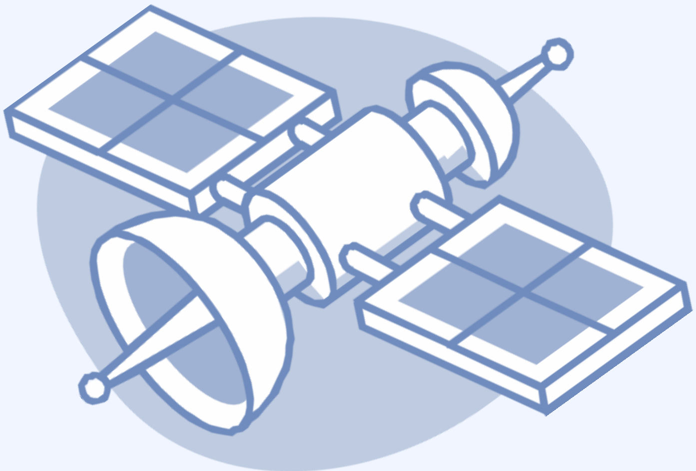
ECE252s

Fundamentals of communication systems project.

Line coding system & BPSK transmitter and Receiver using octave simulator.



Spring 2023

Prepared by: Team 08.

CSE & ECE & EPM students.

Prepared to: Dr. Alaa Eldin Fathy.

The team’s details:

Team Members	ID	Program
Mohamed Ahmed Sayed	2001171	ECE
Eslam Mohamed Marzouk	2000252	CSE
Abdelrahman Ahmed Sayed	2001722	ECE
Youssef Hassanin Mahmoud	2001398	EPM
Ahmed Nader Ahmed	2000056	ECE
Karim Ibrahim Saad	2001118	CSE
Abdelrahman Elsayed Ahmed	2002139	CSE
Muhammad AbdelKhaleq Muhammad	2001052	ECE
Yasmeen Mahmoud Hassan	2001304	EPM
Ahmed Ali Abd Elhakeem	2000047	ECE

The team leader:

Mohamed Ahmed Sayed. **ID: 2001171** **ECE**

Table of contents:

Part1: Line coding system:

Transmitter:

Receiver:

Part2: Binary phase shift-keying:

Part1: Line coding system:

Transmitter:

Step 1 : Generate stream of random bits:

Code:

<u>Used functions:</u>

File: generate_random_bits.m

```
function bit_stream = generate_random_bits( noOfBits )
   % generate random sequence of binary data zeros and ones
   bit_stream = randi( [ 0 , 1 ] , 1 , noOfBits );
end
```

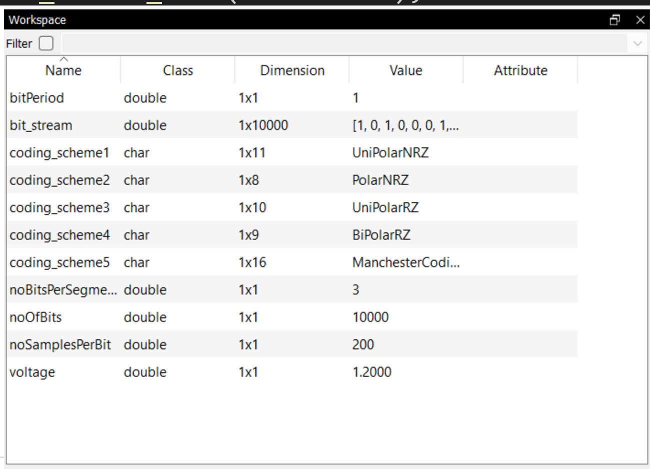
<u>Script:</u>

File: part1.m

```
% Clear all variables and close all figures
clear all;
close all;

%set values
voltage=1.2;

noOfBits=10000;
bitPeriod=1;
noSamplesPerBit=200;
noBitsPerSegments=3;

coding_scheme1='UniPolarNRZ';
coding_scheme2='PolarNRZ';
coding_scheme3='UniPolarRZ';
coding_scheme4='BiPolarRZ';
coding_scheme5='ManchesterCoding';


% generate bit_stream
bit_stream = generate_random_bits( noOfBits );
```

Snapshots:

Workspace:



Name	Class	Dimension	Value	Attribute
bitPeriod	double	1x1	1	
bit_stream	double	1x10000	[1, 0, 1, 0, 0, 0, 1,...	
coding_scheme1	char	1x11	UniPolarNRZ	
coding_scheme2	char	1x8	PolarNRZ	
coding_scheme3	char	1x10	UniPolarRZ	
coding_scheme4	char	1x9	BiPolarRZ	
coding_scheme5	char	1x16	ManchesterCodi...	
noBitsPerSegme...	double	1x1	3	
noOfBits	double	1x1	10000	
noSamplesPerBit	double	1x1	200	
voltage	double	1x1	1.2000	

Step 2: Line code the stream of bits:

Note: this part is containing all the line codes required in step 7 (Uni-polar non return to zero, Polar non return to zero, Uni-polar return to zero, Bipolar return to zero and Manchester coding).

Code:

Function:

File: line_coding.m

```
function [lineCodeVec,timeVec]=line_coding(bit_stream,coding_scheme,voltage,bitPeriod,noSamplesPerBit,fc )
    switch nargin
        %choose according to number of input arguments
        case 5
            %belong to part 1
            [lineCodeVec,timeVec]=Baseband_communication(bit_stream,coding_scheme,voltage,bitPeriod,noSamplesPerBit);
        case 6
            %belong to part 2
            [lineCodeVec,timeVec]=Passband_communication(bit_stream,coding_scheme,voltage,bitPeriod,noSamplesPerBit,fc);
    end
end


%% skip to the needed function in part1
function lineCodeVec = polarNRZ(bit_stream , voltage , timeVec , noSamplesPerBit)
    lineCodeVec = unipolarNRZ(bit_stream , voltage , timeVec , noSamplesPerBit);
    %same as unipolarNRZ but change all the zeros into -ve voltage
    lineCodeVec(lineCodeVec == 0) = -1 * voltage;
end
function lineCodeVec = unipolarRZ(bit_stream , voltage , timeVec , noSamplesPerBit)
    lineCodeVec = zeros(1 , length(timeVec));
    for i = 1 : length(bit_stream)
        if bit_stream(i) == 1
            %+ve voltage for the first half cycle of the bits
            lineCodeVec( ((i - 1) * noSamplesPerBit) + 1 : (i * noSamplesPerBit) - (noSamplesPerBit / 2)) = voltage;
            % 0 voltage for the other half cycle of the bit
            lineCodeVec( (i * noSamplesPerBit) - (noSamplesPerBit / 2) + 1 : i * noSamplesPerBit) = 0;
        end
    end
end
function lineCodeVec = bipolarRZ(bit_stream , voltage , timeVec , noSamplesPerBit)
    lineCodeVec = zeros(1 , length(timeVec));
    flag = 0; % to indicate whether the voltage to be +ve or -ve
    for i = 1 : length(bit_stream)
        if bit_stream(i) == 1
```

```
            if (flag == 0)
                %+ve voltage for the first half cycle of the bits
                lineCodeVec( ((i - 1) * noSamplesPerBit) + 1 : (i * noSamplesPerBit) - (noSamplesPerBit / 2)) = voltage;
                % 0 voltage for the other half cycle of the bit
                lineCodeVec( (i * noSamplesPerBit) - (noSamplesPerBit / 2) + 1 : i * noSamplesPerBit) = 0;
                flag = 1; %update the flag
            elseif(flag == 1)
                %-ve voltage for the first half cycle of the bits
                lineCodeVec( ((i - 1) * noSamplesPerBit) + 1 : (i * noSamplesPerBit) - (noSamplesPerBit / 2)) = -voltage;
                % 0 voltage for the other half cycle of the bit
                lineCodeVec( (i * noSamplesPerBit) - (noSamplesPerBit / 2) + 1 : i * noSamplesPerBit) = 0;
                flag = 0; %update the flag
            end
        end
    end
end
function lineCodeVec = manchesterCoding(bit_stream , voltage , timeVec , noSamplesPerBit)
    lineCodeVec = zeros(1 , length(timeVec));
    for i = 1 : length(bit_stream)
        if bit_stream(i) == 1
            %+ve voltage for the first half cycle of the bits
            lineCodeVec( ((i - 1) * noSamplesPerBit) + 1 : (i * noSamplesPerBit) - (noSamplesPerBit / 2)) = voltage;
            %-ve voltage for the other half cycle of the bit
            lineCodeVec( (i * noSamplesPerBit) - (noSamplesPerBit / 2) + 1 : i * noSamplesPerBit) = -1* voltage;
        else
            %-ve voltage for the first half cycle of the bits
            lineCodeVec( ((i - 1) * noSamplesPerBit) + 1 : (i * noSamplesPerBit) - (noSamplesPerBit / 2)) = -1 * voltage;
            %+ve voltage for the other half cycle of the bit
            lineCodeVec( (i * noSamplesPerBit) - (noSamplesPerBit / 2) + 1 : i * noSamplesPerBit) = voltage;
        end
    end
end
```

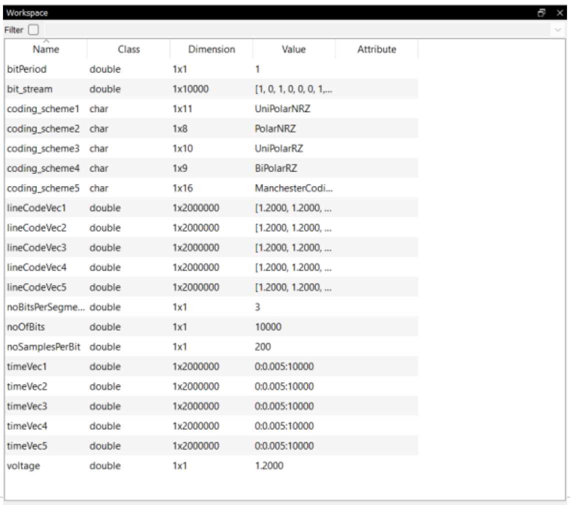
<u>Script:</u>

File: part1.m

```
[lineCodeVec1, timeVec1] = line_coding(bit_stream, coding_scheme1, voltage, bitPeriod, noSamplesPerBit);
[lineCodeVec2, timeVec2] = line_coding(bit_stream, coding_scheme2, voltage, bitPeriod, noSamplesPerBit);
[lineCodeVec3, timeVec3] = line_coding(bit_stream, coding_scheme3, voltage, bitPeriod, noSamplesPerBit);
[lineCodeVec4, timeVec4] = line_coding(bit_stream, coding_scheme4, voltage, bitPeriod, noSamplesPerBit);
[lineCodeVec5, timeVec5] = line_coding(bit_stream, coding_scheme5, voltage, bitPeriod, noSamplesPerBit);

%% plot 5 line coding in time domain
% UniPolarNRZ line coding
figure(1);
plot(timeVec1, lineCodeVec1, 'color', [0.4940 0.1840 0.5560], 'LineWidth', 2);
axis([0 timeVec1(end) 0 - voltage/2 voltage*3/2]);
title("UniPolarNRZ line coding");
legend('UniPolarNRZ');

% Zoomed
figure(2);
plot(timeVec1, lineCodeVec1, 'color', [0.4940 0.1840 0.5560], 'LineWidth', 2);
axis([0 100*bitPeriod 0 - voltage/2 voltage*3/2]);
title("UniPolarNRZ line coding (zoomed to first 100 bits)");
legend('UniPolarNRZ');
grid on;

% PolarNRZ line coding
figure(3);
plot(timeVec2, lineCodeVec2, 'color', [1 0 0], 'LineWidth', 2);
axis([0 timeVec2(end) -3*voltage/2 3*voltage/2]);
title("PolarNRZ line coding");
legend('PolarNRZ');

% Zoomed
figure(4);
plot(timeVec2, lineCodeVec2, 'color', [1 0 0], 'LineWidth', 2);
axis([0 100*bitPeriod -3*voltage/2 3*voltage/2]);
title("PolarNRZ line coding (zoomed to first 100 bits)");
legend('PolarNRZ');
grid on;

% UniPolarRZ line coding
figure(5);
plot(timeVec3, lineCodeVec3, 'color', [0.8500 0.3250 0.0980], 'LineWidth', 2);
axis([0 timeVec3(end) 0 - voltage/2 voltage*3/2]);
title("UniPolarRZ line coding");
legend('UniPolarRZ');
```

```
% Zoomed
figure(6);
plot(timeVec3, lineCodeVec3, 'color', [0.8500 0.3250 0.0980], 'LineWidth', 2);
axis([0 100*bitPeriod 0 - voltage/2 voltage*3/2]);
title("UniPolarRZ line coding (zoomed to first 100 bits)");
legend('UniPolarRZ');

% BiPolarRZ line coding
figure(7);
plot(timeVec4, lineCodeVec4, 'color', [0 0.4470 0.7410], 'LineWidth', 2);
axis([0 timeVec4(end) -3*voltage/2 3*voltage/2]);
title("BiPolarRZ line coding");
legend('BiPolarRZ');

% Zoomed
figure(8);
plot(timeVec4, lineCodeVec4, 'color', [0 0.4470 0.7410], 'LineWidth', 2);
axis([0 100*bitPeriod -3*voltage/2 3*voltage/2]);
title("BiPolarRZ line coding (zoomed to first 100 bits)");
legend('BiPolarRZ');
grid on;

% ManchesterCoding line coding
figure(9);
plot(timeVec5, lineCodeVec5, 'color', [0.6350 0.0780 0.1840], 'LineWidth', 2);
axis([0 timeVec5(end) -3*voltage/2 3*voltage/2]);
title("ManchesterCoding line coding");
legend('ManchesterCoding');

% Zoomed
figure(10);
plot(timeVec5, lineCodeVec5, 'color', [0.6350 0.0780 0.1840], 'LineWidth', 2);
axis([0 100*bitPeriod -3*voltage/2 3*voltage/2]);
title("ManchesterCoding line coding (zoomed to first 100 bits)");
legend('ManchesterCoding');
grid on;
```

Snapshots:
Workspace:



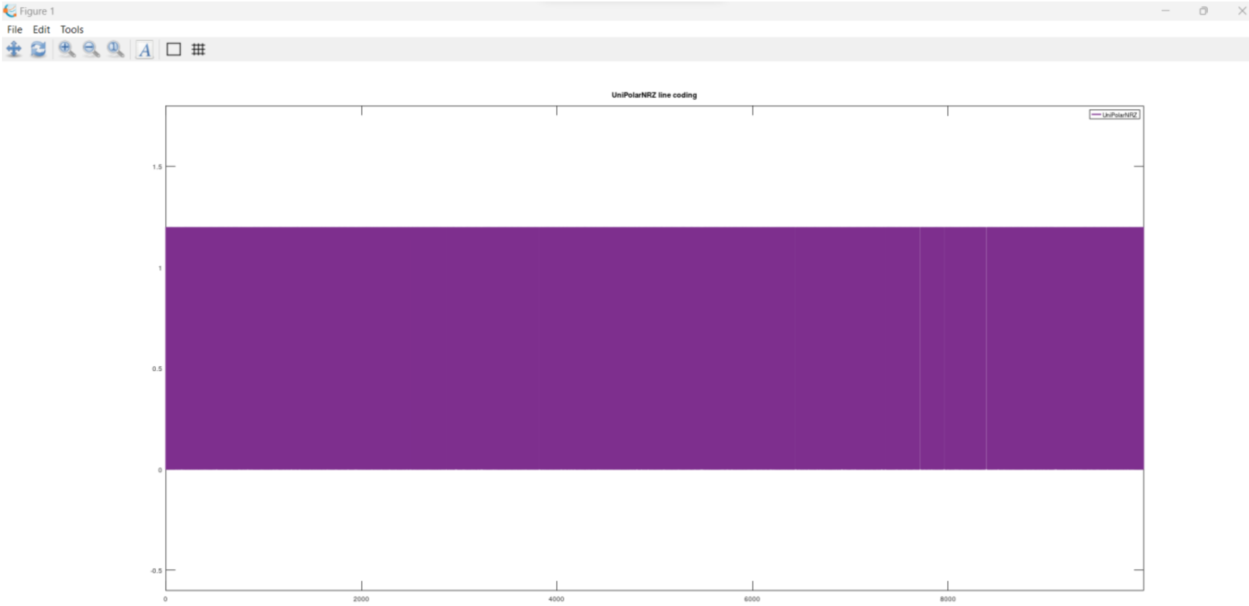
Workspace

Filter

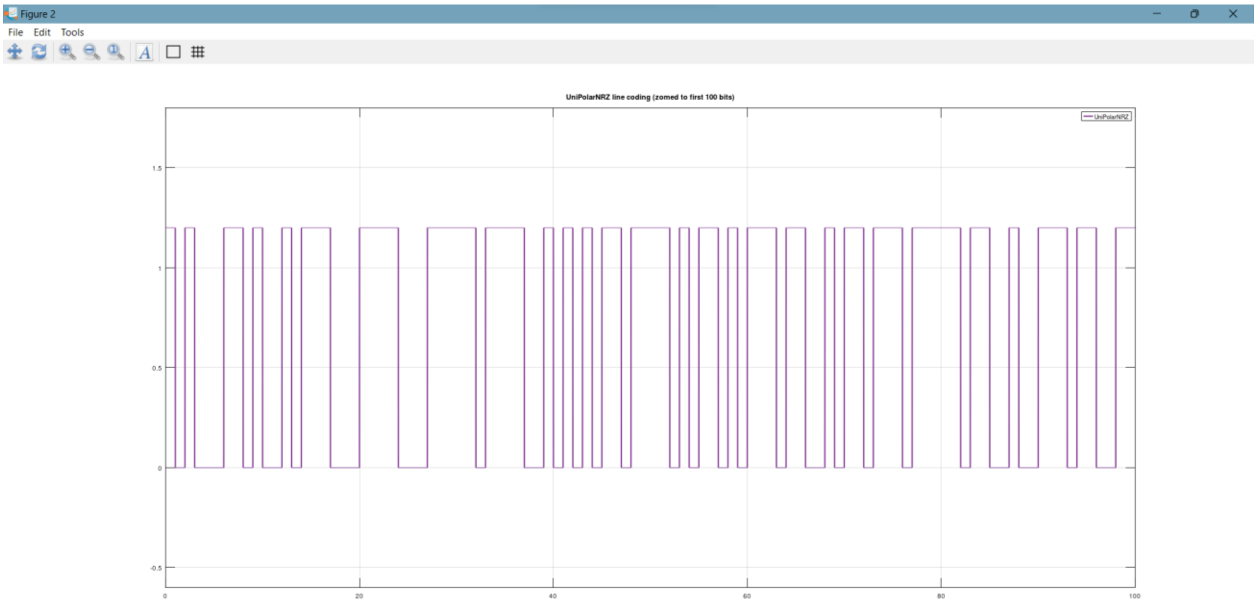
Name	Class	Dimension	Value	Attribute
bitPeriod	double	1x1	1	
bit_stream	double	1x10000	[1, 0, 1, 0, 0, 0, 1,...	
coding_scheme1	char	1x11	UniPolarNRZ	
coding_scheme2	char	1x8	PolarNRZ	
coding_scheme3	char	1x10	UniPolarRZ	
coding_scheme4	char	1x9	BiPolarRZ	
coding_scheme5	char	1x16	ManchesterCodi...	
lineCodeVec1	double	1x2000000	[1.2000, 1.2000, ...	
lineCodeVec2	double	1x2000000	[1.2000, 1.2000, ...	
lineCodeVec3	double	1x2000000	[1.2000, 1.2000, ...	
lineCodeVec4	double	1x2000000	[1.2000, 1.2000, ...	
lineCodeVec5	double	1x2000000	[1.2000, 1.2000, ...	
noBitsPerSegme...	double	1x1	3	
noOfBits	double	1x1	10000	
noSamplesPerBit	double	1x1	200	
timeVec1	double	1x2000000	0:0.005:10000	
timeVec2	double	1x2000000	0:0.005:10000	
timeVec3	double	1x2000000	0:0.005:10000	
timeVec4	double	1x2000000	0:0.005:10000	
timeVec5	double	1x2000000	0:0.005:10000	
voltage	double	1x1	1.2000	

<u>Time domain plot of UniPolar NRZ signal:</u>

Before zoom:

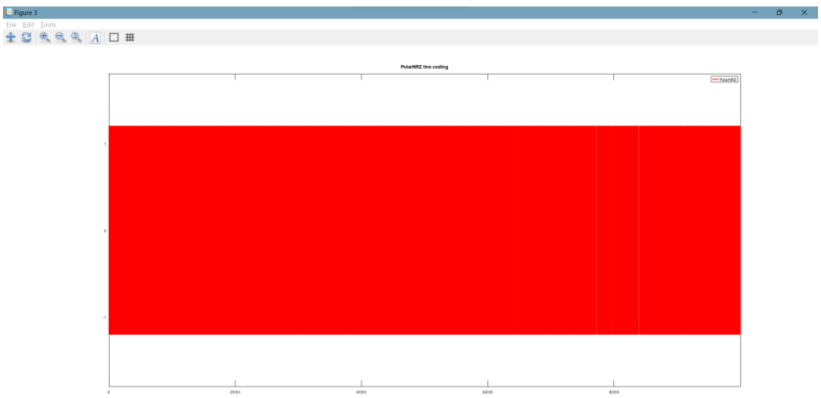


After zoom:

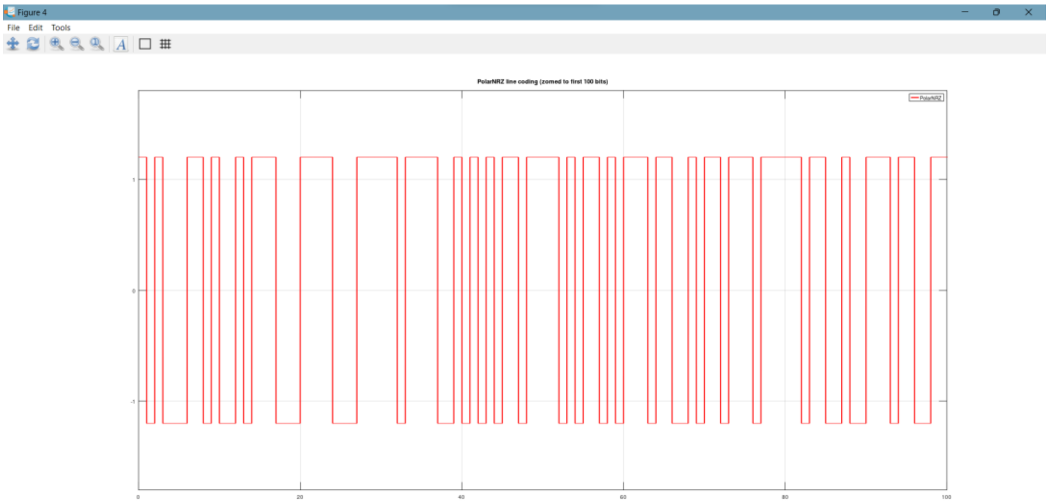


<u>Time domain plot of Polar NRZ signal:</u>

Before zoom:

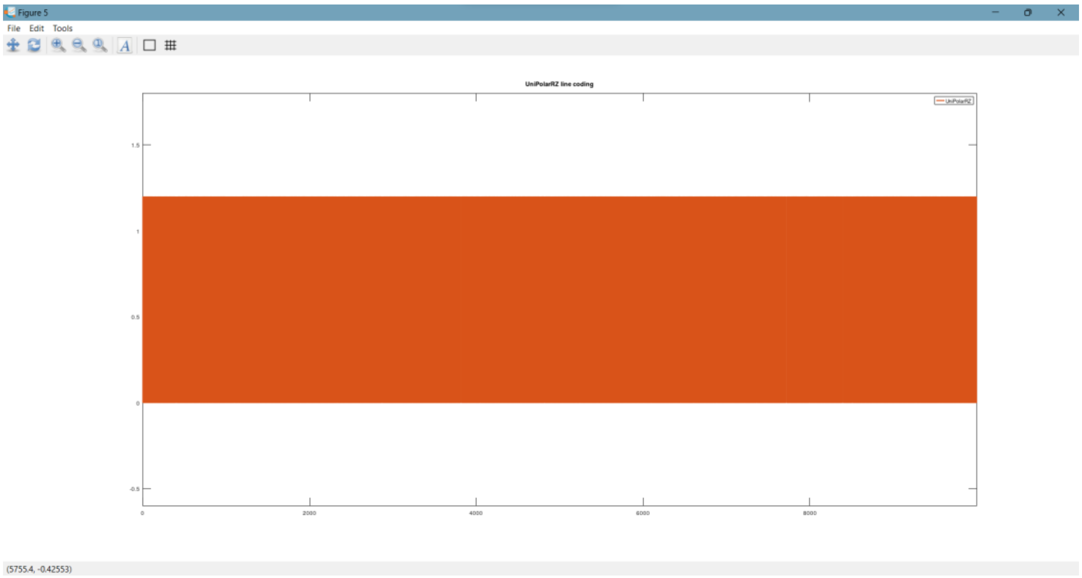


After zoom:

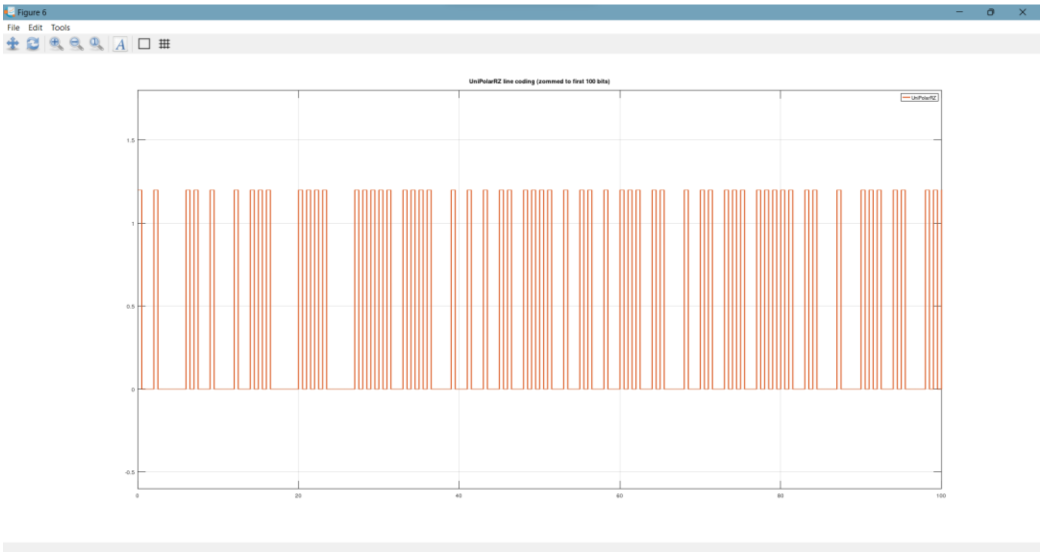


Time domain plot of UniPolar RZ signal:

Before zoom:

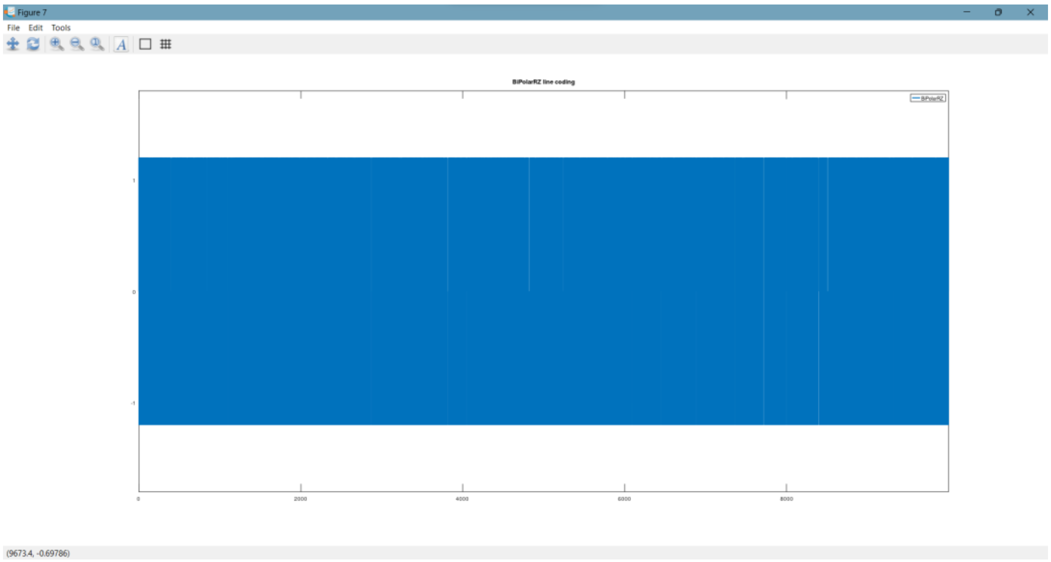


After zoom:

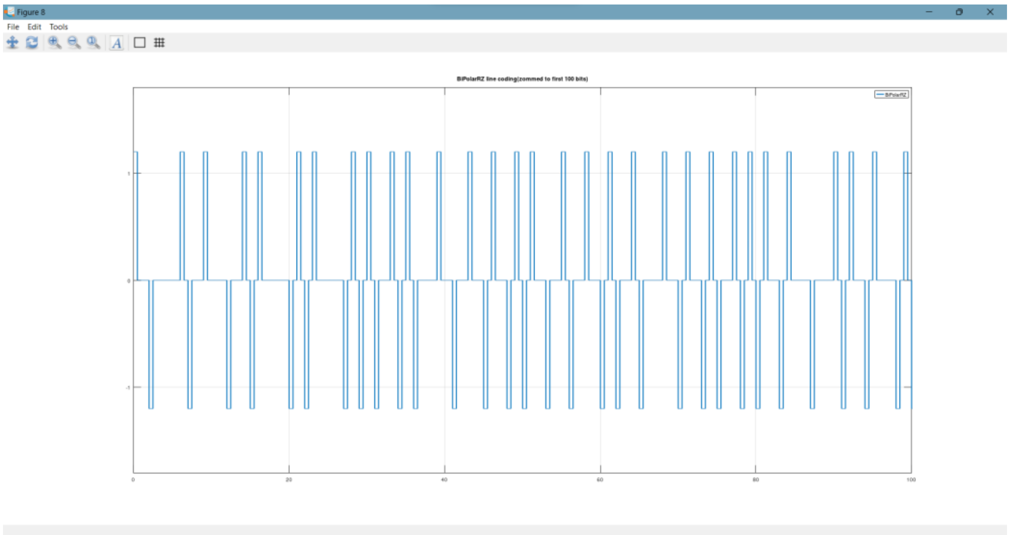


<u>Time domain plot of BiPolar RZ signal:</u>

Before zoom:

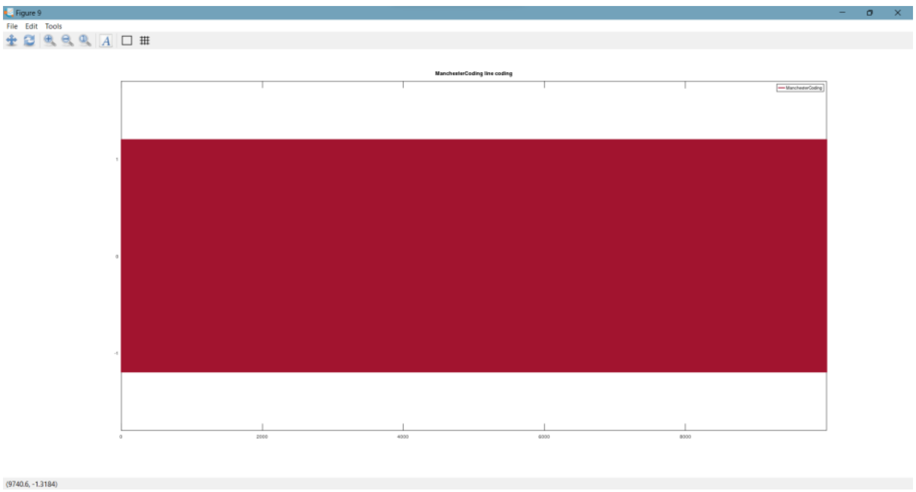


After zoom:



<u>Time domain plot of Manchester signal:</u>

Before zoom:



After zoom:

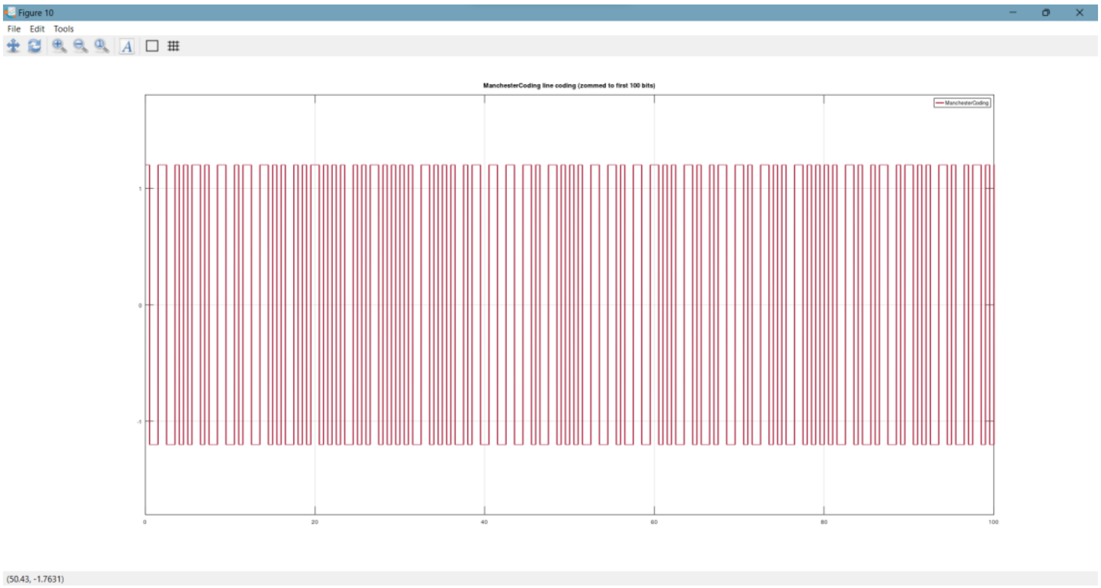


Figure 10
File Edit Tools
ManchesterCoding line coding (zommed to first 100 bits)
ManchesterCoding
1
0
-1
0
20
40
60
80
100
(50.43, -1.7631)

Step 3: Plot the Eye diagram.:

<u>Note:</u> this part is containing all the line codes required in step 7 (Uni-polar non return to zero, Polar non return to zero, Uni-polar return to zero, Bipolar return to zero and Manchester coding).

Code:

<u>Functions:</u>

File: plot_eye_diagram.m

```matlab
function plot_eye_diagram(noBitsPerSegments , noSamplePerBit , lineCodeVec , bitPeriod)
    % segment is a collection of bits we need to shift them to create the eye diagram
    % segmentLength: total samples in a segment
    segmentLength = noSamplePerBit * noBitsPerSegments;
    % periodic time of the samples
    samplePeriod = bitPeriod / noSamplePerBit;
    % time vector is created with a step of samplePeriod, and the end is the total time a segment
    % could be calculated using noBitsPerSegments and bitPeriod, with a step = samplePeriod
    timeVec = (0 : segmentLength - 1) * samplePeriod;
    % to center the zero in the mid
    timeVec = timeVec - (noBitsPerSegments * bitPeriod / 2);
    % total number of segment layers to place on top of each other
    % segmentLayers = total number of samples in line code / total number of samples in the segment
    % it's rounded down to the nearest integer, as we need to keep the size of the segment
    % equal to the size of the time vector in terms of the samples
    segmentLayers = floor(length(lineCodeVec) / segmentLength);
    for i = 1 : segmentLayers
        % find the start and end points of the segment in the line code
        % it's multiplied by the number of bits per segment
        segmentStart = (i - 1) * segmentLength + 1;
        segmentEnd = i * segmentLength;
        % x-axis is the time vector, and the y-axis is the amplitude of each bit
        plot(timeVec , lineCodeVec(segmentStart : segmentEnd),'LineWidth',2);
        % hold to plot all the segments on top of each other
        hold on;
    end
    axis([-1.5 1.5 -bitPeriod bitPeriod]);
end
```

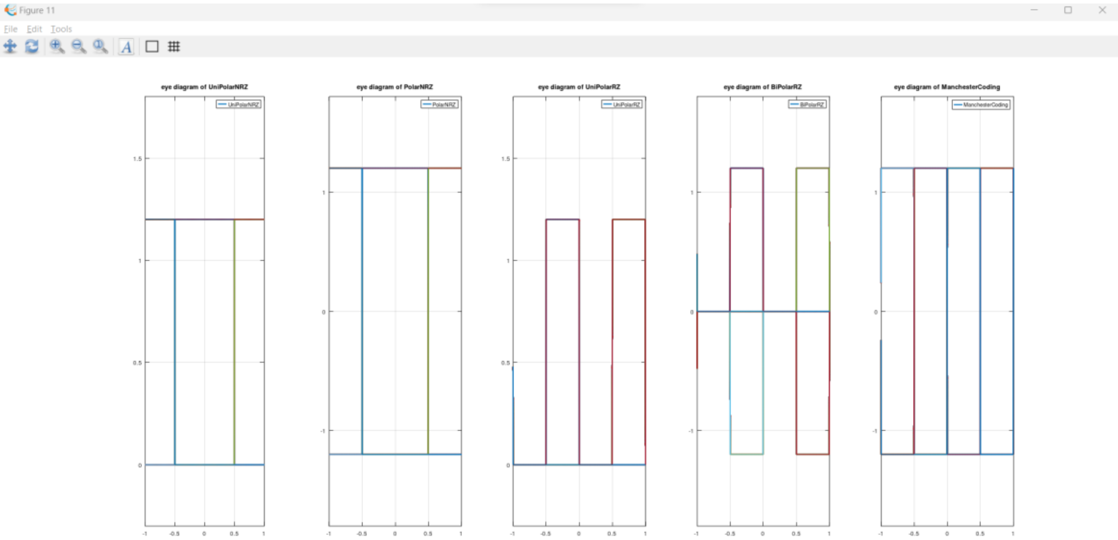
Script:

File: part1.m

```
%% plot eye diagram of line coding(transmitted signal)
% UniPolarNRZ
figure(11);
subplot(1,5,1);
plot_eye_diagram(noBitsPerSegments , noSamplesPerBit , lineCodeVec1 , bitPeriod);
title("eye diagram of UniPolarNRZ");
legend('UniPolarNRZ');
ylim([0-voltage/4 3*voltage/2]);
grid on;

% PolarNRZ
subplot(1,5,2);
plot_eye_diagram(noBitsPerSegments , noSamplesPerBit , lineCodeVec2 , bitPeriod);
title("eye diagram of PolarNRZ");
legend('PolarNRZ');
ylim([-3*voltage/2 3*voltage/2]);
grid on;

% UniPolarRZ
subplot(1,5,3);
plot_eye_diagram(noBitsPerSegments , noSamplesPerBit , lineCodeVec3 , bitPeriod);
title("eye diagram of UniPolarRZ");
legend('UniPolarRZ');
ylim([0-voltage/4 3*voltage/2]);
grid on;

% BiPolarRZ
subplot(1,5,4);
plot_eye_diagram(noBitsPerSegments , noSamplesPerBit , lineCodeVec4 , bitPeriod);
title("eye diagram of BiPolarRZ");
legend('BiPolarRZ');
ylim([-3*voltage/2 3*voltage/2]);
grid on;

% ManchesterCoding
subplot(1,5,5);
plot_eye_diagram(noBitsPerSegments , noSamplesPerBit , lineCodeVec5 , bitPeriod);
title("eye diagram of ManchesterCoding");
legend('ManchesterCoding');
ylim([-3*voltage/2 3*voltage/2]);
grid on;
```

Snapshots:

Eye-diagram of all line coding:



Step 4: Plot the spectral domains of the pulses :

Note: this part is containing all the line codes required in step 7 (Uni-polar non return to zero, Polar non return to zero, Uni-polar return to zero, Bipolar return to zero and Manchester coding).

Code:

Functions:

File: spectral_domain.m

```
function [spectral,f,BW,Spec_Original_line_coding]=spectral_domain(lineCodeVec,bit_stream,noSamplesPerBit,bitPeriod,fc)
    switch nargin
        %choose according to number of input arguments
        case 4
            [spectral,f]=Baseband_communication(lineCodeVec,bit_stream,noSamplesPerBit,bitPeriod);
        case 5
            [spectral,f,BW,Spec_Original_line_coding]=Passband_communication(lineCodeVec,bit_stream,noSamplesPerBit,bitPeriod,fc);
    end
end

%% skip into the needed function for part1

function [spectral,f]=Baseband_communication(lineCodeVec,noSamplesPerBit,bitPeriod,bit_stream)
    %calculation to generate frequency domain
    T = length(bit_stream)*bitPeriod; %simulation time
    df=1/T;  %frequency  step
    fs=noSamplesPerBit/bitPeriod; % sampling frequency
    N=noSamplesPerBit*length(bit_stream);
    %spectrum of Original Digital Signal
    Spec_Original_line_coding = (fftshift(fft(lineCodeVec)))/N;
    % frequency doomain
    if(rem(N,2)==0) %% even
        f = ((-(0.5*fs)): df : ((0.5*fs)-df));%% frequency vector if x/f even
    else %% odd
        f = (-(0.5*fs-0.5*df)) : df :(((0.5*fs)+1)-0.5*df); %% frequecy vector if X/f is odd
    end
    %power spectral of Original Digital Signal
    spectral =abs(Spec_Original_line_coding).^2;
end
```

Script:

File: part1.m

```
%% get power spectrum of line coding(transmitted signal)
[spectral_of_lineCodeVec1,f1]=spectral_domain(lineCodeVec1,noSamplesPerBit,bitPeriod,bit_stream);
[spectral_of_lineCodeVec2,f2]=spectral_domain(lineCodeVec2,noSamplesPerBit,bitPeriod,bit_stream);
[spectral_of_lineCodeVec3,f3]=spectral_domain(lineCodeVec3,noSamplesPerBit,bitPeriod,bit_stream);
[spectral_of_lineCodeVec4,f4]=spectral_domain(lineCodeVec4,noSamplesPerBit,bitPeriod,bit_stream);
[spectral_of_lineCodeVec5,f5]=spectral_domain(lineCodeVec5,noSamplesPerBit,bitPeriod,bit_stream);

%% plot power spectrum of any type of line coding
% UniPolarNRZ
figure(12);
plot(f1,spectral_of_lineCodeVec1,'color',[0.4940 0.1840 0.5560],'LineWidth',2);
title("power spectral of UniPolarNRZ");
xlabel("frequency (Hz)");
ylabel('power spectral density');
legend('UniPolarNRZ');
grid on;

% Zoomed UniPolarNRZ
figure(13);
plot(f1,spectral_of_lineCodeVec1,'color',[0.4940 0.1840 0.5560],'LineWidth',2);
title("power spectral of UniPolarNRZ (zoomed)");
xlabel("frequency (Hz)");
ylabel('power spectral density');
legend('UniPolarNRZ');
axis([-3 3 0 0.0003]);
grid on;

% PolarNRZ
figure(14);
plot(f2,spectral_of_lineCodeVec2,'color',[1 0 0],'LineWidth',2);
title("power spectral of PolarNRZ");
xlabel("frequency (Hz)");
ylabel('power spectral density');
legend('PolarNRZ');
grid on;
axis([-5 5 0 0.0012]);

% UniPolarRZ
figure(15);
```

```
plot(f3,spectral_of_lineCodeVec3,'color',[0.8500 0.3250 0.0980],'LineWidth',2);
title("power spectral of UniPolarRZ");
xlabel("frequency (Hz)");
ylabel('power spectral density');
legend('UniPolarRZ');
grid on;

% Zoomed UniPolarRZ
figure(16);
plot(f3,spectral_of_lineCodeVec3,'color',[0.8500 0.3250 0.0980],'LineWidth',2);
title("power spectral of UniPolarRZ (zoomed)");
xlabel("frequency (Hz)");
ylabel('power spectral density');
legend('UniPolarRZ');
grid on;
axis([-6 6 0 0.0001]);

% BiPolarRZ
figure(17);
plot(f4,spectral_of_lineCodeVec4,'color',[0 0.4470 0.7410],'LineWidth',2);
title("power spectral of BiPolarRZ");
xlabel("frequency (Hz)");
ylabel('power spectral density');
legend('BiPolarRZ');
axis([-15 15 0 0.00025]);
grid on;

% ManchesterCoding
figure(18);
plot(f5,spectral_of_lineCodeVec5,'color',[0.6350 0.0780 0.1840],'LineWidth',2);
title("power spectral of ManchesterCoding");
xlabel("frequency (Hz)");
ylabel('power spectral density');
legend('ManchesterCoding');
axis([-10 10 0 0.0008]);
```

Snapshots:

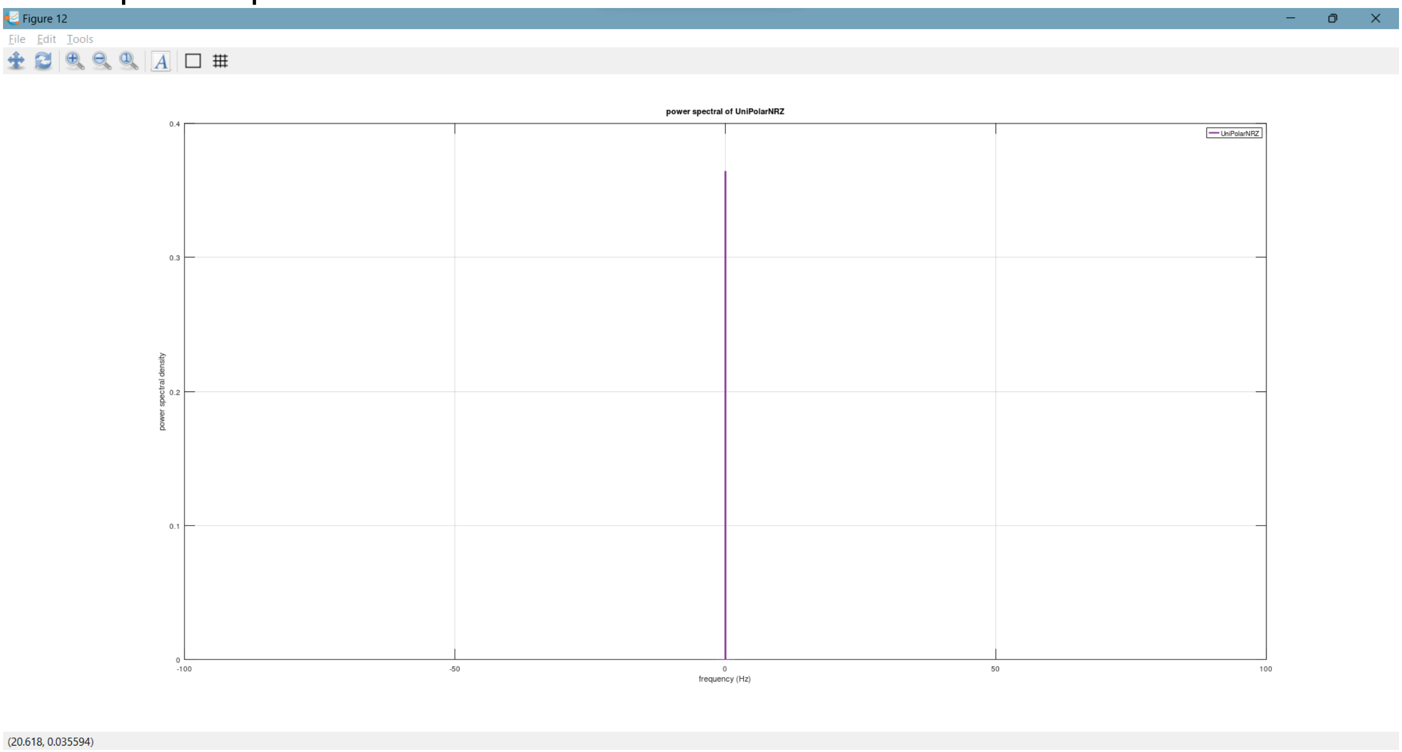
Workspace:

Workspace

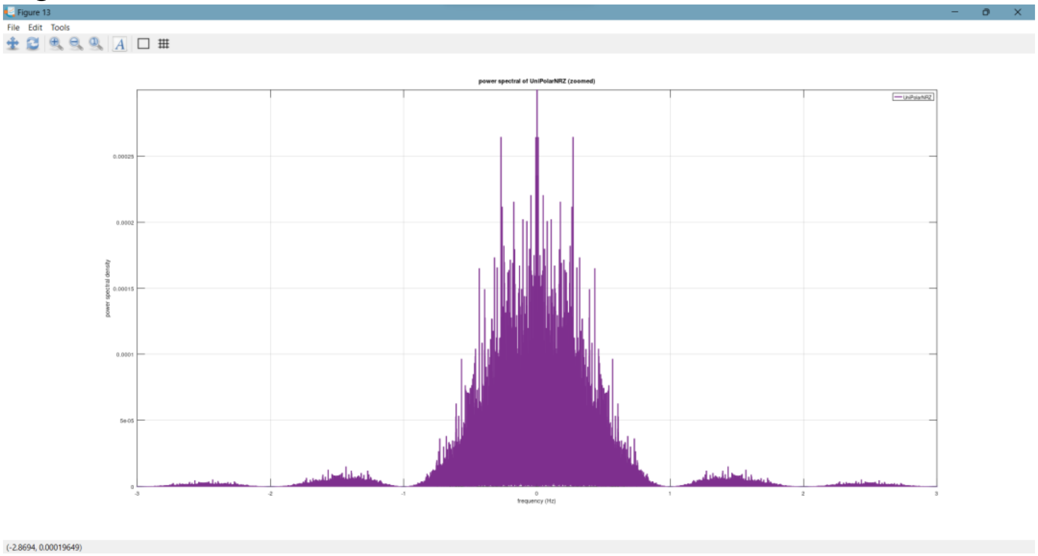
Filter

Name	Class	Dimension	Value	Attribute
bitPeriod	double	1x1	1	
bit_stream	double	1x10000	[1, 0, 1, 0, 0, 0, 1,...	
coding_scheme1	char	1x11	UniPolarNRZ	
coding_scheme2	char	1x8	PolarNRZ	
coding_scheme3	char	1x10	UniPolarRZ	
coding_scheme4	char	1x9	BiPolarRZ	
coding_scheme5	char	1x16	ManchesterCodi...	
f1	double	1x2000000	-100:0.0001:99.9...	
f2	double	1x2000000	-100:0.0001:99.9...	
f3	double	1x2000000	-100:0.0001:99.9...	
f4	double	1x2000000	-100:0.0001:99.9...	
f5	double	1x2000000	-100:0.0001:99.9...	
lineCodeVec1	double	1x2000000	[1.2000, 1.2000, ...	
lineCodeVec2	double	1x2000000	[1.2000, 1.2000, ...	
lineCodeVec3	double	1x2000000	[1.2000, 1.2000, ...	
lineCodeVec4	double	1x2000000	[1.2000, 1.2000, ...	
lineCodeVec5	double	1x2000000	[1.2000, 1.2000, ...	
noBitsPerSegme...	double	1x1	3	
noOfBits	double	1x1	10000	
noSamplesPerBit	double	1x1	200	
spectral_of_line...	double	1x2000000	[0, 4.6797e-17, ...	
spectral_of_line...	double	1x2000000	[0, 1.8719e-16, ...	
spectral_of_line...	double	1x2000000	[0, 1.1699e-17, ...	
spectral_of_line...	double	1x2000000	[0, 9.3060e-21, ...	
spectral_of_line...	double	1x2000000	[0, 4.6187e-24, ...	
timeVec1	double	1x2000000	0:0.005:10000	
timeVec2	double	1x2000000	0:0.005:10000	
timeVec3	double	1x2000000	0:0.005:10000	
timeVec4	double	1x2000000	0:0.005:10000	
timeVec5	double	1x2000000	0:0.005:10000	

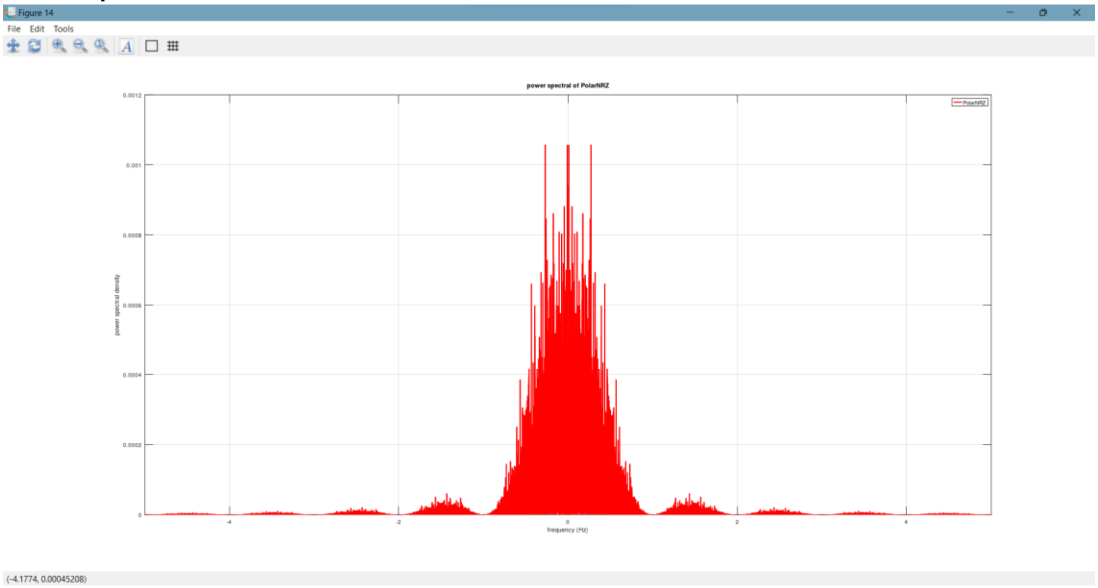
Plot of power spectral of UniPolar NRZ:



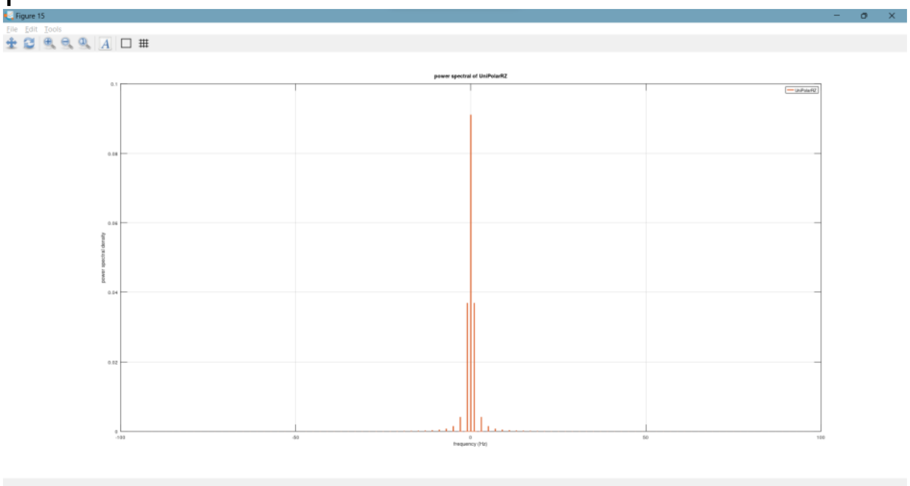
After zooming:



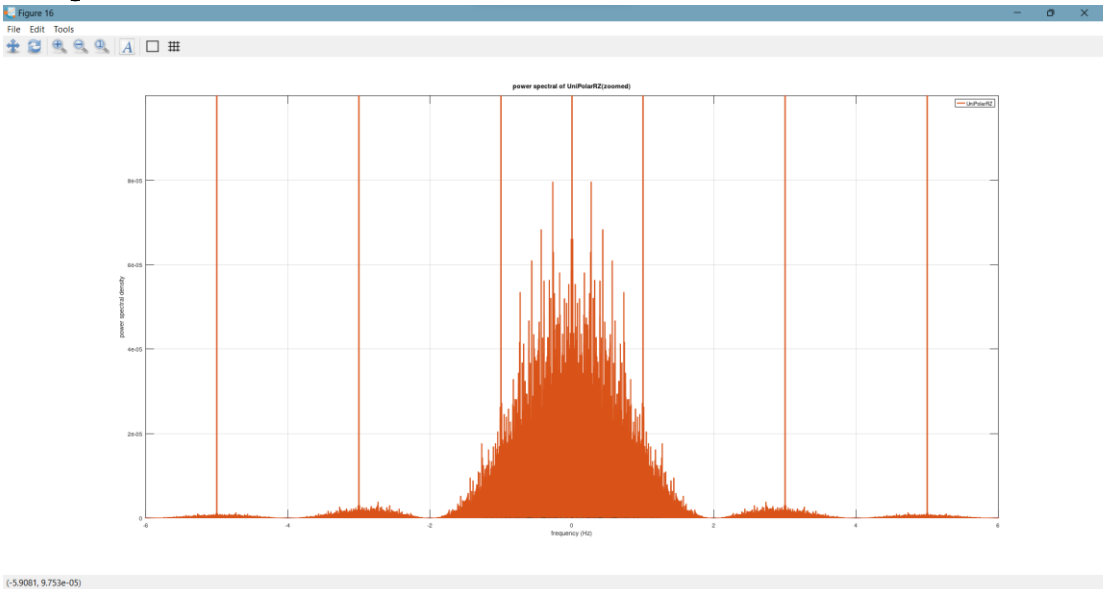
Plot of power spectral of Polar NRZ:



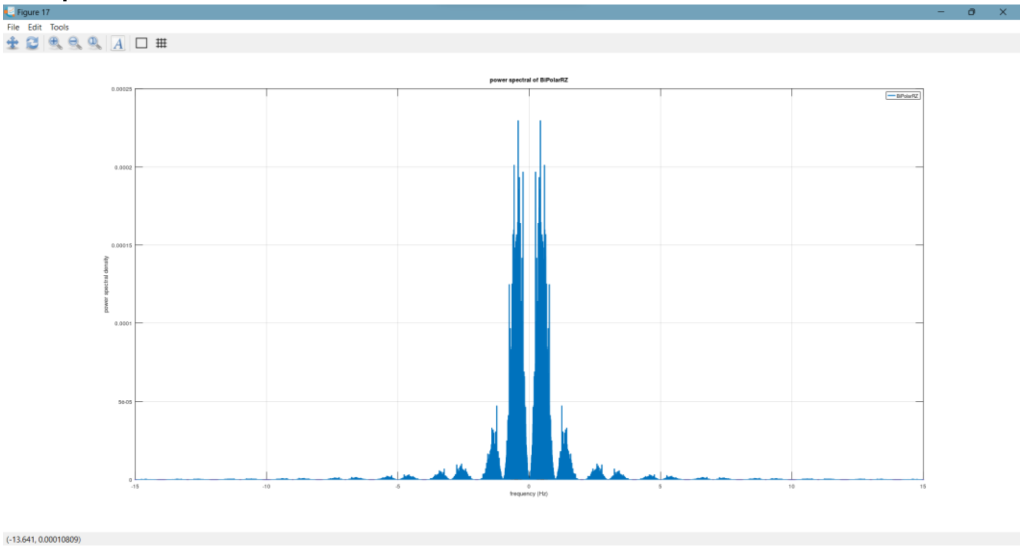
Plot of power spectral of UniPolar RZ:



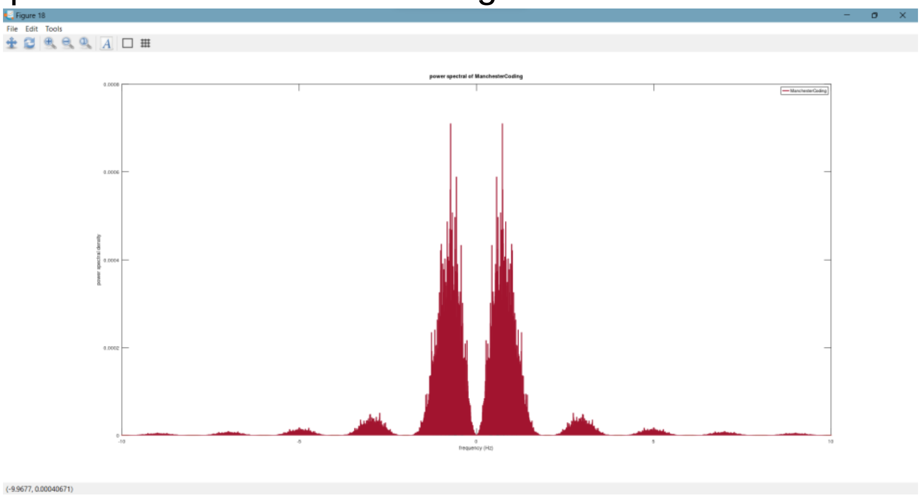
After zooming:



Plot of power spectral of BiPolar RZ:



Plot of power spectral of Manchester Coding:



Receiver:

Step 5-6-8-9-10: design the receiver and add noise to the received signal then calculate BER and error and plot BER verses sigma for each line code:

Code:

<u>Used functions:</u>

File: Sweep_on_value_of_sigma.m

```
function [BER_values, num_errors] = Sweep_on_value_of_sigma(lineCodeVec, voltage, timeVec, coding_scheme, noSamplesPerBit, noOfBits, bit_stream)
    % generate 10 ranges for sigma that range from 0 to the maximum supply voltage
    sigma_ranges = linspace(0, voltage, 10);

    % pre-allocate a vector to store the BER values & num_errors
    BER_values = zeros(1, 10);
    num_errors = zeros(1, 10);

    % loop choose each value of sigma by index i
    for i = 1:10
        % add noise to signal
        received_signal_with_noise = add_noise_to_linecoding(lineCodeVec, sigma_ranges(i), timeVec);

        % path signal through the decision device
        Reciever_output = decision_device(received_signal_with_noise, coding_scheme, voltage, timeVec, noSamplesPerBit, noOfBits);

        % this loop is used to convert Reciever output into binary data
        binary_data = convert_into_Binary_data(Reciever_output, bit_stream, noSamplesPerBit);

        % calculate the BER & num_errors for this value of sigma
        [BER_values(i), num_errors(i)] = BER_device(binary_data, bit_stream); % insert your code for calculating BER & num_errors here
    end
end
```

File: add_noise_to_linecoding.m

```
function received_signal_with_noise = add_noise_to_linecoding(lineCode, sigma, Vectime)
    % this function simulates external noise from the communication channel (telephone line) or noise from the receiver added to the transmitted line coding

    % define your time vector
    t = Vectime;

    % noise
    n = sigma * randn(1, length(t));

     % add the noise to your received signal
     received_signal_with_noise = lineCode + n;
end
```

File: decision_device.m

```
function [Reciever_output] = decision_device(received_signal_with_noise, coding_scheme, voltage, timeVec, noSamplesPerBit, noOfBits)
    % to select the type of line coding by coding scheme
    switch (coding_scheme)
        case 'UniPolarNRZ'
            Reciever_output = r_unipolarNRZ(received_signal_with_noise, voltage, timeVec, noSamplesPerBit, noOfBits);
        case 'PolarNRZ'
            Reciever_output = r_polarNRZ(received_signal_with_noise, voltage, timeVec, noSamplesPerBit, noOfBits);
        case 'UniPolarRZ'
            Reciever_output = r_unipolarRZ(received_signal_with_noise, voltage, timeVec, noSamplesPerBit, noOfBits);
         case 'BiPolarRZ'
             Reciever_output = r_bipolarRZ(received_signal_with_noise, voltage, timeVec, noSamplesPerBit, noOfBits);
         case 'ManchesterCoding'
             Reciever_output = r_manchesterCoding(received_signal_with_noise, voltage, timeVec, noSamplesPerBit, noOfBits);
     end
end

function [Reciever_output] = Master_source_and_comparator (received_signal_with_noise, noOfBits, noSamplesPerBit, timeVec, L, M, P, threshold_1, threshold_2)
    % pre-allocate a vector to store the Reciever_output
    Reciever_output = zeros(1, length(timeVec));
    switch nargin
        case 8
            for i = 1:1:noOfBits
                if (received_signal_with_noise(L) < threshold_1)
                    for k = M:1:P
                        Reciever_output(k) = 0;
                    end
                elseif (received_signal_with_noise(L) > threshold_1)
                    for k = M:1:P
```

```
                        Reciever_output(k) = 1;
                    end
                end
                M = M + noSamplesPerBit;
                P = P + noSamplesPerBit;
                L = L + noSamplesPerBit;
            end
        case 9
            for i = 1:1:noOfBits
                if (received_signal_with_noise(L) < threshold_2)
                    for k = M:1:P
                        Reciever_output(k) = 1;
                    end
                elseif (received_signal_with_noise(L) > threshold_1)
                    for k = M:1:P
                        Reciever_output(k) = 1;
                    end
                elseif (received_signal_with_noise(L) < threshold_1)
                    for k = M:1:P
                        Reciever_output(k) = 0;
                    end
                end
                M = M + noSamplesPerBit;
                P = P + noSamplesPerBit;
                L = L + noSamplesPerBit;
            end
    end
end

function [Reciever_output] = r_unipolarNRZ(received_signal_with_noise, voltage, timeVec, noSamplesPerBit, noOfBits)
    % L decision level of timing circuit
    % M, P time of bit in our time vector
    threshold = voltage / 2;
    L = noSamplesPerBit / 2;
    M = 1;
    P = noSamplesPerBit;
    [Reciever_output] = Master_source_and_comparator(received_signal_with_noise, noOfBits, noSamplesPerBit, timeVec, L, M, P, threshold);
end

function [Reciever_output] = r_polarNRZ(received_signal_with_noise, voltage, timeVec, noSamplesPerBit, noOfBits)
    % L decision level of timing circuit
    % M, P time of bit in our time vector
    threshold = (voltage + (-1 * (voltage))) / 2;
    L = noSamplesPerBit / 2;
    M = 1;
    P = noSamplesPerBit;
    Reciever_output = Master_source_and_comparator(received_signal_with_noise, noOfBits, noSamplesPerBit, timeVec, L, M, P, threshold);
```

```
end

function [Reciever_output] = r_unipolarRZ(received_signal_with_noise, voltage, timeVec, noSamplesPerBit, noOfBits)
    % L decision level of timing circuit
    % M, P time of bit in our time vector
    threshold = voltage / 2;
    L = noSamplesPerBit / 4;
    M = 1;
    P = noSamplesPerBit;
    Reciever_output = Master_source_and_comparator(received_signal_with_noise, noOfBits, noSamplesPerBit, timeVec, L, M, P, threshold);
end

function [Reciever_output] = r_bipolarRZ(received_signal_with_noise, voltage, timeVec, noSamplesPerBit, noOfBits)
    % L decision level of timing circuit
    % M, P time of bit in our time vector
    threshold_1 = voltage / 2;
    threshold_2 = (-1 * (voltage)) / 2;
    L = noSamplesPerBit / 4;
    M = 1;
    P = noSamplesPerBit;
    Reciever_output = Master_source_and_comparator(received_signal_with_noise, noOfBits, noSamplesPerBit, timeVec, L, M, P, threshold_1, threshold_2);
end

function [Reciever_output] = r_manchesterCoding(received_signal_with_noise, voltage, timeVec, noSamplesPerBit, noOfBits)
    % L decision level of timing circuit
    % M, P time of bit in our time vector
    threshold = (voltage + (-1 * (voltage))) / 2;
    L = noSamplesPerBit / 4;
    M = 1;
    P = noSamplesPerBit;
    Reciever_output = Master_source_and_comparator(received_signal_with_noise, noOfBits, noSamplesPerBit, timeVec, L, M, P, threshold);
end
```

File: convert_into_Binary_data.m

```
function [binary_data] = convert_into_Binary_data(Reciever_output, bit_stream, noSamplesPerBit)
    % pre-allocate a vector to store the binary_data
    binary_data = zeros(1, length(bit_stream));

    L = noSamplesPerBit / 2;

    for P = 1:1:length(bit_stream)
        if (Reciever_output(L) == 0)
            binary_data(P) = 0;
        elseif (Reciever_output(L) == 1)
             binary_data(P) = 1;
        end

         L = L + noSamplesPerBit;
     end
end
```

File: BER_device.m

```
function [BER, num_errors] = BER_device(Reciever_output, bit_stream)
    % count the number of errors & calculate BER
    num_errors = sum(Reciever_output ~= bit_stream);
    BER = num_errors / length(bit_stream);
end
```

<u>Script:</u>

File: part1.m

```
figure(19);
sigma_ranges = linspace(0, voltage, 10);
[BER_values1, num_errors1] = Sweep_on_value_of_sigma(lineCodeVec1, voltage, timeVec1, coding_scheme1, noSamplesPerBit, noOfBits, bit_stream);
[BER_values2, num_errors2] = Sweep_on_value_of_sigma(lineCodeVec2, voltage, timeVec2, coding_scheme2, noSamplesPerBit, noOfBits, bit_stream);
[BER_values3, num_errors3] = Sweep_on_value_of_sigma(lineCodeVec3, voltage, timeVec3, coding_scheme3, noSamplesPerBit, noOfBits, bit_stream);
[BER_values4, num_errors4] = Sweep_on_value_of_sigma(lineCodeVec4, voltage, timeVec4, coding_scheme4, noSamplesPerBit, noOfBits, bit_stream);
[BER_values5, num_errors5] = Sweep_on_value_of_sigma(lineCodeVec5, voltage, timeVec5, coding_scheme5, noSamplesPerBit, noOfBits, bit_stream);

semilogy(sigma_ranges, BER_values1, sigma_ranges, BER_values2, sigma_ranges, BER_values3, sigma_ranges, BER_values4, sigma_ranges, BER_values5, 'LineWidth', 2);

grid on;
xlabel('Sigma');
ylabel('BER');
legend({'UniPolarNRZ', 'PolarNRZ', 'UniPolarRZ', 'BiPolarRZ', 'ManchesterCoding'});
```

Snapshots:

Workspace:

Workspace

Workspace

Filter

Name	Class	Dimension	Value	Attribute
BER_values1	double	1x10	[0, 0, 0.010900, ...	
BER_values2	double	1x10	[0, 0, 0, 1.3000e-...	
BER_values3	double	1x10	[0, 0, 0.012900, ...	
BER_values4	double	1x10	[0, 0, 0.018600, ...	
BER_values5	double	1x10	[0, 0, 0, 1.0000e-...	
bitPeriod	double	1x1	1	
bit_stream	double	1x10000	[1, 0, 1, 0, 0, 0, 1,...	
coding_scheme1	char	1x11	UniPolarNRZ	
coding_scheme2	char	1x8	PolarNRZ	
coding_scheme3	char	1x10	UniPolarRZ	
coding_scheme4	char	1x9	BiPolarRZ	
coding_scheme5	char	1x16	ManchesterCodi...	
f1	double	1x2000000	-100:0.0001:99.9...	
f2	double	1x2000000	-100:0.0001:99.9...	
f3	double	1x2000000	-100:0.0001:99.9...	
f4	double	1x2000000	-100:0.0001:99.9...	
f5	double	1x2000000	-100:0.0001:99.9...	
lineCodeVec1	double	1x2000000	[1.2000, 1.2000, ...	
lineCodeVec2	double	1x2000000	[1.2000, 1.2000, ...	
lineCodeVec3	double	1x2000000	[1.2000, 1.2000, ...	
lineCodeVec4	double	1x2000000	[1.2000, 1.2000, ...	
lineCodeVec5	double	1x2000000	[1.2000, 1.2000, ...	
noBitsPerSegme...	double	1x1	3	
noOfBits	double	1x1	10000	
noSamplesPerBit	double	1x1	200	
num_errors1	double	1x10	[0, 0, 109, 657, 1...	
num_errors2	double	1x10	[0, 0, 0, 13, 113, ...	
num_errors3	double	1x10	[0, 0, 129, 646, 1...	
num_errors4	double	1x10	[0, 0, 186, 999, 1...	
num_errors5	double	1x10	[0, 0, 0, 10, 121, ...	
sigma_ranges	double	1x10	[0, 0.1333, 0.266...	
spectral_of_line...	double	1x2000000	[0, 4.6797e-17, ...	
spectral_of_line...	double	1x2000000	[0, 1.8719e-16, ...	
spectral_of_line...	double	1x2000000	[0, 1.1699e-17, ...	
spectral_of_line...	double	1x2000000	[0, 9.3060e-21, ...	
spectral_of_line...	double	1x2000000	[0, 4.6187e-24, ...	
timeVec1	double	1x2000000	0:0.005:10000	
timeVec2	double	1x2000000	0:0.005:10000	
timeVec3	double	1x2000000	0:0.005:10000	
timeVec4	double	1x2000000	0:0.005:10000	
timeVec5	double	1x2000000	0:0.005:10000	
voltage	double	1x1	1.2000	

num_errors vectors:

num_errors1 [1x10 double]

	1	2	3	4	5	6	7	8	9	10	11
1	0	0	109	657	1331	1870	2268	2546	2891	3095	
2											

num_errors2 [1x10 double]

	1	2	3	4	5	6	7	8	9	10	11
1	0	0	0	13	113	357	652	997	1324	1544	
2											

num_errors3 [1x10 double]

	1	2	3	4	5	6	7	8	9	10	11
1	0	0	129	646	1266	1850	2258	2647	2833	3099	

num_errors4 [1x10 double]

	1	2	3	4	5	6	7	8	9	10	11
1	0	0	186	999	1918	2734	3320	3794	4121	4355	

num_errors5 [1x10 double]

	1	2	3	4	5	6	7	8	9	10	11
1	0	0	0	10	121	383	659	979	1308	1574	
2											

Editor | Command Window | Documentation | Variable Editor

sigame_ranges and BER_values vectors:

sigma_ranges [1x10 double]

	1	2	3	4	5	6	7	8	9	10
1	0	0.13333	0.26667	0.4	0.53333	0.66667	0.8	0.93333	1.0667	1.2

BER_values1 [1x10 double]

	1	2	3	4	5	6	7	8	9	10
1	0	0	0.0109	0.0657	0.1331	0.187	0.2268	0.2546	0.2891	0.3095

BER_values2 [1x10 double]

	1	2	3	4	5	6	7	8	9	10	11
1	0	0	0	0.0013	0.0113	0.0357	0.0652	0.0997	0.1324	0.1544	

BER_values3 [1x10 double]

	1	2	3	4	5	6	7	8	9	10
1	0	0	0.0129	0.0646	0.1266	0.185	0.2258	0.2647	0.2833	0.3099

BER_values4 [1x10 double]

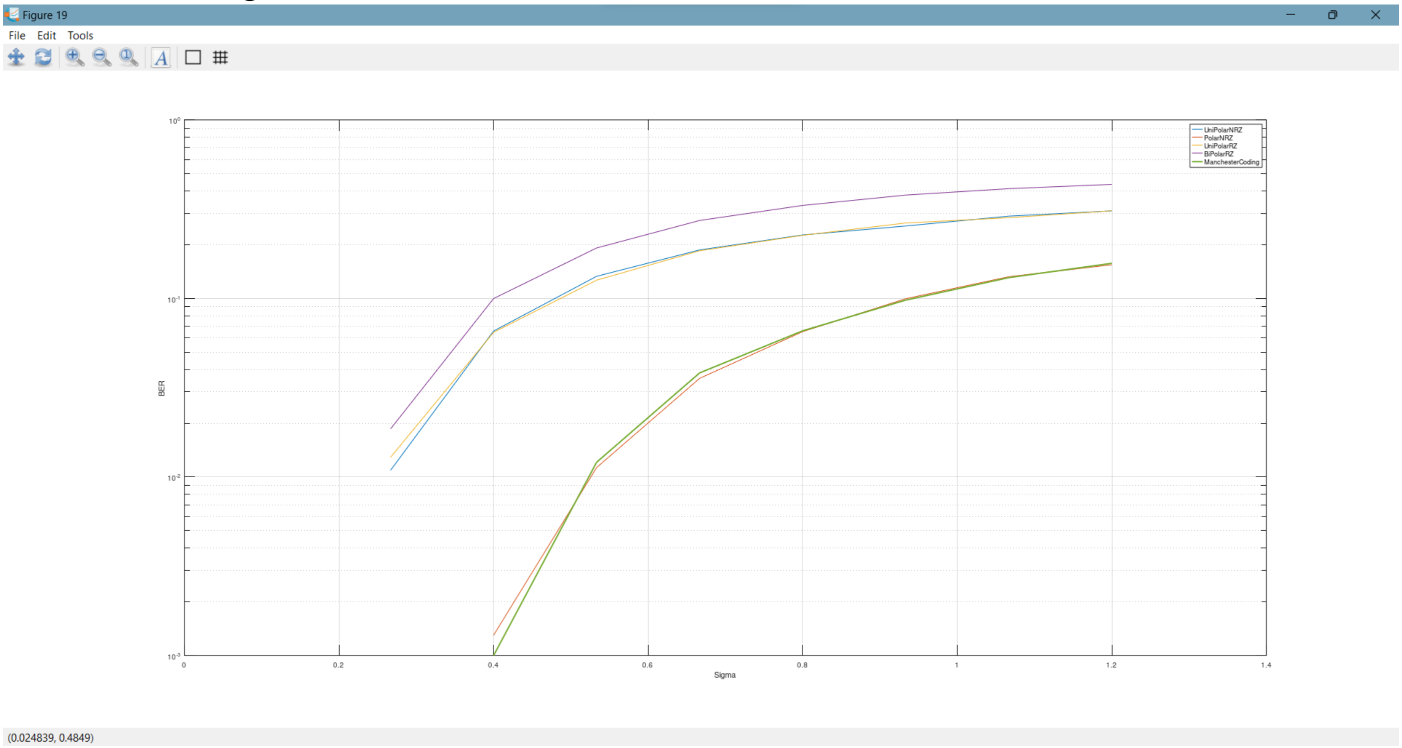
	1	2	3	4	5	6	7	8	9	10	11
1	0	0	0.0186	0.0999	0.1918	0.2734	0.332	0.3794	0.4121	0.4355	

BER_values5 [1x10 double]

	1	2	3	4	5	6	7	8	9	10	11
1	0	0	0	0.001	0.0121	0.0383	0.0659	0.0979	0.1308	0.1574	

Editor | Command Window | Documentation | Variable Editor

Plot BER vs sigma:



Step 11: **(Bonus)** For the case of Bipolar return to zero , design an error detection circuit:

Code:

Functions:

File: Bonus_Sweep_on_value_of_sigma.m

```
function [number_of_detected_errors] = Bonus_Sweep_on_value_of_sigma(lineCodeVec, voltage, timeVec, coding_scheme, noSamplesPerBit, noOfBits)
    % Generate 10 ranges for sigma that range from 0 to the maximum supply voltage
    sigma_ranges = linspace(0, voltage, 10);

    % Pre-allocate a vector to store the BER values & num_errors
    number_of_detected_errors = zeros(1, 10);

    % Loop chooses each value of sigma by index i
    for i = 1:10
        % Add noise to signal
        received_signal_with_noise = add_noise_to_linecoding(lineCodeVec, sigma_ranges(i), timeVec);

        % Path signal through Regenerative_Repeater
        [Repeater_output] = Regenerative_Repeater(received_signal_with_noise, coding_scheme, voltage, timeVec, noSamplesPerBit, noOfBits);

        % Calculate number_of_detected_errors for this value of sigma
        number_of_detected_errors(i) = Error_Detection_Circuit(Repeater_output, voltage, noSamplesPerBit, noOfBits);
    end
end
```

File: Regenerative_Repeater.m

```
function [Repeater_output] = Regenerative_Repeater(received_signal_with_noise, coding_scheme, voltage, timeVec, noSamplesPerBit, noOfBits)
    % This function represents the regenerative repeater block used in telephone lines.
    % When entering the receiver to select the type of line coding by coding scheme.
    switch (coding_scheme)
        case 'BiPolarRZ'
            Repeater_output = r_bipolarRZ(received_signal_with_noise, voltage, timeVec, noSamplesPerBit, noOfBits);
    end
end

function [Repeater_output] = r_bipolarRZ(received_signal_with_noise, voltage, timeVec, noSamplesPerBit, noOfBits)
    % Pre-allocate a vector to store the Receiver_output
    Repeater_output = zeros(1, length(timeVec));
    % L decision level of timing circuit
    % M, P time of bit in our time vector
    threshold_1 = voltage / 2;
    threshold_2 = (-1 * (voltage)) / 2;
    L = noSamplesPerBit / 4;
    M = 1;
    P = noSamplesPerBit / 2;
    for i = 1:1:noOfBits
        if (received_signal_with_noise(L) < threshold_2)
            for k = M:1:P
                Repeater_output(k) = -1 * voltage;
            end
        elseif (received_signal_with_noise(L) > threshold_1)
            for k = M:1:P
                Repeater_output(k) = voltage;
            end
        elseif (received_signal_with_noise(L) < threshold_1)
            for k = M:1:P
                Repeater_output(k) = 0;
            end
        end
        M = M + noSamplesPerBit;
        P = P + noSamplesPerBit;
        L = L + noSamplesPerBit;
    end
end
```

File: Error_Detection_Circuit.m

```
function [number_of_detected_errors] = Error_Detection_Circuit(Repeater_output, voltage, noSamplesPerBit, noOfBits)
    % Error detection circuit for Bipolar return to zero
    number_of_detected_errors = 0;
    L = noSamplesPerBit / 4;

    for i = 1:1:noOfBits
        if (L >= noOfBits * noSamplesPerBit || (L + noSamplesPerBit) >= noOfBits * noSamplesPerBit)
            break;
        end

        % 1000000000...01
        if (Repeater_output(L) == voltage && Repeater_output(L + noSamplesPerBit) == 0)
            if (L >= noOfBits * noSamplesPerBit || (L + noSamplesPerBit) >= noOfBits * noSamplesPerBit)
                break;
            else
                L = L + noSamplesPerBit;
            end

            while (Repeater_output(L) == 0)
            if (L >= noOfBits * noSamplesPerBit || (L + noSamplesPerBit) >= noOfBits * noSamplesPerBit)
                break;
            else
                L = L + noSamplesPerBit;
            end
        end

        if (Repeater_output(L) == voltage)
            number_of_detected_errors = number_of_detected_errors + 1;
        end
        % -100000000...0-1
        elseif (Repeater_output(L) == -1 * voltage && Repeater_output(L + noSamplesPerBit) == 0)
            if (L >= noOfBits * noSamplesPerBit || (L + noSamplesPerBit) >= noOfBits * noSamplesPerBit)
                break;
            else
                L = L + noSamplesPerBit;
            end

            while (Repeater_output(L) == 0)
                if (L >= noOfBits * noSamplesPerBit || (L + noSamplesPerBit) >= noOfBits * noSamplesPerBit)
                    break;
                else
                    L = L + noSamplesPerBit;
```

```
                end
            end

            if (Repeater_output(L) == -1 * voltage)
                number_of_detected_errors = number_of_detected_errors + 1;
            end

        % -1-1
        elseif (Repeater_output(L) == -1 * voltage && Repeater_output(L + noSamplesPerBit) == -1 * voltage)
            number_of_detected_errors = number_of_detected_errors + 1;
            if (L >= noOfBits * noSamplesPerBit || (L + noSamplesPerBit) >= noOfBits * noSamplesPerBit)
                break;
            else
                L = L + noSamplesPerBit;
            end

        % 11
        elseif (Repeater_output(L) == voltage && Repeater_output(L + noSamplesPerBit) == voltage)
            number_of_detected_errors = number_of_detected_errors + 1;
            if (L >= noOfBits * noSamplesPerBit || (L + noSamplesPerBit) >= noOfBits * noSamplesPerBit)
                break;
            else
                L = L + noSamplesPerBit;
            end
        end

        if (L >= noOfBits * noSamplesPerBit || (L + noSamplesPerBit) >= noOfBits * noSamplesPerBit)
            break;
        else
            L = L + noSamplesPerBit;
        end
    end
end
```

Snapshots:

Workspace:

Workspace

Filter

Name	Class	Dimension	Value	Attribute
bitPeriod	double	1x1	1	
noBitsPerSegments	double	1x1	3	
noOfBits	double	1x1	10000	
noSamplesPerBit	double	1x1	200	
voltage	double	1x1	1.2000	
BER_values1	double	1x10	[0, 0, 0.010900, ...	
BER_values2	double	1x10	[0, 0, 0, 1.3000e-...	
BER_values3	double	1x10	[0, 0, 0.012900, ...	
BER_values4	double	1x10	[0, 0, 0.018600, ...	
BER_values5	double	1x10	[0, 0, 0, 1.0000e-...	
coding_scheme3	char	1x10	UniPolarRZ	
num_errors1	double	1x10	[0, 0, 109, 657, 1...	
num_errors2	double	1x10	[0, 0, 0, 13, 113, ...	
num_errors3	double	1x10	[0, 0, 129, 646, 1...	
num_errors4	double	1x10	[0, 0, 186, 999, 1...	
num_errors5	double	1x10	[0, 0, 0, 10, 121, ...	
number_of_detected_errors	double	1x10	[0, 0, 123, 551, 9...	
sigma_ranges	double	1x10	[0, 0.1333, 0.266...	
bit_stream	double	1x10000	[1, 0, 1, 0, 0, 0, 1,...	
coding_scheme1	char	1x11	UniPolarNRZ	
coding_scheme5	char	1x16	ManchesterCodi...	
f1	double	1x2000000	-100:0.0001:99.9...	
f2	double	1x2000000	-100:0.0001:99.9...	
f3	double	1x2000000	-100:0.0001:99.9...	
f4	double	1x2000000	-100:0.0001:99.9...	
f5	double	1x2000000	-100:0.0001:99.9...	
lineCodeVec1	double	1x2000000	[1.2000, 1.2000, ...	
lineCodeVec2	double	1x2000000	[1.2000, 1.2000, ...	
lineCodeVec3	double	1x2000000	[1.2000, 1.2000, ...	
lineCodeVec4	double	1x2000000	[1.2000, 1.2000, ...	
lineCodeVec5	double	1x2000000	[1.2000, 1.2000, ...	
spectral_of_lineCodeVec1	double	1x2000000	[0, 4.6797e-17, ...	
spectral_of_lineCodeVec2	double	1x2000000	[0, 1.8719e-16, ...	
spectral_of_lineCodeVec3	double	1x2000000	[0, 1.1699e-17, ...	
spectral_of_lineCodeVec4	double	1x2000000	[0, 9.3060e-21, ...	
spectral_of_lineCodeVec5	double	1x2000000	[0, 4.6187e-24, ...	
timeVec1	double	1x2000000	0:0.005:10000	
timeVec2	double	1x2000000	0:0.005:10000	
timeVec3	double	1x2000000	0:0.005:10000	
timeVec4	double	1x2000000	0:0.005:10000	
timeVec5	double	1x2000000	0:0.005:10000	
coding_scheme2	char	1x8	PolarNRZ	
coding_scheme4	char	1x9	BiPolarRZ	

Number_of_detected_errors Vector:

number_of_detected_errors [1x10 double]

	1	2	3	4	5	6	7	8	9	10	11
1	0	0	123	551	948	1187	1465	1562	1716	1853	
2											
3											
4											

Part2: binary phase shift-keying:

Transmitter:

Step 1 : Generate stream of random bits:

Code:

<u>Used functions:</u>

File: generate_random_bits.m

```
function bit_stream = generate_random_bits(noOfBits)
    % Generate random sequence of binary data zeros and ones

    bit_stream = randi([0, 1], 1, noOfBits);
end
```

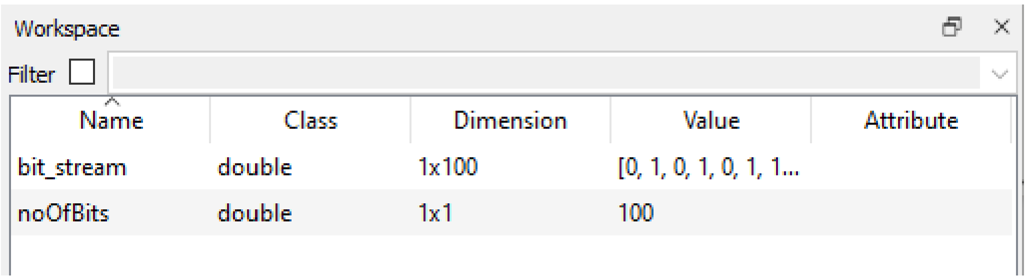
<u>Script:</u>

File: part2.m

```
% MATLAB Script for a Binary PSK (passband communication)

% Clear all variables and close all figures
clear all;
close all;

noOfBits = 100; % Number of time samples

% Generate bit stream
bit_stream = generate_random_bits(noOfBits);
```

Snapshots:

workspace:



Step 2 : Line coding:

Code:

Used functions:

File: line_coding.m

```
function [lineCodeVec, timeVec] = line_coding(bit_stream, coding_scheme, voltage, bitPeriod, noSamplesPerBit, fc)
    switch nargin
        % Choose according to number of input arguments
        case 5
            [lineCodeVec, timeVec] = Baseband_communication(bit_stream, coding_scheme, voltage, bitPeriod, noSamplesPerBit);
        case 6
            % Belongs to Part 2
            [lineCodeVec, timeVec] = Passband_communication(bit_stream, coding_scheme, voltage, bitPeriod, noSamplesPerBit, fc);
    end
end

function [lineCodeVec, timeVec] = Passband_communication(bit_stream, coding_scheme, voltage, bitPeriod, noSamplesPerBit, fc)
    % This function implements the function of polarNRZ block in transmitter

    % Calculations
    noOfBits = length(bit_stream);
    fs = 10 * fc;  % Sampling frequency - Sampling rate - This will define the resolution
    ts = 1 / fs;   % Time step
    t = 0:ts:(noOfBits - 1) * ts;
% Generate time domain
timeVec = [];
for ii = 1:1:length(bit_stream)
timeVec = [timeVec t];
t = t + (noOfBits - 1) * ts;
end

noSamplesPerBit = length(t);

% Generate polarNRZ line coding
switch (coding_scheme)
case 'PolarNRZ'
lineCodeVec = polarNRZ(bit_stream, voltage, timeVec, noSamplesPerBit);
end
end
```

<u>Script:</u>

File: part2.m

```
%% Calculations
fc = 10^9;   % Frequency of Modulating Signal
fs = 10 * fc;  % Sampling frequency - Sampling rate - This will define the resolution
ts = 1 / fs;   % Time step
t = 0:ts:(noOfBits - 1) * ts;   % Time for one bit - Time of one bit = 1 second per one bit
T = (noOfBits) * (noOfBits * ts);   % Simulation time
Rb = noOfBits / T;   % Bit rate = N / simulation time

voltage = 1;
bitPeriod = t(end);
noSamplesPerBit = length(t);
coding_scheme = 'PolarNRZ';

%% The PolarNRZ Signal
[PolarNRZ, timeVec] = line_coding(bit_stream, coding_scheme, voltage, bitPeriod, noSamplesPerBit, fc);

% Plot the PolarNRZ Signal with time
figure;
plot(timeVec, PolarNRZ, 'r', 'LineWidth', 2);
xlabel('Time (s)');
ylabel('Amplitude (V)');
title('PolarNRZ Signal');
axis([0 timeVec(end) -1.5 1.5]);
grid on;
```

Snapshots:

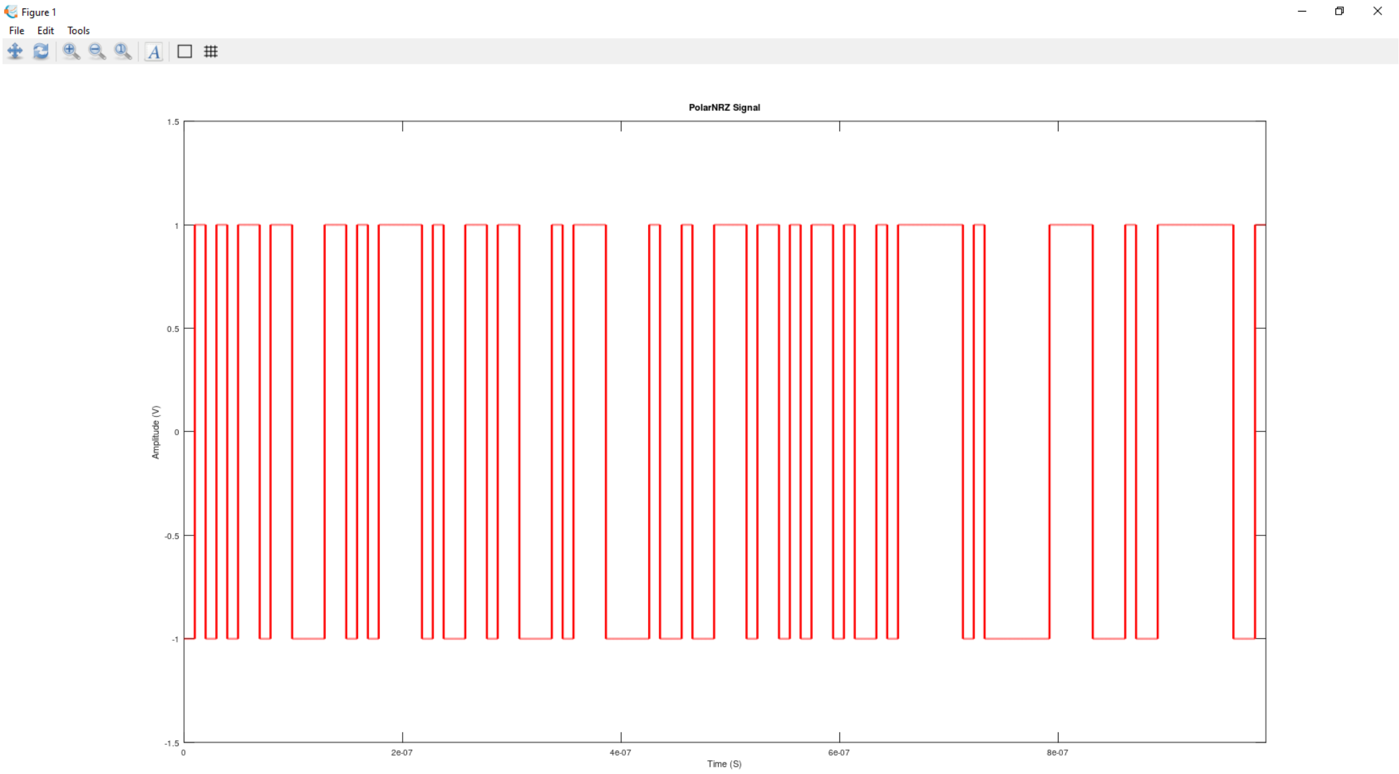
workspace:

Workspace

Filter

Name	Class	Dimension	Value	Attribute
PolarNRZ	double	1x10000	[-1, -1, -1, -1, -1...	
Rb	double	1x1	1.0000e+08	
T	double	1x1	1.0000e-06	
bitPeriod	double	1x1	9.9000e-09	
bit_stream	double	1x100	[0, 1, 0, 1, 0, 1, 1...	
coding_scheme	char	1x8	PolarNRZ	
fc	double	1x1	1.0000e+09	
fs	double	1x1	1.0000e+10	
noOfBits	double	1x1	100	
noSamplesPerBit	double	1x1	100	
t	double	1x100	0:1e-10:9.9e-09	
timeVec	double	1x10000	[0, 1.0000e-10, 2...	
ts	double	1x1	1.0000e-10	
voltage	double	1x1	1	

plot of time domain with PolarNRZ signal:



Step 3 : spectral domain before Modulation:

Code:

Used functions:

File: spectral_domain.m

```
function [spectral, f, BW, Spec_Original_line_coding] = spectral_domain(lineCodeVec, bit_stream, noSamplesPerBit, bitPeriod, fc)
    switch nargin
        % Choose according to number of input arguments
        case 4
            % Belongs to Part 1
            [spectral, f] = Baseband_communication(lineCodeVec, bit_stream, noSamplesPerBit, bitPeriod);
        case 5
            % Belongs to Part 2
            [spectral, f, BW, Spec_Original_line_coding] = Passband_communication(lineCodeVec, bit_stream, noSamplesPerBit, bitPeriod, fc);
    end
end

function [spectral, f, BW, Spec_Original_line_coding] = Passband_communication(lineCodeVec, bit_stream, noSamplesPerBit, bitPeriod, fc)
    % Calculation to generate frequency domain
    noOfBits = length(bit_stream);
    fs = 10 * fc;  % Sampling frequency - Sampling rate - This will define the resolution
    ts = 1 / fs;   % Time step
    T = (noOfBits) * (I apologize for the incomplete response in my previous message. Here's the remainder of the reformatted code starting from line 18 with line numbers:noOfBits * ts); % Simulation time
    Rb = noOfBits / T; % Bit rate = N / simulation time
    BW = Rb; % Polar & NRZ
    df = 1 / T; % Frequency step

% Spectrum of Original Digital Signal
Spec_Original_line_coding = (fftshift(fft(lineCodeVec))) / noOfBits; % We put message in frequency domain because it only shifts function in FFT

% Frequency domain
if (rem(noOfBits,2) == 0) % Even
f = ((-(0.5 * fs)):df:((0.5 * fs) - df)); % Frequency vector if x/f even
else % Odd
f = (-(0.5 * fs - 0.5 * df)) : df :(((0.5 * fs) + 1) - 0.5 * df); % Frequency vector if X/f is odd
end

% Power spectral of Original Digital Signal
spectral = abs(Spec_Original_line_codingand).^2;
end
```

Script:

File: part2.m

```
% Spectrum & Spectral of PolarNRZ Signal

[spectral_PolarNRZ_Signal, f, BW, Spec_PolarNRZ_Signal] = spectral_domain(PolarNRZ, bit_stream, noSamplesPerBit, bitPeriod, fc);

figure;
grid on;

plot(f, abs(Spec_PolarNRZ_Signal));
title('Spectrum of Original PolarNRZ Signal');
xlabel("Frequency (Hz)");
ylabel('Amplitude (V)');

figure;
plot(f, spectral_PolarNRZ_Signal);
title('Power Spectral Density of PolarNRZ Signal');
xlabel("Frequency (Hz)");
ylabel('Power Spectral Density');
```

Snapshots:

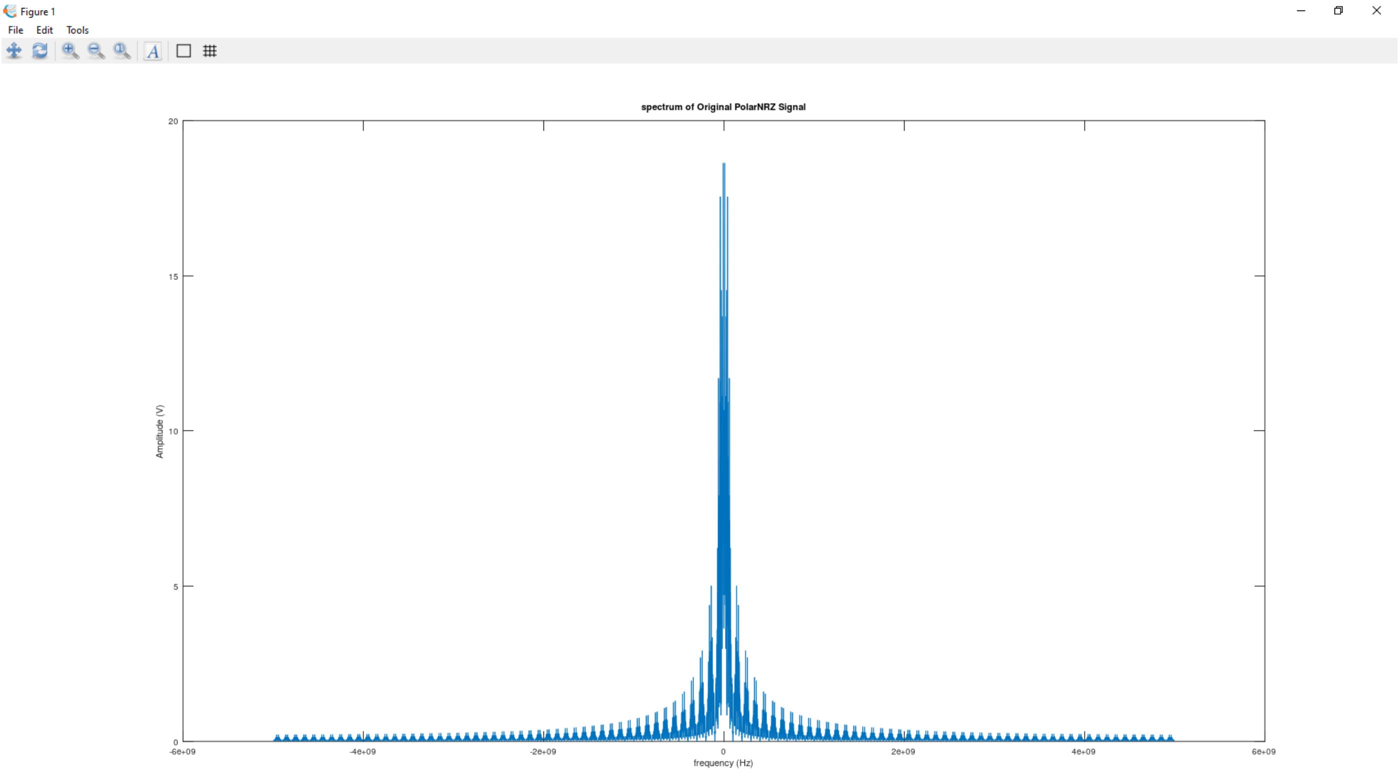
workspace:

Workspace

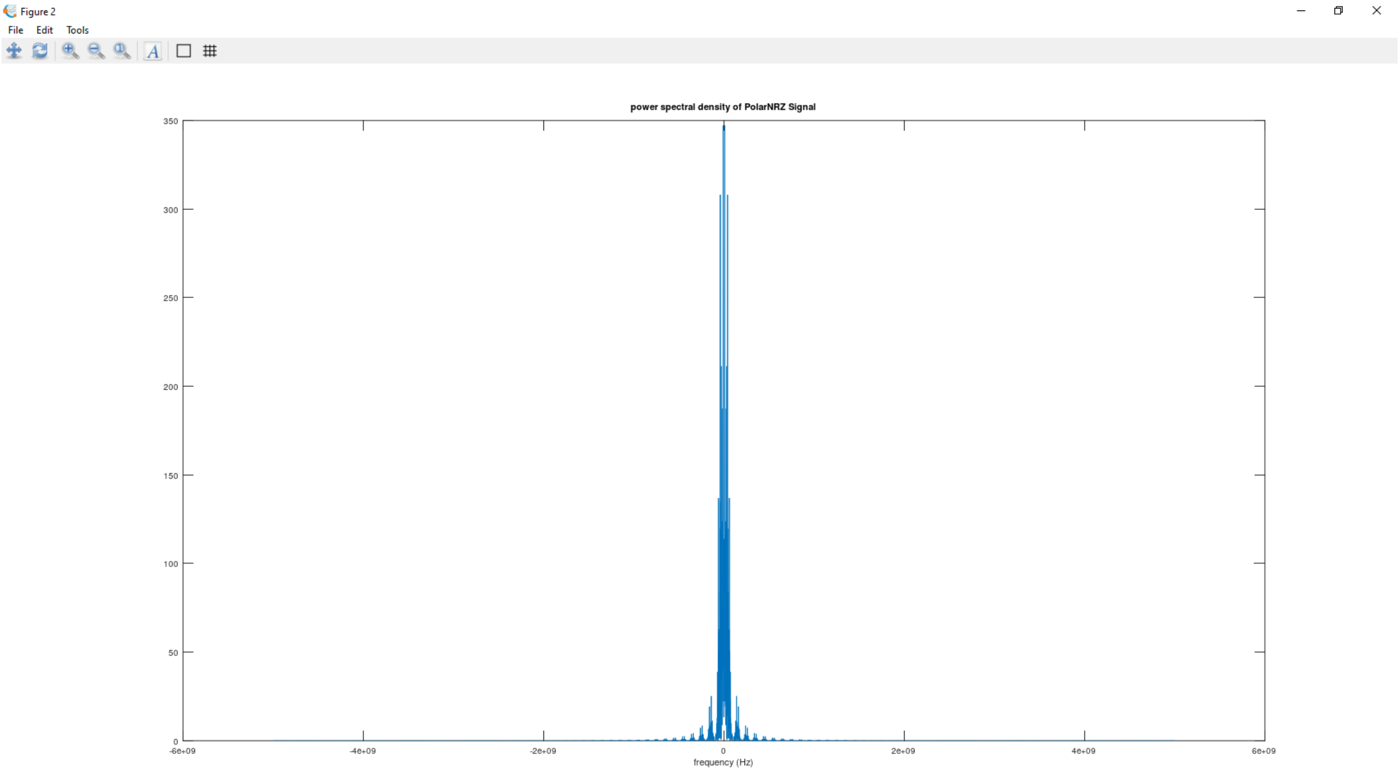
Filter

Name	Class	Dimension	Value	Attribute
BW	double	1x1	1.0000e+08	
PolarNRZ	double	1x10000	[-1, -1, -1, -1, -1...	
Rb	double	1x1	1.0000e+08	
Spec_PolarNRZ...	double	1x10000	[0 + 0i, -2.5426e...	complex
T	double	1x1	1.0000e-06	
bitPeriod	double	1x1	9.9000e-09	
bit_stream	double	1x100	[0, 1, 0, 1, 0, 1, 1...	
coding_scheme	char	1x8	PolarNRZ	
f	double	1x10000	-5e+09:1e+06:4....	
fc	double	1x1	1.0000e+09	
fs	double	1x1	1.0000e+10	
noOfBits	double	1x1	100	
noSamplesPerBit	double	1x1	100	
spectral_PolarN...	double	1x10000	[0, 1.3085e-06, 1...	
t	double	1x100	0:1e-10:9.9e-09	
timeVec	double	1x10000	[0, 1.0000e-10, 2...	
ts	double	1x1	1.0000e-10	
voltage	double	1x1	1	

plot of PolarNRZ signal's spectrum:



plot of Power spectral of PolarNRZ signal:



Step 4 : time domain after Modulation:

Code:

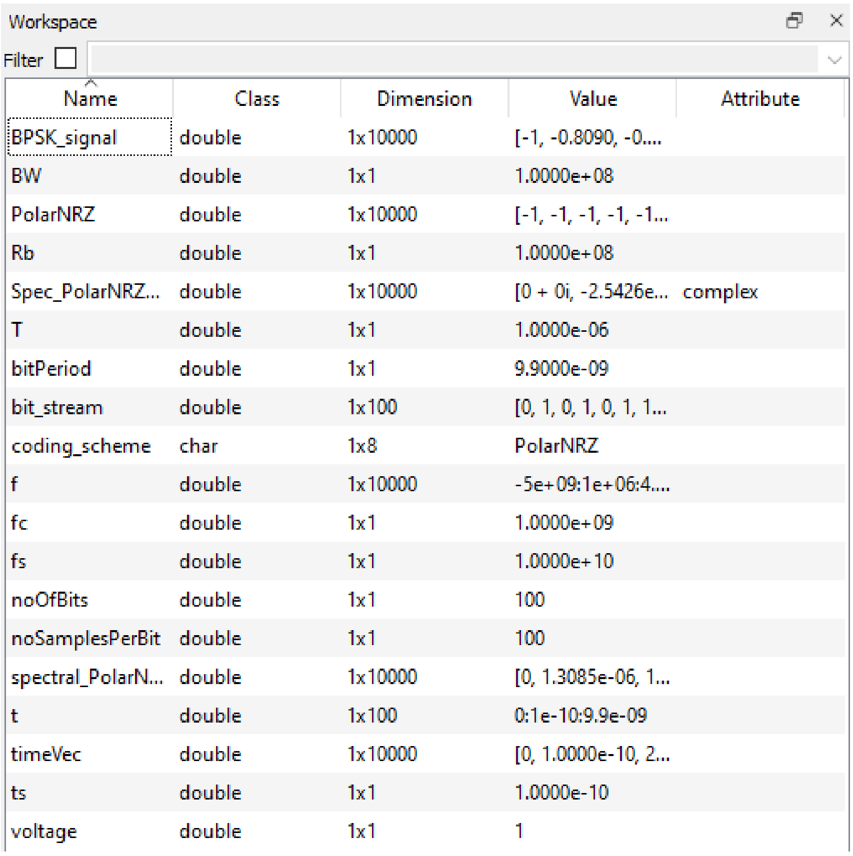
Script:

File: part2.m

```
% The BPSK Signal (modulated signal of PolarNRZ Signal after product modulator in transmitter circuit)
BPSK_signal = PolarNRZ .* cos(2 * pi * fc * timeVec);

% Plot the BPSK Signal (line coding) with time
figure;
plot(timeVec, BPSK_signal, 'LineWidth', 2);
xlabel('Time (s)');
ylabel('Amplitude (V)');
title('BPSK Signal with Two Phase Shifts');
axis([0 timeVec(end) -1.5 1.5]);
grid on;
```

Snapshots:

workspace:

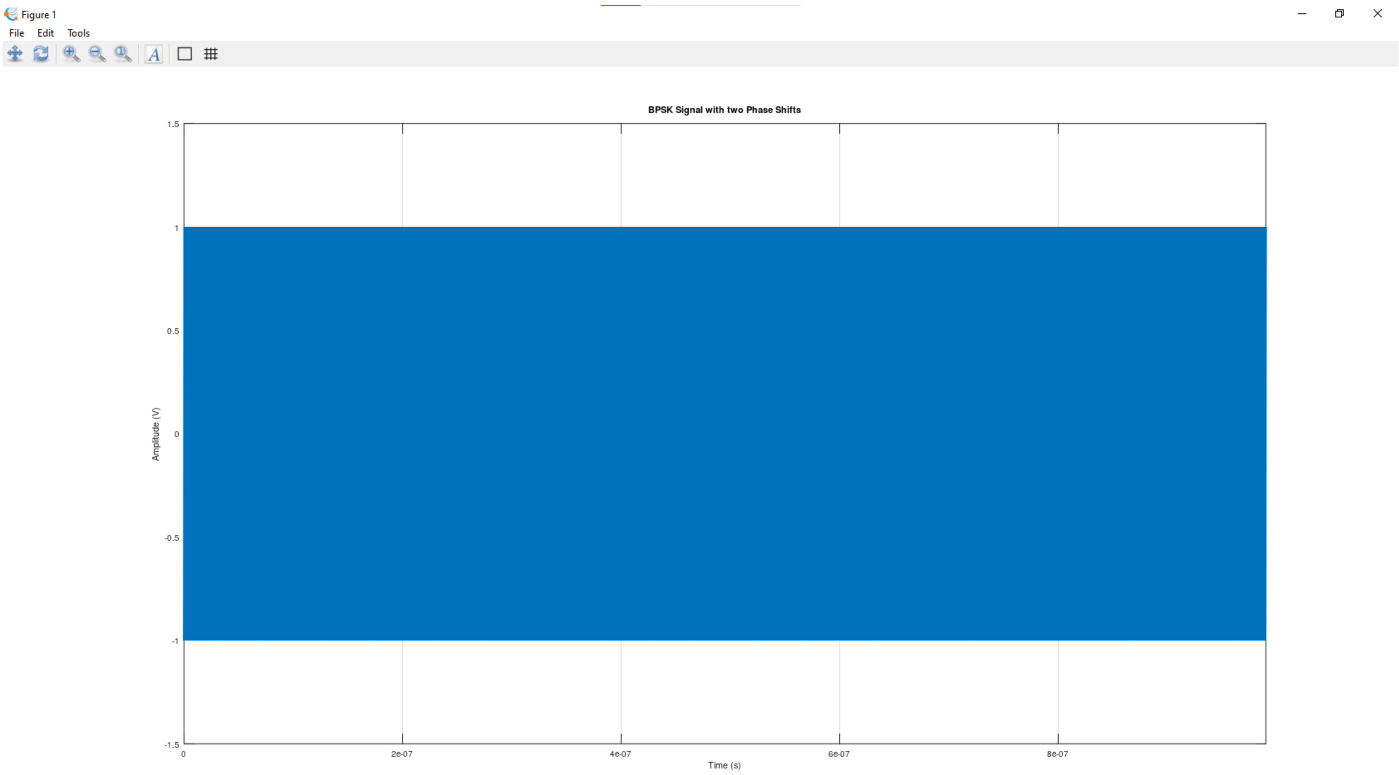


Workspace

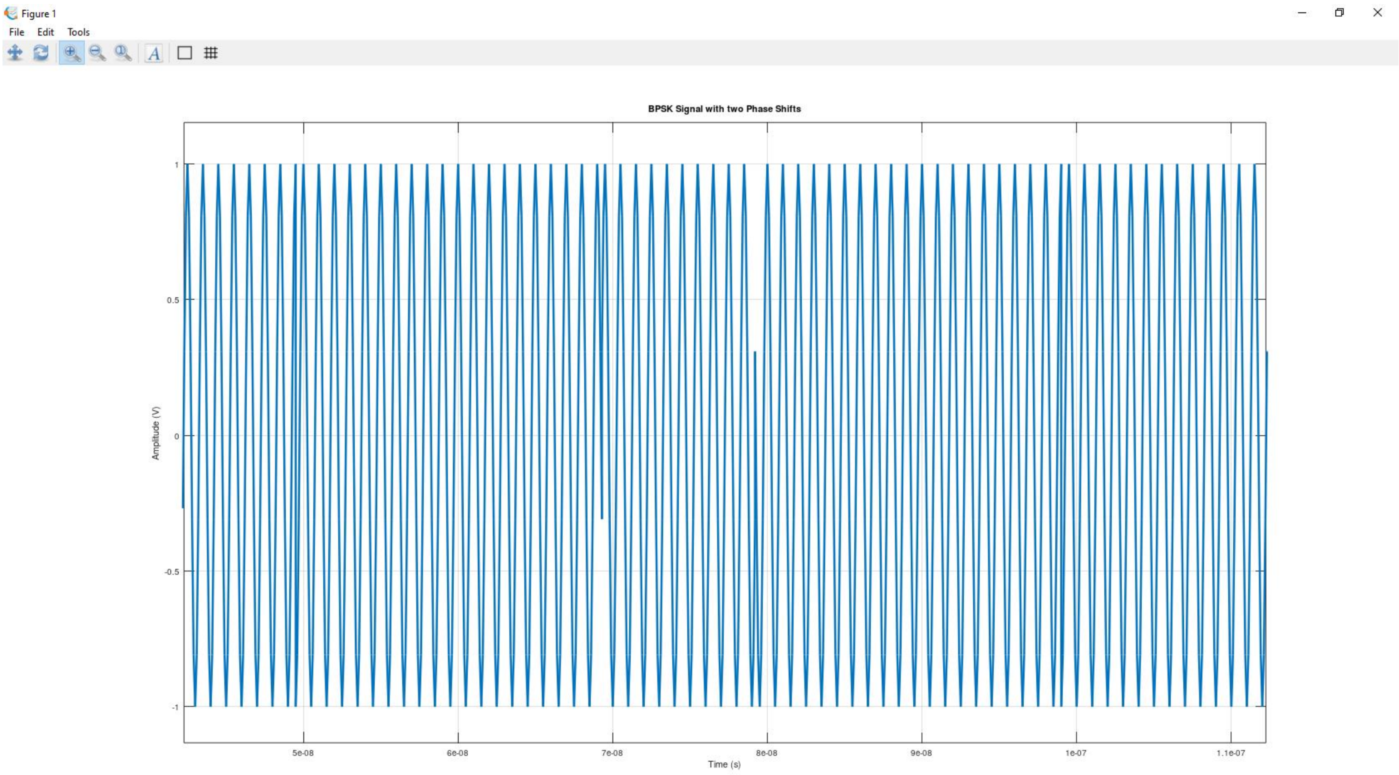
Filter

Name	Class	Dimension	Value	Attribute
BPSK_signal	double	1x10000	[-1, -0.8090, -0....	
BW	double	1x1	1.0000e+08	
PolarNRZ	double	1x10000	[-1, -1, -1, -1, -1...	
Rb	double	1x1	1.0000e+08	
Spec_PolarNRZ...	double	1x10000	[0 + 0i, -2.5426e...	complex
T	double	1x1	1.0000e-06	
bitPeriod	double	1x1	9.9000e-09	
bit_stream	double	1x100	[0, 1, 0, 1, 0, 1, 1...	
coding_scheme	char	1x8	PolarNRZ	
f	double	1x10000	-5e+09:1e+06:4....	
fc	double	1x1	1.0000e+09	
fs	double	1x1	1.0000e+10	
noOfBits	double	1x1	100	
noSamplesPerBit	double	1x1	100	
spectral_PolarN...	double	1x10000	[0, 1.3085e-06, 1...	
t	double	1x100	0:1e-10:9.9e-09	
timeVec	double	1x10000	[0, 1.0000e-10, 2...	
ts	double	1x1	1.0000e-10	
voltage	double	1x1	1	

Plot of BPSK signal with two phase shifts:
Before zoom:



After zoom:



Step 5 : spectrum of the modulated BPSK signal:

Code:

Used functions:

File: spectral_domain.m

```
function [spectral, f, BW, Spec_Original_line_coding] = spectral_domain(lineCodeVec, bit_stream, noSamplesPerBit, bitPeriod, fc)
    switch nargin
        % Choose according to number of input arguments
        case 4
            % Belongs to Part 1
            [spectral, f] = Baseband_communication(lineCodeVec, bit_stream, noSamplesPerBit, bitPeriod);
        case 5
            % Belongs to Part 2
            [spectral, f, BW, Spec_Original_line_coding] = Passband_communication(lineCodeVec, bit_stream, noSamplesPerBit, bitPeriod, fc);
    end
end

function [spectral, f, BW, Spec_Original_line_coding] = Passband_communication(lineCodeVec, bit_stream, noSamplesPerBit, bitPeriod, fc)
    % Calculation to generate frequency domain
    noOfBits = length(bit_stream);
    fs = 10 * fc;  % Sampling frequency - Sampling rate - This will define the resolution
    ts = 1 / fs;   % Time step
    T = (noOfBits) * (I apologize for the incomplete response in my previous message. Here's the remainder of the reformatted code starting from line 18 with line numbers:noOfBits * ts); % Simulation time
    Rb = noOfBits / T; % Bit rate = N / simulation time
    BW = Rb; % Polar & NRZ
    df = 1 / T; % Frequency step

% Spectrum of Original Digital Signal
Spec_Original_line_coding = (fftshift(fft(lineCodeVec))) / noOfBits; % We put message in frequency domain because it only shifts function in FFT

% Frequency domain
if (rem(noOfBits,2) == 0) % Even
f = ((-(0.5 * fs)):df:((0.5 * fs) - df)); % Frequency vector if x/f even
else % Odd
f = (-(0.5 * fs - 0.5 * df)) : df :(((0.5 * fs) + 1) - 0.5 * df); % Frequency vector if X/f is odd
end

% Power spectral of Original Digital Signal
spectral = abs(Spec_Original_line_codingand).^2;
end
```

Script:
File: part2.m

```
% Spectrum & Spectral of BPSK signal

[spectral_BPSK_signal, f, BW, spec_BPSK_signal] = spectral_domain(BPSK_signal,
bit_stream, noSamplesPerBit, bitPeriod, fc);

figure(3);
grid on;

subplot(1, 2, 1);
plot(f, abs(spec_BPSK_signal));
title("Spectrum of Modulated BPSK Signal");
xlabel("Frequency (Hz)");
ylabel('Amplitude (V)');

subplot(1, 2, 2);
plot(f, spectral_BPSK_signal);
title("Power Spectral Density of Modulated BPSK Signal");
xlabel("Frequency (Hz)");
ylabel('Power Spectral Density');
```

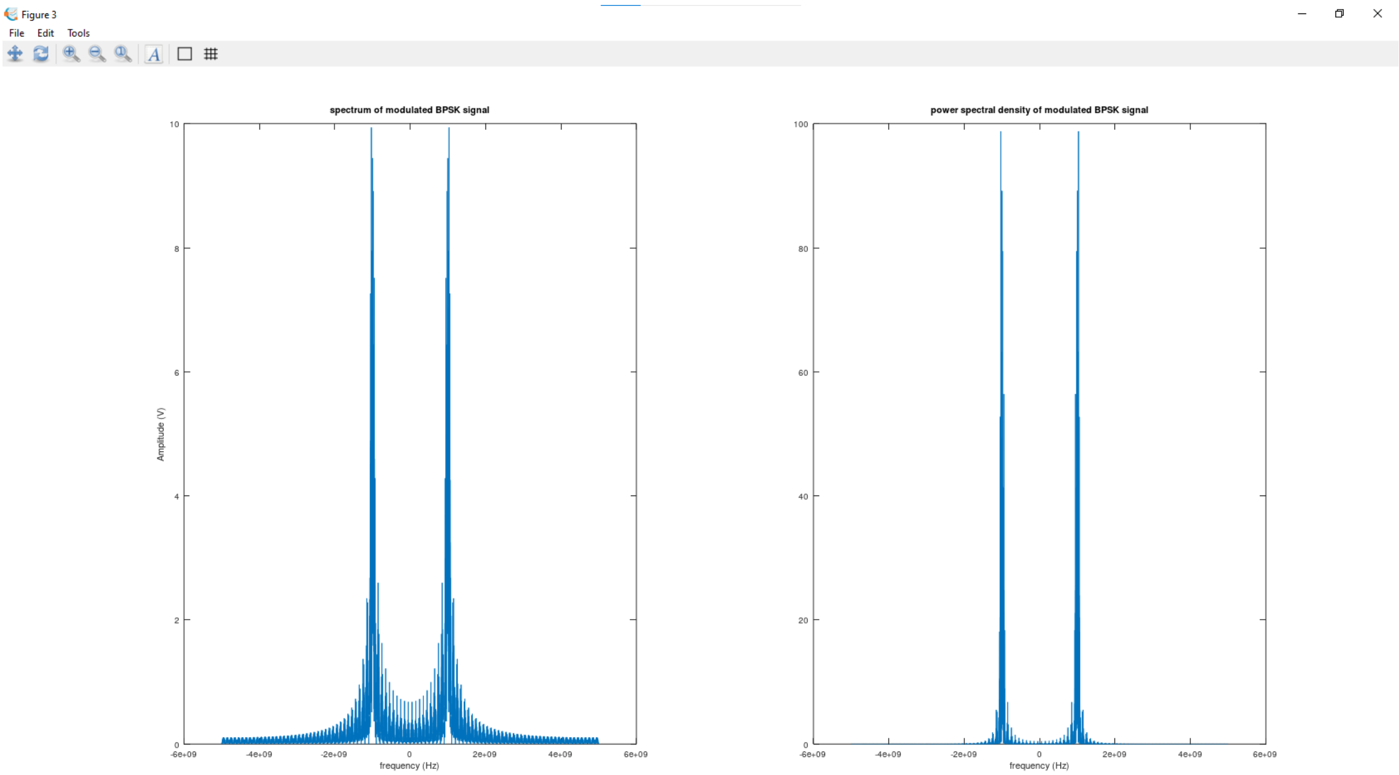
Snapshots:
workspace:

Workspace

Filter

Name	Class	Dimension	Value	Attribute
BPSK_signal	double	1x10000	[-1, -0.8090, -0....	
BW	double	1x1	1.0000e+08	
PolarNRZ	double	1x10000	[-1, -1, -1, -1, -1...	
Rb	double	1x1	1.0000e+08	
Spec_PolarNRZ...	double	1x10000	[0 + 0i, -2.5426e...	complex
T	double	1x1	1.0000e-06	
bitPeriod	double	1x1	9.9000e-09	
bit_stream	double	1x100	[0, 1, 0, 1, 0, 1, 1...	
coding_scheme	char	1x8	PolarNRZ	
f	double	1x10000	-5e+09:1e+06:4....	
fc	double	1x1	1.0000e+09	
fs	double	1x1	1.0000e+10	
noOfBits	double	1x1	100	
noSamplesPerBit	double	1x1	100	
spec_BPSK_signal	double	1x10000	[-1.0083e-13 + ...	complex
spectral_BPSK_s...	double	1x10000	[1.0166e-26, 3.1...	
spectral_PolarN...	double	1x10000	[0, 1.3085e-06, 1...	
t	double	1x100	0:1e-10:9.9e-09	
timeVec	double	1x10000	[0, 1.0000e-10, 2...	
ts	double	1x1	1.0000e-10	
voltage	double	1x1	1	

Plot of spectrum of moduled BPSK signal & Power of spectral moduled BPSK signal:



Receiver:

Step 6 : Design a receiver:

Code:

<u>Used functions:</u>

File: decision_device.m

```
function [Reciever_output] = decision_device(received_signal_with_noise, coding_scheme, voltage, timeVec, noSamplesPerBit, noOfBits)
    % to select the type of line coding by coding scheme
    switch (coding_scheme)
        case 'UniPolarNRZ'
            Reciever_output = r_unipolarNRZ(received_signal_with_noise, voltage, timeVec, noSamplesPerBit, noOfBits);
        case 'PolarNRZ'
            Reciever_output = r_polarNRZ(received_signal_with_noise, voltage, timeVec, noSamplesPerBit, noOfBits);
        case 'UniPolarRZ'
            Reciever_output = r_unipolarRZ(received_signal_with_noise, voltage, timeVec, noSamplesPerBit, noOfBits);
        case 'BiPolarRZ'
            Reciever_output = r_bipolarRZ(received_signal_with_noise, voltage, timeVec, noSamplesPerBit, noOfBits);
        case 'ManchesterCoding'
            Reciever_output = r_manchesterCoding(received_signal_with_noise, voltage, timeVec, noSamplesPerBit, noOfBits);
    end
end

function [Reciever_output] = Master_source_and_comparator(received_signal_with_noise, noOfBits, noSamplesPerBit, timeVec, L, M, P, threshold_1, threshold_2)
    % pre-allocate a vector to store the Reciever_output
    Reciever_output = zeros(1, length(timeVec));
    switch nargin
        case 8
            for i = 1:1:noOfBits
                if (received_signal_with_noise(L) < threshold_1)
                    for k = M:1:P
                        Reciever_output(k) = 0;
                    end
                elseif (received_signal_with_noise(L) > threshold_1)
                    for k = M:1:P
                        Reciever_output(k) = 1;
                    end
                end
                M = M + noSamplesPerBit;
                P = P + noSamplesPerBit;
                L = L + noSamplesPerBit;
            end
        case 9
```

```
            for i = 1:1:noOfBits
                if (received_signal_with_noise(L) < threshold_2)
                    for k = M:1:P
                        Reciever_output(k) = 1;
                    end
                elseif (received_signal_with_noise(L) > threshold_1)
                    for k = M:1:P
                        Reciever_output(k) = 1;
                    end
                elseif (received_signal_with_noise(L) < threshold_1)
                    for k = M:1:P
                        Reciever_output(k) = 0;
                    end
                end
                M = M + noSamplesPerBit;
                P = P + noSamplesPerBit;
                L = L + noSamplesPerBit;
            end
    end
end

function [Reciever_output] = r_unipolarNRZ(received_signal_with_noise, voltage, timeVec, noSamplesPerBit, noOfBits)
    % L decision level of timing circuit
    % M, P time of bit in our time vector
    threshold = voltage / 2;
    L = noSamplesPerBit / 2;
    M = 1;
    P = noSamplesPerBit;
    [Reciever_output] = Master_source_and_comparator(received_signal_with_noise, noOfBits, noSamplesPerBit, timeVec, L, M, P, threshold);
end

function [Reciever_output] = r_polarNRZ(received_signal_with_noise, voltage, timeVec, noSamplesPerBit, noOfBits)
    % L decision level of timing circuit
    % M, P time of bit in our time vector
    threshold = (voltage + (-1 * (voltage))) / 2;
    L = noSamplesPerBit / 2;
    M = 1;
    P = noSamplesPerBit;
    Reciever_output = Master_source_and_comparator(received_signal_with_noise, noOfBits, noSamplesPerBit, timeVec, L, M, P, threshold);
end

function [Reciever_output] = r_unipolarRZ(received_signal_with_noise, voltage, timeVec, noSamplesPerBit, noOfBits)
    % L decision level of timing circuit
    % M, P time of bit in our time vector
    threshold = voltage / 2;
    L = noSamplesPerBit / 4;
```

```
    M = 1;
    P = noSamplesPerBit;
    Reciever_output = Master_source_and_comparator(received_signal_with_noise,
noOfBits, noSamplesPerBit, timeVec, L, M, P, threshold);
end

function [Reciever_output] = r_bipolarRZ(received_signal_with_noise, voltage, timeVec,
noSamplesPerBit, noOfBits)
    % L decision level of timing circuit
    % M, P time of bit in our time vector
    threshold_1 = voltage / 2;
    threshold_2 = (-1 * (voltage)) / 2;
    L = noSamplesPerBit / 4;
    M = 1;
    P = noSamplesPerBit;
    Reciever_output = Master_source_and_comparator(received_signal_with_noise,
noOfBits, noSamplesPerBit, timeVec, L, M, P, threshold_1, threshold_2);
end

function [Reciever_output] = r_manchesterCoding(received_signal_with_noise, voltage,
timeVec, noSamplesPerBit, noOfBits)
    % L decision level of timing circuit
    % M, P time of bit in our time vector
    threshold = (voltage + (-1 * (voltage))) / 2;
    L = noSamplesPerBit / 4;
    M = 1;
    P = noSamplesPerBit;
    Reciever_output = Master_source_and_comparator(received_signal_with_noise,
noOfBits, noSamplesPerBit, timeVec, L, M, P, threshold);
end
```

File: convert_into_Binary_data.m

```
function
[binary_data]=convert_into_Binary_data(Reciever_output,bit_stream,noSamplesPerBit)
    % pre-allocate an vector to store the binary_data
    binary_data = zeros(1, length(bit_stream));
    L = noSamplesPerBit / 2;
    for P = 1:1:length(bit_stream)
        if (Reciever_output(L) == 0)
            binary_data(P) = 0;
        elseif (Reciever_output(L) == 1)
            binary_data(P) = 1;
         end
         L = L + noSamplesPerBit;
     end
end
```

Script:

File: part2.m

```
% demodulated signal (demodulation for BPSK signal after product modulator in recivied circuit)
demodulated_BPSK_signal = BPSK_signal .* cos(2 * pi * fc * timeVec);

% specterum & spectral of recieved signal (demodulated BPSK signal) after product modulator in reciver circuit
[spectral_demodulated_BPSK_signal,f,BW,spec_demodulated_BPSK_signal] = spectral_domain(demodulated_BPSK_signal, bit_stream, noSamplesPerBit, bitPeriod, fc);
figure(4);
grid on;
subplot(1,2,1);
plot(f, abs(spec_demodulated_BPSK_signal));
title("specterum of recieved signal after product modulator in reciver circuit ");
xlabel("frequency (Hz)");
ylabel("Amplitude (V)");
subplot(1,2,2);
plot(f, spectral_demodulated_BPSK_signal);
title("power spectral density of demodulated BPSK signal");
xlabel("frequency (Hz)");
ylabel("power spectral density ");

% LPF (Ideal)
H = abs(f) < (BW);

% specterum &spectral of demodulated BPSK signal After LPF
demodulated_BPSK_signal_After_Lpf = abs(real(H .* spec_demodulated_BPSK_signal));
figure(5);
grid on;
subplot(1,2,1);
plot(f, demodulated_BPSK_signal_After_Lpf);
title("spectrum of demodulated BPSK signal After Lpf");
xlabel("frequency (Hz)");
ylabel("Amplitude (V)");
subplot(1,2,2);
plot(f, (demodulated_BPSK_signal_After_Lpf .^ 2));
title("power spectral density of demodulated BPSK signal After Lpf");
xlabel("frequency (Hz)");
ylabel("power spectral density");

% demodulated BPSK signal AFTER LPF in time domain
demodulated_BPSK_signal_After_Lpf_t = real(ifft(fftshift(H .* spec_demodulated_BPSK_signal) * noOfBits)); %1/N

figure(6);
grid on;
subplot(2,1,1);
plot(timeVec, demodulated_BPSK_signal_After_Lpf_t, 'LineWidth', 2);
axis([0 timeVec(end) -1*voltage voltage]);
title("demodulated BPSK signal After LPF in time domain");
```

```
xlabel("time(s)");
ylabel("Amplitude (V)");

% decision device circuit give wave form in ones and zeros and remove effect of LPF
[Reciever_output] = decision_device(demodulated_BPSK_signal_After_Lpf_t,
coding_scheme, voltage, timeVec, noSamplesPerBit, noOfBits);

subplot(2,1,2);
plot(timeVec, Reciever_output, 'color', [0.6350 0.0780 0.1840], 'LineWidth', 2);
axis([0 timeVec(end) 0 voltage]);
title("Reciever output");
xlabel("time(s)");
ylabel("Amplitude (V)");

% return reciever output in form of binary data
binary_data = convert_into_Binary_data(Reciever_output, bit_stream, noSamplesPerBit);
```

Snapshots:

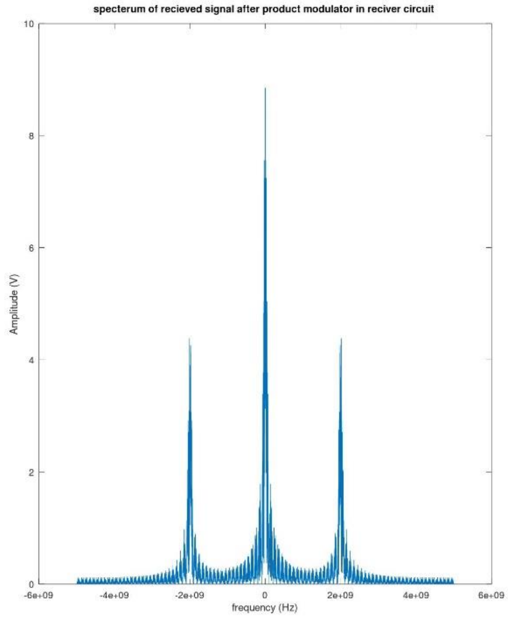
workspace:

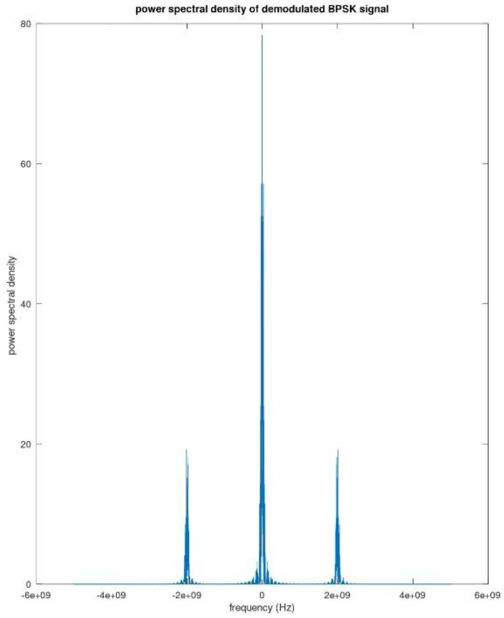
Workspace

Filter

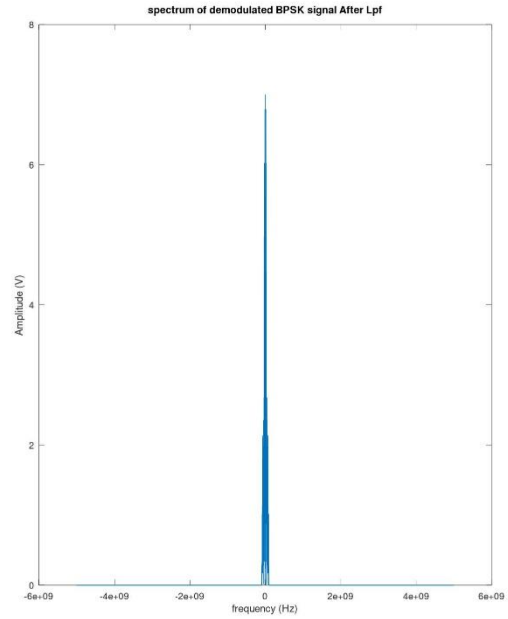
Name	Class	Dimension	Value	Attribute
BPSK_signal	double	1x10000	[-1, -0.8090, -0....	
BW	double	1x1	1.0000e+08	
H	logical	1x10000	[0, 0, 0, 0, 0, 0, 0...	
PolarNRZ	double	1x10000	[-1, -1, -1, -1, -1...	
Rb	double	1x1	1.0000e+08	
Reciever_output	double	1x10000	[0, 0, 0, 0, 0, 0, 0...	
Spec_PolarNRZ...	double	1x10000	[0 + 0i, -2.5426e...	complex
T	double	1x1	1.0000e-06	
binary_data	double	1x100	[0, 1, 0, 1, 0, 1, 1...	
bitPeriod	double	1x1	9.9000e-09	
bit_stream	double	1x100	[0, 1, 0, 1, 0, 1, 1...	
coding_scheme	char	1x8	PolarNRZ	
demodulated_B...	double	1x10000	[-1, -0.6545, -0....	
demodulated_B...	double	1x10000	[0, 0, 0, 0, 0, 0, 0...	
demodulated_B...	double	1x10000	[-0.010618, -0.0...	
f	double	1x10000	-5e+09:1e+06:4....	
fc	double	1x1	1.0000e+09	
fs	double	1x1	1.0000e+10	
noOfBits	double	1x1	100	
noSamplesPerBit	double	1x1	100	
spec_BPSK_signal	double	1x10000	[-1.0083e-13 + ...	complex
spec_demodula...	double	1x10000	[-7.3612e-14 + ...	complex
spectral_BPSK_s...	double	1x10000	[1.0166e-26, 3.1...	
spectral_PolarN...	double	1x10000	[0, 1.3085e-06, 1...	
t	double	1x100	0:1e-10:9.9e-09	
timeVec	double	1x10000	[0, 1.0000e-10, 2...	
ts	double	1x1	1.0000e-10	
voltage	double	1x1	1	

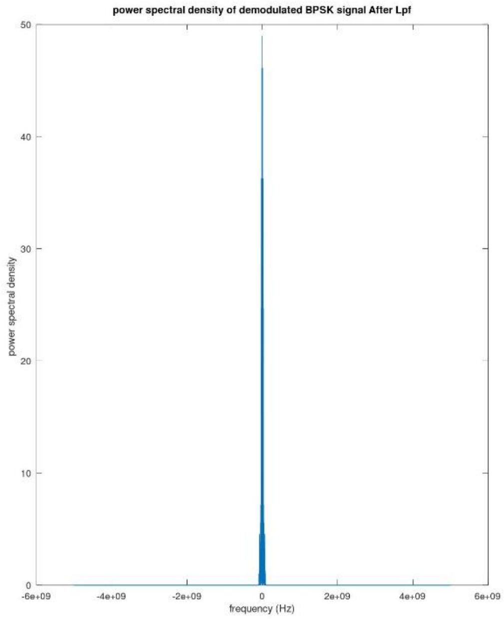
plot of spectrum of moduled signal & Power spectral demoduled BPSK signal:



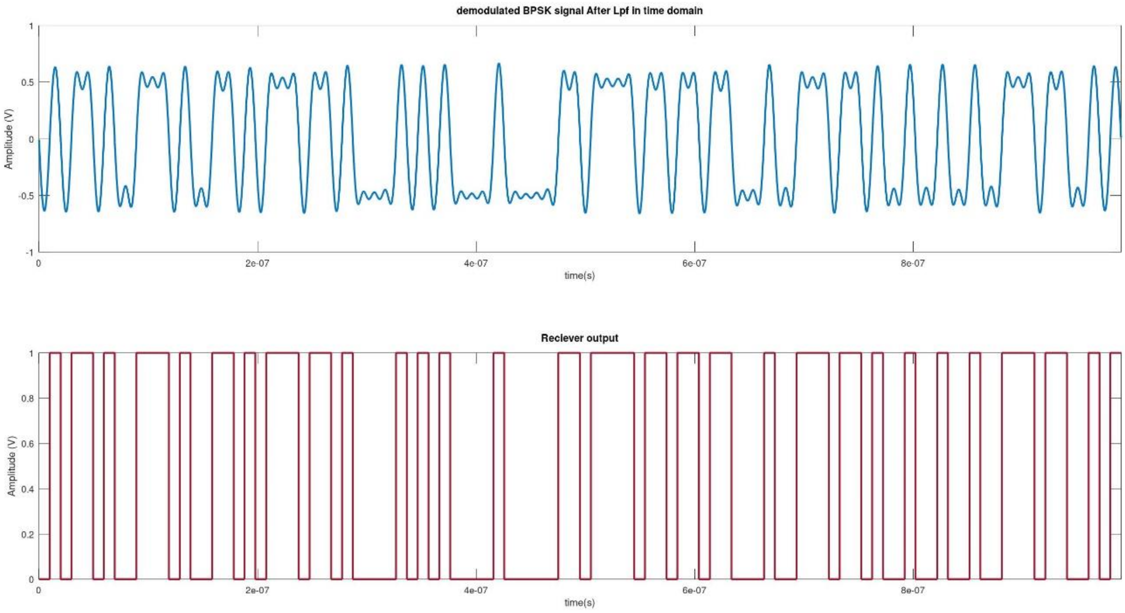


plot of spectrum of BPSK signal after LPF& Power spectral BPSK signal after LPF:





plot of demodulated BPSK signal after LPF in time domain& Output BPSK signal:



Step 7: calculate bit error rate (BER):

Code:

Used functions:

File: BER_device.m

```
function [BER, num_errors] = BER_device(Reciever_output, bit_stream)
    % count the number of errors & calculate BER
    num_errors = sum(Reciever_output ~= bit_stream);
    BER = num_errors / length(bit_stream);
end
```

Script:

File: part2.m

```
% calculate BER & num_errors
[BER, num_errors] = BER_device(binary_data, bit_stream);
```

Snapshots:

workspace:

Workspace

Filter

Name	Class	Dimension	Value	Attribute
BER	double	1x1	0	
BPSK_signal	double	1x10000	[-1, -0.8090, -0....	
BW	double	1x1	1.0000e+08	
H	logical	1x10000	[0, 0, 0, 0, 0, 0, 0...	
PolarNRZ	double	1x10000	[-1, -1, -1, -1, -1...	
Rb	double	1x1	1.0000e+08	
Reciever_output	double	1x10000	[0, 0, 0, 0, 0, 0, 0...	
Spec_PolarNRZ...	double	1x10000	[0 + 0i, -2.5426e...	complex
T	double	1x1	1.0000e-06	
binary_data	double	1x100	[0, 1, 0, 1, 0, 1, 1...	
bitPeriod	double	1x1	9.9000e-09	
bit_stream	double	1x100	[0, 1, 0, 1, 0, 1, 1...	
coding_scheme	char	1x8	PolarNRZ	
demodulated_B...	double	1x10000	[-1, -0.6545, -0....	
demodulated_B...	double	1x10000	[0, 0, 0, 0, 0, 0, 0...	
demodulated_B...	double	1x10000	[-0.010618, -0.0...	
f	double	1x10000	-5e+09:1e+06:4....	
fc	double	1x1	1.0000e+09	
fs	double	1x1	1.0000e+10	
noOfBits	double	1x1	100	
noSamplesPerBit	double	1x1	100	
num_errors	double	1x1	0	
spec_BPSK_signal	double	1x10000	[-1.0083e-13 + ...	complex
spec_demodula...	double	1x10000	[-7.3612e-14 + ...	complex
spectral_BPSK_s...	double	1x10000	[1.0166e-26, 3.1...	
spectral_PolarN...	double	1x10000	[0, 1.3085e-06, 1...	
t	double	1x100	0:1e-10:9.9e-09	
timeVec	double	1x10000	[0, 1.0000e-10, 2...	
ts	double	1x1	1.0000e-10	
voltage	double	1x1	1	

Code as text:

Script Part 1:

part1.m

```
% Clear all variables and close all figures
clear all;
close all;

%set values
voltage=1.2;

noOfBits=10000;
bitPeriod=1;
noSamplesPerBit=200;
noBitsPerSegments=3;

coding_scheme1='UniPolarNRZ';
coding_scheme2='PolarNRZ';
coding_scheme3='UniPolarRZ';
coding_scheme4='BiPolarRZ';
coding_scheme5='ManchesterCoding';


% generate bit_stream
bit_stream = generate_random_bits( noOfBits );

[lineCodeVec1, timeVec1] = line_coding(bit_stream, coding_scheme1, voltage, bitPeriod, noSamplesPerBit);
[lineCodeVec2, timeVec2] = line_coding(bit_stream, coding_scheme2, voltage, bitPeriod, noSamplesPerBit);
[lineCodeVec3, timeVec3] = line_coding(bit_stream, coding_scheme3, voltage, bitPeriod, noSamplesPerBit);
[lineCodeVec4, timeVec4] = line_coding(bit_stream, coding_scheme4, voltage, bitPeriod, noSamplesPerBit);
[lineCodeVec5, timeVec5] = line_coding(bit_stream, coding_scheme5, voltage, bitPeriod, noSamplesPerBit);


%% plot 5 line coding in time domain
% UniPolarNRZ line coding
figure(1);
plot(timeVec1, lineCodeVec1, 'color', [0.4940 0.1840 0.5560], 'LineWidth', 2);
axis([0 timeVec1(end) 0 - voltage/2 voltage*3/2]);
title("UniPolarNRZ line coding");
legend('UniPolarNRZ');

% Zoomed
figure(2);
```

```
plot(timeVec1, lineCodeVec1, 'color', [0.4940 0.1840 0.5560], 'LineWidth', 2);
axis([0 100*bitPeriod 0 - voltage/2 voltage*3/2]);
title("UniPolarNRZ line coding (zoomed to first 100 bits)");
legend('UniPolarNRZ');
grid on;


% PolarNRZ line coding
figure(3);
plot(timeVec2, lineCodeVec2, 'color', [1 0 0], 'LineWidth', 2);
axis([0 timeVec2(end) -3*voltage/2 3*voltage/2]);
title("PolarNRZ line coding");
legend('PolarNRZ');

% Zoomed
figure(4);
plot(timeVec2, lineCodeVec2, 'color', [1 0 0], 'LineWidth', 2);
axis([0 100*bitPeriod -3*voltage/2 3*voltage/2]);
title("PolarNRZ line coding (zoomed to first 100 bits)");
legend('PolarNRZ');
grid on;


% UniPolarRZ line coding
figure(5);
plot(timeVec3, lineCodeVec3, 'color', [0.8500 0.3250 0.0980], 'LineWidth', 2);
axis([0 timeVec3(end) 0 - voltage/2 voltage*3/2]);
title("UniPolarRZ line coding");
legend('UniPolarRZ');

% Zoomed
figure(6);
plot(timeVec3, lineCodeVec3, 'color', [0.8500 0.3250 0.0980], 'LineWidth', 2);
axis([0 100*bitPeriod 0 - voltage/2 voltage*3/2]);
title("UniPolarRZ line coding (zoomed to first 100 bits)");
legend('UniPolarRZ');

% BiPolarRZ line coding
figure(7);
plot(timeVec4, lineCodeVec4, 'color', [0 0.4470 0.7410], 'LineWidth', 2);
axis([0 timeVec4(end) -3*voltage/2 3*voltage/2]);
title("BiPolarRZ line coding");
legend('BiPolarRZ');

% Zoomed
figure(8);
plot(timeVec4, lineCodeVec4, 'color', [0 0.4470 0.7410], 'LineWidth', 2);
axis([0 100*bitPeriod -3*voltage/2 3*voltage/2]);
title("BiPolarRZ line coding (zoomed to first 100 bits)");
legend('BiPolarRZ');
grid on;
```

```


% ManchesterCoding line coding
figure(9);
plot(timeVec5, lineCodeVec5, 'color', [0.6350 0.0780 0.1840], 'LineWidth', 2);
axis([0 timeVec5(end) -3*voltage/2 3*voltage/2]);
title("ManchesterCoding line coding");
legend('ManchesterCoding');

% Zoomed
figure(10);
plot(timeVec5, lineCodeVec5, 'color', [0.6350 0.0780 0.1840], 'LineWidth', 2);
axis([0 100*bitPeriod -3*voltage/2 3*voltage/2]);
title("ManchesterCoding line coding (zoomed to first 100 bits)");
legend('ManchesterCoding');
grid on;
figure(11);
subplot(1,5,1);
plot_eye_diagram(noBitsPerSegments , noSamplesPerBit , lineCodeVec1 , bitPeriod);
title("eye diagram of UniPolarNRZ");
legend('UniPolarNRZ');
ylim([0-voltage/4 3*voltage/2]);
grid on;
% PolarNRZ
subplot(1,5,2);
plot_eye_diagram(noBitsPerSegments , noSamplesPerBit , lineCodeVec2 , bitPeriod);
title("eye diagram of PolarNRZ");
legend('PolarNRZ');
ylim([-3*voltage/2 3*voltage/2]);
grid on;
% UniPolarRZ
subplot(1,5,3);
plot_eye_diagram(noBitsPerSegments , noSamplesPerBit , lineCodeVec3 , bitPeriod);
title("eye diagram of UniPolarRZ");
legend('UniPolarRZ');
ylim([0-voltage/4 3*voltage/2]);
grid on;
% BiPolarRZ
subplot(1,5,4);
plot_eye_diagram(noBitsPerSegments , noSamplesPerBit , lineCodeVec4 , bitPeriod);
title("eye diagram of BiPolarRZ");
legend('BiPolarRZ');
ylim([-3*voltage/2 3*voltage/2]);
grid on;
% ManchesterCoding
subplot(1,5,5);
plot_eye_diagram(noBitsPerSegments , noSamplesPerBit , lineCodeVec5 , bitPeriod);
title("eye diagram of ManchesterCoding");
legend('ManchesterCoding');
ylim([-3*voltage/2 3*voltage/2]);
grid on;
```

```
%% get power spectrum of line coding (transmitted signal)
[spectral_of_lineCodeVec1,f1] = spectral_domain(lineCodeVec1,noSamplesPerBit,bitPeriod,bit_stream);
[spectral_of_lineCodeVec2,f2] = spectral_domain(lineCodeVec2,noSamplesPerBit,bitPeriod,bit_stream);
[spectral_of_lineCodeVec3,f3] = spectral_domain(lineCodeVec3,noSamplesPerBit,bitPeriod,bit_stream);
[spectral_of_lineCodeVec4,f4] = spectral_domain(lineCodeVec4,noSamplesPerBit,bitPeriod,bit_stream);
[spectral_of_lineCodeVec5,f5] = spectral_domain(lineCodeVec5,noSamplesPerBit,bitPeriod,bit_stream);
%% plot power spectrum of any type of line coding
% UniPolarNRZ
figure(12);
plot(f1,spectral_of_lineCodeVec1,'color',[0.4940 0.1840 0.5560],'LineWidth',2);
title("power spectral of UniPolarNRZ");
xlabel("frequency (Hz)");
ylabel('power spectral density');
legend('UniPolarNRZ');
grid on;
% Zoomed UniPolarNRZ
figure(13);
plot(f1,spectral_of_lineCodeVec1,'color',[0.4940 0.1840 0.5560],'LineWidth',2);
title("power spectral of UniPolarNRZ (zoomed)");
xlabel("frequency (Hz)");
ylabel('power spectral density');
legend('UniPolarNRZ');
axis([-3 3 0 0.0003]);
grid on;
% PolarNRZ
figure(14);
plot(f2,spectral_of_lineCodeVec2,'color',[1 0 0],'LineWidth',2);
title("power spectral of PolarNRZ");
xlabel("frequency (Hz)");
ylabel('power spectral density');
legend('PolarNRZ');
grid on;
axis([-5 5 0 0.0012]);
% UniPolarRZ
figure(15);
plot(f3,spectral_of_lineCodeVec3,'color',[0.8500 0.3250 0.0980],'LineWidth',2);
title("power spectral of UniPolarRZ");
xlabel("frequency (Hz)");
ylabel('power spectral density');
legend('UniPolarRZ');
grid on;
```

```

% Zoomed UniPolarRZ
figure(16);
plot(f3,spectral_of_lineCodeVec3,'color',[0.8500 0.3250 0.0980],'LineWidth',2);
title("power spectral of UniPolarRZ (zoomed)");
xlabel("frequency (Hz)");
ylabel('power spectral density');
legend('UniPolarRZ');
grid on;
axis([-6 6 0 0.0001]);

% BiPolarRZ
figure(17);
plot(f4,spectral_of_lineCodeVec4,'color',[0 0.4470 0.7410],'LineWidth',2);
title("power spectral of BiPolarRZ");
xlabel("frequency (Hz)");
ylabel('power spectral density');
legend('BiPolarRZ');
axis([-15 15 0 0.00025]);
grid on;

% ManchesterCoding
figure(18);
plot(f5,spectral_of_lineCodeVec5,'color',[0.6350 0.0780 0.1840],'LineWidth',2);
title("power spectral of ManchesterCoding");
xlabel("frequency (Hz)");
ylabel('power spectral density');
legend('ManchesterCoding');
axis([-10 10 0 0.0008]);

figure(19);
sigma_ranges = linspace(0, voltage, 10);
[BER_values1, num_errors1] = Sweep_on_value_of_sigma(lineCodeVec1, voltage, timeVec1, coding_scheme1, noSamplesPerBit, noOfBits, bit_stream);
[BER_values2, num_errors2] = Sweep_on_value_of_sigma(lineCodeVec2, voltage, timeVec2, coding_scheme2, noSamplesPerBit, noOfBits, bit_stream);
[BER_values3, num_errors3] = Sweep_on_value_of_sigma(lineCodeVec3, voltage, timeVec3, coding_scheme3, noSamplesPerBit, noOfBits, bit_stream);
[BER_values4, num_errors4] = Sweep_on_value_of_sigma(lineCodeVec4, voltage, timeVec4, coding_scheme4, noSamplesPerBit, noOfBits, bit_stream);
[BER_values5, num_errors5] = Sweep_on_value_of_sigma(lineCodeVec5, voltage, timeVec5, coding_scheme5, noSamplesPerBit, noOfBits, bit_stream);

semilogy(sigma_ranges, BER_values1, sigma_ranges, BER_values2, sigma_ranges, BER_values3, sigma_ranges, BER_values4, sigma_ranges, BER_values5, 'LineWidth', 2);

grid on;
xlabel('Sigma');
ylabel('BER');
legend({'UniPolarNRZ', 'PolarNRZ', 'UniPolarRZ', 'BiPolarRZ', 'ManchesterCoding'});
%% BONUS
```

```
[number_of_detected_errors]=Bonus_Sweep_on_value_of_sigma(lineCodeVec4,voltage,timeVec4,coding_scheme4,noSamplesPerBit,noOfBits);
```

Script Part 2 :

part2.m:

```
% octave Script for a Binary PSK (passband communication)
% Clear all variables and close all figures
clear all;
close all;
noOfBits = 100; % Number of time samples
% Generate bit stream
bit_stream = generate_random_bits(noOfBits);
%% Calculations
fc = 10^9;   % Frequency of Modulating Signal
fs = 10 * fc;  % Sampling frequency - Sampling rate - This will define the resolution
ts = 1 / fs;   % Time step
t = 0:ts:(noOfBits - 1) * ts;   % Time for one bit - Time of one bit = 1 second per one bit
T = (noOfBits) * (noOfBits * ts);   % Simulation time
Rb = noOfBits / T;   % Bit rate = N / simulation time
voltage = 1;
bitPeriod = t(end);
noSamplesPerBit = length(t);
coding_scheme = 'PolarNRZ';
%% The PolarNRZ Signal
[PolarNRZ, timeVec] = line_coding(bit_stream, coding_scheme, voltage, bitPeriod, noSamplesPerBit, fc);
% Plot the PolarNRZ Signal with time
figure;
plot(timeVec, PolarNRZ, 'r', 'LineWidth', 2);
xlabel('Time (s)');
ylabel('Amplitude (V)');
title('PolarNRZ Signal');
axis([0 timeVec(end) -1.5 1.5]);
grid on;
% Spectrum & Spectral of PolarNRZ Signal
[spectral_PolarNRZ_Signal, f, BW, Spec_PolarNRZ_Signal] = spectral_domain(PolarNRZ, bit_stream, noSamplesPerBit, bitPeriod, fc);
figure;
grid on;
plot(f, abs(Spec_PolarNRZ_Signal));
title('Spectrum of Original PolarNRZ Signal');
xlabel("Frequency (Hz)");
```

```
ylabel('Amplitude (V)');

figure;
plot(f, spectral_PolarNRZ_Signal);
title('Power Spectral Density of PolarNRZ Signal');
xlabel("Frequency (Hz)");
ylabel('Power Spectral Density');

% The BPSK Signal (modulated signal of PolarNRZ Signal after product modulator in transmitter circuit)
BPSK_signal = PolarNRZ .* cos(2 * pi * fc * timeVec);

% Plot the BPSK Signal (line coding) with time
figure;
plot(timeVec, BPSK_signal, 'LineWidth', 2);
xlabel('Time (s)');
ylabel('Amplitude (V)');
title('BPSK Signal with Two Phase Shifts');
axis([0 timeVec(end) -1.5 1.5]);
grid on;

% Spectrum & Spectral of BPSK signal

[spectral_BPSK_signal, f, BW, spec_BPSK_signal] = spectral_domain(BPSK_signal, bit_stream, noSamplesPerBit, bitPeriod, fc);

figure(3);
grid on;

subplot(1, 2, 1);
plot(f, abs(spec_BPSK_signal));
title("Spectrum of Modulated BPSK Signal");
xlabel("Frequency (Hz)");
ylabel('Amplitude (V)');

subplot(1, 2, 2);
plot(f, spectral_BPSK_signal);
title("Power Spectral Density of Modulated BPSK Signal");
xlabel("Frequency (Hz)");
ylabel('Power Spectral Density');

% demodulated signal (demodulation for BPSK signal after product modulator in recivied circuit)
demodulated_BPSK_signal = BPSK_signal .* cos(2 * pi * fc * timeVec);

% specterum & spectral of recieved signal (demodulated BPSK signal) after product modulator in reciver circuit
[spectral_demodulated_BPSK_signal,f,BW,spec_demodulated_BPSK_signal] = spectral_domain(demodulated_BPSK_signal, bit_stream, noSamplesPerBit, bitPeriod, fc);
figure(4);
grid on;
```

```
subplot(1,2,1);
plot(f, abs(spec_demodulated_BPSK_signal));
title("specterum of recieved signal after product modulator in reciver circuit ");
xlabel("frequency (Hz)");
ylabel("Amplitude (V)");
subplot(1,2,2);
plot(f, spectral_demodulated_BPSK_signal);
title("power spectral density of demodulated BPSK signal");
xlabel("frequency (Hz)");
ylabel("power spectral density ");

% LPF (Ideal)
H = abs(f) < (BW);

% specterum &spectral of demodulated BPSK signal After LPF
demodulated_BPSK_signal_After_Lpf = abs(real(H .* spec_demodulated_BPSK_signal));
figure(5);
grid on;
subplot(1,2,1);
plot(f, demodulated_BPSK_signal_After_Lpf);
title("spectrum of demodulated BPSK signal After Lpf");
xlabel("frequency (Hz)");
ylabel("Amplitude (V)");
subplot(1,2,2);
plot(f, (demodulated_BPSK_signal_After_Lpf .^ 2));
title("power spectral density of demodulated BPSK signal After Lpf");
xlabel("frequency (Hz)");
ylabel("power spectral density ");

% demodulated BPSK signal AFTER LPF in time domain
demodulated_BPSK_signal_After_Lpf_t = real(ifft(fftshift(H .* spec_demodulated_BPSK_signal) * noOfBits)); %1/N

figure(6);
grid on;
subplot(2,1,1);
plot(timeVec, demodulated_BPSK_signal_After_Lpf_t, 'LineWidth', 2);
axis([0 timeVec(end) -1*voltage voltage]);
title("demodulated BPSK signal After LPF in time domain");
xlabel("time(s)");
ylabel("Amplitude (V)");

% decision device circuit give wave form in ones and zeros and remove effect of LPF
[Reciever_output] = decision_device(demodulated_BPSK_signal_After_Lpf_t, coding_scheme, voltage, timeVec, noSamplesPerBit, noOfBits);

subplot(2,1,2);
plot(timeVec, Reciever_output, 'color', [0.6350 0.0780 0.1840], 'LineWidth', 2);
axis([0 timeVec(end) 0 voltage]);
title("output demodulated BPSK signal from recieved circuit");
xlabel("time(s)");
```

```
ylabel("Amplitude (V)");

% return reciever output in form of binary data
binary_data = convert_into_Binary_data(Reciever_output, bit_stream, noSamplesPerBit);

% calculate BER & num_errors
[BER, num_errors] = BER_device(binary_data, bit_stream);
```

Functions codes:

File: generate_random_bits.m

```
function bit_stream = generate_random_bits( noOfBits )
   % generate random sequence of binary data zeros and ones
   bit_stream = randi( [ 0 , 1 ] , 1 , noOfBits );
end
```

File: line_coding.m

```
function [lineCodeVec,timeVec]=line_coding(bit_stream,coding_scheme,voltage,bitPeriod,noSamplesPerBit,fc )
   switch nargin
       %choose according to number of input arguments
       case 5
           %belong to part 1
           [lineCodeVec,timeVec]=Baseband_communication(bit_stream,coding_scheme,voltage,bitPeriod,noSamplesPerBit);
       case 6
           %belong to part 2
           [lineCodeVec,timeVec]=Passband_communication(bit_stream,coding_scheme,voltage,bitPeriod,noSamplesPerBit,fc);
   end
end
function [lineCodeVec, timeVec] = Passband_communication(bit_stream, coding_scheme, voltage, bitPeriod, noSamplesPerBit, fc)
    % This function implements the function of polarNRZ block in transmitter

    % Calculations
    noOfBits = length(bit_stream);
    fs = 10 * fc;  % Sampling frequency - Sampling rate - This will define the resolution
    ts = 1 / fs;   % Time step
    t = 0:ts:(noOfBits - 1) * ts;
% Generate time domain
timeVec = [];
for ii = 1:1:length(bit_stream)
timeVec = [timeVec t];
t = t + (noOfBits - 1) * ts;
end

noSamplesPerBit = length(t);
```

```
% Generate polarNRZ line coding
switch (coding_scheme)
case 'PolarNRZ'
lineCodeVec = polarNRZ(bit_stream, voltage, timeVec, noSamplesPerBit);
end
end
function lineCodeVec = polarNRZ(bit_stream , voltage , timeVec , noSamplesPerBit)
    lineCodeVec = unipolarNRZ(bit_stream , voltage , timeVec , noSamplesPerBit);
    %same as unipolarNRZ but change all the zeros into -ve voltage
    lineCodeVec(lineCodeVec == 0) = -1 * voltage;
end
function lineCodeVec = unipolarRZ(bit_stream , voltage , timeVec , noSamplesPerBit)
    lineCodeVec = zeros(1 , length(timeVec));
    for i = 1 : length(bit_stream)
        if bit_stream(i) == 1
            %+ve voltage for the first half cycle of the bits
            lineCodeVec( ((i - 1) * noSamplesPerBit) + 1 : (i * noSamplesPerBit) - (noSamplesPerBit / 2)) = voltage;
            % 0 voltage for the other half cycle of the bit
            lineCodeVec( (i * noSamplesPerBit) - (noSamplesPerBit / 2) + 1 : i * noSamplesPerBit) = 0;
        end
    end
end
function lineCodeVec = bipolarRZ(bit_stream , voltage , timeVec , noSamplesPerBit)
    lineCodeVec = zeros(1 , length(timeVec));
    flag = 0; % to indicate whether the voltage to be +ve or -ve
    for i = 1 : length(bit_stream)
        if bit_stream(i) == 1
            if (flag == 0)
                %+ve voltage for the first half cycle of the bits
                lineCodeVec( ((i - 1) * noSamplesPerBit) + 1 : (i * noSamplesPerBit) - (noSamplesPerBit / 2)) = voltage;
                % 0 voltage for the other half cycle of the bit
                lineCodeVec( (i * noSamplesPerBit) - (noSamplesPerBit / 2) + 1 : i * noSamplesPerBit) = 0;
                flag = 1; %update the flag
            elseif(flag == 1)
                %-ve voltage for the first half cycle of the bits
                lineCodeVec( ((i - 1) * noSamplesPerBit) + 1 : (i * noSamplesPerBit) - (noSamplesPerBit / 2)) = -voltage;
                % 0 voltage for the other half cycle of the bit
                lineCodeVec( (i * noSamplesPerBit) - (noSamplesPerBit / 2) + 1 : i * noSamplesPerBit) = 0;
                flag = 0; %update the flag
            end
        end
    end
end
```

```
function lineCodeVec = manchesterCoding(bit_stream , voltage , timeVec , noSamplesPerBit)
    lineCodeVec = zeros(1 , length(timeVec));
    for i = 1 : length(bit_stream)
        if bit_stream(i) == 1
            %+ve voltage for the first half cycle of the bits
            lineCodeVec( ((i - 1) * noSamplesPerBit) + 1 : (i * noSamplesPerBit) - (noSamplesPerBit / 2)) = voltage;
            %-ve voltage for the other half cycle of the bit
            lineCodeVec( (i * noSamplesPerBit) - (noSamplesPerBit / 2) + 1 : i * noSamplesPerBit) = -1* voltage;
        else
            %-ve voltage for the first half cycle of the bits
            lineCodeVec( ((i - 1) * noSamplesPerBit) + 1 : (i * noSamplesPerBit) - (noSamplesPerBit / 2)) = -1 * voltage;
            %+ve voltage for the other half cycle of the bit
            lineCodeVec( (i * noSamplesPerBit) - (noSamplesPerBit / 2) + 1 : i * noSamplesPerBit) = voltage;
        end
    end
end
```

File: plot_eye_diagram.m

```
function plot_eye_diagram(noBitsPerSegments , noSamplePerBit , lineCodeVec , bitPeriod)
    % segment is a collection of bits we need to shift them to create the eye diagram
    % segmentLength: total samples in a segment
    segmentLength = noSamplePerBit * noBitsPerSegments;
    % periodic time of the samples
    samplePeriod = bitPeriod / noSamplePerBit;
    % time vector is created with a step of samplePeriod, and the end is the total time a segment
    % could be calculated using noBitsPerSegments and bitPeriod, with a step = samplePeriod
    timeVec = (0 : segmentLength - 1) * samplePeriod;
    % to center the zero in the mid
    timeVec = timeVec - (noBitsPerSegments * bitPeriod / 2);
    % total number of segment layers to place on top of each other
    % segmentLayers = total number of samples in line code / total number of samples in the segment
    % it's rounded down to the nearest integer, as we need to keep the size of the segment
    % equal to the size of the time vector in terms of the samples
    segmentLayers = floor(length(lineCodeVec) / segmentLength);
    for i = 1 : segmentLayers
        % find the start and end points of the segment in the line code
        % it's multiplied by the number of bits per segment
        segmentStart = (i - 1) * segmentLength + 1;
```

```
        segmentEnd = i * segmentLength;
        % x-axis is the time vector, and the y-axis is the amplitude of each bit
        plot(timeVec , lineCodeVec(segmentStart : segmentEnd),'LineWidth',2);
        % hold to plot all the segments on top of each other
        hold on;
    end
    axis([-1.5 1.5 -bitPeriod bitPeriod]);
end
```

File: spectral_domain.m

```
function [spectral, f, BW, Spec_Original_line_coding] = spectral_domain(lineCodeVec, bit_stream, noSamplesPerBit, bitPeriod, fc)
    switch nargin
        % choose according to number of input arguments
        case 4
            [spectral, f] = Baseband_communication(lineCodeVec, noSamplesPerBit, bitPeriod, bit_stream);
        case 5
            [spectral, f, BW, Spec_Original_line_coding] = Passband_communication(lineCodeVec, bit_stream, noSamplesPerBit, bitPeriod, fc);
    end
end

function [spectral, f, BW, Spec_Original_line_coding] = Passband_communication(lineCodeVec, bit_stream, noSamplesPerBit, bitPeriod, fc)
    % Calculation to generate frequency domain
    noOfBits = length(bit_stream);
    fs = 10 * fc;  % Sampling frequency - Sampling rate - This will define the resolution
    ts = 1 / fs;   % Time step
    T = noOfBits * ts; % Simulation time
    Rb = noOfBits / T; % Bit rate = N / simulationtime
    BW = Rb; % Polar & NRZ
    df = 1 / T; % Frequency step

    % Spectrum of Original Digital Signal
    Spec_Original_line_coding = (fftshift(fft(lineCodeVec))) / noOfBits; % We put message in frequency domain because it only shifts function in FFT

    % Frequency domain
    if (rem(noOfBits,2) == 0) % Even
        f = ((-(0.5 * fs)):df:((0.5 * fs) - df)); % Frequency vector if x/f even
    else % Odd
        f = (-(0.5 * fs - 0.5 * df)) : df :(((0.5 * fs) + 1) - 0.5 * df); % Frequency vector if X/f is odd
    end

    % Power spectral of Original Digital Signal
    spectral = abs(Spec_Original_line_codingand).^2;
```

```
end

function [spectral, f] = Baseband_communication(lineCodeVec, noSamplesPerBit, bitPeriod, bit_stream)
    % Calculation to generate frequency domain
    T = length(bit_stream) * bitPeriod; % Simulation time
    df =It  %frequency  step
   fs=noSamplesPerBit/bitPeriod; % sampling frequency
  N=noSamplesPerBit*length(bit_stream);
  %spectrum of Original Digital Signal
 Spec_Original_line_coding = (fftshift(fft(lineCodeVec)))/N;
  % frequency doomain
 if(rem(N,2)==0) %% even
 f = ((-(0.5*fs)): df : ((0.5*fs)-df));%% frequency vector if x/f even
 else %% odd
 f = (-(0.5*fs-0.5*df)) : df :(((0.5*fs)+1)-0.5*df); %% frequecy vector if X/f is odd
 end
  %power spectral of Original Digital Signal
 spectral =abs(Spec_Original_line_coding).^2;
  end
```

File: Sweep_on_value_of_sigma.m

```
function [BER_values, num_errors] = Sweep_on_value_of_sigma(lineCodeVec, voltage, timeVec, coding_scheme, noSamplesPerBit, noOfBits, bit_stream)
    % generate 10 ranges for sigma that range from 0 to the maximum supply voltage
    sigma_ranges = linspace(0, voltage, 10);

    % pre-allocate a vector to store the BER values & num_errors
    BER_values = zeros(1, 10);
    num_errors = zeros(1, 10);

    % loop choose each value of sigma by index i
     for i = 1:10
         % add noise to signal
         received_signal_with_noise = add_noise_to_linecoding(lineCodeVec, sigma_ranges(i), timeVec);

         % path signal through the decision device
         Reciever_output = decision_device(received_signal_with_noise, coding_scheme, voltage, timeVec, noSamplesPerBit, noOfBits);

         % this loop is used to convert Reciever output into binary data
         binary_data = convert_into_Binary_data(Reciever_output, bit_stream, noSamplesPerBit);

         % calculate the BER & num_errors for this value of sigma
         [BER_values(i), num_errors(i)] = BER_device(binary_data, bit_stream); % insert your code for calculating BER & num_errors here
```

```
    end
end
```

File: add_noise_to_linecoding.m

```
function received_signal_with_noise = add_noise_to_linecoding(lineCode, sigma, Vectime)
    % this function simulates external noise from the communication channel (telephone line) or noise from the receiver added to the transmitted line coding

    % define your time vector
    t = Vectime;

    % noise
    n = sigma * randn(1, length(t));

    % add the noise to your received signal
    received_signal_with_noise = lineCode + n;
end
```

File: decision_device.m

```
function [Reciever_output] = decision_device(received_signal_with_noise, coding_scheme, voltage, timeVec, noSamplesPerBit, noOfBits)
    % to select the type of line coding by coding scheme
    switch (coding_scheme)
        case 'UniPolarNRZ'
            Reciever_output = r_unipolarNRZ(received_signal_with_noise, voltage, timeVec, noSamplesPerBit, noOfBits);
        case 'PolarNRZ'
            Reciever_output = r_polarNRZ(received_signal_with_noise, voltage, timeVec, noSamplesPerBit, noOfBits);
        case 'UniPolarRZ'
            Reciever_output = r_unipolarRZ(received_signal_with_noise, voltage, timeVec, noSamplesPerBit, noOfBits);
        case 'BiPolarRZ'
            Reciever_output = r_bipolarRZ(received_signal_with_noise, voltage, timeVec, noSamplesPerBit, noOfBits);
        case 'ManchesterCoding'
            Reciever_output = r_manchesterCoding(received_signal_with_noise, voltage, timeVec, noSamplesPerBit, noOfBits);
    end
end

function [Reciever_output] = Master_source_and_comparator(received_signal_with_noise, noOfBits, noSamplesPerBit, timeVec, L, M, P, threshold_1, threshold_2)
    % pre-allocate a vector to store the Reciever_output
    Reciever_output = zeros(1, length(timeVec));
```

```
    switch nargin
        case 8
            for i = 1:1:noOfBits
                if (received_signal_with_noise(L) < threshold_1)
                    for k = M:1:P
                        Reciever_output(k) = 0;
                    end
                elseif (received_signal_with_noise(L) > threshold_1)
                    for k = M:1:P
                        Reciever_output(k) = 1;
                    end
                end
                M = M + noSamplesPerBit;
                P = P + noSamplesPerBit;
                L = L + noSamplesPerBit;
            end
        case 9
            for i = 1:1:noOfBits
                if (received_signal_with_noise(L) < threshold_2)
                    for k = M:1:P
                        Reciever_output(k) = 1;
                    end
                elseif (received_signal_with_noise(L) > threshold_1)
                    for k = M:1:P
                        Reciever_output(k) = 1;
                    end
                elseif (received_signal_with_noise(L) < threshold_1)
                    for k = M:1:P
                        Reciever_output(k) = 0;
                    end
                end
                M = M + noSamplesPerBit;
                P = P + noSamplesPerBit;
                L = L + noSamplesPerBit;
            end
    end
end

function [Reciever_output] = r_unipolarNRZ(received_signal_with_noise, voltage, timeVec, noSamplesPerBit, noOfBits)
    % L decision level of timing circuit
    % M, P time of bit in our time vector
    threshold = voltage / 2;
    L = noSamplesPerBit / 2;
    M = 1;
    P = noSamplesPerBit;
    [Reciever_output] = Master_source_and_comparator(received_signal_with_noise, noOfBits, noSamplesPerBit, timeVec, L, M, P, threshold);
end
```

```
function [Reciever_output] = r_polarNRZ(received_signal_with_noise, voltage, timeVec, noSamplesPerBit, noOfBits)
    % L decision level of timing circuit
    % M, P time of bit in our time vector
    threshold = (voltage + (-1 * (voltage))) / 2;
    L = noSamplesPerBit / 2;
    M = 1;
    P = noSamplesPerBit;
    Reciever_output = Master_source_and_comparator(received_signal_with_noise, noOfBits, noSamplesPerBit, timeVec, L, M, P, threshold);
end

function [Reciever_output] = r_unipolarRZ(received_signal_with_noise, voltage, timeVec, noSamplesPerBit, noOfBits)
    % L decision level of timing circuit
    % M, P time of bit in our time vector
    threshold = voltage / 2;
    L = noSamplesPerBit / 4;
    M = 1;
    P = noSamplesPerBit;
    Reciever_output = Master_source_and_comparator(received_signal_with_noise, noOfBits, noSamplesPerBit, timeVec, L, M, P, threshold);
end

function [Reciever_output] = r_bipolarRZ(received_signal_with_noise, voltage, timeVec, noSamplesPerBit, noOfBits)
    % L decision level of timing circuit
    % M, P time of bit in our time vector
    threshold_1 = voltage / 2;
    threshold_2 = (-1 * (voltage)) / 2;
    L = noSamplesPerBit / 4;
    M = 1;
    P = noSamplesPerBit;
    Reciever_output = Master_source_and_comparator(received_signal_with_noise, noOfBits, noSamplesPerBit, timeVec, L, M, P, threshold_1, threshold_2);
end

function [Reciever_output] = r_manchesterCoding(received_signal_with_noise, voltage, timeVec, noSamplesPerBit, noOfBits)
    % L decision level of timing circuit
    % M, P time of bit in our time vector
    threshold = (voltage + (-1 * (voltage))) / 2;
    L = noSamplesPerBit / 4;
    M = 1;
    P = noSamplesPerBit;
    Reciever_output = Master_source_and_comparator(received_signal_with_noise, noOfBits, noSamplesPerBit, timeVec, L, M, P, threshold);
end
```

File: convert_into_Binary_data.m

```
function [binary_data] = convert_into_Binary_data(Reciever_output, bit_stream, noSamplesPerBit)
    % pre-allocate a vector to store the binary_data
    binary_data = zeros(1, length(bit_stream));

    L = noSamplesPerBit / 2;

    for P = 1:1:length(bit_stream)
        if (Reciever_output(L) == 0)
            binary_data(P) = 0;
        elseif (Reciever_output(L) == 1)
             binary_data(P) = 1;
        end

         L = L + noSamplesPerBit;
     end
end
```

File: BER_device.m

```
function [BER, num_errors] = BER_device(Reciever_output, bit_stream)
    % count the number of errors & calculate BER
    num_errors = sum(Reciever_output ~= bit_stream);
    BER = num_errors / length(bit_stream);
end
```

File: Regenerative_Repeater.m

```
function [Repeater_output] = Regenerative_Repeater(received_signal_with_noise, coding_scheme, voltage, timeVec, noSamplesPerBit, noOfBits)
    % This function represents the regenerative repeater block used in telephone lines.
    % When entering the receiver to select the type of line coding by coding scheme.
    switch (coding_scheme)
        case 'BiPolarRZ'
            Repeater_output = r_bipolarRZ(received_signal_with_noise, voltage, timeVec, noSamplesPerBit, noOfBits);
    end
end

function [Repeater_output] = r_bipolarRZ(received_signal_with_noise, voltage, timeVec, noSamplesPerBit, noOfBits)
    % Pre-allocate a vector to store the Receiver_output
    Repeater_output = zeros(1, length(timeVec));
    % L decision level of timing circuit
    % M, P time of bit in our time vector
    threshold_1 = voltage / 2;
    threshold_2 = (-1 * (voltage)) / 2;
    L = noSamplesPerBit / 4;
    M = 1;
    P = noSamplesPerBit / 2;
```

```
    for i = 1:1:noOfBits
        if (received_signal_with_noise(L) < threshold_2)
            for k = M:1:P
                Repeater_output(k) = -1 * voltage;
            end
        elseif (received_signal_with_noise(L) > threshold_1)
            for k = M:1:P
                Repeater_output(k) = voltage;
            end
        elseif (received_signal_with_noise(L) < threshold_1)
            for k = M:1:P
                Repeater_output(k) = 0;
            end
        end
        M = M + noSamplesPerBit;
        P = P + noSamplesPerBit;
        L = L + noSamplesPerBit;
    end
end
```

File: Error_Detection_Circuit.m

```
function [number_of_detected_errors] = Error_Detection_Circuit(Repeater_output, voltage, noSamplesPerBit, noOfBits)
    % Error detection circuit for Bipolar return to zero
    number_of_detected_errors = 0;
    L = noSamplesPerBit / 4;

    for i = 1:1:noOfBits
        if (L >= noOfBits * noSamplesPerBit || (L + noSamplesPerBit) >= noOfBits * noSamplesPerBit)
            break;
        end

        % 1000000000...01
        if (Repeater_output(L) == voltage && Repeater_output(L + noSamplesPerBit) == 0)
            if (L >= noOfBits * noSamplesPerBit || (L + noSamplesPerBit) >= noOfBits * noSamplesPerBit)
                break;
            else
                L = L + noSamplesPerBit;
            end

            while (Repeater_output(L) == 0)
            if (L >= noOfBits * noSamplesPerBit || (L + noSamplesPerBit) >= noOfBits * noSamplesPerBit)
                break;
            else
                L = L + noSamplesPerBit;
            end
        end
```

```

        if (Repeater_output(L) == voltage)
            number_of_detected_errors = number_of_detected_errors + 1;
        end
        % -100000000...0-1
        elseif (Repeater_output(L) == -1 * voltage && Repeater_output(L + noSamplesPerBit) == 0)
            if (L >= noOfBits * noSamplesPerBit || (L + noSamplesPerBit) >= noOfBits * noSamplesPerBit)
                break;
            else
                L = L + noSamplesPerBit;
            end

            while (Repeater_output(L) == 0)
                if (L >= noOfBits * noSamplesPerBit || (L + noSamplesPerBit) >= noOfBits * noSamplesPerBit)
                    break;
                else
                    L = L + noSamplesPerBit;
                end
            end

            if (Repeater_output(L) == -1 * voltage)
                number_of_detected_errors = number_of_detected_errors + 1;
            end

        % -1-1
        elseif (Repeater_output(L) == -1 * voltage && Repeater_output(L + noSamplesPerBit) == -1 * voltage)
            number_of_detected_errors = number_of_detected_errors + 1;
            if (L >= noOfBits * noSamplesPerBit || (L + noSamplesPerBit) >= noOfBits * noSamplesPerBit)
                break;
            else
                L = L + noSamplesPerBit;
            end

        % 11
        elseif (Repeater_output(L) == voltage && Repeater_output(L + noSamplesPerBit) == voltage)
            number_of_detected_errors = number_of_detected_errors + 1;
            if (L >= noOfBits * noSamplesPerBit || (L + noSamplesPerBit) >= noOfBits * noSamplesPerBit)
                break;
            else
                L = L + noSamplesPerBit;
            end
        end

```

```
        if (L >= noOfBits * noSamplesPerBit || (L + noSamplesPerBit) >= noOfBits * noSamplesPerBit)
            break;
        else
            L = L + noSamplesPerBit;
        end
    end
end
```

File: Bonus_Sweep_on_value_of_sigma.m

```
function [number_of_detected_errors] = Bonus_Sweep_on_value_of_sigma(lineCodeVec, voltage, timeVec, coding_scheme, noSamplesPerBit, noOfBits)
    % Generate 10 ranges for sigma that range from 0 to the maximum supply voltage
    sigma_ranges = linspace(0, voltage, 10);

    % Pre-allocate a vector to store the BER values & num_errors
    number_of_detected_errors = zeros(1, 10);

    % Loop chooses each value of sigma by index i
    for i = 1:10
        % Add noise to signal
        received_signal_with_noise = add_noise_to_linecoding(lineCodeVec, sigma_ranges(i), timeVec);

        % Path signal through Regenerative_Repeater
        [Repeater_output] = Regenerative_Repeater(received_signal_with_noise, coding_scheme, voltage, timeVec, noSamplesPerBit, noOfBits);

        % Calculate number_of_detected_errors for this value of sigma
        number_of_detected_errors(i) = Error_Detection_Circuit(Repeater_output, voltage, noSamplesPerBit, noOfBits);
    end
end
```