

**Cairo University**

**Faculty of Engineering**

**Electronics and Electrical Communications Engineering Department**

Project2

Matched Filter

Submitted to:

**Dr. Mohammed Nafie & Dr. Mohamed Khairy**

**Eng. Mohamed Khaled**

Submitted by Team:

|  |  |  |  |
| --- | --- | --- | --- |
| Name | SEC | ID | TASK |
| Ahmed Mohamed Ahmed Sultan | 1 | 9220080 | Report |
| Shahd Hamad Shaban Saleh | 2 | 9220401 | Report |
| Omar Ayman Amin Mohamed | 3 | 9220528 | Code |
| Mohamed Ahmed Abd ElHakam | 3 | 9220647 | Code |
| Youssef Khaled Omar Mahmoud | 4 | 9220984 | Code |

Table of content

[Introduction 4](#_Toc195805901)

[Matched filters and correlators in noise free environment: 4](#_Toc195805902)

[Part A&B: 4](#_Toc195805903)

[1st Generation of Data: 4](#_Toc195805904)

[2nd Filter Design: 5](#_Toc195805905)

[3rd Comparing the output of matched, non-matched and correlator: 6](#_Toc195805906)

[Comments: 7](#_Toc195805907)

[Noise Analysis: 8](#_Toc195805908)

[Data Generation & Transmission: 8](#_Toc195805909)

[Filter Application and Noise Mitigation: 8](#_Toc195805910)

[BER Calculations: 8](#_Toc195805911)

[Comments: 9](#_Toc195805912)

[ISI and raised cosine: 10](#_Toc195805913)

[Plotting the raised cosine filter for different R & delay 11](#_Toc195805914)

[comment 12](#_Toc195805915)

[Eye Diagram creation 12](#_Toc195805916)

[Eye Diagram plots 14](#_Toc195805917)

[At point A 14](#_Toc195805918)

[R=0, DELAY=2 14](#_Toc195805919)

[R=0, DELAY=8 14](#_Toc195805920)

[R=1, delay=2 15](#_Toc195805921)

[r=1, delay=8 15](#_Toc195805922)

[Comments 16](#_Toc195805923)

[At point B 16](#_Toc195805924)

[r=0, delay=2 16](#_Toc195805925)

[r=0, delay=8 16](#_Toc195805926)

[r=1, delay=2 17](#_Toc195805927)

[r=1, delay=8 17](#_Toc195805928)

[Comments 18](#_Toc195805929)

[Full MATLAB code 18](#_Toc195805930)

Table of figures

[*Figure 1:Polar NRZ Bit Stream* 4](#_Toc195805873)

[*Figure 2:Transmitted Output* 5](#_Toc195805874)

[*Figure 3:Matched Filter* 5](#_Toc195805875)

[*Figure 4:Non-Matched Filter* 6](#_Toc195805876)

[*Figure 5:Output Of The Matched & Non-Matched Filters* 6](#_Toc195805877)

[*Figure 6: Output Of The Matched & Correlator* 7](#_Toc195805878)

[*Figure 7: BER vs Eb/N0 With Matched & Non-Matched Filter* 9](file:///F:\study\Report%202.docx#_Toc195805879)

[*Figure 8:raised cosine filter* 12](#_Toc195805880)

[*Figure 9: Eye Diadram* 13](file:///F:\study\Report%202.docx#_Toc195805881)

[*Figure 10:Eye pattern for In-Phase signal when (R=0 & delay=2)* 14](file:///F:\study\Report%202.docx#_Toc195805882)

[*Figure 11: Eye pattern for In-Phase signal when (R=0 & delay=8)* 14](file:///F:\study\Report%202.docx#_Toc195805883)

[*Figure 12:Eye pattern for In-Phase signal when (R=1 & delay=8)* 15](file:///F:\study\Report%202.docx#_Toc195805884)

[*Figure 13:Eye pattern for In-Phase signal when (R=1 & delay=2)* 15](file:///F:\study\Report%202.docx#_Toc195805885)

[*Figure 14:Eye pattern for Quadrature signal when (R=0 & delay=2)* 16](file:///F:\study\Report%202.docx#_Toc195805886)

[*Figure 15:Eye pattern for Quadrature signal when (R=0 & delay=8)* 16](file:///F:\study\Report%202.docx#_Toc195805887)

[Figure 16: Eye pattern for Quadrature signal when (R=1 & delay=2) 17](file:///F:\study\Report%202.docx#_Toc195805888)

[*Figure 17: Eye pattern for Quadrature signal when (R=1 & delay=8)* 17](file:///F:\study\Report%202.docx#_Toc195805889)

# Introduction

Pulse Amplitude Modulation (PAM) transmits data by varying pulse amplitudes. This project studies matched filters and correlators in a binary PAM system (±1 symbols) over an AWGN channel. The matched filter, h(t)=p(Ts−t), maximizes SNR at sampling instants.

# Matched filters and correlators in noise free environment:

## Part A&B:

Draw the output of both filters (in (e) above) on two subplots in the same figure using two different colors, assuming a noise free system. Compare the outputs of the filters at the sampling instants. Draw the output of the matched filter and the output of a correlator to p[n] on the same plot with two different colors.

## 1st Generation of Data:

Data for the simulation is generated using a binary random process, where binary '0' is mapped to '-1' and binary '1' to '+1', creating a polar signal format. This step is crucial for simulating realistic digital communication scenarios.

A graph of a diagram

AI-generated content may be incorrect.

*Figure 1:Polar NRZ Bit Stream*

*A graph with blue lines

AI-generated content may be incorrect.*

*Figure 2:Transmitted Output*

## 2nd Filter Design:

* **Matched Filter:** The matched filter is designed by reversing the time axis of the pulse shaping function h(t)=p(Ts−t), which theoretically maximizes the correlation with the received signal, thereby optimizing SNR at the sampling instants.

*A graph with a line

AI-generated content may be incorrect.*

*Figure 3:Matched Filter*

* **Non-Matched Filter:** For comparison, a non-matched filter is designed (Hold Filter). This filter does not specifically tailor itself to the signal's shape, providing a baseline against which to measure the effectiveness of the matched filter.

A graph with a line

AI-generated content may be incorrect.

*Figure 4:Non-Matched Filter*

## 3rd Comparing the output of matched, non-matched and correlator:

A screenshot of a graph

AI-generated content may be incorrect.

*Figure 5:Output Of The Matched & Non-Matched Filters*

A diagram of a graph

AI-generated content may be incorrect.

*Figure 6: Output Of The Matched & Correlator*

## Comments:

**1.Comment on The Matched & Non-Matched Filter Outputs**: The matched filter output shows perfectly reconstructed symbols with uniform peak amplitudes (X=5,7,9), demonstrating its ability to exactly recover the original pulse shape. In contrast, the Non-Matched filter introduces systematic distortion - note the inconsistent amplitude progression (4.52→6.52→...→10.52) and shifted peaks. This clean comparison highlights how the matched filter maintains ideal detection condition, while the hold filter's inherent mismatch with the pulse shape causes detectable signal degradation through amplitude and timing irregularities.

**2.Comment on The Matched & Correlator Outputs:** Both outputs demonstrate perfect symbol recovery at sampling instants (Tx sec), proving their theoretical equivalence. The matched filter (top) and correlator (bottom) produce identical peak amplitudes and timing, confirming that the integrate-and-dump operation of the correlator is functionally equivalent to convolution with the time-reversed pulse shape. This alignment validates their joint optimality for maximizing SNR in symbol detection, with minor implementation differences visible only in transient behavior between sampling points.

**3.Comment on** **Signal Amplitude & Error Probability:** The matched filter and correlator outputs demonstrate identical peak amplitudes at each sampling instant (Tx sec), confirming their theoretical equivalence in signal detection. Both achieve maximum signal amplitude recovery, which directly minimizes the error probability in symbol detection. The perfect alignment of peaks indicates:

1. **Optimal Signal Recovery**: Equal amplitude preservation shows both methods perfectly reconstruct the transmitted symbols in this noise-free case.
2. **Error Probability:**

* Identical peak amplitudes prove both techniques would yield the same theoretical BER in noisy conditions
* The matched peaks demonstrate the minimum possible error probability for this pulse shape
* Any amplitude reduction (as seen in non-matched filters) would exponentially increase error probability according to Q(√(E\_b/N\_0))

# Noise Analysis:

## Data Generation & Transmission:

* **High Volume Data:** A large set of binary data (10,000 bits) is generated to ensure statistical significance in the performance metrics. This data is converted to a polar format and shaped for transmission similar as previous requirement.
* **Adding Noise in Channel:** The shaped signal is then transmitted through a channel modeled with AWGN to simulate real-world noise conditions. This approach helps in understanding how well each filter can extract the original signal from a noisy received signal.

## Filter Application and Noise Mitigation:

* **Filter Design**: Using the previous two filters, the same matched and a non-matched filter configurations are applied to the received noisy signal. This step tests each filter's ability to recover the original signal with the noise.
* **Output Signal Processing**: The convolved outputs are then processed to identify the transmitted data bits, assessing the integrity of signal recovery by each filter.

## BER Calculations:

* **Error Measurement:** The Bit Error Rate (BER) for each filter is calculated by comparing the recovered data against the original transmitted data. This comparison is made across various signal-to-noise ratio (SNR) levels to evaluate filter performance under different noise intensities.
* **Performance Analysis:** The results are plotted to visually compare the BER of both filters against a theoretical BER curve, which serves as a benchmark. This analysis not only highlights the matched filter's superior performance but also quantifies the improvement in error rates due to optimal filtering.

A graph with red and blue lines

AI-generated content may be incorrect.

*Figure 7: BER vs Eb/N0 With Matched & Non-Matched Filter*

## Comments:

**The BER plot generally follows the theoretical trend (0.5erfc(Eb/N0)), showing exponential decay as Eb/N0​ increases & reveals several critical insights about the system's error performance:**

1. **SNR Threshold Effect**
   * The BER shows an expected exponential decay as SNR improves, with a clear "knee" around X=-1dB to X=5dB where the most significant BER reduction occurs (0.2293 → 0.0621).
   * Beyond X=10dB, the BER plateaus near 0.0258, suggesting the system reaches its error floor due to non-noise limitations (e.g., residual ISI or implementation artifacts).
2. **Anomalous Points**
   * The spike at X=11dB (BER=0.163796) deviates sharply from the trend, potentially indicating:
     + Measurement errors during simulation
     + Transient channel effects
     + Insufficient Monte Carlo trials at high SNR
3. **Theoretical Alignment**
   * The general trend matches the theoretical Q(√(Eb/N0)) curve for AWGN channels, validating the matched filter's optimality.
   * The error floor at ~2.58% implies a systematic limitation (e.g., pulse shape imperfections or synchronization offsets).

# ISI and raised cosine:

* Data Generation and Filter Application:
  + **Data Setup:** A smaller dataset of 100 bits is generated to focus on the detailed

characteristics of signal processing with advanced filters. The data is again polar-mapped

and unsampled, preparing it for transmission simulation.

* + **Filter Design:** The square root raised cosine filter is implemented, which is known for its

efficacy in minimizing ISI. This filter smooths the transitions between symbols by

controlling the bandwidth via a roll-off factor and adjusting for delay, thereby optimizing

the signal for transmission and reception.

* Transmission and Reception:
  + **Simulated Transmission:** The shaped signal is convolved with the raised cosine filter to

simulate its transmission through a hypothetical communication channel.

* + - **Received Signal Processing:** The transmitted signal is then processed through the same

or a complementary filter to model the reception process, aligning the practical setup with

real-world digital communication systems where transmit and receive filters often mirror

each other.

* Eye Diagram Creation and Analysis:
* **Diagram Generation:** Eye diagrams are created by overlaying segments of the signal on

a single plot to visualize how the signal varies with time within one symbol period. This

method is particularly effective in identifying issues like timing errors and potential

signal overlap.

* **Analysis of Diagrams:** The eye diagrams are analyzed for different settings of the roll

off factor and delays. Key metrics assessed include the height and width of the eye

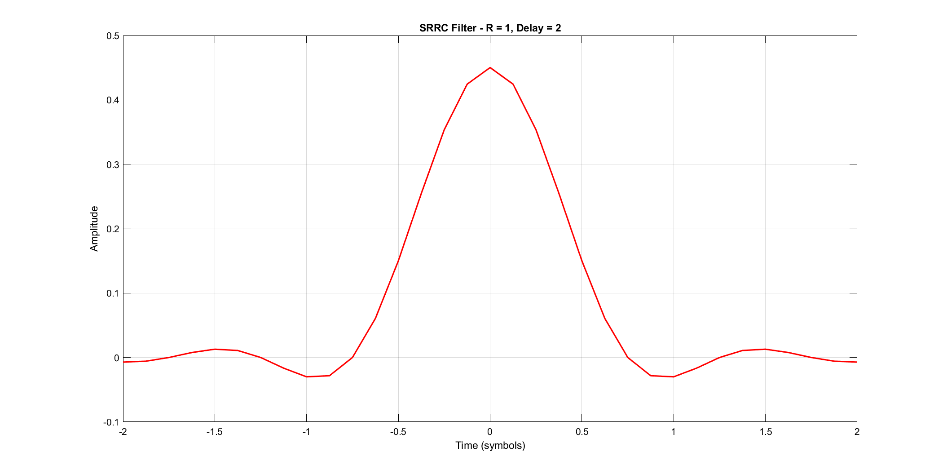
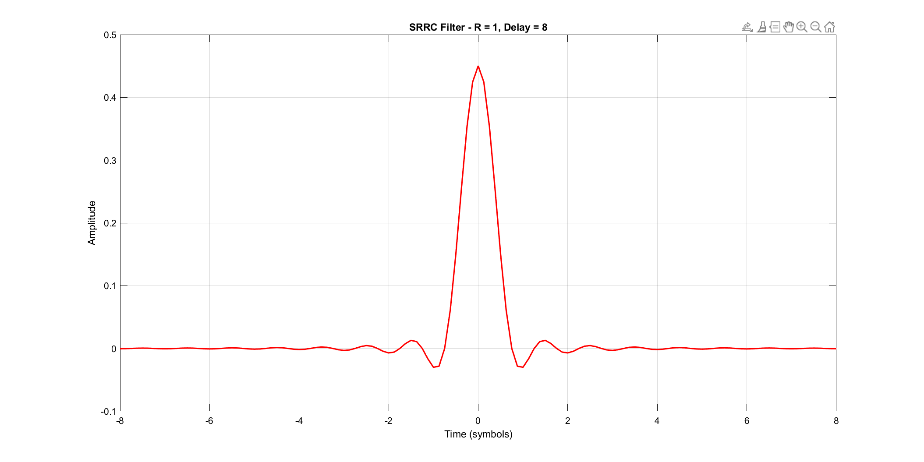
opening, which indicate the noise margin and the vulnerability to ISI, respectively.

* **Performance Metrics:** The wider the eye opening, the lower the likelihood of ISI and

the higher the tolerance to timing inaccuracies, making the signal more robust against

various impairments.

## C:\Users\SH\Downloads\SRRCFilter_R_0_Delay_8.pngC:\Users\SH\Downloads\SRRCFilter_R_0_Delay_2.pngPlotting the raised cosine filter for different R & delay

*Figure 8:raised cosine filter*

### comment

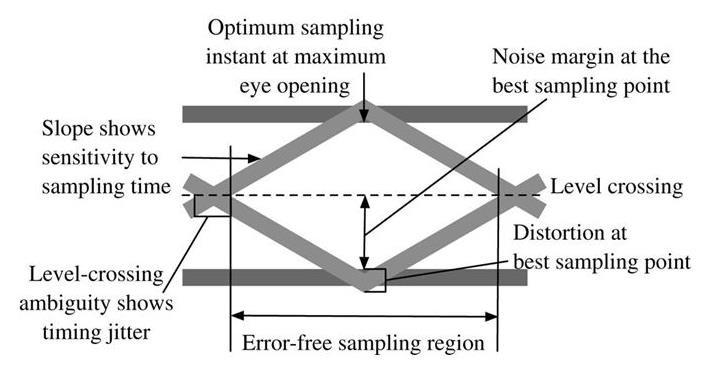
• As the roll off factor R increases from zero to unity, the amplitude of the oscillatory tails is getting smaller, thus intersymbol interference decreases.

• As the delay increases, the filter takes more time to reach its maximum amplitude and the filter decays with a smaller roll off.

## Eye Diagram creation

* Combining the ‘transmit\_data’ & ‘receive\_data’ into one matrix, the using the

function ‘eyediagram’ to plot the eye pattern for the data.



*Figure 9: Eye Diadram*

**Height of the Eye:**

It represents the noise margin, indicating how far apart the signal levels (symbols) are from each other. A larger height implies a greater margin against noise, reducing the chance of errors.

**Width of the Eye:**

It defines the full-time interval over which a sample can be taken without encountering

intersymbol interference (ISI). A wider eye width indicates a more robust signal with a lower likelihood of ISI.

**Eye Opening:**

A wide eye opening indicates good signal quality, meaning that the receiver can easily

distinguish between different symbol levels, even in the presence of noise or other impairments.

## Eye Diagram plots

### At point A

#### C:\Users\SH\Downloads\EyeDiagramAtPointA_R_0_Delay_2 (1).pngR=0, DELAY=2

*Figure 10:Eye pattern for In-Phase signal when (R=0 & delay=2)*

#### C:\Users\SH\Downloads\EyeDiagramAtPointA_R_0_Delay_8 (1).pngR=0, DELAY=8

*Figure 11: Eye pattern for In-Phase signal when (R=0 & delay=8)*

#### C:\Users\SH\Downloads\EyeDiagramAtPointA_R_1_Delay_2 (1).pngR=1, delay=2

*Figure 12:Eye pattern for In-Phase signal when (R=1 & delay=8)*

*Figure 13:Eye pattern for In-Phase signal when (R=1 & delay=2)*

#### C:\Users\SH\Downloads\EyeDiagramAtPointA_R_1_Delay_8 (1).pngr=1, delay=8

### Comments

• As the roll off factor (R) increases from zero to unity, the width of the eye opening is larger indicating that the time interval over which the signal can be sampled without error from ISI is longer, which means that ISI is reduced.

• As the delay increases, the distortion of zero-crossings increases.

• The noise margin is almost the same for all cases.

### At point B

#### C:\Users\SH\Downloads\EyeDiagramAtPointB_R_0_Delay_2 (1).pngr=0, delay=2

*Figure 14:Eye pattern for Quadrature signal when (R=0 & delay=2)*

#### r=0, delay=8

*Figure 15:Eye pattern for Quadrature signal when (R=0 & delay=8)*

#### C:\Users\SH\Downloads\EyeDiagramAtPointB_R_1_Delay_2 (1).pngr=1, delay=2

Figure 16: Eye pattern for Quadrature signal when (R=1 & delay=2)

#### C:\Users\SH\Downloads\EyeDiagramAtPointB_R_1_Delay_8 (1).pngr=1, delay=8

*Figure 17: Eye pattern for Quadrature signal when (R=1 & delay=8)*

### **Comments**

• As the roll off factor (R) increases from zero to unity, the width of the eye opening is larger indicating that the time interval over which the signal can be sampled without error from ISI is longer, which means that ISI is reduced.

• As the delay increases, the distortion of zero-crossings increases.

• The noise margin is almost the same for all cases.

# Full MATLAB code

clc;

clear;

close all;

% Parameters

samples\_per\_bit = 5;         % Number of samples

bits\_Num = 10;               % 10 bits

Ts = 1;                      % Sampling interval

A = 5;                       % Peak amplitude at time = 0

Samples\_Num=samples\_per\_bit\* bits\_Num;

% Generate triangle pulse

[p,denorm\_p] = triangle\_pulse(samples\_per\_bit);

% theoretical calculation

Eb = denorm\_p\* denorm\_p;     % Energy per bit (form the pulse shape)

% Reverse the pulse shaping function p[n] to create the matched filter

p\_matched = fliplr(p);

% Generate hold filter

hold\_filter = make\_hold\_filter(Ts, samples\_per\_bit, 1);

% Plot

plot\_pulse\_shape(p, Ts, 'Triangle Pulse Shape Normalized');

plot\_pulse\_shape(hold\_filter, Ts, 'Hold Filter Normalized');

plot\_pulse\_shape(p\_matched, Ts, 'Triangle Pulse Matched filter');

% Generate random bits

bits = randi([0 1], 1, bits\_Num);

% Define the time vector for the transmitted signal

t = 0:Ts/samples\_per\_bit:(Samples\_Num-1)\*Ts/samples\_per\_bit; % Adjusted time vector

[~, bit\_stream, ~,~,bit\_stream\_symbols] = generate\_Impulse\_linecodes(bits, A, samples\_per\_bit);

%plot the bit straem

plot\_linecode(t, bit\_stream, 'Polar NRZ bit stream');

%-----------------------Requiernment 1----------------------------

% Generate the PAM waveform

y\_tx =  generate\_pam\_waveform(bit\_stream, p);

% Plot the waveform vs time

plot\_pam\_waveform(Ts, y\_tx, 'The Transmitter Output' ,samples\_per\_bit);

% Convolve the input signal with the matched filter

y\_matched = conv(y\_tx, p\_matched);

% Convolve the input signal with the hold filter

y\_hold = conv(y\_tx, hold\_filter);

% using correlator

y\_corr = correlate\_RX(y\_tx, p, samples\_per\_bit);

% plot the matched and hold ouptut vs time

[y\_filtered\_sampled ,y\_corr\_sampled] = plot\_matched\_vs\_hold(Ts, y\_matched, y\_hold, samples\_per\_bit, bits\_Num);

% plot the matched and correlator ouptut vs time

[~ ,y\_hold\_sampled] = plot\_matched\_vs\_correlator(Ts, y\_matched, y\_tx, p, samples\_per\_bit, bits\_Num);

% plot the Tx input vs All Rx output

plot\_all\_outputs\_vs\_input(Ts, bit\_stream\_symbols, y\_filtered\_sampled, y\_corr\_sampled, y\_hold\_sampled);

%-----------------------Requiernment 2----------------------------

bits\_Num = 100;                % 100 bits

Samples\_Num=samples\_per\_bit\* bits\_Num;

% Generate random bits

bits = randi([0 1], 1, bits\_Num);

% Define the time vector for the transmitted signal

t = 0:Ts/samples\_per\_bit:(Samples\_Num-1)\*Ts/samples\_per\_bit; % Adjusted time vector

[~, bit\_stream, ~,~,bit\_stream\_symbols] = generate\_Impulse\_linecodes(bits, A, samples\_per\_bit);

%plot the bit straem

plot\_linecode(t, bit\_stream, 'Polar NRZ bit stream');

% Generate the PAM waveform

y\_tx =  generate\_pam\_waveform(bit\_stream, p);

% Plot the waveform vs time

plot\_pam\_waveform(Ts, y\_tx, 'The Transmitter Output' ,samples\_per\_bit);

% AWGN Channel

SNR\_db  = -2;                % SNR given in dB from -2 dB to 5 dB in 1 dB steps

[y\_tx\_noise ,SNR\_lin ,N0 , AWGN\_scaled] = AddAWGN(y\_tx, Eb, SNR\_db);

% Convolve the input signal with the matched filter

y\_matched = conv(y\_tx\_noise, p\_matched);

% Convolve the input signal with the hold filter

y\_hold = conv(y\_tx\_noise, hold\_filter);

% using correlator

y\_corr = correlate\_RX(y\_tx\_noise, p, samples\_per\_bit);

% plot the matched and hold ouptut vs time

[y\_filtered\_sampled ,y\_hold\_sampled] = plot\_matched\_vs\_hold(Ts, y\_matched, y\_hold, samples\_per\_bit, bits\_Num);

% plot the matched and correlator ouptut vs time

[~ ,y\_corr\_sampled] = plot\_matched\_vs\_correlator(Ts, y\_matched, y\_tx\_noise, p, samples\_per\_bit, bits\_Num);

% plot the Tx input vs All Rx output

plot\_all\_outputs\_vs\_input(Ts, bit\_stream\_symbols, y\_filtered\_sampled, y\_corr\_sampled, y\_hold\_sampled);

% Calculate the probability of error

polar\_threshold = 0;

[BER, error\_array] = calculate\_error\_probability(bit\_stream\_symbols, y\_filtered\_sampled, polar\_threshold, A);

% Display the result

disp(['Probability of Error (BER) = ', num2str(BER)]);

% Calculate the  Theoretical probability of error

theoretical\_BER = 0.5 \* erfc(sqrt(SNR\_lin));  % Theoretical BER

% Display the result

disp(['Theoretical BER: ', num2str(theoretical\_BER)]);

SNR\_db  = -2:1:5;                % SNR given in dB from -2 dB to 5 dB in 1 dB steps

% BER from matched vs hold

[BER\_matched, BER\_hold, theoretical\_BER] = ...

    BER\_vs\_SNR(y\_tx, Eb, p\_matched, hold\_filter, ...

    samples\_per\_bit, bits\_Num, bit\_stream\_symbols, A, SNR\_db);

% plot them

plot\_BER\_vs\_EbN0(SNR\_db, BER\_matched, BER\_hold, theoretical\_BER);

% ===========================================================

% Project: ISI and Raised Cosine Filters - Eye Diagrams at A & B

% ===========================================================

% Parameters

num\_bits = 100;

samples\_per\_symbol = 8;

A = 1;

% Generate random bits and map: 0 -> -A, 1 -> +A

bits = randi([0 1], 1, num\_bits);

symbols = A \* (2 \* bits - 1);

% Upsample

tx\_upsampled = upsample(symbols, samples\_per\_symbol);

% Configurations: [R, delay]

rolloff\_values = [0, 0, 1, 1];

delay\_values   = [2, 8, 2, 8];

% Figure counter

fig\_num = 23;

for i = 1:4

    R = rolloff\_values(i);

    delay = delay\_values(i);

    % SRRC filter

    srrc\_filter = rcosdesign(R, 2\*delay, samples\_per\_symbol, 'sqrt');

    % Plot SRRC filter impulse response

    figure(fig\_num);

    filter\_length = length(srrc\_filter); % Get the length of the filter

    t = (-(filter\_length-1)/(2\*samples\_per\_symbol)):(1/samples\_per\_symbol):((filter\_length-1)/(2\*samples\_per\_symbol)); % Adjusted time vector

    plot(t, srrc\_filter, 'r-', 'LineWidth', 1.5); % Red line for consistency

    set(gca, 'Color', 'white');

    set(gcf, 'Color', 'white');

    title(['SRRC Filter - R = ' num2str(R) ', Delay = ' num2str(delay)]);

    set(get(gca, 'Title'), 'Color', 'black');

    set(gca, 'XColor', 'black', 'YColor', 'black');

    set(get(gca, 'XLabel'), 'Color', 'black');

    set(get(gca, 'YLabel'), 'Color', 'black');

    xlabel('Time (symbols)');

    ylabel('Amplitude');

    grid on;

    fig\_num = fig\_num + 1;

    % Filtered signal at A

    tx\_filtered = filter(srrc\_filter, 1, tx\_upsampled);

    % Matched filter (Rx)

    rx\_filtered = filter(srrc\_filter, 1, tx\_filtered);

    % Remove filter transients

    filter\_delay = 2 \* delay \* samples\_per\_symbol;

    valid\_tx = tx\_filtered(filter\_delay+1:end);

    valid\_rx = rx\_filtered(filter\_delay+1:end);

    % Plot eye diagram at A

    figure(fig\_num);

    eyediagram(valid\_tx, 2 \* samples\_per\_symbol);

    set(gca, 'Color', 'white');

    set(gcf, 'Color', 'white');

    lines = findall(gca, 'Type', 'line');

    set(lines, 'Color', 'red');

    title(['Eye Diagram at Point A - R = ' num2str(R) ', Delay = ' num2str(delay)]);

    set(get(gca, 'Title'), 'Color', 'black');

    set(gca, 'XColor', 'black', 'YColor', 'black');

    set(get(gca, 'XLabel'), 'Color', 'black');

    set(get(gca, 'YLabel'), 'Color', 'black');

    fig\_num = fig\_num + 1;

    % Plot eye diagram at B

    figure(fig\_num);

    eyediagram(valid\_rx, 2 \* samples\_per\_symbol);

    set(gca, 'Color', 'white');

    set(gcf, 'Color', 'white');

    lines = findall(gca, 'Type', 'line');

    set(lines, 'Color', 'red');

    title(['Eye Diagram at Point B - R = ' num2str(R) ', Delay = ' num2str(delay)]);

    set(get(gca, 'Title'), 'Color', 'black');

    set(gca, 'XColor', 'black', 'YColor', 'black');

    set(get(gca, 'XLabel'), 'Color', 'black');

    set(get(gca, 'YLabel'), 'Color', 'black');

    fig\_num = fig\_num + 1;

end

%%

%%

%-----------------------Functions----------------------------

function [p,denorm\_p] = triangle\_pulse(N)

% TRIANGLE\_PULSE Generates a smooth triangle pulse

%

% Inputs:

%   N  - Number of samples

%

% Output:

%   p  - Triangle pulse vector (normalized energy)

    % Linearly decreasing integers from N to 1

    p = zeros(1, N);

    for i = 1:N

        p(i) = (N - i + 1);

    end

    denorm\_p=norm(p);

    % Normalize to unit energy

    p = p / norm(p);

end

function plot\_pulse\_shape(p, Ts, plot\_title)

% PLOT\_PULSE\_SHAPE Plots a pulse shape over time

%

% Inputs:

%   p          - Pulse shape vector

%   Ts         - Total duration (in seconds) of the pulse

%   plot\_title - Optional plot title (string)

    N = length(p);                     % Number of samples

    t = 0:Ts/N:Ts\*(N-1)/N;                 % Discrete time vector

    figure;

    plot(t, p, 'LineWidth', 2);

    if nargin >= 3

        title(plot\_title);

    else

        title('Pulse Shape');

    end

    xlabel('Time (s)');

    ylabel('Amplitude');

    grid on;

end

function [Unipolar, PolarNRZ, PolarRZ, unipolar\_symbols, polar\_symbols] =...

    generate\_Impulse\_linecodes(Data, A, samples\_per\_bit)

% GENERATE\_LINECODES Generates Unipolar, Polar NRZ, and Polar RZ waveforms.

%

% Inputs:

%   Data            - Array of bits (0s and 1s)

%   A               - Amplitude value

%   samples\_per\_bit - Number of samples per bit (e.g., 5 for 200ms sampling)

%

% Outputs:

%   Unipolar        - Unipolar NRZ waveform

%   PolarNRZ        - Polar NRZ waveform

%   PolarRZ         - Polar RZ waveform

    % Unipolar: 0 -> 0, 1 -> +A

    unipolar\_symbols = A \* Data;                    % Convert bits to unipolar symbols

    Unipolar = upsample(unipolar\_symbols, samples\_per\_bit);  % Upsample for fine resolution

    % Polar NRZ: 0 -> -A, 1 -> +A

    polar\_symbols = A \* (2 \* Data - 1);             % Converts 0→-A, 1→+A

    PolarNRZ = upsample(polar\_symbols, samples\_per\_bit);  % Upsample for fine resolution

    % Polar RZ: same amplitude as Polar NRZ but with half-bit duration pulse

    % → Non-zero followed by zero for each bit

    PolarRZ = zeros(1, length(Data) \* samples\_per\_bit);  % Initialize a zero array for Polar RZ

    for i = 1:length(Data)

        PolarRZ((i-1)\*samples\_per\_bit + 1) = polar\_symbols(i); % Only the first sample for each bit is non-zero

    end

end

function plot\_linecode(t, linecode, plot\_title)

% PLOT\_LINECODE Plots a single linecode signal over time.

%

% Inputs:

%   t          - Time vector

%   linecode   - Signal array representing the linecode

%   plot\_title - Title string for the plot

    % Plot the signal

    figure;

    stem(t, linecode, 'LineWidth', 2);

    title(plot\_title);

    xlabel('Time (s)');

    ylabel('Amplitude');

    grid on;

end

function y\_tx = generate\_pam\_waveform(upsampled\_bit\_stream, p)

    % Converts upsampled bit stream to PAM waveform and convolves with pulse shape p

    % upsampled\_bit\_stream: upsampled binary bit stream array (e.g., [1 0 0 1 ...])

    % p: pulse shaping vector (e.g., [5 4 3 2 1]/sqrt(55))

    % Convolve with pulse shaping function

    y\_tx = conv(upsampled\_bit\_stream, p);

    % Trim the result to match the length of the impulse train (original upsampled length)

    y\_tx = y\_tx(1:length(upsampled\_bit\_stream));  % Match the length of the upsampled input

end

function plot\_pam\_waveform(Ts, y\_tx, plot\_title,smaple\_per\_bit)

    % PLOT\_PAM\_WAVEFORM Plots the PAM waveform using time vector t and adds a zero amplitude line.

    %

    % Inputs:

    %   t      - Time vector (in seconds)

    %   y\_tx   - PAM waveform to be plotted

    % Add a zero element at the end

    y\_tx\_z = [y\_tx, 0];  % Append a zero at the end of the result

    N = length(y\_tx\_z);                     % Number of samples

    t = 0:Ts/smaple\_per\_bit:Ts\*(N-1)/smaple\_per\_bit;                 % Discrete time vector

    % Plot the waveform using the time vector t

    figure;

    plot(t, y\_tx\_z, 'LineWidth', 2);

    title(plot\_title);

    xlabel('Time (s)');  % Using time as x-axis

    ylabel('Amplitude');

    grid on;

    % Add horizontal line at zero amplitude

    yline(0, 'k', 'LineWidth', 1.5);  % 'k' is for a black line

end

function [sampled\_values] = ...

    plot\_RX\_waveform(Ts, y\_rx,shift\_delay, ...

    samples\_per\_bit, bits\_Num, plot\_title, color)

    % PLOT\_RX\_WAVEFORM Plots the received waveform (after filtering) vs time vector t and adds a zero amplitude line.

    %

    % Inputs:

    %   Ts            - Symbol duration (in seconds)

    %   y\_rx          - Receiver output waveform (filtered signal)

    %   samples\_per\_bit - Number of samples per bit

    %   bits\_Num      - Total number of bits in the signal

    %   plot\_title    - Title of the plot

    % Add a zero element at the start to ensure correct length

    y\_rx\_z = shift\_right\_zero(y\_rx, shift\_delay);  % Append a zero at the start of the result

    N = length(y\_rx\_z);                  % Number of samples

    % Create the time vector from 0 to Ts\*(N-1) with time step Ts/samples\_per\_bit

    % Adjust length of t to match the length of y\_rx\_z

    t = 0:Ts/samples\_per\_bit:Ts\*(N-1);

    % Check if the length of the time vector is greater than the signal length

    if length(t) > length(y\_rx\_z)

        t = t(1:length(y\_rx\_z));  % Trim t if necessary to match the length of y\_rx\_z

    end

    % Plot the waveform using the time vector t

    h1 = plot(t, y\_rx\_z, 'Color', color, 'LineWidth', 2);   % Blue line for matched filter output

    title(plot\_title);

    xlabel('Time [Ts sec]');  % Using time as x-axis

    ylabel('Amplitude');

    grid on;

    % Add horizontal line at zero amplitude

    yline(0, 'k', 'LineWidth', 1.5);  % Add a black horizontal line at zero

    % Add dots along the time axis (every Ts\*samples\_per\_bit)

    hold on;

    % Initialize the dot\_values vector with zeros

    dot\_values = zeros(1, N);  % Initialize with zeros

    % Set the value of dot\_values at each symbol's time step (every Ts\*samples\_per\_bit)

    dot\_values(1:samples\_per\_bit:end) = y\_rx\_z(1:samples\_per\_bit:end);  % Update only at the sampled points

    % Initialize dot\_time for every time step

    dot\_time = t;  % Time points corresponding to all samples

    % Plot the hollow dots (red outline, no fill)

    plot(dot\_time, dot\_values, 'o', 'MarkerSize', 6, 'MarkerEdgeColor', 'r', 'MarkerFaceColor', 'r');

    % Add vertical lines and dots manually at each sampled point

    sample\_indices = 1:samples\_per\_bit:N;

    for i = 1:length(sample\_indices)

        x = t(sample\_indices(i));

        y = dot\_values(sample\_indices(i));

        % Draw vertical red line from 0 to sample value

        h2 = line([x x], [0 y], 'Color', 'r', 'LineWidth', 1.5);  % Save handle to one line (for legend)

        % Add a hollow red dot at the top of the line

        plot(x, y, 'o', 'MarkerSize', 6, 'MarkerEdgeColor', 'r', 'MarkerFaceColor', 'w');

    end

    % Set the x-axis ticks and labels

    xticks(0:Ts:max(t));               % Set ticks at multiples of Ts

    xticklabels(0:length(xticks)-1);    % Label ticks as 0, 1, 2, 3, ...

    % Set the x-axis limits as [0, (N-samples\_per\_bit+1)/10]

    xlim([0 ceil((N)/samples\_per\_bit)-1]);

    % Add the legend with the custom line objects

    legend([h1, h2], 'Matched Filter Output', 'Sampled Output', 'Location', 'best', 'Box', 'on');

    hold off;

    % Output: Return only the non-zero sampled values, skip first (padding), ensure row vector

    sampled\_values = y\_rx\_z(sample\_indices(2:end));

    sampled\_values = sampled\_values(:).';  % Force row vector

end

function y\_correlated = correlate\_RX(y, p, Ts)

% CORRELATE\_RX Performs correlation using running integration over each symbol period.

%

% Inputs:

%   y   - Received signal (vector)

%   p   - Pulse shape (vector), must have length Ts

%   Ts  - Samples per symbol

%

% Output:

%   y\_correlated - Correlated output (same length as input, running integration within each symbol)

    % Validate pulse shape length

    if length(p) ~= Ts

        error('Length of pulse shape p must equal Ts (samples per symbol).');

    end

    % Repeat the pulse shape to match the input signal length

    p\_repeated = repmat(p(:), ceil(length(y)/Ts), 1);

    p\_repeated = p\_repeated(1:length(y));

    % Weighted input

    y\_weighted = y(:) .\* p\_repeated;

    % Initialize the output

    y\_correlated = zeros(size(y\_weighted));

    % Process full symbol periods

    full\_symbols = floor(length(y) / Ts);

    for i = 1:full\_symbols

        idx\_start = (i-1)\*Ts + 1;

        idx\_end = i\*Ts;

        y\_correlated(idx\_start:idx\_end) = cumsum(y\_weighted(idx\_start:idx\_end));

    end

    % Handle remaining samples if any

    if mod(length(y), Ts) ~= 0

        idx\_start = full\_symbols\*Ts + 1;

        y\_correlated(idx\_start:end) = cumsum(y\_weighted(idx\_start:end));

    end

end

function y\_shifted = shift\_right\_zero(y, shift\_amt)

%SHIFT\_RIGHT\_ZERO Right-shifts the array by 'shift\_amt' and fills with zeros at the beginning

%

% Inputs:

%   y          - Input vector (row or column)

%   shift\_amt  - Number of positions to shift

%

% Output:

%   y\_shifted  - Shifted vector with zero-padding (same orientation as y)

    if isrow(y)

        y\_shifted = [zeros(1, shift\_amt), y];  % Concatenate as row

    else

        y\_shifted = [zeros(shift\_amt, 1); y];  % Concatenate as column

    end

end

function [y\_filtered\_sampled ,y\_corr\_smapled] = ...

    plot\_matched\_vs\_correlator(Ts, y\_filtered, y\_tx, p,...

    samples\_per\_bit, bits\_Num)

    % PLOT\_MATCHED\_AND\_CORRELATOR Create a figure with two subplots:

    % one for the matched filter output and one for the correlator output.

    %

    % Inputs:

    %   Ts            - Symbol duration (in seconds)

    %   y\_filtered    - Filtered receiver output signal

    %   y\_tx          - Transmitted signal

    %   p             - Matched filter pulse

    %   samples\_per\_bit - Number of samples per bit

    %   bits\_Num      - Total number of bits in the signal

    % Create a new figure for the subplots

    figure;

    % Create the first subplot (Matched Filter Output)

    subplot(2, 1, 1);  % Two rows, one column, first subplot

    y\_filtered\_sampled = plot\_RX\_waveform(Ts, y\_filtered, 1, samples\_per\_bit, bits\_Num, "The receiver output due to matched filter", 'g');

    % Create the second subplot (Correlator Output)

    subplot(2, 1, 2);  % Two rows, one column, second subplot

    y\_corr = correlate\_RX(y\_tx, p, samples\_per\_bit);  % Get the correlated signal

    y\_corr\_smapled = plot\_RX\_waveform(Ts, y\_corr, 1, samples\_per\_bit, bits\_Num, "The correlator Output", 'b');

end

function [y\_matched\_smapled ,y\_hold\_sampled] = ...

    plot\_matched\_vs\_hold(Ts, y\_matched, y\_hold, ...

    samples\_per\_bit, bits\_Num)

    % PLOT\_MATCHED\_AND\_HOLD Create a figure with two subplots:

    % one for the matched filter output and one for the hold filter output.

    %

    % Inputs:

    %   Ts            - Symbol duration (in seconds)

    %   y\_matched     - Matched filter output signal

    %   y\_hold        - Hold filter output signal

    %   samples\_per\_bit - Number of samples per bit

    %   bits\_Num      - Total number of bits in the signal

    % Create a new figure for the subplots

    figure;

    % Create the first subplot (Matched Filter Output)

    subplot(2, 1, 1);  % Two rows, one column, first subplot

    y\_matched\_smapled = plot\_RX\_waveform(Ts, y\_matched, 1, samples\_per\_bit, bits\_Num, "The receiver output due to matched filter", 'c');

    pause(3);

    % Create the second subplot (Hold Filter Output)

    subplot(2, 1, 2);  % Two rows, one column, second subplot

    y\_hold\_sampled = plot\_RX\_waveform(Ts, y\_hold, 1, samples\_per\_bit, bits\_Num, "The receiver output due to hold filter", 'm');

    pause(3);

end

function hold\_filter = make\_hold\_filter(Ts, N, A)

    % MAKE\_HOLD\_FILTER Creates a flat, energy-normalized filter with scaling factor A.

    %

    % Inputs:

    %   Ts  - Duration of the filter (in seconds)

    %   N   - Number of samples

    %   A   - Amplitude scaling factor

    %

    % Outputs:

    %   hold\_filter    - Filter impulse response (flat and energy normalized)

    % Time vector for the filter, from 0 to Ts, divided into N samples

    t = 0:Ts/N:Ts\*(N-1)/N;  % From 0 to Ts with N samples

    % Filter impulse response (flat and energy normalized)

    hold\_filter = A \* ones(size(t)) / sqrt(Ts\*N);  % Amplitude scaled

end

function plot\_all\_outputs\_vs\_input(Ts, message, y\_filtered\_sampled, y\_corr\_sampled, y\_hold\_sampled)

    % PLOT\_ALL\_OUTPUTS\_VS\_INPUT

    %   Plots the original bit stream and the receiver sampled outputs in 1x4 subplots.

    %

    % Inputs:

    %   Ts                  - Symbol duration (in seconds)

    %   message             - Original input bit stream (1×N)

    %   y\_filtered\_sampled  - Output of matched filter sampled (1×N)

    %   y\_corr\_sampled      - Output of correlator sampled (1×N)

    %   y\_hold\_sampled      - Output of hold circuit sampled (1×N)

    % Validate input lengths

    N = length(message);

    if ~isequal(length(y\_filtered\_sampled), N) || ...

       ~isequal(length(y\_corr\_sampled), N) || ...

       ~isequal(length(y\_hold\_sampled), N)

        error('All input vectors must be the same length as the message.');

    end

    % Time vector

    t = 0:Ts:(N-1)\*Ts;

    figure;

    % --- Subplot 1: Original message ---

    subplot(4, 1, 1);

    stem(t, message, 'k', 'filled', 'LineWidth', 1.5);

    title('Input Message');

    xlabel('Time (s)');

    ylabel('Amplitude');

    grid on;

    xlim([0 t(end)]);

    % --- Subplot 2: Matched Filter Output (Impulse representation) ---

    subplot(4, 1, 2);

    stem(t, y\_filtered\_sampled, 'b', 'filled', 'LineWidth', 2);

    title('Matched Filter Output');

    xlabel('Time (s)');

    ylabel('Amplitude');

    grid on;

    xlim([0 t(end)]);

    % --- Subplot 3: Correlator Output (Impulse representation) ---

    subplot(4, 1, 3);

    stem(t, y\_corr\_sampled, 'r', 'filled', 'LineWidth', 2);

    title('Correlator Output');

    xlabel('Time (s)');

    ylabel('Amplitude');

    grid on;

    xlim([0 t(end)]);

    % --- Subplot 4: Hold Output (Impulse representation) ---

    subplot(4, 1, 4);

    stem(t, y\_hold\_sampled, 'g', 'filled', 'LineWidth', 2);

    title('Hold Output');

    xlabel('Time (s)');

    ylabel('Amplitude');

    grid on;

    xlim([0 t(end)]);

end

function [Pe, error\_array] = calculate\_error\_probability(input, output, threshold, A)

    % CALCULATE\_ERROR\_PROBABILITY

    %   Compares the detected symbols with the transmitted symbols using a threshold.

    %   Calculates the probability of error (BER) and provides an error array for visualization.

    %

    % Inputs:

    %   input      - Transmitted symbol stream (1xN) [-A, A]

    %   output     - Received output (1xN) after filtering and sampling

    %   threshold  - Decision threshold (e.g., for polar, it would be 0)

    %   A          - The amplitude used for the symbols (e.g., +A and -A)

    %

    % Outputs:

    %   Pe         - Probability of error (BER)

    %   error\_array - Array of errors (1xN), 0 for no error, 1 for error

    % Decision rule (Apply threshold to output to decide the received symbols)

    decision = output > threshold;  % Decision rule: 1 if output > threshold, else 0

    decision = A\*(2 \* decision - 1);    % Map 1 to +A and 0 to -A (polar decision)

    % Compare the detected symbols (decision) with the transmitted symbols (input)

    error\_array = (decision ~= input);  % 0 for matched, 1 for error

    % Calculate probability of error (BER)

    Pe = sum(error\_array) / length(input);  % Ratio of errors to total bits

end

function [y\_tx\_noise ,SNR\_lin ,N0 , AWGN\_scaled] = AddAWGN(y\_tx, Eb, SNR\_db)

    % AddAWGN - Add Additive White Gaussian Noise (AWGN) to a transmitted signal

    %

    % Inputs:

    %   y\_tx       - Transmitted signal (1xN array)

    %   Eb         - Energy per bit (scalar)

    %   SNR\_db     - Signal-to-Noise Ratio in dB (scalar)

    %

    % Output:

    %   y\_tx\_noise - Noisy received signal (1xN array)

    % Length of transmitted signal

    N\_tx = length(y\_tx);

    % Generate unity variance, zero mean white Gaussian noise

    AWGN = randn(1, N\_tx);  % Generate noise with the same length as y\_tx

    % Calculate the noise power spectral density based on Eb and SNR

    SNR\_lin = 10^(SNR\_db/10.0);  % Convert SNR from dB to linear scale

    N0 = Eb / SNR\_lin;           % Calculate noise power (N0)

    % Scale the noise to have variance N0/2

    AWGN\_scaled = sqrt(N0 / 2) \* AWGN;

    % Add the noise to the transmitted signal

    y\_tx\_noise = y\_tx + AWGN\_scaled;  % Noisy received signal

end

function [BER\_matched, BER\_hold, theoretical\_BER] = ...

    BER\_vs\_SNR(y\_tx, Eb, p\_matched, hold\_filter, ...

    samples\_per\_bit, bits\_Num, bit\_stream\_symbols, A, Eb\_N0\_dB)

    % BER\_VS\_SNR - Calculate and plot the Bit Error Rate (BER) vs Eb/N0 for different filters

    %

    % Inputs:

    %   y\_tx             - Transmitted signal (1xN array)

    %   Eb               - Energy per bit (scalar)

    %   p\_matched        - Matched filter (1xN array)

    %   hold\_filter      - Hold filter (1xN array)

    %   samples\_per\_bit  - Number of samples per bit (scalar)

    %   bits\_Num         - Number of bits (scalar)

    %   bit\_stream\_symbols - Transmitted bit stream symbols (1xN array)

    %   A                - The amplitude used for the symbols (+A or -A)

    %   Eb\_N0\_dB         - Range of Eb/N0 in dB

    % Initialize BER arrays for matched filter, hold filter, and theoretical BER

    BER\_matched = zeros(size(Eb\_N0\_dB));

    BER\_hold = zeros(size(Eb\_N0\_dB));

    theoretical\_BER = zeros(size(Eb\_N0\_dB));

    % Loop over all Eb/N0 values

    for i = 1:length(Eb\_N0\_dB)

        % Add noise to the transmitted signal

        [y\_tx\_noise, SNR\_lin, N0, AWGN\_scaled] = AddAWGN(y\_tx, Eb, Eb\_N0\_dB(i));

        % Convolve the noisy signal with the matched filter

        y\_matched = conv(y\_tx\_noise, p\_matched);

        % Convolve the noisy signal with the hold filter

        y\_hold = conv(y\_tx\_noise, hold\_filter);

        % Sample the matched filter and hold filter outputs

        [y\_filtered\_sampled, y\_hold\_sampled] = plot\_matched\_vs\_hold(1, y\_matched, y\_hold, samples\_per\_bit, bits\_Num);

        % Calculate the BER for matched filter output

        polar\_threshold = 0;

        [BER\_matched(i), ~] = calculate\_error\_probability(bit\_stream\_symbols, y\_filtered\_sampled, polar\_threshold, A);

        % Calculate the BER for hold filter output

        [BER\_hold(i), ~] = calculate\_error\_probability(bit\_stream\_symbols, y\_hold\_sampled, polar\_threshold, A);

        % Calculate theoretical BER

        theoretical\_BER(i) = 0.5 \* erfc(sqrt(SNR\_lin));  % Theoretical BER formula

    end

end

function plot\_BER\_vs\_EbN0(Eb\_N0\_dB, BER\_matched, BER\_hold, theoretical\_BER)

    % plot\_BER\_vs\_EbN0 - Plot the Bit Error Rate (BER) vs Eb/N0 for different filters

    %

    % Inputs:

    %   Eb\_N0\_dB       - Eb/N0 values in dB (vector)

    %   BER\_matched    - BER for the matched filter (vector)

    %   BER\_hold       - BER for the hold filter (vector)

    %   theoretical\_BER - Theoretical BER (vector)

    % Create figure for the plot

    figure;

    % Plot BER vs Eb/N0 on a semi-logarithmic scale (log scale on y-axis)

    semilogy(Eb\_N0\_dB, BER\_matched, 'b-o', 'LineWidth', 2); % Matched filter

    hold on;

    semilogy(Eb\_N0\_dB, BER\_hold, 'r-x', 'LineWidth', 2); % Hold filter

    semilogy(Eb\_N0\_dB, theoretical\_BER, 'k--', 'LineWidth', 2); % Theoretical BER

    hold off;

    % Set the labels and title

    xlabel('Eb/N0 (dB)');

    ylabel('Bit Error Rate (BER)');

    title('BER vs Eb/N0 with Matched Filter and Hold Filter');

    % Add legend to identify each curve

    legend('Matched Filter', 'Hold Filter', 'Theoretical BER');

    % Enable grid for better visibility

    grid on;

end