Electrical Engineering Department,

Fourth Year - Communications & Electronics.

EE481 DIGITAL COMMUNICATIONS

Experiment 2

Handling noisy channels: Matched filters

PREPARED BY	SECTION	SEAT.NO.
Mahmoud Mohamed Kamal Ismail	7	250

➢ Contents

➢ List of Figures

1. m Files

1.1. Lab2_script.m

```
%%
% Alexandria University
% Faculty of Engineering
% Electrical and Electronic Engineering Department
%
% Course: Digital Communications Lab
%
% Lab No. 2: Handling noisy channels: Matched filters

%% Simulation parameters

fs = 1e5;                          % Sampling rate (samples per sec)
Ts = 1/fs;                         % Sampling time

N = 102400 - 1;                    % Total number of samples
t_axis = (0:N-1)*Ts;               % Time axis (the same during the entire
experiment)
f_axis = -fs/2:fs/N:fs/2-1/N;      % Frequency axis (the same during the entire
experiment)

%% Part 1-a: Generate a pulse

% Generate one square pulse with the following parameters
Energy_per_bit = 1;                % The total energy of all samples constituting
the square pulse
T_sq = 100*Ts;                     % The duration of the square pulse (an integer
number of sampling times)

x_bits = 1;
x_square =
GenerateSquarePulses(t_axis,T_sq,Energy_per_bit,fs,x_bits,'unipolar');    %
IMPLEMENT THIS: complete the 'RZ' part

%% Show time and frequency plots of the generated pulse
% DO NOT ALTER THIS PART

figure
subplot(2,1,1)

plot(t_axis,x_square,'linewidth',2)
grid on
xlim([0 T_sq*1.2])
xlabel('Time','linewidth',2)
ylabel('Square pulse','linewidth',2)

X_square = GetFreqResponse(x_square,fs);

subplot(2,1,2)
plot(f_axis,abs(X_square),'linewidth',2)
title('Student Figure','linewidth',10)
grid on
xlim([-1/T_sq 1/T_sq]*5)
xlabel('Frequency','linewidth',2)
ylabel('Frequency ressponse magnitude','linewidth',2)
subplot(2,1,1)
title('A square pulse in time and frequency domains','linewidth',10)

%% Part 1-b: AWGN channel effect

% See the effect of an AWGN channel for a particular Eb/No value
Eb_No_db = 0;        % The specified Eb/No value in dB
```

```
%%% WRITE YOUR CODE HERE
% Knowing the value of Eb/No in dB, and for the given energy per bit value
% specified above, find the corresponding value of No.
No = Energy_per_bit/db2mag(Eb_No_db);
%%%

%%% Implement the effect of the AWGN channel
y_square = AWGNChannel(x_square,No,fs);     % IMPLEMENT THIS: the output of the
AWGN channel should be given in y_square.

figure
subplot(2,1,1)
plot(t_axis,x_square,'linewidth',2)
grid on
xlim([0 T_sq*1.2])
xlabel('Time','linewidth',2)
ylabel('Square pulse','linewidth',2)

subplot(2,1,2)
plot(t_axis,y_square,'linewidth',2)
title('Student Figure','linewidth',10)
grid on
xlim([0 T_sq*1.2])
xlabel('Time','linewidth',2)
ylabel('Square pulse','linewidth',2)
subplot(2,1,1)
title('A square pulse in time and the effect of noise','linewidth',10)

%% Part 1-c: See noise effect on multiple bits

% Here, generate square pulses for the sequence of bits 1010, and notice
% how the noise can affect the shape of the pulse generated for the bit
% sequence. Store the sequence of generated pulses in the variable x_square
% and the outpuf of the AWGN channel in y_square.

x_square = zeros(size(t_axis));
y_square = zeros(size(t_axis));
%%% WRITE YOUR CODE HERE
x_bits = [1 0 1 0];
x_square =
GenerateSquarePulses(t_axis,T_sq,Energy_per_bit,fs,x_bits,'unipolar');
y_square = AWGNChannel(x_square,No,fs);
%%%

figure
subplot(2,1,1)
plot(t_axis,x_square,'linewidth',2)
grid on
xlim([0 T_sq*4.2])
xlabel('Time','linewidth',2)
ylabel('Square pulse','linewidth',2)

subplot(2,1,2)
plot(t_axis,y_square,'linewidth',2)
title('Student Figure','linewidth',10)
grid on
xlim([0 T_sq*4.2])
xlabel('Time','linewidth',2)
ylabel('Square pulse','linewidth',2)
subplot(2,1,1)
title('A series of square pulses in time and the effect of
noise','linewidth',10)
```

```
%% Part 2-a: Design a matched filter for unipolar encoding
% Here, you will implement the operation of a matched filter for the square
% pulse. In this section you will visualize the procedure of the matched
% filter one a sequence of 1 bit only, just to check if you have
% implemented the code correctly and to better understand the operation of
% the matched filter.

x_bits = [1];
x_pulse_shaped =
GenerateSquarePulses(t_axis,T_sq,Energy_per_bit,fs,x_bits,'unipolar');
[rec_bits, ht, z_square] =
MatchedFilter(T_sq,Energy_per_bit,fs,x_pulse_shaped,'unipolar'); % IMPLEMENT
THIS: implement the operation of the matched filter. You don't have to
implement the decision part of the matched filter

figure
subplot(3,1,1)
plot(t_axis,x_pulse_shaped,'linewidth',2)
grid on
xlim([0 T_sq*2.2])
xlabel('Time','linewidth',2)
ylabel('Square pulse','linewidth',2)

subplot(3,1,2)
plot(t_axis,[ht zeros(1,length(x_square) - length(ht))],'linewidth',2)
title('Student Figure','linewidth',10)
grid on
xlim([0 T_sq*2.2])
xlabel('Time','linewidth',2)
ylabel('h(k)','linewidth',2)

subplot(3,1,3)
plot(t_axis,z_square(1:length(t_axis)),'linewidth',2)
grid on
xlim([0 T_sq*2.2])
xlabel('Time','linewidth',2)
ylabel('MK output','linewidth',2)
subplot(3,1,1)
title('The Matched Filter operation','linewidth',10)

%% Part 2-b: The decision operation of the MF receiver

% Complete the function MatchedFilter to implement the decision part of the
% MF operation. It should be based on the observation that you made from
% Part 2-a.

%% Part 2-c: Check the BER of the matched filter for unipolar encoding
% In this section you will see the decoding performance, i.e., the BER, of
% the MF you created. The code repeats the operation in the previous
% section but using a long sequence of bits.

N_bits = 2084;

x_bits = 0;
BER_uni = 0;

%%% WRITE YOUR CODE HERE
% Generate a random sequence of N_bits bits and store them in the variable
% x_bits. Then apply the square pulse shape you these bits to obtain the
% sampled sequence, apply AWGN to this sample sequence and then use the MF
% you implemented to decode those bits. Finally, compute the BER of this
% MF and store it in the variable BER_EZ.
%
% Hint: reuse the code from the previous cell. Your code can be as short as
```

```
% 5 lines. You can reuse the function ComputeBER from Experiment 1.
x_bits = randi([0,1],[1,N_bits]);
x_square = GenerateSquarePulses(t_axis, T_sq, Energy_per_bit, fs, x_bits,
'unipolar');
y_square = AWGNChannel(x_square, No, fs);
[rec_bits, ht, z_square] = MatchedFilter(T_sq, Energy_per_bit, fs, y_square,
'unipolar');
BER_uni = sum(bitxor(x_bits,rec_bits)) / N_bits;
%%%

%% Part 3-a: Generate a pulse with bipolar encoding

% Generate one square pulse with the following parameters
Energy_per_bit = 1;             % The total energy of all samples constituting
the square pulse
T_sq = 100*Ts;                  % The duration of the square pulse (an integer
number of sampling times)

x_bits = [1 0];
x_square =
GenerateSquarePulses(t_axis,T_sq,Energy_per_bit,fs,x_bits,'bipolar');
%IMPLEMENT THIS: complete the 'NRZ' part

% DO NOT ALTER THE FOLLOWING PART
figure

subplot(2,1,1)
plot(t_axis,x_square,'linewidth',2)
grid on
xlim([0 T_sq*2.2])
xlabel('Time','linewidth',2)
ylabel('Square pulse','linewidth',2)

X_square = GetFreqResponse(x_square,fs);

subplot(2,1,2)
plot(f_axis,abs(X_square),'linewidth',2)
title('Student Figure','linewidth',10)
grid on
xlim([-1/T_sq 1/T_sq]*5)
xlabel('Frequency','linewidth',2)
ylabel('Frequency ressponse magnitude','linewidth',2)
subplot(2,1,1)
title('A square pulse in time and frequency domains','linewidth',10)

%% Part 3-b: Design a matched filter for bipolar encoding
% Here, you will implement the operation of a matched filter for the square
% pulse with NRZ format. This is very similar to Part X. You will
% visualize the procedure of the matched filter on a sequence of 1 bit to
% check if you have implemented the code correctly and to better
% understand the operation of the matched filter.

x_bits = [1];
x_pulse_shaped =
GenerateSquarePulses(t_axis,T_sq,Energy_per_bit,fs,x_bits,'bipolar');
[rec_bits, ht, z_square] =
MatchedFilter(T_sq,Energy_per_bit,fs,x_pulse_shaped,'bipolar'); % IMPLEMENT
THIS: implement the operation of the matched filter for bipolar encoding. You
don't have to implement the decision part of the matched filter

figure
subplot(3,1,1)
plot(t_axis,x_pulse_shaped,'linewidth',2)
title('The Matched Filter operation','linewidth',10)
```

```
grid on
xlim([0 T_sq*2.2])
xlabel('Time','linewidth',2)
ylabel('Square pulse','linewidth',2)

subplot(3,1,2)
plot(t_axis,[ht zeros(1,length(x_square) - length(ht))],'linewidth',2)
title('Student Figure','linewidth',10)
grid on
xlim([0 T_sq*2.2])
xlabel('Time','linewidth',2)
ylabel('h(k)','linewidth',2)

subplot(3,1,3)
plot(t_axis,z_square(1:length(t_axis)),'linewidth',2)
grid on
xlim([0 T_sq*2.2])
xlabel('Time','linewidth',2)
ylabel('MF output','linewidth',2)

%% Part 3-c: The decision operation of the MF receiver

% Complete the function MatchedFilter to implement the decision part of the
% MF operation. It should be based on the observation that you made from
% Part 3-b.

%% Part 3-d: Check the BER of the matched filter for bipolar
% In this section you will see the decoding performance, i.e., the BER, of
% the MF you created for bipolar encoding. The code repeats the operation
% in the previous section but using a long sequence of bits.

N_bits = 2084;

x_bits = 0;
BER_bi = 0;

%%% WRITE YOUR CODE HERE
% Generate a random sequence of N_bits bits and store them in the variable
% x_bits. Then apply the square pulse shape you these bits to obtain the
% sampled sequence, apply AWGN to this sample sequence and then use the MF
% you implemented to decode those bits. Finally, compute the BER of this
% MF and store it in the variable BER_EZ.
%
% Hint: reuse the code from the previous cell. Your code can be as short as
% 5 lines. You can reuse the function ComputeBER from Experiment 1.
x_bits = randi([0,1],[1,N_bits]);
x_square = GenerateSquarePulses(t_axis, T_sq, Energy_per_bit, fs, x_bits,
'bipolar');
y_square = AWGNChannel(x_square, No, fs);
[rec_bits, ht, z_square] = MatchedFilter(T_sq, Energy_per_bit, fs, y_square,
'bipolar');
BER_bi = sum(bitxor(x_bits,rec_bits)) / N_bits;
%%%

%% THe BER performance of bipolar and unipolar MF receivers versus EB/No
% Here, we will check the performance of the unipolar and bipoloar MFs to
% see which format is better in terms of square pulse shaping in the
% presence of AWGN. The goal here is to repeat the BER computation
% performed above for both formats, but for different values of Eb/No. You
% need to compute the BER for unipolar and bipolar encoding for each value
% of Eb/No in the vector Eb_No_dB_vector provided below. Store the values
% of BER you obtained in the vectors BER_uni and BER_bi.

N_bits = 10000;
```

```matlab
Eb_No_dB_vector = -15:0;

BER_bi = zeros(size(Eb_No_dB_vector));
BER_uni = zeros(size(Eb_No_dB_vector));

%%% WRITE YOUR CODE HERE
x_bits = randi([0,1],[1,N_bits]);

for i = 1:1:length(Eb_No_dB_vector)
  No = Energy_per_bit/db2mag(Eb_No_dB_vector(i));
  x_square = GenerateSquarePulses(t_axis, T_sq, Energy_per_bit, fs, x_bits,
'unipolar');
  y_square = AWGNChannel(x_square, No, fs);
  [rec_bits, ht, z_square,] = MatchedFilter(T_sq, Energy_per_bit, fs, y_square,
'unipolar');
  BER_uni(i) = sum(bitxor(x_bits, rec_bits)) / N_bits;

  x_square = GenerateSquarePulses(t_axis, T_sq, Energy_per_bit, fs, x_bits,
'bipolar');
  y_square = AWGNChannel(x_square, No, fs);
  [rec_bits, ht, z_square] = MatchedFilter(T_sq, Energy_per_bit, fs, y_square,
'bipolar');
  BER_bi(i) = sum(bitxor(x_bits, rec_bits)) / N_bits;
end
%%%

figure
semilogy(Eb_No_dB_vector,BER_bi,'-xk','linewidth',2)
hold on
semilogy(Eb_No_dB_vector,BER_uni,'-ob','linewidth',2)
legend('Bipolar encoding','Unipolar encoding','linewidth',2)
xlabel('Eb/No','linewidth',2)
ylabel('BER','linewidth',2)
```

1.2. GenerateSquarePulses.m

```matlab
function x_square = GenerateSquarePulses(t_axis,T_sq,E_bit,fs,x_bits,type)
%
% Inputs:
%   t_axis:     Time axis
%   T_sq_dur:   Duration of the square pulse in seconds
%   E_bit:      Total energy in all samples of one square pulse
%   fs:         Sampling frequency
%   x_bits:     Sequence of input bits (if not available, then it is equal
%               to 1)
%   type:       Type of bit coding, 'RZ' or 'NRZ' (default is 'RZ')
% Outputs:
%   x_square:   The sequence of samples corresponding to the pulse shaping
%               of the input bits using a square pulse shape
%
% This function takes an input a sequence of bits, x_bits. It then
% generates a sequence of samples in x_square, corresponding to the pulse
% shaping of these bits using square pulses. The parameters of the square
% pulse used for pulse shaping are given in the inputs of the function.
%
% Notes:
%       If x_bits is not specified, it is assumed to be equal to 1
%       bit.
%       If type is not specified, it is assumed to be 'RZ'.
%       x_square is always equal in dimention to t_axis.

if nargin < 5
    x_bits = 1;
    type = 'unipolar';
```

```
end

if nargin < 6
    type = 'unipolar';
end

Ts = 1/fs;
N = length(t_axis);

%%% Generate one square pulse
N_sq = round(T_sq/Ts);
one_square = zeros(1,N_sq);

%%% WRITE YOUR CODE HERE
% Here you should create exactly one square pulse with the specified
% parameters. This square pulse should be stored in the array called
% one_square. The dimensions of this array should be 1 x N_sq_pos. The
% length of the square pulse you generate should be equal to N_sq_pos,
% i.e., the variable one_square should not change dimensions after you
% generate the pulse. Keep in mind that you have to set the energy of the
% square pulse to be equal to E_bit.

one_square(1:N_sq) = ones(1,N_sq);

% YOUR CODE ENDS HERE
%%%

%%% Generate one square pulse for each bit in the variable x_bit
% Here, you should be able to use the pulse you generated in one_square to
% create the final array x_square. This final array should consist of the
% square pulses corresponding to each input bit. Note that the dimensions
% of the x_square should always be equal to the dimensions of t_axis.

switch type
    case ('bipolar')
        %%% This case is for NRZ
        %%% WRITE YOUR CODE HERE
        x_square = zeros(1,N);
        A = sqrt(E_bit/N_sq);
        start = 1;
        for i = 1:1:length(x_bits)
            if     (x_bits(i) == 0)
                x_square(start:(N_sq*i)) = -1*A*one_square(1);
            elseif (x_bits(i) == 1)
                x_square(start:(N_sq*i)) = 1*A*one_square(1);
            end
            start = (N_sq*i)+1;
        end
        % YOUR CODE ENDS HERE

    case ('unipolar')
        %%% This case is for RZ
        %%% WRITE YOUR CODE HERE
        x_square = zeros(1,N);
        A = sqrt((2*E_bit)/N_sq);
        start = 1;
        for i = 1:1:length(x_bits)
            x_square(start:(N_sq*i)) = x_bits(i)*A*one_square(1);
            start = (N_sq*i)+1;
        end
        % YOUR CODE ENDS HERE
end
```

1.3. GetFreqResponse.m

```
function [X, f_axis] = GetFreqResponse(x,fs)
%
% Inputs:
%   x:      Signal in time domain
%   fs:     Sampling frequency
% Outputs:
%   X:      Magnitude of the signal in frequency domain
%   f_axis: Frequency axis
%
% This function generates the magnitude of the signal in the frequency
% domain after performing FFT on the signal in time domain.

N = length(x);
X = fftshift(fft(x));

f_axis = -fs/2:fs/N:fs/2-1/N;
```

1.4. AWGNChannel.m

```
function y = AWGNChannel(x,No,fs)
%
% Inputs:
%   x:      Signal in time domain
%   No:     2 times the noise variance
% Outputs:
%   y:      The output of the AWGN channel for the input x
%
% This function generates the effect of an AWGN channel with noise variance
% No/2 on the input signal x.

y = zeros(size(x));
%%% WRITE YOUR CODE HERE
% Your code should generate the ouptut y which is a noisy version of the
% input x, corrupted by an AWGN noise with variance No/2. Hint: use randn
% as a function for generating Gaussian noise with unit variance.
Noise = sqrt(No/2).*randn(size(x));
y = x + Noise;
%y = awgn(x,0); %snr_db = 0
%%%
```

1.5. MatchedFilter.m

```
function [rec_bits, ht, z_signal] = MatchedFilter(T_sq,E_bit,fs,y_signal,type)
%
% Inputs:
%   T_sq_dur:   Duration of the square pulse in seconds
%   E_bit:      Total energy in all samples of one square pulse
%   fs:         Sampling frequency
%   y_square:   Sequence of samples which correspond to the square pulses
%               of the input bits to be detected
%   type:       Type of bit coding, 'unipolar' or 'bipolar' (default is
%               'unipolar')
% Outputs:
%   rec_bits:   The sequence of bits decoded by the matched filter
%               corresponding to the input y_square
%   ht:         The impulse response of the matched filter
%   z_square:   The output of the convolution operation between the input
%               sequence y_square and the impulse response h_t
%
% This function implements the matched fiilter receiver for a square pulse
% shape. The function takes as input y_square, which contains the samples
```

```
% corresponding to the sequence of square pulse shapes of the input bits.
% The operation of the function is the matched filter operation:
%
%         1- It generates ht, the impulse response of the appropriate
%         matched filter
%         2- It performs a convolution operation between ht and the input
%         sequence y_square. The output of this operation is stored in a
%         variable called z_square
%         3- From the variable z_square, the function makes a decision on
%         the value of each input bit.
%
% Notes and hints:
%       - The dimensions of ht should be equal to N_sq, which is the length
%       of the square pulse used in the generation of the input sequence.
%       - The dimensions of z_square should be equal to the expected length
%       of the output of a convolution operation between two input
%       sequences: y_square and ht.
%       - From z_square (which should be a long vector, longer than the
%       expected number of input bits), the function should decide the
%       value of each input bit and store those in the variable rec_bits.
%       Note therefore that rec_bits should have a smaller length than
%       z_square.
%       - If type is not specified, it is assumed to be 'unipolar'.

if nargin < 5
    type = 'unipolar';
end

Ts = 1/fs;

N_sq = round(T_sq/Ts);          % Length of the square pulse used for pulse
shaping
N_y_signal = length(y_signal);  % Length of the input sequence

N_bits = 0;
%%% WRITE YOUR CODE HERE
% Knowing the length of the square pulse and the length of the input
% sequence of pulses corresponding to the input bits, compute the number of
% input bits and store it in N_bits.
N_bits = (N_y_signal+1) / N_sq;
%%%

ht = [];
z_signal = [];
rec_bits = [];
switch type
    case ('bipolar')
        %%% WRITE YOUR CODE HERE
        % Compute the MF impulse response ht, and the MF output signal
        % z_signal for the bipolar encoding case
        ht(1:N_sq) = 2*sqrt(E_bit/N_sq);
        z_signal   = conv(y_signal,ht);
        %%%

        %%% WRITE YOUR CODE HERE
        % Implement the decision part of the receiver with bipolar encoding
        % which uses z_signal to decide the values of the input bits
        for i = N_sq:N_sq:length(z_signal)
            if (z_signal(i) > 0)
                rec_bits = [rec_bits 1];
            else
                rec_bits = [rec_bits 0];
            end
        end
```

```
        %%%

    case ('unipolar')

        %%% WRITE YOUR CODE HERE
        % Part 2-a: Compute the MF impulse response ht, and the MF output
        % signal z_signal for the unipolar encoding case
        ht(1:N_sq) = sqrt((2*E_bit)/N_sq);
        z_signal   = conv(y_signal,ht);
        %%%

        %%% WRITE YOUR CODE HERE
        % Pat 2-b: Implement the decision part of the receiver with
        % unipolar encoding which uses z_signal to decide the values of the
        % input bits
        for i = N_sq:N_sq:length(z_signal)
            if (z_signal(i) > 1)
                rec_bits = [rec_bits 1];
            else
                rec_bits = [rec_bits 0];
            end
        end
        %%%

end
```

2. Results

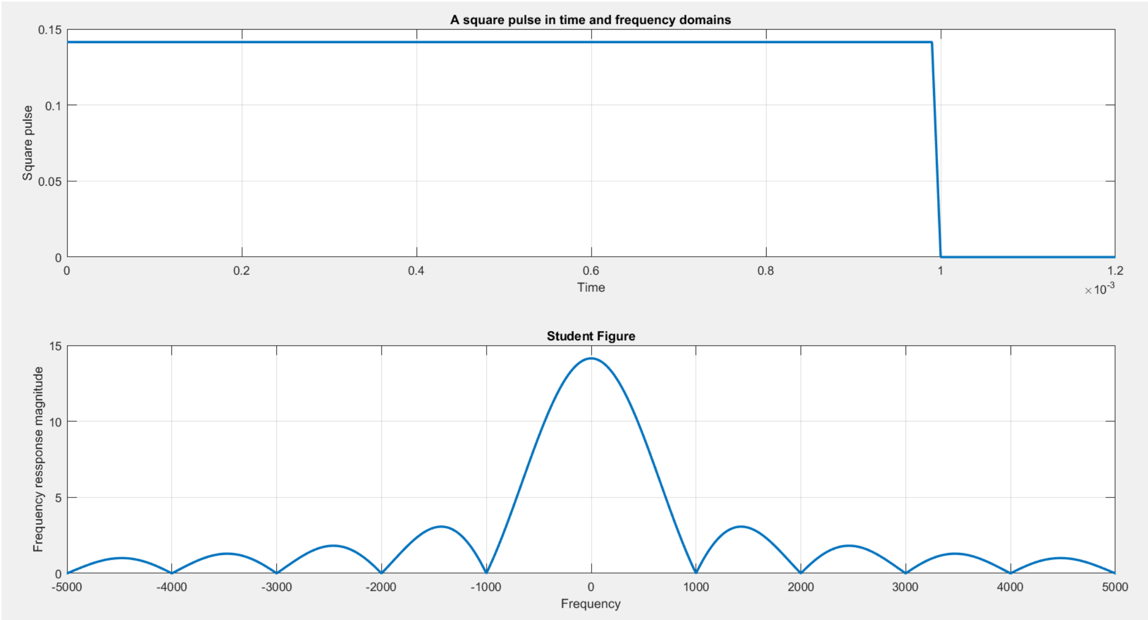

Figure 1: Part 1-a: Generate a square wave pulse (Unipolar square pulse shaping).

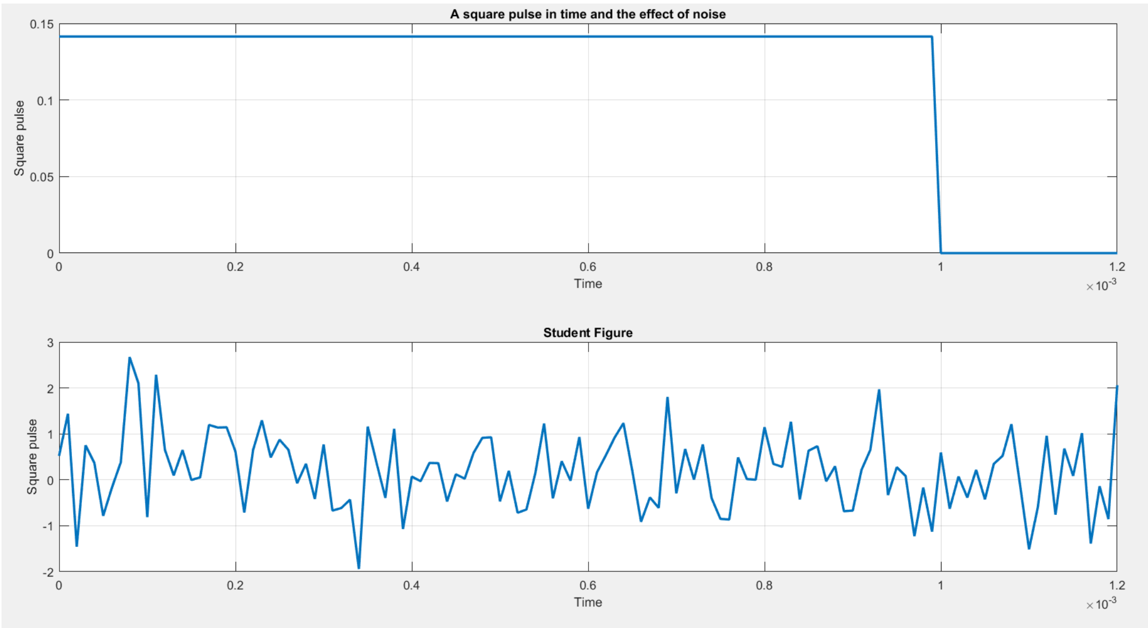

Figure 2: Part 1-b: See the effect of the noise on the signal (Unipolar square pulse shaping).

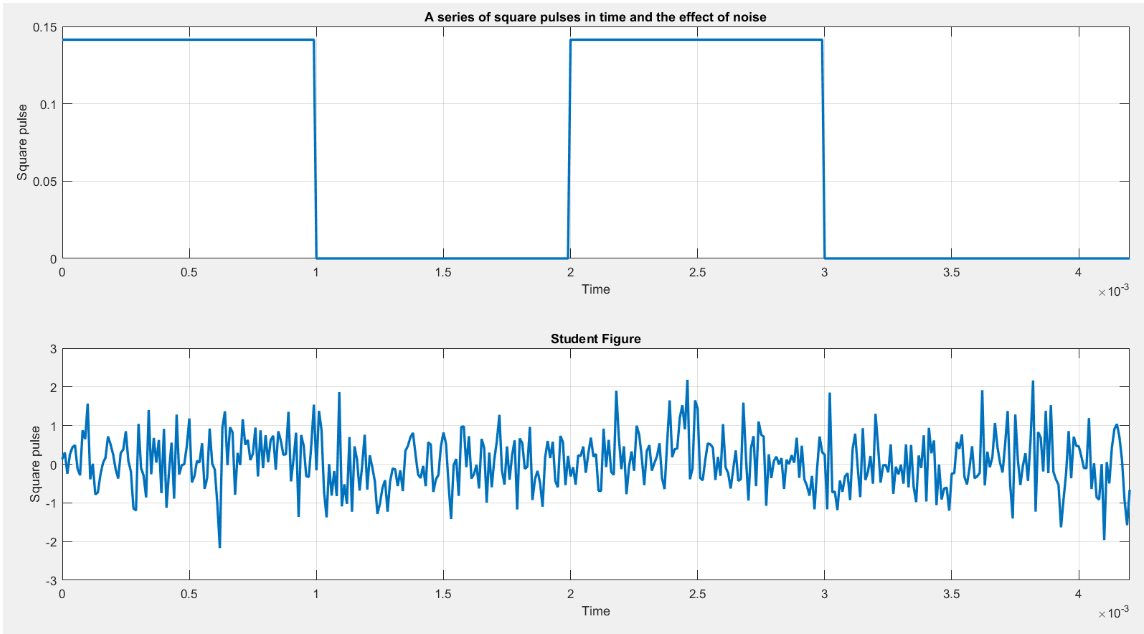

Figure 3: Part 1-c: See the effect of the noise on a sequence of square pulses (Unipolar square pulse shaping).

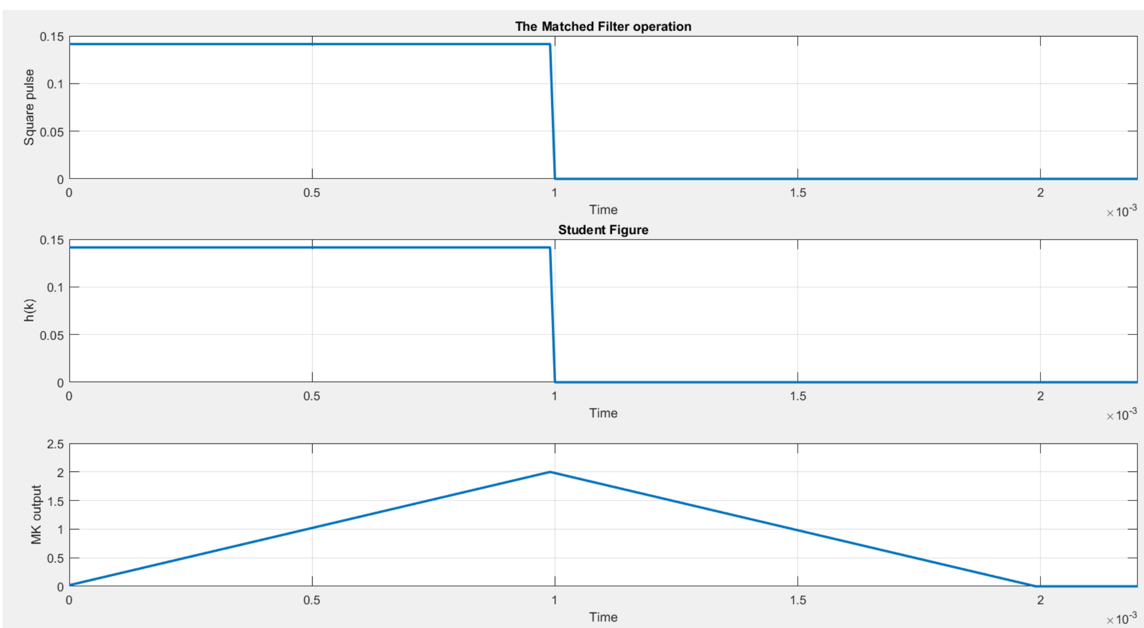

Figure 4: Part 2-b: Design the decision-making module for the matched filter of unipolar encoding.

Figure 5: Part 2-c: Check the BER performance of the MF (Unipolar square pulse shaping).

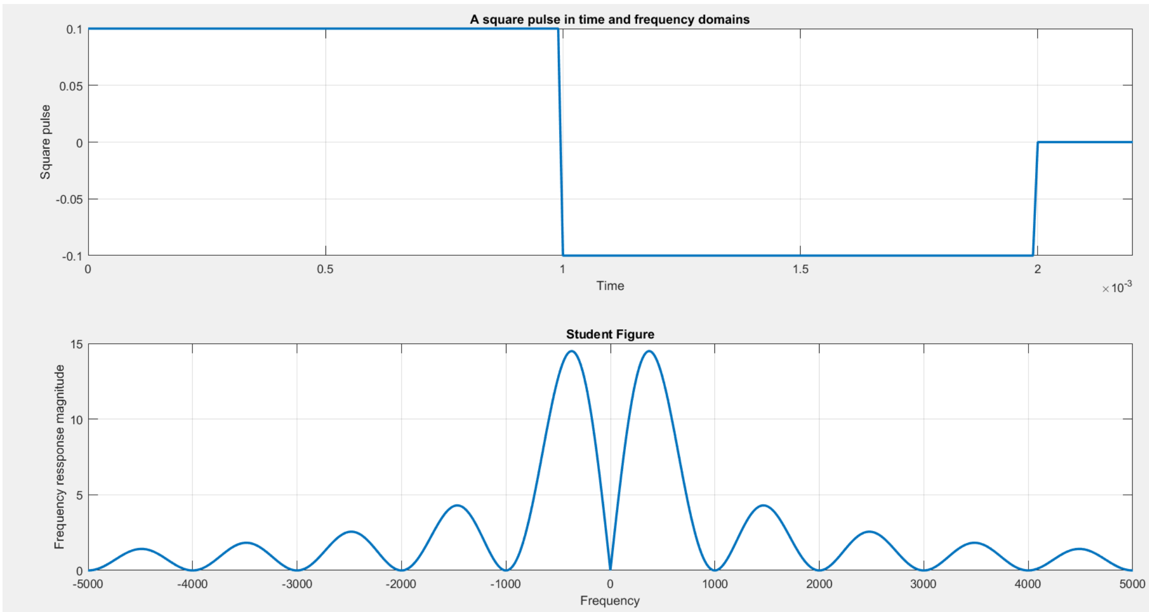

Figure 6: Part 3-a: Generate a square wave pulse (bipolar square pulse shaping).

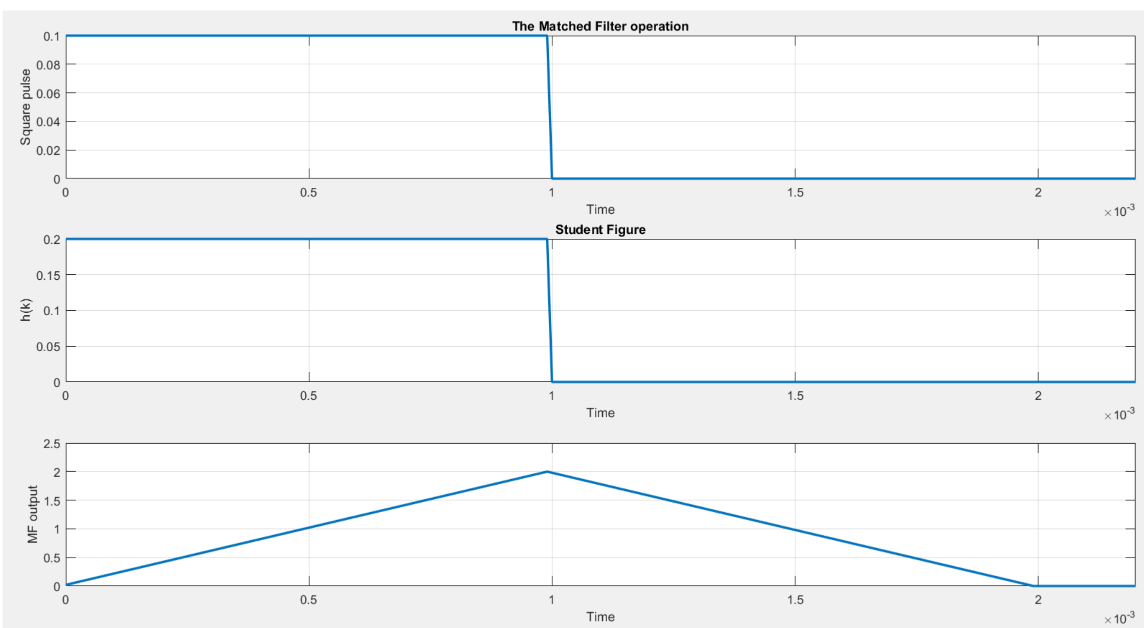

Figure 7: Part 3-b: Design the decision-making module for the matched filter of bipolar encoding.

Figure 8: Part 3-c: Check the BER performance of the MF (bipolar square pulse shaping).

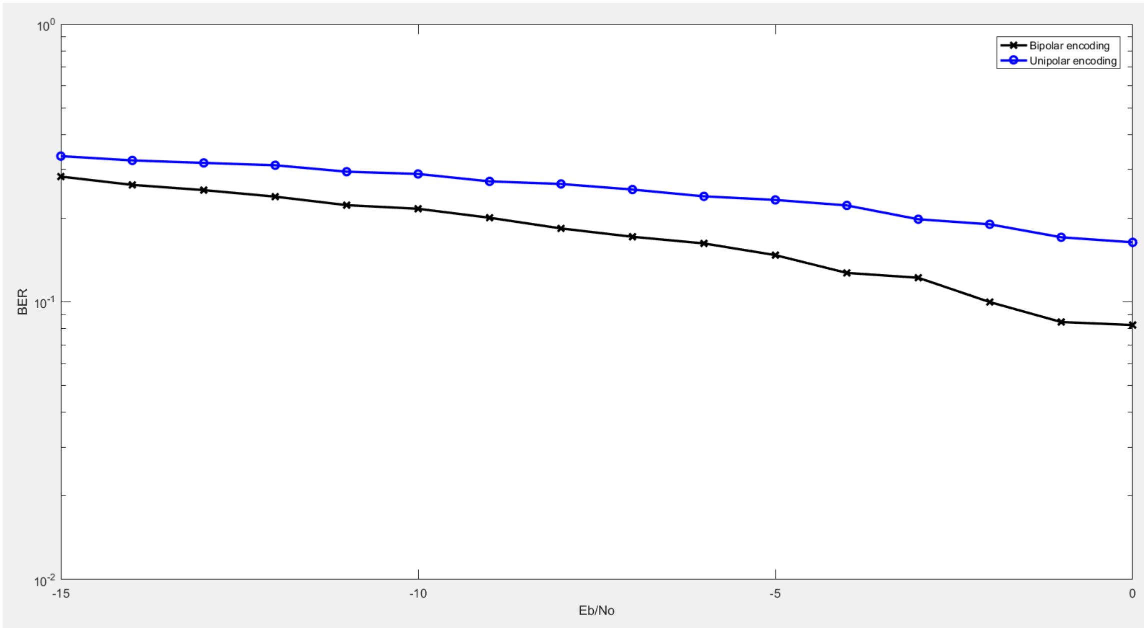

Figure 9: PART 4.