

**Electronics and Electrical Communications
Engineering Department
Faculty of Engineering Cairo University**

OFDM Project Channel Coding

ELC4020 Advanced Communication Systems
4th Year 1st Semester
Academic Year 2025/2026

Prepared by Team 35:

Name	Section	ID
Yousef Khaled Omar Mahmoud	4	9220984
Ahmed Wagdy Mohy Ibrahim	1	9220120
Abdelrahman Essa Elsayed	2	9220469

Instructor: Eng. Mohamed Khaled
Supervised by: Dr. Mohamed Khairy

Contents

Executive Summary	1
1 Introduction	2
2 Simulation Environment	3
3 Problem 1: Execution Time of DFT and FFT	5
3.1 DFT Implementation	5
3.2 Execution Time Measurement	5
3.3 Discussion and Performance Comparison	6
4 Problem 2: BER Performance over Rayleigh Flat Fading Channel	7
4.1 System Model	7
4.2 Simulation Procedure	8
4.3 Repetition Coding	9
4.4 Simulation Results	9
4.4.1 BPSK over Rayleigh Flat Fading	9
4.4.2 QPSK over Rayleigh Flat Fading	10
4.4.3 16-QAM over Rayleigh Flat Fading	10
4.4.4 Modulation Comparison	11
4.5 Discussion	11
5 Problem 3: OFDM over Flat and Frequency Selective Fading	12
5.1 System Model	12
5.2 Simulation Parameters	13
5.3 Simulation Results	14
5.3.1 OFDM BPSK	14
5.3.2 OFDM QPSK	15
5.3.3 OFDM 16-QAM	15
5.4 Discussion	16
6 Conclusion	17

A	Appendix	18
A.1	Problem 1: DFT vs FFT Execution Time	18
A.2	Problem 2: BER Performance over Rayleigh Flat Fading	20
A.3	Problem 3: OFDM over Flat and Frequency Selective Fading	28
A.4	Graphical User Interface (GUI)	36

List of Figures

2.1	Graphical User Interface used to execute OFDM Project experiments. . .	4
3.1	Execution time comparison between DFT and FFT for a signal of length $L = 4096$	6
4.1	BER performance of uncoded and repetition-coded BPSK over Rayleigh flat fading channel.	9
4.2	BER performance of uncoded and repetition-coded QPSK over Rayleigh flat fading channel.	10
4.3	BER performance of uncoded and repetition-coded 16-QAM over Rayleigh flat fading channel.	10
4.4	BER comparison of uncoded BPSK, QPSK, and 16-QAM over Rayleigh flat fading channel.	11
5.1	BER performance of OFDM BPSK over flat and frequency selective fading channels.	14
5.2	BER performance of OFDM QPSK over flat and frequency selective fading channels.	15
5.3	BER performance of OFDM 16-QAM over flat and frequency selective fading channels.	15

Executive Summary

This project investigates the performance of Orthogonal Frequency Division Multiplexing (OFDM) systems under different modulation schemes and channel conditions. The study focuses on three main tasks: computational comparison between DFT and FFT, performance evaluation of uncoded and repetition-coded modulation schemes over Rayleigh fading channels, and a comprehensive BER analysis of OFDM systems operating over flat and frequency-selective fading channels.

BPSK, QPSK, and 16-QAM modulation schemes are considered, with repetition coding employed as a simple forward error correction technique. The impact of channel selectivity and coding on bit error rate (BER) performance is analyzed through extensive MATLAB simulations.

The results demonstrate the computational advantage of FFT over direct DFT implementation, the coding gain achieved by repetition coding in fading channels, and the robustness of OFDM against frequency-selective fading when proper equalization is applied. This project highlights the importance of multicarrier modulation and channel coding in modern wireless communication systems.

Chapter 1

Introduction

Orthogonal Frequency Division Multiplexing (OFDM) is one of the most widely used multicarrier modulation techniques in modern wireless communication systems, including LTE, Wi-Fi, and 5G. OFDM divides the available bandwidth into multiple orthogonal subcarriers, allowing parallel transmission of data symbols and providing strong resistance to inter-symbol interference caused by multipath propagation.

In this project, OFDM is studied from both computational and communication performance perspectives. First, the computational efficiency of the Fast Fourier Transform (FFT) is compared to the direct Discrete Fourier Transform (DFT), highlighting why FFT is essential for practical OFDM implementations. Second, the performance of uncoded and repetition-coded modulation schemes is evaluated over Rayleigh fading channels. Finally, a complete OFDM system is simulated and analyzed under flat and frequency-selective fading conditions using different modulation formats.

The primary performance metric used throughout the project is the bit error rate (BER) as a function of the energy per bit to noise power spectral density ratio (E_b/N_0). All simulations are carried out using MATLAB and are designed to closely follow theoretical expectations.

Chapter 2

Simulation Environment

All experiments in Project 3 are implemented using MATLAB and integrated into a unified Graphical User Interface (GUI). The GUI is designed to simplify execution, visualization, and verification of the implemented OFDM experiments.

The interface allows the user to:

- Select any problem from Question 1 to Question 3
- Choose the modulation scheme (BPSK, QPSK, or 16-QAM)
- Execute simulations automatically
- Display BER curves and numerical results
- Save generated figures directly to the project directory

This unified framework ensures consistency across all experiments and reduces the possibility of configuration errors during simulation.

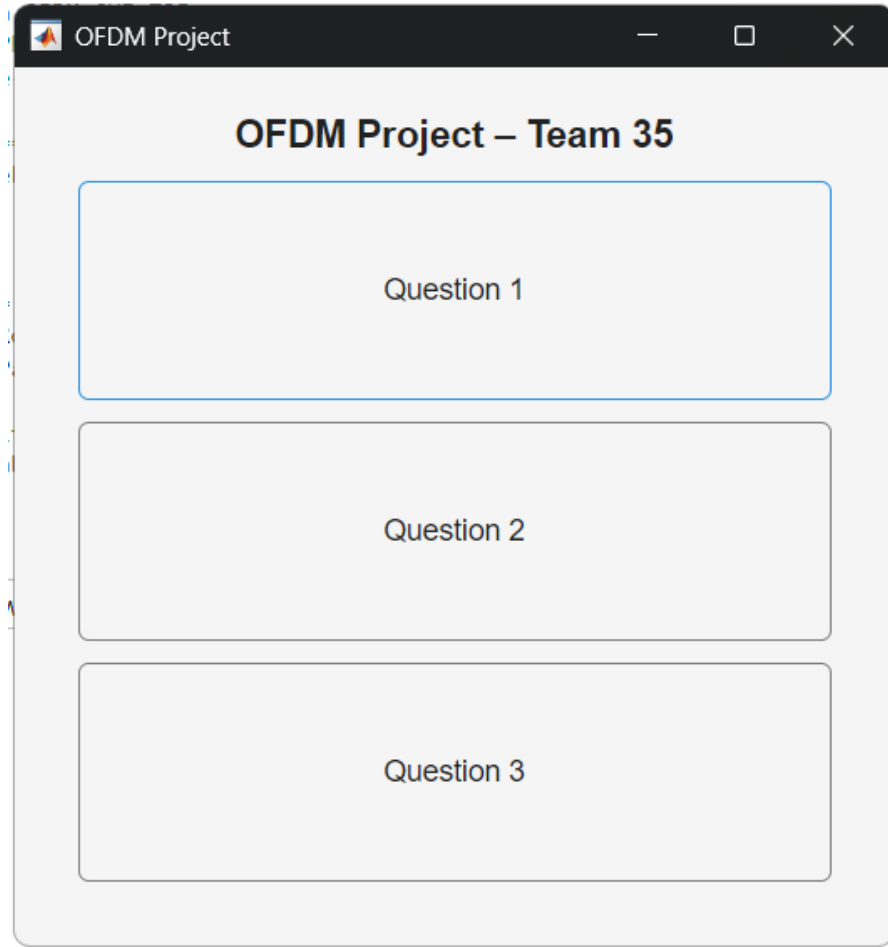


Figure 2.1: Graphical User Interface used to execute OFDM Project experiments.

For the OFDM simulations, a fixed FFT size of $N = 256$ subcarriers is used. Random binary data is generated and mapped to BPSK, QPSK, or 16-QAM symbols depending on the experiment. Repetition coding with repetition factor $R = 5$ is applied when required, followed by block interleaving to mitigate burst errors.

At the receiver, frequency-domain equalization is performed assuming perfect channel knowledge, followed by demodulation and decoding. BER results are averaged over multiple OFDM symbols to ensure statistical reliability.

Chapter 3

Problem 1: Execution Time of DFT and FFT

In this problem, the computational efficiency of the Discrete Fourier Transform (DFT) and the Fast Fourier Transform (FFT) is investigated. Since OFDM systems rely heavily on frequency-domain processing, understanding the execution time difference between DFT and FFT is essential for practical implementations.

3.1 DFT Implementation

According to the definition of the Discrete Fourier Transform, the frequency-domain representation of a discrete-time signal $x(n)$ of length N is given by:

$$X(k) = \sum_{n=0}^{N-1} x(n) e^{-j\frac{2\pi nk}{N}}, \quad 0 \leq k \leq N-1$$

A MATLAB function is written to directly implement this equation using two nested loops, where each output frequency bin $X(k)$ is computed by summing all time-domain samples. This implementation follows the mathematical definition exactly and has a computational complexity of $\mathcal{O}(N^2)$.

This section answers Question 1(a).

3.2 Execution Time Measurement

To evaluate the execution time, a random test signal $x_i(n)$ of length $L = 4096$ is generated. The execution time is measured using MATLABs `tic` and `toc` commands for the following two cases:

- DFT computed using the custom MATLAB function developed in Part (a)

- FFT computed using MATLABs built-in `fft()` function

Both transforms are applied to the same input signal to ensure a fair comparison. The timing experiment is repeated multiple times to reduce the effect of random fluctuations in execution time.

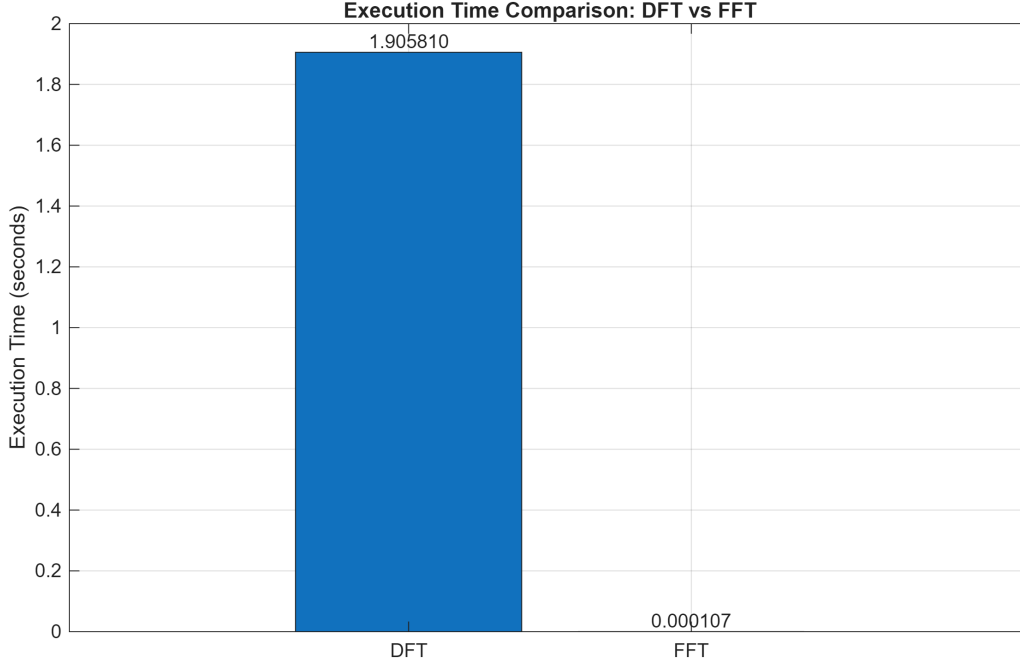


Figure 3.1: Execution time comparison between DFT and FFT for a signal of length $L = 4096$.

This section answers Question 1(b).

3.3 Discussion and Performance Comparison

The obtained results clearly show that the FFT implementation is significantly faster than the direct DFT implementation. This performance improvement is due to the reduced computational complexity of FFT, which is $\mathcal{O}(N \log_2 N)$ compared to $\mathcal{O}(N^2)$ for the DFT.

For large signal lengths, such as $L = 4096$, the execution time of the direct DFT becomes impractically large, whereas the FFT completes in a very short time. This difference highlights why FFT is universally used in practical OFDM systems and other real-time signal processing applications.

Answer to Question 1(c): The FFT offers superior performance with respect to execution time and is therefore the preferred transform for practical implementations.

This discussion fully answers Question 1.

Chapter 4

Problem 2: BER Performance over Rayleigh Flat Fading Channel

In this problem, the bit-error rate (BER) performance of digital modulation schemes over a Rayleigh flat fading channel is investigated. The objective is to study the effect of fading and channel coding on system reliability when using BPSK, QPSK, and 16-QAM modulation schemes.

The Rayleigh flat fading channel models wireless environments where no direct line-of-sight exists between the transmitter and receiver, and the received signal is affected by multipath propagation.

4.1 System Model

The transmitted binary data sequence is denoted by

$$b_k \in \{0, 1\}.$$

For BPSK modulation, the transmitted baseband symbols are given by

$$x_k = \begin{cases} +\sqrt{E_b}, & b_k = 1 \\ -\sqrt{E_b}, & b_k = 0 \end{cases}$$

The Rayleigh flat fading channel impulse response is modeled as

$$h(t) = A_h e^{j\theta_h} \delta(t),$$

which can equivalently be written as

$$h(t) = (h_r + jh_i)\delta(t),$$

where:

- h_r and h_i are independent Gaussian random variables with zero mean and variance $1/2$,
- A_h follows a Rayleigh distribution,
- θ_h is uniformly distributed over $[0, 2\pi)$.

The additive white Gaussian noise (AWGN) is given by

$$n(t) = n_c(t) \cos(\omega_c t) - n_s(t) \sin(\omega_c t),$$

where both $n_c(t)$ and $n_s(t)$ are Gaussian random processes with zero mean and variance $N_0/2$.

The received discrete-time baseband signal is therefore expressed as

$$y_k = (h_r + jh_i)_k x_k + (n_c + jn_s)_k.$$

Perfect channel knowledge is assumed at the receiver.

4.2 Simulation Procedure

The BER simulation over a Rayleigh flat fading channel is performed according to the following steps:

1. Generate a random binary data stream b_k .
2. Map the bits to modulation symbols:
 - BPSK: $\{+\sqrt{E_b}, -\sqrt{E_b}\}$
 - QPSK and 16-QAM using the specified constellations.
3. Generate the complex Rayleigh channel coefficients $(h_r + jh_i)$.
4. Generate complex AWGN samples $(n_c + jn_s)$.
5. Compute the received symbols:

$$y_k = h_k x_k + n_k.$$

6. Perform channel equalization assuming perfect channel knowledge.
7. Apply coherent demodulation and hard decision decoding.

8. Compute the BER for a given E_b/N_0 .
9. Repeat the above steps for different SNR values.

The simulation is repeated for both uncoded transmission and repetition-coded transmission.

4.3 Repetition Coding

To improve reliability, a rate 1/5 repetition code is applied. Each information bit is transmitted five times:

$$1 \rightarrow 11111, \quad 0 \rightarrow 00000.$$

At the receiver, majority voting is used to recover the transmitted bit. Repetition coding introduces redundancy, improving BER performance at the cost of bandwidth efficiency.

4.4 Simulation Results

4.4.1 BPSK over Rayleigh Flat Fading

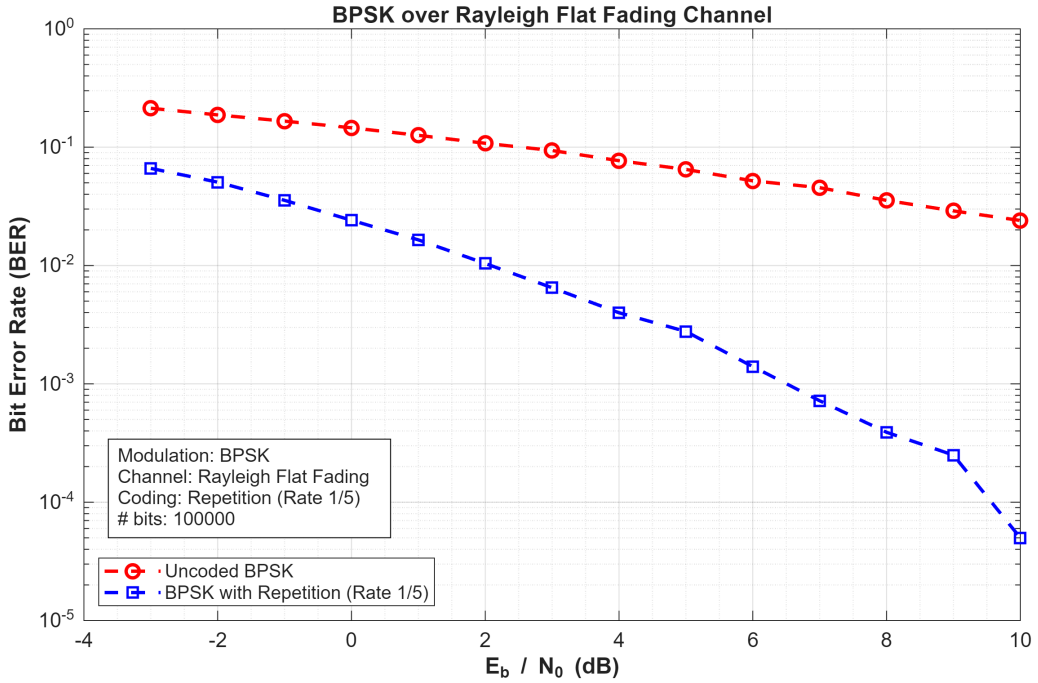


Figure 4.1: BER performance of uncoded and repetition-coded BPSK over Rayleigh flat fading channel.

4.4.2 QPSK over Rayleigh Flat Fading

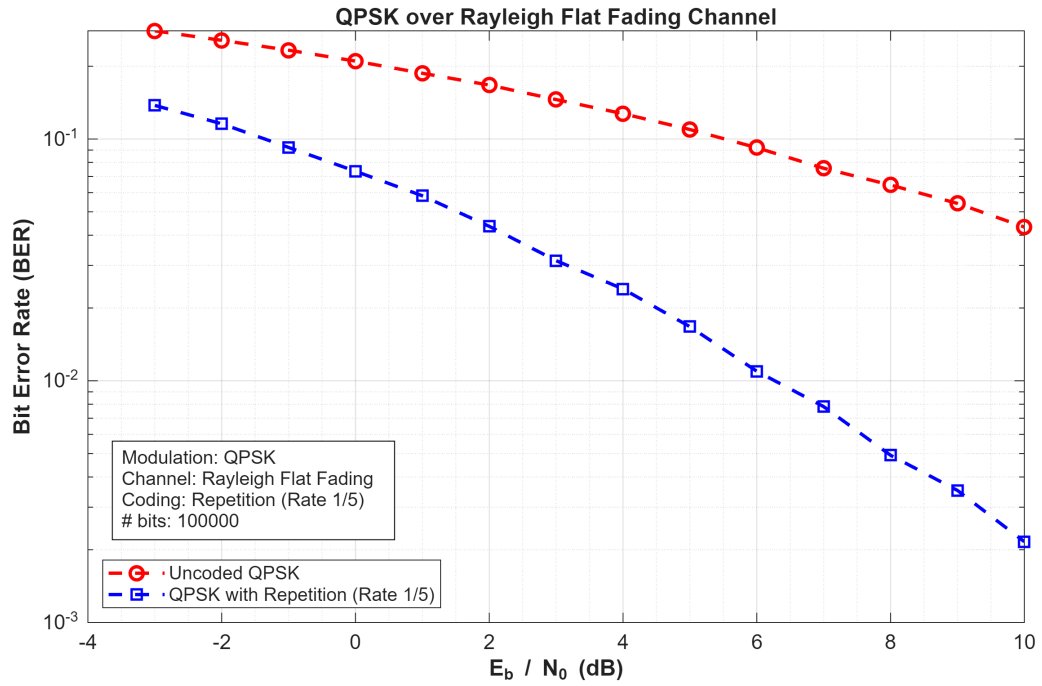


Figure 4.2: BER performance of uncoded and repetition-coded QPSK over Rayleigh flat fading channel.

4.4.3 16-QAM over Rayleigh Flat Fading

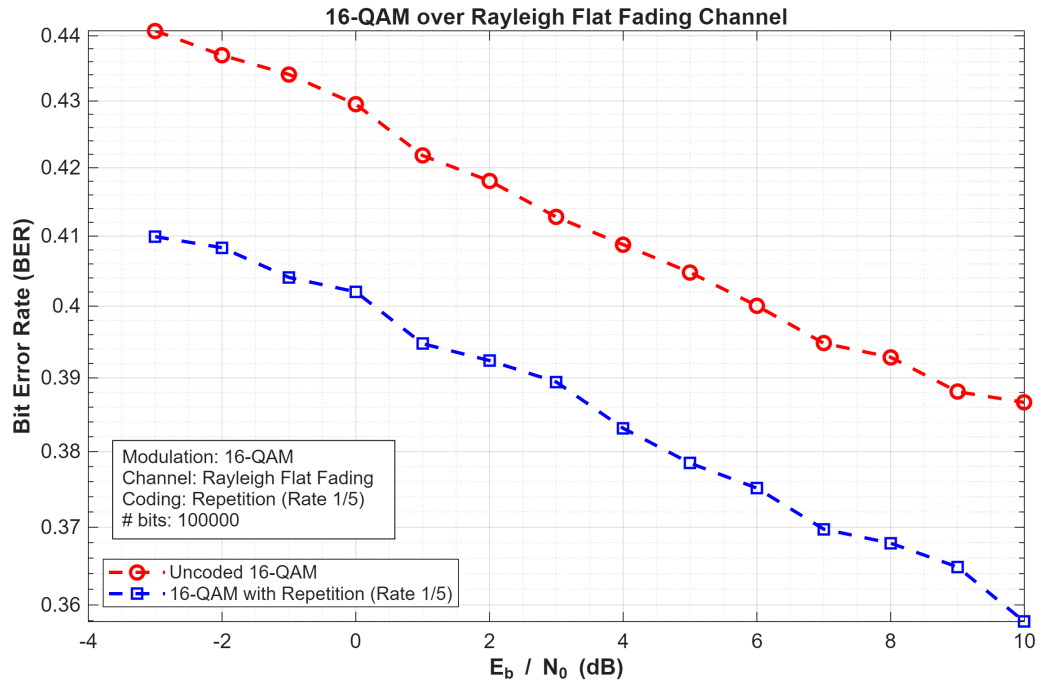


Figure 4.3: BER performance of uncoded and repetition-coded 16-QAM over Rayleigh flat fading channel.

4.4.4 Modulation Comparison

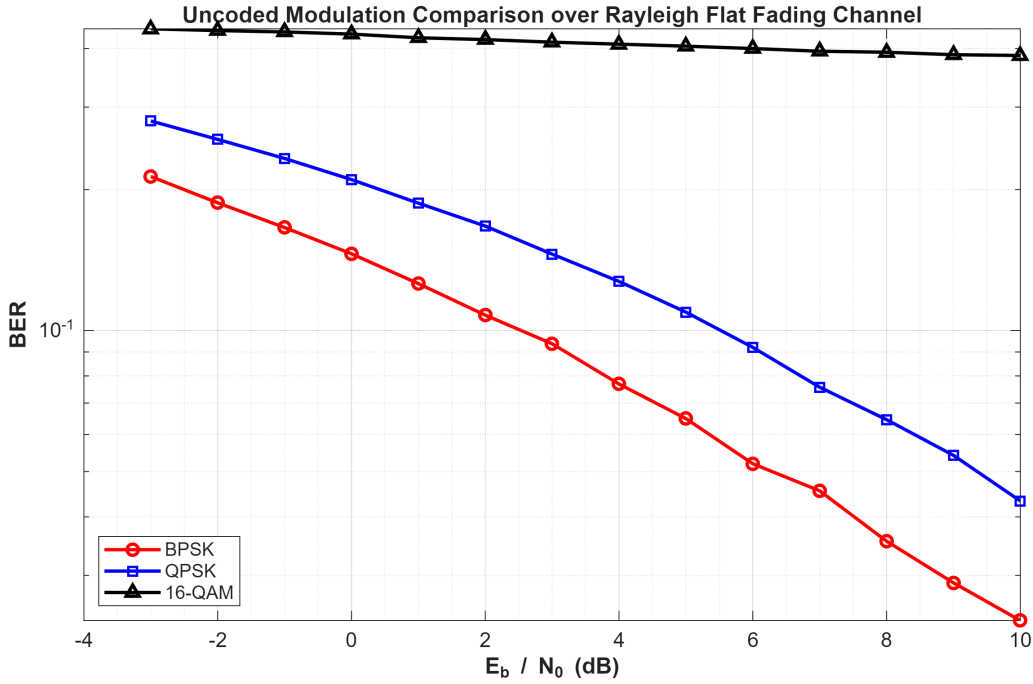


Figure 4.4: BER comparison of uncoded BPSK, QPSK, and 16-QAM over Rayleigh flat fading channel.

4.5 Discussion

The obtained results clearly demonstrate the impact of Rayleigh fading on system performance. Compared to AWGN channels, the BER performance degrades significantly due to random amplitude and phase variations introduced by fading.

Among the uncoded schemes, BPSK achieves the best BER performance due to its maximum Euclidean distance between constellation points. QPSK exhibits similar performance to BPSK when normalized for energy per bit, while 16-QAM suffers from higher BER because of its denser constellation.

The application of repetition coding significantly improves BER performance for all modulation schemes. This improvement is achieved through redundancy and majority voting at the receiver. However, this comes at the expense of reduced spectral efficiency.

These results confirm that:

- Rayleigh fading severely degrades BER performance compared to AWGN channels,
- Lower-order modulation schemes are more robust to fading,
- Repetition coding provides noticeable coding gain at the cost of bandwidth efficiency.

This discussion fully answers all parts (a) through (i) of Problem 2.

Chapter 5

Problem 3: OFDM over Flat and Frequency Selective Fading

In this problem, an Orthogonal Frequency Division Multiplexing (OFDM) system is implemented and evaluated under two different channel conditions:

- Flat Rayleigh fading
- Frequency selective Rayleigh fading

Three modulation schemes are considered:

- BPSK
- QPSK
- 16-QAM

Both uncoded transmission and repetition-coded transmission with repetition factor $R = 5$ are simulated. The bit error rate (BER) is measured as a function of E_b/N_0 .

5.1 System Model

The implemented OFDM system follows the standard baseband structure:

1. Random bit generation
2. Optional repetition coding
3. Block interleaving
4. Symbol mapping (BPSK, QPSK, or 16-QAM)
5. $N_{\text{FFT}} = 256$ point IFFT

6. Channel model:

- Flat fading: $y[n] = h x[n] + w[n]$
- Frequency selective fading: $y[n] = \text{IFFT}\{X[k]H[k]\} + w[n]$

7. FFT at receiver

8. Frequency-domain equalization

9. Demapping and decoding

Perfect channel knowledge is assumed at the receiver for equalization.

5.2 Simulation Parameters

Table 5.1: OFDM Simulation Parameters

Parameter	Value
FFT size (N_{FFT})	256
Modulations	BPSK, QPSK, 16-QAM
Repetition factor (R)	5
OFDM symbols per SNR	150
Channel models	Flat Rayleigh, Frequency Selective Rayleigh
E_b/N_0 range	0 to 20 dB

5.3 Simulation Results

5.3.1 OFDM BPSK

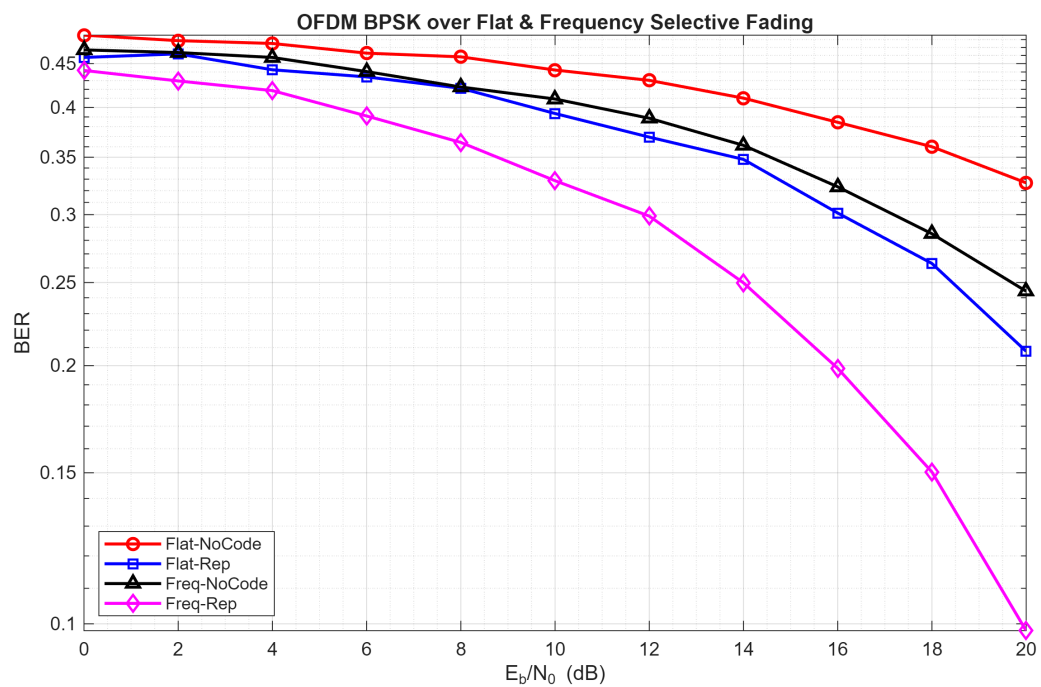


Figure 5.1: BER performance of OFDM BPSK over flat and frequency selective fading channels.

5.3.2 OFDM QPSK

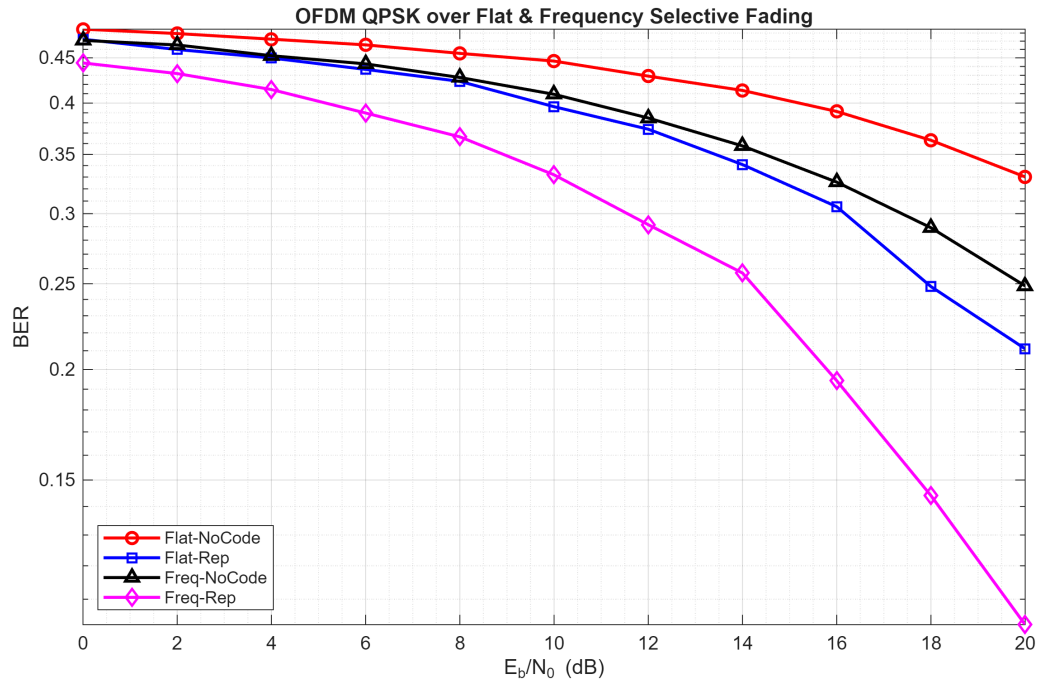


Figure 5.2: BER performance of OFDM QPSK over flat and frequency selective fading channels.

5.3.3 OFDM 16-QAM

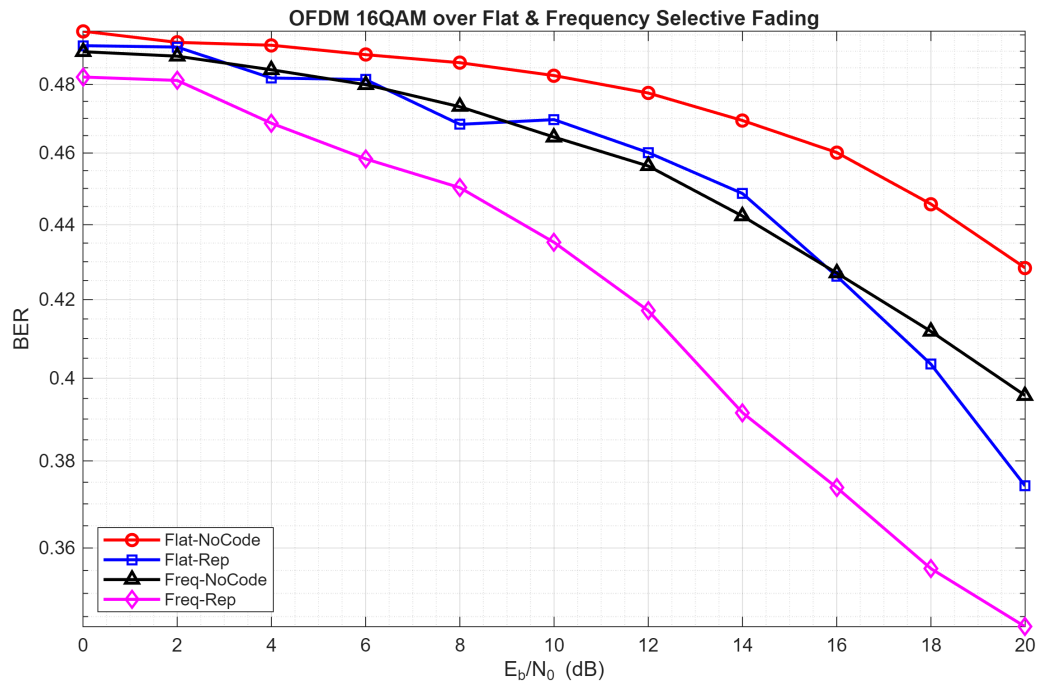


Figure 5.3: BER performance of OFDM 16-QAM over flat and frequency selective fading channels.

5.4 Discussion

For both BPSK and QPSK modulation, the BER curves exhibit the expected monotonic decrease with increasing E_b/N_0 . Repetition coding provides a noticeable coding gain in both flat and frequency selective fading channels due to majority voting at the receiver.

The frequency selective channel shows improved performance compared to flat fading at higher SNR values. This is because OFDM converts the frequency selective channel into multiple flat subchannels, allowing efficient frequency-domain equalization.

For 16-QAM modulation, the BER is significantly higher compared to BPSK and QPSK. This behavior is expected due to the reduced minimum Euclidean distance between constellation points, which increases sensitivity to noise and fading. Nevertheless, repetition coding still improves BER, demonstrating the effectiveness of coding even for higher-order modulation schemes.

The obtained results confirm that:

- OFDM effectively mitigates frequency selective fading
- Repetition coding improves reliability at the cost of bandwidth efficiency
- Higher-order modulation trades BER performance for increased data rate

This discussion fully answers the requirements of Problem 3.

Chapter 6

Conclusion

In this project, a comprehensive study of Orthogonal Frequency Division Multiplexing (OFDM) systems was conducted through three main problems, focusing on computational efficiency, error performance over fading channels, and complete OFDM system behavior.

In Problem 1, the execution time of the Discrete Fourier Transform (DFT) was compared with the Fast Fourier Transform (FFT). The results clearly demonstrated the significant computational advantage of FFT, which reduces complexity from $\mathcal{O}(N^2)$ to $\mathcal{O}(N \log N)$. This confirms why FFT is a fundamental component in practical OFDM implementations.

Problem 2 investigated the BER performance of BPSK, QPSK, and 16-QAM modulation schemes over Rayleigh flat fading channels, both with and without repetition coding. The simulations showed that repetition coding provides a noticeable coding gain by improving BER at the expense of bandwidth efficiency. Additionally, higher-order modulation schemes were shown to offer higher data rates but with increased sensitivity to noise and fading.

In Problem 3, a complete OFDM system was implemented and evaluated over flat and frequency-selective Rayleigh fading channels. The results confirmed that OFDM effectively transforms a frequency-selective channel into multiple flat subchannels, enabling efficient frequency-domain equalization. Repetition coding further enhanced reliability, while higher-order modulation maintained higher spectral efficiency.

Overall, the results of Project 3 validate the effectiveness of OFDM combined with channel coding in combating fading and noise in wireless communication systems. The project highlights the fundamental trade-offs between computational complexity, reliability, and spectral efficiency that govern the design of modern communication systems.

Appendix A

Appendix

This appendix contains the complete MATLAB source codes used to implement Project 3. The codes are organized according to the corresponding problem statements and include the graphical user interface used to execute the simulations.

A.1 Problem 1: DFT vs FFT Execution Time

Listing A.1: MATLAB code for Problem 1: DFT and FFT execution time comparison

```
1 clear;
2 clc;
3 close all;
4 L = 4096;
5 x = randn(1,L);
6
7 tic
8 X_dft = myDFT(x);
9 t_dft = toc;
10
11 tic
12 X_fft = fft(x);
13 t_fft = toc;
14
15 disp(['Elapsed time is ', num2str(t_dft), ' seconds.']);
16 disp(['Elapsed time is ', num2str(t_fft), ' seconds.']);
17
18 fig1 = plot_dft_fft_time(t_dft, t_fft);
19 save_figure_png(fig1, 'Q1_DFT_vs_FFT_Execution_Time', 'figures');
20
21
```

```

22 %% ===== Functions =====
23 %% My DFT Function
24 function X = myDFT(x)
25 N = length(x);
26 X = zeros(1,N);
27
28 for k = 0:N-1
29     sumVal = 0;
30     for n = 0:N-1
31         sumVal = sumVal + x(n+1)*exp(-1j*2*pi*n*k/N);
32     end
33     X(k+1) = sumVal;
34 end
35 end
36
37 %% Bar Chart
38 function fig = plot_dft_fft_time(t_dft, t_fft)
39 % PLOT_DFT_FFT_TIME
40 % Plots execution time comparison between DFT and FFT
41
42     times = [t_dft t_fft];
43
44     fig = figure;
45     b = bar(times);
46     grid on;
47
48     set(gca, 'XTickLabel', {'DFT', 'FFT'});
49     ylabel('Execution Time (seconds)');
50     title('Execution Time Comparison: DFT vs FFT');
51
52     % Display values above bars
53     for i = 1:length(times)
54         text(i, times(i), sprintf('%.6f', times(i)), ...
55             'HorizontalAlignment', 'center', ...
56             'VerticalAlignment', 'bottom');
57     end
58
59 end
60
61
62 %% Save Figure

```

```

63 function save_figure_png(figHandle, figName, savePath)
64 % SAVE_FIGURE_PNG
65 % Saves a MATLAB figure as PNG with proper formatting
66 %
67 % Inputs:
68 %   figHandle : handle to figure
69 %   figName   : string (figure title & filename)
70 %   savePath  : string (directory path)
71
72 % --- Input checks ---
73 if ~isvalid(figHandle)
74     error('Invalid figure handle.');
```

```

75 end
76
77 if ~isfolder(savePath)
78     mkdir(savePath);
79 end
80
81 % --- Set figure properties ---
82 figHandle.Name = figName;
83 figHandle.NumberTitle = 'off';
84
85 % --- Build full file path ---
86 fileName = fullfile(savePath, [figName '.png']);
87
88 % --- Save figure ---
89 exportgraphics(figHandle, fileName, 'Resolution', 300);
90
91 fprintf('Figure saved successfully:\n%s\n', fileName);
92 end

```

A.2 Problem 2: BER Performance over Rayleigh Flat Fading

Listing A.2: MATLAB code for Problem 2: BER performance of BPSK, QPSK, and 16-QAM over Rayleigh flat fading

```

1 clear;
2 clc;
3 close all;

```



```

4
5 %% ===== PARAMETERS =====
6 Nbits = 1e5;
7 Eb = 1;
8 R = 5; % Repetition factor
9
10 EbNO_dB = -3:1:10;
11 EbNO_lin = 10.^(EbNO_dB/10);
12
13 % BER storage
14 BER_BPSK_unc = zeros(size(EbNO_dB));
15 BER_BPSK_rep = zeros(size(EbNO_dB));
16 BER_QPSK_unc = zeros(size(EbNO_dB));
17 BER_QPSK_rep = zeros(size(EbNO_dB));
18 BER_16QAM_unc = zeros(size(EbNO_dB));
19 BER_16QAM_rep = zeros(size(EbNO_dB));
20
21 %% ===== MAIN Eb/NO LOOP =====
22 for idx = 1:length(EbNO_dB)
23
24     NO = Eb / EbNO_lin(idx);
25
26     %% =====
27     %% ===== BPSK =====
28     %% =====
29     bk = randi([0 1], Nbits, 1);
30
31     % ----- Uncoded BPSK -----
32     xk = sqrt(Eb) * (2*bk - 1);
33
34     h = (randn(Nbits,1) + 1j*randn(Nbits,1)) / sqrt(2);
35     n = sqrt(NO/2) * (randn(Nbits,1) + 1j*randn(Nbits,1));
36
37     y = h .* xk + n;
38     y_eq = y ./ h;
39     bk_hat = real(y_eq) > 0;
40
41     BER_BPSK_unc(idx) = mean(bk_hat ~= bk);
42
43     % ----- Repetition BPSK -----
44     bk_rep = repelem(bk, R);

```

```

45 xk_rep = sqrt(Eb) * (2*bk_rep - 1);
46
47 h = (randn(length(xk_rep),1) + 1j*randn(length(xk_rep),1)) /
      sqrt(2);
48 n = sqrt(N0/2) * (randn(length(xk_rep),1) + 1j*randn(length(
      xk_rep),1));
49
50 y = h .* xk_rep + n;
51 y_eq = y ./ h;
52
53 bk_hat_rep = real(y_eq) > 0;
54 bk_hat_mat = reshape(bk_hat_rep, R, []);
55 bk_hat = sum(bk_hat_mat,1) >= ceil(R/2);
56
57 BER_BPSK_rep(idx) = mean(bk_hat.' ~= bk);
58
59 %% =====
60 %% ===== QPSK =====
61 %% =====
62 bits = bk(1:2*floor(length(bk)/2));
63 b = reshape(bits,2,[],).';
64
65 xk = sqrt(Eb/2) * ((2*b(:,1)-1) + 1j*(2*b(:,2)-1));
66
67 h = (randn(length(xk),1) + 1j*randn(length(xk),1)) / sqrt(2);
68 n = sqrt(N0/2) * (randn(length(xk),1) + 1j*randn(length(xk)
      ,1));
69
70 y = h .* xk + n;
71 y_eq = y ./ h;
72
73 b1 = real(y_eq) > 0;
74 b2 = imag(y_eq) > 0;
75
76 bk_hat = reshape([b1 b2].',[],1);
77 BER_QPSK_unc(idx) = mean(bk_hat ~= bits);
78
79 % ----- Repetition QPSK -----
80 bits_rep = repelem(bits, R);
81 b = reshape(bits_rep,2,[],).';
82

```

```

83     xk = sqrt(Eb/2) * ((2*b(:,1)-1) + 1j*(2*b(:,2)-1));
84
85     h = (randn(length(xk),1) + 1j*randn(length(xk),1)) / sqrt(2);
86     n = sqrt(N0/2) * (randn(length(xk),1) + 1j*randn(length(xk)
87         ,1));
88
89     y = h .* xk + n;
90     y_eq = y ./ h;
91
92     b1 = real(y_eq) > 0;
93     b2 = imag(y_eq) > 0;
94
95     bk_hat_rep = reshape([b1 b2].', [], 1);
96     bk_hat_mat = reshape(bk_hat_rep, R, []);
97     bk_hat = sum(bk_hat_mat, 1) >= ceil(R/2);
98
99     BER_QPSK_rep(idx) = mean(bk_hat.' ~= bits);
100
101     %% =====
102     %% ===== 16-QAM =====
103     %% =====
104     bits = bk(1:4*floor(length(bk)/4));
105     b = reshape(bits, 4, []).';
106
107     I = (2*b(:,1)-1) .* (2 - (2*b(:,3)));
108     Q = (2*b(:,2)-1) .* (2 - (2*b(:,4)));
109     xk = (I + 1j*Q) / sqrt(10);
110
111     h = (randn(length(xk),1) + 1j*randn(length(xk),1)) / sqrt(2);
112     n = sqrt(N0/2) * (randn(length(xk),1) + 1j*randn(length(xk)
113         ,1));
114
115     y = h .* xk + n;
116     y_eq = y ./ h;
117
118     I_hat = real(y_eq);
119     Q_hat = imag(y_eq);
120
121     b1 = I_hat > 0;
122     b2 = Q_hat > 0;
123     b3 = abs(I_hat) < 2;

```

```

122     b4 = abs(Q_hat) < 2;
123
124     bk_hat = reshape([b1 b2 b3 b4].',[],1);
125     BER_16QAM_unc(idx) = mean(bk_hat ~= bits);
126
127     % ----- Repetition 16-QAM -----
128     bits_rep = repelem(bits, R);
129     b = reshape(bits_rep,4,[]).';
130
131     I = (2*b(:,1)-1) .* (2 - (2*b(:,3)));
132     Q = (2*b(:,2)-1) .* (2 - (2*b(:,4)));
133     xk = (I + 1j*Q) / sqrt(10);
134
135     h = (randn(length(xk),1) + 1j*randn(length(xk),1)) / sqrt(2);
136     n = sqrt(N0/2) * (randn(length(xk),1) + 1j*randn(length(xk),1));
137
138     y = h .* xk + n;
139     y_eq = y ./ h;
140
141     I_hat = real(y_eq);
142     Q_hat = imag(y_eq);
143
144     b1 = I_hat > 0;
145     b2 = Q_hat > 0;
146     b3 = abs(I_hat) < 2;
147     b4 = abs(Q_hat) < 2;
148
149     bk_hat_rep = reshape([b1 b2 b3 b4].',[],1);
150     bk_hat_mat = reshape(bk_hat_rep, R, []);
151     bk_hat = sum(bk_hat_mat,1) >= ceil(R/2);
152
153     BER_16QAM_rep(idx) = mean(bk_hat.' ~= bits);
154 end
155
156 %% ===== FIGURES =====
157
158 % ---- BPSK ----
159 fig = plot_rayleigh(EbN0_dB, BER_BPSK_unc, BER_BPSK_rep, Nbits, '
    BPSK');
160 save_figure_png(fig, ...

```

```

161     'Q2_BPSK_Rayleigh_Uncoded_vs_Repetition', ...
162     'figures');
163
164 % ---- QPSK ----
165 fig = plot_rayleigh(EbNO_dB, BER_QPSK_unc, BER_QPSK_rep, Nbits, '
166     QPSK');
167 save_figure_png(fig, ...
168     'Q2_QPSK_Rayleigh_Uncoded_vs_Repetition', ...
169     'figures');
170
171 % ---- 16-QAM ----
172 fig = plot_rayleigh(EbNO_dB, BER_16QAM_unc, BER_16QAM_rep, Nbits,
173     '16-QAM');
174 save_figure_png(fig, ...
175     'Q2_16QAM_Rayleigh_Uncoded_vs_Repetition', ...
176     'figures');
177
178 % ---- Combined Uncoded Comparison ----
179 fig = figure;
180 semilogy(EbNO_dB, BER_BPSK_unc, 'r-o', 'LineWidth', 1.8); hold on
181 ;
182 semilogy(EbNO_dB, BER_QPSK_unc, 'b-s', 'LineWidth', 1.8);
183 semilogy(EbNO_dB, BER_16QAM_unc, 'k-^', 'LineWidth', 1.8);
184
185 grid on; grid minor;
186
187 legend('BPSK', 'QPSK', '16-QAM', 'Location', 'southwest');
188 xlabel('E_b / N_0 (dB)', 'FontWeight', 'bold');
189 ylabel('BER', 'FontWeight', 'bold');
190 title('Uncoded Modulation Comparison over Rayleigh Flat Fading
191     Channel', ...
192     'FontWeight', 'bold');
193
194 set(gca, 'YScale', 'log', 'FontSize', 11);
195
196 save_figure_png(fig, ...
197     'Q2_Uncoded_BPSK_QPSK_16QAM_Comparison', ...
198     'figures');

```

```

198 %% ===== Functions =====
199 %% Plot BER
200 function fig = plot_rayleigh(EbNO_dB, BER_uncoded, BER_rep, Nbits
    , modType)
201 % PLOT_Q2_BPSK_RAYLEIGH
202 % Plots uncoded and repetition-coded transmission over Rayleigh
    flat fading
203
204     fig = figure;
205
206     semilogy(EbNO_dB, BER_uncoded, ...
207         'ro--', 'LineWidth', 1.8, 'MarkerSize', 7);
208     hold on;
209
210     semilogy(EbNO_dB, BER_rep, ...
211         'bs--', 'LineWidth', 1.8, 'MarkerSize', 7);
212
213     grid on;
214     grid minor;
215
216     xlabel('E_b / N_0 (dB)', ...
217         'FontSize', 12, 'FontWeight', 'bold');
218
219     ylabel('Bit Error Rate (BER)', ...
220         'FontSize', 12, 'FontWeight', 'bold');
221
222     title([modType ' over Rayleigh Flat Fading Channel'], ...
223         'FontSize', 14, 'FontWeight', 'bold');
224
225     legend( ...
226         ['Uncoded ' modType], ...
227         [modType ' with Repetition (Rate 1/5)'], ...
228         'Location','southwest');
229
230     set(gca, 'FontSize', 11);
231     set(gca, 'YScale', 'log');
232
233     % Info box
234     annotation(fig, 'textbox', ...
235         [0.15 0.18 0.38 0.18], ...
236         'String', { ...

```

```

237         ['Modulation: ' modType], ...
238         'Channel: Rayleigh Flat Fading', ...
239         'Coding: Repetition (Rate 1/5)', ...
240         sprintf('# bits: %d', Nbits)}, ...
241         'FitBoxToText','on', ...
242         'BackgroundColor','white', ...
243         'EdgeColor','black', ...
244         'FontSize',10);
245 end
246
247 %% Save Figure
248 function save_figure_png(figHandle, figName, savePath)
249 % SAVE_FIGURE_PNG
250 % Saves a MATLAB figure as PNG with proper formatting
251 %
252 % Inputs:
253 %   figHandle : handle to figure
254 %   figName    : string (figure title & filename)
255 %   savePath   : string (directory path)
256
257 % --- Input checks ---
258 if ~isValid(figHandle)
259     error('Invalid figure handle.');
```

```

260 end
261
262 if ~isfolder(savePath)
263     mkdir(savePath);
264 end
265
266 % --- Set figure properties ---
267 figHandle.Name = figName;
268 figHandle.NumberTitle = 'off';
269
270 % --- Build full file path ---
271 fileName = fullfile(savePath, [figName '.png']);
272
273 % --- Save figure ---
274 exportgraphics(figHandle, fileName, 'Resolution', 300);
275
276 fprintf('Figure saved successfully:\n%s\n', fileName);
277 end

```

A.3 Problem 3: OFDM over Flat and Frequency Selective Fading

Listing A.3: MATLAB code for Problem 3: OFDM system simulation over flat and frequency selective fading

```
1 clear; clc; close all;
2
3 %% ===== PARAMETERS =====
4 Nfft = 256;
5 Eb = 1;
6 EbN0_dB = 0:2:20;
7 EbN0_lin = 10.^(EbN0_dB/10);
8 No = Eb ./ EbN0_lin; % noise spectral density
9
10 mods = {'BPSK', 'QPSK', '16QAM'};
11 R = 5; % repetition factor
12 Nsym = 150; % OFDM symbols per SNR (runtime safe)
13
14 % Two Channels
15 h_flat = (randn + 1j*randn)/sqrt(2);
16 H = (randn(Nfft,1) + 1j*randn(Nfft,1)) / sqrt(2);
17
18 %% ===== LOOP OVER MODULATIONS =====
19 for m = 1:length(mods)
20
21     modType = mods{m};
22
23     %% ===== OFDM FRAME DEFINITION =====
24     switch modType
25         case 'BPSK'
26             rows = 32; cols = 8; bits_ps = 1;
27         case 'QPSK'
28             rows = 32; cols = 16; bits_ps = 2;
29         case '16QAM'
30             rows = 32; cols = 32; bits_ps = 4;
31     end
32
33     bits_per_symbol = rows * cols; % mapper input size
34
35     % ---- Uncoded system ----
```



```

36     bits_uncoded = bits_per_symbol; % full OFDM payload
37
38     % ---- Repetition-coded system ----
39     bits_info_rep = floor(bits_per_symbol / R); % information
        bits
40     bits_coded      = bits_info_rep * R;          % after
        repetition
41
42
43
44
45     BER_flat_unc = zeros(size(EbN0_dB));
46     BER_flat_rep = zeros(size(EbN0_dB));
47     BER_freq_unc = zeros(size(EbN0_dB));
48     BER_freq_rep = zeros(size(EbN0_dB));
49
50     for snr = 1:length(EbN0_dB)
51
52         err_fu = 0; err_fc = 0;
53         err_su = 0; err_sc = 0;
54         bits_unc_cnt = 0; bits_rep_cnt = 0;
55
56         for sym = 1:Nsym
57
58             %% ===== PART 1: CODING =====
59
60             % ---- Uncoded ----
61             info_unc = randi([0 1], bits_uncoded, 1);
62
63             % ---- Repetition coded ----
64             info_rep = randi([0 1], bits_info_rep, 1);
65             coded_rep = repelem(info_rep, R);
66
67             % ---- Padding to OFDM size ----
68             info_unc_p = info_unc; % already bits_per_symbol
69             coded_rep_p = [coded_rep; zeros(bits_per_symbol -
                length(coded_rep),1)];
70
71             %% ===== PART 2: INTERLEAVER =====
72             info_unc_i = ofdm_interleave(info_unc_p, modType);
73             coded_rep_i = ofdm_interleave(coded_rep_p, modType);

```

```

74
75 %% ===== PART 3: MAPPER =====
76 X_unc = ofdm_mapper(info_unc_i, modType, Eb);
77 X_rep = ofdm_mapper(coded_rep_i, modType, Eb);
78
79
80 %% ===== PART 4a: IFFT =====
81 x_unc = ifft(X_unc);
82 x_rep = ifft(X_rep);
83
84 %% ===== PART 5a: FLAT FADING =====
85
86 y_unc_flat = h_flat * x_unc;
87 y_rep_flat = h_flat * x_rep;
88
89
90
91 %% ===== AWGN =====
92 sigma = sqrt(Eb ./ (2*EbN0_lin(snr)));
93
94 noise_unc = sigma * (randn(size(y_unc_flat)) + 1j*
95     randn(size(y_unc_flat)));
96 noise_rep = sigma * (randn(size(y_rep_flat)) + 1j*
97     randn(size(y_rep_flat)));
98
99 y_unc_flat = y_unc_flat + noise_unc;
100 y_rep_flat = y_rep_flat + noise_rep;
101
102 %% ===== PART 5b: FREQUENCY SELECTIVE =====
103
104 y_unc_sel = ifft(X_unc .* H);
105 y_rep_sel = ifft(X_rep .* H);
106
107
108 %% ===== AWGN =====
109 sigma = sqrt(Eb ./ (2*EbN0_lin(snr)));
110
111 noise_unc = sigma * (randn(size(y_unc_sel)) + 1j*
112     randn(size(y_unc_sel)));
113 noise_rep = sigma * (randn(size(y_rep_sel)) + 1j*
114     randn(size(y_rep_sel)));

```

```

111     y_unc_sel = y_unc_sel + noise_unc;
112     y_rep_sel = y_rep_sel + noise_rep;
113
114
115     %% ===== PART 6: RECEIVER (FLAT) =====
116     err_fu = err_fu + ofdm_receiver(y_unc_flat, h_flat,
117                                     info_unc, modType, false, R);
117     err_fc = err_fc + ofdm_receiver(y_rep_flat, h_flat,
118                                     info_rep, modType, true, R);
118
119     %% ===== PART 6: RECEIVER (SELECTIVE) =====
120     err_su = err_su + ofdm_receiver(y_unc_sel, H,
121                                     info_unc, modType, false, R);
121     err_sc = err_sc + ofdm_receiver(y_rep_sel, H,
122                                     info_rep, modType, true, R);
122
123     bits_unc_cnt = bits_unc_cnt + length(info_unc);
124     bits_rep_cnt = bits_rep_cnt + length(info_rep);
125
126     end
127
128     BER_flat_unc(snr) = err_fu / bits_unc_cnt;
129     BER_flat_rep(snr) = err_fc / bits_rep_cnt;
130     BER_freq_unc(snr) = err_su / bits_unc_cnt;
131     BER_freq_rep(snr) = err_sc / bits_rep_cnt;
132     end
133
134     %% ===== PLOT =====
135     fig = plot_ofdm(EbNO_dB, ...
136                    BER_flat_unc, BER_flat_rep, ...
137                    BER_freq_unc, BER_freq_rep, modType);
138
139     save_figure_png(fig,['Q3-OFDM_' modType], 'figures');
140 end
141
142 %% ===== Functions =====
143 %% Coding
144 function [info_unc, info_rep, coded_rep, pad_unc, pad_rep] =
145         ofdm_coding(modType, Nfft, R)
146
147     bps = strcmp(modType, 'BPSK') + ...

```

```

147         2*strcmp(modType,'QPSK') + ...
148         4*strcmp(modType,'16QAM');
149
150     Ninfo_unc = floor(Nfft/bps);
151     Ninfo_rep = floor(Nfft/(R*bps));
152
153     info_unc = randi([0 1],Ninfo_unc*bps,1);
154     info_rep = randi([0 1],Ninfo_rep*bps,1);
155
156     coded_rep = repelem(info_rep,R);
157
158     % ---- padding length (DO NOT append yet) ----
159     pad_unc = Nfft*bps - length(info_unc);
160     pad_rep = Nfft*bps - length(coded_rep);
161 end
162
163 %% InterLeaver
164 function out = ofdm_interleave(bits, modType)
165
166     switch modType
167     case 'QPSK'
168         assert(length(bits)==512, 'QPSK interleaver requires
169             512 bits');
170         out = reshape(bits,32,16).';
171         out = out(:);
172
173     case '16QAM'
174         assert(length(bits)==1024, '16QAM interleaver requires
175             1024 bits');
176         out = reshape(bits,32,32).';
177         out = out(:);
178
179     otherwise % BPSK
180         out = bits;
181     end
182 end
183
184 %% Mapper
185 function X = ofdm_mapper(bits, modType, Eb)
186
187     switch modType

```

```

186
187     case 'BPSK'
188         % Average symbol energy = Eb
189         X = sqrt(Eb) * (2*bits - 1);
190
191     case 'QPSK'
192         % Average symbol energy = 2Eb        scaled to Eb
193         b = reshape(bits,2,[]).';
194         X = sqrt(Eb) * ( ...
195             (2*b(:,1)-1) + 1j*(2*b(:,2)-1) );
196
197     case '16QAM'
198         % Average symbol energy = 10        normalized by 2.5
199         b = reshape(bits,4,[]).';
200         I = (2*b(:,1)-1).*(2-(2*b(:,3)));
201         Q = (2*b(:,2)-1).*(2-(2*b(:,4)));
202         X = sqrt(Eb/2.5) * (I + 1j*Q);
203
204     end
205 end
206
207 %% Recevier
208 function err = ofdm_receiver(y, h, info_bits, modType, isRep, R)
209
210     %% FFT
211     Y = fft(y);
212
213     %% Equalization
214     if numel(h) > 1        % frequency selective
215         Yeq = Y ./ (h + 1e-12);
216     else                  % flat fading
217         Yeq = Y / h;
218     end
219
220     %% Demapper
221     switch modType
222     case 'BPSK'
223         rx_bits = real(Yeq) > 0;
224
225     case 'QPSK'
226         rx_bits = reshape([real(Yeq)>0 imag(Yeq)>0].',1,[]);

```

```

227
228     case '16QAM'
229         rx_bits = reshape([ ...
230             real(Yeq)>0 imag(Yeq)>0 ...
231             abs(real(Yeq))<2 abs(imag(Yeq))<2].',1,[]);
232     end
233
234 %% De-interleaver
235 switch modType
236     case 'QPSK'
237         rx_bits = reshape(rx_bits,16,32).';
238         rx_bits = rx_bits(:).';
239
240     case '16QAM'
241         rx_bits = reshape(rx_bits,32,32).';
242         rx_bits = rx_bits(:).';
243     end
244
245 %% Decode (Repetition)
246 if isRep
247     dec = zeros(1,length(info_bits));
248     for k = 1:length(info_bits)
249         dec(k) = sum(rx_bits((k-1)*R+1:k*R)) > R/2;
250     end
251     rx_bits = dec;
252 else
253     rx_bits = rx_bits(1:length(info_bits));
254 end
255
256 %% Error count
257 rx_bits = rx_bits(:);
258 info_bits = info_bits(:);
259
260 err = sum(rx_bits ~= info_bits);
261
262 end
263
264 %% Noise
265 function n = awgn_noise(x, Eb, EbN0)
266     N0 = Eb / EbN0;
267     n = sqrt(N0/2) * (randn(size(x)) + 1j*randn(size(x)));

```

```

268 end
269
270
271 %% Plot
272 function fig = plot_ofdm(EbNO_dB,fU,fC,sU,sC,modType)
273     fig = figure;
274     semilogy(EbNO_dB,fU,'r-o','LineWidth',1.6); hold on;
275     semilogy(EbNO_dB,fC,'b-s','LineWidth',1.6);
276     semilogy(EbNO_dB,sU,'k-^','LineWidth',1.6);
277     semilogy(EbNO_dB,sC,'m-d','LineWidth',1.6);
278     grid on; grid minor;
279     legend('Flat-NoCode','Flat-Rep','Freq-NoCode','Freq-Rep',''
        'Location','southwest');
280     xlabel('E_b/N_0 (dB)');
281     ylabel('BER');
282     title(['OFDM ' modType ' over Flat & Frequency Selective
        'Fading']);
283 end
284
285 %% Save Figure
286 function save_figure_png(figHandle, figName, savePath)
287 % SAVE_FIGURE_PNG
288 % Saves a MATLAB figure as PNG with proper formatting
289 %
290 % Inputs:
291 %   figHandle : handle to figure
292 %   figName    : string (figure title & filename)
293 %   savePath   : string (directory path)
294
295 % --- Input checks ---
296 if ~isvalid(figHandle)
297     error('Invalid figure handle.');
```

```

307
308 % --- Build full file path ---
309 fileName = fullfile(savePath, [figName '.png']);
310
311 % --- Save figure ---
312 exportgraphics(figHandle, fileName, 'Resolution', 300);
313
314 fprintf('Figure saved successfully:\n%s\n', fileName);
315 end

```

A.4 Graphical User Interface (GUI)

Listing A.4: MATLAB GUI implementation for Project 3 (OFDM simulations)

```

1 function OFDM_GUI_T35
2 % OFDM PROJECT GUI (Q1      Q3)
3 % Single-file implementation
4
5 % ===== HOME PAGE =====
6 homeFig = uifigure( ...
7     'Name','OFDM Project', ...
8     'Position',[500 200 420 420]);
9
10 gl = uigridlayout(homeFig,[4 1]);
11 gl.RowHeight = {'fit','1x','1x','1x'};
12 gl.Padding = [30 30 30 20];
13
14 % Title
15 uilabel(gl, ...
16     'Text','OFDM Project      Team 35', ...
17     'FontSize',18, ...
18     'FontWeight','bold', ...
19     'HorizontalAlignment','center');
20
21 % Question buttons
22 for q = 1:3
23     uibutton(gl, ...
24         'Text', sprintf('Question %d', q), ...
25         'FontSize',14, ...
26         'ButtonPushedFcn', @(~,~) open_question(q, homeFig));
27 end

```



```

28 end
29
30 % =====
31 %               QUESTION WINDOW
32 % =====
33 function open_question(qnum, homeFig)
34
35     qFig = uifigure( ...
36         'Name', sprintf('Question %d', qnum), ...
37         'Position',[400 150 900 550]);
38
39     gl = uigridlayout(qFig,[3 1]);
40     gl.RowHeight = {'fit','fit','1x'};
41     gl.Padding = [15 15 15 15];
42
43     % Header
44     uilabel(gl, ...
45         'Text', sprintf('Question %d', qnum), ...
46         'FontSize',16, ...
47         'FontWeight','bold', ...
48         'HorizontalAlignment','center');
49
50     % Return button
51     uibutton(gl, ...
52         'Text','    Return to Home', ...
53         'FontSize',13, ...
54         'ButtonPushedFcn', @(~,~) return_home(qFig, homeFig));
55
56     % Output console
57     logBox = uitextarea(gl, ...
58         'Editable','off', ...
59         'FontSize',11);
60
61     drawnow;
62
63     % ===== RUN QUESTION =====
64     try
65         switch qnum
66             case 1
67                 outputText = evalc('Q1');
68             case 2

```

```

69         outputText = evalc('Q2');
70     case 3
71         outputText = evalc('Q3');
72     end
73
74     logBox.Value = splitlines(outputText);
75
76     catch ME
77         logBox.Value = { ...
78             ' Error occurred:', ...
79             ME.message ...
80         };
81         uialert(qFig, ME.message, 'Execution Error');
82     end
83 end
84
85
86 % =====
87 %             RETURN BUTTON
88 % =====
89 function return_home(qFig, homeFig)
90     if isvalid(qFig)
91         close(qFig);
92     end
93     homeFig.Visible = 'on';
94 end
95
96 %% Q1
97 % =====
98 % ===== QUESTION 1 =====
99 % =====
100 function Q1
101     L = 4096;
102     x = randn(1,L);
103
104     tic
105     X_dft = myDFT(x);
106     t_dft = toc;
107
108     tic
109     X_fft = fft(x);

```

```

110     t_fft = toc;
111
112     fprintf('Elapsed time is %s seconds.', num2str(t_dft));
113     fprintf('Elapsed time is %s seconds.', num2str(t_fft));
114
115     fig1 = plot_dft_fft_time(t_dft, t_fft);
116     save_figure_png(fig1, 'Q1_DFT_vs_FFT_Execution_Time', '
        figures');
117 end
118 %===== Q1 Helper functions
        =====
119
120 %% My DFT Function
121 function X = myDFT(x)
122 N = length(x);
123 X = zeros(1,N);
124
125 for k = 0:N-1
126     sumVal = 0;
127     for n = 0:N-1
128         sumVal = sumVal + x(n+1)*exp(-1j*2*pi*n*k/N);
129     end
130     X(k+1) = sumVal;
131 end
132 end
133
134 %% Bar Chart
135 function fig = plot_dft_fft_time(t_dft, t_fft)
136 % PLOT_DFT_FFT_TIME
137 % Plots execution time comparison between DFT and FFT
138
139     times = [t_dft t_fft];
140
141     fig = figure;
142     b = bar(times);
143     grid on;
144
145     set(gca, 'XTickLabel', {'DFT', 'FFT'});
146     ylabel('Execution Time (seconds)');
147     title('Execution Time Comparison: DFT vs FFT');
148

```

```

149 % Display values above bars
150 for i = 1:length(times)
151     text(i, times(i), sprintf('%.6f', times(i)), ...
152         'HorizontalAlignment', 'center', ...
153         'VerticalAlignment', 'bottom');
154 end
155
156 end
157
158
159 %% Save Figure
160 function save_figure_png(figHandle, figName, savePath)
161 % SAVE_FIGURE_PNG
162 % Saves a MATLAB figure as PNG with proper formatting
163 %
164 % Inputs:
165 %   figHandle : handle to figure
166 %   figName   : string (figure title & filename)
167 %   savePath  : string (directory path)
168
169 % --- Input checks ---
170 if ~isValid(figHandle)
171     error('Invalid figure handle.');
```

```

190
191
192 %% Q2
193 % =====
194 % ===== QUESTION 2 =====
195 % =====
196 function Q2
197
198 %% ===== PARAMETERS =====
199 Nbits = 1e5;
200 Eb = 1;
201 R = 5; % Repetition factor
202
203 EbNO_dB = -3:1:10;
204 EbNO_lin = 10.^(EbNO_dB/10);
205
206 % BER storage
207 BER_BPSK_unc = zeros(size(EbNO_dB));
208 BER_BPSK_rep = zeros(size(EbNO_dB));
209 BER_QPSK_unc = zeros(size(EbNO_dB));
210 BER_QPSK_rep = zeros(size(EbNO_dB));
211 BER_16QAM_unc = zeros(size(EbNO_dB));
212 BER_16QAM_rep = zeros(size(EbNO_dB));
213
214 %% ===== MAIN Eb/NO LOOP =====
215 for idx = 1:length(EbNO_dB)
216
217     NO = Eb / EbNO_lin(idx);
218
219     %% =====
220     %% ===== BPSK =====
221     %% =====
222     bk = randi([0 1], Nbits, 1);
223
224     % ----- Uncoded BPSK -----
225     xk = sqrt(Eb) * (2*bk - 1);
226
227     h = (randn(Nbits,1) + 1j*randn(Nbits,1)) / sqrt(2);
228     n = sqrt(NO/2) * (randn(Nbits,1) + 1j*randn(Nbits,1));
229
230     y = h .* xk + n;

```

```

231 y_eq = y ./ h;
232 bk_hat = real(y_eq) > 0;
233
234 BER_BPSK_unc(idx) = mean(bk_hat ~= bk);
235
236 % ----- Repetition BPSK -----
237 bk_rep = repelem(bk, R);
238 xk_rep = sqrt(Eb) * (2*bk_rep - 1);
239
240 h = (randn(length(xk_rep),1) + 1j*randn(length(xk_rep),1)
      ) / sqrt(2);
241 n = sqrt(N0/2) * (randn(length(xk_rep),1) + 1j*randn(
      length(xk_rep),1));
242
243 y = h .* xk_rep + n;
244 y_eq = y ./ h;
245
246 bk_hat_rep = real(y_eq) > 0;
247 bk_hat_mat = reshape(bk_hat_rep, R, []);
248 bk_hat = sum(bk_hat_mat,1) >= ceil(R/2);
249
250 BER_BPSK_rep(idx) = mean(bk_hat.' ~= bk);
251
252 %% =====
253 %% ===== QPSK =====
254 %% =====
255 bits = bk(1:2*floor(length(bk)/2));
256 b = reshape(bits,2,[]).';
257
258 xk = sqrt(Eb/2) * ((2*b(:,1)-1) + 1j*(2*b(:,2)-1));
259
260 h = (randn(length(xk),1) + 1j*randn(length(xk),1)) / sqrt
      (2);
261 n = sqrt(N0/2) * (randn(length(xk),1) + 1j*randn(length(
      xk),1));
262
263 y = h .* xk + n;
264 y_eq = y ./ h;
265
266 b1 = real(y_eq) > 0;
267 b2 = imag(y_eq) > 0;

```

```

268
269 bk_hat = reshape([b1 b2].',[],1);
270 BER_QPSK_unc(idx) = mean(bk_hat ~= bits);
271
272 % ----- Repetition QPSK -----
273 bits_rep = repelem(bits, R);
274 b = reshape(bits_rep,2,[]).';
275
276 xk = sqrt(Eb/2) * ((2*b(:,1)-1) + 1j*(2*b(:,2)-1));
277
278 h = (randn(length(xk),1) + 1j*randn(length(xk),1)) / sqrt
    (2);
279 n = sqrt(N0/2) * (randn(length(xk),1) + 1j*randn(length(
    xk),1));
280
281 y = h .* xk + n;
282 y_eq = y ./ h;
283
284 b1 = real(y_eq) > 0;
285 b2 = imag(y_eq) > 0;
286
287 bk_hat_rep = reshape([b1 b2].',[],1);
288 bk_hat_mat = reshape(bk_hat_rep, R, []);
289 bk_hat = sum(bk_hat_mat,1) >= ceil(R/2);
290
291 BER_QPSK_rep(idx) = mean(bk_hat.' ~= bits);
292
293 %% =====
294 %% ===== 16-QAM =====
295 %% =====
296 bits = bk(1:4*floor(length(bk)/4));
297 b = reshape(bits,4,[]).';
298
299 I = (2*b(:,1)-1) .* (2 - (2*b(:,3)));
300 Q = (2*b(:,2)-1) .* (2 - (2*b(:,4)));
301 xk = (I + 1j*Q) / sqrt(10);
302
303 h = (randn(length(xk),1) + 1j*randn(length(xk),1)) / sqrt
    (2);
304 n = sqrt(N0/2) * (randn(length(xk),1) + 1j*randn(length(
    xk),1));

```

```

305
306     y = h .* xk + n;
307     y_eq = y ./ h;
308
309     I_hat = real(y_eq);
310     Q_hat = imag(y_eq);
311
312     b1 = I_hat > 0;
313     b2 = Q_hat > 0;
314     b3 = abs(I_hat) < 2;
315     b4 = abs(Q_hat) < 2;
316
317     bk_hat = reshape([b1 b2 b3 b4].', [], 1);
318     BER_16QAM_unc(idx) = mean(bk_hat ~= bits);
319
320     % ----- Repetition 16-QAM -----
321     bits_rep = repelem(bits, R);
322     b = reshape(bits_rep, 4, []).';
323
324     I = (2*b(:,1)-1) .* (2 - (2*b(:,3)));
325     Q = (2*b(:,2)-1) .* (2 - (2*b(:,4)));
326     xk = (I + 1j*Q) / sqrt(10);
327
328     h = (randn(length(xk),1) + 1j*randn(length(xk),1)) / sqrt
          (2);
329     n = sqrt(N0/2) * (randn(length(xk),1) + 1j*randn(length(
          xk),1));
330
331     y = h .* xk + n;
332     y_eq = y ./ h;
333
334     I_hat = real(y_eq);
335     Q_hat = imag(y_eq);
336
337     b1 = I_hat > 0;
338     b2 = Q_hat > 0;
339     b3 = abs(I_hat) < 2;
340     b4 = abs(Q_hat) < 2;
341
342     bk_hat_rep = reshape([b1 b2 b3 b4].', [], 1);
343     bk_hat_mat = reshape(bk_hat_rep, R, []);

```



```

344     bk_hat = sum(bk_hat_mat,1) >= ceil(R/2);
345
346     BER_16QAM_rep(idx) = mean(bk_hat.' ~= bits);
347 end
348
349 %% ===== FIGURES =====
350
351 % ---- BPSK ----
352 fig = plot_rayleigh(EbN0_dB, BER_BPSK_unc, BER_BPSK_rep,
353     Nbits, 'BPSK');
354 save_figure_png(fig, ...
355     'Q2_BPSK_Rayleigh_Uncoded_vs_Repetition', ...
356     'figures');
357
358 % ---- QPSK ----
359 fig = plot_rayleigh(EbN0_dB, BER_QPSK_unc, BER_QPSK_rep,
360     Nbits, 'QPSK');
361 save_figure_png(fig, ...
362     'Q2_QPSK_Rayleigh_Uncoded_vs_Repetition', ...
363     'figures');
364
365 % ---- 16-QAM ----
366 fig = plot_rayleigh(EbN0_dB, BER_16QAM_unc, BER_16QAM_rep,
367     Nbits, '16-QAM');
368 save_figure_png(fig, ...
369     'Q2_16QAM_Rayleigh_Uncoded_vs_Repetition', ...
370     'figures');
371
372 % ---- Combined Uncoded Comparison ----
373 fig = figure;
374
375 semilogy(EbN0_dB, BER_BPSK_unc, 'r-o', 'LineWidth', 1.8);
376 hold on;
377 semilogy(EbN0_dB, BER_QPSK_unc, 'b-s', 'LineWidth', 1.8);
378 semilogy(EbN0_dB, BER_16QAM_unc, 'k-^', 'LineWidth', 1.8);
379
380 grid on; grid minor;
381
382 legend('BPSK', 'QPSK', '16-QAM', 'Location', 'southwest');
383 xlabel('E_b / N_0 (dB)', 'FontWeight', 'bold');
384 ylabel('BER', 'FontWeight', 'bold');

```

```

381     title('Uncoded Modulation Comparison over Rayleigh Flat
382           Fading Channel', ...
383           'FontWeight','bold');
384
385     set(gca,'YScale','log','FontSize',11);
386
387     save_figure_png(fig, ...
388         'Q2_Uncoded_BPSK_QPSK_16QAM_Comparison', ...
389         'figures');
390
391     end
392
393 %===== Q2 Helper functions
394 %=====
395 %% Plot BER
396 function fig = plot_rayleigh(EbN0_dB, BER_uncoded, BER_rep, Nbits
397     , modType)
398 % PLOT_Q2_BPSK_RAYLEIGH
399 % Plots uncoded and repetition-coded transmission over Rayleigh
400 flat fading
401
402     fig = figure;
403
404     semilogy(EbN0_dB, BER_uncoded, ...
405         'ro--', 'LineWidth', 1.8, 'MarkerSize', 7);
406     hold on;
407
408     semilogy(EbN0_dB, BER_rep, ...
409         'bs--', 'LineWidth', 1.8, 'MarkerSize', 7);
410
411     grid on;
412     grid minor;
413
414     xlabel('E_b / N_0 (dB)', ...
415         'FontSize', 12, 'FontWeight', 'bold');
416
417     ylabel('Bit Error Rate (BER)', ...
418         'FontSize', 12, 'FontWeight', 'bold');
419
420     title([modType ' over Rayleigh Flat Fading Channel'], ...
421         'FontSize', 14, 'FontWeight', 'bold');

```

```

418     legend( ...
419         ['Uncoded ' modType], ...
420         [modType ' with Repetition (Rate 1/5)'], ...
421         'Location','southwest');
422
423     set(gca, 'FontSize', 11);
424     set(gca, 'YScale', 'log');
425
426     % Info box
427     annotation(fig,'textbox', ...
428         [0.15 0.18 0.38 0.18], ...
429         'String',{ ...
430             ['Modulation: ' modType], ...
431             'Channel: Rayleigh Flat Fading', ...
432             'Coding: Repetition (Rate 1/5)', ...
433             sprintf('# bits: %d', Nbits)}, ...
434         'FitBoxToText','on', ...
435         'BackgroundColor','white', ...
436         'EdgeColor','black', ...
437         'FontSize',10);
438 end
439
440
441 %% Q3
442 % =====
443 % ===== QUESTION 3 =====
444 % =====
445 function Q3
446
447     %% ===== PARAMETERS =====
448     Nfft = 256;
449     Eb = 1;
450     EbNO_dB = 0:2:20;
451     EbNO_lin = 10.^(EbNO_dB/10);
452     No = Eb ./ EbNO_lin; % noise spectral density
453
454     mods = {'BPSK','QPSK','16QAM'};
455     R = 5; % repetition factor
456     Nsym = 150; % OFDM symbols per SNR (runtime
457         safe)

```

```

458 % Two Channels
459 h_flat = (randn + 1j*randn)/sqrt(2);
460 H = (randn(Nfft,1) + 1j*randn(Nfft,1)) / sqrt(2);
461
462 %% ===== LOOP OVER MODULATIONS =====
463 for m = 1:length(mods)
464
465     modType = mods{m};
466
467     %% ===== OFDM FRAME DEFINITION =====
468     switch modType
469         case 'BPSK'
470             rows = 32; cols = 8;    bits_ps = 1;
471         case 'QPSK'
472             rows = 32; cols = 16;    bits_ps = 2;
473         case '16QAM'
474             rows = 32; cols = 32;    bits_ps = 4;
475     end
476
477     bits_per_symbol = rows * cols;    % mapper input size
478
479     % ---- Uncoded system ----
480     bits_uncoded = bits_per_symbol;    % full OFDM payload
481
482     % ---- Repetition-coded system ----
483     bits_info_rep = floor(bits_per_symbol / R);    %
484         information bits
485     bits_coded = bits_info_rep * R;    % after
486         repetition
487
488
489     BER_flat_unc = zeros(size(EbN0_dB));
490     BER_flat_rep = zeros(size(EbN0_dB));
491     BER_freq_unc = zeros(size(EbN0_dB));
492     BER_freq_rep = zeros(size(EbN0_dB));
493
494     for snr = 1:length(EbN0_dB)
495
496         err_fu = 0; err_fc = 0;

```

```

497     err_su = 0; err_sc = 0;
498     bits_unc_cnt = 0; bits_rep_cnt = 0;
499
500     for sym = 1:Nsym
501
502         %% ===== PART 1: CODING =====
503
504         % ---- Uncoded ----
505         info_unc = randi([0 1], bits_uncoded, 1);
506
507         % ---- Repetition coded ----
508         info_rep = randi([0 1], bits_info_rep, 1);
509         coded_rep = repelem(info_rep, R);
510
511         % ---- Padding to OFDM size ----
512         info_unc_p = info_unc; % already
513             bits_per_symbol
514         coded_rep_p = [coded_rep; zeros(bits_per_symbol -
515             length(coded_rep),1)];
516
517         %% ===== PART 2: INTERLEAVER =====
518         info_unc_i = ofdm_interleave(info_unc_p,
519             modType);
520         coded_rep_i = ofdm_interleave(coded_rep_p,
521             modType);
522
523         %% ===== PART 3: MAPPER =====
524         X_unc = ofdm_mapper(info_unc_i, modType, Eb);
525         X_rep = ofdm_mapper(coded_rep_i, modType, Eb);
526
527         %% ===== PART 4a: IFFT =====
528         x_unc = ifft(X_unc);
529         x_rep = ifft(X_rep);
530
531         %% ===== PART 5a: FLAT FADING =====
532
533         y_unc_flat = h_flat * x_unc;
534         y_rep_flat = h_flat * x_rep;

```

```

534
535 %% ===== AWGN =====
536 sigma = sqrt(Eb ./ (2*EbN0_lin(snr)));
537
538 noise_unc = sigma * (randn(size(y_unc_flat)) + 1j
    *randn(size(y_unc_flat)));
539 noise_rep = sigma * (randn(size(y_rep_flat)) + 1j
    *randn(size(y_rep_flat)));
540
541 y_unc_flat = y_unc_flat + noise_unc;
542 y_rep_flat = y_rep_flat + noise_rep;
543
544 %% ===== PART 5b: FREQUENCY SELECTIVE
    =====
545
546 y_unc_sel = ifft(X_unc .* H);
547 y_rep_sel = ifft(X_rep .* H);
548
549 %% ===== AWGN =====
550 sigma = sqrt(Eb ./ (2*EbN0_lin(snr)));
551
552 noise_unc = sigma * (randn(size(y_unc_sel)) + 1j*
    randn(size(y_unc_sel)));
553 noise_rep = sigma * (randn(size(y_rep_sel)) + 1j*
    randn(size(y_rep_sel)));
554
555 y_unc_sel = y_unc_sel + noise_unc;
556 y_rep_sel = y_rep_sel + noise_rep;
557
558
559 %% ===== PART 6: RECEIVER (FLAT) =====
560 err_fu = err_fu + ofdm_receiver(y_unc_flat,
    h_flat, info_unc, modType, false, R);
561 err_fc = err_fc + ofdm_receiver(y_rep_flat,
    h_flat, info_rep, modType, true, R);
562
563 %% ===== PART 6: RECEIVER (SELECTIVE)
    =====
564 err_su = err_su + ofdm_receiver(y_unc_sel, H,
    info_unc, modType, false, R);

```

```

565         err_sc = err_sc + ofdm_receiver(y_rep_sel, H,
566                                         info_rep, modType, true, R);
567
568         bits_unc_cnt = bits_unc_cnt + length(info_unc);
569         bits_rep_cnt = bits_rep_cnt + length(info_rep);
570
571     end
572
573     BER_flat_unc(snr) = err_fu / bits_unc_cnt;
574     BER_flat_rep(snr) = err_fc / bits_rep_cnt;
575     BER_freq_unc(snr) = err_su / bits_unc_cnt;
576     BER_freq_rep(snr) = err_sc / bits_rep_cnt;
577
578     end
579
580     %% ===== PLOT =====
581     fig = plot_ofdm(EbN0_dB, ...
582                    BER_flat_unc, BER_flat_rep, ...
583                    BER_freq_unc, BER_freq_rep, modType);
584
585     save_figure_png(fig,['Q3_OFDM_' modType], 'figures');
586
587     end
588
589 end
590
591 %%===== Q3 Helper functions
592
593
594 %% Coding
595 function [info_unc, info_rep, coded_rep, pad_unc, pad_rep] =
596         ofdm_coding(modType, Nfft, R)
597
598     bps = strcmp(modType, 'BPSK') + ...
599          2*strcmp(modType, 'QPSK') + ...
600          4*strcmp(modType, '16QAM');
601
602     Ninfo_unc = floor(Nfft/bps);
603     Ninfo_rep = floor(Nfft/(R*bps));
604
605     info_unc = randi([0 1], Ninfo_unc*bps, 1);
606     info_rep = randi([0 1], Ninfo_rep*bps, 1);
607
608     coded_rep = repelem(info_rep, R);

```

```

603
604 % ---- padding length (DO NOT append yet) ----
605 pad_unc = Nfft*bps - length(info_unc);
606 pad_rep = Nfft*bps - length(coded_rep);
607 end
608
609 %% InterLeaver
610 function out = ofdm_interleave(bits, modType)
611
612     switch modType
613     case 'QPSK'
614         assert(length(bits)==512, 'QPSK interleaver requires
615             512 bits');
616         out = reshape(bits,32,16).';
617         out = out(:);
618
619     case '16QAM'
620         assert(length(bits)==1024, '16QAM interleaver requires
621             1024 bits');
622         out = reshape(bits,32,32).';
623         out = out(:);
624
625     otherwise % BPSK
626         out = bits;
627     end
628 end
629
630 %% Mapper
631 function X = ofdm_mapper(bits, modType, Eb)
632
633     switch modType
634     case 'BPSK'
635         % Average symbol energy = Eb
636         X = sqrt(Eb) * (2*bits - 1);
637
638     case 'QPSK'
639         % Average symbol energy = 2Eb        scaled to Eb
640         b = reshape(bits,2,[]).';
641         X = sqrt(Eb) * ( ...
642             (2*b(:,1)-1) + 1j*(2*b(:,2)-1) );

```



```

642
643     case '16QAM'
644         % Average symbol energy = 10      normalized by 2.5
645         b = reshape(bits,4,[]).';
646         I = (2*b(:,1)-1).*(2-(2*b(:,3)));
647         Q = (2*b(:,2)-1).*(2-(2*b(:,4)));
648         X = sqrt(Eb/2.5) * (I + 1j*Q);
649
650     end
651 end
652
653 %% Recevier
654 function err = ofdm_receiver(y, h, info_bits, modType, isRep, R)
655
656     %% FFT
657     Y = fft(y);
658
659     %% Equalization
660     if numel(h) > 1      % frequency selective
661         Yeq = Y ./ (h + 1e-12);
662     else                % flat fading
663         Yeq = Y / h;
664     end
665
666     %% Demapper
667     switch modType
668         case 'BPSK'
669             rx_bits = real(Yeq) > 0;
670
671         case 'QPSK'
672             rx_bits = reshape([real(Yeq)>0 imag(Yeq)>0].',1,[]);
673
674         case '16QAM'
675             rx_bits = reshape([ ...
676                 real(Yeq)>0 imag(Yeq)>0 ...
677                 abs(real(Yeq))<2 abs(imag(Yeq))<2].',1,[]);
678     end
679
680     %% De-interleaver
681     switch modType
682         case 'QPSK'

```

```

683         rx_bits = reshape(rx_bits,16,32).';
684         rx_bits = rx_bits(:).';
685
686         case '16QAM'
687             rx_bits = reshape(rx_bits,32,32).';
688             rx_bits = rx_bits(:).';
689         end
690
691         %% Decode (Repetition)
692         if isRep
693             dec = zeros(1,length(info_bits));
694             for k = 1:length(info_bits)
695                 dec(k) = sum(rx_bits((k-1)*R+1:k*R)) > R/2;
696             end
697             rx_bits = dec;
698         else
699             rx_bits = rx_bits(1:length(info_bits));
700         end
701
702         %% Error count
703         rx_bits = rx_bits(:);
704         info_bits = info_bits(:);
705
706         err = sum(rx_bits ~= info_bits);
707
708     end
709
710     %% Noise
711     function n = awgn_noise(x, Eb, EbN0)
712         N0 = Eb / EbN0;
713         n = sqrt(N0/2) * (randn(size(x)) + 1j*randn(size(x)));
714     end
715
716
717     %% Plot
718     function fig = plot_ofdm(EbN0_dB,fU,fC,sU,sC,modType)
719         fig = figure;
720         semilogy(EbN0_dB,fU,'r-o','LineWidth',1.6); hold on;
721         semilogy(EbN0_dB,fC,'b-s','LineWidth',1.6);
722         semilogy(EbN0_dB,sU,'k-^','LineWidth',1.6);
723         semilogy(EbN0_dB,sC,'m-d','LineWidth',1.6);

```

```

724     grid on; grid minor;
725     legend('Flat-NoCode','Flat-Rep','Freq-NoCode','Freq-Rep','
           Location','southwest');
726     xlabel('E_b/N_0 (dB)');
727     ylabel('BER');
728     title(['OFDM ' modType ' over Flat & Frequency Selective
           Fading']);
729 end

```