Fundementals of Communication system Project

Team members

Name	ID
Somaya Ayman El Said Ahmed	2000423
Maya Ahmed Abdullah Ali	2002124
Nada Mohamed Abdelhamid	2001434
Mennatallah Amr Ali	2002111
Sara ashraf abdelhakam	2000337
Rwan Hesham Mohamed	2000009
Asmaa Abdelmotlb yousseff	2000051
Ziad Mohamed Ramadan	2000456
Mayar hamdy Bayoumi	2001435

Table of contents

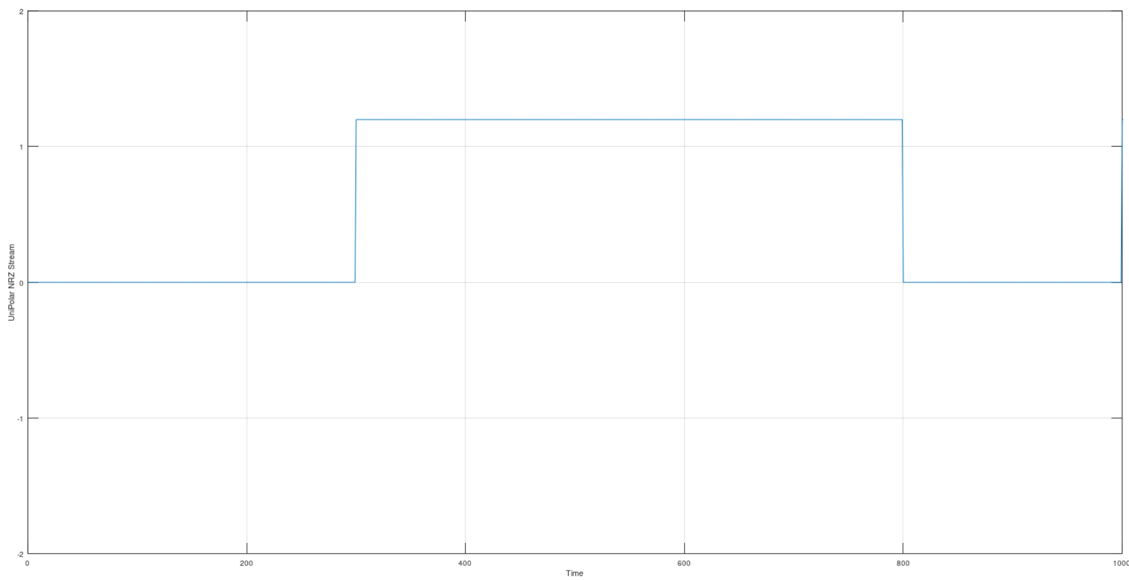
Part one: Transmitter

1.Generate stream of random bits (10,000 bit) (This bit stream should be selected to be random, which means that the type of each bit is randomly selected by the program code to be either ‘1’ or ‘0’).

```
%part one
%transmitter

bitsStream = randi([0 1],1,10e3);
```

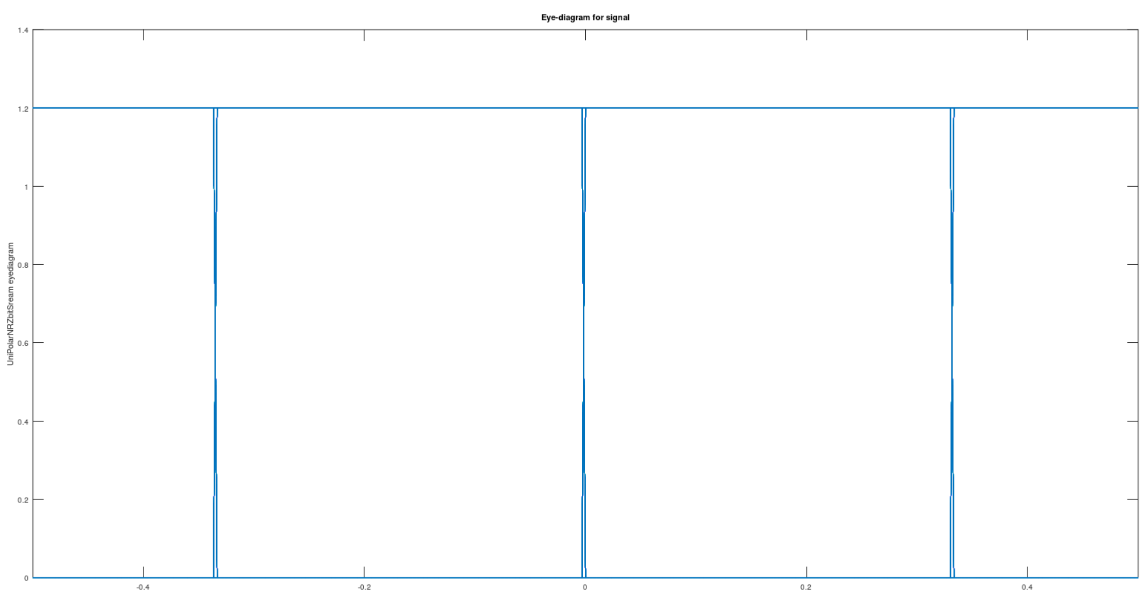
2. Line code the stream of bits (pulse shape) according to Uni-polar non return to zero (Supply voltages are: +1.2 V and -1.2V).

```
%unipolar NRZ line coding
positiveVoltage = 1.2;
UniPolarNRZbitSream = positiveVoltage*bitsStream;

%extend the duration of each bit (each bit will be represented by 100 values)
delay = ones(100,1);
UniPolarNRZbitSream = UniPolarNRZbitSream.* delay;
UniPolarNRZbitSream = reshape(UniPolarNRZbitSream,1,[]);

%time domain
N = length(UniPolarNRZbitSream);
tb = 100;
ts = tb/100;
T = ts * N;
t = 0:ts:((N-1)*ts);
n = length(bitsStream);

%frequency domain
df = 1/T;
Rb = 1/tb;
fs = 1/ts;
if(rem(N,2)==0) %% Even
    f = - (0.5*fs) : df : (0.5*fs-df) ;
else %% Odd
    f = - (0.5*fs-0.5*df) : df : (0.5*fs-0.5*df)
end


%plot the UniPolarNRZbitSream
figure;
plot(t,UniPolarNRZbitSream);
grid on; xlabel("Time"); ylabel("UniPolar NRZ Stream"); xlim([0 1000]);ylim([-2 2])
```



3. Plot the corresponding Eye diagram.

```
%eyediagram
pkg load communications
eyediagram(UniPolarNRZbitSream, 300); ylabel("UniPolarNRZbitSream eyediagram")
```



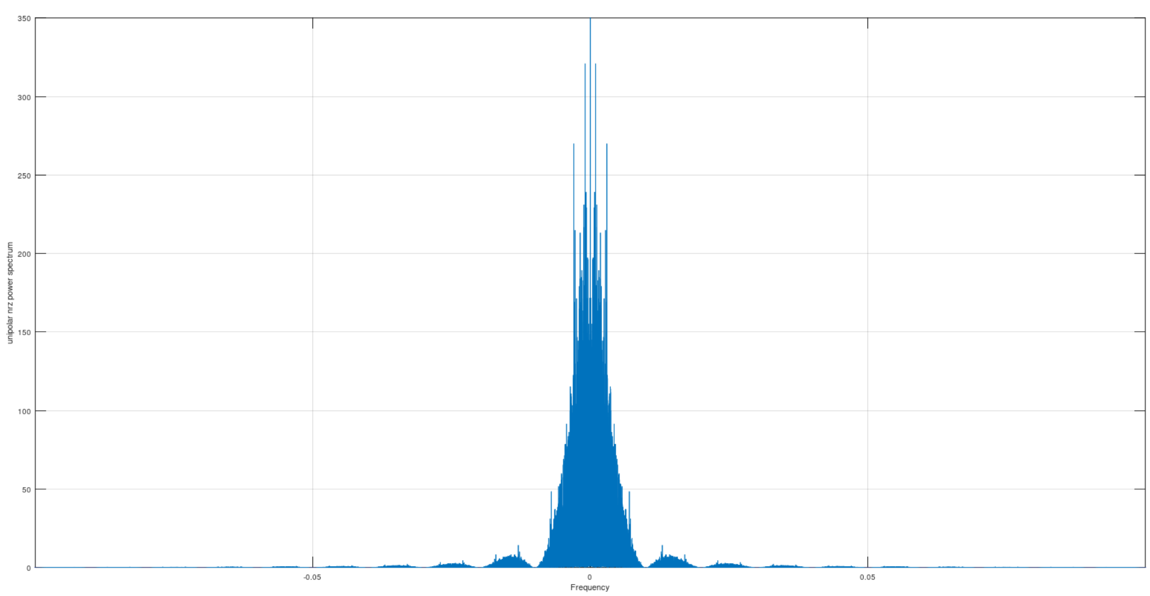
4. Plot the spectral domains of the pulses (square of the Fourier transform).

Function to plot spectral domains

```
function plotSpectraldomains = plotSpectraldomains (lineCodedStream, frequency, ylimit)
% fourier transform and normalize the signal
NN = length(lineCodedStream);
fftbitSream = fftshift(fft(lineCodedStream));
% plot the spectral domain
figure
plot(frequency,abs(fftbitSream.^2)/NN);
grid on;
xlabel("Frequency");
ylabel("Power Spectral density" );
ylim([0 ylimit]);
xlim([-0.1 0.1]);
end
```

Plotting spectral domain of unipolar nrz

```
% Compute the Fourier transform

plotSpectraldomains(UniPolarNRZbitSream,f, 350); ylabel("unipolar nrz power spectrum")
```



Part one: Receiver

5. Design a receiver which consists of a decision device. (The decision device has two inputs: received waveform).

```
% Receiver code for decoding the received signal and calculating BER
%before adding the noise
rx_unipolar_nrz = UniPolarNRZbitSream;
rx_polar_nrz = polar_nrz;
rx_unipolar_rz = unipolar_rz;
rx_bipolar_rz = bipolar_rz;
rx_manchester = Manchester;


%tx bitstream extended
delay = ones(100,1);
extended_bit_stream = bitsStream.* delay;
extended_bit_stream = reshape(extended_bit_stream,1,[]);
```

```
% Decoding the received signal using Uni-polar NRZ
rx_bitsAfterDecision_unipolar_nrz = zeros(1,length(rx_unipolar_nrz));
i = 1;
for i = 1:length(rx_unipolar_nrz)
    if rx_unipolar_nrz(i) > 0.6
       rx_bitsAfterDecision_unipolar_nrz(i) = 1;
    else
       rx_bitsAfterDecision_unipolar_nrz(i) = 0;
    end
    i = i + 1;
end
```

6. Compare the output of the decision level with the generated stream of bits in the transmitter. The comparison is performed by comparing the value of each received bit with the corresponding transmitted bit (step 1) and count number of errors. Then calculate bit error rate (BER) = number of error bits/ Total number of bits.

```
% Calculating BER for each line coding technique
ber_unipolar_nrz = sum(rx_bitsAfterDecision_unipolar_nrz~=extended_bit_stream)/length(extended_bit_stream);
ber_polar_nrz = sum(rx_bitsAfterDecision_polar_nrz~=extended_bit_stream)/length(extended_bit_stream);
ber_unipolar_rz = sum(rx_bitsAfterDecision_unipolar_rz~=extended_bit_stream)/length(extended_bit_stream);
ber_bipolar_rz = sum(rx_bitsAfterDecision_bipolar_rz~=extended_bit_stream)/length(extended_bit_stream);
ber_manchester = sum(rx_bitsAfterDecision_manchester~=extended_bit_stream)/length(extended_bit_stream);
% Displaying the results
disp(['BER for Uni-polar NRZ = ' num2str(ber_unipolar_rz)]);
disp(['BER for Polar NRZ = ' num2str(ber_polar_nrz)]);
disp(['BER for Uni-polar RZ = ' num2str(ber_unipolar_rz)]);
disp(['BER for Bipolar RZ = ' num2str(ber_bipolar_rz)]);
disp(['BER for Manchester = ' num2str(ber_manchester)]);
```

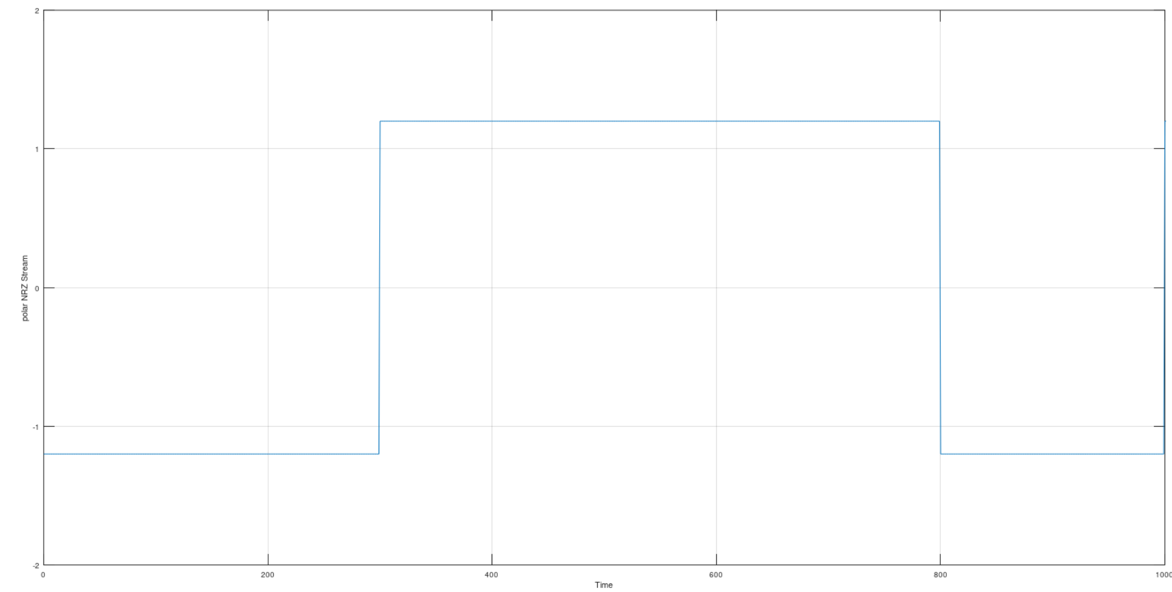
```
BER for Uni-polar NRZ = 0
BER for Polar NRZ = 0
BER for Uni-polar RZ = 0
BER for Bipolar RZ = 0
BER for Manchester = 0
```

7. Repeat the previous steps for different line coding (Polar non return to zero, Uni-polar return to zero, Bipolar return to zero and Manchester coding)

Polar non return to zero

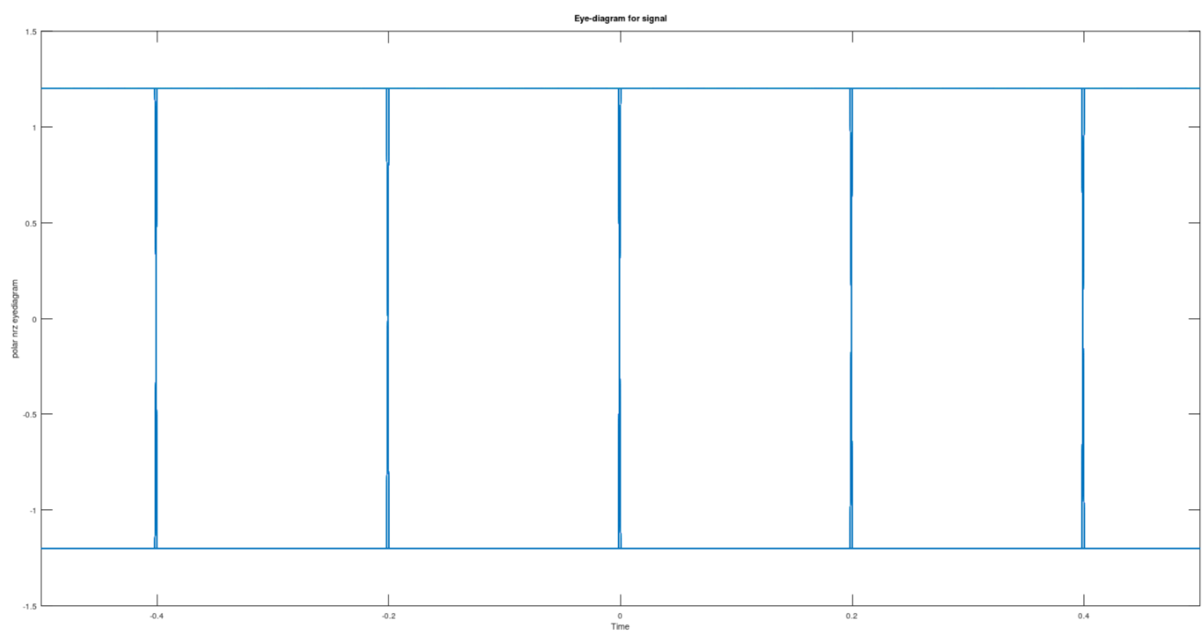
Line code the stream of bits (pulse shape) according to polar non return to zero (Supply voltages are: +1.2 V and -1.2V).

```
% Line encoding using Polar NRZ

polar_nrz = positiveVoltage*(bitsStream*2-1); % Convert 0's to -1.2's and 1's to +1.2's


%extend the duration of each bit (each bit will be represented by 100 values)
delay = ones(100,1);
polar_nrz = polar_nrz.* delay;
polar_nrz = reshape(polar_nrz,1,[]);

%plot the polar_nrz
figure;
plot(t,polar_nrz);
grid on; xlabel("Time"); ylabel("polar NRZ Stream"); xlim([0 1000]);ylim([-2 2])
```



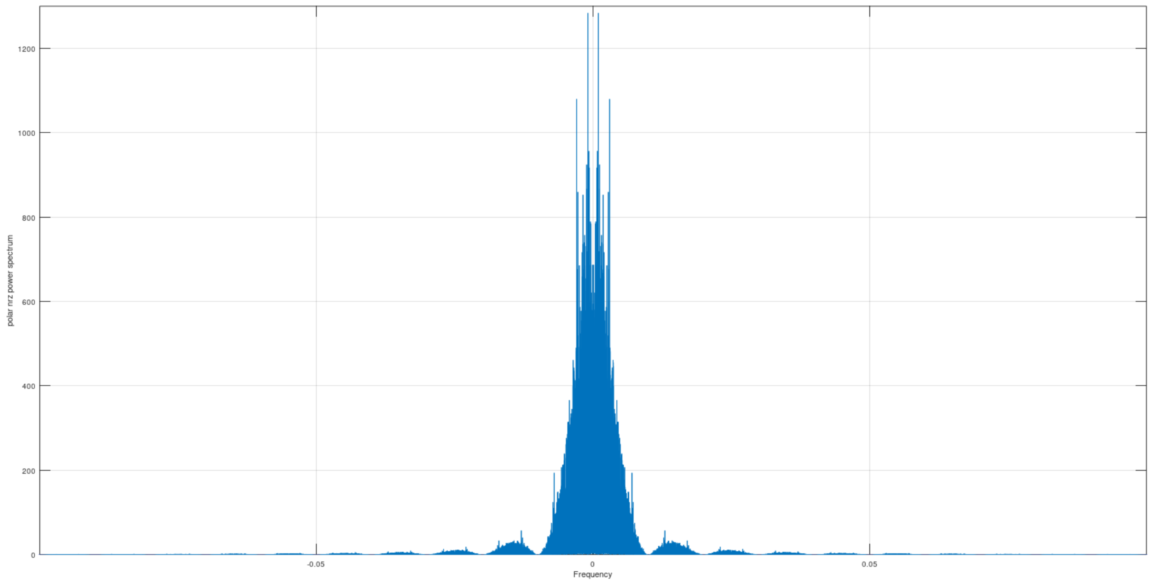
Plot the corresponding Eye diagram.

```
%eye diagram
eyediagram(polar_nrz, 500); ylabel("polar nrz eyediagram")
```



Plotting spectral domain of polar nrz

```
% Compute the Fourier transform

% fourier transform and normalize the signal
plotSpectraldomains(polar_nrz,f,1300); ylabel("polar nrz power spectrum")
```



Design a decision device and calculate the ber for polar nrz

```
% Decoding the received signal using polar NRZ
rx_bitsAfterDecision_polar_nrz = zeros(1,length(rx_polar_nrz));
i = 1;
for i = 1:100:length(rx_polar_nrz)
    if rx_polar_nrz(i) > 0
        rx_bitsAfterDecision_polar_nrz(i:i+99) = 1;
    else
        rx_bitsAfterDecision_polar_nrz(i:i+99) = 0;
    end
    i = i + 1;
end
```

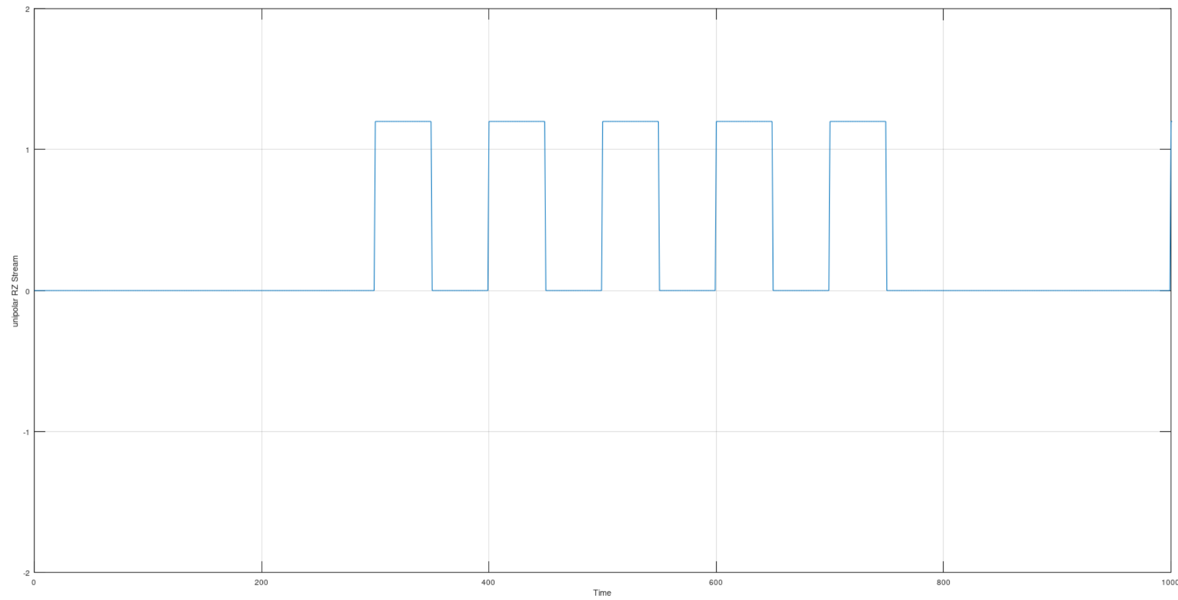
```
% Calculating BER for each line coding technique
ber_unipolar_nrz = sum(rx_bitsAfterDecision_unipolar_nrz~=extended_bit_stream)/length(extended_bit_stream);
ber_polar_nrz = sum(rx_bitsAfterDecision_polar_nrz~=extended_bit_stream)/length(extended_bit_stream);
ber_unipolar_rz = sum(rx_bitsAfterDecision_unipolar_rz~=extended_bit_stream)/length(extended_bit_stream);
ber_bipolar_rz = sum(rx_bitsAfterDecision_bipolar_rz~=extended_bit_stream)/length(extended_bit_stream);
ber_manchester = sum(rx_bitsAfterDecision_manchester~=extended_bit_stream)/length(extended_bit_stream);
% Displaying the results
disp(['BER for Uni-polar NRZ = ' num2str(ber_unipolar_rz)]);
disp(['BER for Polar NRZ = ' num2str(ber_polar_nrz)]);
disp(['BER for Uni-polar RZ = ' num2str(ber_unipolar_rz)]);
disp(['BER for Bipolar RZ = ' num2str(ber_bipolar_rz)]);
disp(['BER for Manchester = ' num2str(ber_manchester)]);
```

```
BER for Uni-polar NRZ = 0
BER for Polar NRZ = 0
BER for Uni-polar RZ = 0
BER for Bipolar RZ = 0
BER for Manchester = 0
```

Uni-polar return to zero

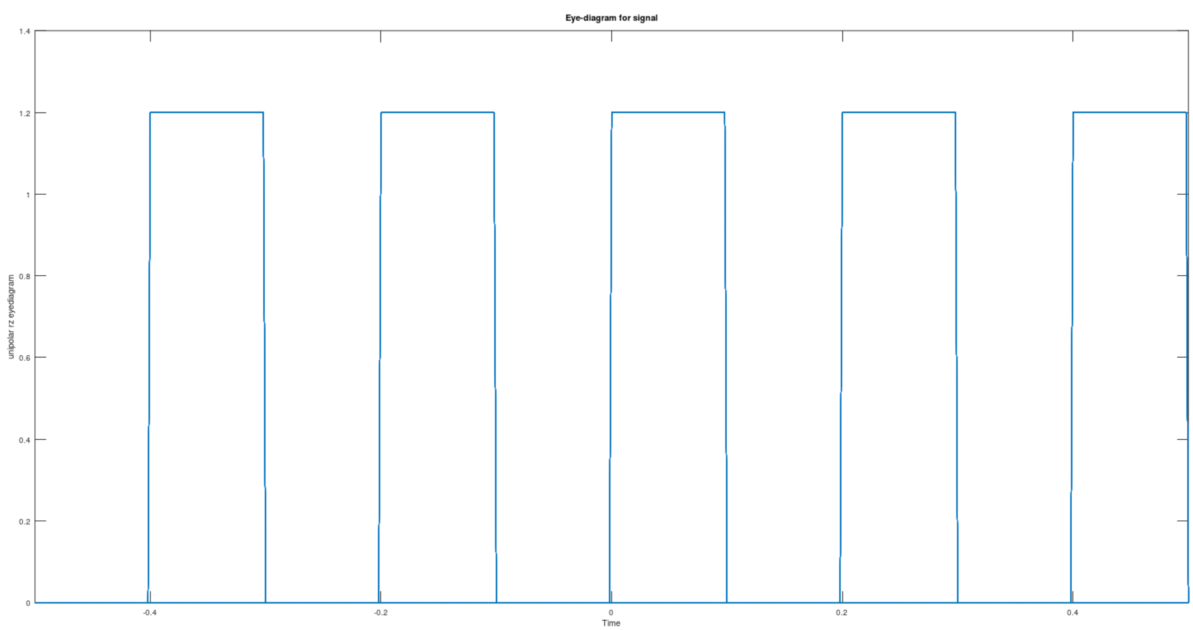
Line code the stream of bits (pulse shape) according to uni-polar return to zero (Supply voltages are: +1.2 V and -1.2V).

```
% Line encoding using Uni-polar RZ
unipolar_rz = positiveVoltage*bitsStream;

%extend the duration of each bit (each bit will be represented by 100 values)
delay = ones(100,1);
unipolar_rz = unipolar_rz.* delay;
unipolar_rz = reshape(unipolar_rz,1,[]);

% Line encoding using Uni-polar RZ
for i = 1:length(unipolar_rz)
    if unipolar_rz(i) == positiveVoltage
      i = i +50;
      j = 0;
      unipolar_rz(i) = 0;
      j++;
      if j == 50
        continue;
      end

    else
      unipolar_rz(i) = 0;
end
end


%plot the unipolar_rz
figure;
plot(t,unipolar_rz);
grid on; xlabel("Time"); ylabel("unipolar RZ Stream"); xlim([0 1000]);ylim([-2 2])
```



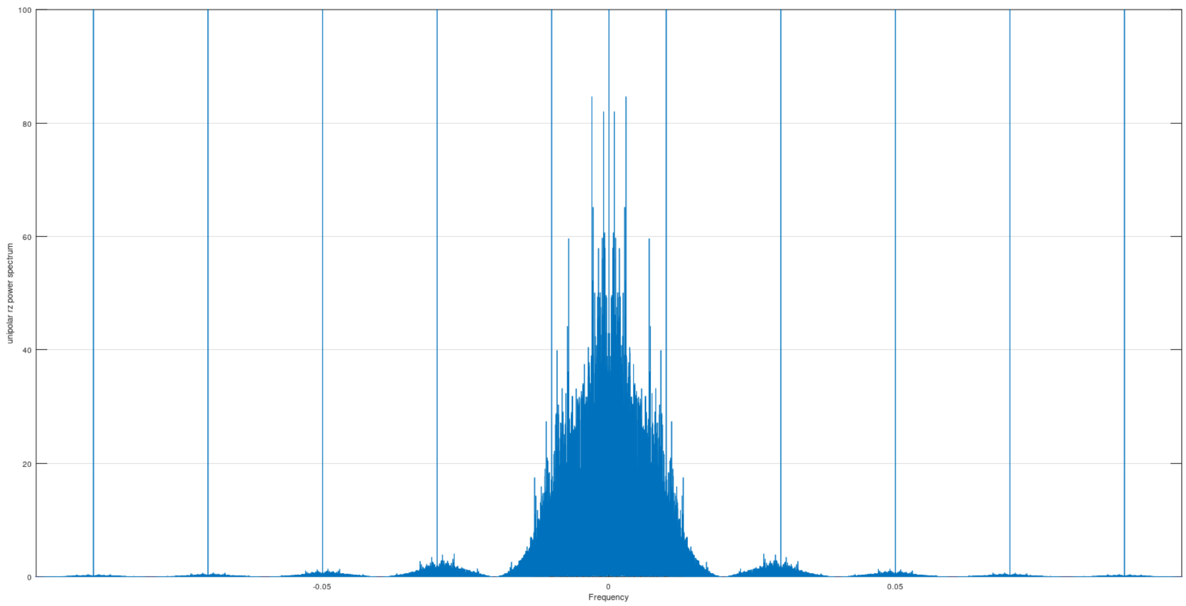
Plot the corresponding Eye diagram.

```
%eye diagram unipolar rz
eyediagram(unipolar_rz, 500); ylabel("unipolar rz eyediagram")
```



Plotting spectral domain of unipolar rz

```
% Compute the Fourier transform

% fourier transform and normalize the signal
plotSpectraldomains(unipolar_rz,f,100); ylabel("unipolar rz power spectrum")
```



Design a decision device and calculate the ber for uni-polar rz

```
% Decoding the received signal using Uni-polar RZ
rx_bitsAfterDecision_unipolar_rz = zeros(1,length(rx_unipolar_rz));
i = 1;
for i = 1:100:length(rx_unipolar_rz)
    if rx_unipolar_rz(i) > 0.6
       rx_bitsAfterDecision_unipolar_rz(i:i+99) = 1;
    else
       rx_bitsAfterDecision_unipolar_rz(i:i+99) = 0;
    end
    i = i + 1;
end
```

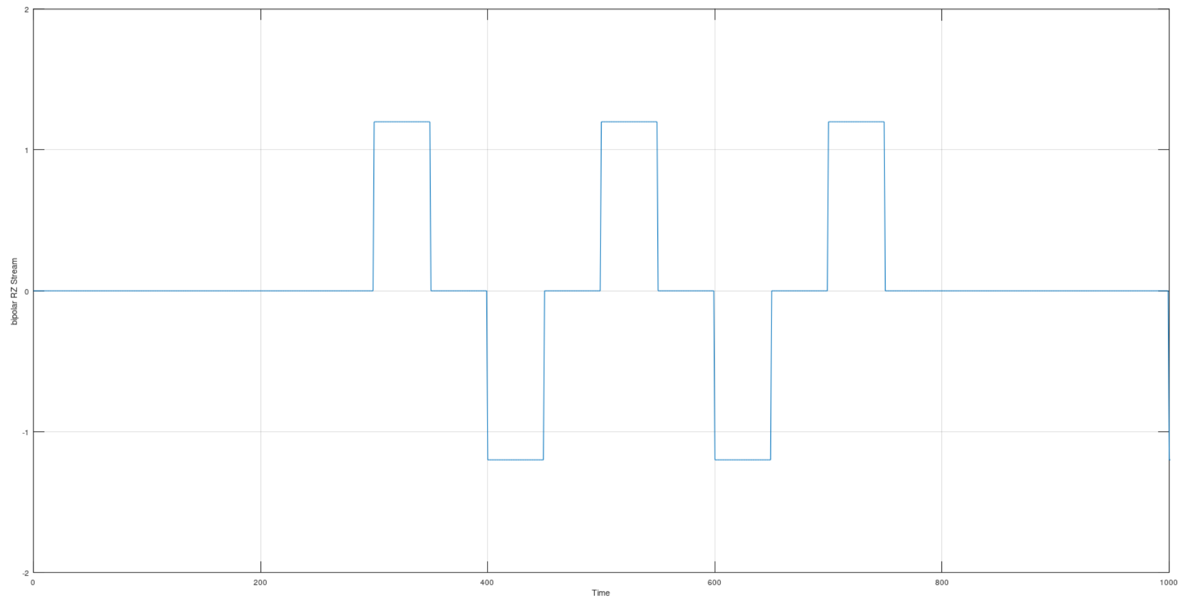
```
% Calculating BER for each line coding technique
ber_unipolar_nrz = sum(rx_bitsAfterDecision_unipolar_nrz~=extended_bit_stream)/length(extended_bit_stream);
ber_polar_nrz = sum(rx_bitsAfterDecision_polar_nrz~=extended_bit_stream)/length(extended_bit_stream);
ber_unipolar_rz = sum(rx_bitsAfterDecision_unipolar_rz~=extended_bit_stream)/length(extended_bit_stream);
ber_bipolar_rz = sum(rx_bitsAfterDecision_bipolar_rz~=extended_bit_stream)/length(extended_bit_stream);
ber_manchester = sum(rx_bitsAfterDecision_manchester~=extended_bit_stream)/length(extended_bit_stream);
% Displaying the results
disp(['BER for Uni-polar NRZ = ' num2str(ber_unipolar_rz)]);
disp(['BER for Polar NRZ = ' num2str(ber_polar_nrz)]);
disp(['BER for Uni-polar RZ = ' num2str(ber_unipolar_rz)]);
disp(['BER for Bipolar RZ = ' num2str(ber_bipolar_rz)]);
disp(['BER for Manchester = ' num2str(ber_manchester)]);
```

```
BER for Uni-polar NRZ = 0
BER for Polar NRZ = 0
BER for Uni-polar RZ = 0
BER for Bipolar RZ = 0
BER for Manchester = 0
```

Bipolar return to zero

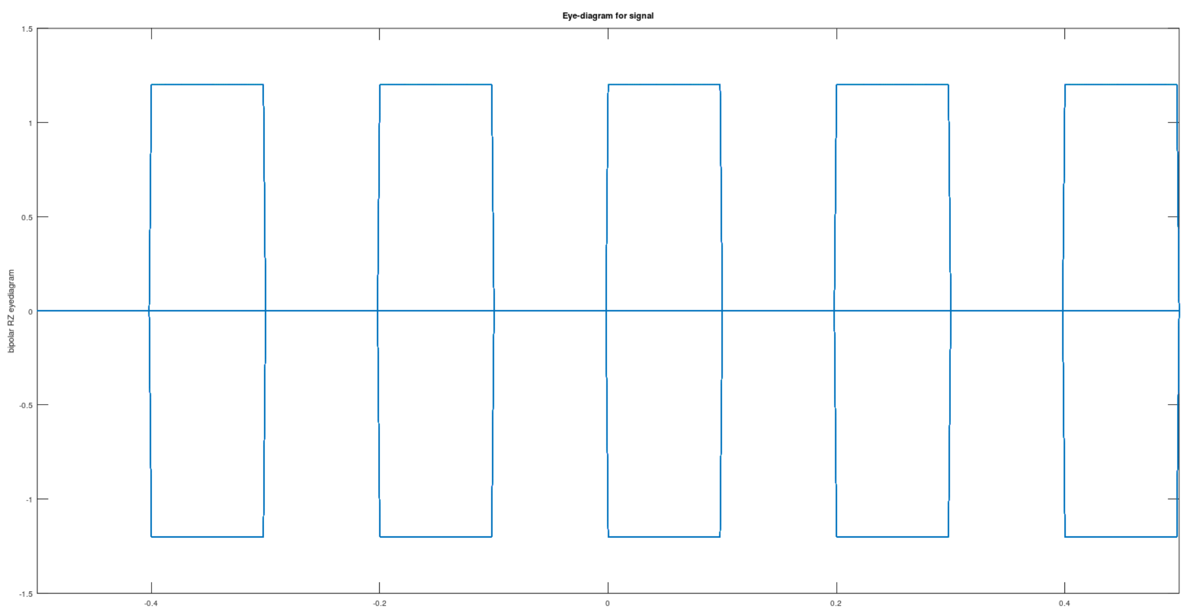
Line code the stream of bits (pulse shape) according to bi-polar return to zero (Supply voltages are: +1.2 V and -1.2V).

```
% Line encoding using Bipolar RZ

prev_polarity = 1;
for i = 1:length(bitsStream)
    if bitsStream(i) == 1
        BipolarbitsStream(i) = bitsStream(i)*prev_polarity;
        prev_polarity = -prev_polarity;
    else
        BipolarbitsStream(i) = 0;
    end
end

%extend the duration of each bit (each bit will be represented by 100 values)
delay = ones(100,1);
bipolar_rz = BipolarbitsStream*positiveVoltage;
bipolar_rz = bipolar_rz.* delay;
bipolar_rz = reshape(bipolar_rz,1,[]);

% Line encoding using Bipolar RZ
for i = 1:length(bipolar_rz)
    if bipolar_rz(i) != 0
      i = i +50;
       j = 0;
      bipolar_rz(i) = 0;
      j++;
      if j == 50
        continue;
      end
    else
      bipolar_rz(i) = 0;
end
end


%plot the bipolar_rz
figure;
plot(t,bipolar_rz);
grid on; xlabel("Time"); ylabel("bipolar RZ Stream"); xlim([0 1000]);ylim([-2 2])
```



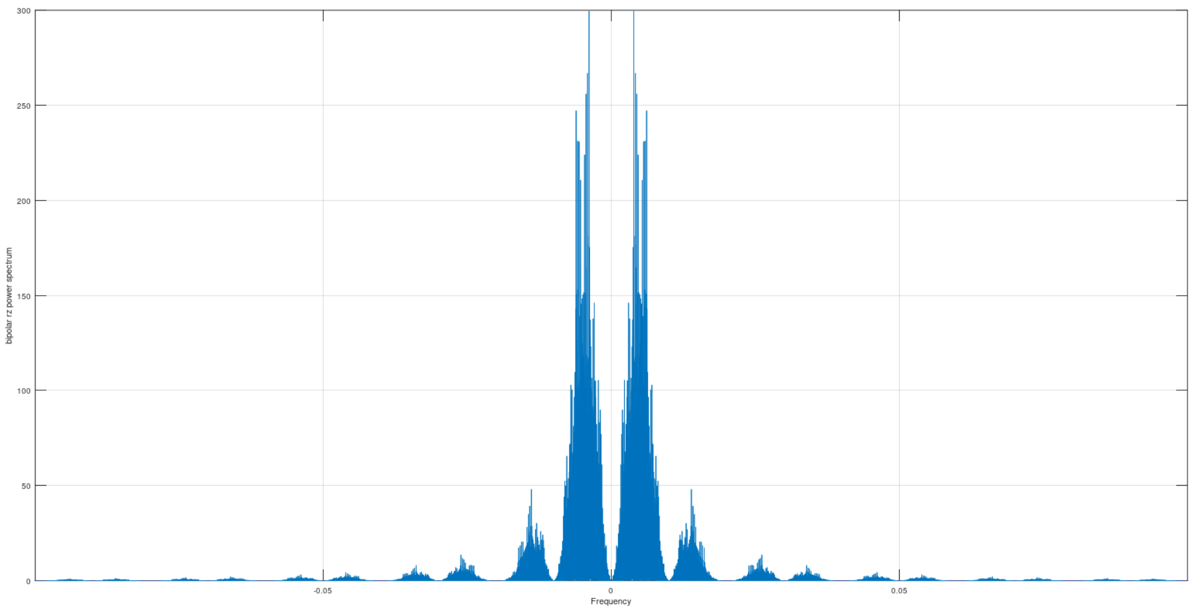
Plot the corresponding Eye diagram.

```
%eye diagram bipolar_rz
eyediagram(bipolar_rz, 500); ylabel("bipolar RZ eyediagram")
```



Plotting spectral domain of bipolar rz

```
% Compute the Fourier transform bipolar rz

% fourier transform and normalize the signal
plotSpectraldomains(bipolar_rz,f,300); ylabel("bipolar rz power spectrum")
```



Design a decision device and calculate the ber for bi-polar rz

```
% Decoding the received signal using bi-polar RZ
rx_bitsAfterDecision_bipolar_rz = zeros(1,length(rx_bipolar_rz));

i = 1;
for i = 1:100:length(rx_bipolar_rz)
    if rx_bipolar_rz(i) > 0.6 || rx_bipolar_rz(i) < -0.6
       rx_bitsAfterDecision_bipolar_rz(i:i+99) = 1;
    else
       rx_bitsAfterDecision_bipolar_rz(i:i+99) = 0;
    end
    i = i + 1;
end
```

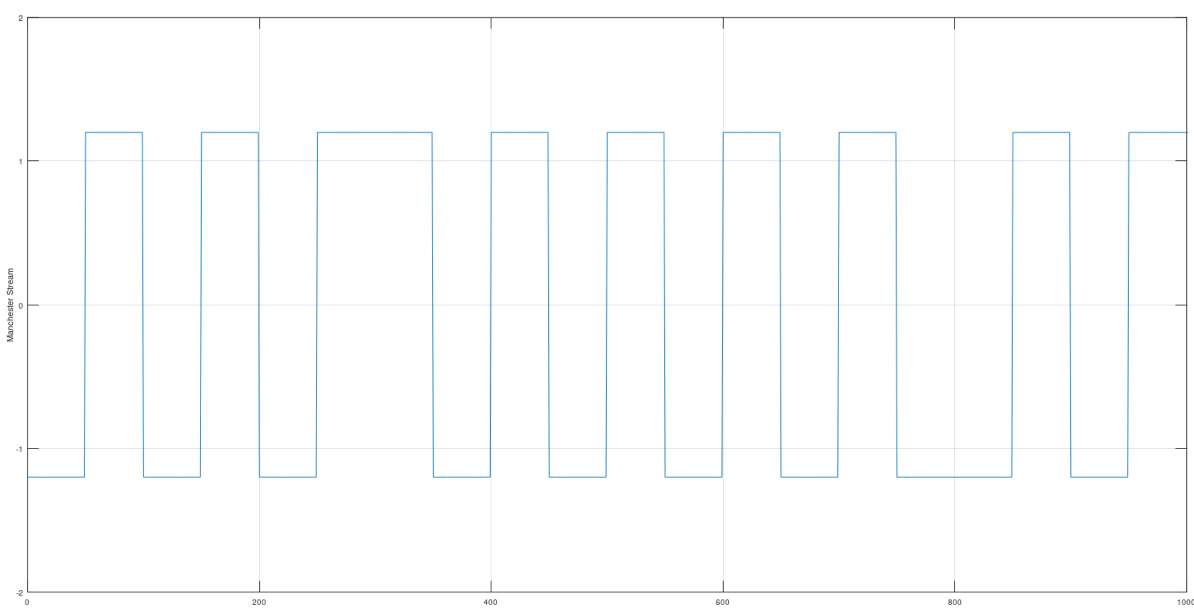
```
% Calculating BER for each line coding technique
ber_unipolar_nrz = sum(rx_bitsAfterDecision_unipolar_nrz~=extended_bit_stream)/length(extended_bit_stream);
ber_polar_nrz = sum(rx_bitsAfterDecision_polar_nrz~=extended_bit_stream)/length(extended_bit_stream);
ber_unipolar_rz = sum(rx_bitsAfterDecision_unipolar_rz~=extended_bit_stream)/length(extended_bit_stream);
ber_bipolar_rz = sum(rx_bitsAfterDecision_bipolar_rz~=extended_bit_stream)/length(extended_bit_stream);
ber_manchester = sum(rx_bitsAfterDecision_manchester~=extended_bit_stream)/length(extended_bit_stream);
% Displaying the results
disp(['BER for Uni-polar NRZ = ' num2str(ber_unipolar_rz)]);
disp(['BER for Polar NRZ = ' num2str(ber_polar_nrz)]);
disp(['BER for Uni-polar RZ = ' num2str(ber_unipolar_rz)]);
disp(['BER for Bipolar RZ = ' num2str(ber_bipolar_rz)]);
disp(['BER for Manchester = ' num2str(ber_manchester)]);
```

```
BER for Uni-polar NRZ = 0
BER for Polar NRZ = 0
BER for Uni-polar RZ = 0
BER for Bipolar RZ = 0
BER for Manchester = 0
```

Manchester

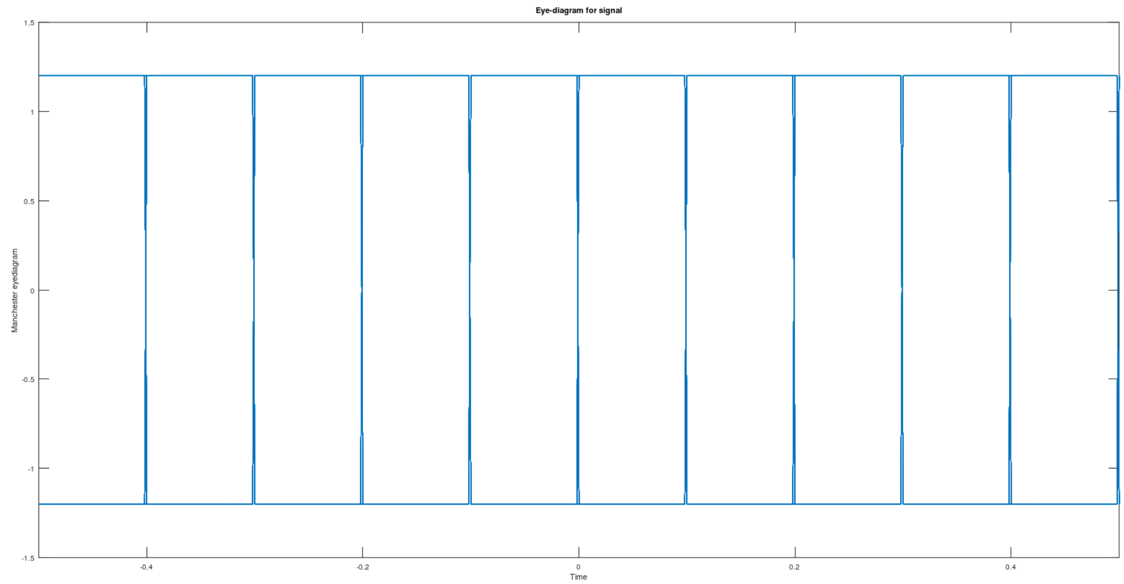
Line code the stream of bits (pulse shape) according to manchester (Supply voltages are: +1.2 V and -1.2V).

```
% Line encoding using Manchester
Manchester = positiveVoltage*(bitsStream*2-1); % Convert 0's to -1.2's and 1's to +1.2's

%extend the duration of each bit (each bit will be represented by 100 values)
delay = ones(100,1);
Manchester = Manchester.* delay;
Manchester = reshape(Manchester,1,[]);

% Line encoding using Manchester
for i = 1:length(bitsStream)
    if bitsStream(i) == 0
        Manchester((i-1)*100+1:50+(i-1)*100) = -1.2;
        Manchester(51+(i-1)*100:i*100) = 1.2;
     else
        Manchester((i-1)*100+1:50+(i-1)*100) = 1.2;
        Manchester(51+(i-1)*100:i*100) = -1.2;
    end
end

%plot the Manchester
figure;
plot(t,Manchester(1:N));
grid on; xlabel("Time"); ylabel("Manchester Stream"); xlim([0 1000]);ylim([-2 2])
```



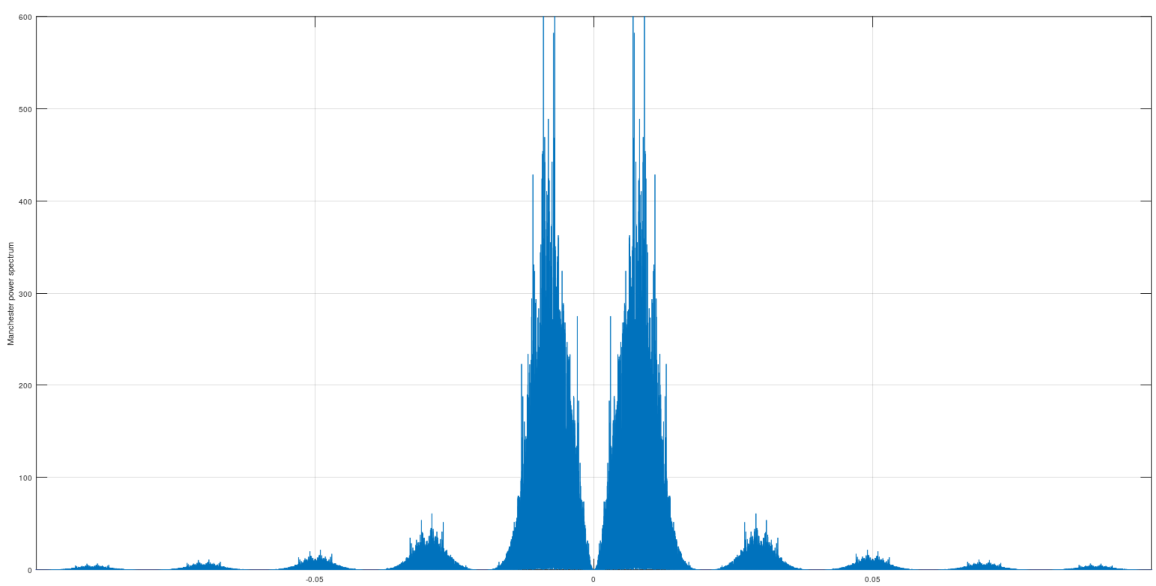
Plot the corresponding Eye diagram.

```
%eye diagram manchester
eyediagram(Manchester, 500); ylabel("Manchester eyediagram")
```



Plotting spectral domain of Manchester

```
% fourier transform and normalize the signal
plotSpectraldomains(Manchester,f,600); ylabel("Manchester power spectrum")
```



Design a decision device and calculate the ber for Manchester.

```
% Decoding the received signal using manchester
rx_bitsAfterDecision_manchester = zeros(1,length(rx_manchester));
i = 1;
for i = 1:100:length(rx_manchester)
    if rx_manchester(i) >  0
       rx_bitsAfterDecision_manchester(i:i+99) = 1;
    else
       rx_bitsAfterDecision_manchester(i:i+99) = 0;
    end
    i = i + 1;
end

% Calculating BER for each line coding technique
ber_unipolar_nrz = sum(rx_bitsAfterDecision_unipolar_nrz~=extended_bit_stream)/length(extended_bit_stream);
ber_polar_nrz = sum(rx_bitsAfterDecision_polar_nrz~=extended_bit_stream)/length(extended_bit_stream);
ber_unipolar_rz = sum(rx_bitsAfterDecision_unipolar_rz~=extended_bit_stream)/length(extended_bit_stream);
ber_bipolar_rz = sum(rx_bitsAfterDecision_bipolar_rz~=extended_bit_stream)/length(extended_bit_stream);
ber_manchester = sum(rx_bitsAfterDecision_manchester~=extended_bit_stream)/length(extended_bit_stream);
% Displaying the results
disp(['BER for Uni-polar NRZ = ' num2str(ber_unipolar_rz)]);
disp(['BER for Polar NRZ = ' num2str(ber_polar_nrz)]);
disp(['BER for Uni-polar RZ = ' num2str(ber_unipolar_rz)]);
disp(['BER for Bipolar RZ = ' num2str(ber_bipolar_rz)]);
disp(['BER for Manchester = ' num2str(ber_manchester)]);
```

```
BER for Uni-polar NRZ = 0
BER for Polar NRZ = 0
BER for Uni-polar RZ = 0
BER for Bipolar RZ = 0
BER for Manchester = 0
```

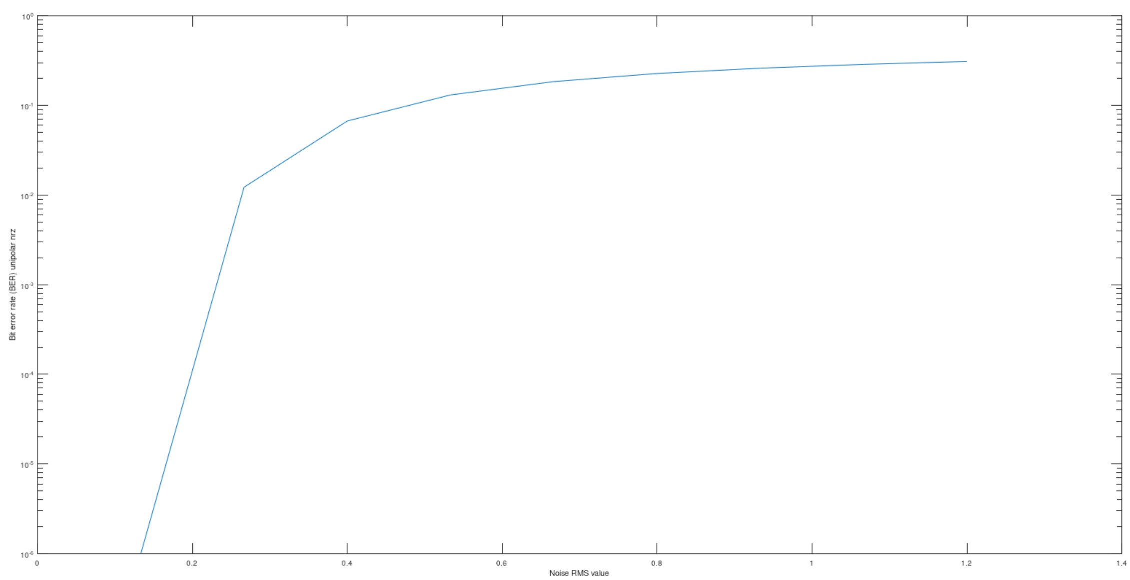
8. Add noise to the received signal (Hint: use n = sigma * randn(1,length(t)),where t is time vector and sigma is the noise rms value).

9. Sweep on the value of sigma (10 values ranges from 0 to the maximum supply voltage) and calculate the corresponding BER for each value of sigma.

10. Repeat the previous steps for different line coding and plot BER versus sigma for the different line coding in the same figure, where y-axis is in the log scale (Hint: use semilogy).

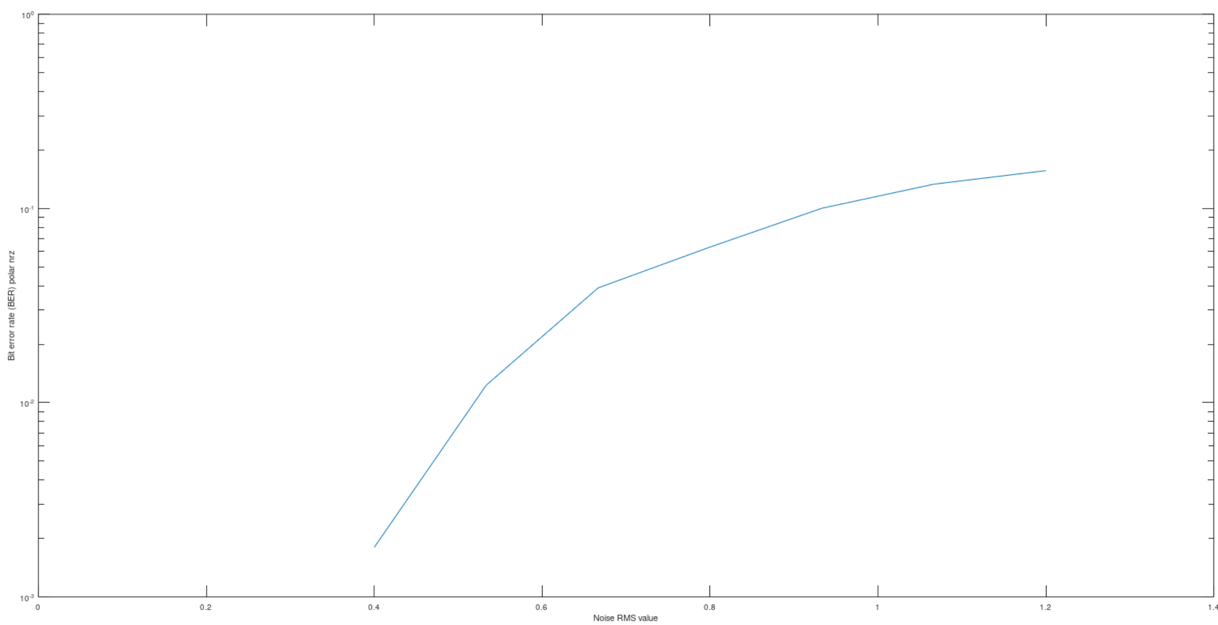
Unipolar NRZ

```
% Decoding the received signal using Uni-polar NRZ
rx_bitsAfterDecision_unipolar_nrz = zeros(1,length(rx_unipolar_nrz));
ber_unipolar_nrz = zeros(1, 10);
for j = 1:length(sigma)
rx_unipolar_nrz =  sigma(j).* randn(1,length(t)) + UniPolarNRZbitSream;
i = 1;
for i = 1:length(rx_unipolar_nrz)
    if rx_unipolar_nrz(i) > 0.6
       rx_bitsAfterDecision_unipolar_nrz(i) = 1;
    else
       rx_bitsAfterDecision_unipolar_nrz(i) = 0;
    end
    i = i + 1;
end
ber_unipolar_nrz(j) = sum(rx_bitsAfterDecision_unipolar_nrz~=extended_bit_stream)/length(extended_bit_stream);
end
%plot semilogy
figure;
semilogy(sigma, ber_unipolar_nrz);xlabel('Noise RMS value');
ylabel('Bit error rate (BER) unipolar nrz');
```



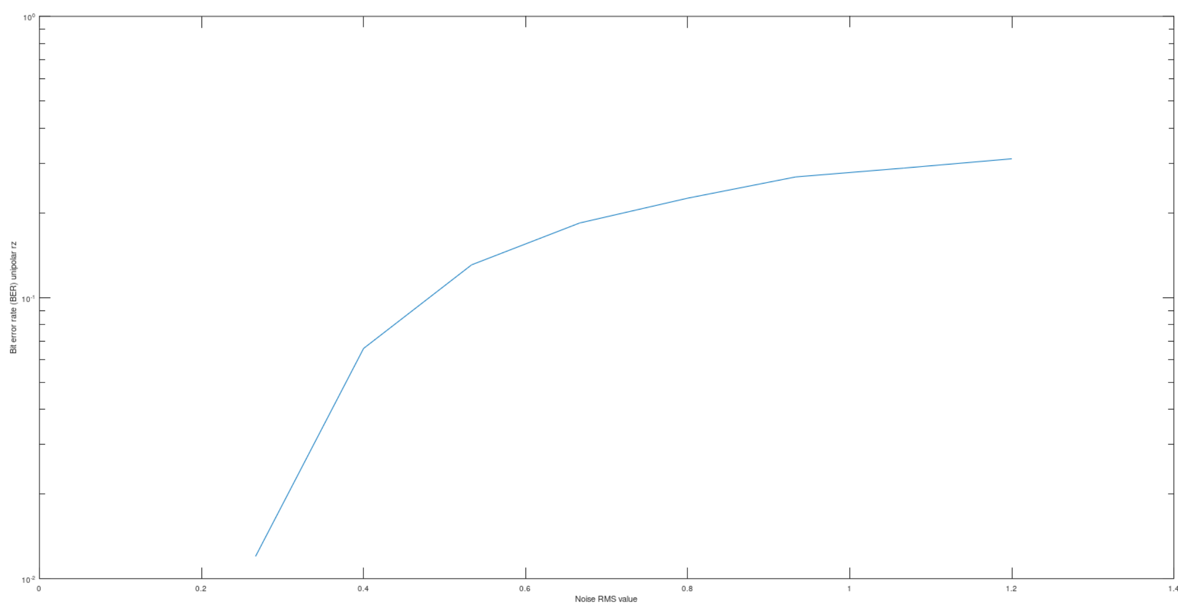
Polar non return to zero

```
% Decoding the received signal using polar NRZ
rx_bitsAfterDecision_polar_nrz = zeros(1,length(rx_polar_nrz));
ber_polar_nrz = zeros(1, 10);
for j = 1:length(sigma)
rx_polar_nrz =  sigma(j).* randn(1,length(t)) + polar_nrz;
i = 1;
for i = 1:100:length(rx_polar_nrz)
    if rx_polar_nrz(i) > 0
       rx_bitsAfterDecision_polar_nrz(i:i+99) = 1;
    else
       rx_bitsAfterDecision_polar_nrz(i:i+99) = 0;
    end
    i = i + 1;
    end
    ber_polar_nrz(j) = sum(rx_bitsAfterDecision_polar_nrz~=extended_bit_stream)/length(extended_bit_stream);
end
%plot semilogy
figure;
semilogy(sigma, ber_polar_nrz);xlabel('Noise RMS value');
ylabel('Bit error rate (BER) polar nrz');
```



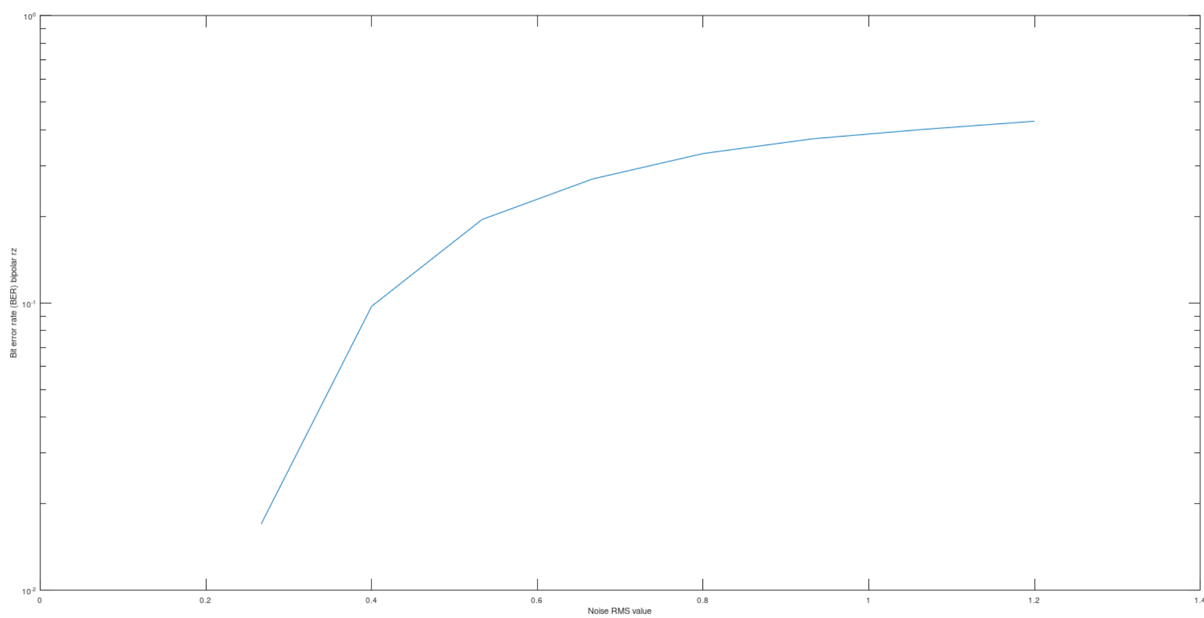
Uni-polar return to zero

```
% Decoding the received signal using Uni-polar RZ
rx_bitsAfterDecision_unipolar_rz = zeros(1,length(rx_unipolar_rz));
ber_unipolar_rz = zeros(1, 10);
for j = 1:length(sigma)
rx_unipolar_rz =  sigma(j).* randn(1,length(t)) + unipolar_rz;
i = 1;
for i = 1:100:length(rx_unipolar_rz)
    if rx_unipolar_rz(i) > 0.6
       rx_bitsAfterDecision_unipolar_rz(i:i+99) = 1;
    else
       rx_bitsAfterDecision_unipolar_rz(i:i+99) = 0;
    end
    i = i + 1;
end
ber_unipolar_rz(j) = sum(rx_bitsAfterDecision_unipolar_rz~=extended_bit_stream)/length(extended_bit_stream);
end
%plot semilogy
figure;
semilogy(sigma, ber_unipolar_rz);xlabel('Noise RMS value');
ylabel('Bit error rate (BER) unipolar rz');
```



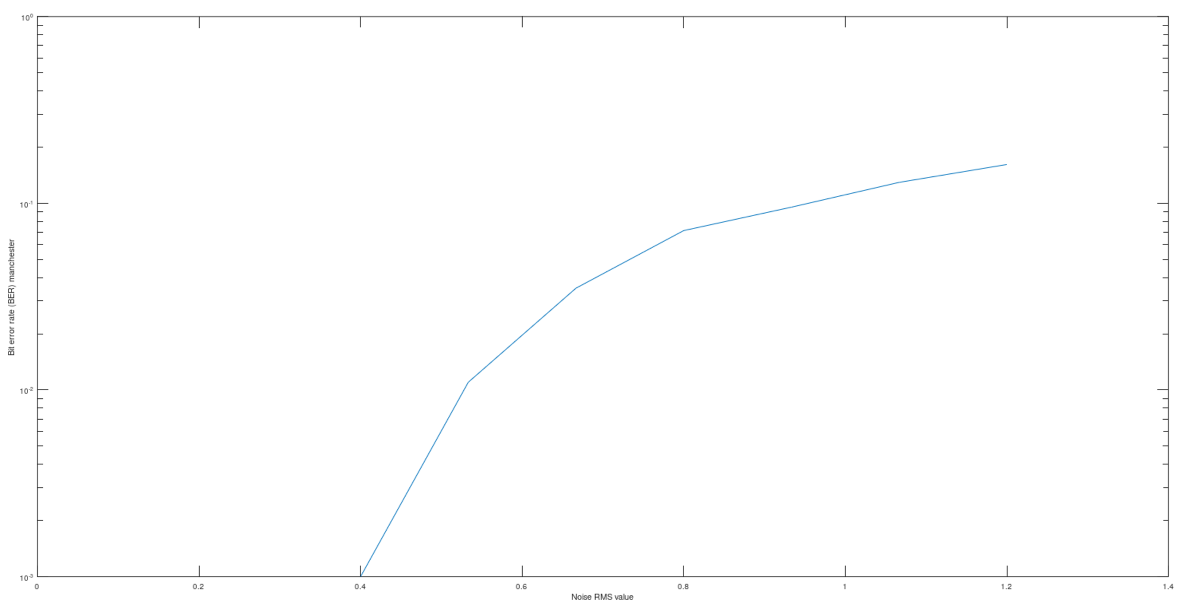
Bipolar return to zero

```
% Decoding the received signal using bi-polar RZ
rx_bitsAfterDecision_bipolar_rz = zeros(1,length(rx_bipolar_rz));
%save values for rx bits with noise and rx bits after decision
rxBiPolar = zeros(10, length(t));
rxBiPolarDecision = zeros(10, length(t));
ber_bipolar_rz = zeros(1, 10);
for j = 1:length(sigma)
rx_bipolar_rz =  sigma(j).* randn(1,length(t)) + bipolar_rz;
rxBiPolar(j,:) = rx_bipolar_rz;
i = 1;
for i = 1:100:length(rx_bipolar_rz)
    if rx_bipolar_rz(i) > 0.6 || rx_bipolar_rz(i) < -0.6
       rx_bitsAfterDecision_bipolar_rz(i:i+99) = 1;
    else
       rx_bitsAfterDecision_bipolar_rz(i:i+99) = 0;
    end
    i = i + 1;
  end
  rxBiPolarDecision(j,:) = rx_bitsAfterDecision_bipolar_rz;
  ber_bipolar_rz(j) = sum(rx_bitsAfterDecision_bipolar_rz~=extended_bit_stream)/length(extended_bit_stream);
end
%plot semilogy
figure;
semilogy(sigma, ber_bipolar_rz);xlabel('Noise RMS value');
ylabel('Bit error rate (BER) bipolar rz');
```



Manchester

```
% Decoding the received signal using manchester
rx_bitsAfterDecision_manchester = zeros(1,length(rx_manchester));
ber_manchester = zeros(1, 10);
for j = 1:length(sigma)
rx_manchester =  sigma(j).* randn(1,length(t)) + Manchester;
i = 1;
for i = 1:100:length(rx_manchester)
    if rx_manchester(i) >  0
       rx_bitsAfterDecision_manchester(i:i+99) = 1;
    else
       rx_bitsAfterDecision_manchester(i:i+99) = 0;
    end
    i = i + 1;
  end
  ber_manchester(j) = sum(rx_bitsAfterDecision_manchester~=extended_bit_stream)/length(extended_bit_stream);

end
%plot semilogy
figure;
semilogy(sigma, ber_manchester);xlabel('Noise RMS value');
ylabel('Bit error rate (BER) manchester');
```



Displaying BER at different sigma for all line codes

```
% Calculating BER for each line coding technique
% Displaying the results
disp(['BER for Uni-polar NRZ with noise = ' num2str(ber_unipolar_rz)]);
disp(['BER for Polar NRZ with noise = ' num2str(ber_polar_nrz)]);
disp(['BER for Uni-polar RZ with noise = ' num2str(ber_unipolar_rz)]);
disp(['BER for Bipolar RZ with noise = ' num2str(ber_bipolar_rz)]);
disp(['BER for Manchester with noise = ' num2str(ber_manchester)]);
```

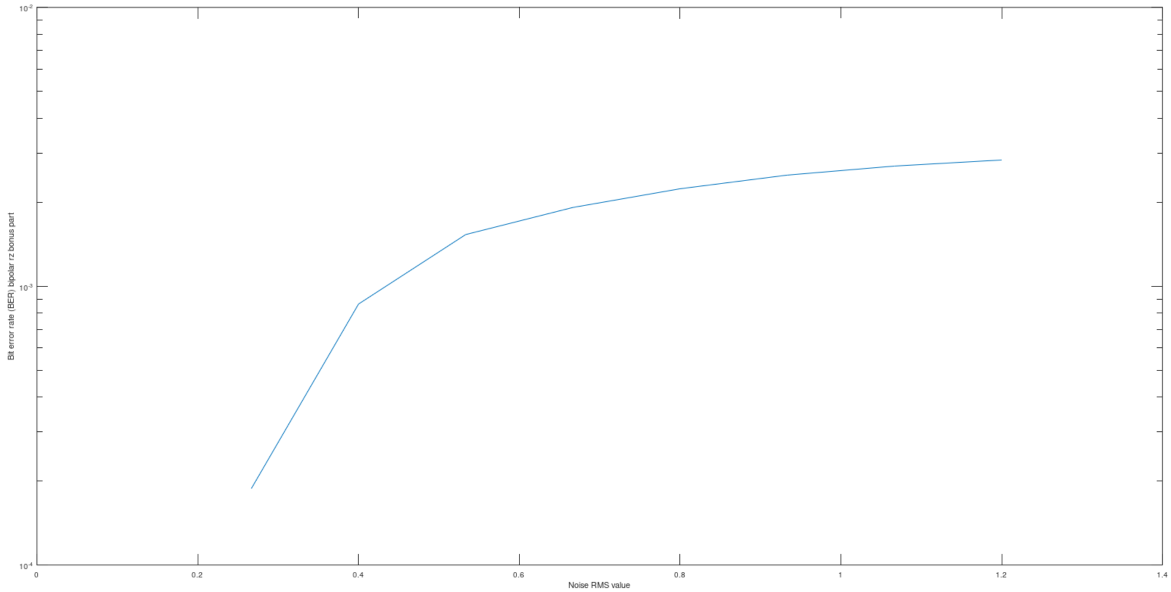
```
BER for Uni-polar NRZ with noise = 0      0      0.012   0.0659  0.1307  0.1841  0.2256  0.2685  0.2886  0.3115
BER for Polar NRZ with noise = 0      0      0       0.0018  0.0123  0.039   0.0633  0.1003  0.1334  0.1566
BER for Uni-polar RZ with noise = 0      0      0.012   0.0659  0.1307  0.1841  0.2256  0.2685  0.2886  0.3115
BER for Bipolar RZ with noise = 0      0      0.017   0.0973  0.1951  0.2699  0.3309  0.3727  0.4017  0.4287
BER for Manchester with noise = 0      0      0       0.001   0.011   0.0351  0.0714  0.0953  0.1294  0.1613
```

Bonus

11. (Bonus) For the case of Bipolar return to zero , design an error detection circuit. Count the number of detected errors in case of different number of sigma (Use the output of step 8).

```
%bonus
% Error detection

detected_bits = zeros(1, 10);
for j = 1:length(sigma)
    previousPolarity = -1;
    for i = 50:100:length(t) %start from mid of bit
        if rxBiPolar(j,i) > 0.6 || rxBiPolar(j,i) < -0.6 %check if consecutive ones have same polarity
            if sign(rxBiPolar(j,i)) == previousPolarity
                detected_bits(j) = detected_bits(j) + 1;
            end
            previousPolarity = sign(rxBiPolar(j,i));
        end
    end
    disp(['Number of detected errors for sigma ' num2str(sigma(j)) ' is ' num2str(detected_bits(j))]);
end
%plot semilogy
figure;
semilogy(sigma, detected_bits/length(extended_bit_stream));xlabel('Noise RMS value');
ylabel('Bit error rate (BER) bipolar rz bonus part');
```

```
Number of detected errors for sigma 0 is 0
Number of detected errors for sigma 0.13333 is 0
Number of detected errors for sigma 0.26667 is 188
Number of detected errors for sigma 0.4 is 863
Number of detected errors for sigma 0.53333 is 1532
Number of detected errors for sigma 0.66667 is 1916
Number of detected errors for sigma 0.8 is 2236
Number of detected errors for sigma 0.93333 is 2503
Number of detected errors for sigma 1.0667 is 2697
Number of detected errors for sigma 1.2 is 2836
>>
```



Part II

Using the previous codes, you can implement binary phase shift-keying BPSK transmitter and receiver.

Code:

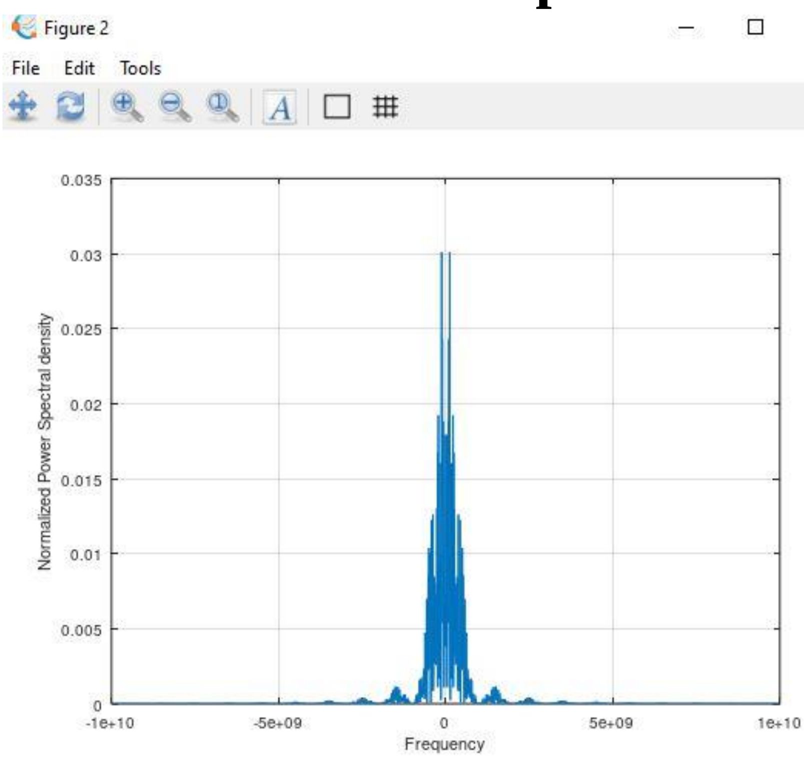
```
%Transmitter

% generate stream of 100 random bits

bits = randi([0 1],1,100);
% make the zeros into -1 for polar non return to zero line coding
for index = 1:length(bits)
  if bits(index) == 0
    bits(index)= -1;
  endif
end
% extend the duration of each bit (each bit will be represented by 100 values)

delay = ones(100,1);

pulses = bits.* delay;

pulses = reshape(pulses,1,[]);
```

```
%time domain
N = length(pulses);
ts = 1e-11;
T = ts * N;
tb = 100* ts;
t = 0:ts:T-ts;

%frequency domain
df = 1/T;
fs = 1/ts;
Rb = 1/tb;
%N is even
f = -(0.5 * fs):df: (0.5 * fs -df);

% fourier transform and normalize the signal
PULSES = fftshift(fft(pulses))/N;
% plot the spectral domain
PULSES = PULSES .* PULSES;
figure
plot(f,abs(PULSES)); grid on; xlabel("Frequency"); ylabel("Normalized Power Spectral density"); xlim([-
1e10 1e10]);

%BPSK modulation
fc = 1e9;
carrier = cos(2 * pi * fc * t);
modPulses =  pulses .* carrier;
%plot modulated BPSK with time
figure
```

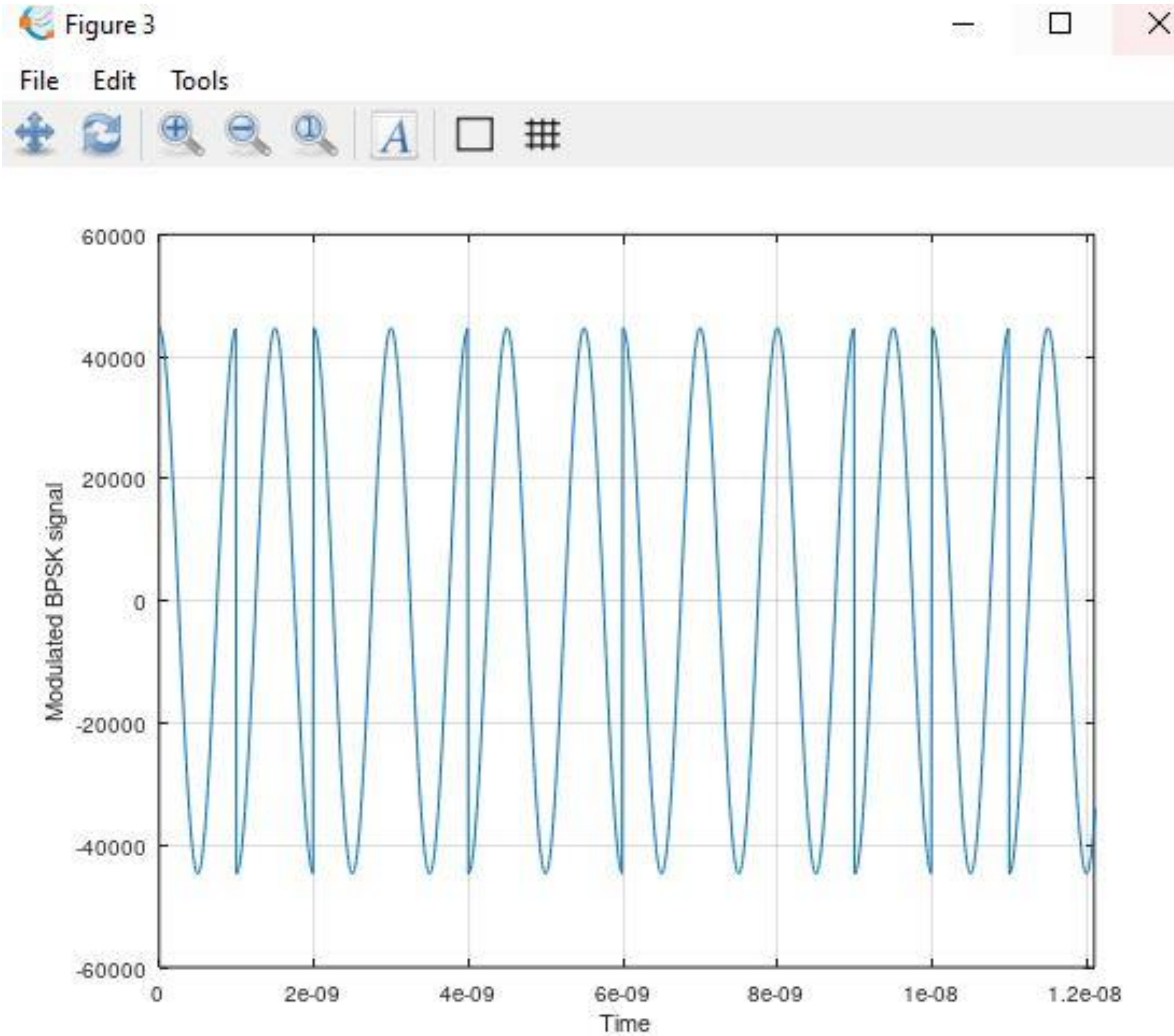
```
plot(t,modPulses); grid on; xlabel("Time"); ylabel("Modulated BPSK signal"); xlim([0 10e-9]); ylim([-1.1 1.1]);


%Plot spectrum of the modulated BPSK signal

MODPULSES = fftshift(fft(modPulses))/N ;

MODPULSES = MODPULSES .* MODPULSES;

figure

plot(f,abs(MODPULSES)); grid on; xlabel("Frequency"); ylabel("Spectrum of modulated BPSK signal");xlim([-1e10 1e10]);




%Receiver
carrier2 = cos(2 * pi * fc * t);

modPulses2 =  modPulses .* carrier2;

MODPULSES2 = fftshift(fft(modPulses2))/N;

MODPULSES2 = MODPULSES2 .* MODPULSES2;


%low pass filter

h = abs(f)<1e9+50;

MODPULSES2 = h.* MODPULSES2;

MODPULSES2 = sqrt(MODPULSES2);

pulses2 = real(ifft(fftshift(MODPULSES2))* N);


average = ones(1,100);

for index = 1:length(average)

  average(index) = pulses(100 * index -50);     % get mid point of each pulse

end

%Decision device

decision = sign(average);
```

```
%Compare output of decision level with the generated stream of bits
errorBits = 0;
for index = 1:length(bits)
   if bits(index) != decision(index)
      errorBits++;
  endif
end
bitErrorRate = errorBits/length(bits);
```

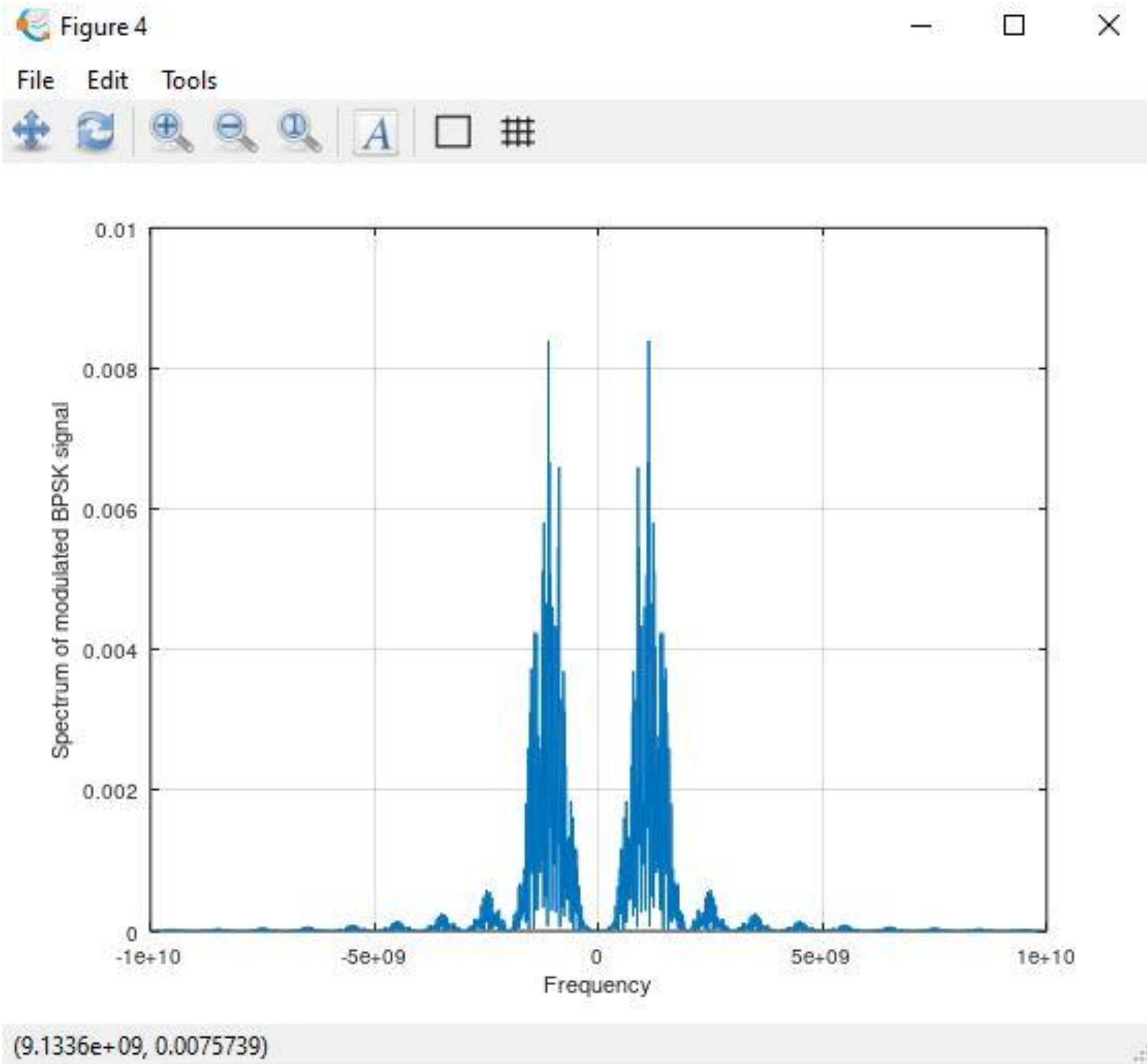
Transmitter:
3. Plot the spectral domains



4. Plot the time domain of the modulated BPSK signal (fc = 1GHz).



5. Plot the spectrum of the modulated BPSK signal.



Receiver:

7. Compare the output of decision level with the generated stream of bits in the transmitter.