Digital Communications

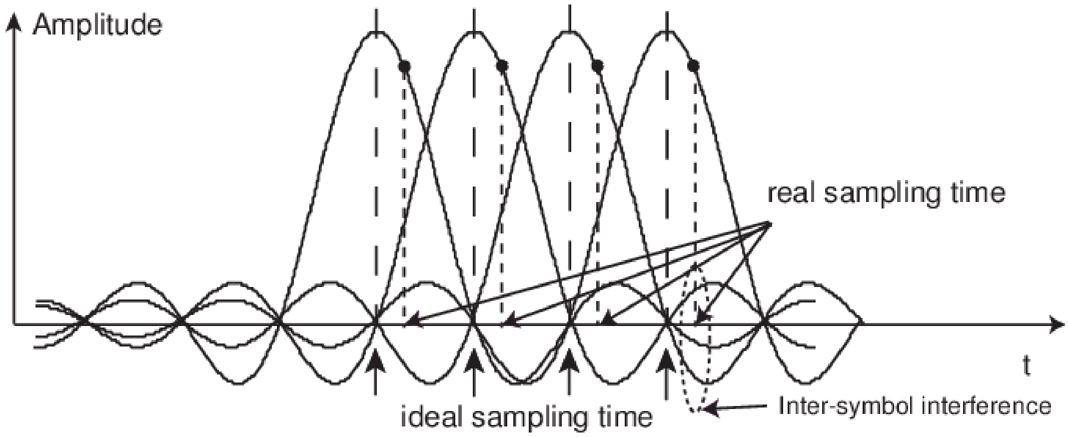
Lab (3)

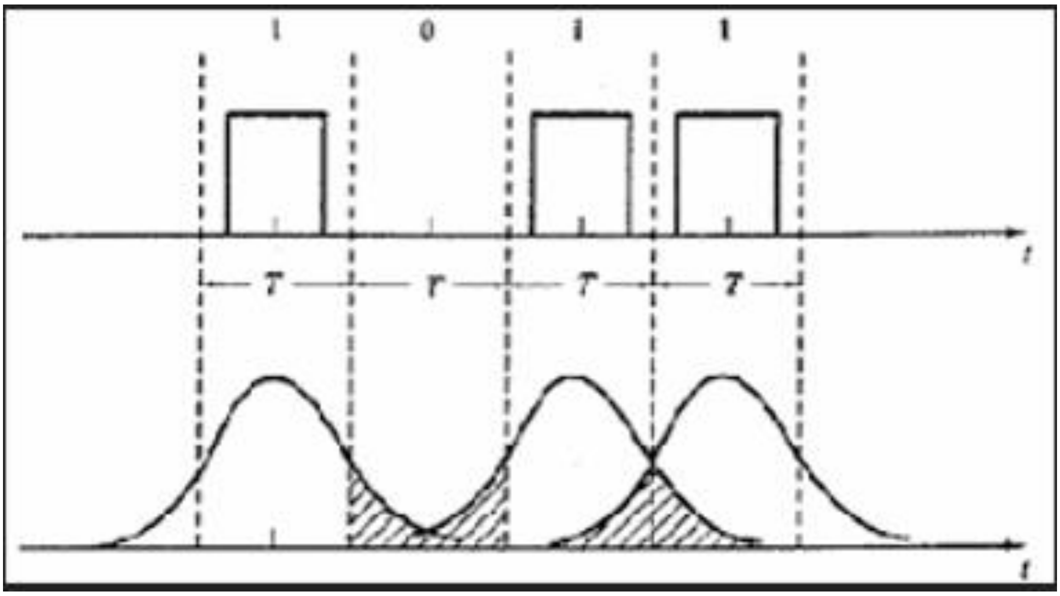
Team Members

Menatallah Moustafa	6234
Farah Ahmed	6274
Alaa Mohamed Abdel Hamid	6473
Rawan Hindawy	6491
Nouran Hisham	6532
Kareem sabra	6594
Esraa Mahmoud	6597
Nourhan Waleed	6609
Seif Mohamed	6624
Fatema Moharam	6655

Part (1)

Inter-Symbol Interference due to band-limited channels:





The code:

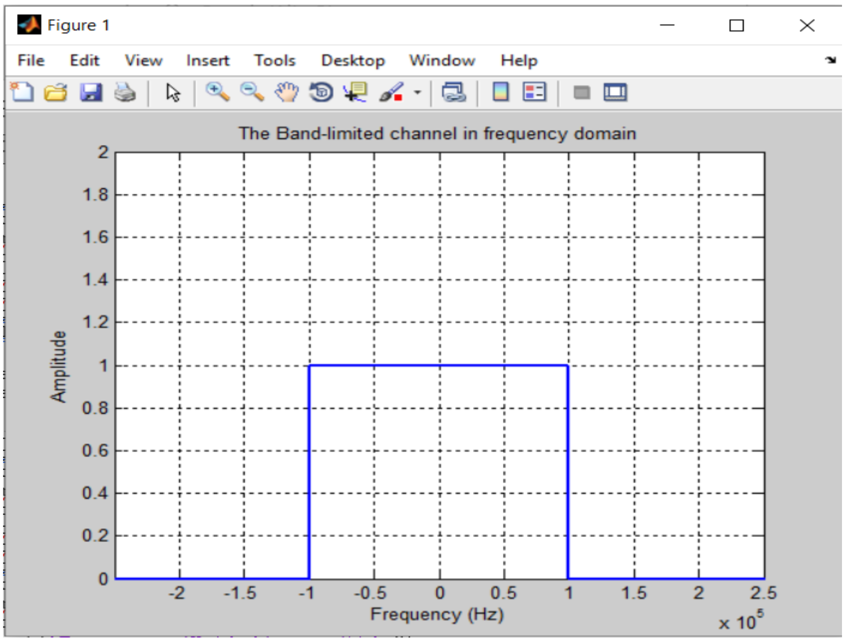
```
clc
close all
clear all

fs = 1e7;
Ts = 1/fs;
N = 1e7;
time_axis = (0:N-1)*Ts;

freq_axis = -fs/2:fs/N:fs/2-1/N;
B = 100e3;
T = 2/B;
```

```
%% Creating Band-limited channel

one_square = ones(1,200e3);
zero_me = zeros(1,9800e3/2);
Band_limited_channel= [zero_me one_square zero_me];

figure
plot(freq_axis,Band_limited_channel,'linewidth',2)
grid on
ylim([0 2])
xlim([-1/T 1/T]*5)
xlabel('Frequency (Hz)','linewidth',2)
ylabel('Amplitude','linewidth',2)
title('The Band-limited channel in frequency domain','linewidth',10)


%% Generating first square pulse

x_bits = [1];
pulse1 = rectpuls(time_axis-1/B,T);
pulse1_length = length(pulse1);
pulse1_fft = (1/200)*fftshift(fft(pulse1));
freq_axis = -fs/2:fs/pulse1_length:fs/2-1/pulse1_length;


%% Plotting first square pulse

figure
subplot(2,1,1)
plot(time_axis,pulse1,'b','linewidth',2); hold on;
grid on
xlim([0 T*4.2])
ylim([0 2])
xlabel('Time (s)','linewidth',2)
ylabel('Amplitude','linewidth',2)

subplot(2,1,2)
plot(freq_axis,abs(pulse1_fft),'b','linewidth',2); hold on;
grid on
ylim([0 2])
xlim([-1/T 1/T]*5)
xlabel('Frequency (Hz)','linewidth',2)
ylabel('Amplitude','linewidth',2)
subplot(2,1,1)
title('Square pulse before passing through channel','linewidth',10)


%% Passing first square pulse through the Band-limited channel

pulse1_after_chann = pulse1_fft .* Band_limited_channel;
pulse1_after_chann_T =100* ifft(ifftshift(pulse1_after_chann));

figure
subplot(2,1,1)
plot(time_axis,pulse1_after_chann_T,'b','linewidth',2); hold on;
grid on
xlim([0 T*5])
xlabel('Time (s)','linewidth',2)
ylabel('Amplitude','linewidth',2)
```

```
subplot(2,1,2)
plot(freq_axis,abs(pulse1_after_chann),'b','linewidth',2); hold on;
grid on
xlim([-1/T 1/T]*5)
ylim([0 2])
xlabel('Frequency (Hz)','linewidth',2)
ylabel('Amplitude','linewidth',2)
subplot(2,1,1)
title('Square pulse after passing through channel','linewidth',10)


%% Generating second square pulse

x_bits = [0 1];
pulse2 = rectpuls(time_axis-3/B,T);
pulse2_length = length(pulse2);
pulse2_fft =(1/200)* fftshift(fft(pulse2));


%% Plotting the 2 square pulses

figure
subplot(2,1,1)
plot(time_axis,pulse1,'b','linewidth',2); hold on;
plot(time_axis,pulse2,'r','linewidth',2); hold on;
grid on
xlim([0 T*4.2])
ylim([0 2])
xlabel('Time (s)','linewidth',2)
ylabel('Amplitude','linewidth',2)
title('Square pulses before passing through channel','linewidth',10)


%% Passing the 2 square pulses through the Band-limited channel

pulse2_after_chann = pulse2_fft .* Band_limited_channel;
pulse2_after_chann_T =100* ifft(ifftshift(pulse2_after_chann));

subplot(2,1,2)
plot(time_axis,pulse1_after_chann_T,'b','linewidth',2); hold on;
plot(time_axis,pulse2_after_chann_T,'r','linewidth',2); hold on;
grid on
xlim([0 T*5])
xlabel('Time (s)','linewidth',2)
ylabel('Amplitude','linewidth',2)
title('Square pulses after passing through channel','linewidth',10)


%% creating Sinc function and passing it through the band-limited
channel

time_axis = (-N/2:N/2-1)*Ts;
y = sinc(time_axis*B);
y1 = [zeros(1,10000) y(1:9990000)];
y1_length = length(y1);
y1_f = (1/100)*fftshift(fft(y1));
freq_axis = -fs/2:fs/y1_length:fs/2-1/y1_length;
```

```
figure
subplot(2,1,1)
plot(time_axis,y1,'b','linewidth',2); hold on;
xlim([0.0008 0.0012])
grid on
xlabel('Time (s)','linewidth',2)
ylabel('Amplitude','linewidth',2)
subplot(2,1,2)
plot(freq_axis,abs(y1_f),'b','linewidth',2); hold on;
xlim([-80000 80000])
grid on
xlabel('Frequency (Hz)','linewidth',2)
ylabel('Amplitude','linewidth',2)
subplot(2,1,1)
title('Sinc function before passing through channel','linewidth',10)

y1_after_ch = y1_f .* Band_limited_channel;
y1_after_ch_T =100* ifft(ifftshift(y1_after_ch));

figure
subplot(2,1,1)
plot(time_axis,y1_after_ch_T,'b','linewidth',2); hold on;
xlim([0.0008 0.0012])
grid on
xlabel('Time (s)','linewidth',2)
ylabel('Amplitude','linewidth',2)
subplot(2,1,2)
plot(freq_axis,abs(y1_after_ch),'b','linewidth',2); hold on;
xlim([-80000 80000])
grid on
xlabel('Frequency (Hz)','linewidth',2)
ylabel('Amplitude','linewidth',2)
subplot(2,1,1)
title('Sinc function after passing through channel','linewidth',10)
```

The output:



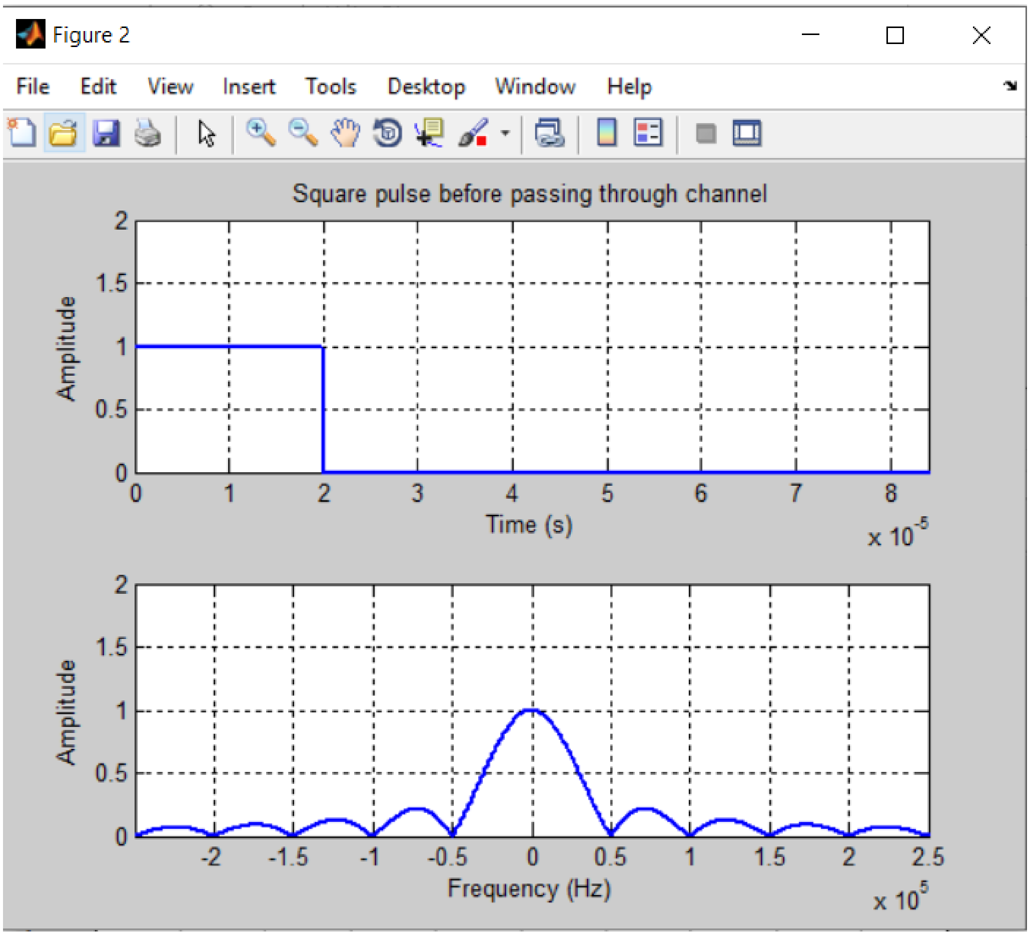


Figure 2
File Edit View Insert Tools Desktop Window Help
Square pulse before passing through channel
Amplitude
Time (s)
x 10^-5
Amplitude
Frequency (Hz)
x 10^5

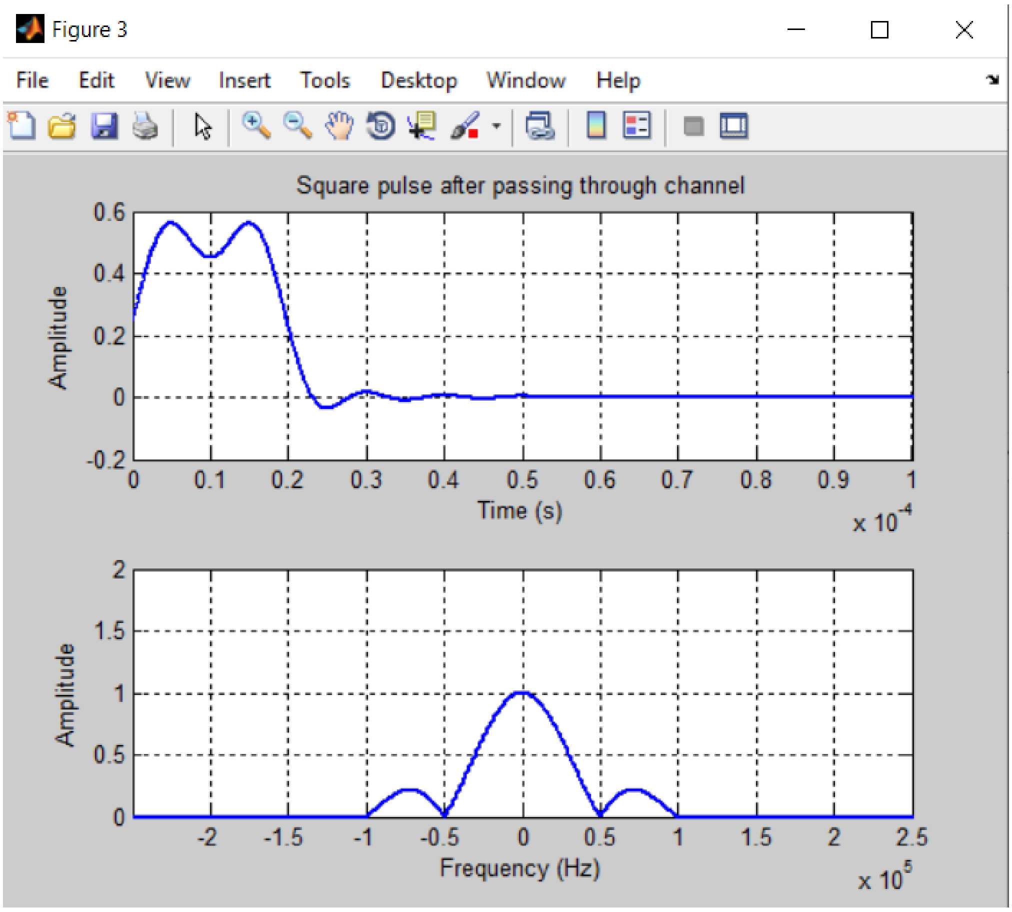


Figure 3
File Edit View Insert Tools Desktop Window Help
Square pulse after passing through channel
Amplitude
Time (s)
x 10^-4
Amplitude
Frequency (Hz)
x 10^5

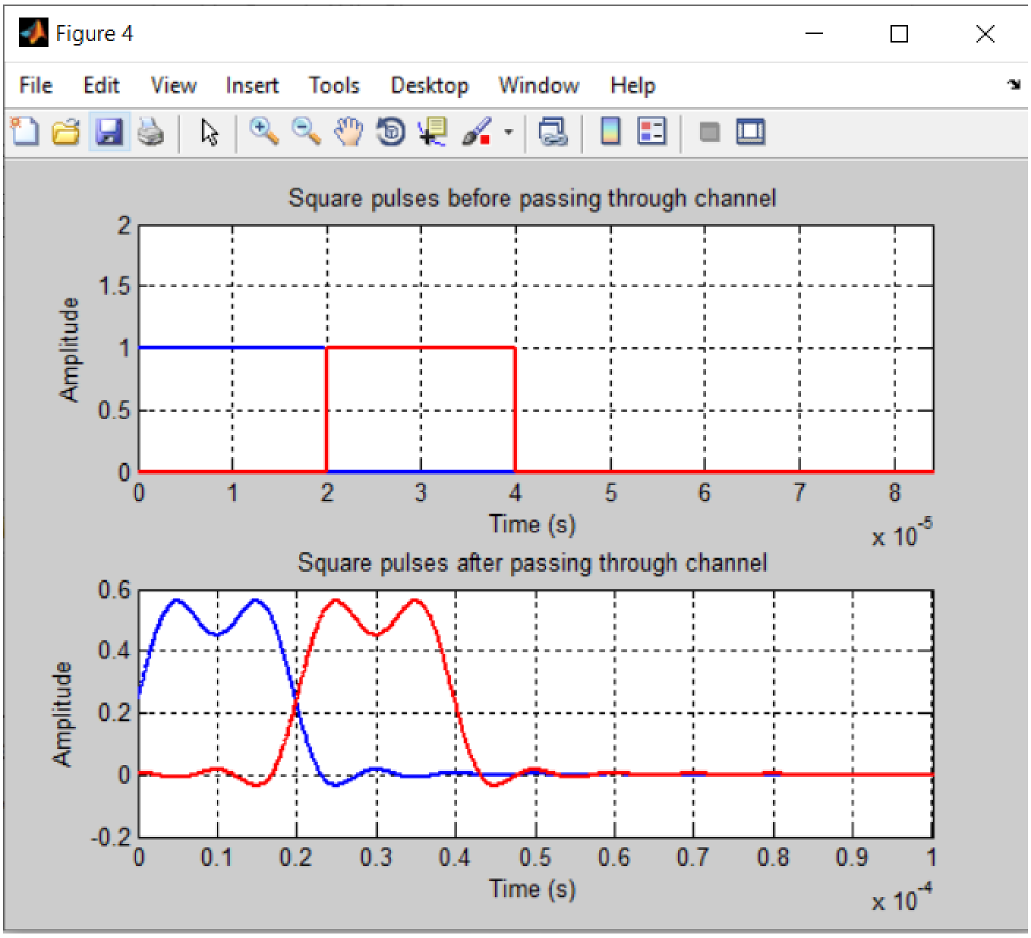


Figure 4
File Edit View Insert Tools Desktop Window Help
Square pulses before passing through channel
Amplitude
Time (s)
x 10^-5
Square pulses after passing through channel
Amplitude
Time (s)
x 10^-4

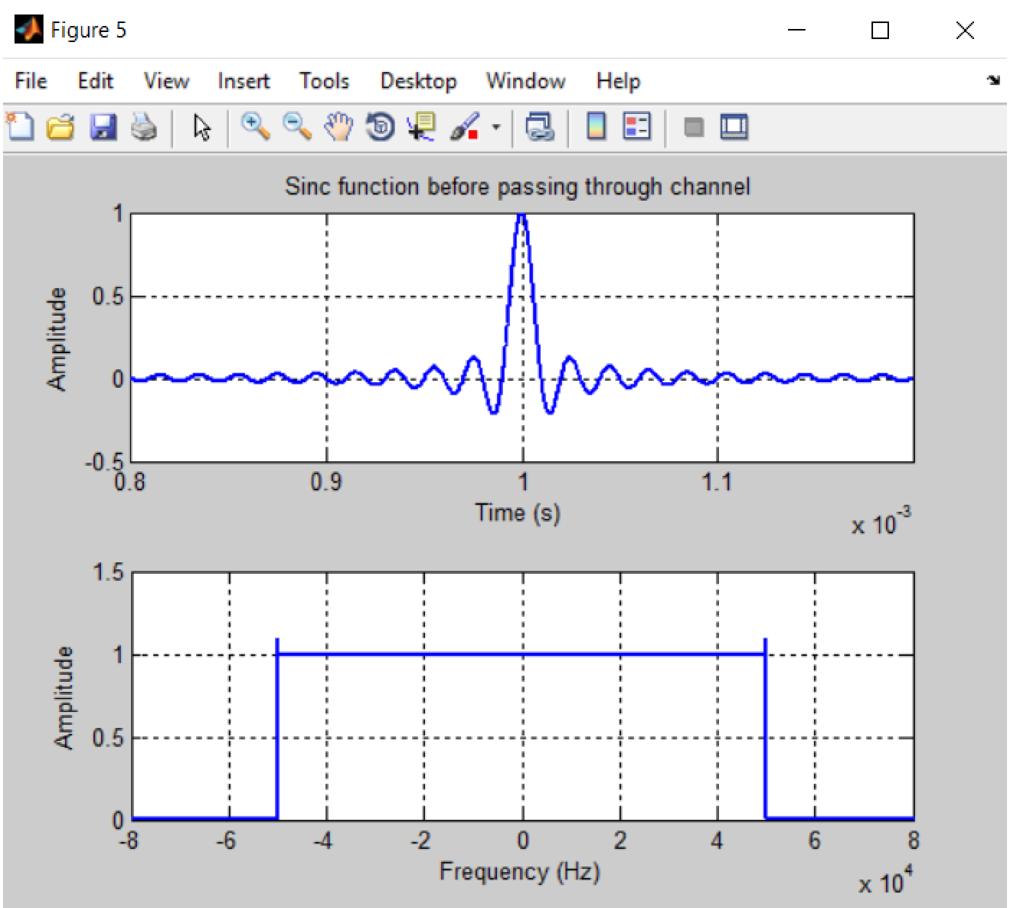
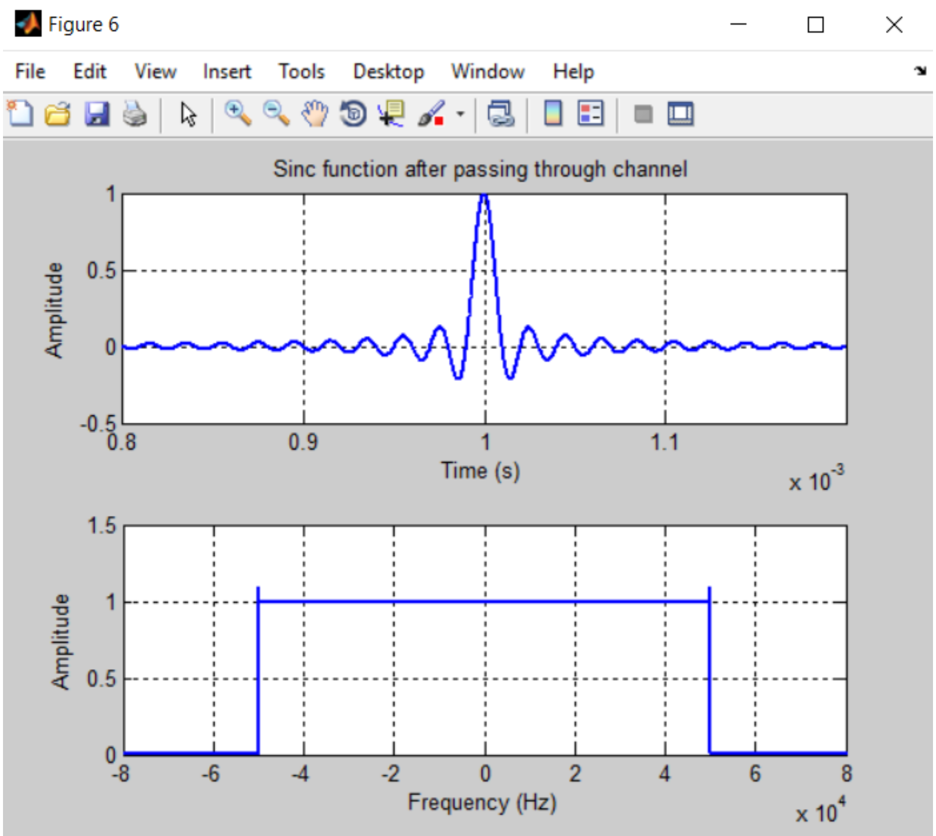


Figure 5
File Edit View Insert Tools Desktop Window Help
Sinc function before passing through channel
Amplitude
Time (s)
x 10^-3
Amplitude
Frequency (Hz)
x 10^4



ISI can negatively affect the detection performance of multiple consecutive signals. To combat the effect of ISI in band-limited channels, one cannot use square pulses anymore. Instead, there are other pulse shapes that are better suited for such channels. We need to investigate such solutions.

So, we use raised cosine to solve the problem of ISI

Raised Cosine Filter:

The raised-cosine filter is an implementation of a low-pass Nyquist filter.

It can be produced in terms :

A) Time Domain :

$$p(t) = \operatorname{sinc}(2Wt)\frac{\cos(2\pi\alpha Wt)}{1-16\alpha^2W^2t^2}$$

B) Frequency Domain:

$$P(f) = \begin{cases} \dfrac{1}{2W}, & 0 \le |f| < f_1 \\ \dfrac{1}{4W}\left\{1+\cos\left[\dfrac{\pi}{2W\alpha}(|f|-f_1)\right]\right\}, & f_1 \le |f| < 2W - f_1 \\ 0, & |f| \ge 2W - f_1 \end{cases}$$

Parameters of Raised Cosine :

Bandwidth : *W*

Rb/2= 2*Tb*

Roll of factor $\alpha = 1 - f/W$

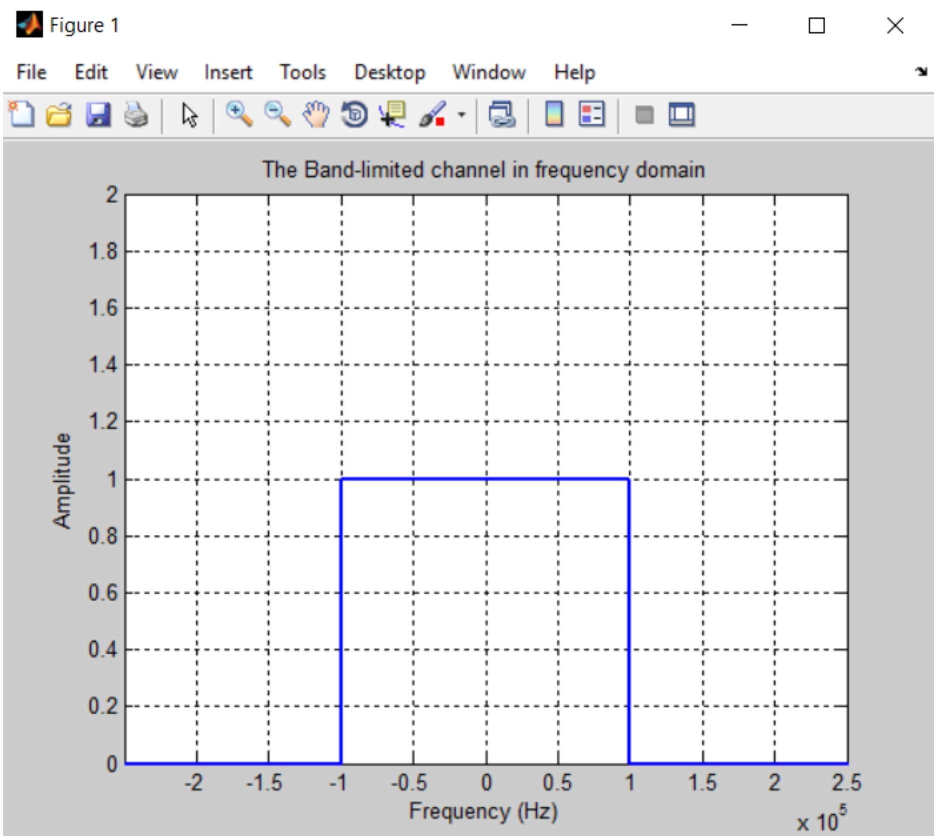
The main parameter of a raised cosine filter is its roll-off factor, which indirectly specifies the bandwidth of the filter. Ideal raised cosine filters have an infinite number of taps. Therefore, practical raised cosine filters are windowed.

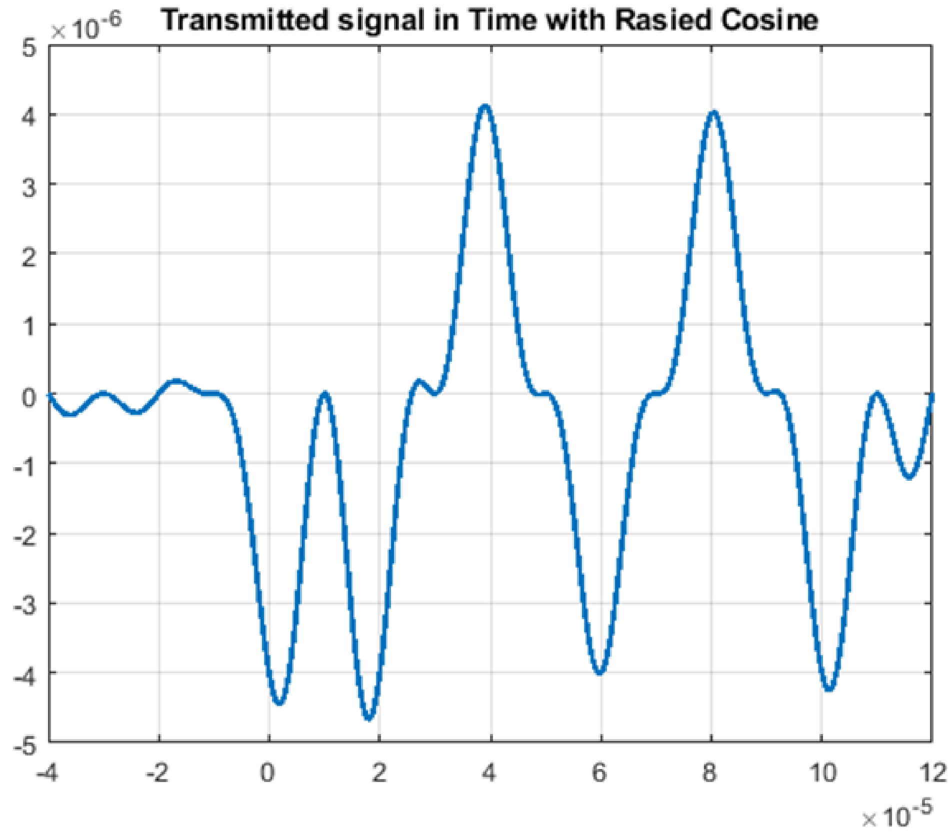
The code:

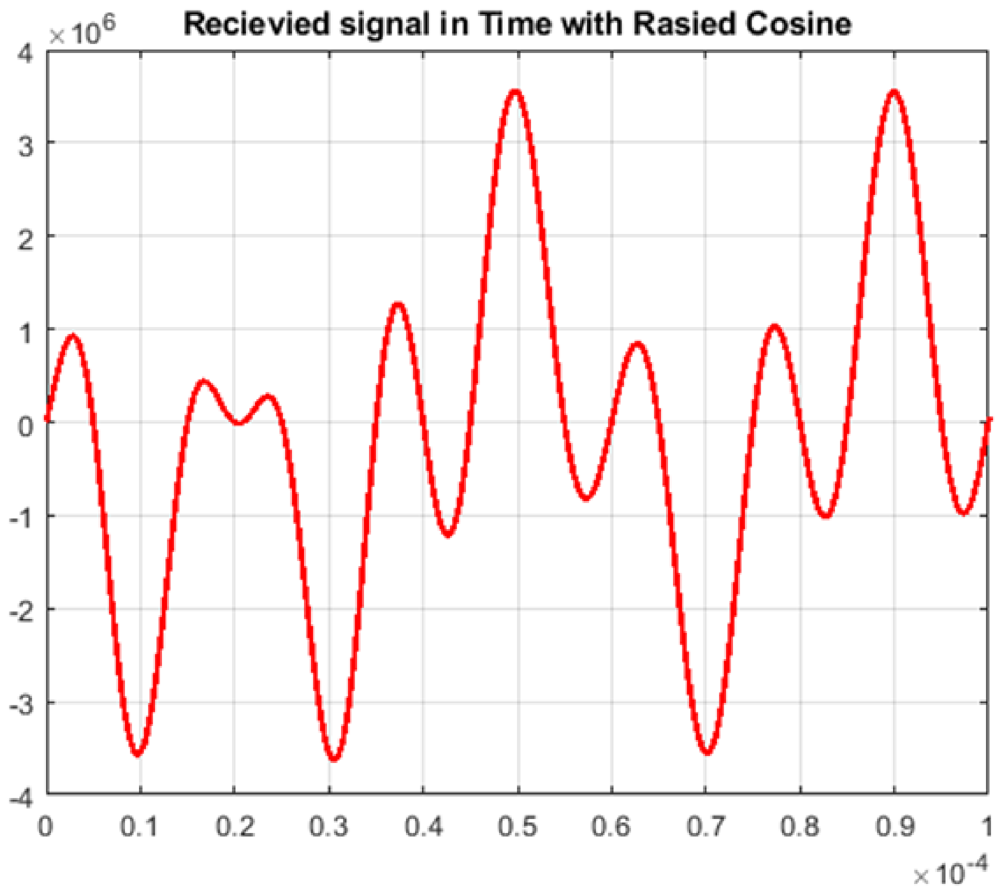
```
clc
close all
clear all

fs = 1e7;
Ts = 1/fs;
N = 1e7;
time_axis = (0:N-1)*Ts;
freq_axis = -fs/2:fs/N:fs/2-1/N;
B = 100e3;
T = 2/B;

%% Creating Band-limited channel

one_square = ones(1,200e3);
zero_me = zeros(1,9800e3/2);
Band_limited_channel= [zero_me one_square zero_me];

figure
plot(freq_axis,Band_limited_channel,'linewidth',2)
grid on
ylim([0 2])
xlim([-1/T 1/T]*5)
xlabel('Frequency (Hz)','linewidth',2)
ylabel('Amplitude','linewidth',2)
title('The Band-limited channel in frequency domain','linewidth',10)

%% Generating raised cosine

x_bits = [1 1 0 1 0 1];

alpha = 0.5;
W = 1/T;
n = 10000000 ;

t_axis = (0:n-1)*Ts;
N2 = length(t_axis);
x_square = zeros(1, N2);
N_sq = round(T/Ts);
t_axis_sh = ((-(n-1)/2):((n-1)/2))*Ts;
one_raised_cos = sinc(2*W*t_axis_sh).*cos(2*pi*alpha*W*t_axis_sh)./(1-
(4*alpha*W*t_axis).^2);
```

```
for i = 1:length(x_bits)

        if x_bits(i) == 1
            x_square = x_square + circshift(one_raised_cos' , N_sq*(i-1))';
        else
            x_square = x_square - circshift(one_raised_cos' , N_sq*(i-1))';
        end
end

%% Plotting raised cosine

figure
plot(t_axis_sh,x_square , 'linewidth',2)
xlim([-T*2 T*6])
title('Transmitted signal in Time with Rasied Cosine','linewidth',10)
grid on

%% Passing raised cosine through the Band-limited channel

x_square_fft = fftshift(fft(x_square));
x_square_after_chann = x_square_fft .* Band_limited_channel;
x_square_after_chann_T = ifft(ifftshift(x_square_after_chann));
figure
plot(time_axis,x_square_after_chann_T,'r','linewidth',2);
xlim([-T*2 T*6])
title('Recievied signal in Time with Rasied Cosine','linewidth',10)
grid on
```

The output:

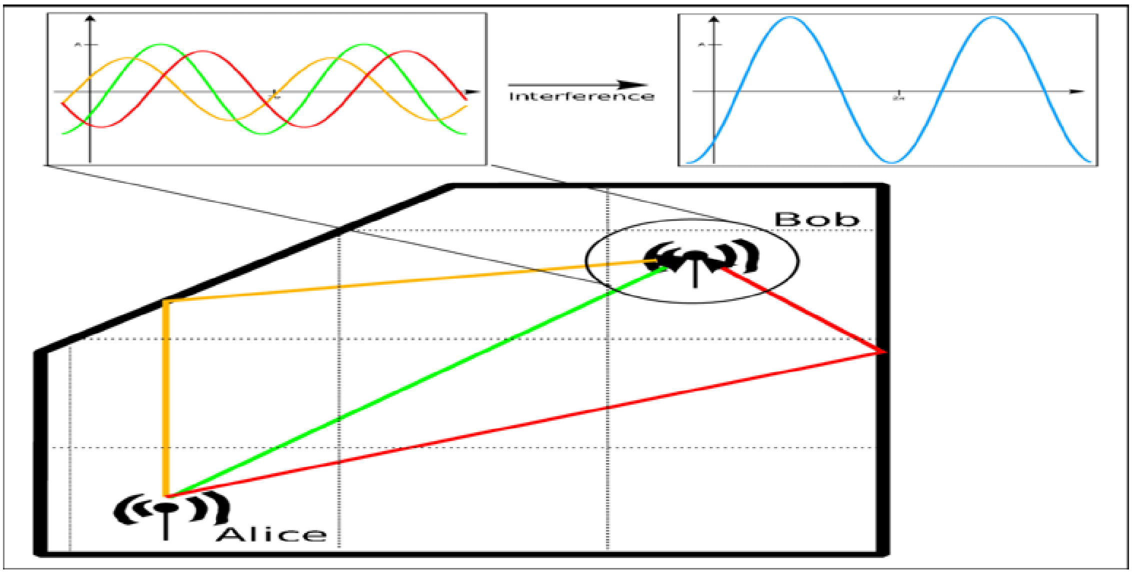



Transmitted signal in Time with Rasied Cosine
×10^-6
5
4
3
2
1
0
-1
-2
-3
-4
-5
-4
-2
0
2
4
6
8
10
12
×10^-5


Recievied signal in Time with Rasied Cosine
×10^6
4
3
2
1
0
-1
-2
-3
-4
0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
0.9
1
×10^-4

Part (2)

Inter-Symbol Interference due to multi-path channels



The code:

```
%Transmitted signal is x
% received signal is y
% N is the awgn noise that corrupts y
% our function is Y = HX+N where H(i) is the channel effect of the i+1
% channel and is represented by a 2D matrix of dimension L*L

%initializing variables
Eb = 1;
%dimension of matrix, could be changed, I used 5 for debugging
L = 50;
H = zeros(L,L);
%channel function that we're going to use  % e^-0.5*x^2
channel_function = exp(-0.5*[0:L-1].^2)';
%channel_function = repmat(channel_function,1,1);
%creating an array of gaussian distribution numbers, mean =0, variance
=1
h = randn(L,1);
%i added the other randn as the graph looked weird
% awgn noise 11 values
No = [0 0.001 0.005 0.01 0.05 0.1 0.25 0.5 0.7 0.9 1];
% initializing empty array to store the BER values inside the loop for
plotting
temporary_BER = [];
% initializing empty array for BER/noise
BER = [];
h = abs(h).*channel_function;
```

```
%filling matrix
i=1;
j = 1;
for k = 1:L
    for m = i:-1:1
        H(k,j) = h(m);
        j = j +1;
    end
    j = 1;
    i = i + 1;
end
%disp(h)
%disp(H)
%inverse_H = inv(H);   %inversing the matrix

%calculating BER
for a = No
    %calculating noise power
    N = sqrt(a/2)*randn(L,1);
    %calculating the BER 11 times for each noise
    for i = 1:11
        %returning the non zero values with no repitition
        non_zero_values = setdiff(-1:1, 0);
        x = non_zero_values( randi(length(non_zero_values), L, 1) );
        %recieved bits/ signal
        y = H*x' + N;
        %recieved bits plus noise
        x_received =inv(H)*y;
        D = zeros(size(x_received));
        for k = 1:L
            if x_received(k) <= 0
        %polar type, if it's less than zero it's -1, if it's more it's -
1
                D(k)= -1 ;
            else
                D(k) = 1;
            end
        end
        n = 0;
        for k = 1:L
            %checking with initial x to calculate BER
            if D(k) ~= x(k)
                n = n + 1;
            end
        end
        temporary_BER = [temporary_BER n/L];
    end
    %calculating the mean BER for each noise
    BER = [BER mean(temporary_BER)];
    %reseting the array for the new noise
    temporary_BER = [];
end
plot(Eb./No, BER)
xlabel('Eb/No')
ylabel('BER')
xlim([0,100])
```

The output:

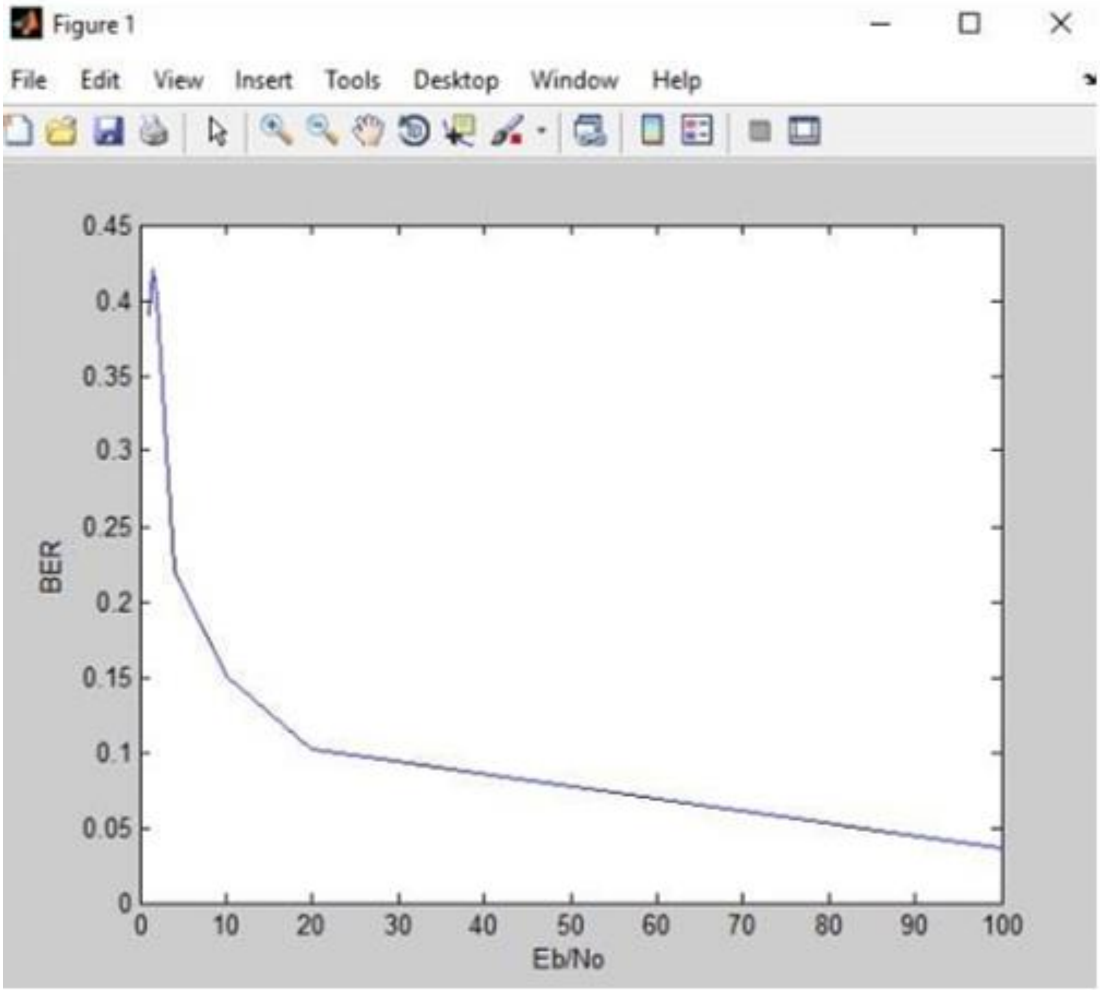


Figure 1
File Edit View Insert Tools Desktop Window Help
0.45
0.4
0.35
0.3
0.25
0.2
0.15
0.1
0.05
0
BER
0 10 20 30 40 50 60 70 80 90 100
Eb/No