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

In this part we define sampling rate, the Bandwidth of the channel and the duration of the pulse

This file can be opened as a Live Script. For more information, see Creating Live Scripts.

```
close all
clc
fs = 100*(10^5);
ts = 1/ fs ;
B = 100 * 1000;
T = 2/B;
drtn = T*fs;
drtn = cast(drtn,'uint32');
%all = drtn*2
t_axis = linspace(0,2*T,drtn*2);
f_axis = linspace(-.5*fs ,.5*fs,2*drtn);
```

In this part we will generate square pulses and sinc signals “for zero ISI”

We will start with the square pulses :

## In time domain:

We generated square pulses with zeros and ones

```
%% Generate pulses
%first pulse
square1=[zeros(1,drtn) ones(1,drtn)];
%second pulse
square2=[ones(1,drtn) zeros(1,drtn)];

figure;
subplot(3,1,1)
plot(t_axis ,square1,'linewidth',2)
hold on
plot(t_axis ,square2,'linewidth',2)
title('The square pulses in time domain');
legend('Pulse 1','Pulse 2');
```

## In frequency domain :

By using FFT for the square pulse in time domain it turned to frequency domain

```
%% Frequancy response of square pulses
square1_freq = fftshift(fft(square1));
square2_freq = fftshift(fft(square2));

subplot(3,1,2)
plot(f_axis,abs(square1_freq));
hold on
plot(f_axis ,abs(square2_freq));
xlim([-1000000,1000000])
title('The square pulses in frequancy domain');
legend('Pulse 1','Pulse 2');
```

## Creating the band-limited channel :

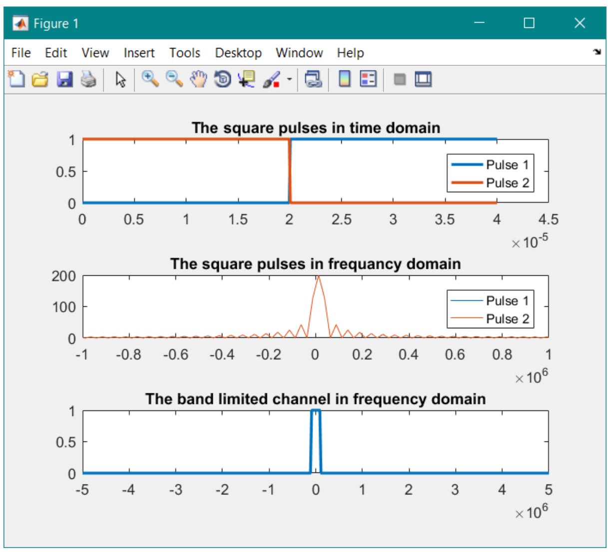
In this part we implement the band-limited channel by introducing some variables

- ValueOfOneSample which carries the time taken for each sample
- no_of_zeros which carries the number of zero samples of the filter where it is equal to half of the number of samples minus the band width of the channel filter

Finally we implement our channel by the ones and zeros functions to get the desired filter

```
%% Band limited channel
%frequency
drtn = cast(drtn,'double');
valueOfOneSample=2*drtn*(ts);
numb_of_zeros = (fs/2- B)*valueOfOneSample;
numb_of_zeros = cast(numb_of_zeros,'uint32');
limited_channel_freq = [zeros( 1, numb_of_zeros) (ones(1,2*drtn-2*numb_of_zeros))  zeros( 1, numb_of_zeros) ];

subplot(3,1,3)
plot(f_axis ,limited_channel_freq,'linewidth',2)
title('The band limited channel in frequency domain');
```

Plots of square pulses in time and frequency domain and the band-limited channel :

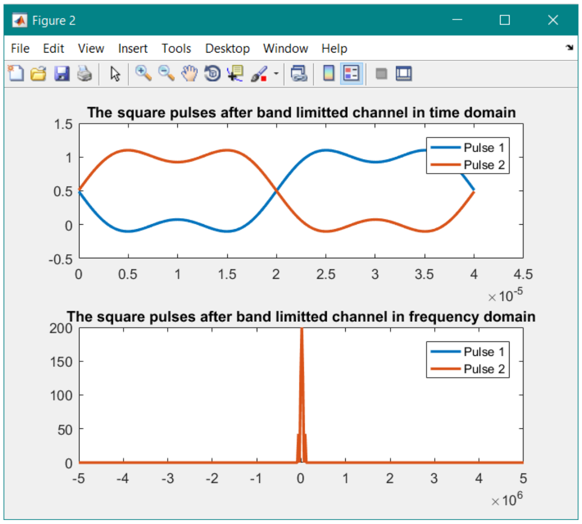



Next step, we pass the square wave in frequency domain into the channel and by using IFFT it converted to time domain :

```
%% Passing the pulses to the the band limitted channel
%frequency domain
out_square1_freq = square1_freq.* limited_channel_freq ;
out_square2_freq = square2_freq.* limited_channel_freq ;

%time domain
out_square1 = real(ifft(ifftshift(out_square1_freq )));
out_square2 = real(ifft(ifftshift(out_square2_freq))) ;

figure;
subplot(2,1,1)
plot(t_axis,out_square1,'linewidth',2);
hold on
plot(t_axis ,out_square2,'linewidth',2);
title('The square pulses after band limitted channel in time domain');
legend('Pulse 1','Pulse 2');
```

```
subplot(2,1,2)
plot(f_axis,abs(out_square1_freq),'linewidth',2);
hold on
plot(f_axis ,abs(out_square2_freq),'linewidth',2)
title('The square pulses after band limitted channel in frequency domain');
legend('Pulse 1','Pulse 2');
```

Plots after passing two square pulses through the band limited channel :




Adding additive white gaussian noise to square pulses:

```
%% Adding additive white gaussian noise to square pulses
%time domain
square1_noise = Computenoise(out_square1,drtn);
square2_noise = Computenoise(out_square2,drtn);

figure;
subplot(2,1,1)
plot(t_axis,square1_noise,'linewidth',2);
hold on
plot(t_axis,square2_noise,'linewidth',2);
title('The square pulses after band limitted channel with AWGN in time domain');
legend('Pulse 1','Pulse 2');

%frequency domain
square1_noise_freq = fftshift(fft(square1_noise));
square2_noise_freq = fftshift(fft(square2_noise));
```

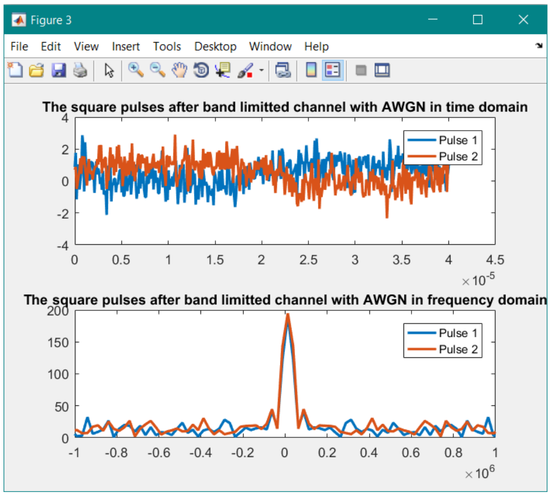
```

subplot(2,1,2)
plot(f_axis,abs(square1_noise_freq),'linewidth',2);
hold on
plot(f_axis,abs(square2_noise_freq),'linewidth',2);
xlim([-10^6,10^6]);
title('The square pulses after band limitted channel with AWGN in frequency domain');
legend('Pulse 1','Pulse 2');
```

We create a function for the AWGN and called it Computenoise:

To add noise to the pulses

```
function AWGN = Computenoise(sig,drtn)
No=1;
AWGN= sig + sqrt((No/2))*randn(1,length(sig));
```

Plots after adding AWGN to the pulses in time and frequency domain:




## For no ISI we generated sinc signal :

## Explain what is the mathematical criterion that ensures no ISI.

For nullifying the ISI terms, with an impulse of unit value applied at t=0 to the combined filters h(t), the samples of the h(t) at the output of the filter combination should be 1 at the sampling instant t=0 and zero at all other sampling instants kT_b (k !=0).

## Describe one or more pulse shapes that ensure that such condition is met

- the signal that avoids ISI with the least amount of bandwidth is a sinc pulse of bandwidth F/ 2.
- any triangular waveform („Δ" function) with a width that is less than 2Tb will also satisfy the condition.
- raised–cosine pulses that satisfy the Nyquist criterion and require slightly larger bandwidth than what a sinc pulse (which requires the minimum bandwidth ever) requires.

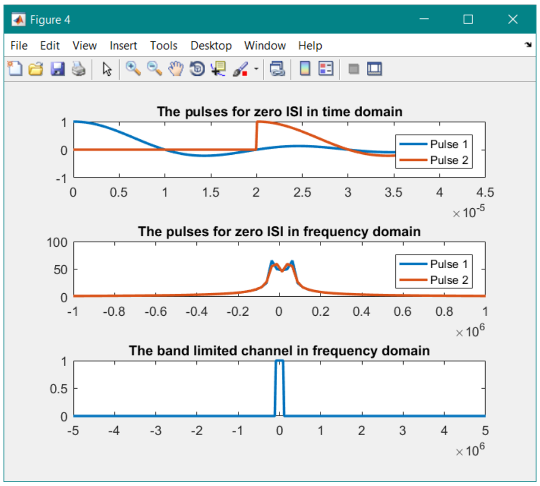
First sinc signal starts from zero and second starts from 201 after a vector of zeros in time domain

```
%% For zero ISI generating pulses
%sinc or triangle
%time domain
sinc_siganl = sinc((t_axis.*B));
n=length(sinc_siganl);
sinc_siganl1= sinc_siganl;
sinc_siganl2= [zeros(1,200) sinc_siganl(1:n-200) ];

figure;
subplot(3,1,1)
plot(t_axis , sinc_siganl1,'linewidth',2 )
hold on
plot(t_axis , sinc_siganl2,'linewidth',2 )
title('The pulses for zero ISI in time domain');
legend('Pulse 1','Pulse 2');
```

And in frequency domain we converted it by FFT

```
%frequancy domain
sinc_siganl1_freq = fftshift(fft(sinc_siganl1));
sinc_siganl2_freq = fftshift(fft(sinc_siganl2));

subplot(3,1,2)
plot(f_axis , abs(sinc_siganl1_freq),'linewidth',2 )
hold on
plot(f_axis , abs(sinc_siganl2_freq),'linewidth',2 )
xlim([-1000000,1000000]);
title('The pulses for zero ISI in frequency domain');
legend('Pulse 1','Pulse 2');

subplot(3,1,3)
plot(f_axis ,limited_channel_freq,'linewidth',2)
title('The band limited channel in frequency domain');
```

Plots of both sinc signals in time and frequency domain

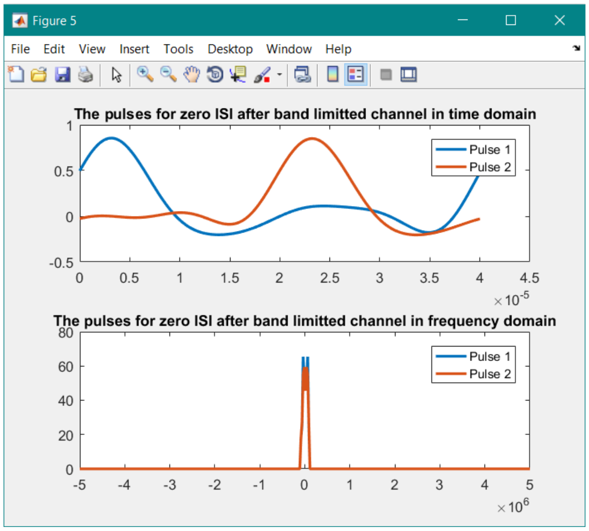



And after passing both of sinc signals through the band limited channel in frequency domain and time domain

```
%% After band channel for zero ISI
%frequency domain
out_sinc_siganl1_freq = sinc_siganl1_freq .* limited_channel_freq;
out_sinc_siganl2_freq = sinc_siganl2_freq .* limited_channel_freq;
%time domain
out_sinc_siganl1 = real(ifft(ifftshift(out_sinc_siganl1_freq )));
out_sinc_siganl2 = real(ifft(ifftshift(out_sinc_siganl2_freq)) ;

figure;
subplot(2,1,1)
plot(t_axis,out_sinc_siganl1,'linewidth',2);
hold on
plot(t_axis ,out_sinc_siganl2,'linewidth',2);
title('The pulses for zero ISI after band limitted channel in time domain');
legend('Pulse 1','Pulse 2');
```

```

subplot(2,1,2)
plot(f_axis,abs(out_sinc_siganl1_freq),'linewidth',2);
hold on
plot(f_axis ,abs(out_sinc_siganl2_freq),'linewidth',2)
title('The pulses for zero ISI after band limitted channel in frequency domain');
legend('Pulse 1','Pulse 2');
```

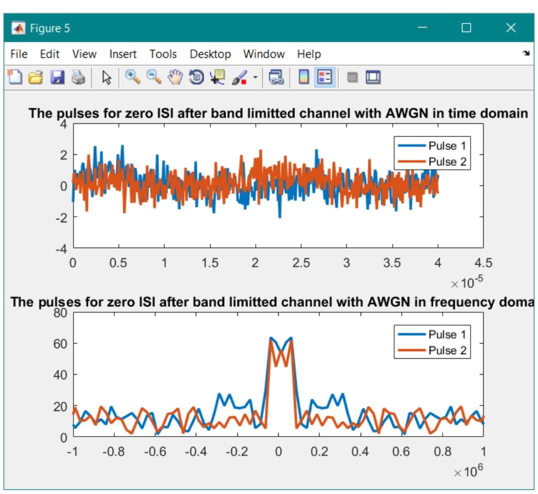
Plots after passing both of sinc signals through the band limited channel in frequency domain and time domain




passing both of sinc signals through the AWGN channel in frequency domain and time domain

```
%% Adding additive white gaussian noise to pulses for zero ISI
%time domain
pulse1_noise = Computenoise(out_sinc_siganl1,drtn);
pulse2_noise = Computenoise(out_sinc_siganl2,drtn);

figure;
subplot(2,1,1)
plot(t_axis,pulse1_noise,'linewidth',2);
hold on
plot(t_axis,pulse2_noise,'linewidth',2);
title('The pulses for zero ISI after band limitted channel with AWGN in time domain');
legend('Pulse 1','Pulse 2');

%frequency domain
pulse1_noise_freq = fftshift(fft(pulse1_noise));
pulse2_noise_freq = fftshift(fft(pulse2_noise));

subplot(2,1,2)
plot(f_axis,abs(pulse1_noise_freq),'linewidth',2);
hold on
plot(f_axis,abs(pulse2_noise_freq),'linewidth',2);
xlim([-10^6,10^6]);
title('The pulses for zero ISI after band limitted channel with AWGN in frequency domain');
legend('Pulse 1','Pulse 2');
```

Plots after passing both of sinc signals through the AWGN channel in frequency domain and time domain



To compute BER we created function and called it ComputeBER :

It compares the bit in the generated sinc signal before noise and the same bit after passing it through the noise.

```
function BER = ComputeBER(bit_seq,rec_bit_seq)
E=0;
N = length ( bit_seq );
for i = 1:1: N
    if  bit_seq (i) ~= rec_bit_seq (i)
        E=E+1;
    end
end
BER = E / N ;
end
```

## In the main code :

```
%% Computing BER
%BER for the pulses with zero ISI
fprintf('BER for the pulses with zero ISI is : ');
AWGN_out3 = out(pulse1_noise,drtn);
AWGN_out4 = out(pulse2_noise,drtn);
BER_pulse1 = ComputeBER(AWGN_out3,sinc_siganl1)
BER_pulse2 = ComputeBER(AWGN_out4,sinc_siganl2)
```

We created a function called out to convert the signal bits into zeros and ones “ any value greater than or equal zero will be one , and other values will be zero “

```
function AWGN_out = out(sgn,drtn)
AWGN_out=zeros(size(sgn));
n=1;
for i= drtn/2:drtn:length(sgn)
 if( sgn(1,i)>= 0 )
 AWGN_out(n)=1;
 else
 AWGN_out(n)=0;
 end
 n=n+1;
end
```

## And BER will be :

```
New to MATLAB? See resources for Getting Started.
BER for the pulses with zero ISI is :
BER_pulse1 =

     1


BER_pulse2 =

    0.5025

fx >>
```

```
pulse2_no
pulse2_no
sinc_sigan
sinc_sigan
sinc_sigan
sinc_sigan
sinc_sigan
square1
square1_fr
square1_n
square1_n
square2
square2_fr
square2_n
```