QPSK

ZAINAB RAZA 2022102013 **ADITHI SAMUDRALA 2022102065**

1. INTRODUCTION

Communication System :

A communication model describes the process of how information is transmitted and received between a sender and a receiver.

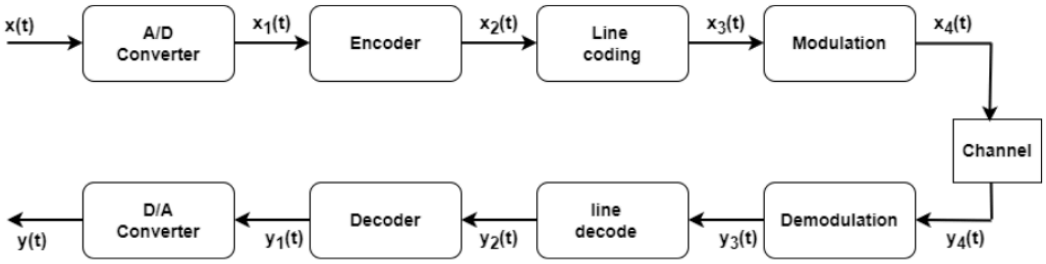


Figure 1: System model

The Quadrature Phase Shift Keying (**QPSK**) is a common Phase Shift Keying digital modulation, due to its simplicity, excellent power and bandwidth efficiency.It allows the signal to carry twice as much information as ordinary PSK using the same bandwidth. In this project we have simulated the QPSK Communication Model of the type-2 kind where the symbols are shifted by π/4.

2. A/D converter

In this section, the audio file is converted into binary data. To do this, audio data is first scaled to fit within the range of a 16-bit signed integer. Then, it is converted into binary representation, where each 16-bit integer is converted into a 16-character string of binary digits. Binary representation is concatenated into a row vector after which it is converted into a numeric vector consisting of 0's and 1's.

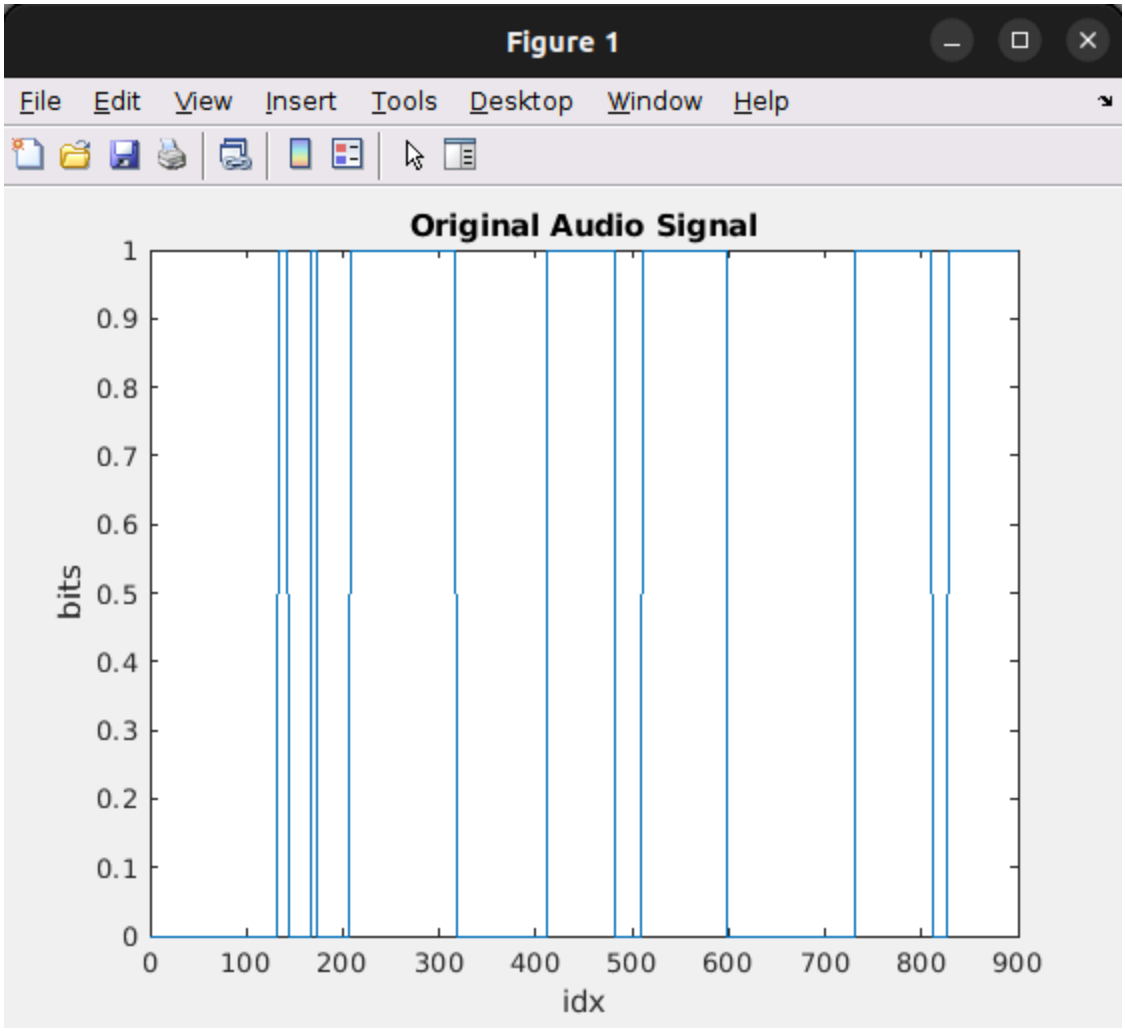


Figure 1: A/D converter output

3. Encoder

The encoder block maps the incoming bitstream from the A/D converter as per the QPSK constellation shifted counterclockwise by π/4 i.e. it takes two consecutive bits and maps them to appropriate values.

Mapping:

(0, 0) → (a, a); (1, 0) → (−a, a); (1, 1) → (-a, −a); (0, 1) → (a, −a).

In our case: a = 1

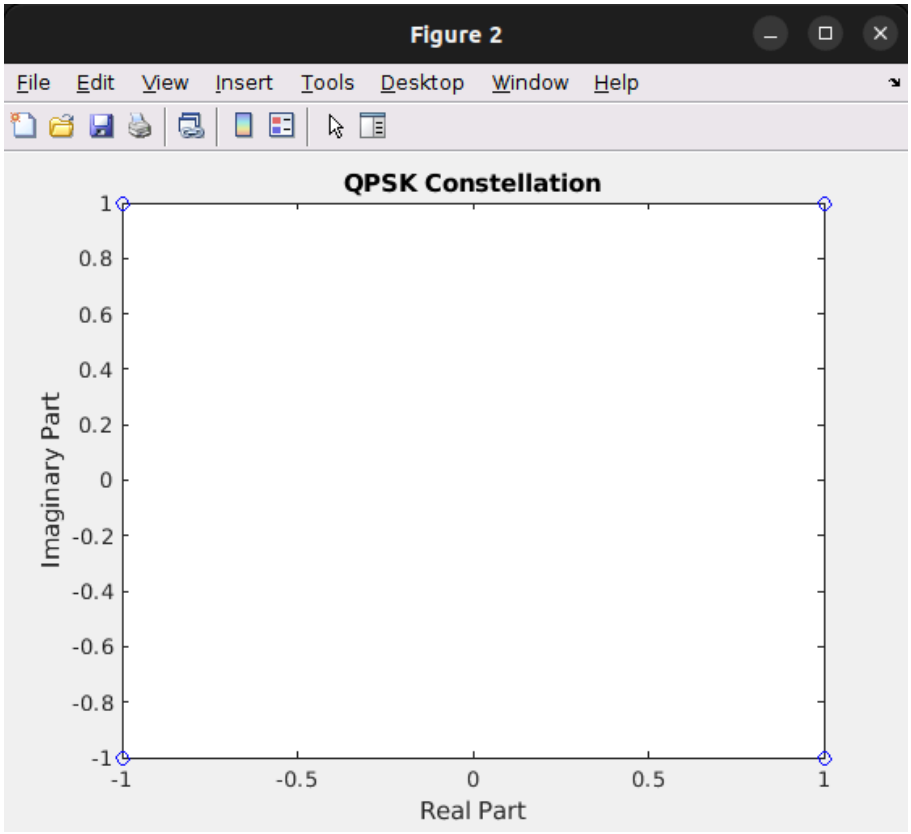
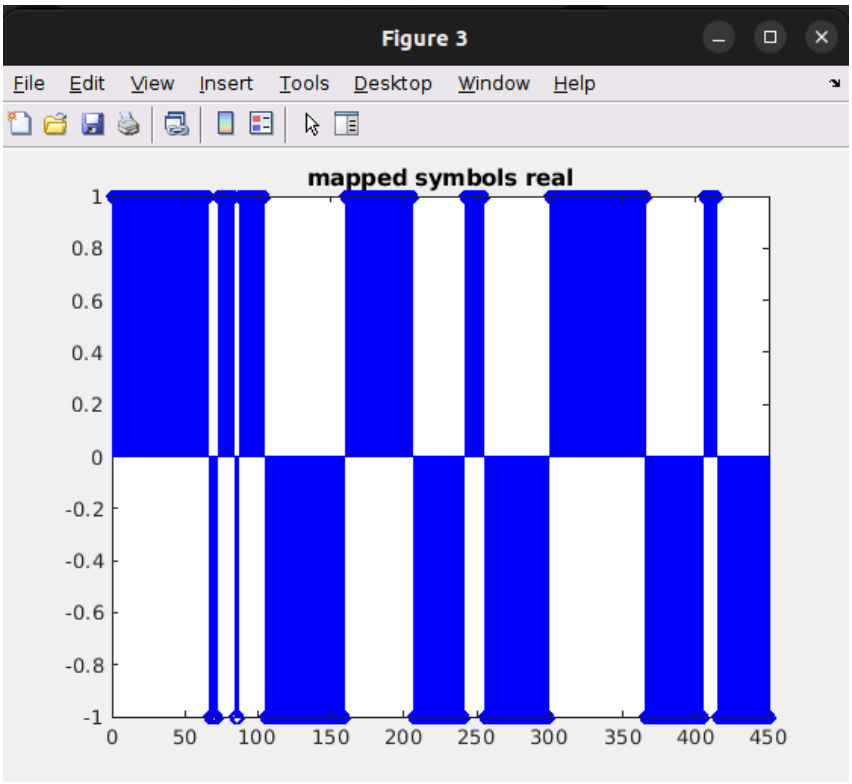


Figure 2: QPSK mapping without upsampling



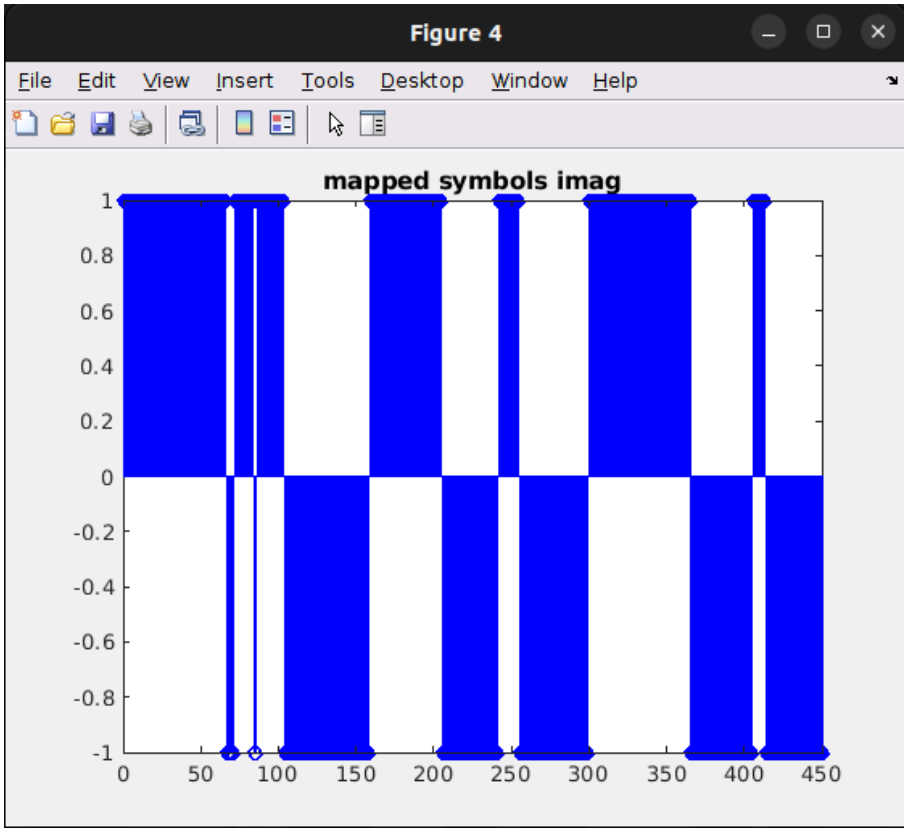


Figure 4: mapped symbols imaginary part

4. Line Coding

We convert binary data into actual baseband waveforms for further modulation for transmission.

Output of line coder:

$$\mathbf{x}_3(t) = \sum_k a_k p(t - kT_b),$$

We're asked to line code using two different pulses:

Rectangular Pulse:

a. The rectangular pulse has a constant amplitude within its duration and then abruptly drops to zero. It's represented by a rectangle in the time domain.
b. For line coding with a rectangular pulse:
    i. When the bit is 1, the pulse is transmitted with its full amplitude for the duration of Tb/2
    ii. When the bit is 0, there's no transmission, so the signal remains at zero.

iii. We took the length as 20.

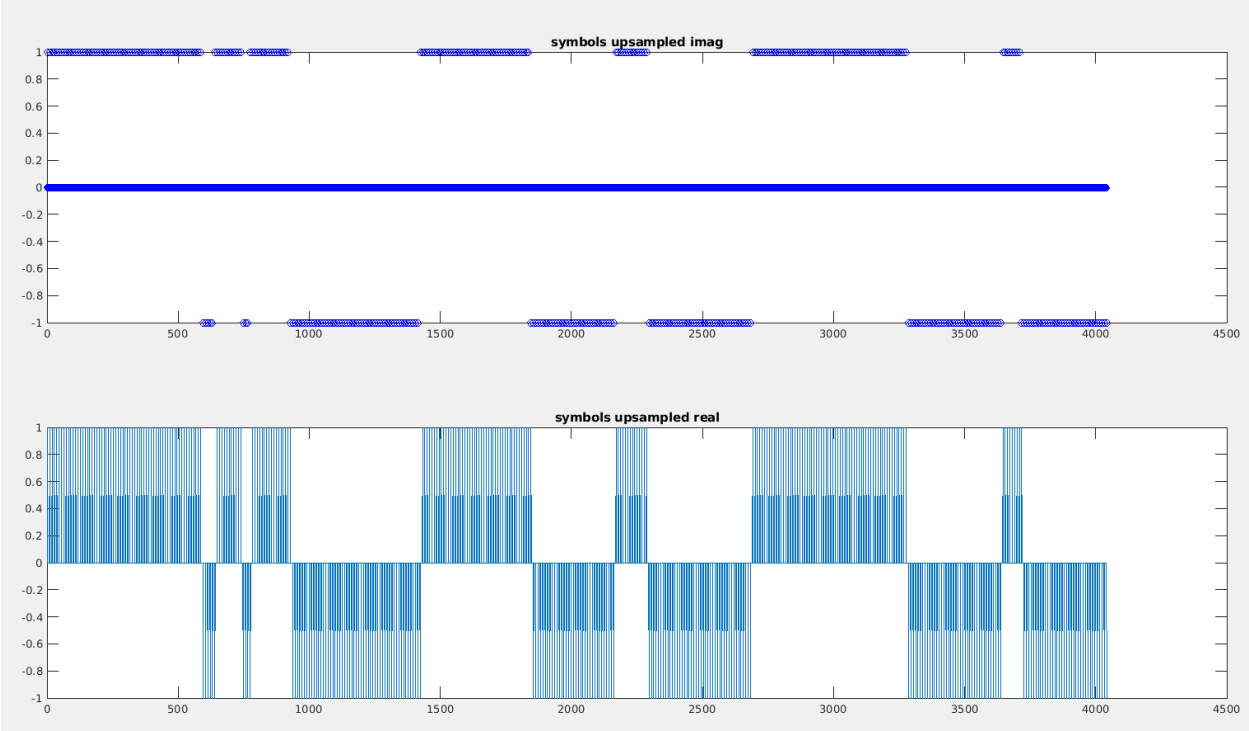
Raised Cosine Pulse:

c. The raised cosine pulse is a smoother pulse shape compared to the rectangular pulse. It's designed to minimize intersymbol interference.
d. For line coding with a raised cosine pulse:
    i. Similar to the rectangular pulse, when the bit is 1, the pulse is transmitted with its full amplitude for the duration of Tb
    ii. However, instead of an abrupt transition to zero, the raised cosine pulse smoothly transitions to zero at the edges. This helps in reducing the bandwidth of the transmitted signal and mitigating distortion.
    iii. We took the length as 9.

Before we line code, we oversample our data

- Oversampling factor of 10 for rectangular pulse
- Oversampling factor of 9 for raised cosine pulse

Then we convolute with the pulses for line coder output



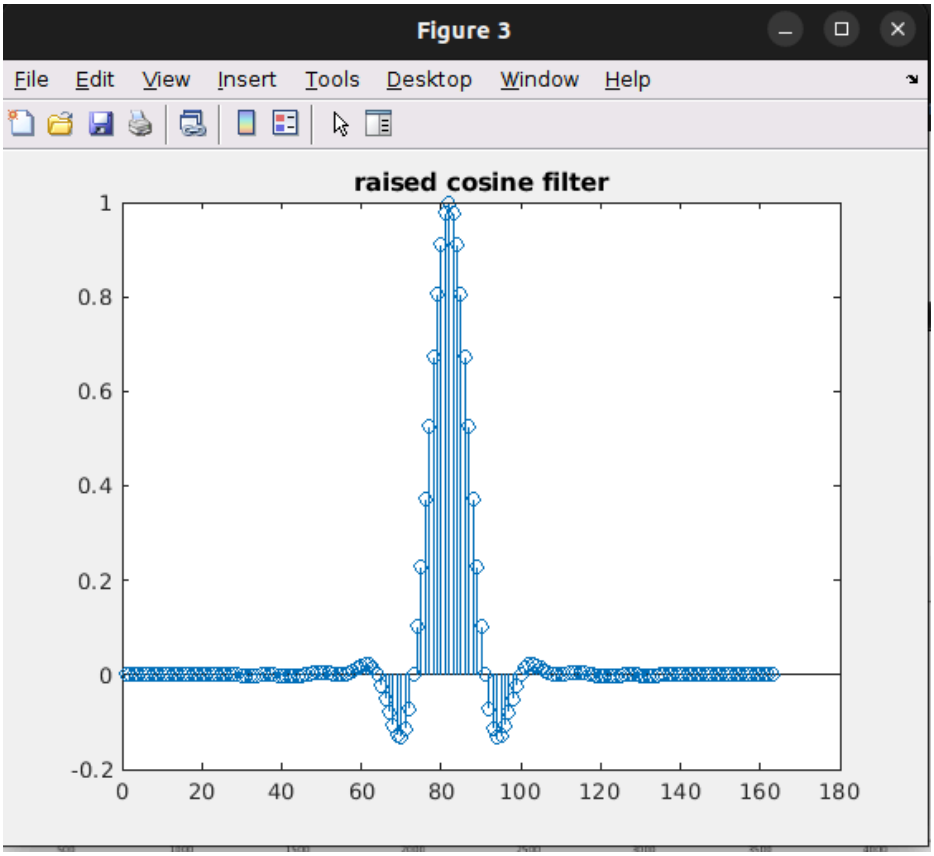


Figure 3
File Edit View Insert Tools Desktop Window Help
raised cosine filter
1
0.8
0.6
0.4
0.2
0
-0.2
0 20 40 60 80 100 120 140 160 180

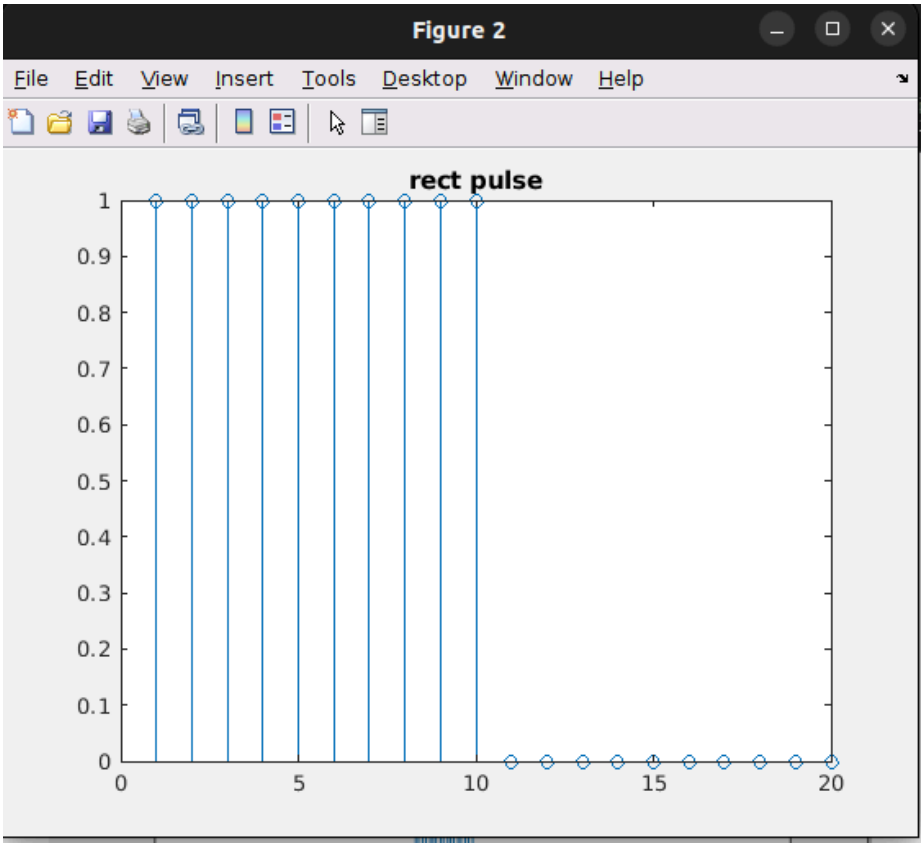
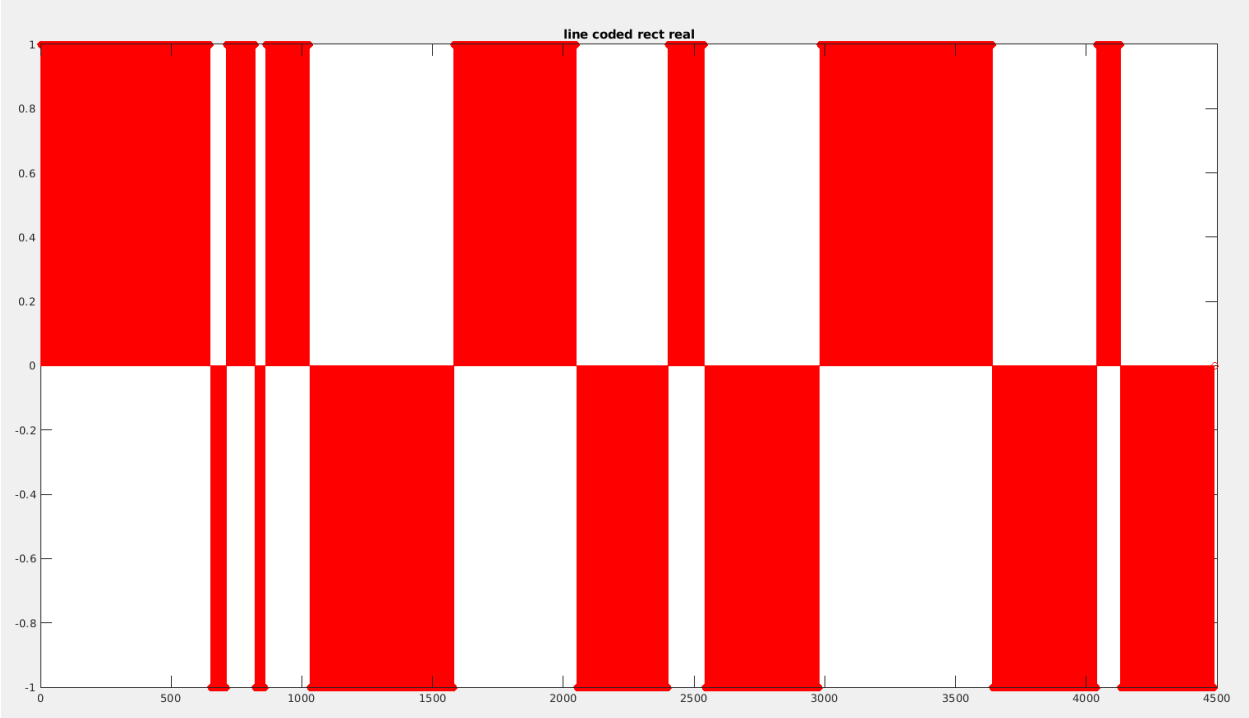
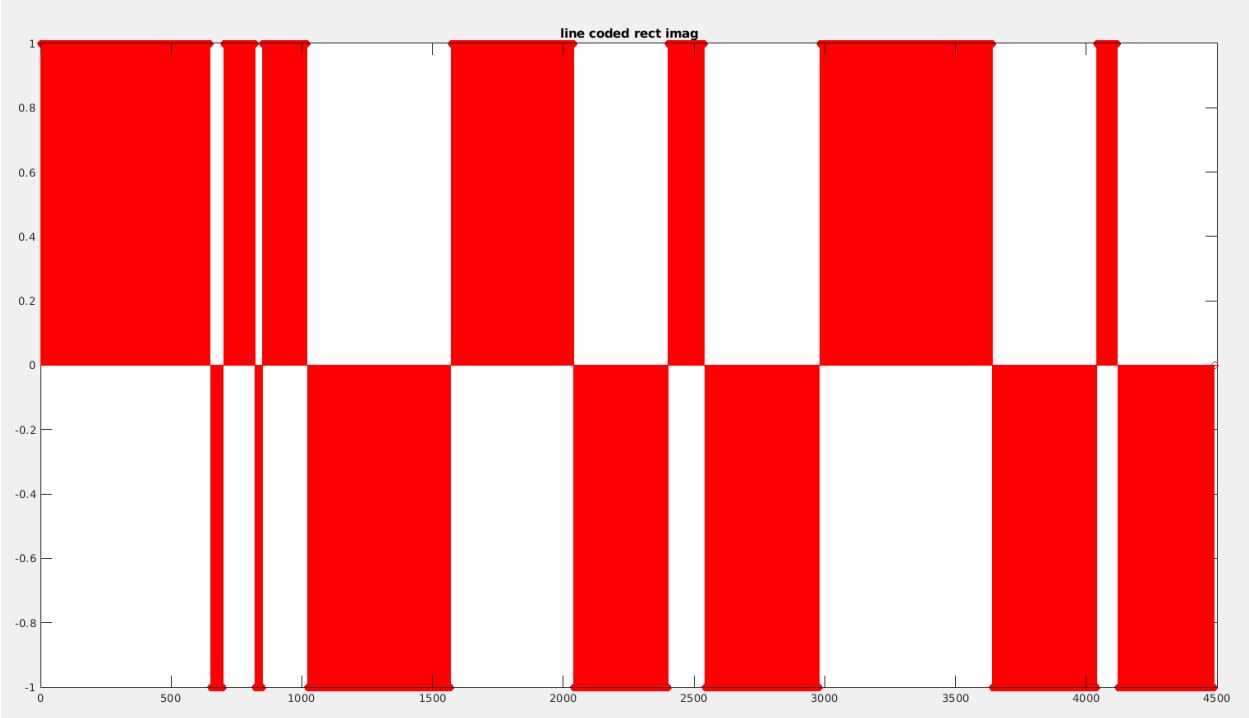
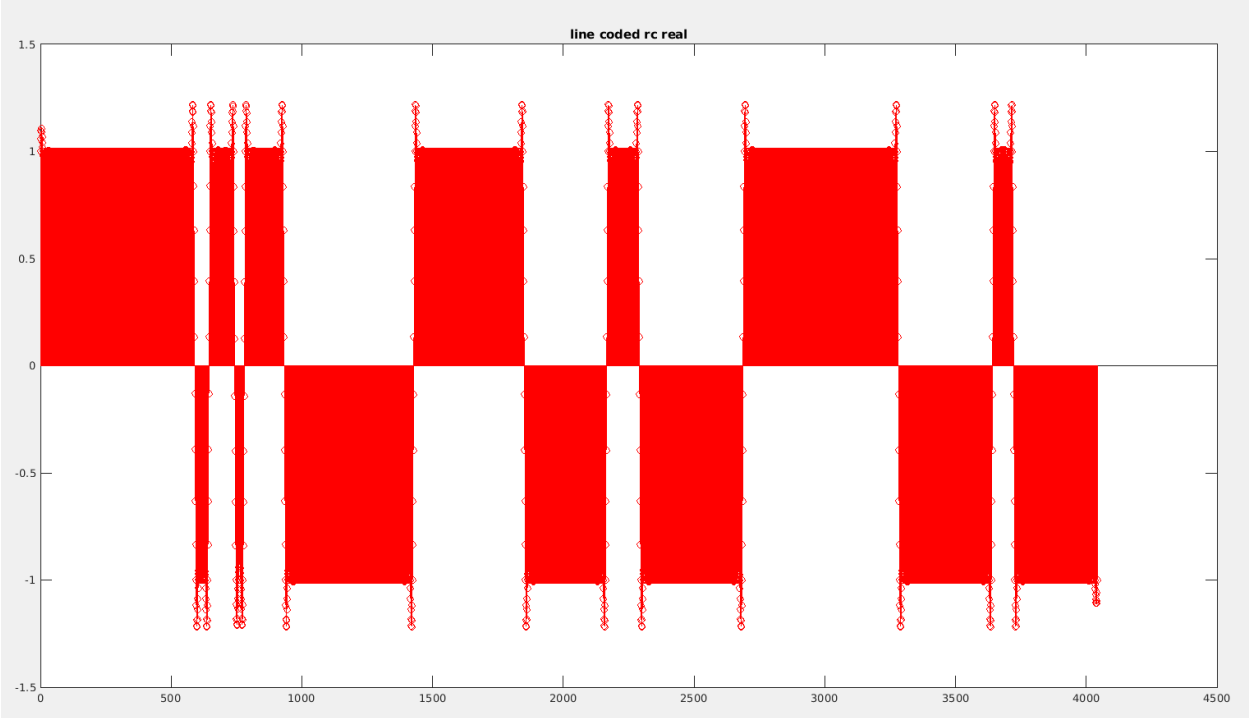
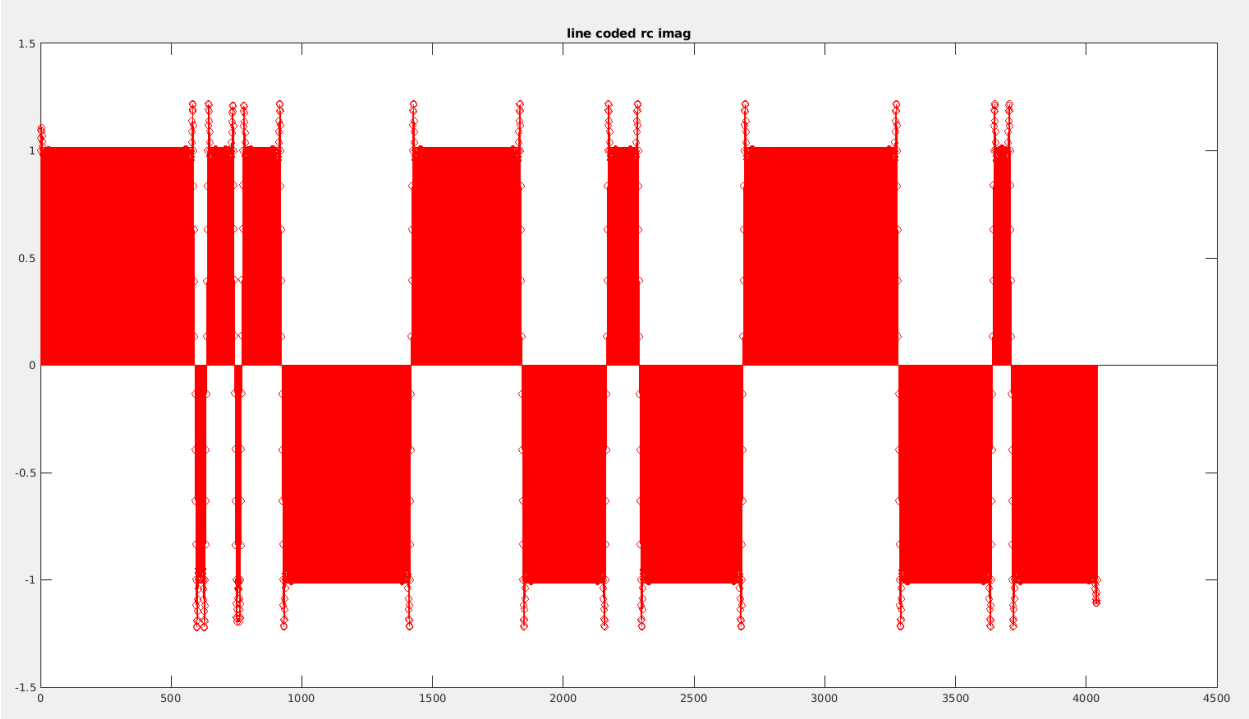


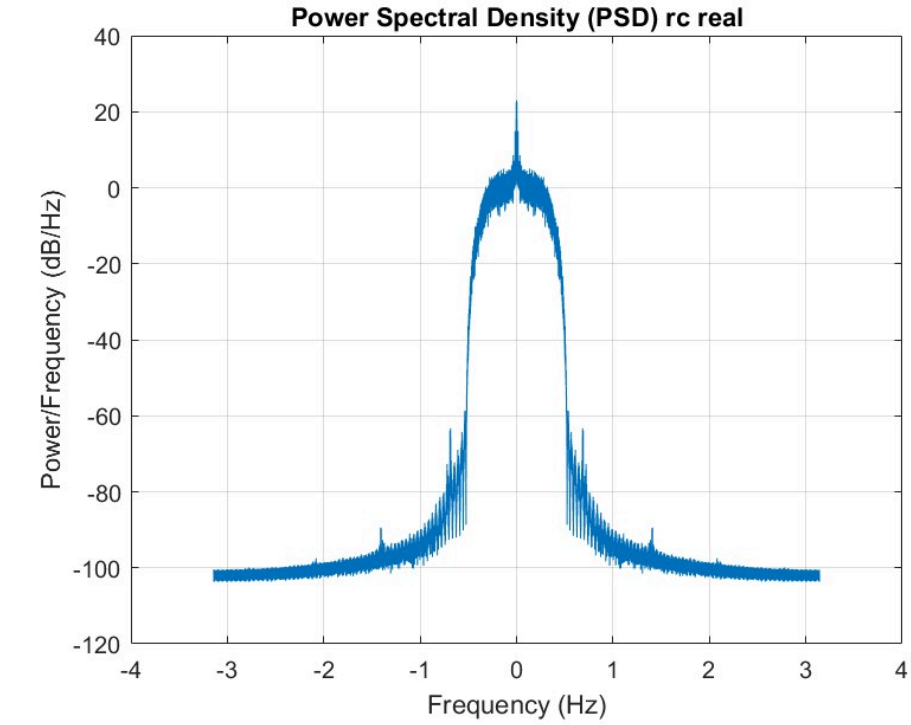
Figure 2
File Edit View Insert Tools Desktop Window Help
rect pulse
1
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1
0
0 5 10 15 20

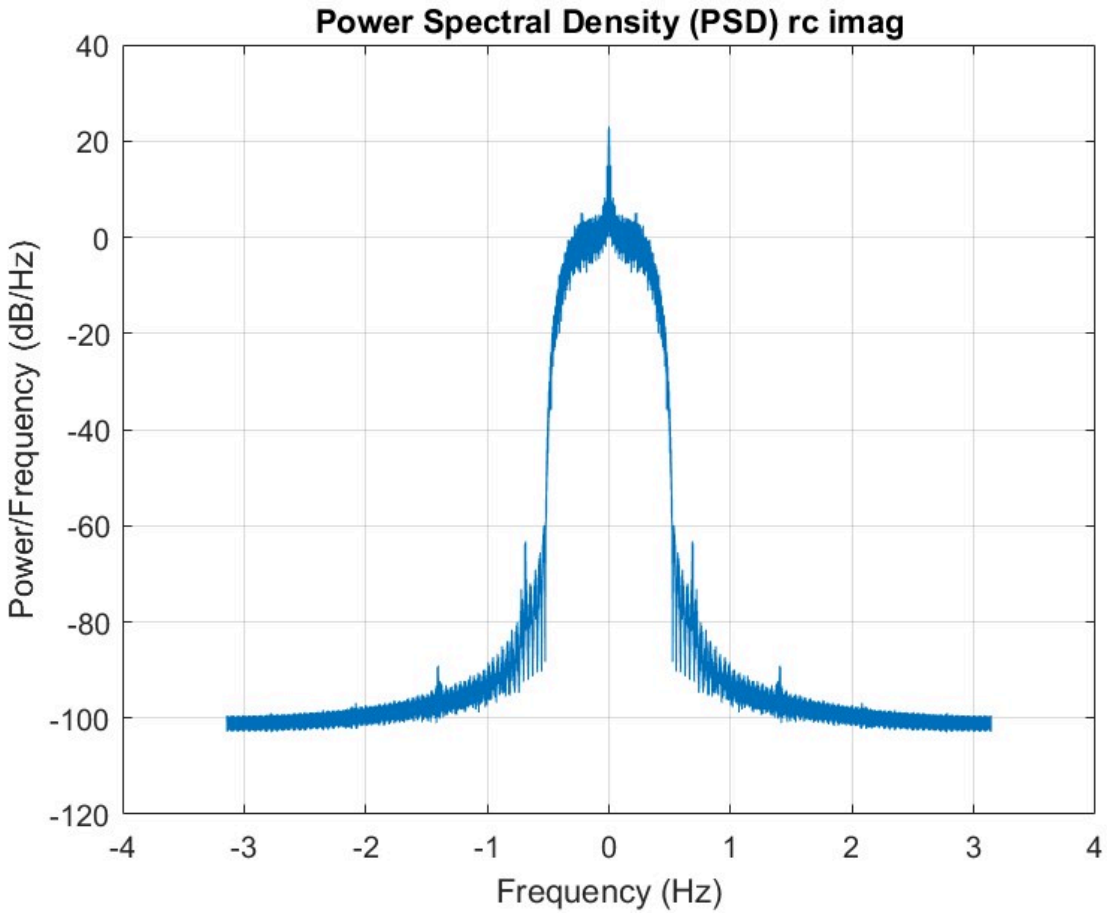

line coded rect real
1
0.8
0.6
0.4
0.2
0
-0.2
-0.4
-0.6
-0.8
-1
0
500
1000
1500
2000
2500
3000
3500
4000
4500

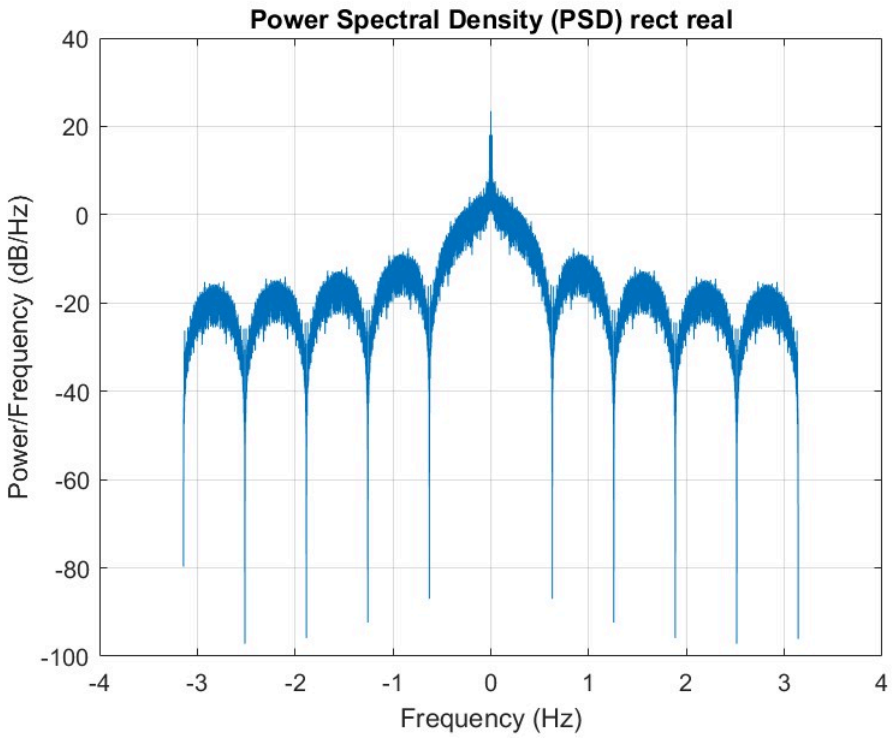

line coded rect imag
1
0.8
0.6
0.4
0.2
0
-0.2
-0.4
-0.6
-0.8
-1
0
500
1000
1500
2000
2500
3000
3500
4000
4500

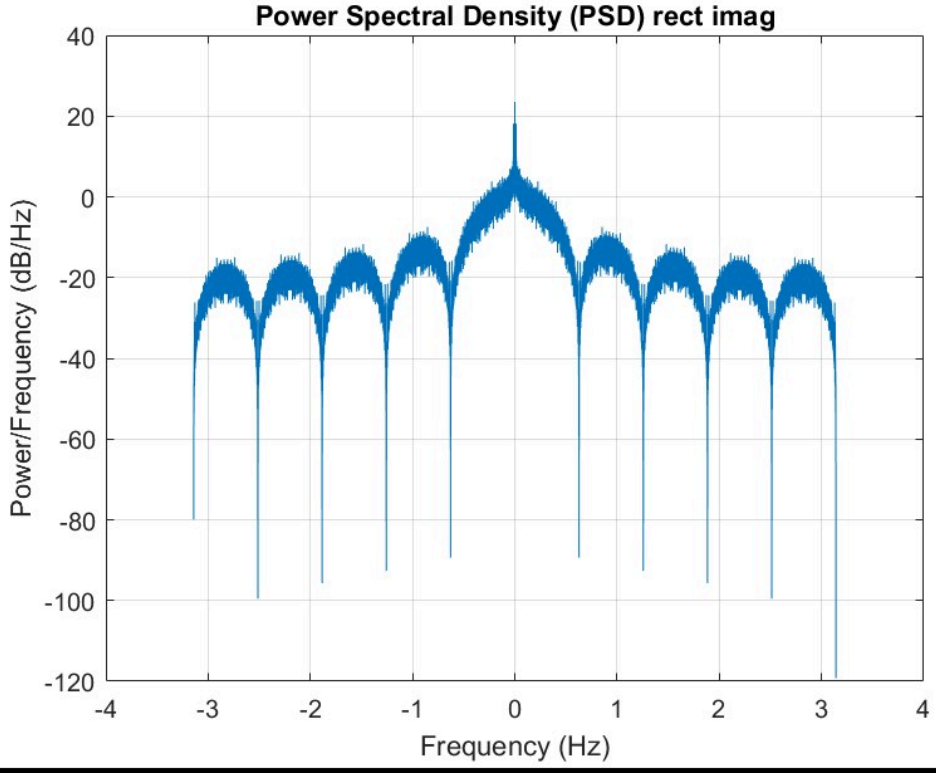

line coded rc real
1.5
1
0.5
0
-0.5
-1
-1.5
0
500
1000
1500
2000
2500
3000
3500
4000
4500


line coded rc imag
1.5
1
0.5
0
-0.5
-1
-1.5
0
500
1000
1500
2000
2500
3000
3500
4000
4500


Power Spectral Density (PSD) rc real
Power/Frequency (dB/Hz)
Frequency (Hz)
40
20
0
-20
-40
-60
-80
-100
-120
-4
-3
-2
-1
0
1
2
3
4


Power Spectral Density (PSD) rc imag
Power/Frequency (dB/Hz)
Frequency (Hz)
40
20
0
-20
-40
-60
-80
-100
-120
-4
-3
-2
-1
0
1
2
3
4


Power Spectral Density (PSD) rect real
Power/Frequency (dB/Hz)
Frequency (Hz)
40
20
0
-20
-40
-60
-80
-100
-4
-3
-2
-1
0
1
2
3
4

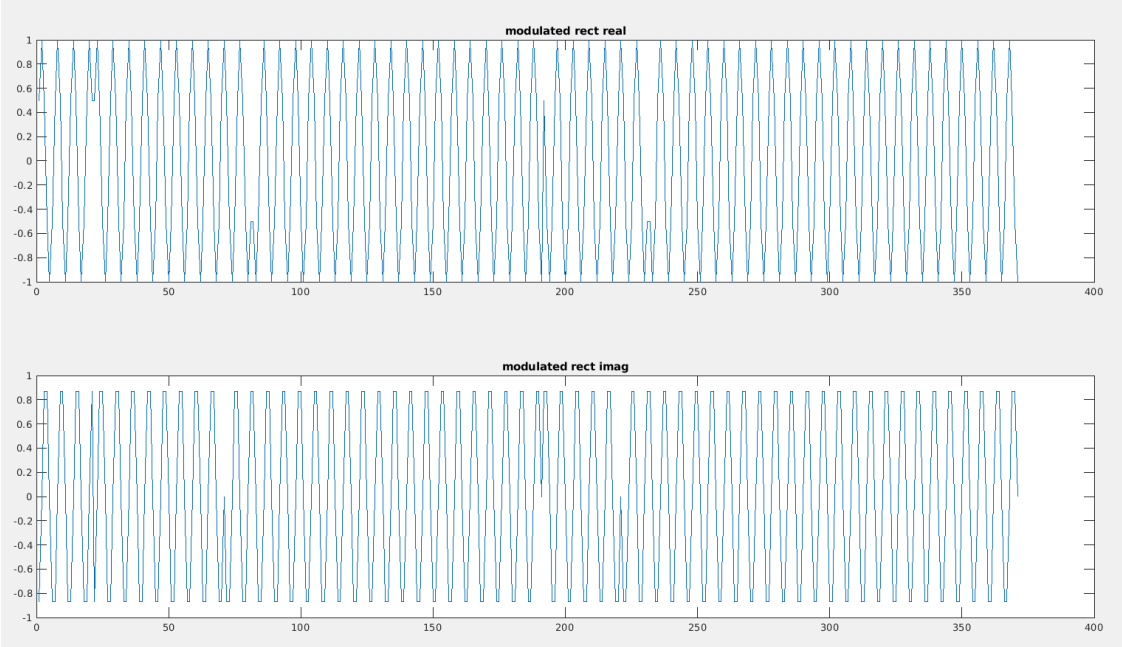

Power Spectral Density (PSD) rect imag
Power/Frequency (dB/Hz)
Frequency (Hz)
40
20
0
-20
-40
-60
-80
-100
-120
-4
-3
-2
-1
0
1
2
3
4

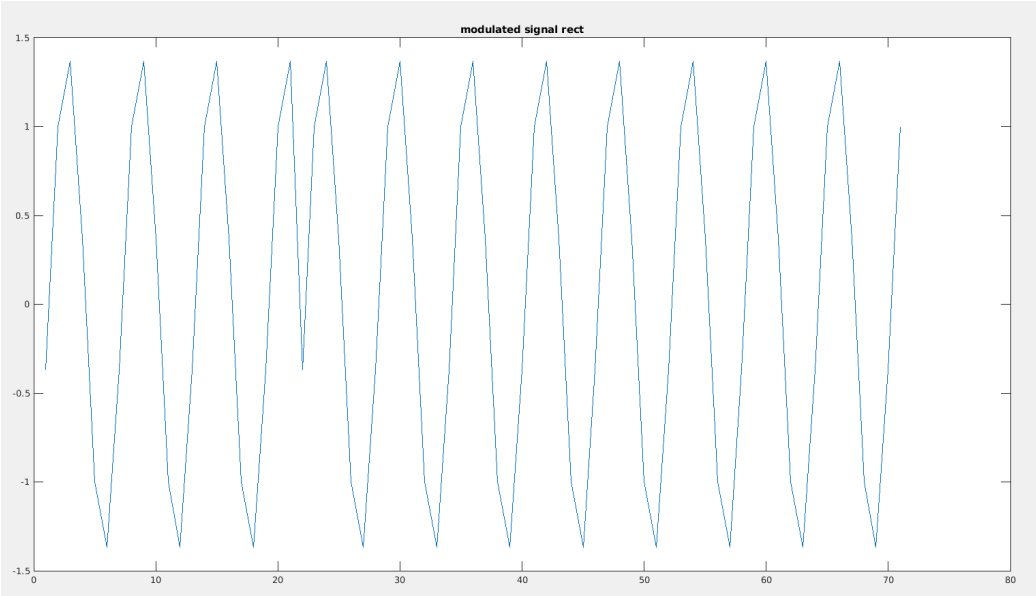
5. Modulation

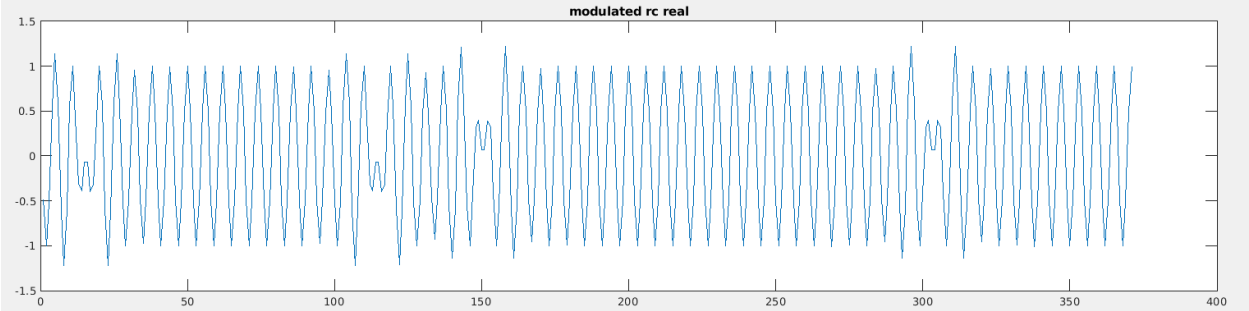
QPSK modulated signal is given by:

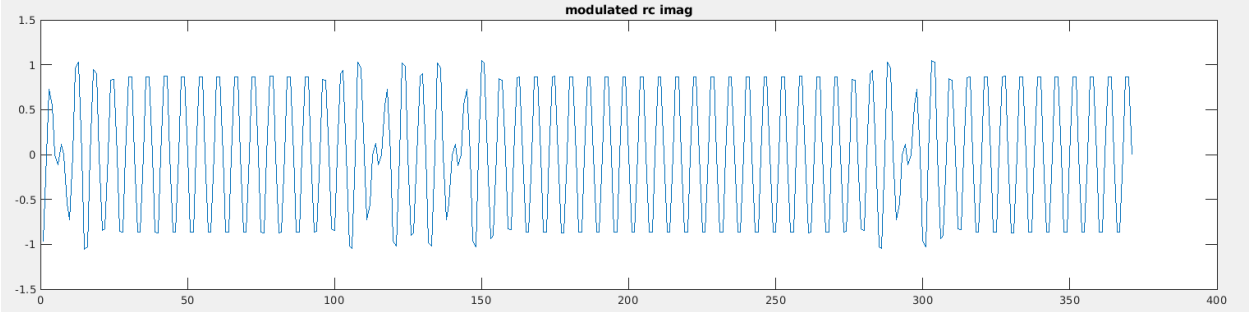
$$\phi_{PSK}(t) = a_m \sqrt{\frac{2}{T_b}} \cos \omega_c t + b_m \sqrt{\frac{2}{T_b}} \sin \omega_c t$$

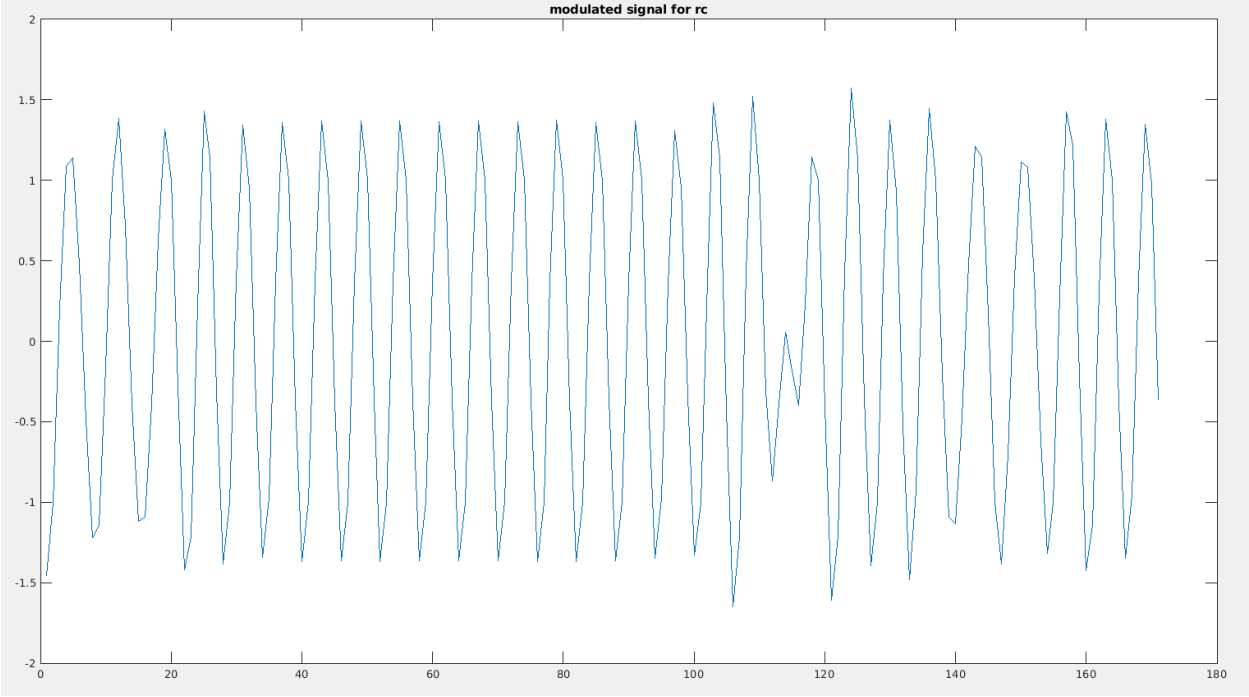
am are in-phase bits
bm are quadrature bits
Sent into modulator block from line coding block.






modulated rc real
1.5
1
0.5
0
-0.5
-1
-1.5
0
50
100
150
200
250
300
350
400


modulated rc imag
1.5
1
0.5
0
-0.5
-1
-1.5
0
50
100
150
200
250
300
350
400


modulated signal for rc
2
1.5
1
0.5
0
-0.5
-1
-1.5
-2
0
20
40
60
80
100
120
140
160
180

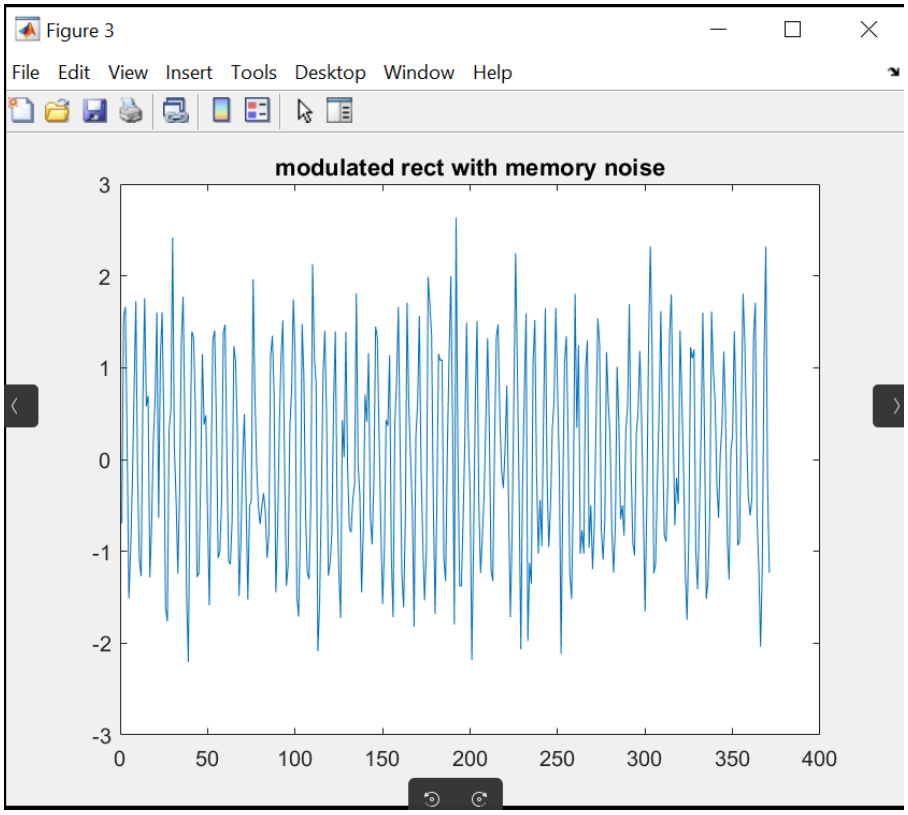


Figure 3
File Edit View Insert Tools Desktop Window Help
modulated rect with memory noise
3
2
1
0
-1
-2
-3
0 50 100 150 200 250 300 350 400

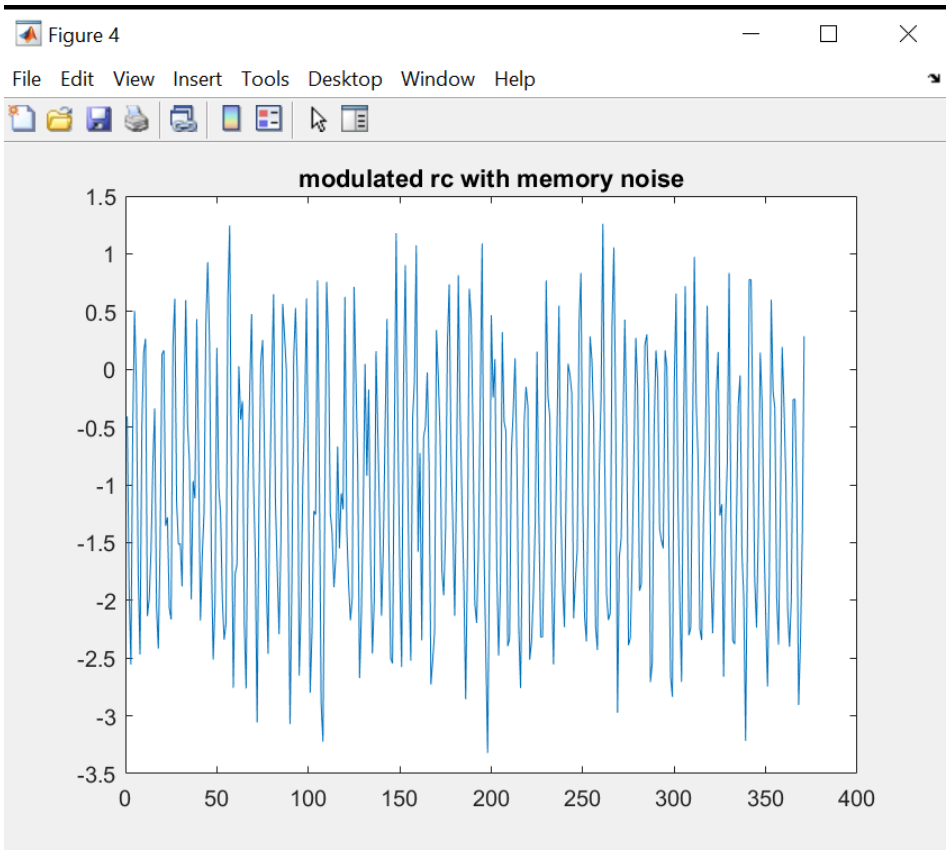
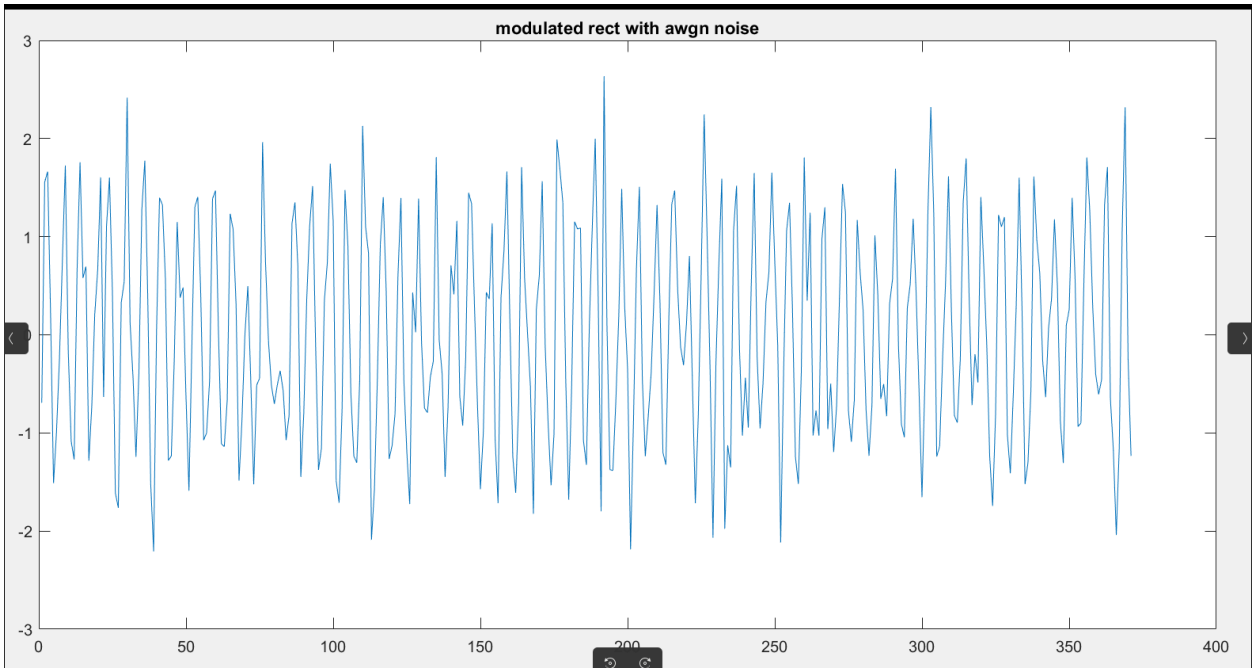
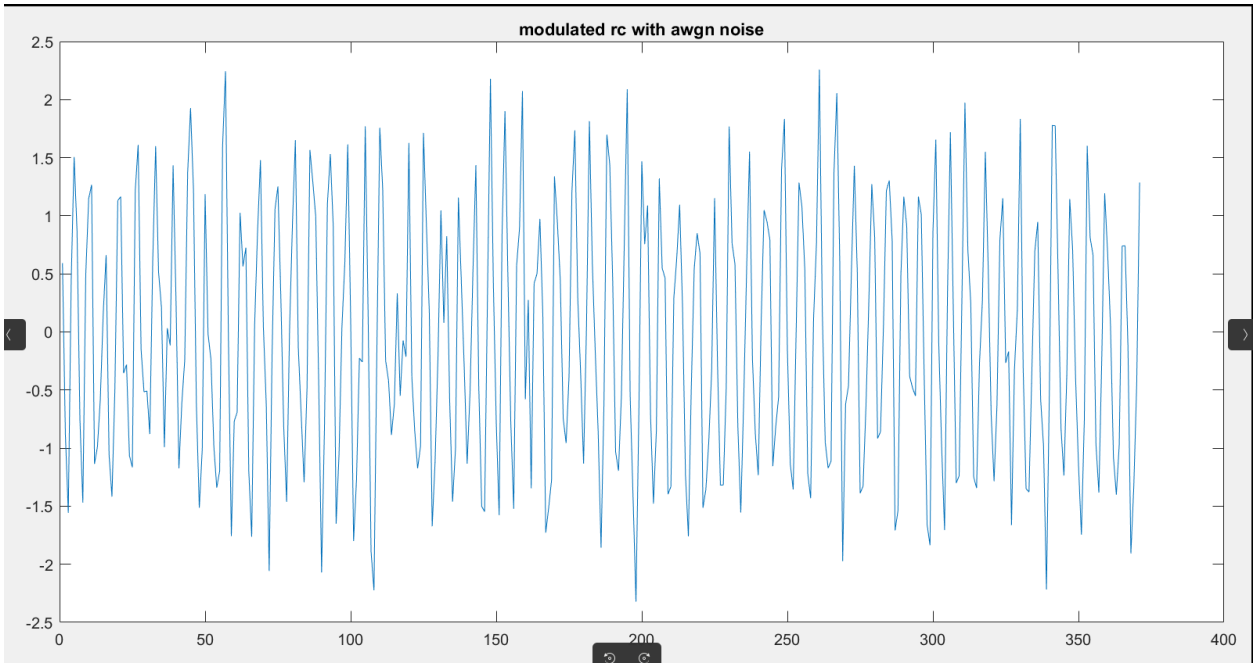
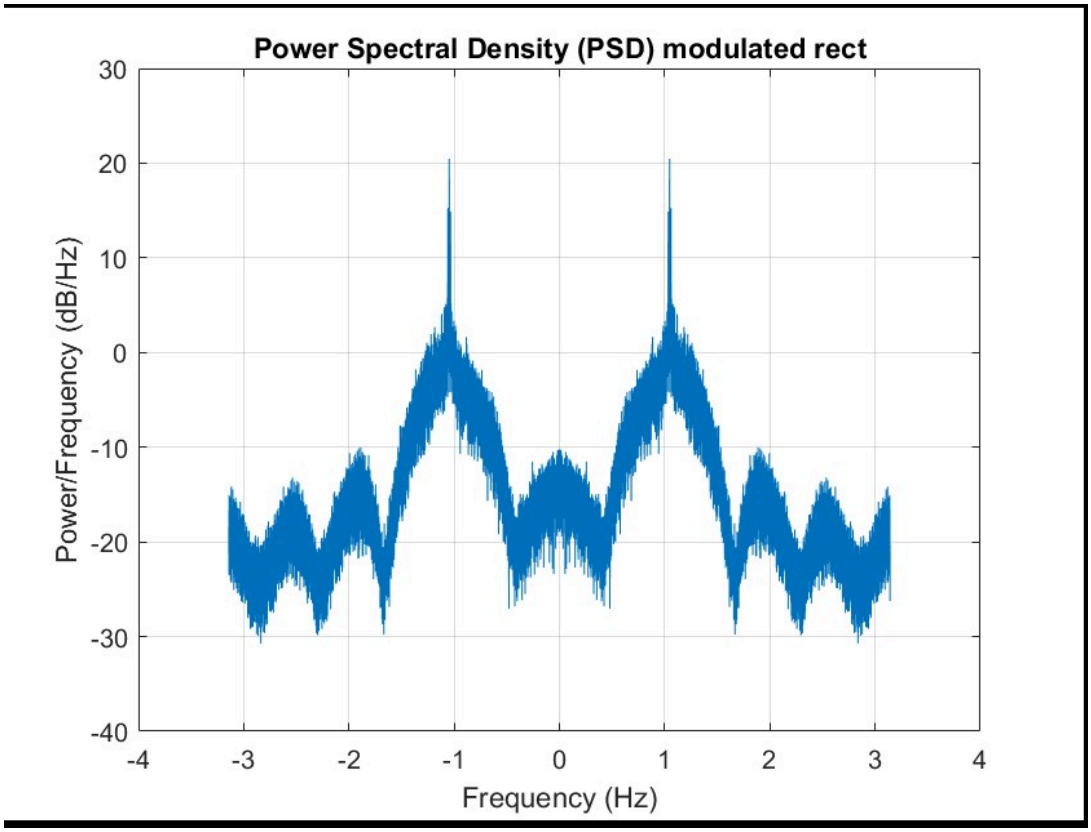
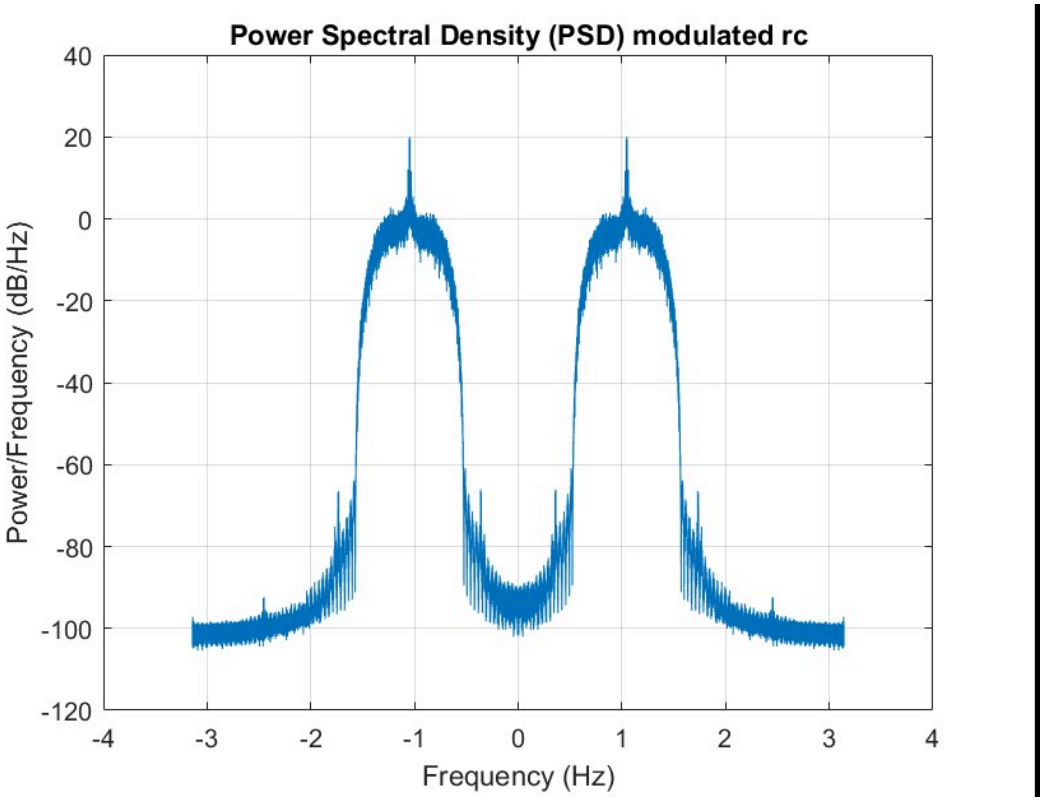


Figure 4
File Edit View Insert Tools Desktop Window Help
modulated rc with memory noise
1.5
1
0.5
0
-0.5
-1
-1.5
-2
-2.5
-3
-3.5
0 50 100 150 200 250 300 350 400


modulated rect with awgn noise
3
2
1
-1
-2
-3
0
50
100
150
200
250
300
350
400


modulated rc with awgn noise
2.5
2
1.5
1
0.5
0
-0.5
-1
-1.5
-2
-2.5
0
50
100
150
200
250
300
350
400


Power Spectral Density (PSD) modulated rect
Power/Frequency (dB/Hz)
Frequency (Hz)
30
20
10
0
-10
-20
-30
-40
-4
-3
-2
-1
0
1
2
3
4


Power Spectral Density (PSD) modulated rc
Power/Frequency (dB/Hz)
Frequency (Hz)
40
20
0
-20
-40
-60
-80
-100
-120
-4
-3
-2
-1
0
1
2
3
4

6. Channel

After modulating the signal, we pass our signal through a memoryless channel and one with memory.

1. Memoryless AWGN channel

$$r(t) = s(t) + n(t)$$

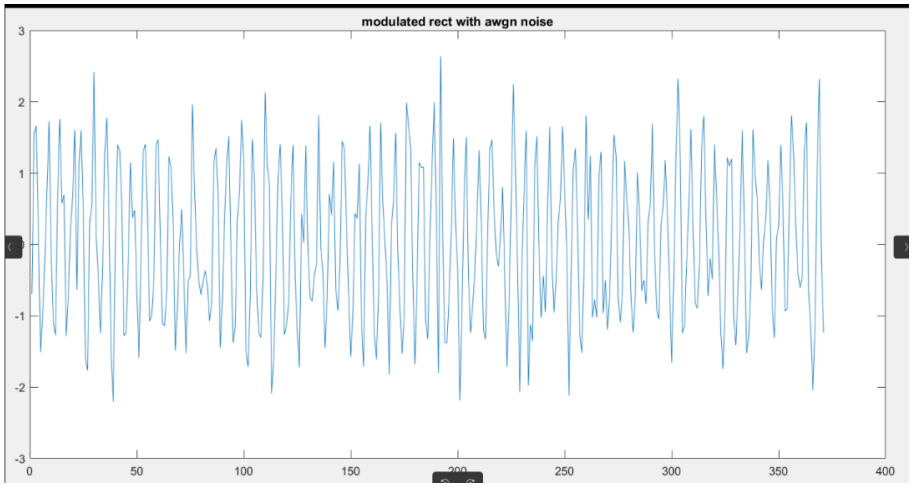
2. AWGN channel with memory

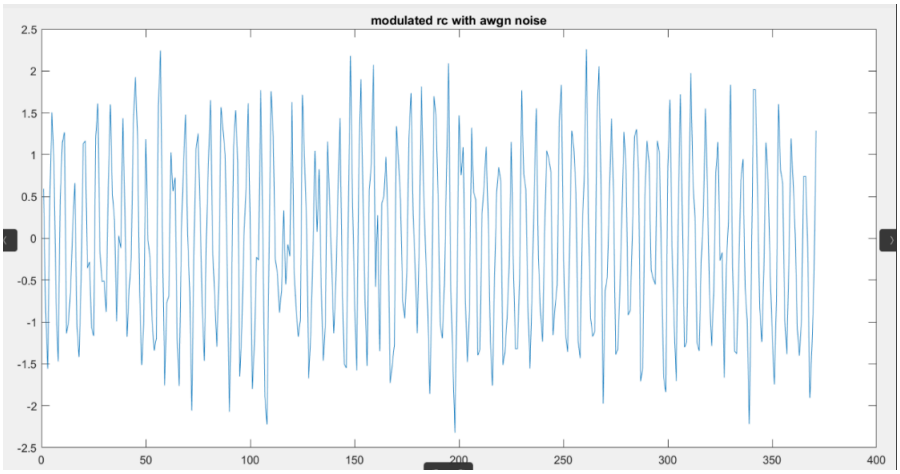
$$r(t) = h(t) * s(t) + n(t)$$

where, $h(t) = a\delta(t) + (1-a)\delta(t - bT_b)$

n(t) is the awgn noise that is added. To add noise, SNR=6 is taken as we infer from the Pe vs SNR plot that when SNR is 6, there is 0 Pe, hence reducing probability of error. Higher SNR values indicate better signal quality and vice versa. At SNR=20, we have almost perfect quality of signal.

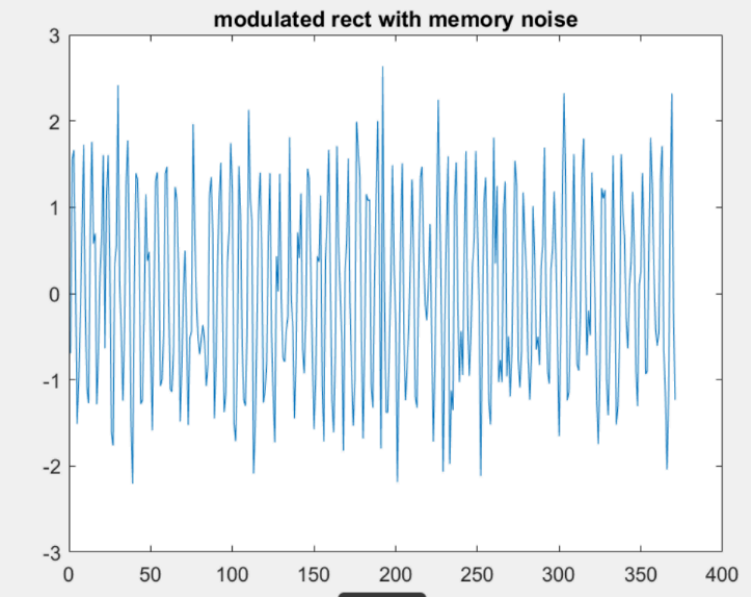
1. Memoryless:

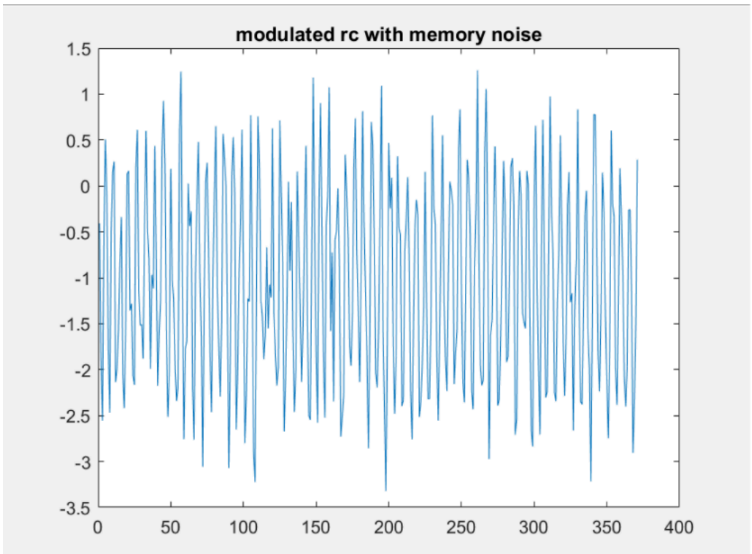




As we can see our modulated plots are distorted because of the noise. Each noise sample is independent and identically distributed (i.i.d.).

2. With Memory : a=0.6, b=1

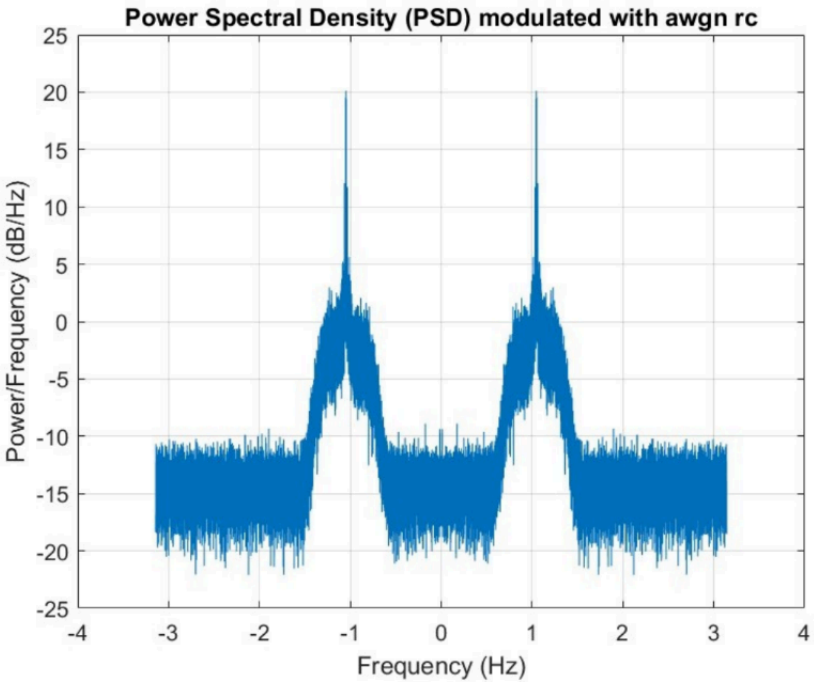




The memory influences the signal based on past values due to the convolution operation.
As we can see it incorporates memory effects or correlations in the noise, hence can provide a more accurate representation of the actual channel conditions. Real-world communication channels often exhibit memory effects due to phenomena such as multipath propagation, fading, or interference from neighboring signals.

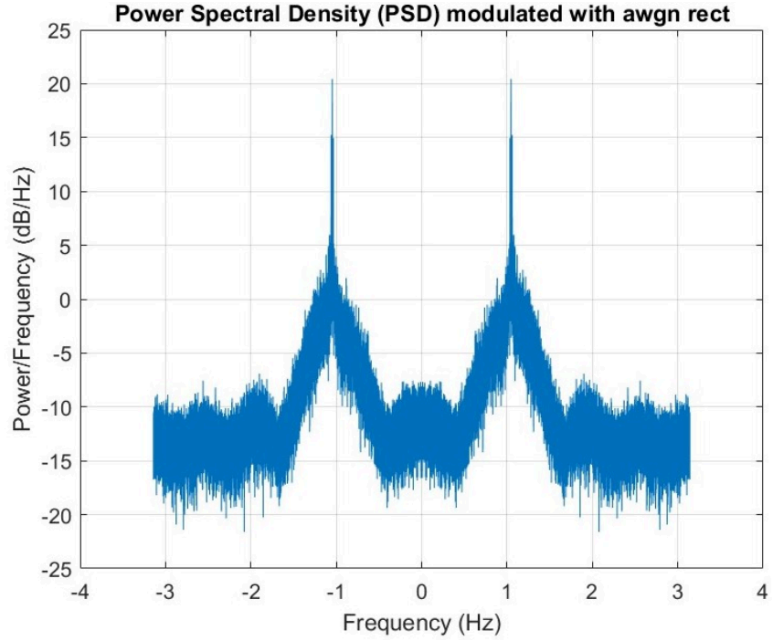
Memoryless channels generally tend to have lower Pe as compared to channels with Memory and require less complex receivers as compared to channels with memory.
Channels with memory, therefore, pose greater challenges in digital communication systems, demanding more sophisticated signal processing to maintain performance, unlike memoryless channels that are simpler to manage.

Below is the PSD of modulated signal with raised cosine filter and added AWGNnoise :



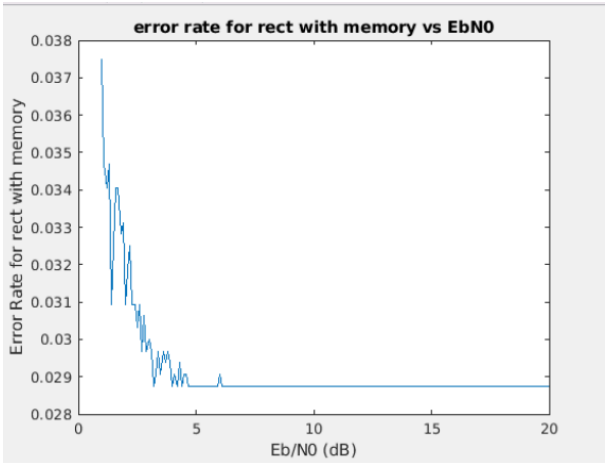
As AWGN has a flat spectrum, meaning it has equal power at all frequencies. Consequently, its PSD is constant across all frequencies. The overall PSD of the signal after adding AWGN will exhibit the same peaks as the modulated signal, but the noise floor will be elevated due to the presence of AWGN. Hence, we see flat sidebands.

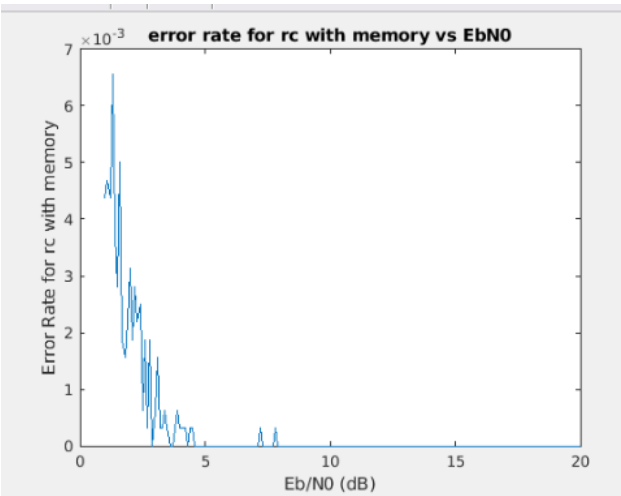
Below is the same using rect filter :



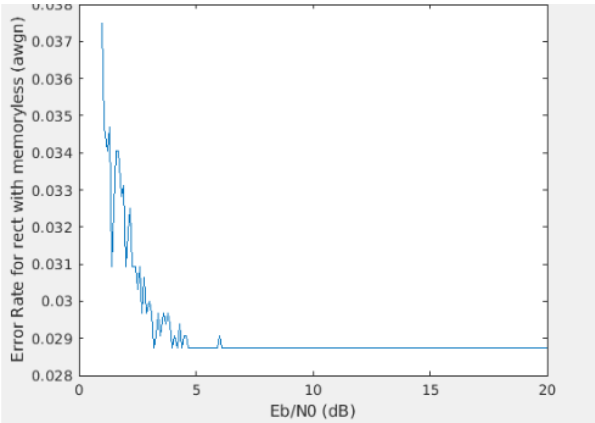
The PSD of the modulated signal with AWGN exhibits higher spectral leakage and potentially higher noise floor due to the sharp frequency response of the rectangular filter.

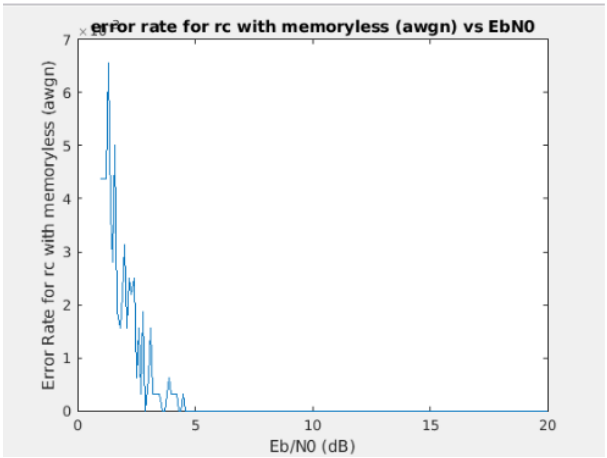
PE VS SNR AFTER NOISE : They don’t look as smooth as we took the first 50000 samples only but the plot will look like this only (it was taking us way too long to generate for the entire dataset). We notice that :





Error Rate for rect with memoryless vs Eb/N0

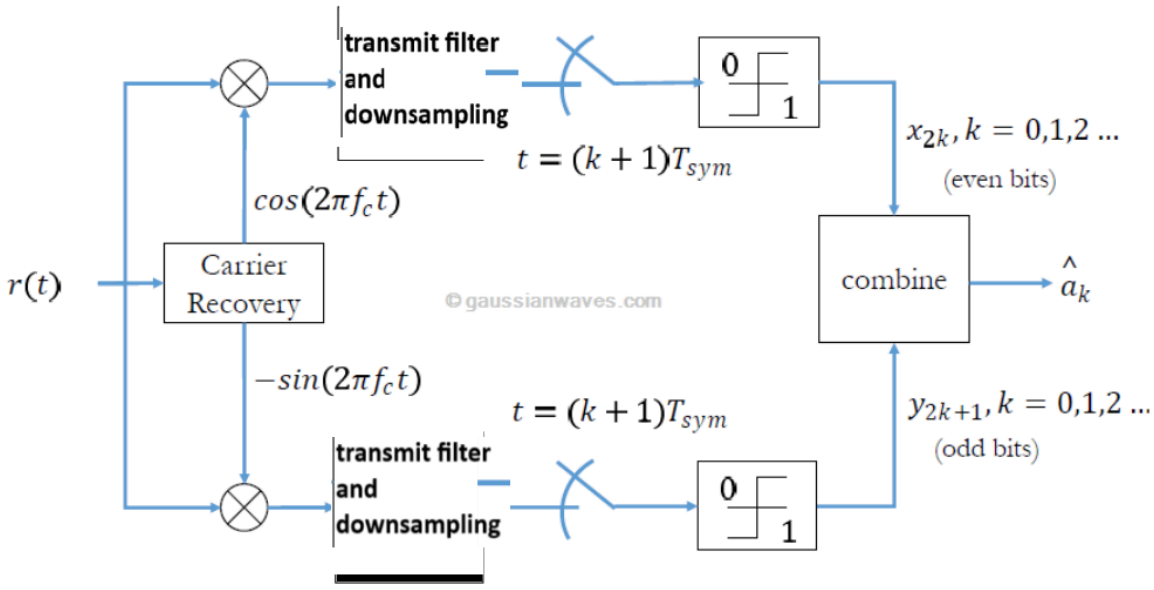




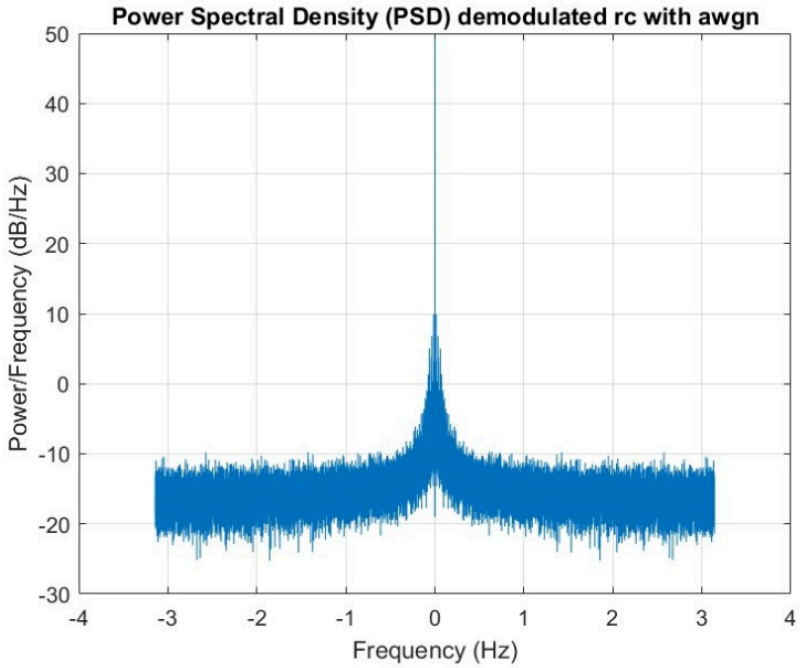
We notice that as SNR increases, the probability of error almost goes to zero. As RC is one of the best filters to combat ISI, we realize it nicely goes to 0 but rect doesn't go exactly zero but reduces significantly.

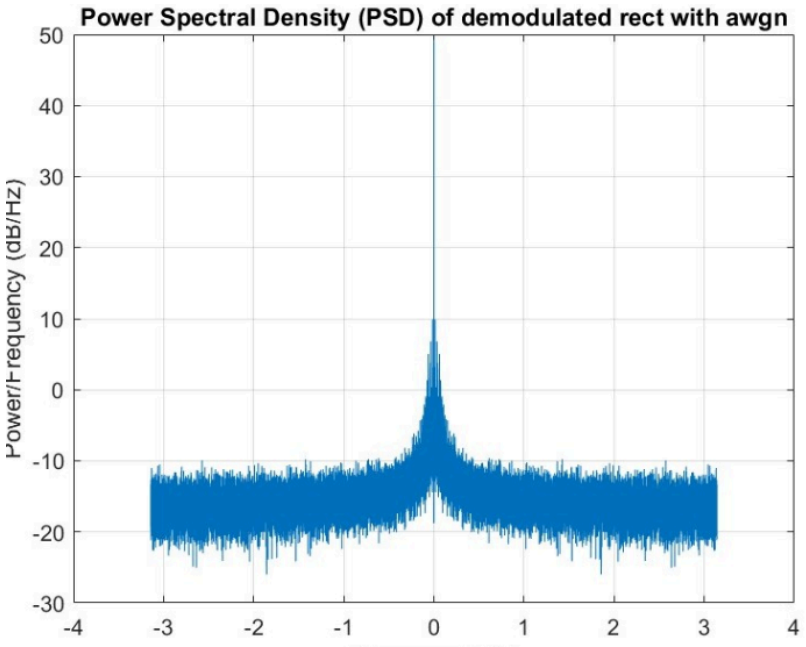
7. Demodulation :

For demodulation we multiplied the inphase received signal with the cos signal and the quadrature phase received signal with the sin signal to receive our signals back so that we get separate inphase and quad phase demodulated which can then be added to get final demodulation.



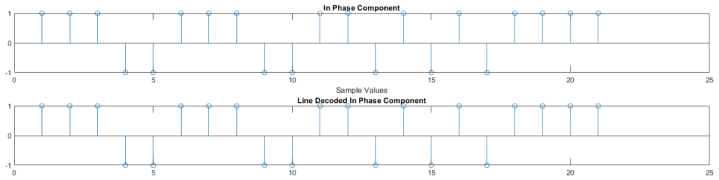
This block diagram explains how to get original bits back so demodulation, line decoding and decoder is all explained in this block.


Power Spectral Density (PSD) demodulated rc with awgn
Power/Frequency (dB/Hz)
Frequency (Hz)
50
40
30
20
10
0
-10
-20
-30
-4
-3
-2
-1
0
1
2
3
4

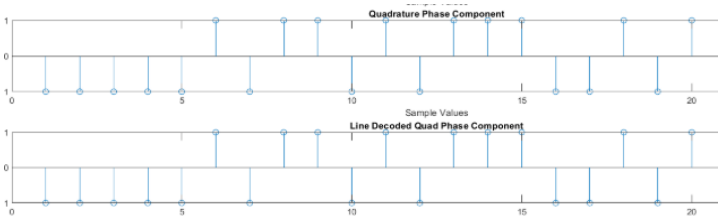

Power Spectral Density (PSD) of demodulated rect with awgn
Power/Frequency (dB/Hz)
50
40
30
20
10
0
-10
-20
-30
-4
-3
-2
-1
0
1
2
3
4

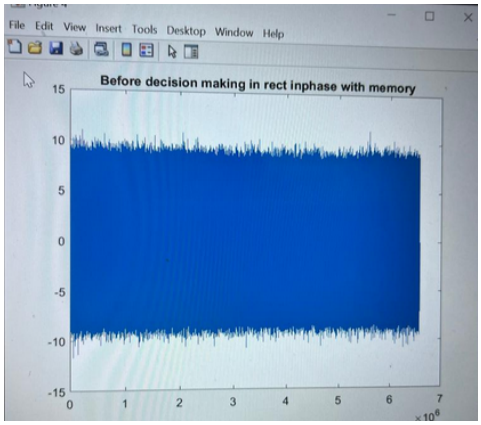
8. Line Decoding :

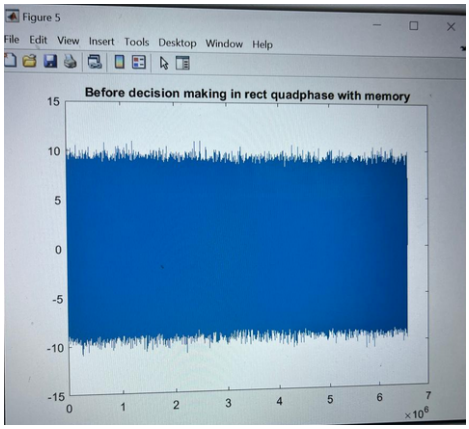
For memory :



For memory:







9. Decoder

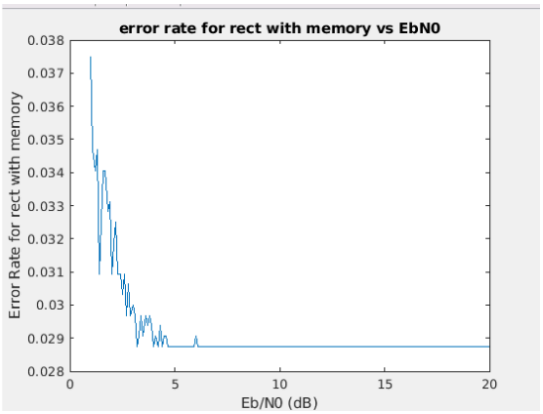
This section provides the ultimate decision statistics based on the in-phase and quadrature signals received from the line decoder. The mapping employed here is opposite to that of the encoder. Specifically, when both in-phase and quadrature bits equal 1, the decoded signal bits are determined to be (0,0).

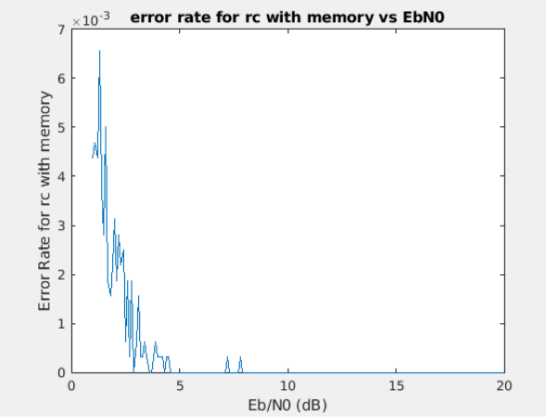
The constellation mapping after decoding is delineated as follows for both channel realizations.

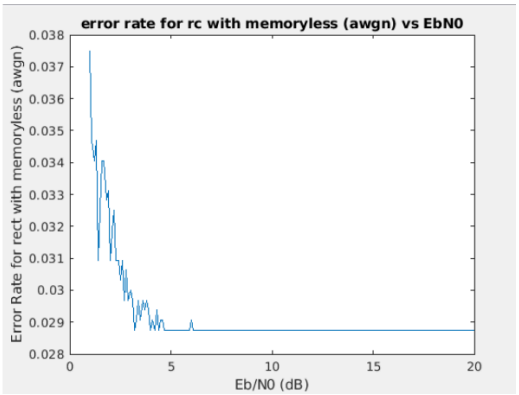
This is our decision making :

```
for z = 1:length(numeric_vector)/2
    if(downsample_rc_in(z)<0 && downsample_rc_qd(z)<0)
        Rx_in_rc = 1;
        Rx_qd_rc = 1;
    elseif(downsample_rc_in(z)<0 && downsample_rc_qd(z)>0)
        Rx_in_rc = 1;
        Rx_qd_rc = 0;
    elseif(downsample_rc_in(z)>0 && downsample_rc_qd(z)>0)
        Rx_in_rc = 0;
        Rx_qd_rc = 0;
    elseif(downsample_rc_in(z)>0 && downsample_rc_qd(z)<0)
        Rx_in_rc = 0;
        Rx_qd_rc = 1;
    end
    demodu_rc(k)=Rx_in_rc;
    demodu_rc(k+1) = Rx_qd_rc;
       k=k+2;
end
```

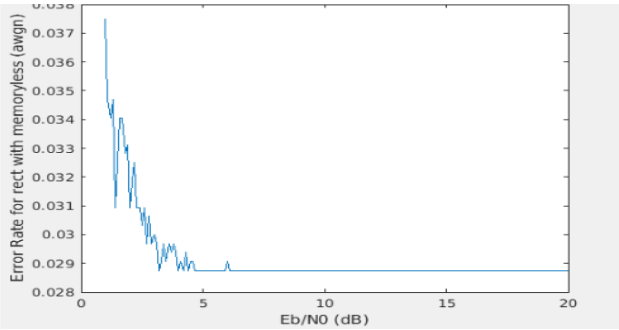
We are doing $\begin{matrix} \text{Decision} : p(t) \text{ if } r > 0 \\ -p(t) \text{ if } r < 0 \end{matrix}$ as QPSK has polar and for polar Ep=Eq and we know a = Ep-Eq/2 so a=0 always.



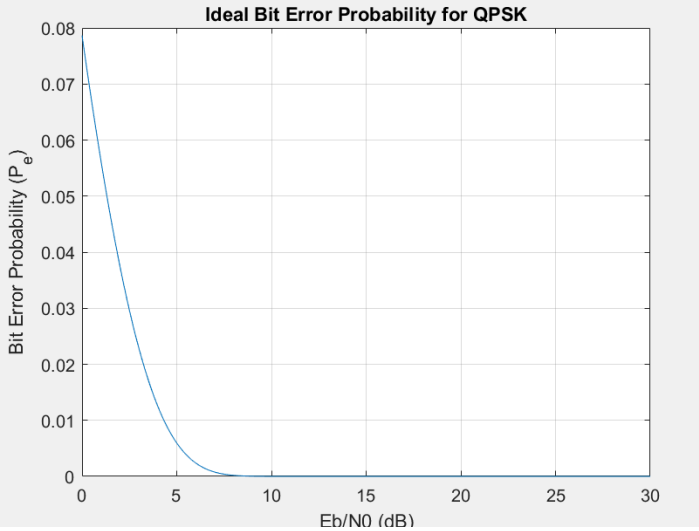




Error rate for rect with memoryless

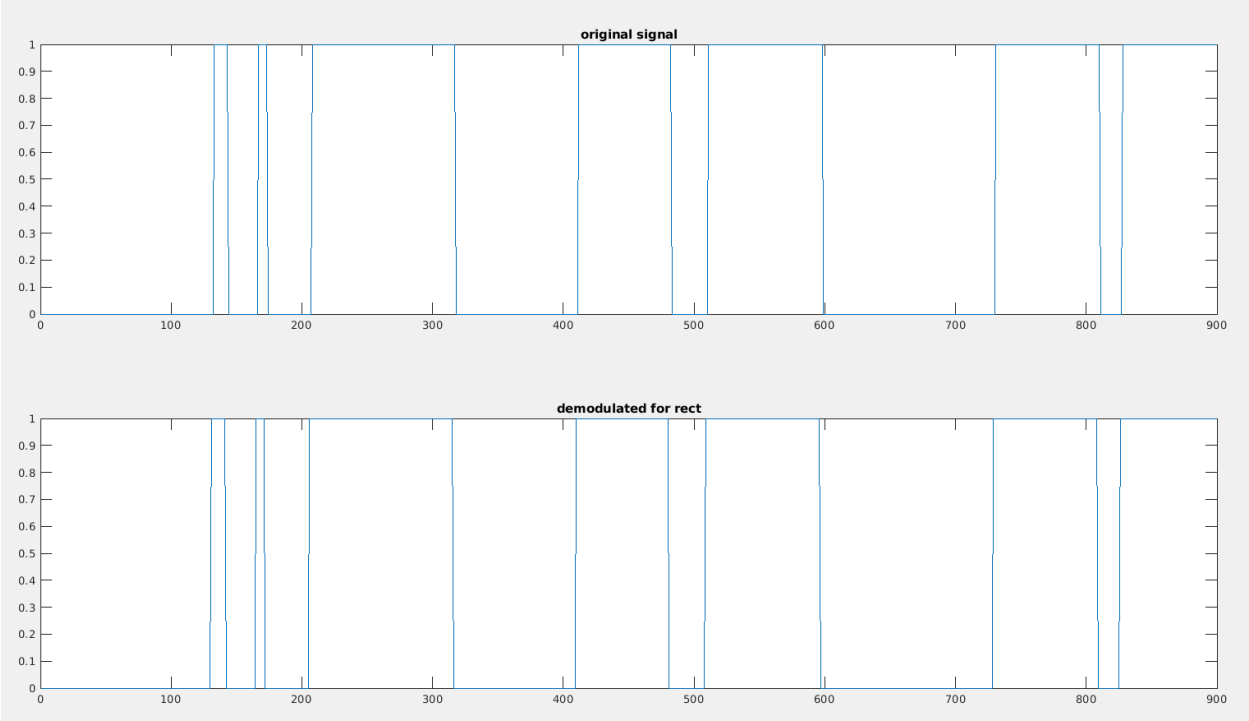


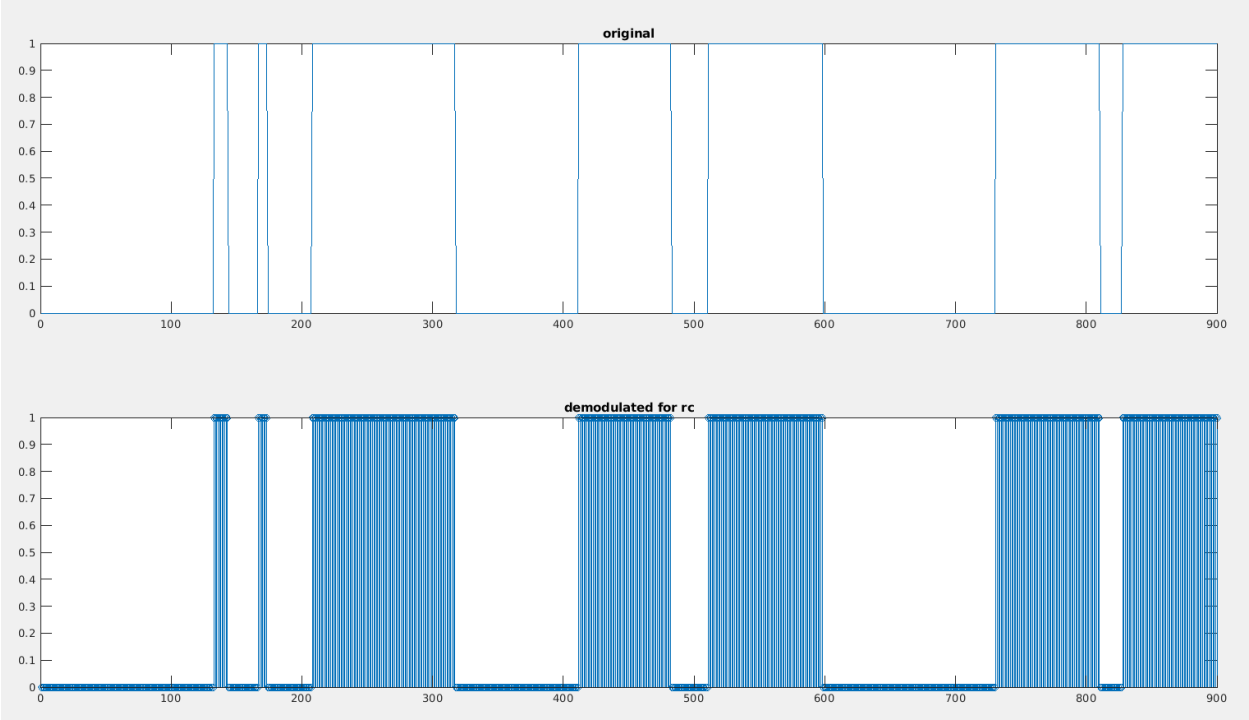
ANALYTICAL ERROR CALCULATION :



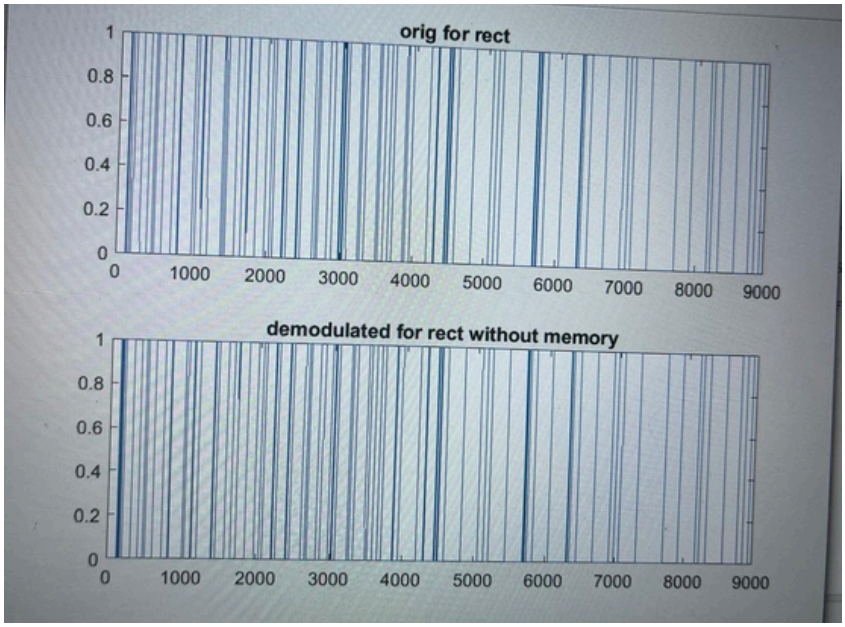
We can observe that the error is very small, which means our signal is well demodulated, increasing SNR makes it better.

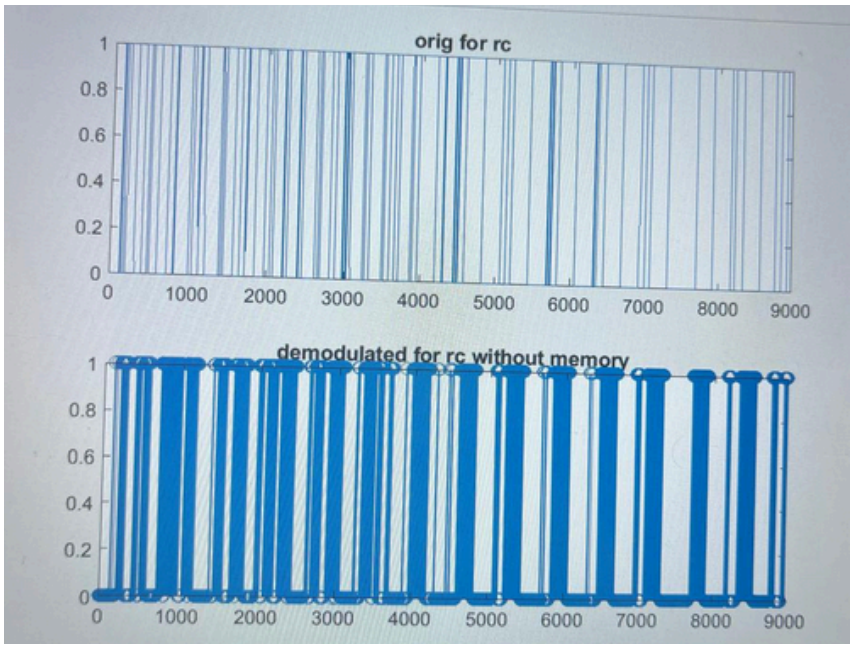
WITHOUT NOISE DEMODULATION:



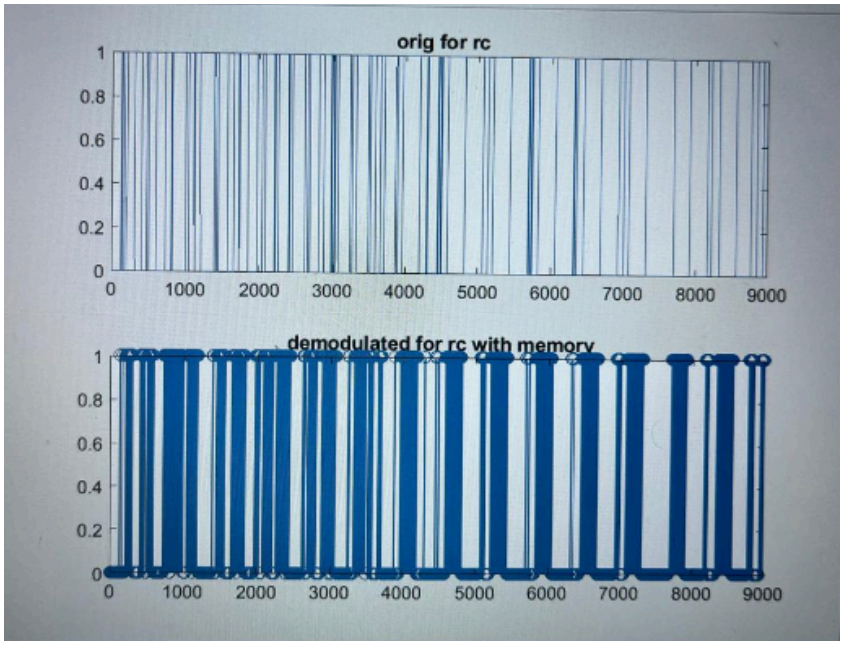


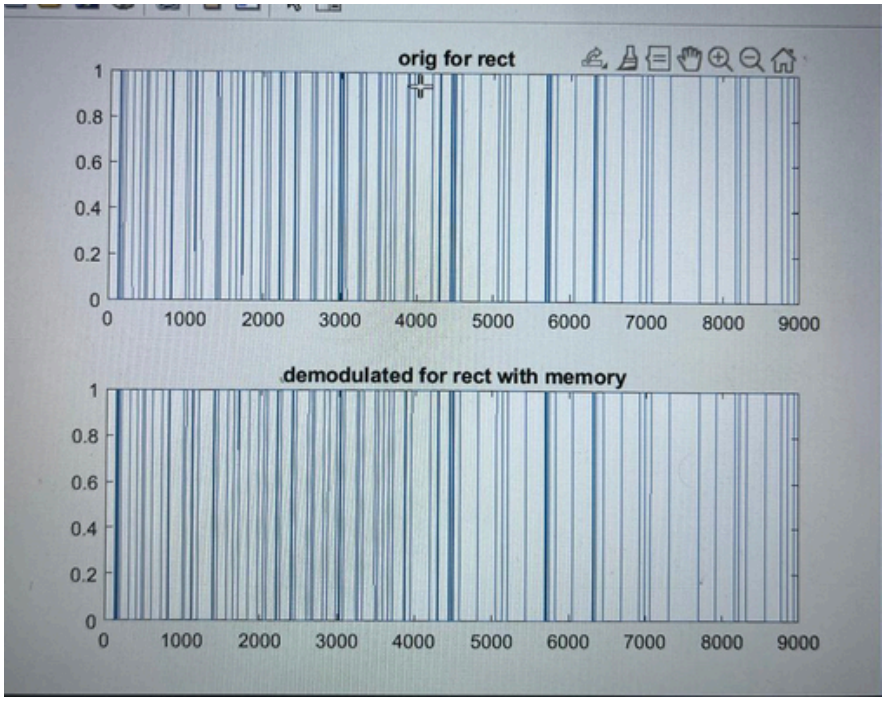
AWGN NOISE ADDED DEMODULATION:





MEMORY NOISE DEMODULATION:





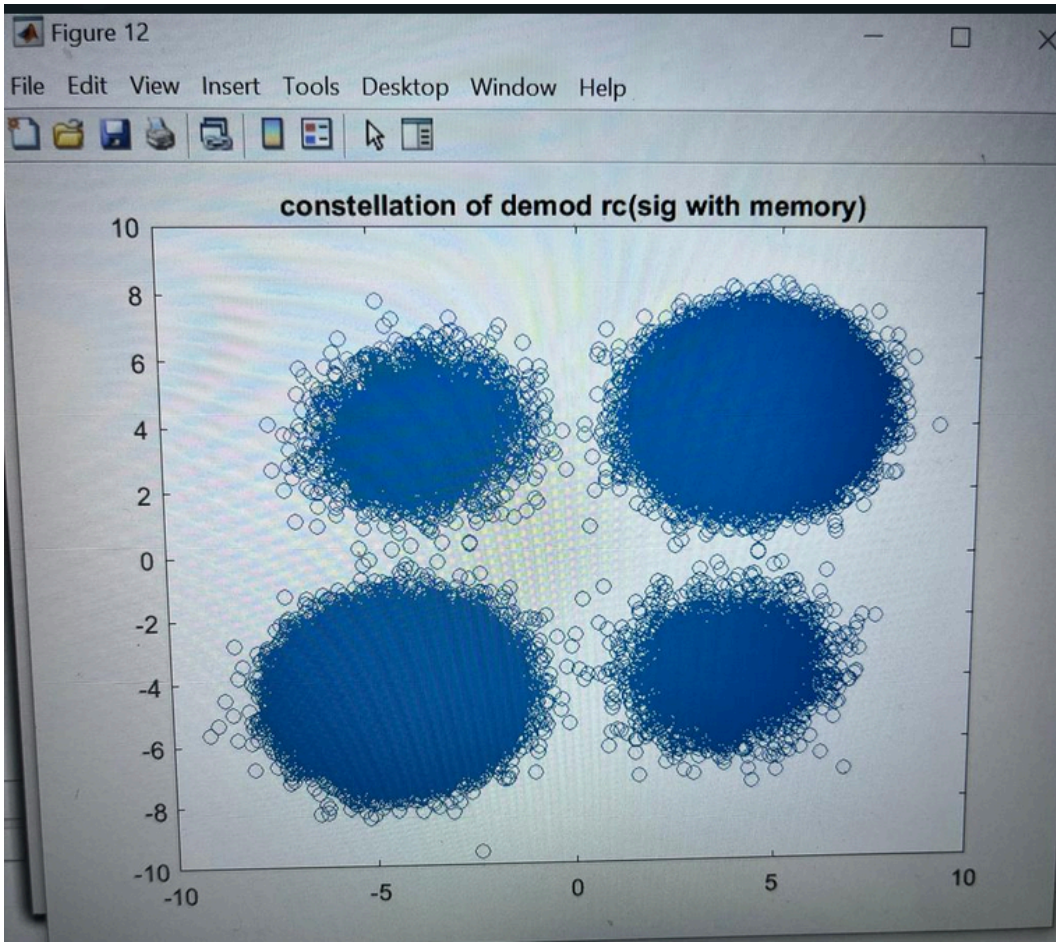


Figure 12
File Edit View Insert Tools Desktop Window Help
constellation of demod rc(sig with memory)
10
8
6
4
2
0
-2
-4
-6
-8
-10
-10
-5
0
5
10

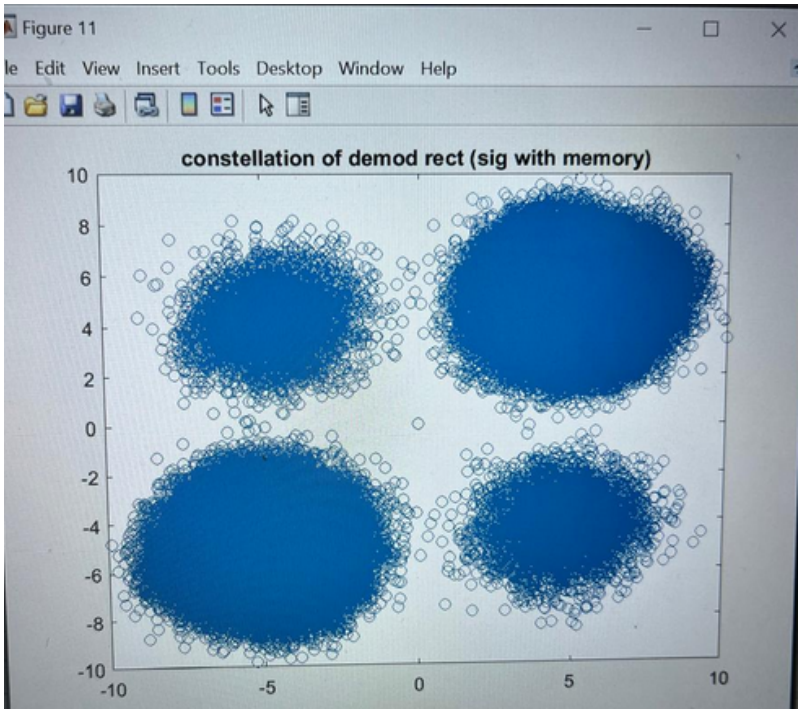


Figure 11
le Edit View Insert Tools Desktop Window Help
constellation of demod rect (sig with memory)
10
8
6
4
2
0
-2
-4
-6
-8
-10
-10
-5
0
5
10

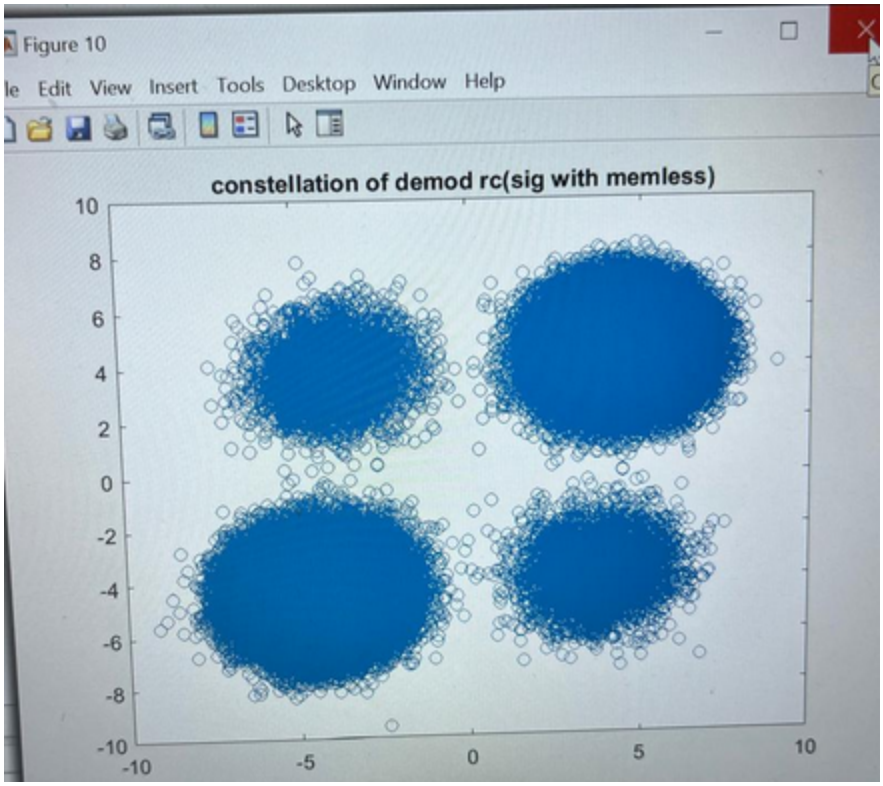


Figure 10
le Edit View Insert Tools Desktop Window Help
constellation of demod rc(sig with memless)
10
8
6
4
2
0
-2
-4
-6
-8
-10
-10
-5
0
5
10

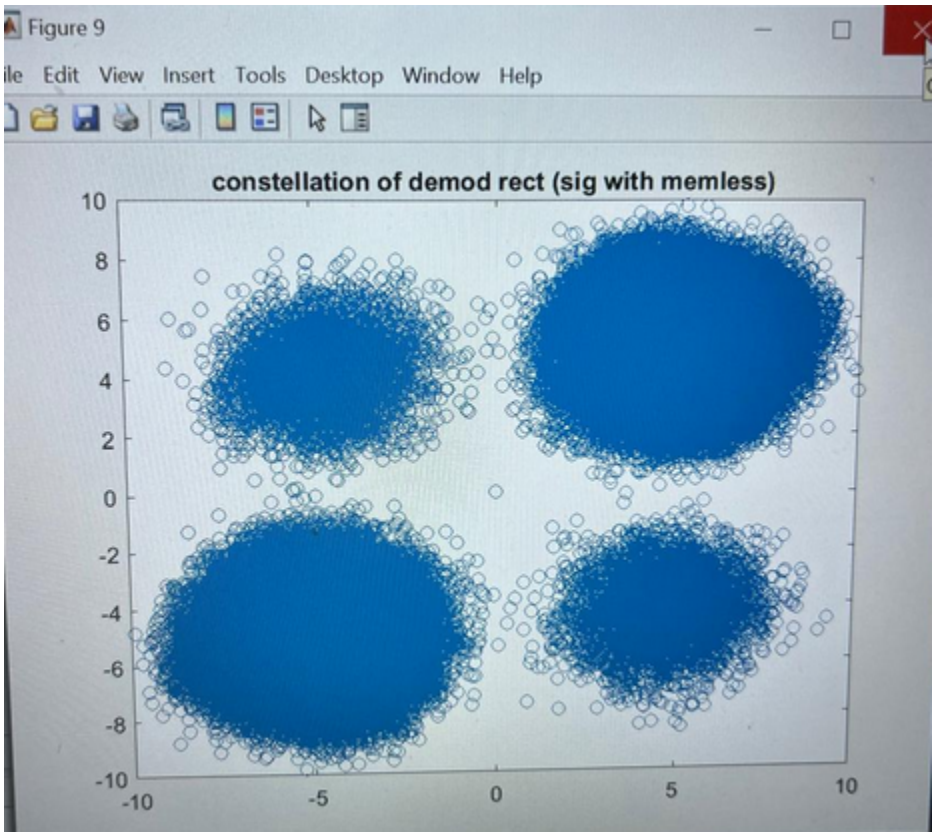


Figure 9
ile Edit View Insert Tools Desktop Window Help
constellation of demod rect (sig with memless)
10
8
6
4
2
0
-2
-4
-6
-8
-10
-10
-5
0
5
10

10. D/A Converter

1. **Convert Numeric Array to Character Array**:
    - `char_array_m = num2str(demodu_m_rc);`
        - `demodu_m_rc` likely represents the demodulated data obtained after decoding. This data is first converted from numeric values to a character array using `num2str`. Each numeric value is converted into its corresponding character representation.
2. **Remove Spaces**:
    - `char_array_m = strrep(char_array_m.', ' ', '');`
        - This step removes any spaces from the character array. It's common to represent binary data as strings with spaces in between each binary digit. This line removes these spaces.
3. **Reshape to Matrix**:
    - `binary_matrix_m = reshape(char_array_m.', [], 16);`
        - The character array is reshaped into a matrix format where each row represents a 16-bit binary number. This step assumes that each binary number is represented by 16 characters in the character array.
4. **Convert Binary to Decimal**:
    - `audio_integers_m = bin2dec(binary_matrix_m);`
        - Each row in the binary matrix is converted into its decimal equivalent. This converts the binary representation back into numeric values.
5. **Typecast to int16**:
    - `audio_reconstructed_m = typecast(uint16(audio_integers_m), 'int16');`
        - The decimal values are typecasted to 16-bit signed integers (`int16`). This step ensures that the data is represented in the appropriate format for audio signals.
6. **Normalize to [-1, 1]**:
    - `audio_reconstructed_normalized_m = double(audio_reconstructed_m) / 32767;`
        - The reconstructed audio data is normalized to the range [-1, 1] by dividing each sample by the maximum value representable by a 16-bit signed integer (32767).
7. **Play Reconstructed Audio**:
    - `sound(audio_reconstructed_normalized_m, fs);`

- Finally, the reconstructed audio signal is played using the `sound` function, with the sampling frequency `fs`.

Overall, this code block takes the decoded binary data, converts it back to audio samples, and plays the reconstructed audio signal.