Applied signal processing laboratory

(A.Y. 2024/25)

Assignment 2

Digital filter design

Authors:
Castorina Giovanni (307594)
Fanton Vittoria (308625)
Giordano Andrea (310679)

Contents

1 Introduction

In this report, different topics of signal processing are analyzed and implemented.
The first section deals with the design of digital filters, starting from standard implementations and moving towards a custom solution through the creation of the function `my_filter()`. After its implementation, `my_filter()` is tested and compared with the built-in functions to evaluate its performance and highlight possible differences.

The second section focuses on FIR filter design using the window method. Different types of filters are developed, including band-pass and band-stop filters, and particular attention is given to the use of the Kaiser window for the design of filters with specific performance requirements. For each case, the design steps are explained, and the resulting frequency responses are analyzed.

In the third section, the filtering techniques developed are applied to the processing of biosignals. Starting from an original signal, different filters are applied to remove noise and isolate the useful components. A low-pass filter is used to eliminate high-frequency noise, while a high-pass filter is applied to remove low-frequency components. After preprocessing, additional analyses are carried out to compute the pulse rate and the oxygen saturation.

In general, throughout the report, theoretical concepts are directly linked to their practical implementation.

2 Filter Design

2.1 Design and Implementation

The first exercise focuses on the design and analysis of digital filters applied to a composite signal. Initially, a signal is synthesized by combining a low-frequency cosine, a frequency-modulated component involving a Dirichlet function and additive white Gaussian noise. The task begins with generating a 20-second-long signal sampled at 1 kHz, followed by the estimation of its power spectral density using Welch's method. The subsequent steps involve designing and analyzing a band-pass filter centered around 250 Hz using three different techniques, Butterworth, Elliptic, and equiripple FIR, each chosen to illustrate different filter characteristics in terms of selectivity, ripple, and transition steepness.

In addition, a Chebyshev type I low-pass filter with a 6 Hz pass-band is designed and its frequency response plotted. Finally, a custom function, `my_filter`, is developed to filter an input signal based on a digital filter represented in the z-transform form of a linear constant-coefficient difference equation (LCCDE). This function is tested under various scenarios to confirm its accuracy and robustness.

Therefore, the first step is to define the signal `X(t)` as follows:

```
x = cos(2*pi*f1*t) + 3*(D3.^2).*cos(2*pi*f2*t) + W;
```

According to the following specifications:

- $f_1 = 5$ Hz;
- $f_2 = 250$ Hz;
- $B_D = 10$ Hz;
- $f_s = 1$ kHz;
- D_3 is a Dirichlet function of order 3;
- W is a zero-mean white Gaussian noise with variance $\sigma^2 = 10$.

Moreover, the time-axis t is specifically defined to generate a 20 seconds portion of the signal. Subsequently, the power spectral density (PSD) $10log_{10}|S_x(f)|$ was plotted in logarithmic scale, as reported in Figure 1, where the vector $S_x(f)$ represent the Welch periodogram obtained segmenting the signal into 25 Hamming windows with 50% overlapping.

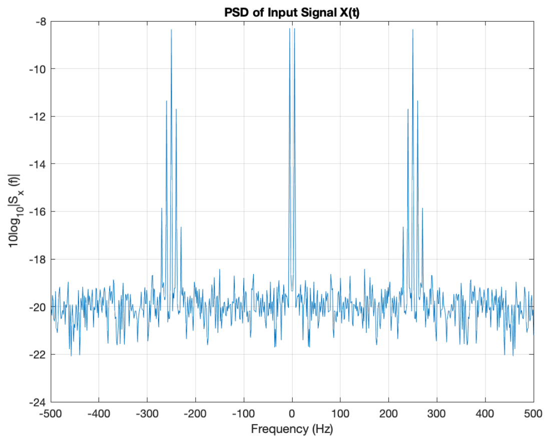


Figure 1: Power Spectral Density (PSD) of the input signal `X(t)` computed using Welch's method. The signal is segmented into 25 Hamming windows with 50% overlap over a 20-second duration, and the PSD is displayed in logarithmic scale.

Next, a Butterworth band-pass filter is designed to have a pass-band of 80 Hz centered at 250 Hz. This is realized through the built-in function `buttord()` first, in order to estimate the filter order, and the function `butter()` then to obtain the actual transfer function coefficients. The first function takes as inputs the following parameters:

- Wp: Pass-band cutoff frequencies, defined as $f_{central} \pm \frac{passband}{2}$ and then normalized by $\frac{f_s}{2}$;
- Ws: Stop-band cutoff frequencies, defined as $f_{central} \pm (\frac{passband}{2} + transition\_band)$ and then normalized by $\frac{f_s}{2}$;
- Rp: Pass-band ripple, specified to be 1 dB;
- Rs: Stop-band attenuation, specified to be 60 dB;

And the outputs obtained are the lowest filter order and the normalized cutoff frequencies, which are then used, after denormalizing the cutoff frequencies multiplying them by $\frac{f_s}{2}$, as inputs for `butter()`, together with the filter type set to "*bandpass*". This function returns as outputs the transfer function coefficients `b` and `a`, from which the frequency response of the digital filter can be obtained and plotted against frequency using the `freqz()` function. The frequency response is evaluated over 1024 points and the resulting plot is shown in Figure 2.

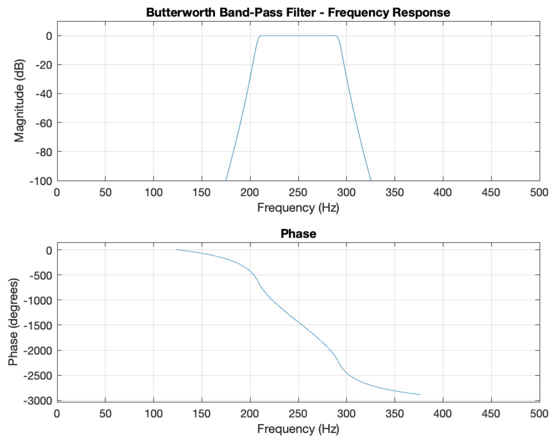


Figure 2: Frequency response of the Butterworth band-pass filter computed using the built-in `freqz()` function over 1024 points. The magnitude is limited from -100 dB to 10 dB, clearly depicting the filter's pass-band centered around 250 Hz.

Finally, the output signal `Y(t)` is obtained by filtering the input signal `X(t)` through the following command:

```
y = filter(b, a, x);
```

And its PSD is then plotted from the periodogram obtained applying once again the Welch method, as shown in Figure 3.

Moreover, two additional band-pass filters with the same specifications are designed and plotted. The first among these is an Elliptic filter, which follows the same procedure explained for the Butterworth filter. In this case, the built-in functions `ellipord()` and `ellip()` are used to estimate the required filter order and derive the corresponding transfer function coefficients. After computing the transfer function coefficients, the signal `X(t)` is processed using the `filter()` function. Subsequently, Welch's method is utilized, with the same parameter settings as before, to compute the periodogram of the filtered output `Y(t)`. Figure 4 displays both the frequency response (4.a) and the power spectral density (PSD) (4.b) of the resulting signal.

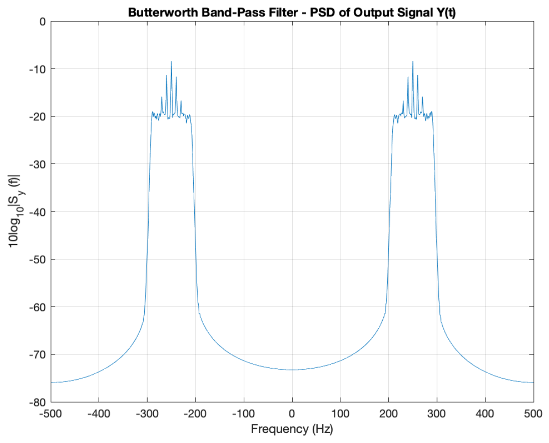
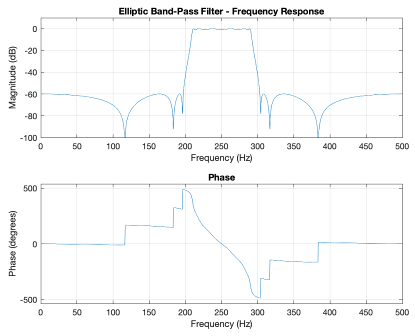
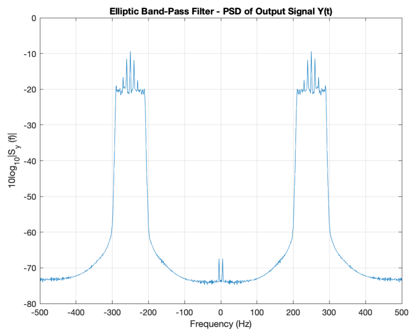


Figure 3: Power Spectral Density (PSD) of the output signal `Y(t)` after filtering with the Butterworth band-pass filter. Welch's method was applied by segmenting the 20-second signal into 25 Hamming windows with 50% overlap, demonstrating the filter's effect on the signal's spectral content.



(a)



(b)

Figure 4: (a) Frequency response of the Elliptic band-pass filter, computed using `freqz()` over 1024 points, with the magnitude constrained from -100 dB to 10 dB, which highlights the filter's frequency characteristics; (b) Power Spectral Density (PSD) of the output signal `Y(t)` obtained after processing with the Elliptic band-pass filter. Welch's method was applied by segmenting the 20-second signal into 25 Hamming windows with 50% overlap, demonstrating the filter's effect on the signal's spectral content.

Next, an equiripple FIR filter is designed for the same band-pass specifications, but using a procedure tailored for FIR filter design, which from the theory have the denominator coefficients vector set to `a = 1`. First, the vector of cutoff frequencies is defined by scaling the stop-band and pass-band edge frequencies with $\frac{f_s}{2}$, as shown by:

```
f_fir = [Ws(1) Wp Ws(2)]*(fs/2);
```

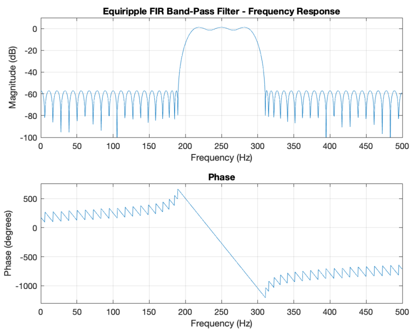
The desired amplitude response is set to `a = [0 1 0]`, meaning that the stop-bands should have zero amplitude and the pass-band should have unity gain. The design tolerances or deviations for the three bands are specified through the vector:

```
dev = [10^(-Rs/20) 1-10^(-Rp/20) 10^(-Rs/20)];
```

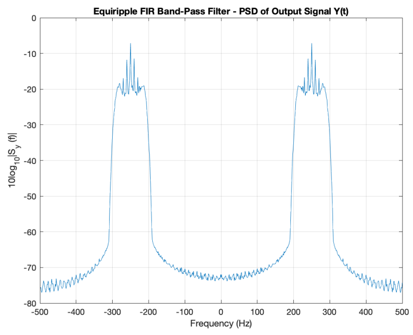
which reflects the stop-band attenuation (Rs) and the pass-band ripple (Rp). The built-in function `firpmord()` is then used to estimate the required filter order (n), normalized frequency points (f_o), amplitude response (a_o), and weights (w), using the defined frequency vector (f_fir), amplitude vector (a), deviation vector (dev), and sampling frequency (fs):

```
[n,fo,ao,w] = firpmord(f_fir,a,dev,fs);
```

Finally, the equiripple FIR filter is constructed using the `firpm()` function with the estimated parameters, and its frequency response is plotted over 1024 points. After obtaining the filter coefficients `b`, the filter is applied to the input signal `X(t)` with the command `filter()` and the periodogram of the filtered signal `Y(t)` is then estimated using Welch's method with the same parameters as before. Figure 5 illustrates the frequency response (5.a) and power spectral density (PSD) (5.b) of the output signal for this filter, allowing for direct comparison with those obtained from the Elliptic design.



(a)



(b)

Figure 5: (a) Frequency response of the equiripple FIR band-pass filter computed with `freqz()` over 1024 points. The plot displays the filter's response with the magnitude limited from -100 dB to 10 dB; (b) Power Spectral Density (PSD) of the output signal `Y(t)` after filtering with the equiripple FIR band-pass filter. Welch's method was applied by segmenting the 20-second signal into 25 Hamming windows with 50% overlap, demonstrating the filter's effect on the signal's spectral content.

Subsequently, a Chebyshev type I low-pass filter is designed following a procedure analogous to the one described earlier, though with modifications suited to a low-pass configuration. In this case, the pass-band is defined from `f = 0` up to a designated edge frequency. Consequently, the pass-band cutoff frequency vector comprises a single element, the pass-band edge frequency, set to 6 Hz, while the stop-band cutoff frequency is determined by adding the 8 Hz transition band to this edge frequency. The specifications for the pass-band ripple (1 dB) and stop-band attenuation (60 dB) remain unchanged. Using the function `freqz()`, its frequency response is then plotted over 1024 points, with the magnitude limited from -100 dB to 10 dB, as shown in Figure 6.

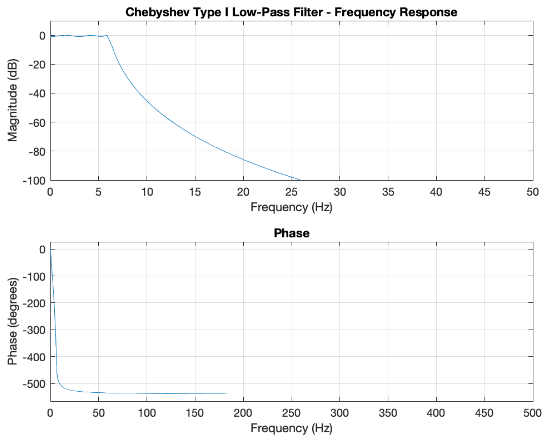


Figure 6: Frequency response of the Chebyshev type I low-pass filter computed using the `freqz()` function over 1024 points. The plot is constrained to a magnitude range from -100 dB to 10 dB and frequencies up to 50 Hz, highlighting the low-pass characteristics with a cutoff at 6 Hz and an 8 Hz transition band.

2.2 Comparison between filters

When comparing the three band-pass filters (Butterworth, Elliptic, and equiripple FIR) designed for the same specifications, there are different key points to highlight:

Selectivity (Steepness of Transition):

The frequency response plots shows that the Elliptic filter has the sharpest transition between the pass-band and the stop-band. This steep slope means high selectivity, which efficiently distinguishes between the desired frequency components and unwanted ones. In contrast, the Butterworth filter, characterized by its nearly flat pass-band, displays a more gradual transition. Moreover, the equiripple FIR filter also achieves a relatively steep transition, but it does so at the expense of a significantly higher filter order compared to the Elliptic design.

Pass-band Ripple:

The Elliptic filter, as observed in its frequency response, introduces noticeable ripples within the pass-band due to its equal-ripple design. Thanks to these deliberate oscillations it can meet stringent transition specifications with fewer coefficients. Conversely, the Butterworth filter maintains a nearly ideal flat response in the pass-band, with minimal ripple. The equiripple FIR filter shows moderate ripple in the pass-band, which is a typical compromise of linear phase designs (see the phase plot of Figure 5.a).

Stop-band Attenuation:

Although all three filters are designed to achieve approximately 60 dB of stop-band attenuation, their practical behaviors differ noticeably. The Butterworth filter exhibits a nearly linear decay in magnitude with no ripple in the stopband, continuing well below -100 dB, even though the plot is limited to -100 dB. In contrast, the equiripple FIR filter stabilizes around -60 dB in the stopband but shows a distinct ripple that, while somewhat periodic, is not perfectly so; the magnitude oscillates from slightly above -60 dB (approximately -57/58 dB) up to nearly -100 dB. The Elliptic filter, on the other hand, not only provides a sharp transition but also exhibits a ripple in the stopband that appears to stabilize asymptotically at around -60 dB.

Filter Order:

The order of the filter represents its computational complexity and the number of coefficients used. Observations from the plotted responses show that the Elliptic filter meets the design specifications with the lowest order, using its inherent ripple to produce a sharp transition. The Butterworth filter, designed for a maximally flat response, requires a higher order relative to the Elliptic filter. Meanwhile, the equiripple FIR filter generally demands the highest order, as it is evident from its frequently oscillating frequency response, to achieve the same level of performance in terms of both selectivity and stop-band attenuation.

Table 1 provides a concise summary of the filter performance across these key parameters.

Filter Type	**Selectivity**	**Pass-band Ripple**	**Stop-band Attenuation**	**Filter Order**
Butterworth	Moderate (gradual transition)	Minimal	Very High ($\sim$ linear decay)	Intermediate ($n = 18$)
Elliptic	Highest (sharp transition)	Moderate	$\approx$-60 dB (ripple stabilizing asymptotically)	Lowest ($n = 6$)
Equiripple FIR	High (steep transition)	Significant	-57/58 dB to -100 dB (non-perfect periodic ripple)	Highest ($n = 99$)

Table 1: Comparison of filter characteristics based on frequency response and PSD observations.

2.3 Custom function `my_filter()`

In addition to the conventional filter designs presented above, a custom MATLAB function named `my_filter()` was developed. This function implements a digital filter based on the z-transform representation of a linear constant-coefficient difference equation (LCCDE):

$$y(n) = \sum_{k=0}^{N_b-1} b(k+1)\,x(n-k) \;-\; \sum_{m=1}^{N_a-1} a(m+1)\,y(n-m),$$

with $a(1) = 1$. This function performs digital filtering while also revealing how the algorithm works internally, detailing step by step how the recursion is carried out and how initial conditions are managed.

A few key aspects of the `my_filter` implementation are as follows:

- **Stability check:** Before performing any filtering, the function checks the stability of the filter by computing the roots of the denominator polynomial defined by the coefficient vector `a`. An error is raised if any root has a magnitude greater than or equal to one and the function terminates. This is achieved by the following code:

```
if any(abs(roots(a)) >= 1)
    errordlg('The filter is not stable!', 'Error');
    y = [];
    return
end
```

- **Zero-padding for initial conditions:** To correctly compute the output samples during this recursive filtering, the input signal `x` is zero-padded on the left. To determine the number of zero samples to be added, the maximum of the lengths of the coefficient vectors minus one is taken. This guarantees that the initial conditions are properly handled.

- **Single loop filtering:** The zero-padded input vector is processed in one pass to satisfy the requirement of using a single `for` loop. The coefficient vectors are flipped so a sliding window can move over the input `x` and the past outputs `y`. At each step, the flipped numerator coefficients are multiplied by the matching part of `x`, and the flipped denominator coefficients by the matching part of `y`. Using `min()` makes sure that only available samples at the start and end are used, so no extra boundary checks are needed. The core implementation is:

```
b_flipped = fliplr(b);
a_flipped = fliplr(a);
for i = 1:length(x)
    y(i+Na-1) = sum(b_flipped(1:min(end, length(x)- ...
    i+1)).* x(i:min(Nb+i-1, end))) - sum(a_flipped(...
    1:min(end-1, length(y) - i)).*y(i: min(Na+i-2, ...
    end-1)));
end
```

- **Output cutting:** After the filtering process, the additional zero-padding is removed so that the final output `y` matches the original length of `x`.

In summary, the `my_filter` function shows a direct approach to implementing digital filtering. It performs a pre-filter stability check, then zero-pads according to the initial conditions, and finally executes the LCCDE within a single `for` loop. In the following steps, this custom function will be tested under various scenarios, including comparisons with built-in MATLAB commands and usage with unstable coefficients, in order to ensure that `my_filter` works correctly.

2.4 Testing of `my_filter()`

As an initial validation of `my_filter()`, the impulse response of the previously designed Chebyshev Type I low-pass filter (coefficients $b_{\text{chev}}, a_{\text{chev}}$, see Figure 6) was computed. A Kronecker delta, zero-padded to $M = 100$ samples, was filtered using `my_filter(b_chev, a_chev, delta)` and compared against MATLAB's built-in `impz(b_chev, a_chev, M)` in the plot shown in Figure 7. The signal filtered using `my_filter()` was plotted using `stem()`, with the horizontal axis representing the sample index n, which goes from 0 up to $M - 1$. As it is clear from the figure, for each sample both responses coincide exactly; validating the correctness of the custom function.

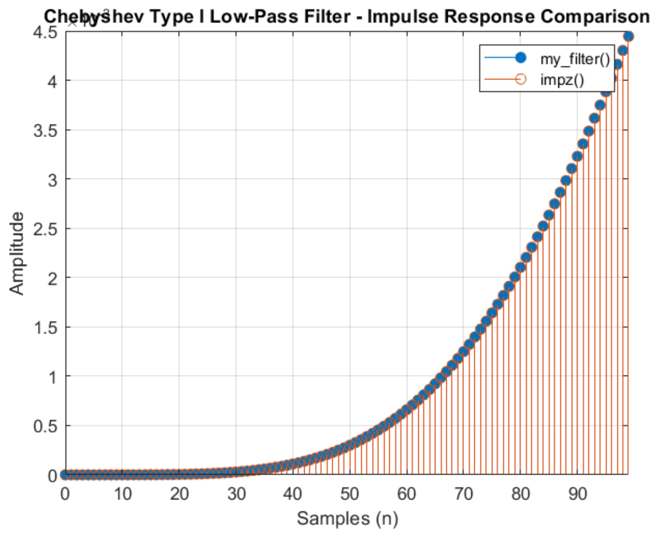


Figure 7: Comparison of the Chebyshev Type I low-pass filter impulse response computed by `impz()` (red dots) and by `my_filter()` (blue circles). The horizontal axis denotes the sample index n.

Subsequently, a 4-second test signal `X(t)` containing a 5 Hz cosine, a Dirichlet-modulated component, and white Gaussian noise was processed with both

```
Y_a = my_filter(b_chev, a_chev, x);
Y_b = filter(b_chev, a_chev, x);
```

Figures 8 and 9 display the results for noise variances $\sigma^2 = 1$ and $\sigma^2 = 0.1$, respectively. In each case, the two traces are indistinguishable, confirming that `my_filter()` reproduces MATLAB's native `filter()` output exactly. Cursors placed on every other peak, spanning two full periods, exhibit a total separation of approximately 0.4 s, which corresponds to a 5 Hz sinusoid (two periods in 0.4 s implies $f = 2/0.4 = 5\,\text{Hz}$).

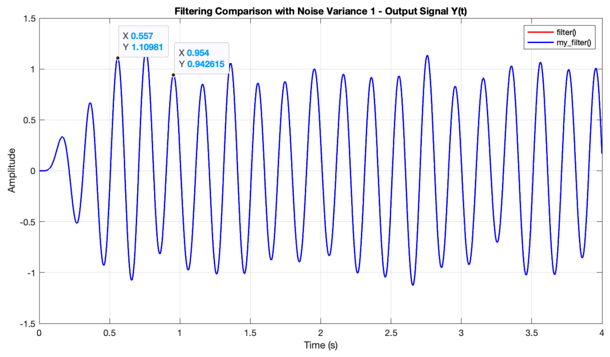


Figure 8: Filtering comparison for noise variance $\sigma^2 = 1$: MATLAB's `filter()` (red) versus `my_filter()` (blue).

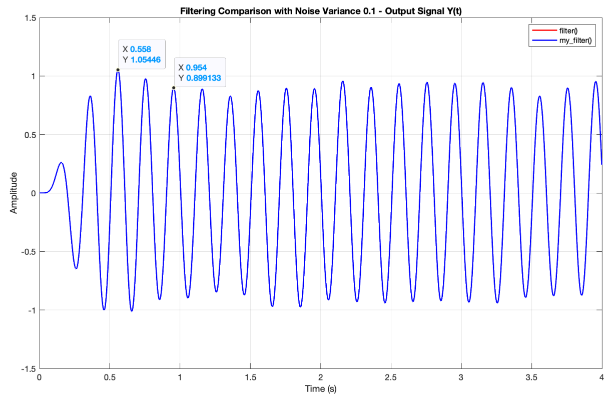
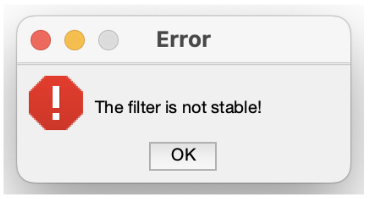


Figure 9: Filtering comparison for noise variance $\sigma^2 = 0.1$: MATLAB's `filter()` (red) versus `my_filter()` (blue).

Finally, stability was tested by supplying the coefficient set

```
b_error = [1, 1.3, 0.49, -0.013, -0.029];
a_error = [1, -0.4326, -1.6656, 0.1253, 0.2877];
```

whose pole–zero plot (Figure 10) reveals poles outside the unit circle. Invocation of `my_filter(b_error, a_error, x)` immediately produces the dialog


Error

The filter is not stable!

OK

stating "The filter is not stable!" and returns no output, thereby satisfying the stability-check requirement.

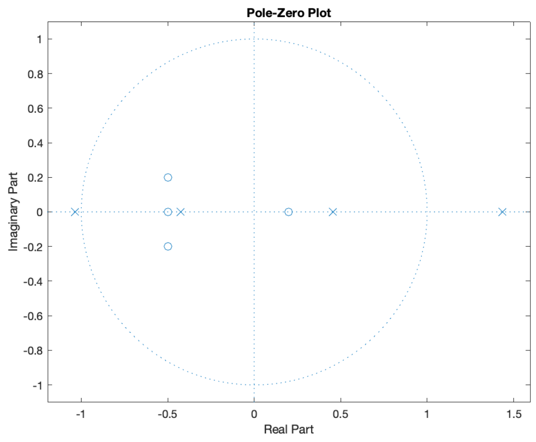


Figure 10: Pole–zero plot for the unstable coefficient set, showing poles outside the unit circle.

3 Window method for FIR filter design

The goal of the exercise was to design different FIR filter using the window method.

3.1 Band-pass filter

In the first section, the aim was to design a band-pass filter with the following characteristics:

- Passband region: 70 Hz centered at 200 Hz
- Transition bands: 30 Hz
- Minimum stopband attenuation: $A_s = 50$ dB

considering $f_s = 1$ kHz as the sampling frequency.
Since the window method was required, the first step was to select a window capable of satisfying the design specifications with the minimum number of coefficients. Therefore, the minimum required stopband attenuation was used for the selection. Given that the attenuation had to exceed 50 dB, the Hamming window was identified as the first suitable option, as it provides a minimum stopband attenuation of 53 dB.

In order to create properly the window, the stopband and the passband were explicitely defined according to the specifications.
The passband was centered at 200 Hz and had a total width of 70 Hz. For further computations, the passband and the stopband were normalized in [0, π]. Therefore, the normalized passband was defined as:

$$W_p = \left[\frac{f_{\text{cent}} - \frac{\text{passband\_length}}{2}}{f_s}, \frac{f_{\text{cent}} + \frac{\text{passband\_length}}{2}}{f_s}\right] \cdot 2\pi$$

where $f_{\text{cent}} = 200$ Hz, `passband_length` $= 70$ Hz, and f_s is the sampling frequency. Substituting the values:

$$W_p = \left[\frac{200 - \frac{70}{2}}{f_s}, \frac{200 + \frac{70}{2}}{f_s}\right] \cdot 2\pi = \frac{1}{f_s}\,[165,\ 235] \cdot 2\pi$$

Proceeding with the definition of the stopband, taking into account a transition band of 30 Hz, it was defined as:

$$W_s = \left[\frac{f_{\text{cent}} - \frac{\text{passband\_length}}{2} - \text{trans\_band}}{f_s}, \frac{f_{\text{cent}} + \frac{\text{passband\_length}}{2} + \text{trans\_band}}{f_s} \right] \cdot 2\pi$$

where `trans_band` = 30 Hz. Substituting the values:

$$W_s = \left[\frac{200 - \frac{70}{2} - 30}{f_s}, \frac{200 + \frac{70}{2} + 30}{f_s} \right] \cdot 2\pi = \frac{1}{f_s} [135,\ 265] \cdot 2\pi$$

Furthermore, the cutoff frequencies were determined as the midpoints between the passband and stopband. In MATLAB, this was obtained as:

```
fc_norm = (Wp + Ws)/(2*2*pi)
```

where W_p and W_s are the vectors containing the edges of the passband and stopband, respectively. Since both W_p and W_s are normalized in $[0, \pi]$, the resulting cutoff frequencies are also normalized with respect to f_s. In this case, the computed normalized cutoff frequencies were:

$$f_{c_1} = 0.15 \quad \text{and} \quad f_{c_2} = 0.25$$

Converting these values back to Hertz by multiplying with f_s (with $f_s = 10^3$ Hz) yields:

$$f_{c_{1_{Hz}}} = 0.15 \cdot f_s = 150\,\text{Hz} \quad \text{and} \quad f_{c_{2_{Hz}}} = 0.25 \cdot f_s = 250\,\text{Hz}$$

These cutoff frequencies are exactly the expected midpoints between the passband and the stopband limits.

In addition, in order to obtain the number of required coefficients N to define the window, the required formula was for Hamming:

$$N = \left\lceil \frac{6.6\pi}{\frac{B_{t_{Hz}}}{f_s} \cdot 2\pi} \right\rceil$$

in which B_t represented the transition band in Hertz. Substituting the values:

$$N = \left\lceil \frac{6.6\pi}{30} \cdot \frac{f_s}{2\pi} \right\rceil = 110$$

Therefore, the number of required coefficients is 110.
The corresponding window function, after having defined all the parameters accurately, is obtained following the formula for the Hamming window:

$$w_{\text{Hamming}}[n] = 0.54 - 0.46 \cos\left(\frac{2\pi n}{N-1}\right)$$

given n defined as the interval of values [0 : N-1].
It is clear that the window defined in this way is not causal. As a consequence, it is required to defined the delay in samples, denoted as M, such that it makes the filter causal:

$$M = \left\lceil \frac{N-1}{2} \right\rceil = \left\lceil \frac{110-1}{2} \right\rceil = 55$$

Once the window has been defined, using the spectral transformation formulas, and the delay has been found, the final filter can be obtained by multiplying the ideal impulse response with the window function:

$$h[n] = h_{\text{id}}[n] \cdot w_{\text{Hamming}}[n]$$

where $h_{\text{id}}[n]$ is the impulse response of the ideal passband filter.
In particular, the impulse response of an ideal passband filter is defined as the difference between two ideal low-pass filters with different cutoff frequencies. It can be expressed as:

$$h_{\text{id}}[n] = 2f_{c_1} \cdot \text{sinc}\left(2f_{c_1}(n-M)\right) - 2f_{c_2} \cdot \text{sinc}\left(2f_{c_2}(n-M)\right)$$

where f_{c_1} and f_{c_2} are the normalized lower and upper cutoff frequencies, respectively, and M is the delay found in the previous point.
Once the filter coefficients $h[n]$ have been obtained through windowing, the impulse response can be evaluated and visualized using the following MATLAB commands:

```
imp_res = impz(h, 1, 1024, fs)
```

Here, `impz` is used to compute the impulse response of the FIR filter defined by numerator coefficients $h[n]$ and denominator equal to 1 (since the filter is FIR), over 1024 samples and with sampling frequency f_s. Subsequently, the function `freqz` is used to compute the frequency response $H(f)$ of the filter.

```
[H, f] = freqz(h, 1, 1024, fs)
```

The resulting impulse response `imp_res` and frequency response magnitude $|H(f)|^2$ can be plotted to verify if the characteristics of the filter correspond to the one expected. Specifically, the impulse response is shown in Figure 11 and the frequency response magnitude is in Figure 12.

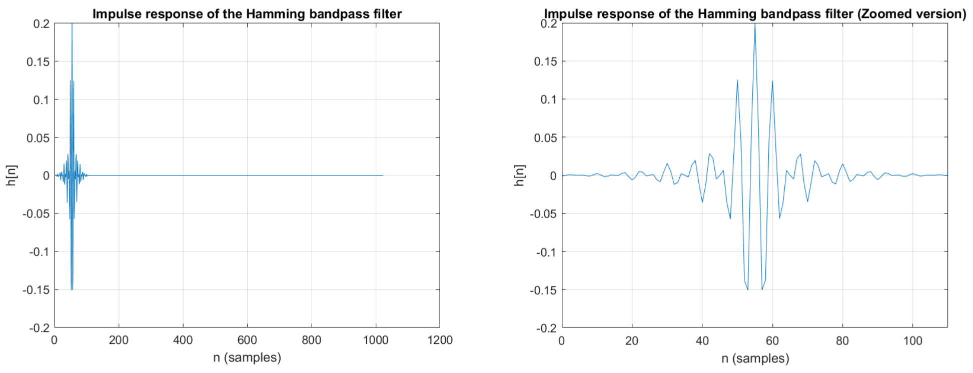


Figure 11: Impulse response h[n] of the bandpass filter obtained through the Hamming window. The figure on the right displays the response over the interval n in [0,N], while the full-length response is shown on the left.

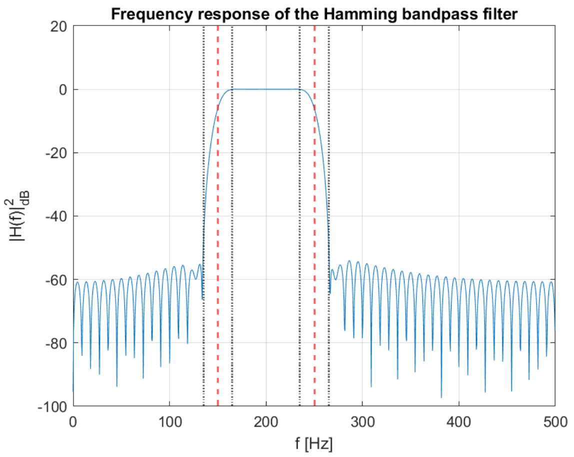


Figure 12: Frequency response $|H(f)|^2$ of the bandpass filter obtained through the Hamming window. The black lines represent the frequency edges, while the red line the cutoff frequencies

Focusing on Figure 11, it can be observed that the impulse response is not centered at n = 0. This is because it has been shifted by M samples to make the system causal, resulting in the response being centered at n = M.

Additionally, it has finite duration due to the filtering process and it is of length N. Overall, it exhibits the oscillatory behavior characteristic of a sinc function multiplied by a window function, specifically a Hamming window, which smooths the side lobes and limits the duration of the impulse response.

In addition, analyzing Figure 12, it is evident that the filter behaves as a bandpass filter, as expected. This is confirmed by the presence of a 0 dB region confined within a specific frequency range, corresponding to the passband defined in the specifications. A more detailed analysis shows that the attenuation in the stopbands meets the expected criteria: the frequency ranges from 0 to 135 Hz and from 265 Hz to 500 Hz is below -50 dB, thus satisfying the requirements. Furthermore, the cutoff frequencies are located precisely midway between the passband and stopband edges, as anticipated. Finally, the transition bands are each 30 Hz wide, in full agreement with the design specifications.

3.2 Band-stop filter

The same procedure applied for the band-pass filter is now extended to the design of the band-stop filter. Specifically, the band-stop filter is defined with the following characteristics:

- Passband region: from 0 Hz to 255 Hz and from 395 Hz to 500 Hz
- Transition bands: 40 Hz
- Minimum stopband attenuation: $A_s = 70$ dB

As in the previous section, the minimum required stopband attenuation was used for the selection of a window that satisfies the requirements. Given that the attenuation had to exceed 70 dB, the Blackman window was identified as the only suitable option, as it provides a minimum stopband attenuation of 74 dB.

To correctly design the window, the passband and stopband were explicitly defined according to the given specifications. They were defined normalized in MATLAB as follows:

```
Wp = [0, 255, 395, 500]/(fs)*2*pi;
Ws = [295, 355]/(fs)*2*pi;
```

According to the specification, the passband was defined up to 255 Hz and from 395 Hz onwards. A transition band of 40 Hz was introduced on each side of the stopband region, resulting in an effective stopband between 295 Hz and 355 Hz.

The cutoff frequencies were again defined as the midpoints between the passband and stopband edges. In MATLAB, this was computed as:

```
fc_norm = (Wp(2:3) + Ws)/(2*2*pi);
```

Here, the second and third elements of the "Wp" vector are the values 255 Hz and 395 Hz, which correspond to the end of the lower passband and the beginning of the upper passband. We used these values for the computation because they represent the frequencies at which the transition bands begin. Since both Wp and Ws are defined in normalized form in $[0, \pi]$, the resulting cutoff frequencies are also normalized in $[0,\ \frac{1}{2}]$:

$$f_{c_1} = 0.28 \quad \text{and} \quad f_{c_2} = 0.38$$

To understand whether the obtained values are consistent with the specifications, they can be converted back to Hertz by multiplying by the sampling frequency f_s:

$$f_{c_1} = 0.28 \cdot f_s = 280\,\text{Hz} \quad \text{and} \quad f_{c_2} = 0.38 \cdot f_s = 380\,\text{Hz}$$

These values correspond exactly to the midpoints between the passband and stopband edges, confirming their correctness.
In order to find the required number of coefficients N, the formula for the Blackman window is used:

$$N = \left\lceil \frac{11\pi}{\frac{B_{t_{Hz}}}{f_s} \cdot 2\pi} \right\rceil = \left\lceil \frac{11\pi}{40} \cdot \frac{10^3}{2\pi} \right\rceil = 138$$

Therefore, the number of required coefficients is 138.
The corresponding window function, after having defined all the parameters accurately, is obtained following the formula for the Blackman window:

$$w_{\text{Blackman}}[n] = 0.42 - 0.5\cos\left(\frac{2\pi n}{N-1}\right) + 0.08\cos\left(\frac{4\pi n}{N-1}\right)$$

given n defined as the interval of values [0 : N-1].
Again, it is required to compute the delay in samples to make the window causal:

$$M = \left\lceil \frac{N-1}{2} \right\rceil = \left\lceil \frac{138-1}{2} \right\rceil = 69$$

Once the window has been defined, using the spectral transformation formulas, the final filter can be obtained by multiplying the ideal impulse response with the window function:

$$h[n] = h_{\text{id}}[n] \cdot w_{\text{Blackman}}[n]$$

The impulse response of an ideal stopband filter is defined as:

$$h_{\text{id}}[n] = \delta(n-M) - \left(2f_{c_2}\,\text{sinc}\left(2f_{c_2}(n-M)\right) - 2f_{c_1}\,\text{sinc}\left(2f_{c_1}(n-M)\right)\right)$$

where M is the delay found in the previous point and f_{c_1} and f_{c_2} are the normalized cutoff frequencies.
Once the filter coefficients $h[n]$ are obtained through windowing, the impulse response can be evaluated using `imp_res = impz(h, 1, 1024, fs)`. Subsequently, the function `freqz` is used to compute the frequency response $H(f)$ of the filter (`[H, f] = freqz(h, 1, 1024, fs)`).

The resulting impulse response `imp_res` and frequency response magnitude $|H(f)|^2$ are shown respectively in Figure 13 and in Figure 14.

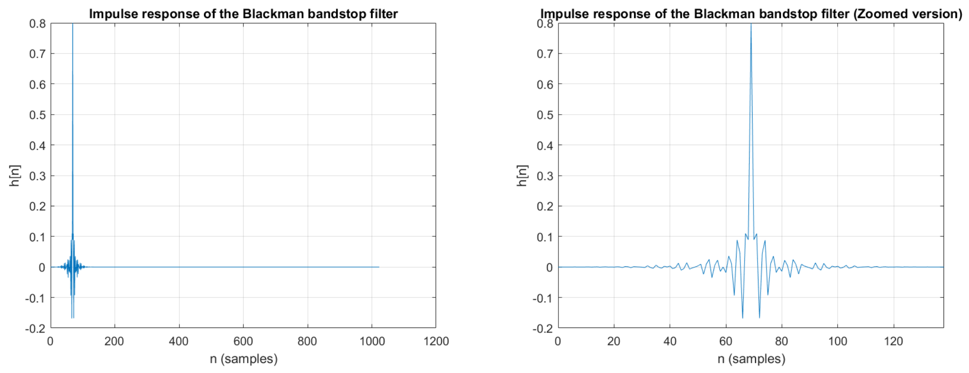


Figure 13: Impulse response h[n] of the bandstop filter obtained through the Blackman window. The figure on the right displays the response over the interval n in [0,N], while the full-length response is shown on the left.

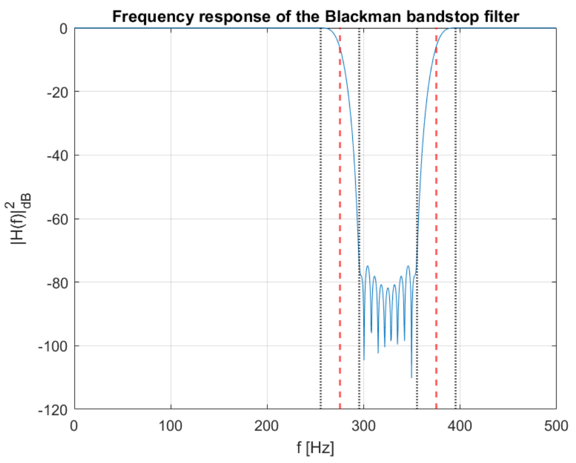


Figure 14: Frequency response $|H(f)|^2$ of the bandstop filter obtained through the Blackman window. The black lines represent the frequency edges, while the red line the cutoff frequencies

Analyzing Figure 13, similarly to Figure 11, it can be observed that the impulse response is centered at $n = M$ to ensure causality and exhibits the typical behavior of a sinc function multiplied by a window function, in this case a Blackman window. In addition, the response is nonzero only over N samples, corresponding to the length of the window.
Moreover, focusing on Figure 14, it is clear that it represents the spectrum of a bandstop filter, as expected. This is shown by the fact that there is an attenuated region, specifically between 295 and 355 Hz, which totally corresponds to the expected stopband region. A more detailed analysis shows that the attenuation totally satisfies the requirements since, in the expected stopband region, the filter remains consistently below -70 dB. Furthermore, as already checked for the passband filter, the cutoff frequencies are exactly in the middle of the transition bands, defined by the black dotted lines.

3.3 Kaiser window filter

The goal of the last part of the exercise is to build a custom function, called `my_kaiser_filter()`, that, given specific input parameters, designs and applies a filter using a Kaiser window. The parameters are:

- the required stop-band attenuation A_s
- the transition band B_t
- the sampling frequency f_s

- the cutoff frequency f_c
- a char array for the type of filter $(-lp, -hp, -bp, -sp)$

The outputs are:

- the FIR filter coefficients b_k
- the number of coefficients N

The first parameter that was defined in the function was β, which is the one that controls the stop-band attenuation, following the formula:

$$\beta = \begin{cases} 0.1102 \cdot (A_s - 8.7) & \text{if } A_s > 50 \\ 0.5842 \cdot (A_s - 21)^{0.4} + 0.07886 \cdot (A_s - 21) & \text{if } 21 \leq A_s < 50 \\ 0 & \text{if } A_s < 21 \end{cases}$$

Then, the cutoff frequency was normalized by f_s and the number of coefficients N was obtained through the following formula:

$$N = \left\lceil \frac{A_s - 8}{2.285 \cdot (\omega_s - \omega_p)} \right\rceil \quad \text{with} \quad \omega_s, \omega_p \text{ in } [0, \pi]$$

Again, in order to make the filter causal, the delay M was obtained with:

$$M = \left\lceil \frac{N-1}{2} \right\rceil$$

All the previously explained steps were performed equally for all types of filters, as the only distinction between them lies in the ideal impulse response h_{ideal}, which was defined based on the specific filter type as follows:

$$h_{\text{ideal}} = \begin{cases} 2f_c \,\text{sinc}\,(2f_c(n-M)) & \text{if type = '-lp'} \\ 2f_{c_2} \,\text{sinc}\,(2f_{c_2}(n-M)) - 2f_{c_1} \,\text{sinc}\,(2f_{c_1}(n-M)) & \text{if type = '-bp'} \\ \delta(n-M) - 2f_c \,\text{sinc}\,(2f_c(n-M)) & \text{if type = '-hp'} \\ \delta(n-M) - (2f_{c_2} \,\text{sinc}\,(2f_{c_2}(n-M)) - 2f_{c_1} \,\text{sinc}\,(2f_{c_1}(n-M))) & \text{if type = '-sp'} \end{cases}$$

If the filter type inserted as input is not valid, the function creates an error window defined as follows:

```
errordlg('Invalid filter type', 'ERROR')
```

In general, after having chosen the ideal impulse response, the Kaiser window is obtained with the following formula:

```
w_Kaiser = besseli(0, beta * sqrt(1 - ((2*n - N + 1)/(N -
                1)).^2)) / besseli(0, beta)
```

in which `besseli()` is the zero-order modified Bessel function of the first kind. The filter coefficients are subsequently obtained multiplying the window and the ideal impulse response as:

```
Bk = h_id.*w_Kaiser
```

Finally, the function is tested for two different types of filter: highpass and bandpass.

Highpass filter

It was required to set the sampling frequency to $f_s = 4$ kHz and use the custom function to create a high-pass filter with the following characteristics:

- Stop-band attenuation: $A_s = 40$ dB
- Transition band: $B_T = 200$ Hz
- Cut-off frequency: $f_c = 1.6$ kHz

After having inserted all these values as inputs of the function, the values of N and M are computed, according to the instructions presented previously. Moreover, β is computed following the formula $0.5842 \cdot (A_s - 21)^{0.4} + 0.07886 \cdot (A_s - 21)$ since $21 \leq A_s < 50$, and h_{ideal} is obtained with $h_{id} = \delta(n - M) - 2f_c \operatorname{sinc}(2f_c(n - M))$, given that it is a high-pass filter.

The resulting coefficients are then checked with the result of the following command:

```
b1 = fir1(N-1, Wn, kaiser(N, beta), 'high')
```

which computes the impulse response of a high-pass FIR filter of order N-1, with normalized cutoff frequency W_n, using a Kaiser window of length N and shape parameter β. This function is, in fact, used as a reference to verify that the manually calculated coefficients b_k match those obtained through the standard MATLAB implementation. In order to visually compare the built-in function and the custom one, the frequency responses are computed using `[H, f] = freqz(h,1,1024,fs)` and the magnitude $|H(f)|^2$ is shown in Figure 15.

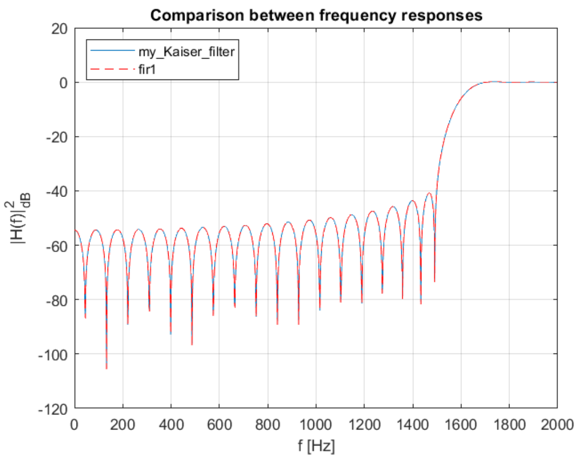


Figure 15: Comparison between the frequency responses $|H(f)|^2$ of the highpass filter, made with a Kaiser window, obtained with the custom function and the MATLAB built-in function

Analyzing Figure 15, since only one line is visible, it is possible to affirm that the results obtained with the custom function and the built-in function are totally overlapped, showing perfect equality. Both results correspond to a highpass filter, as expected, which has as characteristics the stopband attenuation of at least 40 dB and the transition band of 200 Hz. Therefore, the results perfectly satisfy the requirements.

Bandpass filter

The same steps used for the highpass filter are now performed for a bandpass filter with the following characteristics:

- Stop-band attenuation: $A_s = 60$ dB
- Transition band: $B_T = 100$ Hz
- Cut-off frequencies: $f_1 = 800$ Hz and $f_2 = 1.2$ kHz
- Sampling frequency: $f_s = 4$ kHz

Inserting all the parameters as inputs of the custom function, the values of N and M are computed, as well as β, which is obtained with the formula $0.1102 \cdot (A_s - 8.7)$, since $A_s > 50$.
The impulse response of the filter is chosen, according to instructions, as $h_{id} = 2f_{c_2}\,\mathrm{sinc}\,(2f_{c_2}(n-M)) - 2f_{c_1}\,\mathrm{sinc}\,(2f_{c_1}(n-M))$. The coefficients b_k are computed and checked with the result of the command:

```
b2 = fir1(N-1,Wn,kaiser(N,beta),bandpass)
```

which is the standard MATLAB implementation to compute the FIR bandpass filter using a Kaiser window with all the specific characteristics.
In order to practically compare the MATLAB implementation and the custom function, the frequency responses are computed using
`[H, f] = freqz(h,1,1024,fs)` and the magnitude is shown in Figure 16.

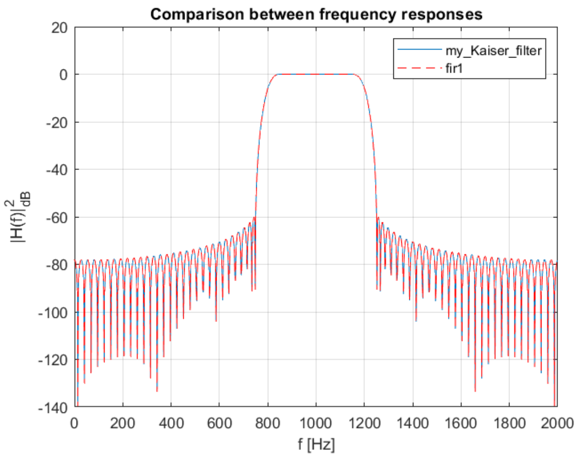


Figure 16: Comparison between the frequency responses $|H(f)|^2$ of the bandpass filter, made with a Kaiser window, obtained with the custom function and the MATLAB built-in function

Focusing on Figure 16, it is clear that, as in the case of the highpass filter, the results obtained through the custom and the built-in function perfectly correspond. Analyzing the stopband attenuation, it is clear that it respects the requirement of -60 dB and, in addition, the passband is centered at 1000 Hz, with cutoff frequencies (800 and 1200 Hz) matching exactly the expected ones, as well as the transition band perfectly corresponding to the expectations.

4 Processing of biosignals

In this exercise, the signal processing techniques studied previously in the course were applied to the analysis of a real case scenario: the output signal of a pulse oximeter. From the signals generated by the pulse oximeter, we were interested in calculating the pulse rate and the blood oxygen saturation level. The pulse oximeter measures these quantities by calculating the amount of light absorbed by a finger, using two different wavelengths: RED light and INFRARED light. Therefore, our analysis was performed simultaneously on both signals.

4.1 Original signal

We were given a `txt` file containing the values measured for both the RED and INFRARED signals. The sampling frequency of 100 Hz is equal for all the signals. We were asked to work only on a limited time window, consisting in $T_{window} = 60$ seconds, discarding the value measured in the first 10 seconds, as this may contains errors due to settling time. The data provided included output signals over a much longer duration than the interval we were interested in. Therefore, it was necessary to truncate the original signals, removing the first 10 seconds and trimming the end as well, in order to isolate exactly the portion of the signal corresponding to the desired time window. Using the relationship $N = T \cdot f_s$, where T is the observation time interval, it was easy to determine where to cut the signals. In this specific case $T_{start} = 10$ and $T_{end} = T_{start} + T_{window} = 70$. To plot the obtained signal, a time vector t was defined, as a vector of points in the range $[0, 60]$ and constant spacing $\frac{1}{f_s}$. In Figure 17 are shown both the RED and the INFRARED signals versus time: the two signals behave similarly, both showing a sinusoidal trend.

4.2 Apply low-pass filter

Measurements taken by the oximeter are affected by high frequency noise, mainly due to muscular movement. It is medically proven that the maximum pulse rate is of 180 bpm, or, equivalently, in Hertz $\frac{180\ bpm}{60\ s} = 3\ Hz$. We hence proceed to design a Butterworth low-pass filter (digital IIR) filter to remove such noise, with the following specifications:

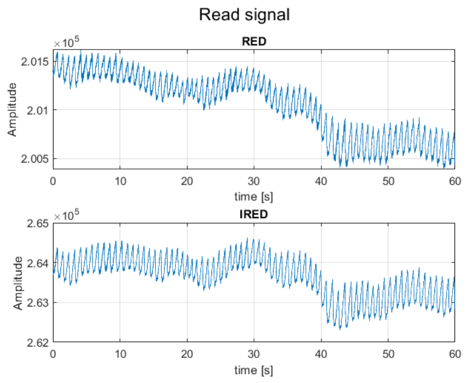


Figure 17: 60 second of both the RED and INFRARED signals as read by the pulse oximeter

- pass-band frequency edge: $\Omega_p = 3\ Hz$
- stop-band frequency edge: $\Omega_s = 6\ Hz$
- pass-band ripple: $R_p = 1\ dB$
- stop-band attenuation:$A_s = 60\ dB$

Equation 1 reports the equation representing the frequency response of the analogue Butterworth low-pass filter of order N and cut-off frequency Ω_c. Naturally, the cut-off frequency must be in the transition band, thus in between the pass-band and the stop-band edges defined above.

$$|H_a(j\Omega)|^2 = \frac{1}{1 + (\frac{\Omega}{\Omega_c})^{2N}} \tag{1}$$

To determine the filter order, the formula reported in Equation 2 was used. In this formula both the frequencies ω_s, ω_p are expressed as angular frequencies ($\omega = 2\pi\Omega$): we estimated the minimum order of the filter to be $N = 11$. Once the filter order N was determined, we were able to calculate the two possible values for the cut-off frequency: $\omega_{c,1}$ and $\omega_{c,2}$. The first, $\omega_{c,1}$, corresponds to the angular cut-off frequency required to precisely satisfy the pass-band constraints (both in terms of the band edge and ripple), while the second, $\omega_{c,2}$, is chosen to meet the stop-band specifications.
Equation 3 and equation 4 reports respectively the formula used to compute the two possible cut-off frequency. Actually, as final value for the low-pass filter cut-off frequency we used the mean value of the two: $\omega_c = \frac{\omega_{c,1}+\omega_{c,2}}{2}$,

that, expressed in Hertz is $\Omega_c = 3.196\ Hz$.

$$N = \left\lceil \frac{\log_{10}\left(\frac{10^{\frac{R_p}{10}}-1}{10^{\frac{A_s}{10}}-1}\right)}{2\log_{10}\left(\frac{\Omega_p}{\Omega_s}\right)} \right\rceil \tag{2}$$

$$\omega_c = \frac{\omega_p}{\sqrt[2N]{10^{\frac{R_p}{10}}-1}} \tag{3}$$

$$\omega_c = \frac{\omega_s}{\sqrt[2N]{10^{\frac{A_s}{10}}-1}} \tag{4}$$

Figure 18 represents the squared magnitude of the analogue filter H_a (Equation 1). Analyzing the response, it is possible to see that all the requirements of the low-pass filter (passband and stopband frequency edges, passband ripple and minimum stopband attenuation) are met. The frequency axis consists of 1001 points linearly spaced between 0 Hz and 40 Hz.

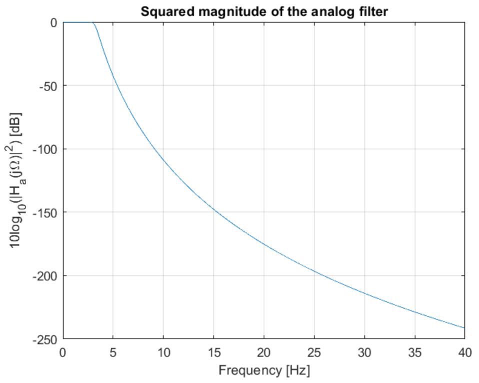
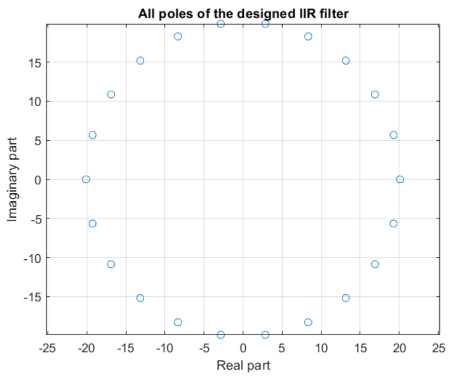


Figure 18: Squared magnitude of the analogue filter

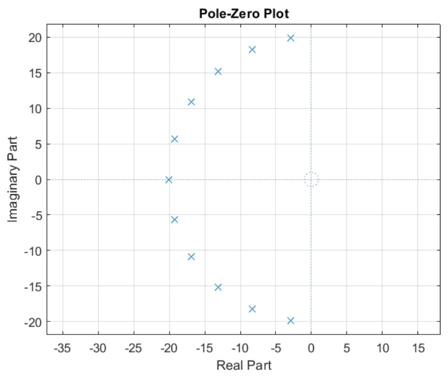
To be able to obtain the digital filter frequency response, the poles of the analogue transfer function must be known. All the poles were calculated, according to the formula reported in Equation 5, for $k = [0, N-1]$ and then we selected only the poles that lie in the left half-plane `s`, which are the one with negative real part. Figure 19 shows on the left all the poles of $H_a(j\Omega)$, on the right the poles of the transfer function $H_a(s)$ plotted using the MATLAB build-in command `zplane`. The transfer function $H_a(s)$

is computed using the MATLAB function `zp2tf`, that converts the zeroes, poles and gain of the filter parameters to the transfer function (returning numerator and denominator as vector). Being a low-pass filter, the transfer function has no zeroes, hence an empty vector was passed to the `zp2tf` function; the gain of the filter is defined as: Ω_c^N. By comparing the two plots in Figure 19, it is clear that, as expected, all the poles of the transfer function $H_a(s)$ lie in the left half-plane s, ensuring causality. Additionally, the poles coincide between the two plots and are positioned along a circumference of radius Ω_c.

$$p_k = \Omega_c \cdot e^{j\frac{k\pi}{N}} \cdot e^{j\frac{\frac{\pi(N+1)}{2}}{N}} \tag{5}$$



(a) All poles of the transfer function calculated using Eq 5



(b) Poles of H_a(S) plotted using `zplane` function

Figure 19

To generate the digital filter, the impulse invariance method was employed. This method calculate the impulse response $h[n]$ of the digital filter by sampling $h_a(t)$, the impulse response of the analogue filter. In MATLAB, it is possible to obtain directly $H(z)$ from $H_a(s)$ by using the command `impinvar`. This command returns the numerator and the denominator of the digital filter taking as inputs the numerator and the denominator of the analogue filter (obtained using `zp2tf`) and the sampling frequency f_S. Figure 20 shows the filter frequency and phase response plotted using the `freqz` command over 1024 points. Looking at the magnitude response, and comparing it with the one of the analogue filter (Figure 18) it is easy to notice that for low frequency these are identical, but as the frequency increases, the attenuation of the digital filter stays constant, while in the case of the analogue filter it

continues to decrease. This is due to aliasing, which is the main drawback of the impulse invariance method. In our specific case, due to the filter design, the effect of the aliasing is negligible: the attenuation of high frequency component is still more than 200 dB, thus very low.
The designed digital low-pass filter is then used to filter out the high-frequency components from both the RED and INFRARED signals, using the command `filtfilt`, a function that performs zero-phase digital filtering by processing the given signal both in the forward and in the reverse directions.

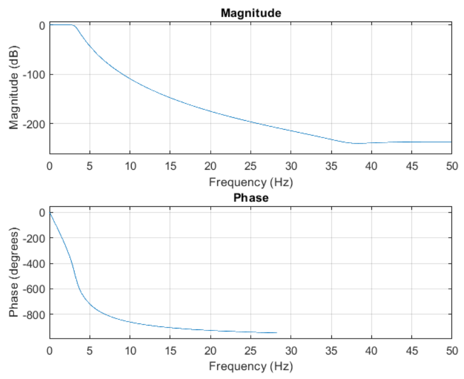


Figure 20: Transfer function of the digital Butterworth low-pass filter using the impulse invariance method

4.3 Apply high-pass filter

In order to be able to compute the pulse rate and the AC components of the signals, the DC components need to be attenuated, hence the use of a high-pass filter. We used a high-pass FIR filter design with window method, following the same procedure explained in the Section 3 of this assignment. The specification of the filter are as follows:

- stop-band frequency edge: $\Omega_s = 0.05\ Hz$
- pass-band frequency edge: $\Omega_p = 0.75\ Hz$
- window type: Hamming

Since the window type is known, we can obtain the number of coefficient N required to define the window: since we are using a Hamming window, $N = \left\lceil \frac{6.6\pi}{\frac{\Omega_s - \Omega_p}{fs} \cdot 2\pi} \right\rceil = 472$. As usual, for FIR filter, the cut-off is taken in the middle point of the transition band, thus $f_c = \frac{\Omega_s+\Omega_p}{2} = 0.4\ Hz$. Recalling the definition of the Hamming window

$$w_{Hamming}[n] = 0.54 - 0.46 \cos\left(\frac{2\pi n}{N-1}\right)$$

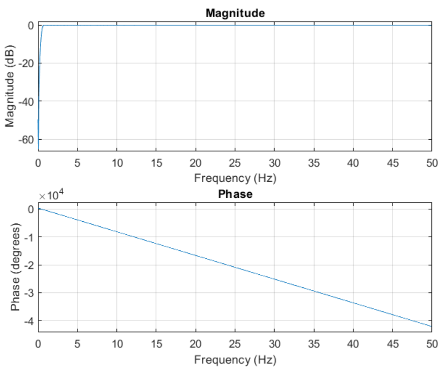
and the ideal impulse response for a high-pass filter

$$h_{id} = \delta(n-M) - 2f_c sinc\left(2f_c(n-M)\right)$$

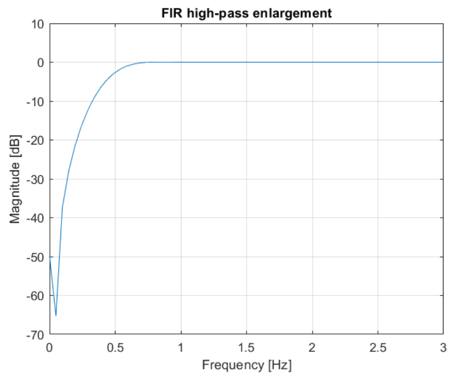
with the delay:

$$M = \left\lceil \frac{N-1}{2} \right\rceil = 236$$

we were able to design the desired high-pass filter, whose frequency response, obtained with the `freqz` is shown in Figure 21. Due to the very low pass-band frequency edge (we would like to remove solely the DC components of the signal, without modifying the AC, Figure 21.b reports an enlargement between 0 Hz and 3 Hz of the high-pass filter, so that it is possible to see both the stop-band and the transition band, otherwise undistinguishable from the pass-band.



(a)

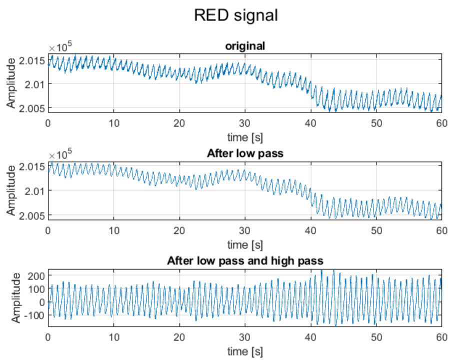


(b)

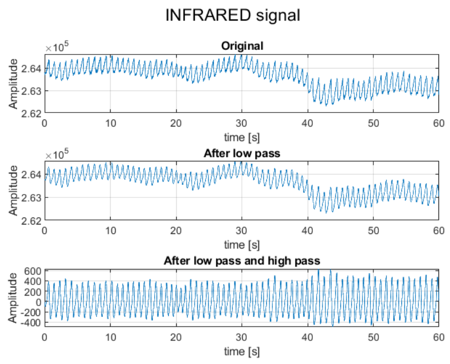
Figure 21: FIR high-pass filter designed using a Hamming window. On the right an enlargement on low-frequencies

The designed high-pass filter is then applied on the low-pass filtered signals, both RED and INFRARED, to obtain a DC free signal (purely oscillatory). To filter the signals, we again used the `filtfilt` command.

Figure 22a shows the filtered RED signal compared to the original one, while Figure 22b shows the same but for the INFRARED signal. For both the signal, the output of the low-pass has no noticeable difference with respect to the original, meaning that the noise is negligible; the signal after the low pass, as expected, presents no DC component, thus it is centered around zero.



(a) Original and filtered version of the RED signal



(b) Original and filtered version of the INFRARED signal

Figure 22

4.4 Compute the pulse rate

To compute the pulse rate over the analyzed time window, we need to work with the AC components of the RED signal. In particular, what is interesting to us, is the frequency at which the FFT of this signal presents its maximum. To find this value, we computed the FFT of the high-pass output using as number of samples the next power of 2, in this way the algorithm (Radix-2) is optimized. To find the next power of 2 the MATLAB function `nextpow2` was employed. Increasing the number of points, also means that the frequency resolution is increased, thus the step between two consecutive frequencies, δf, is decreased. The frequency vector was redefined using the new value for the step equal to $\delta f = \frac{f_s}{N_{FFT}}$ with $N_{FFT} = 2^{nextpow2(length(red\_filt2))}$ where `red_filt2` is the output of the high-pass filter.
Using `max` we were able to get the index where the FFT is maximum, consequently the frequency at which this happens. Once the frequency was found, to obtain the pulse rate, expressed as BPM (Beats Per Minute) the peak

frequency was multiplied by 60, resulting in a heart rate of 70.312 BPM. Figure 23 shows the Fourier transform of the doubled filtered signal.

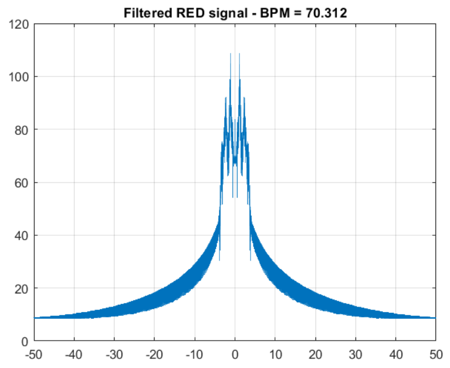


Figure 23: Spectrum of the RED signal after the high-pass filter, highlighting the pulse rate

4.5 Saturation computation

The formula to compute the saturation level is:

$$SaO_2 = 110 - 25\bar{R}$$

where $\bar{R}$ is defined as

$$\bar{R} = \left\langle \frac{\frac{I_{AC}(RED)}{I_{DC}(RED)}}{\frac{I_{AC}(INFRARED)}{I_{DC}(INFRARED)}} \right\rangle$$

The value of R was calculated every second and then we took the mean value. To determine the value of R we firstly need to calculate the actual value of all the currents. The command `findpeaks` was used to find the local maxima of the signal. When we were interested in locating the minimum, we used the same command, but we change the sign of the vector, remembering to then change again the sign of the output. This trick works because changing the sign of the vector means studying a signal symmetric to the original around the y-axis, thus the local minimums of the original signal are now the local maxima of the flipped version. The syntax of this command is as follows:

```
[maximum_value, maximum_location] = findpeaks(signal)
```

if we are looking for the maximum or

```
[minimum_value, minimum_location] = findpeaks(-signal)
```

if we want to retrieve the minimum, where of course the `maximum_location` is the index where the corresponding maximum is located in the input vector. This value is needed so that we can also know the time at which each maximum happens.
When dealing with AC components, we worked with the high-pass filtered signal; when dealing with DC, we worked with the low-pass output. Due to the oscillatory nature of AC, we were interested in finding both the values for minima and maxima, when speaking of DC, we were interested only in the minimum.
Once the point of interest were all determined, using the function `interp1`, we create the best fitting function that passes for these points; in particular we were interested in computing the DC current, hence passing from the DC minima, the AC current, passing from the AC minima and maxima, oscillating, and the overall current, that consist of the DC component and the AC components. All the interpolation were done using the *spline* method.
Since the syntax to compute the interpolating curve is slightly different depending on what the desired output is, we report here all the different variation done.
DC current:

```
I_DC = interp1(t(DC_location), -DC_values, t, "spline");
```

As explained before, it is `-DC_values` because when looking for the minima, we need to flip the original signal. AC current:

```
I_AC = interp1([t(AC_min_location), t(AC_max_location)], [-
    AC_min_values; AC_max_val], t, "spline");
```

Current:

```
I = interp1([t(AC_min_loc), t(AC_max_loc)], [-DC_val; -DC_val +
    AC_min_val + AC_max_val], t, "spline");
```

where t is the time vector defined at the beginning of the analysis.
Since we want to compute the value of R each second, we need to perform

sampling on all the currents: calling $\delta n = 1s \cdot f_s$ the resolution with the new sampling period of 1 second, we obtained a vector $n = 0 : \delta n :$ length(I) with the indexes of the sampled time. From there, by applying the formula reported previously we calculated $\bar{R}$ and consequently the saturation level, that we found to be $SaO_2 = 98.196\%$.
Finally in Figure 24 we reported, other than the saturation level, the interpolating curves: on the top there are the plots concerning the RED signal, on the bottom the ones concerning the INFRARED signal. The two subplots on the left are the whole current, which is the DC component (red dots) and the AC component (black oscillating curve); the two on the right shows only the AC component of the current: here the red dots are the local minima and maxima found using the `findpeaks` function, the black line is the interpolating curve obtained with the `interp1` function and *spline* method.

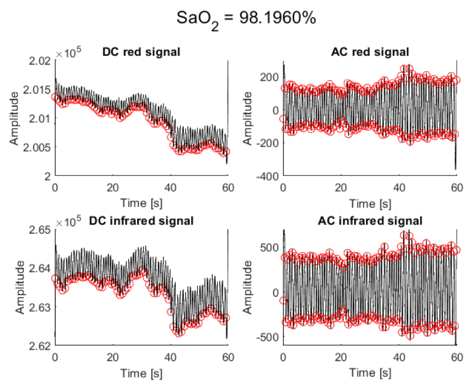


Figure 24: Interpolating curves and saturation level

5 Conclusion

In conclusion, throughout the three sections it was possible to practically apply the signal processing techniques studied during the second part of the course. In the first exercise, different filter design methods were implemented and compared, highlighting their differences in terms of selectivity, ripple, and transition steepness. In the second exercise, the work was extended by designing a FIR filter using the window method, focusing on how the choice of the window affects the filter's frequency response. In the third exercise, all the acquired skills were applied to a real-world scenario involving the processing of physiological signals acquired from a pulse oximeter, demonstrating the effectiveness of the designed filters in reducing noise, isolating the components of interest, and enabling the extraction of vital parameters such as heart rate and blood oxygen saturation. Overall, the work highlighted the importance of accurate filter design and confirmed that the theoretical knowledge studied in class can be successfully applied to practical real-world problems.