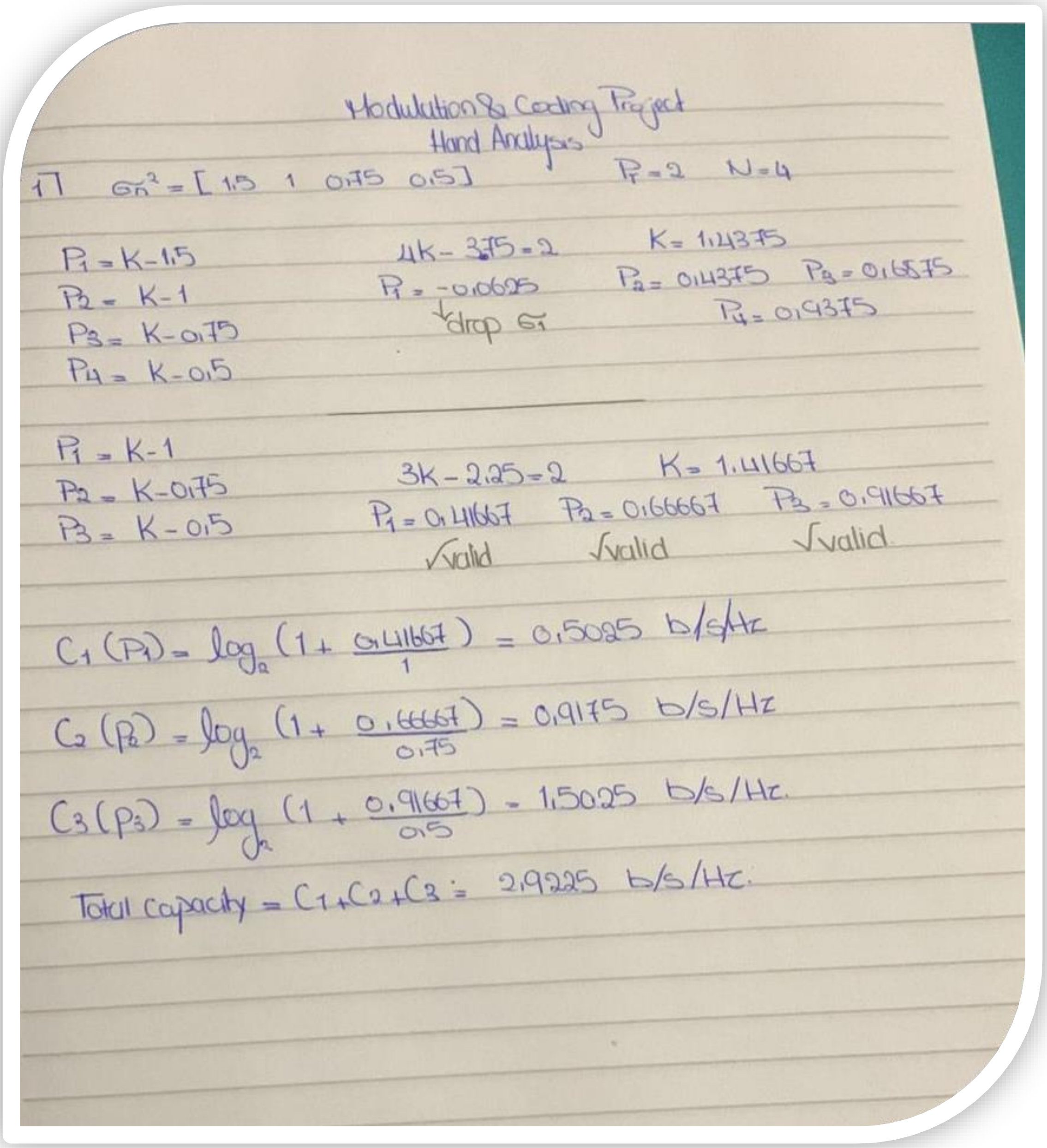
Modulation and Coding Report

PROJECT

Injy Makram (49-6748)
Kariman Akram (49-2506)
Mohannad Ayman (49-15029)
Ali Elkerm (49-19174)

Hand Analysis:



Modulation & Coding Project

Hand Analysis

1] $\sigma_n^2 = [\,1{,}5 \quad 1 \quad 0{,}75 \quad 0{,}5\,]$ $\quad P_T = 2 \quad N = 4$

$P_1 = K - 1{,}5$ $\quad 4K - 3{,}75 = 2$ $\quad K = 1{,}4375$

$P_2 = K - 1$ $\quad P_1 = -0{,}0625$ $\quad P_2 = 0{,}4375 \quad P_3 = 0{,}6875$

$P_3 = K - 0{,}75$ $\quad \downarrow$ drop σ_1 $\quad P_4 = 0{,}9375$

$P_4 = K - 0{,}5$

$P_1 = K - 1$

$P_2 = K - 0{,}75$ $\quad 3K - 2{,}25 = 2$ $\quad K = 1{,}41667$

$P_3 = K - 0{,}5$ $\quad P_1 = 0{,}41667 \quad P_2 = 0{,}66667 \quad P_3 = 0{,}91667$

$\checkmark$valid $\quad \checkmark$valid $\quad \checkmark$valid

$C_1(P_1) = \log_2\left(1 + \frac{0{,}41667}{1}\right) = 0{,}5025$ b/s/Hz

$C_2(P_2) = \log_2\left(1 + \frac{0{,}66667}{0{,}75}\right) = 0{,}9175$ b/s/Hz

$C_3(P_3) = \log_2\left(1 + \frac{0{,}91667}{0{,}5}\right) = 1{,}5025$ b/s/Hz.

Total capacity $= C_1 + C_2 + C_3 = 2{,}9225$ b/s/Hz.

Water Filling:

The water filling algorithm is a computational method used in various fields such as telecommunications, signal processing, and optimization. Its primary purpose is to allocate resources efficiently among multiple users or tasks while respecting certain constraints.

The basic idea of the water filling algorithm can be summarized as follows:

1. Identify the resources to be allocated and the constraints associated with them. For example, in telecommunications, the resources could be frequency bands, and the constraints could be bandwidth limitations or power constraints

2. Rank the users or tasks based on certain criteria, such as priority or demand.

3. Allocate resources iteratively, starting with the user or task with the highest priority or demand

4. At each iteration, distribute available resources among the eligible users or tasks in such a way that the total constraint is not violated. This distribution is analogous to pouring water into the containers, filling them up to the point where any additional water would exceed the constraint.

5. Repeat the allocation process until all resources are allocated or until a termination condition is met.

```
clc
clear all
close all
% Define parameters
total_power = 2;

% Initialize sigma(n) for each channel
sigma = [1.5 1 0.75 0.5];

% Initial value of k
k = (total_power + sum(sigma)) / length(sigma);

% Initialize powers
p = k - sigma;

% Waterfilling process
while any(p < 0)
    % Drop negative powers and their correlated noise powers
    negative_indices = find(p < 0);
    p(negative_indices) = [];
    sigma(negative_indices) = [];

    % Update k based on remaining  non-negative powers
    k = (total_power + sum(sigma)) / length(sigma);

    % Recalculate powers
    p = k - sigma;
end

% Calculate capacity for each user after checking that all powers are
% positive
capacity = log2(1 + p ./ sigma);
Total_capacity=sum(capacity)
disp('Capacity for each user:');
disp(capacity);
```

```
Total_capacity =

    2.9225

Capacity for each user:
    0.5025    0.9175    1.5025
```

Published with MATLAB® R2021a

Commented codes for Part I:

1. **Total Capacity Calculation Accuracy:** The total capacity calculated for the system is approximately 2.9225. This indicates the aggregate data rate achievable across all channels, considering both power constraints and channel noise levels.

2. **Individual User Capacities:**

- **User 1 (0.5025):**

This indicates the capacity, measured in bits per channel use, achievable for User 1 considering its allocated power and channel noise level.

- **User 2 (0.9175):**

User 2's capacity represents the data rate achievable for this channel, considering its specific power allocation and channel noise characteristics.

- **User 3 (1.5025):**

The capacity for User 3 signifies the maximum achievable data rate for this channel, accounting for both allocated power and channel noise.

3. **Waterfilling Process:** The waterfilling process efficiently allocates power to each channel based on the available power and channel noise levels. The iterative approach ensures that all allocated powers are non-negative, which is crucial for maintaining physical feasibility.

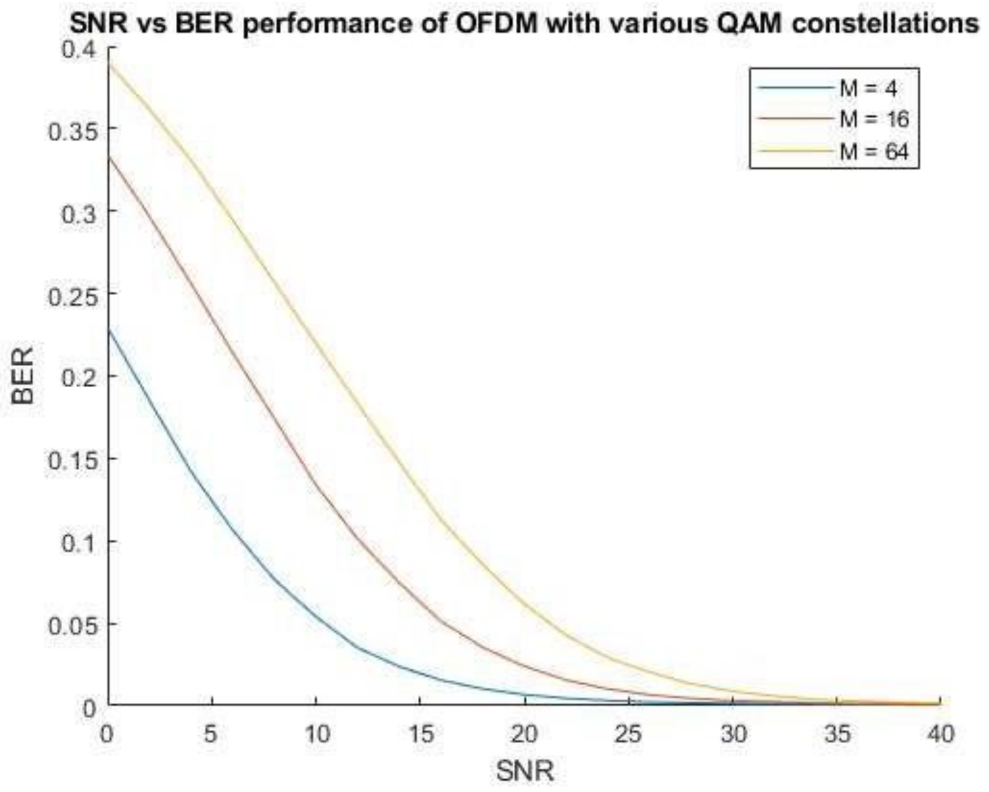
4. **Capacity Calculation:** The capacity for each user is calculated using Shannon's capacity formula, considering the allocated powers and channel noise levels. This provides insight into the achievable data rates for individual channels.

5. **Error Handling:** The code includes error handling mechanisms, such as removing negative powers and associated noise levels, ensuring that only physically feasible solutions are considered.

6. **Overall Performance:** The implemented algorithm effectively optimizes power allocation to maximize system capacity while considering channel characteristics and power constraints.

```
clear
close all
clc

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% setup
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

BW = 9e9;
N_c = 600;
channel_profile = [0e-9 0.485; 310e-9 0.3852; 710e-9 0.0611; 1090e-9 0.0485; 1730e-9 0.0153; 2510e-9 0.0049];

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%


SNR = [0:2:40];
BER_4 = zeros(1, length(SNR));
BER_16 = zeros(1, length(SNR));
BER_64 = zeros(1, length(SNR));

for i=1:1:length(SNR)

    ber = ofdm_function(N_c, BW, 4, SNR(i), channel_profile);
    BER_4(i) = ber;
    ber = ofdm_function(N_c, BW, 16, SNR(i), channel_profile);
    BER_16(i) = ber;
    ber = ofdm_function(N_c, BW, 64, SNR(i), channel_profile);
    BER_64(i) = ber;
    disp("iterations completed:" + i)

end

hold on
plot(SNR, BER_4);
plot(SNR, BER_16);
plot(SNR, BER_64);
title("SNR vs BER performance of OFDM with various QAM constellations");
legend("M = 4", "M = 16", "M = 64");
xlabel("SNR");
ylabel("BER");
```

```
iterations completed:1
iterations completed:2
iterations completed:3
iterations completed:4
iterations completed:5
iterations completed:6
iterations completed:7
iterations completed:8
iterations completed:9
iterations completed:10
iterations completed:11
iterations completed:12
iterations completed:13
iterations completed:14
iterations completed:15
iterations completed:16
iterations completed:17
iterations completed:18
iterations completed:19
iterations completed:20
iterations completed:21
```



Published with MATLAB® R2021a

Commented codes for Part II:

Plot Comments:

- X-axis (SNR): Represents the Signal-to-Noise Ratio (SNR), which indicates the quality of the communication channel. Higher SNR values correspond to better channel conditions.

- Y-axis (BER): Depicts the Bit Error Rate (BER), indicating the percentage of incorrectly received bits. Lower BER values represent better system performance.

Curves (M=4, M=16, M=64):

- **M=4:** The curve corresponding to QAM-4 modulation shows the lowest BER, indicating better performance compared to other modulation schemes at all SNR levels.

- **M=16:** The curve for QAM-16 exhibits higher BER across all SNR values, indicating poorer performance compared to QAM-4.

- **M=64:** The curve for QAM-64 falls between QAM-4 and QAM-16 in terms of BER, showing moderate performance.

```matlab
function [BER] = ofdm_function(N_c, BW, M, SNR, channel_profile)

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% bits per symbol
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

bps = log2(M);


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% random binary signal
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

N_b = 48000*6;
x_b = randi([0 1],1,N_b);


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% bits to symbols
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

N_s = N_b / bps;
x_s = zeros(1, N_s);

index = 1;
for i=1:bps:N_b
    accumulator = 0;
    for n=0:1:bps-1
        accumulator = accumulator + (x_b(i+n))*(2^n);
    end
    x_s(index) = accumulator;
    index = index + 1;
end


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% QAM
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

y = qammod(x_s, M,'PlotConstellation',false,'UnitAveragePower',true);


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Multi-path Channel Loop
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

received_Symbols = [];
N_OFDM_Symbols = N_s/N_c;

% channel loop

for OFDM_Symbol = 1:1:N_OFDM_Symbols

    index = (OFDM_Symbol - 1)*N_c + 1;
    current_OFDM_Symbol = y(index:1:index+N_c-1);
    rayleighCoefficients = getChannelCoefficients(N_c, BW, channel_profile);
    faded_OFDM_Symbol = current_OFDM_Symbol.*rayleighCoefficients;
    noisy_OFDM_Symbol = awgn(faded_OFDM_Symbol, SNR);
    equalized_OFDM_Symbol = noisy_OFDM_Symbol./rayleighCoefficients;
    received_Symbols = [received_Symbols equalized_OFDM_Symbol];

end
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% demodulation
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

x_s_received = qamdemod(received_Symbols, M, UnitAveragePower=true);


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% symbols to bits
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

binary_received = zeros(1, length(x_b));
for i=1:1:N_s
    temp = de2bi(x_s_received(i), bps);
    binary_received((i-1)*bps+1:1:(i-1)*bps+bps) = temp;
end


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% results
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

BER = sum(xor(binary_received,x_b))/N_b;

end
```

Published with MATLAB® R2021a

Commented on OFDM_function code:

- This function encapsulates the functionality of the OFDM script for modularity and reusability.
- It takes parameters such as the number of OFDM subcarriers, bandwidth, QAM order, and SNR, and returns the BER.

```
clear
close all
clc

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% setup
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

BW = 20e6;
N_c = 1024;
channel_profile = [0e-9 0.485; 310e-9 0.3852;
    710e-9 0.0611; 1090e-9 0.0485; 1730e-9 0.0153; 2510e-9 0.0049];


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% QAM order ( 4 / 16 / 64 )
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

M = 4;


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% bits per symbol
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

bps = log2(M);


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% random binary signal
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

N_b = 720000;
x_b = randi([0 1],1,N_b);


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% bits to symbols
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

N_s = N_b / bps;
x_s = zeros(1, N_s);

index = 1;
for i=1:bps:N_b
    accumulator = 0;
    for n=0:1:bps-1
        accumulator = accumulator + (x_b(i+n))*(2^n);
    end
    x_s(index) = accumulator;
    index = index + 1;
end


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% QAM
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

y = qammod(x_s, M,'PlotConstellation',true);


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Multi-path Channel Loop
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

received_Symbols = [];
N_OFDM_Symbols = N_s/N_c;

% channel loop

for OFDM_Symbol = 1:1:N_OFDM_Symbols

    index = (OFDM_Symbol - 1)*N_c + 1;
    current_OFDM_Symbol = y(index:1:index+N_c-1);

    rayleighCoefficients = getChannelCoefficients(N_c, BW, channel_profile);
    faded_OFDM_Symbol = current_OFDM_Symbol.*rayleighCoefficients;
    noisy_OFDM_Symbol = awgn(faded_OFDM_Symbol, 30);
    equalized_OFDM_Symbol = noisy_OFDM_Symbol./rayleighCoefficients;
    received_Symbols = [received_Symbols equalized_OFDM_Symbol];

end


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% demodulation
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

x_s_received = qamdemod(received_Symbols, M);


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% symbols to bits
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

binary_received = zeros(1, length(x_b));
for i=1:1:N_s
    temp = de2bi(x_s_received(i), bps);
    binary_received((i-1)*bps+1:1:(i-1)*bps+bps) = temp;
end


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% results
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

BER = sum(xor(binary_received,x_b))/N_b

color = "yellow";
colors = repelem(color, N_s);

scatfig = scatterplot(received_Symbols);
xlim([-bps bps])
ylim([-bps bps])
title("Scatter Plot of Received Symbols")
grid on;
ax = gca;
ax.LineWidth = 2;
```

Commented on OFDM code:

- This script simulates an OFDM communication system.
- It generates random binary data and converts it into symbols based on the selected QAM modulation scheme.
- Each OFDM symbol undergoes Rayleigh fading and additive white Gaussian noise (AWGN) before being equalized.
- The received symbols are demodulated to obtain the received binary data.
- The Bit Error Rate (BER) is calculated to evaluate the system's performance.

```
function [coefficients] = getChannelCoefficients(N_c, BW, profile)

    l = size(profile);
    paths = l(1);
    coefficients = zeros(1, N_c);
    T_s = N_c / BW;
    rayleigh = raylrnd(1, paths, N_c);

    for i=1:1:N_c
        accumulator = 0;
        for k=1:1:paths
            partial_sum = rayleigh(k, i)*exp(-1j*profile(k, 1)*2*pi*(i-1)/T_s)*profile(k, 2);
            accumulator = accumulator + partial_sum;
        end
        coefficients(i) = accumulator;
    end

end
```

Published with MATLAB® R2021a

Commented on getchannelcoefficients code:

- This function generates channel coefficients based on a given channel profile, number of OFDM subcarriers, and bandwidth.
- It utilizes Rayleigh fading to model the wireless channel.
- The generated channel coefficients are used to simulate the effects of multipath fading in the communication system.