Report

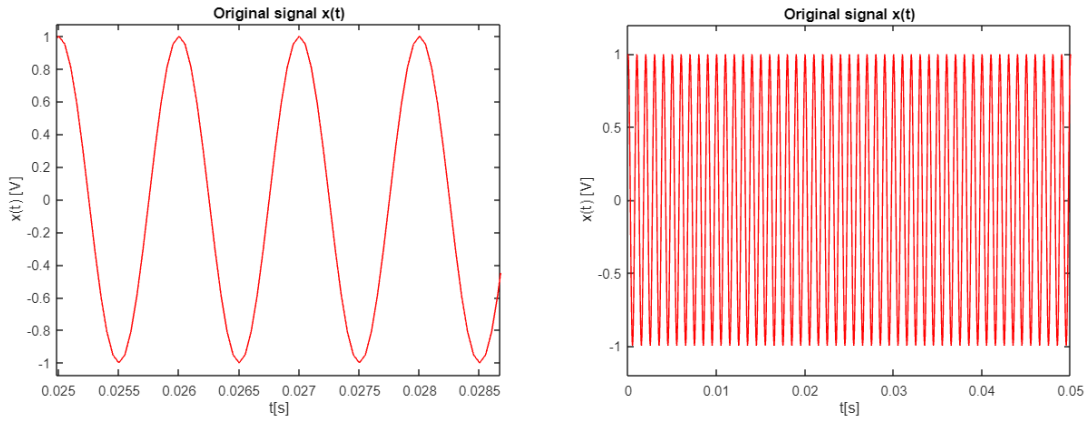
1. Activity1: Filtering example using a finite response filter (FIR)

```
close all % close all plots
clear all % clear all variables
clc % clear the screen
```

1.1 Generation of the sinusoid and plot

Firstly, I have created a sinusoid with Amplitude A=1 with the code below.

```
A=1; % Amplitude of the sinusoid [V]
f0=1000; % Frequency of the sinusoid [Hz]
fs=20000; % sampling frequency
duration=0.05; % signal duration in seconds

%% Generation of Sinusoidal signal
[t,x,N]=SinusoidalSource_2023(A,f0,duration,fs); % generation of the signal

figure
plot(t,x,"r")
xlabel("t[s]")
ylabel("x(t) [V]")
title("Original signal x(t)")
axis([min(t) max(t) 1.2*min(x) 1.2*max(x)])
```

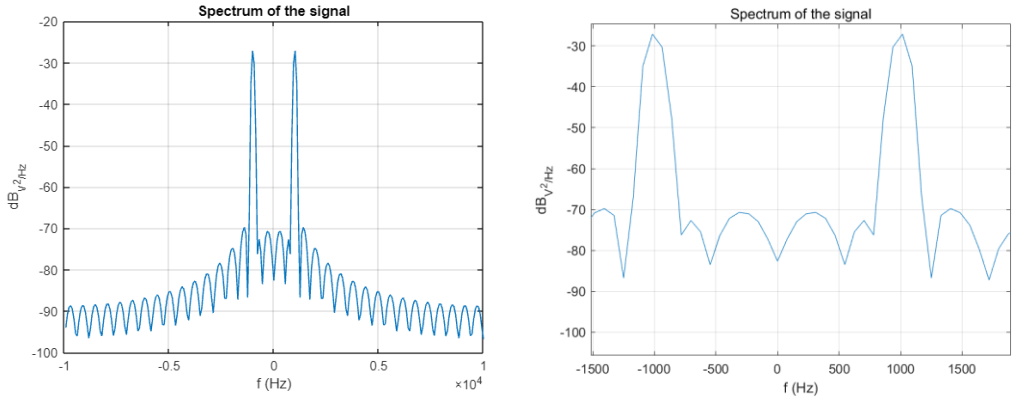


These plots represent the sine wave that we have created, since his amplitude is A=1, we can affirm that his signal power is equal to A^2/2, so equal to 0.5.

```
signal_power=0.5*A^2;
fprintf('sinusoid power [V^2]=%f',signal_power)
```

```
sinusoid power [V^2] =0.500000
```

```
figure
PlotSpectrum(x,fs);
title('Spectrum of the signal');
```



These two plots above represent the spectrum of the signal: the spectrum of a signal shows how much power is contained in each of its harmonic or spectral components or the frequency spectrum of the signal. A plot of the frequency components on the x-axis and attendant Power in that frequency on the y-axis is called the Power Spectrum of the signal. Its units are watts per Hertz. The spectrum of the sine wave shows that power is concentrated at the carrier frequency, since here we have a two-sides Power Spectrum, the power is mostly concentrated on -1000Hz and 1000Hz and that the total power is the sum of the powers in both the negative and positive terms.

1.2 Noise Generation

The thermal noise affecting communications is dubbed white Gaussian noise (WGN): the mean value is zero, all samples are uncorrelated, hence the power spectral density (PSD) is frequency flat, and the probability density function (pdf) is.

$$P(x) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{x^2}{2\sigma^2}}$$

Now, we are going to generate a Gaussian noise with the code below.

```
sigma=0.6; % standard deviation of the noise. The noise power is sigma^2 [V^2]
noise=sigma*randn(1,N); % generation of the noise
```

In this case, N is the length of the signal generated before and it represents the number of Samples, more sample we have, the noise that we are going to create is closer to the Gaussian function.

```
fprintf('Noise power [v^2]= %f',sigma^2)
```

```
Noise power [v^2] = 0.360000
```

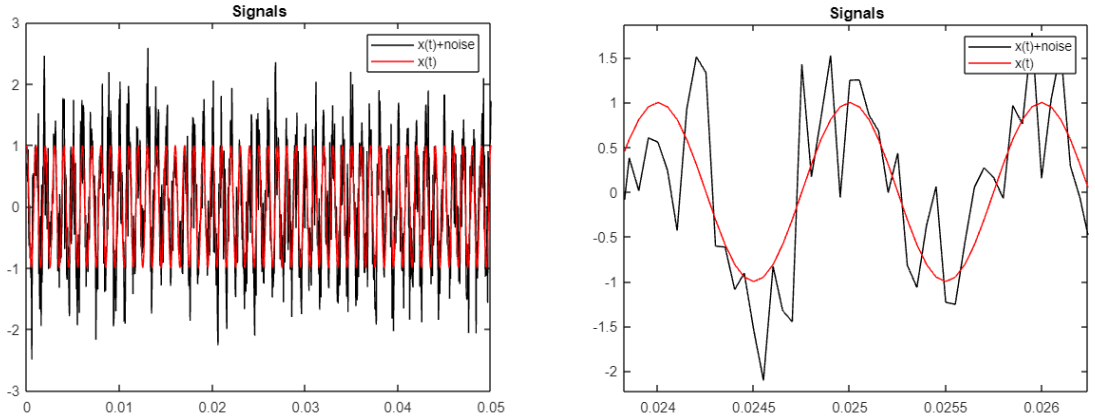
```
x_noisy=x+noise; % add Gaussian noise to the sinusoid x created in the section
before
signal_to_noise_ratio_dB=10*log10(signal_power/(sigma^2))
```

```
signal_to_noise_ratio_dB = 1.4267
```

```
fprintf('SNR[dB]=%f',signal_to_noise_ratio_dB)
```

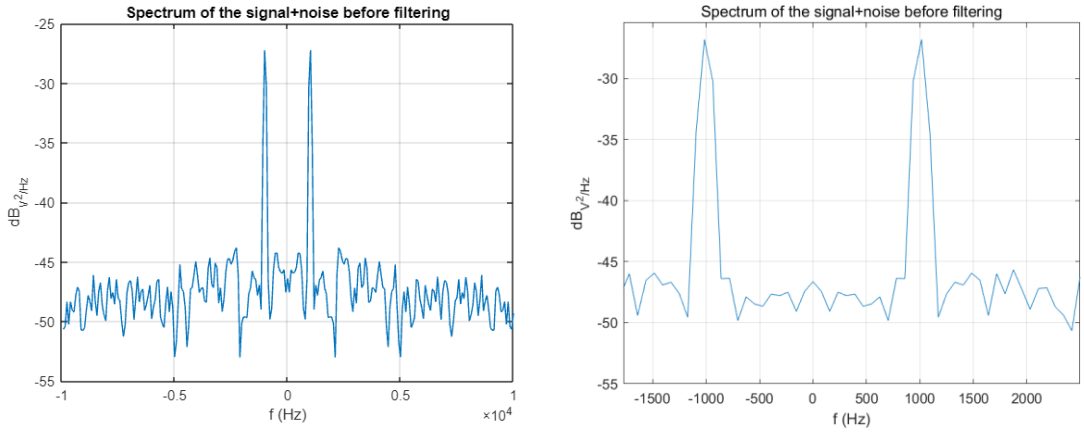
```
SNR [dB]=1.426675
```

```
figure
plot(t,x_noisy,'k') % plot the signal with noise x(t)+noise
hold on
plot(t,x,'r') % plot the noise
legend('x(t)+noise','x(t)')
title('Signals');
```



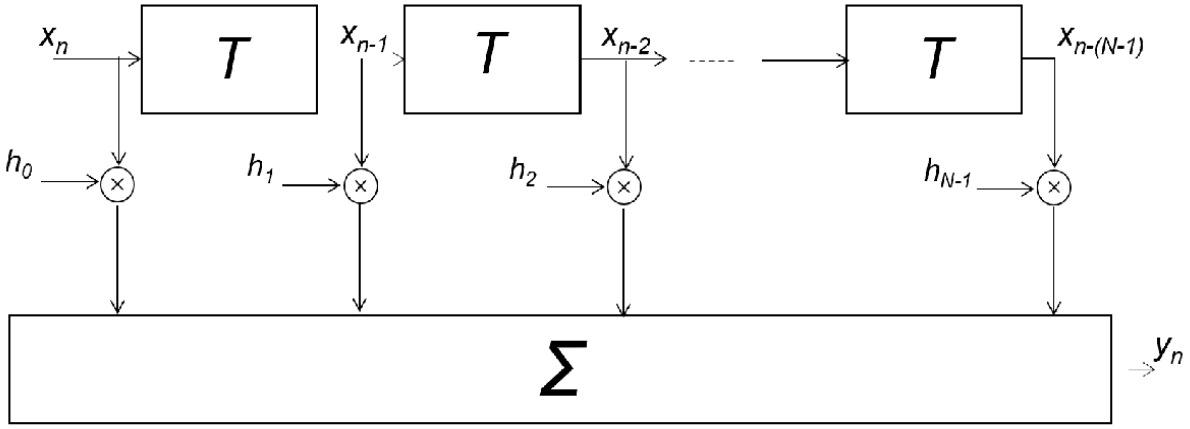
This plot above represents in Black the original signal x(t) plus the noise, so comparing the black signal with the red one, we can see that we have a worse behavior; the noise interferes a lot the original signal.

```
figure
PlotSpectrum(x_noisy,fs);
title('Spectrum of the signal+noise before filtering');
```



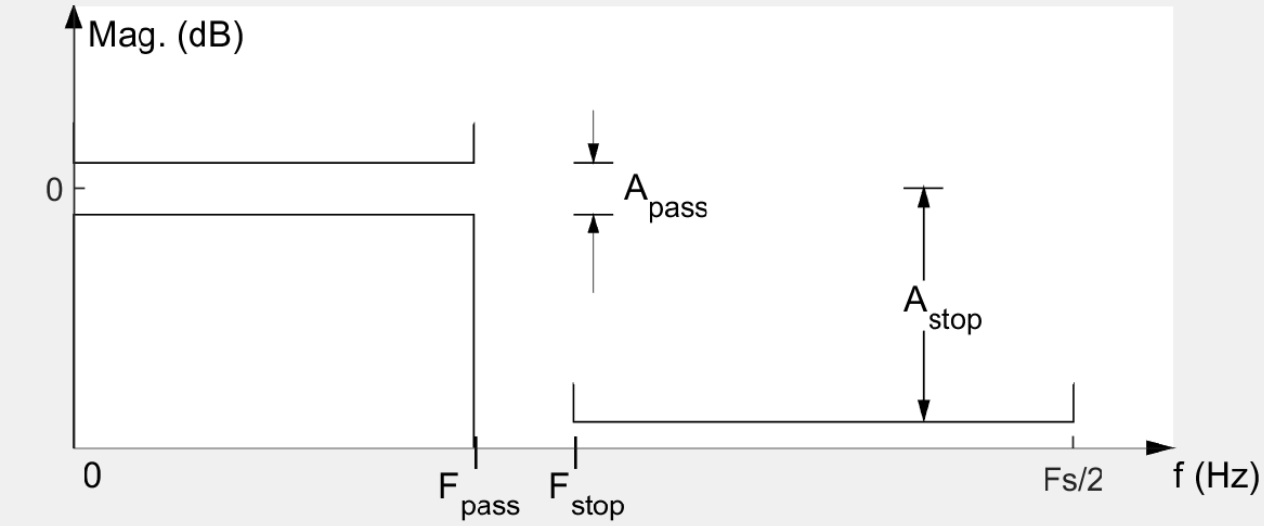
There plots above represent the power spectrum of the sinusoid after adding the noise, we can observe that the power is still more concentrated in -1000Hz and 1000Hz.

1.3 FIR_filter_architecture




```
Nf=400; % number of FIR filter taps; higher this number, better the filter is
```

1.3.1 Lowpass filter design

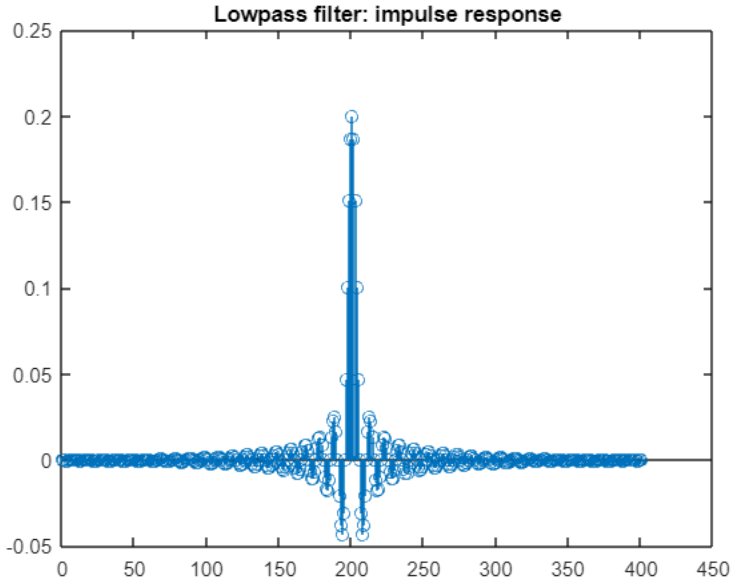



```
%% cut frequency for a low-pass filter
Fpass=2000; % 3 dB cut frequency Why 3db? Why 2000 Hz?
h_lowpass=fir1(Nf,Fpass/(0.5*fs)); % filter design
```

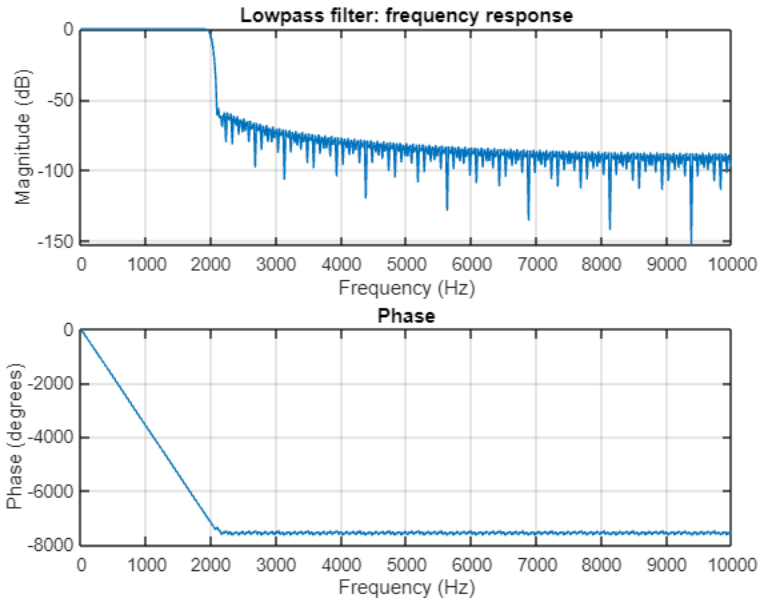
The low pass filter is a device that isolate the signals which have frequencies higher than the cutoff frequencies, it permits only signals with frequencies lower than the cutoff frequency.

1.3.2 Filter impulse response and frequency response (transfer function)

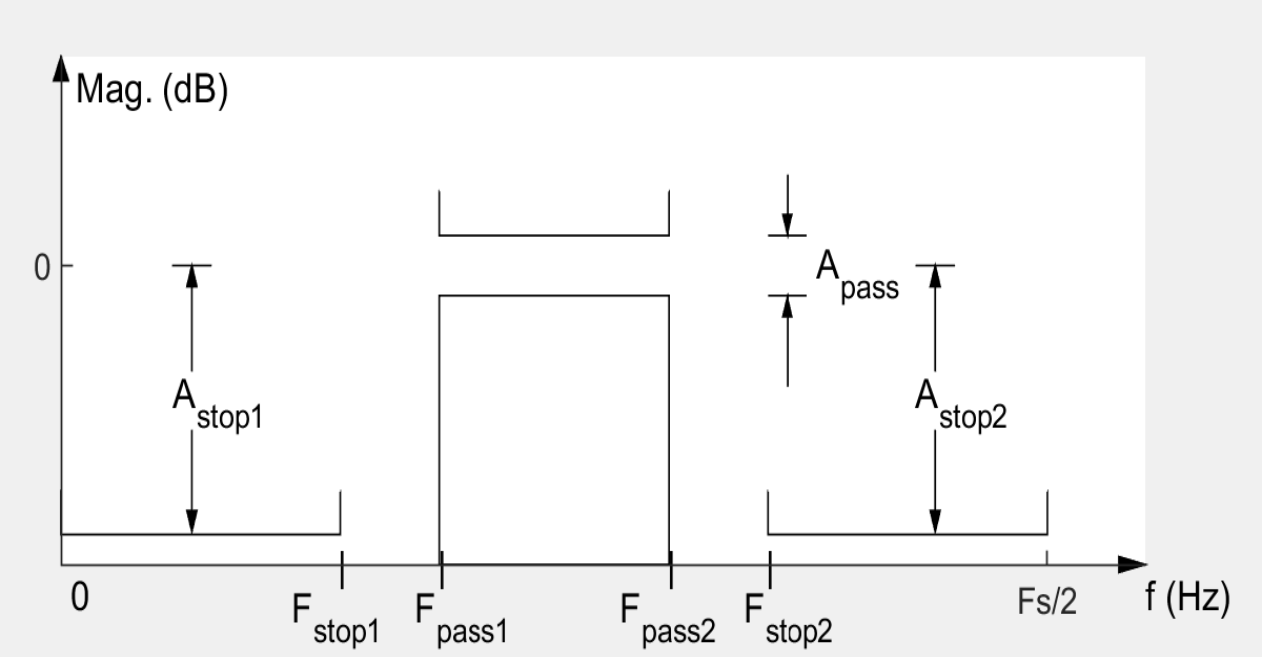
```
stem(h_lowpass) % filter taps (coefficients), that is, filter impulse response

title('Lowpass filter: impulse response')
```




```
freqz(h_lowpass,1,[],fs); % plot the frequency response
title("Lowpass filter: frequency response")
```



1.3.3 Pass band filter design

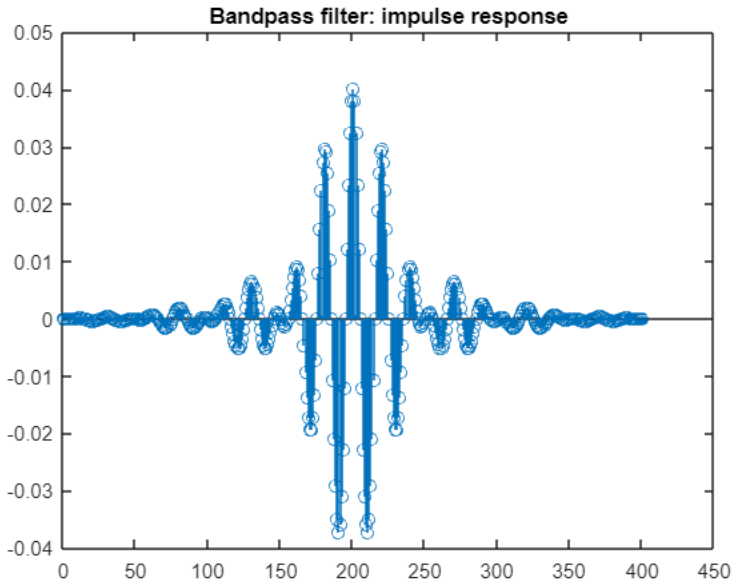


A band pass filter is defined as a device that allows frequencies within a specific frequency range and reject frequencies outside that range. The Band Pass Filter has two cutoff frequencies. The first cutoff frequency is from a higher pass filter. This will decide the higher frequency limit of a band that is known as the higher cutoff frequency (Fpass2). The second cutoff frequency is from the low pass filter. This will decide the lower frequency limit of the band and that is known as lower cutoff frequency (Fpass1).

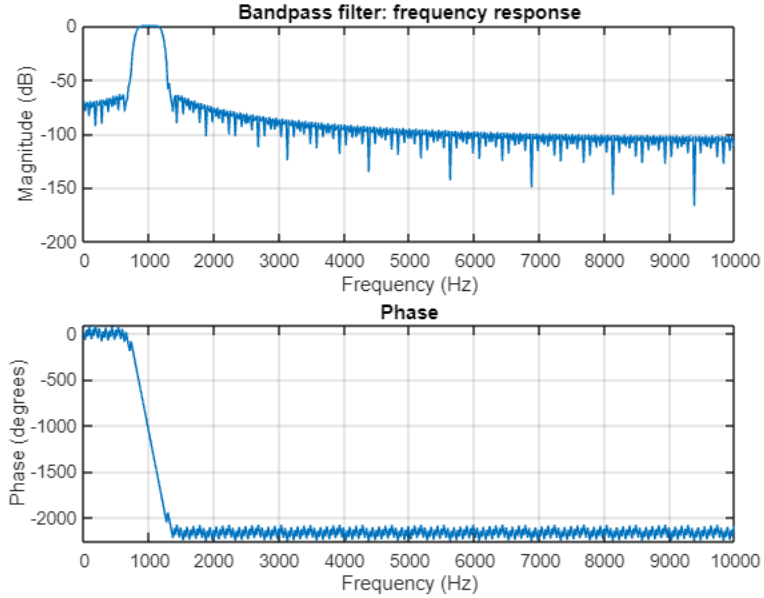
```
 Fpass1=800; % low cut frequency of the filter
 Fpass2=1200; % high cut frequency of the filter
 h_bandpass=fir1(Nf,[Fpass1/(0.5*fs) Fpass2/(0.5*fs)], "bandpass"); % filter
design
```

1.3.4 Filter impulse response and frequency response (transfer function)

```
 stem(h_bandpass) %filter taps (coefficients), that is, filter impulse response
 title('Bandpass filter: impulse response')
```




```
freqz(h_bandpass,1,[],fs); % plot the frequency response
title('Bandpass filter: frequency response')
```



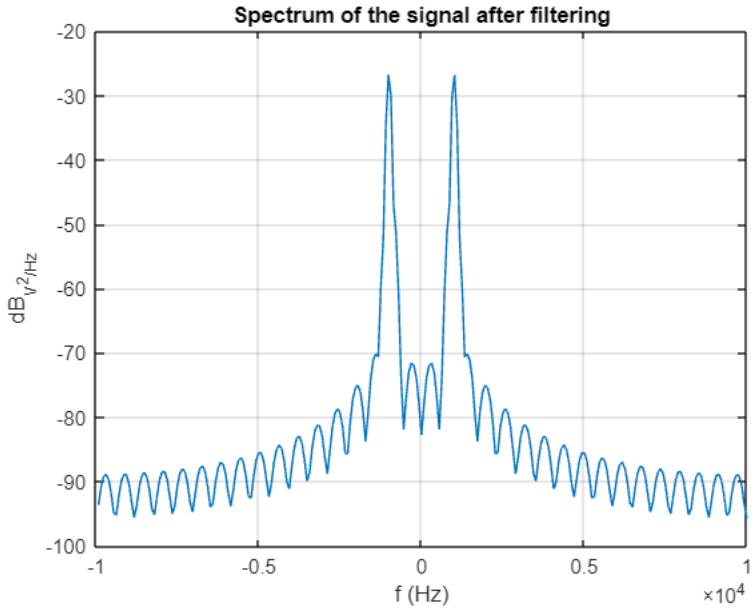
1.3.5Filtering

```
y=conv(x_noisy,h_bandpass,'same'); % filter the signal with the bandpass filter
% y=conv(x_noisy,h_lowpass,'same'); % filter the signal with the lowpass filter
```

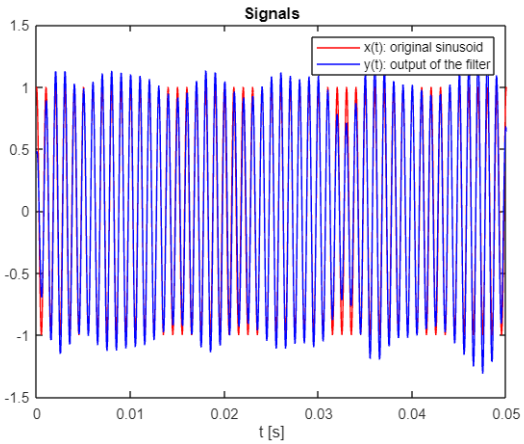
With conv () you do the convolution.

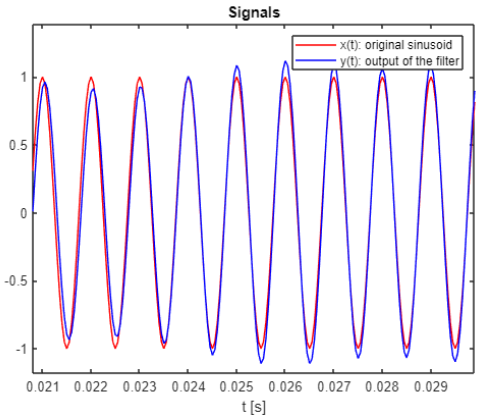
1.3.6Plots of the outputs of the bandpass filter

```
figure
PlotSpectrum(y,fs);
title('Spectrum of the signal after filtering');
```



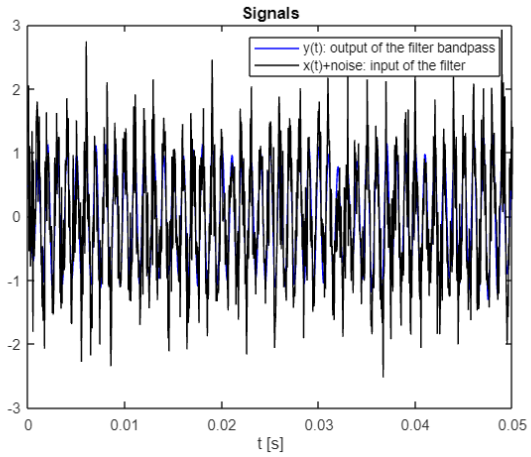

```
figure
plot(t,x,"r")
hold on
plot(t,y,'b')
xlabel('t [s]')
legend ('x(t): original sinusoid','y(t): output of the filter')
title('Signals');
```

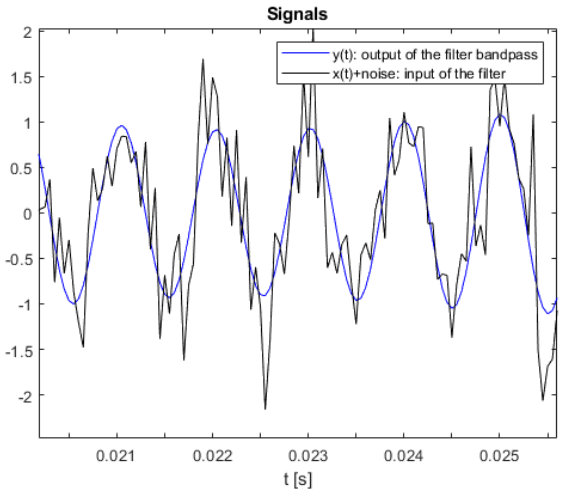




In this plot we can observe the comparison between the original signal and the output of the band-pass filter. We can see that even if there are still some small differences between the two sinusoids but the noise is almost deleted.

```
figure
plot(t,y,'b')
hold on
plot(t,x_noisy,'k')
xlabel('t [s]')
legend ('y(t): output of the filter bandpass','x(t)+noise: input of the filter')
title('Signals');
```

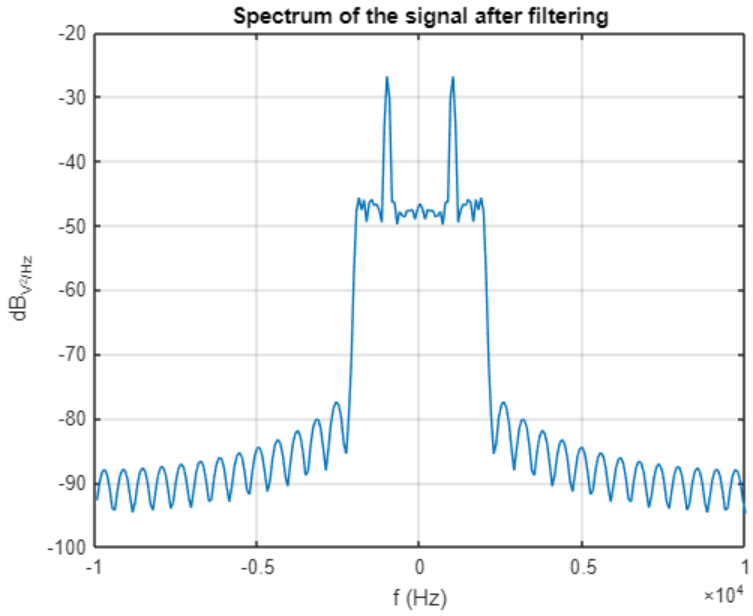




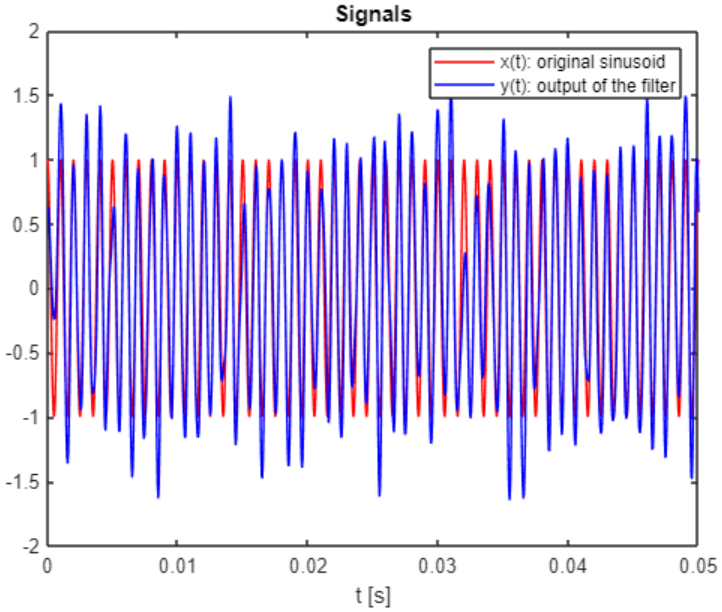
Making the comparison between the input of the filter and the output, we can see that the noise is almost deleted.

1.3.7Plots of the outputs of the lowpass filter

```
figure
PlotSpectrum(y,fs);
title('Spectrum of the signal after filtering');
```

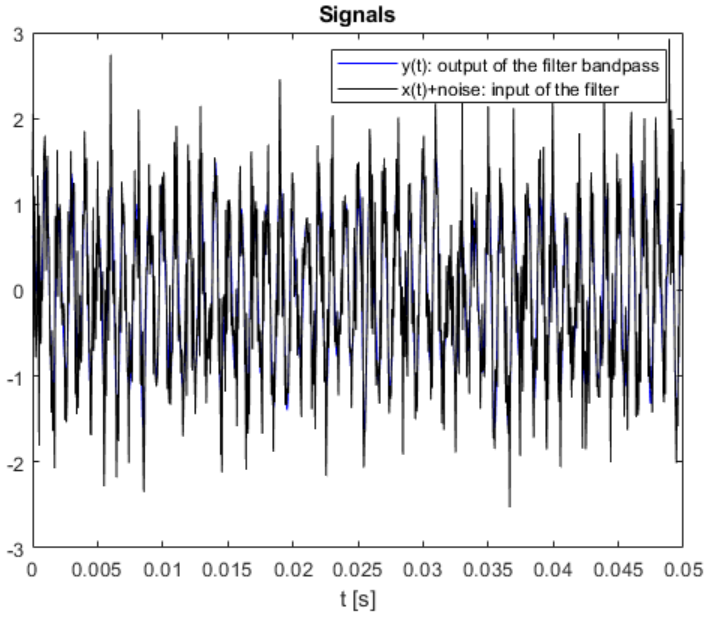



```
figure
plot(t,x,"r")
hold on
plot(t,y,'b')
xlabel('t [s]')
legend ('x(t): original sinusoid','y(t): output of the filter')
title('Signals');
```



Even not doing the comparison plot between the output of lowpass and band pass filter, we can clearly observe that the Band Pass Filter shows an output whose trend is closer to the original signal x(t). In fact, unlike the low pass filter which only pass signals of a low frequency range, a band pass filter passes signals within a certain “band” or “spread” of frequencies without distorting the input signal or introducing extra noise.

```
figure
plot(t,y,'b')
hold on
plot(t,x_noisy,'k')
xlabel('t [s]')
legend ('y(t): output of the filter bandpass','x(t)+noise: input of the filter')
title('Signals');
```



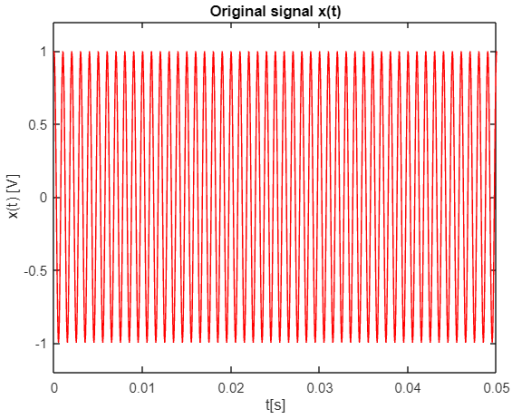
1.4 Compare the outputs for different powers of the sine wave.

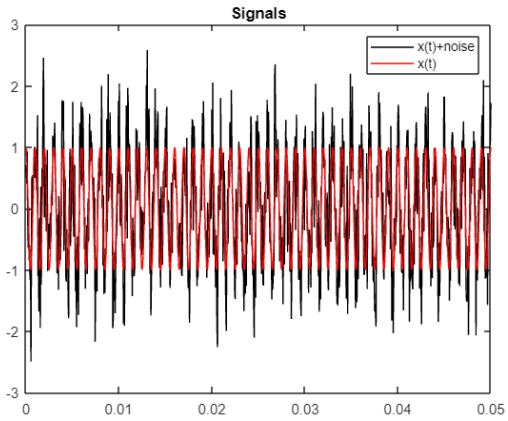
We know that in order to obtain different powers of the sine wave, we have to change the amplitude of the sine.

In this comparison, we always consider the noise with standard deviation equal to 0.6, so the noise power is 0,36.

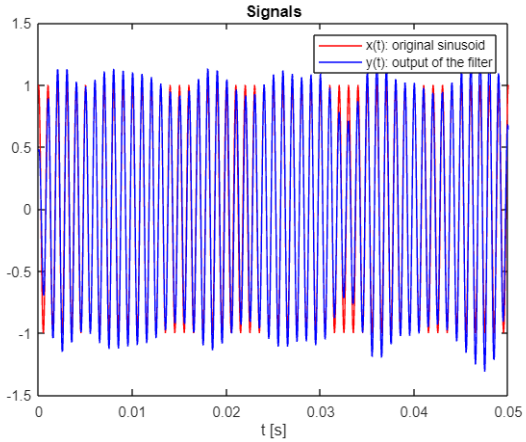
Here, we have the sine wave with A=1 and P=0.5.

Adding to this signal the noise, we get a interfered signal with `signal_to_noise_ratio_dB = 1.4267`

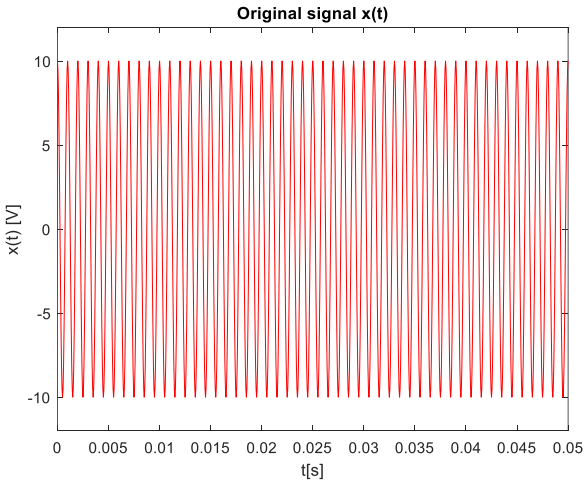




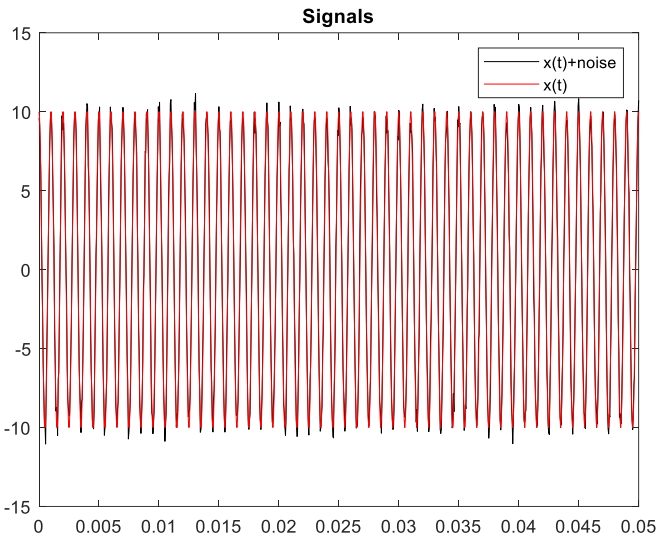
In order to eliminate the noise we use the band pass filter, and this is the output of the filter.

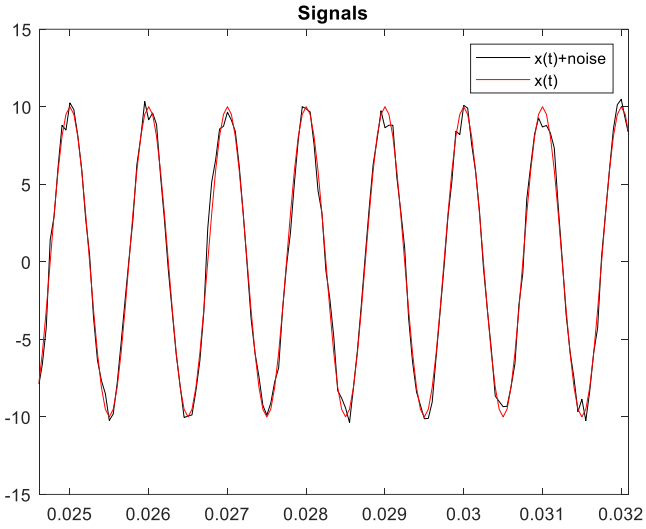


Now, let's increase the amplitude, so the power of the sine wave.

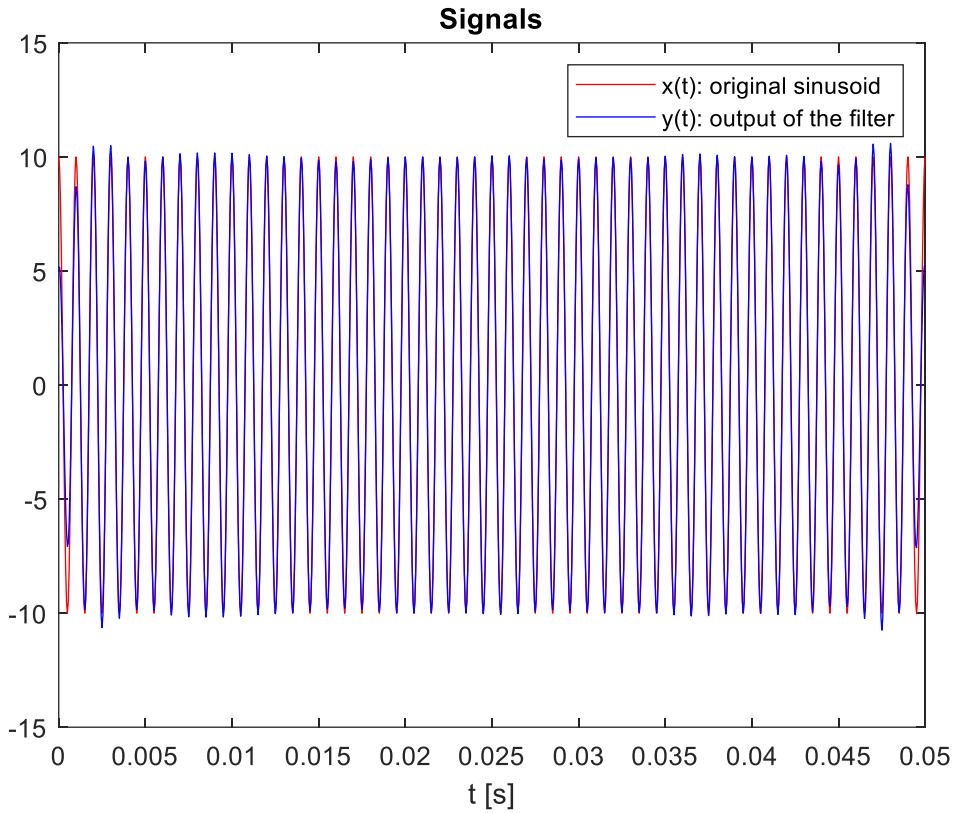


This is the plot of the signal with Amplitude A=10, so his power is `sinusoid power [V^2] =50.0000`
Also here we add to the sine wave the noise, and we can see that the noise does not affect a lot the sine wave, the black sine wave is almost equal to the red one.

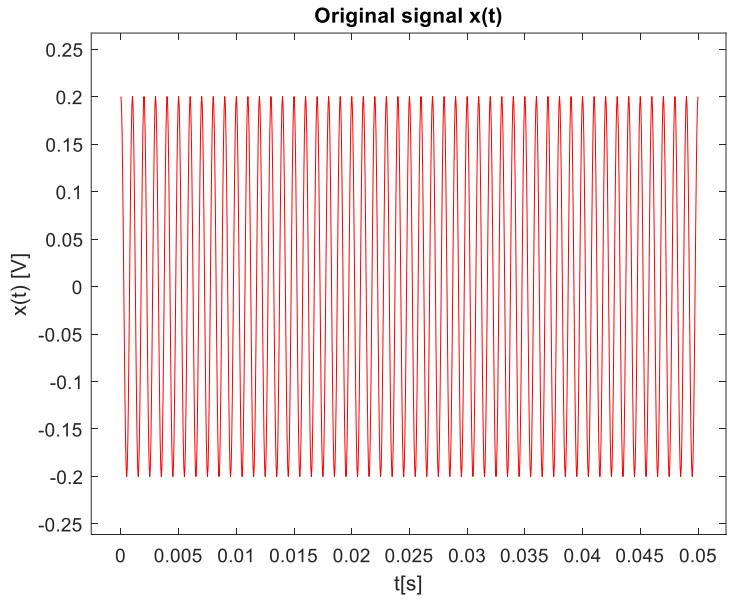




In fact, the sine wave with higher power has a `signal_to_noise_ratio_dB = 21.4267`. SNR is an important parameter that affects the performance and the quality of systems that process or transmit signals: the higher the ratio, the better the signal quality. A higher SNR means that the signal can be easier be detected or interpreted, a lower SNR means that the signal is corrupted by the noise and might be difficult to distinguish or recover.
For this reason, here we have a higher SNR than the previous one, so the signal is not corrupted as much as in case we have amplitude 1. So, we will have also the output of the filter, that attenuate the noise, almost equal to the original signal.

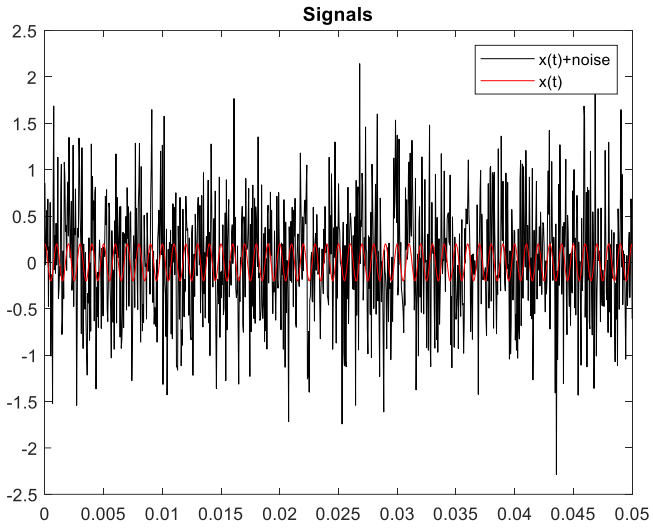


What happens if we decrease the power? It's is clear that the power is much lower and we will have a lower SNR. Let's observe a sine wave with A = 0.2.

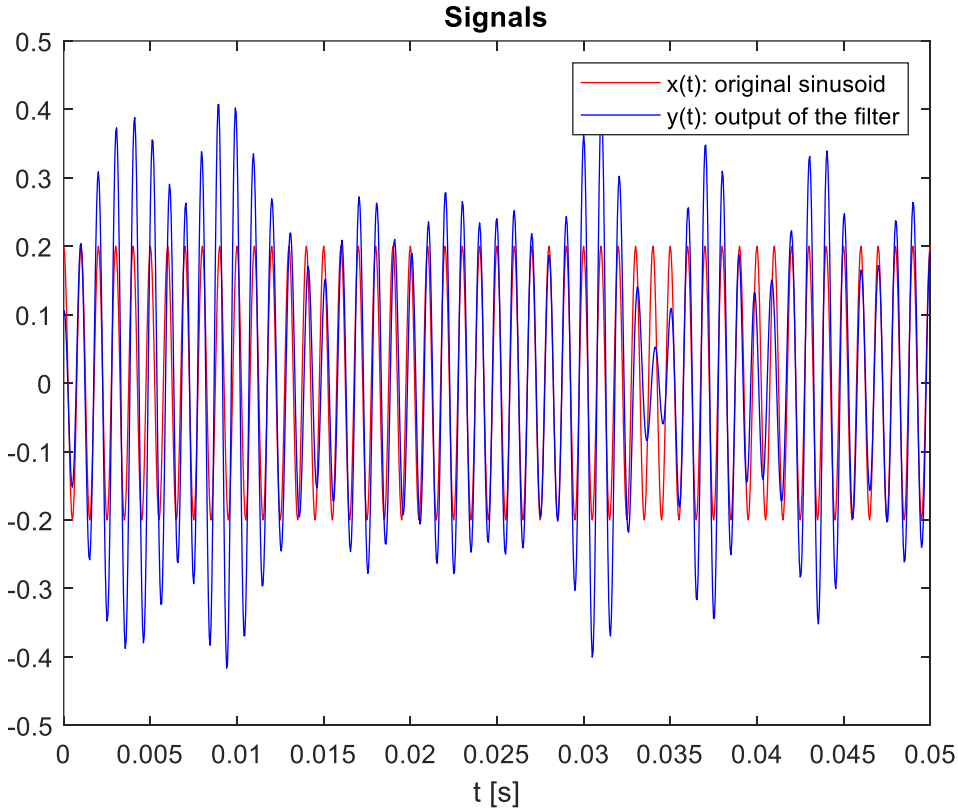


This signal has `sinusoid power [V^2] =0.02000`

Here, the effect of the noise is more evident, the signal is more disrupted by the noise. In fact, his SNR is `signal_to_noise_ratio_dB = -12.5527`, a negative value.



Also, for this reason, the output of the filter might have more differences in respect to the original signal; it is very difficult to detect the original sinusoid because the power of the noise is higher than the power of the sinusoid so we have more interference.



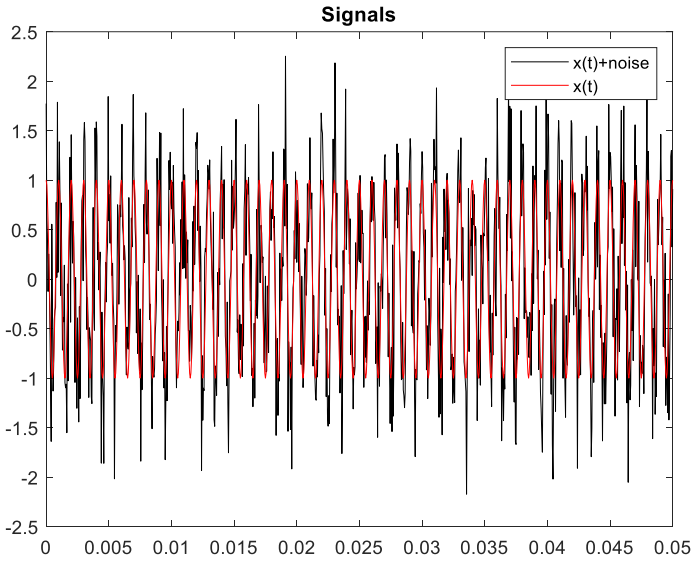
1.5 What happens if we increase the standard deviation?

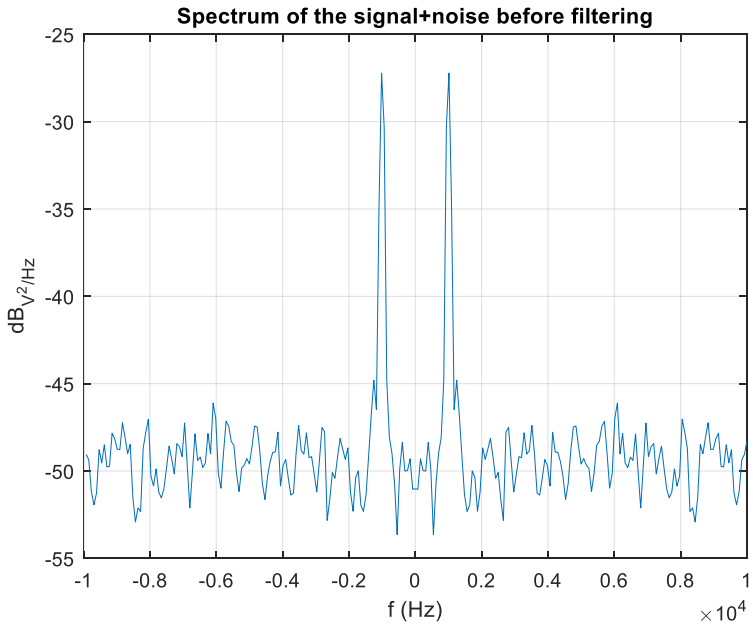
`The sigma value represents the standard deviation: a low sigma indicates that the values tend to be close to the mean or expected value of the set, instead a high`

standard deviation indicates that the values are spread out over a wider range. In fact, the standard deviation is a measure of how far the signal fluctuates from the mean and the variance represents the power of this fluctuation. Furthermore, the variance is given by sigma^2. So, increasing the standard deviation, we get a higher noise power, so the noise will corrupt more the signal.

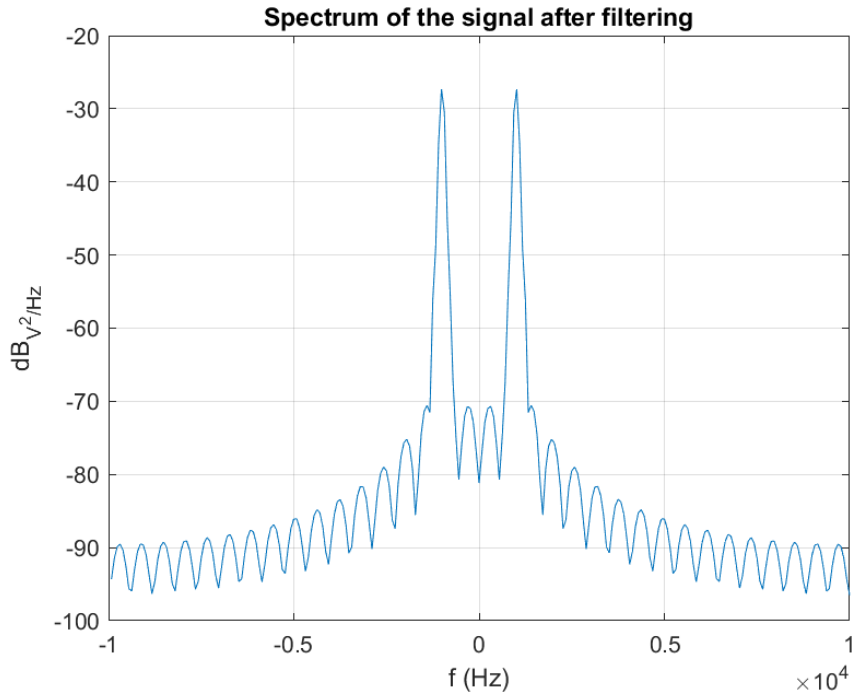
Let’s consider a signal with A=1 and sinusoid power = 0.5.

We start to consider standard deviation equal to 0.5.
σ=0.5 and we have a Noise power = 0.25. In this case we have SNR [dB]=3.010300

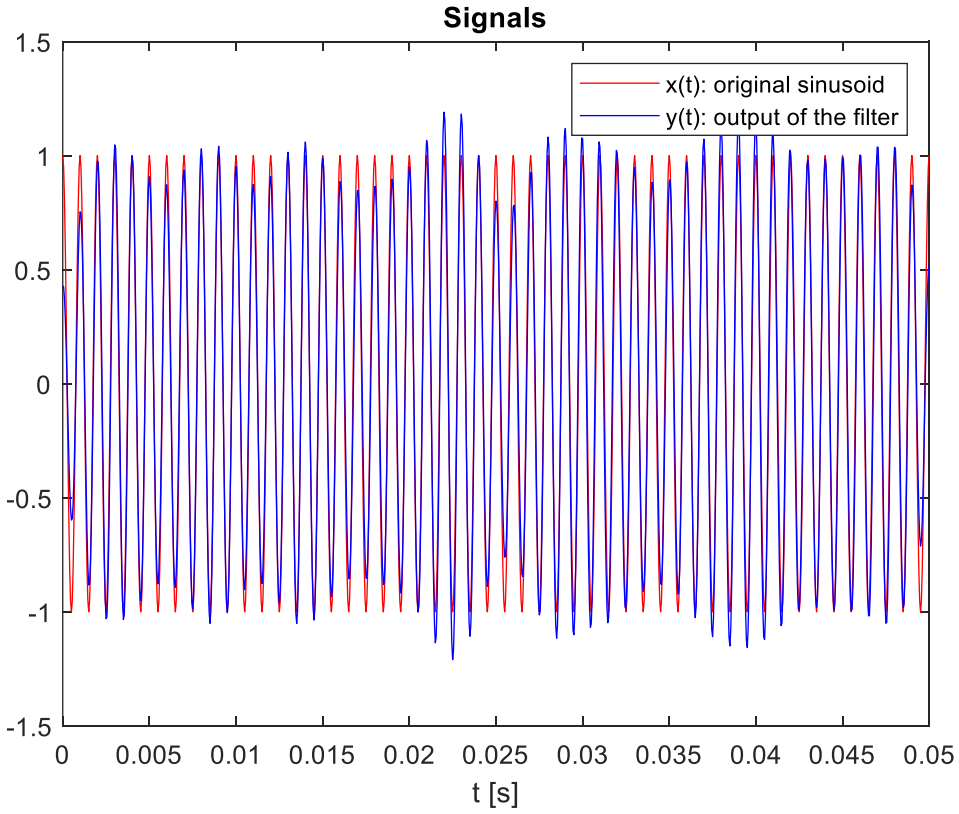




Here we can see that the floor in Spectrum plot in the two side before the filter is almost between -55 dB and -45 dB, but after filtering we will get a Spectrum like this:



We can see that the two floor of the Spectrum after the filtering becomes much lower and closer to the spectrum of the original sine wave.

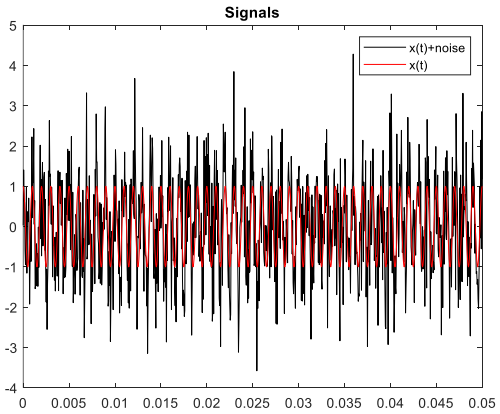


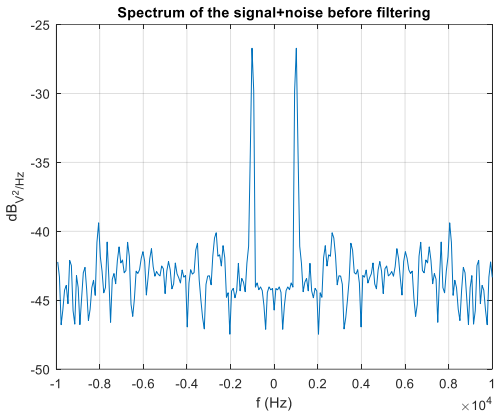
Now let’s increase the standard deviation:

σ=1

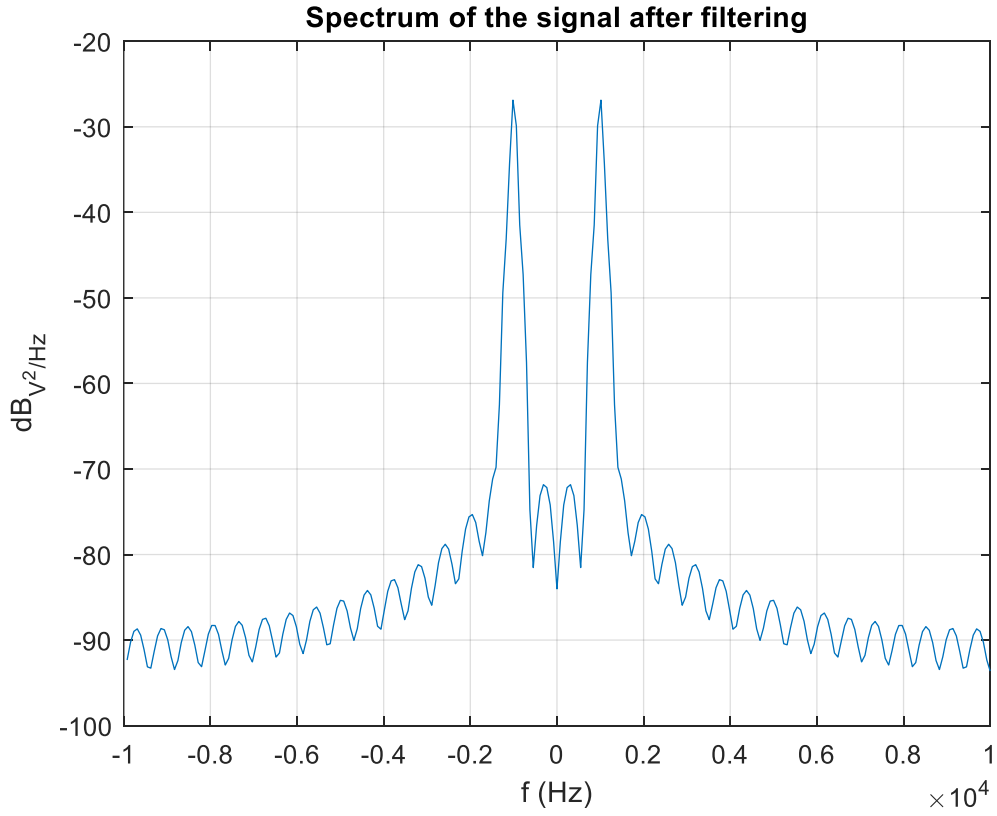
Noise power [v^2]= 1.000000

SNR[dB]=-3.0103





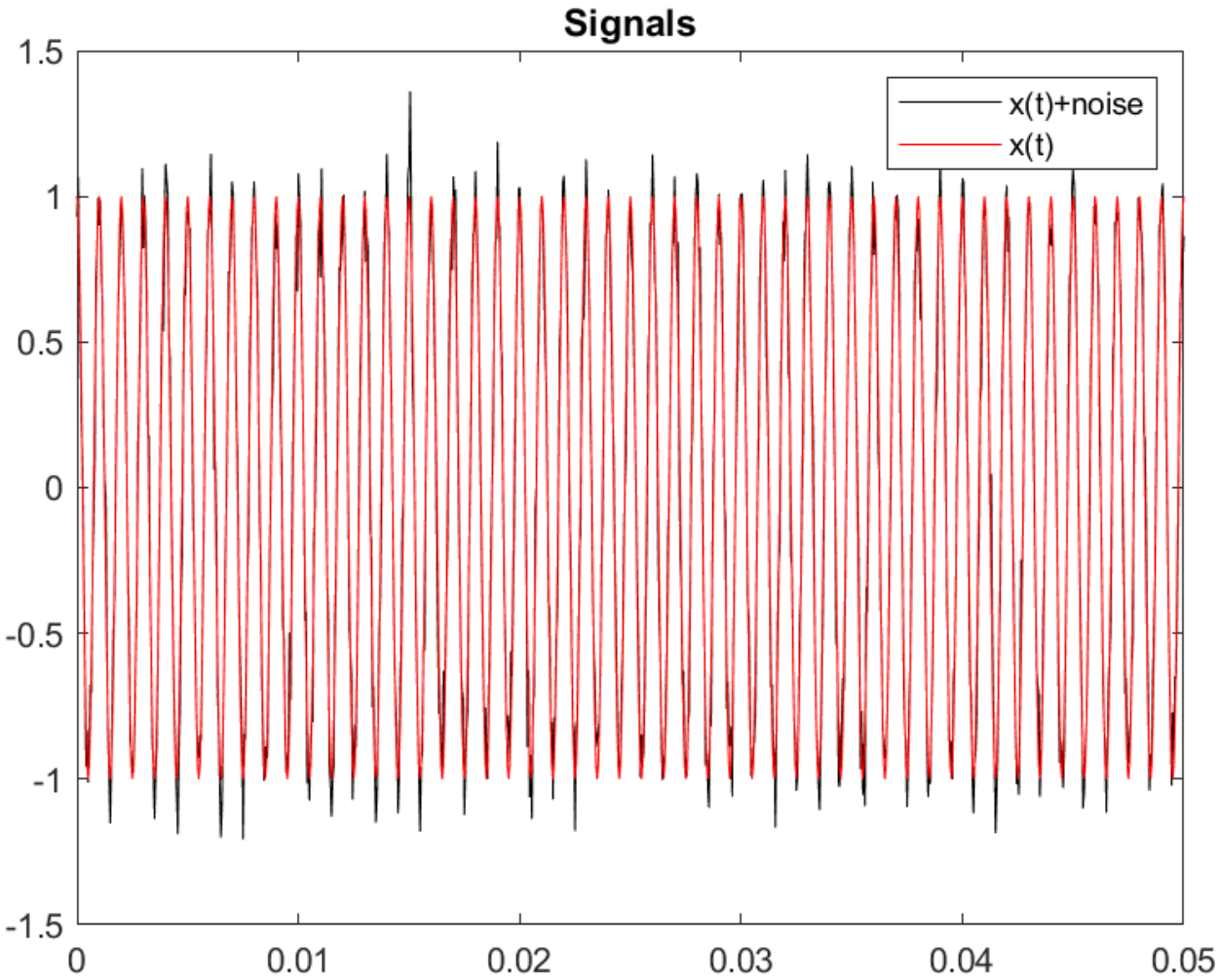
Since, we have SNR [dB]=-3.0103, the signal is more affected by the noise and the floor in the two side of the filter is almost between -47 dB and -40 dB, comparing to the previous case it is higher and more distant in respect to the Spectrum of the spectrum of the original signal. Like in the previous case, after the filtering the two floor is closer to the floor of the spectrum of the original sinusoid



Now let’s decrease the standard deviation:

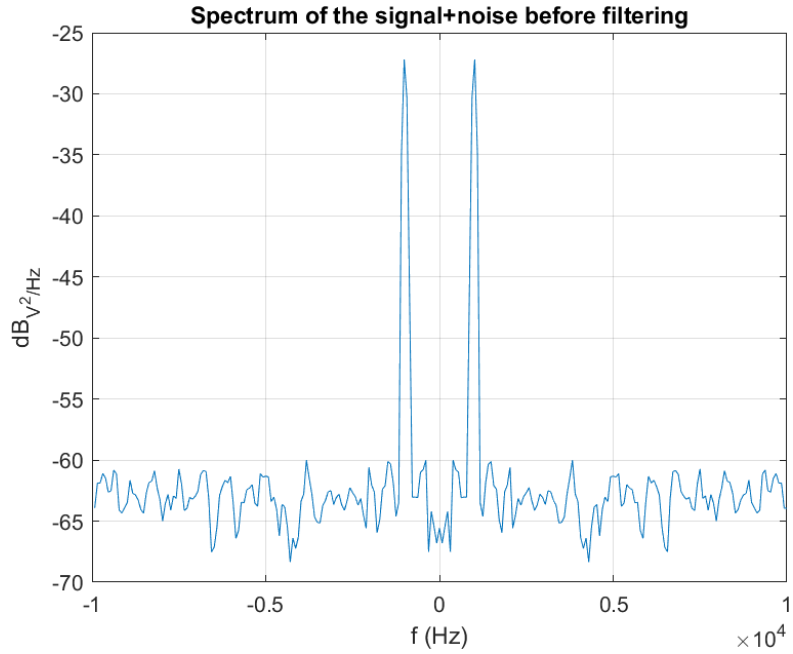
σ=0.1

```
Noise power [v^2]= 1.000000
SNR[dB]=-3.0103
```

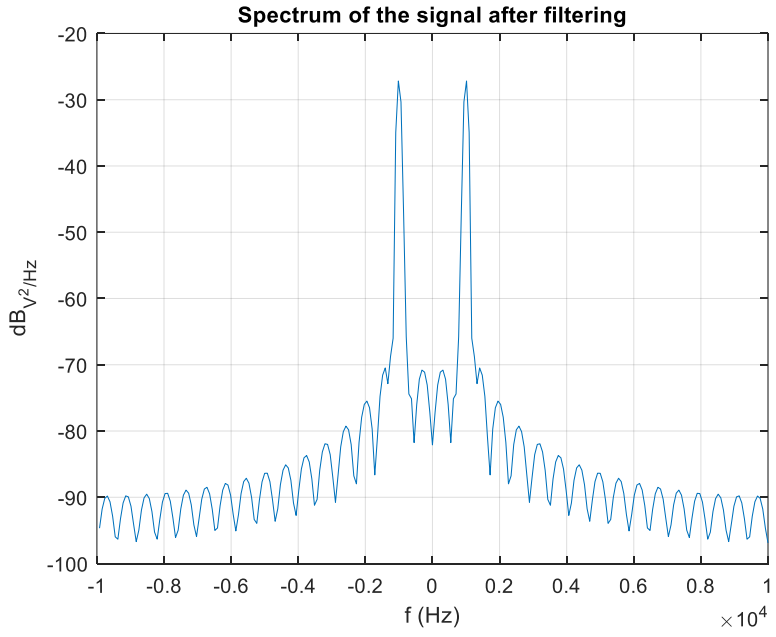


The more the standard deviation is close to 0, the same is the interference of the noise to the signal. Here, we can in fact, neglect the effect of the noise. We almost have the same signal.

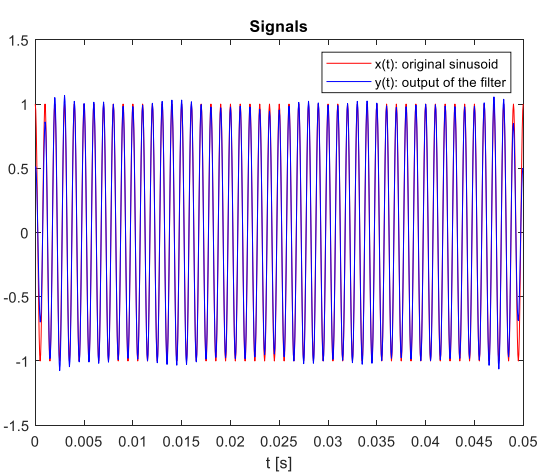
We can also see that the two floor in the Spectrum of the signal with noise before filtering is much lower than the previous two case in which we can higher standard deviation, so higher noise, the effect of the noise is more evident. Here, these two floors have values between -67 dB and -60 db.

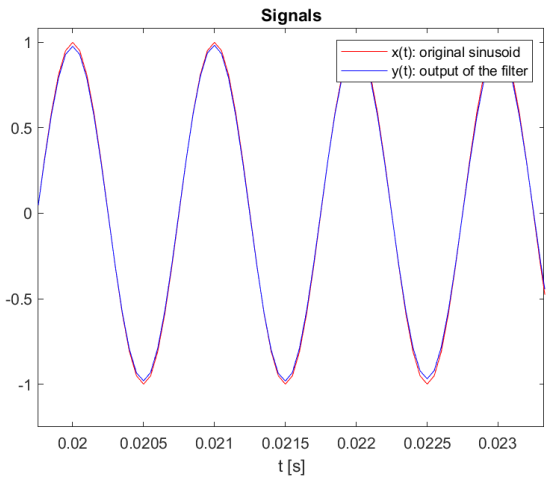


After the filtering, we have this plot for the Spectrum, the difference between the values in the floor of the Spectrum before and after filtering is just -20 to -30 db.



Since, the noise is not so evident in this case we get after the filtering almost the same sinusoid, it is much simpler to detect the original sinusoid for the filter.



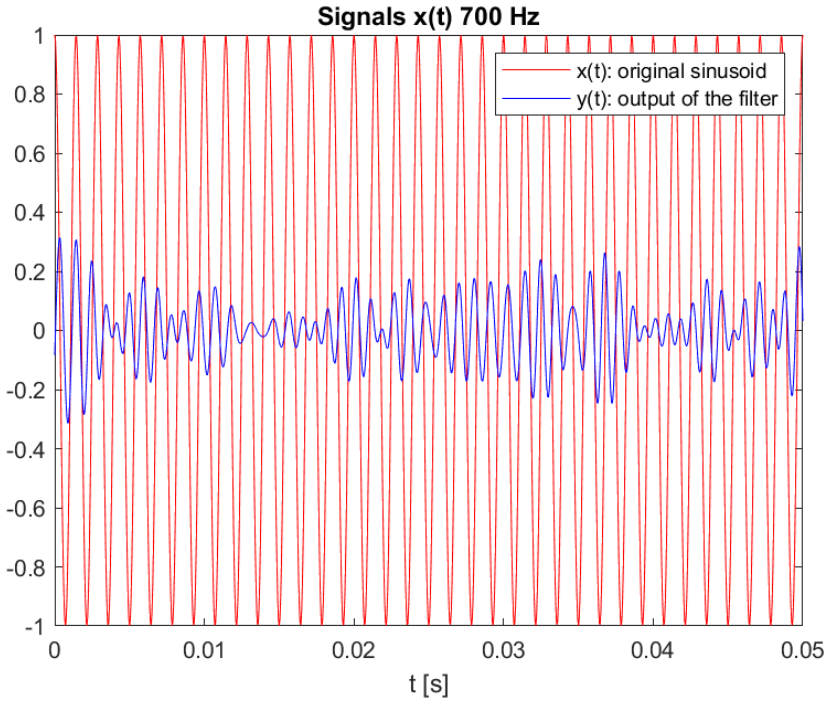


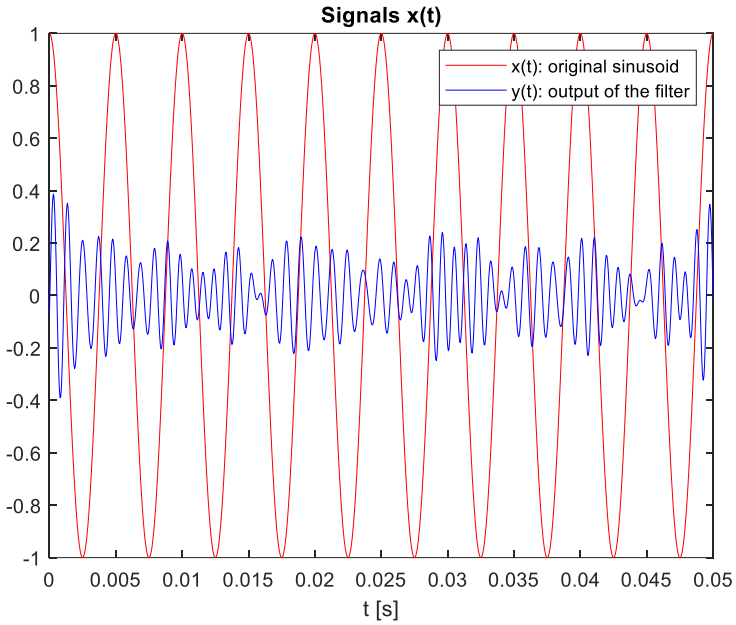
1.6 For sine wave, choose a frequency that does not fall within the passband of the filter and observe the outputs.

As we know, a band pass filter is a device that passes frequencies within a certain range and rejects frequencies outside that range.

Here, we have considered the higher cutoff frequency (Fpass2) equal to 1200Hz and the lower cutoff frequency (Fpass1) equal to 800Hz.

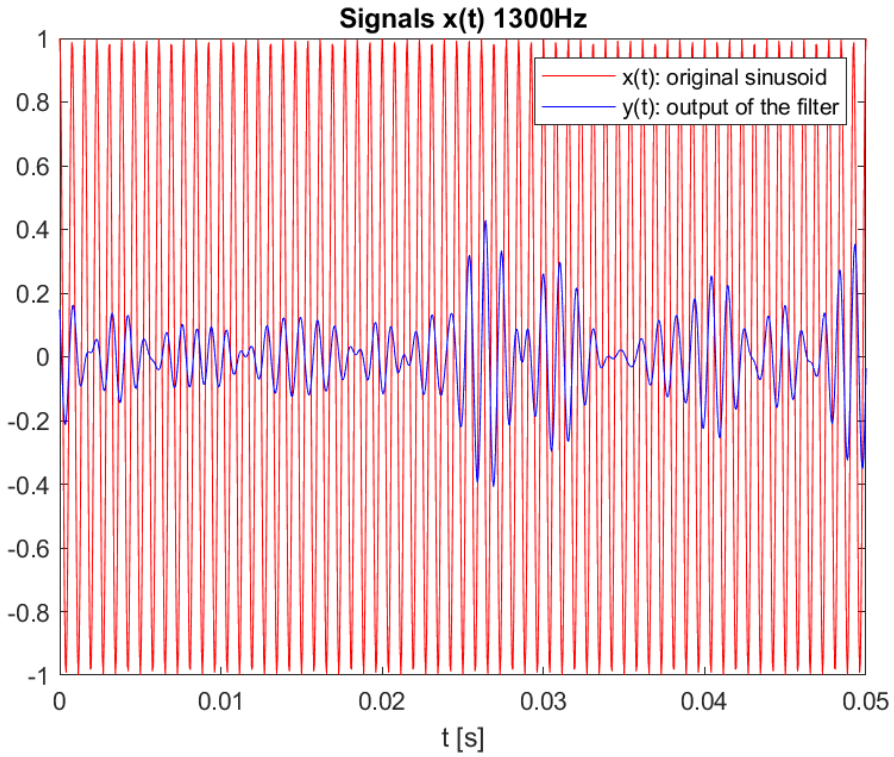
If now, we choose a sinusoid with a frequency higher than 1200 and lower than 800, it does not fall within the passband of the filter.

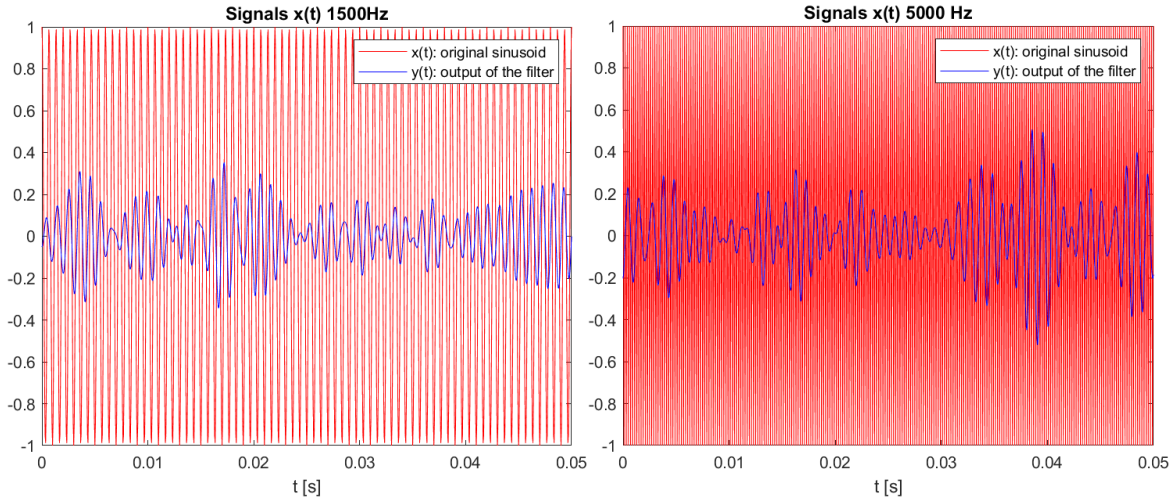




Here, is the output of bandpass filter of the sine wave with frequency equal to 700 plus noise, the blue sinusoid is our output and it is clearly not correctly filtered by the filter, we get a very different signal in respect to the original sinusoid.

The same result we will get also when we consider the frequency of the original sinusoid a value higher than 1200 Hz




Signals x(t) 1500Hz
x(t): original sinusoid
y(t): output of the filter
t [s]
Signals x(t) 5000 Hz
x(t): original sinusoid
y(t): output of the filter
t [s]

2. Activity2: QAM modulation and Demodulation

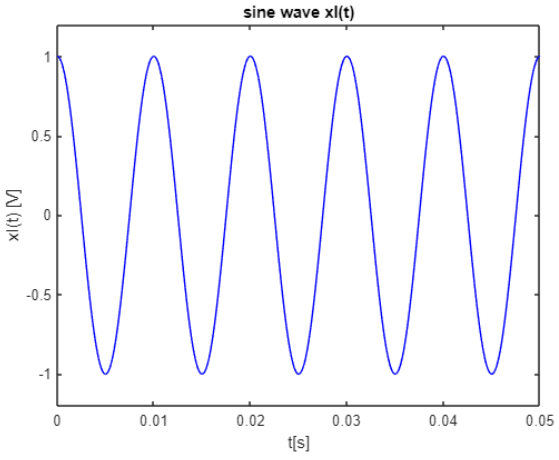
```
close all % close all plots
clear all % clear all variables
clc % clear the screen

fs=30000; % sampling frequency
A=1; % amplitude of the modulating in-phase and quadrature sine waves
T=0.05; % duration (sec) of the in-phase and quadrature sine waves
f1=100; % frequency of the modulating sine wave xI in the in-phase path
f2=300; % frequency of the modulating sine wave xQ in the quadrature path
fc=3000; % carrier frequency
```

2.1 Generation of the sine waves

Now I'll generate and plot the modulating sine waves xI and xQ by means of the SinusoidalSource_2023 function: these two sine waves have amplitude A, frequency f1 for wave xI and frequency f2 for wave xQ, the duration is T and sampling frequency fs.

Firstly, I'll generate the sinusoid xI:

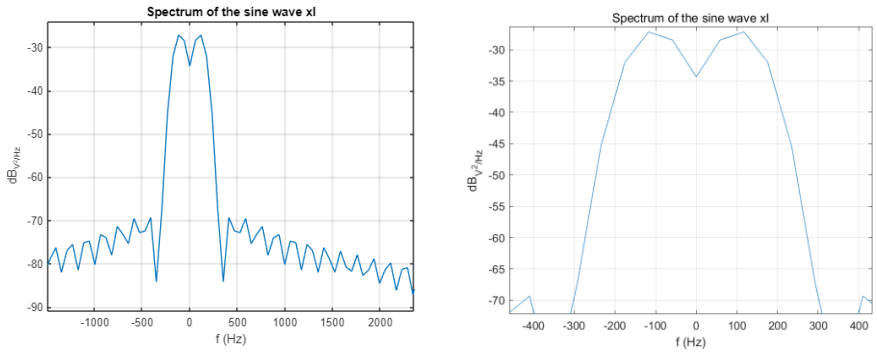
```
[t,xI,N]=SinusoidalSource_2023(A,f1,T,fs); % generation of the signal

% N = Number of samples of the signal
% t = Sampled time axis
% xI is the wave generated
figure
plot(t,xI,"b")
xlabel("t[s]")
ylabel("xI(t) [V]")
title("sine wave xI(t)")
axis([min(t) max(t) 1.2*min(xI) 1.2*max(xI)])
```



```
signal_power=0.5*A^2;
fprintf('sinusoid power [V^2]=%f',signal_power)
```

```
sinusoid power [V^2] =0.500000
```

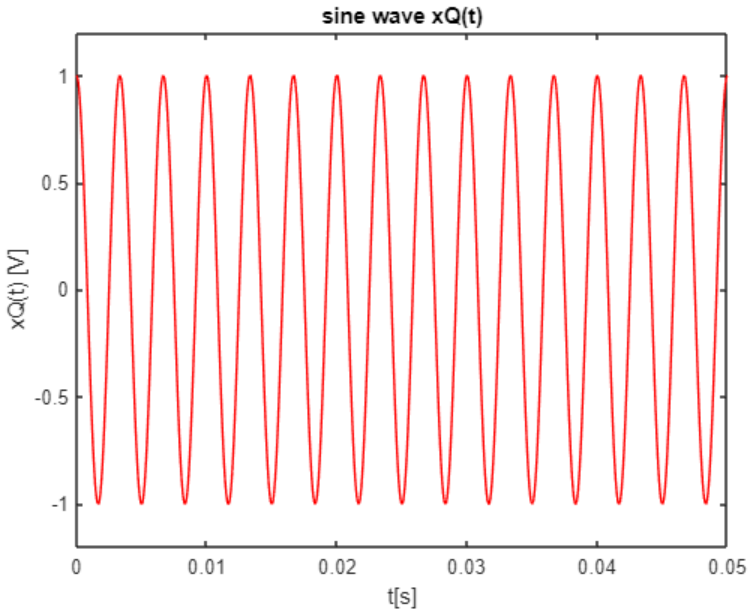
```
figure
PlotSpectrum(xI,fs);
title('Spectrum of the sine wave xI');
```



This plot above represents the spectrum of the sine wave xI, the spectrum of the sine wave shows that power is concentrated -100Hz and 100Hz.

Now, I am going to create the sinusoid xQ:

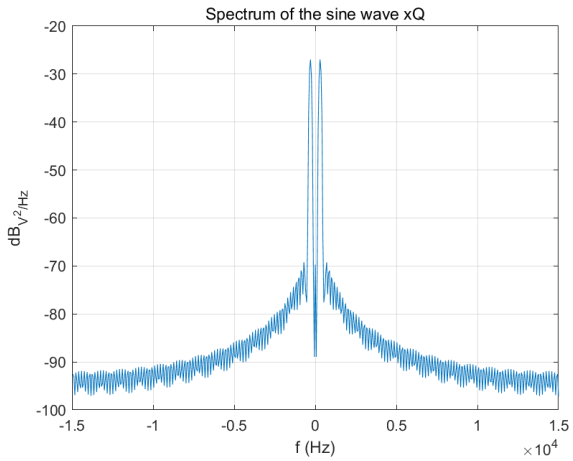
```
[t,xQ,N]=SinusoidalSource_2023(A,f2,T,fs); % generation of the signal

% N = Number of samples of the signal
% t = Sampled time axis
% xI is the wave generated
figure
plot(t,xQ,"r")
xlabel("t[s]")
ylabel("xQ(t) [V]")
title("sine wave xQ(t)")
axis([min(t) max(t) 1.2*min(xQ) 1.2*max(xQ)])
```

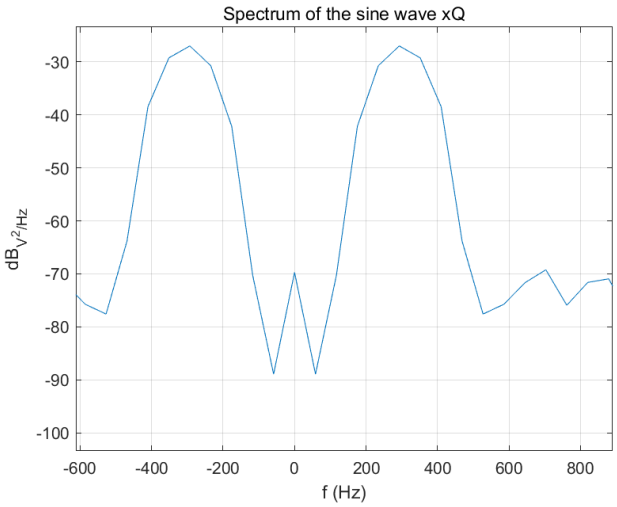



```
signal_power=0.5*A^2;
fprintf('sinusoid power [V^2]=%f',signal_power)
```

```
sinusoid power [V^2]=0.500000
```

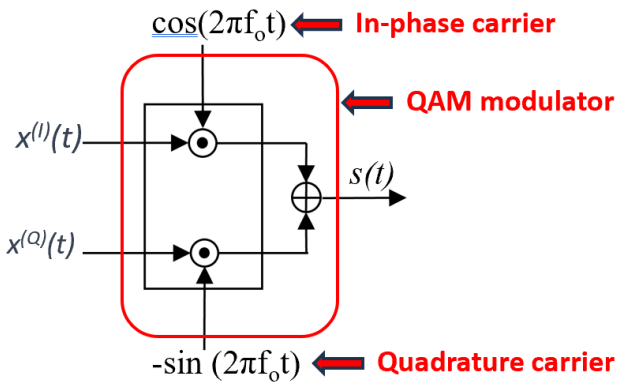
```
figure
PlotSpectrum(xQ,fs);
title('Spectrum of the sine wave xQ');
```



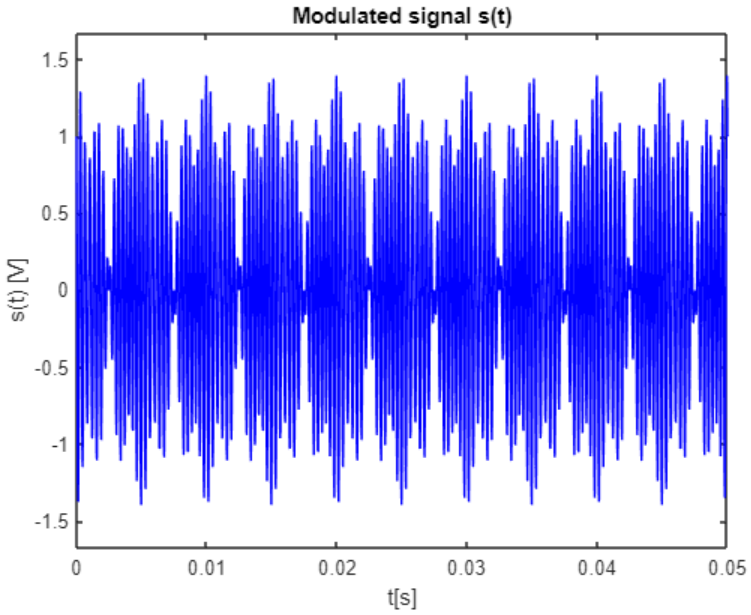


Here, instead,the spectrum of xQ shows that the power is mostly concentrated in -300Hz and 300Hz.

2.2 QAM modulation xI and xQ with carrier frequency



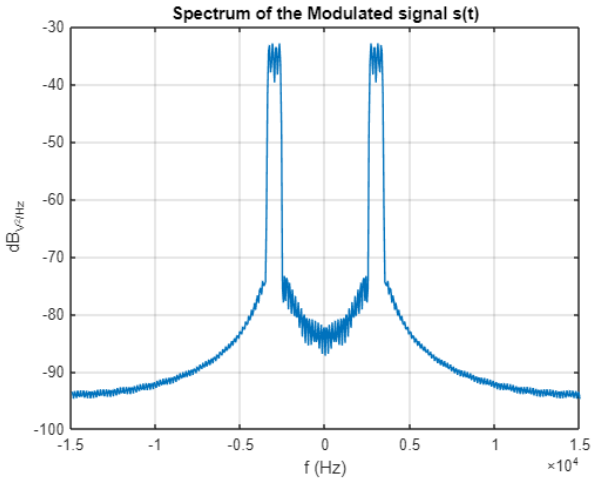

```
 s = ModQAM_2023(xI,xQ,fc,T,fs); % modulate the signal with xI and xQ in carrier
frequency fc
 figure
 plot(t,s,"b")
 xlabel("t[s]")
 ylabel("s(t) [V]")
 title("Modulated signal s(t)")
 axis([min(t) max(t) 1.2*min(s) 1.2*max(s)])
```

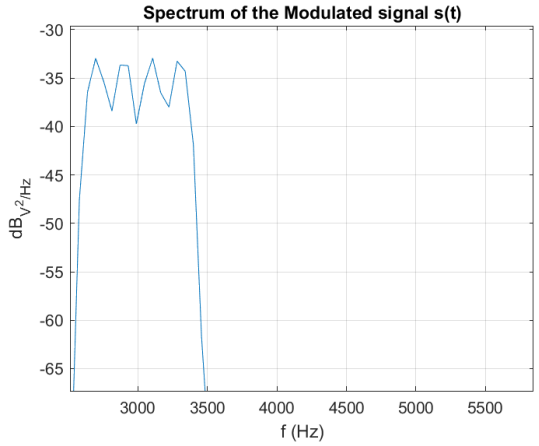



```
 signal_power=0.5*A^2;
 fprintf('sinusoid power [V^2]=%f',signal_power)
```

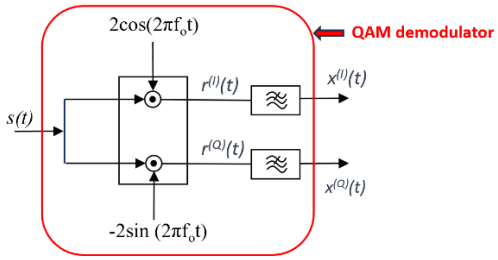
```
sinusoid power [V^2]=0.500000
```

```
figure
PlotSpectrum(s,fs);
title('Spectrum of the Modulated signal s(t)');
```



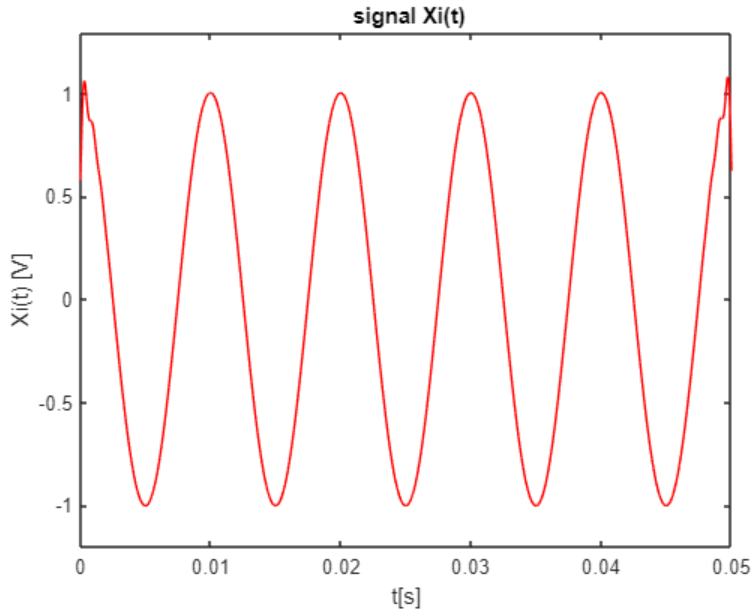


2.3 QAM demodulation



This demodulation function, given at the input the samples of the QAM signal s with carrier frequency fc, operates as QAM demodulator providing the samples of the corresponding in-phase and quadrature signals Xi and Xq. The variable psi represents the phase offset of the carrier used by the demodulator relative to the carrier used by the modulator. In the ideal case it is psi=0. As optional output it shall provide also the delay introduced by the low-pass filter in terms of number of samples.

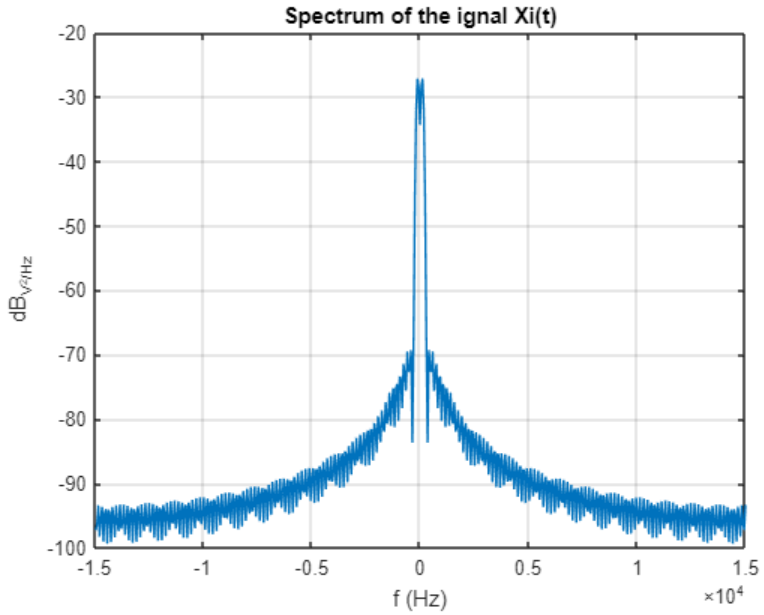
```
psi=0; % the ideal case, without offset;
[Xi,Xq,Delay]=DeModQAM_2023(s,fc,T,fs,psi); % demodulation of the signal
figure
plot(t,Xi,"r")
xlabel("t[s]")
ylabel("Xi(t) [V]")
title(" signal Xi(t)")
axis([min(t) max(t) 1.2*min(Xi) 1.2*max(Xi)])
```



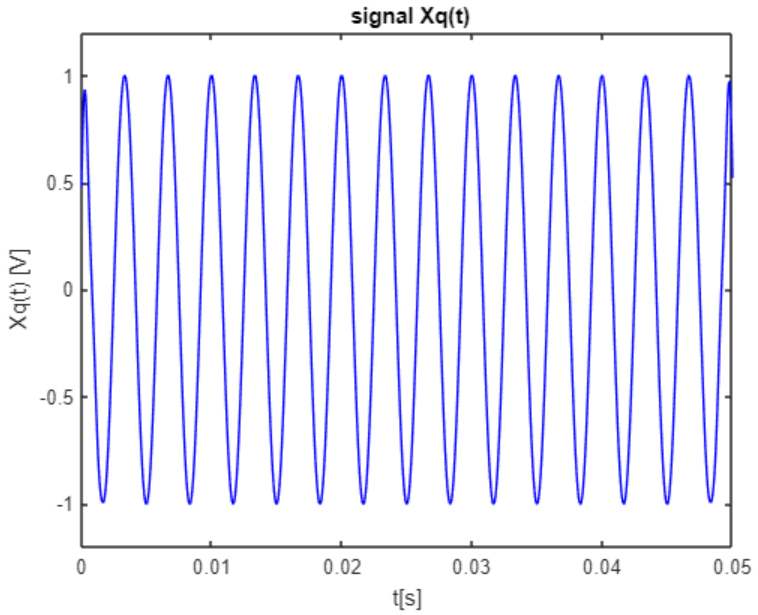

```
signal_power=0.5*A^2;
fprintf('sinusoid power [V^2]=%f',signal_power)
```

```
sinusoid power [V^2]=0.500000
```

```
figure
PlotSpectrum(Xi,fs);
title('Spectrum of the ignal Xi(t)');
```



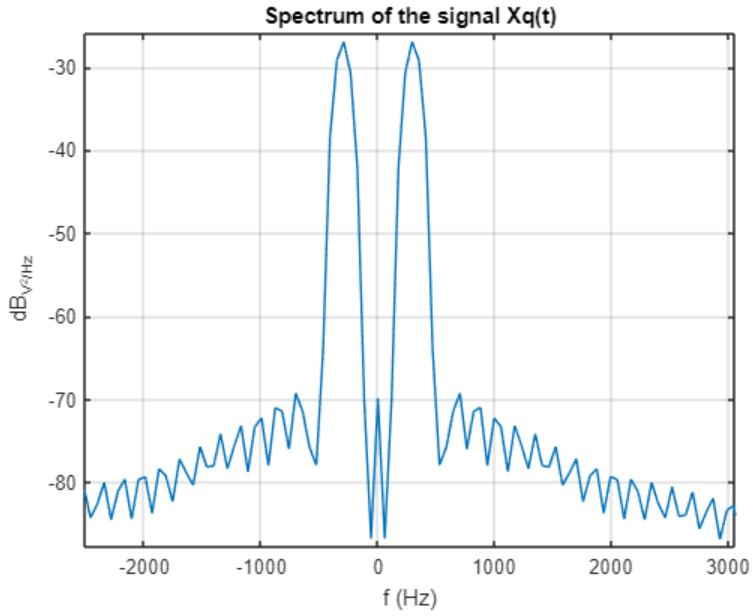
```
figure
plot(t,Xq,"b")
xlabel("t[s]")
ylabel("Xq(t) [V]")
title(" signal Xq(t)")
axis([min(t) max(t) 1.2*min(Xq) 1.2*max(Xq)])
```



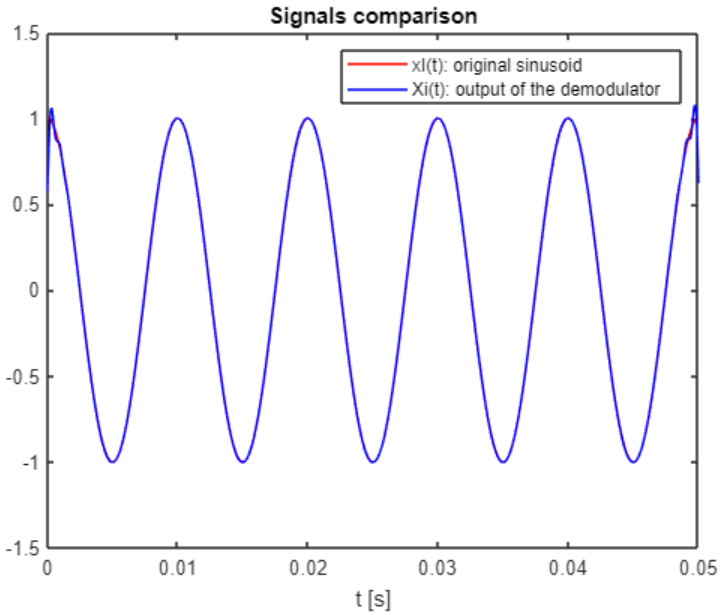

```
signal_power=0.5*A^2;
fprintf('sinusoid power [V^2]=%f',signal_power)
```

```
sinusoid power [V^2]=0.500000
```

```
figure
PlotSpectrum(Xq,fs);
title('Spectrum of the signal Xq(t)');
```




```
figure
plot(t,xI,"r")
hold on
plot(t,Xi,'b')
xlabel('t [s]')
legend ('xI(t): original sinusoid','Xi(t): output of the demodulator')
title('Signals comparison');
```



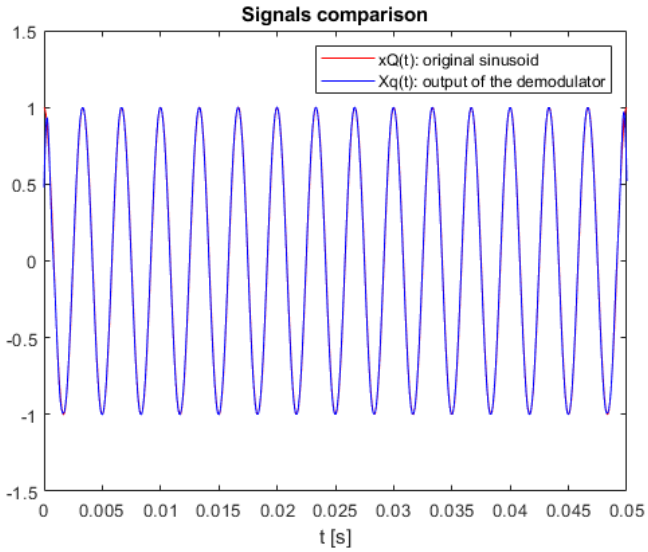
Here, I have made a comparison between the signal at the output of the demodulation function and the signal xI that I have created at the beginning with the function

SinusoidalSource_2023; as we can see the two functions are the same, since here we have considered the ideal case, with offset zero.

$$r^{(I)}(t) = x^{(I)}(t)\cos(\varphi) + x^{(Q)}(t)\sin(\varphi)$$

At the output of the demodulator, for the in-phase component we have only contribution from this component $x^{(I)}(t)\cos(\varphi)$, since the term $sin\ (0)$ will eliminate the quadrature component.

```
figure
plot(t,xQ,"r")
hold on
plot(t,Xq,'b')
xlabel('t [s]')
legend ('xQ(t): original sinusoid','Xq(t): output of the demodulator')
title('Signals comparison');
```



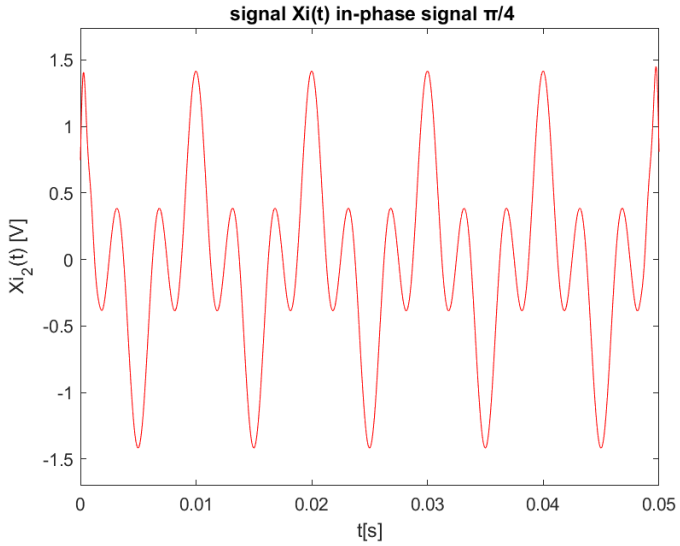
Also here, comparing the result Xq of demodulator and the original signal xQ created with the function SinusoidalSource_2023, as considering offset zero, the two waves coincide.

$$r^{(Q)}(t) = -x^{(I)}(t)\sin(\varphi) + x^{(Q)}(t)\cos(\varphi)$$

Here, $sin(0)$ will eliminate the in-phase component so the result of the demodulation gives us the quadrature component that coincided with the original xQ sinusoid.

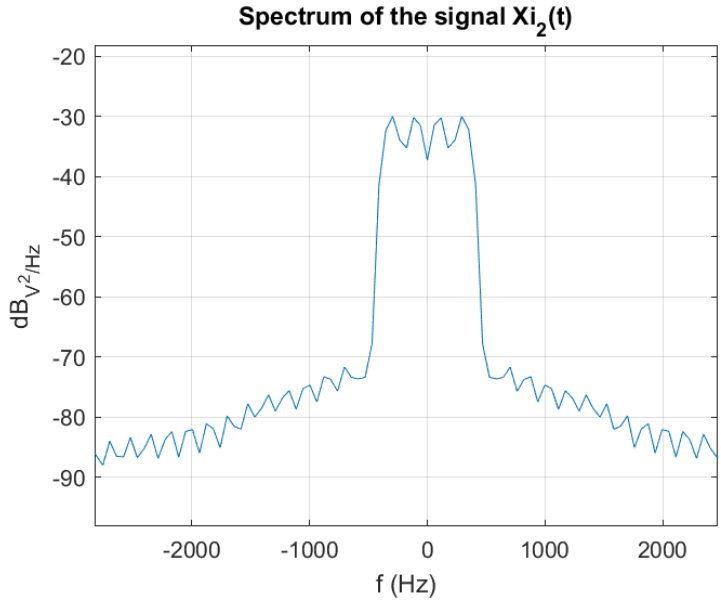
2.4 If the offset is not zero and it is $\boldsymbol{\frac{\pi}{4}}$

In case we have the offset between the carrier phases of the transmitter and the receiver equal to $\boldsymbol{\frac{\pi}{4}}$ then we get the in-phase signal like this

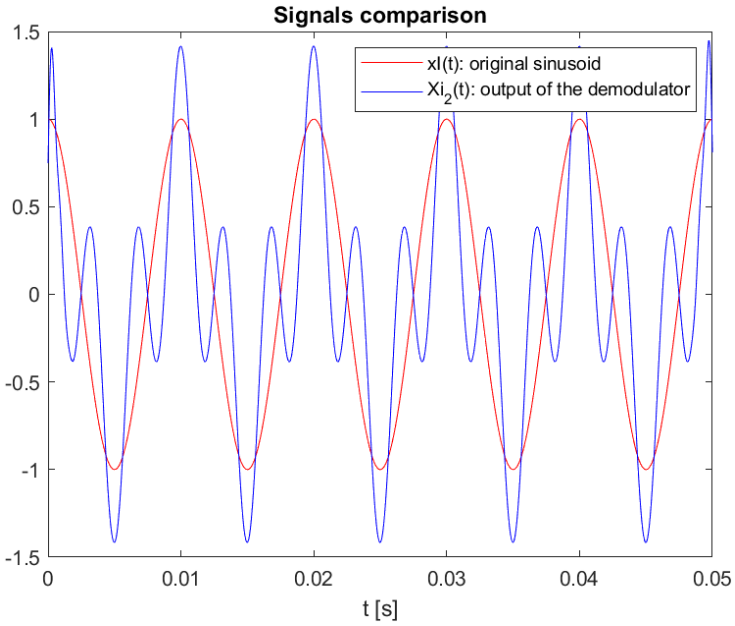


We have here not only the contribution of the in-phase part of s(t) but also the quadrature part of the s(t). In fact, from the mathematical point of view, the result of demodulation of the signal s(t) is $r^{(I)}(t) = x^{(I)}(t)\cos(\varphi) + x^{(Q)}(t)\sin(\varphi)$

So in case , φ is zero, we only have in-phase contribution, but in this case we have also the quadrature contribution, because $\sin\frac{\pi}{4} = \frac{1}{\sqrt{2}}$ and $\cos\frac{\pi}{4} = \frac{1}{\sqrt{2}}$, we have both contribution of in-phase and quadrature part of the signal s(t).



We have in the Spectrum four peak instead of two, in fact we have two signal that contribute in Xi(t).

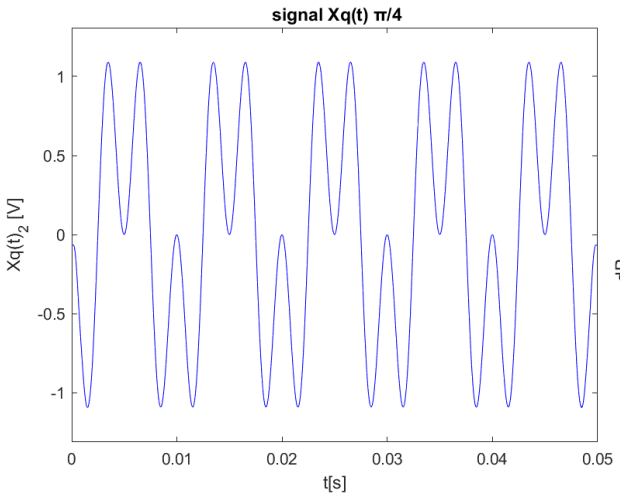


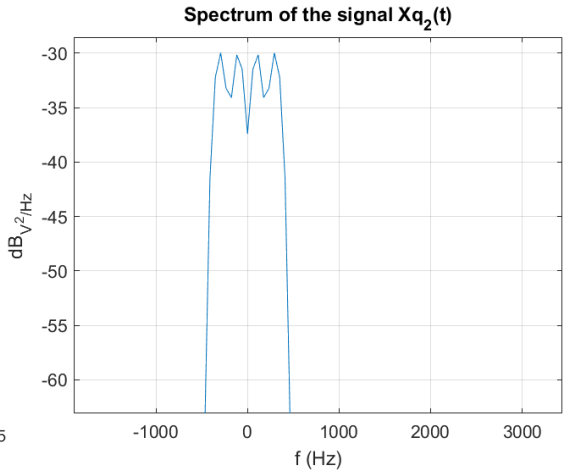
The same is for the quadrature case, we have here not only the contribution of quadrature the part of s(t) but also the in-phase part of the s(t). In fact, from the mathematical point of view, the result of demodulation of the signal s(t) is

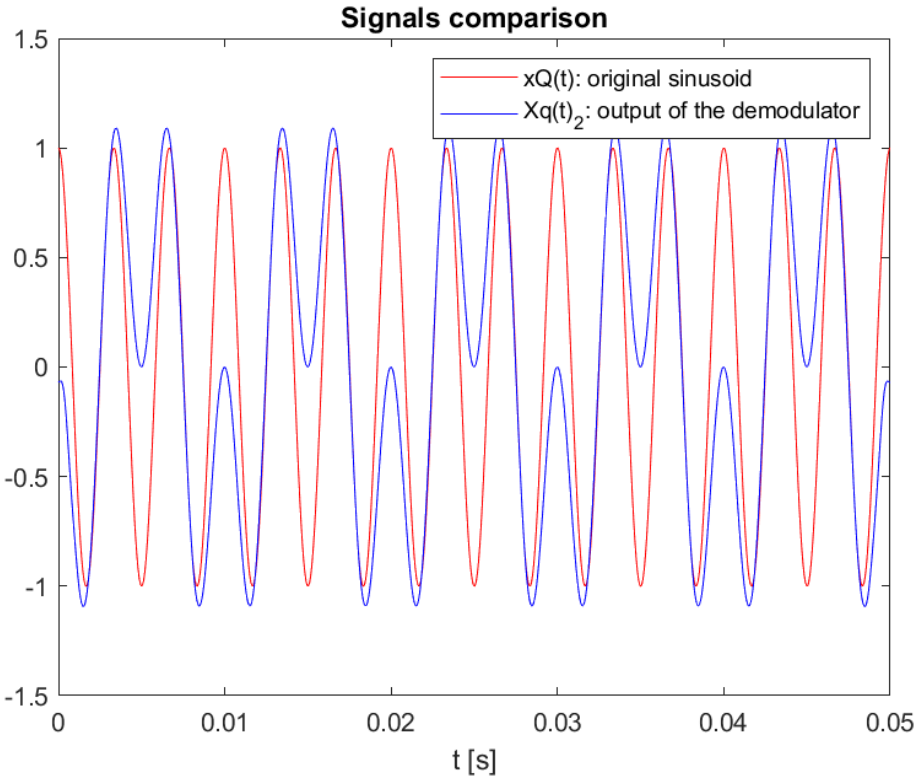
$$r^{(Q)}(t) = -x^{(I)}(t)\sin(\varphi) + x^{(Q)}(t)\cos(\varphi)$$

$\sin\frac{\pi}{4} = \frac{1}{\sqrt{2}}$ and $\cos\frac{\pi}{4} = \frac{1}{\sqrt{2}}$,

We have both contribution of in-phase and quadrature part of the signal s(t).

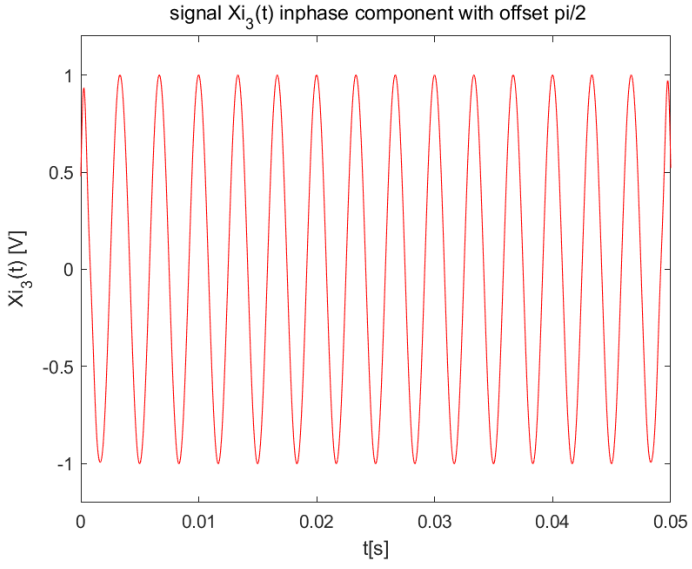




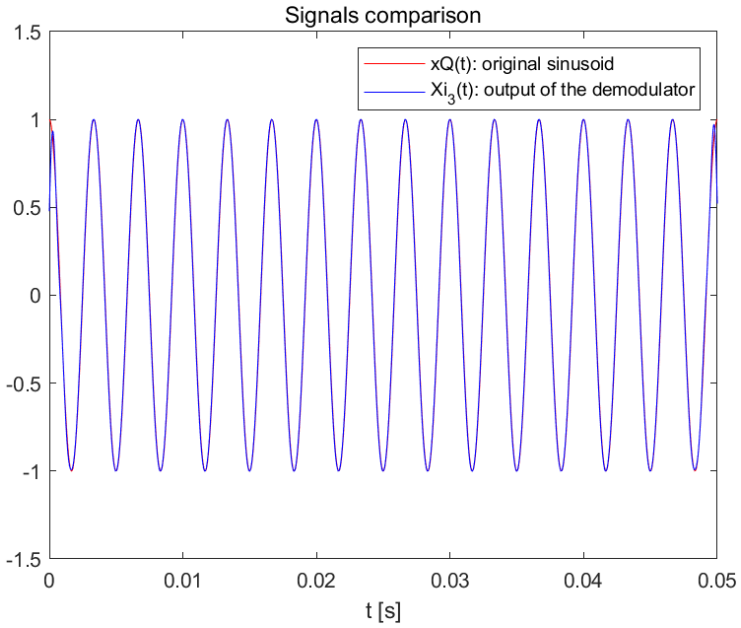


2.5 If the offset is not zero and it is $\frac{\pi}{2}$

In case we have the offset between the carrier phases of the transmitter and the receiver equal to $\frac{\pi}{2}$ then we get the in-phase signal like this :

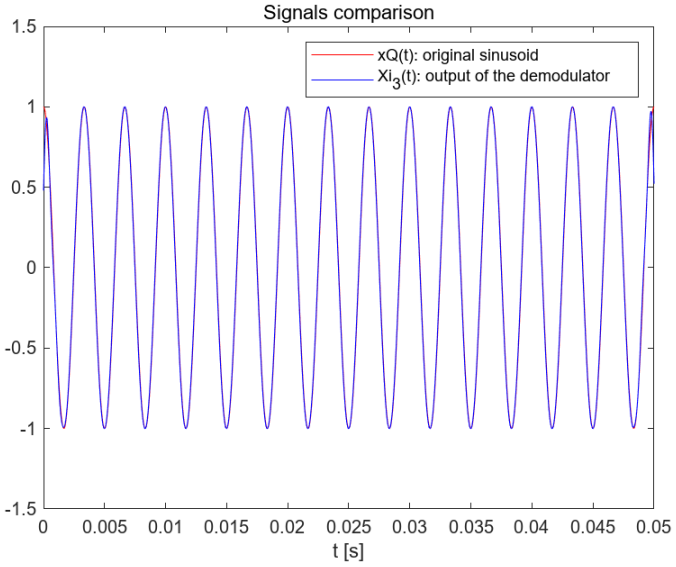


And its quadrature signal like this:

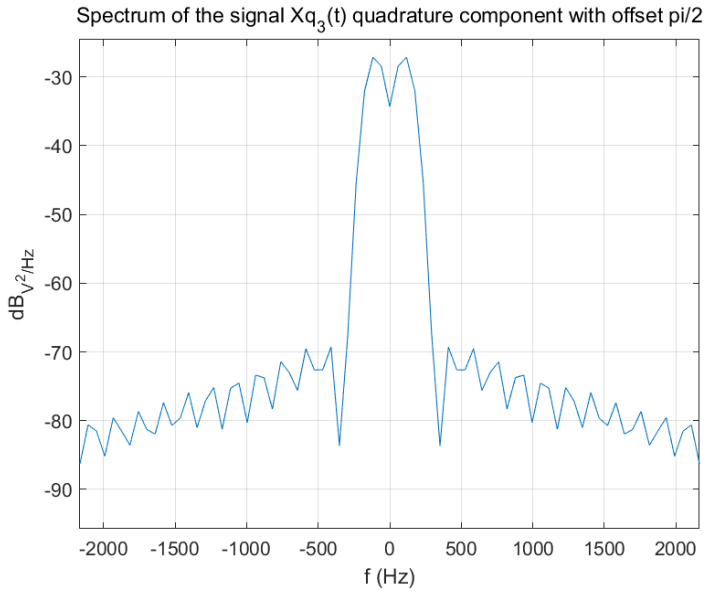


In in-phase signal:

$r^{(I)}(t) = x^{(I)}(t)\cos(\varphi) + x^{(Q)}(t)\sin(\varphi)$, since the $sin\left(\frac{\pi}{2}\right) = 1$ and $\cos\left(\frac{\pi}{2}\right) = 0$, in Xi(t), the in-phase result of the demodulation function, we have only a quadrature contribution, in fact, as we can see below the in-phase signal Xi coincides with the original quadrature sinusoid.

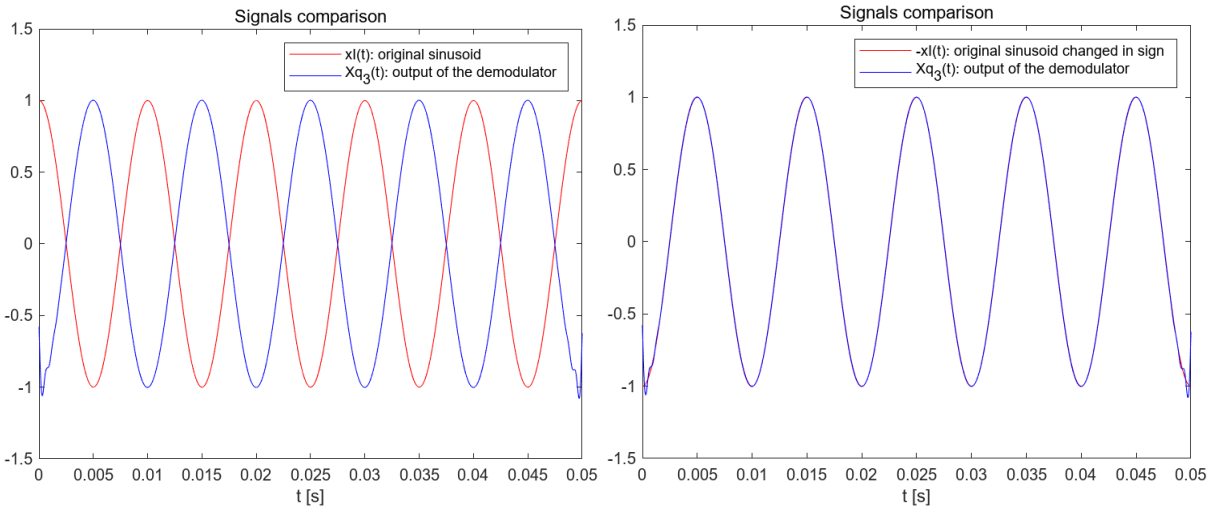


Since here we have for the in-phase signal only contribution of one signal (quadrature one) , we have the spectral of XI coinciding with the spectrum of original xQ wave.

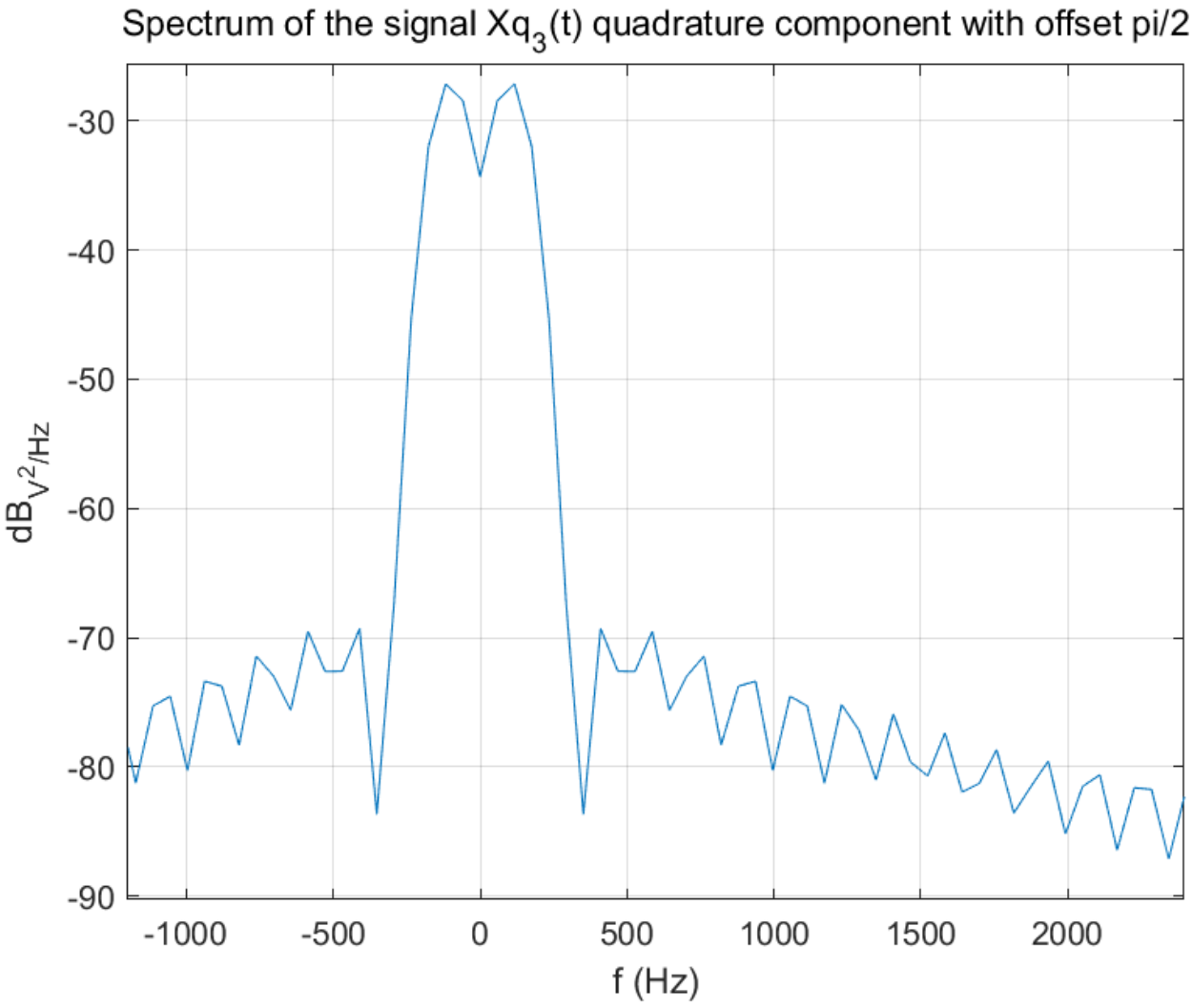


In the quadrature signal, $r^{(Q)}(t) = -x^{(I)}(t)\sin(\varphi) + x^{(Q)}(t)\cos(\varphi)$, since the $sin\left(\frac{\pi}{2}\right) = 1$ and $\cos\left(\frac{\pi}{2}\right) = 0$, we have only contribution from the in-phase component, but inverted in sign.

In fact the quadrature signal that we get from the demodulation function is exactly equal to the original in-phase component changed in sign.



As we have only contribution from one signal, in the spectral plot of Xq we have only two peaks, like in the spectral plot of original in-phase wave:


Spectrum of the signal Xq3(t) quadrature component with offset pi/2
dBV2/Hz
-30
-40
-50
-60
-70
-80
-90
-1000
-500
0
500
1000
1500
2000
f (Hz)

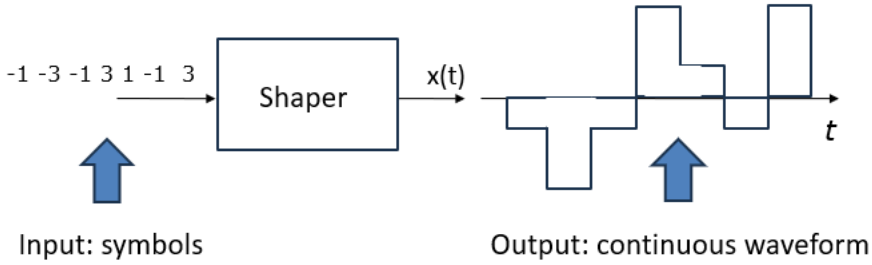
3. PAM Modulator

In communication, we often associate bits with symbols and to be transmitted, symbols must be associated with a continuous waveform, namely, a PAM signal *x(t).* The PAM signal is generated starting from the **symbols** and a **basic pulse** g(t).

$$x(t) = \sum_k x_k g(t - kT)$$

symbol

In order to do this operation, we need a Pulse Shape:



Pulse shaping is the process of changing a transmitted pulses' waveform to optimize the signal for its intended purpose or the communication channel. This is often done by limiting the bandwidth of the transmission and filtering the pulses to control intersymbol interference.

Using the function **PAMModulator_2023(symbols,nsps,Nf,pulsetype,rolloff)** we generate our PAM signal x(t) corresponding to the symbols provided in the vector **symbols**. The parameter **Nf** represents the number of samples of the basic pulse used to generate the PAM signal.

The parameter **pulse type** can be equal to

- 'RECT', to use a rectangular pulse.
- 'ROOTRAISEDCOSINE' to use a root raised cosine pulse. The **roll-off** factor ∈ [0,1] must be introduced.

The output PAM signal shall have **nsps** samples per symbol. The second output parameter **Delay** indicates delay introduced by the pulse shaping filter.

```
clc
clear all
```

3.1 ROOT-RAISED-COSINE PULSE TYPE

Here we are going to generate a PAM signal using a Root raised cosine pulse type. The raised-cosine filter has a smoother frequency response compared to the rectangular pulse filter, which helps in reducing intersymbol interference ISI in the communication channel. And the non-zero

portion of the frequency spectrum of its simplest form (Rolloff=1) is a cosine function, 'raised' up to sit above the frequency (horizontal) axis.

We define a vector of complex symbols:

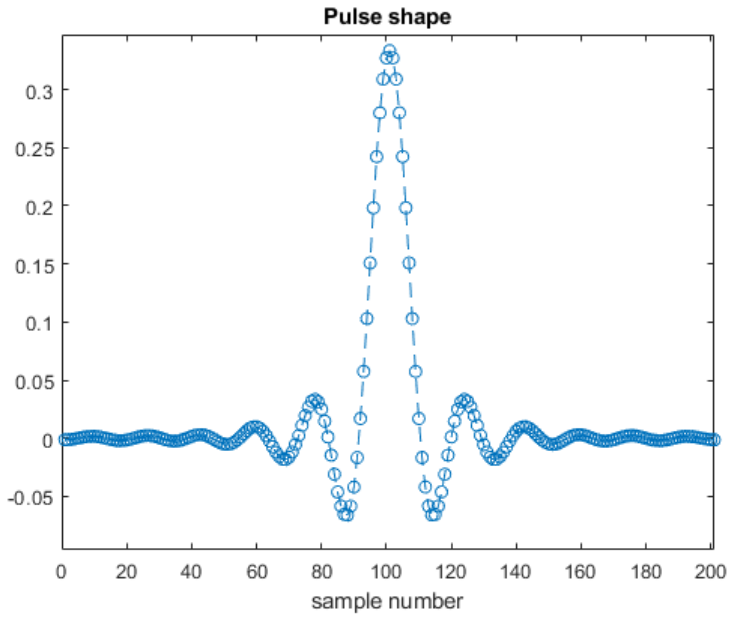
```
symbols=[1+j -1-j -1+j 1-j -1-j 1+j 1+j -1-j 1+j -1+j -1+j 1-j 1+j -1-j 1+j 1-j];
```

Below, we are defining all the other parameters of the function:

```
 Nf=200; % number of samples of the basic pulse
 nsps=10; % number of samples of the PAM signal in a symbol time
 rolloff=0.2; % roll-off factor for the root raised cosine pulse
 pulsetype="ROOTRAISEDCOSINE"; % choose the pulse type by uncommenting the
corresponding line

 [x,h,~]=PAMModulator_2023(symbols,nsps,Nf,pulsetype,rolloff);
```

After generating the PAM signal with the above function, we plot the pulse shape:

```
 figure
 plot(h,'--o')
 xlabel('sample number')
 title('Pulse shape')
```



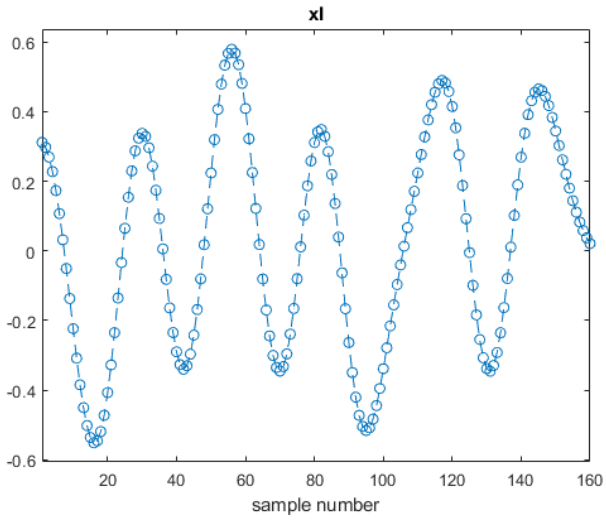
Since the symbols that we transmit are complex, we can plot the real and the imaginary part of the PAM modulated signal.

3.1.1 Now, we are going to extract the real part of the signal

```
figure
plot(real(x3),'--o')
xlabel('sample number')
title('xI')
real(symbols)
```

```
ans = 1×16
     1    -1    -1     1    -1     1     1    -1     1    -1    -1     1
1 ⋯
```

```
axis([1 length(x3) 1.1*min(real(x3)) 1.1*max(real(x3))])
```

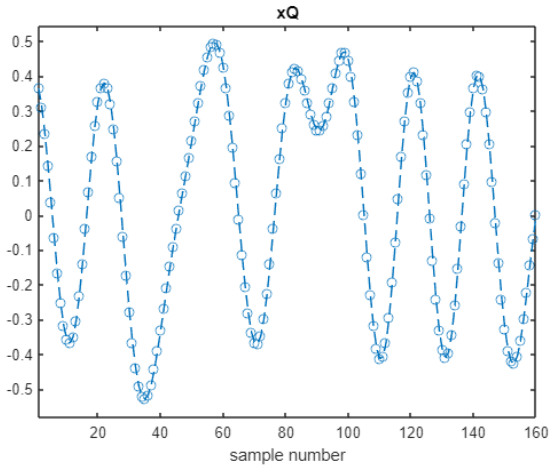


3.1.2 Now, we are going to extract the imaginary part of the signal

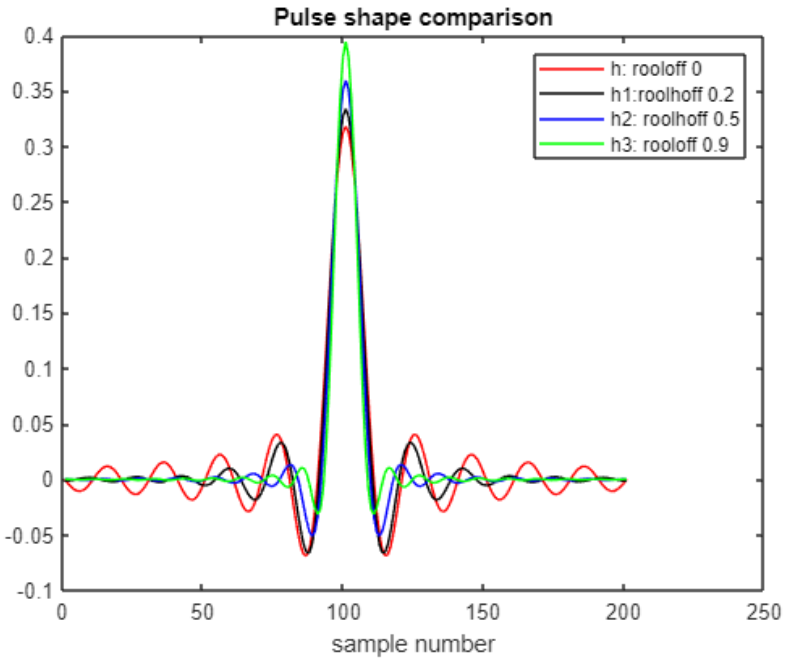
```
figure
plot(imag(x3),'--o')
xlabel('sample number')
title('xQ')
imag(symbols)
```

```
ans = 1×16
     1    -1     1    -1    -1     1     1    -1     1     1     1    -1
1 ⋯
```

```
axis([1 length(x3) 1.1*min(imag(x3)) 1.1*max(imag(x3))])
```

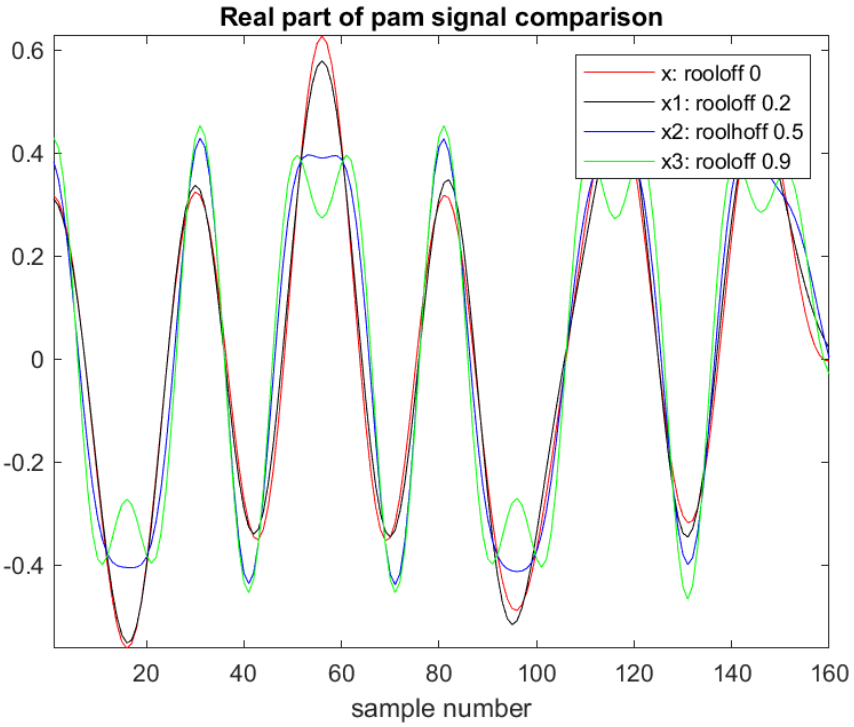


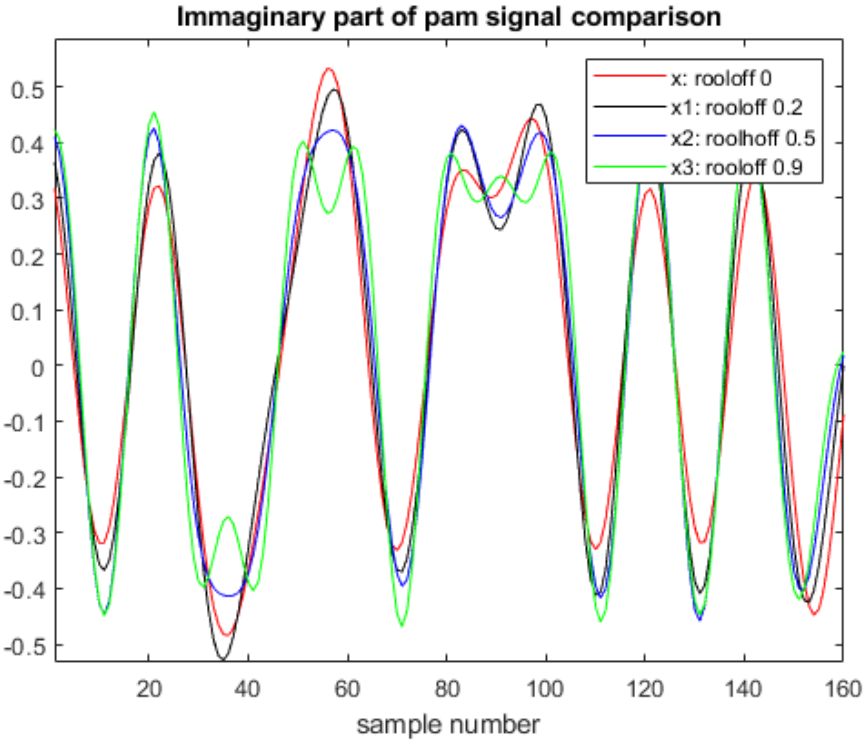
3.1.3 Let's increase and decrease the roll-off factor, how does the Pulse shape change?



The main parameter of a raised cosine filter is its **roll-off factor**, which indirectly specifies the bandwidth of the filter. The **Rolloff factor** is a measure of the excess bandwidth of the filter, i.e., the bandwidth occupied beyond the Nyquist bandwidth of 1/2T, where 1/T is symbol rate.

As we can see from the plot, when we decrease the roll-factor, we have more oscillations in the time domain, what is a non-desired characteristic in principle. In fact, low values of roll-off factor allow for a more efficient use of the spectrum but increase the ISI.

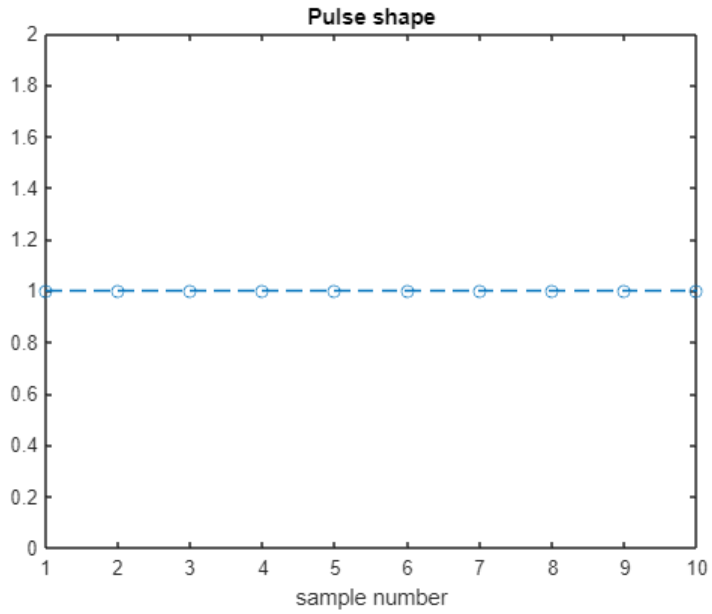

Real part of pam signal comparison
x: rooloff 0
x1: rooloff 0.2
x2: roolhoff 0.5
x3: rooloff 0.9
0.6
0.4
0.2
0
-0.2
-0.4
20
40
60
80
100
120
140
160
sample number


Immaginary part of pam signal comparison
x: rooloff 0
x1: rooloff 0.2
x2: roolhoff 0.5
x3: rooloff 0.9
0.5
0.4
0.3
0.2
0.1
0
-0.1
-0.2
-0.3
-0.4
-0.5
20
40
60
80
100
120
140
160
sample number

3.2 'RECT' PULSE TYPE

Now, we are going to generate a PAM signal using a Rect pulse type:

```
pulsetype="RECT";
[x3,h,Delay]=PAMModulator_2023(symbols,nsps,Nf,pulsetype,rolloff);

figure
plot(h,'--o')
xlabel('sample number')
title('Pulse shape')
```



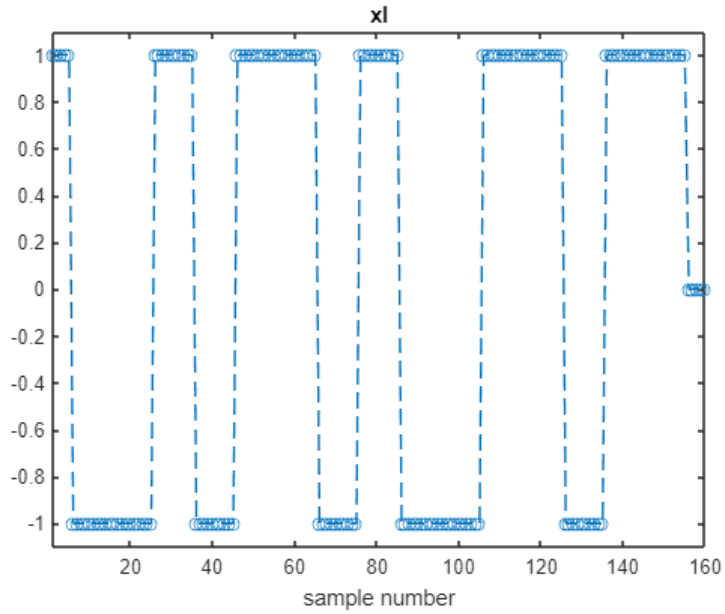
In rectangular pulse, we need that the number of samples of the rect function are equal the symbol length and the filter's coefficients are the samples of the basic pulse. Since the basic pulse has amplitude, we set equal to 1, here we have only 1 every sample.

3.2.1 Now, we are going to extract the real part of the signal.

```
figure
plot(real(x3),'--o')
xlabel('sample number')
title('xI')
real(symbols)
```

```
ans = 1×16
     1    -1    -1     1    -1     1     1    -1     1    -1    -1     1
1 ⋯
```

```
axis([1 length(x3) 1.1*min(real(x3)) 1.1*max(real(x3))])
```

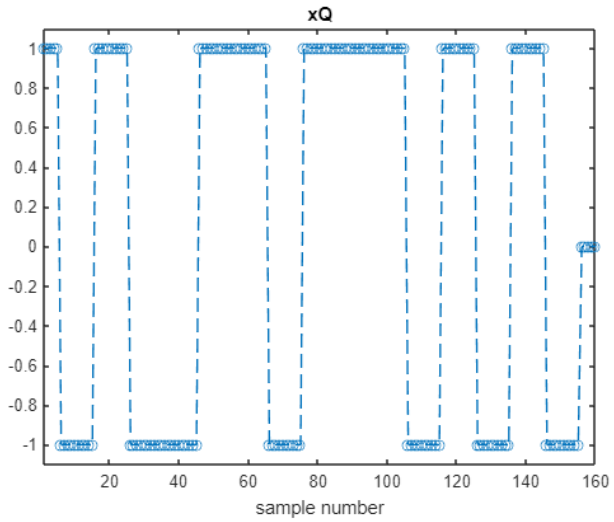


3.2.2 Now, we are going to extract the imaginary part of the signal

```
figure
plot(imag(x3),'--o')
xlabel('sample number')
title('xQ')
imag(symbols)
```

```
ans = 1×16
     1    -1     1    -1    -1     1     1    -1     1     1     1    -1
1 ⋯
```

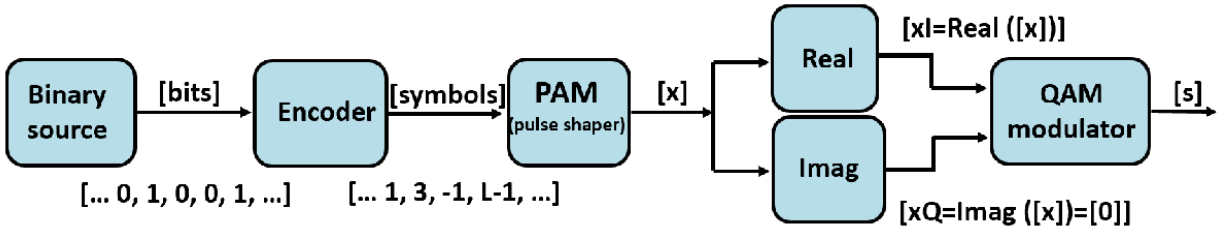
```
axis([1 length(x3) 1.1*min(imag(x3)) 1.1*max(imag(x3))])
```



4. Activity 4: L-ASK MODULATOR

```
clear all % clear all varables
close all % close all plots
clc % clear the screen in the command window
warning('off','all')
```

4.1 L-ASK transmitter



In this activity we would like to implement a transmitter that is able to transmit bits; as we know we cannot directly send bits in our radio channel, firstly we have to encode them into symbols that are sent by transmitting a fixed-amplitude carrier wave at a fixed frequency for a specific time duration. We are going to encode these symbols through the L-ASK (Amplitude-Shift-Keying) modulation, which takes a sequence of $\log_2(L)$ bits and maps it in one of the L symbols. The L-ASK (Amplitude-Shift-Keying) modulation, which takes a sequence of $\log_2(L)$ bits and maps it in one of the L symbols.

The transmitter that we are going to implement is composed by five parts:

1. Binary source generator
2. Gray Encoder
3. Pulse shaper
4. Generation of the in-phase and quadrature signals
5. QAM modulator.

We are defining here the parameter that we need to have in order to generate a signal.

We can select the dimension of the L-ASK constellation, we define this value changing how many bits are associated to each symbol (bits_per_symbol=1,2,3): the number of levels of the modulation alphabet is given by this following relation: **L=2^bits_per_symbol.**

```
%Signal specifications
bits_per_symbol=2;       % number of bits that correspond to a symbol
L=2^bits_per_symbol       % number of levels of the modulation alphabet
```

```
L = 4
```

We are able also to define how many bits we want to transmit, its bit rate as consequence also its symbol rate.

```
nbits=1000*bits_per_symbol % number of bits to be transmitted
```

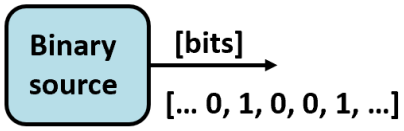
```
nbits = 2000
```

```
Br=500;                    % [bits/s] bit rate
fs=5000;                   % [samples/s] sampling frequency
fc=3000;                   % [Hz] carrier frequency
Bs=Br/log2(L)              % [symbols/s] Symbol rate
```

```
Bs = 250
```

```
nsps=fs/Bs;                % samples per symbol
```

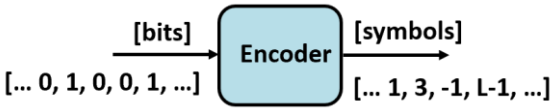
4.1.1Source bits generation




```
source_bits=BinarySource_2023(nbits); % vector with the source bits
```

The vector **source_bits** is composed by independent and identically distributed bits with alphabet {0,1}; the value of certain bit do not influence the value of the other bits and the probability of having outcome 1 and the probability of having outcome 0 is the same. We have 50% chance to have 1 and 50% chance to have 0.

4.1.2 Symbols generation with Gray mapping




```
symbols=Encoder_2023(source_bits, L);
```

```
table = 4×3
      0      0     -3
      0      1     -1
      1      1      1
      1      0      3
bit_table = 4×2
      0      0
      0      1
      1      1
      1      0
symbol_table = 4×1
     -3
     -1
      1
      3
```

Through this MATLAB function **Encoder_2023 (source_bits, nlevels)** we are going to encode our bits, given as input the vector **source_bits**. In order to get the output vector symbols we use the multi-level symmetric alphabet {−1, +1, −3, +3, . . . − L + 1, L − 1} of L =**nlevels** levels (L = 2, 4, 8), according to the gray mapping rule. According to the gray code rule, adjacent symbols has just one bit of distance, means the adjacent sequences of bits differ by just one bit.

For example, in case L is equal to 4 (=2^2), the symbol -3 is associated to the couple of bits 00, while the adjacent bits -1 is associated to the couple of bits 01 and as we can see they differ only by just one bit. Also, between symbol -1 and 1 they differ only by one bit (01 and 11).

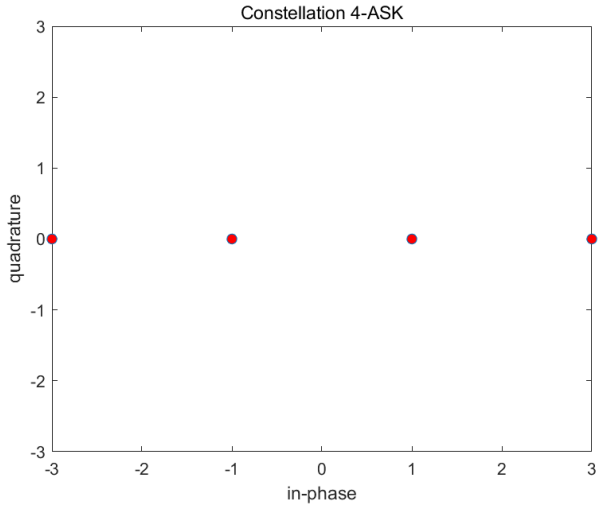
4.1.3 Plot of the constellation

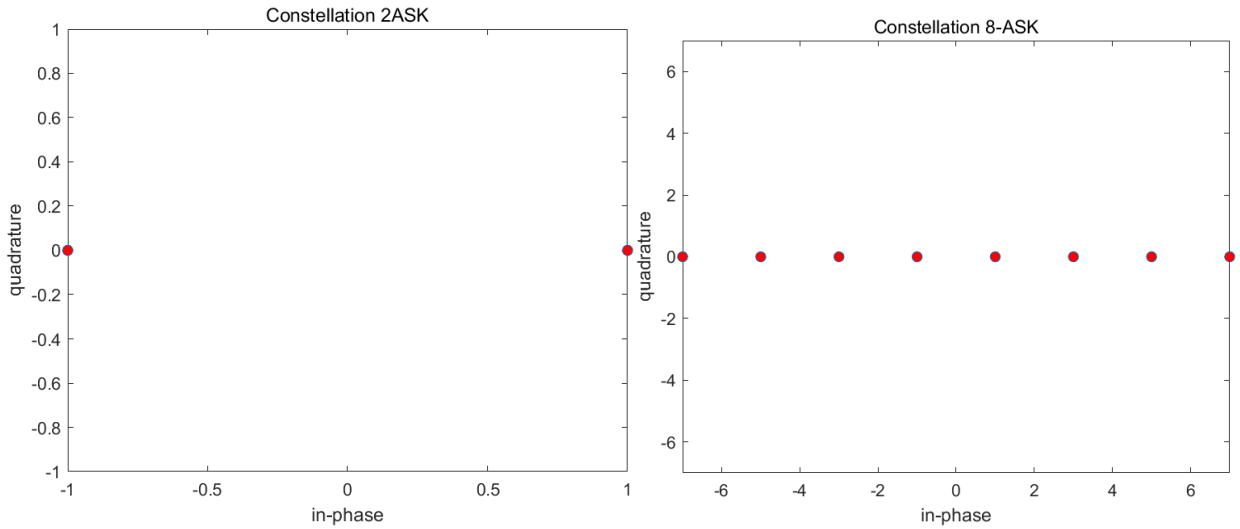
The constellation diagram can often demonstrate how the amplitude and phase of signals or symbols differ.

In the constellation diagram, the in-phase and quadrature components of the complex envelope are represented by the x- and y-axis respectively.

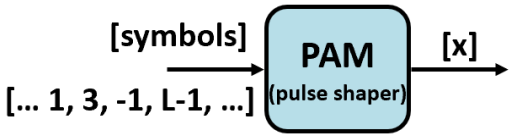
The L-ASK constellation has symbols that have only in-phase component and symbols take value from -L+1 to +L+1, their spacing is always equal between each other and is always equal to 2. Since the spacing represents also how well a receiver can distinguish between all potential symbols in the presence of noise, increasing L means that we are increasing the L-1 thresholds, and the receiver can more easily choose the wrong symbols.

```
figure
plot(real(symbols),imag(symbols),'o','MarkerFaceColor','r'),
axis ([-L+1 L-1 -L+1 L-1]);
title('Constellation');
xlabel('in-phase');
ylabel('quadrature');
```





4.1.4 Generation of the baseband PAM signal



The symbols are modulated through a Pulse shaper which gives a baseband analog signal. So, we are defining which pulse type we are going to use to create this PAM signal.

```
Nf=5*nsps; % number of filter coefficients (=five time the symbol duration)
pulsetype='ROOTRAISEDCOSINE'; % pulsetype: 'RECT' or 'ROOTRAISEDCOSINE'
switch pulsetype
    case 'ROOTRAISEDCOSINE'
        rolloff=0.8; % roll-off factor for root raised cosine pulses
        bandwidth_Hz=0.5*Bs*(1+rolloff) % bandwidth of the baseband signal
    case 'RECT'
        rolloff=[];
        bandwidth_first_lobe_Hz=Bs
end
```

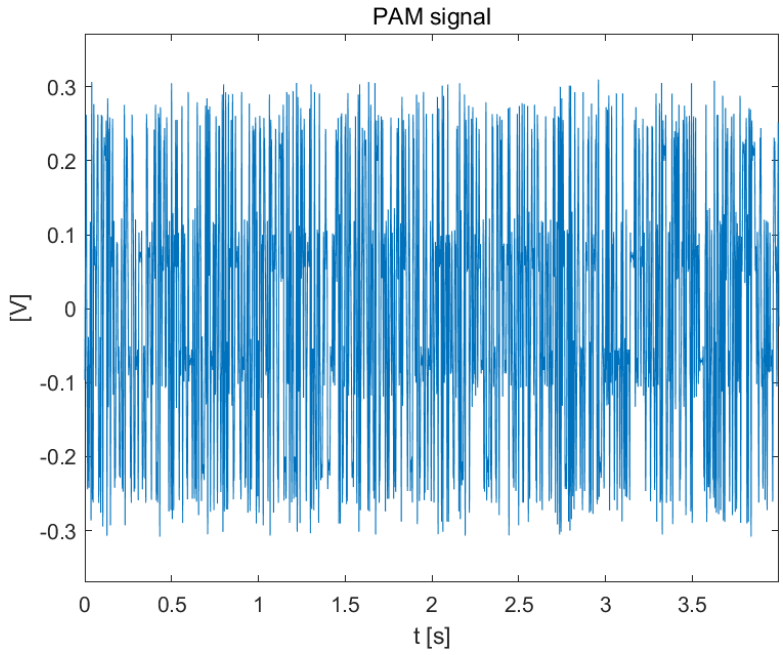
```
bandwidth_Hz = 225
```

```
% ------------------------------------------------------------------------
x=PAMModulator_2023(symbols,nsps,Nf,pulsetype,rolloff);
Ts=1/fs;% sampling interval
T=Ts*(length(x)-1); % signal duration
```

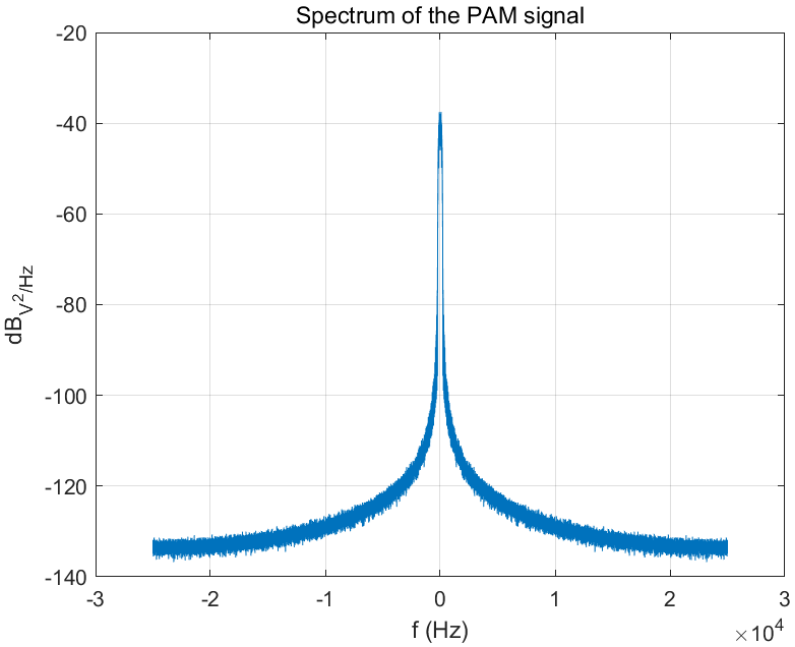
4.1.4.1Plot of the baseband PAM signal and of its spectrum

```
figure
t=0:1/fs:T;
```

```
plot(t,x)
xlabel('t [s]')
ylabel ('[V]')
title('PAM signal')
axis([min(t) max(t) 1.2*min(x) 1.2*max(x)])
```



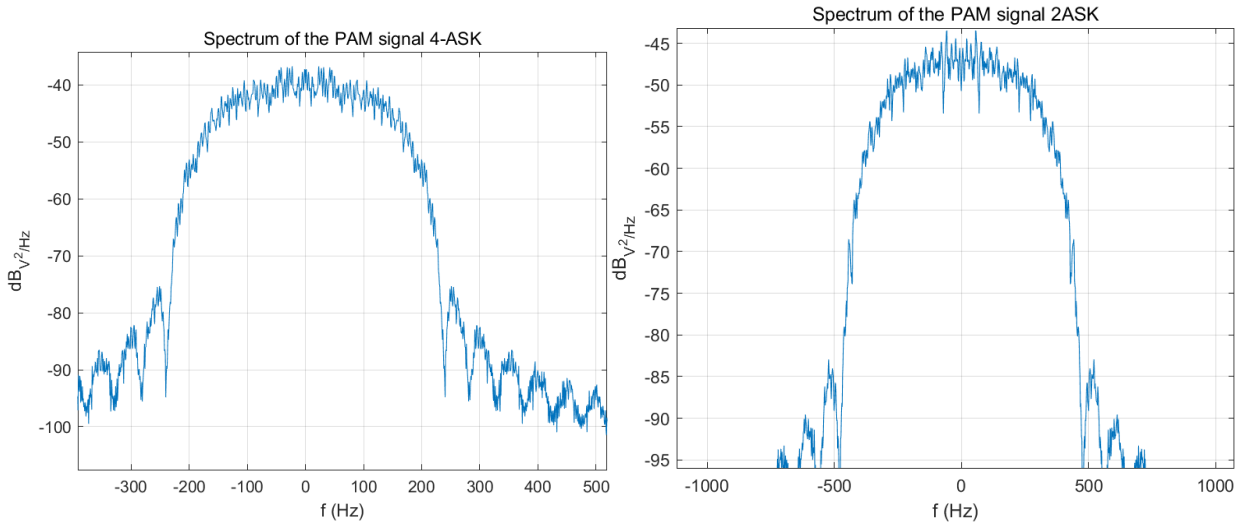

```
figure
PlotSpectrum(x,fs);
title('Spectrum of the PAM signal')
```

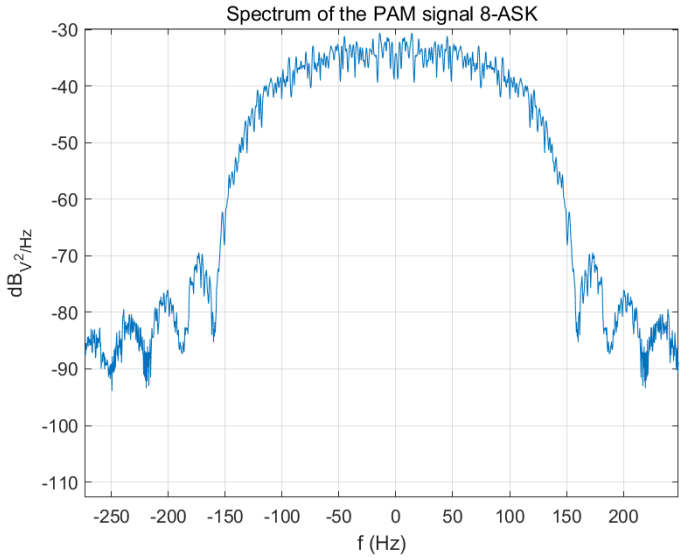


4.1.4.2 Spectrum of the PAM signal with different L levels

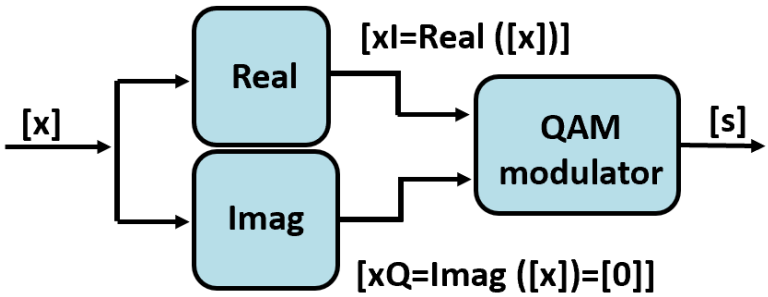
The bandwidth of the PAM signal is proportional to the symbol rate at which symbols enter the pulse shaper. Since the PAM signal that we generate is baseband, as we can see from the plot, the spectrum is always centered in 0Hz.

Since the bandwidth of the PAM signal is proportional to the symbol rate, and symbol rate decrease by a factor log2(L) every time we increase L also the bandwidth decreases by factor log2(L). Having bitrate Br=500, we have for the 8-ASK a lower bandwidth around 160 Hz than the case of 4-ASK (around 225 Hz) and 2-ASK (near 500 Hz). In L=2, the symbol rate corresponds exactly to the bit rate, so that we are using the maximum bandwidth (near 500 Hz).





4.1.5 L-ASK modulation through the QAM modulator



We use a QAM modulator to convert the signal from base-band to pass-band with carrier frequency fc; since the symbols that we are transmitting are real, we only have in-phase signals, we set the quadrature component as zero.

```
xI=x % in-phase signal
```

```
xI = 1×200000
   -0.0964   -0.0962   -0.0960   -0.0958   -0.0955   -0.0952   -0.0949…
```

```
xQ=zeros(1,length(x)) % the quadrature component is zero
```

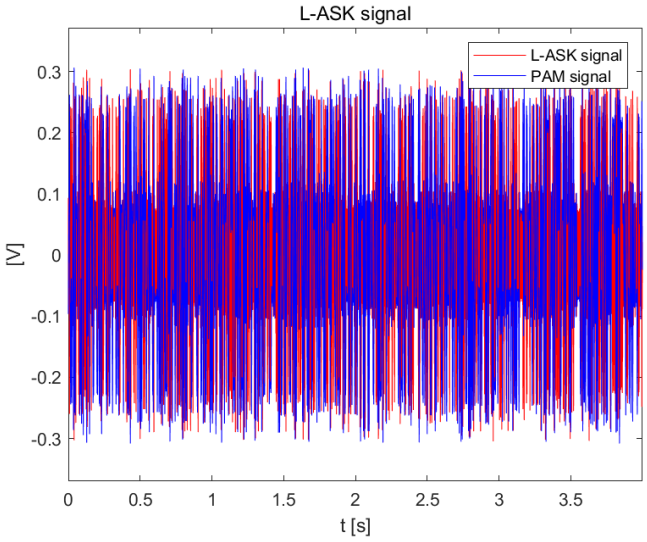
```
xQ = 1×200000
   0     0     0     0     0     0     0     0     0     0     0     0 …
```

```
s=ModQAM_2023(xI,xQ,fc,T,fs)
```

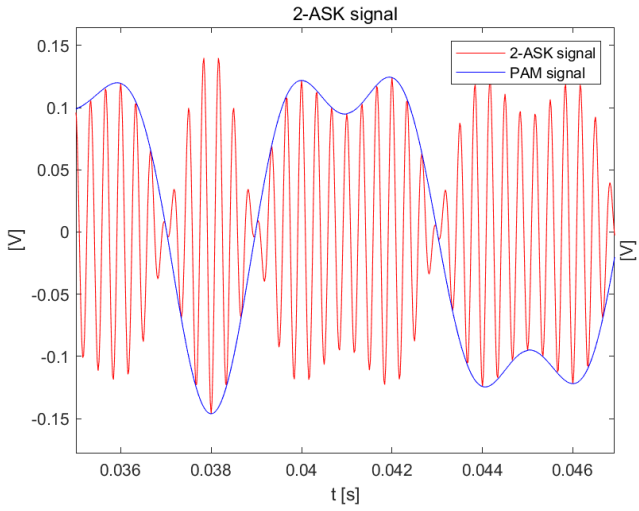
```
s = 1×200000
  -0.0964   -0.0962   -0.0960   -0.0958   -0.0955   -0.0952   -0.0949 …
```

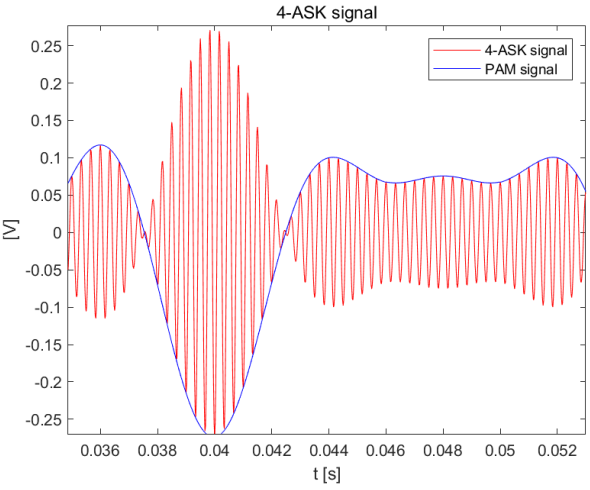
4.1.6 Plot of the L-ASK signal and of its spectrum

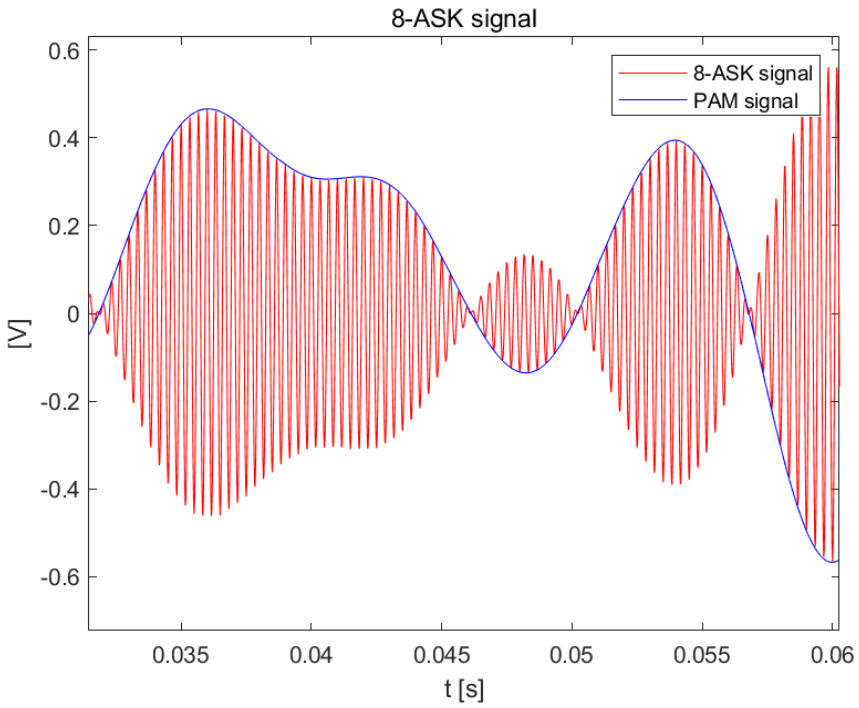
```
figure
plot(t,s,'r')
hold on
plot(t,x,'b')
axis([min(t) max(t) 1.2*min(x) 1.2*max(x)])
xlabel('t [s]')
ylabel ('[V]')
title('L-ASK signal')
legend('L-ASK signal','PAM signal')
```



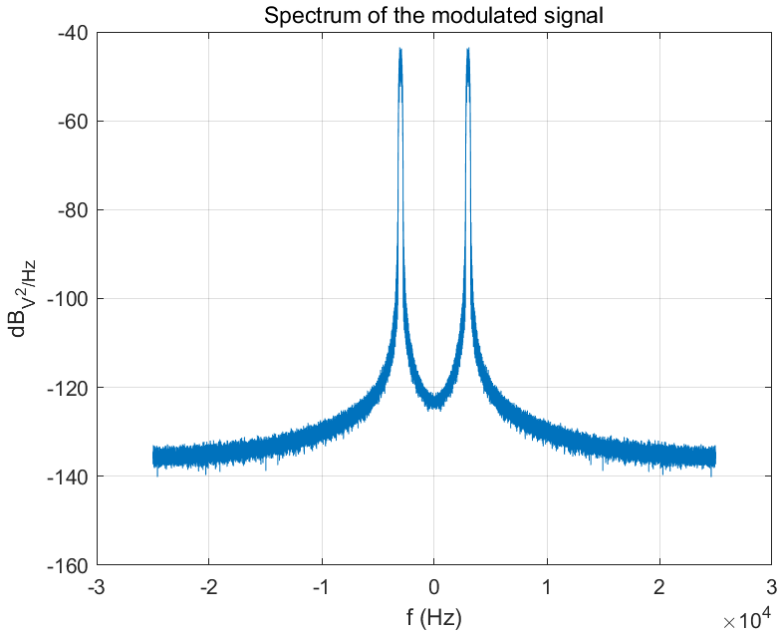
The L-ASK signal (in red) is the one which is really transmitted on the radio medium. It's possible to see that the envelope of this signal is the baseband PAM signal (in blue). This envelope is the important part of the wave because it contains our symbols so our bits of information. We can observe that when the symbols from positive become negative or from negative become positive, the L-ASK signal changes its phase of π . Increasing L, we can transmit more symbols, but the level of amplitude will also increase. In 2-ASK the maximum amplitude is 0.15V instead in 8-ASK we can reach the 0.6V of amplitude, which means we must feed the transmitter with a higher power.



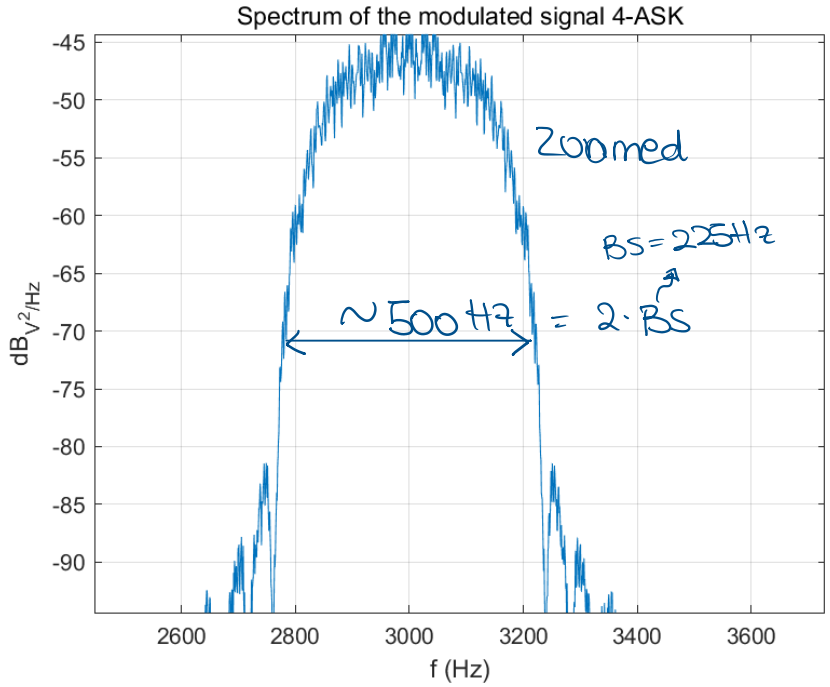


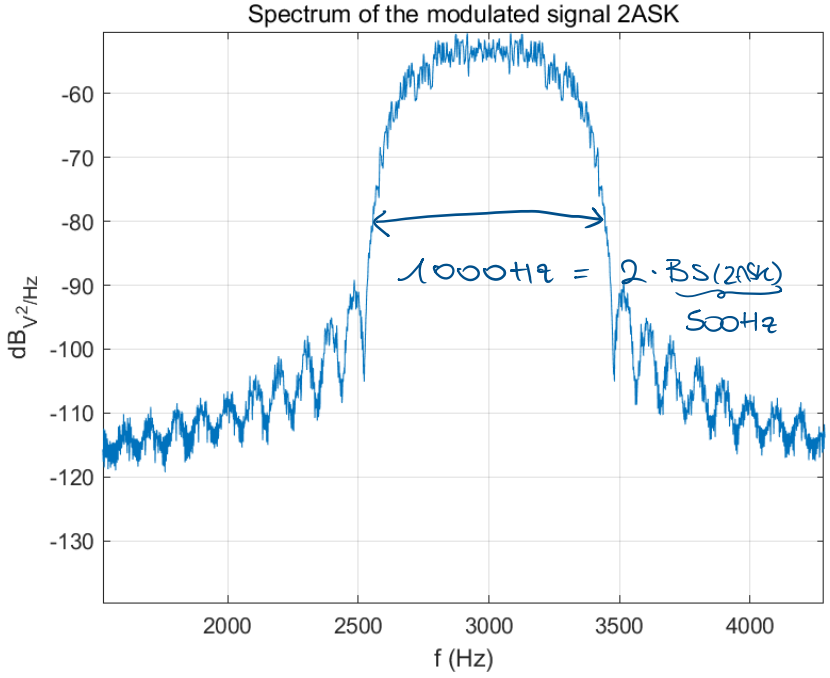


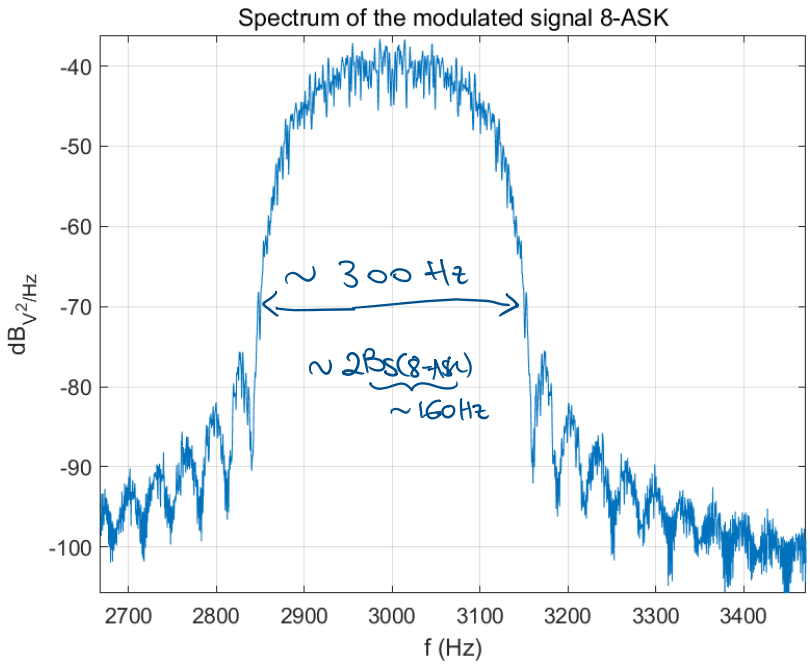

```
figure
PlotSpectrum(s,fs);
title('Spectrum of the modulated signal')
```



The spectrum of the modulated signal shows that since they are no more baseband signal, now they are centered in frequency fc=+-3000hz and divided in amplitude by 2 (in linear units). The bandwitdh is the double of that in PAM signal's spectrum, but the previous considerations on bandwidth for different values of L still hold.


Spectrum of the modulated signal 4-ASK
dB V²/Hz
-45
-50
-55
-60
-65
-70
-75
-80
-85
-90
2600
2800
3000
3200
3400
3600
f (Hz)
Zoomed
BS = 225Hz
≈ 500 Hz = 2 · BS


Spectrum of the modulated signal 2ASK
dB V²/Hz
-60
-70
-80
-90
-100
-110
-120
-130
2000
2500
3000
3500
4000
f (Hz)
1000Hz = 2 · BS(2ASK)
500Hz



4.1.7 Output the signal to the audio card.

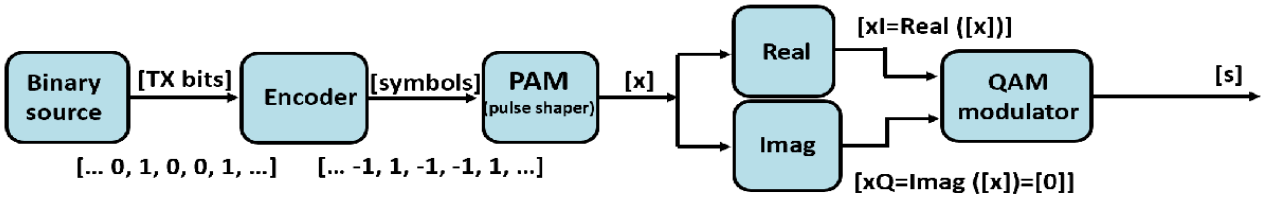
```
sound(s,fs);
```

5. Activity 5: 2-ASK system (both transmitter and receiver)

In this activity, we are going to implement a system that transmit and receive binary code; we use in both side a 2-ASK system. We also add inside the system a AWGN block that creates noise that will be filtered in the receiver side of the system, but as we will observe in the constellation plot, the noise will let us receive symbols not exactly ones or minus ones. We will see how to minimize this effect of the noise.

```
clear all % clear all variables
close all % close all plots
clc % clear the screen in the command window
```

5.1 Transmitter



Signal specifications

```
bits_per_symbol=1;            % number of bits that correspond to a symbol
L=2^bits_per_symbol;          % number of levels of the modulation alphabet
nbits=100000*bits_per_symbol; % number of bits to be transmitted
Br=500;                       % [bits/s] bit rate
fs=50000;                     % [samples/s] sampling frequency
Ts=1/fs;                      % sampling time[s].
fc=3000;                      % [Hz] carrier frequency
Bs=Br/log2(L);                % [symbols/s] Symbol rate
nsps=fs/Bs;                   % samples per symbol
```

Pulse shaper specifications

```
Nf=5*nsps; % number of filter coefficients for the pulse shaper
pulsetype='RECT';

switch pulsetype
    case 'ROOTRAISEDCOSINE'          % root raised cosine pulse
    rolloff=0.8                      % roll-off factor for root raised cosine
pulses
    bandwidth_Hz=0.5*Bs*(1+rolloff); % bandwidth of the baseband signal
    case 'RECT'                      % rectangular pulse,
    rolloff=[];
```

```
    bandwidth_first_lobe_Hz=Bs;
end
```

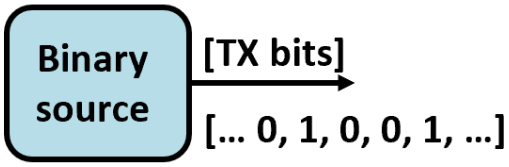
Demodulator specifications

```
psi=0; % carrier offset at the receiver
```

5.1.1 Monte Carlo simulation

```
index=1;
for EbN0_dB=1:15 % Run a Monte Carlo simulation for 10 different values of SNR
```

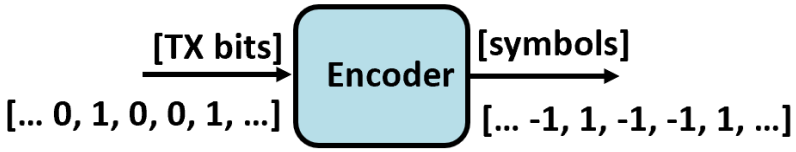
5.1.1.2 Source bits generation



With the code below, we are going to generate a Binary Source bit with Bit rate Br=500 bits/s.

```
source_bits=BinarySource_2023(nbits); % vector with the source bits
```

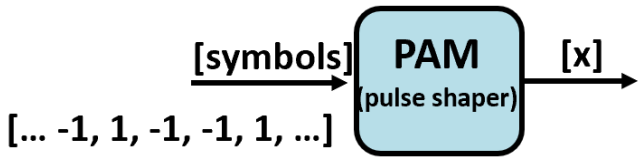
5.1.1.3 Symbols generation with Gray mapping



Now, we are going to convert the binary code into symbols using the Gray mapping.

```
symbols=Encoder_2023(source_bits, L);
```

5.1.1.4 Generation of the baseband PAM signal

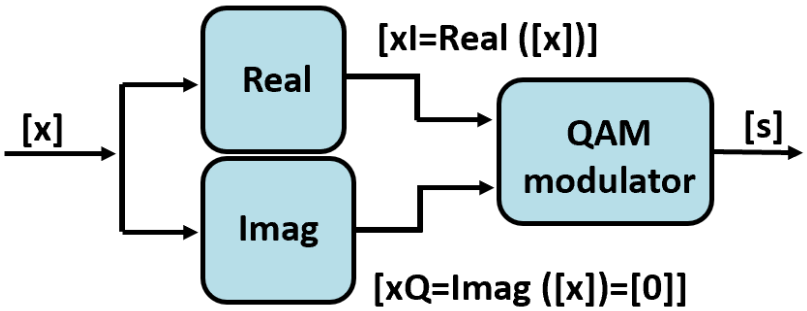


We are now generating a PAM signal that will allow us to send the signal to the receiver.

```
[x,h,DelayPAM]=PAMmodulator_2023(symbols,nsps,Nf,pulsetype,rolloff);
```

```
T=Ts*(length(x)-1); % signal duration
```

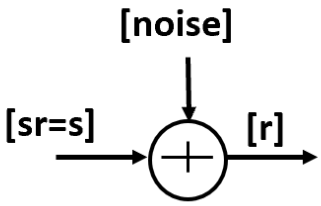
5.1.1.5 2-ASK modulation through the QAM modulator



In this part of the code, we modulate the signal with QAM modulator: as input we have in-phase signal xI and quadrature signal xQ. xI is the real part of x (output of the Pulse Shaper) while xQ is the imaginary part of x, but since we have x composed just by real components, we have all the quadrature components are zero.

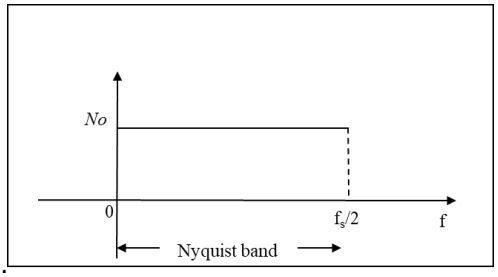
```
xI=x; % in-phase signal
xQ=zeros(1,length(x)); % the quadrature component is zero
s=ModQAM_2023(xI,xQ,fc,T,fs);
```

5.1.1.6 The AWGN Channel



The AWGN Channel generated an additive White Gaussian noise, so the noise generated is added to the input signal sr. The input of this channel is equal to the modulated signal s and the signal.

$sr = s$ is characterized by Energy E_b in a bit interval T_b. The noise that we generate is characterized by power spectral density N_0.



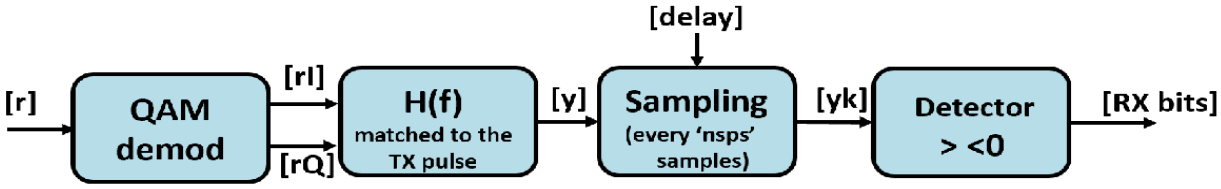
At the output of the channel, the signal is characterized by the Signal-to-Noise Ratio SNR:

$$\mathrm{SNR} = \frac{E_b}{N_0}$$

5.1.1.7 Noise generation and sum with the received signal.

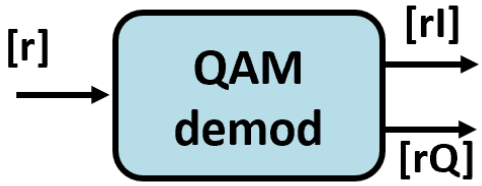
```
sr=s;
P=(1/length(sr))*sum(sr.^2); % Estimation of the mean power of the signal
EbN0=10^(0.1*EbN0_dB);       % SNR expressed in linear units
sigman=sqrt(P*fs/(2*Br*EbN0)); % calculation of the noise's standard deviation
noise=sigman.*randn(1,length(sr)); % generation of the Gaussian noise
r=sr+noise; % addition of the noise to the received signal
```

5.1.1.8 Receiver



At the receiver, at first, we demodulate the signal r, so we get two signals: in-phase signal rI and quadrature signal rQ. Then, we introduce a matching filter that try its best to remove the noise, so we obtain signal y. This signal y is introduced to a Sampler that samples the signal every Ts seconds. After having sampled the received signal, we use the Detector to detect bits of ones and zeros, because of the noise we do not receive exactly symbols 1 or -1, using the detector we get a binary signal RX.

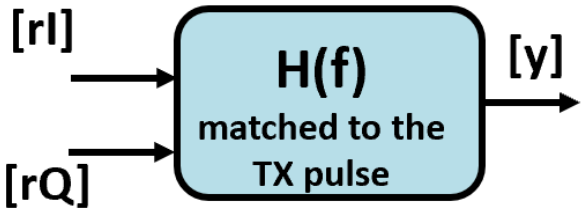
5.1.1.9 QAM demodulator



With the QAM demodulation, we spit the input signal into in-phase component and quadrature component. This QAM Demodulator is characterized by its Delay $\mathrm{DelayQAM}$.

```
[rI,rQ,DelayQAM]=DeModQAM_2023(r,fc,T,fs,psi); % demodulate the signal
```

5.1.1.10 Matched filter

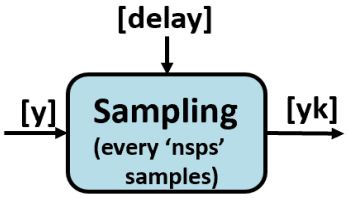


To remove the noise introduced by the AWGN channel, we introduce in the receiver the Matched Filter. The Matched filter is designed to match the shape of the transmitted pulse, with the goal of maximizing the SNR of the received signal. The SNR is a measure of the ratio of the signal power to the noise power. In a noise environment, the signal power is usually lower than the noise power, the goal of the matched filter is to increase the SNR of the received signal by amplifying the signal and reducing the noise. The matched filter is a time-reversed copy of the transmitted pulse (RECT in this case). This means that the filter output is the convolution of the received signal with the time-reversed pulse.

We use the filter used in the pulse shaper, so consider the same delay $\mathrm{DelayPAM}$.

```
r=rI+1i*rQ;
y=conv(r,h)/nsps; % filtering with the matched filter
```

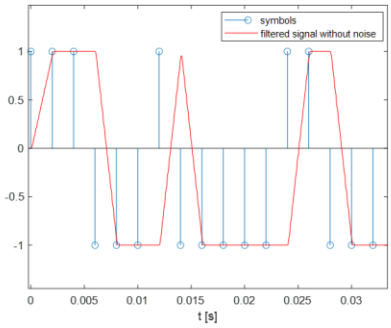
5.1.1.11Sampler



In the Sampler, we sample every $nsps$ samples starting from an initial delay. So, we do not sample in the instant when the symbols were generated, but we start from an initial delay, and is due to:

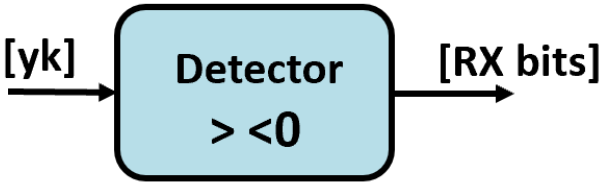
1. the pulse shaper: $\mathrm{DelayPAM}$.
2. the filter in the QAM demodulator: $\mathrm{DelayQAM}$
3. the matched filter at the receiver: $\mathrm{DelayPAM}$.

We sample the first sample not at $t = 0s$ but at $t = delay\ [s]$ and after Ts seconds we sample the second symbols and so one.




```
% compute the overall delay introduced by the TX and RX filters
delay=round (2*DelayPAM+DelayQAM);
% sample the signal every symbol time accounting for the delay
yk = y(delay:nsps:nsps*nbits+delay-1);
```

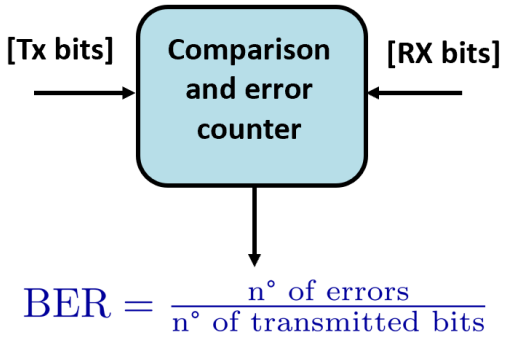
5.1.1.12 Detector



Due to the noise, we do not receive exact symbols -1 or 1, so here using the Detector we check whether the sample is > 0 (hence $bit = 1$) or < 0 (hence $bit = 0$).

```
RXdata=real(yk) > 0; % take only the real part
```

5.1.1.13 Bit Error Rate (BER) estimate



Since we know the transmitted bits and now, we also get the received bits, we could calculate the Bits Error Rate; The bit-error rate (BER) is an estimate of the theoretical bit error probability P_b

```
noe=sum(abs(source_bits-RXdata)); % Errors count
nod=length(source_bits);
```

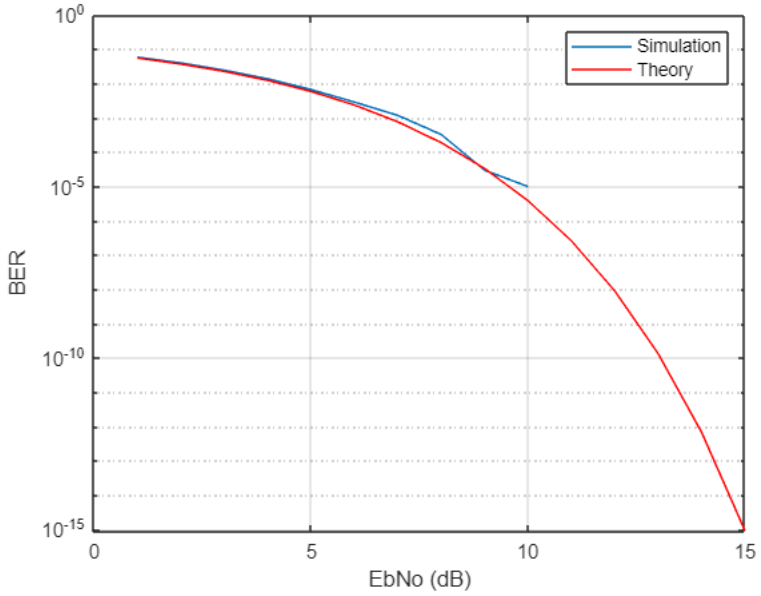
```
SNRdB(index)=EbN0_dB;
ber(index)=noe/nod; % Estimate the BER
fprintf('EbNo=%.1f (dB) BER=%g\n',SNRdB(index),ber(index));
index=index+1;
% End of Monte Carlo simulation
end
```

```
EbNo=1.0 (dB) BER=0.05932
EbNo=2.0 (dB) BER=0.04032
EbNo=3.0 (dB) BER=0.02471
EbNo=4.0 (dB) BER=0.01395
EbNo=5.0 (dB) BER=0.00675
EbNo=6.0 (dB) BER=0.00291
EbNo=7.0 (dB) BER=0.00119
EbNo=8.0 (dB) BER=0.00033
EbNo=9.0 (dB) BER=3e-05
EbNo=10.0 (dB) BER=1e-05
EbNo=11.0 (dB) BER=0
EbNo=12.0 (dB) BER=0
EbNo=13.0 (dB) BER=0
EbNo=14.0 (dB) BER=0
EbNo=15.0 (dB) BER=0
```

5.2 Plots

5.3 BER vs EbN0 plot

```
figure
semilogy(SNRdB,ber);
xlabel('EbNo (dB)');
ylabel('BER');
grid on
hold on
SNR=10.^(0.1*SNRdB);
Pb=0.5*erfc(sqrt(SNR));
plot(SNRdB,Pb,'r')
legend('Simulation', 'Theory')
```

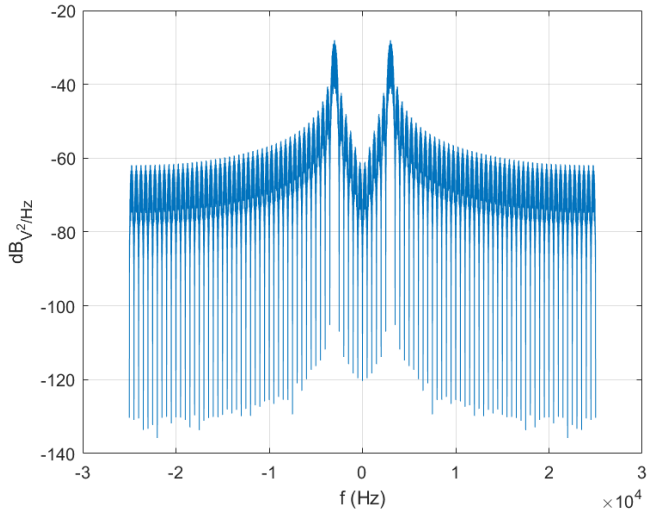


In this plot, we compare the BER provided by the simulator with the theoretical Pb: $P_b = \frac{1}{2}\text{erf}\sqrt{\frac{E_b}{N_0}}$.

The red curve represents the theoretical P_b and the blue one is the BER provided by the simulator, and for value of BER higher, we have a quite accurate simulation of P_b, they are almost coincident, but for value of lower BER we have much higher difference, because we do not transmit for example 10^7 bits so we haven't an accurate estimation. To improve the accuracy for low bit error rates we need to increase the number of transmitted bits.

5.4 Spectrum plot

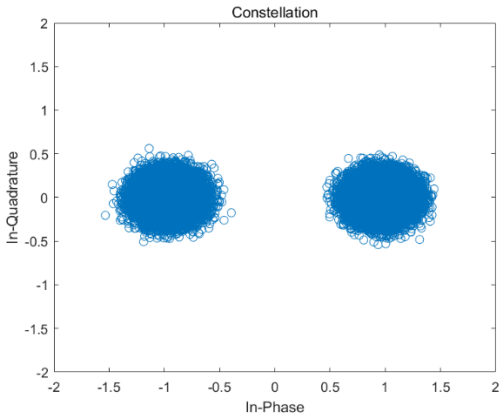
```
figure
PlotSpectrum_2023(s,fs);
```



5.5 Constellation

This code shows us the constellation of the signal sampled with respect to the last value of SNR.

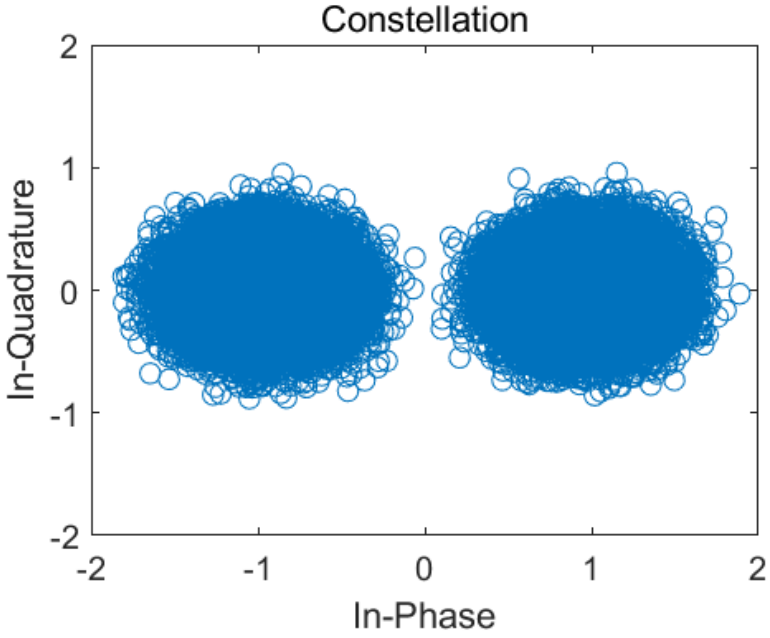
```
x=real(yk);
y=imag(yk);
figure
plot(x,y,'o'), axis ([-2 2 -2 2]);
title('Constellation');
xlabel('In-Phase');
ylabel('In-Quadrature');
```



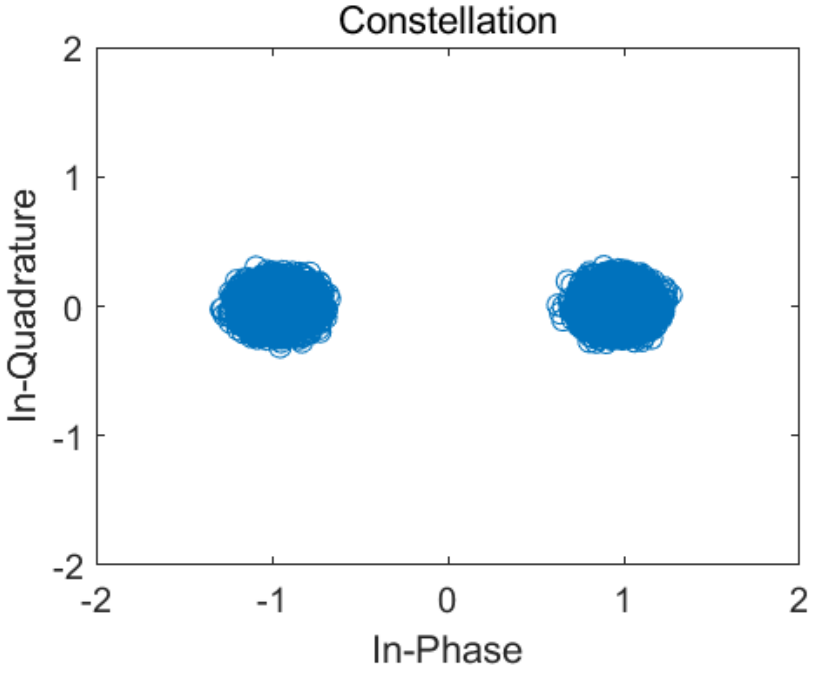
In this constellation plots, we can observe two clouds that represents how the signals sent are distributed by the noise, in fact we do not get exact value of -1 and 1.

If we increase EbN0 we get smaller clouds, so our message is less disrupted by the noise.

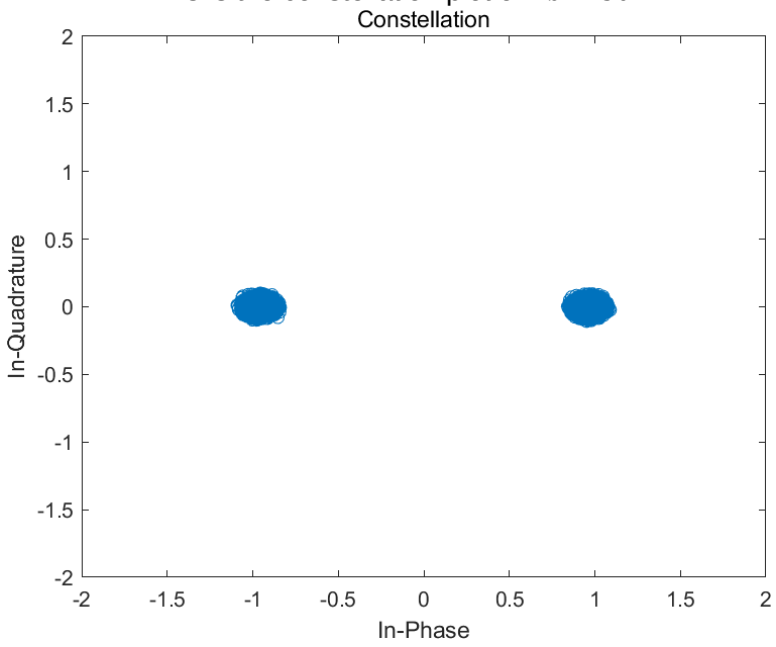
This is the constellation plot of $Eb = 10$



This is the constellation plot of $Eb = 20$



This is the constellation plot of $Eb = 30$



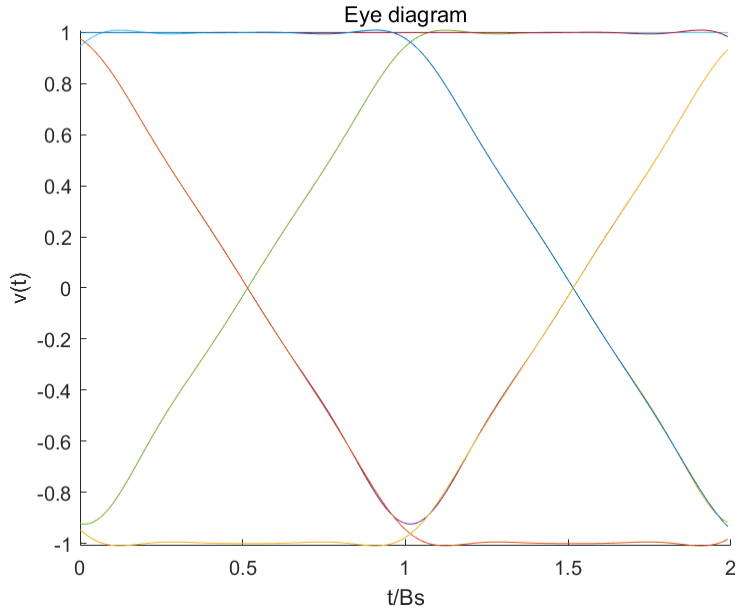
The sampled symbols are more closer to -1 and 1 as we increse the power of a signal.

5.6 Eye diagram

In the following code, we are going to consider that the receiver is repetitively sampled,i.e. so we consider several different sequence of symbols (ones and minus ones) that we sample with the sampler. The eye diagram is simply a graphical display of a serial data signal with respect to time that shows a pattern that resembles an eye. Indeed, what we want is that in the instant in which the

sampler at receiver samples the signal , this eye is open, i.e. the sampler has to detect high value o low value, so in the sample instant the important is the receiver detect value close or equal to 1 ore close or equal -1 as consequence we can see the eye at sample instant open.

```
H=2; % number of observation symbols
[yi,yq,DelayQAM]=DeModQAM_2023(sr,fc,T,fs,psi); % demodulate the signal
vt0=conv(yi,h)/nsps;
mm=max(abs(vt0));
x1=0:H*nsps-1;
x2=x1./nsps;
figure;
hold on
for iii=1:10
x=x1+iii*nsps+delay-1;
y=real(vt0(x)); % plot the in-phase via
plot (x2,y), axis([0 H -mm mm]);
end
title('Eye diagram');
xlabel('t/Bs');
ylabel('v(t)');
```



6. Activity 6: Coded Digital Transmission Chain

In this activity, we build a complete **coded BPSK digital transmission system** with **Hamming code** and test its performance.

6.1 2-ASK system with Hamming coding.
Environment and signal specification

With the following codes, we are defining the environment with which we implement our 2-ASK system, defining also the signal specification (number of bits to be transmitted, number of bits that correspond to a symbol etc...), the pulse shaper specification (which is the shape of the PAM signal) and demodulation specification. What is new here is the codes that checks the number of bits to by coded by the Hamming encoder.

```
% ************* Initialization of the environment*
clear all
close all
% ************* Signal specifications*
bits_per_symbol=1; % number of bits that correspond to a symbol
L=2^bits_per_symbol; % number of levels of the modulation alphabet
nbits=100000*bits_per_symbol; % number of bits to be transmitted
Br=500; % [bits/s] bit rate
fs=50000; % [samples/s] sampling frequency
Ts=1/fs; % sampling time[s].
fc=3000; % [Hz] carrier frequency
Bs=Br/log2(L); % [symbols/s] Symbol rate
nsps=fs/Bs; % samples per symbol
% ************* Pulse shaper specifications*
Nf=5*nsps; % number of filter coefficients for the pulse shaper
pulsetype='RECT';
switch pulsetype
case 'ROOTRAISEDCOSINE'
rolloff=0.8 % roll-off factor for root raised cosine pulses
bandwidth_Hz=0.5*Bs*(1+rolloff); % bandwidth of the baseband signal
case 'RECT' % rectangula pulse
rolloff=[];
bandwidth_first_lobe_Hz=Bs;
end
% ************* Demodulator specifications*
psi=0; % carrier offset at the receiver
% ************* Check on the number of bits to be coded by the Hamming encoder*
if rem(nbits,4)~=0 % The number of bits must be a multiple of 4
disp('The number of bits to e transmitted must be a multiple of 4')
return % The execution is stopped
end
```

As in the previous activity, the Monte Carlo simulation takes different SNR and gives as results the BER. Also, the communication scheme is still the same, unless that in both the scheme of the transmitter and the receiver we have to add the blocks that take care of the Hamming coding.

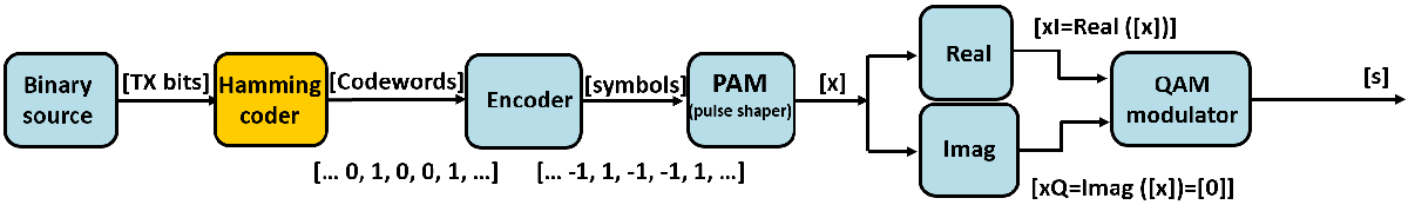
Hamming code is a set of error-correction codes that can be used to **detect and correct the errors** that can occur when the data is moved or stored from the sender to the receiver. In this activity, we are using the traditional (7,4) Hamming code: it adds three additional check bits to every four data bits of the message. Hamming's (7,4) algorithm can correct any single-bit error or detect all single-bit and two-bit errors.

For this reason, we must check the number of bits must be a multiple of 4.

6.2 Monte Carlo simulation

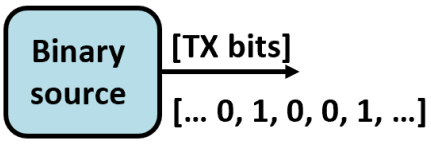
```
index=1;
for EbN0_dB=1:10 % Run a Monte Carlo simulation for 10 different values of SNR
```

6.2.1 Transmitter



We use a transmitter that we have developed in the previous activity but what is new is the ***Hamming coder Block***. We have to put an Hamming coder block after the bit source. The Hamming coder will give a new sequence of codewords with the coded bits. Each bit of this new sequence is then encoded in a symbol based on the 2-ASK modulation. Each symbol is then associated to a PAM waveform in the pulse shaper and at the end, after the QAM block, is transmitted to the AWGN channel.

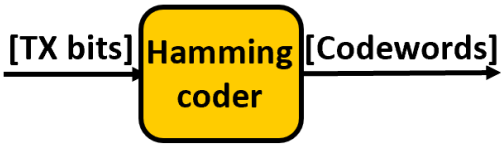
6.2.2 Binary Source



We are generating the i.i.d. bits to transmit.

```
 source_bits=BinarySource_2023(nbits); % generation of a vector with the source
bits
```

6.2.3 Hamming Coder




```
codedbits=HammingCoder_2023(source_bits); % (7,4) Hamming coder
nCodedBits=length(codedbits);
```

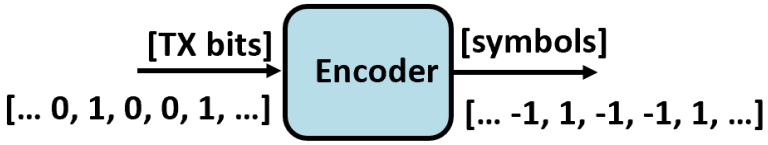
Through this Hamming Coder Block, the bits generated are coded with (7,4) Hamming code. Each packet of 4 bits corresponds to a new packet (codeword) of 7 bits based on this table.

x	codeword
0000	0000000
0001	0001111
0010	0010011
0011	0011100
0100	0100101
0101	0101010
0110	0110110
0111	0111001
1000	1000110
1001	1001001
1010	1010101
1011	1011010
1100	1100011
1101	1101100
1110	1110000
1111	1111111

In this way, our bits are more protected from the noise introduced by the channel, and the receiver is more able to detect the correct transmitted stream.

6.2.4 Encoder

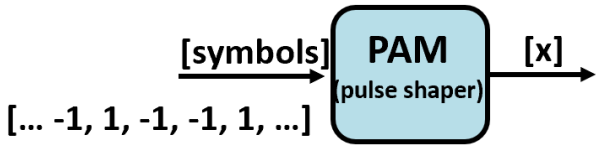
Now, we associate our ***codebits*** with symbols according to gray rule.




```
% ************* Symbols generation with Gray mapping*
symbols=Encoder_2023(codedbits, L);
```

6.2.5 Pulse Shaper

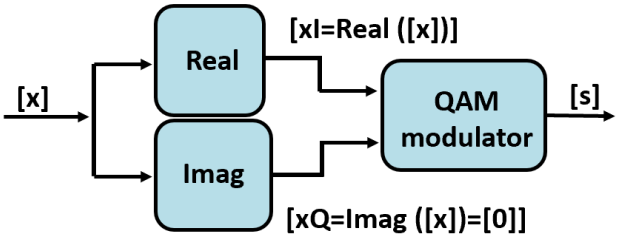
Now, we are going to generate the PAM signal that allow us to transmit our symbols.




```
 [x,h,DelayPAM]=PAMmodulator_2023(symbols,nsps,Nf,pulsetype,rolloff);
T=Ts*(length(x)-1); % signal duration
```

6.2.6 QAM modulator

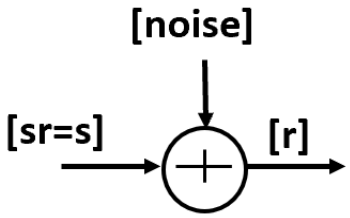
The PAM lowpass signal in input is converted to a bandpass signal by the QAM modulator.




```
%[ ************* 2-ASK modulation through the QAM modulator*
xI=x; % in-phase signal
xQ=zeros(1,length(x)); % the quadrature component is zero
s=ModQAM_2023(xI,xQ,fc,T,fs);
```

6.2.7 AWGN channel

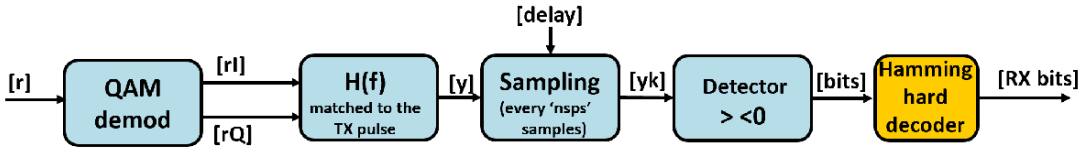
Now the signal from the transmitter is send to the noisy channel constructed with the given SNR.




```
% ************ Noise generation and sum with the received signal*
sr=s;
P=(1/length(sr))*sum(sr.^2); % Estimation of the mean power of the signal
EbN0=10^(0.1*EbN0_dB); % SNR expressed in linear units
sigman=sqrt(P*fs/(2*Br*EbN0)); % calculation of the noise's standard deviation
noise=sigman.*randn(1,length(sr)); % generation of the Gaussian noise
r=sr+noise; % addition of the noise to the received signal
```

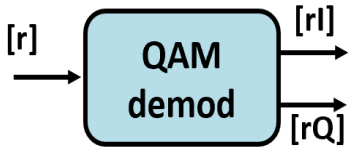
6.2.8 Receiver

The receiver is the same as the one in the Activity 5, except now we have to add a Hamming decoder block after the symbol detector.



6.2.9 QAM demodulator

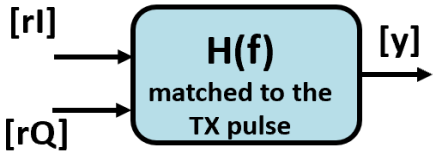
As usual thanks to noise we can have components of the received signal in the quadrature path.



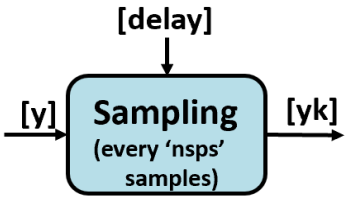

```
% ************* QAM demodulator*
[rI,rQ,DelayQAM]=DeModQAM_2023(r,fc,T,fs,psi); % demodulate the signal
```

6.2.10 Matched Filter and Sampler

The filter and sampler are still equal to what we have developed in Activity 5.




```
% ************* Matched filter*
r=rI+1i*rQ;
y=conv(r,h)/nsps; % filtering with the matched filter
```

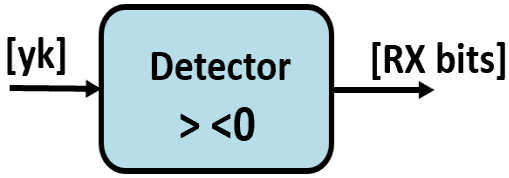



```
% ************* Sampler*
% compute the overall delay introduced by the TX and RX filters
delay=round (2*DelayPAM+DelayQAM);
```

```
% sample the signal every symbol time accounting for the delay
yk = y(delay:nsps:nsps*nCodedBits+delay-1);
```

6.2.11 BPSK symbol detector

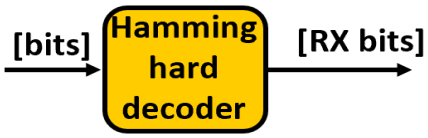
The samples are now compared to the threshold set to zero (because of 2-ASK constellation) in the detector.




```
% ************* Detector*
b=real(yk) > 0; % take only the real part
```

6.2.12 Hamming hard decoder

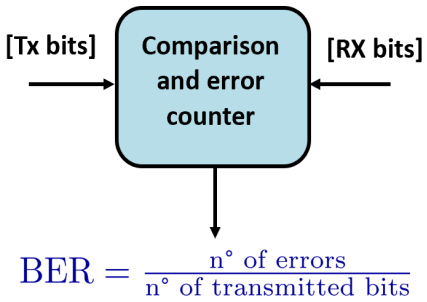
Because in the transmitter we have introduced a Hamming coder, now after the detector what we have received are codewords and in order to get the original sequence of bits, we need to decode the codewords sequence using the Hamming hard decoder.




```
% ************* Hamming decoder*
RXdata=HammingHardDecoder_2023(double(b)); % hard decoding
```

6.2.13 BER estimation

As in Activity 5 we now compare the received detected bits and the original sequence of bits from the transmitter's binary source. Counting the erroneous bits, the result we obtain is the bit error rate.

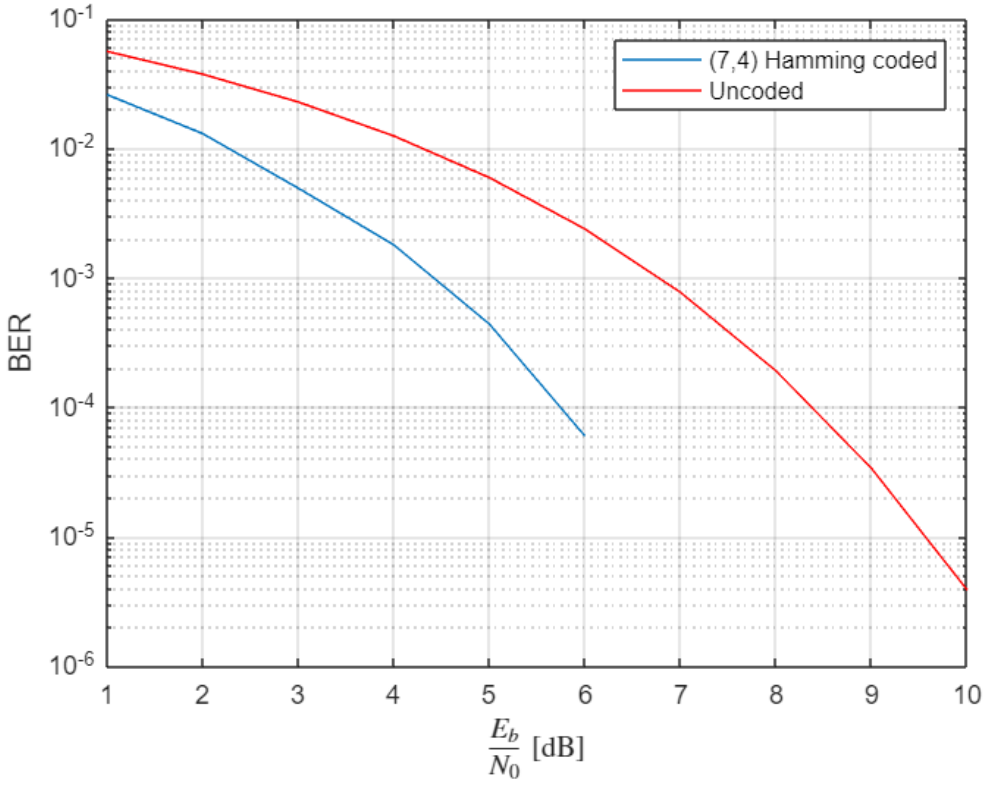


```
 % ************* Bit Error Rate (BER) estimate*
     noe=sum(abs(source_bits-RXdata)); % Errors count
     nod=length(source_bits);
     SNRdB(index)=EbN0_dB;
     ber(index)=noe/nod; % Estimate the BER
     fprintf('EbNo=%.1f (dB) BER=%g\n',SNRdB(index),ber(index));
     index=index+1;
 % ************* End of Monte Carlo simulation*
 end
```

```
EbNo=1.0 (dB) BER=0.02605
EbNo=2.0 (dB) BER=0.01304
EbNo=3.0 (dB) BER=0.00494
EbNo=4.0 (dB) BER=0.0018
EbNo=5.0 (dB) BER=0.00044
EbNo=6.0 (dB) BER=6e-05
EbNo=7.0 (dB) BER=0
EbNo=8.0 (dB) BER=0
EbNo=9.0 (dB) BER=0
EbNo=10.0 (dB) BER=0
```

6.2 BER's plot of Coded vs Uncoded scheme

```
 % ************* BER vs EbN0 plot*
 figure
 semilogy(SNRdB,ber);
 xlabel('$\frac{E_b}{N_0}$ [dB]','HorizontalAlignment','center','Interpreter','l
atex');
 ylabel('BER');
 grid on
 hold on
 SNR=10.^(0.1*SNRdB);
 Pb=0.5*erfc(sqrt(SNR));
 plot(SNRdB,Pb,'r')
 legend('(7,4) Hamming coded', 'Uncoded')
```



We can see on the plot that the Hamming coded BPSK system is better than the uncoded BPSK system. In fact, the blue curve, which displays the BER for the Hamming coded system, stays always under the red curve. It means that using Hamming coded BPSK system, the probability of detecting wrong bits is much lower than an uncoded BPSK system.