In the name of God

Sharif University of Technology

Department of electrical engineering

Telecommunication Systems Lesson Exercise 4 MATLAB Simulation

Dr. Behroozi

Kasra Fallah

97109987

بهمن 99

سوال1)

1.1) I write the division and composition function as follows: Using the remainder divided by two, I do this easily. I put them in a row in the combine.

```
function [ a , b ] = Divide(x)
    for i = 1:length(x)
        if mod(i,2) == 0
            b (i/2) = x(i);
        end
        if mod(i,2) == 1
            a ((i+1)/2) = x(i);
        end
    end

end

function out = Combine(input1,input2)
out=[];
for i=1:1:length(input1)
    out = [out,input1(i),input2(i)];
end
end
```

We also execute the pulse material function easily by using condition and loop and stacking. (1.1

```
function out = PulseShaping( input , one , zero)
    out = [];
    for i = 1:length(input)
        if input(i) == 1 ;
            out = [out ,one];
        end

        if input(i) == 0 ;
          out = [out ,zero];
        end
    end
    end
```

We construct the modulation function by multiplying the signal by the cosine and the sine(1.2

```
function out = AnalogMod( input1 , input2 , Fs ,Fc)
    for i = 1: length(input1)
        out(i) = input1(i) * cos(  2*pi*Fc * i / Fs)...
            +input2(i) * sin(  2* pi* Fc * i / Fs);
    end
end
```

We construct the channel function, which, as mentioned, is an intermediate filter, with two ت(1.3 evils on the frequencies.

```
function out = Channel( input , Fs ,Fc ,Bw)
    num = length( input );
    fft_signal = fftshift (fft(input));
    F = -Fs/2 : Fs/num : Fs/2 - Fs/num;
    for i = 1:length(fft_signal)
```

```
        if abs(F(i)-Fc)<=Bw/2
            out_fft(i) = fft_signal(i);

        elseif abs(F(i)+Fc)<=Bw/2
            out_fft(i) = fft_signal(i);
        else
            out_fft(i) = 0;
        end
    end
    out  = ifft(ifftshift(out_fft));
end
```

We also do the modulation by multiplying by sine and cosine and filtering the bandwidth (1.4

```
function [x1,x2]=AnalogDemod(input,Fs,Bw,Fc)
t=1/Fs:1/Fs:(length(input)*1/Fs);
Cos_vec = cos(2*pi*Fc*t);
Sin_vec = sin(2*pi*Fc*t);
x1=2*input.*Cos_vec;
x2=2*input.*Sin_vec ;
X1=fftshift(fft(x1));
X2=fftshift(fft(x2));
F=linspace(0,Fs,length(X1))-Fs/2;
for i=1:1:length(X1)
   if(abs(F(i))>Bw)
       X1(i)=0;
       X2(i)=0;
   end
end
x1=ifft(ifftshift(X1));
x2=ifft(ifftshift(X2));
end
```

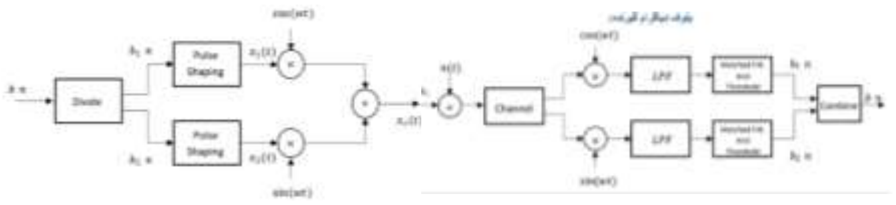
To do this, we need to go beyond the principles of wrist filtering by calculating the input (1.5
correlation with the signal and calculating it with half the energy of the sample signal. We
compare them to half the sample energy.

```
function [cor_one,cor_zero,out]=MatchedFilt(input,one,zero)   (1.6
psd_one = one.*one;
Energy=sum(psd_one);
cor_one=conv(input,one);
cor_zero=conv(input,zero);
out=zeros(1,length(input)/length(zero));
for i=1:length(out)
    if cor_one(i*length(zero))>=Energy/2
        out(i)=1;
    end
end
end
```

Part II

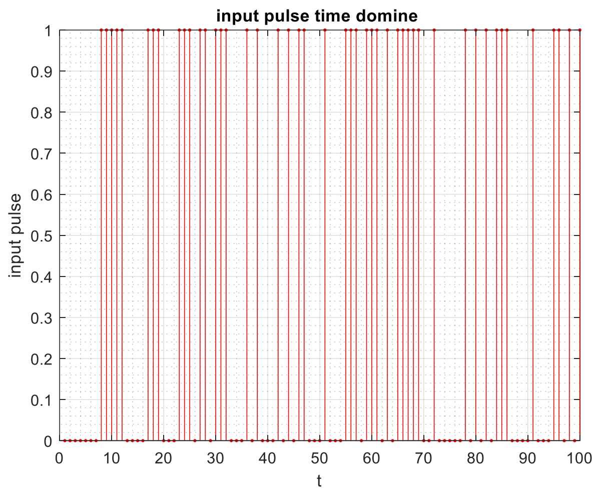
First part: no noise

First, we generate a random signal with the following method and channel it to the channel.

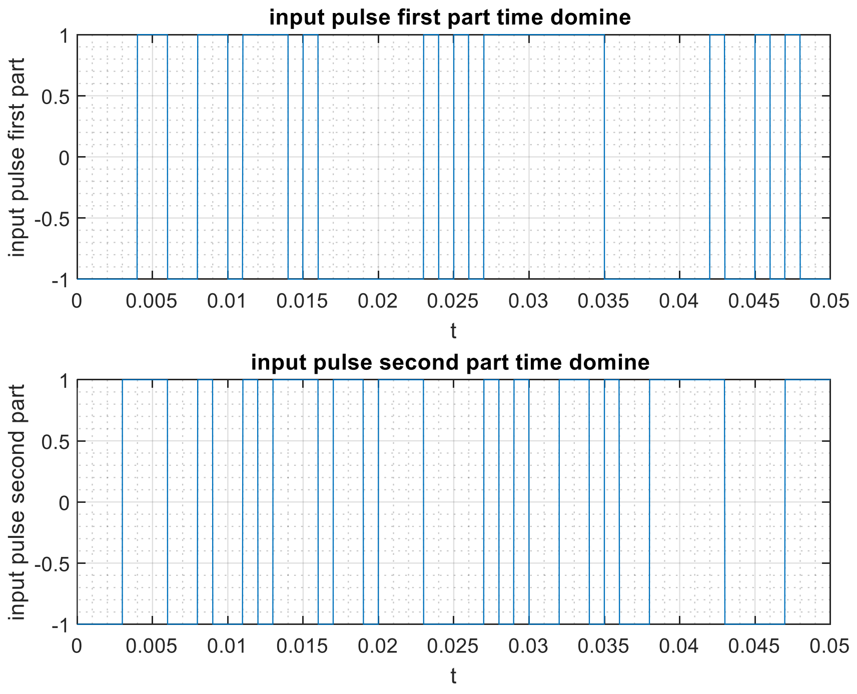


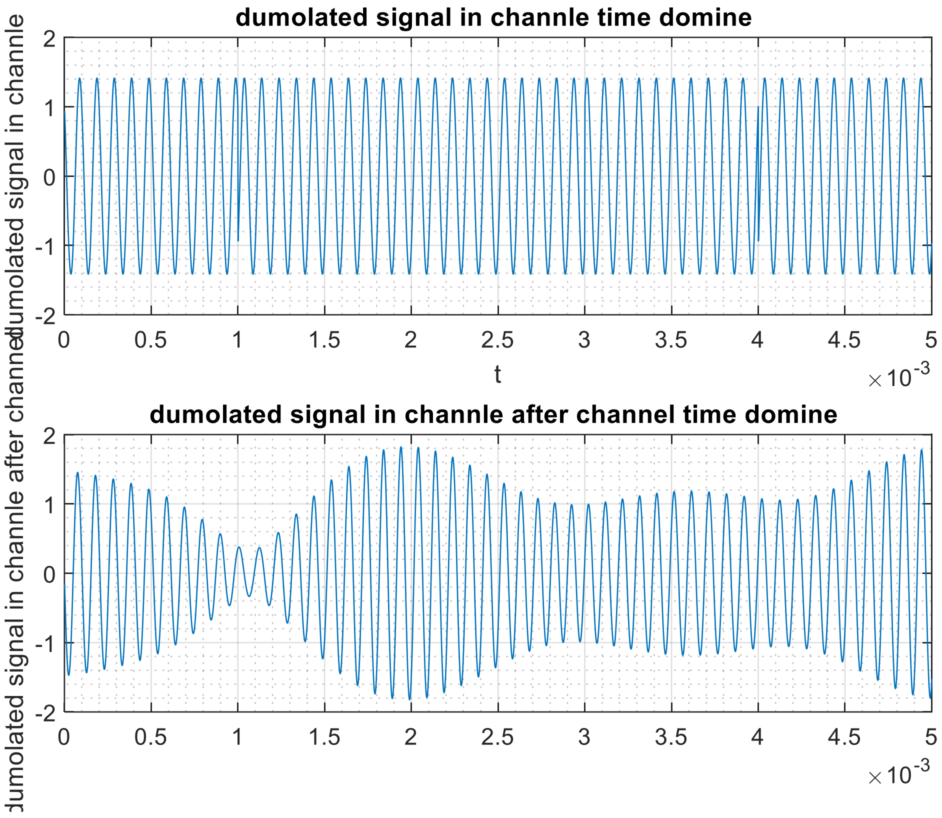
The first part of the random signal generated by the relation

```
x=(sign(rand(1,100)-.5)+1)/2
```

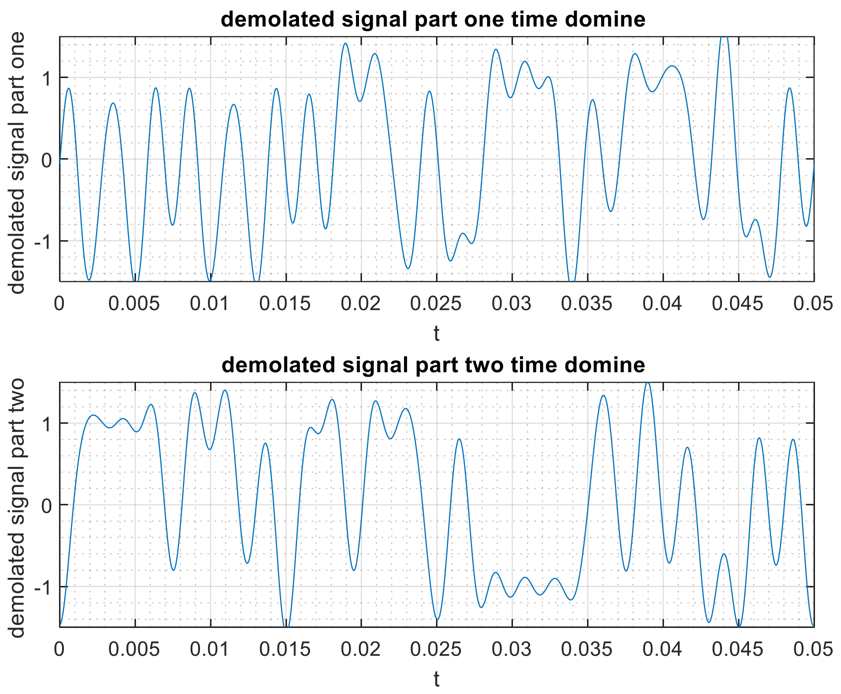


After splitting the signal



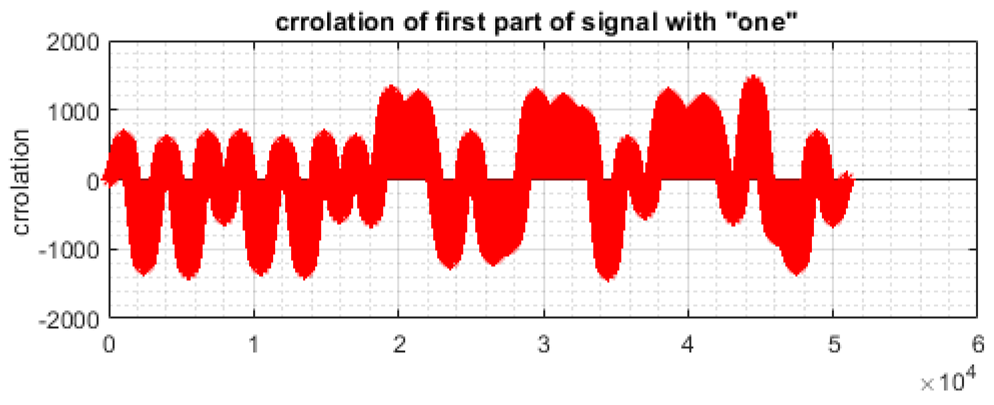


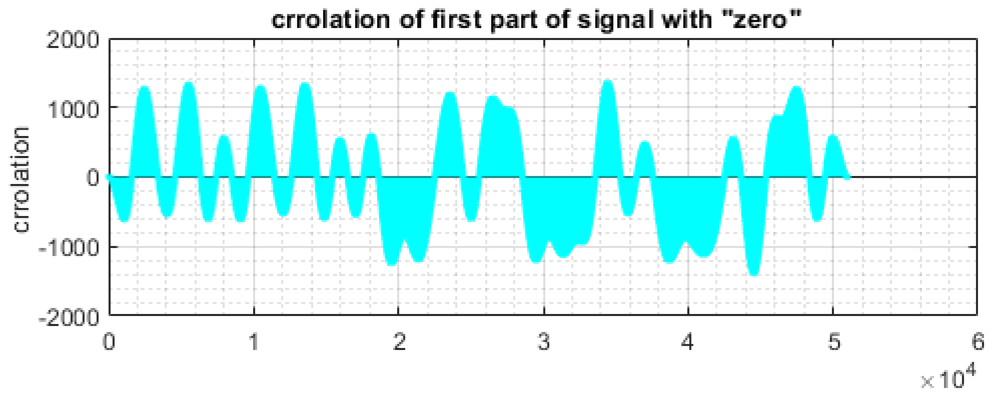
Above is the first diagram of the channel input signal and the second is the signal passing through the channel, ie filtered.

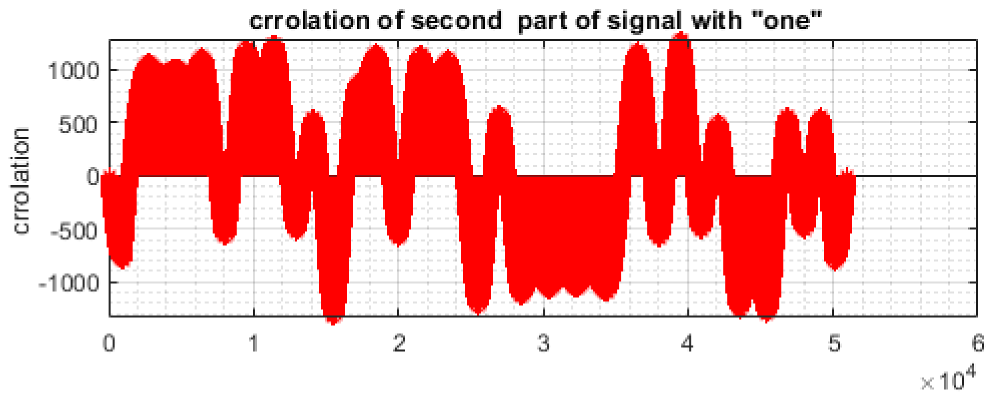


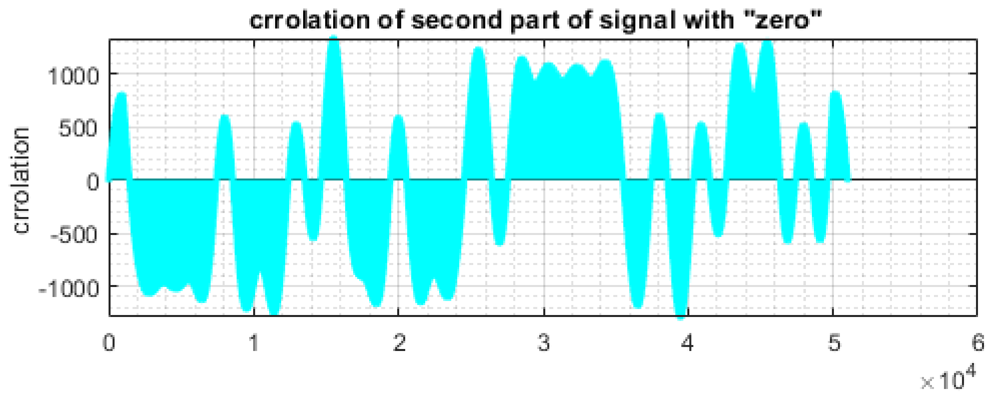
These are the two demodulated signals output from the channel, which I showed in a short time to make the details clearer.

The two diagrams below the correlation image of the partial signals and the two are each with samples one and zero, respectively, which, as I see, are also the location of zeros and ones.

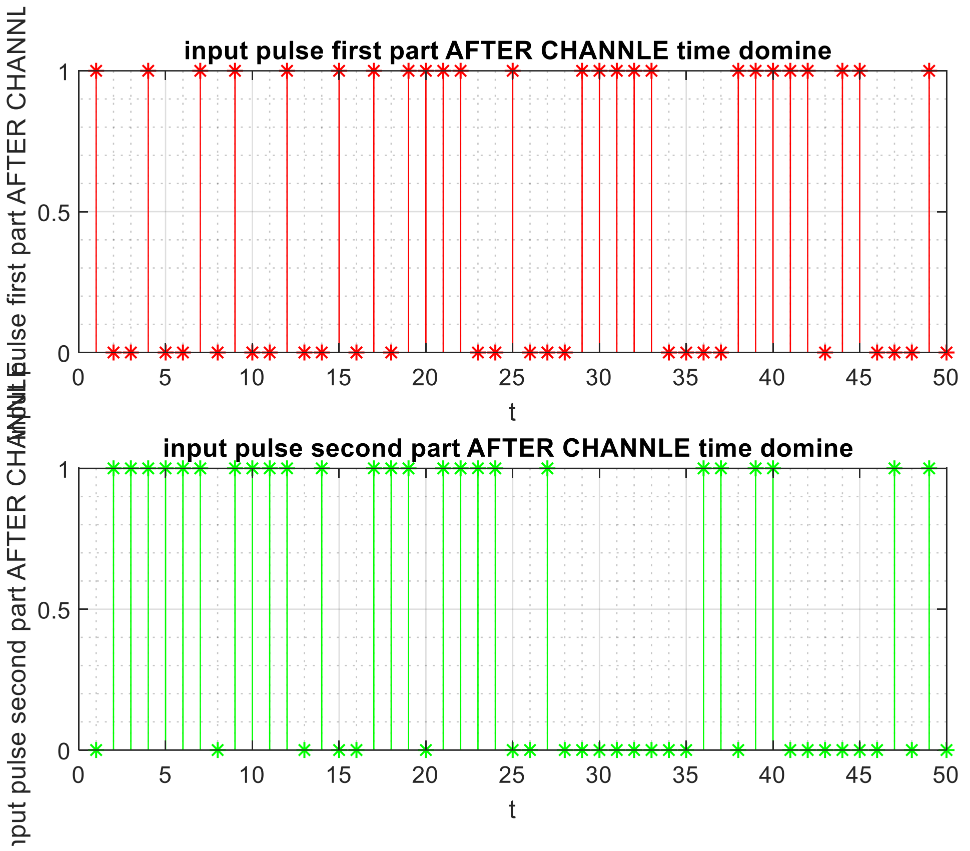




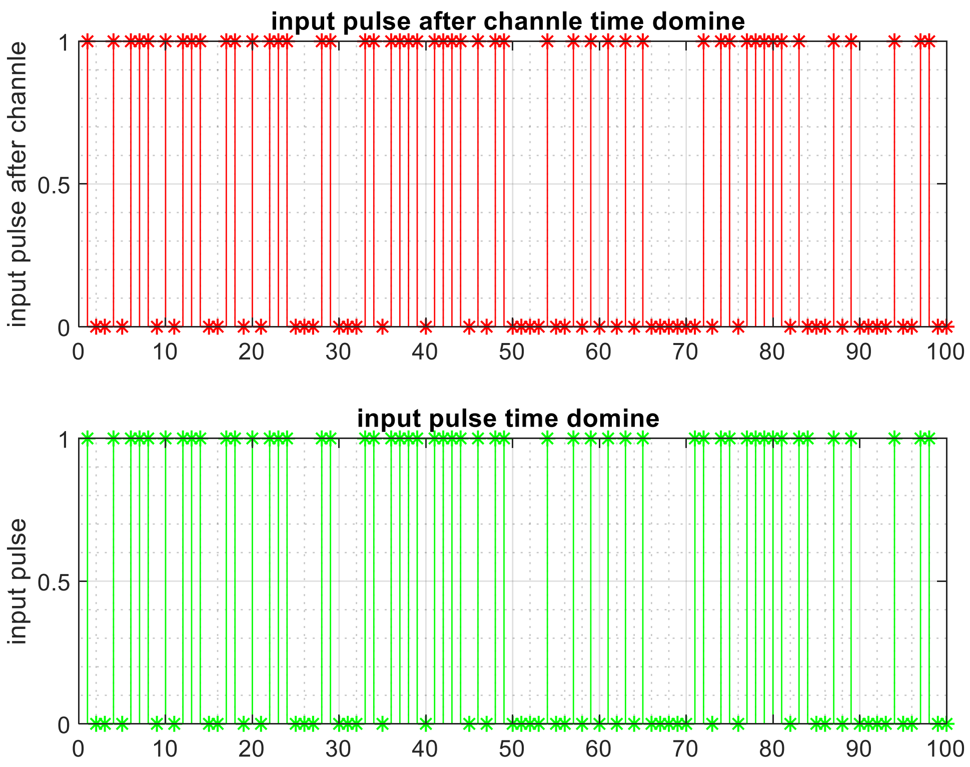




Now we estimate the even and odd signals from the channel



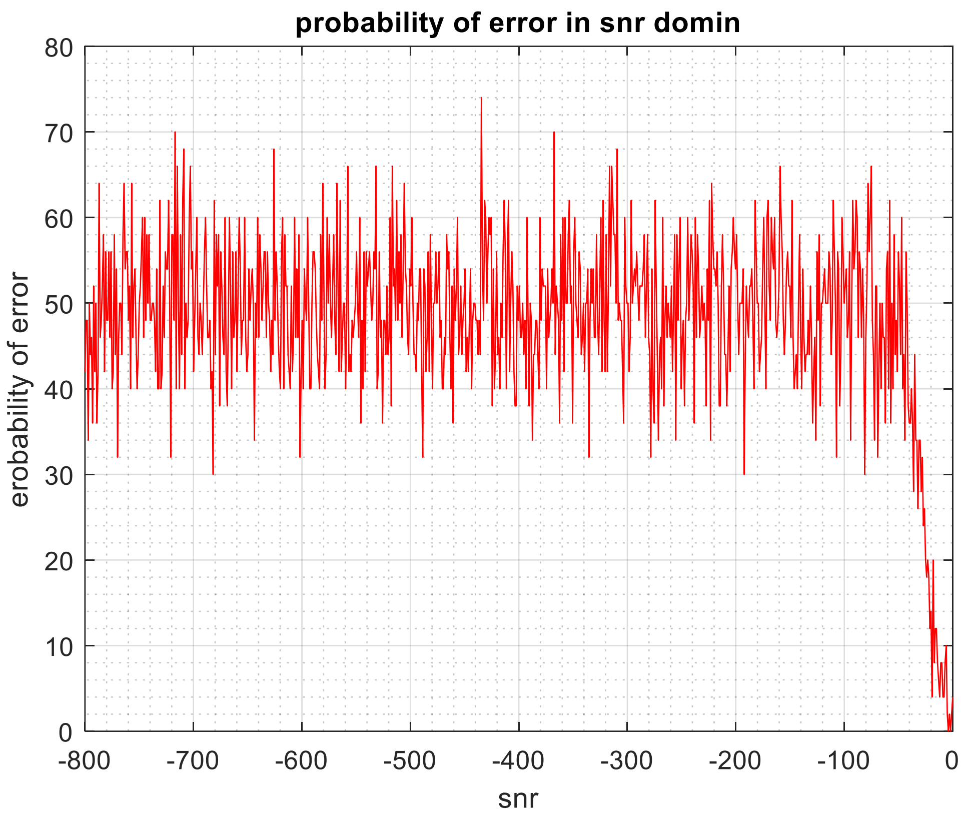
Finally, the final signal that the previous channel I see, our information is protected against the sent signal (bottom)



Part Two with Noise

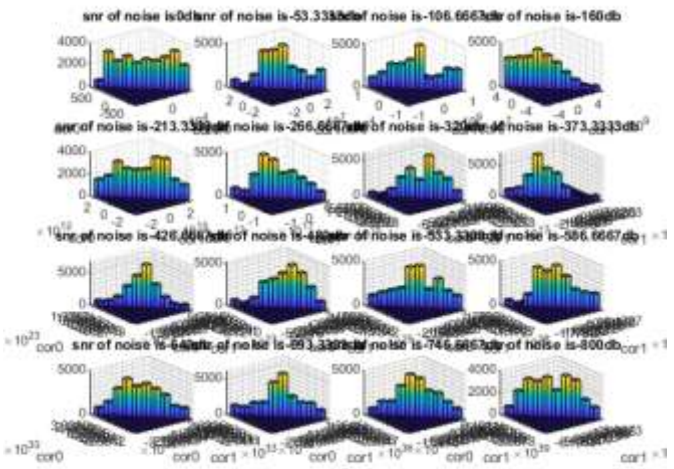
In this section, we want to plot the probability of error in terms of noise. The awgn function works based on snr, so by changing the value of snr in a loop for a random sequence of 50, we calculate this error. It took the code!

I have brought the output result below



Clearly, the limit behavior of the function is clear. From some value onwards, there is so much noise that it no longer has a signal, and in practice, the probability of error is limited by a 50% probability, which is one-half of the bit transmission.

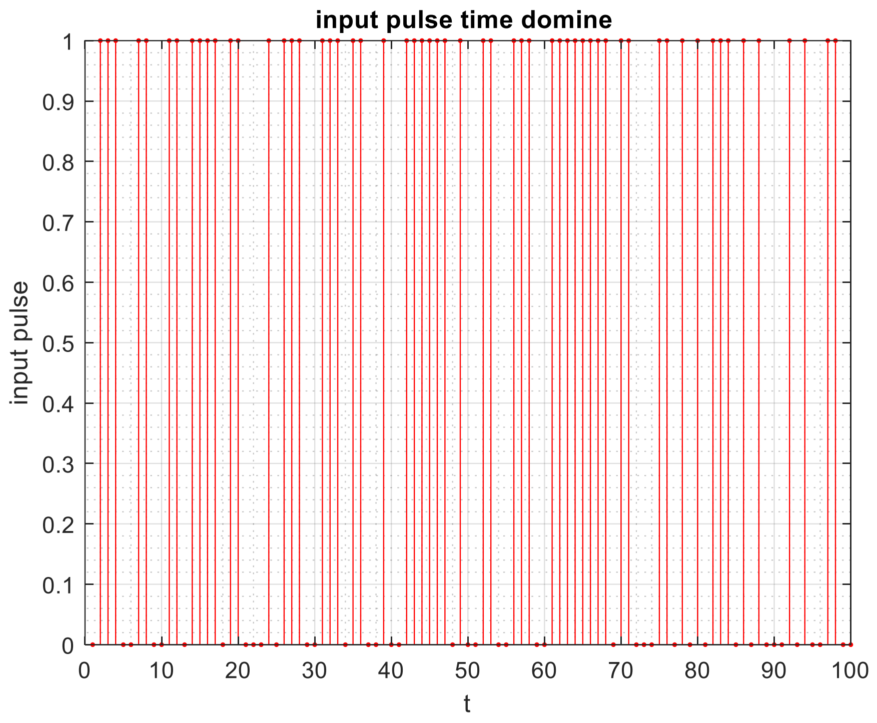
Part c) I calculated the histocram diagram for zero and one correlations and plotted it on a diagram for mixed SNRs for 16 different SNRs

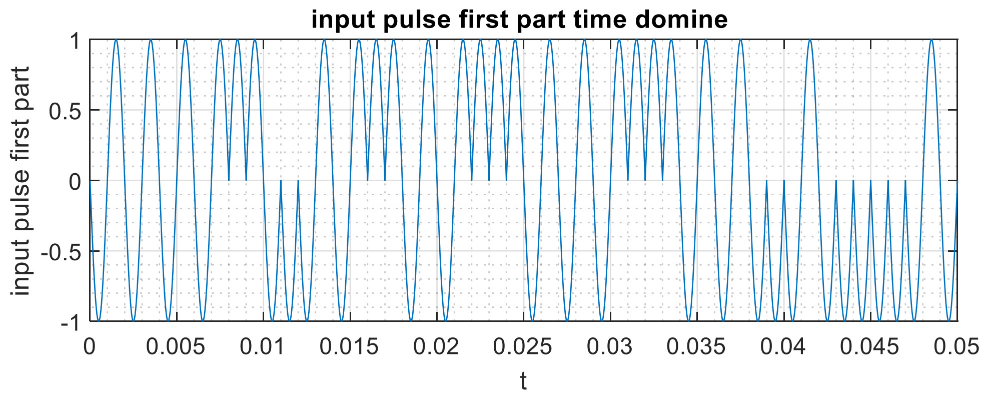


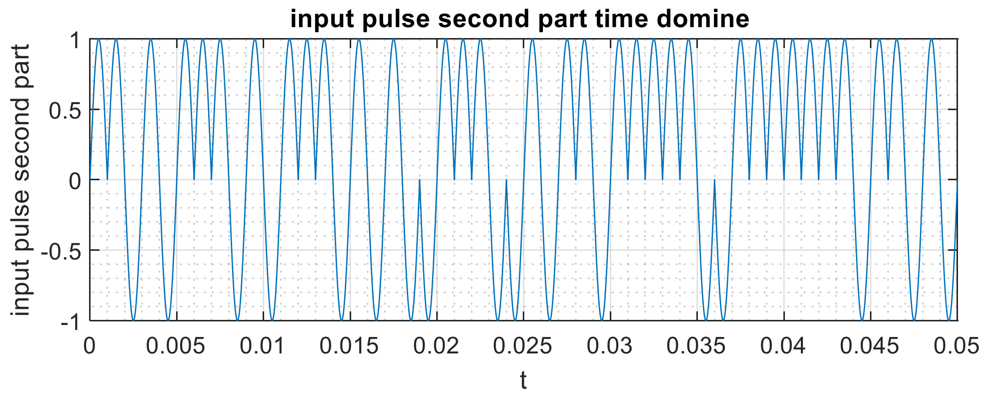
.

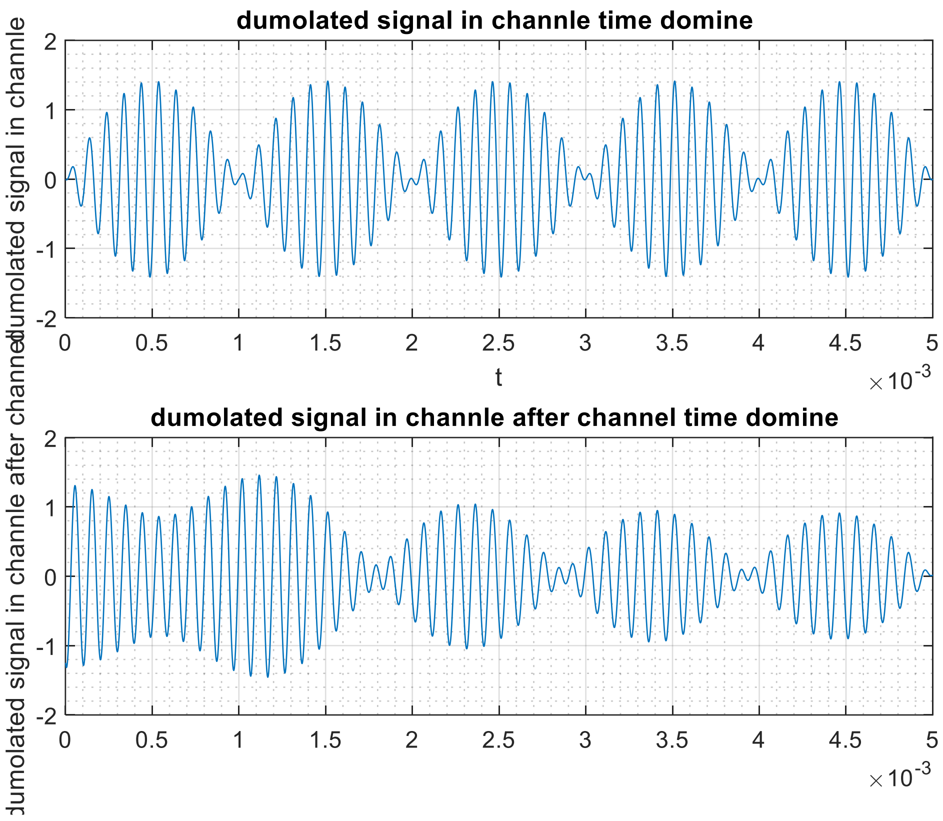
Part 4: Repeat all the steps with 500hz cytosine

The first part with 500 Hz

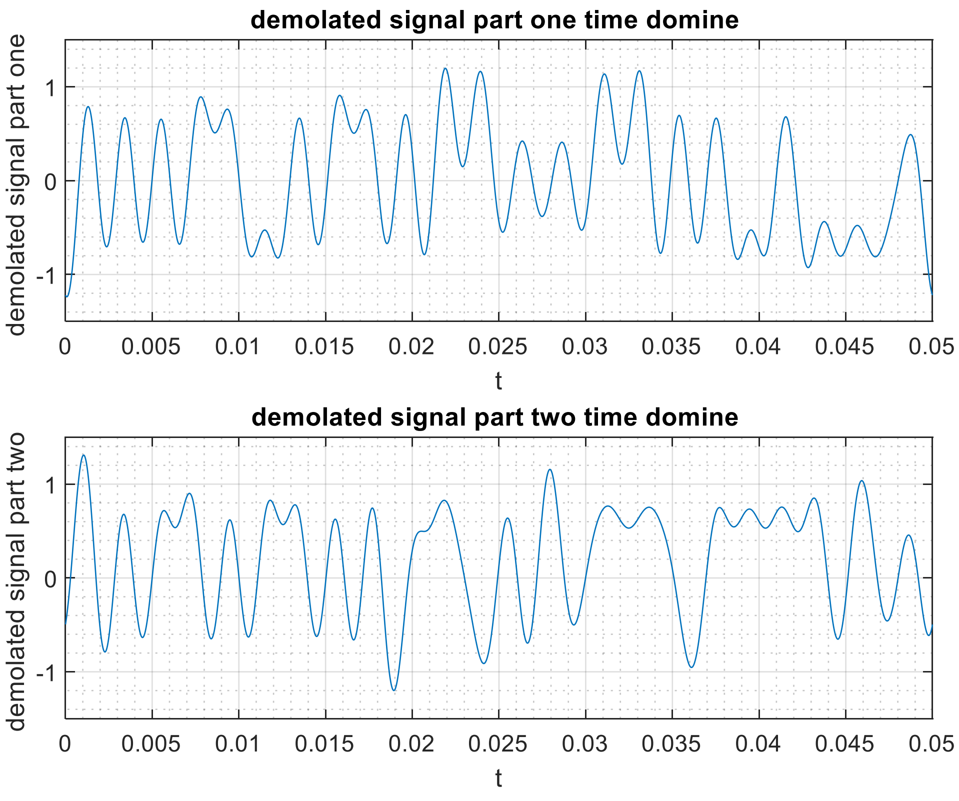




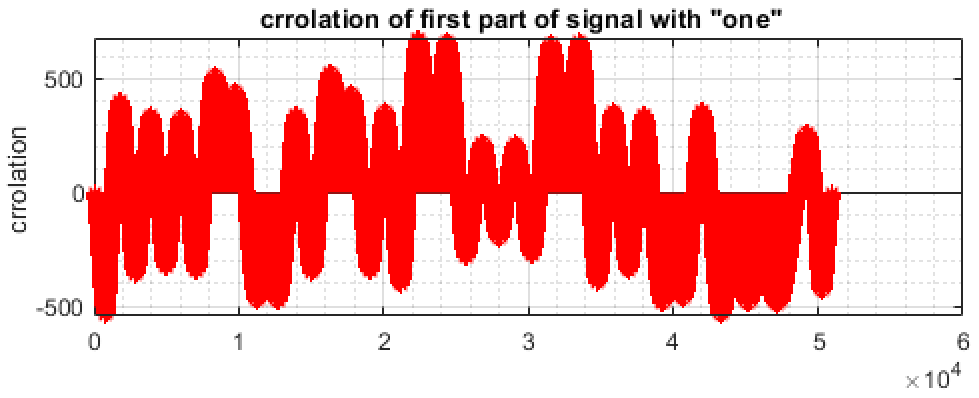


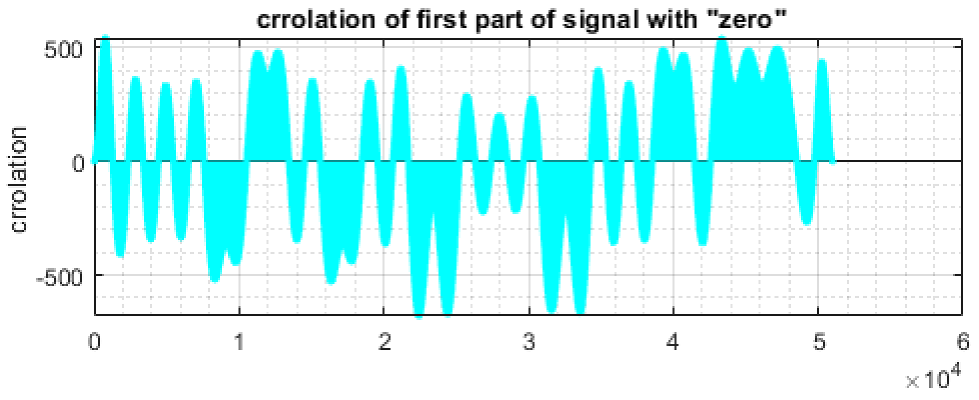


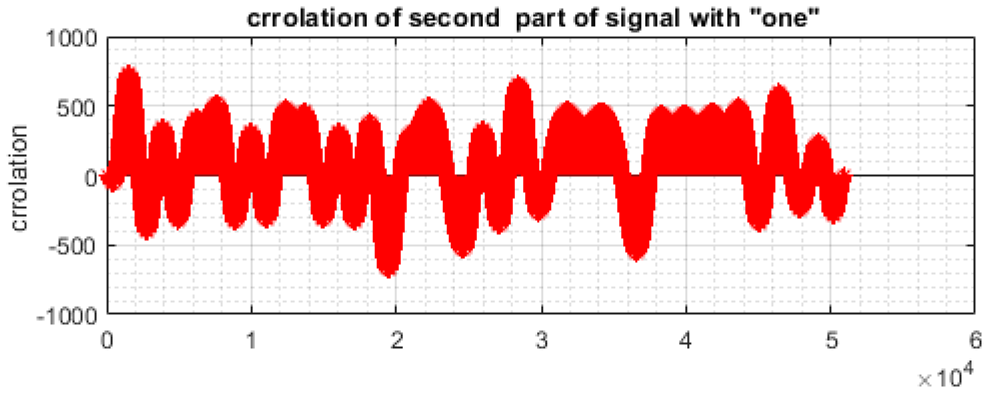
dumolated signal in channle time domine
dumolated signal in channle
t
×10^-3
dumolated signal in channle after channel time domine
dumolated signal in channle after channel
×10^-3

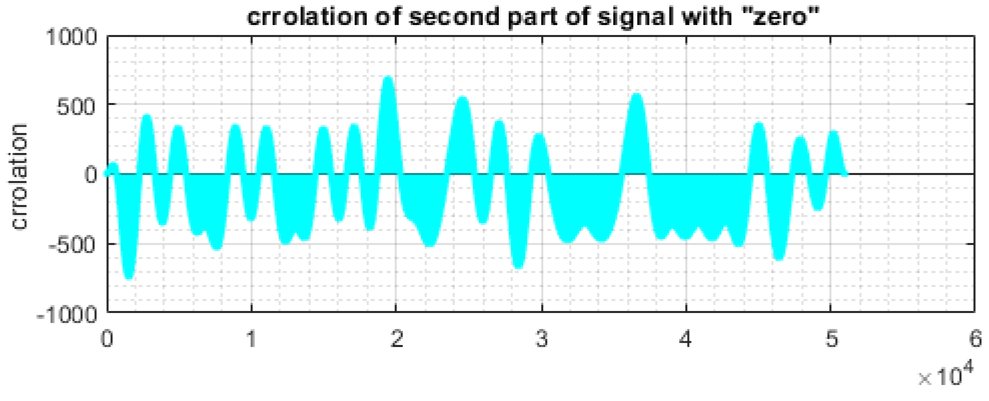


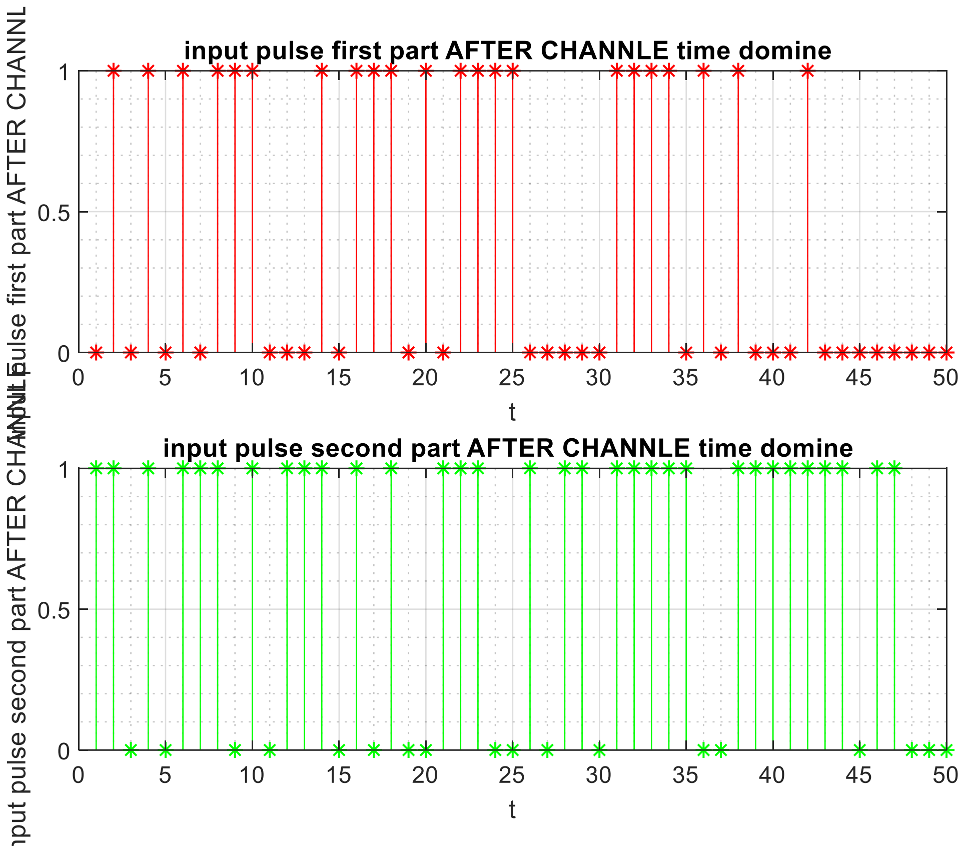
demolated signal part one time domine
demolated signal part one
t
demolated signal part two time domine
demolated signal part two
t

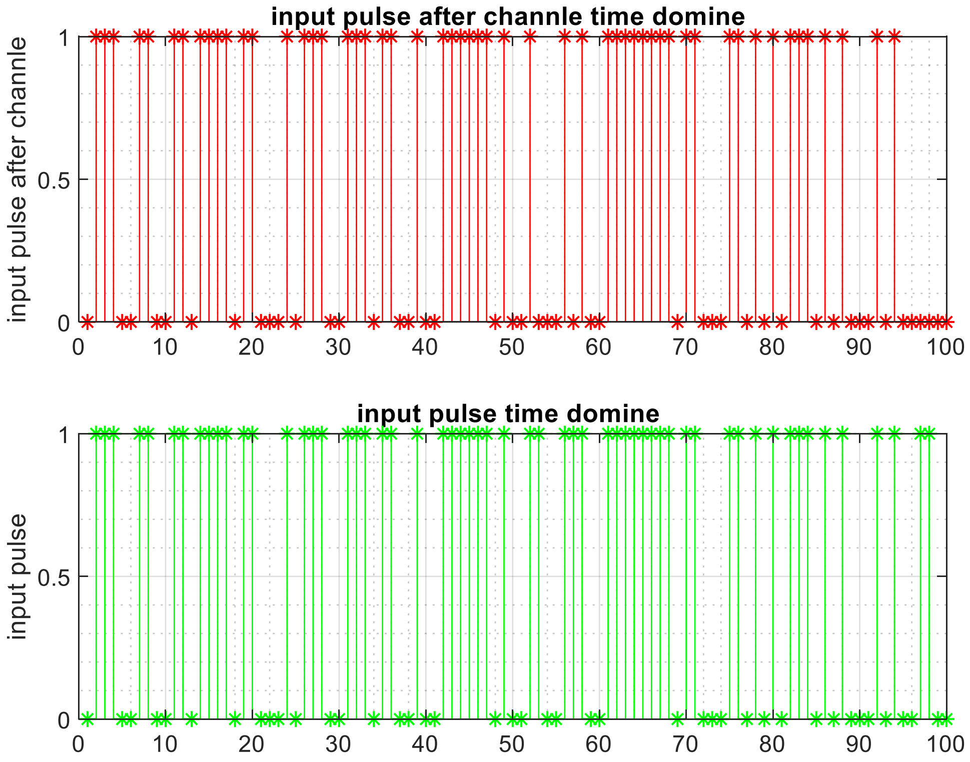

crrolation of first part of signal with "one"
crrolation
500
0
-500
0
1
2
3
4
5
6
×10^4


crrolation of first part of signal with "zero"
crrolation
500
0
-500
0
1
2
3
4
5
6
×10^4

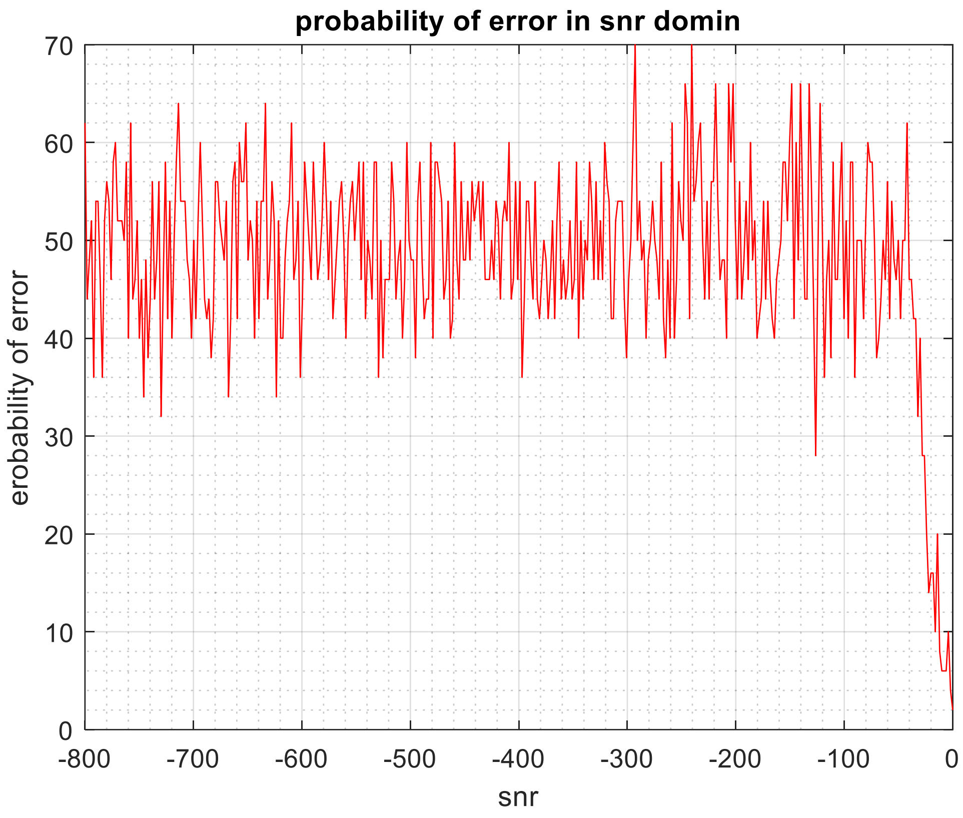

crrolation of second part of signal with "one"
crrolation
1000
500
0
-500
-1000
0
1
2
3
4
5
6
×10^4


crrolation of second part of signal with "zero"
crrolation
1000
500
0
-500
-1000
0
1
2
3
4
5
6
×10^4


input pulse first part AFTER CHANNLE time domine
input pulse first part AFTER CHANNL
t
input pulse second part AFTER CHANNLE time domine
input pulse second part AFTER CHANNLE
t

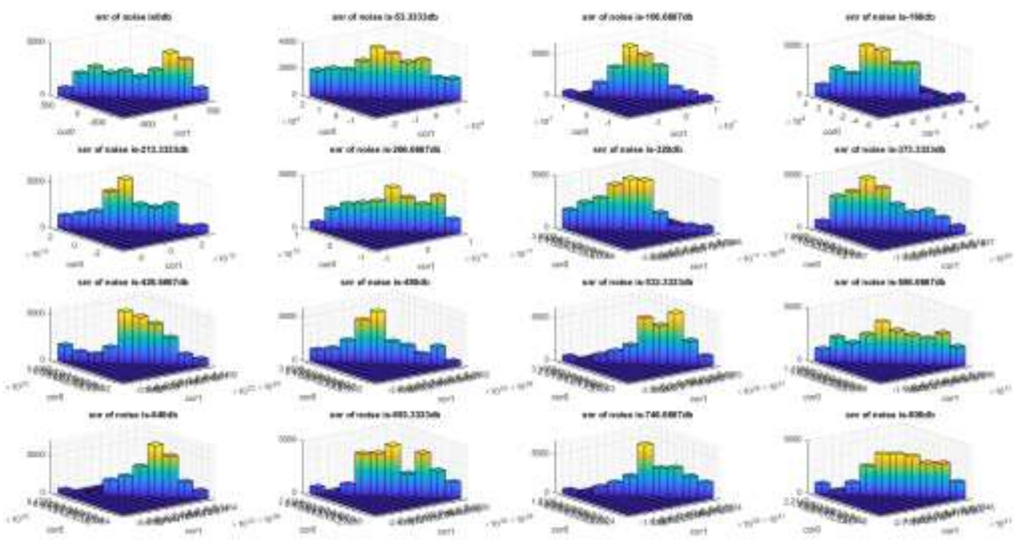

input pulse after channle time domine
input pulse after channle
input pulse time domine
input pulse

Part II Noise Analysis



The saturated behavior is clearly clear.

And finally, the histogram for the outputs of two pairs of ranks



third part)
For this section, we first execute two functions

```
function out = SourceGenerator(x)
 out = [];
    for i = 1:length(x)
        out = [out,de2bi(x(i),8)];
    end
 end

function out = OutputDecoder(x)
 out = [];
    for i = 1:8:length(x)
        out =
[out,bi2de([x(i),x(i+1),x(i+2),x(i+3),x(i+4),x(i+5),x(i+6),x(i+7)])];
    end
 end
```

We can easily generate 8-bit data
It takes hours to execute this code for a 100-digit entry (800bi), and its error can simply be measured by dividing the error by the sum of the squares divided by the maximum sum of the squares, assuming the number of zeros is equal to one.

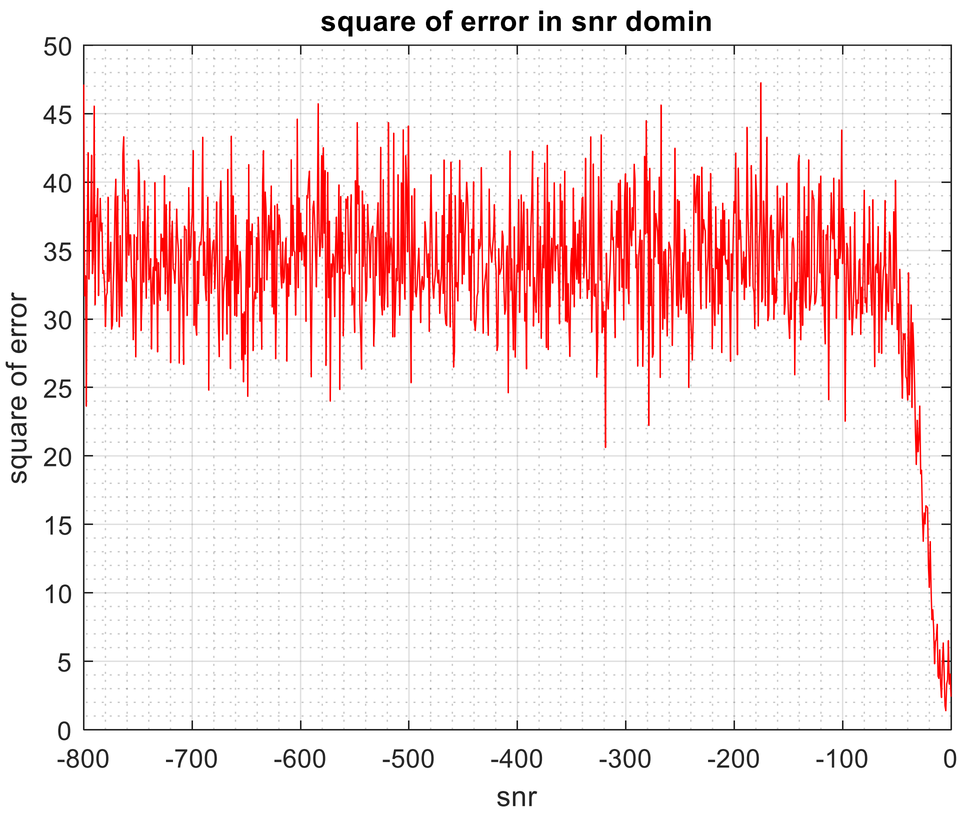
To run this part, when I first ran the program, it took hours to run, so I changed the amount of filtering modification, such as CHANNEL and ANALOGDEMOD, whose search algorithms added a value of a range that was supposed to be 0.

```
function out = Channel( input , Fs ,Fc ,Bw)
    num = length( input );
    fft_signal = fftshift (fft(input));
    F = -Fs/2 : Fs/num : Fs/2 - Fs/num;
    out_fft = zeros(1,length(fft_signal));
    min2 = floor((Fc-Bw/2+Fs/2)/(Fs/num));
    max2 = floor((Fc+Bw/2+Fs/2)/(Fs/num));
    min1 = floor((-Fc-Bw/2+Fs/2)/(Fs/num));
    max1 = floor(-Fc+Bw/2+Fs/2)/(Fs/num);
    for i = min1-100:1:max1+100
        if abs(F(i)-Fc)<=Bw/2
            out_fft(i) = fft_signal(i);

        elseif abs(F(i)+Fc)<=Bw/2
            out_fft(i) = fft_signal(i);
        else
            out_fft(i) = 0;
        end
    end
    for i = min2-100:1:max2+100
        if abs(F(i)-Fc)<=Bw/2

            out_fft(i) = fft_signal(i);

        elseif abs(F(i)+Fc)<=Bw/2
            out_fft(i) = fft_signal(i);
        else
            out_fft(i) = 0;
        end
    end
    out  = ifft(ifftshift(out_fft));
end
```

I determined the important points, which included 4 points, and I determined them by hand and reduced the search area to reduce the program time, as well as for the ANALOG DEMOD function.

```
function [x1,x2]=AnalogDemod(input,Fs,Bw,Fc)
t=1/Fs:1/Fs:(length(input)*1/Fs);
Cos_vec = cos(2*pi*Fc*t);
Sin_vec = sin(2*pi*Fc*t);
x1=2*input.*Cos_vec;
x2=2*input.*Sin_vec ;
X1=fftshift(fft(x1));
X2=fftshift(fft(x2));
F=linspace(0,Fs,length(X1))-Fs/2;
min = ((Fs/2)-(Bw/2))/(Fs/length(X1));
max = ((Fs/2)+(Bw/2))/(Fs/length(X1));
out1 = zeros(1,length(X1));
out2 = zeros(1,length(X1));
for i=min-100:1:max+100
   if(abs(F(i))<Bw)
        out1(i)=X1(i);
        out2(i)=X2(i);
   end
end
x1=ifft(ifftshift(out1));
x2=ifft(ifftshift(out2));
end
```

Now run the program for 1000 points and draw the sum of your squares according to the given SNR



The limit behavior is clearly due to the fact that the noise at the maximum state is limited and can not be greater than that.

Manually calculate the total sum of squares

$$E((X - X_{IN})^2 | \lim_{n \to \infty} N) = 2 * (2^8 - 1)^2 \cong 130000$$

As we can get from the program, this value is about 120,000, which is a number close to this number.