Digital Communication

Inter-Symbol interference

Mohamed Ayman Sallam - 6144
Yosra Emad – 6206
Youssef Mohamed Sabry - 6239
Youssef Hassan - 6259
Nour Eldine Hazem - 6261
Abd El Rahman Mohamed - 6262
Omar Alaa – 6294
Hazem Hashem - 6657
Anas Ahmed - 6659

Part 1: Inter-Symbol Interference due to band-limited channels

Code

```
fs = 1e6; % Sampling frequency
N = 1e5;  % Total number of Samples
Ts = 1/fs; % Sampling Time
t = (0:N-1)*Ts; % Main Time Axis
f_axis = -fs/2:fs/N:fs/2-1/N; % Generating f_axis for the pulse
TIME_AXIS_SIZE = length(t); % The length of the time axis


% Channel bandwidth frequency
B = 100 * 10^3;

%-----------------------------------------------------------------------------
% Graph Specs
f_limit = fs * 0.55;
t_limit = (2/B)*1.4;


%-----------------------------------------------------------------------------
-
% Generate Square pulse in Time Domain

N_square_pulse = (2/B) * fs; % Number of samples in the square pulse
first_square_pulse = ones(1, N_square_pulse); % Generate Train of ones

% Put the square signal on the main time axis
first_square_pulse = [first_square_pulse zeros(1, N - N_square_pulse)];

% Square pulse in the freq domain
square_pulse_freq = 1/fs* fftshift(fft(first_square_pulse));


figure
subplot(2,1,1)

% Plot the pulse in time domain
plot(t,first_square_pulse)
grid on
xlim([0 t_limit * 5])
xlabel('Time (s)')
ylabel('Amplitude')
title('Square Signal Pulse (Time Domain)')

% Plot the pulse in freq domain
subplot(2,1,2)
plot(f_axis, abs(square_pulse_freq))
grid on
xlim([-f_limit f_limit])
xlabel('Frequency (Hz)')
```

```
ylabel('Amplitude')
title('Square Signal Pulse (Freq Domain)')

%-----------------------------------------------------------------------------
-
% Generate The Band Limited Channel

% Generate the filter in freq domain
bl_channel_freq = rectpuls(f_axis , 2*B);

% Get the channel in the time domain
bl_channel = real(1/fs*ifft(ifftshift(bl_channel_freq)));

t_sinc_bl_channel = (-N/2:(N-1)/2)*Ts;

% TODO: FIX THE TIME DOMAIN CHANNEL

figure

% Plot the channel in the time domain
subplot(2,1,1)
plot(t_sinc_bl_channel,bl_channel)
grid on
title('Band Limited Channel (Time Domain)')
xlim([-t_limit * 5 t_limit * 5])
xlabel('Time (s)')
ylabel('Amplitude')

% Plot the channel in the freq domain
subplot(2,1,2)
plot(f_axis, abs(bl_channel_freq))
title('Band Limited Channel (Frequency Domain)')
xlim([-f_limit f_limit])
xlabel('Frequency (Hz)')
ylabel('Amplitude')

%-----------------------------------------------------------------------------
-
% Filter the signal by passing through the channel


% Convultion of the square signal by the band limited channel (filtering)
filtered_square_pulse = conv(first_square_pulse, bl_channel);

% Trimming the function to fit the time axis
filtered_square_pulse = filtered_square_pulse(1:TIME_AXIS_SIZE);

% Generate frequency domain of the filtered signal
filtered_square_pulse_freq = 1/fs*fftshift(fft(filtered_square_pulse));


figure;
subplot(2,1,1)
plot(t, filtered_square_pulse)
```

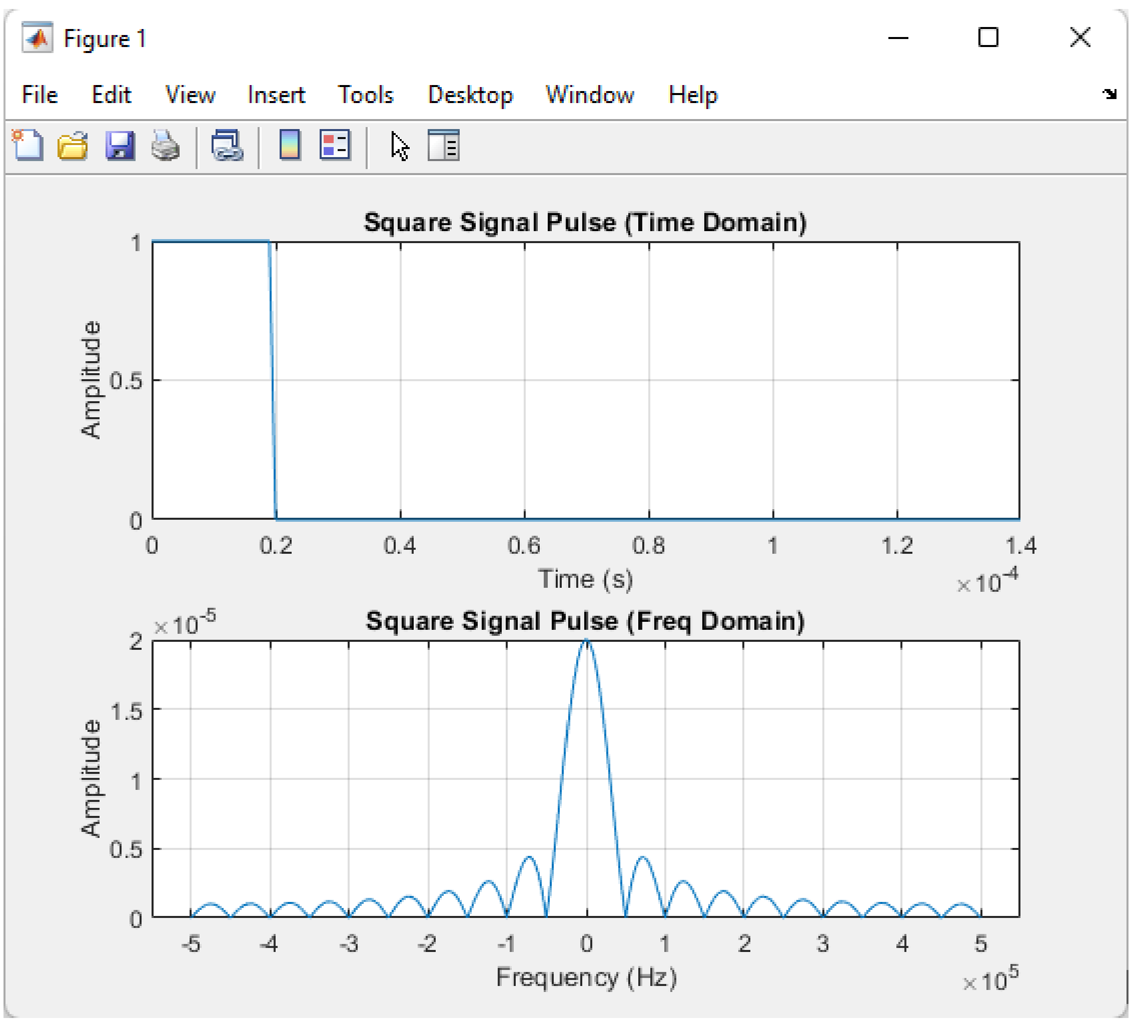
```
title('Filtered Square Pulse (Time Domain)')
xlim([0 t_limit * 5])
xlabel('Time (s)')
ylabel('Amplitude')


% Plot the channel in the freq domain
subplot(2,1,2)
plot(f_axis, abs(filtered_square_pulse_freq))
title('Band Limited Channel (Frequency Domain)')
xlim([-f_limit * .7 f_limit* .7])
ylabel('Amplitude')
xlabel('Frequency (Hz)')

%----------------------------------------------------------------------------
--------
% Generate the two consectuive pulses

second_square_pulse = ones(1, N_square_pulse); % Generate Train of ones
second_square_pulse = [zeros(1, N_square_pulse) second_square_pulse zeros(1,
N - 2* N_square_pulse)];

figure;
plot(t, first_square_pulse, 'r')
hold on
plot(t, second_square_pulse, 'b')
title('Filtered Square Pulse (Time Domain)')
xlim([0 t_limit * 5])
xlabel('Time (s)')
ylabel('Amplitude')

%----------------------------------------------------------------------------
--------
% Convlute the two cons pulses with the band limited channel

% Repeat what we have done for the first pulse
filtered_second_square_pulse = conv(second_square_pulse, bl_channel);
filtered_second_square_pulse =
filtered_second_square_pulse(1:TIME_AXIS_SIZE);
filtered_second_quare_pulse_freq =
1/fs*fftshift(fft(filtered_second_square_pulse));

figure;

% Plot the two pulses in time domain
subplot(2,1,1)
plot(t, filtered_square_pulse, 'r')
hold on
plot(t, filtered_second_square_pulse, 'b')
title('Filtered Two Pulses (Time Domain)')
xlim([0 t_limit * 5])
xlabel('Time (s)')
ylabel('Amplitude')

% Plot the two pulses in freq domain
```

```
subplot(2,1,2)
plot(f_axis, abs(filtered_square_pulse_freq), 'r')
hold on
plot(f_axis, abs(filtered_second_quare_pulse_freq), 'b')
title('Filtered Two Pulses (Freq Domain))')
xlim([-f_limit * .7 f_limit* .7])
ylabel('Amplitude')
xlabel('Frequency (Hz)')

% ---------------------------------------------------------------------------
----------
% Investigating a solution to the ISI
% Raised Cosine solution (Ideal Solution)

beta = 0.5;

t_rcf = (-(N-1)/2: ((N-1)/2))*Ts;
raised_cosine_filter = (sinc(2*t_rcf*(B/2)).*
cos(2*pi*beta*(B/2)*t_rcf))./(1-(4*beta*t_rcf*(B/2)).^2);
figure
plot(t, raised_cosine_filter)

% Generate Signal
raised_cosine_signal = zeros(1, N) + raised_cosine_filter +
circshift(raised_cosine_filter' , N_square_pulse)';
raised_cosine_signal = raised_cosine_signal(1:TIME_AXIS_SIZE);
raised_cosine_signal_freq = 1/fs*fftshift(fft(raised_cosine_signal));

% Pass Signal through channel
filtered_rfc = conv(raised_cosine_signal, bl_channel);
filtered_rfc = filtered_rfc(1:TIME_AXIS_SIZE);
filtered_rfc_freq = 1/fs* fftshift(fft(filtered_rfc));

% Plot signal before passing through channel
figure
% time domain
subplot(2,1,1)
plot(t, raised_cosine_signal)
title('Transmitted Raised Cosine Signal (Time Domain)')
xlim([0 t_limit])
xlabel('Time (s)')
ylabel('Amplitude')

% freq domain
subplot(2,1,2)
plot(f_axis, abs(raised_cosine_signal_freq))
title('Transmitted Raised Cosine Signal (Freq Domain))')
xlim([-f_limit f_limit])
ylabel('Amplitude')
xlabel('Frequency (Hz)')

% Plot signal after passing through channel
figure
% time domain
subplot(2,1,1)
```

```
plot(t, filtered_rfc)
title('Received Raised Cosine Signal (Time Domain)')
xlim([0 t_limit])
xlabel('Time (s)')
ylabel('Amplitude')

% freq domain
subplot(2,1,2)
plot(f_axis, abs(filtered_rfc_freq))
title('Received Raised Cosine Signal (Freq Domain))')
xlim([-f_limit f_limit])
ylabel('Amplitude')
xlabel('Frequency (Hz)')
```

Graphs

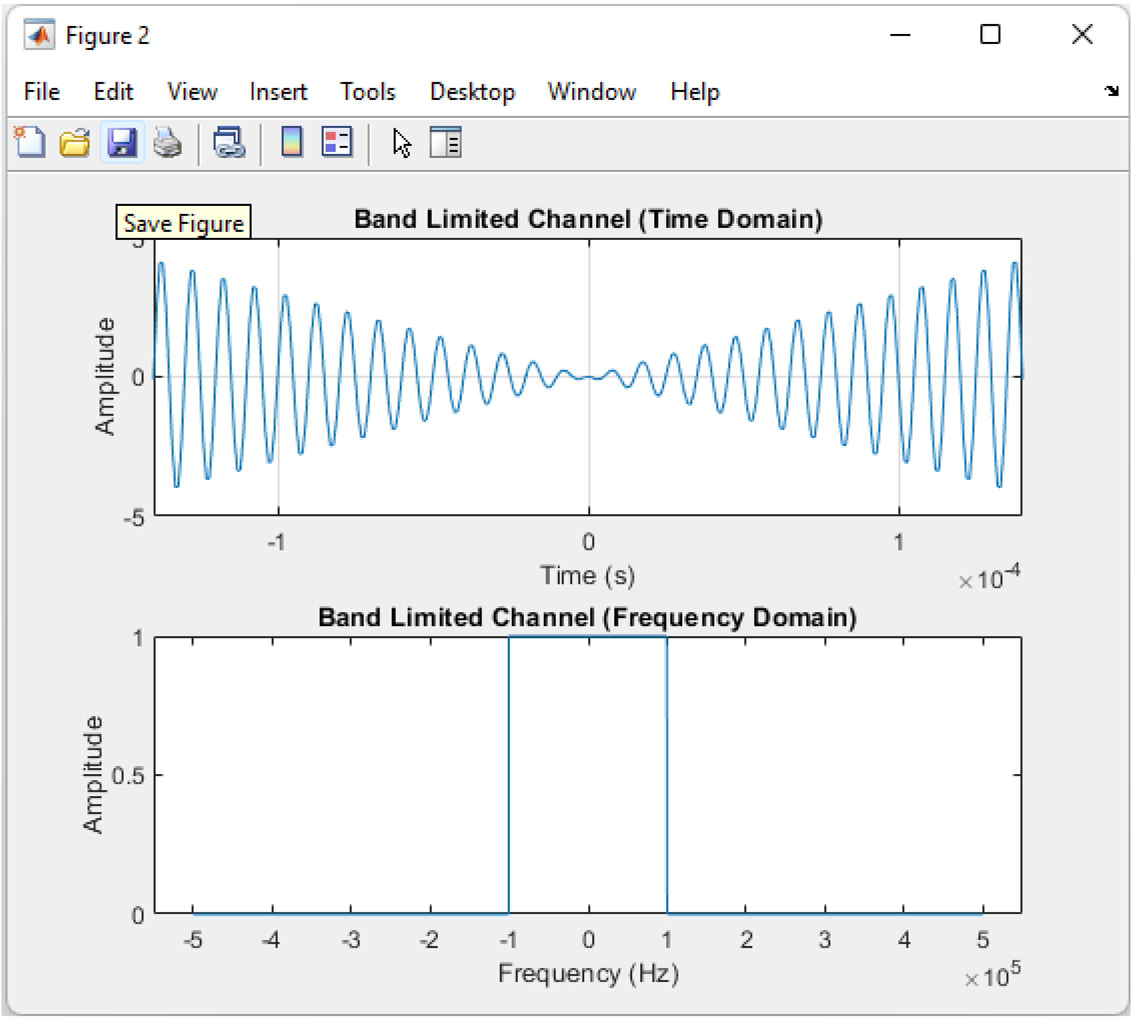
Figure 1



Explanation:

Showing the square signal pulse in time domain and frequency domain with duration T = 2/B, before it passes through the channel.

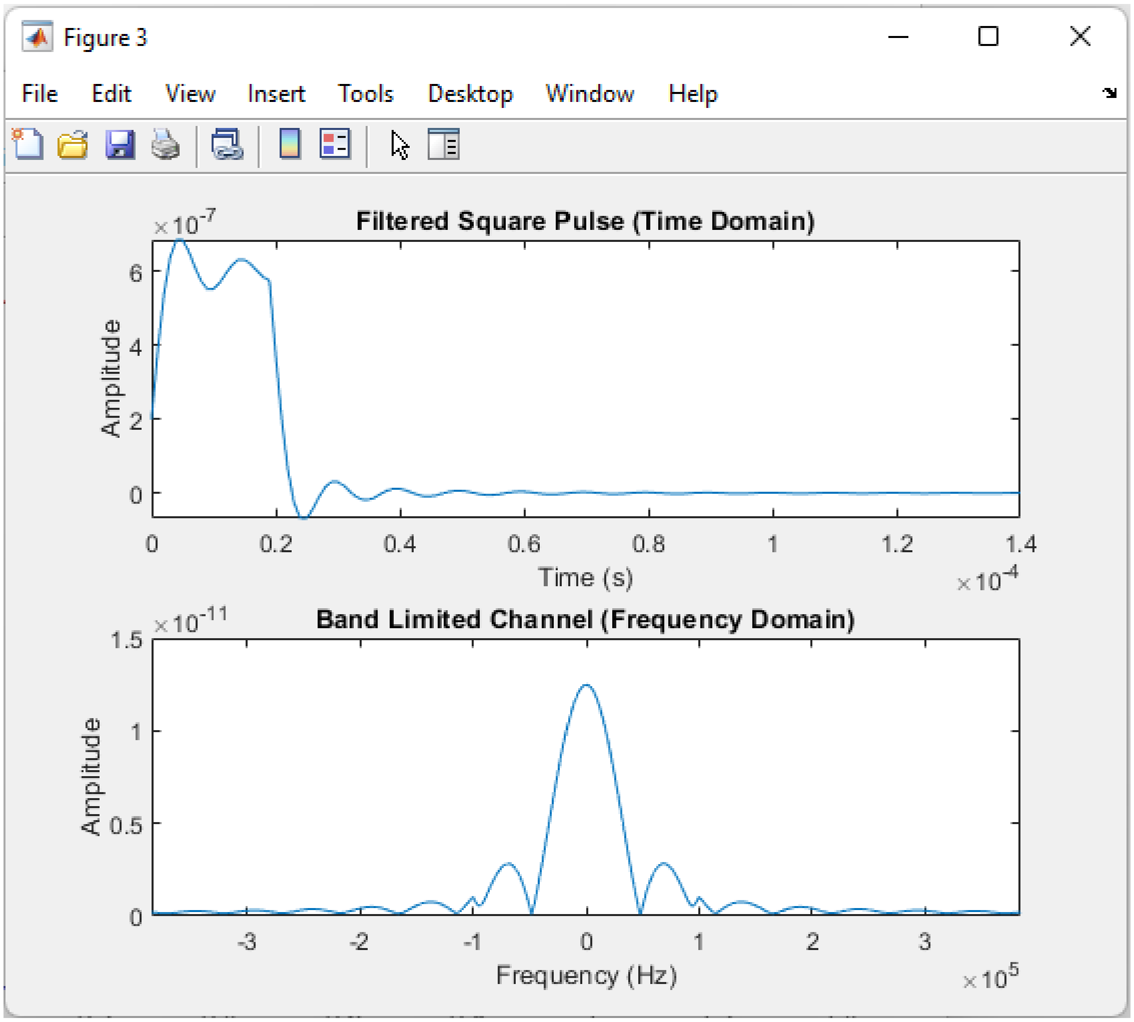
Figure 2



Explanation:

Showing the band limited channel in time domain and frequency domain

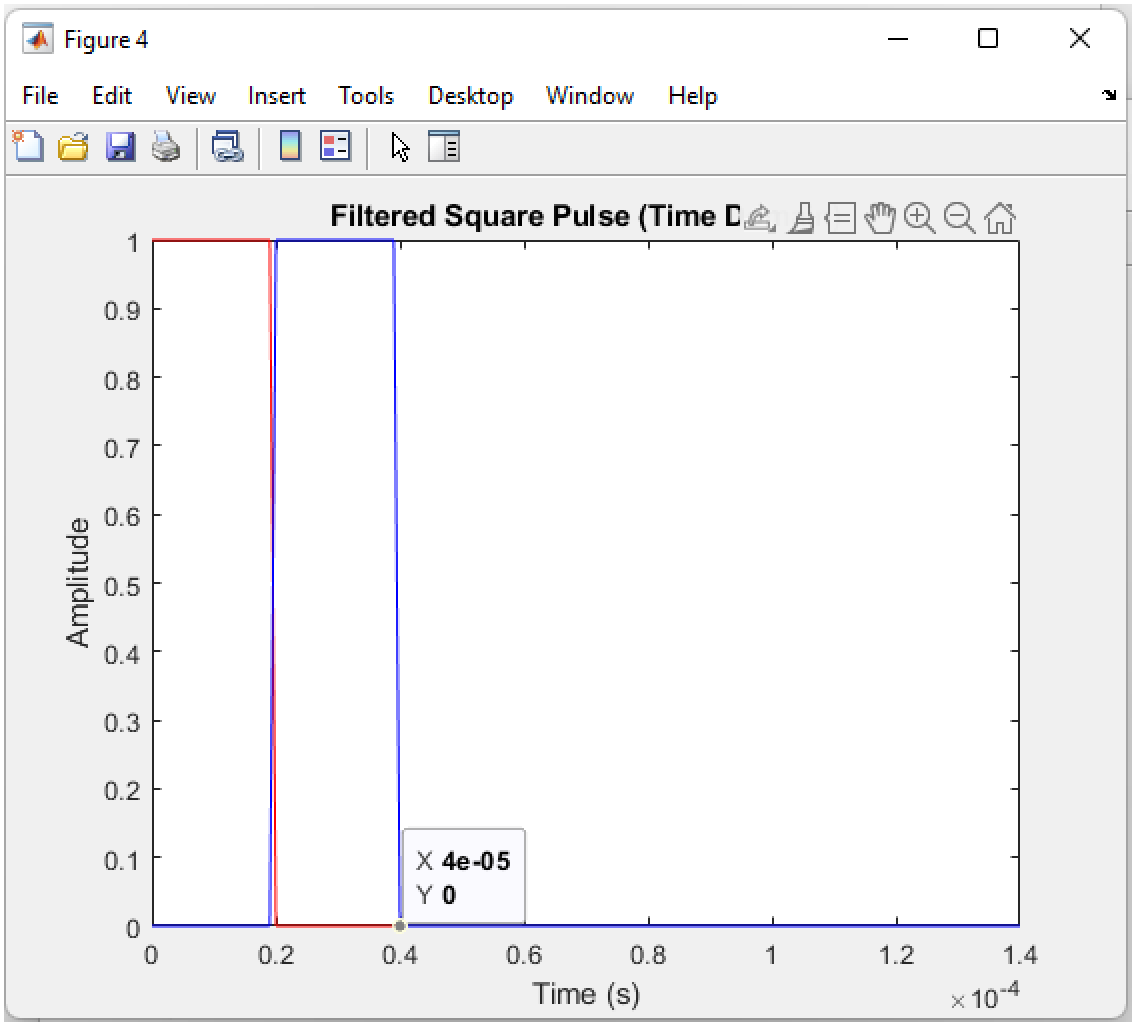
Figure 3



Explanation:

Showing the square signal pulse after it passes through the channel and has been filtered. Distortion of the signal can be noticed where the signal has a value after 0.2×10^{-4} seconds even though it should end at 0.2 seconds.

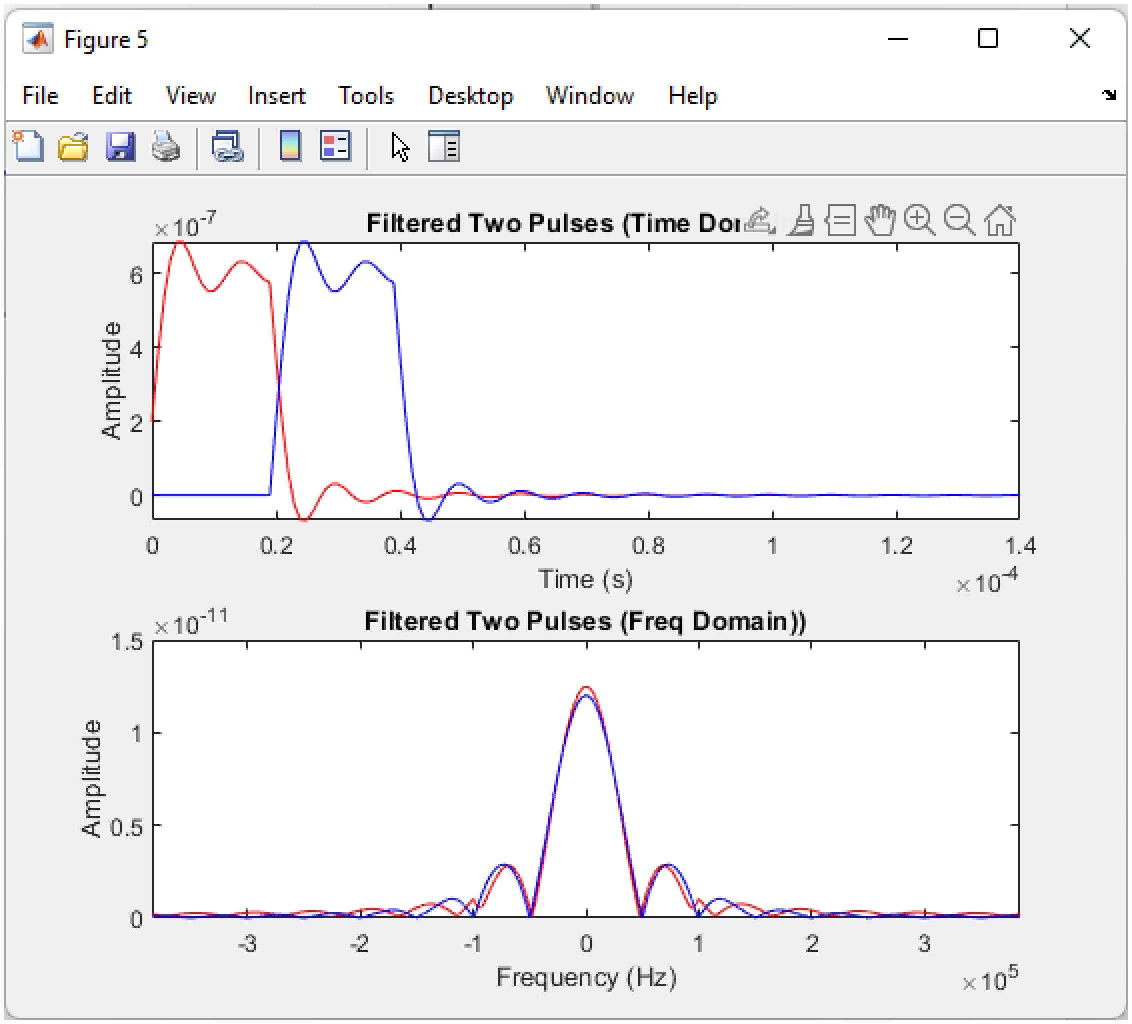
Figure 4



Explanation:

Showing plots for the second square pulse. the shapes of the two pulses are distinguishable on the plots using a different color.

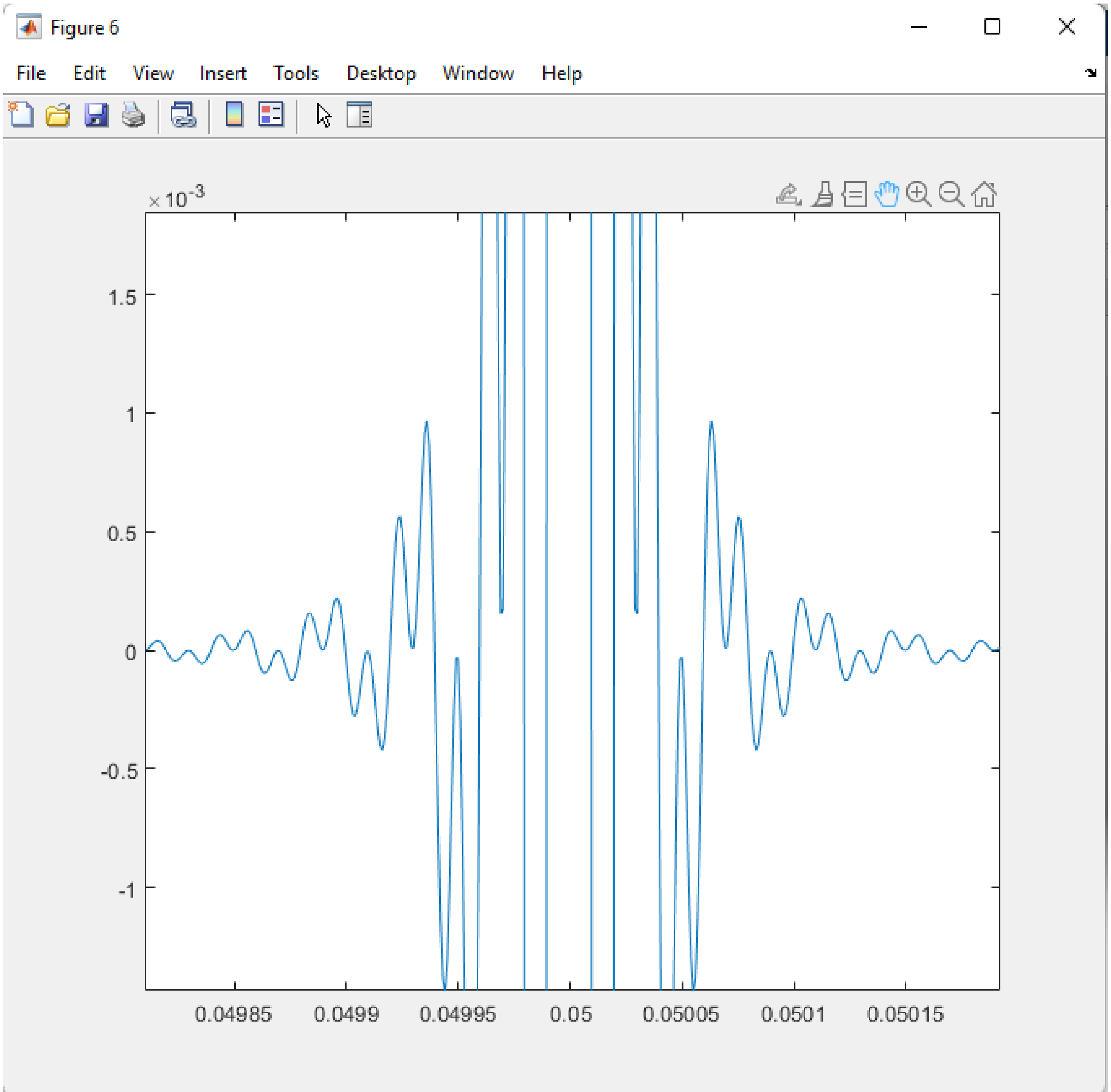
Figure 5



Explanation:

Showing the two square signals pulses after they pass through the channel and have been filtered. Inter-symbol interference happened as shown in the figure. This is due to the band-limited channel.

Figure 6

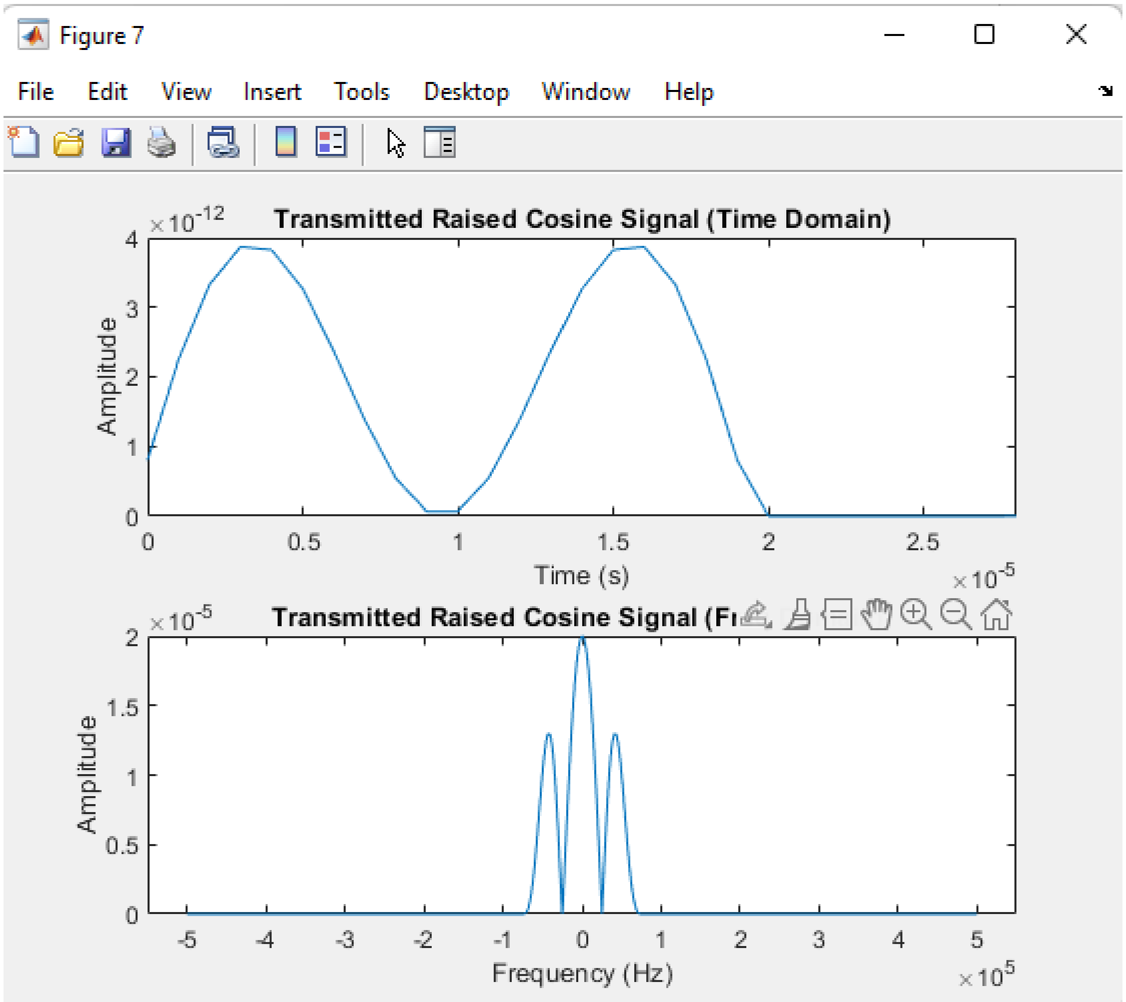


Explanation:

Showing the raised cosine filter, it is a very narrow pulse with the following equation

$$\text{filter} = \frac{sinc(2tW).\,cos(2\pi\beta W * t)}{1 - (4 * \beta t W)^2}$$

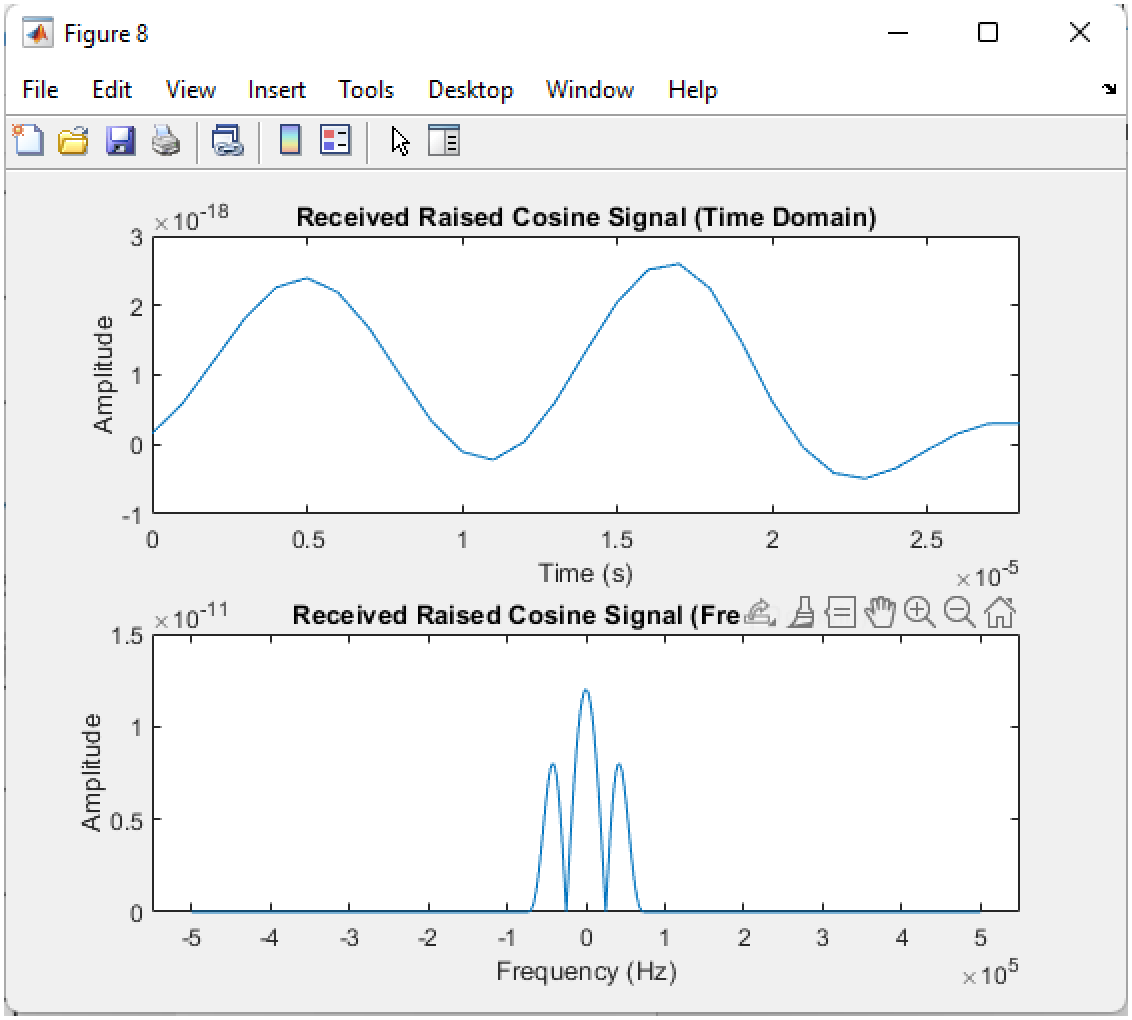
Figure 7



Explanation:

Showing the transmitted raised cosine signal in time domain and frequency domain. The filter converted the shape of the two pulses as shown in figure to be able to retrieve the pulses again.

Figure 8



Explanation:

Showing the received raised cosine signal in time domain and frequency domain.

The received signal can be retrieved with ease due to applying the raised cosine filter. No ISI occurred.

Nyquist Criterion for ISI

$$h(nT_s) = \begin{cases} 1; & n = 0 \\ 0; & n \neq 0 \end{cases}$$

Part 2: Inter-Symbol Interference due to multi-path channels

Code

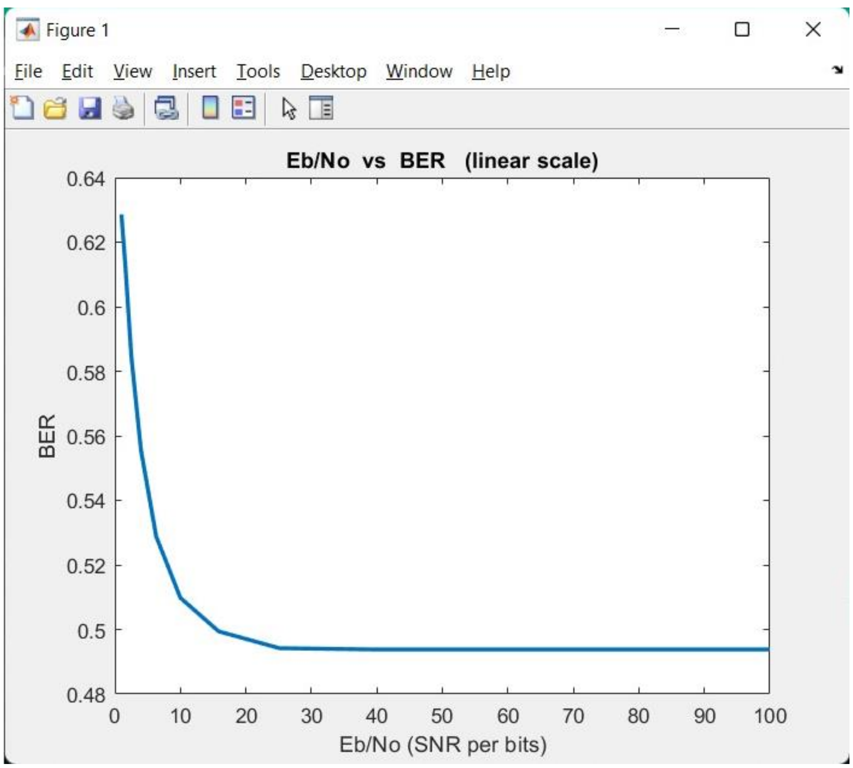
```
clear all
clc

%Number of paths
L = 50;

%number of bits
number_of_bits=5000;
%Generating random bit signal of size 5000
random_bits=randi([0 1],number_of_bits,1);

%BPSK modulater (0-> -1) & 1 not changed
for i = 1 : 1 : length(number_of_bits)
  if (random_bits(i) == 0 )
      random_bits(i) = -1;
  end
end

%assuming that the coefficients of the channel are Complex Gaussian with zero
mean and variance 1.
variance = 1;
%h is the channel effect on the transmitted signal
h = sqrt(variance/2) * (randn(L,1)+1i*rand(L,1));

%the effect of the length of each path & the attenuation
attenuation = exp(-0.5*[0:L-1])';
h = abs(h).*attenuation;

if (number_of_bits >= L)
    h = [h ; zeros(number_of_bits - L , 1)];
else
    h = h(1:number_of_bits);
end

H = zeros(length(h),length(h));

for i = 1:length(h)
    for k = 1:i
        H(i,k) = h(i-k+1);
    end
end

                %%%%%%%%%%%%%%%%%%%%For Testing%%%%%%%%%%%%%%
%noise (AWGN)
% noise = 0.05;    %assumption
% sigma = sqrt(noise);
% N = sigma*randn(number_of_bits,1);
%
%
% %Received signal after adding noise
```

```
% Y = ( H * random_bits) + N;

%passing the received signal through the equalizer
% H_inv = inv(H);
% orignal_signal = H_inv * Y;

% %removing noise from equalized signal
%     for k=1:1:number_of_bits
%         if (orignal_signal(k) > 0)
%             orignal_signal(k) = 1;
%         else
%             orignal_signal(k) = -1;
%         end
%     end

%BER
% error_bits = length( find(orignal_signal ~= random_bits) );
% BER_vector = error_bits/(length(random_bits));
            %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

H_inv = inv(H);
Eb=1; %given Energy
Eb_No_dB = (0:2:20)';          %% assumption
Eb_No = 10.^(Eb_No_dB/10);
No_vec = (Eb)./ Eb_No;
BER_vec = zeros(length(No_vec),1);
N_trail = 6;
for count = 1:length(No_vec)
    sigma = sqrt(No_vec(count));
    Noise = sigma*randn(number_of_bits,1);
    BER_vector = zeros(1,N_trail);
    for s = 1:N_trail        %calculate the BER 6 times for each noise ang
get the average
        Y = ( H * random_bits ) + Noise;
        orignal_signal = H_inv * Y;
        for k=1:1:number_of_bits
            if (orignal_signal(k) > 0)
                orignal_signal(k) = 1;
            else
                orignal_signal(k) = -1;
            end
        end
        error_bits = length( find(orignal_signal ~= random_bits) );
        BER_vector(s) = error_bits/(length(random_bits));
    end

    BER_vec(count) = (sum(BER_vector))/N_trail;

end
             %---------------------------plot---------------------------
%Linear:
figure(1)
plot(Eb_No , BER_vec,'linewidth',2);
xlabel('Eb/No (SNR per bits)')
ylabel('BER')
title('Eb/No  vs  BER   (linear scale)')
```

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

%DB:
figure(2)
semilogy(Eb_No_dB,BER_vec,'linewidth',2);
grid on;       hold on;
xlabel('Eb/No(dB) -SNR per bits-')
ylabel('BER')
title('Eb/No(dB)  vs  BER   (dB scale)')
```

Graphs

First Run



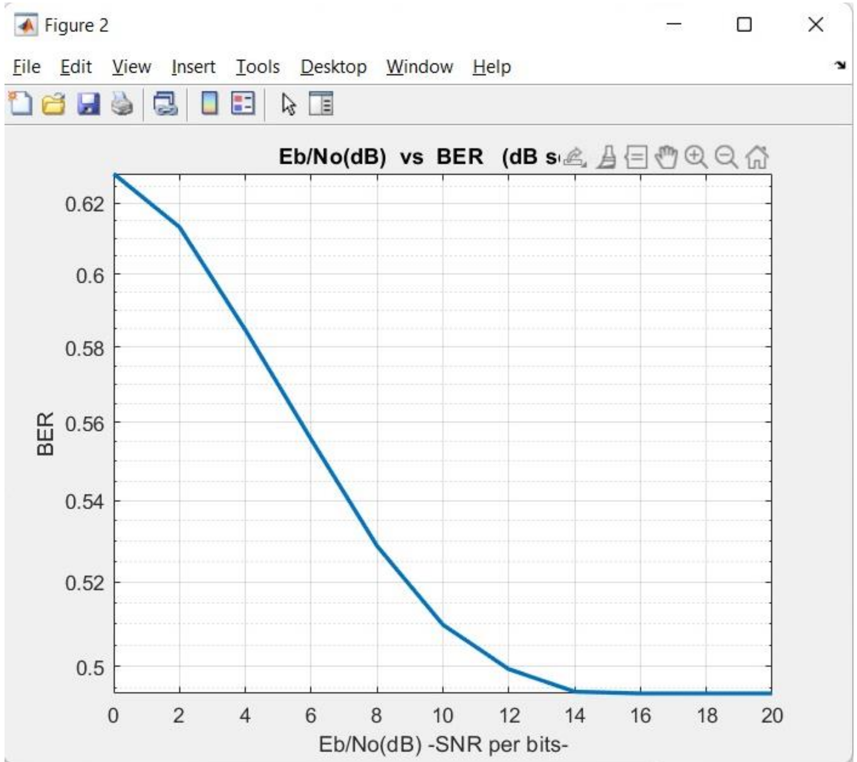
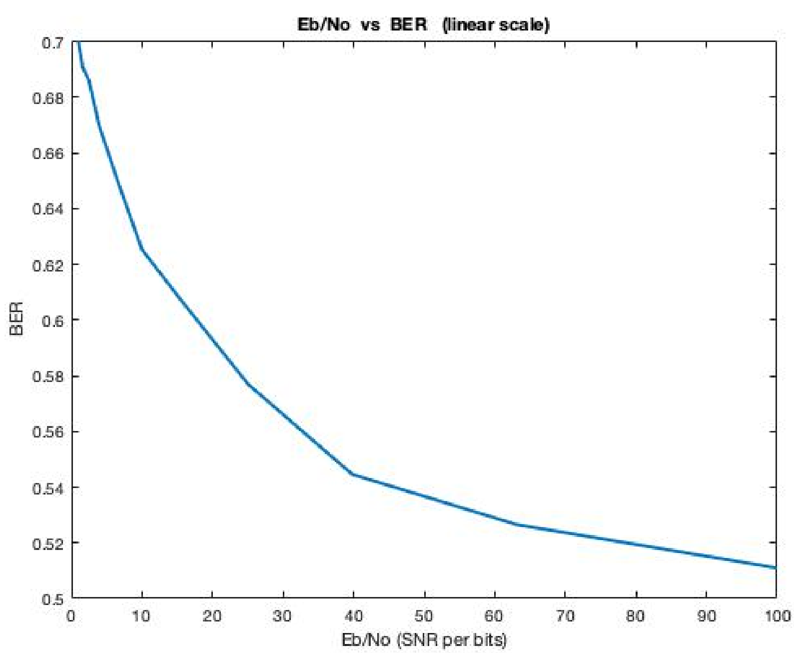
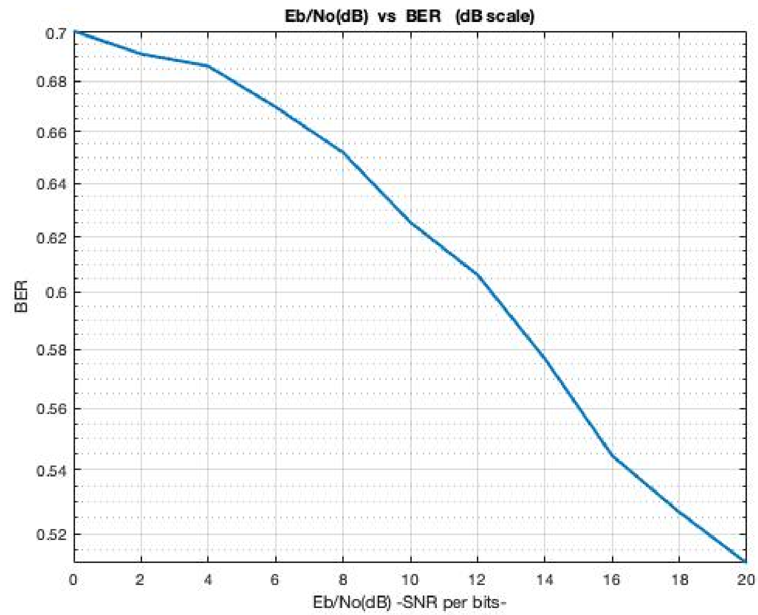


Figure 2
File Edit View Insert Tools Desktop Window Help
Eb/No(dB) vs BER (dB s
BER
0.62
0.6
0.58
0.56
0.54
0.52
0.5
0 2 4 6 8 10 12 14 16 18 20
Eb/No(dB) -SNR per bits-

second run

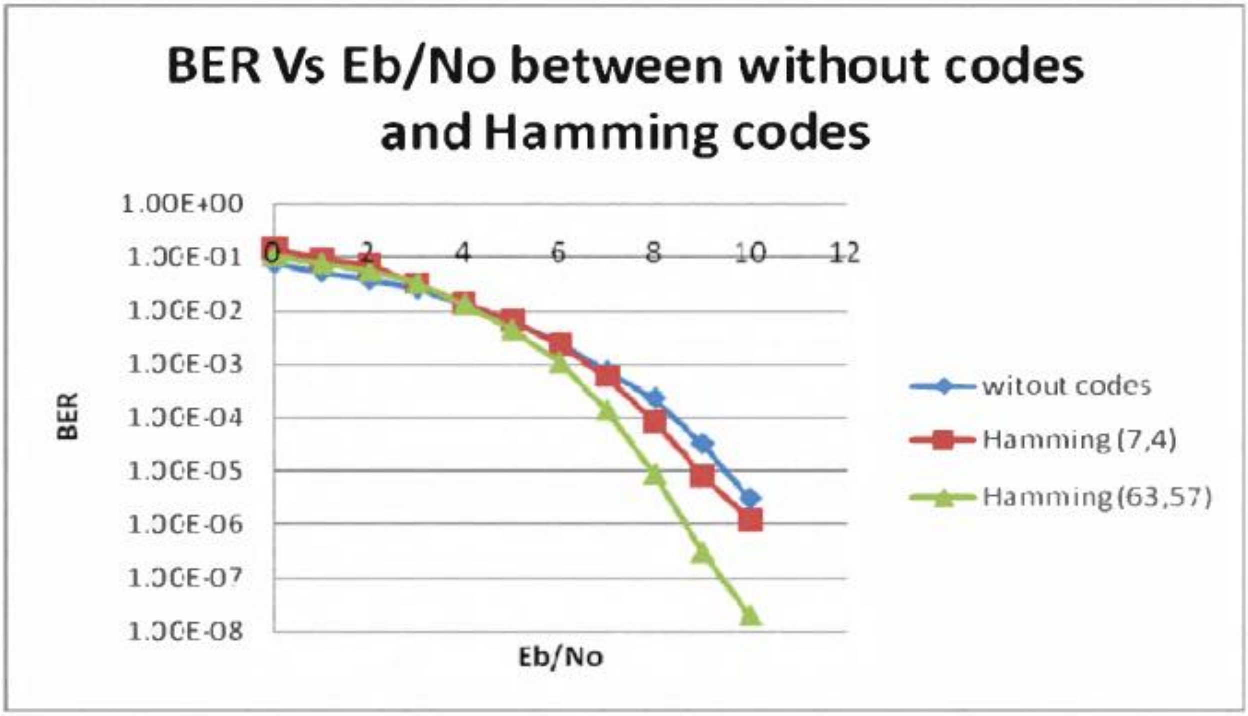


Eb/No vs BER (linear scale)
BER
0.7
0.68
0.66
0.64
0.62
0.6
0.58
0.56
0.54
0.52
0.5
0 10 20 30 40 50 60 70 80 90 100
Eb/No (SNR per bits)



Explanation

With every run it generates random bits and random noise therefore the 2 figures are different every run



Estimating the transmitted symbols on the receiver side

- Estimation of the real transmitted sequence is done by Zero Forcing Equalizer.
- If the receiver knows Y, H, and the statistics of AWGN:
- Theoretical way:
    1. Calculate the inverse of the matrix H.
    2. Multiply Y by H to get the product matrix as the equalized sequence.
    3. We could subtract the AWGN from the product or we could leave it pretending that the receiver doesn't know the statistics of AWGN.
    4. The transmitted sequence X is estimated
- Practical Way:
    1. Use an encoding error correction scheme.
    2. Encode the signal.
    3. Use Maximum Likelihood Equalizer.
    4. Transmit Signal
    5. Receive the Signal
    6. Decode the received Signal.