [657 435];
trellis = poly2trellis(constLength , polynomial);

encoded_data_Conv = convenc(RandomSignal , trellis);

%% Modulation :

%% first repetition
Rep3_Mod_Data = pskmod(encoded_data_3,2);
PTX_R_3=mean(Rep3_Mod_Data.^2);
Rep3_Mod_Data = Rep3_Mod_Data*sqrt(1/3);

Rep5_Mod_Data = pskmod(encoded_data_5,2);
PTX_R_5=mean(Rep5_Mod_Data.^2);
Rep5_Mod_Data = Rep5_Mod_Data*sqrt(1/5);

Rep11_Mod_Data = pskmod(encoded_data_11,2);
PTX_R_11=mean(Rep11_Mod_Data.^2);
Rep11_Mod_Data = Rep11_Mod_Data*sqrt(1/11);
%% second Linear block code
LBC_Mod_Data= pskmod(encoded_data_LBC,2);
PTX_LBC= mean(LBC_Mod_Data.^(2));
LBC_Mod_Data = LBC_Mod_Data*sqrt(1.75);
%% third Convolutional Codes

Conv_Mod_Data = pskmod(encoded_data_Conv,2);
PTX_conv= mean(Conv_Mod_Data.^(2));
Conv_Mod_Data = Conv_Mod_Data*sqrt(1/2);
%% fourth uncoded
uncoded_Mod_Data=pskmod(RandomSignal,2);
PTX_uncoded = mean(uncoded_Mod_Data.^2);

%% adding noise + demodulation + decoding + error detection


%% adding noise


for n=1:length(SNR)

     snr_i=10^(SNR(n)/10);
     % noise repetition
     noise_rep3=sqrt(1/(2*snr_i))*( randn(size(Rep3_Mod_Data))
+1j*randn(size(Rep3_Mod_Data)));
     noise_rep5=sqrt(1/(2*snr_i))*( randn(size(Rep5_Mod_Data))
+1j*randn(size(Rep5_Mod_Data)));
     noise_rep11=sqrt(1/(2*snr_i))*( randn(size(Rep11_Mod_Data))
+1j*randn(size(Rep11_Mod_Data)));
     %noise lbc
     noise_LBC=sqrt(1/(2*snr_i))*( randn(size(LBC_Mod_Data))
+1j*randn(size(LBC_Mod_Data)));
     %noise conv
     noise_conv=sqrt(1/(2*snr_i))*( randn(size(Conv_Mod_Data))
+1j*randn(size(Conv_Mod_Data)));
     %uncoded
     %noise_BPSK=sqrt(1/(2 * snr_i) *( randn(size(RandomSignal)) + 1j *
randn(size(RandomSignal)) ) ) ; %PRK noise
```

```
    %recieved repetition
    Rx_Rep3 = Rep3_Mod_Data + noise_rep3;
    Rx_Rep5 = Rep5_Mod_Data + noise_rep5;
    Rx_Rep11 = Rep11_Mod_Data + noise_rep11;
    %recieved repetition
    Rx_LBC = awgn(LBC_Mod_Data ,SNR (n), 'measured');
    %recieved repetition
    Rx_conv = Conv_Mod_Data + noise_conv;
    %recieved uncoded
    Rx_uncoded= awgn(uncoded_Mod_Data ,SNR (n), 'measured');


    %% demodulation


        %% first repetition

    Rep3_Demod_Data = pskdemod(Rx_Rep3,2);
    Rep5_Demod_Data = pskdemod(Rx_Rep5,2);
    Rep11_Demod_Data = pskdemod(Rx_Rep11,2);
    %% second Linear block code
    LBC_Demod_Data = pskdemod(Rx_LBC,2);

        %% third Convolutional Codes

    Conv_Demod_Data = pskdemod(Rx_conv,2);

        %% fourth Convolutional Codes

    uncoded_Demod = pskdemod(Rx_uncoded,2);



    %% Decoding

         %% first repetition

    for i=1:NumSNR
        Rep3_Dec_Data(i)= sum(Rep3_Demod_Data(3*i-2:3*i))>=2;
        Rep5_Dec_Data(i)= sum(Rep5_Demod_Data(5*i-4:5*i))>=3;
        Rep11_Dec_Data(i)= sum(Rep11_Demod_Data(11*i-10:11*i))>=6;


    end
         %% second Linear block code
%{
     mat2 = vec2mat (LBC_Demod_Data,m);
    [y,r] = size(mat2);
    LBC_dec_Data =[];

    for j  = 1 : y
       p = decode(mat2(j,:),m,k,'linear/binary',G_M,d_t);
       LBC_dec_Data = [LBC_dec_Data , p] ;
    end
%}
 LBC_dec_Data= decode(LBC_Demod_Data,7,4,'hamming/binary');
         %% third Convolutional Codes

    Conv_Dec_Data = vitdec(Conv_Demod_Data,trellis,traceBack,'trunc','hard');
```

```
%% error and ber

          %% first repetition

     [numoferrors_rep3(n),ratio_rep_3(n)]=biterr(RandomSignal,Rep3_Dec_Data);
     [numoferrors_rep5(n),ratio_rep_5(n)]=biterr(RandomSignal,Rep5_Dec_Data);
     [numoferrors_rep11(n),ratio_rep_11(n)]=biterr(RandomSignal,Rep11_Dec_Data);

          %% second Linear block code

     [numoferrors_LBC(n),ratio_LBC(n)]=biterr(RandomSignal,LBC_dec_Data);



          %% third Convolutional Codes

     [numoferrors_conv(n),ratio_conv(n)]= biterr(RandomSignal,Conv_Dec_Data);
          %% fourth Convolutional Codes
     [numoferrors_uncoded(n),ratio_uncoded(n)]=
biterr(RandomSignal,uncoded_Demod);



end

figure
semilogy(SNR,ratio_rep_3,'r','linewidth',1.5,'DisplayName','rep3') %plotting BER
VS SNR
hold on
semilogy(SNR,ratio_rep_5,'b','linewidth',1.5,'DisplayName','rep5') %plotting BER
vs SNR
hold on
semilogy(SNR,ratio_rep_11,'g','linewidth',1.5,'DisplayName','rep11') %plotting
BER vs SNR
hold on
semilogy(SNR,ratio_conv,'k','linewidth',1.5,'DisplayName','conv') %plotting BER
vs SNR
hold on
semilogy(SNR,ratio_uncoded,'m','linewidth',1.5,'DisplayName','uncoded')
%plotting BER vs SNR
hold on
semilogy(SNR,ratio_LBC,'y','linewidth',1.5,'DisplayName','linear') %plotting BER
vs SNR

ylabel('BER'), xlabel('SNR') ,grid on;
hold off
legend;
```



BER
SNR
rep3
rep5
rep11
conv
uncoded
linear