Cairo University
Faculty of Engineering

Department of Computer Engineering

ELC 3070 – Spring 2024

Communications 3

Project #3

Matched Filter

Submitted to

Dr. Mohamed Khairy

Dr. Mohamed Nafea

Eng. Mohamed Khaled

Submitted by

Team: 90

Name	Sec	BN
فتحى مصطفي فتحى عبد الحميد	3	19
عمر احمد عبد الكريم عبد الظاهر	3	8

Contents

Figures

Phase shift key

Phase Shift Keying (PSK) is a digital modulation technique used in telecommunications to encode and decode digital data in the phase of a carrier signal. In PSK, the phase of the carrier signal is altered to represent different symbols or bits

BPSK

Binary Phase Shift Keying (BPSK) employs two distinct phase shifts, 0° and 180°, to symbolize binary 0 and 1 correspondingly.

Initially, the transmitted bits were mapped to the Polar Non-Return-to-Zero (NRZ) scheme, where '1' corresponds to '1' and '0' corresponds to '-1' at the transmitter. At the receiver, since the transmitted data carries the same energy, the decision boundary is set at the origin. Thus, if the received data is greater than zero, it is demapped as '1', otherwise as '0'. Subsequently, the received data after demapping is compared with the sent data before mapping. The number of differences between them is computed and divided by the total number of sent bits to determine the actual Bit Error Rate (BER) due to Additive White Gaussian Noise (AWGN) in the channel.

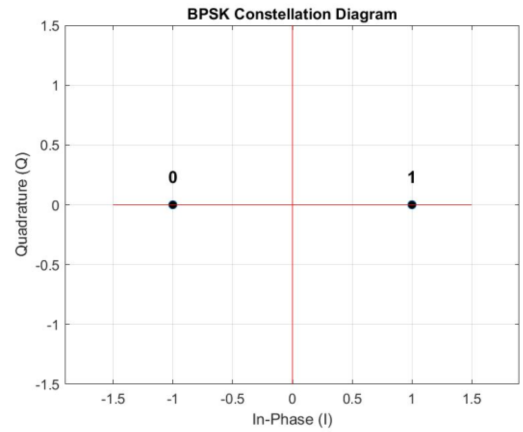

Figure 1 - BPSK Constellation

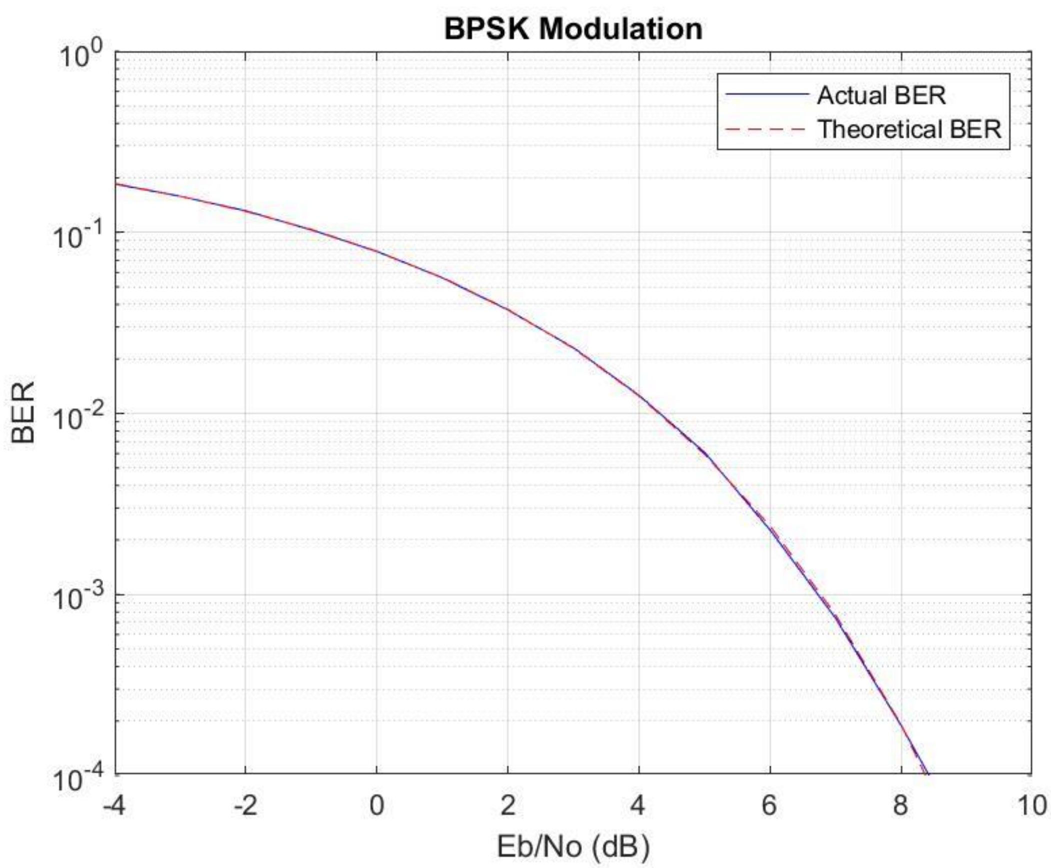

Figure 2 - BPSK BER

Theoretical

$$S_1(t) = \sqrt{\frac{2E_b}{T_b}}\cos(2\pi f_c t\,)\,,\;\; S_2(t) = -\sqrt{\frac{2E_b}{T_b}}\cos(2\pi f_c t\,)$$

$$BER = \frac{1}{2}\; erfc(\sqrt{\frac{E_b}{N_o}})$$

Figure 2 illustrates the correlation between the theoretical and observed Bit Error Rate (BER) of Binary Phase Shift Keying (BPSK) modulation. This data was obtained from transmitting and receiving 540,000 bits in the presence of Additive White Gaussian Noise (AWGN). Additionally, the figure showcases the reduction in BER as the Signal-to-Noise Ratio (SNR) increases.

QPSK – Gray:

Quadrature Phase Shift Keying (QPSK) utilizes four distinct phase shifts, 45°, 135°, 225°, and 315°, to represent two bits per symbol. Each phase shift corresponds to a unique combination of two bits, allowing for the transmission of binary data at twice the rate compared to Binary Phase Shift Keying (BPSK). Gray coding is utilized to ensure that adjacent symbols differ by only one bit, enhancing error robustness during transmission and reception.

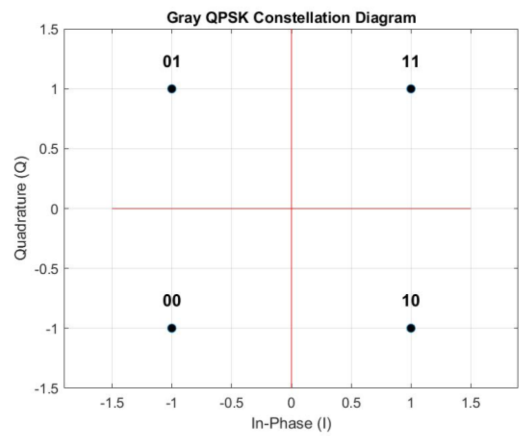

Figure 3 – Gray QPSK Constellation

At the transmitter, the bits were mapped to Polar Non-Return-to-Zero (NRZ) and sent in the form $\cos(\theta) \pm$ j $\sin(\theta)$. At the receiver, the decision regions were defined such that each quarter corresponds to a specific symbol: the first quarter is mapped to '11', the second quarter to '01', the third quarter to '00', and the fourth quarter to '10'. Subsequently, the transmitted data was compared with the received data to compute the Bit Error Rate (BER).

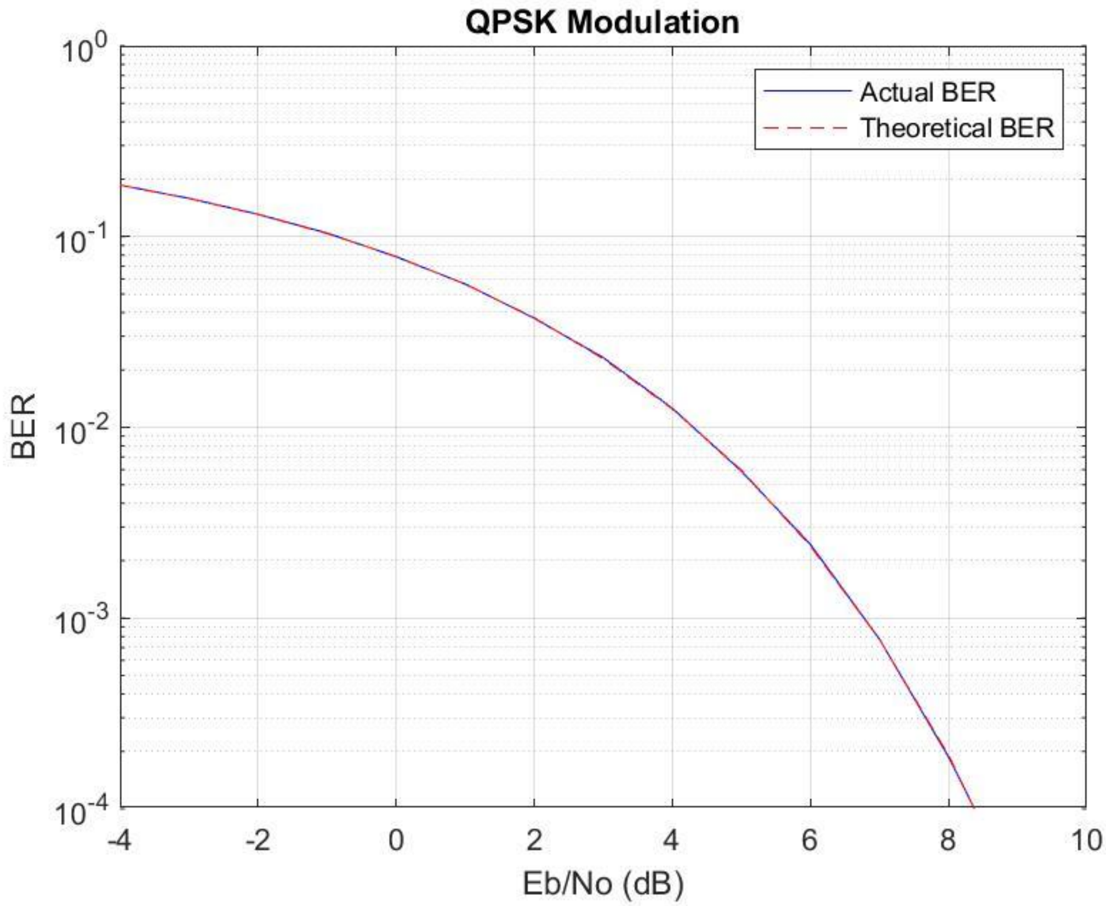

Figure 4 – Gray QPSK BER

Theoretical

$$S_i(t) = \sqrt{\frac{2E}{T}} \cos\left(2\pi f_c t + (2i-1)\frac{2\pi}{4}\right)$$

$$BER = \frac{1}{2}\, erfc\left(\sqrt{\frac{E_b}{N_o}}\right)$$

Figure 4 depicts the relationship between the theoretical and observed Bit Error Rate (BER) of Quadrature Phase Shift Keying (QPSK) modulation of 540,000 bits. According to theoretical analysis, QPSK exhibits identical BER performance to Binary Phase Shift Keying (BPSK) but with half the bandwidth, leading to improved transmission efficiency.

M – PSK:

8-Phase Shift Keying (8PSK) employs eight distinct phase shifts, namely 0°, 45°, 90°, 135°, 180°, 225°, 270°, and 315°, to encode three bits per symbol. Each phase shift corresponds to a unique combination of three bits, enabling the transmission of binary data at a higher rate compared to Quadrature Phase Shift Keying (QPSK).

8PSK data is transmitted and received similarly to QPSK, except that it encodes three bits per symbol instead of two. At the receiver, the decision boundary is depicted in Figure 5.

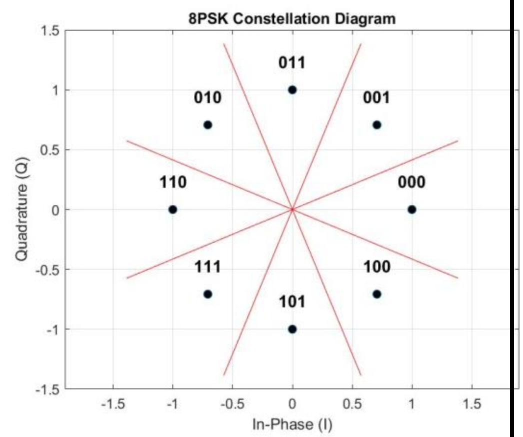

Figure 5 - 8PSK Constellation

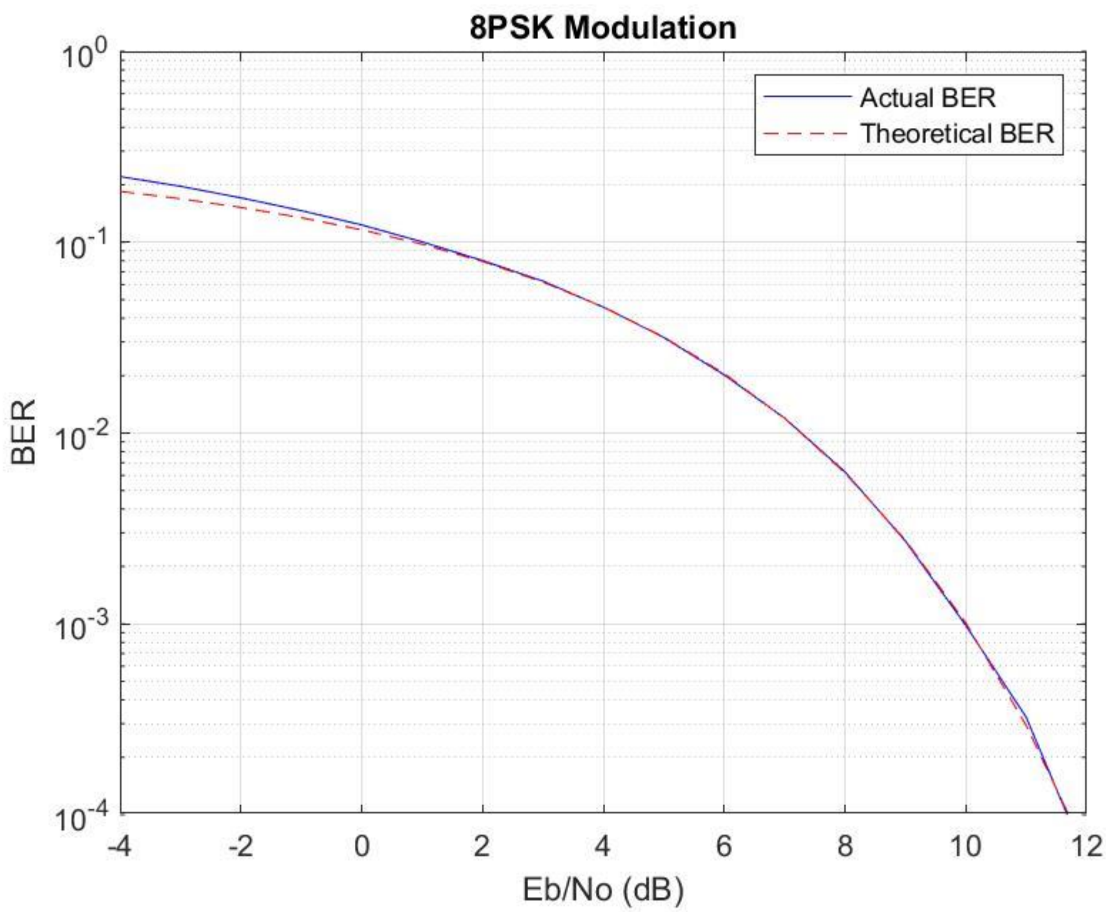

Figure 6 - 8PSK BER

Theoretical

$$\boldsymbol{S_i(t) = \sqrt{\frac{2E}{T}}\ \cos\left(2\pi f_c t\ + (i-1)\frac{2\pi}{m}\right)}$$

$$BER = \frac{1}{log_2(M)} erfc\left(\sqrt{\frac{log_2(M)Eb}{N_O}}\ \sin\left(\frac{\pi}{M}\right)\right)$$

$$BER = \frac{1}{3} erfc\left(\sqrt{\frac{3Eb}{N_O}}\ \sin\left(\frac{\pi}{3}\right)\right)$$

Figure 6 illustrates the similarity between the actual and theoretical Bit Error Rate (BER) of Eight Phase Shift Keying (8PSK) modulation of 540,000 bits despite differences at low Signal-to-Noise Ratio (SNR), where the theoretical BER lacks a tighter bound to the actual due to the removal of a squared term (Pe) during the approximation to the erfc formula. This difference is particularly noticeable at low SNR levels. Additionally, 8PSK exhibits a narrower bandwidth compared to both QPSK and BPSK.

Amplitude Modulation:

Amplitude Shift Keying (ASK) is a digital modulation technique employed in telecommunications to encode and decode digital data by varying the amplitude of a carrier signal. In ASK, different amplitude levels represent distinct symbols or bits.

M – QAM:

16-QAM (Quadrature Amplitude Modulation) utilizes a grid pattern in the complex plane comprising sixteen unique symbol points. Each symbol point represents a specific combination of four bits, facilitating higher data transmission rates compared to simpler modulation techniques such as BPSK or QPSK. These points are arranged equidistantly from one another, optimizing bandwidth utilization and enhancing data throughput. In 16-QAM, Gray coding is utilized to enhance data transmission reliability.

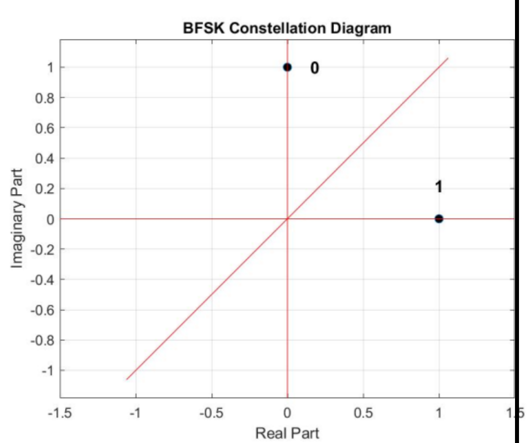

Figure 7 - 16QAM Constellation

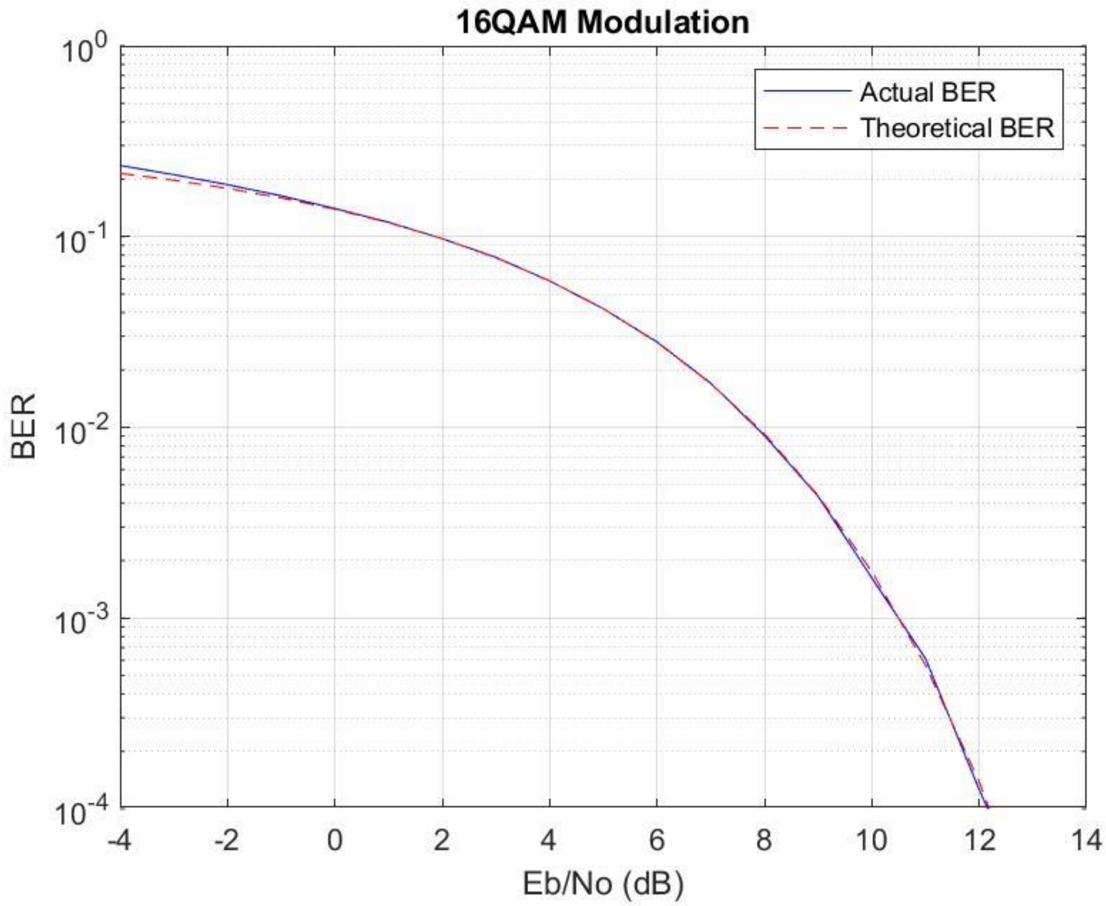

Figure 8 - - 16QAM BER

Theoretical:

$$\boldsymbol{S_i(t)} = \sqrt{\frac{2E_o}{\boldsymbol{T}}}\, a_{\boldsymbol{i}}\, \mathbf{cos}(\boldsymbol{2\pi f_c t}\,) - \sqrt{\frac{2E_o}{\boldsymbol{T}}}\, b_{\boldsymbol{i}} \sin(\boldsymbol{2\pi f_c t}\,), \qquad 0 \le t \le T, \qquad i = 1,2,3, \dots M$$

$$a_i = \pm 1, \pm 3\,, \qquad b_i = \pm 1, \pm 3$$

$$\boldsymbol{BER} = \frac{\mathbf{3}}{\mathbf{8}} \boldsymbol{erfc}\left(\sqrt{\frac{\boldsymbol{E_b}}{\mathbf{2.5}\boldsymbol{N_O}}}\right)$$

Figure 8 shows how similar the simulation and theoretical Bit Error Rate (BER) of 16-QAM of 540,000 bits are, but there are some differences. Especially when the Signal-to-Noise Ratio (SNR) is low, the theoretical BER is lower than the actual one because of a missing squared term in the formula. This difference is more obvious when the SNR is low and there are lots of nearby symbols, like in 16-QAM and 8-PSK.

Modulations:

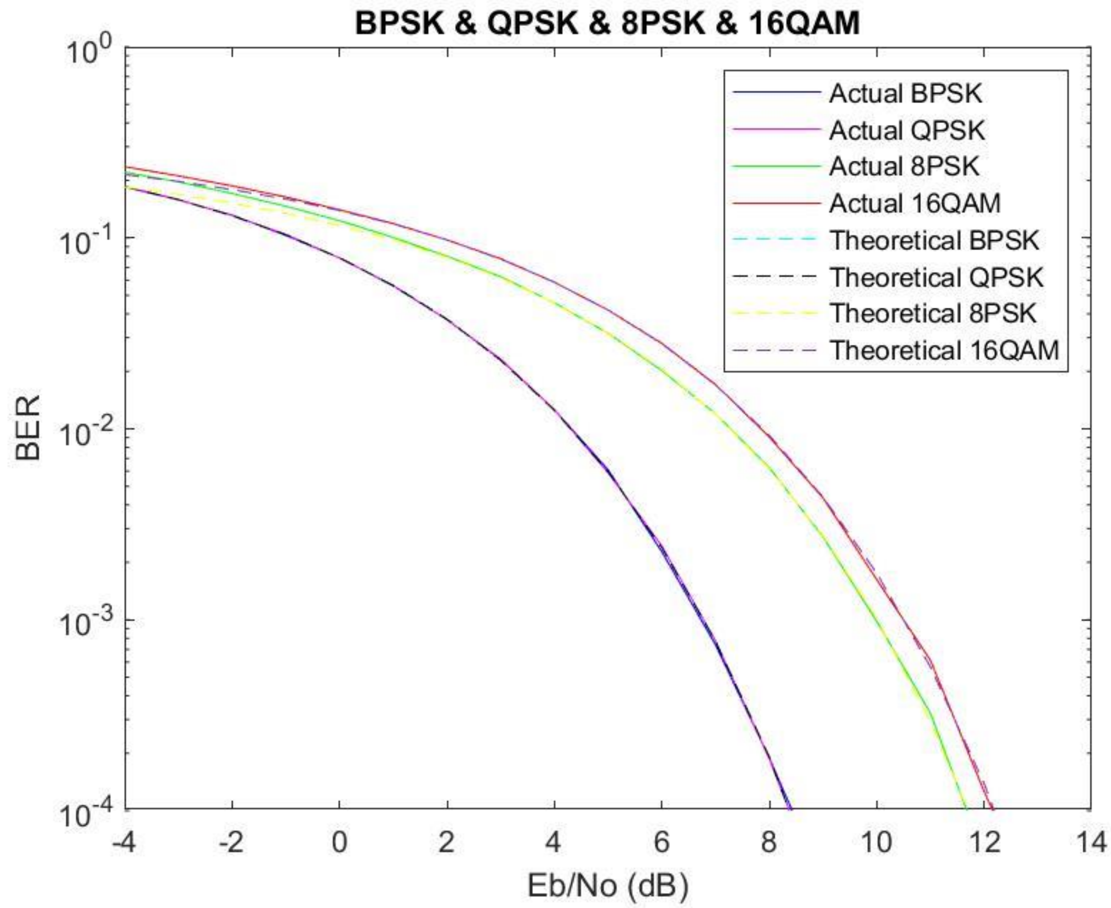

Figure 9 - BPSK Vs QPSK Vs 8PSK Vs 16QAM BER

Figure 9 presents a comparison among four different constellations. Both BPSK and QPSK demonstrate the expected similarity in Bit Error Rate (BER), confirming that QPSK is more efficient due to its narrower bandwidth compared to BPSK. However, 8PSK exhibits a higher bit error rate than QPSK. This is because, beyond QPSK, increasing the number of bits improves bandwidth efficiency at the expense of increasing the BER.

16QAM, although having the largest BER, demonstrates the highest bandwidth efficiency. It is also the most effective scheme for transmitting a large number of bits, as its constellation optimally utilizes space, resulting in lower energy consumption compared to other modulation schemes when transmitting the same large number of bits.

QPSK - Non Gray

The Non Gray encoding of QPSK resembles the Gray QPSK method mentioned previously, with the key difference being that adjacent symbols differ by more than one bit, resulting in an increased bit error rate.

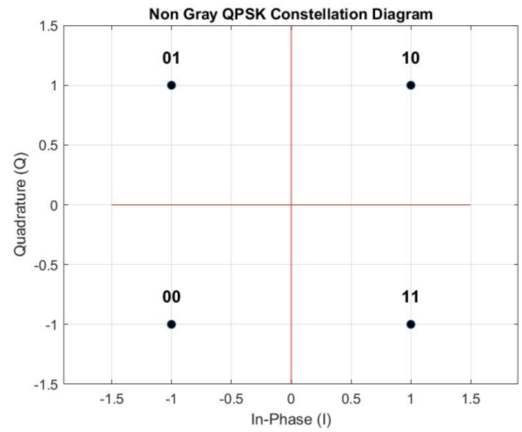

Figure 10 - Non Gray QPSK Constellation

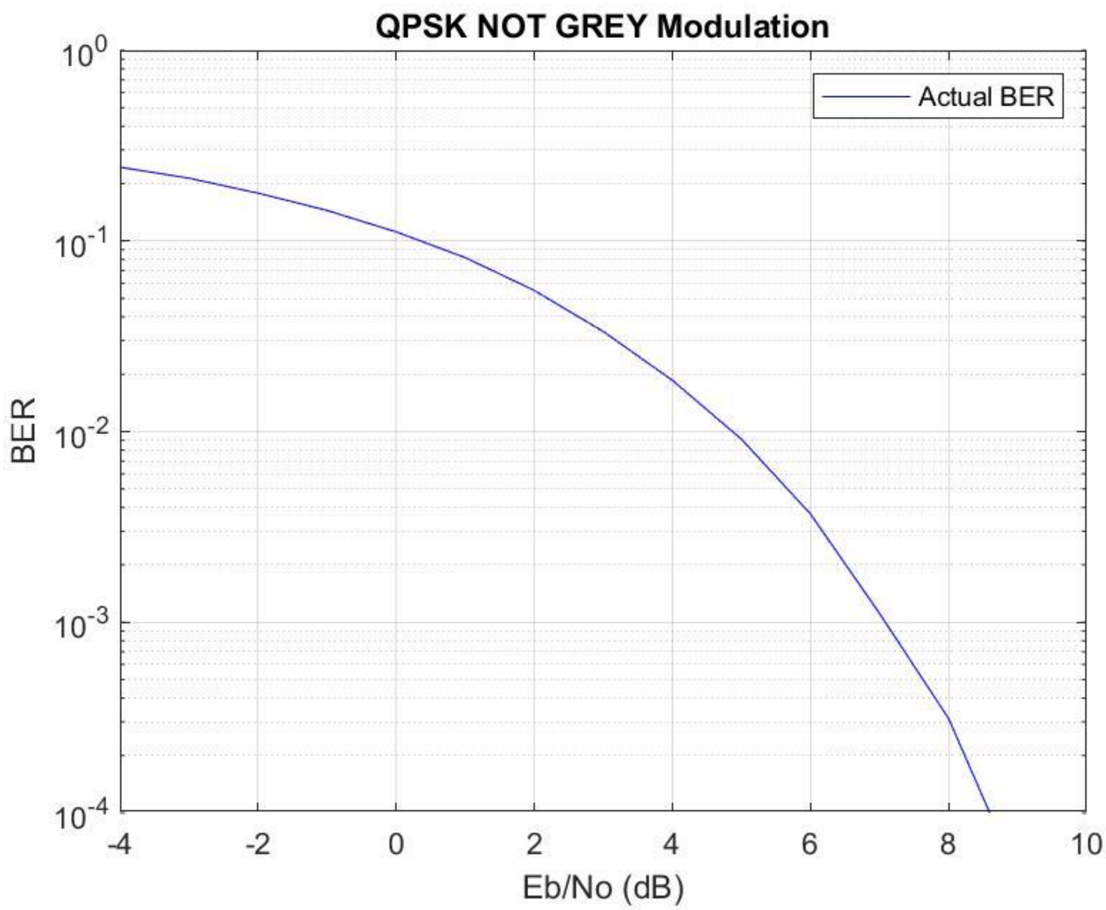

Figure 11 - Non Gray QPSK BER

QPSK – Gray Vs Not Gray:

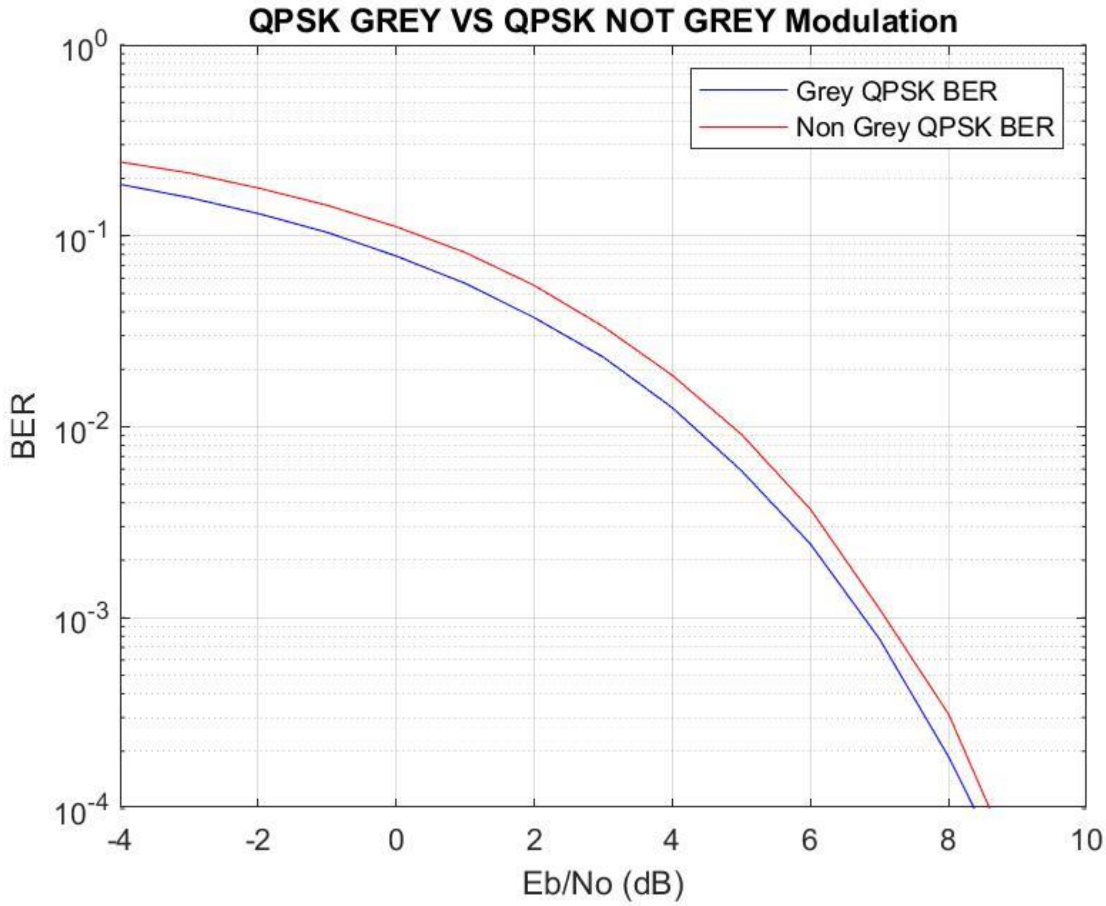

Figure 12 - Gray Vs Non Gray QPSK BER

Gray QPSK ensures that adjacent symbols differ by only one bit, leading to smoother transitions between symbols and lower Bit Error Rate (BER). In contrast, Non-Gray QPSK allows symbols to differ by more than one bit, resulting in larger transitions between adjacent symbols and consequently higher BER, as illustrated in Figure 12.

Frequency shift key

Frequency Shift Keying (FSK) is a digital modulation technique used in telecommunications to encode and decode digital data by varying the frequency of a carrier signal. In FSK, different frequency values represent distinct symbols or bits.

BPSK

Binary Frequency Shift Keying (BFSK) uses two distinct frequencies to represent binary 0 and 1. The carrier signal alternates between these frequencies to encode digital data. Typically, one frequency represents binary 0, while the other represents binary 1.

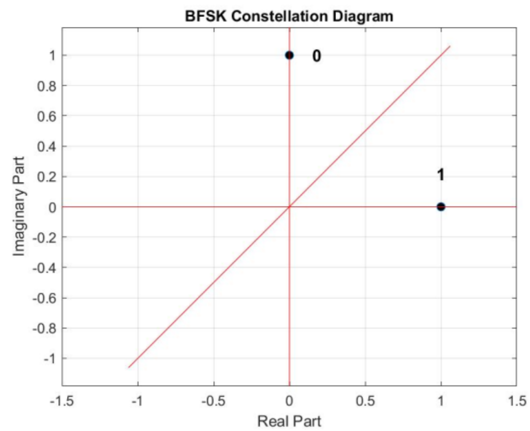

Figure 13 - BFSK Constellation

In Binary Frequency Shift Keying (BFSK), two basic functions are used at the transmitter. A binary 1 is encoded to a carrier frequency $F_c + \frac{1}{T_b}$, corresponding to a signal along the x-axis. On the other hand, a binary 0 is encoded to a carrier frequency $F_c + \frac{2}{T_b}$, corresponding to a signal along the y-axis.

At the receiver, the decision boundary is determined, as depicted in Figure 13, with a line angle of 14 degrees. Subsequently, the transmitted and received data are compared to calculate the Bit Error Rate (BER).

Theoretical

Basis functions:

$$\varphi_i = \sqrt{\frac{2}{T_b}} \cos\left(2\,\pi\,(\,f_c + \frac{i}{T_b}\right)t)\ , \qquad f_c = \frac{n_c}{T_b} \quad , \qquad i = 1{,}2$$

$$\varphi_1 = \sqrt{\frac{2}{T_b}} \cos\left(2\,\pi\,(\,f_c + \frac{1}{T_b}\right)t)\ , \quad \varphi_2 = \sqrt{\frac{2}{T_b}} \cos\left(2\,\pi\,(\,f_c + \frac{2}{T_b}\right)t)$$

Bassband equivalent signals:

$$S_i(t) = \sqrt{\frac{2E_b}{T_b}} \cos\left(2\,\pi\,(\,f_c + \frac{i}{T_b}\right)t)\ , \qquad \boldsymbol{0 \le t \le T}, \qquad i = 1{,}2$$

$$Let\ f_1 = f_c\ , \qquad f_2 = f_c + \frac{1}{T_b}\ , \qquad \Delta f_2 = \frac{2}{T_b}$$

$$S_1(t) = \sqrt{\frac{2E_b}{T_b}} \cos(2\,\pi\ f_1\,t)\,, \qquad S_2(t) = \sqrt{\frac{2E_b}{T_b}} \cos(2\,\pi\ f_2\,t)$$

$$S_1(t) = \sqrt{\frac{2E_b}{T_b}} \cos(2\,\pi\ f_c\,t)$$

$$S_2(t) = \sqrt{\frac{2E_b}{T_b}} \cos(2\pi \ f_c \ t) \cos(2\pi \ \Delta f_2 \ t) - \sqrt{\frac{2E_b}{T_b}} \cos(2\pi \ f_c \ t) \cos(2\pi \ \Delta f_2 \ t)$$

$$S_{1BB}(t) = \sqrt{\frac{2E_b}{T_b}}$$

$$S_{2BB}(t) = \sqrt{\frac{2E_b}{T_b}} \cos(2\pi \ \Delta f_2 \ t) + j\sqrt{\frac{2E_b}{T_b}} \cos(2\pi \ \Delta f_2 \ t)$$

Bit error rate:

$$BER = \frac{1}{2} \, erfc \, (\sqrt{\frac{E_b}{2N_o}})$$

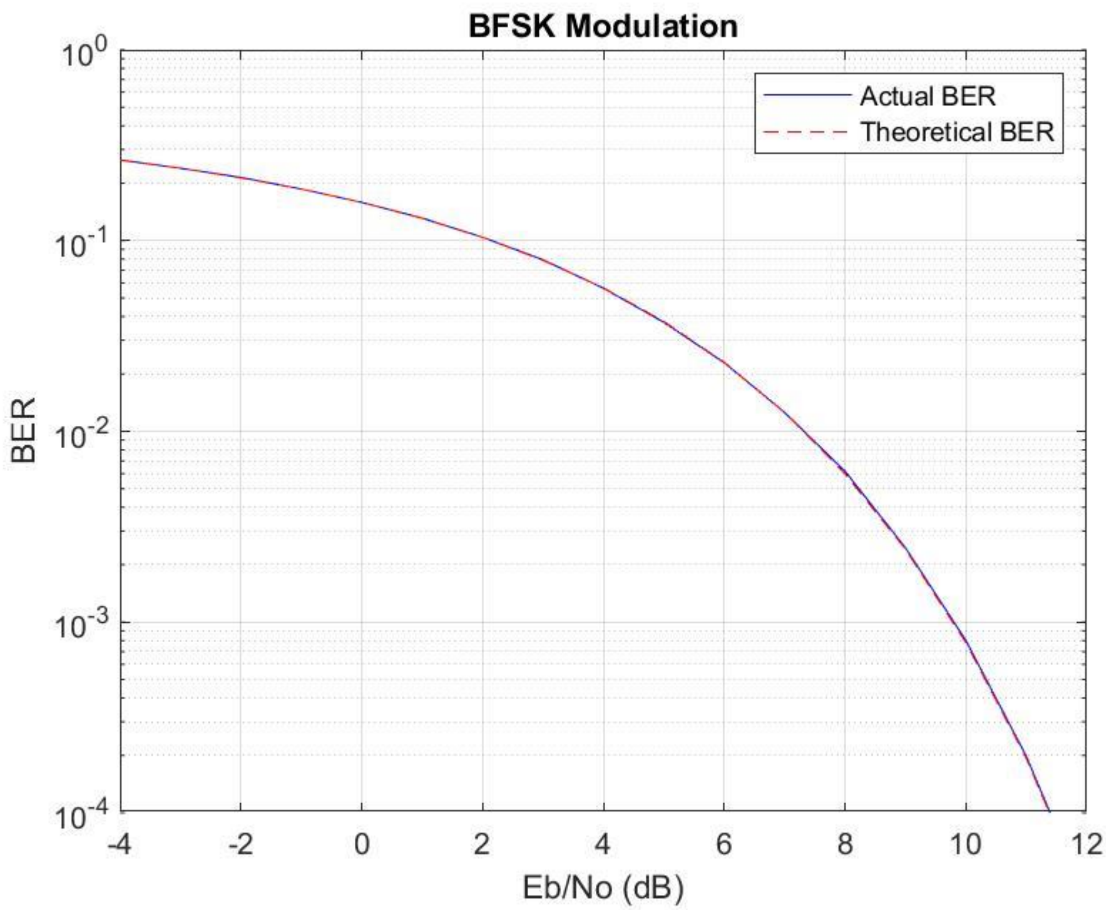

Figure 14 - BFSK BER

Figure 14 illustrates the correlation between the theoretical and observed Bit Error Rate (BER) of Binary Frequency Shift Keying (BFSK) modulation of 540,000 bits.

Theoretical PSD

$$S_B(f) = \frac{1}{2} * \left[\delta\left(f - \frac{1}{2\,T_b}\right) + \delta\left(f + \frac{1}{2\,T_b}\right) \right] + \frac{8E_b \cos^2(\pi T_b f)}{\pi^2\,(4\,T_b^2\,f^2 - 1)^2}$$

The actual PSD of BPSK is determined by generating multiple realizations, mapping them to the baseband equivalent, computing the autocorrelation of the ensemble, and finally, performing a Fourier transform on the autocorrelation to derive the PSD.

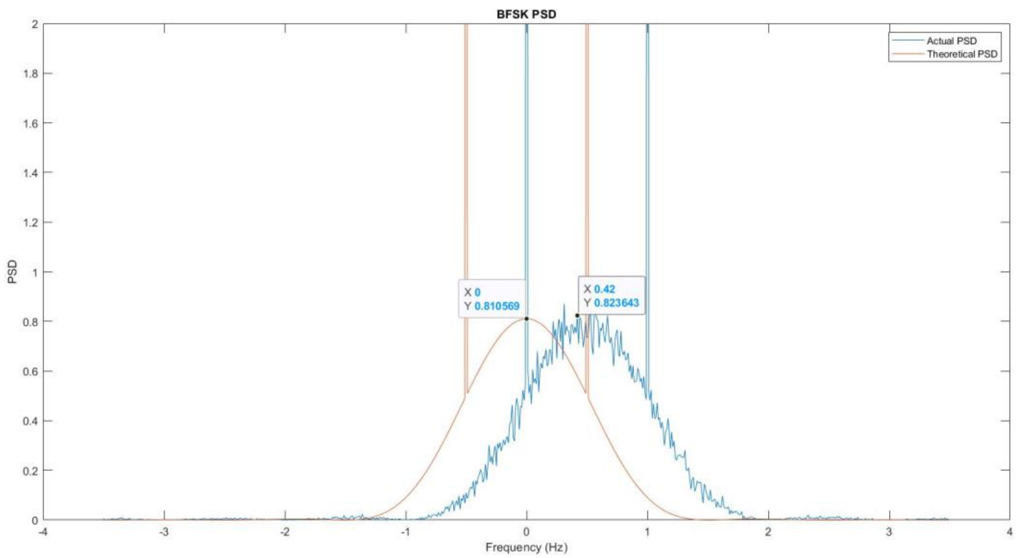

Figure 15 - BFSK PSD

Figure 15 illustrates the comparison between the theoretical and actual Power Spectral Density (PSD) of Binary Frequency Shift Keying (BFSK). A shift of 0.5 is observed between the theoretical and simulated plots. This discrepancy arises because the theoretical equations assume f_1 at $\frac{-1}{2T_b}$. and f_2 at $\frac{1}{2T_b}$., whereas the simulation assumes f_1 at 0 and f_2 at $\frac{1}{T_b}$. However, both plots exhibit similar behavior with identical peak values and two delta functions at f_1 and f_2 .

MATLAB Code

```
%%%%%%%%%%%project Communications 3%%%%%%%%%%%%%%
close all;
clear;
clc;


num_bits=540000;                  %define number of bits
random_bits=randi([0 1],1,num_bits);        %generate
random data

%%%%%%%%BPSK%%%%%%
Eb_BPSK=1;
symbol_BPSK=zeros(1,num_bits);
Demapped_BPSK=zeros(1,num_bits);
%%%%%%%%%maping BPSK%%%%%%%
symbol_BPSK=random_bits.*2-1;      %%maping random data to
Bpsk
%%%%%%%%%Noisy Channel%%%%%%%
SNR_range=[-4:14];

N0=Eb_BPSK./(10.^(SNR_range/10));
for i=1:length(SNR_range)
AWGN_channel=randn(1,num_bits)*sqrt(N0(i)/2);
%%generate gaussian noise
data_BPSK_channel=symbol_BPSK+AWGN_channel;   %%add noise
to symbols

%%demapping%%
for n=1:num_bits
if data_BPSK_channel(n)>=0
Demapped_BPSK(n)=1;
else
Demapped_BPSK(n)=0;
end
end
num_error_BPSK=0;
for n=1:num_bits
if Demapped_BPSK(n)~=random_bits(n)
num_error_BPSK=num_error_BPSK+1;
end
end
```

```
BER_BPSK(i)=num_error_BPSK/num_bits;
end
BER_BPSK_theoritcal=0.5*erfc(sqrt(10.^(SNR_range/10)));
figure
semilogy(SNR_range,BER_BPSK,'b') ;
hold on;
semilogy( SNR_range , BER_BPSK_theoritcal ,'r--') ;
xlabel('Eb/No (dB)');
ylabel('BER');
ylim([10^-4 , 1]);
legend('Actual BER' , 'Theoretical BER ') ;
grid on
title('BPSK Modulation');
saveas(gcf, fullfile('G:\Comm3', 'BPSK.jpg'),'jpg');

%%%%%%%%%%%%%%%%%%QPSK%%%%%%%%%%%%%%%%%%
symbol_QPSK=zeros(1,(num_bits/2));          %two bits are
represented in one symbol
Demapped_QPSK=zeros(1,num_bits);
%%%%%maping%%%%%%%
for i=1:2:num_bits
if [random_bits(i) random_bits(i+1)]==[1 1]
symbol_QPSK(floor(i/2+1))=1+1j;
elseif [random_bits(i) random_bits(i+1)]==[0 1]
symbol_QPSK(floor(i/2+1))=-1+1j;
elseif [random_bits(i) random_bits(i+1)]==[1 0]
symbol_QPSK(floor(i/2+1))=1-1j;
elseif [random_bits(i) random_bits(i+1)]==[0 0]
symbol_QPSK(floor(i/2+1))=-1-1j;
end
end
Eb_QPSK=1;
Es_QPSK=2*Eb_QPSK;
N0_QPSK=Eb_QPSK./(10.^(SNR_range/10));
for i=1:length(SNR_range)
AWGN_channel_QPSK=randn(1,num_bits/2)*sqrt(N0_QPSK(i)/2)+1
j*randn(1,num_bits/2)*sqrt(N0_QPSK(i)/2);     %%generate
gaussian noise
data_QPSK_channel=symbol_QPSK+AWGN_channel_QPSK;   %%add
noise to symbols

%%Demapping%%
for n=1:num_bits/2
```

```
if real(data_QPSK_channel(n)) >=0 &&
imag(data_QPSK_channel(n)) >=0
Demapped_QPSK(n*2-1)=1;
Demapped_QPSK(n*2)=1;
elseif  real(data_QPSK_channel(n)) >=0 &&
imag(data_QPSK_channel(n))<0
Demapped_QPSK(n*2-1)=1;
Demapped_QPSK(n*2)=0;
elseif  real(data_QPSK_channel(n)) <0 &&
imag(data_QPSK_channel(n))>=0
Demapped_QPSK(n*2-1)=0;
Demapped_QPSK(n*2)=1;
elseif  real(data_QPSK_channel(n)) <0 &&
imag(data_QPSK_channel(n))<0
Demapped_QPSK(n*2-1)=0;
Demapped_QPSK(n*2)=0;
end
end
num_error_QPSK=0;
for k=1:num_bits
if Demapped_QPSK(k)~=random_bits(k)
num_error_QPSK=num_error_QPSK+1;
end
end
BER_QPSK(i)=num_error_QPSK/num_bits;
end
BER_QPSK_theortical=0.5*erfc(sqrt(10.^(SNR_range/10)));
figure
semilogy(SNR_range,BER_QPSK,'b') ;
hold on;
semilogy( SNR_range , BER_QPSK_theortical,'r--') ;
xlabel('Eb/No (dB)');
ylabel('BER');
ylim([10^-4 , 1]);
legend('Actual BER' , 'Theoretical BER ') ;
grid on
title('QPSK Modulation');
saveas(gcf, fullfile('G:\Comm3', 'QPSK.jpg'),'jpg');



%%%%%%%%%%% QPSK NOT GRAY CODE%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%QPSK%%%%%%%%%%%%%%%%%%%
```

```
symbol_QPSK_not_grey=zeros(1,(num_bits/2));          %two
bits are represented in one symbol
Demapped_QPSK_not_grey=zeros(1,num_bits);
%%%%%maping%%%%%%%
for i=1:2:num_bits
if [random_bits(i) random_bits(i+1)]==[1 1]
symbol_QPSK_not_grey(floor(i/2+1))=1-1j;
elseif [random_bits(i) random_bits(i+1)]==[0 1]
symbol_QPSK_not_grey(floor(i/2+1))=-1+1j;
elseif [random_bits(i) random_bits(i+1)]==[1 0]
symbol_QPSK_not_grey(floor(i/2+1))=1+1j;
elseif [random_bits(i) random_bits(i+1)]==[0 0]
symbol_QPSK_not_grey(floor(i/2+1))=-1-1j;
end
end
Eb_QPSK_not_grey=1;
Es_QPSK_not_grey=2*Eb_QPSK_not_grey;
N0_QPSK_not_grey=Eb_QPSK_not_grey./(10.^(SNR_range/10));
for i=1:length(SNR_range)
AWGN_channel_QPSK_not_grey=randn(1,num_bits/2)*sqrt(N0_QPS
K_not_grey(i)/2)+1j*randn(1,num_bits/2)*sqrt(N0_QPSK_not_g
rey(i)/2);     %%generate gaussian noise
data_QPSK_channel_not_grey=symbol_QPSK_not_grey+AWGN_chann
el_QPSK_not_grey;   %%add noise to symbols

%%Demapping%%
for n=1:num_bits/2
if real(data_QPSK_channel_not_grey(n)) >=0 &&
imag(data_QPSK_channel_not_grey(n)) >=0
Demapped_QPSK_not_grey(n*2-1)=1;
Demapped_QPSK_not_grey(n*2)=0;
elseif  real(data_QPSK_channel_not_grey(n)) >=0 &&
imag(data_QPSK_channel_not_grey(n))<0
Demapped_QPSK_not_grey(n*2-1)=1;
Demapped_QPSK_not_grey(n*2)=1;
elseif  real(data_QPSK_channel_not_grey(n)) <0 &&
imag(data_QPSK_channel_not_grey(n))>=0
Demapped_QPSK_not_grey(n*2-1)=0;
Demapped_QPSK_not_grey(n*2)=1;
elseif  real(data_QPSK_channel_not_grey(n)) <0 &&
imag(data_QPSK_channel_not_grey(n))<0
Demapped_QPSK_not_grey(n*2-1)=0;
Demapped_QPSK_not_grey(n*2)=0;
end
```

```
end
num_error_QPSK_not_grey=0;
for k=1:num_bits
if Demapped_QPSK_not_grey(k)~=random_bits(k)
num_error_QPSK_not_grey=num_error_QPSK_not_grey+1;
end
end
BER_QPSK_not_grey(i)=num_error_QPSK_not_grey/num_bits;
end
%BER_QPSK_theortical_not_grey=erfc(sqrt(10.^(SNR_range/10)
));
figure;
semilogy(SNR_range,BER_QPSK_not_grey,'b') ;
hold on;
%semilogy( SNR_range , BER_QPSK_theortical_not_grey,'r--')
;
xlabel('Eb/No (dB)');
ylabel('BER');
ylim([10^-4 , 1]);
legend('Actual BER' , 'Theoretical BER ') ;
grid on
title('QPSK NOT GREY Modulation');
saveas(gcf, fullfile('G:\Comm3',
'QPSK_not_grey.jpg'),'jpg');
figure;    %grey QPSK and NOT GREY QPSK
semilogy(SNR_range,BER_QPSK,'b') ;
hold on;
semilogy( SNR_range , BER_QPSK_not_grey,'r') ;
xlabel('Eb/No (dB)');
ylabel('BER');
ylim([10^-4 , 1]);
legend('Grey QPSK BER' , 'Non Grey QPSK BER') ;
grid on
title('QPSK GREY VS QPSK NOT GREY Modulation');
saveas(gcf, fullfile('G:\Comm3', 'greyVsnot.jpg'),'jpg');



%%%%%%%%%%%%%%8PSK%%%%%%%%%%%%%%%%
symbol_8PSK=zeros(1,(num_bits/3));          %three bits
are represented in one symbol
Demapped_8PSK=zeros(1,num_bits);
%%%%%maping%%%%%%%
for i=1:3:num_bits
```

```
if [random_bits(i) random_bits(i+1) random_bits(i+2)]==[0
0 0]
symbol_8PSK(floor(i/3+1))=cos(0)+1j*sin(0);
elseif [random_bits(i) random_bits(i+1)
random_bits(i+2)]==[0 0 1]
symbol_8PSK(floor(i/3+1))=cos(pi/4)+1j*sin(pi/4);
elseif [random_bits(i) random_bits(i+1)
random_bits(i+2)]==[0 1 1]
symbol_8PSK(floor(i/3+1))=cos(pi/2)+1j*sin(pi/2);
elseif [random_bits(i) random_bits(i+1)
random_bits(i+2)]==[0 1 0]
symbol_8PSK(floor(i/3+1))=cos(3*pi/4)+1j*sin(3*pi/4);
elseif [random_bits(i) random_bits(i+1)
random_bits(i+2)]==[1 1 0]
symbol_8PSK(floor(i/3+1))=cos(pi)+1j*sin(pi);
elseif [random_bits(i) random_bits(i+1)
random_bits(i+2)]==[1 1 1]
symbol_8PSK(floor(i/3+1))=cos(5*pi/4)+1j*sin(5*pi/4);
elseif [random_bits(i) random_bits(i+1)
random_bits(i+2)]==[1 0 1]
symbol_8PSK(floor(i/3+1))=cos(6*pi/4)+1j*sin(6*pi/4);
elseif [random_bits(i) random_bits(i+1)
random_bits(i+2)]==[1 0 0]
symbol_8PSK(floor(i/3+1))=cos(7*pi/4)+1j*sin(7*pi/4);
end
end
Es_8PSK=1;
Eb_8PSK=Es_8PSK/3;
N0_8PSK=Eb_8PSK./(10.^(SNR_range/10));
for i=1:length(SNR_range)
AWGN_channel_8PSK=randn(1,num_bits/3)*sqrt(N0_8PSK(i)/2)+1
j*randn(1,num_bits/3)*sqrt(N0_8PSK(i)/2);     %%generate
gaussian noise
data_8PSK_channel=symbol_8PSK+AWGN_channel_8PSK;   %%add
noise to symbols

%%Demapping%%
%%%%%% demaping here be check region using angles
for n=1:num_bits/3
if angle(data_8PSK_channel(n)) >=-pi/8 &&
angle(data_8PSK_channel(n)) <pi/8
Demapped_8PSK(n*3-2)=0;
Demapped_8PSK(n*3-1)=0;
Demapped_8PSK(n*3)=0;
```

```
elseif angle(data_8PSK_channel(n)) >=pi/8 &&
angle(data_8PSK_channel(n)) <3*pi/8
Demapped_8PSK(n*3-2)=0;
Demapped_8PSK(n*3-1)=0;
Demapped_8PSK(n*3)=1;
elseif angle(data_8PSK_channel(n)) >=3*pi/8 &&
angle(data_8PSK_channel(n)) <5*pi/8
Demapped_8PSK(n*3-2)=0;
Demapped_8PSK(n*3-1)=1;
Demapped_8PSK(n*3)=1;
elseif angle(data_8PSK_channel(n)) >=5*pi/8 &&
angle(data_8PSK_channel(n)) <7*pi/8
Demapped_8PSK(n*3-2)=0;
Demapped_8PSK(n*3-1)=1;
Demapped_8PSK(n*3)=0;
elseif angle(data_8PSK_channel(n)) >=-7*pi/8 &&
angle(data_8PSK_channel(n)) <-5*pi/8
Demapped_8PSK(n*3-2)=1;
Demapped_8PSK(n*3-1)=1;
Demapped_8PSK(n*3)=1;
elseif angle(data_8PSK_channel(n)) >=-5*pi/8 &&
angle(data_8PSK_channel(n)) <-3*pi/8
Demapped_8PSK(n*3-2)=1;
Demapped_8PSK(n*3-1)=0;
Demapped_8PSK(n*3)=1;
elseif angle(data_8PSK_channel(n)) >=-3*pi/8 &&
angle(data_8PSK_channel(n)) <-pi/8
Demapped_8PSK(n*3-2)=1;
Demapped_8PSK(n*3-1)=0;
Demapped_8PSK(n*3)=0;
else
Demapped_8PSK(n*3-2)=1;
Demapped_8PSK(n*3-1)=1;
Demapped_8PSK(n*3)=0;

end
end
num_error_8PSK=0;
for k=1:num_bits
if Demapped_8PSK(k)~=random_bits(k)
num_error_8PSK=num_error_8PSK+1;
end
end
BER_8PSK(i)=num_error_8PSK/num_bits;
```

```
end
BER_8PSK_theortical=1/3*erfc(sqrt(3*(10.^(SNR_range/10)))*
sin(pi/8));
figure
semilogy(SNR_range,BER_8PSK,'b') ;
hold on;
semilogy( SNR_range , BER_8PSK_theortical,'r--') ;
xlabel('Eb/No (dB)');
ylabel('BER');
ylim([10^-4 , 1]);
legend('Actual BER' , 'Theoretical BER ') ;
grid on
title('8PSK Modulation');
saveas(gcf, fullfile('G:\Comm3', 'EightPSK.jpg'),'jpg');



%%%%%%%%%%%%%%%%%%16 QAM%%%%%%%%%%%%%%%%%


symbol_16QAM=zeros(1,(num_bits/4));          %FOUR bits
are represented in one symbol
Demapped_16QAM=zeros(1,num_bits);
%%%%%maping%%%%%%%
for i=1:4:num_bits
if [random_bits(i) random_bits(i+1) random_bits(i+2)
random_bits(i+3)]==[0 0 0 0]
symbol_16QAM(floor(i/4+1))=-3-3j;
elseif [random_bits(i) random_bits(i+1) random_bits(i+2)
random_bits(i+3)]==[0 0 0 1]
symbol_16QAM(floor(i/4+1))=-3-1j;
elseif [random_bits(i) random_bits(i+1) random_bits(i+2)
random_bits(i+3)]==[0 0 1 0]
symbol_16QAM(floor(i/4+1))=-3+3j;
elseif [random_bits(i) random_bits(i+1) random_bits(i+2)
random_bits(i+3)]==[0 0 1 1]
symbol_16QAM(floor(i/4+1))=-3+1j;
elseif [random_bits(i) random_bits(i+1) random_bits(i+2)
random_bits(i+3)]==[0 1 0 0]
symbol_16QAM(floor(i/4+1))=-1-3j;
elseif [random_bits(i) random_bits(i+1) random_bits(i+2)
random_bits(i+3)]==[0 1 0 1]
symbol_16QAM(floor(i/4+1))=-1-1j;
```

```
elseif [random_bits(i) random_bits(i+1) random_bits(i+2)
random_bits(i+3)]==[0 1 1 0]
symbol_16QAM(floor(i/4+1))=-1+3j;
elseif [random_bits(i) random_bits(i+1) random_bits(i+2)
random_bits(i+3)]==[0 1 1 1]
symbol_16QAM(floor(i/4+1))=-1+1j;
elseif [random_bits(i) random_bits(i+1) random_bits(i+2)
random_bits(i+3)]==[1 0 0 0]
symbol_16QAM(floor(i/4+1))=3-3j;
elseif [random_bits(i) random_bits(i+1) random_bits(i+2)
random_bits(i+3)]==[1 0 0 1]
symbol_16QAM(floor(i/4+1))=3-1j;
elseif [random_bits(i) random_bits(i+1) random_bits(i+2)
random_bits(i+3)]==[1 0 1 0]
symbol_16QAM(floor(i/4+1))=3+3j;
elseif [random_bits(i) random_bits(i+1) random_bits(i+2)
random_bits(i+3)]==[1 0 1 1]
symbol_16QAM(floor(i/4+1))=3+1j;
elseif [random_bits(i) random_bits(i+1) random_bits(i+2)
random_bits(i+3)]==[1 1 0 0]
symbol_16QAM(floor(i/4+1))=1-3j;
elseif [random_bits(i) random_bits(i+1) random_bits(i+2)
random_bits(i+3)]==[1 1 0 1]
symbol_16QAM(floor(i/4+1))=1-1j;
elseif [random_bits(i) random_bits(i+1) random_bits(i+2)
random_bits(i+3)]==[1 1 1 0]
symbol_16QAM(floor(i/4+1))=1+3j;
elseif [random_bits(i) random_bits(i+1) random_bits(i+2)
random_bits(i+3)]==[1 1 1 1]
symbol_16QAM(floor(i/4+1))=1+1j;
end
end
Es_16QAM=10;
Eb_16QAM=Es_16QAM/4;
N0_16QAM=Eb_16QAM./(10.^(SNR_range/10));
for i=1:length(SNR_range)
AWGN_channel_16QAM=randn(1,num_bits/4)*sqrt(N0_16QAM(i)/2)
+1j*randn(1,num_bits/4)*sqrt(N0_16QAM(i)/2);
%%generate gaussian noise
data_16QAM_channel=symbol_16QAM+AWGN_channel_16QAM;
%%add noise to symbols

%%Demapping%%
%%%%%% demaping here be check region
```

```
for n=1:num_bits/4
if real(data_16QAM_channel(n))<-2 &&
imag(data_16QAM_channel(n))<-2
Demapped_16QAM(n*4-3)=0;
Demapped_16QAM(n*4-2)=0;
Demapped_16QAM(n*4-1)=0;
Demapped_16QAM(n*4)=0;
elseif real(data_16QAM_channel(n))<-2 &&
imag(data_16QAM_channel(n))>=-2
&&imag(data_16QAM_channel(n))<0
Demapped_16QAM(n*4-3)=0;
Demapped_16QAM(n*4-2)=0;
Demapped_16QAM(n*4-1)=0;
Demapped_16QAM(n*4)=1;
elseif real(data_16QAM_channel(n))<-2 &&
imag(data_16QAM_channel(n))>=0
&&imag(data_16QAM_channel(n))<2
Demapped_16QAM(n*4-3)=0;
Demapped_16QAM(n*4-2)=0;
Demapped_16QAM(n*4-1)=1;
Demapped_16QAM(n*4)=1;
elseif real(data_16QAM_channel(n))<-2 &&
imag(data_16QAM_channel(n))>=2
Demapped_16QAM(n*4-3)=0;
Demapped_16QAM(n*4-2)=0;
Demapped_16QAM(n*4-1)=1;
Demapped_16QAM(n*4)=0;
elseif real(data_16QAM_channel(n))>=-2 &&
imag(data_16QAM_channel(n))<-2
&&real(data_16QAM_channel(n))<0
Demapped_16QAM(n*4-3)=0;
Demapped_16QAM(n*4-2)=1;
Demapped_16QAM(n*4-1)=0;
Demapped_16QAM(n*4)=0;
elseif real(data_16QAM_channel(n))>=-2 &&
imag(data_16QAM_channel(n))>=-2
&&real(data_16QAM_channel(n))<0 &&
imag(data_16QAM_channel(n))<0
Demapped_16QAM(n*4-3)=0;
Demapped_16QAM(n*4-2)=1;
Demapped_16QAM(n*4-1)=0;
Demapped_16QAM(n*4)=1;
```

```
elseif real(data_16QAM_channel(n))>=-2 &&
imag(data_16QAM_channel(n))<2 &&
real(data_16QAM_channel(n))<0 &&
imag(data_16QAM_channel(n))>0
Demapped_16QAM(n*4-3)=0;
Demapped_16QAM(n*4-2)=1;
Demapped_16QAM(n*4-1)=1;
Demapped_16QAM(n*4)=1;
elseif real(data_16QAM_channel(n))>=-2 &&
imag(data_16QAM_channel(n))>=2 &&
real(data_16QAM_channel(n))<0
Demapped_16QAM(n*4-3)=0;
Demapped_16QAM(n*4-2)=1;
Demapped_16QAM(n*4-1)=1;
Demapped_16QAM(n*4)=0;
elseif real(data_16QAM_channel(n))>=0 &&
imag(data_16QAM_channel(n))<-2  &&
real(data_16QAM_channel(n))<2
Demapped_16QAM(n*4-3)=1;
Demapped_16QAM(n*4-2)=1;
Demapped_16QAM(n*4-1)=0;
Demapped_16QAM(n*4)=0;
elseif real(data_16QAM_channel(n))>=0 &&
imag(data_16QAM_channel(n))>=-2 &&
real(data_16QAM_channel(n))<2 &&
imag(data_16QAM_channel(n))<0
Demapped_16QAM(n*4-3)=1;
Demapped_16QAM(n*4-2)=1;
Demapped_16QAM(n*4-1)=0;
Demapped_16QAM(n*4)=1;
elseif real(data_16QAM_channel(n))>=0 &&
imag(data_16QAM_channel(n))>=0 &&
real(data_16QAM_channel(n))<2 &&
imag(data_16QAM_channel(n))<2
Demapped_16QAM(n*4-3)=1;
Demapped_16QAM(n*4-2)=1;
Demapped_16QAM(n*4-1)=1;
Demapped_16QAM(n*4)=1;
elseif real(data_16QAM_channel(n))>=0 &&
imag(data_16QAM_channel(n))>=2
&&real(data_16QAM_channel(n))<2
Demapped_16QAM(n*4-3)=1;
Demapped_16QAM(n*4-2)=1;
```

```
Demapped_16QAM(n*4-1)=1;
Demapped_16QAM(n*4)=0;
elseif real(data_16QAM_channel(n))>=2 &&
imag(data_16QAM_channel(n))<-2
Demapped_16QAM(n*4-3)=1;
Demapped_16QAM(n*4-2)=0;
Demapped_16QAM(n*4-1)=0;
Demapped_16QAM(n*4)=0;
elseif real(data_16QAM_channel(n))>=2 &&
imag(data_16QAM_channel(n))>=-2 &&
imag(data_16QAM_channel(n))<0
Demapped_16QAM(n*4-3)=1;
Demapped_16QAM(n*4-2)=0;
Demapped_16QAM(n*4-1)=0;
Demapped_16QAM(n*4)=1;
elseif real(data_16QAM_channel(n))>=2 &&
imag(data_16QAM_channel(n))>=0 &&
imag(data_16QAM_channel(n))<2
Demapped_16QAM(n*4-3)=1;
Demapped_16QAM(n*4-2)=0;
Demapped_16QAM(n*4-1)=1;
Demapped_16QAM(n*4)=1;
elseif real(data_16QAM_channel(n))>=2 &&
imag(data_16QAM_channel(n))>=2
Demapped_16QAM(n*4-3)=1;
Demapped_16QAM(n*4-2)=0;
Demapped_16QAM(n*4-1)=1;
Demapped_16QAM(n*4)=0;
end
end
num_error_16QAM=0;
for k=1:num_bits
if Demapped_16QAM(k)~=random_bits(k)
num_error_16QAM=num_error_16QAM+1;
end
end
BER_16QAM(i)=num_error_16QAM/num_bits;
end
BER_16QAM_theortical=3/8*erfc(sqrt((10.^(SNR_range/10))/2.
5));
figure
semilogy(SNR_range,BER_16QAM,'b') ;
hold on;
```

```
semilogy( SNR_range , BER_16QAM_theortical,'r--') ;
ylim([10^-2,1]);
xlabel('Eb/No (dB)');
ylabel('BER');
ylim([10^-4 , 1]);
legend('Actual BER' , 'Theoretical BER ') ;
grid on
title('16QAM Modulation');
saveas(gcf, fullfile('G:\Comm3', 'SixteebQAM.jpg'),'jpg');

%%%%%%%%%%%Plot all%%%%%%%%%%%%%%%%%

figure
semilogy(SNR_range,BER_BPSK ,'Color','b') ;
hold on;
semilogy(SNR_range,BER_QPSK , 'Color','m') ;
semilogy(SNR_range,BER_8PSK , 'Color','g') ;
semilogy(SNR_range,BER_16QAM , 'Color','r') ;
semilogy( SNR_range , BER_BPSK_theoritcal  ,'--' ,
'Color', 'c') ;
semilogy( SNR_range , BER_QPSK_theortical ,'--' ,
'Color','k' ) ;
semilogy( SNR_range , BER_8PSK_theortical ,'--' , 'Color',
'y') ;
semilogy( SNR_range , BER_16QAM_theortical  ,'--' ,
'Color',[0.5, 0.2, 0.8]) ;
xlabel('Eb/No (dB)');
ylabel('BER');
ylim([10^-4 , 1]);
legend('Actual BPSK' , 'Actual QPSK' , 'Actual 8PSK' ,
'Actual 16QAM' , 'Theoretical BPSK' , 'Theoretical QPSK' ,
'Theoretical 8PSK' , 'Theoretical 16QAM') ;
title('BPSK & QPSK & 8PSK & 16QAM');
saveas(gcf, fullfile('G:\Comm3', 'ALL.jpg'),'jpg');
hold off;
%%%%%%%%%%%%%%BFSK%%%%%%%%%%%%%%%%%%%%
Eb_BFSK=1;
symbol_BFSK=zeros(1,num_bits);
Demapped_BFSK=zeros(1,num_bits);
%%%%%%%%%maping BFSK%%%%%%%
for k=1:num_bits
if random_bits(k)== 1
symbol_BFSK(k)=1;
else
```

```
symbol_BFSK(k)=1j;
end
end

%%%%%%%%%Noisy Channel%%%%%%%
N0_BFSK=Eb_BFSK./(10.^(SNR_range/10));
for i=1:length(SNR_range)
AWGN_channel_BFSK=randn(1,num_bits)*sqrt(N0_BFSK(i)/2)+1j*
randn(1,num_bits)*sqrt(N0_BFSK(i)/2);     %%generate
gaussian noise
data_BFSK_channel=symbol_BFSK+AWGN_channel_BFSK;   %%add
noise to symbols

%%demapping%%
for n=1:num_bits
if angle(data_BFSK_channel(n))>=-3*pi/4 &&
angle(data_BFSK_channel(n))<pi/4
Demapped_BFSK(n)=1;
else
Demapped_BFSK(n)=0;
end
end
num_error_BFSK=0;
for n=1:num_bits
if Demapped_BFSK(n)~=random_bits(n)
num_error_BFSK=num_error_BFSK+1;
end
end
BER_BFSK(i)=num_error_BFSK/num_bits;
end
BER_BFSK_theoritcal=0.5*erfc(sqrt(10.^(SNR_range/10)/2));
figure
semilogy(SNR_range,BER_BFSK,'b') ;
hold on;
semilogy( SNR_range , BER_BFSK_theoritcal ,'r--') ;
xlabel('Eb/No (dB)');
ylabel('BER');
ylim([10^-4 , 1]);
legend('Actual BER' , 'Theoretical BER ') ;
grid on
title('BFSK Modulation');
saveas(gcf, fullfile('G:\Comm3', 'BFSK.jpg'),'jpg');
%%%%%%%%%%%%%%%%%PSD BFSK%%%%%%%%%%%%%%%%%%%%%%
```

```
ensemble_size = 25000;
num_bits_PSD = 100;
Eb_BFSK = 1;
bit_time = 7;
sampling_time = 1;
%numbers of samples in on bit=7
num_samples_bit = bit_time / sampling_time;
num_samples = num_samples_bit*num_bits_PSD;

S1_BB =sqrt(2*Eb_BFSK/(bit_time));
step = (bit_time/num_samples_bit);
t = 0: step :bit_time - (bit_time/num_samples_bit);
S2_BB =
sqrt(2*Eb_BFSK/(bit_time))*cos(2*pi*(1/bit_time)*t)+1i*sqr
t(2*Eb_BFSK/(bit_time))*sin(2*pi*(1/bit_time)*t);



data_PSD = randi([0,1], ensemble_size, num_bits_PSD+1);
%secondly, storing the data after transforming it form
bits to symbols
data_symbols=zeros(ensemble_size,(num_bits_PSD+1)*num_samp
les_bit);
for i=1:ensemble_size
     for j = 1:num_bits_PSD+1
        if data_PSD(i,j)
            for k = 0 : num_samples_bit-1
                data_symbols(i,num_samples_bit*(j-1)+1+k)=
S2_BB(k+1);
            end
        else
            for k = 0 : num_samples_bit-1
                data_symbols(i,num_samples_bit*(j-1)+1+k)=
S1_BB;
             end
        end
     end
end

td = randi([0,(num_samples_bit-1)],ensemble_size,1);
data = zeros(ensemble_size, num_samples);
for i = 1:ensemble_size
    data(i,:)= data_symbols(i, td(i)+1 : num_samples +
td(i));
```

```
end
stat_autocorr = zeros(size(data(1,1:end)));
for i = (-num_samples/2+1) : num_samples/2
    stat_autocorr(1,num_samples/2+i) =
sum((conj(data(:,num_samples/2)) .*
data(:,num_samples/2+i))) / ensemble_size;
end
%stat_autocorr = cat (2,
conj(fliplr(stat_autocorr(2:num_samples))),
stat_autocorr);
k = -num_samples + 1: num_samples - 1;
k = [1:700];
fs = 1;
Freq = (-num_bits_PSD*7/2:num_bits_PSD*7/2-1) *
fs/(num_bits_PSD*7) ;
%Scale by Tb
FrequencyNoramlize= Freq*bit_time;
FirstDelta = 100*double(abs(Freq - 1/(2*bit_time)) <
0.00001); % Tolerance set to 0.00001
SecondDelta = 100*double(abs(Freq + 1/(2*bit_time)) <
0.00001); % Tolerance set to 0.00001
PSD_BFSK_theoritcal = (2/bit_time)*(FirstDelta
+SecondDelta) + (8*cos(pi*bit_time*Freq).^2)./(pi^2
*(4*bit_time^2 * Freq.^2 -1).^2);
figure('Name','PSD');
%%%Normalize by 4
ActaulPSDNormalize = abs(fftshift(fft(stat_autocorr)));
TheoritcalPSDNormalize = PSD_BFSK_theoritcal;
plot(FrequencyNoramlize,ActaulPSDNormalize);
hold on;
plot (FrequencyNoramlize,TheoritcalPSDNormalize);
title("BFSK PSD");
xlabel("Frequency (Hz)");
ylabel("PSD");
ylim([0,2]);
legend('Actual PSD' , 'Theoretical PSD')
saveas(gcf, fullfile('G:\Comm3', 'PSD.jpg'),'jpg');

%%%%%%%
```