Lab 4: Introduction to Signal Processing - Filters

This laboratory requires the following equipment:

- Matlab

The laboratory duration is approximately 3 hours. Although this laboratory is not graded, we encourage you to take your own personal notes as the lab verification test might leverage results acquired during this laboratory session. For any questions, please contact us at **sis-ta@groupes.epfl.ch**

Information

In the following, text you will find several exercises and questions.

- The notation **S** means that the question can be solved using only additional simulation.
- The notation **Q** means that the question can be answered theoretically, without any simulation.
- The notation **I** means that the problem has to be solved by implementing a piece of code and performing a simulation.
- The notation **B** means that the question is optional and should be answered if you have enough time at your disposal.

Outline

This lab continues on the subject of signal processing, and in particular discusses filtering. In Part 1, you will apply different filters to signals synthesized in Matlab, while in Part 2 you will design your own filters using the Filter Design and Analysis Tool and apply them to real signals.

If you don't have it already, please install the Signal Processing Toolbox in Matlab from the Add-Ons-menu.

Getting Started

To start with this lab, you will need to download the material available on Moodle. Download `lab04.tar.gz` or `lab04.zip` and extract it in your home directory (you can type `tar xvfz lab04.tar.gz`). Now start Matlab and change your "Current directory" to be `lab04/part01/`.

Part 1: Filtering of signals generated in Matlab

The first part of this lab uses the `SamplingReconstruction.m` file located in the `part01` directory of the `lab04` folder. There are five places in this file where parameters can be modified:

- The first location corresponds to the signal creation (lines 18-19 in the source code). The signal is built through a summation of multiple sine functions. You can define the parameters of the different sine functions: `f(n)` stands for the frequency of the n^{th} sine and `a(n)` stands for its amplitude. The signal can be built with an unlimited number of sine functions.

- At the second location (line 43), `Fmax` stands for the maximal frequency displayed in the FFT plots.

- At the third location (lines 65-70), you can specify if you want to add a filter. There are two types of filters: a simple first order low pass filter and a Butterworth filter whose order can be modified. There are three parameters:
  - `filterType` selects the kind of filter you want,
  - `Fc` sets the cutoff frequency of the filters,
  - `filterOrder` sets the filter order (only for the Butterworth filter).

A Butterworth filter is a type of filter designed to have a maximally flat frequency response in the passband. It can be trivially modified to function as a low pass, high pass, band stop or band pass filter.
The order of a filter can be identified in the Bode plot: for an *n*-order filter, the decibel roll-off will be 20**n* dB per decade.

- At the fourth place (line 134), `fs` stands for the sampling frequency used for the resampling. You can specify an array of sampling frequencies if you want. For this lab, we will leave it as an empty vector.

- At the last place (lines 158-159) you can select the interpolation function used in reconstructing the original signal from the collected samples. It can be either `wsSignalReconstruction` or `linearSignalReconstruction` for Whittaker-Shannon or linear interpolation, respectively.

The Matlab script first generates the signal and computes the Fast Fourier Transform (FFT), an efficient algorithm for carrying out the Discrete Fourier Transform (DFT), to observe the frequencies contained in the signal. Then, it filters it if the according parameters have been set. It then resamples the signal at frequency `fs` and reconstructs the signal using the selected interpolation formula.

1. **(I):** Create a signal made of two sines, with respective frequencies of 0.5 and 7 Hz, and amplitude of 1 for both. Add a simple low pass filter with a cutoff frequency (`Fc`) of 3 Hz. Compare the FFT of the original signal with the FFT of the filtered signal. Are there any differences? Explain the changes from the Bode plot.

2. **(I):** Keep the same cutoff frequency and change to a high pass Butterworth filter of order 1. Compare the FFT of the original signal with the FFT of the filtered signal. Are there any differences? Explain the changes from the Bode plot.

3. **(I):** Change the filter type back to a low pass Butterworth filter, and run the script for filter orders of 1 and 3. Compare the two filtered signals. Are there any differences? Explain the changes from the Bode plots.

4. **(I)**: Increase the cutoff frequency to 100 Hz and compare the Bode plots for a simple low pass filter and 1st and 3rd order low pass Butterworth filters. What is different in the frequency response? What are the differences in the phase response? Explain the changes from the Bode plot.

5. **(Q)**: What parameter of the Butterworth filter do we have to change to have a better attenuation in high frequencies? What is the trade-off when changing that parameter? *Hint: check the phase plot of the filters that you designed.*

Part 2: Filter and Fourier transform on sound samples

The second part of this lab uses the Matlab scripts and sound data located in the `part02` directory. Change your current directory to the latter.

The files `sample1.wav`, `sample2.wav` and `sample3.wav` are audio files sampled at 11025 Hz taken from movie dialogs. You can load them and listen to them in Matlab. First connect your headphones to the computer and **set its volume to a low level initially**. You can later set it higher if needed, but **be careful since some of the sounds recorded are not very pleasant**. Headphone outputs are muted in most lab's computers. You should go to System Settings and un-check the mute option.

In order to load a file into a Matlab array use the function `audioread('Filename')` (`Filename` includes the extension, .wav). To play it you can use the function `sound(var,Fs)` where `Fs` is the frequency used to sample the sound and needed to reproduce at the appropriate speed. Load and listen to `sample1.wav` setting the right frequency:

```
>> [y, fs] = audioread('sample1.wav');
```

```
>> sound(y, fs);
```

For versions of Matlab older than R2012b, use `wavread` instead of `audioread`.

6. **(S)**: Use the provided Matlab function `analyzeSoundSignal(y,fs)` to look at the time and frequency domain of the signal you just played. How does the spectrum look like? Where can you find the voice components?

7. **(I)**: Now play the same sound signal again setting higher and lower frequencies as `Fs` (e.g., 7000, 14000, 18000). How does the voice sound? Why does the pitch change?

8. **(S)**: Now load `sample2.wav`, listen to it (**watch out, sound might not be pleasant**) and look at how it looks in the time and frequency domain. The signal is composed from a dialogue from the same movie and an added signal. Can you guess which kind of signal is it? Where is it found in the frequency domain? *Hint: use the provided Matlab function `analyzeSoundSignal` to look at the time and frequency domain of the signal and uncomment the line 18 to set the y axis as needed.*

9. **(Q)**: As you could probably guess, `sample2.wav` is the sum of a signal representing a voice and a sinusoid of high frequency. If we want to remove the sinusoid signal in order to better listen the speaker, what kind of filter would we need? Low-pass, high-pass, band stop or band pass? Consider that we might be able to discard sound components above a certain frequency and still be able to recognize a voice.

One way to implement a digital filter is by using the following formula. This form of a digital filter is known as the *Direct Finite Impulse Response (FIR) Form*.

$$y[n] = \sum_{k=0}^{M} b_k x[n-k]$$

This means that we get our actual filtered value `y[n]` by summing up `M` previous unfiltered values `x[n-k]` where each previous value has been multiplied with the corresponding coefficient b_k. When we design a filter the final result is this set of coefficients b_k. The number of coefficients (`M`) is called the filter order. The higher the filter order the more the *real* filter approaches an *ideal* filter.

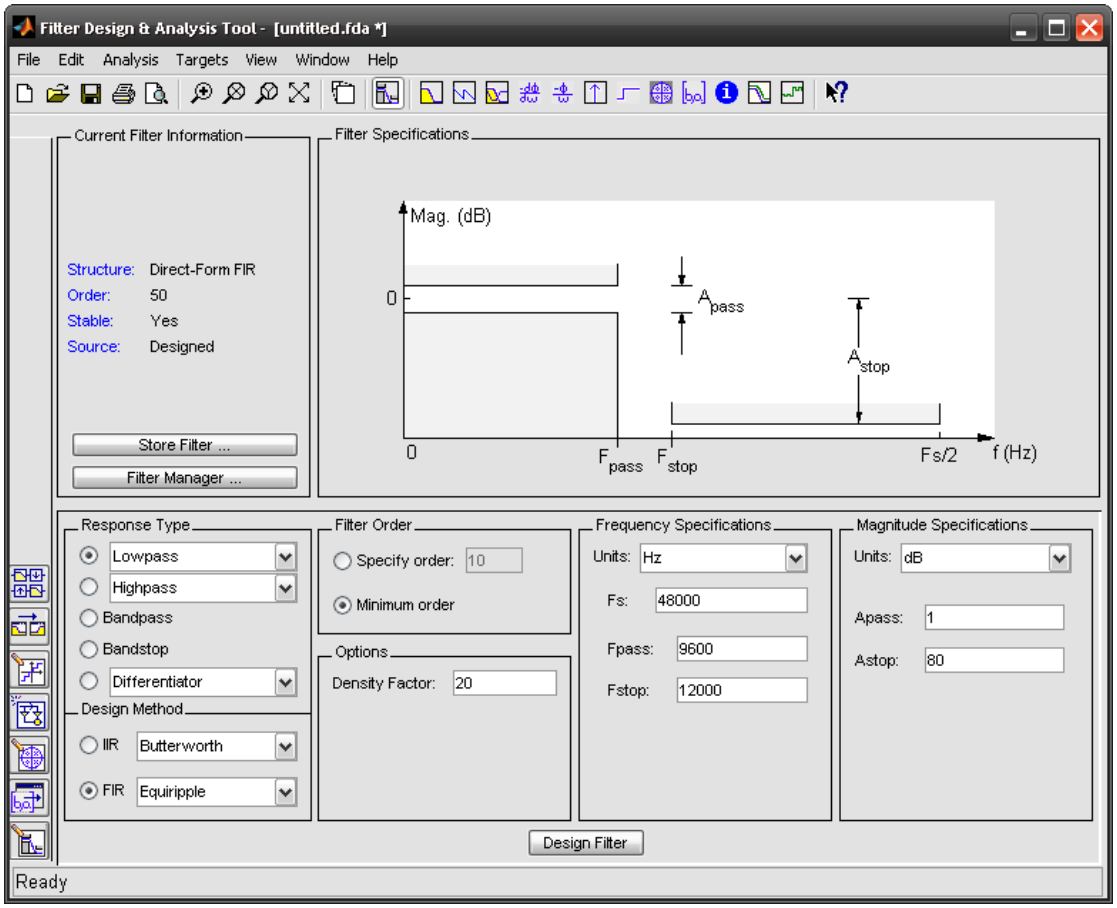
10. **(Q)**: Suppose you have a filter with coefficients $b = [-2,1,0,-1,2]$defined for k = [0:4]. Write the analytical equation of the filtered signal y as a function of x.

We will now use Matlab's filter design tool to compute the coefficients for the filter we need. We decided to use initially a low-pass filter and keep only frequencies below 2 kHz. Later we will use a stop-band filter at 2 kHz.

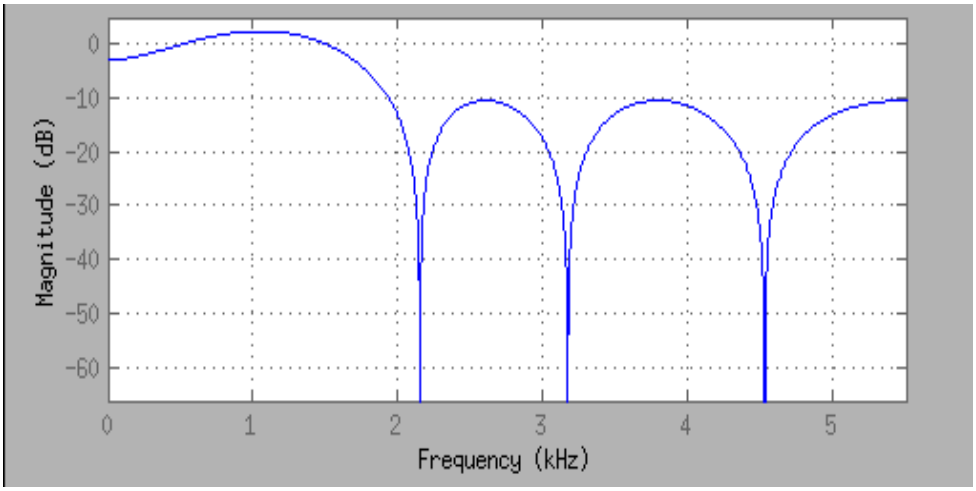
Open Matlab's filter design tool by typing `filterDesigner` at the Matlab prompt. The window which opens should look like the figure below. Now we set the following parameters (read the following paragraph very carefully):

- Make sure that in `Response type` we select `low-pass` as we would like to design a low-pass filter.
- In `Design Method` select `FIR equiripple`.
- Now mark `Specify order` and set it to `10`. (More on the importance of the filter order in a later question)
- Next, we will have to tell the design tool at which frequency our signal has been sampled. In our case it is 11025 Hz, so set F_S to 11025 and make sure that `Units` is set to `Hz`.

- We want our filter to filter out all the high-frequencies above 1900 Hz, in this way we will filter the sinusoid at 2 kHz, and keep most of the voice frequencies. `Fpass` indicates the frequency up to which the signal should be unfiltered, so we set `Fpass` to `1700 Hz`.
- If we would have an ideal filter everything above `Fpass` would be completely filtered out. In the real world however, we can never completely filter out anything. `Fstop` indicates the frequency at which the higher frequencies have been filtered out to a certain degree. Set this value to `1950 Hz`.
- By now clicking on `Design Filter` at the bottom `filterDesigner` calculates the coefficients of the filter corresponding to your specifications. This may take a few seconds.



After the filter computation is done, you should now see the magnitude response of the filter in the upper right window which should look like this.



Note how frequencies above `Fstop` are still present as the magnitude is not $-\infty$, which would correspond to perfect filtering.

Note: In order to obtain the frequency response of a digital filter whose coefficients are known in z-domain, Matlab's `freqz()` *function can also be used. Furthermore,* `freqz()` *can also be utilized with* `designfilt()` *function to obtain the frequency response similar to* `filterDesigner` *tool.*

11. **(I)**: Change `Specify order` from `10` to `140`, then hit `Design Filter` to recompute the filter. What happens to magnitude response?

12. **(Q)**: Which filter is "better"? The one with order 10 or the one with order 140?

13. **(Q)**: What could be the disadvantage of a filter with a high filter order? *Hint: look at the phase plot of the filters you designed in the* `filterDesigner`.

Note: You can switch the plot to show magnitude and/or phase response with the buttons in the toolbar above. These buttons are shown below:

Now we would like to apply the filter with order 140 to our sound signal in order to listen properly to the voices.

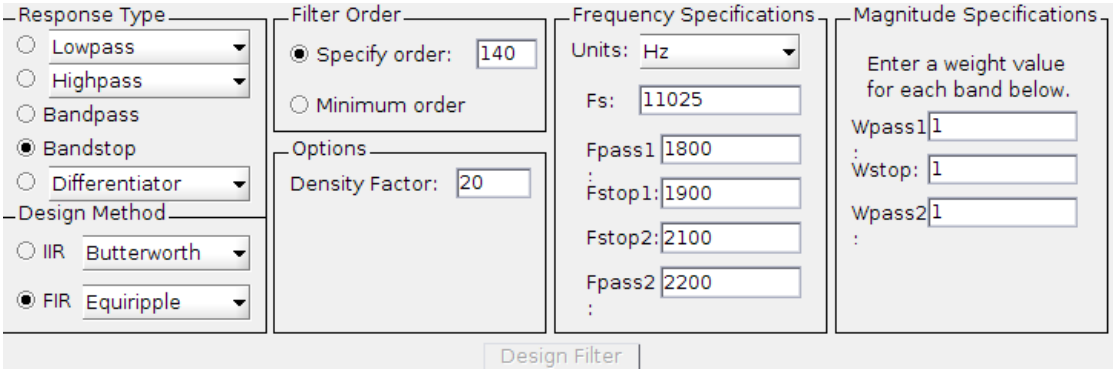
14. **(I)**: Export the computed coefficients to the Matlab workspace:

- On the window of Filter Design & Analysis Tool, click on `File > Export ...`
- Click on the `Export` button.
- In your workspace you should now have a variable called `Num` which contains the 141 coefficients.

15. **(I)**: Filter the data using the Matlab routine `filter_data.m`, play the resulting signal and look at its time and frequency domains. Can you listen properly to the voice? Is the tone at 2 kHz still present? Compare also with the original signal.

```
y_filt=filter_data(y,Num);
sound(y_filt, fs);
analyzeSoundSignal(y_filt,fs);
```

By implementing a low-pass filter we filtered some high frequency components that might be present initially. Now again using `filterDesigner` we will design a stop-band filter, which will remove the frequency components in a determined band. You can start with the following settings, and explore the resulting filter:



16. **(I)**: Find your desired filter by tuning if needed the above parameters, export it as in Question **14** and repeat Question **15** for the new filter. How does the frequency domain plot compare for the three signals (original, filtered in Question **15** and filtered here)?

17. **(I)**: Lastly you will work with `sample3.wav` file which corresponds to a voice of a well-known movie with two sine signals of different frequencies added. Using `analyzeSoundSignal`, determine these two frequencies and later implement the filtering needed to clean the signal. *Hint: you might want to design and apply two different filters using the `filterDesigner` and apply them sequentially.* Do you know to whom the voice corresponds?