Cairo University, Faculty of Engineering
Electronics and Electrical Communications
Department (EECE)

Digital Communication

Project2_ELC3070

Team:13

Name	ID	Sec
احمد محمد إبراهيم محمد عبده	9220078	1
روان خالد محمد محمد	9220303	2
محمد خالد محمد شتا	9203230	3
احمد خلف محمد علي	9220036	1
خالد محمد رمضان علي	9221589	2

Submitted to: Eng. Mohamed Khaled

Date: 18-4-2025

Table of Contents

Table of figures

The Role of each member

We all contributed to writing the report.

Name	ID
احمد محمد إبراهيم عبده	Pulse Shaping and Noise Generation
روان خالد محمد محمد	Matched Filter, Rectangular Filter, and Correlator
محمد خالد محمد شتا	Ber Calculation
احمد خلف محمد علي	ISI and Raised Cosine Filter
خالد محمد رمضان علي	Ber Calculation

Introduction:

In modern digital communication systems, the efficient transmission and reliable detection of signals in the presence of noise and distortion are crucial for maintaining high performance. One of the core techniques used in digital baseband transmission is Pulse Amplitude Modulation (PAM), where binary information is transmitted using modulated pulses shaped by a pulse shaping filter. The shape of the pulse plays a significant role in controlling intersymbol interference (ISI) and optimizing signal detection at the receiver.

To detect the transmitted PAM signals with minimal error, the receiver employs filters that maximize the Signal-to-Noise Ratio (SNR) at the sampling instants. The most optimal among these is the matched filter, which is derived based on the transmitted pulse and is proven to provide the highest SNR in an Additive White Gaussian Noise (AWGN) channel. Additionally, the correlator, which performs integration over the pulse duration, can be used as an alternative approach with similar performance in noise-free environments. Furthermore, to combat ISI introduced by the communication channel or improper pulse shaping, raised cosine filters are employed at both the transmitter and receiver. These filters help achieve zero ISI at the sampling instants when their parameters are chosen.

Problem description:

This project aims to simulate and analyze the performance of binary PAM communication systems under both noise-free and noisy environments in MATLAB. The simulation involves the use of a predefined normalized pulse shaping function and examines different receiver structures including the matched filter and simple correlator.

The goals of the project are:

1. **Matched Filter, Hold filter and Correlator Comparison**:
   Simulate a binary PAM system using a normalized discrete pulse. Compare the outputs of a matched filter, Hold filter and a correlator in a noise-free scenario to evaluate their performance at ideal sampling instants.
2. **BER Analysis in AWGN Channel**:
   Extend the simulation to a noisy environment by introducing additive white Gaussian noise (AWGN) to the transmitted signal. Analyze and compare the Bit Error Rate (BER) performance of two different receiver filters.
3. **ISI Investigation with Raised Cosine Filters**:
   Investigate the effect of intersymbol interference by using square root raised cosine filters at both transmitter and receiver. Generate and analyze eye patterns for different roll-off factors and filter delays to study their impact on signal clarity and timing recovery.

Control Flags:

To make and study our outputs, we use a set of key settings in MATLAB that guide how everything works. Here's what they are and why they matter:

- **Symbol Duration (T)**:

Ts = 1; defines the symbol duration in seconds, representing the time interval for transmitting one symbol. This parameter controls the system's symbol rate (1/Ts symbols per second) and scales the time axis for sampling and plotting. A larger Ts slows down the symbol rate, stretching the signal in time, while a smaller Ts increases the rate, compressing the signal. It's critical for aligning the up sampling, filtering, and sampling processes.

- **Samples per Symbol**:

Samples_per_symbol = 5; specifies the number of samples per symbol, determining the up-sampling factor and the resolution of the digital signal (Ts). This parameter controls how many discrete points represent each symbol in the up sampled signal, affecting the accuracy of pulse shaping and filtering. A higher value increases resolution but also computational complexity, while a lower value risks aliasing or loss of signal fidelity.

- **Number of Bits**:

Num_bits = 10; sets the number of random binary bits to generate for the input signal. This parameter controls the length of the transmitted sequence, directly impacting the duration of the simulation and the number of symbols processed. A larger Num_bits extends the signal for analysis, while a smaller value reduces it.

- **Pulse Shaping Filter**:

p = [5 4 3 2 1]/sqrt(55); defines the normalized pulse shape used for transmitting symbols. This parameter shapes the signal to control its bandwidth and intersymbol interference. The specific coefficients [5 4 3 2 1] create a triangular-like pulse, and normalization ensures unit energy. Changing p alters the signal's spectral properties and affects the matched filter's performance.

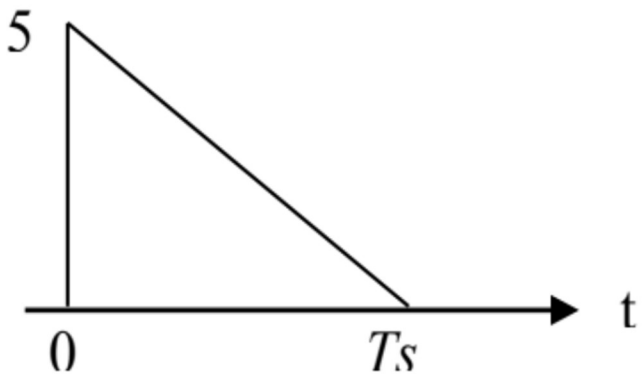

Figure 1. Pulse shaping filter

Data Generation:

The data is generated using the following commands:

```
bits = randi([0 1], 1, Num_bits);

signal = 2*bits - 1;  % Converts 0->-1 and 1->+1

upsampled_signal = upsample(signal, samples_per_symbol); %(insert 4 zeros)

y = conv(upsampled_signal, p, 'full'); % Convolve with pulse shaping function
```

The first line generates a row vector of length 10 containing random binary values (0 or 1) with equal probability. Each element represents a single bit in the input data sequence.

- **Generating bipolar signal:**

The second line maps the binary bits to a bipolar signal, converting (0 to -1) and (1 to +1).

- **Up Sampling:**

The next line up samples the bipolar signal by inserting 4 zeros between each symbol. The up-sample function increases the sampling rate by a factor of 5.

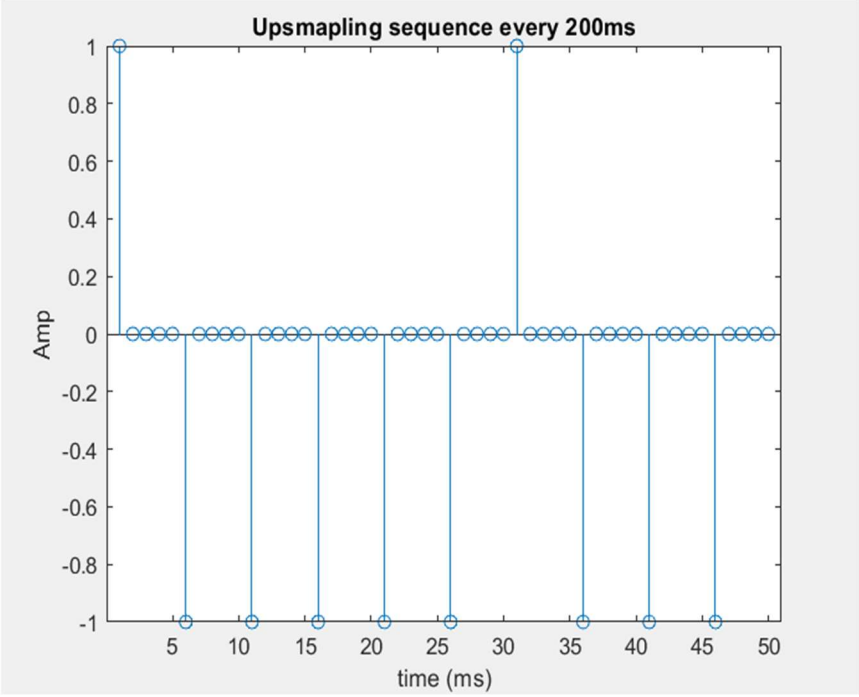

Figure 2. Up sampled data form

- **Convolution with pulse shaping filter:**

The last line convolves the up sampled signal with the pulse shaping filter [p] to produce the transmitted waveform [y].

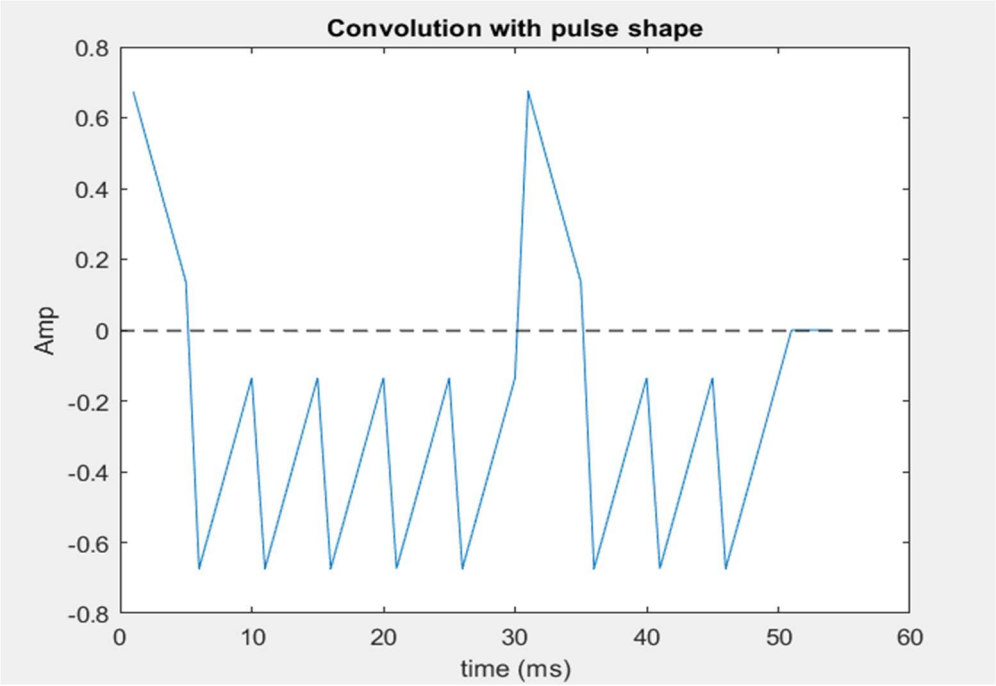

Figure 3. Convolution of random bits with pulse shaping filter

Note: The 'full' option ensures the convolution includes all output samples, accounting for the overlap between pulses. The result is a continuous-time-like waveform of the transmitted signal.

Matched Filter:

- **Theoretical base:**

A matched filter is a linear filter designed to maximize the signal-to-noise ratio (SNR) at the output of a receiver in the presence of additive white Gaussian noise (AWGN). It is widely used in digital communication systems to detect known signals embedded in noise. The theoretical foundation of the matched filter comes from the principle that the filter's impulse response should be matched to the transmitted signal's waveform to optimize detection.

- **Definition and Derivation**

Consider a transmitted signal $s(t)$ that is received in the presence of noise $n(t)$, so the received signal is: $$r(t) = s(t) + n(t)$$

The goal is to design a filter with impulse response $h(t)$ such that the output SNR at a specific sampling time T is maximized. The output of the filter at time [t] is given by the convolution:

$$y(t) = r(t) * h(t) = \int_{-\infty}^{\infty} r(\tau)h(t-\tau)\,d\tau$$

At the sampling instant [t = T] , the output is:

$$y(T) = \int_{-\infty}^{\infty} r(\tau)h(T-\tau)\,d\tau$$

This consists of a signal component (from $s(t)$) and a noise component (from $n(t)$). The SNR is defined as the ratio of the squared signal output to the noise variance at [t=T]. To maximize SNR, the filter $h(t)$ should emphasize the signal while minimizing the noise contribution.

Using the Cauchy-Schwarz inequality, it was proved that the SNR is maximized when the filter's response is proportional to the time-reversed and delayed version of the transmitted signal:

$$h(t) = k \cdot s(T_S - t)$$

Substituting $h(t)$ in integration , the output becomes:

$$y(t) = \int_{-\infty}^{\infty} r(\tau)\,.s(T_S - (t-\tau))\,d\tau = \int_{-\infty}^{\infty} r(\tau)\,.s(T_S - t + \tau)\,d\tau$$

This integral represents the correlation of the received signal $r(t)$ with the time-reversed pulse $s(T_S - t)$, producing a peak at [t=Ts] when $r(t)$ contains the transmitted pulse $s(t)$, maximizing the signal-to-noise ratio (SNR) at the sampling instant.

- **Operation in discrete**

In discrete time, for a received signal r[n] and the matched filter h[n] , is our reversed normalized pulse shape [1 2 3 4 5] and for 5 samples per bit, The output will be:

$$y[n] = \sum_{k=0}^{4} r(n-k) \times \frac{[\,1\;2\;3\;4\;5\,][k+1]}{\sqrt{55}}$$

Where the term $[k+1]$ is used to index the specific number from reversed pulse matrix.

This computes a weighted sum of 5 consecutive samples of $r(t)$, where the weights are the reversed pulse coefficients and normalized, producing peaks at the correct sampling points that align with the transmitted symbols (±1).

- **Simulation base:**

```
% Matched filter
h_matched = fliplr(p);

y_matched = conv(y, h_matched, 'full');
```

Matched Filter design:
The first line creates the matched filter impulse response [h] by reversing the time-domain pulse shaping filter [p]. The function fliplr (flip left-right) takes the vector [p] and reverses its order.

Convolution with matched filter:
The second line convolves the received waveform [y] the output of the pulse shaping, with the matched filter to produce the filtered output [y_matched].

Hold Filter:

- **Theoretical base:**

A simple hold filter, also known as a rectangular filter, boxcar filter, or integrate-and-dump filter in communication systems, is a basic receiver filter that averages or integrates the received signal over a fixed time window, typically the symbol duration. Unlike the matched filter, which is optimally designed to maximize the signal-to-noise ratio (SNR) by correlating the received signal with the time-reversed transmitted pulse, the hold filter is a simpler, suboptimal approach that assumes a constant signal value over the symbol period.

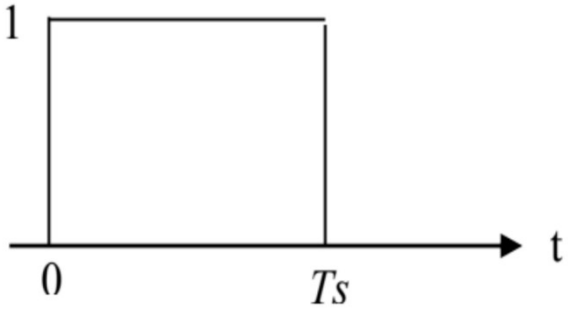

Figure 4. Required Hold filter design

- **Definition and Purpose**

The hold filter's impulse response is a rectangular pulse, uniform over a duration equal to the symbol period (or a multiple of the sampling interval). In continuous time, if the symbol duration is [Ts], the hold filter's impulse response can be expressed as:

$$h(t) = \begin{cases} 1/\sqrt{T_S} & 0 \le t \le T_S \\ 0 & otherwise \end{cases}$$

Here, the factor $1/\sqrt{T_S}$ normalizes the filter's energy:

$$\int_0^{T_S} (1/\sqrt{T_S})^2 \, dt = 1$$

The hold filter's purpose is to integrate the received signal over the symbol duration to estimate the transmitted symbol's amplitude. It assumes the signal is relatively constant over the symbol period, which holds true for rectangular pulses but is less accurate for shaped pulse.

The output of the hold filter is obtained by convolving the received signal $r(t)$ with $h(t)$:

$$y(t) = r(t) * h(t) = \int_{-\infty}^{\infty} r(\tau) h(t-\tau) \, d\tau$$

➢ **Operation in discrete**

In discrete time, for a received signal r[n] and filter h(n) , the output is:

$$y[n] = \sum_{k=0}^{4} r(n-k) \times \frac{1}{\sqrt{5}}$$

This computes a normalized sum of 5 consecutive samples, effectively averaging the signal over one symbol period.

➢ **Comparing Hold to Matched Filter**

Unlike the matched filter, whose impulse response $h(t) = s(T_S - t)$ is tailored to the transmitted pulse shape $s(t)$, the hold filter uses a uniform response, ignoring the specific shape of the transmitted pulse. This makes it:

- **Suboptimal for SNR**: The hold filter does not maximize SNR unless the transmitted pulse is perfectly rectangular and aligned with the filter's window. For non-rectangular pulses (like p), it captures less signal energy and is more susceptible to noise.
- **Simpler to Implement**: The hold filter requires minimal computation (just averaging), making it attractive for low-complexity receivers.
- **Sensitive to Timing Errors**: Misalignment between the filter window and symbol boundaries can degrade performance, as it doesn't emphasize the pulse's peak like a matched filter.
- **Intersymbol Interference (ISI)**: For pulses with tails (like p), the hold filter may include energy from adjacent symbols, increasing ISI compared to a matched filter designed to minimize it.

- **Simulation base:**

```
% Alternative Simple hold filter (all ones, normalized)
h_alt = ones(1,5)/sqrt(5);

y_alt = conv(y, h_alt, 'full');
```

Hold Filter design;

The first line defines the impulse response of the simple hold filter, as a row vector of five ones, with normalized energy.

 Convolution with hold filter:

The second line convolves the received waveform [y] the pulse-shaped signal, with the hold filter to produce the filtered output [y_alt].

Correlator:

• Theoretical base:

In communication systems, a correlator is used to identify transmitted symbols by sliding the known pulse shape over the received signal, producing peaks when the pulse aligns with a symbol. The correlator is closely related to the matched filter, as both perform similar operations, but they differ in implementation and interpretation.

➢ Definition and Purpose

The cross-correlation between a received signal $r(t)$ and the pulse shape $s(t)$ is defined in continuous time as:

$$y(\tau) = \int_{-\infty}^{\infty} r(\tau)s(t-\tau)\, d\tau$$

The integral measures how well $r(t)$ matches $s(t)$ when shifted by τ. Peaks in $y(\tau)$ indicate the presence of symbols at specific shifts.

➢ Operation

In discrete time, for a received signal r[n] and pulse p[m], the correlation at shift n is:

$$y[n] = \sum_{m} y(n+m).p(m)$$

The correlator's output at a discrete time index [n] approximates this by computing the dot product between a segment of the received signal and the pulse, adjusting for alignment. The goal is to detect the transmitted symbols (±1 in the bipolar signal) by finding where [r] (derived from p) correlates strongly with [p].

➢ Comparing Correlator to Matched Filter

The matched filter, with impulse response $h(t) = s(T_S - t)$, is tailored to the transmitted pulse shape $s(t)$, while the correlator computes the cross-correlation $y(\tau) = \int r(t)s(t-\tau)\, dt$, directly measuring similarity with $s(t)$, This makes them:

- **Equivalent for SNR**: Both maximize SNR in AWGN channel, as the correlator's output at [τ=0] matches the matched filter's at [t=Ts], fully capturing the signal energy of pulses.

- **Similar Complexity**: Both require correlating the received signal with the pulse, involving comparable computations, though the correlator's shift-based approach may emphasize timing estimation.
- **Equal Timing Sensitivity**: Misalignment in sampling (matched filter) or shift (correlator) degrades performance equally, as both rely on precise alignment with $s(t)$.
- **Identical ISI Handling**: For pulses like [p], both minimize ISI when sampled correctly, producing equivalent outputs at symbol boundaries, unlike suboptimal filters that ignore pulse shape.

- **Simulation base:**

```
% Correlator filter
y_correlator = zeros(samples_per_symbol * Num_bits, 1);  % Output size: 5 * 10 = 50
y_reshaped = reshape(y(:, 1:50), 5, 10);  % 5 samples, 10 bits

% Loop over each bit
for n = 1:Num_bits
    % Loop over each correlation output for the current bit
    for i = 1:samples_per_symbol
        sum_result = 0;
        % Compute correlation: sum of p(m) * y_reshaped(m, n) for m = 1 to i
        for m = 1:i
            sum_result = sum_result + p(m) * y_reshaped (m, n);
        end
        % Store result in y_correlator
        y_correlator((n-1)*samples_per_symbol + i) = sum_result;
    end
end
```

 Pre-allocation: Allocates memory for the y_correlator vector to store the correlation results, ensuring efficient memory usage.

 Reshaping the Input Matrix : Reshapes a portion of the input data into a matrix, where each column represents a bit, and each row corresponds to a sample of that bit.

 First Loop (Over Bits): Loops over each bit, processing them one by one.

 Second Loop (Over Samples): Loops over the samples for each bit, processing each sample sequentially.

 Third Loop (Summing Correlation): For each sample, it sums the product of the signal and the corresponding bit samples up to that sample.

 Storing the Result: Stores the calculated correlation result in the y_correlator vector at the correct position.

Requirement 1:

Q1) Draw the output of both matched and hold filters on two subplots in the same figure using two different colors, assuming a noise free system. Compare the output of the filters at the sampling instants.

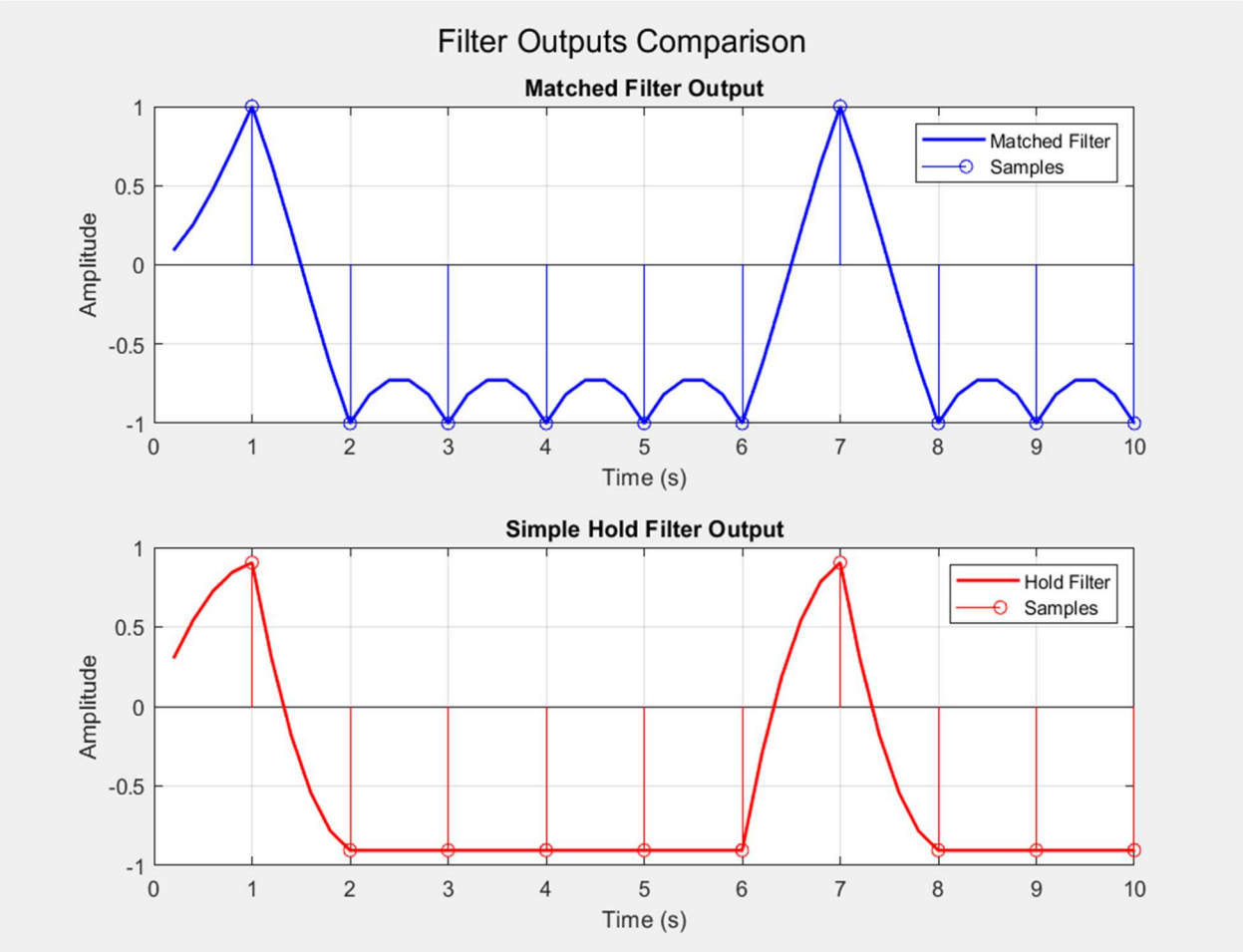

Figure 5. Matched filter vs Hold filter

 Matched Filter Output: The matched filter maximizes the signal at the sampling instants, producing a peak output that aligns with the correlation of the input signal and its matched pulse shape. At the sampling instant, the matched filter output achieves its maximum value, reflecting optimal detection performance.

 Hold Filter Output: The hold filter maintains the input signal's value constant over each sampling period, resulting in a piecewise constant output. At the sampling instant, the hold filter's output doesn't show the peak enhancement seen in the matched filter.

The visual distinction in the plots, with the matched filter showing sharper peaks and the hold filter displaying flat segments, underscores these differences in their time-domain behavior.

Q2) Draw the output of the matched filter and the output of a correlator to p[n] on the same plot with two different colors.

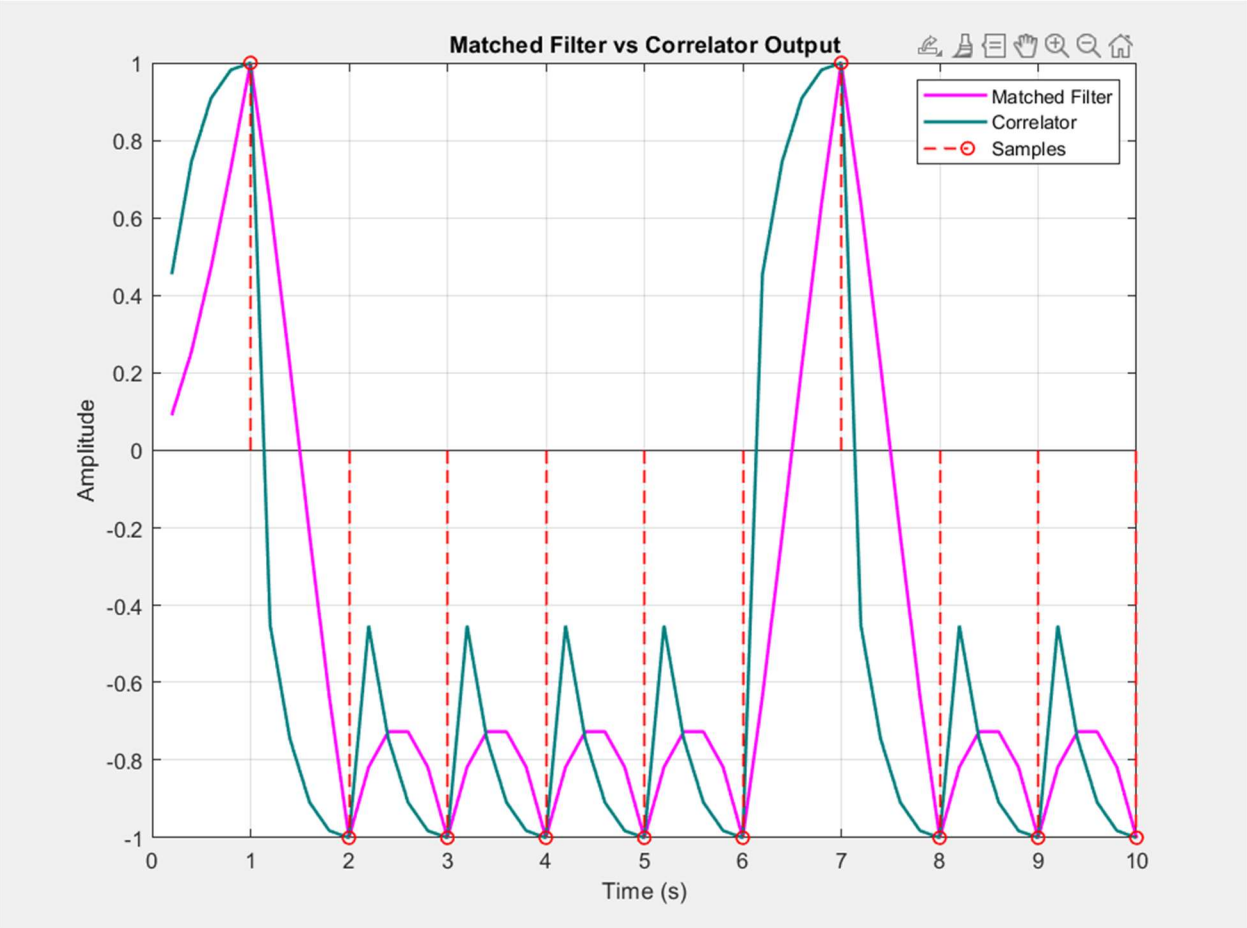

Figure 6. Matched filter Vs correlator outputs

 Despite their different implementations, both approaches yield identical results at the symbol sampling instant. This means that when sampling at the correct decision times, the output amplitudes from both filters are the same. This confirms that both techniques effectively extract the transmitted signal energy aligned with the expected pulse shape.

BER & Noise Analysis

In this section, we extend the simulation to a noisy environment by introducing Additive White Gaussian Noise (AWGN) to the transmitted signal. The objective is to analyze and compare the Bit Error Rate (BER) performance of two different receiver filters: matched filter - simple hold filter (with the following response). This analysis helps evaluate the effectiveness of each filter under realistic channel conditions.

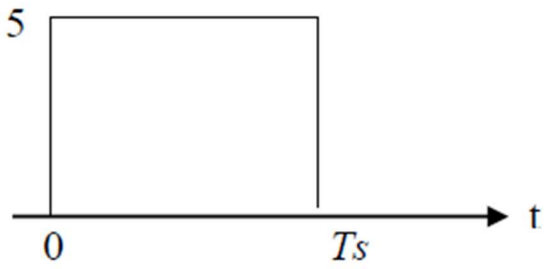

Figure 7: Required Hold filter design

- **Theoretical base:**

After generating 10,000 random bits instead of 10, the same steps are taken to get the transmitted signal y[n], and depending on signal to noise ratio SNR = [-2, -1, … 5] (in dB) AWGN is generated with variance $\sigma^2 = \frac{N_0}{2}$, where N_0 is the noise power spectral density.

$$\frac{E_b}{N_0} = 10^{SNR/10}$$

Where E_b is bit energy, and each bit is represented by the normalized pulse p[n]

$$\therefore E_b = 1\ (Energy\ unit) \rightarrow N_0 = \frac{1}{E_b/N_0}$$

Then, received signal before the filter block is: rx_signal[n] = y[n] + AWGN

We now need to check which filter is better in guessing the true value transmitted after exposing it to noise added in the channel, so we filter the noisy signal received by both filters: matched filter - hold filter.

The filtered outputs are sampled at every T_s (symbol duration) to extract the corresponding output sample values. These samples are then passed to the decision-making stage, where a decision threshold of 0 is used; samples greater than 0 are mapped to 1, while those less than 0 are mapped to -1.

How can we decide which filter is better in minimizing the probability of error?

We calculate the bit error rate BER on a large scale of transmitted bits (10,000 bits) for both filters. The filter with less BER value is better, where BER is calculated by dividing the number of bit errors by the total number of bits.

Theoretical BER is calculated using the formula: $BER = \frac{1}{2}\, erfc(\sqrt{\frac{E_b}{N_0}})$

- **Simulation base:**

```
% Adding Noise and Calculating BER
% Initialize BER arrays for matched filter, hold filters, and theoretical
BER_matched = zeros(size(EbN0_dB));
BER_hold = zeros(size(EbN0_dB));
BER_theo = zeros(size(EbN0_dB));

% Matched filter
h_matched = fliplr(p);
h_hold = ones(1,5) / sqrt(5);  % Simple hold filter with 5 samples, un-normalized
h_hold = h_hold/(5*sqrt(5)); % Simple hold filter with 5 samples, normalized

% Sampling points
sampling_indices = (1:Num_bits) * samples_per_symbol;

% Loop over Eb/N0 values
for idx = 1:length(EbN0_dB)
    EbN0 = 10^(EbN0_dB(idx)/10);  % Convert Eb/N0 fr%om dB to linear scale
    N0 = 1 / EbN0;                % Since pulse p is normalized, Eb = 1

    % Generate Gaussian noise with appropriate variance
    noise = sqrt(N0/2) * randn(size(y));  % Generate AWGN noise
    rx_signal = y + noise;                % Received signal after adding noise

    % Matched filter output
    rx_matched = conv(rx_signal, h_matched, 'full');
    matched_samples = rx_matched(sampling_indices);

    % Hold filter output
    rx_hold = conv(rx_signal, h_hold, 'full');
    hold_samples = rx_hold(sampling_indices);

    % Decision making (threshold = 0)
    matched_decisions = matched_samples > 0;
    hold_decisions = hold_samples > 0;

    % Calculate BER for matched filter
    BER_matched(idx) = sum(matched_decisions ~= bits) / Num_bits;

    % Calculate BER for hold filter
    BER_hold(idx) = sum(hold_decisions ~= bits) / Num_bits;

    % Calculate Theoretical BER value
    BER_theo(idx) = 0.5 * erfc(sqrt(EbN0));
end
```

 Pre-allocation: Allocates memory for the BER arrays (BER_matched, BER_hold, and BER_theo) to store the BER values for each SNR value.

 h_matched: The time reversed version of the pulse shaping filter.

h_hold: A basic uniform (rectangular) filter with normalized amplitude, used to simulate a simple non-ideal receiver filter.

sampling_indices: Picks sampling points after filtering. Since we use 5 samples per symbol, every 5th sample corresponds to one transmitted symbol.

 Main loop over SNR: Iterates over each SNR value in dB, Converts SNR from dB to linear scale, assumes the bit energy $E_b = 1$, derive $N_o = \frac{1}{E_b/N_o}$

Generating AWGN: Generates additive white Gaussian noise with zero mean and variance $\frac{N_o}{2}$. The noise is added to the transmitted signal to simulate a noisy channel..

 Filtering the noisy signals: Applies convolution between the received signal and both filters (h_matched and h_hold). tores the filtered results in rx_matched and rx_hold.

 Sampling results: Extracts symbol values by sampling the filtered signals at symbol instants. The outputs are stored in matched_samples and hold_samples.

 Decision making (threshold = 0): Applies a decision rule: if the sampled value > 0, the detected bit is 1; otherwise, it is 0.

 BER Calculation: Compares detected bits with the transmitted bits. Computes the Bit Error Rate (BER) as: $BER = \frac{Number\ of\ Bit\ Errors}{Total\ Bits}$

 Theoretical BER: Calculated using the standard formula for BPSK in AWGN:

$$BER_{theoretical} = \frac{1}{2}\ erfc(\sqrt{\frac{E_b}{N_0}})$$

Requirement 2:

BER Performance Comparison

The BER performance of the matched filter and the hold filter is compared against the theoretical BER curve. The results are plotted in Figure 8 below.

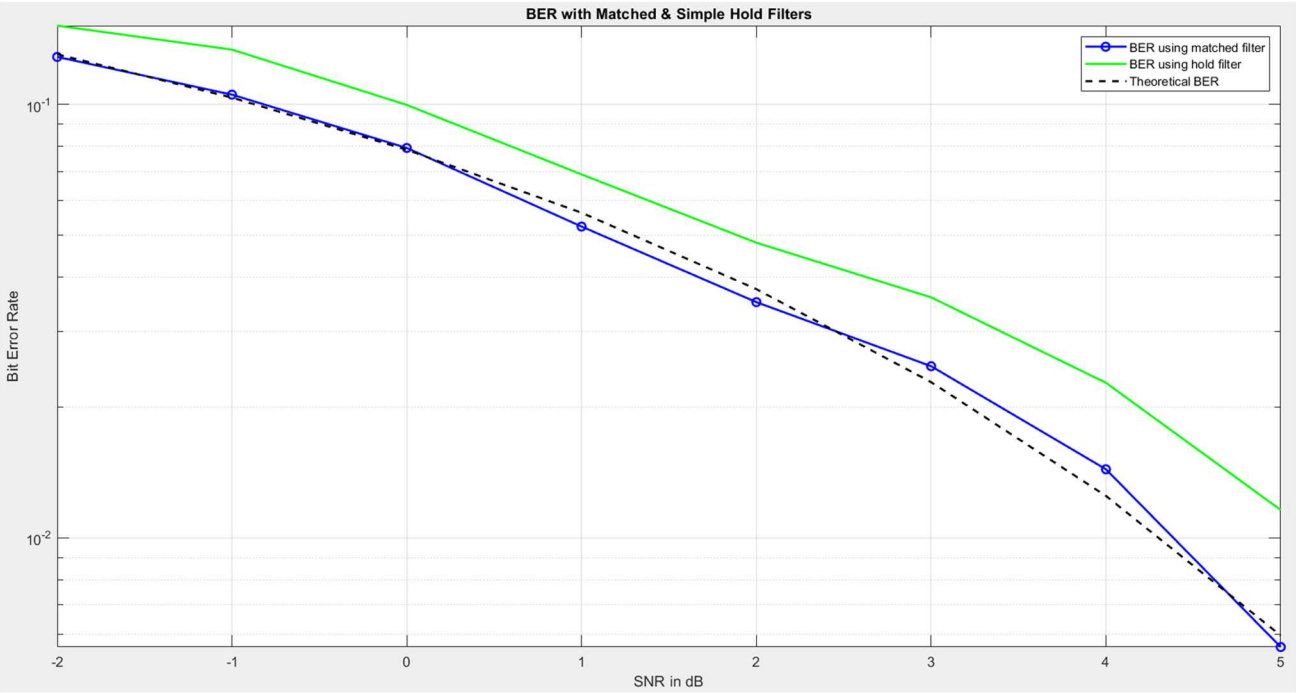

Figure 8: BER vs SNR for matched & hold filters

1. **Matched Filter:** The blue curve represents the measured BER for the matched filter's output. It closely follows the theoretical BER curve, indicating that the matched filter is maximizing the output Signal-to-Noise Ratio (SNR) at the sampling instants, hence it is the optimum filter.
2. **Hold Filter:** The green curve shows the measured BER for the hold filter's output. Compared to the matched filter, the hold filter exhibits higher BER values, due to its suboptimal design, which assumes a constant signal value over the symbol period without considering the transmitted pulse shape.
3. **Theoretical BER:** The dashed black line represents the theoretical BER, providing a standard for optimal performance.

Quantitative Analysis

The following table summarizes the measured BER values for both filters at selected SNR levels:

SNR	MATCHED FILTER	HOLD FILTER	THEORETICAL
-2	0.1284	0.1551	0.1306
-1	0.1021	0.1287	0.1038
0	0.0785	0.1012	0.0786
1	0.0604	0.0781	0.0563
2	0.0376	0.0522	0.0375
3	0.0212	0.0319	0.0229
4	0.0129	0.0209	0.0125
5	0.0049	0.0115	0.0060

Table 1: BER Values at Selected SNR Levels

From the table, it is shown that the matched filter achieves BER values very close to the theoretical limit, while the hold filter exhibits higher error rates, particularly at lower SNR values.

Practical Outcomes:

- In real-world communication systems, the matched filter is preferred due to its ability to achieve near-optimal performance.
- While the hold filter is simpler to implement, its suboptimal performance makes it less suitable for scenarios with high noise levels.

In conclusion, the BER analysis confirms the superiority of the matched filter in terms of error performance under AWGN conditions. The results highlight the importance of designing receiver filters that are tailored to the transmitted signal's characteristics to achieve a reliable communication system.

ISI &&Raised Cosine

- **Theoretical base**

A **raised cosine (RC) filter** is designed to satisfy the Nyquist criterion and reduce ISI while maintaining bandwidth efficiency.

Impulse Response:

$$h(t) = \frac{sin(\pi t/T)}{\pi t/T} \cdot \frac{\cos(\pi \beta t/T)}{1-(2\beta t/T)^2}$$

- T: Symbol period β: Roll-off factor ($0 \leq \beta \leq 1$)

- **Simulation base:**

```
% ISI && Raised cosine
data_length = 100;       % Length of the data (bits)
data = randi([0, 1], 1, data_length);
Data_sequence_maped = 2*data - 1; % maping the output to 1 or -1 %
Data_sequence_UpSample = upsample(Data_sequence_maped, samples_per_symbol);

i=1;
for R = [0, 1]
    for delay = [2, 8]
            h = rcosdesign(R, delay, samples_per_symbol ,"sqrt");
        % Apply square root raised cosine first
            A_Tx = filter(h ,1 , Data_sequence_UpSample);
        % ideal channel free noise mean H(s)=1not make any change
            B_Rx = filter(h ,1 , A_Tx); % Apply square root raised cosine first
        %The outcome was applying raised cosine to the data.
        % draw rsosine filter
            figure(i);
            i = i + 7;
            plot(h);
            title("Square Root Raised Cosine Filter (R: " + num2str(R) + ", Delay: " +
num2str(delay) + ")");
            xlabel("Time Samples");
            ylabel("Amplitude");
            ylim([min(h)-0.01 max(h)+0.01]);
         % draw Eye diagram
             eye_fig_TX=eyediagram(A_Tx(delay * samples_per_symbol + 1:end - delay *
samples_per_symbol)', 2 * samples_per_symbol);
             set(eye_fig_TX,'Name',"A_Tx_eyediagram for R :" + R + " Delay: " + delay);
             eye_fig_RX=eyediagram(B_Rx(delay * samples_per_symbol +1:end - delay *
samples_per_symbol)', 2 * samples_per_symbol);
             set(eye_fig_RX,'Name',"B_Rx_eyediagram for R :" + R + " Delay: " + delay);
    end
end
```

- We tested two values for the roll-off factor (0 and 1) and two values for the delay (2 and 8) to observe their impact on the filter and signal.
- For each case, we designed a Square Root Raised Cosine (SRRC) filter and applied it to the upsampled data at the transmitter side.
- The signal was passed through an ideal channel with no distortion, then filtered again at the receiver using the same SRRC filter.
- Using the SRRC filter on both sides results in a full Raised Cosine response, which helps eliminate intersymbol interference (ISI).
- We plotted the filter response to visualize the effect of changing the roll-off factor and delay.
- We removed the transient parts at the beginning and end of the signal that caused by the filter delay to ensure accurate results during analysis.
- Eye diagrams were generated before and after the receiver filter to analyze the signal quality and ISI levels.
- All plots were labeled with the corresponding values of R and delay for easy comparison.

Requirement 3

plot the eye pattern for the data length of 100 bits at points A and B For the 4 cases mentioned above.

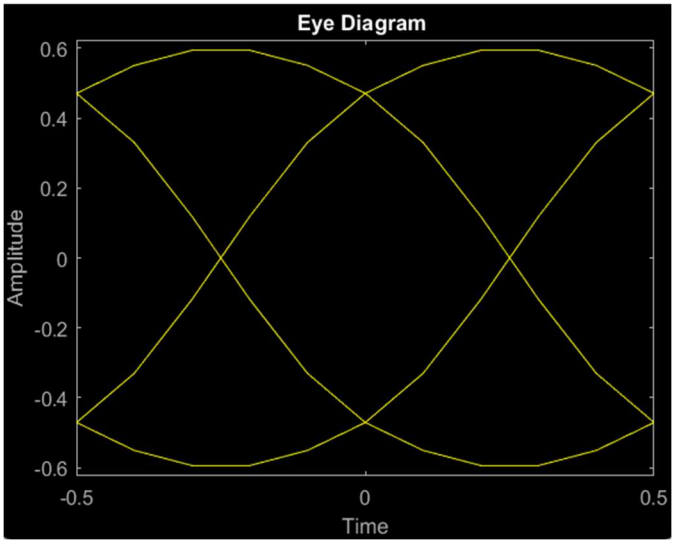

Figure 9:A_TX R_0_delay2

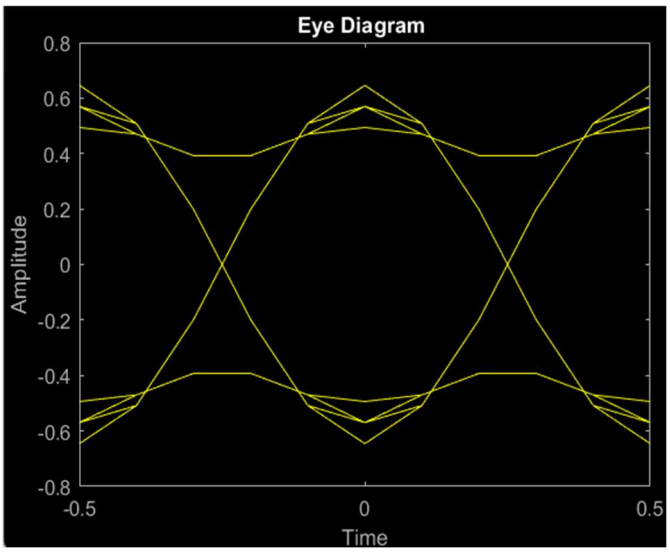

Figure 10:A_TX R_1_delay2

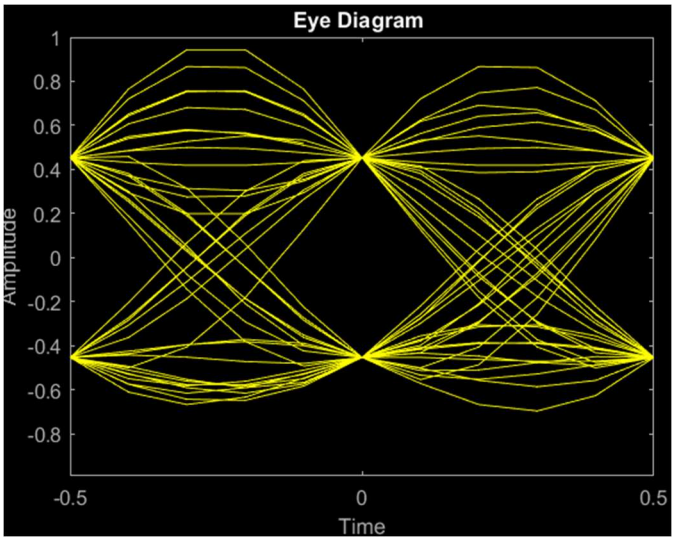

Figure 11:A_TX R_0_delay8

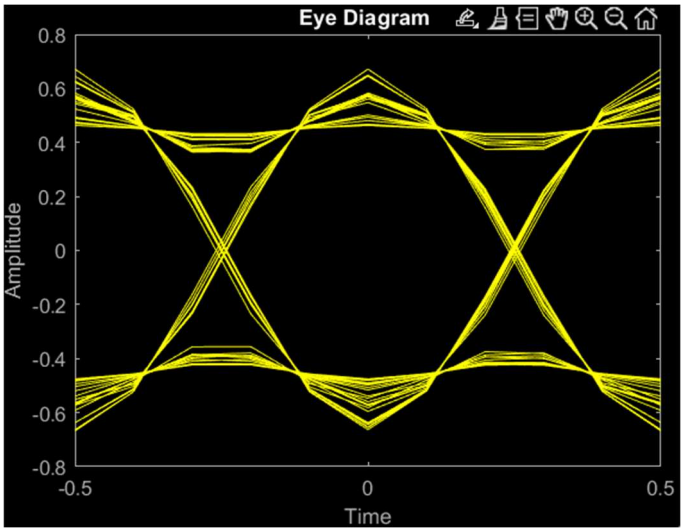

Figure 12:A_TX R_1_delay8

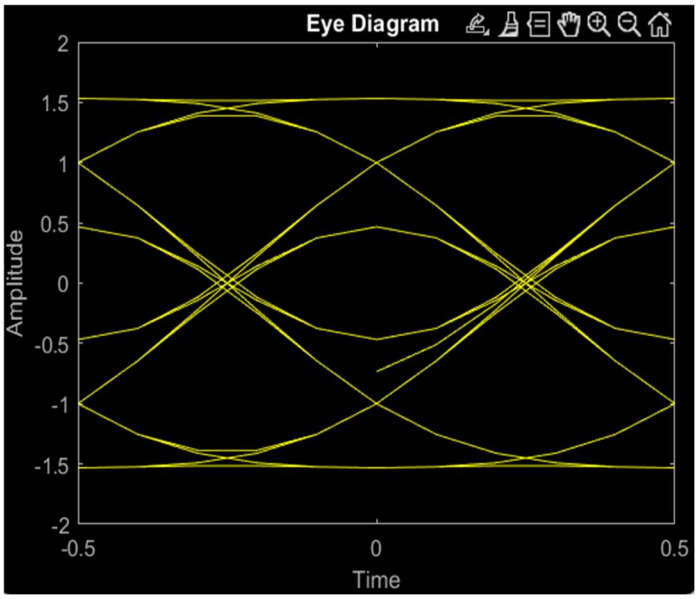

Figure 13. B_RX R_0_delay2

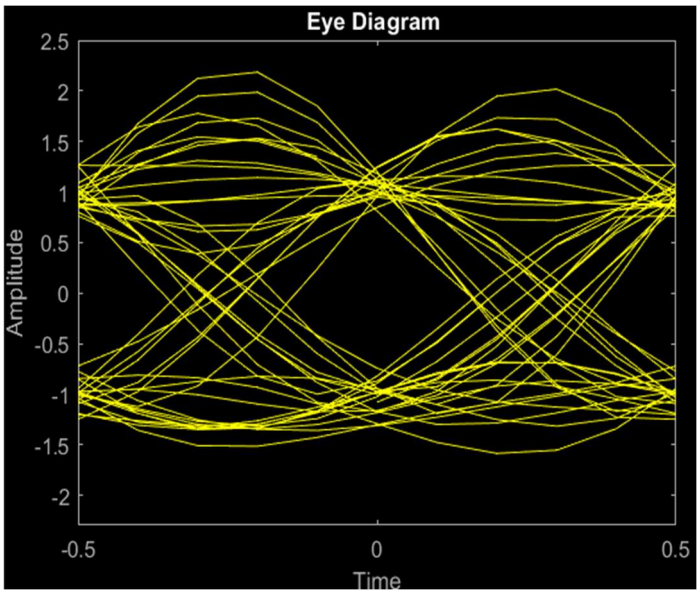

Figure 14. B_RX R_0_delay8

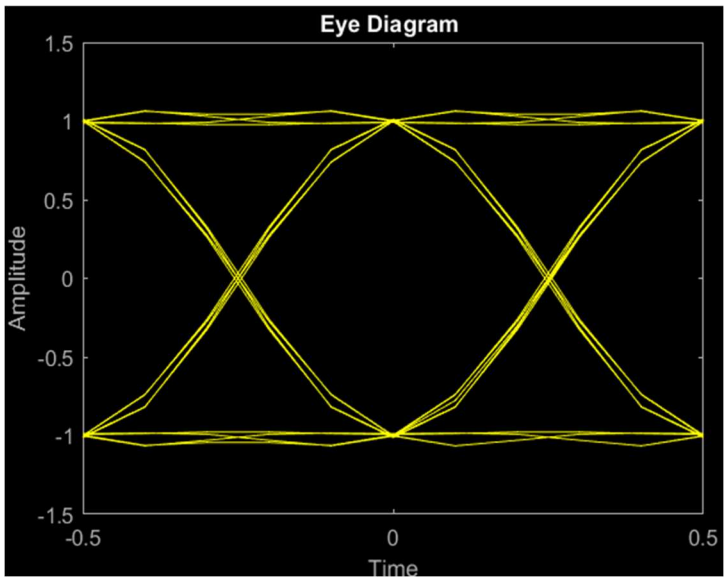

Figure 15:B_RX R_1_delay2

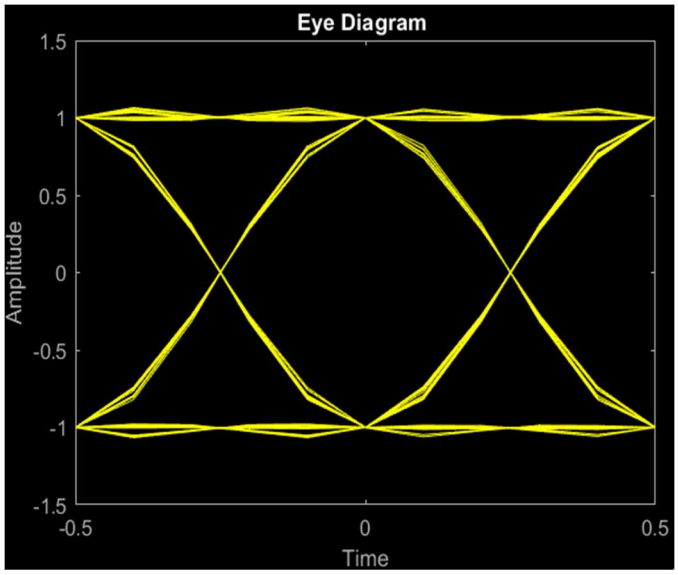

Figure 16:B_RX R_1_delay8

- The eye height shows the noise margin and how far the symbols are from each other. A larger eye height means the system can tolerate more noise.
- From our results, we observed that when the roll-off factor R is set to 1, the eye opening is wider and inter-symbol interference (ISI) is reduced.
- We also found that when the filter delay is increased to 8, the eye diagram looks more symmetrical and cleaner, which indicates better pulse shaping and improved signal quality.

Full MATLAB Code:

```
clc; clear; close all;
%% Requirmnet 1
% Parameters
T = 1;  % Symbol duration in seconds
samples_per_symbol = 5;
Ts = 0.2; % sampling duartion in seconds
Num_bits = 10;
% Define normalized pulse
p = [5 4 3 2 1]/sqrt(55);

% Generate random binary bits (0s and 1s)
bits = randi([0 1], 1, Num_bits);

% Convert to +1/-1
signal = 2*bits - 1;  % Converts 0->-1 and 1->+1

% Upsample the signal (insert 4 zeros between samples)
upsampled_signal = upsample(signal, samples_per_symbol);
% Convolve with pulse shaping function
y = conv(upsampled_signal, p, 'full');
```

```
figure;
stem(upsampled_signal, 'LineWidth', 1.5);
title("Upsmapling sequence every 200ms");
xlabel("time (ms)");
ylabel("Amp");

figure;
plot(y, 'LineWidth', 1.5);
yline(0, 'k--', 'LineWidth', 1);  % Add a dashed black line at y = 0
title("Convolution with pulse shape");
xlabel("time (ms)");
ylabel("Amp");

% Matched filter
h_matched = fliplr(p);

% Alternative Simple hold filter (all ones, normalized)
h_alt = ones(1,5)/sqrt(5);  % Simple hold filter with 5 samples, normalized

% Filter the signal
y_matched = conv(y, h_matched, 'full');
y_alt = conv(y, h_alt, 'full');

% Correlator filter
y_reshaped = reshape(y(:, 1:Num_bits * samples_per_symbol), samples_per_symbol,
Num_bits);  % size(y_reshaped) = [5, 10], 5 samples, 10 bits
y_correlator = zeros(samples_per_symbol * Num_bits, 1);  % Output size: 5 * 10 = 50

% Loop over each bit
for n = 1:Num_bits
    % Loop over each correlation output for the current bit
    for i = 1:samples_per_symbol
        sum_result = 0;
        % Compute correlation: sum p(m) at each sample * y_reshaped(m, n) for each
sample within a bit in output y
        for m = 1:i
            sum_result = sum_result + p(m) * y_reshaped(m, n);
        end
        % Store result in y_correlator (dumb summation at T) for each bit to then
start over
        y_correlator((n-1)*samples_per_symbol + i) = sum_result;
    end
end

% Sampling points
sampling_indices = (1:Num_bits) * samples_per_symbol;
% Extract matched and alt samples at symbol instants
matched_samples = y_matched(sampling_indices);
alt_samples = y_alt(sampling_indices);

% Time vector adjusting
time_vector_matched = (0:length(y_matched)-1) * Ts + 0.2; % added 0.2 filter delay to
overcome the transition from 50ms domain to 10sec domain
time_vector_alt = (0:length(y_alt)-1) * Ts + 0.2;
```

```
time_vector_corr = (0:length(y_correlator)-1) * Ts + 0.2;

% Plot: Matched Filter vs Simple Hold Filter
figure;
subplot(2,1,1);
plot(time_vector_matched , y_matched, 'b-', 'LineWidth', 1.5); hold on;
stem(sampling_indices * Ts, matched_samples, 'bo', 'LineWidth', 1);
title('Matched Filter Output');
xlabel('Time (s)');
xlim([0,10]);
ylabel('Amplitude');
legend('Matched Filter','Samples');
grid on;

subplot(2,1,2);
plot(time_vector_alt, y_alt, 'r-', 'LineWidth', 1.5); hold on;
stem(sampling_indices * Ts, alt_samples, 'ro', 'LineWidth', 1);
title('Simple Hold Filter Output');
xlabel('Time (s)');
xlim([0,10]);
ylabel('Amplitude');
legend('Hold Filter','Samples');
grid on;
set(gcf, 'Position', [100 100 800 600]);
sgtitle('Filter Outputs Comparison');

% Plot: Matched Filter vs Correlator
figure;
plot(time_vector_matched,y_matched, 'm-', 'LineWidth', 1.5); hold on;
plot(time_vector_corr,y_correlator, 'Color', '[0 0.5 0.5]', 'LineWidth', 1.5);
stem(sampling_indices * Ts, matched_samples, 'r--o', 'LineWidth', 1);
title('Matched Filter vs Correlator Output');
xlabel('Time (s)');
xlim([0,10]);
ylabel('Amplitude');
legend('Matched Filter', 'Correlator', 'Samples');
grid on;
set(gcf, 'Position', [100 100 800 600]);

%% Requirment 2
% Parameters
Num_bits = 10000;
EbN0_dB = -2:1:5;

% Generate random binary bits (0s and 1s)
bits = randi([0 1], 1, Num_bits);

% Convert to +1/-1
signal = 2*bits - 1;  % Converts 0->-1 and 1->+1

% Upsample the signal (insert 4 zeros between samples)
upsampled_signal = upsample(signal, samples_per_symbol);
% Convolve with pulse shaping function
y = conv(upsampled_signal, p, 'full');
```

```
% Adding Noise and Calculating BER
% Initialize BER arrays for matched filter, hold filters, and theoretical
BER_matched = zeros(size(EbN0_dB));
BER_hold = zeros(size(EbN0_dB));
BER_theo = zeros(size(EbN0_dB));

% Matched filter
h_matched = fliplr(p);
h_hold = 5 * ones(1,5);  % Simple hold filter with 5 samples, un-normalized
h_hold = h_hold/(5*sqrt(5)); % Simple hold filter with 5 samples, normalized

% Sampling points
sampling_indices = (1:Num_bits) * samples_per_symbol;

% Loop over Eb/N0 values
for idx = 1:length(EbN0_dB)
    EbN0 = 10^(EbN0_dB(idx)/10);  % Convert Eb/N0 fr%om dB to linear scale
    N0 = 1 / EbN0;                % Since pulse p is normalized, Eb = 1

    % Generate Gaussian noise with appropriate variance
    noise = sqrt(N0/2) * randn(size(y));  % Generate AWGN noise
    rx_signal = y + noise;                % Received signal after adding noise

    % Matched filter output
    rx_matched = conv(rx_signal, h_matched, 'full');
    matched_samples = rx_matched(sampling_indices);

    % Hold filter output
    rx_hold = conv(rx_signal, h_hold, 'full');
    hold_samples = rx_hold(sampling_indices);

    % Decision making (threshold = 0)
    matched_decisions = matched_samples > 0;
    hold_decisions = hold_samples > 0;

    % Calculate BER for matched filter
    BER_matched(idx) = sum(matched_decisions ~= bits) / Num_bits;

    % Calculate BER for hold filter
    BER_hold(idx) = sum(hold_decisions ~= bits) / Num_bits;

    % Calculate Theoretical BER value
    BER_theo(idx) = 0.5 * erfc(sqrt(EbN0));
end

%  Plotting the BER vs SNR
figure;
semilogy(EbN0_dB, BER_matched, 'bo-', 'LineWidth', 1.5); hold on;
semilogy(EbN0_dB, BER_hold, 'g-', 'LineWidth', 1.5); hold on;
semilogy(EbN0_dB, BER_theo, 'k--', 'LineWidth', 1.5); % Theoretical BER (dashed line)
title('BER with Matched & Simple Hold Filters');
xlabel('SNR in dB');
ylabel('Bit Error Rate');
legend('BER using matched filter', 'BER using hold filter', 'Theoretical BER');
grid on;
```

```
%% Requirmnet 3
% ISI && Raised cosine
data_length = 100;       % Length of the data (bits)
data = randi([0, 1], 1, data_length);
Data_sequence_maped = 2*data - 1; % maping the output to 1 or -1 %
Data_sequence_UpSample = upsample(Data_sequence_maped, samples_per_symbol);

i=1;
for R = [0, 1]
    for delay = [2, 8]
            h = rcosdesign(R, delay, samples_per_symbol ,"sqrt");
        % Apply square root raised cosine first
            A_Tx = filter(h ,1 , Data_sequence_UpSample);
        % ideal channel free noise mean H(s)=1not make any change
            B_Rx = filter(h ,1 , A_Tx); % Apply square root raised cosine first
        %The outcome was applying raised cosine to the data.
        % draw rsosine filter
            figure(i);
            i = i + 7;
            plot(h);
            title("Square Root Raised Cosine Filter (R: " + num2str(R) + ", Delay: "
+ num2str(delay) + ")");
            xlabel("Time Samples");
            ylabel("Amplitude");
            ylim([min(h)-0.01 max(h)+0.01]);
         % draw Eye diagram
             eye_fig_TX=eyediagram(A_Tx(delay * samples_per_symbol + 1:end - delay *
samples_per_symbol)', 2 * samples_per_symbol);
             set(eye_fig_TX,'Name',"A_Tx_eyediagram for R :" + R + " Delay: " +
delay);
             eye_fig_RX=eyediagram(B_Rx(delay * samples_per_symbol +1:end - delay *
samples_per_symbol)', 2 * samples_per_symbol);
             set(eye_fig_RX,'Name',"B_Rx_eyediagram for R :" + R + " Delay: " +
delay);
    end
end
```