Department of Electronics and Electrical

Communications Engineering

Faculty of Engineering

Cairo University

[DFT & OFDM system project]

4th Year
1st Semester – Academic year 2024/2025

Presented by

Name	SEC	BN	ID
أحمد عادل يونس سيد	1	17	9213073
كريم أيمن محمد فخر الدين	3	22	9210836

Under supervision of

Prof. Mohamed Khairy

Eng / Mohamed Nady

➢ Problem 1: Execution time of DFT and FFT

- In this section, it is required to compare between the execution time of Discrete Fourier Transform (DFT) and Fast Fourier Transform (FFT) where the discrete Fourier transform X(k) represented by this equation: $X(k) = \sum_{n=0}^{N-1} x(n)\, e^{\frac{-j2\pi nk}{N}} \quad 0 \le k \le N-1$

```
clc
clear
close all;

%% First part (Execution time of DFT and FFT)
L = 8192;
x = rand(1, L);       % Xi[n] random signal

% Measure execution time for DFT {x[n]}
tic;
X_dft = DFT(x);
dft_time = toc;
fprintf('Execution Time for DFT = %.4f seconds \n', dft_time);

% Measure execution time for FFT {x[n]}
tic;
X_fft = fft(x);       % Calculate fast fourier transform directly using built-in function fft()
fft_time = toc;
fprintf('Execution Time for FFT = %.4f seconds\n', fft_time);

%% DFT implementation Function
function X = DFT(x)
  N = length(x);
  X = zeros(1, N);
  for k = 0:N-1
    for n = 0:N-1
      X(k+1) = X(k+1) + x(n+1) * exp(-1j * 2 * pi * n * k / N);
    end
  end
end
```

- After simulating the code, we found that the execution time of DFT function is **much higher** than FFT built-in function as shown in figure (1). Therefore, FFT offers superior performance with respect to the execution time.

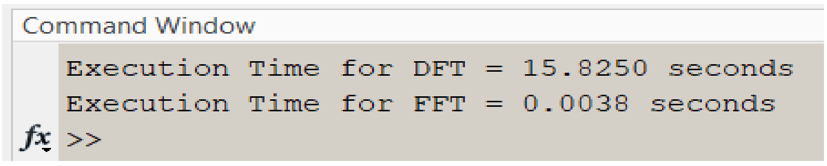
Command Window
```
Execution Time for DFT = 15.8250 seconds
Execution Time for FFT = 0.0038 seconds
>>
```

Figure (1): Simulation result

➢ Problem 2: Bit-error rate performance for BPSK & QPSK & 16-QAM over Rayleigh flat fading channel

```
clc
clear
close all;

%% − − − − − − − − − −  Generating the Random Data Sequence − − − − − − − − − − − − %%

Number_of_Bits = 120000;
Eb = 1;             % Eb represent the bit eneregy
Eb_QAM = 2.5;
SNR_range_db = (−5: 15);  % where SNR_range = Eb/No
SNR_range_linear = 10 .^ (SNR_range_db/10);
No = Eb ./ SNR_range_linear;
Random_Data = randi([0 1] , 1 , Number_of_Bits);  % generate all the random bits

%% − − − − − − − − − − −BPSK Modulation Scheme − − − − − − − − − − − − − − −%%

% − − − − − − − − − − − − − − − − − −Declare BPSK (Nocode) − − − − − − − − − − − − − − − −%
BPSK = Mapper(Random_Data, 1, Eb); % Symbol_Bits = 1 , Eb = 1

% − − − − − − − − − − − − − BPSK Channel stage (No code) − − − − − − − − − − − − − %
BPSK_recieved = zeros(1, length(BPSK));
BER_BPSK = zeros(1, length(SNR_range_db));
h_BPSK
= (1/sqrt(2)) .
∗ complex(randn(1, length(BPSK)) , randn(1, length(BPSK)));  % channel  with zero mean & variance = 1/2

for i = 1 : length(SNR_range_db)
  noise_BPSK = sqrt(No(i)/2).∗ complex(randn(1, length(BPSK)) , randn(1, length(BPSK))) ;
  BPSK_recieved = (BPSK .∗ h_BPSK + noise_BPSK)./ h_BPSK;
  BPSK_demapped = DeMapper(BPSK_recieved, 1);      % calling the demapper function

  % − − − − − − − − − − BER calculation for the BPSK Scheme(No code) − − − − − − − − − − %
  error_bits_BPSK = 0;
  for j = 1 : length(Random_Data)
    if(BPSK_demapped(j) ~ = Random_Data(j))
      error_bits_BPSK = error_bits_BPSK + 1;
    end
  end
  BER_BPSK(1, i) = error_bits_BPSK/Number_of_Bits;
end
% − − − − − − − − − − − − − − − − − −Declare BPSK (Rep Code) − − − − − − − − − − − − − − − −%
BPSK_rep_bstream = SymbolRepetitionEncoder(Random_Data, 1);
BPSK_rep = Mapper(BPSK_rep_bstream, 1, Eb);

% − − − − − − − − − − − − − BPSK Channel stage (Rep code) − − − − − − − − − − − − − %
BPSK_rep_recieved = zeros(1, length(BPSK_rep));
BER_BPSK_rep = zeros(1, length(SNR_range_db));

h_BPSK_rep = (1/sqrt(2)) .∗ complex(randn(1, length(BPSK_rep)) , randn(1, length(BPSK_rep)));

for i = 1 : length(SNR_range_db)
  noise_BPSK_rep = sqrt(No(i)/2).∗ complex(randn(1, length(BPSK_rep)) , randn(1, length(BPSK_rep))) ;
  BPSK_rep_recieved = (BPSK_rep .∗ h_BPSK_rep + noise_BPSK_rep)./ h_BPSK_rep;
      BPSK_rep_demapped = DeMapper(BPSK_rep_recieved, 1);      % calling the demapper function
 BPSK_rep_decoded = majority_decoder(BPSK_rep_demapped, 1);      % Apply the majority decoder function
```

```
% − − − − − − − − − − BER calculation for the BPSK Scheme(Rep code) − − − − − − − − − − %
  error_bits_BPSK  = 0;
  for j  = 1 : length(Random_Data)
    if(BPSK_rep_decoded(j) ~ = Random_Data(j))
      error_bits_BPSK  = error_bits_BPSK  + 1;
    end
  end
  BER_BPSK_rep(1, i)  = error_bits_BPSK/Number_of_Bits;
end

% − − − − − − − − − − − − − − − − − − − −plotting the BER for BPSK − − − − − − − − − − − − − − −%
figure;
semilogy(SNR_range_db , BER_BPSK,'k','linewidth', 1.5);
hold on;
semilogy(SNR_range_db , BER_BPSK_rep,'r','linewidth', 1.5);
%hold off;
title('BER of BPSK Modulation Scheme');
xlabel('Eb/No (dB)');
grid on;
legend('BER without repetition coding','BER with repetition coding','Location','southwest')

%% − − − − − − − − − − − − QPSK Modulation Scheme  − − − − − − − − − − − − −%%

% − − − − − − − − − − − − − − − − − − − Declare QPSK (No code) − − − − − − − − − − − − − − %
QPSK  = Mapper(Random_Data, 2, Eb);

% − − − − − − − − − − − − − − − − − − − −QPSK channel stage(No code) − − − − − − − − − − − − −%
QPSK_recieved  = zeros(1, length(QPSK));
BER_QPSK  = zeros(1, length(SNR_range_db));
h_QPSK  = (1/sqrt(2)) .∗ complex(randn(1, length(QPSK)) , randn(1, length(QPSK)));

for i  = 1 : length(SNR_range_db)
  noise_QPSK  = sqrt(No(i)/2).∗ complex(randn(1, length(QPSK)) , randn(1, length(QPSK)));
  QPSK_recieved  = (QPSK .∗ h_QPSK  + noise_QPSK) ./ h_QPSK;
  QPSK_demapped  = DeMapper(QPSK_recieved, 2);     % Calling demapper function

  % − − − − − − − − − − −BER calculation for QPSK (No code) − − − − − − − − − − − − − − −%
  error_bits_QPSK  = 0;
  inc_var  = 1;
  for j  = 1 : length(Random_Data)/2
    for k  = 1 : 2
      if(QPSK_demapped(j, k) ~ = Random_Data(inc_var))
        error_bits_QPSK  = error_bits_QPSK  + 1;
      end
      inc_var  = inc_var  + 1;
    end
  end
  BER_QPSK(1, i)  = error_bits_QPSK/Number_of_Bits;
end

% − − − − − − − − − − − − − − − − − − − Declare QPSK (Rep code) − − − − − − − − − − − − − − %
QPSK_rep_bstream  =  SymbolRepetitionEncoder(Random_Data, 2);
QPSK_rep  = Mapper(QPSK_rep_bstream, 2, Eb);

% − − − − − − − − − − − − − − − − − − − −QPSK channel stage(Rep code) − − − − − − − − − − − − − − − − − − %
QPSK_rep_recieved  = zeros(1, length(QPSK_rep));
BER_QPSK_rep  = zeros(1, length(SNR_range_db));
h_QPSK_rep  = (1/sqrt(2)) .∗ complex(randn(1, length(QPSK_rep)) , randn(1, length(QPSK_rep)));

for i  = 1 : length(SNR_range_db)
  noise_QPSK_rep  = sqrt(No(i)/2).∗ complex(randn(1, length(QPSK_rep)) , randn(1, length(QPSK_rep)));
  QPSK_rep_recieved  = (QPSK_rep .∗ h_QPSK_rep  + noise_QPSK_rep) ./ h_QPSK_rep;
  QPSK_demapped_rep = DeMapper(QPSK_rep_recieved, 2);     % Calling demapper function
  QPSK_decoded_rep = majority_decoder(reshape(QPSK_demapped_rep', 1, []), 2);
  numSymbols = length(QPSK_decoded_rep) / 2;
  QPSK_decoded_rep = inverseReshape(QPSK_decoded_rep, [numSymbols, 2]);

  % −−−−−−−−−−−−−− BER calculation for QPSK (Rep code) −−−−−−−−−−−−−− %
  error_bits_QPSK = 0;
  inc_var = 1;
  for j = 1 : length(Random_Data)/2
    for k = 1 : 2
      if(QPSK_decoded_rep(j, k) ~= Random_Data(inc_var))
        error_bits_QPSK = error_bits_QPSK + 1;
      end
      inc_var = inc_var + 1;
    end
  end
  BER_QPSK_rep(1, i) = error_bits_QPSK/Number_of_Bits;
end

% −−−−−−−−−−−−−−−−−−− plotting the BER for QPSK −−−−−−−−−−−−−−−−−−−−%
figure;
semilogy(SNR_range_db , BER_QPSK , 'k', 'linewidth', 1.5);
hold on;
semilogy(SNR_range_db , BER_QPSK_rep , 'r', 'linewidth', 1.5);
title('BER of QPSK Modulation Scheme');
xlabel('Eb/No (dB)');
grid on;
legend('BER without repetition coding', 'BER with repetition coding', 'Location', 'southwest')

%% −−−−−−−−−−−− 16 − QAM Modulation Scheme −−−−−−−−−−−−−−−− %%

% −−−−−−−−−−−− Declare 16 − QAM(No code) −−−−−−−−−−−−−−−−−−−%
MQAM = Mapper(Random_Data, 4, Eb_QAM);

% −−−−−−−−−−−−−−−− 16 − QAM channel stage(No code) −−−−−−−−−−−−%
MQAM_recieved = zeros(1, length(MQAM));
BER_MQAM = zeros(1, length(SNR_range_db));
h_MQAM = (1/sqrt(2)) .* complex(randn(1, length(MQAM)) , randn(1, length(MQAM)));

for i = 1 : length(SNR_range_db)
  noise_MQAM = sqrt(No(i)/2).* complex(randn(1, length(MQAM)) , randn(1, length(MQAM)));
  MQAM_recieved = (MQAM .* h_MQAM + noise_MQAM) ./ h_MQAM;
  MQAM_demapped = DeMapper(MQAM_recieved, 4);         % Calling demapper function

  % −−−−−−−−−−BER calculation for 16 − QAM scheme(No code) −−−−−−−−−−%
  error_bits_MQAM = 0;
  inc_var = 1;
  for j = 1 : length(Random_Data)/4
    for k = 1 : 4
      if(MQAM_demapped(j, k) ~= Random_Data(inc_var))
        error_bits_MQAM = error_bits_MQAM + 1;
      end
      inc_var = inc_var + 1;
    end
  end
  BER_MQAM(1, i) = error_bits_MQAM/Number_of_Bits;
end

% −−−−−−−−−−−− Declare 16 − QAM(Rep Code) −−−−−−−−−−−−−−−−−− %
MQAM_rep_bstream = SymbolRepetitionEncoder(Random_Data, 4);
MQAM_rep = Mapper(MQAM_rep_bstream, 4, Eb_QAM);

% −−−−−−−−−−−−−−−− 16 − QAM channel stage(Rep code) −−−−−−−−−−−−%
```

```
MQAM_rep_recieved = zeros(1, length(MQAM_rep));
BER_MQAM_rep = zeros(1, length(SNR_range_db));
h_MQAM_rep = (1/sqrt(2)) .* complex(randn(1, length(MQAM_rep)) , randn(1, length(MQAM_rep)));

for i = 1 : length(SNR_range_db)
  noise_MQAM_rep = sqrt(No(i)/2).* complex(randn(1, length(MQAM_rep)) , randn(1, length(MQAM_rep)));
  MQAM_rep_recieved = (MQAM_rep .* h_MQAM_rep + noise_MQAM_rep) ./ h_MQAM_rep;
  MQAM_demapped_rep = DeMapper(MQAM_rep_recieved, 4);  % Calling demapper function
  MQAM_decoded_rep = majority_decoder(reshape(MQAM_demapped_rep', 1, []),4);
  numSymbols = length(MQAM_decoded_rep) / 4;
  MQAM_decoded_rep = inverseReshape(MQAM_decoded_rep, [numSymbols, 4]);

  % − − − − − − − − − − − − −BER calculation for 16 − QAM scheme(Rep code) − − − − − − − − − − %
  error_bits_MQAM = 0;
  inc_var = 1;
  for j = 1 : length(Random_Data)/4
    for k = 1 : 4
      if(MQAM_decoded_rep(j, k) ~ = Random_Data(inc_var))
        error_bits_MQAM = error_bits_MQAM + 1;
      end
      inc_var = inc_var + 1;
    end
  end
  BER_MQAM_rep(1, i) = error_bits_MQAM/Number_of_Bits;
end

% − − − − − − − − − − − − − − − − − − − plotting the BER for 16 − QAM − − − − − − − − − − − − − − − − − %
figure;
semilogy(SNR_range_db , BER_MQAM, 'k', 'linewidth', 1.5);
hold on;
semilogy(SNR_range_db , BER_MQAM_rep, 'r', 'linewidth', 1.5);
title('BER of 16 − QAM Modulation Scheme');
xlabel('Eb/No (dB)');
grid on;
legend('BER without repetition coding', 'BER with repetition coding', 'Location', 'southwest')


%% − − − − − − − − − − − −Mapper function − − − − − − − − − − − − − %%
function mapped_constellations = Mapper(inputstream, symbolBits, Eb)       % Eb represents the info energy
Number_of_Bits = length(inputstream);
mapped_constellations = zeros(1, Number_of_Bits/symbolBits);

switch symbolBits
  case 1 % BPSK mapping
    mapped_constellations = sqrt(Eb) ∗ (2 ∗ inputstream − 1);
  case 2 % QPSK mapping
    for i = 1: symbolBits: Number_of_Bits
      if(inputstream(i) == 0 && inputstream(i + 1) == 0 )
        mapped_constellations((i + 1)/2) = sqrt(Eb) ∗ complex(−1, −1);
      elseif(inputstream(i) == 0 && inputstream(i + 1) == 1)
        mapped_constellations((i + 1)/2) = sqrt(Eb) ∗ complex(−1,1);
      elseif(inputstream(i) == 1 && inputstream(i + 1) == 0)
        mapped_constellations((i + 1)/2) = sqrt(Eb) ∗ complex(1, −1);
      elseif(inputstream(i) == 1 && inputstream(i + 1) == 1)
        mapped_constellations((i + 1)/2) = sqrt(Eb) ∗ complex(1,1);
      end
    end
  case 4 % 16 − QAM mapping
    real_MQAM = zeros(1, length(inputstream)/symbolBits);
    img_MQAM = zeros(1, length(inputstream)/symbolBits);
    for i = 1: symbolBits: Number_of_Bits
      if(inputstream(i) == 0 && inputstream(i + 1) == 0)        % First two bits control the real part
        real_MQAM((i + 3)/4) = −3;
        elseif(inputstream(i) == 0 && inputstream(i + 1) == 1)
          real_MQAM((i + 3)/4) = −1;
        elseif(inputstream(i) == 1 && inputstream(i + 1) == 1)
          real_MQAM((i + 3)/4) = 1;
        elseif(inputstream(i) == 1 && inputstream(i + 1) == 0)
          real_MQAM((i + 3)/4) = 3;
        end
        if(inputstream(i + 2) == 0 && inputstream(i + 3) == 0)       % Second two bits control the imaginary
          img_MQAM((i + 3)/4) = −3;
        elseif(inputstream(i + 2) == 0 && inputstream(i + 3) == 1)
          img_MQAM((i + 3)/4) = −1;
        elseif(inputstream(i + 2) == 1 && inputstream(i + 3) == 1)
          img_MQAM((i + 3)/4) = 1;
        elseif(inputstream(i + 2) == 1 && inputstream(i + 3) == 0)
          img_MQAM((i + 3)/4) = 3;
        end
      mapped_constellations((i + 3)/4) = sqrt(Eb) ∗ complex(real_MQAM((i + 3)/4), img_MQAM((i + 3)/4));
      end
end

end

%% − − − − − − − − − − − − Repeatition encoder − − − − − − − − − − − − − −  %%
function encodedStream = SymbolRepetitionEncoder(inputStream, symbolBits)
  % Reshape the input stream into symbols
  numSymbols = length(inputStream) / symbolBits;
  symbols = reshape(inputStream, symbolBits, numSymbols)′;
  repeatedSymbols = repelem(symbols, 3, 1);                    % Repeats each symbol by 3 times
  % Flatten the repeated symbols back into a binary stream
  encodedStream = repeatedSymbols′;
  encodedStream = encodedStream(: )′;
end

%% − − − − − − − − − −  DeMapper Function − − − − − − − − − − − − − − %%
function demapped_data = DeMapper (received_data , symbolBits)
  Eb = 1;
  Eb_QAM = 1;
  demapped_data = zeros(1, length(received_data));
  if symbolBits == 1        % BPSK case
    BPSK_table = sqrt(Eb) ∗ [complex(−1,0), complex(1,0)];
    for j = 1 : length(received_data)
      [~ , Min_index] = min(abs(received_data(j) − BPSK_table));
      demapped_data(j) = Min_index − 1;
    end
    demapped_data = de2bi(demapped_data, 1, 'left − msb');
  elseif symbolBits == 2      % QPSK case
    QPSK_table = sqrt(Eb) ∗ [complex(−1, −1), complex(−1,1), complex(1, −1), complex(1,1)];
    for j = 1 : length(received_data)
      [~ , Min_index] = min(abs(received_data(j) − QPSK_table));
      demapped_data(j) = Min_index − 1;
    end
    demapped_data = de2bi(demapped_data, 2, 'left − msb');
  elseif symbolBits == 4      % 16 − QAM case
    MQAM_table
= sqrt(Eb_QAM)
∗ [complex(−3, −3), complex(−3, −1), complex(−3,3), complex(−3,1), complex(−1, −3), complex(−1, −1),
 complex(−1,3), complex(−1,1), complex(3, −3), complex(3, −1), complex(3,3), complex(3,1), complex(1, −3),
complex(1, −1), complex(1,3), complex(1,1)];
    for j = 1 : length(received_data)
      [~ , Min_index] = min(abs(received_data(j) − MQAM_table));
      demapped_data(j) = Min_index − 1;
    end
    demapped_data = de2bi(demapped_data, 4, 'left − msb');
  end
end
%% -------------- Decoding Function ------------- %%
function decoded_bitstream = majority_decoder (encoded_bitstream ,symbolBits )
  hash_table = zeros(1,2^symbolBits);
  iterations = 0;
  j = 1;
  decoded_bitstream = zeros(1, length(encoded_bitstream)/3);      % Assume 3 – repeatition code
  for i = 1 : symbolBits : length(encoded_bitstream)
    iterations = iterations + 1 ;
    % Convert the symbol into the correspoding decimal into a hashtable
    % to see the no. of occurrences of such symbol for decision
    index = bin2dec(num2str((encoded_bitstream(i: i + symbolBits – 1)))) + 1;
    hash_table(index) = hash_table(index) + 1;
    if(iterations == 3)
      %decide based on the most no. of occurrences
      % then get the corrsponding binary symbol
      index_of_most_repeated_symbol = find(hash_table == max(hash_table),1);
      binaryStr = dec2bin(index_of_most_repeated_symbol – 1, symbolBits);
      decoded_bitstream(j: j + symbolBits – 1) = double(binaryStr) – '0';
      j = j + symbolBits;
      hash_table = zeros(1,2^symbolBits); % Reset the hash table for the next set of symbols
      iterations = 0;
    end
  end
end

%% ---------- Inverse reshape Function ---------- %%
function originalMatrix = inverseReshape(rowVector, originalSize)
  % Reshape the row vector into the transposed matrix shape
  transposedMatrix = reshape(rowVector, fliplr(originalSize));

  % Transpose the matrix to restore the original order
  originalMatrix = transposedMatrix.';
end
```

❖ The above code explains the steps of BER calculation over Rayleigh fading channel:

1) Generate Random Bit stream $b_k = [0, 1, 0, 1, \dots]$.

2) Generate BPSK symbols based on the bit stream $x_k = [-\sqrt{E_b}\, , \sqrt{E_b}\, , -\sqrt{E_b}\, , \sqrt{E_b}\, , \dots]$ and generate the QPSK & 16-QAM symbols based on their constellation.

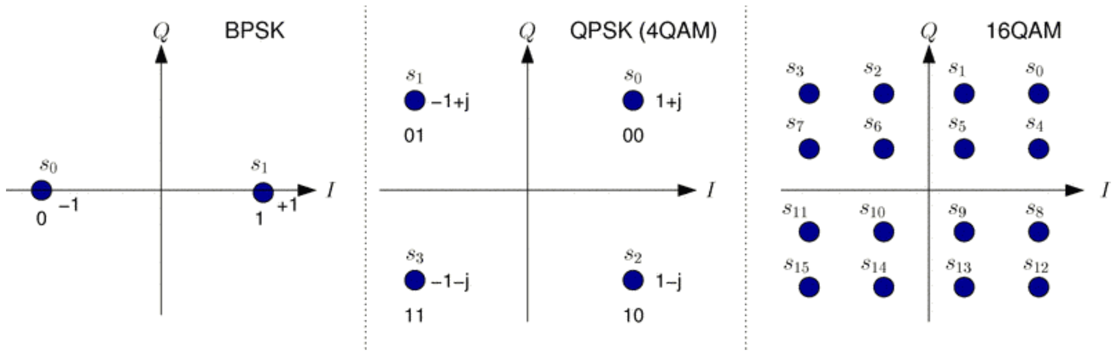

Figure (2): Constellation diagram for BPSK, QPSK, 16-

3) Generate the complex channel vector $(h_r + jh_i)$ and complex noise vector $(n_c + jn_s)$ based on the distributions explained above.

4) Compute the received symbol vector as follows:

$$\boxed{\boldsymbol{y_k = (h_r + jh_i)_k * x_k + (n_c + jn_s)_k \quad where\ k = 0, 1, 2, \ldots}}$$

5) Compensate for the channel gain at the receiver (assuming the channel is **known** at the receiver), apply correlator and make hard decision decoding to estimate the transmitted bit stream $(\hat{b}_k)$.

6) Compute the bit-error rate (BER) for each SNR value $[\boldsymbol{SNR} = \frac{\boldsymbol{E_b}}{\boldsymbol{N_o}}]$.

7) Plot the BER against SNR.

8) Repeat the above steps using a rate 1/3 repetition code. This is done by transmitting every “1” as three “1’s” and every “0” as three “0’s”.

- After simulating the code, we produce BER graph for each modulation scheme (BPSK, QPSK, 16-QAM) where each graph compares certain modulation scheme without any channel coding vs with channel coding (repetition 3 coding).

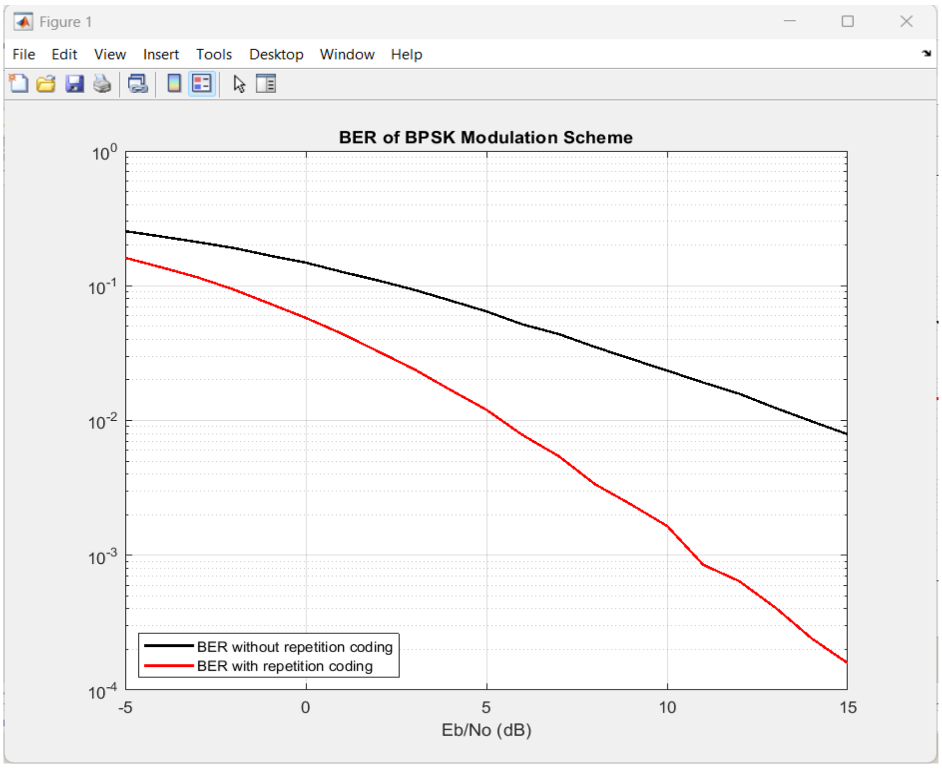

Figure (3): BER vs SNR for BPSK modulation scheme

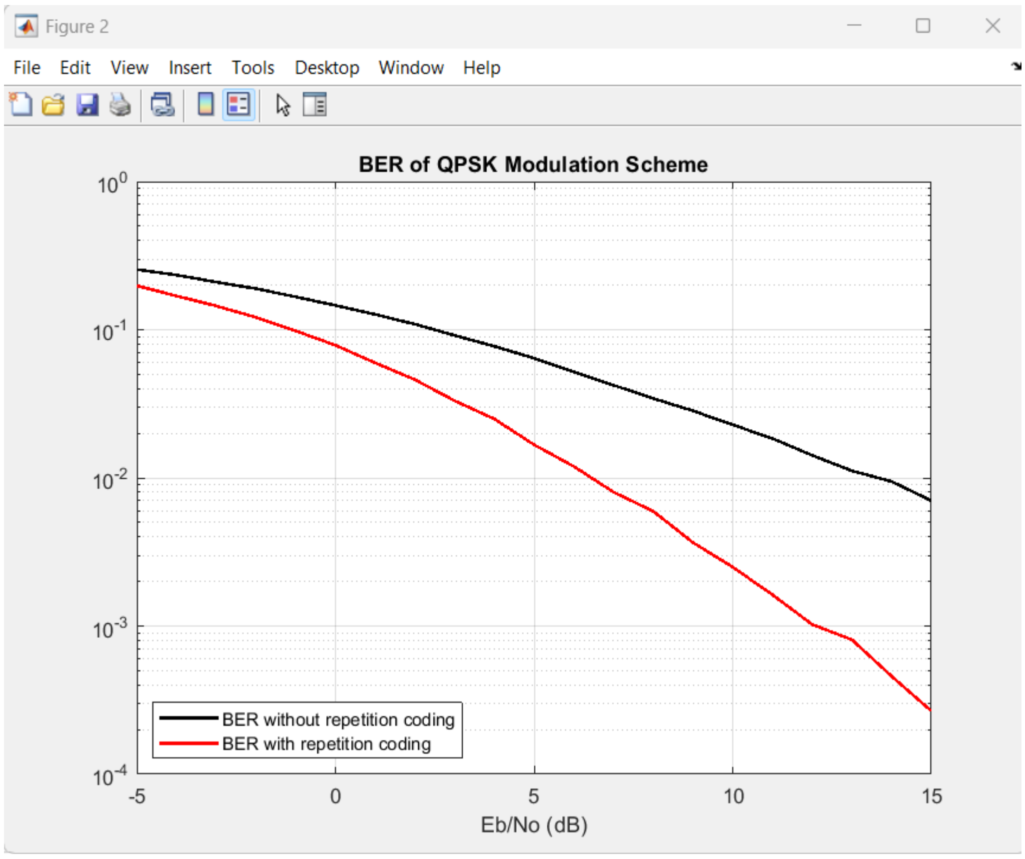

Figure (4): BER vs SNR for QPSK modulation scheme

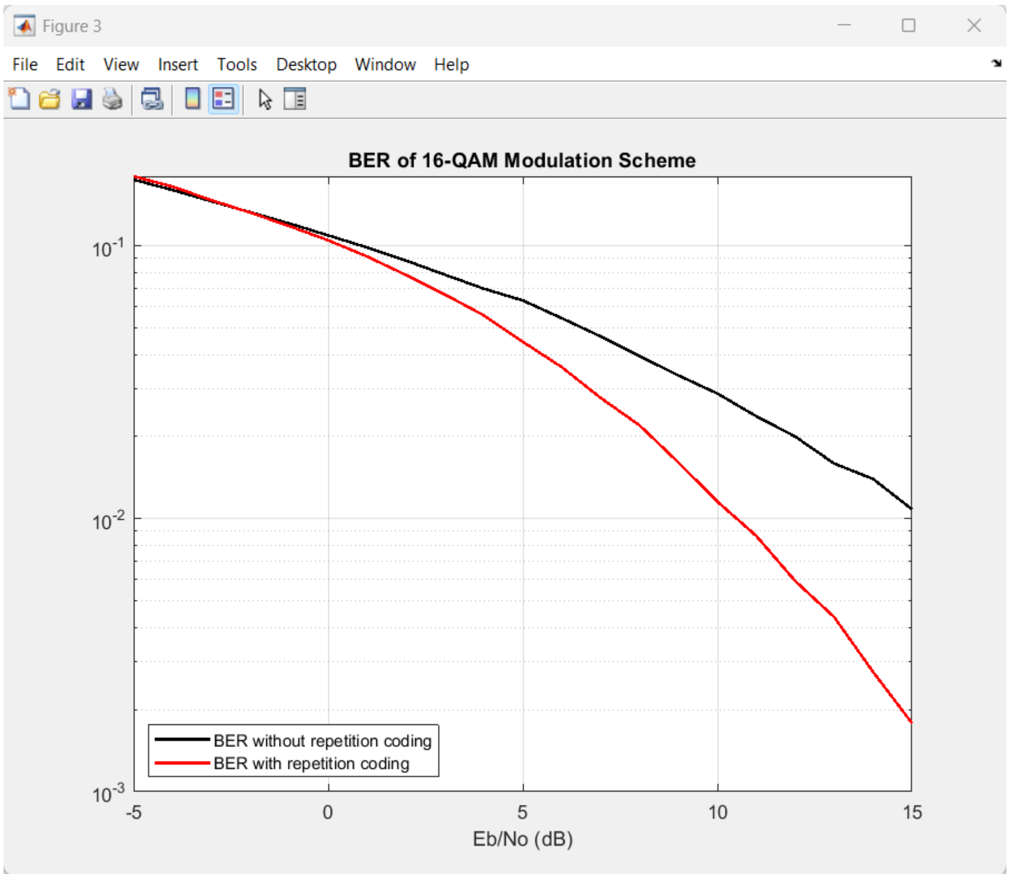

Figure (5): BER vs SNR for 16-QAM modulation scheme

✓ From the above graphs we notice that:

1. The BER in case of repetition coding is **better than** BER with no coding and this is what we expect as in repetition coding, each bit is transmitted multiple times (e.g., three times for 3-repetition coding) so, the receiver uses majority decoding logic to decide the bit value. single-bit error in any of the three bits still results in correct decoding using majority logic.

2. BER for BPSK is very close to BER for QPSK and the reason behind that is the theoretical value of the BER for BPSK **equal to** that of QPSK

$$\boxed{BER_{BPSK} = BER_{QPSK} = \frac{1}{2} * erfc(\sqrt{SNR})}$$

➢ Problem 3: OFDM system simulation

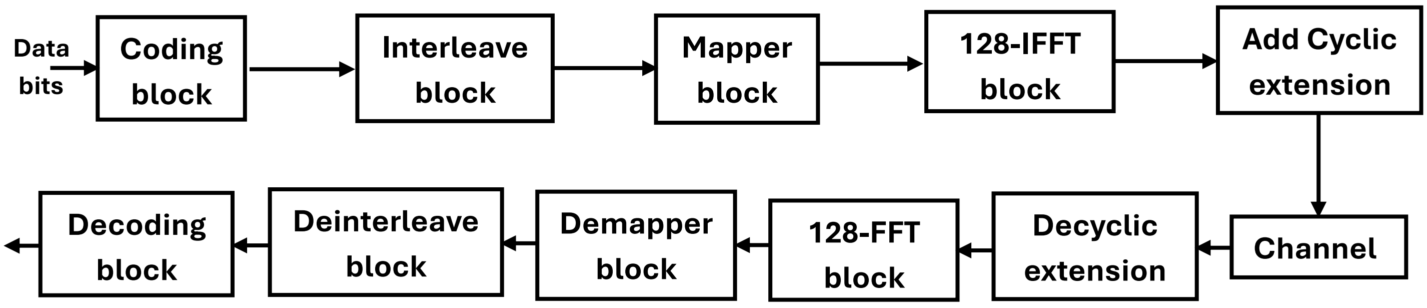

Figure (6): OFDM system block diagram

✓ Role of each block in brief:

1. **Coding block** → The input to the system is a stream of binary data bits is coded first (for channel coding "repetition" only). This coding can be applied to the data bits to improve BER.

2. **Interleave block** → The data bits are interleaved to spread the impact of burst errors caused by channel fading. In our case we use interleaver with size (16*16) for QPSK, and the size of the interleaver is (32*16) for 16-QAM.

3. **Mapper block** → The interleaved bits are mapped to complex symbols (e.g., QPSK, 16-QAM) using a constellation mapping scheme mentioned above in figure (2).

4. **IFFT block →** The mapped symbols are grouped into blocks of 128 symbols and transformed into the time domain using an Inverse Fast Fourier Transform function ifft ().

5. **Cyclic extension block →** The last part of the IFFT output is appended to the beginning of the OFDM symbol, creating a cyclic prefix. Where the length of cyclic prefix = 25% of IFFT length (in our case cyclic prefix length = 32). This process helps mitigate Inter-Symbol Interference (ISI) caused by channel delay spread.

6. **Channel →** The OFDM symbols are transmitted through the channel. Two channel models are considered, which are Rayleigh Flat Fading & Frequency Selective Fading.

7. **Receiver blocks →** All blocks at the Rx perform the reverse operation performed by the transmitter blocks mentioned before.

```
clc
clear
close all;

%% ------------ QPSK & 16 - QAM with No coding ---------------- %%

Number_of_Bits = 435200;    % 43520 , 435200 , 1044480
Random_Data = randi([0 1],1, Number_of_Bits);
Eb = 1;
Eb_QAM = 2.5 * Eb;
SNR_range_db = (-5: 15);

% ------------------------ Interleaver ---------------------%
for i = 1: 256: Number_of_Bits
   QPSK_intrlv(1, i: i + 255) = matintrlv(Random_Data(1, i: i + 255),16,16);     % for QPSK Modulation
end

for i = 1: 512: Number_of_Bits
   QAM_intrlv(1, i: i + 511) = matintrlv(Random_Data(1, i: i + 511),32,16);     % for 16 - QAM Modulation
end

% ------------------------- -Mapper ------------------------ -%
QPSK_mapped = Mapper(QPSK_intrlv, 2, Eb);
QAM_mapped = Mapper(QAM_intrlv, 4, Eb_QAM);

% -------------------------- IFFT ----------------------- %
QPSK_IFFT = zeros(1, length(QPSK_mapped));
for i = 1: 128: length(QPSK_mapped)
  QPSK_IFFT(1, i: i + 127) = ifft(QPSK_mapped(1, i: i + 127),128);
end

QAM_IFFT = zeros(1, length(QAM_mapped));
for i = 1: 128: length(QAM_mapped)
  QAM_IFFT(1, i: i + 127) = ifft(QAM_mapped(1, i: i + 127),128);
end

% ----------------------- Cyclic Extension ------------------ %
count1 = 1;
QPSK_CYC = [];         % Initialize the QPSK_CYC matrix
```

```
for i = 1:128:(Number_of_Bits/2)
  % Add cyclic prefix with size 32 (cyclic prefix length) and append to QPSK_IFFT to produce 160 bit block QPSK_
+ 32)
  QPSK_CYC((160 * (count1 − 1) + 1):(count1 * 160))
                = [QPSK_IFFT(1,((i + 96):(i + 127))),QPSK_IFFT(1,(i:(i + 127)))];
  count1 = count1 + 1;
end

count2 = 1;
QAM_CYC = [];          % Initialize the QAM_CYC matrix
for i = 1:128:(Number_of_Bits/4)
  % Add cyclic prefix and append to QPSK_cyclic
  QAM_CYC((160 * (count2 − 1) + 1):(count2 * 160))
                = [QAM_IFFT(1,((i + 96):(i + 127))),QAM_IFFT(1,(i:(i + 127)))];
  count2 = count2 + 1;
end

% − − − − − − − − − − − − − − − − − − − − − − − − − − − Channel − − − − − − − − − − − − − − − − − − − − − − − %

% outputs from Rayleigh flat fading channel
QPSK_ch = Flat_channel(QPSK_CYC,SNR_range_db);
QAM_ch = Flat_channel(QAM_CYC,SNR_range_db);

% outputs from Frequency selective (FS) Fading channel
QPSK_ch_FS = FS_channel(QPSK_CYC,SNR_range_db);
QAM_ch_FS = FS_channel(QAM_CYC,SNR_range_db);

% ======================= At Reciever ========================== %

% − − − − − − − − − − − − − − − − − − − − − −De − Cyclic Extension − − − − − − − − − − − − − − − − − − − − − −%
QPSK_DeCYC = Decyclic(QPSK_ch,SNR_range_db);        % First case for QPSK [flat channel]
QPSK_DeCYC_FS
= Decyclic(QPSK_ch_FS,SNR_range_db);    % Second case for QPSK [frequency selective channel]
QAM_DeCYC = Decyclic(QAM_ch,SNR_range_db);          % Third case for QAM [flat channel]
QAM_DeCYC_FS
= Decyclic(QAM_ch_FS,SNR_range_db);     % Fourth case for QAM [frequency selective channel]

% − − − − − − − − − − − − − − − − −FFT − − − − − − − − − − − − − − − − −%
QPSK_FFT = FFT(QPSK_DeCYC,SNR_range_db);
QPSK_FFT_FS = FFT(QPSK_DeCYC_FS,SNR_range_db);
QAM_FFT = FFT(QAM_DeCYC,SNR_range_db);
QAM_FFT_FS = FFT(QAM_DeCYC_FS,SNR_range_db);

% − − − − − − − − − − − − − − − − −Demapper − − − − − − − − − − − − − − − − −%
QPSK_demapped = DeMapper(QPSK_FFT,SNR_range_db,2,Eb);
QPSK_demapped_FS = DeMapper(QPSK_FFT_FS,SNR_range_db,2,Eb);
QAM_demapped = DeMapper(QAM_FFT,SNR_range_db,4,Eb_QAM);
QAM_demapped_FS = DeMapper(QAM_FFT_FS,SNR_range_db,4,Eb_QAM);

% − − − − − − − − − − − − Deinterleaver − − − − − − − − − − − − − − − − − − −%
QPSK_Deintrlv = Deinterleaver(QPSK_demapped,SNR_range_db,2);
QPSK_FS_Deintrlv = Deinterleaver(QPSK_demapped_FS,SNR_range_db,2);
QAM_Deintrlv = Deinterleaver(QAM_demapped,SNR_range_db,4);
QAM_FS_Deintrlv = Deinterleaver(QAM_demapped_FS,SNR_range_db,4);

% − − − − − − − − − − − − −BER Calculation − − − − − − − − − − − − − − − − − − %
BER_QPSK = compute_BER(QPSK_Deintrlv ,Random_Data,SNR_range_db);
BER_QPSK_FS = compute_BER(QPSK_FS_Deintrlv ,Random_Data,SNR_range_db);
BER_QAM = compute_BER(QAM_Deintrlv ,Random_Data,SNR_range_db);
BER_QAM_FS = compute_BER(QAM_FS_Deintrlv ,Random_Data,SNR_range_db);


%%  − − − − − − − − − − − − − − − − QPSK & 16 − QAM with Repeatition coding − − − − − − − − − − − − −%%
```

```
QPSK_rep  =  SymbolRepetitionEncoder(Random_Data, 2);
QAM_rep  =  SymbolRepetitionEncoder(Random_Data, 4);

% − − − − − − − − − − − − − − − − − − − − − − − Interleaver (Repetition) − − − − − − − − − − − − − − − − − − − −%
for i = 1: 255: length(QPSK_rep)
   intrlv_out  =  matintrlv([QPSK_rep(1, i: i + 254),0],16,16);     % for QPSK Modulation with repeatiton
   if(i == 1)
    QPSK_rep_intrlv  =  intrlv_out;
   else
    QPSK_rep_intrlv  =  [QPSK_rep_intrlv intrlv_out];
   end
end

for i = 1: 510: length(QAM_rep)
   intrlv_out  =  matintrlv([QAM_rep(1, i: i + 509),0,0],32,16);     % for 16 − QAM Modulation with repeatition
   if(i == 1)
     QAM_rep_intrlv  =  intrlv_out;
   else
     QAM_rep_intrlv  =  [QAM_rep_intrlv intrlv_out];
   end
end

% − − − − − − − − − − − − − − − − − Mapper (Repetition) − − − − − − − − − − − − − − − − − −%
QPSK_rep_mapped  =  Mapper(QPSK_rep_intrlv, 2, Eb/sqrt(3));       % Same energy per info
QAM_rep_mapped  =  Mapper(QAM_rep_intrlv, 4, Eb_QAM/sqrt(3));      % Same energy per info

% − − − − − − − − − − − − − − − IFFT (Repetition) − − − − − − − − − − − − − − − − − −%
QPSK_rep_IFFT  =  zeros(1, length(QPSK_rep_mapped));
for i = 1: 128: length(QPSK_rep_mapped)
  QPSK_rep_IFFT(1, i: i + 127)  =  ifft(QPSK_rep_mapped(1, i: i + 127),128);
end

QAM_IFFT  =  zeros(1, length(QAM_rep_mapped));
for i = 1: 128: length(QAM_rep_mapped)
  QAM_rep_IFFT(1, i: i + 127)  =  ifft(QAM_rep_mapped(1, i: i + 127),128);
end

% − − − − − − − − Cyclic Extension (Repetition) − − − − − − − − − − − − − − − %
count3  =  1;
QPSK_rep_CYC  =  [];           % Initialize the QPSK_CYC matrix
for i  =  1: 128: length(QPSK_rep_mapped)
  QPSK_rep_CYC((160 ∗ (count3  −  1)  +  1): (count3  ∗  160))
                     =  [QPSK_rep_IFFT(1, ((i  +  96): (i  +  127))), QPSK_rep_IFFT(1, (i: (i  +  127)))];
  count3  =  count3  +  1;
end

count4  =  1;
QAM_rep_CYC  =  [];
for i  =  1: 128: length(QAM_rep_mapped)
  QAM_rep_CYC((160  ∗  (count4  −  1)  +  1): (count4  ∗  160))
                     =  [QAM_rep_IFFT(1, ((i  +  96): (i  +  127))), QAM_rep_IFFT(1, (i: (i  +  127)))];
  count4  =  count4  +  1;
end

% − − − − − − − − − − − − − − − − − − −Channel − − − − − − − − − − − − − − − − − − − − − − − − − %
% outputs from Rayleigh flat fading channel
QPSK_rep_ch  =  Flat_channel(QPSK_rep_CYC, SNR_range_db);    % for QPSK
QAM_rep_ch  =  Flat_channel(QAM_rep_CYC, SNR_range_db);      % for QAM

% outputs from Frequency selective (FS) Fading channel
QPSK_rep_ch_FS  =  FS_channel(QPSK_rep_CYC, SNR_range_db);   % for QPSK
QAM_rep_ch_FS  =  FS_channel(QAM_rep_CYC, SNR_range_db);     % for QAM
```

```
% ==================== At Reciever ======================== %

% −−−−−−−−−−−−−−−De − Cyclic Extension −−−−−−−−−−−−−−−−%
% First case for QPSK [flat channel]
QPSK_rep_DeCYC  =  Decyclic(QPSK_rep_ch,SNR_range_db);
% Second case for QPSK [frequency selective channel]
QPSK_rep_DeCYC_FS  =  Decyclic(QPSK_rep_ch_FS,SNR_range_db);
% Third case for QAM [flat channel]
QAM_rep_DeCYC  =  Decyclic(QAM_rep_ch,SNR_range_db);
% Fourth case for QAM [frequency selective channel]
QAM_rep_DeCYC_FS  =  Decyclic(QAM_rep_ch_FS,SNR_range_db);

% −−−−−−−−−−−−−−−−−− FFT (Repitition) −−−−−−−−−−−−−−−−−−−−%
QPSK_rep_FFT  =  FFT(QPSK_rep_DeCYC,SNR_range_db);       % QPSK [Flat − Fading]
QPSK_rep_FFT_FS  =  FFT(QPSK_rep_DeCYC_FS,SNR_range_db);  % QPSK [Frequency Slective]

QAM_rep_FFT  =  FFT(QAM_rep_DeCYC,SNR_range_db);         % QAM [Flat − Fading]
QAM_rep_FFT_FS  =  FFT(QAM_rep_DeCYC_FS,SNR_range_db);    % QAM [Frequency Slective]

% −−−−−−−−−−−−−−−−−Demapper (Repitition) −−−−−−−−−−−−−−−−%
QPSK_rep_demapped  =  DeMapper(QPSK_rep_FFT,SNR_range_db,2,Eb);          % QPSK [Flat − Fading]
QPSK_rep_demapped_FS
                =  DeMapper(QPSK_rep_FFT_FS,SNR_range_db,2,Eb);       % QPSK [Frequency Slective]

QAM_rep_demapped  =  DeMapper(QAM_rep_FFT,SNR_range_db,4,Eb_QAM);         % QAM [Flat − Fading]
QAM_rep_demapped_FS
                =  DeMapper(QAM_rep_FFT_FS,SNR_range_db,4,Eb_QAM);     % QAM [Frequency Slective]

% −−−−−−−−−−−−Deinterleaver(Repitition) −−−−−−−−−−−−−−−%
QPSK_rep_Deintrlv  =  Deinterleaver(QPSK_rep_demapped,SNR_range_db,2);
QPSK_rep_FS_Deintrlv  =  Deinterleaver(QPSK_rep_demapped_FS,SNR_range_db,2);
QAM_rep_Deintrlv  =  Deinterleaver(QAM_rep_demapped,SNR_range_db,4);
QAM_rep_FS_Deintrlv  =  Deinterleaver(QAM_rep_demapped_FS,SNR_range_db,4);

% Remove the padding done at the interleaving stage at the Tx
QPSK_rep_Deintrlv  =  pad_removal(QPSK_rep_Deintrlv,2);
QPSK_rep_FS_Deintrlv  =  pad_removal(QPSK_rep_FS_Deintrlv,2);
QAM_rep_Deintrlv  =  pad_removal(QAM_rep_Deintrlv,4);
QAM_rep_FS_Deintrlv  =  pad_removal(QAM_rep_FS_Deintrlv,4);

% −−−−−−−−−−− Majority Decoding −−−−−−−−−−−−−−−−−−%
for i = 1:length(SNR_range_db)
  QPSK_rep_decoded(i,:)  =  majority_decoder(QPSK_rep_Deintrlv(i,:),2);
  QPSK_rep_FS_decoded(i,:)  =  majority_decoder(QPSK_rep_FS_Deintrlv(i,:),2);
  QAM_rep_decoded(i,:)  =  majority_decoder(QAM_rep_Deintrlv(i,:),4);
  QAM_rep_FS_decoded(i,:)  =  majority_decoder(QAM_rep_FS_Deintrlv(i,:),4);
end

% −−−−−−−−−−−−BER Calculation (Repitition) −−−−−−−−−−−−−%
BER_QPSK_rep  =  compute_BER(QPSK_rep_decoded ,Random_Data,SNR_range_db);
BER_QPSK_rep_FS  =  compute_BER(QPSK_rep_FS_decoded,Random_Data,SNR_range_db);
BER_QAM_rep  =  compute_BER(QAM_rep_decoded ,Random_Data,SNR_range_db);
BER_QAM_rep_FS  =  compute_BER(QAM_rep_FS_decoded ,Random_Data,SNR_range_db);


% −−−−−−−−−−−−Plotting BER vs SNR(Eb/No) −−−−−−−−−−−−−−− %
figure;
semilogy(SNR_range_db ,BER_QPSK ,'k','linewidth',1.5);
hold on;
semilogy(SNR_range_db ,BER_QPSK_rep ,'r','linewidth',1.5);
xlabel('Eb/No (dB)');
grid on;
```

```
title('BER of Flat fading channel QPSK');
legend('without coding','with Repitition coding','location','southwest')

figure;
semilogy(SNR_range_db ,BER_QPSK_FS ,'k','linewidth',1.5);
hold on;
semilogy(SNR_range_db ,BER_QPSK_rep_FS ,'r','linewidth',1.5);
xlabel('Eb/No (dB)')
grid on;
title('BER of frequency selective channel QPSK');
legend('without coding','with Repitition coding','location','southwest')

figure;
semilogy(SNR_range_db ,BER_QAM ,'k','linewidth',1.5);
hold on;
semilogy(SNR_range_db ,BER_QAM_rep ,'r','linewidth',1.5);
xlabel('Eb/No (dB)');
grid on;
title('BER of Flat fading channel 16 − QAM');
legend('without coding','with Repitition coding','location','southwest')

figure;
semilogy(SNR_range_db ,BER_QAM_FS ,'k','linewidth',1.5);
hold on;
semilogy(SNR_range_db ,BER_QAM_rep_FS ,'r','linewidth',1.5);
xlabel('Eb/No (dB)');
grid on;
title('BER of frequency selective channel 16 − QAM');
legend('without coding','with Repitition coding','location','southwest')


%% − − − − − − − − − − − − − − − − − − Repeatition encoder − − − − − − − − − − − − − − − − − %%
function encodedStream = SymbolRepetitionEncoder(inputStream, symbolBits)
  % Reshape the input stream into symbols
  numSymbols = length(inputStream) / symbolBits;
  symbols = reshape(inputStream, symbolBits, numSymbols)';

  % Repeat each symbol 3 times
  repeatedSymbols = repelem(symbols, 3, 1);

  % Flatten the repeated symbols back into a binary stream
  encodedStream = repeatedSymbols';
  encodedStream = encodedStream(:)';
end

%% − − − − − − − − − − −Mapper function − − − − − − − − − − − − − − −%%
function mapped_constellations = Mapper(inputstream, symbolBits, Eb)        % Eb represents the info energy
Number_of_Bits = length(inputstream);
mapped_constellations = zeros(1, Number_of_Bits/symbolBits);

switch symbolBits
  case 2          % QPSK mapping
    for i = 1: symbolBits: Number_of_Bits
      if (inputstream(i) == 0 && inputstream(i + 1) == 0 )
        mapped_constellations((i + 1)/2) = sqrt(Eb) ∗ complex(−1, −1);
      elseif (inputstream(i) == 0 && inputstream(i + 1) == 1)
        mapped_constellations((i + 1)/2) = sqrt(Eb) ∗ complex(−1,1);
      elseif (inputstream(i) == 1 && inputstream(i + 1) == 0)
        mapped_constellations((i + 1)/2) = sqrt(Eb) ∗ complex(1, −1);
      elseif (inputstream(i) == 1 && inputstream(i + 1) == 1)
        mapped_constellations((i + 1)/2) = sqrt(Eb) ∗ complex(1,1);
      end
    end
  case 4          % 16 − QAM mapping
    real_MQAM  =  zeros(1, length(inputstream)/symbolBits);
    img_MQAM  =  zeros(1, length(inputstream)/symbolBits);
    for i  =  1: symbolBits: Number_of_Bits
      if (inputstream(i)  ==  0 && inputstream(i + 1)  ==  0)         % First two bits control the real part
        real_MQAM((i + 3)/4)  =  −3;
      elseif (inputstream(i)  ==  0 && inputstream(i + 1)  ==  1)
        real_MQAM((i + 3)/4)  =  −1;
      elseif (inputstream(i)  ==  1 && inputstream(i + 1)  ==  1)
        real_MQAM((i + 3)/4)  =  1;
      elseif (inputstream(i)  ==  1 && inputstream(i + 1)  ==  0)
        real_MQAM((i + 3)/4)  =  3;
      end
      if (inputstream(i + 2)  ==  0 && inputstream(i + 3)  ==  0)        % Second two bits control the imaginary
        img_MQAM((i + 3)/4)  =  −3;
      elseif (inputstream(i + 2)  ==  0 && inputstream(i + 3)  ==  1)
        img_MQAM((i + 3)/4)  =  −1;
      elseif (inputstream(i + 2)  ==  1 && inputstream(i + 3)  ==  1)
        img_MQAM((i + 3)/4)  =  1;
      elseif (inputstream(i + 2)  ==  1 && inputstream(i + 3)  ==  0)
        img_MQAM((i + 3)/4)  =  3;
      end
    mapped_constellations((i + 3)/4)  =  sqrt(Eb)  ∗  complex(real_MQAM((i + 3)/4), img_MQAM((i + 3)/4));
    end
end

end

%% − − − − − − − − − Rayleigh flat fading channel Function − − − − − − − − − − − − − −%%
function Received_data  =  Flat_channel(cyclic_input, SNR_range_db)

SNR_range_linear  =  zeros(1, length(SNR_range_db));
Received_data  =  zeros(length(SNR_range_db), length(cyclic_input));
  for i  =  1: length(SNR_range_db)
    SNR_range_linear(i)  =  10 ^ (SNR_range_db(i)/10);
    for j  =  1 :  160 :  length(cyclic_input)
      Eavg  =  mean(abs(cyclic_input(j: j + 159)). ^2);
      h_channel  =  (1/sqrt(2)) .∗  complex(randn(1,160) , randn(1,160));
      noise  =  sqrt(Eavg/(2 ∗ SNR_range_linear(i))) .∗  complex(randn(1,160) , randn(1,160));
      Received_data(i, j: j + 159)  =  (cyclic_input(j: j + 159) .∗  h_channel  +  noise) ./ h_channel;
    end
  end
end

%%  − − − − − − − − − − − − −Frequency selective fading channel Function − − − − − − − − − − −%%
function Received_data  =  FS_channel(cyclic_input, SNR_range_db)
Cyclic_symbols  =  160;
Number_of_subchannels  =  2;
channel_length  =  Cyclic_symbols/Number_of_subchannels;
gain  =  [0.5,0.5];  % FS gain of the subchannels (2 subchannels) (equal channel assignment)
% Each Rayleigh subchannel is scaled with different gain
ch1_time  =  gain(1) ∗ (1/sqrt(2))  ∗  complex(randn(1, channel_length), randn(1, channel_length));
ch2_time  =  gain(2) ∗ (1/sqrt(2))  ∗  complex(randn(1, channel_length), randn(1, channel_length));
data_due_channel  =  zeros(1, (length(cyclic_input)/channel_length) ∗ (2 ∗ Cyclic_symbols − 2));

indx  =  1;
SNR_range_linear  =  zeros(1, length(SNR_range_db));
% Received_data  =  zeros(length(SNR_range_db), length(cyclic_input));
  for i =  1: length(SNR_range_db)
    SNR_range_linear(i)  =  10 ^ (SNR_range_db(i)/10);
    for j  =  1: channel_length: length(cyclic_input)
      if (mod(indx, 2) == 1) % Assign to channel 1
        data_due_channel(j: j + channel_length − 1)  =  cyclic_input(j: j + channel_length − 1) .∗  ch1_time;
```

```
            Eavg  =  mean(abs(cyclic_input(j: j + channel_length − 1)). ^2);
            noise  =  sqrt(Eavg/(2 ∗ SNR_range_linear(i))) .
                       ∗ complex(randn(1, channel_length) , randn(1, channel_length));
            Received_data(i, j: j + channel_length − 1)
                       =  (data_due_channel(j: j + channel_length − 1)  +  noise) ./ ch1_time;
        else           % Assign to channel 2
            data_due_channel(j: j + channel_length − 1)  =  cyclic_input(j: j + channel_length − 1) .∗  ch2_time;
            Eavg  =  mean(abs(cyclic_input(j: j + channel_length − 1)). ^2);
            noise  =  sqrt(Eavg/(2 ∗ SNR_range_linear(i))) .
                       ∗ complex(randn(1, channel_length) , randn(1, channel_length));
            Received_data(i, j: j + channel_length − 1)
                       =  (data_due_channel(j: j + channel_length − 1)  +  noise) ./ ch2_time;
        end
        indx  =  indx  + 1;
      end
   end
end

%% − − − − − − − − − − − Decyclic extension Function − − − − − − − − − − − − − − − − − −%%
function decyclic_out  =  Decyclic(input_data, SNR_range_db)

%decyclic_out  =  zeros(length(SNR_range_db), (Number_of_Bits/symbolBits));
% decyclic_out  =  zeros(length(SNR_range_db), size(input_data, 2));
   for j = 1: length(SNR_range_db)
      for i = 160: 160: size(input_data, 2)
         decyclic_out(j, ((i/160) ∗ 128) − 127: (i/160) ∗ 128) = input_data(j, i − 127: i);
      end
   end
end

%% − − − − − − − − − − − − − − − −FFT Function (At Rx) − − − − − − − − − − − − − − − −%%
function out_data_fft  =  FFT(input_data, SNR_range_db)

out_data_fft  =  zeros(length(SNR_range_db), size(input_data, 2));       % Initialize output array

   for j  =  1 :  length(SNR_range_db)
      for i  =  1: 128: size(input_data, 2)
         out_data_fft(j, i: i + 127)  =  fft(input_data(j, i: i + 127), 128);
      end
   end
end

%% − − − − − − − − − − − − − DeMapper Function − − − − − − − − − − − − − − − − − − − −%%
function demapped_data_binary  =  DeMapper (received_data , SNR_range_db, symbolBits, Eb)

demapped_data_binary  =  zeros(length(SNR_range_db), size(received_data, 2) ∗ symbolBits);
demapped_data_decimal  =  zeros(length(SNR_range_db), size(received_data, 2));

   for i  =  1 :  length(SNR_range_db)
      switch symbolBits
         case 2     % QPSK case
            QPSK_table  =  sqrt(Eb)  ∗  [complex(−1, −1), complex(−1,1), complex(1, −1), complex(1,1)];
            for j  =  1 :  size(received_data, 2)
               [~ , Min_index]  =  min(abs(received_data(i, j)  −  QPSK_table));
               demapped_data_decimal(i, j)  =  Min_index  −  1;
            end
            demapped_data_binary(i, : )  =  reshape(de2bi(demapped_data_decimal(i, : ),2, 'left − msb')', 1, []);
         case 4     % 16 − QAM case
            MQAM_table
=  sqrt(Eb)
∗ [complex(−3, −3), complex(−3, −1), complex(−3,3), complex(−3,1), complex(−1, −3), complex(−1, −1),
complex(−1,3), complex(−1,1), complex(3, −3), complex(3, −1), complex(3,3), complex(3,1), complex(1, −3),
complex(1, −1), complex(1,3), complex(1,1)];
```

```
        for j  =  1 :  size(received_data, 2)
          [~ , Min_index]  =  min(abs(received_data(i, j)  −  MQAM_table));
          demapped_data_decimal(i, j)  =  Min_index  −  1;
        end
        demapped_data_binary(i, : )  =  reshape(de2bi(demapped_data_decimal(i, : ),4, 'left − msb')', 1, []);
    end
  end
end

%% − − − − − − − − − − − − − − − − Deinterleaver Function − − − − − − − − − − − − − − − − − − %%
function deintrlv_out  =  Deinterleaver(demapped_data, SNR_range_db, symbolBits)

deintrlv_out  =  zeros(length(SNR_range_db), size(demapped_data, 2));      % Initialize the output

  for j  =  1 :  length(SNR_range_db)
    for i  =  1: (128 ∗ symbolBits): size(demapped_data, 2)
      deintrlv_out(j, i: i + (128 ∗ symbolBits) − 1)
                 =  matdeintrlv(demapped_data(j, i: i + (128 ∗ symbolBits) − 1), symbolBits ∗ 8, 16);
    end
  end
end

%% − − − − − − − − − − − − − − −Pad Removal function (In case of Repition code) − − − − − − − − − − − − %%
function Deintrlv_final  =  pad_removal(Deintrlv_output, symbolBits)

  for i  =  1: size(Deintrlv_output, 1)
    k = 1;
    if(symbolBits  ==  2) % QPSK  − −−>  every 256 bits remove 1 bit from the end
      for j   =  1: 256: size(Deintrlv_output, 2)
        frame  =  Deintrlv_output(i, j: j + 255);
        Deintrlv_final(i, k: k + 254)  =  frame(1: end − 1);
        k = k + 255;
      end
    else % 16 − QAM  − −−>  every 512 bits remove 2 bits from the end
      for j   =  1: 512: size(Deintrlv_output, 2)
        frame  =  Deintrlv_output(i, j: j + 511);
        Deintrlv_final(i, k: k + 509)  =  frame(1: end − 2);
        k = k + 510;
      end
    end
  end
end

%% − − − − − − − − − − − −  Decoding Function  − − − − − − − − − − − − − − − − − %%
function [decoded_bitstream]  =  majority_decoder (encoded_bitstream , symbolBits )
  hash_table  =  zeros(1,2^symbolBits);
  iterations  =  0;
  j = 1;
  decoded_bitstream  =  zeros(1, length(encoded_bitstream)/3); %% Assume 3 − repeatition code
  for i  =  1 :  symbolBits :  length(encoded_bitstream)
    iterations  =  iterations  +  1 ;
    % Convert the symbol into the correspoding decimal into a hashtable
    % to see the no. of occurrences of such symbol for decision
    index  =  bin2dec(num2str((encoded_bitstream(i: i + symbolBits − 1)))) + 1;
    hash_table(index)  =  hash_table(index)  +  1;
    if(iterations == 3)
       %decide based on the most no. of occurrences
       % then get the corrsponding binary symbol
       index_of_most_repeated_symbol = find(hash_table == max(hash_table),1);
       binaryStr  =  dec2bin(index_of_most_repeated_symbol − 1, symbolBits);
       decoded_bitstream(j: j + symbolBits − 1)  =  double(binaryStr) − '0';
       j  =  j  +  symbolBits;
       hash_table  =  zeros(1,2^symbolBits); % Reset the hash table for the next set of symbols
      iterations = 0;
    end
  end
end

%% ---------------BER Calculation function----------------%%
function BER_value = compute_BER(received_data, Random_data, SNR_range_db)

BER_value = zeros(1, length(SNR_range_db));
  for i = 1 : length(SNR_range_db)
    BER_value(1, i) = length(find(xor(received_data(i,:), Random_data) == 1))/length(Random_data);
  end
end
```

- After simulating the above code, we produce 4 curves as we have four possibilities (QPSK with flat channel, QPSK with frequency selective channel, 16-QAM with flat channel, 16-QAM with frequency selective channel)

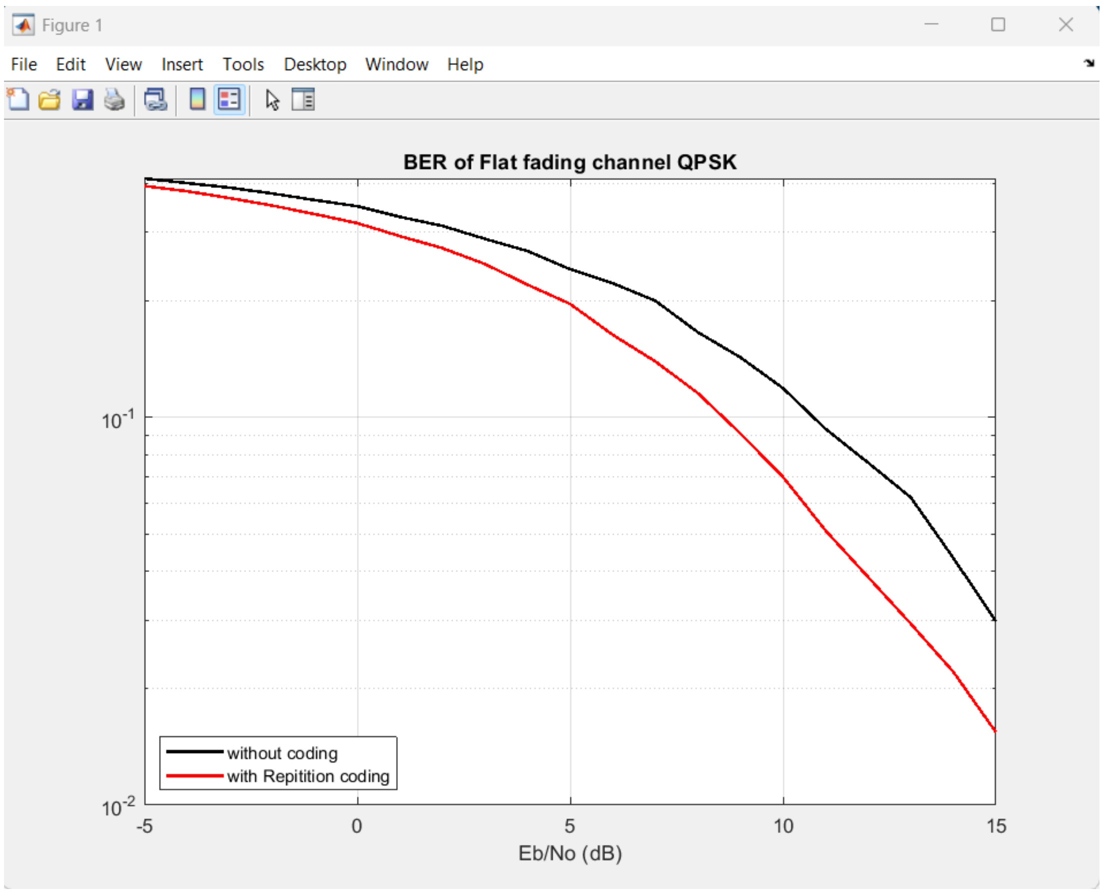

Figure (7): BER of flat fading channel for QPSK modulation

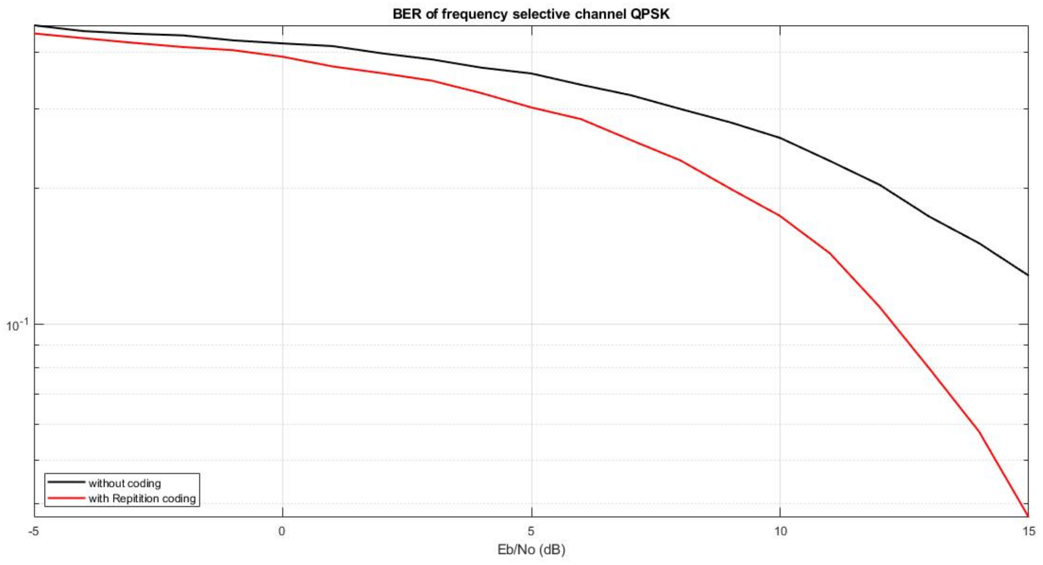

Figure (8): BER of frequency selective fading channel for QPSK modulation

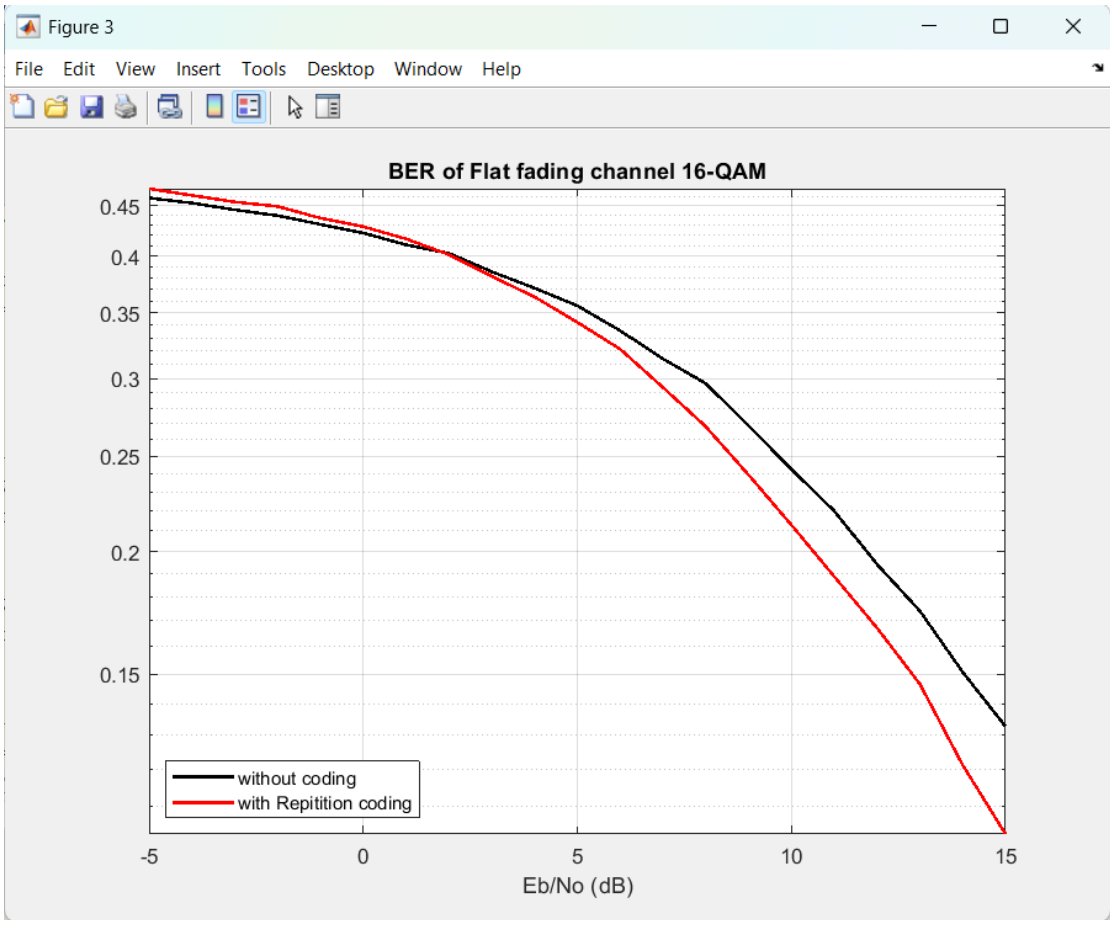

Figure (9): BER of flat fading channel for 16-QAM modulation

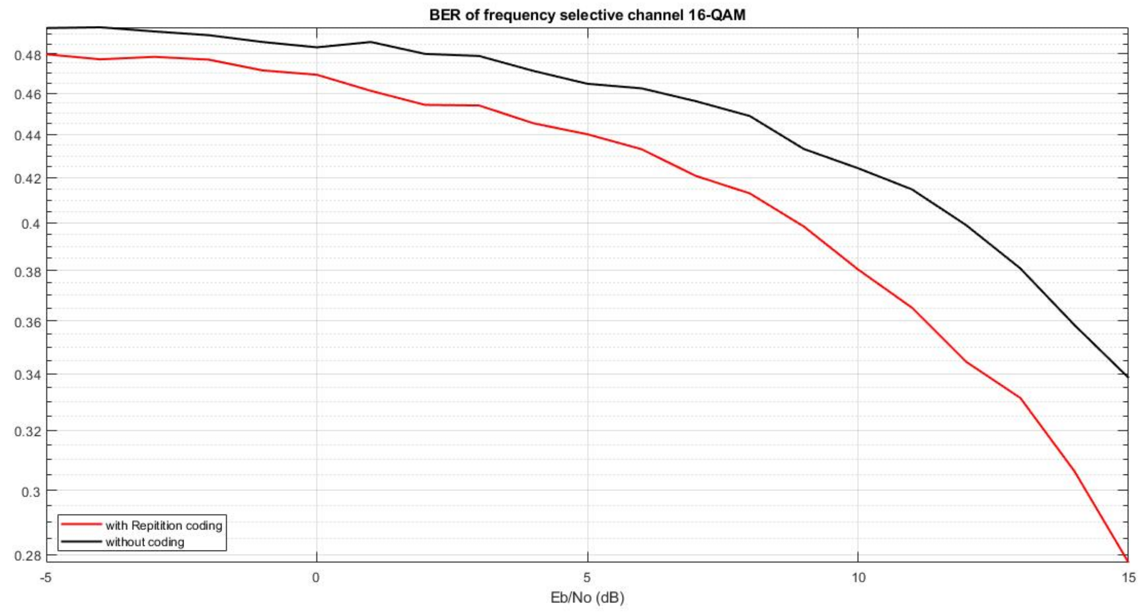

Figure (10): BER of frequency selective fading channel for 16-QAM modulation

✓ From the above graphs we notice that:

1. **Channel vs. Modulation Scheme:**

   Flat Fading Channel: For both QPSK and 16-QAM, the flat fading channel performs better overall compared to the frequency-selective channel. This is expected as flat fading introduces uniform distortion across all frequencies, which is easier to equalize.

   Frequency-Selective Channel: This channel shows a more significant degradation in BER, especially for higher-order modulation (16-QAM). Frequency selectivity causes varying interference across different frequency components, making equalization harder and increasing BER.

2. **Effect of Coding with the Modulation Scheme on BER:**

   QPSK: Repetition coding demonstrates noticeable improvement in BER for both flat and frequency-selective channels. This shows the resilience of lower-order modulation when combined with simple error correction techniques like repetition coding.

   16-QAM: Repetition coding still improves the BER, but the improvement is less pronounced compared to QPSK. This reflects the inherent vulnerability of higher-order modulation to noise and fading, which limits the coding gain.

3. **Insights on Modulation and Coding:**

   Higher-Order Modulation: As seen in 16-QAM, while it provides higher data rates, it is more sensitive to noise and fading. This trade-off makes it less effective in harsh channel conditions, even with coding.

   Error-Correcting Codes: Simple coding techniques like repetition coding offer significant BER improvements, particularly for robust modulation schemes like QPSK. However, advanced coding methods may be needed for higher-order schemes to maximize performance gains.

4. **General Observations:**

   QPSK offers better BER under challenging conditions, 16-QAM provides higher data rates but demands more sophisticated error-correction strategies.