Cairo University, Faculty of Engineering
Electronics and Electrical Communications Department (EECE)

ELC 3070 – Spring 2025
Digital Communications
Project #3

Submitted to
Dr. Mohammed Nafie
Dr. Mohammed Khairy
Eng. Mohammed Khaled
Team 24

Name	ID	Code	Role
عبد الرحمن احمد محمد عبد اللطيف	9220457	233	Simulation of BPSK & QPSK & 8-PSK & 16-QAM
شهاب الدين طارق فؤاد محمد	9220392	224	Simulation of BFSK
عمر احمد رجب بدير	9220513	244	Simulation of BPSK & QPSK & 8-PSK & 16-QAM

Table of Contents

Table of figures:

Introduction

This project investigates the performance of digital modulation schemes in a single-carrier communication system using MATLAB simulation. The studied modulations include BPSK, QPSK (Gray and non-Gray coded), 8PSK, and 16QAM. Each scheme is evaluated over an AWGN channel, and its Bit Error Rate (BER) is compared to theoretical expectations. The goal is to analyze how modulation type and constellation mapping affect reliability and error performance across varying $\frac{E_b}{N_o}$ values.

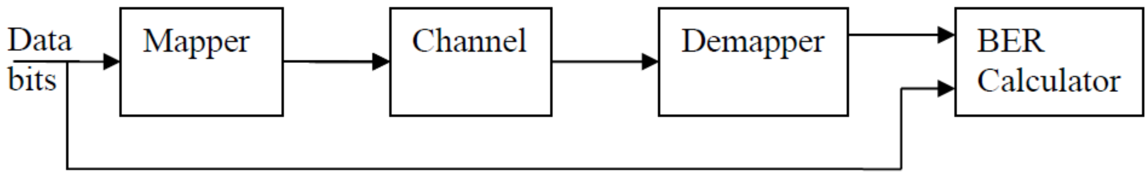

Figure 1: Single Carrier System

Simulated BER of all types:

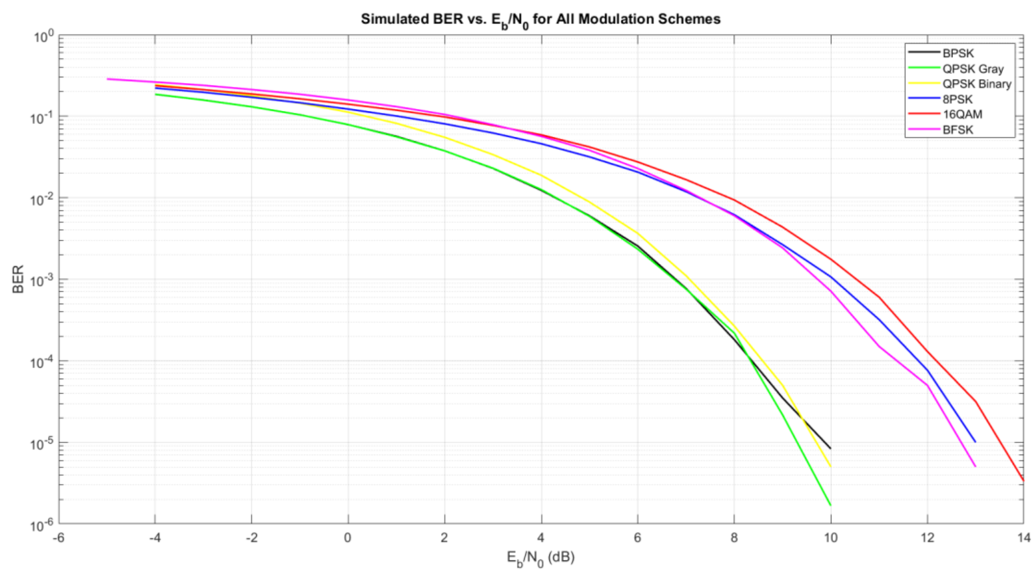

Figure 2: simulated BER of all types

Simulations and results

1) BPSK

Design:

Mapper:

The mapper for BPSK is simple we just convert binary data into modulated symbols suitable for transmission.
Each bit is represented by a phase shift:
0 is modulated into -1 and 1 is modulated into 1(180-degree shift between them).

Channel:

The modulated signal passes through AWGN (Additive White Gaussian Noise) channel.
The channel adds gaussian noise. Its amplitude has a normal distribution.

De-Mapper:

The de-mapper performs the inverse of the mapper, it converts received noisy symbols back into binary bits by comparing them to a threshold at 0.
For BPSK:
If the received symbol is > 0 then bit is 1
If the received symbol is < 0 then bit is 0

BER Analysis:

After demapping the received symbols back into bits, we calculated the accuracy of the transmission by comparing the detected bits after the demapper to the original transmitted bits. Any mismatches between them represent bit errors because of the noise or distortion in the channel.
To calculate it, we compute the Bit Error Rate (BER), which is defined as the ratio of the number of bit errors to the total number of transmitted bits.

Theoretical BER:

We calculated the theoretical BER to compare the results to that when the system does not face noise.

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

Average energy per symbol:

$$E_s = \frac{\sum energy\ of\ symbol}{num\ of\ symbols} = \frac{1^2 + (-1)^2}{2} = 1$$

Bit energy:

- $E_b = \frac{avg\ E_s}{num\ of\ bits} = \frac{1}{1} = 1$
- $N_o = \frac{E_b}{10^{\frac{snr}{10}}}$ $,SNR\ range = [-4:14]$

Results:

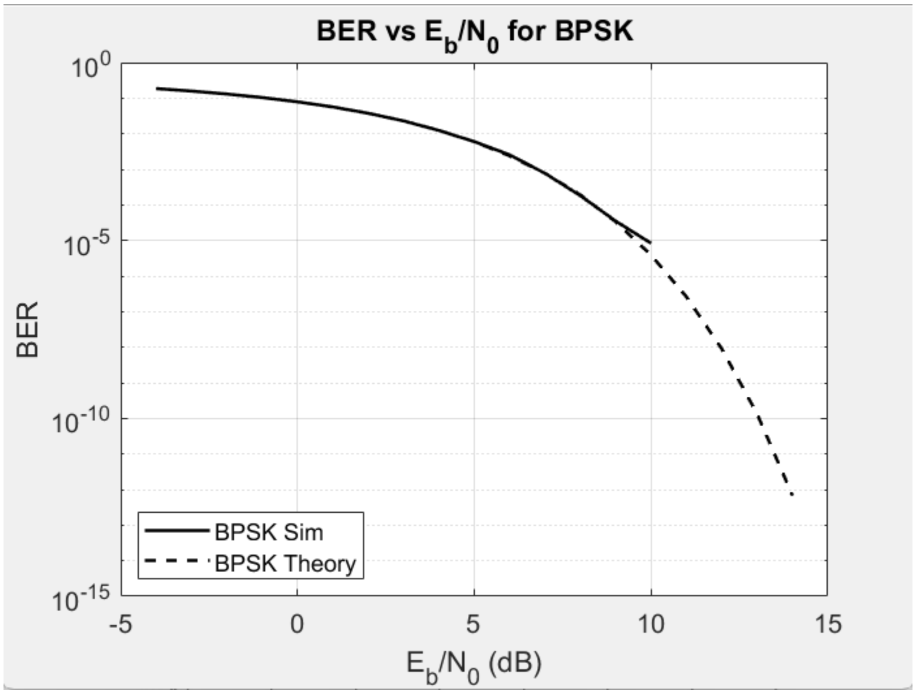

Figure 3: BPSK BER simulated and theoretical VS Eb/No

Comments:

First thing we notice that the simulation almost matches the theoretical indicating the accuracy of BPSK.
The second thing is the fact that the BER is decreasing significantly as we increasing the SNR means that the BPSK is not affected that much by noise.
The graph doesn't continue to 14 dB as with high SNR the error is almost zero among the number of bits and in a log-scale plot, log10(0) is undefined so it disappears.
This can be solved by increasing the number of bits, but it's not related to the project aim.

2) QPSK

Design:

Mapper:

Grey coded:

QPSK maps 2 bits per symbol by shifting the phase of the carrier signal to one of those values:
45°, 135°, 225°, or 315°
In grey coded QPSK we want the difference between each symbol(2-bits) and the nearest symbol (2-bits) to be just 1-bit to reduce the chances of errors in bits
The required Gray-coded QPSK mapper:
11 to 45° (1 + j)
01 to 135° (-1 + j)
00 to 225° (-1 - j)
10 to 315° (1 - j)

Binary coded:

Same as the grey coded but with a 2-bit change between most symbols
The required Binary-coded QPSK mapper:

10 to 45° (1 + j)	01 to 135° (-1 + j)
00 to 225° (-1 - j)	11 to 315° (1 - j)

Channel:

The modulated signal passes through AWGN (Additive White Gaussian Noise) channel.
The channel adds gaussian noise. Its amplitude has a normal distribution.

De-Mapper:

Finds which ideal constellation point the received symbol is closest to.
Maps that point back to the corresponding 2-bit symbol (using gray coding order/Binary coding).
The closest constellation point is found based on the real and imaginary parts of the signal.

BER Analysis:

Like before getting the BER we compare the demapped data with the original data and count the errors then divide them by the total number of the transmitted bits.

Theoretical BER:

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

Average energy per symbol:

$$E_s = \frac{\sum energy\ of\ symbol}{num\ of\ symbols} = \frac{4*(1+1)}{4} = 2$$

Bit energy:

- $E_b = \frac{avg\ E_s}{num\ of\ bits} = \frac{2}{2} = 1$
- $N_o = \frac{E_b}{10^{\frac{snr}{10}}}$ $,SNR\ range = [-4:14]$

Results:

Grey coded:

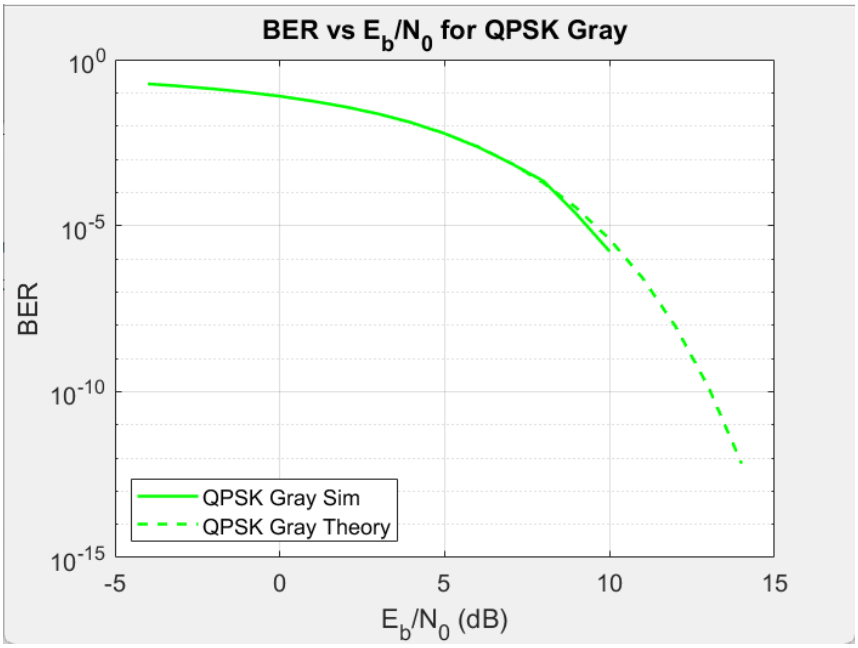

Figure 4:Gray coded QPSK BER simulated and theoretical VS Eb/No

Comments:

The BER from simulation almost matches the theoretical BER.
Like BPSK, the BER decreases with increasing SNR, indicating good performance at dealing with wanted signals.
The 1-bit change between each symbol makes less error rate among symbols.

Binary coded:

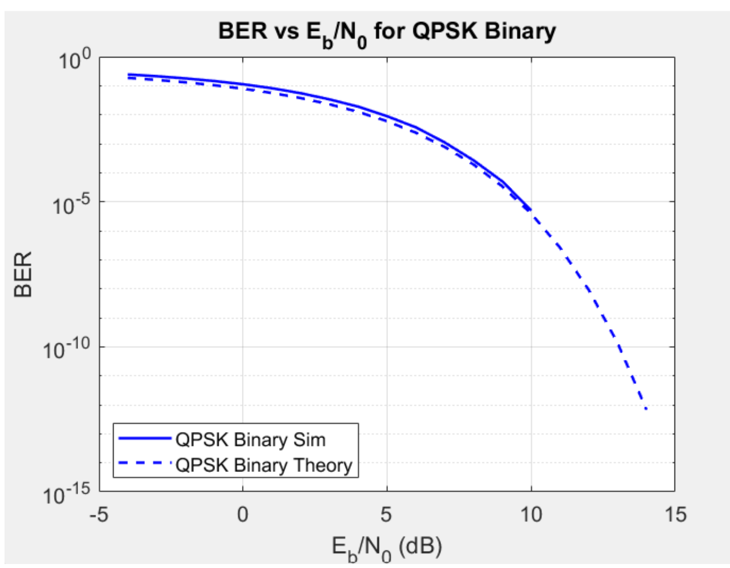

Figure 5: Binary coded QPSK BER simulated and theoretical VS Eb/No

Comments:

The simulated BER is higher than theoretical BER.
The multiple changes in symbol bits resulted in more bit errors.
And like all before the BER decreases with increasing SNR, indicating good performance at dealing with wanted signals.

Binary coded VS Grey coded:

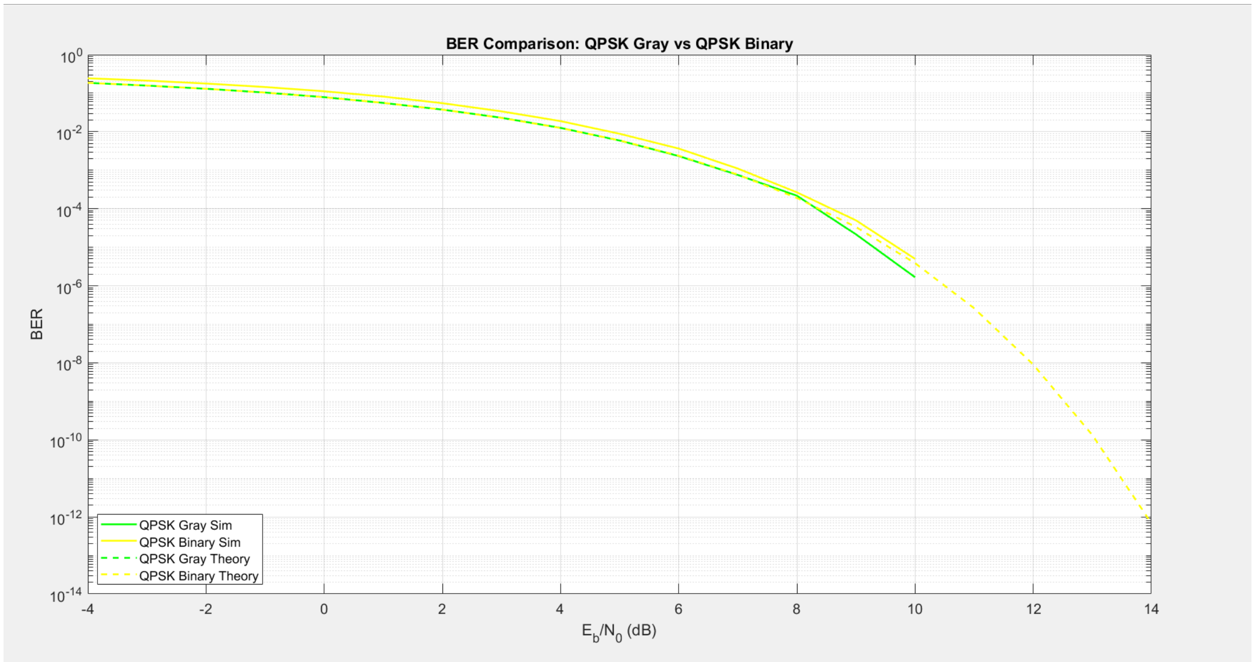

Figure 6: grey coded vs binary coded

Comments:

From the graphs it became clear that the grey coding resulted much less BER as one bit change between each symbol has a real effect on the errors.
Even though the binary coding is simpler to implement but its BER is bad.
As the SNR increases both approaches the theoretical BER.

3) 8PSK

Design:

Mapper:

The symbol consists of 3-bits. Each symbol gets mapped on a certain point on a circle with 45 degrees difference between each point (360/8).
Based on the required consultation map the mapper maps the symbol to:

- $000\ to\ 0° \ (1 + 0j)$
- $001\ to\ 45° \ (\sqrt{2}/2 + j\sqrt{2}/2)$
- $011\ to\ 90° \ (0 + j1)$
- $010\ to\ 135° \ (-\sqrt{2}/2 + j\sqrt{2}/2)$
- $110\ to\ 180° \ (-1 + 0j)$
- $111\ to\ 225° \ (-\sqrt{2}/2 - j\sqrt{2}/2)$
- $101\ to\ 270° \ (0 - j1)$
- $100\ to\ 315° \ (\sqrt{2}/2 - j\sqrt{2}/2)$

Channel:

Like before, The modulated signal passes through AWGN (Additive White Gaussian Noise) channel.
The channel adds gaussian noise. Its amplitude has a normal distribution.

De-Mapper:

The de-mapper receives the noisy signal and determines which of the 8 constellation points the received symbol is closest to (min distance). Once the closest point is found, it retrieves the corresponding 3-bit group that was originally sent.

BER Analysis:

Like before, we just compare the demapped data with the original and count the error.
Then the BER is the bit error count divided by the total transmitted bits.

Theoretical BER:

Theoretical BER = $\frac{1}{3} erfc(\sqrt{\frac{6*E_b}{N_o}} * \sin\left(\frac{\pi}{8}\right) * \frac{1}{\sqrt{2}}\ )$

Average energy per symbol:

$$E_s = \frac{\sum energy\ of\ symbol}{num\ of\ symbols} = \frac{8 * (1)^2}{8} = 1$$

Bit energy:

- $E_b = \frac{avg\ E_s}{num\ of\ bits} = \frac{1}{\log_2 8} = \frac{1}{3}$
- $N_o = \frac{E_b}{10^{\frac{snr}{10}}}\ \ , SNR\ range = [-4{:}\ 14]$

Results:

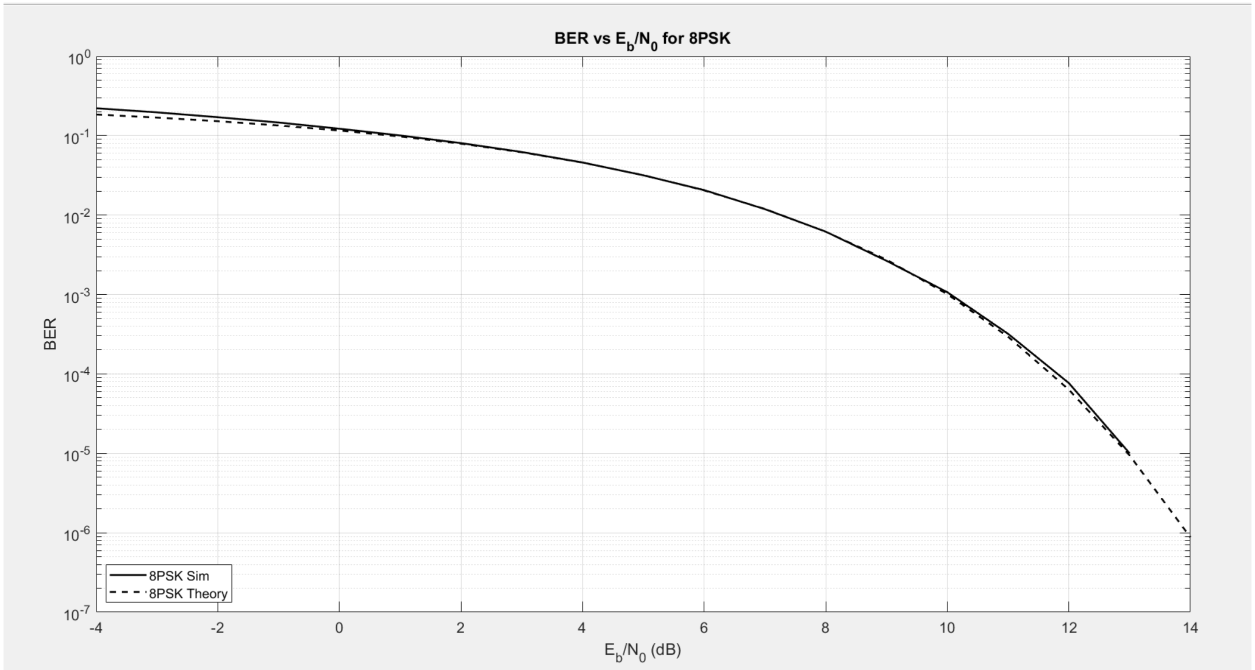

Figure 7: 8PSK BER simulated and theoretical VS Eb/No

Comments:

- At low SNR values (-5 to 2 dB), the simulated BER is higher than theoretical because noise can cause symbols to be misclassified into positions, leading to multiple bit errors.
- The theoretical BER assumes gray coding and mostly single-bit errors, which isn't accurate at low SNR. As SNR increases symbol decisions improve, and the simulated BER approaches the theoretical curve.
- The BER decreases as SNR increases.

4) 16QAM:

Design:

Mapper:

The 16QAM mapper takes groups of 4 bits and maps them to one of 16 unique symbols in the complex plane.
These symbols are arranged in a square grid.
The horizontal (I) and vertical (Q) axes represent different amplitudes.
Each constellation point is represented by 4 bits: b0b1b2b3
The constellation has 4 rows and 4 columns in a square grid with values at -3, -1, 1, 3.
The code is made to match the symbols on their points based on this constellation map

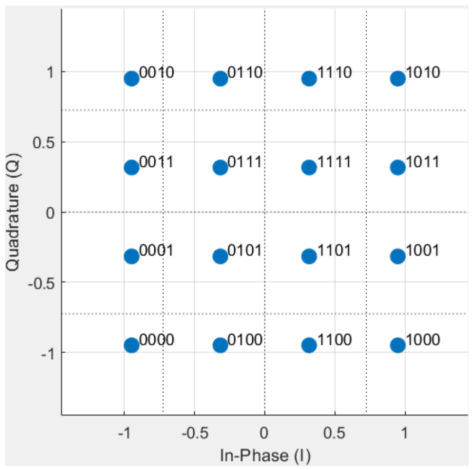

Figure 8: 16QAM constellation map

- The first two bits (b0b1) determine column (I value).
- The last two bits (b2b3) determine row (Q value).

Channel:

Like before, The modulated signal passes through AWGN (Additive White Gaussian Noise) channel.
The channel adds gaussian noise. Its amplitude has a normal distribution.

De-Mapper:

At the receiver, the noisy signal is compared against all 16 possible constellation points. The closest point (based on distance) is chosen as the received symbol. Its corresponding 4-bit binary value is then output.

BER Analysis:

Like before, we just compare the demapped data with the original and count the error.
Then the BER is the bit error count divided by the total transmitted bits.

Theoretical BER:

Theoretical BER = $\frac{3}{3} erfc(\sqrt{\frac{E_b}{2.5*N_o}})$

Average energy per symbol:

$$E_s = \frac{\sum energy\ of\ symbol}{num\ of\ symbols} = \frac{4*(\sqrt{2})^2 + 4*(3\sqrt{2})^2 + 8*(\sqrt{10})^2}{16} = 10$$

Bit energy:

- $E_b = \frac{avg\ E_s}{num\ of\ bits} = \frac{10}{\log_2 16} = \frac{10}{4} = 2.5$
- $N_o = \frac{E_b}{10^{\frac{snr}{10}}}$ $, SNR\ range - 4{:}\ 14]$

Results:

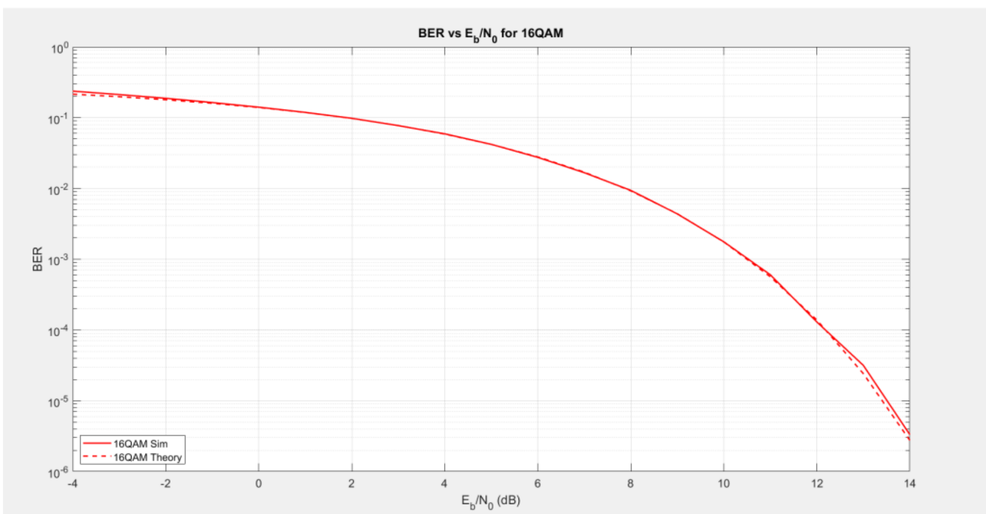

Figure 9: 16QAM BER simulated and theoretical VS Eb/No

Comments:

- Discrepancy at low SNR ($\frac{E_b}{N_o}$ < 2 dB): The simulated Bit Error Rate (BER) exceeds the theoretical prediction due to the increased likelihood of multiple bit errors caused by high noise levels, whereas the theory assumes primarily single-bit errors.
- Gray coding assumption: The theoretical BER relies on the assumption that Gray coding results in only one bit flip per symbol error. However, this assumption becomes invalid at low SNR, where multiple bits may be corrupted.
- Convergence at high SNR ($\frac{E_b}{N_o}$ > 4 dB): At higher SNR values, the simulated and theoretical BER curves align closely, supporting the validity of the single-bit error approximation in low-noise conditions.
- BER trend with $\frac{E_b}{N_o}$: As expected in QAM modulation schemes, the BER drops exponentially as $\frac{E_b}{N_o}$ increases.

Theoretical BPSK:

- the symbol set is: {±1, ±3}
- Average Energy per Symbol $(E_s) = \frac{\sum Simbol\ Energy}{Numbers\ of\ simbols} = \frac{2\times4\ +18\times4\ +\ 10\times8}{16} = 10$
- Bit Energy $(E_b) = \frac{Avg\ E_s}{Bits\ per\ simbol} = \frac{10}{log_2(16)} = 2.5$
- Theoretical BER $= \frac{3}{8}\ erfc\left(\sqrt{\frac{E_b}{2.5\times N_o}}\right)$
- $N_o = \frac{E_b}{10^{\frac{SNR_{dB}}{10}}}$, SNR range = [-4:14]

5) BFSK:

Design:

Mapping:

In the BFSK modulation scheme, each binary bit from the input data stream is mapped to a complex symbol representing one of two frequencies. If the binary bit is 0, it is mapped to a symbol with a phase angle of 0^o degree corresponding to the base frequency (f0). otherwise, it is mapped to a symbol with a phase angle of 90^o degree which is mapped to f1.

Channel:

The simulation introduces (AWGN) to a real-world channel condition. Noise power spectral density is determined based on the Signal-to-Noise Ratio (Eb/No) values specified for simulation.

De-Mapping:

After reception, the received signal is de-mapped to recover the original binary data. De-mapping involves analyzing the phase of each received symbol. If the phase angle falls within a certain range (-135^o to 45^o),
the de-mapped bit is considered "0"; otherwise, it's considered "1".
As shown in the signal's constellations:

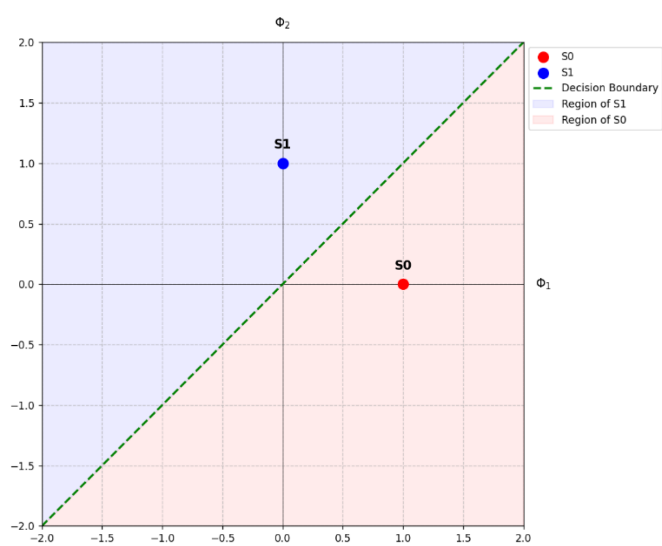

Figure 10: BFSK Signals Constellations

Theoretical BER:

Theoretical BER values are computed using mathematical formulas based on the AWGN channel model. These provide a reference for evaluating the performance of the BFSK modulation under different SNR conditions.

Theoretical BER of BFSK = $\frac{1}{2} erfc(\sqrt{\frac{E_b}{2*N_o}})$, where $E_b = 1$, N_o is varied by the SNR values [-5 ,15] dB

BER Analysis:

It is calculated by comparing the de-mapped binary data with the original input data. It quantifies the accuracy of data transmission, representing the ratio of incorrectly received bits to the total number of transmitted bits.

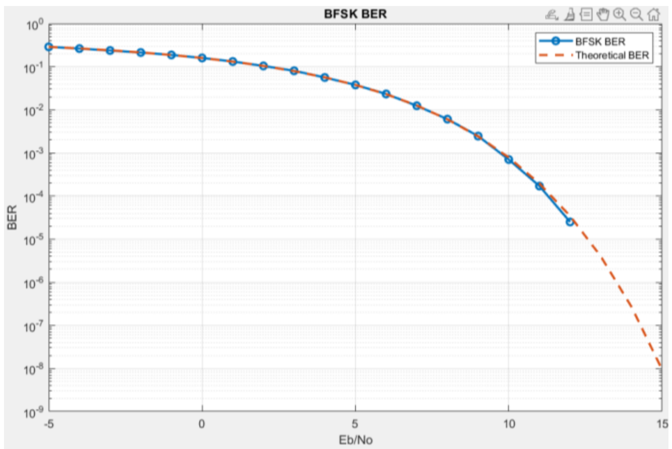

Figure 11: BER of BFSK

Comments:

BER of the simulated and theoretical are nearly equal and aligned.
the behavior of digital modulation schemes, As shown, the BER decreases as SNR increases.

Basis Functions of the Signal Set:

The BFSK Signals given are:

- $S1 = \sqrt{\frac{2Eb}{Tb}} \cos(2\pi f1t) \quad 0 < t < Tb$
- $S2 = \sqrt{\frac{2Eb}{Tb}} \cos(2\pi f2t) \quad 0 < t < Tb$

f_i is given by $\frac{i+nc}{Tb}$, let $f_1 = 0$, $f_2 = 1/ T_b$ to make them orthogonal and has frequency shift = $1/ T_b$.
therefor the basis function are two basis sinusoidal functions with two different frequencies f_1 & f_2, which will be mapped as follows:
"0" is mapped to f_1 (where $f_1 = 0$)
"1" is mapped to f_2 (where $f_2 = 1/ T_b$)

therefor the basis functions are:

- $\emptyset 1(t) = \sqrt{\frac{2}{Tb}}\ cos(2\pi f1t)$
- $\emptyset 2(t) = \sqrt{\frac{2}{Tb}}\ cos(2\pi f2t)$

Therefor the signal sets are:

- $S_1 = \sqrt{Eb} * \emptyset_1(t) \qquad 0 < t < T_b$
- $S_2 = \sqrt{Eb} * \emptyset_2(t) \qquad 0 < t < T_b$

Baseband Equivalent Signals:

Since the signals set are:

- $S1 = \sqrt{\frac{2Eb}{Tb}}\, cos(2\pi f1t)\ \ 0 < t < Tb$
- $S2 = \sqrt{\frac{2Eb}{Tb}}\, cos(2\pi f2t)\ \ 0 < t < Tb$

Let the $f1 = fo$ & $f2 = fo + \Delta f$ (the fo (carrier frequency) will be 0 and Δf will be $1/T_b$ as mentioned before)

Therefor the 1st equivalent signals are:

$$S1 = \sqrt{\frac{2Eb}{Tb}}\, cos(2\pi ft)\ \ 0 < t < Tb$$

The 2nd equivalent signal is**:**

- $S2 = \sqrt{\frac{2Eb}{Tb}}\, cos(2\pi(fo + \Delta f)t)$
- $= \sqrt{\frac{2Eb}{Tb}}\ [cos(2\pi\Delta ft)cos(2\pi fot) - sin(2\pi\Delta ft)sin(2\pi fot)]$
-

Therefore, the Equivalent Signals at the Baseband:

- $Sibb = Real\,\{Si\ \ e^{2\pi fot}\}$
- $S1bb = \sqrt{\frac{2Eb}{Tb}}\ + 0\,i$
- $S2bb = \sqrt{\frac{2Eb}{Tb}}\ [cos(2\pi\Delta ft) + jsin(2\pi\Delta ft)]$

Power Spectral Density of BFSK Signals:

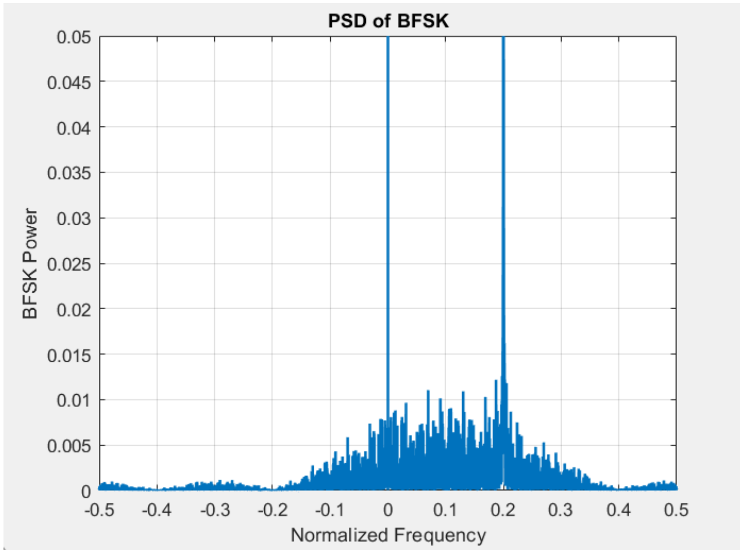

Figure 12: PSD of BFSK

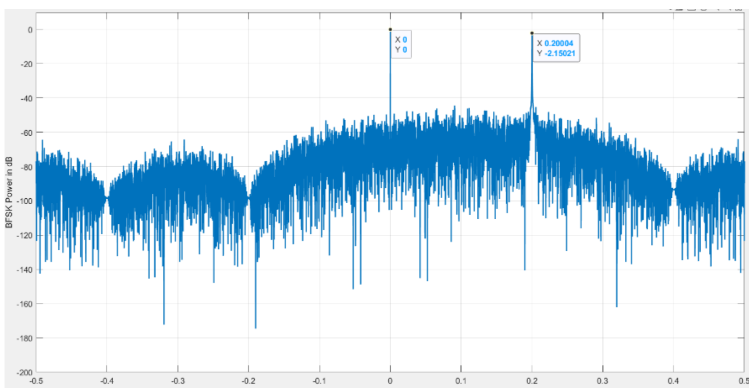

Figure 13: PSD of BSK in dB

Comments:

- The Power Spectral Density (PSD) of BFSK exhibits two distinct delta functions, one is the DC component positioned at zero frequency and the other at a frequency separation of (1/ Tb), where Tb is the bit duration (Tb =5 as defined in the code).

- The amplitude of the deltas depends on Eb and Tb, and the plot is normalized with the data length = 5000 bits (5 bits per sample * 1000 bits per realization)

- As shown the PSD of BFSK is narrow, focusing most of its energy around these two delta functions peaks.

Theoretical and simulated BER:

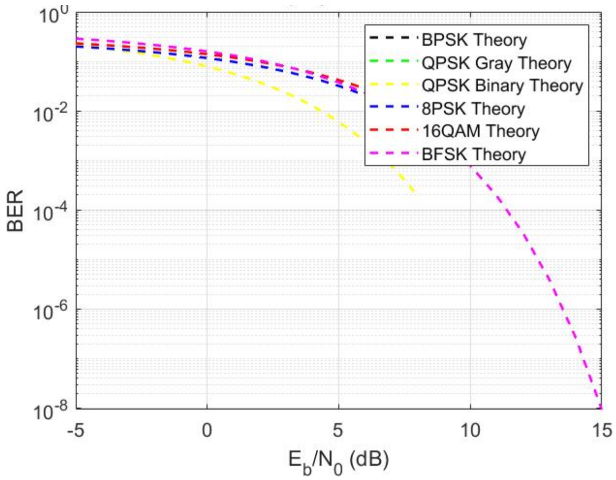

Figure 14: Theoretical BER for all types

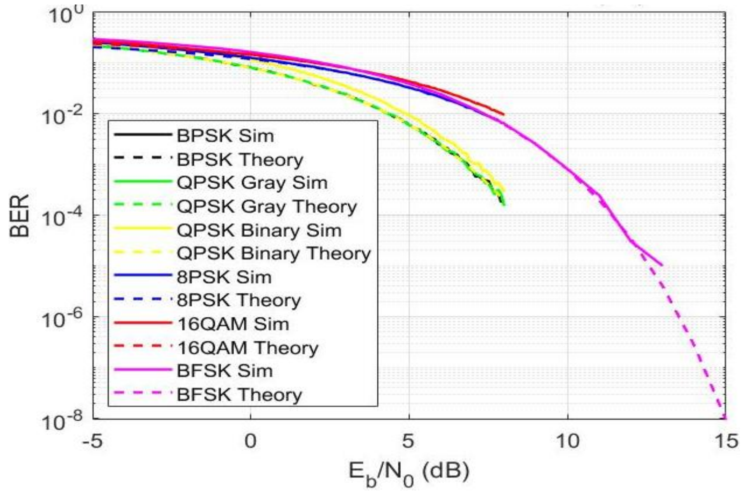

Figure 15: Theoretical & simulated BER for all types

Table 1: Comparison between all types

Modulation Type	BER Performance	Bandwidth Efficiency	Comments
BPSK	Very good (very low BER)	Low (1 bit/s/Hz)	Due to its simple two-phase encoding, it is highly resilient in noisy or long-range channels, but it transmits only 1 bit per symbol.
QPSK	Good (low BER)	Moderate (2 bits/s/Hz)	Gray coding effectively balances BER and data rate, doubling BPSK's throughput without a substantial rise in errors.
8-PSK	Moderate BER	High (3 bits/s/Hz)	Provides 3 bits per symbol for improved bandwidth efficiency, but its compact constellation results in a higher BER at lower SNRs.
16-QAM	Low (Higher BER)	Very High (4 bits/s/Hz)	Offers high spectral efficiency (4 bits per symbol), making it suitable for high-speed data transmission, but requires a strong SNR to preserve signal quality.
BFSK	High (low BER)	Low	It is straightforward, consumes little energy, and is resistant to noise, making it well-suited for low-cost systems despite inefficient bandwidth utilization.

- BPSK and QPSK demonstrate the most favorable BER performance, with nearly identical results owing to their comparable error characteristics.
- BFSK delivers slightly lower performance than BPSK/QPSK but still achieves reliable error rates, particularly at low to mid-range SNR levels.
- 8PSK and 16-QAM exhibit increased BER at lower SNRs due to their greater modulation complexity.
- Higher-order modulation schemes like 16-QAM offer increased data throughput at the cost of reduced BER performance.
- The close match between simulated and theoretical BER curves for all modulation schemes validates the accuracy of the simulation models.
- BER consistently declines in an exponential manner as SNR increases, which aligns with typical behavior in digital modulation techniques.

Code:

```
% clc; clear;
close all;

%% Parameters
numBits = 600000;
EbNo_dB = -4:1:14; %SNR
EbNo = 10.^(EbNo_dB/10);
sqrtEbNo = sqrt(EbNo);

% Initialize BER arrays
berSim = zeros(6, length(EbNo_dB));     % 1: BPSK, 2: QPSK Gray, 3:
QPSK non-Gray, 4: 8PSK, 5: 16QAM
berTheory = zeros(6, length(EbNo_dB));

% Generate random data
dataBits = randi([0 1], 1, numBits);

%% ----- BPSK -----
bpskSymbols = 2 * dataBits - 1;
for i = 1:length(EbNo)
    noise = randn(1, numBits) / sqrt(2*EbNo(i));
    received = bpskSymbols + noise;
    detected = received > 0;
    berSim(1,i) = mean(dataBits ~= detected);
end
berTheory(1,:) = qfunc(sqrt(2)*sqrtEbNo);

%% ----- QPSK Gray -----
dataBitsGray = dataBits;
if mod(numBits, 2) ~= 0
    dataBitsGray = [dataBitsGray, 0];
    numBits = length(dataBitsGray);
end
numSymbols = numBits/2;
dataQPSK = reshape(dataBitsGray, 2, numSymbols).';
% QPSK Gray Mapping according to specified constellation:
% 11 to 45Â° (1+j)/sqrt(2)
% 01 to 135Â° (-1+j)/sqrt(2)
% 00 to 225Â° (-1-j)/sqrt(2)
% 10 to 315Â° (1-j)/sqrt(2)
map_gray = [(1+1j)/sqrt(2), (-1+1j)/sqrt(2), (-1-1j)/sqrt(2), (1-
1j)/sqrt(2)];
symbols = zeros(1, numSymbols);

for i = 1:numSymbols
    dibit = dataQPSK(i,:);
    if isequal(dibit, [1 1])
        symbols(i) = map_gray(1);  % 45Â° 11
    elseif isequal(dibit, [0 1])
        symbols(i) = map_gray(2);  % 135Â° 01
```

```
    elseif isequal(dibit, [0 0])
        symbols(i) = map_gray(3);  % 225Â°  00
    elseif isequal(dibit, [1 0])
        symbols(i) = map_gray(4);  % 315Â° 10
    end
end
for i = 1:length(EbNo)
    %noise = (randn(1,length(numSymbols)) +
1j*randn(1,length(numSymbols))) / sqrt(2*EbNo(i));
    noise = (randn(1, numSymbols) + 1j*randn(1, numSymbols))/sqrt(2);
    noiseScaled = noise / sqrt(2*EbNo(i));
    rx = symbols  + noiseScaled;

    detectedBits = zeros(1, numBits);
    for j = 1:numSymbols
        distances = abs(rx(j) - map_gray);
        [~, idx] = min(distances);

        if idx == 1
            detectedBits(2*j-1:2*j) = [1 1];  % 45Â°
        elseif idx == 2
            detectedBits(2*j-1:2*j) = [0 1];  % 135Â°
        elseif idx == 3
            detectedBits(2*j-1:2*j) = [0 0];  % 225Â°
        elseif idx == 4
            detectedBits(2*j-1:2*j) = [1 0];  % 315Â°
        end
    end
    berSim(2,i) = sum(detectedBits ~= dataBits) / numBits;
    %berSim(2,i) = mean(dataBitsGray ~= reshape(detectedBits.', 1,
[]));
end

berTheory(2,:) = 0.5 * erfc(sqrtEbNo);
%berTheory(2,:) = qfunc(sqrt(2)*sqrtEbNo);

%% ----- QPSK Non-Gray -----
dataBitsBinary = dataBits;
if mod(numBits, 2) ~= 0
    dataBitsBinary = [dataBitsBinary, 0];
    numBits = length(dataBitsBinary);
end
numSymbols = numBits/2;
dataQPSK = reshape(dataBitsBinary, 2, numSymbols).';
% QPSK Gray Mapping according to specified constellation:
% 10 to 45Â° (1+j)/sqrt(2)
% 01 to 135Â° (-1+j)/sqrt(2)
% 00 to 225Â° (-1-j)/sqrt(2)
% 11 to 315Â° (1-j)/sqrt(2)
angles = [45, 135, 225, 315] * pi/180;
map_binary = cos(angles) + 1j*sin(angles);
```

```
symbols = zeros(1, numSymbols);

for i = 1:numSymbols
    dibit = dataQPSK(i,:);
    if isequal(dibit, [1 0])
        symbols(i) = map_binary(1);  % 45Â° 10
    elseif isequal(dibit, [0 1])
        symbols(i) = map_binary(2);  % 135Â° 01
    elseif isequal(dibit, [0 0])
        symbols(i) = map_binary(3);  % 225Â°  00
    elseif isequal(dibit, [1 1])
        symbols(i) = map_binary(4);  % 315Â° 11
    end
end
for i = 1:length(EbNo)
    %noise = (randn(1,length(numSymbols)) +
1j*randn(1,length(numSymbols))) / sqrt(2*EbNo(i));
    noise = (randn(1, numSymbols) + 1j*randn(1, numSymbols))/sqrt(2);
    noiseScaled = noise / sqrt(2*EbNo(i));
    rx = symbols  + noiseScaled;

    detectedBits = zeros(1, numBits);
    for j = 1:numSymbols
        distances = abs(rx(j) - map_binary);
        [~, idx] = min(distances);

        if idx == 1
            detectedBits(2*j-1:2*j) = [1 0];  % 45Â°
        elseif idx == 2
            detectedBits(2*j-1:2*j) = [0 1];  % 135Â°
        elseif idx == 3
            detectedBits(2*j-1:2*j) = [0 0];  % 225Â°
        elseif idx == 4
            detectedBits(2*j-1:2*j) = [1 1];  % 315Â°
        end
    end
    berSim(3,i) = sum(detectedBits ~= dataBits) / numBits;
    %berSim(2,i) = mean(dataBitsGray ~= reshape(detectedBits.', 1,
[]));
end

berTheory(3,:) = 0.5 * erfc(sqrtEbNo);
%berTheory(2,:) = qfunc(sqrt(2)*sqrtEbNo);

%% ----- 8PSK -----
%getting data ready
if mod(numBits, 3) ~= 0
    padding = 3 - mod(numBits, 3);
    dataBits = [dataBits, zeros(1, padding)];
    numBits = length(dataBits);
end
```

```
numSymbols = numBits/3;
data_8PSK = reshape(dataBits, 3, numSymbols).';

angles = [0, 45, 90, 135, 180, 225, 270, 315] * pi/180;
map_8psk = cos(angles) + 1j*sin(angles);

%mapping
symbols = zeros(1, numSymbols);
for i = 1:numSymbols
    tribit = data_8PSK(i,:);
    if isequal(tribit, [0 0 0])
        symbols(i) = map_8psk(1);      % 0Â°
    elseif isequal(tribit, [0 0 1])
        symbols(i) = map_8psk(2);      % 45Â°
    elseif isequal(tribit, [0 1 1])
        symbols(i) = map_8psk(3);      % 90Â°
    elseif isequal(tribit, [0 1 0])
        symbols(i) = map_8psk(4);      % 135Â°
    elseif isequal(tribit, [1 1 0])
        symbols(i) = map_8psk(5);      % 180Â°
    elseif isequal(tribit, [1 1 1])
        symbols(i) = map_8psk(6);      % 225Â°
    elseif isequal(tribit, [1 0 1])
        symbols(i) = map_8psk(7);      % 270Â°
    elseif isequal(tribit, [1 0 0])
        symbols(i) = map_8psk(8);      % 315Â°
    end
end

for i = 1:length(EbNo)
    % noise
    noise = (randn(1, numSymbols) + 1j*randn(1, numSymbols))/sqrt(2);
    noiseScaled = noise / sqrt(3*EbNo(i));

    % Add noise signal
    rx = symbols + noiseScaled;

    % Demapping
    detectedBits = zeros(1, numBits);
    for j = 1:numSymbols
        distances = abs(rx(j) - map_8psk);
        [~, idx] = min(distances);%get min distance
        % Map signal
        if idx == 1
            detectedBits(3*j-2:3*j) = [0 0 0];      % 0Â°
        elseif idx == 2
            detectedBits(3*j-2:3*j) = [0 0 1];      % 45Â°
        elseif idx == 3
            detectedBits(3*j-2:3*j) = [0 1 1];      % 90Â°
        elseif idx == 4
```

```
            detectedBits(3*j-2:3*j) = [0 1 0];      % 135Â°
        elseif idx == 5
            detectedBits(3*j-2:3*j) = [1 1 0];      % 180Â°
        elseif idx == 6
            detectedBits(3*j-2:3*j) = [1 1 1];      % 225Â°
        elseif idx == 7
            detectedBits(3*j-2:3*j) = [1 0 1];      % 270Â°
        elseif idx == 8
            detectedBits(3*j-2:3*j) = [1 0 0];      % 315Â°
        end
    end

    % Calculate BER
    berSim(4,i) = sum(detectedBits ~= dataBits) / numBits;
end

berTheory(4,:)  = (1/3)*erfc(sqrt(3*EbNo)*sin(pi/8));


%% ----- 16-QAM -----
%edit data to be ready for 16QAM
if mod(numBits, 4) ~= 0
    padding = 4 - mod(numBits, 4);
    dataBits = [dataBits, zeros(1, padding)];
    numBits = length(dataBits);
end

numSymbols = numBits/4;
data16QAM = reshape(dataBits, 4, numSymbols).';

%axis values
I_values = [-3, -1, 1, 3];
Q_values = [3, 1, -1, -3];

qam_map = zeros(16, 1);
bit_patterns = zeros(16, 4);
idx = 1;

%mapper-------------------------
for q_idx = 1:4
    for i_idx = 1:4
        % Calculate coordinates
        i_val = I_values(i_idx);
        q_val = Q_values(q_idx);

        %map the bits to their points
        % First two bits (b0b1) determine column (I value)
        % Last two bits (b2b3) determine row (Q value)
        if i_idx == 1 %left column
            b0b1 = [0 0];
        elseif i_idx == 2
```

```
            b0b1 = [0 1];
        elseif i_idx == 3
            b0b1 = [1 1];
        else %last right column
            b0b1 = [1 0];
        end

        if q_idx == 1 % top row
            b2b3 = [1 0];
        elseif q_idx == 2
            b2b3 = [1 1];
        elseif q_idx == 3
            b2b3 = [0 1];
        else % bottom row
            b2b3 = [0 0];
        end

        % Store pattern
        bit_patterns(idx, :) = [b0b1, b2b3];

        %store conseltation points
        qam_map(idx) = (i_val + 1j*q_val);

        idx = idx + 1;
    end
end

% Normalize constellation points to have power equals 1
avg_power = mean(abs(qam_map).^2);
scale_factor = sqrt(avg_power);
qam_map = qam_map / scale_factor;


% Map 4-bit patterns to 16-QAM symbols
symbols = zeros(1, numSymbols);
for i = 1:numSymbols
    pattern = data16QAM(i, :);

    % Find the index of the pattern in our matrix
    for j = 1:16
        if all(pattern == bit_patterns(j, :))
            symbols(i) = qam_map(j);
            break;
        end
    end
end

%noise + demapper
for i = 1:length(EbNo)
    % Generate noise
    noise = (randn(1, numSymbols) + 1j*randn(1, numSymbols))/sqrt(2);
```

```
    noiseScaled = noise * sqrt((1/4)/EbNo(i));

    % Add noise
    rx = symbols + noiseScaled;

    % Demapping based on min distance
    detectedBits = zeros(1, numBits);
    for j = 1:numSymbols
        distances = abs(rx(j) - qam_map);
        [~, idx] = min(distances);

        % Map index back to its pattern
        detectedBits(4*j-3:4*j) = bit_patterns(idx, :);
    end

    %BER
    berSim(5,i) = sum(detectedBits ~= dataBits) / numBits;
end

berTheory(5,:) = (3/8)*erfc(sqrt(EbNo/2.5));


%% BFSK

%Parameters
BFSKnumBits = 200000;
BFSK_Eb=1; % assuming the energy per bit equals 1
BFSK_EbNo_dB = -5:1:15;
BFSK_EbNo = 10.^(BFSK_EbNo_dB/10);
BFSK_Noise_PSD = BFSK_Eb./(10.^(BFSK_EbNo_dB/10));
BER_BFSK = zeros(1, length(BFSK_EbNo_dB));
Theoretical_BER_BFSK = zeros(1, length(BFSK_EbNo_dB));

% Generate random data
BFSK_binraydataBits = randi([0 1], 1, BFSKnumBits);
%-------------------------------------------

% BFSK Mapping
BFSK_data=zeros(1,BFSKnumBits);
for i = 1 : (BFSKnumBits)
    if BFSK_binraydataBits(i)  == 0
        BFSK_data(i) = cos(0)+1i*sin(0); % mapping the '0' value to a
symbol has phase = 0 degree
    else
        BFSK_data(i) = cos(pi/2)+1i*sin(pi/2); % mapping the '1'
value to a symbol has phase = 90 degree
    end
end
%-------------------------------------------

%Looping over the EbNo values and calc the BER at each value
```

```
for i = 1:length(BFSK_EbNo_dB)
    % Generate Complex Noise has variance equals sqrt(No/2) in the I
& Q components
    BFSK_Noise = randn(1, BFSKnumBits) * sqrt(BFSK_Noise_PSD(i) / 2)
+ 1i .* randn(1,BFSKnumBits) * sqrt(BFSK_Noise_PSD(i) / 2);

    % Add noise to transmitted signal
    BFSK_Received_signal = BFSK_data + BFSK_Noise;


%% BFSK Demapper
 BFSK_Received_data= zeros(1, BFSKnumBits);
    for j = 1:BFSKnumBits
        % the region of the value '0' is betweem phases [45,-135]
        if (angle(BFSK_Received_signal(j)) >= -3*pi/4) &&
(angle(BFSK_Received_signal(j)) <= pi/4 )
            BFSK_Received_data(j) = 0;
        else % the region of the value '1' is betweem phases [45,225]
            BFSK_Received_data(j) = 1;
        end
    end


    %% BER of BFSK


    % Calculate BFSK BER
    BFSK_Error = abs(BFSK_Received_data -
BFSK_binraydataBits(1:BFSKnumBits));
    BER_BFSK(i) = sum(BFSK_Error) / BFSKnumBits; %

    % Calculate Theoretical BFSK BER
    Theoretical_BER_BFSK(i) = (1/2) * erfc(sqrt(1 / (2 *
BFSK_Noise_PSD(i))));



end

%-------------------------------------------------------------------
-

%% Plotting BFSK BER
figure;
semilogy(BFSK_EbNo_dB,BER_BFSK , '-o', 'linewidth', 2) ;
hold on
semilogy(BFSK_EbNo_dB, Theoretical_BER_BFSK ,'--','linewidth',2) ;
xlabel('Eb/No');
ylabel('BER');
legend('BFSK BER' , 'Theoretical BER ') ;
grid on
```

```
title('BFSK BER');

%--------------------------------------------------------------------
-

%% Auto-Corr. of 100 BFSK Realizations each is 1000 bits
num_realizations = 100;  % Number of realizations
bits_per_realization = 1000;     % Number of bits per realization
samples_per_bit=5; % Tb=5
upsampled_bfsk_data_length=bits_per_realization * samples_per_bit;



bfsk_binarydata = randi([0 1], 1, bits_per_realization);

bfsk_data_realizations = zeros(num_realizations,
bits_per_realization);
bfsk_data_realizations(1,:)= bfsk_binarydata;  % First realization
(no shift)

for r = 2:num_realizations
    % Apply a circular shift
    shift_amount = randi([1, bits_per_realization-1]);
    circshifted_data = circshift(bfsk_binarydata, [0, shift_amount]);
    bfsk_data_realizations(r,:) = circshifted_data;
end


Tb = samples_per_bit;
f0 = 0;             % Normalized frequency for '0'
f1 = 1 / Tb;   % Normalized frequency (1/Tb) for '1'
t = (0:Tb-1);  % Time samples per bit
 symbol_energy=sqrt(2 * BFSK_Eb / Tb);
mapped_bfsk_data = zeros(num_realizations,
upsampled_bfsk_data_length);

for r = 1:num_realizations
    for c = 1:bits_per_realization
        % indexing the data by the length of Tb (upsampling by factor
of 5)
        indx_start = (c-1)*Tb + 1;
        indx_end = c*Tb;

        if bfsk_data_realizations(r, c) == 0
            symbol = symbol_energy*(cos(2*pi*f0*t) +
1j*sin(2*pi*f0*t)); % mapping '0' to f0
        else
            symbol = symbol_energy*(cos(2*pi*f1*t) +
1j*sin(2*pi*f1*t)); % mapping '1' to f1
        end
```

```
        mapped_bfsk_data(r, indx_start:indx_end) = symbol;
    end
end
BFSK_Auto_Corr = bfsk_autocorr_func(mapped_bfsk_data);
BFSK_Auto_Corr_flipped = [fliplr(conj(BFSK_Auto_Corr(2:end)))
BFSK_Auto_Corr];

%---------------------------------------------------------------------
-


%% BFSK PSD
N = length(BFSK_Auto_Corr_flipped);
BFSK_PSD = abs(fftshift(fft(BFSK_Auto_Corr_flipped, N)));
BFSK_PSD = BFSK_PSD / max(BFSK_PSD);
fs = 5 * f1; % normalized by 5 as the mapped data is upsampled by
factor of 5 due to that the Tb equals 5
f = linspace(-fs/2, fs/2, N);
figure;
plot(f,BFSK_PSD, 'LineWidth', 1.5);
grid on;
xlabel('Normalized Frequency');
ylabel('BFSK Power');
title('PSD of BFSK');
ylim([0 0.05]);
figure;
plot(f,10*log(BFSK_PSD), 'LineWidth', 1.5);
grid on;
xlabel('Normalized Frequency');
ylabel('BFSK Power in dB');
title('PSD of BFSK in dB');
ylim([-200 10]);
xlim([-0.5 0.5]);

%----------------------------------
%% ----- Plot: Simulated BERs -----
figure;
semilogy(EbNo_dB, berSim(1,:), 'k-', 'LineWidth', 1.5); hold on;
semilogy(EbNo_dB, berSim(2,:), 'g-', 'LineWidth', 1.5);
semilogy(EbNo_dB, berSim(3,:), 'y-', 'LineWidth', 1.5);
semilogy(EbNo_dB, berSim(4,:), 'b-', 'LineWidth', 1.5);
semilogy(EbNo_dB, berSim(5,:), 'r-', 'LineWidth', 1.5);
semilogy(BFSK_EbNo_dB,BER_BFSK,'m-','linewidth',1.5) ;
xlabel('E_b/N_0 (dB)'); ylabel('BER'); grid on;
title('Simulated BER vs. E_b/N_0 for All Modulation Schemes');
legend('BPSK', 'QPSK Gray', 'QPSK Binary', '8PSK', '16QAM', 'BFSK');
set(gca, 'FontSize', 12);

%% ----- Plot: Theoretical BERs -----
figure;
semilogy(EbNo_dB, berTheory(1,:), 'k--', 'LineWidth', 1.5); hold on;
semilogy(EbNo_dB, berTheory(2,:), 'g--', 'LineWidth', 1.5);
```

```
semilogy(EbNo_dB, berTheory(3,:), 'y--', 'LineWidth', 1.5);
semilogy(EbNo_dB, berTheory(4,:), 'b--', 'LineWidth', 1.5);
semilogy(EbNo_dB, berTheory(5,:), 'r--', 'LineWidth', 1.5);
semilogy(BFSK_EbNo_dB,Theoretical_BER_BFSK,'m--','linewidth',1.5) ;
xlabel('E_b/N_0 (dB)'); ylabel('BER'); grid on;
title('Theoretical BER vs. E_b/N_0 for All Modulation Schemes');
legend('BPSK Theory', 'QPSK Gray Theory', 'QPSK Binary Theory', '8PSK
Theory', '16QAM Theory', 'BFSK Theory');
set(gca, 'FontSize', 12);


%% ----- Plot: Simulated vs. Theoretical Combined -----
figure;
semilogy(EbNo_dB, berSim(1,:), 'k-', EbNo_dB, berTheory(1,:), 'k--',
'LineWidth', 1.5); hold on;
semilogy(EbNo_dB, berSim(2,:), 'g-', EbNo_dB, berTheory(2,:), 'g--',
'LineWidth', 1.5);
semilogy(EbNo_dB, berSim(3,:), 'y-', EbNo_dB, berTheory(3,:), 'y--',
'LineWidth', 1.5);
semilogy(EbNo_dB, berSim(4,:), 'b-', EbNo_dB, berTheory(4,:), 'b--',
'LineWidth', 1.5);
semilogy(EbNo_dB, berSim(5,:), 'r-', EbNo_dB, berTheory(5,:), 'r--',
'LineWidth', 1.5);
semilogy(BFSK_EbNo_dB, BER_BFSK, 'm-',BFSK_EbNo_dB,
Theoretical_BER_BFSK, 'm--', 'LineWidth', 1.5);
xlabel('E_b/N_0 (dB)'); ylabel('BER'); grid on;
title('Simulated & Theoretical BER vs. E_b/N_0');
legend({'BPSK Sim', 'BPSK Theory', ...
        'QPSK Gray Sim', 'QPSK Gray Theory', ...
        'QPSK Binary Sim', 'QPSK Binary Theory', ...
        '8PSK Sim', '8PSK Theory', ...
        '16QAM Sim', '16QAM Theory', ...
        'BFSK Sim', 'BFSK Theory'}, ...
        'Location', 'southwest');
set(gca, 'FontSize', 12);

%% plot for every type separatly
modNames = {'BPSK', 'QPSK Gray', 'QPSK Binary', '8PSK', '16QAM'};
colors = {'k', 'g', 'b', 'k', 'r'};

for m = 1:5
    figure;
    semilogy(EbNo_dB, berSim(m,:), [colors{m} '-'], 'LineWidth',
1.5); hold on;
    semilogy(EbNo_dB, berTheory(m,:), [colors{m} '--'], 'LineWidth',
1.5);
    xlabel('E_b/N_0 (dB)');
    ylabel('BER');
    title(['BER vs E_b/N_0 for ', modNames{m}]);
    legend([modNames{m} ' Sim'], [modNames{m} ' Theory'], 'Location',
'southwest');
```

```
    grid on;
    set(gca, 'FontSize', 12);
end

figure;
semilogy(EbNo_dB, berSim(2,:), 'g-', 'LineWidth', 1.5); hold on;
semilogy(EbNo_dB, berSim(3,:), 'y-', 'LineWidth', 1.5);
semilogy(EbNo_dB, berTheory(2,:), 'g--', 'LineWidth', 1.5);
semilogy(EbNo_dB, berTheory(3,:), 'y--', 'LineWidth', 1.5);

xlabel('E_b/N_0 (dB)');
ylabel('BER');
title('BER Comparison: QPSK Gray vs QPSK Binary');
legend('QPSK Gray Sim', 'QPSK Binary Sim', ...
       'QPSK Gray Theory', 'QPSK Binary Theory', ...
       'Location', 'southwest');
grid on;
set(gca, 'FontSize', 12);


%% BFSK Auto-Correlation Function
function autocorr_values = bfsk_autocorr_func(bfsk_data_matrix)

    first_col = bfsk_data_matrix(:, 1);
    num_realizations = size(bfsk_data_matrix, 1);


    num_samples = size(bfsk_data_matrix, 2);
    autocorr_values = zeros(1, num_samples);

    for col = 1:num_samples
        current_col = bfsk_data_matrix(:, col);
        dot_product = sum(current_col .* conj(first_col));
        autocorr_values(col) = dot_product / num_realizations;
    end
end
```