Faculty of Engineering
Cairo University

CAIRO UNIVERSITY

Digital Communications modulation project (project #3)

Submitted by:

	Name	Sec	Bn	ID
1	عمر محمد طلبه محمد	3	6	9202985
2	عمر محمد مصطفى عبداللطيف	3	8	9202989

Submitted to:
Dr. Muhammed Khairi
Eng. Muhammed Wael

ELC3070

Constellation of different modulation techniques:

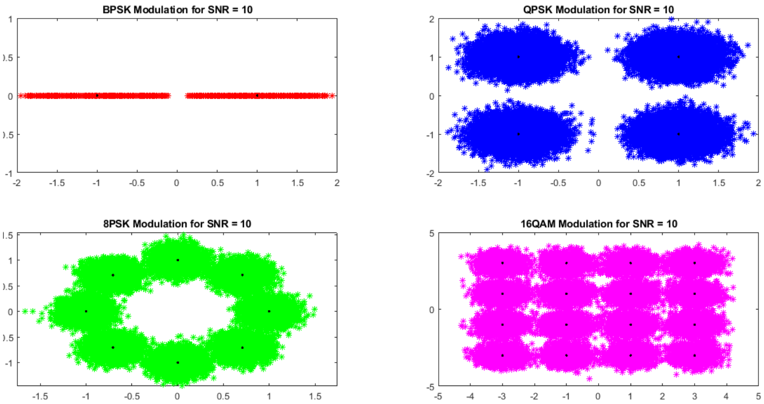

Figure 1: Constellation of the BPSK, QPSK, 8PSK, 16QAM,

Bit error rate (BER) of different modulation techniques:

I. Binary phase shift keying (BPSK)

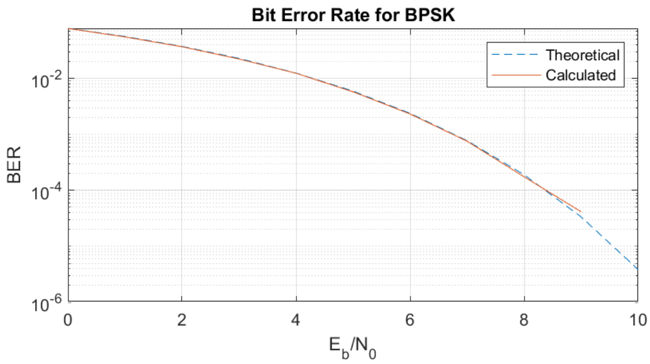

Figure 2: Bit error rate (BER) of BPSK

The theoretical BER of the BPSK is computed given by:

$$BER_{theoritical} = 0.5\ erfc\left(\sqrt{\frac{E_b}{N_0}}\right), where\ E_b = 1.$$

II. Quadrature phase shift keying (QPSK)

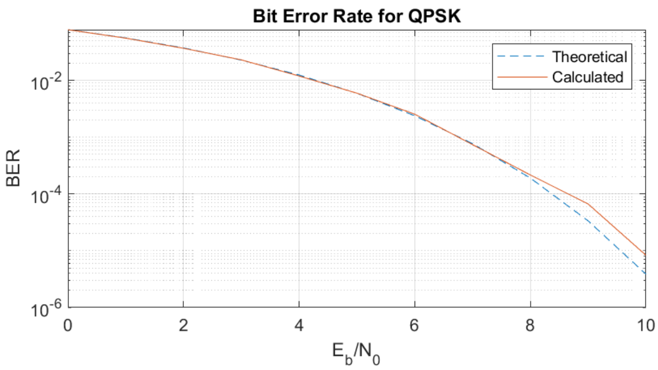

Figure 3: Bit error rate (BER) of QPSK

The theoretical BER of the QPSK is computed given by:

$$BER_{theoritical} = 0.5\ erfc\left(\sqrt{\frac{E_b}{N_0}}\right), where\ E_b = 1.$$

III. Eight-ary phase shift keying (8PSK)

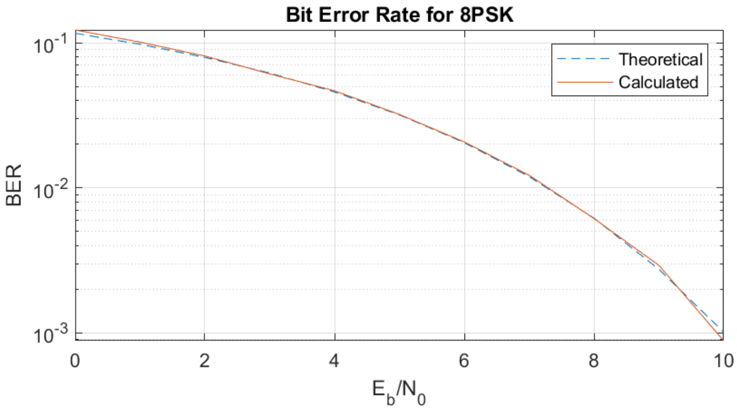

Figure 4: Bit error rate (BER) of 8PSK

The theoretical BER of the QPSK is computed given by:

$$BER_{theoritical} = \frac{1}{3} erfc\left(\sqrt{\frac{E}{N_0}} \sin\left(\frac{\pi}{M}\right)\right), where\ E = \log_2(M)\, E_b\ \ and\ E_b = \frac{1}{3}$$

So, the law is reduced to:

$$BER_{theoritical} = \frac{1}{3} erfc\left(\sqrt{\frac{1}{N_0}} * \sin\left(\frac{\pi}{8}\right)\right)$$

IV. 16-Quadrature amplitude modulation (16QAM)

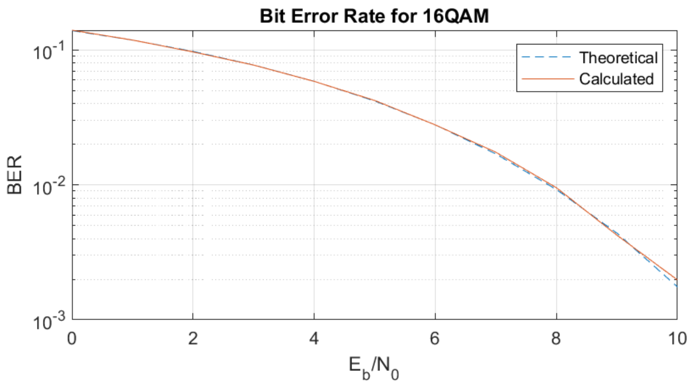

Figure 4: Bit error rate (BER) of 16QAM

The theoretical BER of the 16QAM is computed given by:

$$BER_{theoritical} = \frac{1.5}{4} erfc\left(\sqrt{\frac{E_b}{N_0}}\right), where\ E_b = 2.5$$

V. BPSK, QPSK, 8PSK and 16QAM modulations on the same graph.

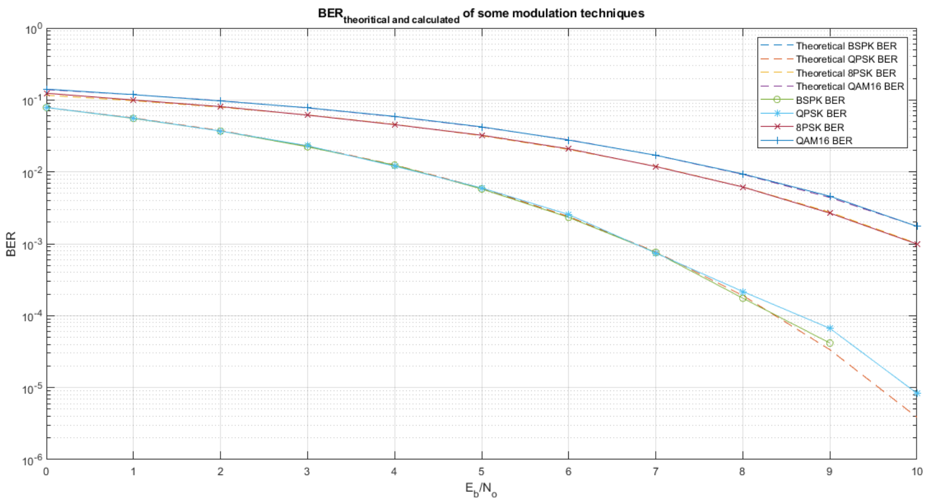

Figure 5: (BER) of BPSK, QPSK, 8PSK, and 16QAM

Comment:

- As it is noticed from the previous graph that the 16QAM modulation technique obtains the worst BER of all other simulated modulation techniques. Therefore, as the number of bits per symbol increases, it leads to deterioration in the BER. It is concluded as the 4 bits per symbol (16QAM) is the worst after it the 3 bits per symbol (8PSK).
- The BER in the BPSK and QPSK is quite similar in response.
- By increasing the signal to noise ratio (SNR) which is computed by:

$$SNR = \frac{E_b}{N_0}$$

It leads to a better performance. In other words, less BER.

VI. BER of grey encoded QPSK and another encoding method of QPSK

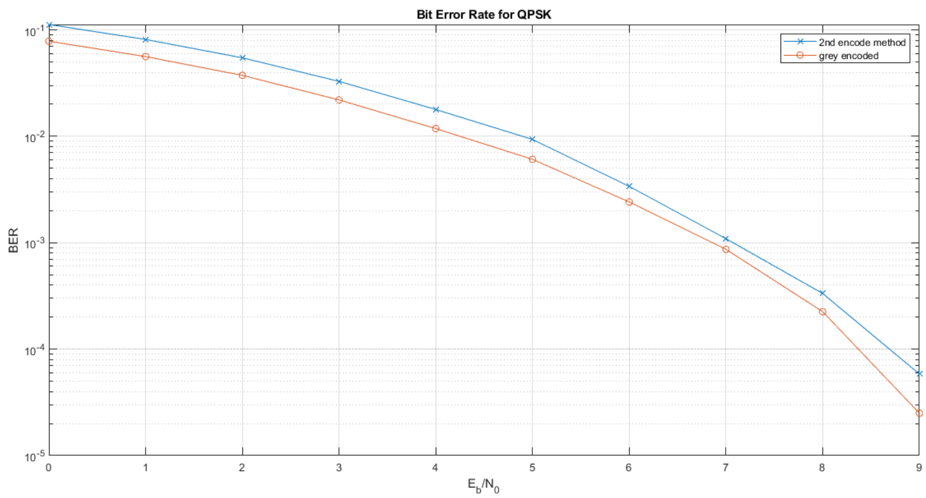

Figure 6: (BER) of grey QPSK and other encoding method

Comment:

It was concluded from the graph that the grey encoding method has obtained a better BER than the other encoding method as when the error happens it may change one bit yet the other way of encoding leads to a higher probability of error.

Binary Frequency Shift Keying (BFSK)

System Basis function

$$\phi_i(t) = \sqrt{\frac{2}{T_b}}\cos(2\pi f_i t) \quad \text{where } f_i = \frac{n_c + i}{T_\mathrm{b}}, i = 1,2$$

Where T_b represents the bit duration or the time taken to transmit a single bit. It is a constant value that defines the duration of each bit in the BFSK signal. And n_c represents the center frequency offset. It is a constant value that determines the separation between the two carrier frequencies used in BFSK modulation

Base band equivalent signal

$$s(t) = \sqrt{\frac{2E_b}{T_b}}\cos(2\pi f_1 t) \ \text{ or } \ \sqrt{\frac{2E_b}{T_b}}\cos(2\pi f_2 t)$$

You can suppose that the carrier frequency is any frequency that is close to the frequencies of interest. You can take it as f1, f2, or (f1+f2)/2. Assuming fc = f1**,** In this case

$$s_1(t) = \sqrt{\frac{2E_b}{T_b}}\cos(2\pi f_c t)$$

$$\begin{aligned} s_2(t) &= \sqrt{\frac{2E_b}{T_b}}\cos(2\pi(f_c + \Delta f)t) \\ &= \sqrt{\frac{2E_b}{T_b}}\cos(2\pi f_c t)\cos(2\pi\Delta f t) - \sqrt{\frac{2E_b}{T_b}}\sin(2\pi f_c t)\sin(2\pi\Delta f t) \end{aligned}$$

So base band equivalent signal will be

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

$$s_{2BB}(t) = \sqrt{\frac{2E_b}{T_b}}\cos(2\pi\Delta f t) + j\sqrt{\frac{2E_b}{T_b}}\sin(2\pi\Delta f t) = \sqrt{\frac{2E_b}{T_b}}e^{(\mathrm{j}2\pi\Delta f t)}$$

Constellation of BFSK Modulation:

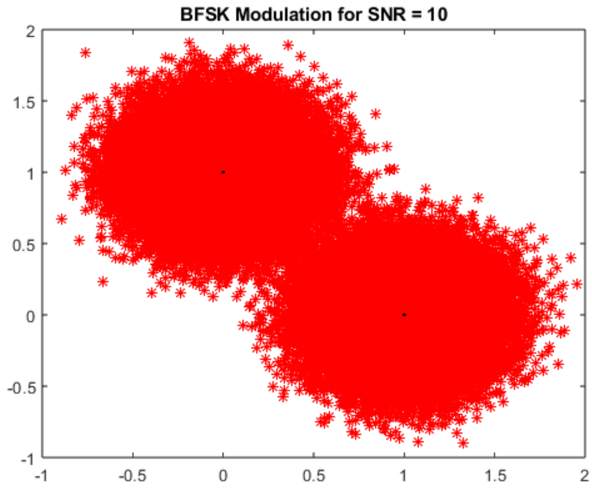

Figure 7 Constellation of BFSK Modulation

Bit error rate (BER) of BFSK Modulation:

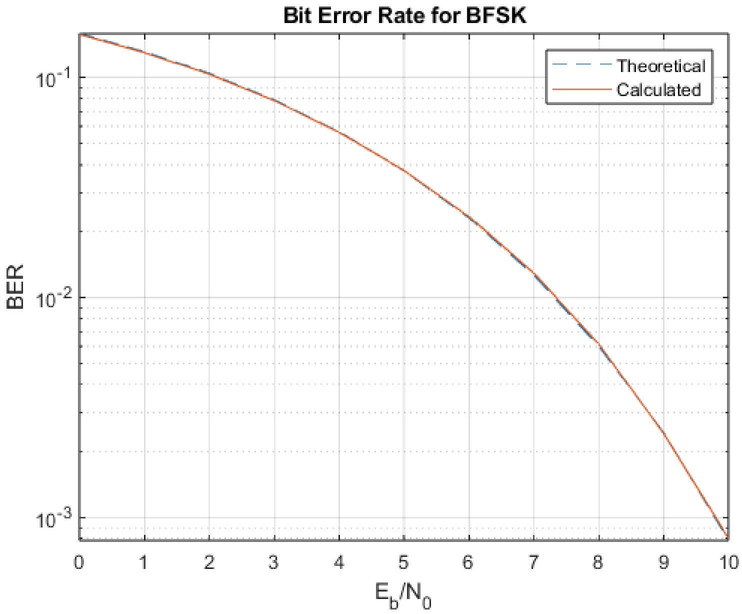

Figure 8 Bit error rate (BER) of BFSK Modulation

The theoretical BER of the BPSK is computed given by:

$$BER_{theoritical} = 0.5\, erfc\left(\sqrt{\frac{E_b}{2N_0}}\right), where\ E_b = 1$$

Power spectral density (PSD) of BFSK Modulation:

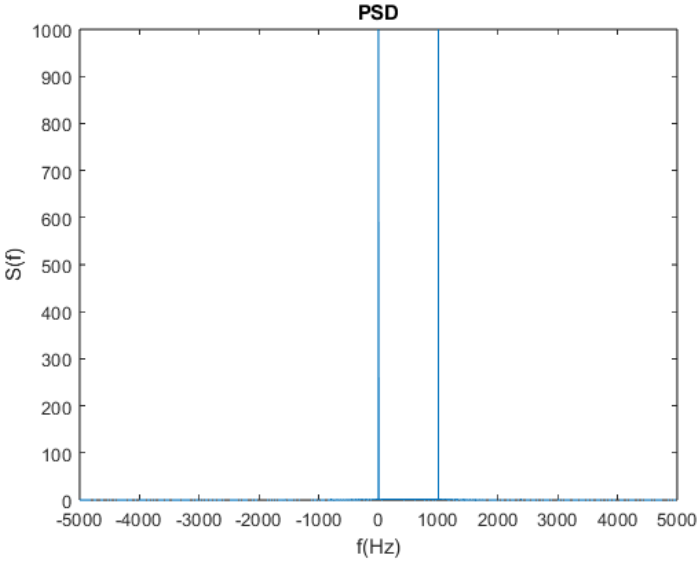

Figure 9 Power spectral density (PSD) of BFSK Modulation

two deltas are observed at the DC frequency (representing the zero bit, denoted as or $s_{1BB}(t)$)) and at 1000Hz (representing the one bit, denoted as or $s_{2BB}(t)$) These delta functions satisfy the following properties: $s_{1BB}(t) = \sqrt{\frac{2E_b}{T_b}}$is constant (DC), while $s_{2BB}(t) = \sqrt{\frac{2E_b}{T_b}} e^{(\mathrm{j}2\pi\Delta f t)}$, which is similar to s_{1BB} but shifted in frequency by Δf. In this case, Δf is equal to $1/T_b$, where Tb is 1e-3 seconds, resulting in $\Delta f = 1000\ Hz$.

MATLAB code

Please be advised that the version of MATLAB used in this project is R2021a (9.10.0.1602886). Furthermore, the code is accessible on GitHub via the following link (GitHub repository will be made public by the project deadline day).

Please note that the first code snippet provided does not include the implementation of the BFSK modulation technique. The BFSK modulation code is present in a separate code snippet, which is the second one.

```
%{
* FILE DESCRIPTION
* File: source_code.m
* Digital communication project-assignment
* Description: Simulation of different modulation techniques (BPSK, QPSK, 8PSK, 16QAM)
* Author:Omar Muhammad Mustafa, Omar Muhammad Tolba
* Date: 6th May 2023
%}
%% clear all WorkSpace Variables and Command Window
clc;
clear ;
close all;

%% initialization
Bits_Number = 1.2e5;                                        % number of bits to be generated
SNR_max_value = 10;

%% Generating the data bits
Bit_Stream = randi([0 1],1,Bits_Number);

%% BPSK Modulation
%initializations needed for Simulation of BPSK
BPSK_Eb = ((1 +1)/ 2) / 1;                               %Calculating Eb of BPSK modulation
BPSK_BER_Theo = zeros(1,SNR_max_value);                  %Vector to store the theortical BER for
different SNR channel
BPSK_BER = zeros(1,SNR_max_value);                       %Vector to store the calculated BER for
different SNR channel

%Mapper
BPSK_symbolStream = 2 * Bit_Stream - 1;                  %The sent symbol is either 1 or -1
%Channel (AWGN)
BPSK_channelNoise = zeros(length(Bit_Stream),SNR_max_value); %Matrix to store the noise in
each channel
BPSK_No = zeros(1,SNR_max_value + 1);

%Calculating No of the channel for different SNR
for SNR_dB = 1 : (SNR_max_value + 1)
    %Generating different noise vector for different channels with different SNR
    noise = randn(1,length(Bit_Stream));
    BPSK_No(SNR_dB) = BPSK_Eb/10.^((SNR_dB-1)/10);
    BPSK_channelNoise(:,SNR_dB) = noise * sqrt (BPSK_No(SNR_dB)/2); %scale the noise.
end

%Demapper
BPSK_demappedSymbol = zeros(1,length(Bit_Stream));   %Vector to store the demapped stream

for SNR_dB = 1 : (SNR_max_value + 1)
    BPSK_DataRx = BPSK_symbolStream + BPSK_channelNoise(:,SNR_dB)'; %Recieved Data

    %Demapping the recieved symbol stream for each channel
    for counter = 1 : Bits_Number
        if(BPSK_DataRx(1,counter) > 0)
            BPSK_demappedSymbol(1,counter) = 1;
        else
            BPSK_demappedSymbol(1,counter) = 0;
        end
    end
    [N_BER_BPSK,BPSK_BER(SNR_dB)] = symerr(BPSK_demappedSymbol,Bit_Stream); %Calculated BER
    BPSK_BER_Theo(SNR_dB)= 0.5 * erfc(sqrt(1/BPSK_No(SNR_dB))); %Theoritical BER
end
%Plotting constillation of BPSK
figure(1)
subplot(2,2,1,'LineWidth',3)
plot(real(BPSK_DataRx),imag(BPSK_DataRx),'r*',real(1),imag(0),'k.',real(-1),imag(0),'k.');
title('BPSK Modulation for SNR = 10')
```

```
%Plotting BER
figure(2)
subplot(2,2,1,'LineWidth',3)
EbN0_dB = 1:1:(SNR_max_value + 1) ;
%Plotting theoritical BER
semilogy((EbN0_dB - 1),BPSK_BER_Theo,'--')
%Plotting calculated BER
hold on
semilogy((EbN0_dB - 1),BPSK_BER,'-')
grid on
ylabel('BER')
xlabel('E_b/N_0')
title('Bit Error Rate for BPSK')
legend('Theoretical','Calculated');

%% QPSK
%initializations needed for Simulation of BPSK
QPSK_Eb = ((4 * 2) / 4 ) / 2;
QPSK_BER_Theo = zeros(1,SNR_max_value);             %Vector to store the theortical BER for
different SNR channel
QPSK_BER = zeros(1,SNR_max_value);                  %Vector to store the calculated BER for
different SNR channel
QPSK_BER_encode2 = zeros(1,SNR_max_value);                  %Vector to store the calculated
BER for different SNR channel


%Mapper
%Grouping the binary data into groups of 2 bits
QPSK_reshaped_binary_data = reshape(Bit_Stream,2,[])';

%Mapping the input data to QPSK symbols grey encoded
%  0 0 -> -1-1i
%  0 1 -> -1+1i
%  1 0 ->  1-1i
%  1 1 ->  1+1i
QPSK_map = [-1-1i, -1+1i, 1-1i, 1+1i];
%  0 0 -> -1-1i
%  0 1 -> -1+1i
%  1 0 ->  1+1i
%  1 1 ->  1-1i
QPSK_map_encode2 = [-1-1i, -1+1i,1+1i, 1-1i];


%The bi2de function is used to convert the binary data to decimal values,
%which are then used as indices to look up the corresponding QPSK symbol in the mapping
table
QPSK_data = QPSK_map(bi2de(QPSK_reshaped_binary_data,'left-msb')+1);
QPSK_data_encode2 = QPSK_map_encode2(bi2de(QPSK_reshaped_binary_data,'left-msb')+1);

%Channel (AWGN)
QPSK_channelNoise_real = zeros(length(Bit_Stream)/2,SNR_max_value); %Matrix to store the
real noise in each channel
QPSK_channelNoise_complex = zeros(length(Bit_Stream)/2,SNR_max_value); %Matrix to store the
complex noise in each channel

QPSK_No = zeros(1,SNR_max_value + 1);

%Calculating No of the channel for different SNR
for SNR_dB = 1 : (SNR_max_value + 1)
    %Generating different noise vector for different channels with different SNR
    QPSK_No(SNR_dB) = QPSK_Eb/10.^((SNR_dB-1)/10);

    noise_I = randn(1,length(Bit_Stream)/2); %[todo] get it out of the for loop
    QPSK_channelNoise_real(:,SNR_dB) = noise_I * sqrt (QPSK_No(SNR_dB)/2); %scale the
noise.

    noise_Q = randn(1,length(Bit_Stream)/2); %[todo] get it out of the for loop
    QPSK_channelNoise_complex(:,SNR_dB) = noise_Q * sqrt (QPSK_No(SNR_dB)/2); %scale the
noise.
end

%Demapper
QPSK_demappedBits = zeros(1,length(Bit_Stream));   %Vector to store the demapped stream
QPSK_demappedBits_encode2 = zeros(1,length(Bit_Stream));   %Vector to store the demapped
stream
```

```
QPSK_recieved_Bits = zeros((Bits_Number/2),2);
QPSK_recieved_Bits_encode2 = zeros((Bits_Number/2),2);


for SNR_dB = 1 : (SNR_max_value + 1)

    %Recieved Data
    QPSK_DataRx = (real(QPSK_data)+ QPSK_channelNoise_real(:,SNR_dB)') ... %[todo] get this
out of the loop
    + 1i *(imag(QPSK_data)+ QPSK_channelNoise_complex(:,SNR_dB)');

    QPSK_DataRx_encode2 = (real(QPSK_data_encode2)+ QPSK_channelNoise_real(:,SNR_dB)')
...%[todo] get this out of the loop
    + 1i *(imag(QPSK_data_encode2)+ QPSK_channelNoise_complex(:,SNR_dB)');

    %Demapping the recieved symbol stream for each channel
    %Grey encoded
    for counter = 1 : Bits_Number/2
        if(real(QPSK_DataRx(counter)) > 0)
            QPSK_recieved_Bits(counter,1) = 1;
        else
            QPSK_recieved_Bits(counter,1) = 0;
        end

        if(imag(QPSK_DataRx(counter)) > 0)
            QPSK_recieved_Bits(counter,2) = 1;
        else
            QPSK_recieved_Bits(counter,2) = 0;
        end
    end
    QPSK_demappedBits = reshape(QPSK_recieved_Bits',1,[]);

    %Demapping the recieved symbol stream for each channel
    %second encoding method
    for counter = 1 : Bits_Number/2
        if(imag(QPSK_DataRx_encode2(counter)) > 0)
            if(real(QPSK_DataRx_encode2(counter)) > 0)
                QPSK_recieved_Bits_encode2(counter,1) = 1;
                QPSK_recieved_Bits_encode2(counter,2) = 0;
            else
                QPSK_recieved_Bits_encode2(counter,1) = 0;
                QPSK_recieved_Bits_encode2(counter,2) = 1;
            end
        else
            if(real(QPSK_DataRx_encode2(counter)) > 0)
                QPSK_recieved_Bits_encode2(counter,1) = 1;
                QPSK_recieved_Bits_encode2(counter,2) = 1;
            else
                QPSK_recieved_Bits_encode2(counter,1) = 0;
                QPSK_recieved_Bits_encode2(counter,2) = 0;
            end
        end
    end
    QPSK_demappedBits_encode2 = reshape(QPSK_recieved_Bits_encode2',1,[]);

    [N_BER_QPSK,QPSK_BER(SNR_dB)] = symerr(QPSK_demappedBits,Bit_Stream); %Calculated BER
    [N_BER_QPSK_2,QPSK_BER_encode2(SNR_dB)] = symerr(QPSK_demappedBits_encode2,Bit_Stream);
%Calculated BER
    QPSK_BER_Theo(SNR_dB)= 0.5 * erfc(sqrt(1/QPSK_No(SNR_dB))); %Theoritical BER
end

%Plotting constillation of QPSK
figure(1)
subplot(2,2,2,'LineWidth',3)
plot(real(QPSK_DataRx),imag(QPSK_DataRx),'b*',real(1),imag(1i),'k.',real(-1),imag(-1i),'k.'
...
    ,real(1),imag(-1i),'k.',real(-1),imag(1i),'k.');
title('QPSK Modulation for SNR = 10')

%Plotting BER
figure(2)
subplot(2,2,2,'LineWidth',3)
EbN0_dB = 1:1:(SNR_max_value + 1) ;
%Plotting theoritical BER
semilogy((EbN0_dB - 1),QPSK_BER_Theo,'--')
%Plotting calculated BER
```

```
hold on
semilogy((EbN0_dB - 1),QPSK_BER,'-')
grid on
ylabel('BER')
xlabel('E_b/N_0')
title('Bit Error Rate for QPSK')
legend('Theoretical','Calculated');

figure (4)
EbN0_dB = 1:1:(SNR_max_value + 1) ;
%Plotting theoritical BER
semilogy((EbN0_dB - 1),QPSK_BER_encode2,'-x')
%Plotting calculated BER
hold on
semilogy((EbN0_dB - 1),QPSK_BER,'-o')
grid on
ylabel('BER')
xlabel('E_b/N_0')
title('Bit Error Rate for QPSK')
legend('2nd encode method','grey encoded');

%% 8PSK
%initializations needed for Simulation of BPSK
M = 8;                  %Numbers of symboles
log2M = log2( M );      %Numbers of bit that represnts 8PSK
M8PSK_Eb = 1/3;
M8PSK_BER_Theo = zeros(1,SNR_max_value);             %Vector to store the theortical BER
for different SNR channel
M8PSK_BER = zeros(1,SNR_max_value);                  %Vector to store the calculated BER
for different SNR channel

%Mapper
%Grouping the binary data into groups of 3 bits
M8PSK_reshaped_binary_data = reshape(Bit_Stream,log2M,[])';

%Mapping the input data to 8PSK symbols grey encoded
%  1. 0 0 0-> theta = 0
%  2. 0 0 1-> theta = 45
%  3. 0 1 1-> theta = 90
%  4. 0 1 0-> theta = 135
%  5. 1 1 0-> theta = 180
%  6. 1 1 1-> theta = 225
%  7. 1 0 1-> theta = 270
%  8. 1 0 0->  theta = 315
% Each symbol is represented by S(i) = 1* exp(itheta)
M8PSK_mapper = [[0,0,0];[0,0,1];[0,1,1];[0,1,0];[1,1,0];[1,1,1];[1,0,1];[1,0,0]];

M8PSK_data = zeros(length(M8PSK_reshaped_binary_data),1); %Vector to store the complex
value of symbols of 8PSK

%Loop over each row in the reshaped data
for M8PSK_reshaped_binary_data_row = 1 : length(M8PSK_reshaped_binary_data)
    %Loop over rows of 8PSK mapper to map to the right angle
    for M8PSK_mapper_row = 1 : length(M8PSK_mapper)
        %Check if the row of the reshaped data equals to the mapper row
        if isequal(M8PSK_reshaped_binary_data(M8PSK_reshaped_binary_data_row,:)...
                ,M8PSK_mapper(M8PSK_mapper_row,:))
            %Then assign the complex value correspondes to the corresponding angle
            %Calculate the corresponding angle
            M8PSK_Theta = (M8PSK_mapper_row - 1) * (2*pi / M);
            %Store the complex value that represnets the symbol
            M8PSK_data(M8PSK_reshaped_binary_data_row) = exp(1i * M8PSK_Theta);
            break; %Break on matching the rows
        end
    end
end

%Channel (AWGN)
M8PSK_channelNoise_real = zeros(size(M8PSK_data,1),SNR_max_value); %Matrix to store the
real noise in each channel
M8PSK_channelNoise_complex = zeros(size(M8PSK_data,1),SNR_max_value); %Matrix to store the
complex noise in each channel

M8PSK_No = zeros(1,SNR_max_value + 1);

%Calculating No of the channel for different SNR
```

```
for SNR_dB = 1 : (SNR_max_value + 1)
    %Generating different noise vector for different channels with different SNR
    M8PSK_No(SNR_dB) = M8PSK_Eb/10.^((SNR_dB-1)/10);

    noise_I = randn(1,size(M8PSK_data,1));
    M8PSK_channelNoise_real(:,SNR_dB) = noise_I * sqrt (M8PSK_No(SNR_dB)/2); %scale the
noise.

    noise_Q = randn(1,size(M8PSK_data,1));
    M8PSK_channelNoise_complex(:,SNR_dB) = noise_Q * sqrt (M8PSK_No(SNR_dB)/2); %scale the
noise.
end
%Demapper
M8PSK_demappedBits = zeros(1,length(Bit_Stream));   %Vector to store the demapped stream of
bits
M8PSK_recieved_Bits = zeros((Bits_Number/3),3);
for SNR_dB = 1 : (SNR_max_value + 1)
    %Recieved Data
    M8PSK_DataRx = ((real(M8PSK_data)+ M8PSK_channelNoise_real(:,SNR_dB)) ...
    + 1i *(imag(M8PSK_data)+ M8PSK_channelNoise_complex(:,SNR_dB)));
    %Demapping the recieved symbol stream for each channel
        %Demapping by computing the angle of the recieved bit and decide the
        %corresponding symbol based on the decision reagin.
        for counter = 1 : size(M8PSK_DataRx,1) %Loop on each symbol of the recieved data.
            Rx_symbol_angle = angle(M8PSK_DataRx(counter)); %Calculate the angle of the
symbol in radian
            %Return the angle ot the positve value if it's negative
            if(Rx_symbol_angle < 0)
                Rx_symbol_angle = Rx_symbol_angle + 2*pi;
            end
            %Compare the angle of the symbol with the angles of the descion regions
            %if 22.5°>= angle || 337.5°<= angle
            if((Rx_symbol_angle <= pi/8) || (Rx_symbol_angle >= 15* pi/8))
                %which means 0 0 0 case
                M8PSK_recieved_Bits(counter, :) = M8PSK_mapper (1,:);
            else
                for bounder = 1 : 2 : 13
                    if((Rx_symbol_angle > bounder * pi/8)&&(Rx_symbol_angle <= (bounder +
2) * pi/8))
                        M8PSK_recieved_Bits(counter, :) = M8PSK_mapper (((bounder + 1)/2) +
1 ,:);
                        break;
                    end
                end
            end
        end
    %Reshape the recieved bits to one vector bit stream
    M8PSK_demappedBits = reshape(M8PSK_recieved_Bits',1,[]);

    [N_BER_M8PSK , M8PSK_BER(SNR_dB)] = symerr(M8PSK_demappedBits,Bit_Stream); %Calculated
BER
    M8PSK_BER_Theo(SNR_dB)= erfc(sqrt(1/M8PSK_No(SNR_dB)) * sin(pi/8))/3; %Theoritical BER
end

%Plotting constillation of 8PSK
figure(1)
subplot(2,2,3,'LineWidth',3)
plot(real(M8PSK_DataRx),imag(M8PSK_DataRx),'g*')
hold on
% Plotting the symbols on the constalation
M8PSK_sPoints = zeros(1,M);
for counter = 1:M
    Theta = (counter - 1) * (2*pi / M);
    M8PSK_sPoints(counter) = exp(1i * Theta);
    plot(real(M8PSK_sPoints(counter)),imag(M8PSK_sPoints(counter)),'k.');
    hold on
end
hold off
title('8PSK Modulation for SNR = 10')

%Plotting BER
figure(2)
subplot(2,2,3,'LineWidth',3)
EbN0_dB = 1:1:(SNR_max_value + 1) ;
%Plotting theoritical BER
semilogy((EbN0_dB - 1),M8PSK_BER_Theo,'--')
```

```
%Plotting calculated BER
hold on
semilogy((EbN0_dB - 1),M8PSK_BER,'-')
grid on
ylabel('BER')
xlabel('E_b/N_0')
title('Bit Error Rate for 8PSK')
legend('Theoretical','Calculated');

%% 16QAM
%initializations needed for Simulation of 16QAM
QAM16_Eb = 2.5;
QAM16_BER_Theo = zeros(1,SNR_max_value);            %Vector to store the theortical BER
for different SNR channel
QAM16_BER = zeros(1,SNR_max_value);                 %Vector to store the calculated BER
for different SNR channel

%Mapper
%Grouping the binary data into groups of 4 bits
QAM16_reshaped_binary_data = reshape(Bit_Stream,4,[])';

%Mapping every four bits in one symbol coded in grey code
%Mapping table
QAM16_map = [-3-3i, -3-1i, -3+3i, -3+1i,... % 0000 -> -3 -3i | 0001 -> -3 - 1i | 0010 -> -3
+ 3i | 0011 -> -3 + 1i
             -1-3i, -1-1i, -1+3i, -1+1i,... % 0100 -> -1 -3i | 0101 -> -1 - 1i | 0110 -> -1
+ 3i | 0111 -> -1 + 1i
              3-3i,  3-1i,  3+3i,  3+1i,... % 1000 ->  3 -3i | 1001 ->  3 - 1i | 1010 ->  3
+ 3i | 1011 ->  3 + 1i
              1-3i,  1-1i,  1+3i,  1+1i];   % 1100 ->  1 -3i | 1101 ->  1 - 1i | 1110 ->  1
+ 3i | 1111 ->  1 + 1i
%Map the bits to the symbol
QAM16_data = QAM16_map(bi2de(QAM16_reshaped_binary_data, 'left-msb') + 1);

%Channel (AWGN)
QAM16_channelNoise_real = zeros(length(QAM16_data),SNR_max_value); %Matrix to store the
real noise in each channel
QAM16_channelNoise_complex = zeros(length(QAM16_data),SNR_max_value); %Matrix to store the
complex noise in each channel

%Calculating No of the channel for different SNR
QAM16_No = zeros(1,SNR_max_value + 1);

for SNR_dB = 1 : (SNR_max_value + 1)
    %Generating different noise vector for different channels with different SNR
    QAM16_No(SNR_dB) = QAM16_Eb./10.^((SNR_dB-1)/10);

    noise_I = randn(1,length(QAM16_data));
    QAM16_channelNoise_real(:,SNR_dB) = noise_I * sqrt (QAM16_No(SNR_dB)/2); %scale the
noise.

    noise_Q = randn(1,length(QAM16_data));
    QAM16_channelNoise_complex(:,SNR_dB) = noise_Q * sqrt (QAM16_No(SNR_dB)/2); %scale the
noise.
end
%Demapper
QAM16_demappedBits = zeros(1,length(Bit_Stream));   %Vector to store the demapped bit
stream
QAM16_recieved_Bits = zeros(length(QAM16_data),4);

for SNR_dB = 1 : (SNR_max_value + 1)

    %Recieved Data
    QAM16_DataRx = (real(QAM16_data)+ QAM16_channelNoise_real(:,SNR_dB)') ...
    + 1i *(imag(QAM16_data)+ QAM16_channelNoise_complex(:,SNR_dB)');

    %Demapping the recieved symbol stream for each channel
    for counter = 1 : length(QAM16_DataRx)
        %Assigning the real part (b0b1xx)
        if(real(QAM16_DataRx(1,counter)) > 2)      %Symbol = 10xx
            QAM16_recieved_Bits(counter,1) = 1;
            QAM16_recieved_Bits(counter,2) = 0;
        elseif(real(QAM16_DataRx(counter)) > 0)    %Symbol = 11xx
            QAM16_recieved_Bits(counter,1) = 1;
            QAM16_recieved_Bits(counter,2) = 1;
        elseif(real(QAM16_DataRx(counter)) > -2)   %Symbol = 01xx
```

```
            QAM16_recieved_Bits(counter,1) = 0;
            QAM16_recieved_Bits(counter,2) = 1;
        else                                        %Symbol = 00xx
            QAM16_recieved_Bits(counter,1) = 0;
            QAM16_recieved_Bits(counter,2) = 0;
        end
        %Assigning the complex part (xxb2b3)
        if(imag(QAM16_DataRx(1,counter)) > 2)       %Symbol = xx10
            QAM16_recieved_Bits(counter,3) = 1;
            QAM16_recieved_Bits(counter,4) = 0;
        elseif(imag(QAM16_DataRx(counter)) > 0)     %Symbol = xx11
            QAM16_recieved_Bits(counter,3) = 1;
            QAM16_recieved_Bits(counter,4) = 1;
        elseif(imag(QAM16_DataRx(counter)) > -2)    %Symbol = xx01
            QAM16_recieved_Bits(counter,3) = 0;
            QAM16_recieved_Bits(counter,4) = 1;
        else                                        %Symbol = xx00
            QAM16_recieved_Bits(counter,3) = 0;
            QAM16_recieved_Bits(counter,4) = 0;
        end
    end
    %Reshape the recieved bits to  vector
    QAM16_demappedBits = reshape(QAM16_recieved_Bits',1,[]);

    [N_BER_QAM16,QAM16_BER(SNR_dB)] = symerr(QAM16_demappedBits,Bit_Stream); %Calculated
BER
    QAM16_BER_Theo(SNR_dB)=  3/8 * erfc(sqrt(1./(1.*QAM16_No(SNR_dB)))); %Theoritical BER
end

%Plotting constillation of 16QAM
figure(1)
subplot(2,2,4,'LineWidth',3)
plot(real(QAM16_DataRx),imag(QAM16_DataRx),'m*')
hold on
%Plotting the symbols on the constalation
for row = -3: 2 : 3
    for col = -3: 2 : 3
        plot(real(row),imag(1i* col),'k.');
        hold on
    end
end
hold off
title('16QAM Modulation for SNR = 10')

%Plotting BER
figure(2)
subplot(2,2,4,'LineWidth',3)
EbN0_dB = 1:1:(SNR_max_value + 1) ;
%Plotting theoritical BER
semilogy((EbN0_dB - 1),QAM16_BER_Theo,'--')
%Plotting calculated BER
hold on
semilogy((EbN0_dB - 1),QAM16_BER,'-')
grid on
ylabel('BER')
xlabel('E_b/N_0')
title('Bit Error Rate for 16QAM')
legend('Theoretical','Calculated');

%% Plotting all the BER, theoritcal and calculated, on the same graph
figure(3)
EbN0_dB = 1:1:(SNR_max_value + 1) ;
%Plotting theoritical BPSK_BER
semilogy((EbN0_dB - 1),BPSK_BER_Theo,'--')
hold on
%Plotting theoritical QPSK_BER
semilogy((EbN0_dB - 1),QPSK_BER_Theo,'--')
hold on
%Plotting theoritical 8PSK_BER
semilogy((EbN0_dB - 1),M8PSK_BER_Theo,'--')
hold on
%Plotting theoritical 16QAM_BER
semilogy((EbN0_dB - 1),QAM16_BER_Theo,'--')
hold on
%Plotting theoritical BFSK_BER
%---- Plot the BFSK BER here -----
```

```
%hold on
%Plotting calculated BPSK_BER
semilogy((EbN0_dB - 1),BPSK_BER,'-o')
hold on
%Plotting calculated QPSK_BER
semilogy((EbN0_dB - 1),QPSK_BER,'-*')
hold on
%Plotting calculated 8PSK_BER
semilogy((EbN0_dB - 1),M8PSK_BER,'-x')
hold on
%Plotting calculated 16QAM_BER
semilogy((EbN0_dB - 1),QAM16_BER,'-+')
hold off
grid on
legend('Theoretical BSPK BER','Theoretical QPSK BER','Theoretical 8PSK BER',...
    'Theoretical QAM16 BER','BSPK BER','QPSK BER','8PSK BER','QAM16 BER');
xlabel('E_{b}/N_{o}');
ylabel('BER');
title("BER_{theoritical and calculated} of some modulation techniques");
```

BFSK Code

```
%{
* FILE DESCRIPTION
* File: BFSK.m
* Digital communication project-assignment
* Description: Simulation of BFSK modulation technique
* Author:Omar Muhammad Mustafa, Omar Muhammad Tolba
* Date: 6th May 2023
%}
%% clear all WorkSpace Variables and Command Window
clc;
clear ;
close all;
%% initialization
Bits_Number = 1.2e5+1;                                % number of bits to be generated
SNR_max_value = 10;
N_Samples = 10;     %  Number of samples
N_Realization = 10;  % Number of realization

%% Generating the data bits
Bits_Sent = randi([0 1],N_Realization,Bits_Number);
Bit_Stream = Bits_Sent(1,:);

%% Initialzie Parameters
BFSK_Eb = ((1 +1)/ 2) / 1;                       %Calculating Eb of BFSK modulation
BFSK_BER_Theo = zeros(1,SNR_max_value);          %Vector to store the theortical BER for
different SNR channel
BFSK_BER = zeros(1,SNR_max_value);               %Vector to store the calculated BER for
different SNR channel

%% Mapper f1 and f2 need to be orthogonal at each other
BFSK_symbolStream = zeros(1,Bits_Number);
for i = 1 : Bits_Number
    if(Bit_Stream(i) == 1) % Map to f1 (1,0i)
        BFSK_symbolStream(i)=1;
    else
        BFSK_symbolStream(i)=1i;% Map to f1 (0,i)
    end
end
%% Channel (AWGN)
BFSK_channelNoise_real = zeros(length(Bit_Stream),SNR_max_value); %Matrix to store the real
noise in each channel
BFSK_channelNoise_complex = zeros(length(Bit_Stream),SNR_max_value); %Matrix to store the
complex noise in each channel

BFSK_No = zeros(1,SNR_max_value + 1);

noise_I = randn(1,length(Bit_Stream)); % in-phase noise
noise_Q = randn(1,length(Bit_Stream)); % quadrature phase noise


%Calculating No of the channel for different SNR
for SNR_dB = 1 : (SNR_max_value + 1)
    %Generating different noise vector for different channels with different SNR
```

```
    BFSK_No(SNR_dB) = BFSK_Eb/10.^((SNR_dB-1)/10);
    BFSK_channelNoise_real(:,SNR_dB) = noise_I * sqrt (BFSK_No(SNR_dB)/2); %scale the
noise. [TODO](check variance)) ; %scale the noise.
    BFSK_channelNoise_complex(:,SNR_dB) = noise_Q * sqrt (BFSK_No(SNR_dB)/2); %scale the
noise. [TODO](check variance)) ; %scale the noise.
end

%% DeMapper
BFSK_ReceivedBits = zeros(1,length(Bit_Stream));   %Vector to store the demapped stream
BFSK_DemappedBits = zeros(1,length(Bit_Stream));   %Vector to store the demapped stream


for SNR_dB = 1 : (SNR_max_value + 1)
    % Recieved Data for the last channel noise [10]
    BFSK_DataRx = (real(BFSK_symbolStream) + BFSK_channelNoise_real(:,SNR_dB)')...
        +1i * (imag(BFSK_symbolStream) +BFSK_channelNoise_complex(:,SNR_dB)') ;

    % Demapping the recieved symbol stream for each channel
    for counter = 1 : Bits_Number
        if((angle(BFSK_DataRx(1,counter)) <= pi/4) && (angle(BFSK_DataRx(1,counter)) >= -
(3*pi/4) ))
            BFSK_ReceivedBits(1,counter) = 1+0i;
            BFSK_DemappedBits(1,counter)= 1;
        else
            BFSK_ReceivedBits(1,counter) = 0+1i;
            BFSK_DemappedBits(1,counter) = 0;
        end
    end
    [N_BER_BPSK,BFSK_BER(SNR_dB)] = symerr(BFSK_DemappedBits,Bit_Stream); %Calculated BER
    BFSK_BER_Theo(SNR_dB)= 0.5 * erfc(sqrt(BFSK_Eb/(2*BFSK_No(SNR_dB)))); %Theoritical BER
end

% Plotting constillation of BPSK
figure(1)
subplot(1,1,1,'LineWidth',3)
plot(real(BFSK_DataRx),imag(BFSK_DataRx),'r*',real(sqrt(BFSK_Eb)),imag(0),'k.',real(0),imag
(1i*sqrt(BFSK_Eb)),'k.');
title('BFSK Modulation for SNR = 10')

%Plotting BER
figure(2)
subplot(1,1,1,'LineWidth',3)
EbN0_dB = 1:1:(SNR_max_value + 1) ;
%Plotting theoritical BER
semilogy((EbN0_dB - 1),BFSK_BER_Theo,'--')
%Plotting calculated BER
hold on
semilogy((EbN0_dB - 1),BFSK_BER,'-')
grid on
ylabel('BER')
xlabel('E_b/N_0')
title('Bit Error Rate for BPSK')
legend('Theoretical','Calculated');

%% PSD
Tb = 1e-3;
f1 = 1e3;
f2 = f1 + (1/Tb);
t =0:(1/N_Samples)*Tb:Tb*(1-1/N_Samples); % sampling interval
s1bb = repelem(sqrt((2*BFSK_Eb)/Tb),N_Samples);  %Mapping zero
s2bb = sqrt((2*BFSK_Eb)/Tb) * exp(2*1i*pi*t*(1/Tb)); % Mapping 1
BFSK_Mapped_Realization1 = zeros(1,N_Samples*Bits_Number);
BFSK_Mapped_Ensemble=zeros(N_Realization,N_Samples*Bits_Number);
mapped_ensamble_delayed = [];

for realiz_counter = 1 : N_Realization
    for bit_count = 1 : Bits_Number
        if(Bits_Sent(realiz_counter,bit_count)== 0)
            BFSK_Mapped_Ensemble(realiz_counter,(bit_count-1)*N_Samples + 1 :
N_Samples*bit_count) = s1bb;
        else
            BFSK_Mapped_Ensemble(realiz_counter,(bit_count-1)*N_Samples + 1 :
N_Samples*bit_count) = s2bb;
        end
    end
    %%% adding random delay to the ensamble %%%
```

```
    delay = randi([1,N_Samples]) + 1; % delay = 1 means there is no delay, hence we add a
+1 offset
    mapped_ensamble_delayed = vertcat(mapped_ensamble_delayed,
BFSK_Mapped_Ensemble(realiz_counter, delay : end - N_Samples + (delay -1) ));

end
%% PSD calculation
for column = 1: size(mapped_ensamble_delayed,2)
    R_tau(column) =(1/N_Realization) * sum(mapped_ensamble_delayed(:, 1) .*
(mapped_ensamble_delayed(:, column)));
end

%% plottion PSD

%%Bandwidth
L = length(R_tau);     % Length of signal
n = 2^nextpow2(L);
T = 0.1*Tb;              % Sampling period
Fs = 1/T;              % Sampling frequency
t = (0:L-1)*T;         % Time vector
%Compute the Fourier transform of the signal.
PSD = fft(R_tau,n);
PSD = fftshift(PSD);
f = Fs * (-n/2:n/2-1)/n;
%Compute the two-sided spectrum P2
P2 = abs(PSD/n).^2;
Smoothed_PSD = smooth(P2); %Smooth the curve from the noise, to be plotted
figure (4)
subplot(1,1,1,'LineWidth',5)
plot(f,Smoothed_PSD)
ylim([0 1e3])
title("PSD")
xlabel("f(Hz)")
ylabel("S(f)")
% End of File [BFSK.m]
```