Digital Lab 2 Project

Names	ID
Mohamed Hassan Hassan Khalil	18011433
Mohamed Nasser Mohamed Amr	18011633
Ali Ramadan Ramadan Younes	18011079
Mahmoud Ahmed Ali Ahmed Omara	18011657
Mohamed Emad Younes yehia	18012503
Abanob Morkos Gad Elsaid	18010004
Abdelrahman Saber Mohamed	17011037
Mohamed Abdelsalam Abdelmotleb	18011535

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

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

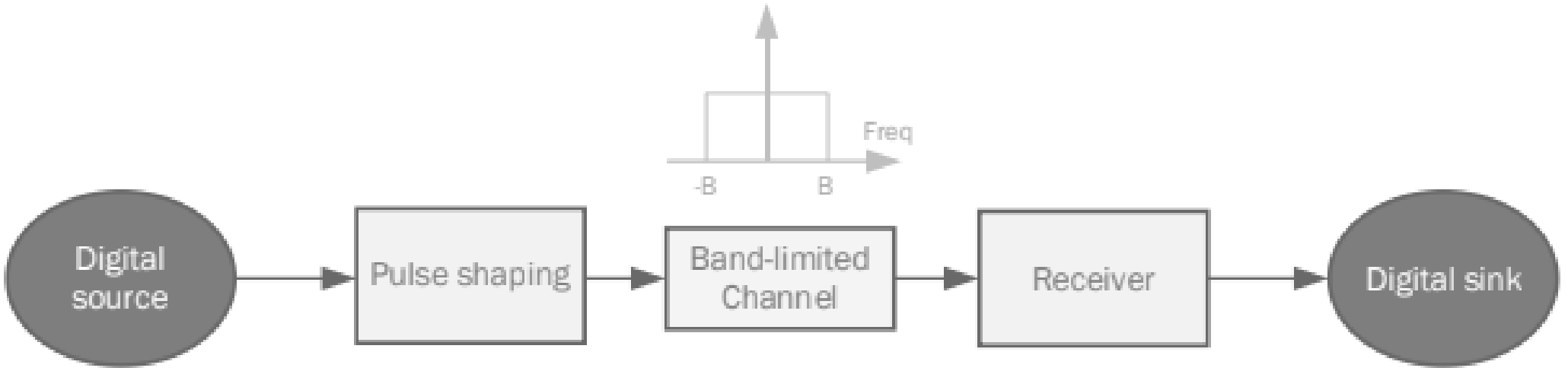


Figure 1 Communication system with a band-limited channel with bandwidth B.

***Guidelines:** The first thing you need to show here is the effect of a band-limited flat channel on the square signal. You need to create a band-limited channel such as the one in Figure 1, with a band $B = 100\ kHz$.*

Then generate a square pulse of duration $T = 2/B$, pass it through the filter, and then look at the output.

You need here to show the signal before and after the channel, both in time and in frequency.

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

***Guidelines:** The second thing you need to show is the effect of two consecutive square signals as they pass through the channel. Consider the same channel and the same square pulse duration as before. The plots you need to show here are in time domain only. Namely,*

1. *Show two plots of the first square pulse: one before it passes through the channel, and one after.*
2. *On top of the two previous plots, show similar plots for the second square pulse. Plot this pulse in the two plots using a different color, so that the shapes of the two pulses are distinguishable on the plots.*
3. *Note that you will have to pass the two squares separately, i.e., you cannot create the two pulses together in the same vector and pass that into the channel. If you do it this way, you won't be able to clearly distinguish the two pulses.*

The procedure that you need to follow to generate the plots required above are shown in Figure 2.

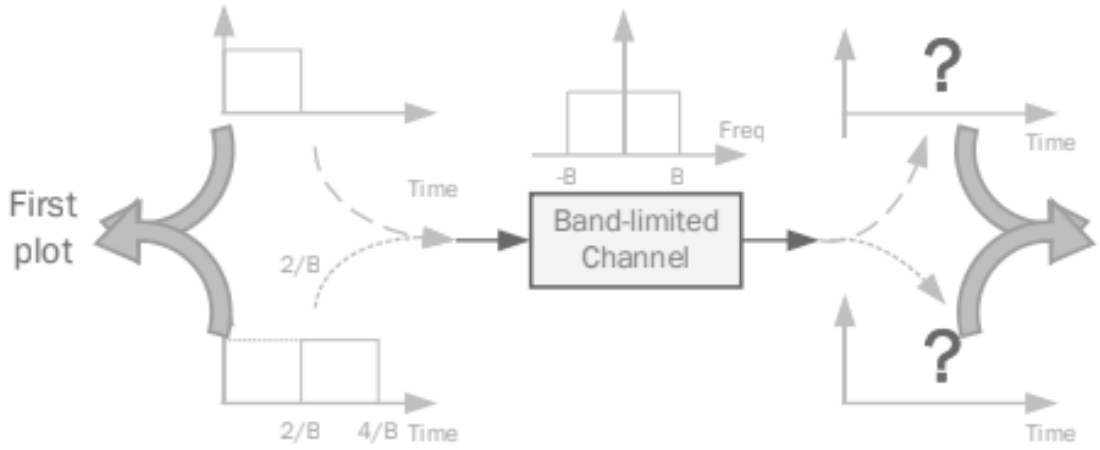


Figure 2 How to generate the two plots described above.

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

1. *Explain what is the mathematical criterion that ensures no ISI.*
2. *Describe one or more pulse shapes that ensure that such condition is met.*
    a. *For one or more of these pulse shapes, show (at least for one of those pulse shapes) plots of the pulse shape before and after the channel, both in time and frequency.*

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

The Code and Results

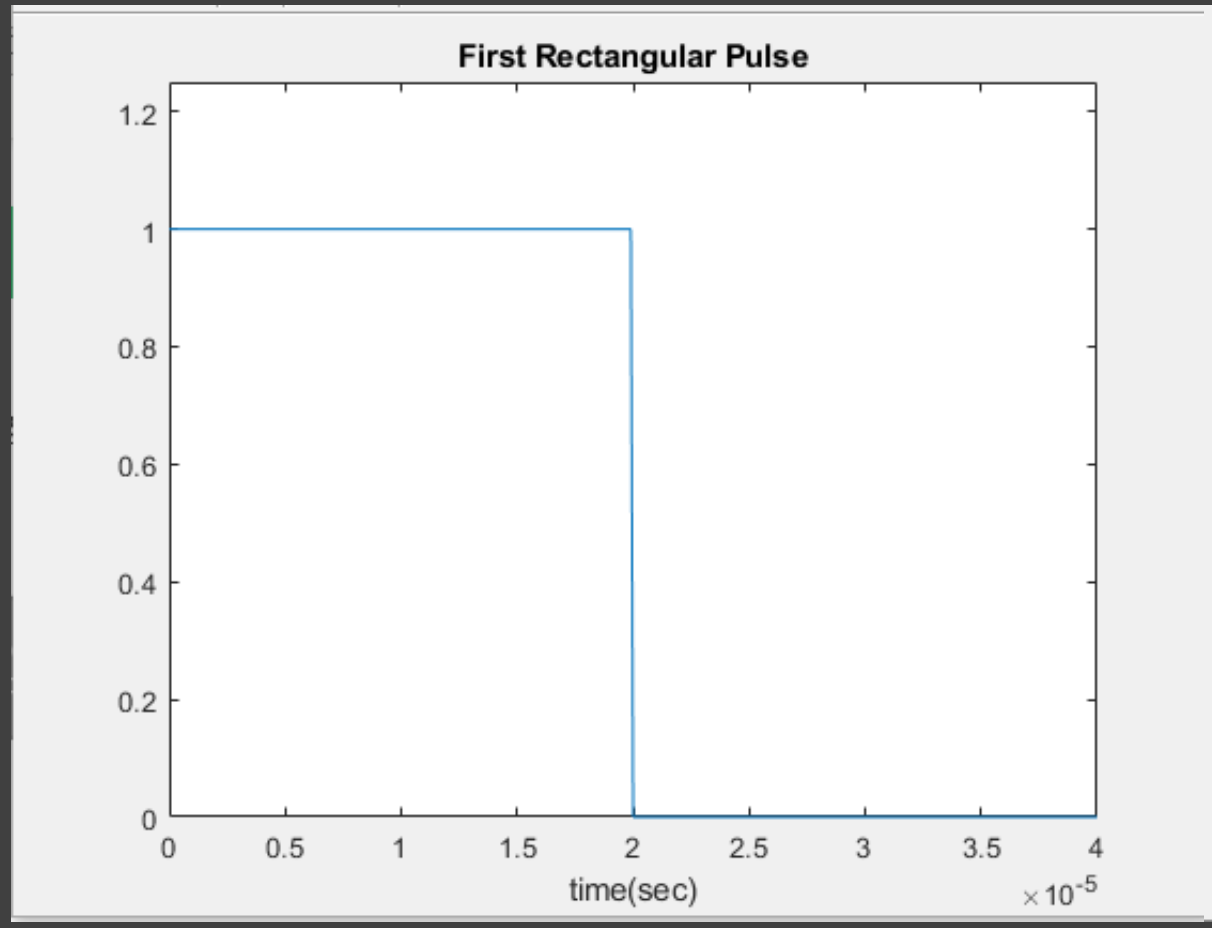
```
clear all
clc
close all
%Intro to Digital communication Lab 2
%Part 1
%---------------------------------------
%%Basic Parameters
%---------------------------------------
B = 1e5;     %B = bandwidth
T = 2/B;     %Pulse Duration
fs = 1e7;    %Sampling Frequency
Ts = 1/fs;   %Sampling time
N = ((round(T*fs))*50)-1;
```

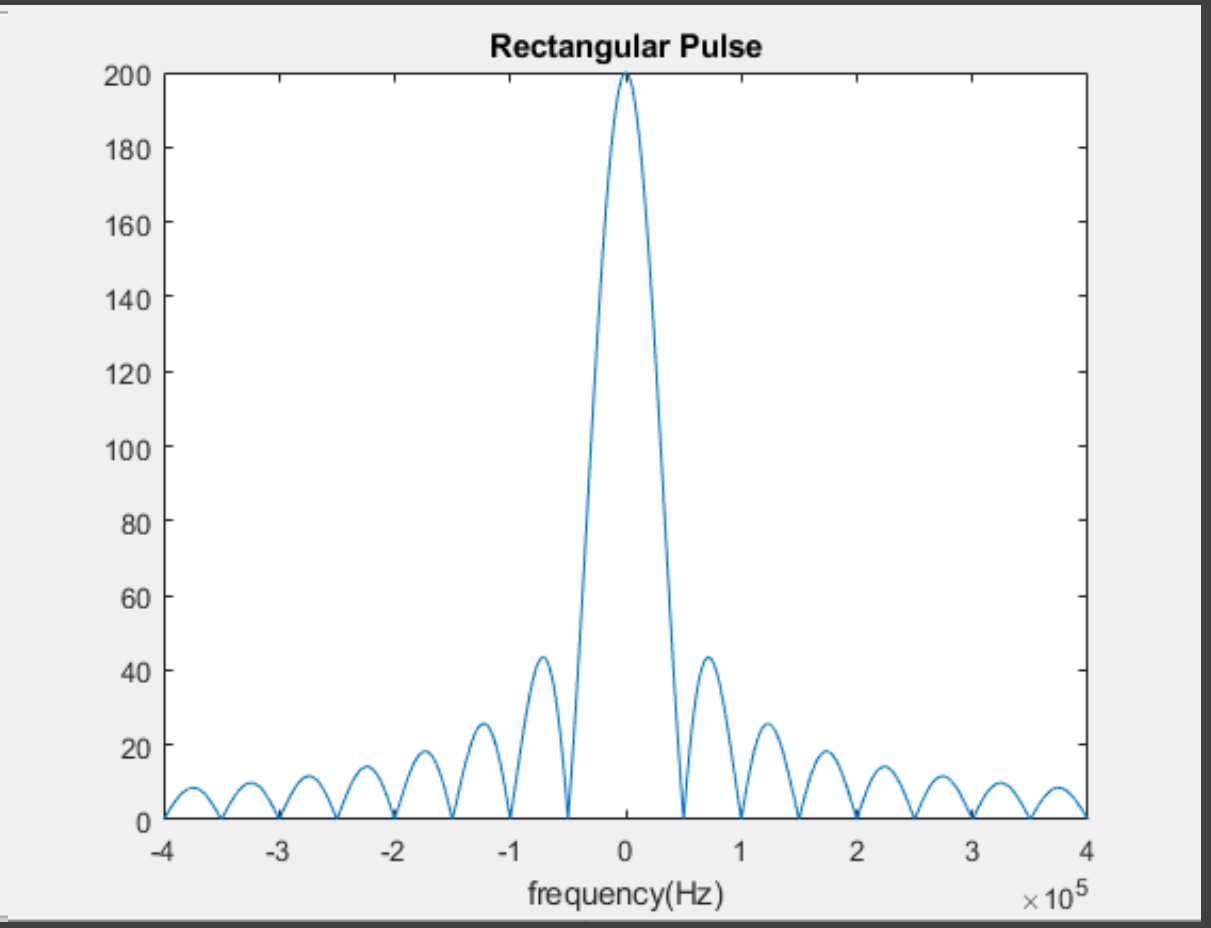
```
%% Section 1
%---------------------------------------
%Generate Pulse Signal

%%%Input Time Domain%%%
bits = ones(1,round(T*fs));
zero = zeros(1,N - length(bits));
sq_pulsel = [bits zero];
t = (0 : N-1)*Ts;
figure;
plot(t,sq_pulsel)
xlabel('time(sec)')
title('First Rectangular Pulse')
xlim([0 0.4e-4])
ylim([0 1.25])
```

```
%%%Input Frequency Domain%%%
sq_pul_f = (fftshift(fft(sq_pulsel)));
f = linspace(-fs/2 , fs/2 , N);
figure;
plot(f,(abs(sq_pul_f)))
xlabel('frequency(Hz)')
title('Rectangular Pulse')
xlim([-4*B 4*B])
```

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

%%%%%%%%%Pulse in Time and Frequency Domains%%%%%%%%%%

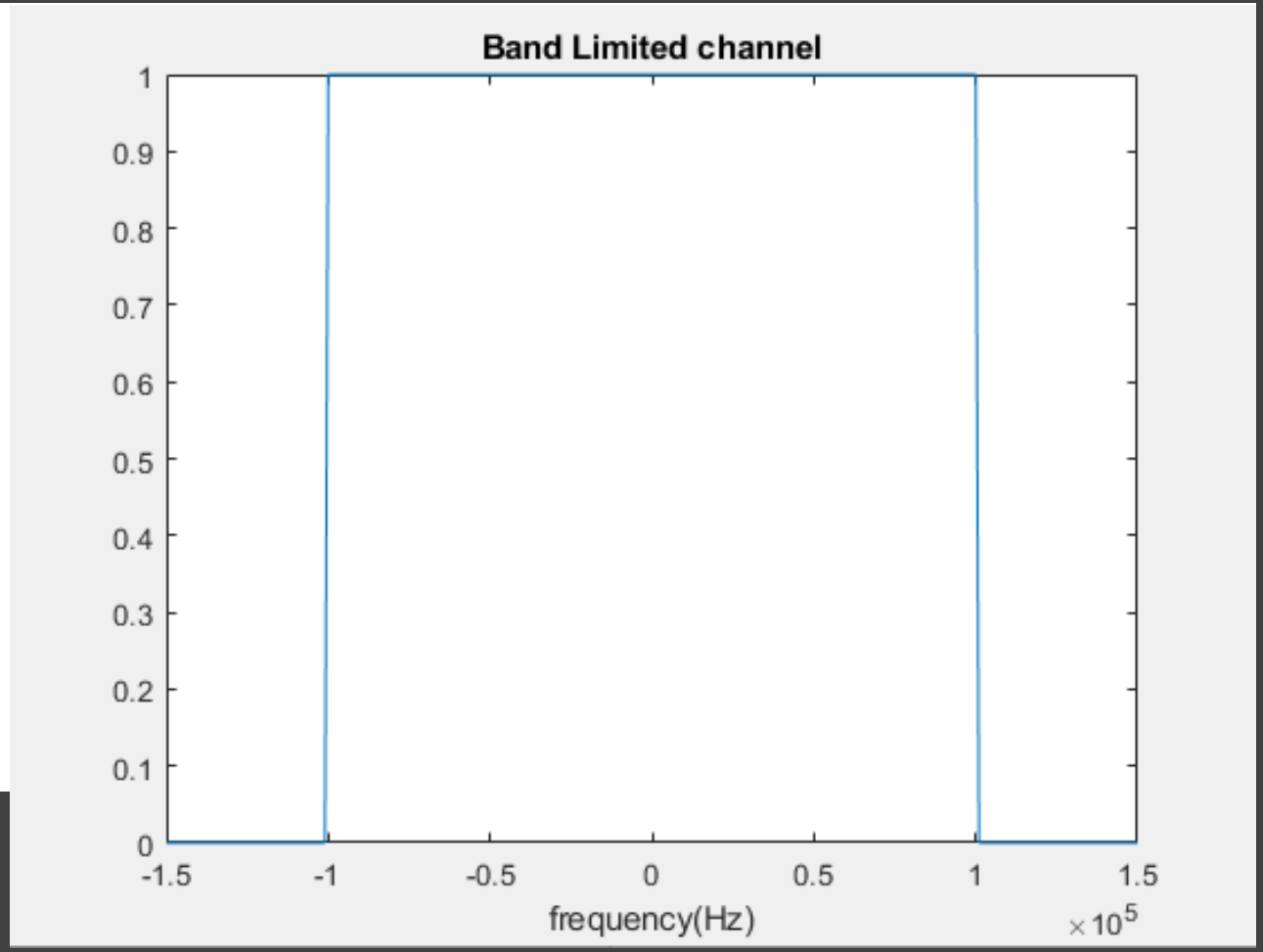




Input Signal NRZ with 1/Tb High level and 1/Tb low level

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

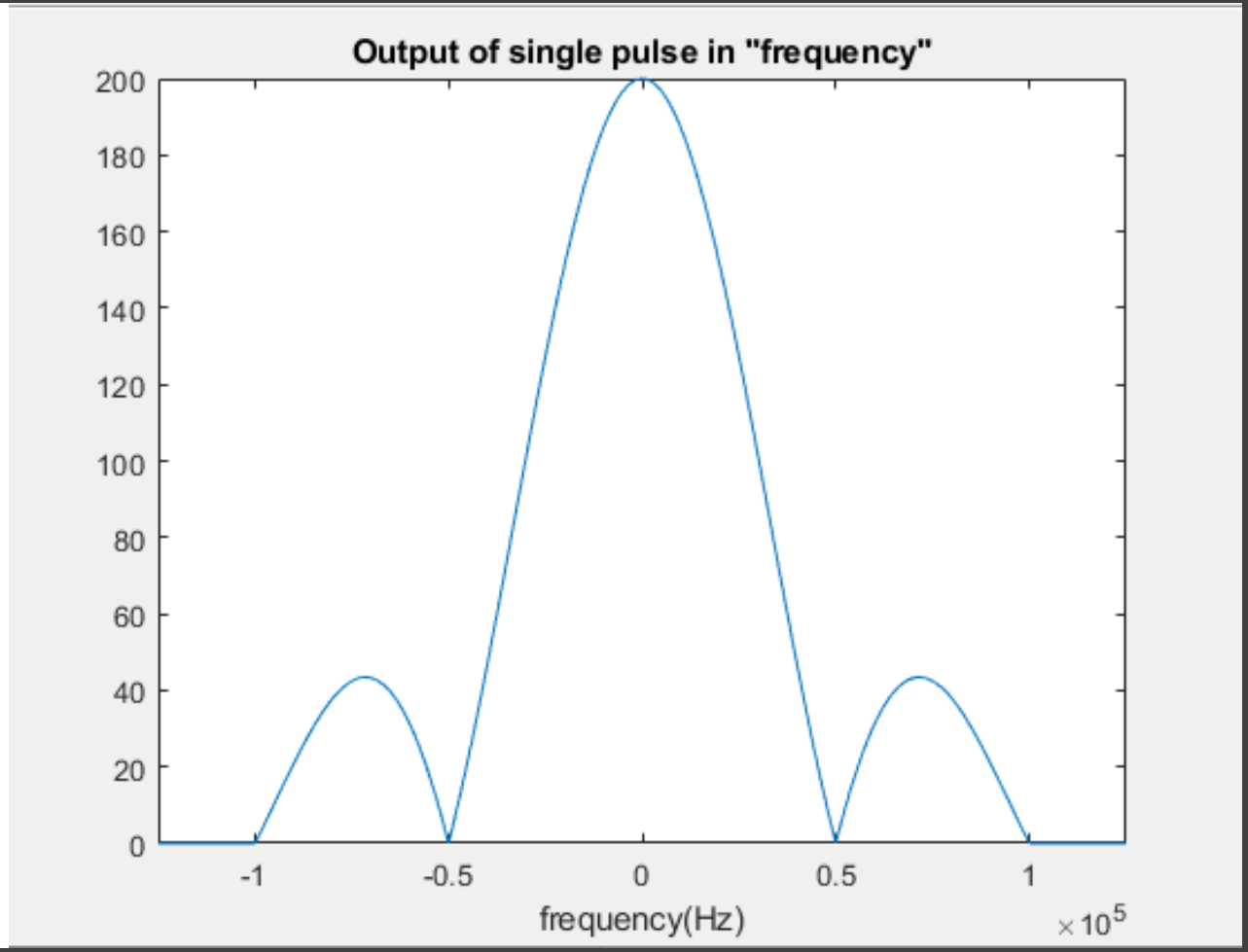
```
%% Section 2
%-------------------------------------
%%%Channel Creation%%%
channel = sq_pul_f;
zeros_p1 = round(N/2) - round(N*(B*Ts));
zeros_p2 = round(N/2) + round(N*(B*Ts));
channel([1:(zeros_p1) (zeros_p2):end])=0;
channel([(zeros_p1):(zeros_p2)]) = 1;
figure;
plot(f,channel)
xlabel('frequency(Hz)')
title('Band Limited channel')
xlim([-1.5*B 1.5*B])
```



%%%Band Limited Channel ranged from –B to B Bandwidth%%%

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

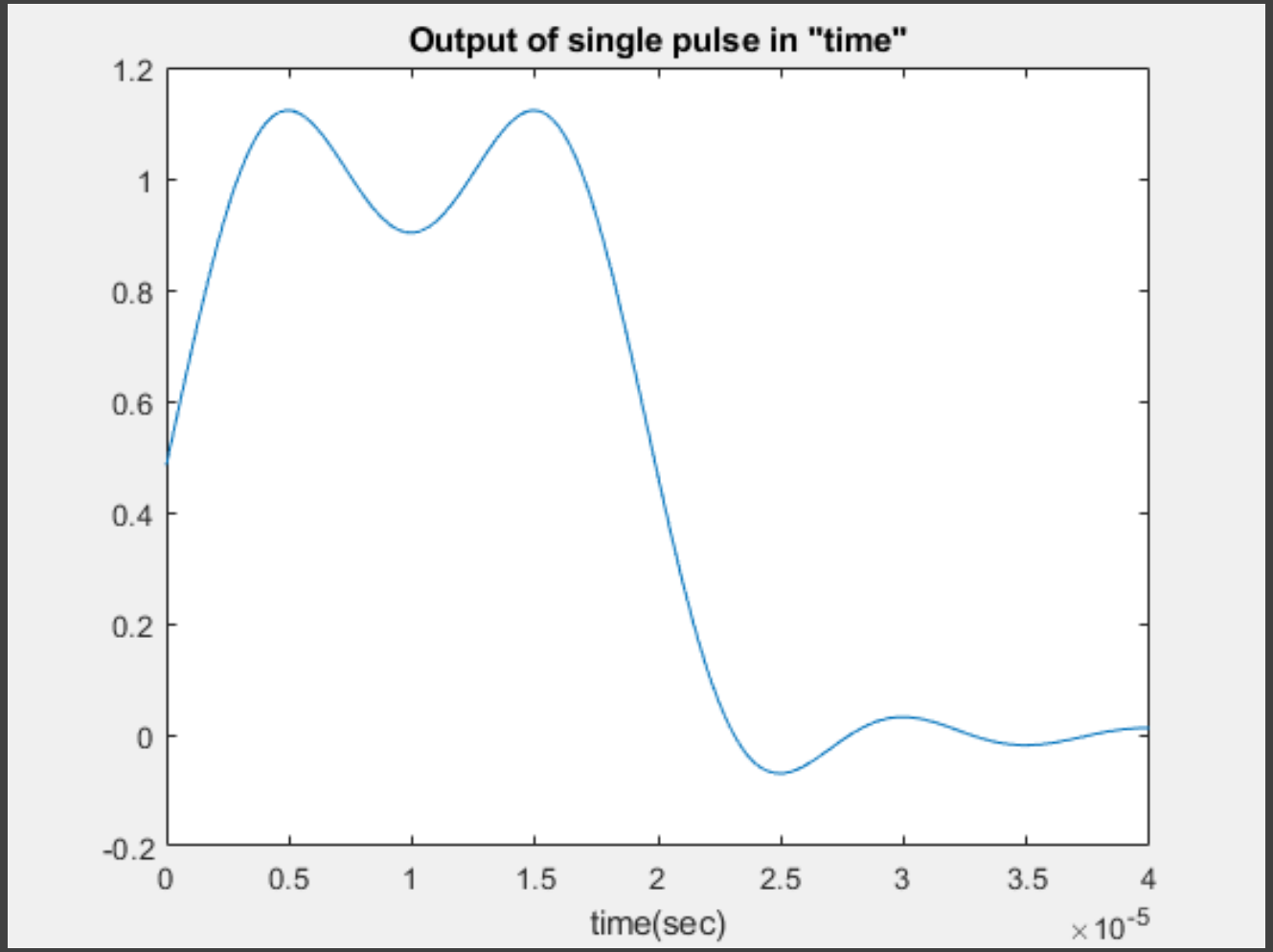
```
%% Section 3
%--------------------------------------
%%%Output of the Channel%%%
outl_f = sq_pul_f.*channel;


%%%Output Frequecny Domain%%%
figure;
plot(f,abs(outl_f))
xlabel('frequency(Hz)')
title('Output of single pulse in "frequency"')
xlim([-1.25*B 1.25*B])


%%%Output Time Domain%%%
outl = ifft(ifftshift(outl_f));
figure;
plot(t,outl)
xlabel('time(sec)')
title('Output of single pulse in "time"')
xlim([0 0.4e-4])
```



%%%After passing the Input through Band Limited channel Sinc function limited at – B and B%%%

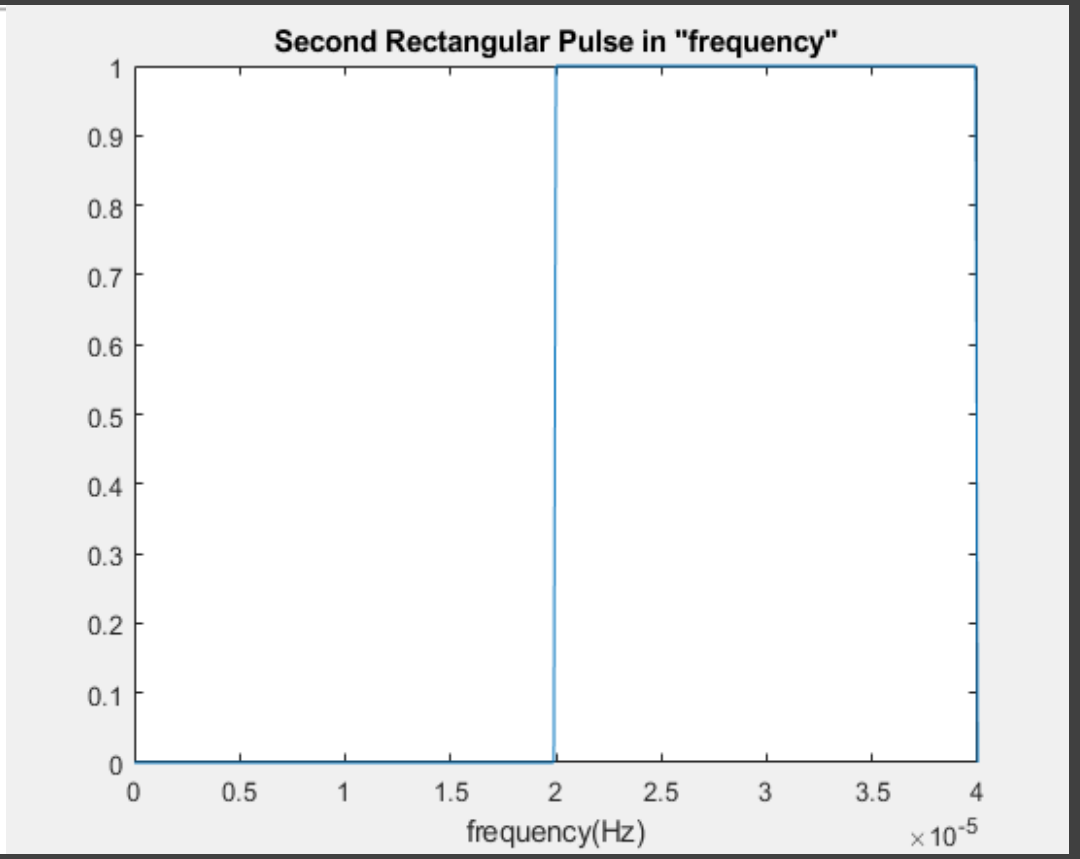
Part 1 : *Inter-Symbol Interference due to band-limited channels*

%%%Due to Channel the Message was distorted with ripples (in Time Domain)%%%



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

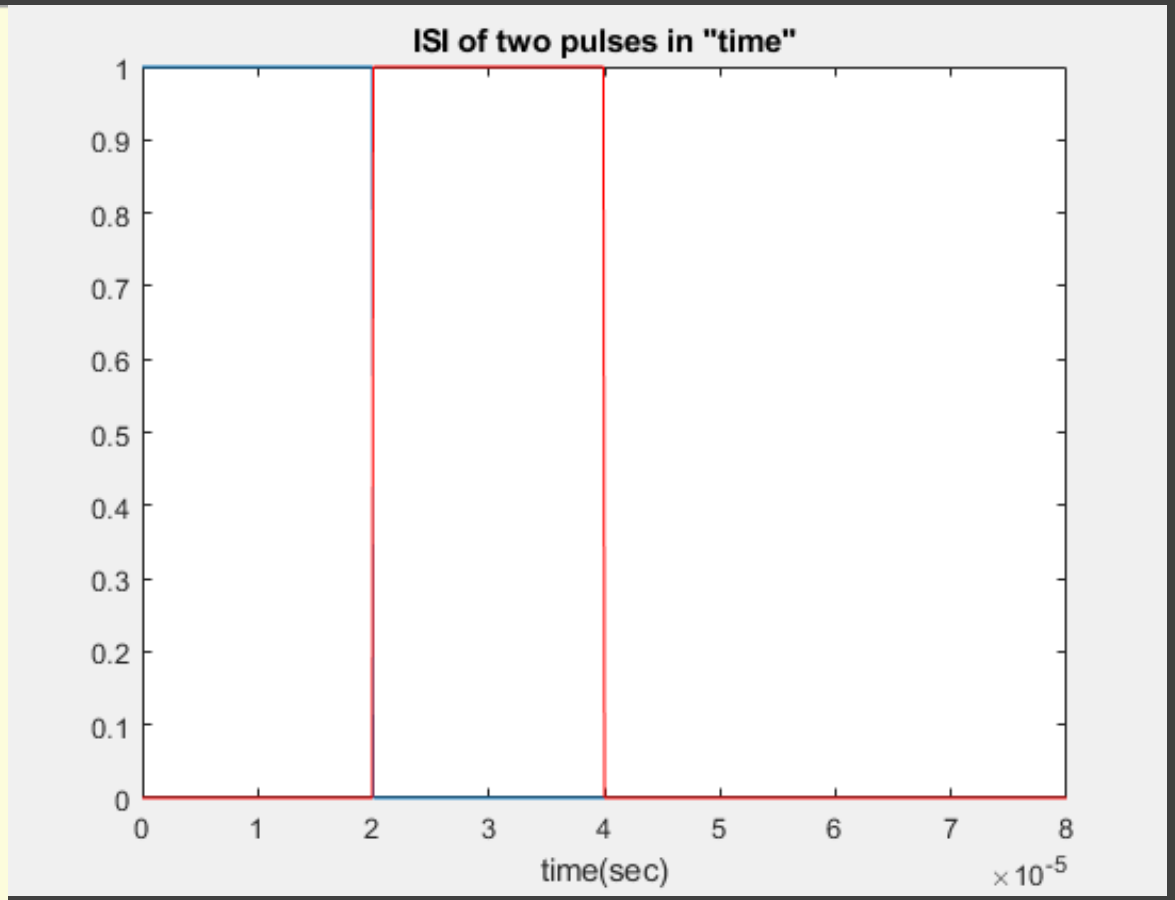
```
%% Section 4
%-------------------------------------
sq_pulse2 = circshift(sq_pulse1,round(T*fs));
sq_pul_f2 = fftshift(fft(sq_pulse2));
figure;
plot(t,sq_pulse2)
xlabel('frequency(Hz)')
title('Second Rectangular Pulse in "frequency"')
xlim([0 0.4e-4])
figure;
plot(t,sq_pulse1)
hold on
plot(t,sq_pulse2,'r')
xlabel('time(sec)')
title('ISI of two pulses in "time"')
xlim([0 0.8e-4])
```



%%%Generation of the Second Pulse which follows the first one immediately (in Frequency domain)%%%

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

```
%% Section 5
%-------------------------------------
%%%Output Frequecny Domain%%%
out2_f = sq_pul_f2.*channel;
figure;
plot(f,out1_f)
hold on
plot(f,out2_f,'r')
xlabel('frequency(Hz)')
title('ISI of Output  in "frequency"')
xlim([-1.25*B 1.25*B])


%%%Output Time Domain%%%
out2 = ifft(ifftshift(out2_f));
sum_2_pulses = out1 + out2;
figure;
plot(t,out1)
hold on
plot(t,out2,'r')
hold on
plot(t,sum_2_pulses,'-.','color',[0 1 0])
xlabel('time(sec)')
title('ISI of Output in "time"')
xlim([0 1e-4])
```

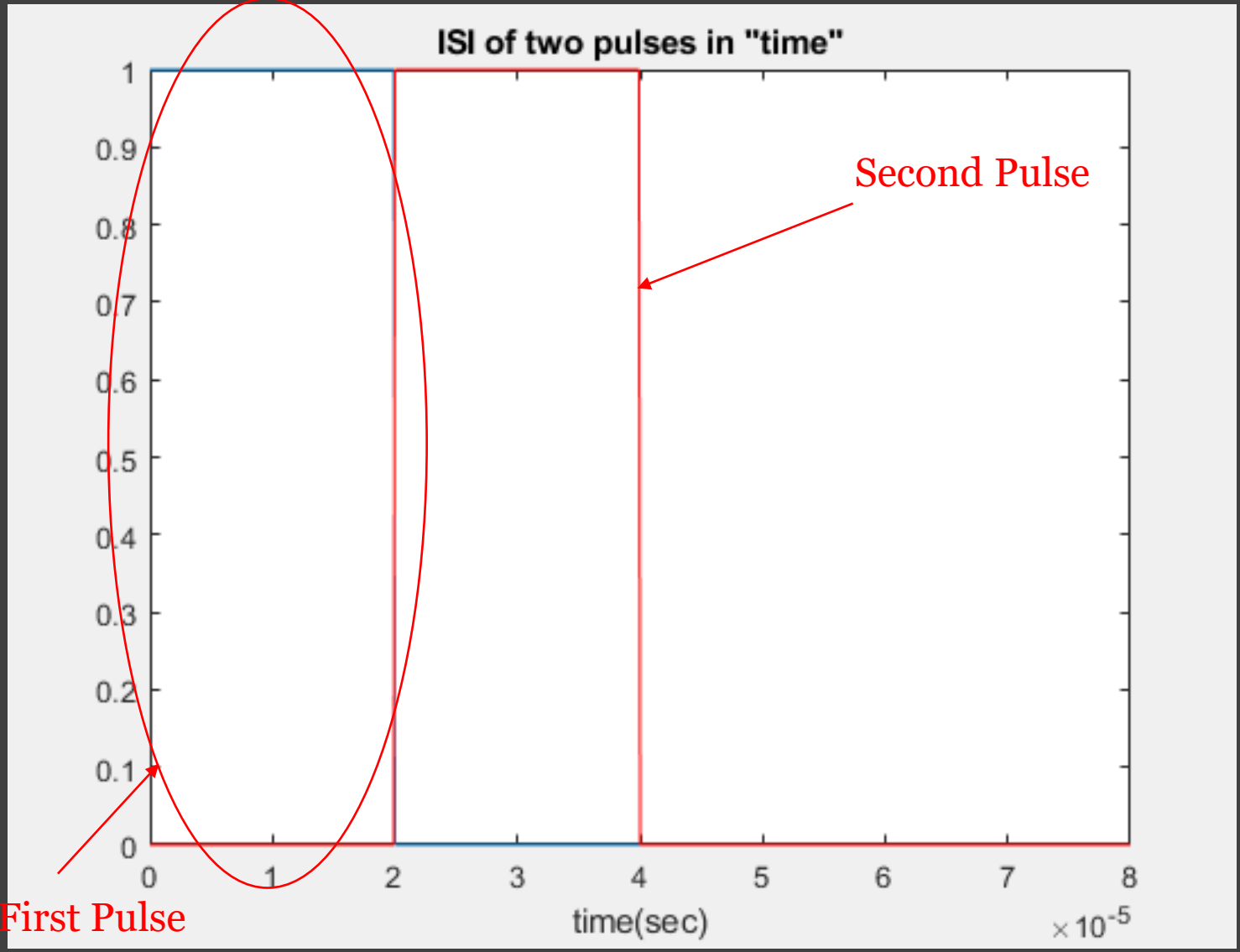


%%%Plotting The Two Pulses to make future operation on them (in time domain)%%%

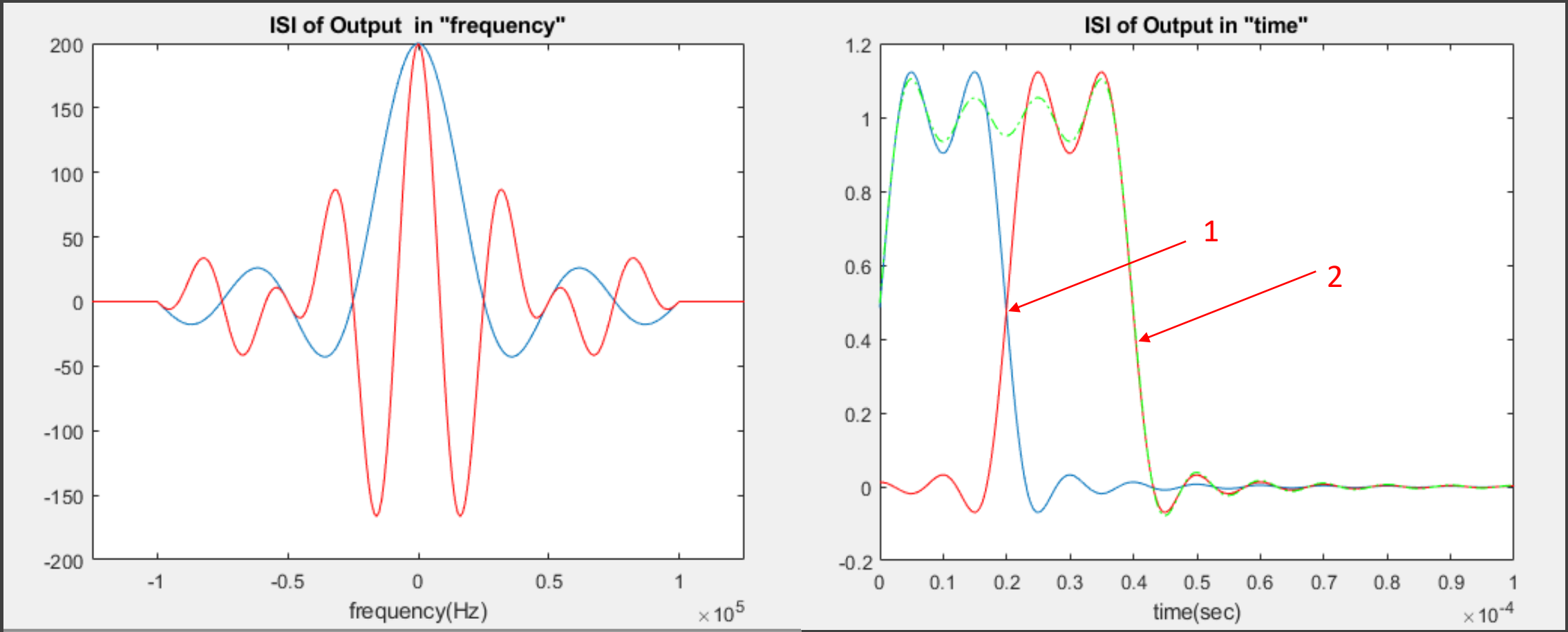
Part 1 : *Inter-Symbol Interference due to band-limited channels*

%%%Plotting The Two Pulses to make future operation on them (in time domain)

To be clearer%%%



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

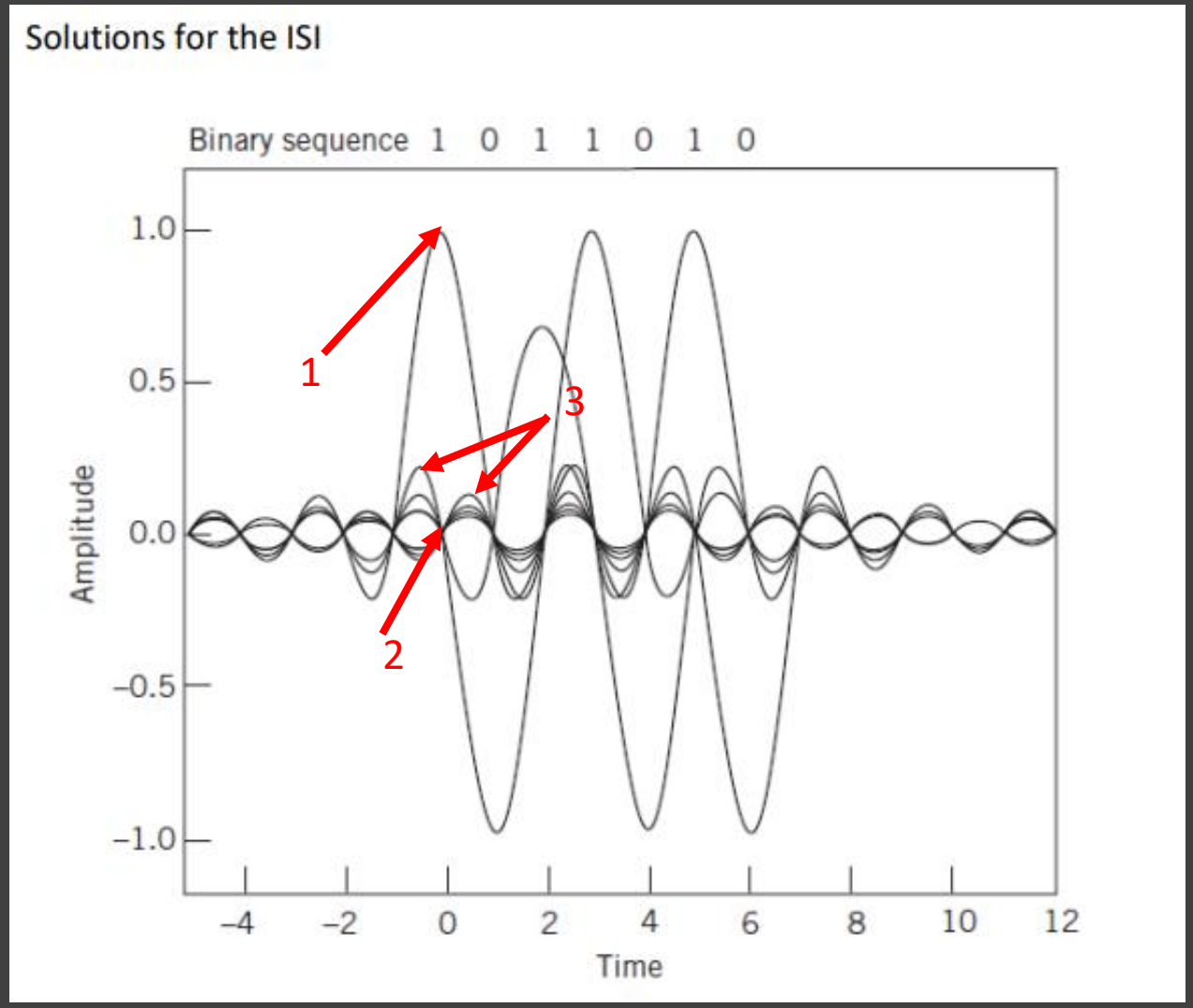


%%%Inter-symbol Interference Occurrence , here what we need?
We need that at point 1 for example to be the value of first pulse only and value of second pulse equals zero while at point 2 the value of second pulse only and value of first pulse equals zero but in reality, ISI occurred and resulted that many points have value of the two pulses.

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

%%%From the Solutions of ISI %%%

First method : If the generated signal is sinc function instead of square pulse where at the values shown(no.1) for specific sinc function there are zeros (no.2) for the other signals actually there are some very small values of other signals(no.3), but sill very low ISI compared to square pulse signal



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

a) Ideal Nyquist

a. Time

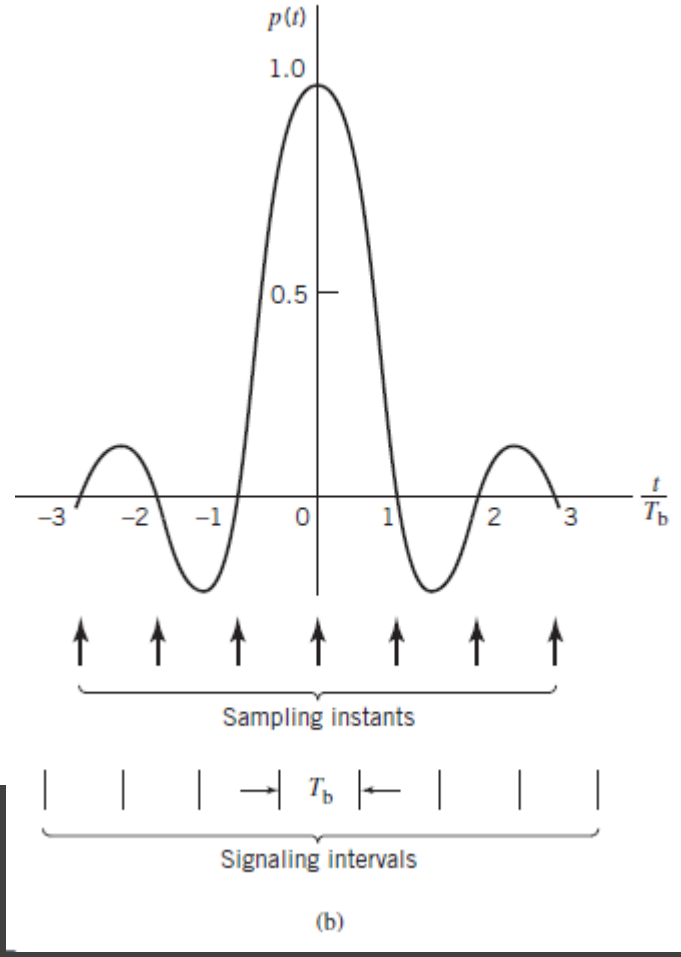
$$p(t) = \frac{\sin(2\pi Wt)}{2\pi Wt}$$

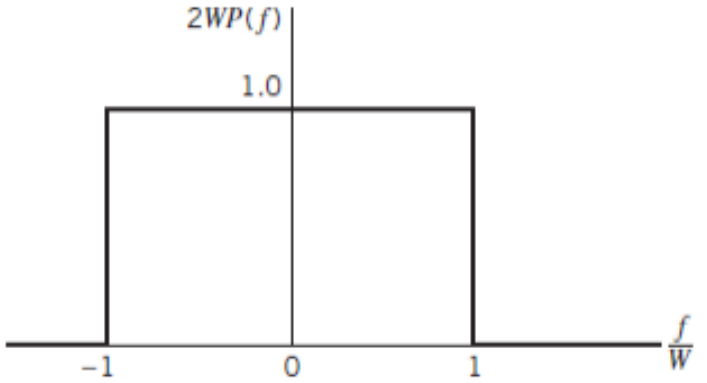
$$= \text{sinc}(2Wt)$$

b. Frequency

$$P(f) = \begin{cases} \frac{1}{2W}, & -W < f < W \\ 0, & |f| > W \end{cases}$$

$$= \frac{1}{2W}\text{rect}\left(\frac{f}{2W}\right)$$





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

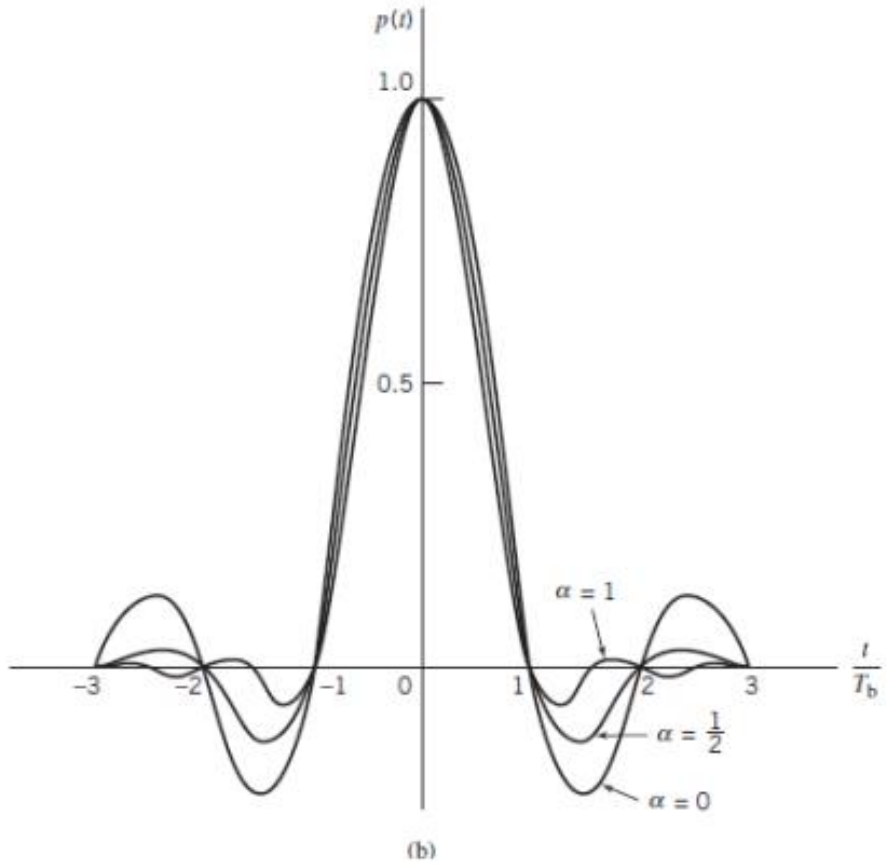
**Second method :
Same idea as sinc function, but the difference thing that we multiplied by factor which has variable (alpha) = roll off factor, this value @alpha = 0 it will be the (First method), but @alpha = 1 then ripples will vanish faster in which this method is much better than first method where the very small values which interfaces will be nearly equal to zero
Also there are additional NULLS appeared which is much better.

b) Raised Cosine

a. Time

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

(a)

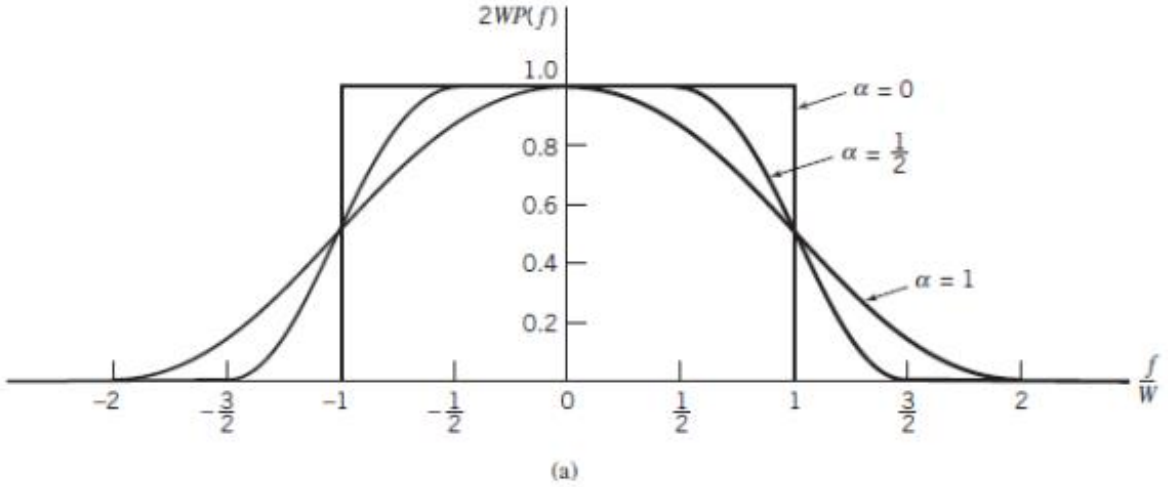


(b)

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

b. Frequency

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



(a)

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

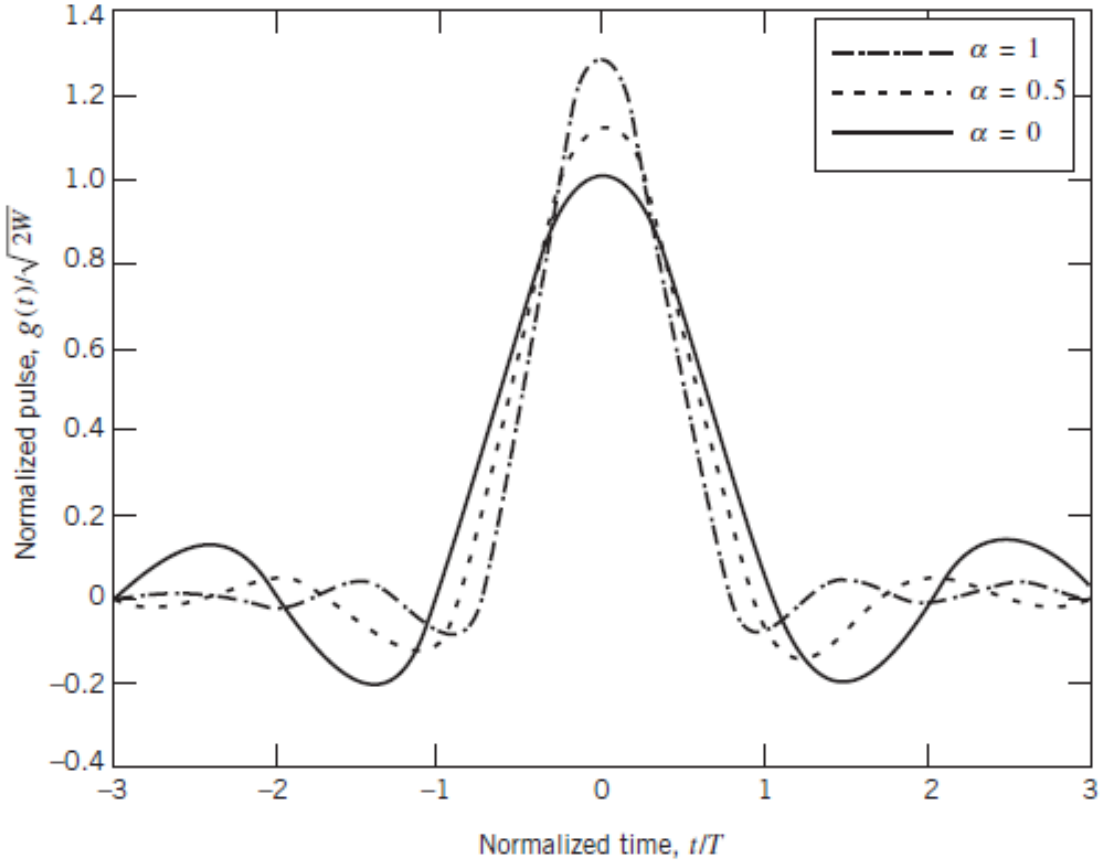
**Third method :
improve spectrum efficiency by more than 75% , but roll off factor can change the amplitude of the Signal in addition to the ripple decaying

(a)

c) Square Root Raised Cosine

a. Time

$$g(t) = \frac{\sqrt{2W}}{1-(8\alpha Wt)^2}\left\{\frac{\sin[2\pi W(1-\alpha)t]}{2\pi Wt} + \frac{4\alpha}{\pi}\cos[2\pi W(1+\alpha)t]\right\}$$

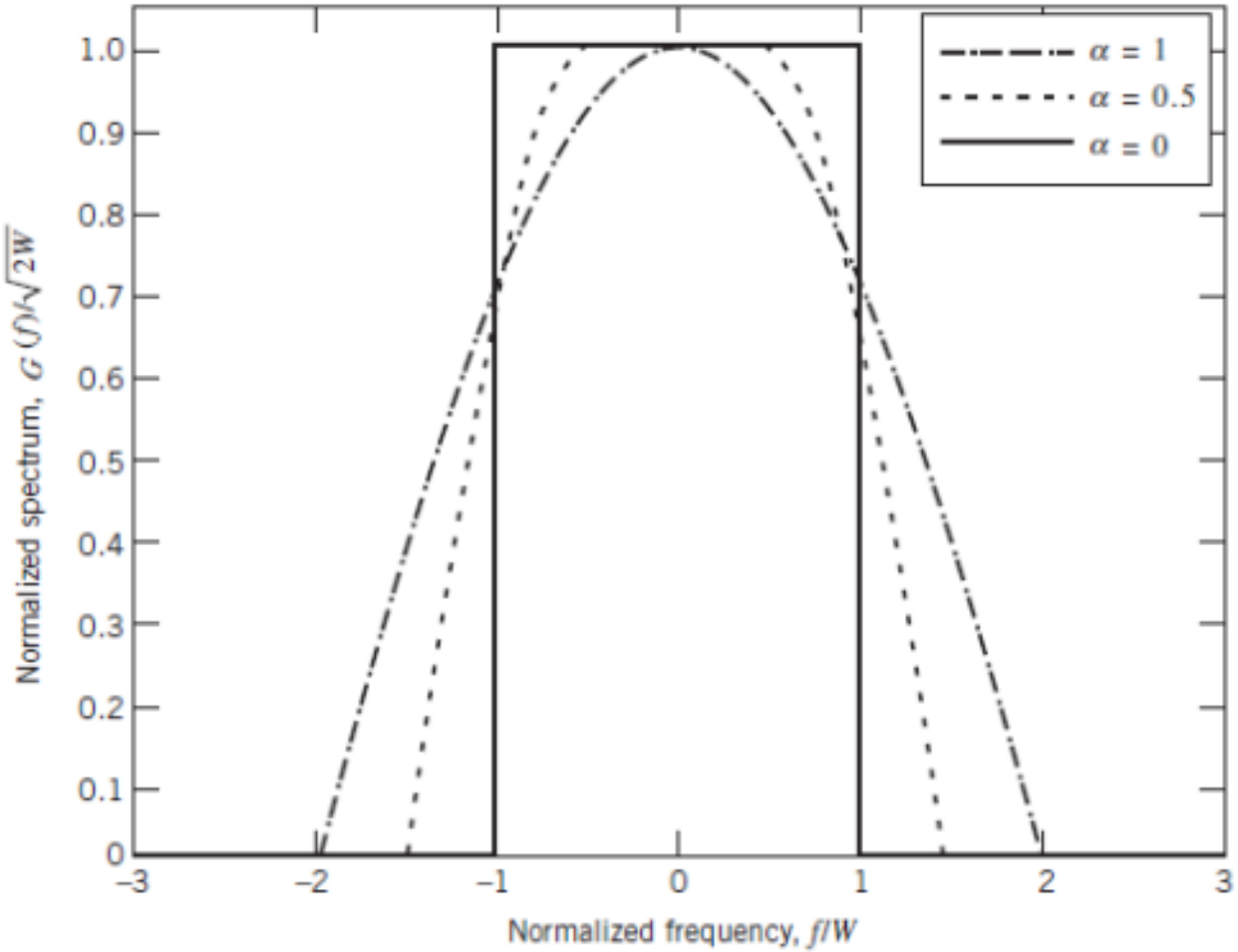


Part 1 : *Inter-Symbol Interference due to band-limited channels*

b. Frequency

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

At roll off factor approaches to value 1 the curve approaches to cosine wave form



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

Part 2 : *Inter-Symbol Interference due to Multi-Path Channels*

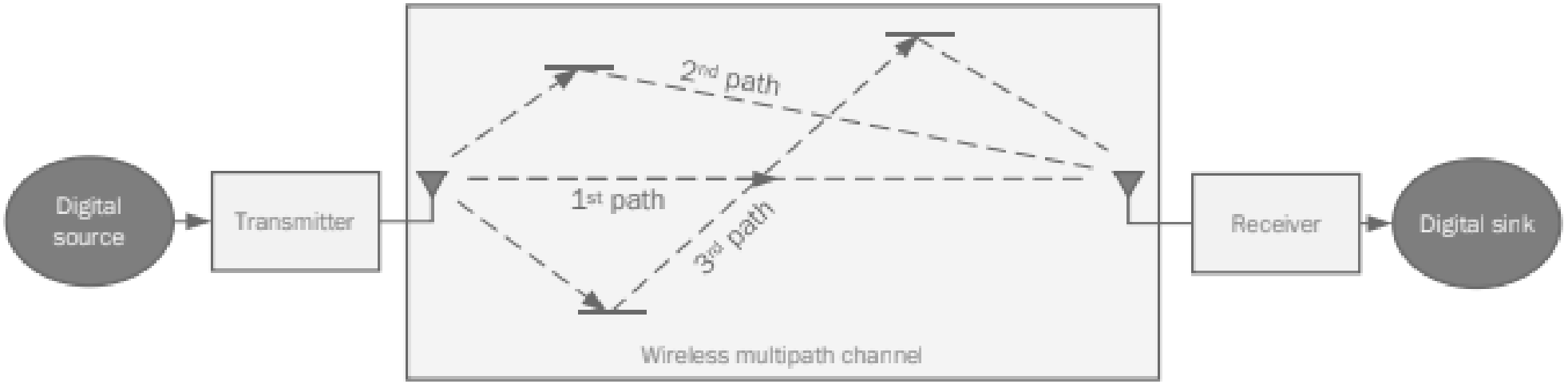


Figure 3 Signal propagation in a wireless multipath channel.

Part 2 : *Inter-Symbol Interference due to Multi-Path Channels*

So, your goal is to answer the following question:

Knowing Y, H, and the statistics of the AWGN noise (i.e., mean and variance), what is the best way of estimating the transmitted symbols X?

Guidelines: The goal here is to give an answer to the previous question. Note that there are several techniques to solve this equation in the literature. You can investigate one or more of these solutions. Please perform simulations to show the BER vs E_b/N_o performance of the techniques you consider for estimating X. In doing these simulations, you can assume that the transmitted symbols are BPSK symbols with energy $E_b = 1$. You can also assume that the coefficients of the channel are Complex Gaussian with zero mean and variance 1.

Part 2 : *Inter-Symbol Interference due to Multi-Path Channels*

```
%intro to Lab 2
%Part 2
%%%%%%%%%%%% generation of Transmitted symbols (BPSK) %%%%%%%%%%%%%
%%
L = 4000;      %No of paths
X_Tx = [];
X_Tx = randi([0 1],[L 1]);
A=length(X_Tx);
for i= 1 : A
    if X_Tx(i) == 0
        X_Tx(i)= -1;
    end
end
```

%%Here, Transmitted (X) is random sequence of {0,1} then convert it to BPSK (bipolar) to {-1,1}
%%

Part 2 : *Inter-Symbol Interference due to Multi-Path Channels*

```
function [h] = MultipathChannel(L,N)

% How to use:
%
% h = MultipathChannel(L) - generates a vector h of length Lx1 containing L
% channel coefficients for the L paths
% h = MultipathChannel(L) - generates a matrix h of dimention LxN, where
% each column corresponds to L channel coefficients for L paths

if nargin < 2
    N = 1;
end

h = randn(L,N) + 1i*randn(L,N);

power_profile = exp(-0.5*[0:L-1])';

power_profile = repmat(power_profile,1,N);

h = abs(h).*power_profile;
end
```

```
%%
%%%%%%%%%%%% generation Matrix of channel coeffients %%%%%%%%%%%%

coeff=MultipathChannel(L,1);
H= zeros(L,L);
for i =1:L
    for j = i:L
        H(j,i) = coeff(j - i +1);
    end
end
V = H * X_Tx ;
Eb_No_db = 0;       % The specified Eb/No value in dB
Energy_per_bit=1;
No=Energy_per_bit/( 10^(Eb_No_db/10) );
noise= randn(size(V))*sqrt(No/2); %generate Noise
Y = (H * X_Tx) + noise ; %getting the received signal Y
```

%%Generation of Channel effect coefficients of paths (h) and whole Effect matrix H
%%

Part 2 : *Inter-Symbol Interference due to Multi-Path Channels*

```
%%
%%%%%%%%%%%% Estimation of transmitted signal from Received signal %%%%%%%%%%%%
Z= inv(H); %Equalize channel effect
X = Z * Y;  %Received X from Y
X_Estimated =[];
%Decision Maker
for i=1:A
    if X(i) > 0
        B =1;
    else
        B=-1;
    end
    X_Estimated =[X_Estimated ; B];   %Estimated_Transmitted_signal
end
%Calculation of BER between transmitted symbols & estimated signal when
%variance = 1
BER = ComputeBER(X_Tx, X_Estimated);
```

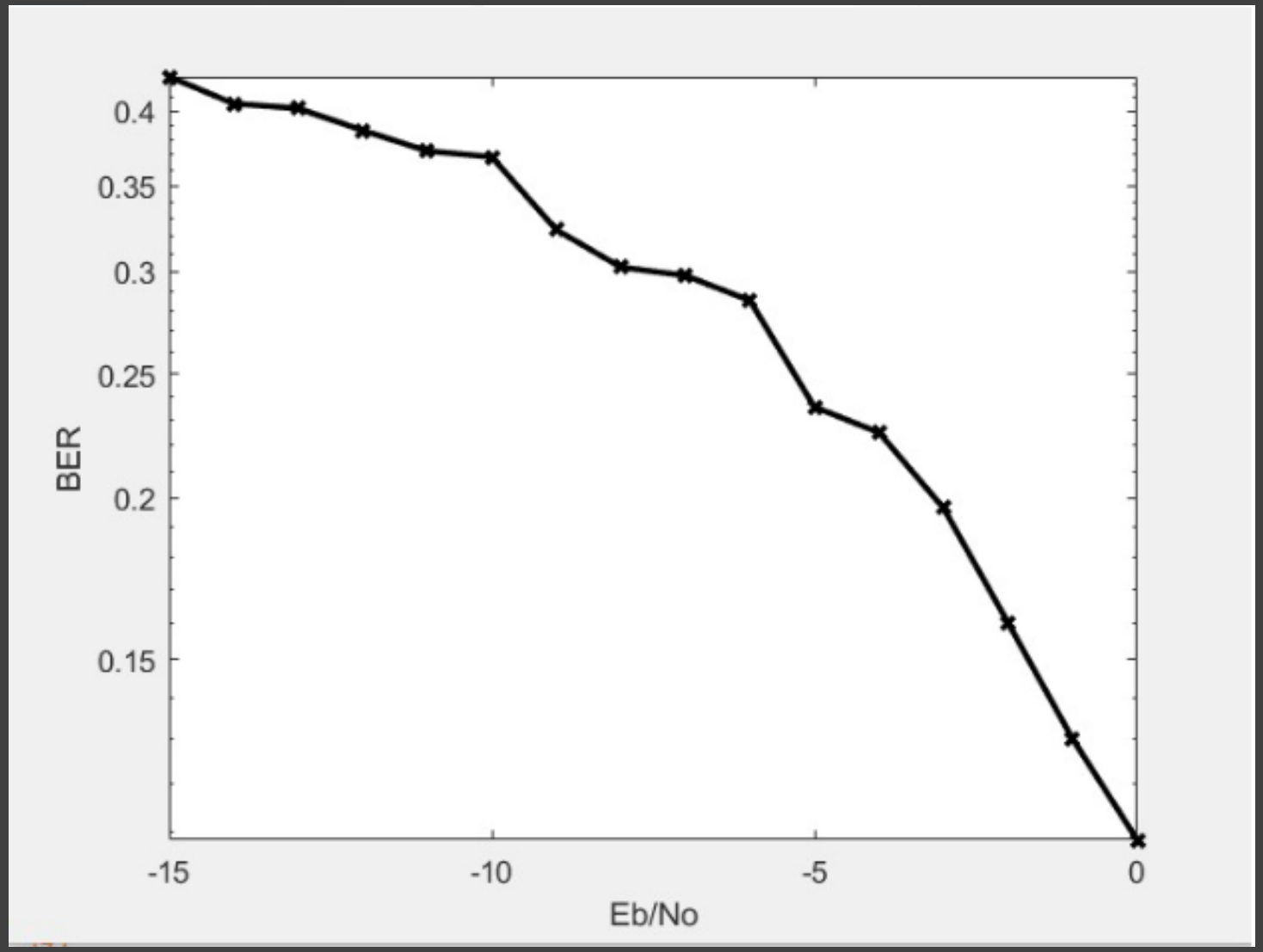
%%Here, we estimate Transmitted (X_Estimated) signal from Received Signal (Y) then convert it to be polar digital {-1,1} %%

Part 2 : *Inter-Symbol Interference due to Multi-Path Channels*

```
%%
%%%%%%%%%%%% Estimation of BER vs Eb/No %%%%%%%%%%%%%
Eb_No_dB_vector = -15:0;
BER1=zeros(size(Eb_No_dB_vector));

for i= 1:length(Eb_No_dB_vector)

No=Energy_per_bit/( 10^(Eb_No_dB_vector(i)/10) );
noise= randn(size(V))*sqrt(No/2);
Y = (H * X_Tx) + noise ;
Z= inv(H);
X= Z * Y;
X_Estimated =[];
for j=1:A
    if X(j) > 0
        B =1;
    else
        B =-1;
    end
X_Estimated =[X_Estimated ; B];
end
BER1(i) = ComputeBER(X_Tx, X_Estimated);
end
%Plotting BER vs Eb/No
figure
semilogy(Eb_No_dB_vector,BER1,'-xk','linewidth',2)
xlabel('Eb/No','linewidth',2)
ylabel('BER','linewidth',2)
```

Part 2 : *Inter-Symbol Interference due to Multi-Path Channels*

BER Representation



Part 3 : Comparison of coding Techniques

Part 3 : *Comparison Coding Techniques*

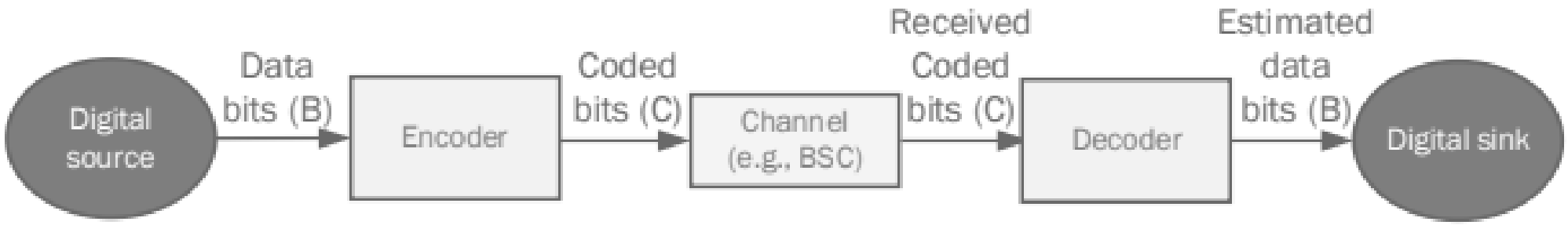


Figure 4 The process of channel coding.

Repetition code: in this code, each information bit is repeated L times. If the coding rate of the repetition code is r, this means that each information bit is repeated $L = \lceil \frac{1}{r} \rceil$ times. The decoding algorithm suitable for BSC is a majority rule vote.

Part 3 : *Comparison Coding Techniques*

Repetition Code Main Script

Steps :

1-Generate bits

2-Generate sequence of samples for each bit

3- Generate the received samples after passing through (Bit flipping channel)

4-Decode received bits then get BER

```
%Lab 2 intro to Digital Communication
%Part 3
%--------------------------------------------
N_bits = 1000; % Total number of bits
p      = 0.5;   % Channel parameter (probability of bit flipping)
```

```
%% Repetition Code
%-------------------------
% System parameters
fs  = 5;     % Number of samples per symbol (bit)

% Generate a bit sequence
bit_seq = GenerateBits(N_bits); % Generate a sequence of bits equal to the total number of bits

% Generate samples from bits
sample_seq = GenerateSamples(bit_seq,fs); % IMPLEMENT THIS: Generate a sequence of samples for each bit

% Pass the sample sequence through the channel
rec_sample_seq = BSC(sample_seq,fs,p,'independent');   % Generate the received samples after passing through the bit flipping channel

% Decode bits from received bit sequence
rec_bit_seq = DecodeBitsFromSamples(rec_sample_seq,fs);    % IMPLEMENT THIS: Decode the received bits

% Compute the BER
BER_Rep = ComputeBER(bit_seq,rec_bit_seq);   % Calculate the bit error rate
```

Part 3 : *Comparison Coding Techniques*

Generate Bits Function

```
function bit_seq = GenerateBits(N_bits)
%
% Inputs:
%   N_bits:     Number of bits in the sequence
% Outputs:
%   bit_seq:    The sequence of generated bits
%
% This function generates a sequence of bits with length equal to N_bits

%%% WRITE YOUR CODE HERE
bit_seq = randi([0 1],1,N_bits);  %N random bits generator
%%%
```

Part 3 : *Comparison Coding Techniques*

Generate Samples Function (Encoder)

```
function sample_seq = GenerateSamples(bit_seq,fs)
%
% Inputs:
%   bit_seq:    Input bit sequence
%   fs:         Number of samples per bit
% Outputs:
%   sample_seq: The resultant sequence of samples
%
% This function takes a sequence of bits and generates a sequence of
% samples as per the input number of samples per bit

sample_seq = zeros(size(bit_seq*fs));

%%% WRITE YOUR CODE FOR PART 2 HERE
sample_seq = repelem(bit_seq,1,fs);
%%%
```

Part 3 : *Comparison Coding Techniques*

BSC channel Function

```
function rec_sample_seq  = BSC(sample_seq,fs,p,channel_type)
%
% Inputs:
%   sample_seq:     The input sample sequence to the channel
%   fs:             The sampling frequency used to generate the sample sequence
%   p:              The bit flipping probability
%   channel_type:   The type of channel, 'independent' or 'correlated'
% Outputs:
%   rec_sample_seq: The sequence of sample sequence after passing through the channel
%
% This function takes the sample sequence passing through the channel, and
% generates the output sample sequence based on the specified channel type
% and parameters

sample_seq      = ~~sample_seq;
rec_sample_seq  = zeros(size(sample_seq));
rec_sample_seq  = ~~rec_sample_seq;

if (nargin <= 3)
    channel_type = 'independent';
end

switch channel_type
```

Part 3 : *Comparison Coding Techniques*

BSC channel Function (Cont.)

```
switch channel_type

    case 'independent'
        channel_effect = rand(size(rec_sample_seq))<=p;
    case 'correlated'
        channel_effect = rand(1,length(rec_sample_seq)/fs)<=p;
        channel_effect = repmat(channel_effect,fs,1);
        channel_effect = channel_effect(:)';
end

rec_sample_seq = xor(sample_seq,channel_effect);
rec_sample_seq = rec_sample_seq + 0;
```

Part 3 : *Comparison Coding Techniques*

BER Computation for Repetition Code

```
function BER = ComputeBER(bit_seq,rec_bit_seq)
%
% Inputs:
%   bit_seq:     The input bit sequence
%   rec_bit_seq: The output bit sequence
% Outputs:
%   BER:         Computed BER
%
% This function takes the input and output bit sequences and computes the
% BER


%   E:           number of error bits
E = 0;
for i=1:length(bit_seq)
    if rec_bit_seq(i) ~= bit_seq(i)

        E = E + 1 ;

    end
end

BER = E/length(bit_seq);
%%%
```

Part 3 : *Comparison Coding Techniques*

Decoder of Repetition code

```
function rec_bit_seq = DecodeBitsFromSamples(rec_sample_seq,fs)
%
% Inputs:
%   rec_sample_seq: The input sample sequence to the channel
%   case_type:      The sampling frequency used to generate the sample sequence
%   fs:             The bit flipping probability
% Outputs:
%   rec_sample_seq: The sequence of sample sequence after passing through the channel
%
% This function takes the sample sequence after passing through the
% channel, and decodes from it the sequence of bits based on the considered
% case and the sampling frequence

if (nargin <= 2)
    fs = 1;
        count=1;
        for i=1:5:length(rec_sample_seq)
           array=rec_sample_seq(i:i+4);
           if (sum(array)>=3)
          rec_bit_seq(count) = 1;
          else rec_bit_seq(count) = 0;
          end
           count=count+1;
        end
end
```

Part 3 : *Comparison Coding Techniques*

Results @ P = 0 .1

BER_Rep	0.0100
bit_seq	*1x1000 double*
fs	5
N_bits	1000
p	0.1000
rec_bit_seq	*1x1000 double*
rec_sample_seq	*1x5000 double*
sample_seq	*1x5000 double*

Part 3 : *Comparison Coding Techniques*

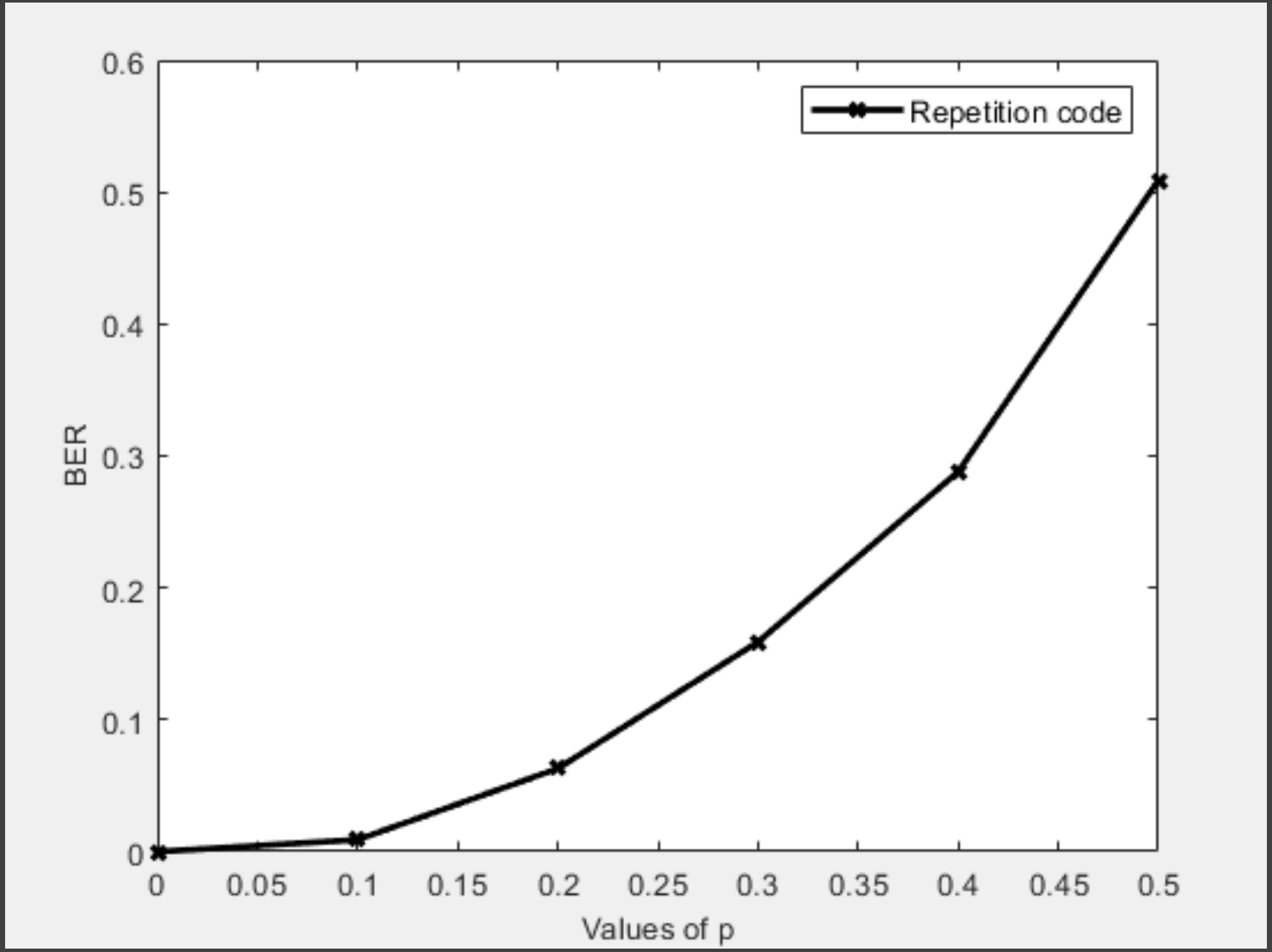
Results @ P = 0 .5

So, if we increase the probability of flipping of the channel then Bit error rate increases until 0.5 at p = 0.5

BER_Rep	0.5050
bit_seq	*1x1000 double*
fs	5
N_bits	1000
p	0.5000
rec_bit_seq	*1x1000 double*
rec_sample_seq	*1x5000 double*
sample_seq	*1x5000 double*

Part 3 : *Comparison Coding Techniques*

BER representation of Repetition code



Part 3 : *Comparison Coding Techniques*

Convolutional code

Convolutional code: this code generates a sequence of coded bits corresponding to an input stream of information bits. The encoding of convolutional codes is based on a given Generator polynomial, and the decoding of convolutional codes is using Viterbi algorithm. Feel free to choose any Generator polynomial you may think is useful. However, choosing a realistic convolutional code (e.g., one that is used in the LTE standard for example?) would show good effort from your side!

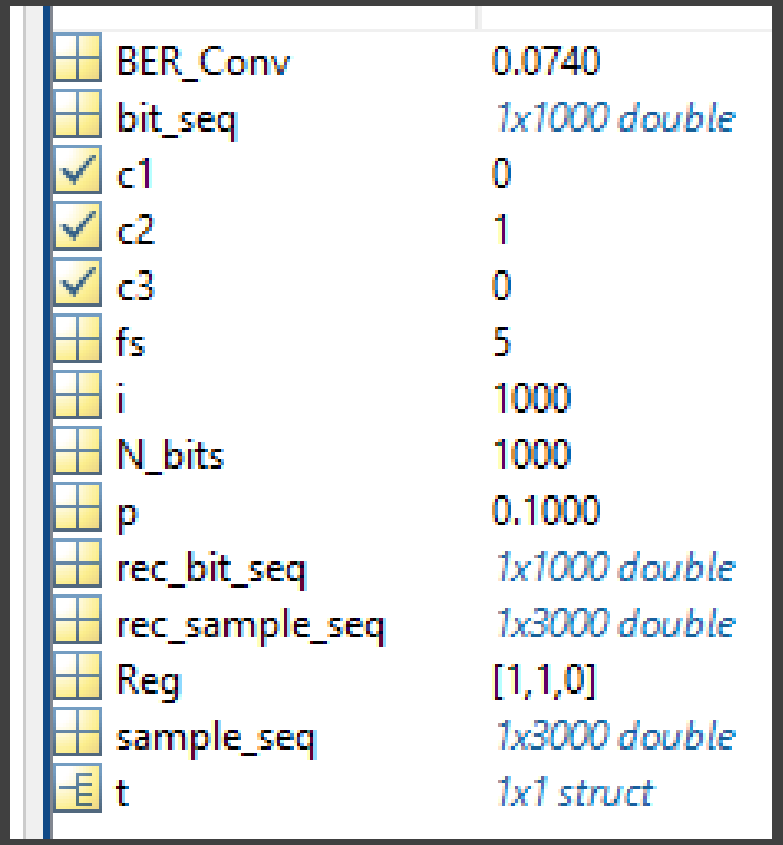
Part 3 : *Comparison Coding Techniques*

Convolutional code main script (Encoder and Decoder)

```
%% Convolutional Code
%-------------------------
%Encoder
Reg = [0 0 0];
bit_seq = randi([0 1],1,N_bits);
sample_seq = zeros(1,2*N_bits);
for i = 1:N_bits
    Reg = circshift(Reg,1);
    Reg(1) = bit_seq(i);
    c1 = xor(Reg(1),Reg(2));
    c2 = xor(Reg(2),Reg(3));
    c3 = xor(c1,Reg(3));
    sample_seq(3*i-2) = c1;
    sample_seq(3*i-1) = c2;
    sample_seq(3*i) = c3;
end

t = poly2trellis([3],[6 3 7]);

% Pass the sample sequence through the channel
rec_sample_seq = BSC(sample_seq,2,0.1);   % Generate the received samples after passing through the bit flipping channel

%Decoder
rec_bit_seq = vitdec(rec_sample_seq,t,1,'trunc','hard');

% Compute the BER
BER_Conv = ComputeBER(rec_bit_seq,bit_seq);   % Calculate the bit error rate
```

Part 3 : *Comparison Coding Techniques*

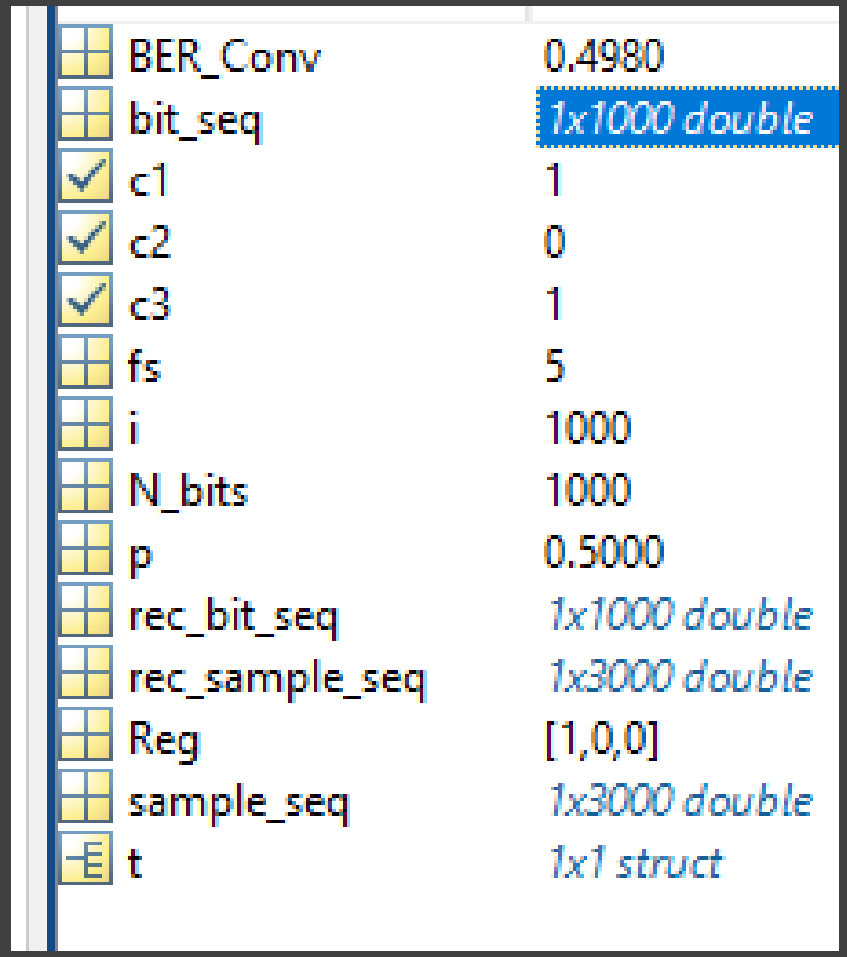
Results @ P = 0 .1
BER = 0.0740



Part 3 : *Comparison Coding Techniques*

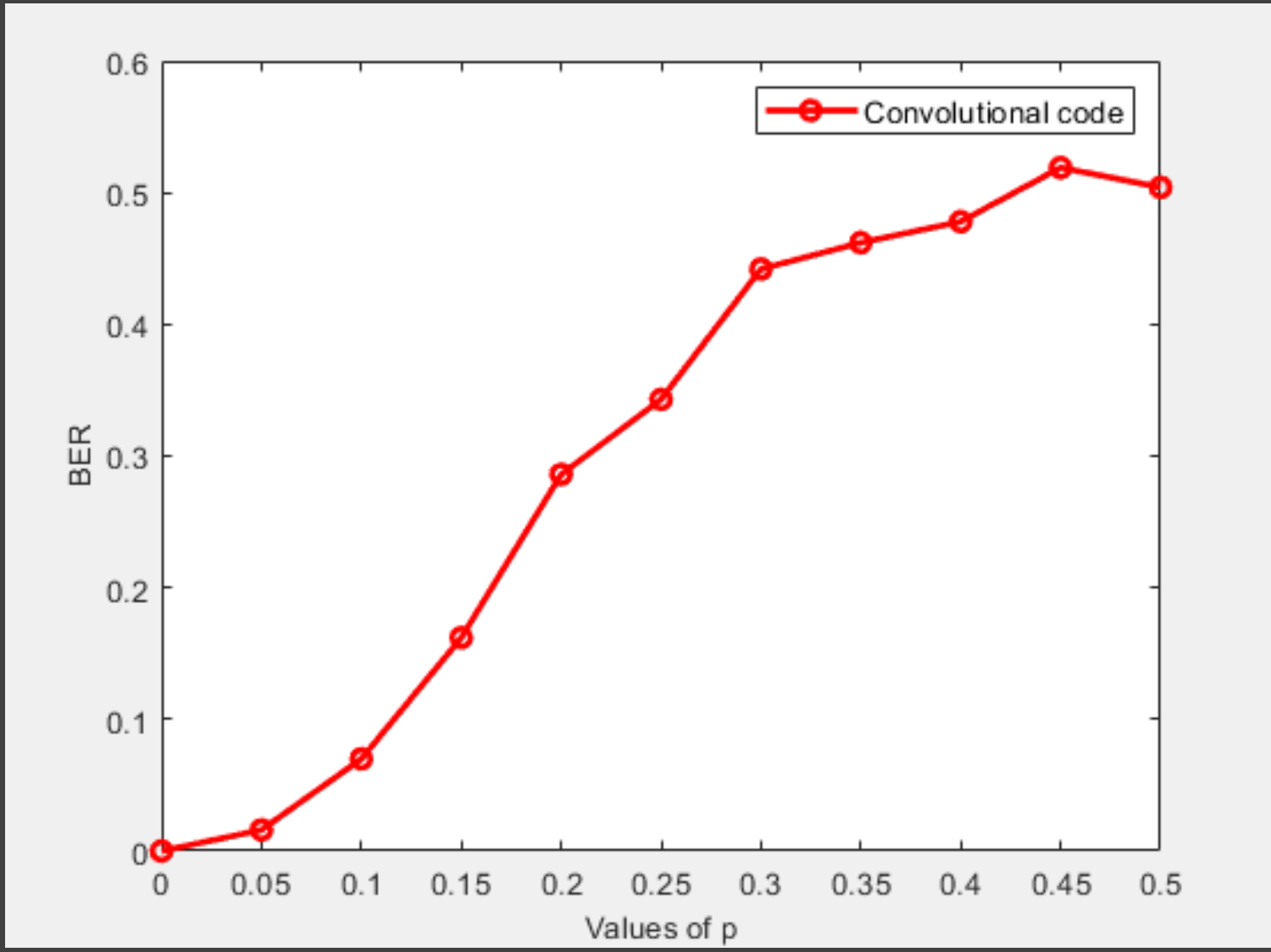
Results @ P = 0 .5
BER = 0.4980

So, if we increase the probability of flipping of the channel then Bit error rate increases until 0.5 at p = 0.5



Part 3 : *Comparison Coding Techniques*

BER representation of Repetition code



Part 3 : *Comparison Coding Techniques*

For Reliability sequence , The values ordered from worst to best in matrix 1 x 1024

```
%Reliability sequence (Frozen positions)

Frozen_pos =[0 1 2 4 8 16 32 3 5 64 9 6 17 10 18 128 12 33 65 20 256 34 24 36 7 129 66 512 11 40 68 130 ...
   19 13 48 14 72 257 21 132 35 258 26 513 80 37 25 22 136 260 264 38 514 96 67 41 144 28 69 42 ...
   516 49 74 272 160 520 288 528 192 544 70 44 131 81 50 73 15 320 133 52 23 134 384 76 137 82 56 27 ...
   97 39 259 84 138 145 261 29 43 98 515 88 140 30 146 71 262 265 161 576 45 100 640 51 148 46 75 266 273 517 104 162 ...
   53 193 152 77 164 768 268 274 518 54 83 57 521 112 135 78 289 194 85 276 522 58 168 139 99 86 60 280 89 290 529 524 ...
   196 141 101 147 176 142 530 321 31 200 90 545 292 322 532 263 149 102 105 304 296 163 92 47 267 385 546 324 208 386 150 153 ...
   165 106 55 328 536 577 548 113 154 79 269 108 578 224 166 519 552 195 270 641 523 275 580 291 59 169 560 114 277 156 87 197 ...
   116 170 61 531 525 642 281 278 526 177 293 388 91 584 769 198 172 120 201 336 62 282 143 103 178 294 93 644 202 592 323 392 ...
   297 770 107 180 151 209 284 648 94 204 298 400 608 352 325 533 155 210 305 547 300 109 184 534 537 115 167 225 326 306 772 157 ...
   656 329 110 117 212 171 776 330 226 549 538 387 308 216 416 271 279 158 337 550 672 118 332 579 540 389 173 121 553 199 784 179 ...
   228 338 312 704 390 174 554 581 393 283 122 448 353 561 203 63 340 394 527 582 556 181 295 285 232 124 205 182 643 562 286 585 ...
   299 354 211 401 185 396 344 586 645 593 535 240 206 95 327 564 800 402 356 307 301 417 213 568 832 588 186 646 404 227 896 594 ...
   418 302 649 771 360 539 111 331 214 309 188 449 217 408 609 596 551 650 229 159 420 310 541 773 610 657 333 119 600 339 218 368 ...
   652 230 391 313 450 542 334 233 555 774 175 123 658 612 341 777 220 314 424 395 673 583 355 287 183 234 125 557 660 616 342 316 ...
   241 778 563 345 452 397 403 207 674 558 785 432 357 187 236 664 624 587 780 705 126 242 565 398 346 456 358 405 303 569 244 595 ...
   189 566 676 361 706 589 215 786 647 348 419 406 464 680 801 362 590 409 570 788 597 572 219 311 708 598 601 651 421 792 802 611 ...
   602 410 231 688 653 248 369 190 364 654 659 335 480 315 221 370 613 422 425 451 614 543 235 412 343 372 775 317 222 426 453 237 ...
   559 833 804 712 834 661 808 779 617 604 433 720 816 836 347 897 243 662 454 318 675 618 898 781 376 428 665 736 567 840 625 238 ...
   359 457 399 787 591 678 434 677 349 245 458 666 620 363 127 191 782 407 436 626 571 465 681 246 707 350 599 668 790 460 249 682 ...
   573 411 803 789 709 365 440 628 689 374 423 466 793 250 371 481 574 413 603 366 468 655 900 805 615 684 710 429 794 252 373 605 ...
   848 690 713 632 482 806 427 904 414 223 663 692 835 619 472 455 796 809 714 721 837 716 864 810 606 912 722 696 377 435 817 319 ...
   621 812 484 430 838 667 488 239 378 459 622 627 437 380 818 461 496 669 679 724 841 629 351 467 438 737 251 462 442 441 469 247 ...
   683 842 738 899 670 783 849 820 728 928 791 367 901 630 685 844 633 711 253 691 824 902 686 740 850 375 444 470 483 415 485 905 ...
   795 473 634 744 852 960 865 693 797 906 715 807 474 636 694 254 717 575 913 798 811 379 697 431 607 489 866 723 486 908 718 813 ...
```

Part 3 : *Comparison Coding Techniques*

Steps

1-Generating an input of a random sequence of K bits

2-Generating frozen bits from reliability sequence equal to (N-K)

3-generating Generator Matrix (G2) with size 2x2

4-Generating (G) by polar transformation (G2 with every previous G) ((kronecker product))

5-Encoding Input signal by polar transformation

$$G_{2^n} = \begin{bmatrix} 1 & 0 \\ 1 & 1 \end{bmatrix}^{\otimes n}$$

```
%% Polar Transformation Encoder
%Generates a sequence of bits
n=2;
N=2^n;
K=0.5*N;
bit_seq=randi([0,1],1,K);
%Generates Input signal
R = Frozen_pos(Frozen_pos<=N); %reliability sequence for N
U=zeros(1,N);
U(R(N-K+1 : end))=bit_seq;
%Generates a Generator Matrix N*N

G2=[1 0 ; 1 1];
G=G2;
for j=1:(n-1)
   G=kron(G2,G);  %Polar transformation
end
%Encode Input Signal using Polar Transform
X= U*G;
X_TX=[];
for i=1:length(X)
    if mod(X(i),2) == 0
     X_TX=[X_TX 0];
    else
     X_TX=[X_TX 1];
    end
end
```

Part 3 : *Comparison Coding Techniques*

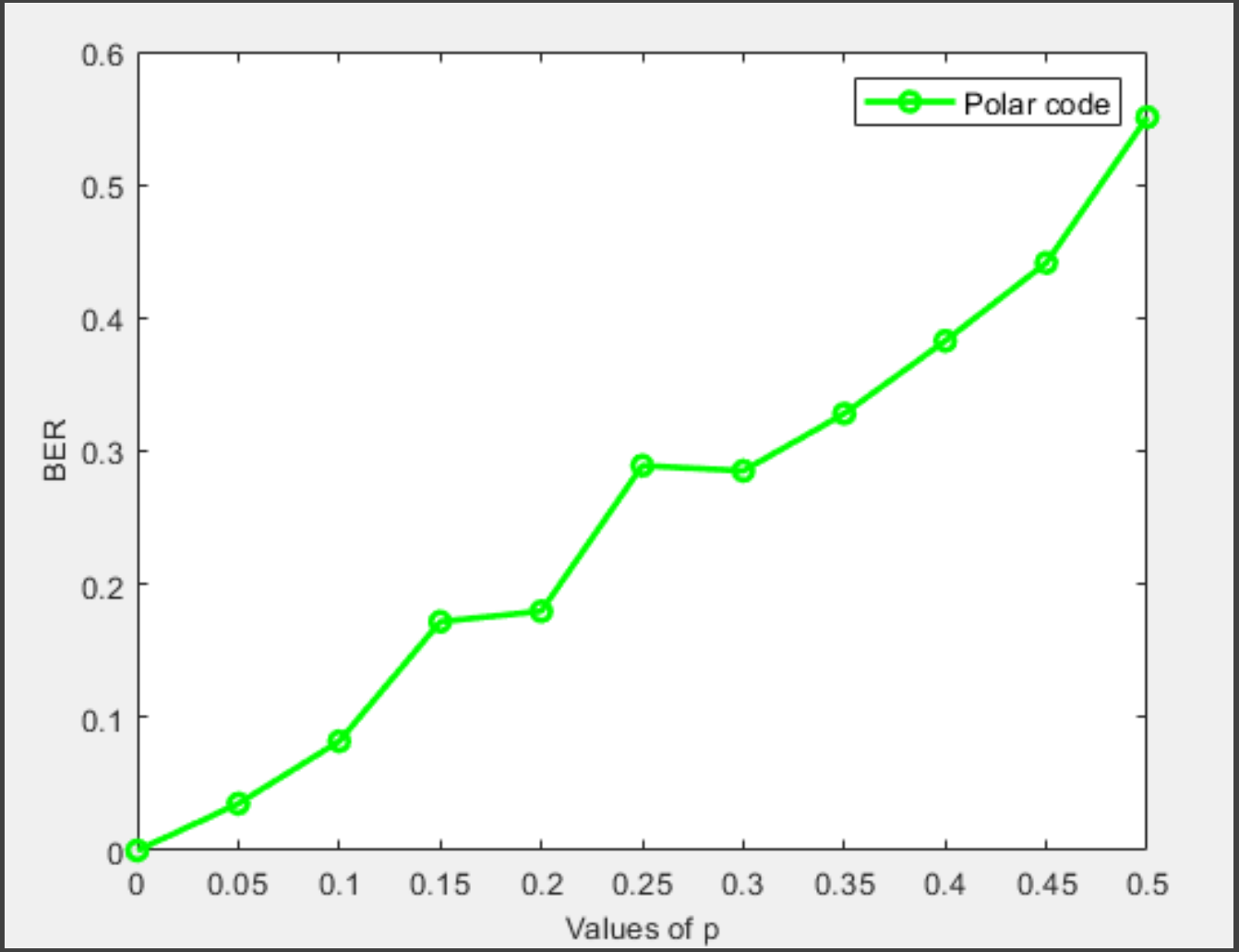
```
%% Decoder (Successive Cancellation)


%%BSC channel effect
channel_effect = rand(size(X_TX))<= p;
X_RX = xor(X_TX,channel_effect);
```

Part 3 : *Comparison Coding Techniques*

```
%BER computation
p_vect          = 0:0.05:0.5;
BER_3  = zeros(size(p_vect));
for p_ind =1:length(p_vect)

    channel_effect = rand(size(X_TX))<= p_vect(p_ind);
    X_RX = xor(X_TX,channel_effect);
    BER_3(p_ind) = numel(find(X_TX ~= X_RX))/length(X_TX);
end

figure(3)
plot(p_vect,BER_3,'o-g','linewidth',2);
xlabel('Values of p','fontsize',10)
ylabel('BER','fontsize',10)
legend('Polar code','fontsize',10)

%Repetition, Convolution and Polar Codes BER
figure(4)
plot(p_vect,BER_1,'x-k','linewidth',2); hold on
plot(p_vect,BER_2,'o-r','linewidth',2); hold on
plot(p_vect,BER_3,'d-b','linewidth',2); hold on

xlabel('Values of p','fontsize',10)
ylabel('BER','fontsize',10)
legend('Rep','Conv','Polar','fontsize',10)
```

Part 3 : *Comparison Coding Techniques*

BER representation of Polar code



Part 3 : *Comparison Coding Techniques*

BER representation of three curves at same plot