CAIRO UNIVERSITY
FACULTY OF ENGINEERING
ELECTRONICS & ELECT. COMM. DEPT

COMMUNICATIONS
FOURTH YEAR
FIRST SEMESTER 2024-2025

ELC4020 - Advanced Communication Systems

OFDM Report

Presented by:

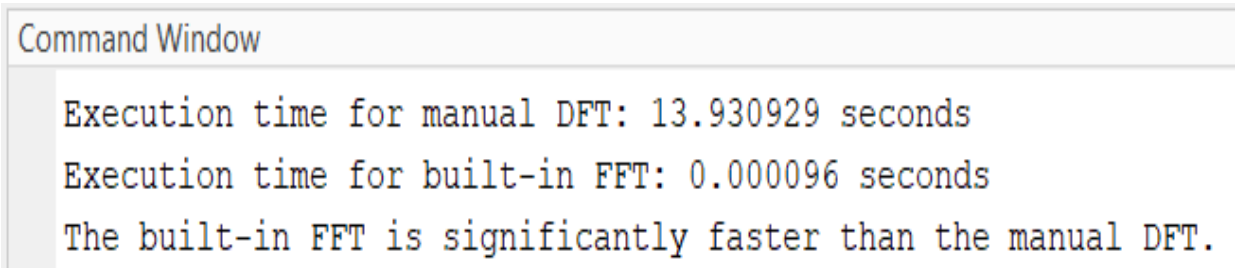
Name	Section	B.N.	ID
Mariam Ahmed Kamal Mamdouh	4	17	9211131
Yasmin Hany Momtaz Abo El-Magd	4	40	9203721

Presented to:

Dr. Mohammed Khairy
Eng. Mohamed Nady

Problem 1: Execution time of DFT and FFT

1. The output



2. MATLAB code

```
%% Problem 1: Execution time of DFT and FFT
clc;
clear all;
close all;

% Part (a): Implement DFT directly
L = 8192; % Length of the signal
xi = rand(1, L); % Generate random test signal

% Initialize DFT result
N = length(xi);
X_dft = zeros(1, N); % Preallocate result array for DFT

% Start timing for DFT
tic;

for k = 0:N-1
    for n = 0:N-1
        X_dft(k+1) = X_dft(k+1) + xi(n+1) * exp(-1j * 2 * pi * k * n
/ N);
    end
end
time_dft = toc; % Stop timing for DFT

% Part (b): Compute FFT using built-in MATLAB function
tic; % Start timing for FFT
X_fft = fft(xi);
time_fft = toc; % Stop timing for FFT

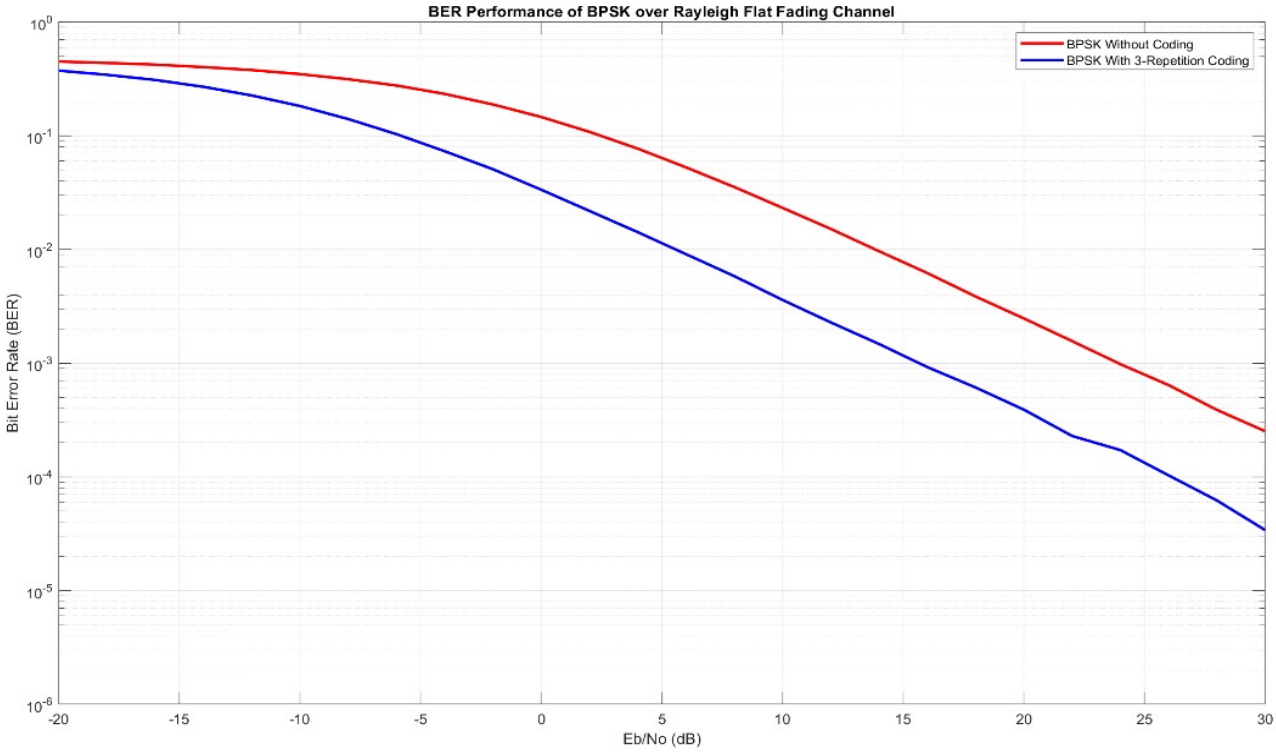
% Display results
fprintf('Execution time for manual DFT: %.6f seconds\n', time_dft);
fprintf('Execution time for built-in FFT: %.6f seconds\n',
time_fft);
```

3. Comment

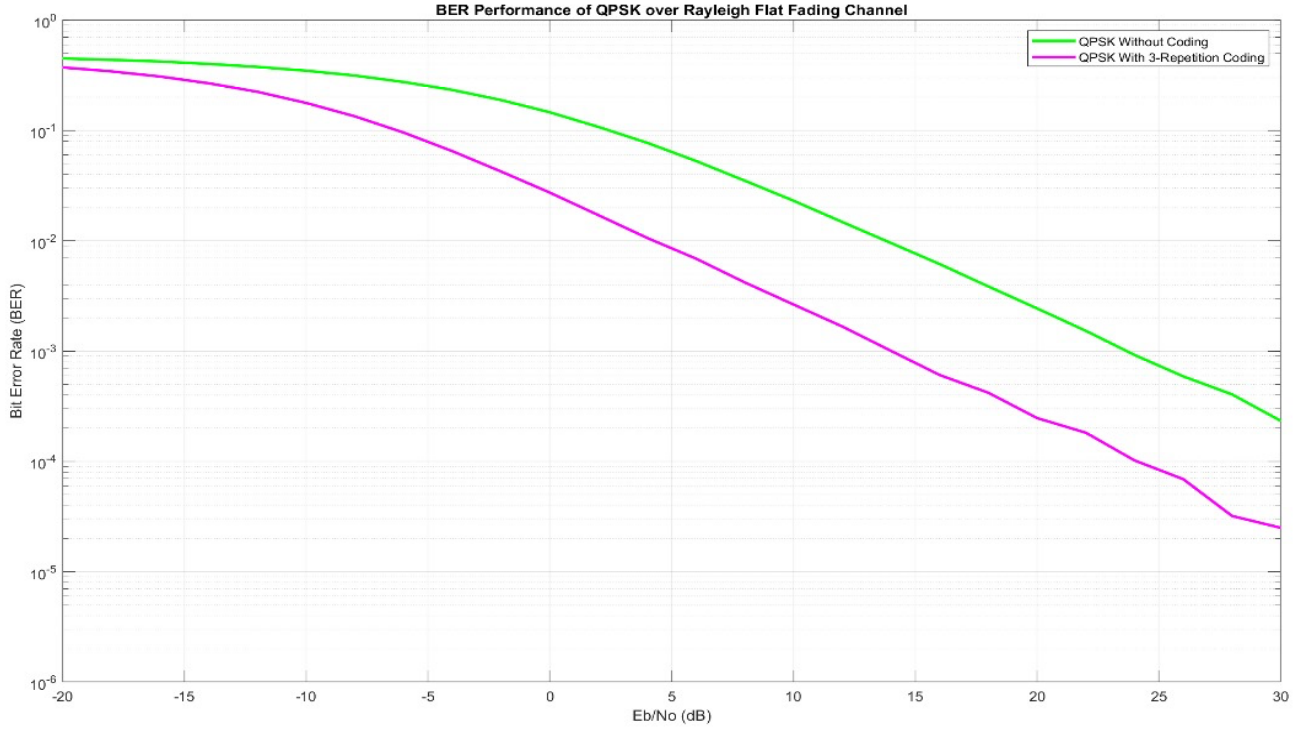
- Obviously FFT take much less time than DFT function , that's due to using nested summations and the original Fourier Transform definition in DFT , while in FFT symmetry and periodicity properties of sinusoidal functions are used which helps so much with computation speed.

Problem 2: Bit-error rate performance for BPSK and 16-QAM over Rayleigh Flat Fading Channel

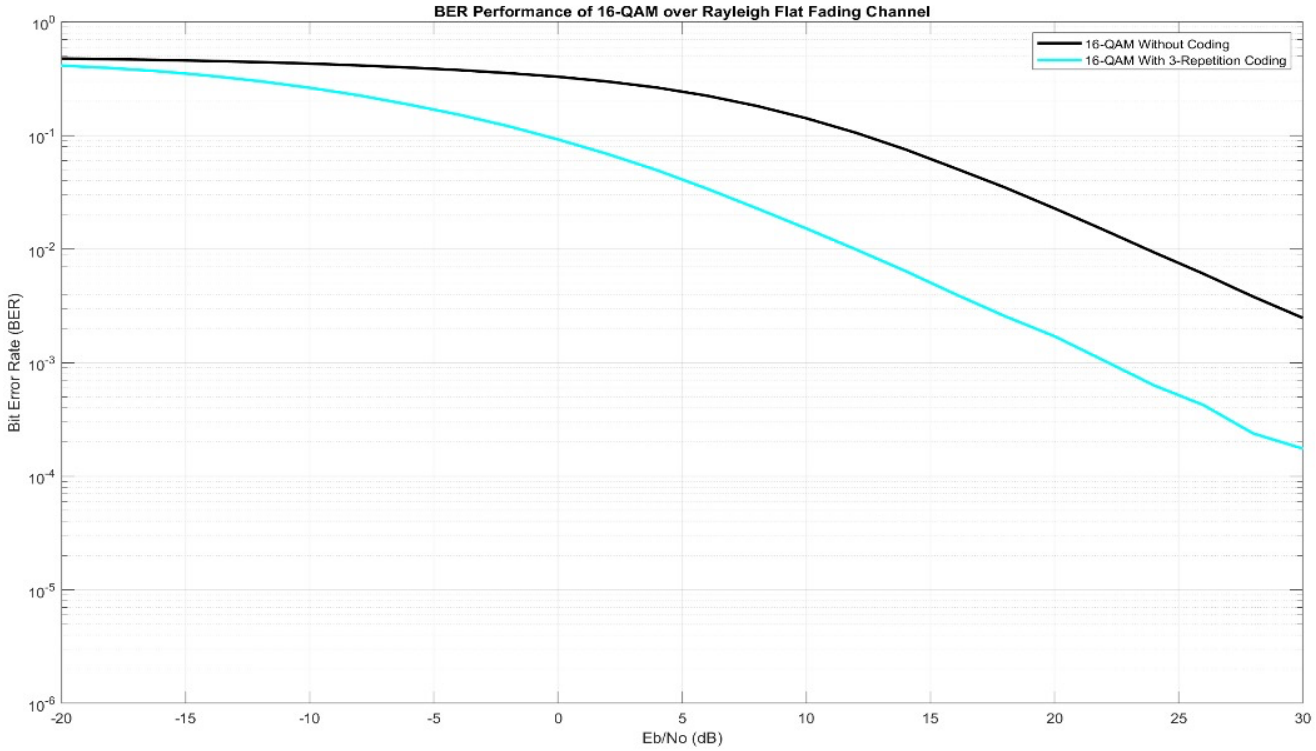
1. **The output**
   A. **BER vs SNR for BPSK without and with 3-repetition coding**



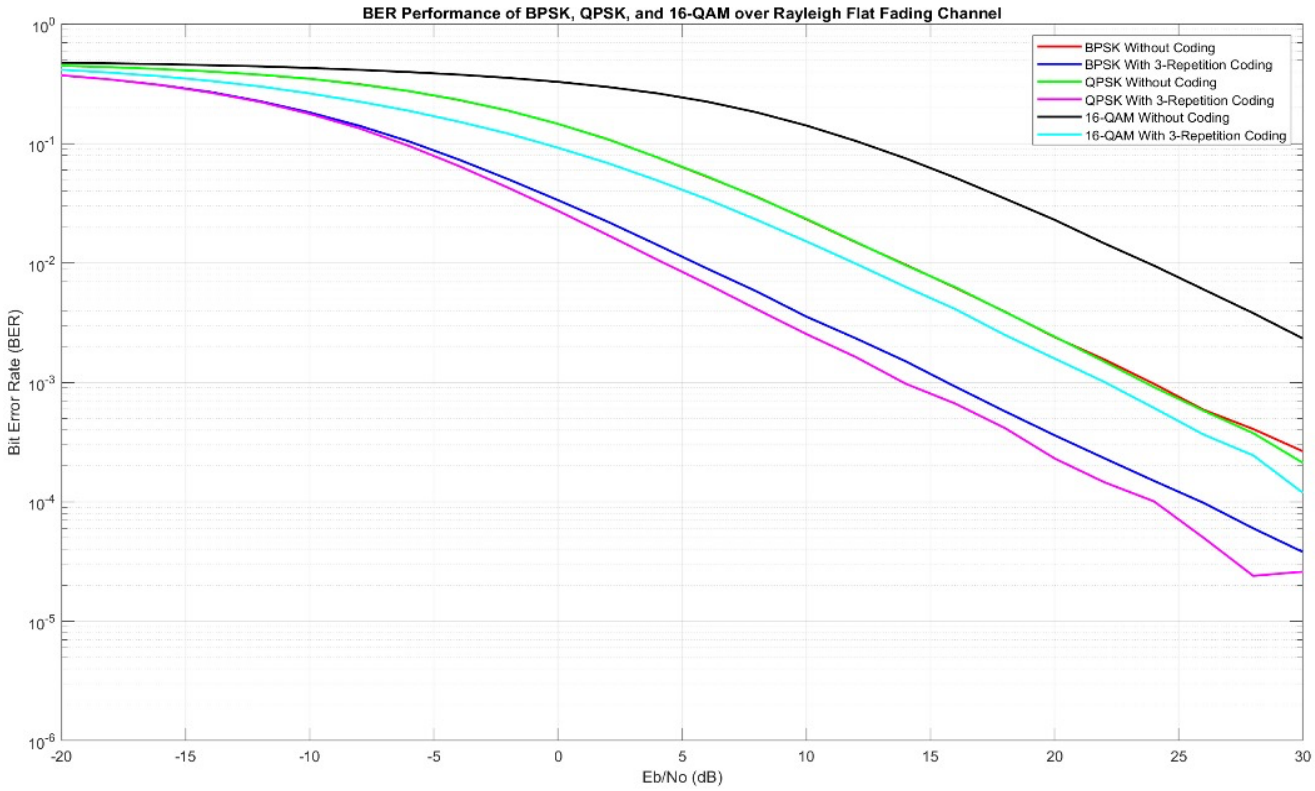
   B. **BER vs SNR for QPSK without and with 3-repetition coding**



C. BER vs SNR for 16-QAM without and with 3-repetition coding



D. BER vs SNR for BPSK, QPSK, and 16-QAM without and with 3-repetition coding



2. MATLAB code

```
%% Problem 2: BER Performance Simulation for BPSK, QPSK, and 16-QAM
clc;
clear all;
close all;

%% Simulation parameters
bits_num = 1000000;  % Number of bits
Eb_No_dB = -20:2:30;  % Eb/No (SNR) range in dB
Eb = 1;  % Energy per Bit
Eb_rep = Eb/3;  % Energy per bit for 3-repetition case

%% Converting Eb/No from dB to linear
Eb_No_linear = 10.^((Eb_No_dB)/10);

%% Random Bit Stream Generation
bk = randi([0 1], 1, bits_num);  % Generate a random binary
sequence of 0s and 1s

%% Repetition parameters
rep_factor = 3;   % Repetition factor (3 times for each bit)
bits_num_rep = rep_factor * bits_num;
bk_coded = repelem(bk, rep_factor);

%% Rayleigh Flat Fading Channel Generation
sigma = sqrt(1/2);  % Standard deviation for Gaussian components
(variance = 1/2)
hr = sigma * randn(1, bits_num);  % Real part (Gaussian with mean
0, variance 1/2)
hi = sigma * randn(1, bits_num);  % Imaginary part (Gaussian with
mean 0, variance 1/2)
h = hr + 1i * hi;  % Complex channel impulse response

%% Rayleigh Flat Fading Channel Generation for 3-repetition coding
h_coded = repelem(h, 3);

%% BER calculations
BER_BPSK_without_coding = zeros(size(Eb_No_dB));
BER_BPSK_with_coding = zeros(size(Eb_No_dB));
BER_QPSK_without_coding = zeros(size(Eb_No_dB));
BER_QPSK_with_coding = zeros(size(Eb_No_dB));
BER_16QAM_without_coding = zeros(size(Eb_No_dB));
BER_16QAM_with_coding = zeros(size(Eb_No_dB));


%% Case(a): BPSK without coding %%
for i = 1:length(Eb_No_dB)
    %% The Mapper (BPSK modulation)
    xk_BPSK = 2 * bk - 1; % BPSK mapping: 0 -> -1, 1 -> 1
```

```
% Noise variance calculation based on Eb/No
    No = Eb / Eb_No_linear(i);

    % Generate AWGN noise
    noise = sqrt(No / 2) * (randn(1, bits_num) + 1j * randn(1,
bits_num));

    % Transmit through Rayleigh Flat Fading Channel
    yk_BPSK = xk_BPSK .* h + noise;

    %% The Receiver: Channel inversion
    % Apply channel inversion (Assuming the channel is known at the
receiver)
    xk_hat_BPSK = yk_BPSK ./ h; % Inverse channel equalization

    % The Demapper: Hard Decision Decoding
    bk_hat_BPSK = real(xk_hat_BPSK) > 0; % Decision based on
threshold (0)

    %% Compute BER without coding
    errors = sum(bk_hat_BPSK ~= bk);  % Count bit errors
    BER_BPSK_without_coding(i) = errors / bits_num;  % Simulated
BER
end

%% Case(b): BPSK with 3-repetition coding %%
for i = 1:length(Eb_No_dB)
    %% The Mapper (BPSK modulation)
    xk_BPSK_coded = 2 * bk_coded - 1;

    %% Generating and Adding AWGN
    % Noise variance calculation based on Eb/No
    No = Eb_rep / Eb_No_linear(i);

    % Generate AWGN noise
    noise_coded = sqrt(No / 2) * (randn(1, bits_num_rep) + 1j *
randn(1, bits_num_rep));

    % Transmit through Rayleigh Flat Fading Channel
    yk_BPSK_coded = xk_BPSK_coded .* h_coded + noise_coded;

    %% The Receiver: Channel inversion
    % Apply channel inversion (Assuming the channel is known at the
receiver)
    xk_hat_BPSK_coded = yk_BPSK_coded ./ h_coded; % Inverse channel
equalization

    % The Demapper: Hard Decision Decoding
    bk_hat_coded = real(xk_hat_BPSK_coded) > 0; % Decision based on
threshold (0)
```

```
    %% Majority Decoding for 3-repetition coding
    bk_hat_decoded = sum(reshape(bk_hat_coded, rep_factor,
bits_num), 1) >= 2; % Majority vote decoding

    %% Compute BER with 3-repetition coding
    errors_coded = sum(bk_hat_decoded ~= bk);  % Count bit errors
    BER_BPSK_with_coding(i) = errors_coded / bits_num;  % Simulated
BER
end

%% Case(c): QPSK without coding %%
for i = 1:length(Eb_No_dB)
    %% The Mapper (QPSK modulation with Gray coding)
    qpsk_symbols = (2*bk(1:2:end) - 1) + 1j*(2*bk(2:2:end) - 1);

    % Noise variance calculation based on Eb/No
    No = Eb / Eb_No_linear(i);

    % Generate AWGN noise
    noise = sqrt(No / 2) * (randn(1, bits_num/2) + 1j * randn(1,
bits_num/2));

    % Transmit through Rayleigh Flat Fading Channel
    yk_QPSK = qpsk_symbols .* h(1:2:end) + noise;

    %% The Receiver: Channel inversion
    xk_hat_QPSK = yk_QPSK ./ h(1:2:end); % Inverse channel
equalization

    % The Demapper: Hard Decision Decoding using Gray coding
    bk_hat_QPSK = zeros(1, bits_num);
    I_channel = real(xk_hat_QPSK) > 0; % Decision for I-channel
    Q_channel = imag(xk_hat_QPSK) > 0; % Decision for Q-channel
    bk_hat_QPSK(1:2:end) = I_channel;
    bk_hat_QPSK(2:2:end) = Q_channel;

    %% Compute BER without coding
    errors = sum(bk_hat_QPSK ~= bk);  % Count bit errors
    BER_QPSK_without_coding(i) = errors / bits_num;  % Simulated
BER
end

%% Case(d): QPSK with 3-repetition coding %%
bk_coded_QPSK = repelem(bk, 3);
for i = 1:length(Eb_No_dB)
    %% The Mapper (QPSK modulation with Gray coding)
    qpsk_symbols_coded = (2*bk_coded_QPSK(1:2:end) - 1) +
1j*(2*bk_coded_QPSK(2:2:end) - 1);

    %% Generating and Adding AWGN
    No = Eb_rep / Eb_No_linear(i);
```

```
% Generate AWGN noise
    noise_coded = sqrt(No / 2) * (randn(1, bits_num_rep/2) + 1j *
randn(1, bits_num_rep/2));

    % Transmit through Rayleigh Flat Fading Channel
    yk_QPSK_coded = qpsk_symbols_coded .* h_coded(1:2:end) +
noise_coded;

    %% The Receiver: Channel inversion
    xk_hat_QPSK_coded = yk_QPSK_coded ./ h_coded(1:2:end); %
Inverse channel equalization

    % The Demapper: Hard Decision Decoding using Gray coding
    bk_hat_QPSK_coded = zeros(1, bits_num_rep);
    I_channel = real(xk_hat_QPSK_coded) > 0; % Decision for I-
channel
    Q_channel = imag(xk_hat_QPSK_coded) > 0; % Decision for Q-
channel
    bk_hat_QPSK_coded(1:2:end) = I_channel;
    bk_hat_QPSK_coded(2:2:end) = Q_channel;

    %% Majority Decoding for 3-repetition coding
    bk_hat_decoded = sum(reshape(bk_hat_QPSK_coded, rep_factor,
bits_num), 1) >= 2; % Majority vote decoding

    %% Compute BER with 3-repetition coding
    errors_coded = sum(bk_hat_decoded ~= bk);  % Count bit errors
    BER_QPSK_with_coding(i) = errors_coded / bits_num;  % Simulated
BER
end


M = 16; % 16-QAM
N = log2(M); % Bits per symbol
constellation = [-3 -1 1 3] + 1j*[-3 -1 1 3]';
constellation = constellation(:); % Flatten matrix to vector
constellation = constellation / sqrt(mean(abs(constellation).^2));
% Normalize power

%% Case(e): 16-QAM without coding %%
for i = 1:length(Eb_No_dB)
    %% Map bits to symbols using Gray-coded 16-QAM
    bit_groups = reshape(bk(1:N*floor(bits_num/N)), N, []).';
    symbol_indices = bi2de(bit_groups, 'left-msb') + 1;
    qam_symbols = constellation(symbol_indices).';

    % Noise variance calculation based on Eb/No
    No = Eb / Eb_No_linear(i);

    % Generate AWGN noise
    noise = sqrt(No / 2) * (randn(1, length(qam_symbols)) + 1j *
randn(1, length(qam_symbols)));
```

```
% Transmit through Rayleigh Flat Fading Channel
    yk_16QAM = qam_symbols .* h(1:floor(bits_num/N)) + noise;

    %% The Receiver: Channel inversion
    xk_hat_16QAM = yk_16QAM ./ h(1:floor(bits_num/N)); % Inverse
channel equalization

    % Demap symbols to bits using minimum distance decoding
    distances = abs(xk_hat_16QAM(:) - constellation.');
    [~, decoded_idx] = min(distances, [], 2);

    % Dynamically determine required columns
    required_columns = ceil(log2(max(decoded_idx)));
    bk_hat = de2bi(decoded_idx - 1, required_columns, 'left-msb');
    bk_hat = reshape(bk_hat.', 1, []);

    % Compute BER
    errors = sum(bk_hat(1:bits_num) ~= bk(1:bits_num));
    BER_16QAM_without_coding(i) = errors / bits_num;
end

%% Case(f): 16-QAM with 3-repetition coding %%
for i = 1:length(Eb_No_dB)
    %% Map bits to symbols using Gray-coded 16-QAM
    bit_groups_coded =
reshape(bk_coded(1:N*floor(bits_num_rep/N)), N, []).';
    symbol_indices_coded = bi2de(bit_groups_coded, 'left-msb') +
1;
    qam_symbols_coded = constellation(symbol_indices_coded).';

    % Noise variance calculation based on Eb/No
    No = Eb_rep / Eb_No_linear(i);

    % Generate AWGN noise
    noise_coded = sqrt(No / 2) * (randn(1,
length(qam_symbols_coded)) + 1j * randn(1,
length(qam_symbols_coded)));

    % Transmit through Rayleigh Flat Fading Channel
    yk_16QAM_coded = qam_symbols_coded .*
h_coded(1:floor(bits_num_rep/N)) + noise_coded;

    %% The Receiver: Channel inversion
    xk_hat_16QAM_coded = yk_16QAM_coded ./
h_coded(1:floor(bits_num_rep/N)); % Inverse channel equalization

    % Demap symbols to bits using minimum distance decoding
    distances_coded = abs(xk_hat_16QAM_coded(:) -
constellation.');
    [~, decoded_idx_coded] = min(distances_coded, [], 2);
```

```
% Dynamically determine required columns
    required_columns = ceil(log2(max(decoded_idx)));
    bk_hat = de2bi(decoded_idx - 1, required_columns, 'left-msb');
    bk_hat = reshape(bk_hat.', 1, []);

    % Compute BER
    errors = sum(bk_hat(1:bits_num) ~= bk(1:bits_num));
    BER_16QAM_without_coding(i) = errors / bits_num;
end

%% Case(f): 16-QAM with 3-repetition coding %%
for i = 1:length(Eb_No_dB)
    %% Map bits to symbols using Gray-coded 16-QAM
    bit_groups_coded =
reshape(bk_coded(1:N*floor(bits_num_rep/N)), N, []).';
    symbol_indices_coded = bi2de(bit_groups_coded, 'left-msb') +
1;
    qam_symbols_coded = constellation(symbol_indices_coded).';

    % Noise variance calculation based on Eb/No
    No = Eb_rep / Eb_No_linear(i);

    % Generate AWGN noise
    noise_coded = sqrt(No / 2) * (randn(1,
length(qam_symbols_coded)) + 1j * randn(1,
length(qam_symbols_coded)));

    % Transmit through Rayleigh Flat Fading Channel
    yk_16QAM_coded = qam_symbols_coded .*
h_coded(1:floor(bits_num_rep/N)) + noise_coded;

    %% The Receiver: Channel inversion
    xk_hat_16QAM_coded = yk_16QAM_coded ./
h_coded(1:floor(bits_num_rep/N)); % Inverse channel equalization

    % Demap symbols to bits using minimum distance decoding
    distances_coded = abs(xk_hat_16QAM_coded(:) -
constellation.');
    [~, decoded_idx_coded] = min(distances_coded, [], 2);

    % Dynamically determine required columns
    required_columns_coded = ceil(log2(max(decoded_idx_coded)));
    bk_hat_coded = de2bi(decoded_idx_coded - 1,
required_columns_coded, 'left-msb');
    bk_hat_coded = reshape(bk_hat_coded.', 1, []);

    % Majority decoding for 3-repetition coding
    bk_hat_decoded = sum(reshape(bk_hat_coded, rep_factor,
bits_num), 1) >= 2;

    % Compute BER
    errors_coded = sum(bk_hat_decoded ~= bk);
    BER_16QAM_with_coding(i) = errors_coded / bits_num;
end
```

```
%% Plot results (BER vs. Eb/No)
%% Cases(a) and (b): BPSK without and with 3-repetition coding
respectively
figure;
semilogy(Eb_No_dB, BER_BPSK_without_coding, 'r-', 'LineWidth', 2);
hold on;
semilogy(Eb_No_dB, BER_BPSK_with_coding, 'b-', 'LineWidth', 2);
grid on;
xlabel('Eb/No (dB)');
ylabel('Bit Error Rate (BER)');
ylim([10^-6 1]);
legend('BPSK Without Coding', 'BPSK With 3-Repetition Coding');
title('BER Performance of BPSK over Rayleigh Flat Fading
Channel');
hold off;

%% Cases(c) and (d): QPSK without and with 3-repetition coding
respectively
figure;
semilogy(Eb_No_dB, BER_QPSK_without_coding, 'g-', 'LineWidth', 2);
hold on;
semilogy(Eb_No_dB, BER_QPSK_with_coding, 'm-', 'LineWidth', 2);
grid on;
xlabel('Eb/No (dB)');
ylabel('Bit Error Rate (BER)');
ylim([10^-6 1]);
legend('QPSK Without Coding', 'QPSK With 3-Repetition Coding');
title('BER Performance of QPSK over Rayleigh Flat Fading
Channel');
hold off;

%% Cases(e) and (f): 16-QAM without and with 3-repetition coding
respectively
figure;
semilogy(Eb_No_dB, BER_16QAM_without_coding, 'k-', 'LineWidth',
2);
hold on;
semilogy(Eb_No_dB, BER_16QAM_with_coding, 'c-', 'LineWidth', 2);
grid on;
xlabel('Eb/No (dB)');
ylabel('Bit Error Rate (BER)');
ylim([10^-6 1]);
legend('16-QAM Without Coding', '16-QAM With 3-Repetition
Coding');
title('BER Performance of 16-QAM over Rayleigh Flat Fading
Channel');
hold off;
```

```
%% BER vs. Eb/No for all cases
figure;
semilogy(Eb_No_dB, BER_BPSK_without_coding, 'r-', 'LineWidth',
1.5);
hold on;
semilogy(Eb_No_dB, BER_BPSK_with_coding, 'b-', 'LineWidth', 1.5);
semilogy(Eb_No_dB, BER_QPSK_without_coding, 'g-', 'LineWidth',
1.5);
semilogy(Eb_No_dB, BER_QPSK_with_coding, 'm-', 'LineWidth', 1.5);
semilogy(Eb_No_dB, BER_16QAM_without_coding, 'k-', 'LineWidth',
1.5);
semilogy(Eb_No_dB, BER_16QAM_with_coding, 'c-', 'LineWidth', 1.5);
grid on;
xlabel('Eb/No (dB)');
ylabel('Bit Error Rate (BER)');
ylim([10^-6 1]);
legend('BPSK Without Coding', 'BPSK With 3-Repetition Coding', ...
'QPSK Without Coding', 'QPSK With 3-Repetition Coding', ...
    '16-QAM Without Coding', '16-QAM With 3-Repetition Coding');
title('BER Performance of BPSK, QPSK, and 16-QAM over Rayleigh
Flat Fading Channel');
hold off;
```

3. Comment

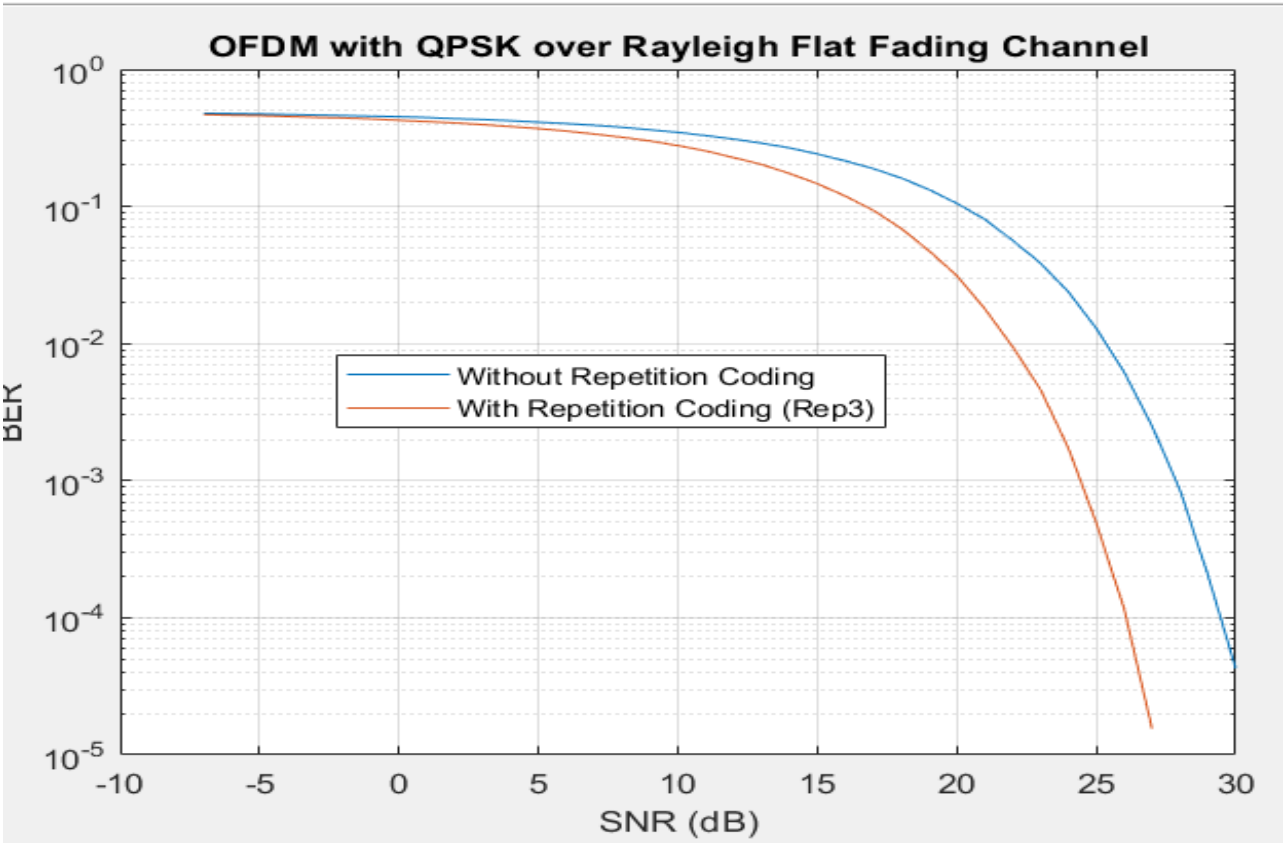
- Each modulation scheme reacts differently over the Rayleigh Flat Fading Channel. The sensitivity to such fading depends on the number of symbols in the modulation scheme and the distance between constellation points.
- The higher the modulation order, the more sensitive to Rayleigh Flat Fading Channel.
- **16-QAM modulation scheme** is the most sensitive scheme to Rayleigh Flat Fading Channel where the constellation points in 16-QAM are closely spaced, resulting in smaller Euclidean distances between symbols.
- **QPSK modulation scheme** has moderate sensitivity to the channel, while **BPSK modulation scheme** has the least sensitivity to the channel.
- From BER point of view;
  BPSK modulation scheme is the best scheme as it has the lowest BER.
  QPSK modulation scheme has moderate BER.
  16-QAM modulation scheme has the highest BER which is not good for the signal.
- There is a trade-off between the BER and sensitivity to the Rayleigh Flat Fading Channel.

Problem 3: OFDM system simulation

1. **The output**

Case A:

1. **QPSK over Rayleigh Flat Fading Channel**



- **MATLAB code**

```
clear all;
clc;
clear all;

% Parameters
numBits = 258048;         % Total number of bits (adjusted for
divisibility)
blockSize = 256;          % Block size for interleaving
numBlocks = numBits / blockSize;
N = 128;                  % Number of subcarriers (for IFFT)
cyclicPrefixLength = 32; % Length of cyclic prefix
snrRange = -7:30;         % SNR range (in dB)
Eb = 1;                   % Energy per bit

 % Generate random data
inputData = randi([0 1], numBits, 1); % Column vector
```

```
% Interleave data block-by-block
interleavedData = zeros(numBits, 1);
for i = 1:numBlocks
    block = inputData((i-1)*blockSize + 1:i*blockSize);
    interleavedBlock = interleaver(block', 'QPSK'); % Row vector
    interleavedData((i-1)*blockSize + 1:i*blockSize) =
interleavedBlock'; % Back to column
end

% Map interleaved bits to QPSK symbols
qpskSymbols = qpsk_mapper(interleavedData, Eb);

% Reshape QPSK symbols into blocks of size N for IFFT
numSymbols = length(qpskSymbols);
numBlocksIFFT = ceil(numSymbols / N);
paddedSymbols = [qpskSymbols; zeros(N * numBlocksIFFT -
numSymbols, 1)]; % Zero-pad
qpskSymbolsMatrix = reshape(paddedSymbols, N, []);

% Perform IFFT
timeDomainSignal = ifft(qpskSymbolsMatrix);

% Add cyclic prefix
cyclicPrefix = timeDomainSignal(end-cyclicPrefixLength+1:end, :);
cyclicSignal = [cyclicPrefix; timeDomainSignal];

% Generate Rayleigh flat fading channel in the frequency domain
hFreq = (1/sqrt(2)) * (randn(N, size(cyclicSignal, 2)) + 1j *
randn(N, size(cyclicSignal, 2))); % Frequency domain channel
response

% Rayleigh flat fading channel and noise simulation
ber1 = zeros(size(snrRange));
for snrIdx = 1:length(snrRange)
    snr = snrRange(snrIdx);
    noisePower = Eb / (10^(snr / 10)); % Adjust for QPSK (2
bits/symbol)

    % Add AWGN to the time-domain signal
    noise = sqrt(noisePower / 2) .* (randn(size(cyclicSignal)) +
1j * randn(size(cyclicSignal)));
    receivedSignal = cyclicSignal + noise;

    % Remove cyclic prefix
    receivedSignal = receivedSignal(cyclicPrefixLength+1:end, :);

    % Perform FFT
    freqDomainSignal = fft(receivedSignal);

    % Apply Rayleigh fading in the frequency domain
    fadedSignal = freqDomainSignal .* hFreq;
```

```
% Equalize (compensate for fading)
    equalizedSignal = fadedSignal ./ hFreq;

    % Demap QPSK symbols
    demappedBits = qpsk_demapper(equalizedSignal(:), Eb);

    % Deinterleave the bits
    outputBits = zeros(numBits, 1);
    for i = 1:numBlocks
        block = demappedBits((i-1)*blockSize + 1:i*blockSize);
        deinterleavedBlock = deinterleaver(block', 'QPSK');
        outputBits((i-1)*blockSize + 1:i*blockSize) =
deinterleavedBlock';
    end

    % Calculate BER
    bitErrors = sum(inputData ~= outputBits);
    ber1(snrIdx) = bitErrors / numBits;
end

%% Repetition encoding (rep-3)
encodedData = repelem(inputData, 3);

% Interleave data block-by-block
numEncodedBits = length(encodedData);
interleavedData = zeros(numEncodedBits, 1);
numBlocksEncoded = numEncodedBits / blockSize;

for i = 1:numBlocksEncoded
    block = encodedData((i-1)*blockSize + 1:i*blockSize);
    interleavedBlock = interleaver(block', 'QPSK'); % Row vector
    interleavedData((i-1)*blockSize + 1:i*blockSize) =
interleavedBlock'; % Back to column
end

% Map interleaved bits to QPSK symbols
qpskSymbols = qpsk_mapper(interleavedData, Eb);

% Reshape QPSK symbols into blocks of size N for IFFT
numSymbols = length(qpskSymbols);
numBlocksIFFT = ceil(numSymbols / N);
paddedSymbols = [qpskSymbols; zeros(N * numBlocksIFFT -
numSymbols, 1)]; % Zero-pad
qpskSymbolsMatrix = reshape(paddedSymbols, N, []);

% Perform IFFT
timeDomainSignal = ifft(qpskSymbolsMatrix);

% Add cyclic prefix
cyclicPrefix = timeDomainSignal(end-cyclicPrefixLength+1:end, :);
cyclicSignal = [cyclicPrefix; timeDomainSignal];
```

```
% Generate Rayleigh flat fading channel in the frequency domain
hFreq = (1/sqrt(2)) * (randn(N, size(cyclicSignal, 2)) + 1j *
randn(N, size(cyclicSignal, 2))); % Frequency domain channel
response

% Rayleigh flat fading channel and noise simulation
ber = zeros(size(snrRange));
for snrIdx = 1:length(snrRange)
    snr = snrRange(snrIdx);
    noisePower = Eb / (10^(snr / 10)); % Adjust for QPSK (2
bits/symbol)

    % Add AWGN to the time-domain signal
    noise = sqrt(noisePower / 2)  .* (randn(size(cyclicSignal)) +
1j*randn(size(cyclicSignal)));
    receivedSignal = cyclicSignal + noise;

    % Remove cyclic prefix
    receivedSignal = receivedSignal(cyclicPrefixLength+1:end, :);

    % Perform FFT
    freqDomainSignal = fft(receivedSignal);

    % Apply Rayleigh fading in the frequency domain
    fadedSignal = freqDomainSignal .* hFreq;

    % Equalize (compensate for fading)
    equalizedSignal = fadedSignal ./ hFreq;

    % Demap QPSK symbols
    demappedBits = qpsk_demapper(equalizedSignal(:), Eb);

    % Deinterleave the bits
    outputBits = zeros(numEncodedBits, 1);
    for i = 1:numBlocksEncoded
        block = demappedBits((i-1)*blockSize + 1:i*blockSize);
        deinterleavedBlock = deinterleaver(block', 'QPSK');
        outputBits((i-1)*blockSize + 1:i*blockSize) =
deinterleavedBlock';
    end

    % Decode repetition (rep-3)
    decodedBits = mode(reshape(outputBits, 3, []).', 2);

    % Calculate BER
    bitErrors = sum(inputData ~= decodedBits);
    ber(snrIdx) = bitErrors / numBits;
end
```

```
% Plot BER vs SNR
figure;
semilogy(snrRange, ber1, '-o', 'DisplayName', 'Without Repetition
Coding');
hold on;
semilogy(snrRange, ber, '-s', 'DisplayName', 'With Repetition
Coding (Rep3)');
xlabel('SNR (dB)');
ylabel('BER');
grid on;
title('OFDM with QPSK over Rayleigh Flat Fading Channel');
legend;
hold off;

function interleavedData = interleaver(inputData,
modulationScheme)
    switch modulationScheme
        case 'QPSK'
            rows = 16;
            cols = 16;
        case '16QAM'
            rows = 32;
            cols = 16;
        otherwise
            error('Unsupported modulation scheme. Use "QPSK" or
"16QAM".');
    end

    interleaverSize = rows * cols;
    if length(inputData) ~= interleaverSize
        error('Input data length must be %d for %s.',
interleaverSize, modulationScheme);
    end

    dataMatrix = reshape(inputData, rows, cols);
    interleavedMatrix = dataMatrix';
    interleavedData = interleavedMatrix(:)';
end

function deinterleavedData = deinterleaver(interleavedData,
modulationScheme)
    switch modulationScheme
        case 'QPSK'
            rows = 16;
            cols = 16;
        case '16QAM'
            rows = 32;
            cols = 16;
        otherwise
            error('Unsupported modulation scheme. Use "QPSK" or
"16QAM".');
    end
```

```
deinterleaverSize = rows * cols;
    if length(interleavedData) ~= deinterleaverSize
        error('Interleaved data length must be %d for %s.',
deinterleaverSize, modulationScheme);
    end

    interleavedMatrix = reshape(interleavedData, cols, rows);
    deinterleavedMatrix = interleavedMatrix';
    deinterleavedData = deinterleavedMatrix(:)';
end

function symbols = qpsk_mapper(bits, Eb)
    if mod(length(bits), 2) ~= 0
        error('Input length must be even.');
    end

    bit_pairs = reshape(bits, 2, []).';
    Es = 2 * Eb;
    norm_factor = sqrt(Es / 2);

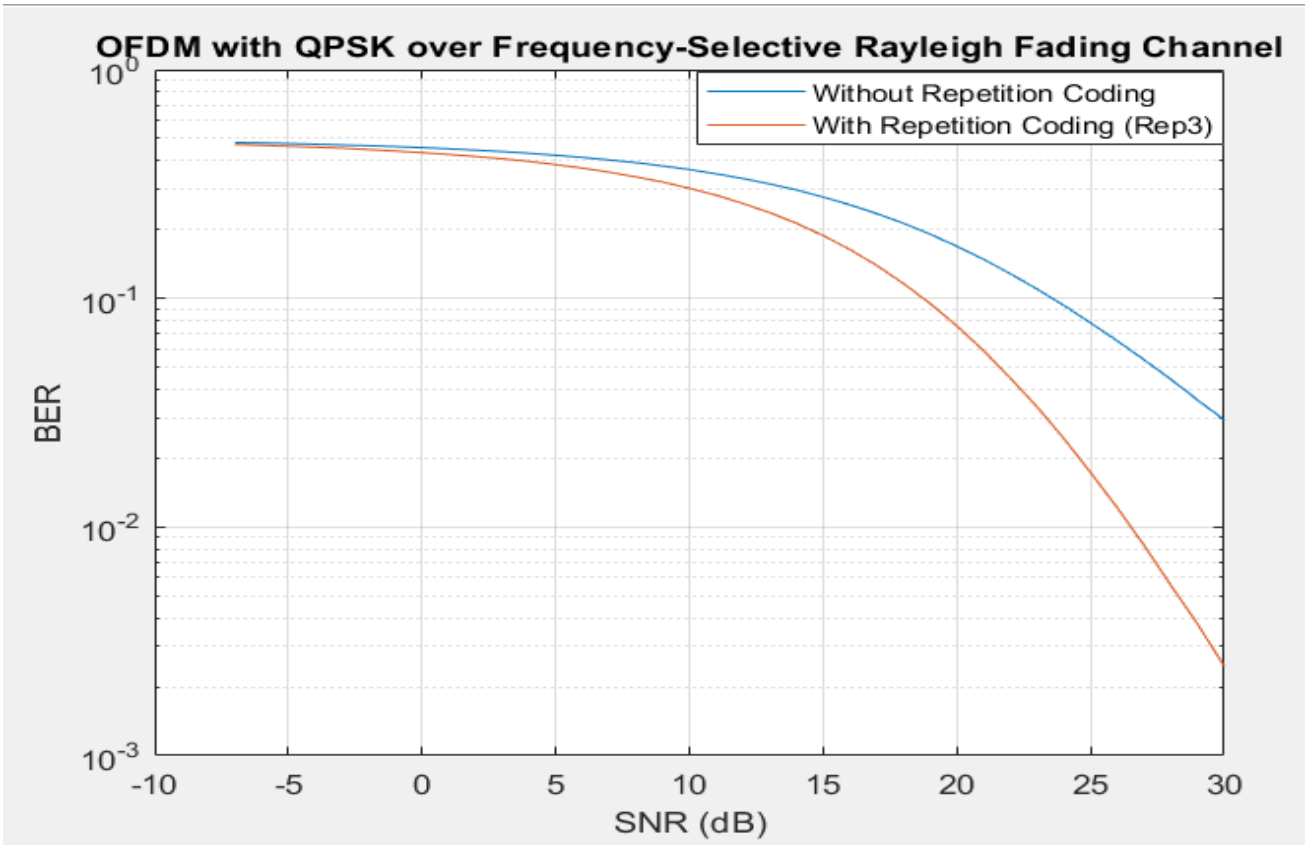
    symbols = zeros(size(bit_pairs, 1), 1);
    symbols(bit_pairs(:, 1) == 0 & bit_pairs(:, 2) == 0) = -1 -
1j;
    symbols(bit_pairs(:, 1) == 0 & bit_pairs(:, 2) == 1) = -1 +
1j;
    symbols(bit_pairs(:, 1) == 1 & bit_pairs(:, 2) == 1) =  1 +
1j;
    symbols(bit_pairs(:, 1) == 1 & bit_pairs(:, 2) == 0) =  1 -
1j;

    symbols = symbols * norm_factor;
end

function bits = qpsk_demapper(symbols, Eb)
    Es = 2 * Eb;
    norm_factor = sqrt(Es / 2);
    symbols = symbols / norm_factor;

    bits = zeros(length(symbols) * 2, 1);
    bits(1:2:end) = real(symbols) > 0;
    bits(2:2:end) = imag(symbols) > 0;
end
```

2. QPSK over Frequency selective:



- **MATLAB code**

```
clear all;
clc;

% Parameters
numBits = 2580480;        % Total number of bits (adjusted for
divisibility)
blockSize = 256;          % Block size for interleaving
numBlocks = numBits / blockSize;
N = 128;                  % Number of subcarriers (for IFFT)
cyclicPrefixLength = 32; % Length of cyclic prefix
snrRange = -7:30;         % SNR range (in dB)
Eb = 1;                   % Energy per bit

% Number of groups for frequency-selective fading
numGroups = 16;           % Number of groups for frequency-
selective fading
subchannelsPerGroup = N / numGroups; % Subchannels per group

% Generate random data
inputData = randi([0 1], numBits, 1); % Column vector
```

```
% Interleave data block-by-block
interleavedData = zeros(numBits, 1);
for i = 1:numBlocks
    block = inputData((i-1)*blockSize + 1:i*blockSize);
    interleavedBlock = interleaver(block', 'QPSK'); % Row vector
    interleavedData((i-1)*blockSize + 1:i*blockSize) =
interleavedBlock'; % Back to column
end

% Map interleaved bits to QPSK symbols
qpskSymbols = qpsk_mapper(interleavedData, Eb);

% Reshape QPSK symbols into blocks of size N for IFFT
numSymbols = length(qpskSymbols);
numBlocksIFFT = ceil(numSymbols / N);
paddedSymbols = [qpskSymbols; zeros(N * numBlocksIFFT -
numSymbols, 1)]; % Zero-pad
qpskSymbolsMatrix = reshape(paddedSymbols, N, numBlocksIFFT);

% Generate frequency-selective Rayleigh fading matrix
hGroup = (1/sqrt(2)) * (randn(numGroups, numBlocksIFFT) + 1j *
randn(numGroups, numBlocksIFFT));
hFreqDomain = zeros(N, numBlocksIFFT);
for groupIdx = 1:numGroups
    hFreqDomain((groupIdx-1)*subchannelsPerGroup +
1:groupIdx*subchannelsPerGroup, :) = ...
        repmat(hGroup(groupIdx, :), subchannelsPerGroup, 1);
end

% Apply Rayleigh fading in frequency domain
fadedSignalFreqDomain = qpskSymbolsMatrix .* hFreqDomain;

% Transform to time domain
timeDomainSignal = ifft(fadedSignalFreqDomain);

% Add cyclic prefix
cyclicPrefix = timeDomainSignal(end-cyclicPrefixLength+1:end, :);
fadedSignalCyclic = [cyclicPrefix; timeDomainSignal];

% Frequency-selective fading channel and noise simulation
ber1 = zeros(size(snrRange));
for snrIdx = 1:length(snrRange)
    snr = snrRange(snrIdx);
    noisePower = Eb / (10^(snr / 10)); % Adjust for QPSK (2
bits/symbol)

    % Add AWGN
    noise = sqrt(noisePower / 2)  .*
(randn(size(fadedSignalCyclic)) + 1j *
randn(size(fadedSignalCyclic)));
    receivedSignal = fadedSignalCyclic + noise;
```

```
% Remove cyclic prefix
    receivedSignal = receivedSignal(cyclicPrefixLength+1:end, :);

    % Perform FFT
    freqDomainSignal = fft(receivedSignal);

    % Equalize (compensate for fading)
    equalizedSignal = freqDomainSignal ./ hFreqDomain;

    % Demap QPSK symbols
    demappedBits = qpsk_demapper(equalizedSignal(:), Eb);

    % Deinterleave the bits
    outputBits = zeros(numBits, 1);
    for i = 1:numBlocks
        block = demappedBits((i-1)*blockSize + 1:i*blockSize);
        deinterleavedBlock = deinterleaver(block', 'QPSK');
        outputBits((i-1)*blockSize + 1:i*blockSize) =
deinterleavedBlock';
    end

    % Calculate BER
    bitErrors = sum(inputData ~= outputBits);
    ber1(snrIdx) = bitErrors / numBits;
end

%% Repetition encoding (rep-3)
encodedData = repelem(inputData, 3);

% Interleave data block-by-block
numEncodedBits = length(encodedData);
interleavedData = zeros(numEncodedBits, 1);
numBlocksEncoded = numEncodedBits / blockSize;

for i = 1:numBlocksEncoded
    block = encodedData((i-1)*blockSize + 1:i*blockSize);
    interleavedBlock = interleaver(block', 'QPSK'); % Row vector
    interleavedData((i-1)*blockSize + 1:i*blockSize) =
interleavedBlock'; % Back to column
end

% Map interleaved bits to QPSK symbols
qpskSymbols = qpsk_mapper(interleavedData, Eb);

% Reshape QPSK symbols into blocks of size N for IFFT
numSymbols = length(qpskSymbols);
numBlocksIFFT = ceil(numSymbols / N);
paddedSymbols = [qpskSymbols; zeros(N * numBlocksIFFT -
numSymbols, 1)]; % Zero-pad
qpskSymbolsMatrix = reshape(paddedSymbols, N, numBlocksIFFT);
```

```
% Generate frequency-selective Rayleigh fading matrix
hGroup = (1/sqrt(2)) * (randn(numGroups, numBlocksIFFT) + 1j *
randn(numGroups, numBlocksIFFT));
hFreqDomain = zeros(N, numBlocksIFFT);
for groupIdx = 1:numGroups
    hFreqDomain((groupIdx-1)*subchannelsPerGroup +
1:groupIdx*subchannelsPerGroup, :) = ...
        repmat(hGroup(groupIdx, :), subchannelsPerGroup, 1);
end

% Apply Rayleigh fading in frequency domain
fadedSignalFreqDomain = qpskSymbolsMatrix .* hFreqDomain;

% Transform to time domain
timeDomainSignal = ifft(fadedSignalFreqDomain);

% Add cyclic prefix
cyclicPrefix = timeDomainSignal(end-cyclicPrefixLength+1:end, :);
fadedSignalCyclic = [cyclicPrefix; timeDomainSignal];

% Frequency-selective fading channel and noise simulation
ber2 = zeros(size(snrRange));
for snrIdx = 1:length(snrRange)
    snr = snrRange(snrIdx);
    noisePower = Eb / (10^(snr / 10)); % Adjust for QPSK (2
bits/symbol)

    % Add AWGN
    noise = sqrt(noisePower / 2)  .*
(randn(size(fadedSignalCyclic)) + 1j *
randn(size(fadedSignalCyclic)));
    receivedSignal = fadedSignalCyclic + noise;

    % Remove cyclic prefix
    receivedSignal = receivedSignal(cyclicPrefixLength+1:end, :);

    % Perform FFT
    freqDomainSignal = fft(receivedSignal);

    % Equalize (compensate for fading)
    equalizedSignal = freqDomainSignal ./ hFreqDomain;

    % Demap QPSK symbols
    demappedBits = qpsk_demapper(equalizedSignal(:), Eb);

    % Deinterleave the bits
    outputBits = zeros(numEncodedBits, 1);
    for i = 1:numBlocksEncoded
        block = demappedBits((i-1)*blockSize + 1:i*blockSize);
        deinterleavedBlock = deinterleaver(block', 'QPSK');
        outputBits((i-1)*blockSize + 1:i*blockSize) =
deinterleavedBlock';
    end
```

```
% Decode repetition (rep-3)
    decodedBits = mode(reshape(outputBits, 3, []).', 2);

    % Calculate BER
    bitErrors = sum(inputData ~= decodedBits);
    ber2(snrIdx) = bitErrors / numBits;
end

% Plot BER vs SNR
figure;
semilogy(snrRange, ber1, '-', 'DisplayName', 'Without Repetition
Coding');
hold on;
semilogy(snrRange, ber2, '-', 'DisplayName', 'With Repetition
Coding (Rep3)');
xlabel('SNR (dB)');
ylabel('BER');
grid on;
title('OFDM with QPSK over Frequency-Selective Rayleigh Fading
Channel');
legend;
hold off;

function interleavedData = interleaver(inputData,
modulationScheme)
    switch modulationScheme
        case 'QPSK'
            rows = 16;
            cols = 16;
        case '16QAM'
            rows = 32;
            cols = 16;
        otherwise
            error('Unsupported modulation scheme. Use "QPSK" or
"16QAM".');
    end

    interleaverSize = rows * cols;
    if length(inputData) ~= interleaverSize
        error('Input data length must be %d for %s.',
interleaverSize, modulationScheme);
    end

    dataMatrix = reshape(inputData, rows, cols);
    interleavedMatrix = dataMatrix';
    interleavedData = interleavedMatrix(:)';
end

function deinterleavedData = deinterleaver(interleavedData,
modulationScheme)
    switch modulationScheme
        case 'QPSK'
            rows = 16;
            cols = 16;
```

```
        case '16QAM'
            rows = 32;
            cols = 16;
        otherwise
            error('Unsupported modulation scheme. Use "QPSK" or
"16QAM".');
    end

    deinterleaverSize = rows * cols;
    if length(interleavedData) ~= deinterleaverSize
        error('Interleaved data length must be %d for %s.',
deinterleaverSize, modulationScheme);
    end

    interleavedMatrix = reshape(interleavedData, cols, rows);
    deinterleavedMatrix = interleavedMatrix';
    deinterleavedData = deinterleavedMatrix(:)';
end

function symbols = qpsk_mapper(bits, Eb)
    if mod(length(bits), 2) ~= 0
        error('Input length must be even.');
    end

    bit_pairs = reshape(bits, 2, []).';
    Es = 2 * Eb;
    norm_factor = sqrt(Es / 2);

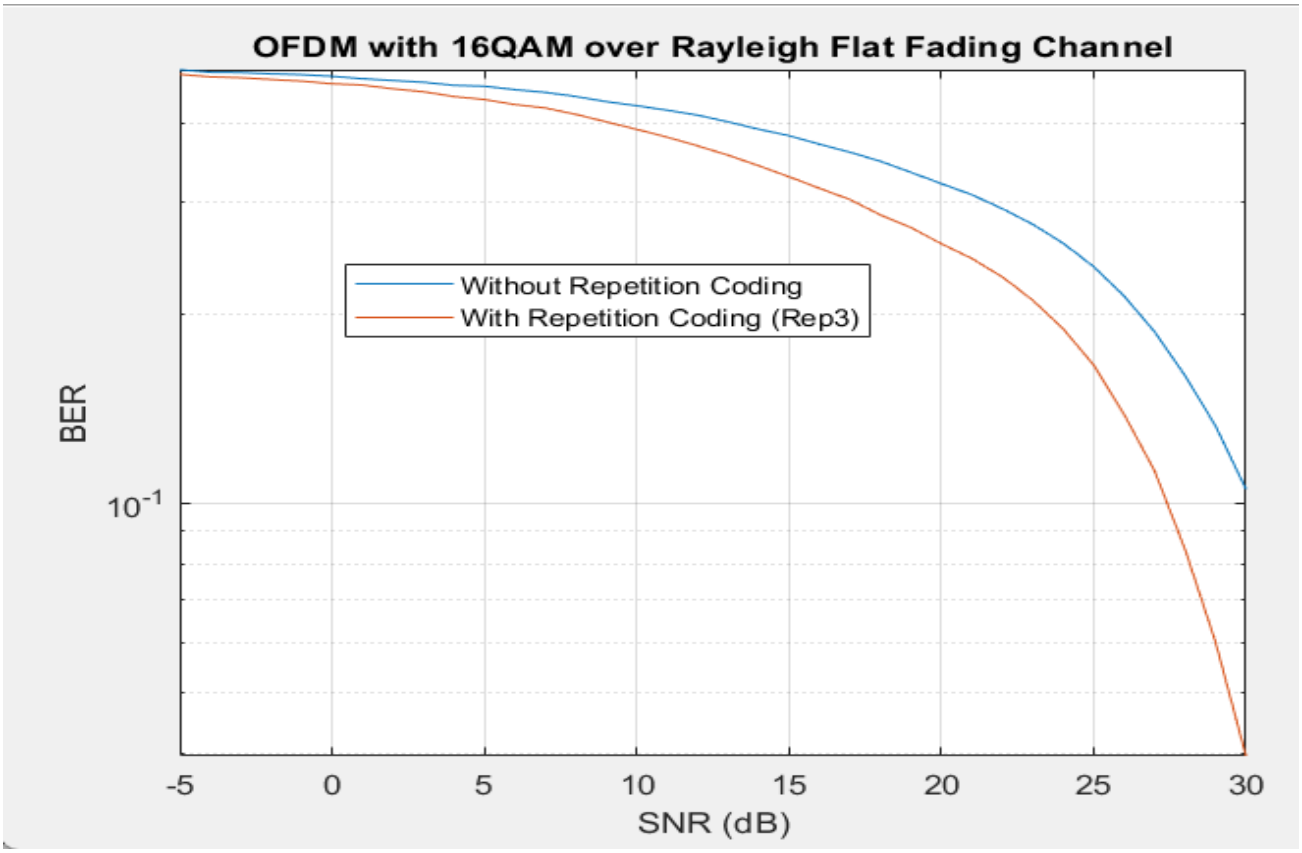
    symbols = zeros(size(bit_pairs, 1), 1);
    symbols(bit_pairs(:, 1) == 0 & bit_pairs(:, 2) == 0) = -1 -
1j;
    symbols(bit_pairs(:, 1) == 0 & bit_pairs(:, 2) == 1) = -1 +
1j;
    symbols(bit_pairs(:, 1) == 1 & bit_pairs(:, 2) == 1) =  1 +
1j;
    symbols(bit_pairs(:, 1) == 1 & bit_pairs(:, 2) == 0) =  1 -
1j;

    symbols = symbols * norm_factor;
end

function bits = qpsk_demapper(symbols, Eb)
    Es = 2 * Eb;
    norm_factor = sqrt(Es / 2);
    symbols = symbols / norm_factor;

    bits = zeros(length(symbols) * 2, 1);
    bits(1:2:end) = real(symbols) > 0;
    bits(2:2:end) = imag(symbols) > 0;
end
```

3. 16QAM over Rayleigh



- **MATLAB code**

```
clc;
clear all;

% Parameters
numBits = 258048;        % Total number of bits (adjusted for
divisibility)
blockSize = 512;         % Block size for interleaving
numBlocks = numBits / blockSize;
N = 128;                 % Number of subcarriers (for IFFT)
cyclicPrefixLength = 32; % Length of cyclic prefix
snrRange = -5:30;        % SNR range (in dB)
Eb = 1;                  % Energy per bit

% Generate random data
inputData = randi([0 1], numBits, 1); % Column vector

% Interleave data block-by-block
interleavedData = zeros(numBits, 1);
for i = 1:numBlocks
    block = inputData((i-1)*blockSize + 1:i*blockSize);
    interleavedBlock = interleaver(block', '16QAM'); % Row vector
    interleavedData((i-1)*blockSize + 1:i*blockSize) =
interleavedBlock'; % Back to column
end
```

```
% Map interleaved bits to 16-QAM symbols
qamSymbols = qam16_mapper(interleavedData, Eb);

% Reshape QAM symbols into blocks of size N for IFFT
numSymbols = length(qamSymbols);
numBlocksIFFT = ceil(numSymbols / N);
paddedSymbols = [qamSymbols; zeros(N * numBlocksIFFT -
numSymbols, 1)]; % Zero-pad
qamSymbolsMatrix = reshape(paddedSymbols, N, []);

% Perform IFFT
timeDomainSignal = ifft(qamSymbolsMatrix);

% Add cyclic prefix
cyclicPrefix = timeDomainSignal(end-cyclicPrefixLength+1:end, :);
cyclicSignal = [cyclicPrefix; timeDomainSignal];

% Generate Rayleigh flat fading channel in the frequency domain
hFreq = (1/sqrt(2)) * (randn(N, size(cyclicSignal, 2)) + 1j *
randn(N, size(cyclicSignal, 2))); % Frequency domain channel
response

% Rayleigh flat fading channel and noise simulation
ber1 = zeros(size(snrRange));
for snrIdx = 1:length(snrRange)
    snr = snrRange(snrIdx);
    noisePower = Eb / (10^(snr / 10)); % Adjust for 16-QAM (4
bits/symbol)

    % Add AWGN to the time-domain signal
    noise = sqrt(noisePower / 2) .* (randn(size(cyclicSignal)) +
1j * randn(size(cyclicSignal)));
    receivedSignal = cyclicSignal + noise;

    % Remove cyclic prefix
    receivedSignal = receivedSignal(cyclicPrefixLength+1:end, :);

    % Perform FFT
    freqDomainSignal = fft(receivedSignal);

    % Apply Rayleigh fading in the frequency domain
    fadedSignal = freqDomainSignal .* hFreq;

    % Equalize (compensate for fading)
    equalizedSignal = fadedSignal ./ hFreq;
    % Remove zero padding
    equalizedSignal = equalizedSignal(:); % Convert to column
vector
    equalizedSignal = equalizedSignal(1:numSymbols); % Retain
only original symbols
```

```
% Demap 16-QAM symbols
    demappedBits = qam16_demapper(equalizedSignal(:), Eb);

    % Deinterleave the bits
    outputBits = zeros(numBits, 1);
    for i = 1:numBlocks
        block = demappedBits((i-1)*blockSize + 1:i*blockSize);
        deinterleavedBlock = deinterleaver(block', '16QAM');
        outputBits((i-1)*blockSize + 1:i*blockSize) =
deinterleavedBlock';
    end

    % Calculate BER
    bitErrors = sum(inputData ~= outputBits);
    ber1(snrIdx) = bitErrors / numBits;
end

% Repetition Encoding (Rep-3)
encodedData = repelem(inputData, 3);

% Interleave data block-by-block
numEncodedBits = length(encodedData);
numBlocksEncoded = numEncodedBits / blockSize;
interleavedData = zeros(numEncodedBits, 1);
for i = 1:numBlocksEncoded
    block = encodedData((i-1)*blockSize + 1:i*blockSize);
    interleavedBlock = interleaver(block', '16QAM'); % Row vector
    interleavedData((i-1)*blockSize + 1:i*blockSize) =
interleavedBlock'; % Back to column
end

% Map interleaved bits to 16-QAM symbols
qamSymbols = qam16_mapper(interleavedData, Eb);

% Reshape QAM symbols into blocks of size N for IFFT
numSymbols = length(qamSymbols);
numBlocksIFFT = ceil(numSymbols / N);
paddedSymbols = [qamSymbols; zeros(N * numBlocksIFFT -
numSymbols, 1)]; % Zero-pad
qamSymbolsMatrix = reshape(paddedSymbols, N, []);

% Perform IFFT
timeDomainSignal = ifft(qamSymbolsMatrix);

% Add cyclic prefix
cyclicPrefix = timeDomainSignal(end-cyclicPrefixLength+1:end, :);
cyclicSignal = [cyclicPrefix; timeDomainSignal];

% Generate Rayleigh flat fading channel in the frequency domain
hFreq = (1/sqrt(2)) * (randn(N, size(cyclicSignal, 2)) + 1j *
randn(N, size(cyclicSignal, 2))); % Frequency domain channel
response
```

```
% Rayleigh flat fading channel and noise simulation
ber = zeros(size(snrRange));
for snrIdx = 1:length(snrRange)
    snr = snrRange(snrIdx);
    noisePower = Eb / (10^(snr / 10)); % Adjust for 16-QAM (4
bits/symbol)

    % Add AWGN to the time-domain signal
    noise = sqrt(noisePower / 2) .* (randn(size(cyclicSignal)) +
1j * randn(size(cyclicSignal)));
    receivedSignal = cyclicSignal + noise;

    % Remove cyclic prefix
    receivedSignal = receivedSignal(cyclicPrefixLength+1:end, :);

    % Perform FFT
    freqDomainSignal = fft(receivedSignal);

    % Apply Rayleigh fading in the frequency domain
    fadedSignal = freqDomainSignal .* hFreq;

    % Equalize (compensate for fading)
    equalizedSignal = fadedSignal ./ hFreq;
    % Remove zero padding
    equalizedSignal = equalizedSignal(:); % Convert to column
vector
    equalizedSignal = equalizedSignal(1:numSymbols); % Retain
only original symbols

    % Demap 16-QAM symbols
    demappedBits = qam16_demapper(equalizedSignal(:), Eb);

    % Deinterleave the bits
    outputBits = zeros(numEncodedBits, 1);
    for i = 1:numBlocksEncoded
        block = demappedBits((i-1)*blockSize + 1:i*blockSize);
        deinterleavedBlock = deinterleaver(block', '16QAM');
        outputBits((i-1)*blockSize + 1:i*blockSize) =
deinterleavedBlock';
    end

    % Decode Repetition (Rep-3)
    decodedBits = mode(reshape(outputBits, 3, []).', 2);

    % Calculate BER
    bitErrors = sum(inputData ~= decodedBits);
    ber(snrIdx) = bitErrors / numBits;
end
```

```
% Plot BER vs SNR
figure;
semilogy(snrRange, ber, '-s', 'DisplayName', '16-QAM with
Repetition Coding (Rep-3)');
xlabel('SNR (dB)');
ylabel('BER');
grid on;
title('OFDM with 16-QAM and Repetition Coding (Rep-3) over
Rayleigh Flat Fading Channel');
legend;
hold off;

% Plot BER vs SNR
figure;
semilogy(snrRange, ber1, '-', 'DisplayName', 'Without Repetition
Coding');
hold on;
semilogy(snrRange, ber, '-', 'DisplayName', 'With Repetition
Coding (Rep3)');
xlabel('SNR (dB)');
ylabel('BER');
grid on;
title('OFDM with 16QAM over Rayleigh Flat Fading Channel');
legend;
hold off;

function interleavedData = interleaver(inputData,
modulationScheme)
    switch modulationScheme
        case 'QPSK'
            rows = 16;
            cols = 16;
        case '16QAM'
            rows = 32;
            cols = 16;
        otherwise
            error('Unsupported modulation scheme. Use "QPSK" or
"16QAM".');
    end

    interleaverSize = rows * cols;
    if length(inputData) ~= interleaverSize
        error('Input data length must be %d for %s.',
interleaverSize, modulationScheme);
    end

    dataMatrix = reshape(inputData, rows, cols);
    interleavedMatrix = dataMatrix';
    interleavedData = interleavedMatrix(:)';
end
```

```
function deinterleavedData = deinterleaver(interleavedData,
modulationScheme)
    switch modulationScheme
        case 'QPSK'
            rows = 16;
            cols = 16;
        case '16QAM'
            rows = 32;
            cols = 16;
        otherwise
            error('Unsupported modulation scheme. Use "QPSK" or
"16QAM".');
    end

    deinterleaverSize = rows * cols;
    if length(interleavedData) ~= deinterleaverSize
        error('Interleaved data length must be %d for %s.',
deinterleaverSize, modulationScheme);
    end

    interleavedMatrix = reshape(interleavedData, cols, rows);
    deinterleavedMatrix = interleavedMatrix';
    deinterleavedData = deinterleavedMatrix(:)';
end

function symbols = qam16_mapper(bits, Eb)
    % Ensure the input length is a multiple of 4
    if mod(length(bits), 4) ~= 0
        error('Input length must be a multiple of 4.');
    end

    % Reshape bits into groups of 4
    bit_groups = reshape(bits, 4, []).';

    % Calculate the scaling factor based on Eb
    scaling_factor = sqrt(Eb / 10); % 10 is the average energy
for 16-QAM symbols

% Map each bit group to a 16-QAM symbol
    symbol_map = [-3 - 3j, -3 - 1j, -3 + 1j, -3 + 3j, ...
                  -1 - 3j, -1 - 1j, -1 + 1j, -1 + 3j, ...
                   1 - 3j,  1 - 1j,  1 + 1j,  1 + 3j, ...
                   3 - 3j,  3 - 1j,  3 + 1j,  3 + 3j];

    binary_labels = de2bi(0:15, 4, 'left-msb'); % Generate all
possible 4-bit labels
    symbols = zeros(1, size(bit_groups, 1)); % Preallocate
symbols array
```

```
    for i = 1:size(bit_groups, 1)
        b = bit_groups(i, :);
        index = find(ismember(binary_labels, b, 'rows'));
        symbols(i) = scaling_factor * symbol_map(index);
    end
end

function bits = qam16_demapper(symbols, Eb)
    % Calculate the scaling factor based on Eb
    scaling_factor = sqrt(Eb / 10);

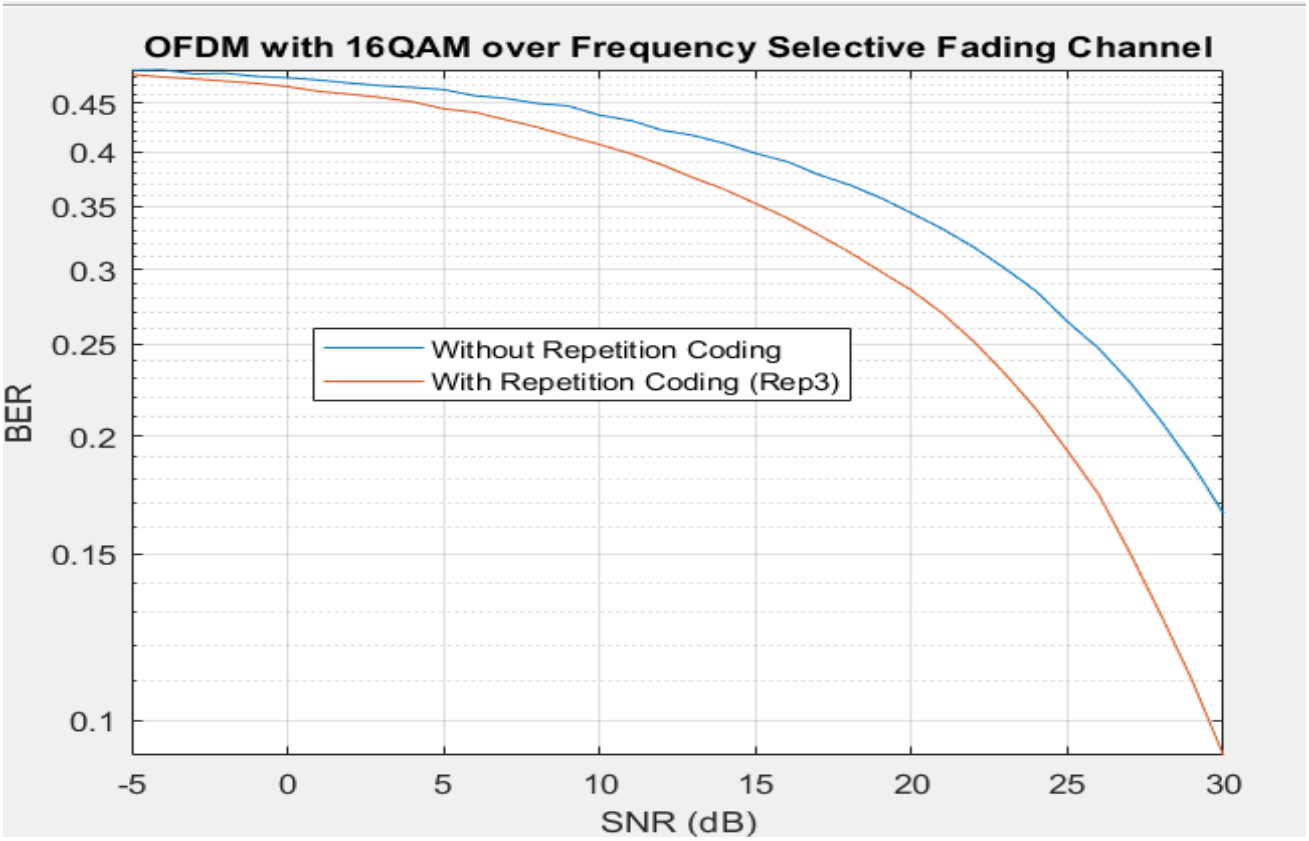
    % Normalize the received symbols
    normalized_symbols = symbols / scaling_factor;

    % Define the constellation
    symbol_map = [-3 - 3j, -3 - 1j, -3 + 1j, -3 + 3j, ...
                  -1 - 3j, -1 - 1j, -1 + 1j, -1 + 3j, ...
                   1 - 3j,  1 - 1j,  1 + 1j,  1 + 3j, ...
                   3 - 3j,  3 - 1j,  3 + 1j,  3 + 3j];
    binary_labels = de2bi(0:15, 4, 'left-msb'); % Generate all
possible 4-bit labels

    % Initialize bits array
    bits = zeros(length(symbols) * 4, 1);

    for i = 1:length(normalized_symbols)
        % Find the closest constellation point
        [~, idx] = min(abs(normalized_symbols(i) - symbol_map));
        bits(4*i-3:4*i) = binary_labels(idx, :);
    end
end
```

4. 16QAM over Frequency selective



- **MATLAB code**

```
clc;
clear all;

% Parameters
numBits = 258048;        % Total number of bits (adjusted for
divisibility)
blockSize = 512;         % Block size for interleaving
numBlocks = numBits / blockSize;
N = 128;                 % Number of subcarriers (for IFFT)
cyclicPrefixLength = 32; % Length of cyclic prefix
snrRange = -5:30;        % SNR range (in dB)
Eb = 1;                  % Energy per bit
numGroups = 16;          % Number of groups for frequency-
selective fading
subchannelsPerGroup = N / numGroups; % Subchannels per group

% Generate random data
inputData = randi([0 1], numBits, 1); % Column vector
```

```
% Interleave data block-by-block
interleavedData = zeros(numBits, 1);
for i = 1:numBlocks
    block = inputData((i-1)*blockSize + 1:i*blockSize);
    interleavedBlock = interleaver(block', '16QAM'); % Row
vector
    interleavedData((i-1)*blockSize + 1:i*blockSize) =
interleavedBlock'; % Back to column
end

% Map interleaved bits to 16-QAM symbols
qamSymbols = qam16_mapper(interleavedData, Eb);

% Reshape QAM symbols into blocks of size N for IFFT
numSymbols = length(qamSymbols);
numBlocksIFFT = ceil(numSymbols / N);
paddedSymbols = [qamSymbols; zeros(N * numBlocksIFFT -
numSymbols, 1)]; % Zero-pad
qamSymbolsMatrix = reshape(paddedSymbols, N, []);

% Generate 16 Rayleigh fading values
hGroup = (1/sqrt(2)) * (randn(numGroups, numBlocksIFFT) + 1j *
randn(numGroups, numBlocksIFFT));

% Apply the same hGroup to 10 subchannels each
hFreqDomain = zeros(N, numBlocksIFFT);
for groupIdx = 1:numGroups
    hFreqDomain((groupIdx-1)*subchannelsPerGroup +
1:groupIdx*subchannelsPerGroup, :) = ...
        repmat(hGroup(groupIdx, :), subchannelsPerGroup, 1);
end

% Apply Rayleigh fading in frequency domain
fadedSignalFreqDomain = qamSymbolsMatrix .* hFreqDomain;

% Transform to time domain
timeDomainSignal = ifft(fadedSignalFreqDomain);

% Add cyclic prefix
cyclicPrefix = timeDomainSignal(end-cyclicPrefixLength+1:end,
:);
fadedSignalCyclic = [cyclicPrefix; timeDomainSignal];

% Frequency-selective fading channel and noise simulation
ber = zeros(size(snrRange));
for snrIdx = 1:length(snrRange)
    snr = snrRange(snrIdx);
    noisePower = Eb / (10^(snr / 10)); % Adjust for 16-QAM (4
bits/symbol)
```

```
    % Add AWGN
    noise = sqrt(noisePower / 2) .*
(randn(size(fadedSignalCyclic)) + 1j *
randn(size(fadedSignalCyclic)));
    receivedSignal = fadedSignalCyclic + noise;

    % Remove cyclic prefix
    receivedSignal = receivedSignal(cyclicPrefixLength+1:end,
:);

    % Perform FFT
    freqDomainSignal = fft(receivedSignal);

    % Equalize (compensate for fading)
    equalizedSignal = freqDomainSignal ./ hFreqDomain;

    % Remove zero padding
    equalizedSignal = equalizedSignal(:); % Convert to column
vector
    equalizedSignal = equalizedSignal(1:numSymbols); % Retain
only original symbols

    % Demap 16-QAM symbols
    demappedBits = qam16_demapper(equalizedSignal(:), Eb);

    % Deinterleave the bits
    outputBits = zeros(numBits, 1);
    for i = 1:numBlocks
        block = demappedBits((i-1)*blockSize + 1:i*blockSize);
        deinterleavedBlock = deinterleaver(block', '16QAM');
        outputBits((i-1)*blockSize + 1:i*blockSize) =
deinterleavedBlock';
    end

    % Calculate BER
    bitErrors = sum(inputData ~= outputBits);
    ber(snrIdx) = bitErrors / numBits;
end

% Plot BER vs SNR
semilogy(snrRange, ber, 'b-o');
grid on;
xlabel('SNR (dB)');
ylabel('BER');
title('BER vs SNR for 16-QAM in Frequency-Selective Fading
Channel');
```

```
% Repetition Encoding (Rep-3)
encodedData = repelem(inputData, 3);

% Interleave data block-by-block
numEncodedBits = length(encodedData);
numBlocksEncoded = numEncodedBits / blockSize;
interleavedData = zeros(numEncodedBits, 1);
for i = 1:numBlocksEncoded
    block = encodedData((i-1)*blockSize + 1:i*blockSize);
    interleavedBlock = interleaver(block', '16QAM'); % Row
vector
    interleavedData((i-1)*blockSize + 1:i*blockSize) =
interleavedBlock'; % Back to column
end

% Map interleaved bits to 16-QAM symbols
qamSymbols = qam16_mapper(interleavedData, Eb);

% Reshape QAM symbols into blocks of size N for IFFT
numSymbols = length(qamSymbols);
numBlocksIFFT = ceil(numSymbols / N);
paddedSymbols = [qamSymbols; zeros(N * numBlocksIFFT -
numSymbols, 1)]; % Zero-pad
qamSymbolsMatrix = reshape(paddedSymbols, N, []);

% Generate 16 Rayleigh fading values
hGroup = (1/sqrt(2)) .* (randn(numGroups, numBlocksIFFT) + 1j *
randn(numGroups, numBlocksIFFT));

% Apply the same hGroup to 10 subchannels each
hFreqDomain = zeros(N, numBlocksIFFT);
for groupIdx = 1:numGroups
    hFreqDomain((groupIdx-1)*subchannelsPerGroup +
1:groupIdx*subchannelsPerGroup, :) = ...
        repmat(hGroup(groupIdx, :), subchannelsPerGroup, 1);
end

% Apply Rayleigh fading in frequency domain
fadedSignalFreqDomain = qamSymbolsMatrix .* hFreqDomain;

% Transform to time domain
timeDomainSignal = ifft(fadedSignalFreqDomain);

% Add cyclic prefix
cyclicPrefix = timeDomainSignal(end-cyclicPrefixLength+1:end,
:);
fadedSignalCyclic = [cyclicPrefix; timeDomainSignal];

% Frequency-selective fading channel and noise simulation
ber1 = zeros(size(snrRange));
```

```
for snrIdx = 1:length(snrRange)
    snr = snrRange(snrIdx);
    noisePower = Eb / (10^(snr / 10)); % Adjust for 16-QAM (4
bits/symbol)

    % Add AWGN
    noise = sqrt(noisePower / 2) .*
(randn(size(fadedSignalCyclic)) + 1j *
randn(size(fadedSignalCyclic)));
    receivedSignal = fadedSignalCyclic + noise;

    % Remove cyclic prefix
    receivedSignal = receivedSignal(cyclicPrefixLength+1:end,
:);

    % Perform FFT
    freqDomainSignal = fft(receivedSignal);

    % Equalize (compensate for fading)
    equalizedSignal = freqDomainSignal ./ hFreqDomain;

    % Remove zero padding
    equalizedSignal = equalizedSignal(:); % Convert to column
vector
    equalizedSignal = equalizedSignal(1:numSymbols); % Retain
only original symbols

    % Demap 16-QAM symbols
    demappedBits = qam16_demapper(equalizedSignal(:), Eb);

    % Deinterleave the bits
    outputBits = zeros(numEncodedBits, 1);
    for i = 1:numBlocksEncoded
        block = demappedBits((i-1)*blockSize + 1:i*blockSize);
        deinterleavedBlock = deinterleaver(block', '16QAM');
        outputBits((i-1)*blockSize + 1:i*blockSize) =
deinterleavedBlock';
    end

    % Decode Repetition (Rep-3)
    decodedBits = mode(reshape(outputBits, 3, []).', 2);

    % Calculate BER
    bitErrors = sum(inputData ~= decodedBits);
    ber1(snrIdx) = bitErrors / numBits;
end
```

```
% Plot BER vs SNR
figure;
semilogy(snrRange, ber, '-', 'DisplayName', 'Without Repetition
Coding');
hold on;
semilogy(snrRange, ber1, '-', 'DisplayName', 'With Repetition
Coding (Rep3)');
xlabel('SNR (dB)');
ylabel('BER');
grid on;
title('OFDM with 16QAM over Frequency Selective Fading
Channel');
legend;
hold off;

function interleavedData = interleaver(inputData,
modulationScheme)
    switch modulationScheme
        case 'QPSK'
            rows = 16;
            cols = 16;
        case '16QAM'
            rows = 32;
            cols = 16;
        otherwise
            error('Unsupported modulation scheme. Use "QPSK" or
"16QAM".');
    end

    interleaverSize = rows * cols;
    if length(inputData) ~= interleaverSize
        error('Input data length must be %d for %s.',
interleaverSize, modulationScheme);
    end

    dataMatrix = reshape(inputData, rows, cols);
    interleavedMatrix = dataMatrix';
    interleavedData = interleavedMatrix(:)';
end

function deinterleavedData = deinterleaver(interleavedData,
modulationScheme)
    switch modulationScheme
        case 'QPSK'
            rows = 16;
            cols = 16;
        case '16QAM'
            rows = 32;
            cols = 16;
        otherwise
            error('Unsupported modulation scheme. Use "QPSK" or
"16QAM".');
    end
```

```
    deinterleaverSize = rows * cols;
    if length(interleavedData) ~= deinterleaverSize
        error('Interleaved data length must be %d for %s.',
deinterleaverSize, modulationScheme);
    end

    interleavedMatrix = reshape(interleavedData, cols, rows);
    deinterleavedMatrix = interleavedMatrix';
    deinterleavedData = deinterleavedMatrix(:)';
end
function symbols = qam16_mapper(bits, Eb)
    % Ensure the input length is a multiple of 4
    if mod(length(bits), 4) ~= 0
        error('Input length must be a multiple of 4.');
    end

    % Reshape bits into groups of 4
    bit_groups = reshape(bits, 4, []).';

    % Calculate the scaling factor based on Eb
    scaling_factor = sqrt(Eb / 10); % 10 is the average energy
for 16-QAM symbols

    % Map each bit group to a 16-QAM symbol
    symbol_map = [-3 - 3j, -3 - 1j, -3 + 1j, -3 + 3j, ...
                  -1 - 3j, -1 - 1j, -1 + 1j, -1 + 3j, ...
                   1 - 3j,  1 - 1j,  1 + 1j,  1 + 3j, ...
                   3 - 3j,  3 - 1j,  3 + 1j,  3 + 3j];

    binary_labels = de2bi(0:15, 4, 'left-msb'); % Generate all
possible 4-bit labels
    symbols = zeros(1, size(bit_groups, 1)); % Preallocate
symbols array

    for i = 1:size(bit_groups, 1)
        b = bit_groups(i, :);
        index = find(ismember(binary_labels, b, 'rows'));
        symbols(i) = scaling_factor * symbol_map(index);
    end
end

function bits = qam16_demapper(symbols, Eb)
    % Calculate the scaling factor based on Eb
    scaling_factor = sqrt(Eb / 10);

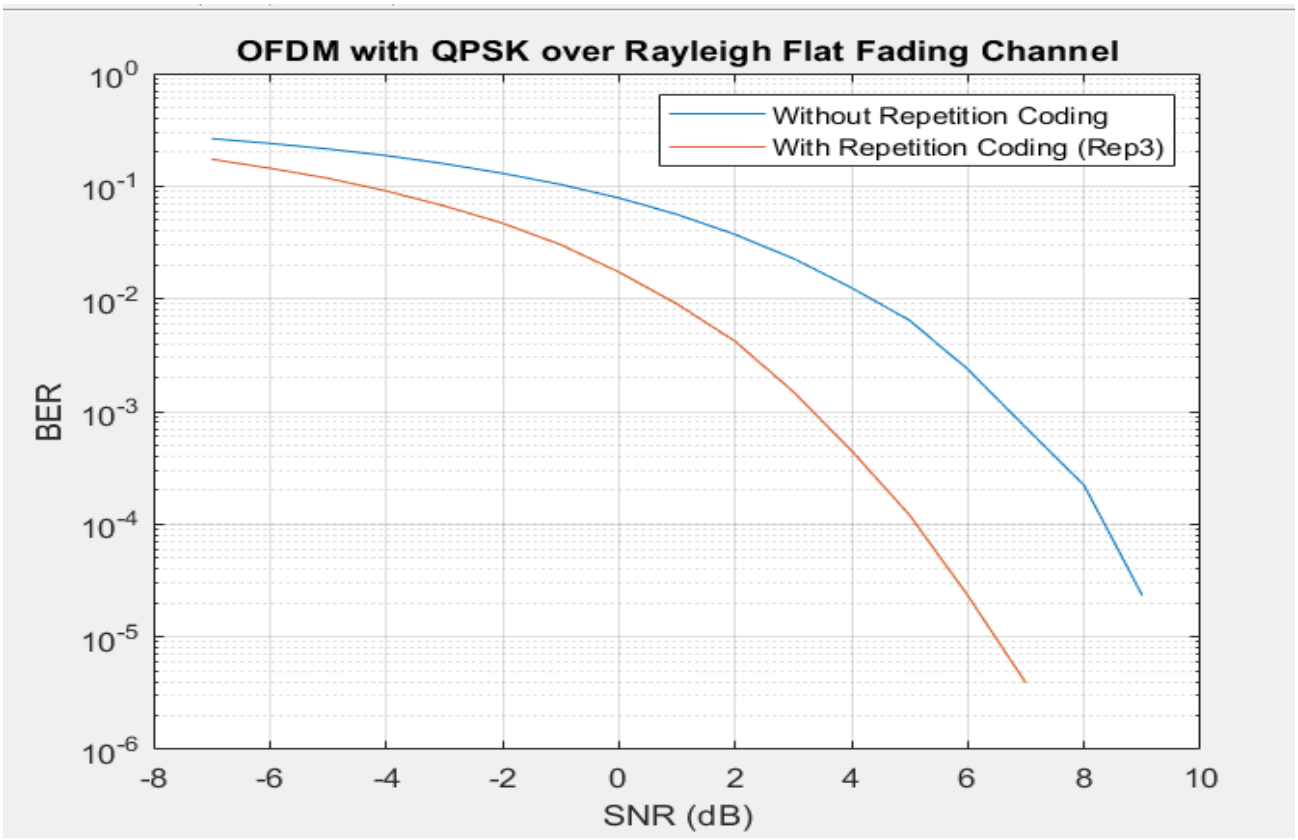
    % Normalize the received symbols
    normalized_symbols = symbols / scaling_factor;
```

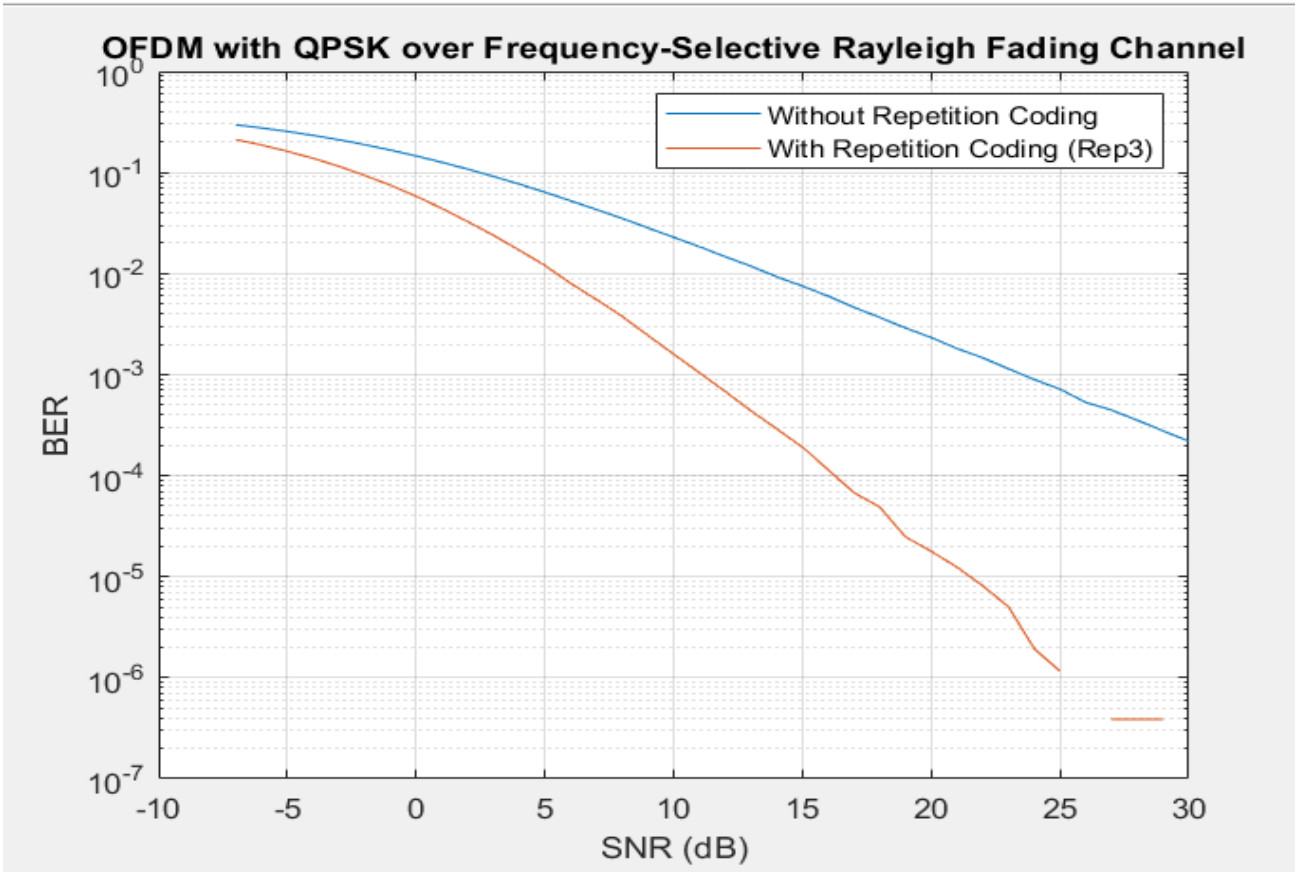
```
    % Define the constellation
    symbol_map = [-3 - 3j, -3 - 1j, -3 + 1j, -3 + 3j, ...
                  -1 - 3j, -1 - 1j, -1 + 1j, -1 + 3j, ...
                   1 - 3j,  1 - 1j,  1 + 1j,  1 + 3j, ...
                   3 - 3j,  3 - 1j,  3 + 1j,  3 + 3j];
    binary_labels = de2bi(0:15, 4, 'left-msb'); % Generate all
possible 4-bit labels

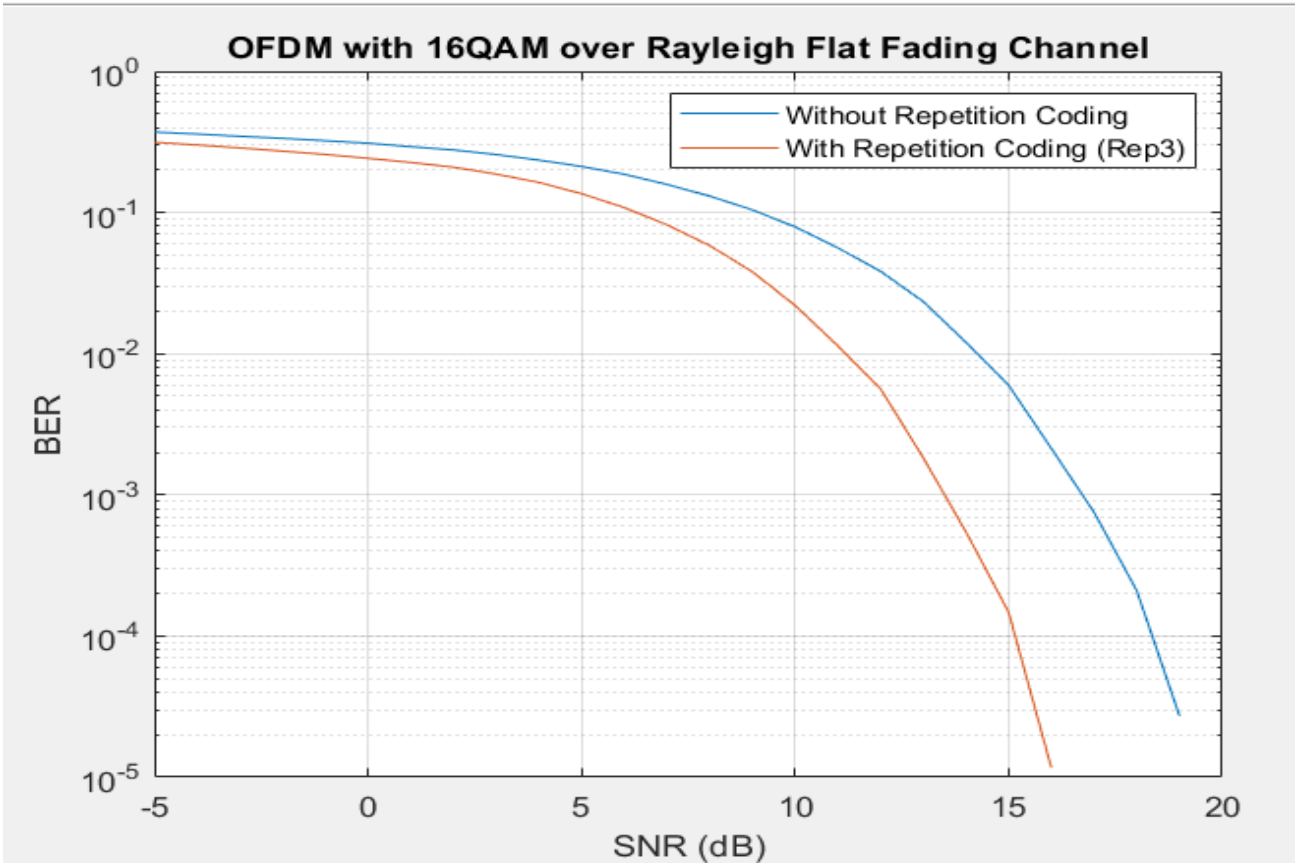
    % Initialize bits array
    bits = zeros(length(symbols) * 4, 1);

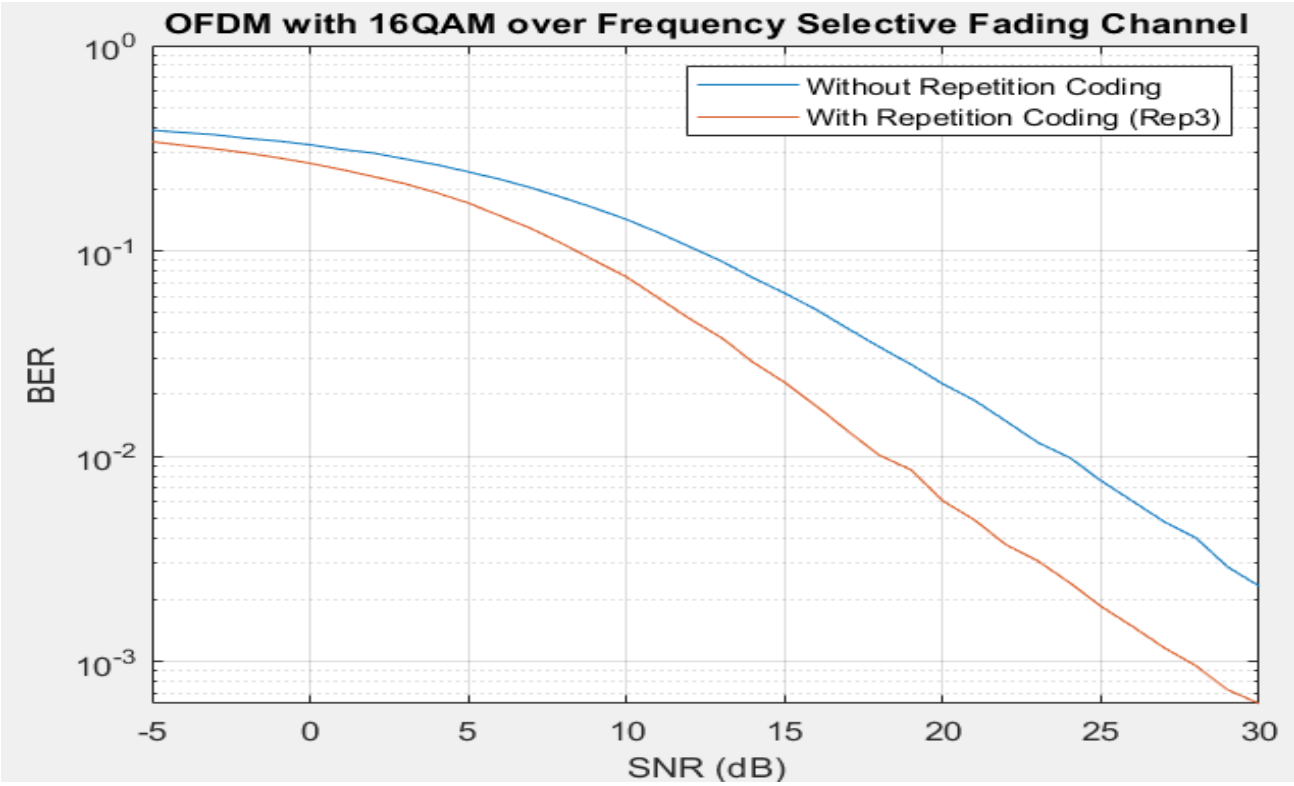
    for i = 1:length(normalized_symbols)
        % Find the closest constellation point
        [~, idx] = min(abs(normalized_symbols(i) - symbol_map));
        bits(4*i-3:4*i) = binary_labels(idx, :);
    end
end
```

Case B: Normalized noise results:




OFDM with QPSK over Frequency-Selective Rayleigh Fading Channel
Without Repetition Coding
With Repetition Coding (Rep3)
BER
SNR (dB)
10^0
10^-1
10^-2
10^-3
10^-4
10^-5
10^-6
10^-7
-10
-5
0
5
10
15
20
25
30


OFDM with 16QAM over Rayleigh Flat Fading Channel
Without Repetition Coding
With Repetition Coding (Rep3)
BER
SNR (dB)
10^0
10^-1
10^-2
10^-3
10^-4
10^-5
-5
0
5
10
15
20



2. Code Comments

- 16 different values of h were generated to represent frequency selective fading then use same h for each 10 sub-channels in each frame.
- Frequency selective fading states variations in channel response for various frequencies.
- I've done two cases one with normalization on noise by multiplying by 1/sqrt(128) and once without so we provided both results ,as there is no contradiction between them but all the difference is in the BER values but the logic behind them is the same.
- ```
  noise = sqrt(noisePower / 2).*(1/sqrt(128)) .*
  (randn(size(cyclicSignal)) + 1j *
  randn(size(cyclicSignal)));
  ```

3. Results and Observations:

- OFDM advantage is robustness to frequency selective fading and that's what makes it powerful.
- After coding better BER is achieved due to bit redundancy and capability to correct errors.
- When applied frequency selective fading on the system the performance wasn't satisfying , but the OFDM robustness against frequency selective fading takes place when an error correction capability is used too ,as bits repetition used in our case offers better BER due to spreading the same information on more than a subcarrier which offers redundancy over various fades (non-burst) which offers capability to correct and not being affected by the frequency selective fading and makes the system more robust to errors.
- It's obvious that when we have problems with fading as frequency selective fading for example, it's better to decrease the modulation order (number of symbols) as recognized in BER of frequency selective case of QPSK and 16QAM that QPSK offered better BER. That is what is known as BPSK for example is sent over more faded subchannels to reduce BER.
  ```