FILTERING GUIDE

Filtering biomedical signals using Matlab

23 DE ABRIL DE 2019
DAVID GONZÁLEZ MÁRQUEZ
Universidad San Pablo CEU

Filtering and removing noise

1-Obtain the signal that we want to convert

In this case, we have two main possibilities, that we already have the signal in a format readable for Matlab or that we need to convert the signal (MIT Format, for example).

Let say that I downloaded the record 222 from the MIT-BIH Arrhythmia Database. (https://www.physionet.org/physiobank/database/mitdb/)

I have the three files of the MIT Format but I need to convert them to something that I can read.

For doing that I will use the command (in the virtual machine) rdsamp.

```
rdsamp -r 222 >222.txt
```

Now I can read that file.

For example, we could write

```
signalComplete=load('222.txt')
```

And we can check using

```
>>> size(signalComplete)
ans =

   650000        3
```

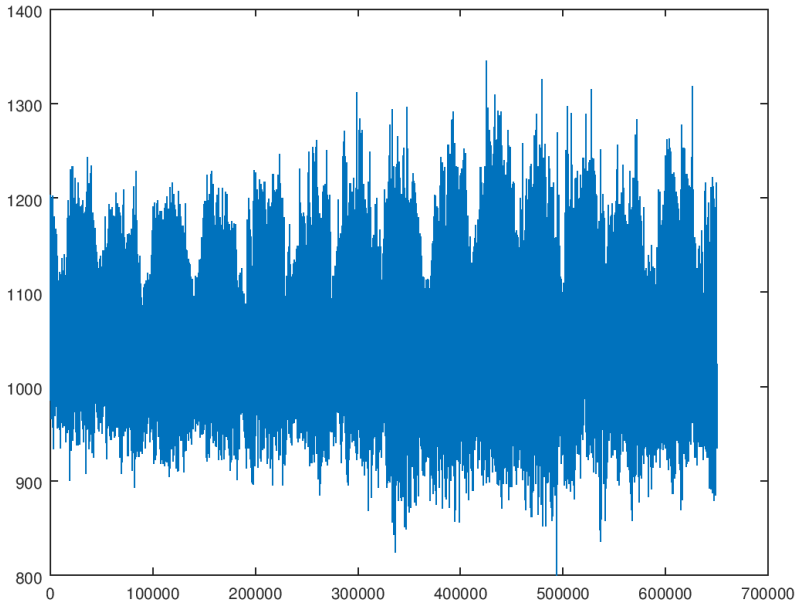
That should return the number of samples and the number of signals.

In this case, we have three signals because when we converted from the MIT Format we included a new column with the index of each sample.

To check if the signal was correctly loaded we plot the signal.

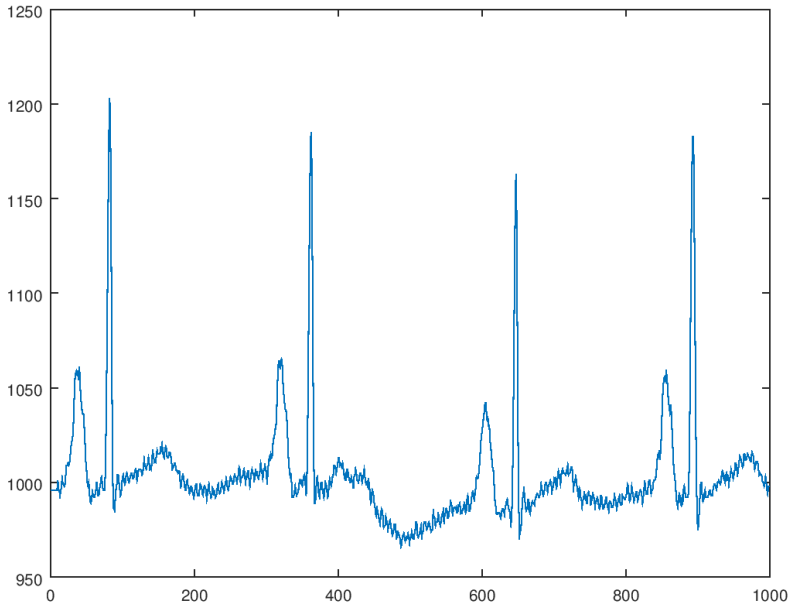
(We only plot the second signal)

```
plot(signalComplete(:,2))
```



If the signal is long maybe is better to only plot some samples to see it better.

```
plot(signalComplete(1:1000,2))
```



We can now how much time we are plotting using the sampling frequency (present in the original file) (NumberOfSamples/SamplingFrequency=seconds;1000/360=2.7778 seconds)

2-Create our filter

Typically, there are several ways of creating a filter.

Probably, the simplest filter and the most famous is the Butterworth filter.

```
[b, a] = butter (n, wc, "high")
```

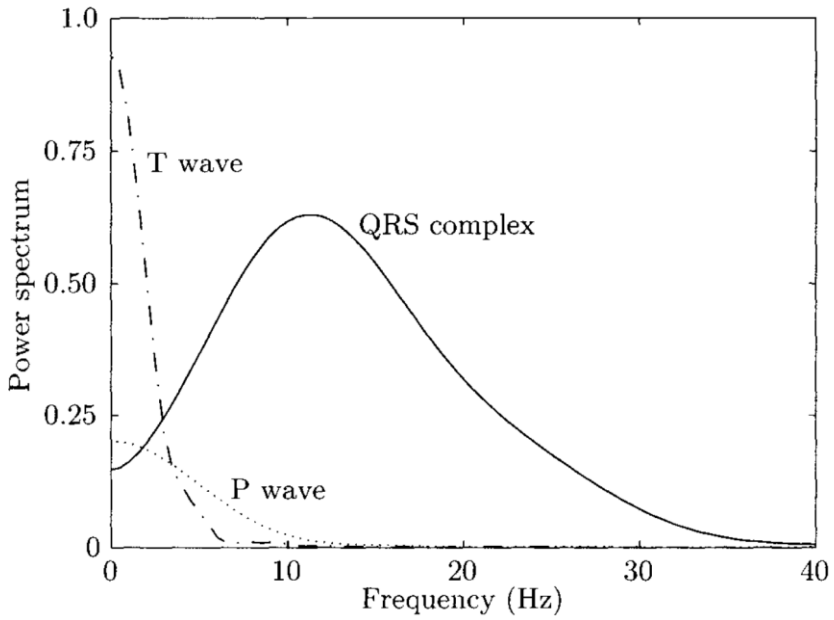
This commands obtains the coefficients for a n-order high-pass filter with a cutoff frequency=wc.

wc is the normalized cutoff frequency (between 0 and 1)

In this case, for the ECG we want to filter the baseline wander (very low frequency noise).

We should choose a cutoff frequency that allow us to remove most of the noise but that at the same time don't delete the parts of the signal that we want to preserve.

In the bibliography, authors usually choose a cutoff frequency of around 0.5Hz for the high pass filter. Using this frequency we should filter the baseline wander without removing a big part of the T and P waves.



The sampling frequency was in the header of the original signal, and in this case is 360Hz.

So, to create our filter for the baseline wander we can apply the following commands:

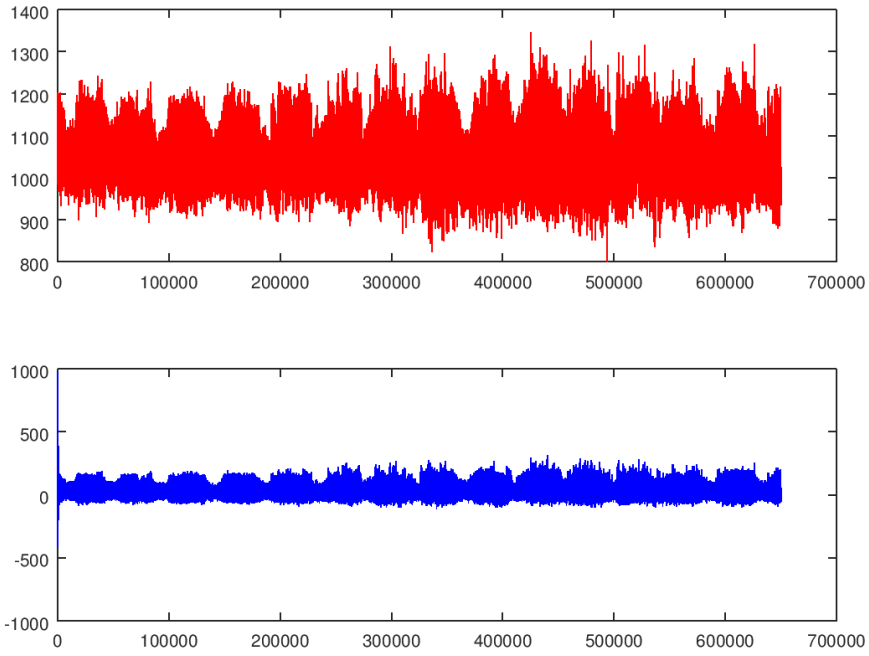
```
Fs=360
[b,a]=butter(6,(0.5)/(Fs/2),'high')
```

Now we should apply this filter over our signal:

```
signalNotFiltered=signalComplete(:,2);
signalFiltered= filter(b,a,signalNotFiltered);
```

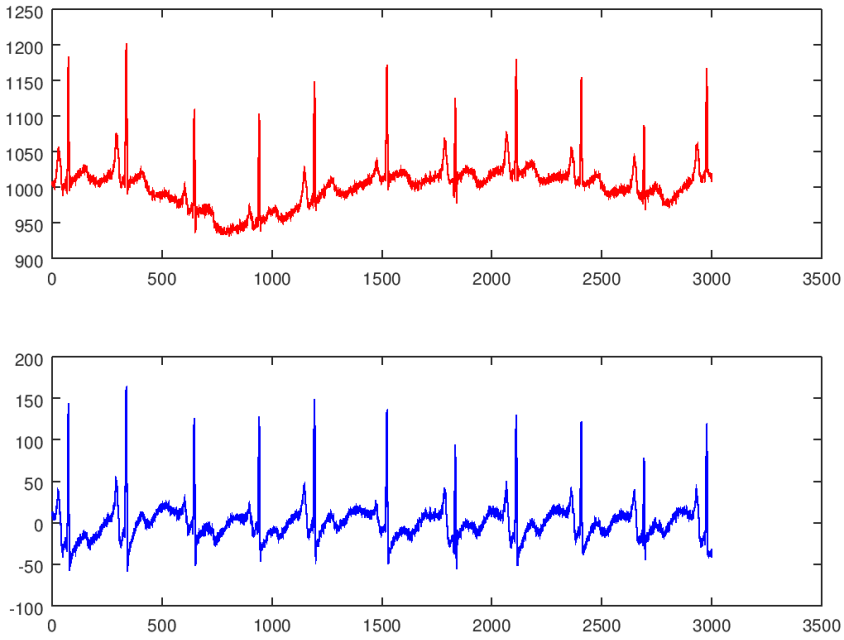
We can now plot both signals to check if we removed the baseline wander:

```
subplot(2,1,1)
plot(signalNotFiltered,'color','red')
subplot(2,1,2)
plot(signalFiltered,'color','blue')
```



It seems that the signal has changed. To see better the difference after filtering we can plot only some heartbeats.

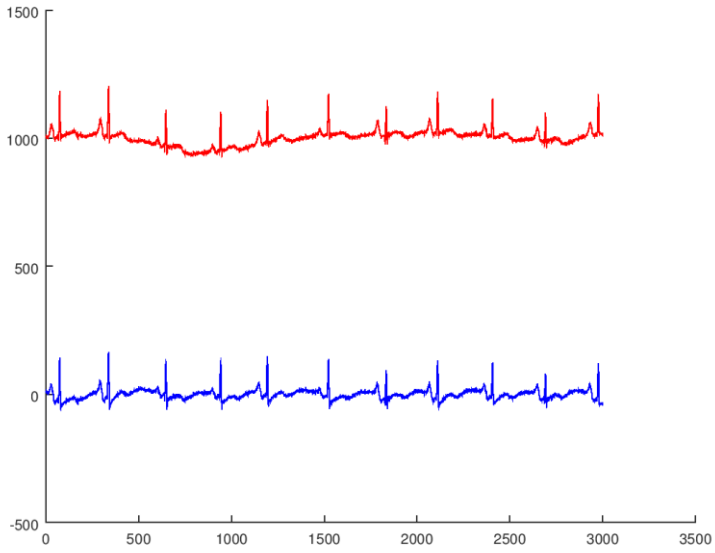
```
subplot(2,1,1)
plot(signalNotFiltered(2000:5000),'color','red')
subplot(2,1,2)
plot(signalFiltered(2000:5000),'color','blue')
```



As you can see, we are indeed removing most of the baseline wander noise but at the cost of disturbing slightly the signal (making P waves bigger).

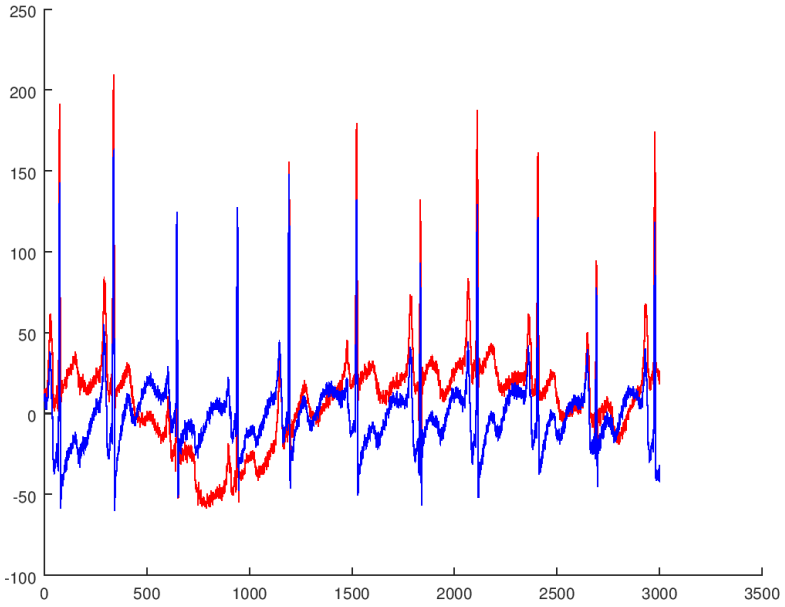
If we want, we can also plot both signals together:

```
subplot(1,1,1)
hold('on')
plot(signalNotFiltered(2000:5000),'color','red')
plot(signalFiltered(2000:5000),'color','blue')
```



The original signal has a static component that is removed after filtering, to print one signal over the other we can subtract the means.

```
newplot()

subplot(1,1,1)

hold('on')

plot(signalNotFiltered(2000:5000)-
mean(signalNotFiltered),'color','red')

plot(signalFiltered(2000:5000)- mean(signalFiltered),'color','blue')
```



To remove high frequency noise, we can proceed in an analogous way, but we should use a low-pass filter with a cutoff frequency around 40-45Hz.

3-Obtaining the frequency response of our filter

As we are filtering in frequency it should useful to obtain a representation of our filters and our signals in frequency also.

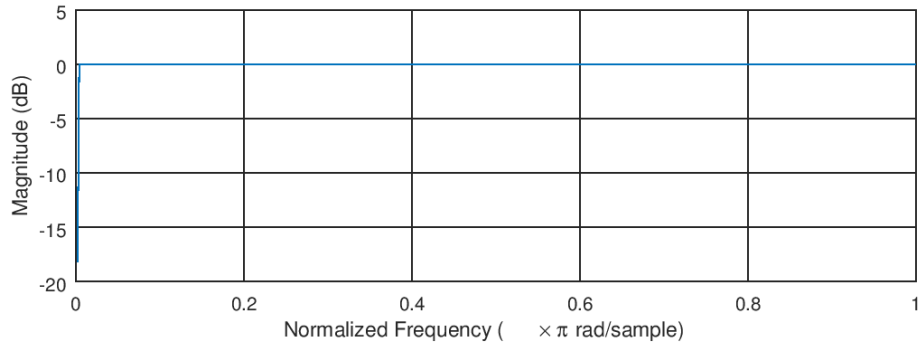
There are several ways of doing this in Matlab/Octave.

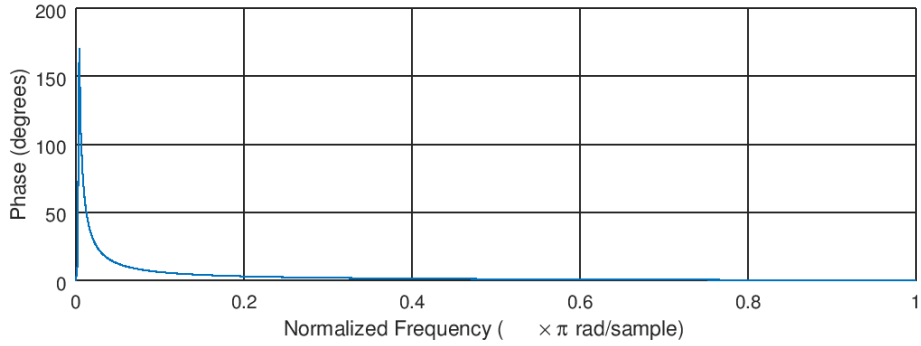
Bear in mind that some commands are limited to certain versions of Matlab, and that typically you will need the 'Signal Processing Toolbox' (normally installed with Matlab)

For filters, we have the command *freqz* that show the frequency response of the filter and the phase response.

```
[b,a]=butter(6,(0.5)/(Fs/2),'high')
```

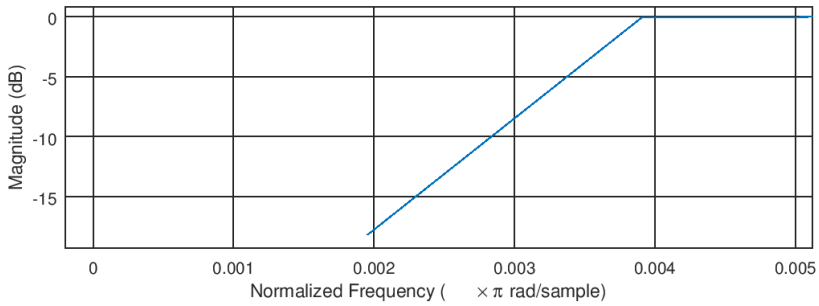
```
freqz(b,a)
```

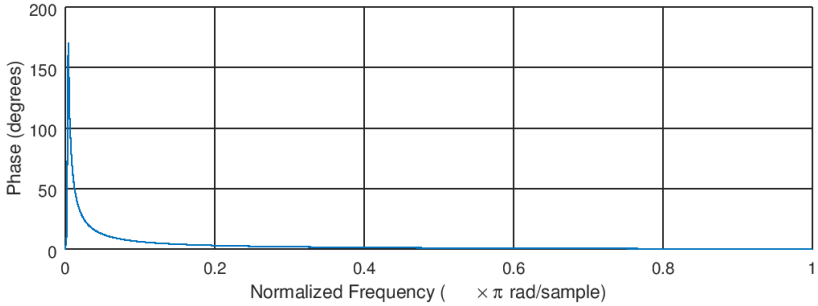




In this case as the cutoff frequency is quite low in frequency is difficult to see anything.

However, if we make zoom in frequency we will see this:

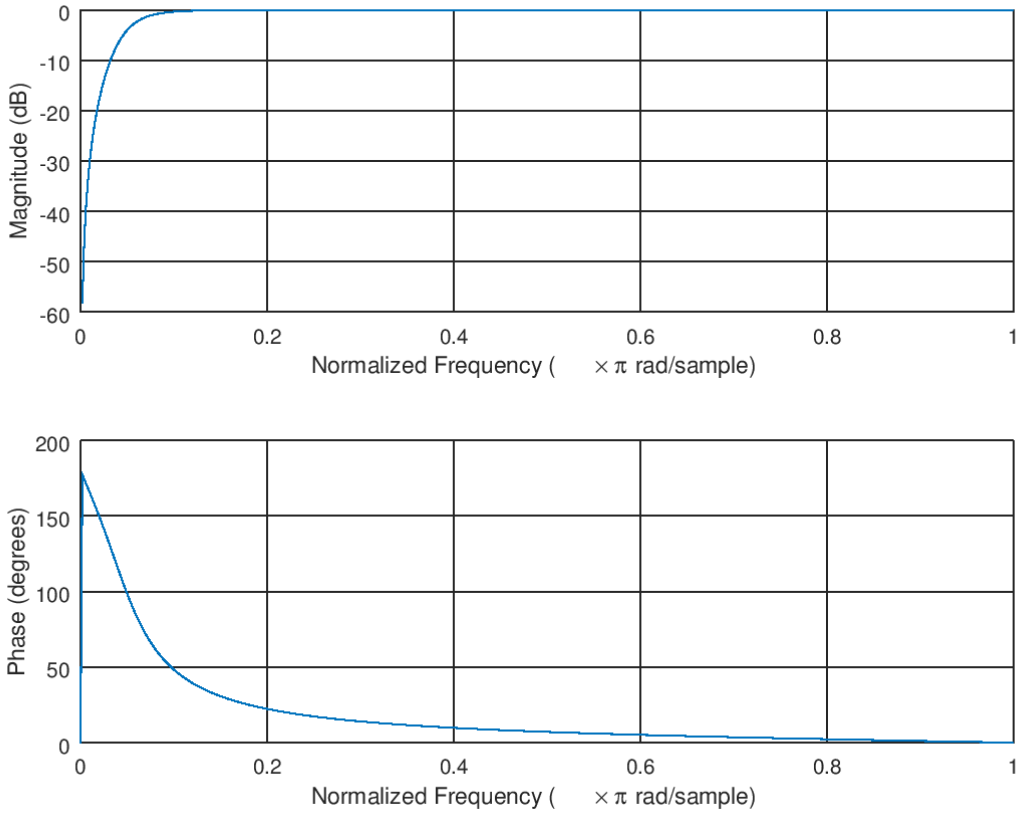




The frequency response change between 0.002 and 0.004 (normalized frequency). In the range of the cutoff frequency that we established 0.5/(360/2)=0.0028

Let's try now with a higher cutoff frequency and a lower order for the filter.

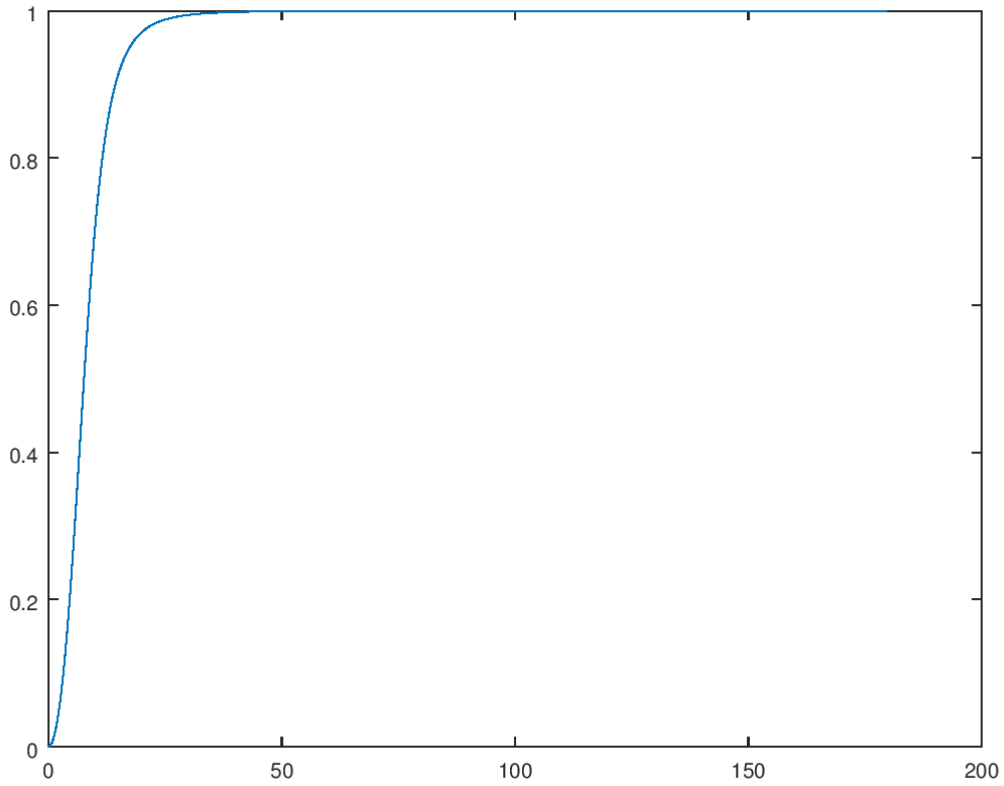
```
[b,a]=butter(2,(10)/(Fs/2),'high')
freqz(b,a)
```



Now the cutoff frequency is 10/(360/2)=0.055556, and as we have decremented the order of the filter now is less similar to the ideal filter.

We could also print the frequency response with the axis already converted to our frequencies.

```
Fs=360
[h,w]=freqz(b,a)
newplot()
w=(w/pi)*(Fs/2)
plot(w,abs(h))
```



Another option, instead of using *freqz* is to obtain ourselves the response of the filter.

The general transfer function of a filter is given by:

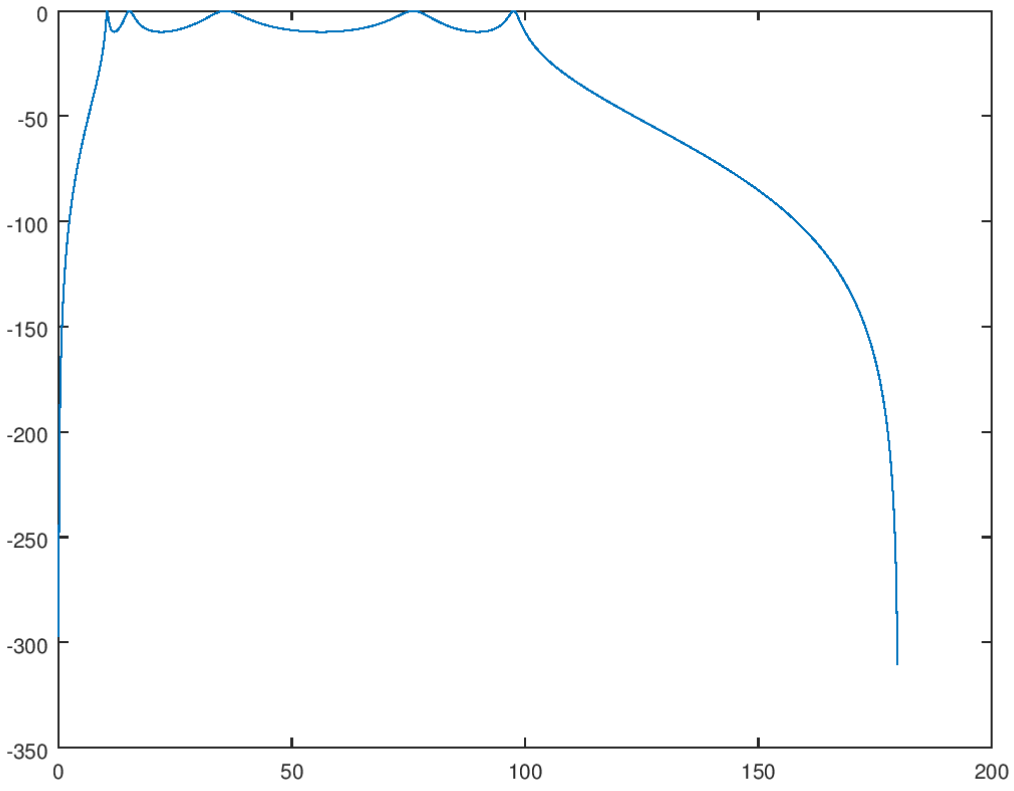
$$H(z) = \frac{\sum_{k=0}^{M} b_k z^{-k}}{\sum_{k=0}^{N} a_k z^{-k}}$$

Where we substitute z=e^(jw) to evaluate the transfer function on the unit circle:

$$H(e^{j\omega}) = \frac{\sum_{k=0}^{M} b_k e^{-j\omega k}}{\sum_{k=0}^{N} a_k e^{-j\omega k}}$$

```
%% Calculate frequency response

Fs=360

[b,a]=cheby1(5,10,[(10)/(Fs/2) (100)/(Fs/2)] ) %In this case we use
a Chebyshev filter bandpass (ripple should appear)

N = 1024; % Number of points used to evaluate the filter
```

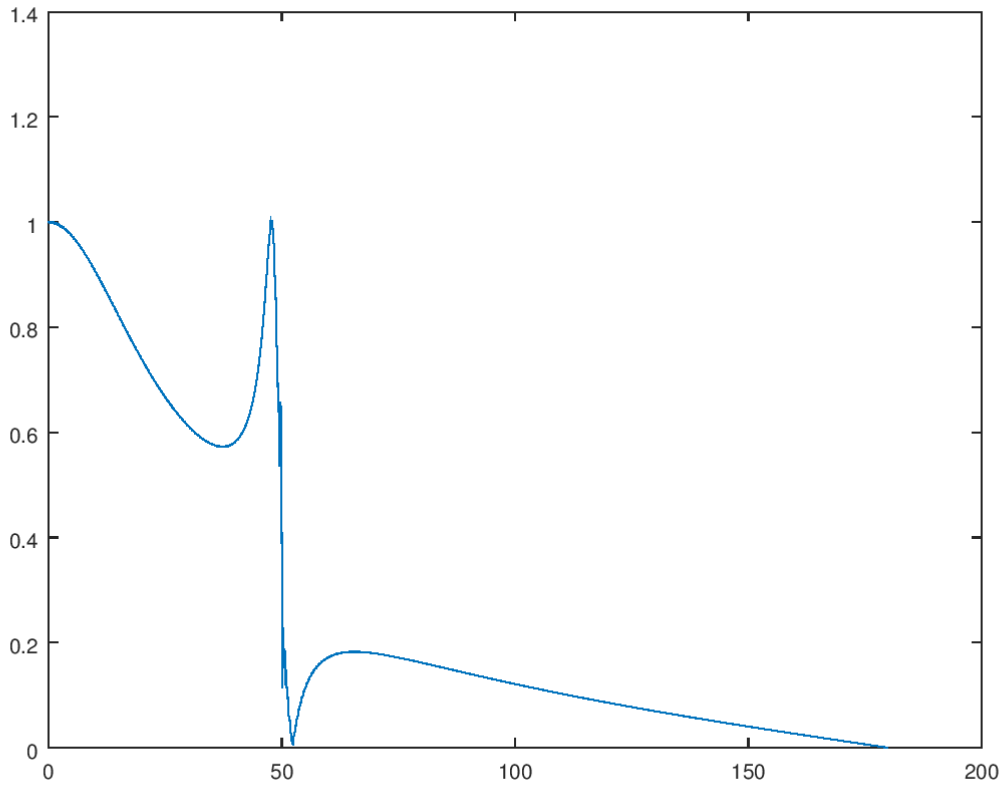
```
upp = pi; % We evaluate only until 180 (Fs/2)

w = linspace(0, pi, N+1); %Vector of frequencies that will be used
to evaluate the filter.

w(end) = []; %We remove the last element (pi)

ze = exp(-1j*w); % Pre-compute all exponents

H = polyval(b, ze)./polyval(a, ze); % Evaluate in the filter
polynomials using our values

Ha = abs(H);

Hdb  = 20*log10(Ha/1); % Convert to dB scale

wn   = w/pi;

%Plot

wn=wn*(360/2);%To show frequencies between 0 and 180.

plot(wn, Hdb) %Our filter between 10 and 100
```



Another way of obtaining the filter response is by using the fundamental property of the filter. That the filter response is a Fourier Transform of its impulse response δ(t).

Then, if we feed an impulse signal δ(t) to our filter we can calculate the response of the filter using the FFT.

```
N = 1024; % Number of points in our impulse used to evaluate the
filter

[b,a]=ellip(5,5,15,50/(Fs/2)) % In this case we have a low-pass
elliptical filter with ripples of 5 and 15 dB and a cutoff frequency
of 50 Hz.


d = [zeros(1,N) 1 zeros(1,N-1)]; %Our impulse signal

h = filter(b, a, d); %We filter our impulse

HH = abs(fft(h)); %FFT of the signal filtered

HH = HH(1:N); %We select only the points that we want

w_plot = linspace(0, (Fs/2), N+1); %Vector of labels for our plot

w_plot (end) = []; %We remove the last element


plot(w_plot,HH)
```

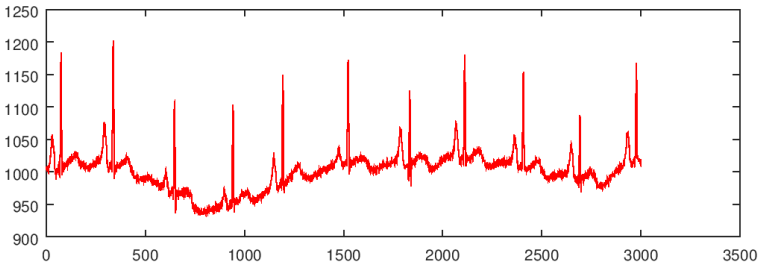


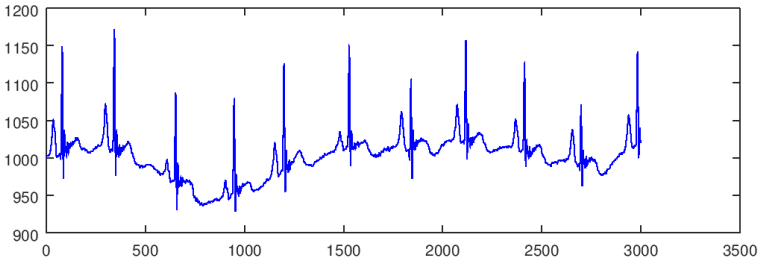
4-Obtaining the frequency representation of a signal

Sometimes is useful to obtain a frequency representation of our signal. This representation will allow us to check if the frequencies that we want to remove have disappeared or not and if our filter is working.

First of all, we define our signal and our filter. In this case, we are going to remove the high frequency noise of the ECG using a Chebyshev filter.

```
%%We load our signal
Fs=360;
signalComplete=load('222.txt');
signalNotFiltered=signalComplete(:,2);
%%We create the filter
[b,a] = cheby1(5,2,45/(Fs/2)); %Low pass filter at 45 Hz with a
ripple of 10 dB

%And we apply the filter
signalFiltered= filter(b,a,signalNotFiltered);

%And we represent our signal in time.
subplot(2,1,1)
plot(signalNotFiltered(2000:5000),'color','red')
subplot(2,1,2)
plot(signalFiltered(2000:5000),'color','blue')
```





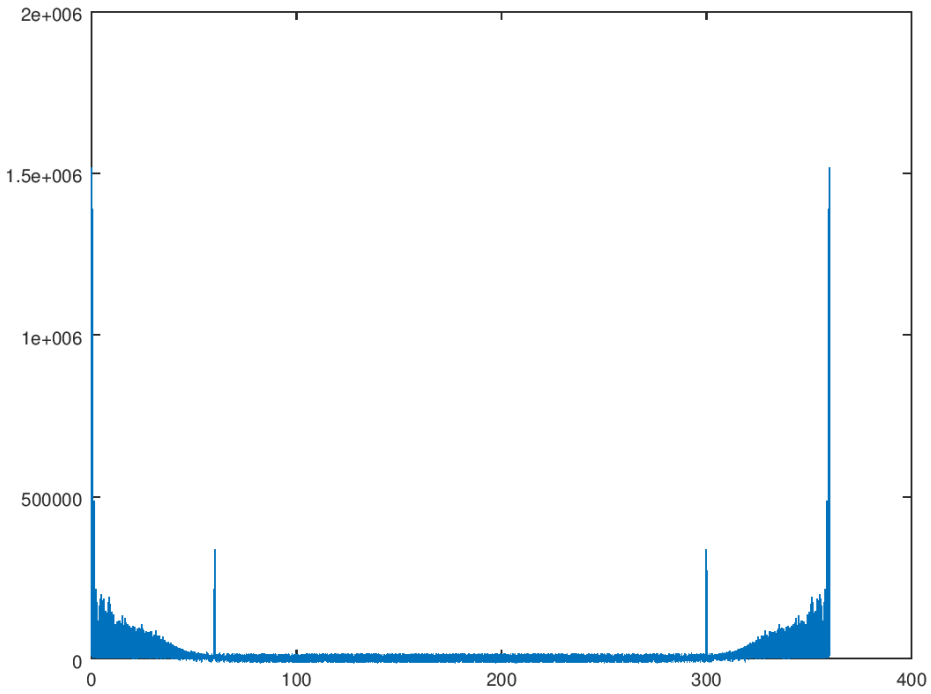
We can see that the high frequency noise has been reduced in time, but it will be better to check if in frequency there is also a difference.

For obtaining the frequency representation we will use the FFT.

```
N= length(signalNotFiltered)

ffSignalNotFiltered = abs(fft(signalNotFiltered,N)); %FFT of the
signal filtered

w_plot = linspace(0, Fs, N+1); %Vector of labels for our plot with
all the Fs

w_plot (end) = []; %We remove the last element

plot(w_plot, ffSignalNotFiltered)
```

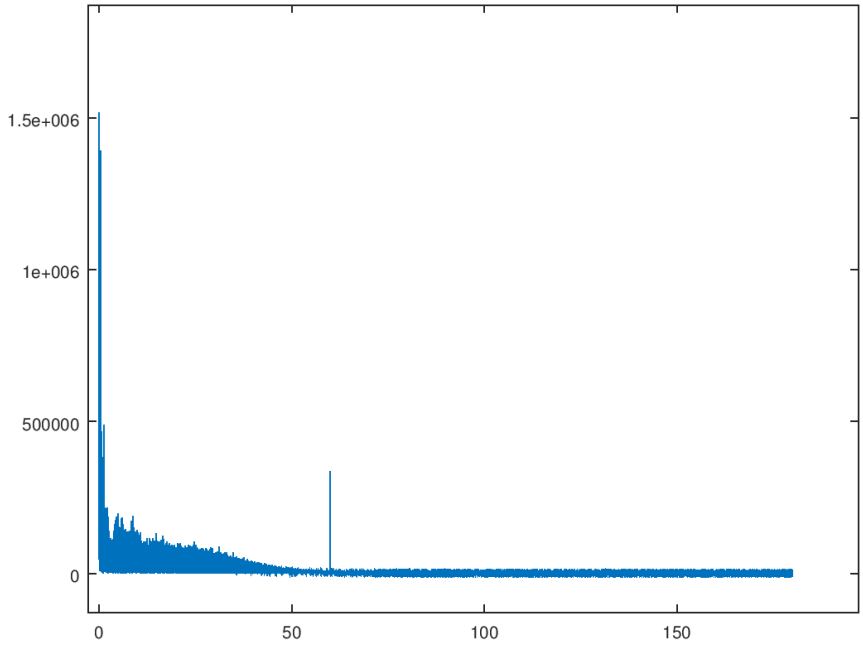
However, if we plot like this is difficult to see anything so we can plot after the two or three first samples (this peak is due to the DC component):

```
plot(w_plot(2:end), ffSignalNotFiltered(2:end))
```



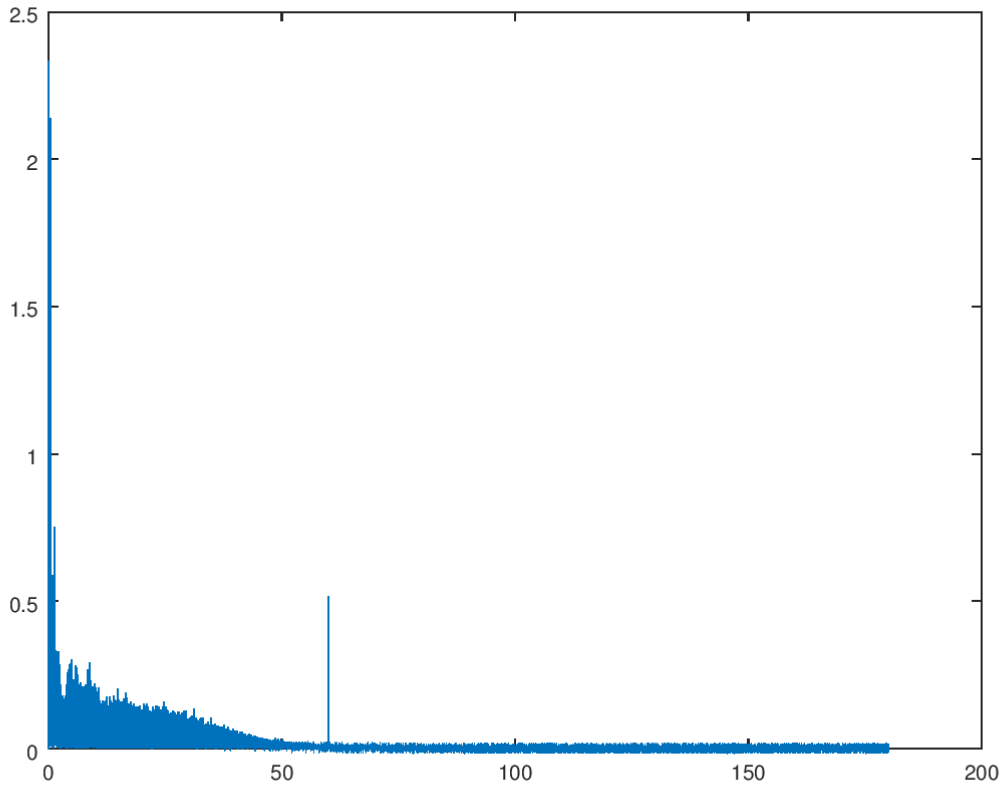
Here we can see something at least, but if we represent only half of the spectrum we will see it better.

```
plot(w_plot(2:N/2), ffSignalNotFiltered(2:N/2))
```



To obtain a better representation we are going to remove the DC component of the signal before applying the FFT. We will also divide the signal by N to obtain the Magnitude Response in the correct units.

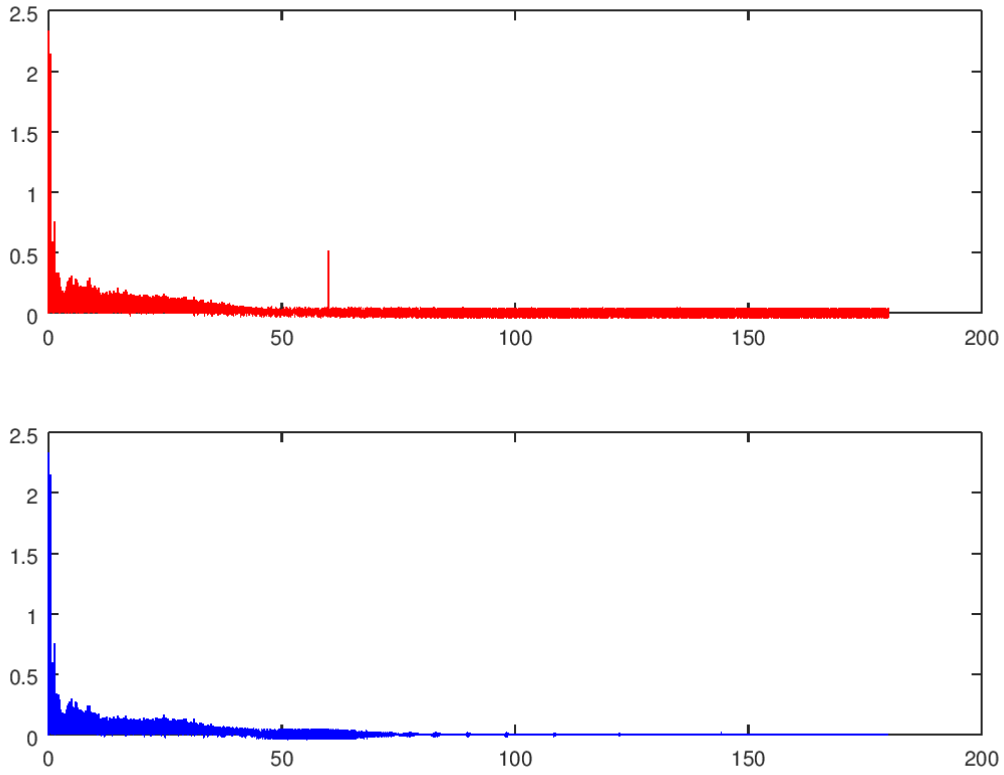
```
N= length(signalNotFiltered)

ffSignalNotFiltered = abs(fft(signalNotFiltered-
mean(signalNotFiltered),N)); %FFT of the signal filtered

w_plot = linspace(0, Fs, N+1); %Vector of labels for our plot with
all the Fs

w_plot (end) = []; %We remove the last element

plot(w_plot(1:N/2), (ffSignalNotFiltered(1:N/2)/N))
```



Using this procedure, we can plot the spectrum of the signal before and after filtering.

```
N= length(signalNotFiltered)

ffSignalNotFiltered = abs(fft(signalNotFiltered-
mean(signalNotFiltered),N)); %FFT of the signal filtered

ffSignalFiltered = abs(fft(signalFiltered-mean(signalFiltered),N));
%FFT of the signal filtered


w_plot = linspace(0, Fs, N+1); %Vector of labels for our plot with
all the Fs
```

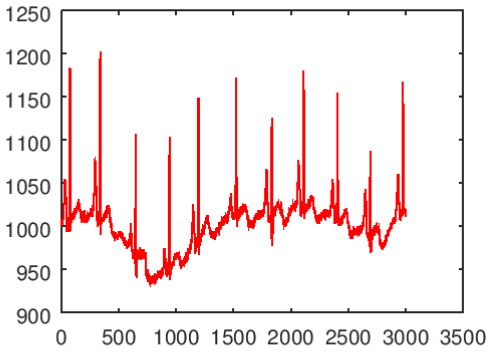
```
w_plot (end) = []; %We remove the last element
subplot(2,1,1)
plot(w_plot(1:N/2), (ffSignalNotFiltered(1:N/2)/N),'color','red')
subplot(2,1,2)
plot(w_plot(1:N/2), (ffSignalFiltered(1:N/2)/N),'color','blue')
```

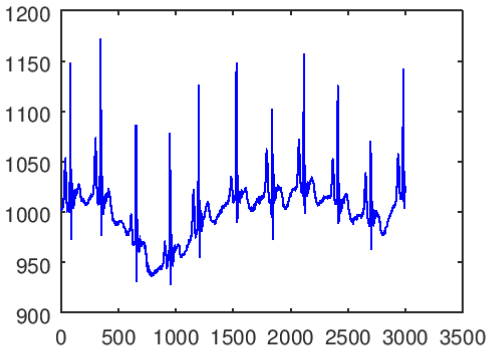


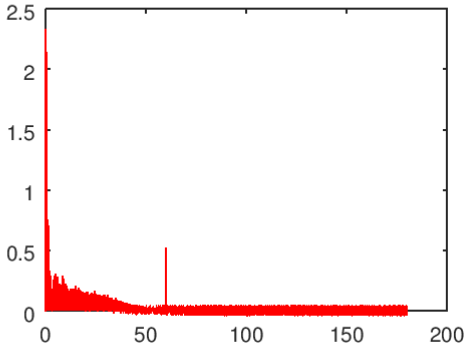
And here finally we can see that our filter is working and removing the high-frequency noise.

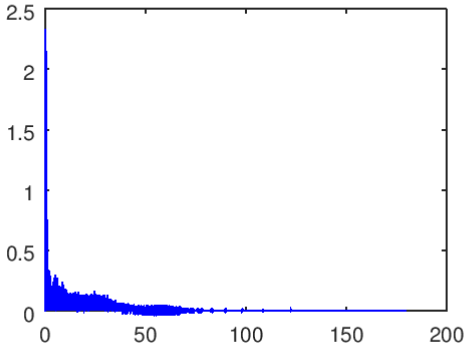
We could plot time and frequency at the same time if we wanted.:

```
subplot(2,2,1)
plot(signalNotFiltered(2000:5000),'color','red')
subplot(2,2,2)
plot(signalFiltered(2000:5000),'color','blue')

subplot(2,2,3)
plot(w_plot(1:N/2), (ffSignalNotFiltered(1:N/2)/N),'color','red')
subplot(2,2,4)
plot(w_plot(1:N/2), (ffSignalFiltered(1:N/2)/N),'color','blue')
```


1250
1200
1150
1100
1050
1000
950
900
0
500
1000
1500
2000
2500
3000
3500


1200
1150
1100
1050
1000
950
900
0
500
1000
1500
2000
2500
3000
3500


2.5
2
1.5
1
0.5
0
0
50
100
150
200


2.5
2
1.5
1
0.5
0
0
50
100
150
200

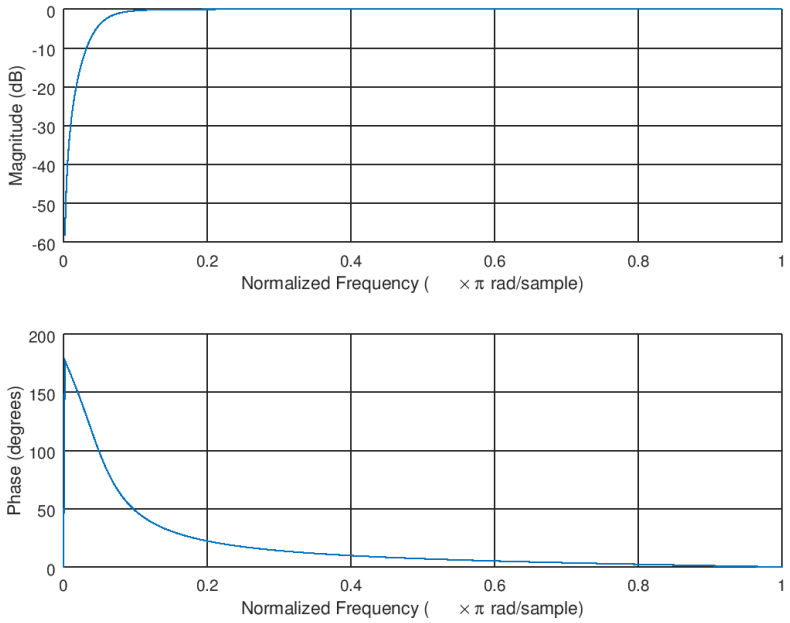
5-Some details about filtering

Zero-phase filtering

The filters that we used until now, usually don't have a constant phase response.

For example, if we look at this Butterworth filter

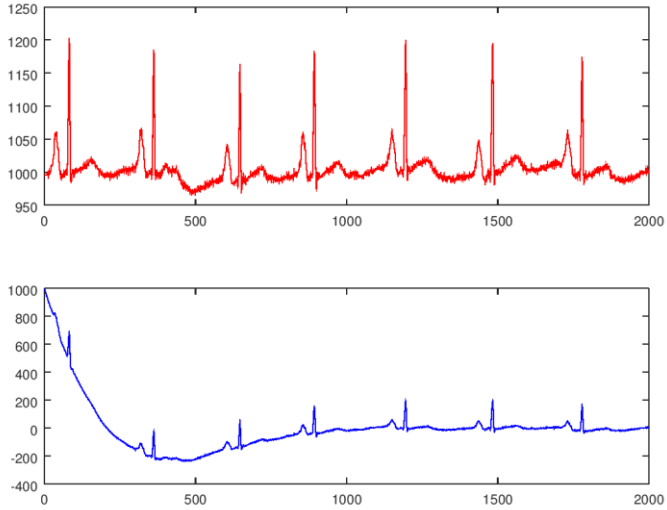
```
[b,a]=butter(2,(10)/(Fs/2),'high')
freqz(b,a)
```



We can see that the phase response is not constant.

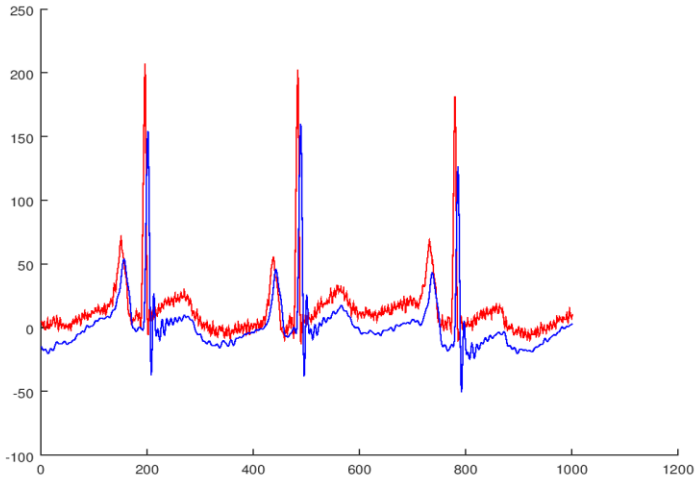
If we look closely at the signal filtered we can see some delay caused by the filter and a small distortion at the start of the signal.

```
Fs=360;
[b,a]=butter(2,(0.3)/(Fs/2),'high') %Filtering only a little


signalComplete=load('222.txt');
signalNotFiltered=signalComplete(:,2);
signalFiltered= filter(b,a,signalNotFiltered);


subplot(2,1,1)
plot(signalNotFiltered(1:2000),'color','red')
subplot(2,1,2)
plot(signalFiltered(1:2000),'color','blue')
```



To emphasize the distortion, we are going to apply two filters, low-pass and high-pass.

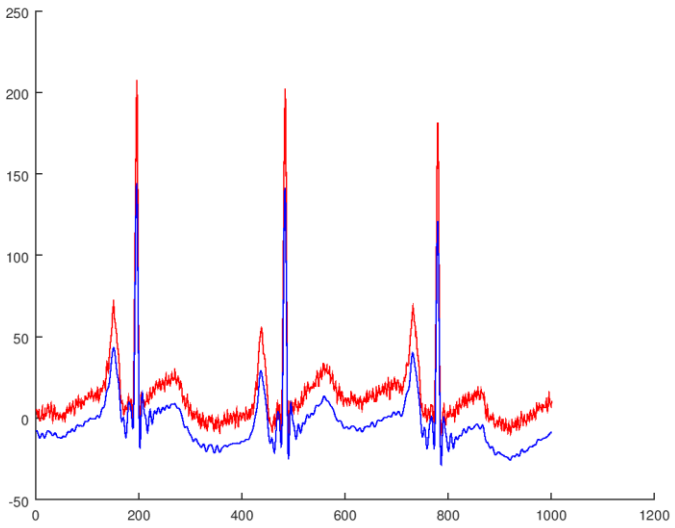
```
[b,a]=butter(2,(0.3)/(Fs/2),'high') %Filtering only a little

signalFiltered= filter(b,a,signalNotFiltered);

[b,a] = cheby1(5,2,45/(Fs/2)); %Low pass filter at 45 Hz with a
ripple of 2 dB

signalFilteredFiltered= filter(b,a, signalFiltered);

subplot(1,1,1)

newplot()

hold('on')

plot(signalNotFiltered(1000:2000)-
mean(signalNotFiltered),'color','red')

plot(signalFilteredFiltered (1000:2000),'color','blue')
```



If we look closely we can see that the peaks are not in the same place in both signals.

To avoid this effect, the simplest solution is to filter the signal both in forward and reverse directions. Matlab and Octave allow us to do this operation by using the command *filtfilt*.

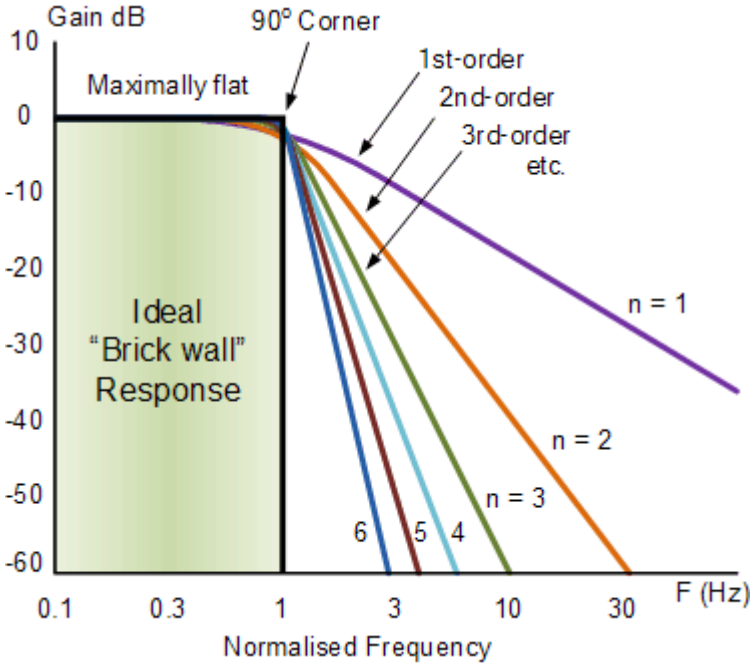
```
[b,a]=butter(2,(0.3)/(Fs/2),'high') %Filtering only a little

signalFiltered= filtfilt(b,a,signalNotFiltered);

[b,a] = cheby1(5,2,45/(Fs/2)); %Low pass filter at 45 Hz with a
ripple of 10 dB

signalFilteredFiltered= filtfilt (b,a, signalFiltered);


subplot(1,1,1)

newplot()

hold('on')

plot(signalNotFiltered(1000:2000)-
mean(signalNotFiltered),'color','red')

plot(signalFilteredFiltered (1000:2000),'color','blue')
```



Now the peaks of both signals are perfectly aligned.

Order of filters

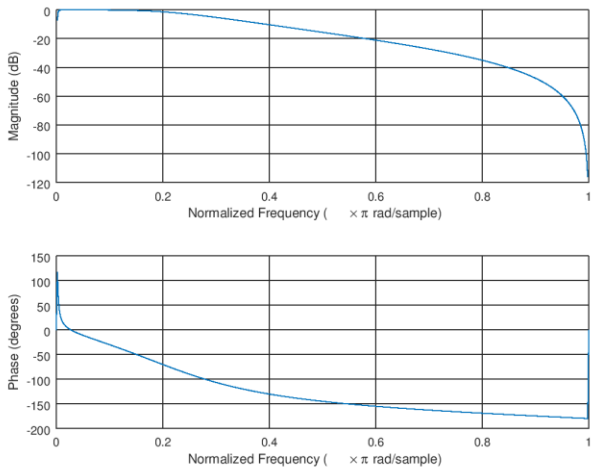
By increasing the order of a filter, we can change the attenuation curve of the filter and try to make it more similar to our ideal filter.



For example, we can see the difference in our filter between order 2 and 8.

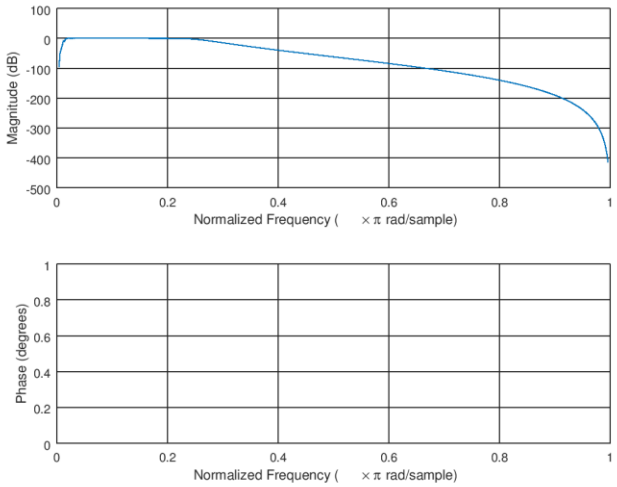
```
[b,a]=butter(2,[(0.5)/(Fs/2) (45)/(Fs/2)]) %Filtering low and high frequency noise.
```

```
freqz(b,a)
```

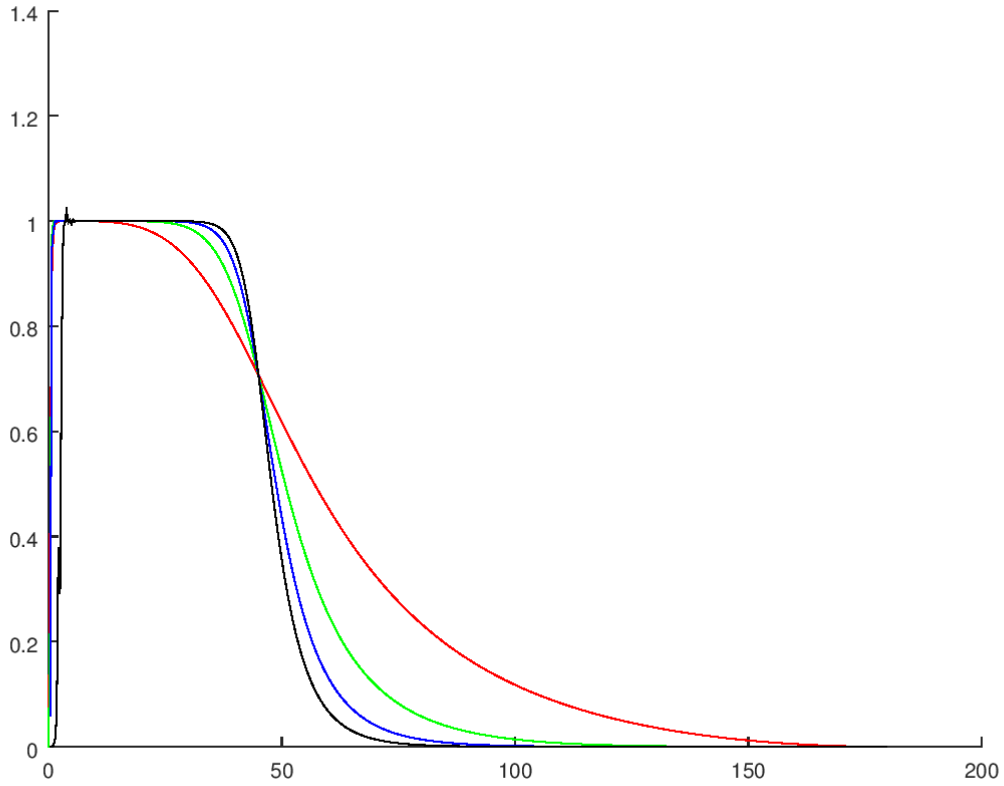



```
[b,a]=butter(8,[(0.5)/(Fs/2) (45)/(Fs/2)]) %Filtering low and high frequency noise.
```

```
freqz(b,a)
```




```
Fs=360

[b,a]=butter(2,[(0.5)/(Fs/2) (45)/(Fs/2)]) %Filtering low and high
frequency noise.

[h1,w1]=freqz(b,a)

[b,a]=butter(4,[(0.5)/(Fs/2) (45)/(Fs/2)]) %Filtering low and high
frequency noise.

[h2,w2]=freqz(b,a)

[b,a]=butter(6,[(0.5)/(Fs/2) (45)/(Fs/2)]) %Filtering low and high
frequency noise.

[h3,w3]=freqz(b,a)

[b,a]=butter(8,[(0.5)/(Fs/2) (45)/(Fs/2)]) %Filtering low and high
frequency noise.

[h4,w4]=freqz(b,a)


newplot()

hold('on')

plot((w1/pi)*(Fs/2),abs(h1),'color','red')

plot((w2/pi)*(Fs/2),abs(h2),'color','green')

plot((w3/pi)*(Fs/2),abs(h3),'color','blue')

plot((w4/pi)*(Fs/2),abs(h4),'color','black')
```



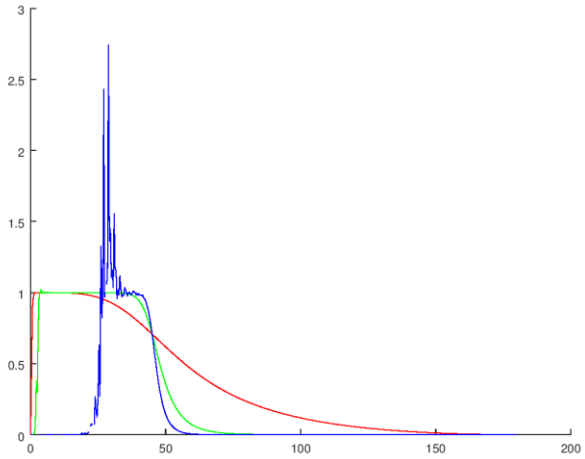
As we can see with a higher order the filter is more similar to the ideal filter. The curve of the filter is less smooth and we are making more abrupt the filtering around the cutoff frequency.

(The graph can be different depending on your computer and program)

However, there are also problems associated with increasing the order of the filter. The calculus is each time more complex and we could overrun the finite precision of our computer. Also, as we increase the order of the filter stability problems start to appear.

```
Fs=360

[b,a]=butter(2,[(0.5)/(Fs/2) (45)/(Fs/2)]); %Filtering low and high
frequency noise.

[h1,w1]=freqz(b,a);

[b,a]=butter(8,[(0.5)/(Fs/2) (45)/(Fs/2)]); %Filtering low and high
frequency noise.

[h2,w2]=freqz(b,a);

[b,a]=butter(16,[(0.5)/(Fs/2) (45)/(Fs/2)]); %Filtering low and high
frequency noise.

[h3,w3]=freqz(b,a);
```

```
newplot()
hold('on')
plot((w1/pi)*(Fs/2),abs(h1),'color','red')
plot((w2/pi)*(Fs/2),abs(h2),'color','green')
plot((w3/pi)*(Fs/2),abs(h3),'color','blue')
```



In this case we are not able to calculate correctly the filter of 16-order.

6-Signals typical filters.

ECG

Low-frequency filtering

The AHA (American Heart Association) said that in routine filters we should only use a cutoff frequency of 0.05Hz, with the objective of preserving the fidelity of repolarization in the ST segment.

However, it also said that this requirement could be relaxed to 0.67Hz for linear filters with zero phase distortion.

The ANSI (Organization for standardization) also endorsed this limits.

High-frequency filtering

The ANSI/AAMI standard in this case is to recommend a high-frequency cutoff of a least 150Hz for all standard 12-lead ECGs. For children this cutoff frequency should be extended to 250Hz in children.

However, if the ECG is not going to be used to measure amplitude for diagnostic classification a cutoff frequency of 40Hz could be used in some especial cases. Nevertheless, the devices should alert the user in case of using a cutoff frequency lower that the recommendation of 150Hz and if needed the filtering using 40Hz should be done afterwards.

EMG

Low-frequency filtering

Typical frequencies for the high pass filters are between 5 and 20 Hz (surface EMG).

ISEK (International Society of Electrophysiology and Kinesiology) recommends a high-pass filter with a cutoff frequency of 5Hz for surface EMG and of 1500Hz for intramuscular (needle) EMG.

High-frequency filtering

Typical values for filtering the high frequencies are between 200Hz and 1000Hz (surface EMG)

ISEK recommends a low-pass filter with a cutoff frequency of 500Hz for surface EMG. For intramuscular EMG there is no recommendation.

EEG

In the EEG, typically we use bandpass filters to process each type of wave separately.

(There bands are approximate, there is no consensus on this)

Infra-Low (<0.5Hz)

Delta Waves (0.5Hz to 3Hz)

Theta-Waves (3Hz to 8Hz)

Alpha Waves (8Hz to 12Hz)

Beta Waves (12Hz to 38Hz)

Gamma Waves (38Hz to 42Hz)

Low-frequency filtering

To reduce the effect of voltage from the galvanic skin response across the head frequencies below 5Hz could be removed. However, we are perturbing the theta waves and removing the delta waves. In small animals the frequencies of the waves are even lower so we should use a lower cutoff frequency.

High-frequency filtering

The activity of the brain is mostly limited to the band of the 0-40Hz. So usually low pass filters are applied to remove higher frequencies. Typically, the cutoff frequency of those filters is around 45Hz to remove the possible powerline interference at 50/60Hz.