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

% Name: Rania Hamada Mohammed
% ID: 79

%% 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 / (10 .^ (Eb_No_db/10));

%%%

%%% 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 = ComputeBER(x_bits, rec_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 = ComputeBER(x_bits, rec_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;
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 unipolar
x_square_uni = GenerateSquarePulses(t_axis,T_sq,Energy_per_bit,fs,x_bits,'unipolar');
for i=1:length(Eb_No_dB_vector)
    No = Energy_per_bit ./ (10 .^ (Eb_No_dB_vector(i)/10));
    y_square_uni = AWGNChannel(x_square_uni,No,fs);
    [rec_bits, ht, z_square] = MatchedFilter(T_sq,Energy_per_bit,fs,y_square_uni,'unipolar');
    BER_uni(i) = ComputeBER(x_bits, rec_bits);
end
```

```
% for bipolar
x_square_bi = GenerateSquarePulses(t_axis,T_sq,Energy_per_bit,fs,x_bits,'bipolar');
for i=1:length(Eb_No_dB_vector)
    No = Energy_per_bit ./ (10 .^ (Eb_No_dB_vector(i)/10));
    y_square_bi = AWGNChannel(x_square_bi,No,fs);
    [rec_bits, ht, z_square] = MatchedFilter(T_sq,Energy_per_bit,fs,y_square_bi,'bipolar');
    BER_bi(i) = ComputeBER(x_bits, rec_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')
xlabel('Eb/No','linewidth',2)
ylabel('BER','linewidth',2)
```