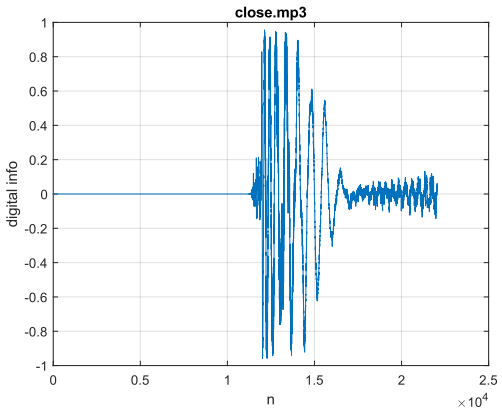
Student: MIRKO TENORE
ID: 311756

Signal analysis and processing
Lab Experience Session 3

Exercise 1

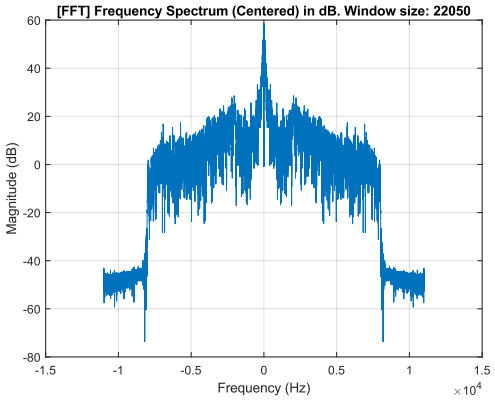


Analysis of the energy spectra generated by an audio file containing a song, in this case “c l o s e” by J. Cole, in MATLAB.

Firstly, the sampling frequency F_s was determined using the `audioread` function and was found to be **44100 KHz**, in line with known standards. The audio file was divided into time windows initially of half a second and then of integer multiples of a second (one second of audio corresponds to F_s samples) and the spectrum of the beginning of the song was analyzed.

It was generated in two ways (these first plots are related to the first 0.5 seconds of the audio file):

· Using the built-in function `fft()`

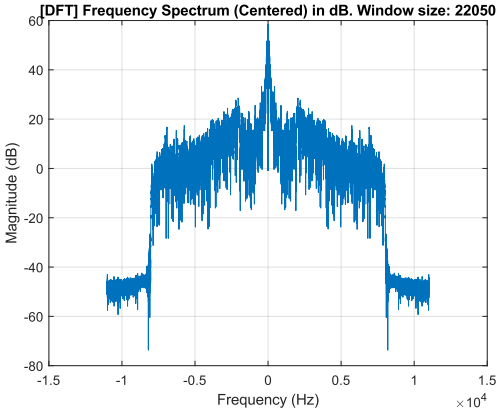


The computation time is very low even for a high number of samples, since `fft()` is based on a very optimized algorithm, the Fast Fourier Transform. `fftshift()` is used to rearrange the resulting vector so that the negative and positive frequencies are symmetrically distributed around zero.

Resulting computational times (`M` is the number of samples):

```
[FFT] M: 22050   - Elapsed time is 0.000435 seconds.
[FFT] M: 88200   - Elapsed time is 0.001203 seconds.
[FFT] M: 2205000 - Elapsed time is 0.029185 seconds.
[FFT] M: 8820000 - Elapsed time is 0.143223 seconds.
```

· Manually implementing the DFT



Given the audio signal $x(n)$ with W samples (equal to the size of the chosen window), the DFT was obtained as:

$$X(k) = \sum_{n=0}^{W} x(n) e^{-j2\pi n \frac{k}{W}}, \forall \mathrm{k} = 0, 1, \dots, \mathrm{W} - 1$$

Comparison between computational times (`M` is the number of samples):

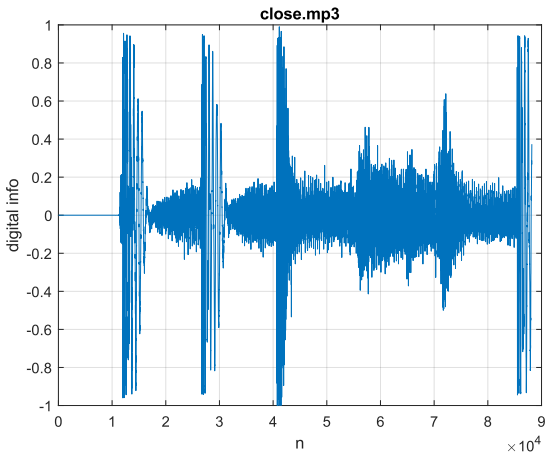
```
[FFT] M: 22050 - Elapsed time is 0.000750 seconds.
[DFT] M: 22050 - Elapsed time is 91.524448 seconds.
[FFT] M: 44100 - Elapsed time is 0.001866 seconds.
[DFT] M: 44100 - Elapsed time is 301.938532 seconds.
[FFT] M: 88200 - Elapsed time is 0.001066 seconds.
[DFT] M: 88200 - Elapsed time is 1152.182626 seconds.
```

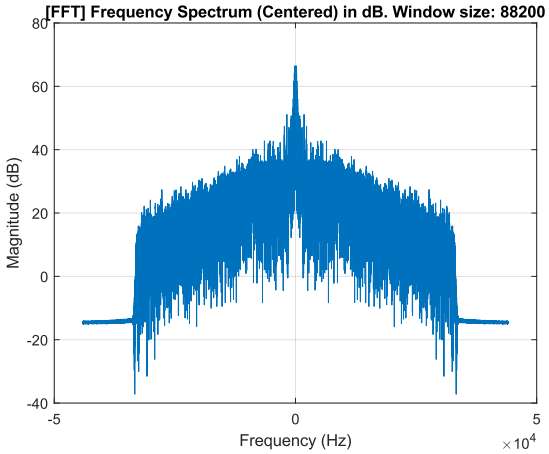
As can be seen from the elapsed times, the implementation of the **manual DFT requires much more time**, being a non-optimized algorithm with **complexity $O(W^2)$** where W is the number of samples to be processed, so it’s much more computationally expensive.

The energy spectra, calculated as $E = |X_k|^2$, were plotted in dB.

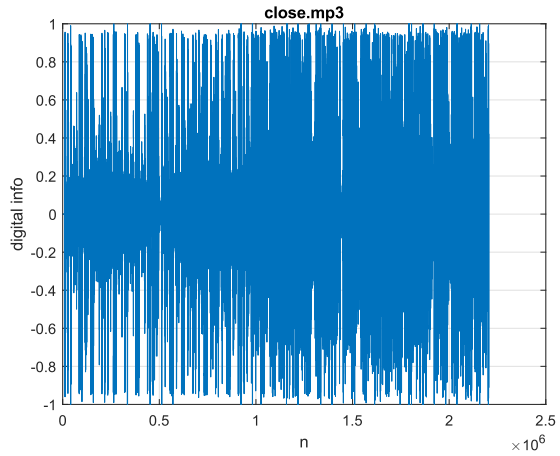
The next page shows some plots considering different lengths for the time window, starting from the first sample of the song.

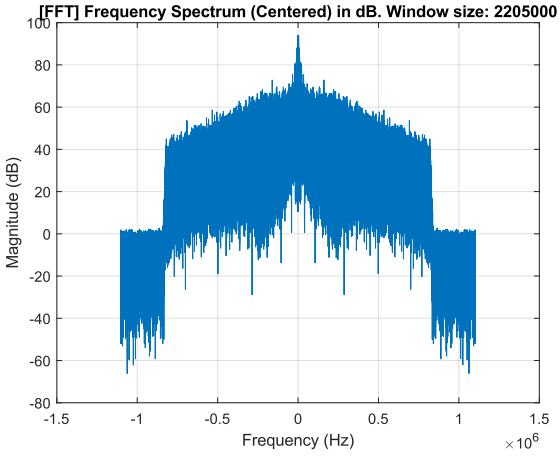
Considering a 2 second window (88200 samples)



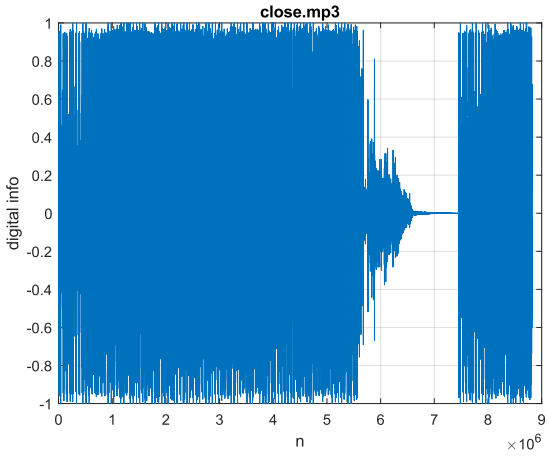


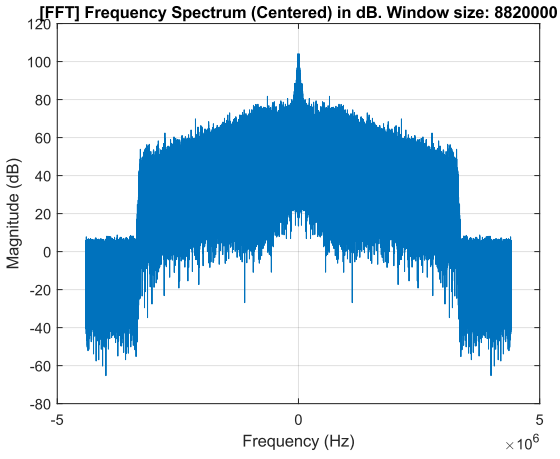
Considering a 50 second window (2205000 samples)





Considering a 200 second window (8820000 samples)

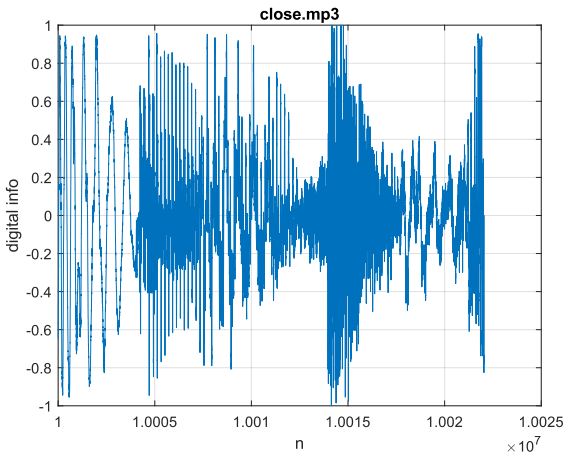


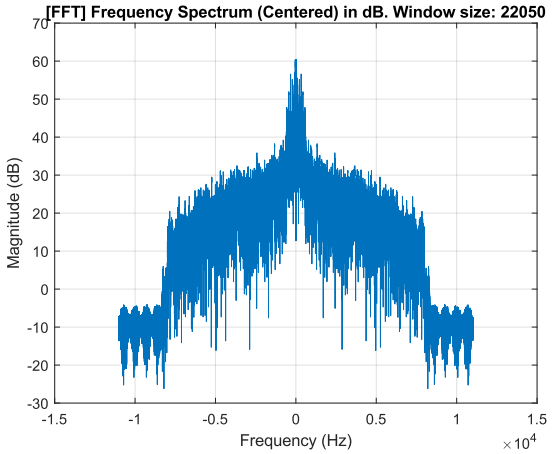


The first thing we notice is that a **larger window width** corresponds to a larger amplitude and therefore **higher energy values**. Clearly, these energy values also **depend on the signal window** being analyzed: at the beginning, lower values will be noted compared to the "livelier" refrain.

Another time window from a middle point of the audio file was analyzed and reported below.

Considering a 0.5 second window (88200 samples) in the middle of the song

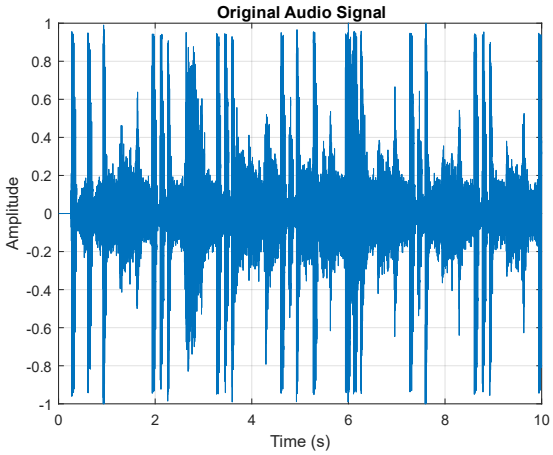


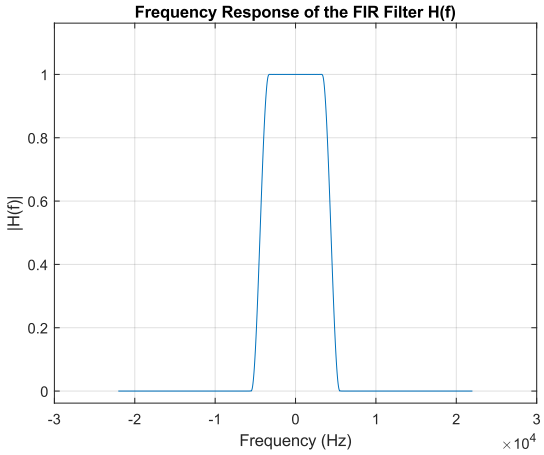


The spectrum of these few seconds, compared to the one previously analyzed of the same duration but at the beginning of the audio, is less dense, while during the chorus it becomes **more dynamic and populated**, reflecting a more intense or energetic musical section.

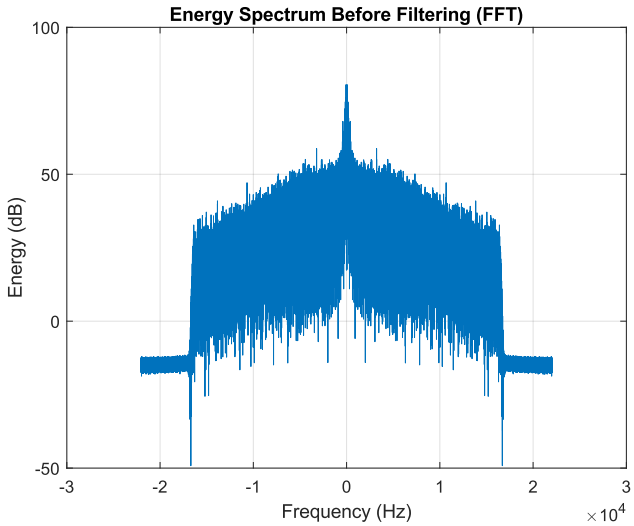
Exercise 2

A portion of the music file of exercise 1, in this case 10 seconds, was filtered using the given FIR filter by importing into MATLAB the vector **h** that contains its impulse response.





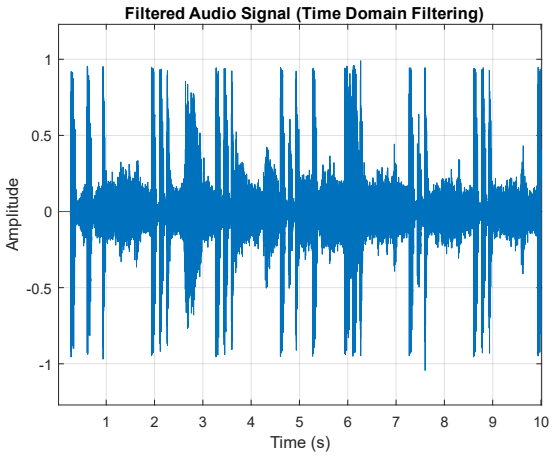
The spectrum of the original audio signal before applying the filter is reported below.

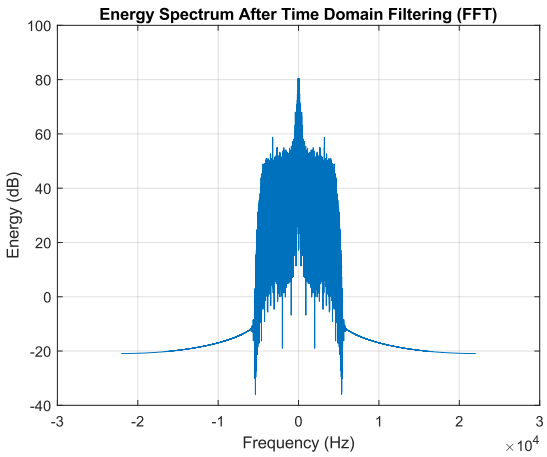


Filtering was performed in two ways:

· Using convolution in time domain

Simply using the built-in MATLAB function `conv()` between the 10-second audio segment and the vector `h`.





Even cutting a significant portion of the high frequencies, the filtered signal is visually like the non-filtered one in time domain.

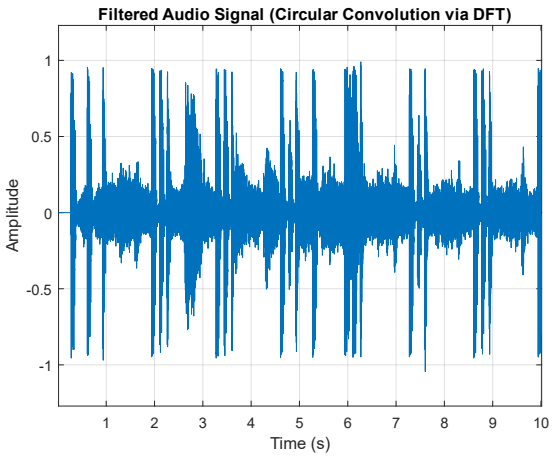
· Using the circular convolution in frequency domain

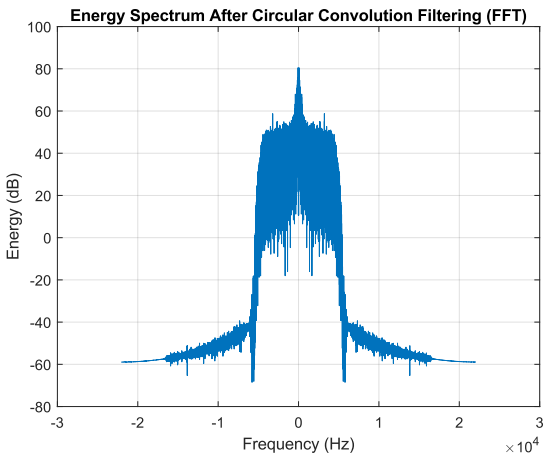
This approach becomes an alternative even with non-periodic signals thanks to the addition of zero samples to the arrays containing the audio signal and to `h`, such that the arrays have size equal to (or greater than) $N_x + N_h - 1$ in order not to have overlapping. This operation is called **zero padding**.

Done this, if $x[n]$ is the audio signal, the filtered signal at the output can be computed with DFTs as:

$$Y(k) = X(k)H(k)$$

just using `fft()` and `fftshift()` in the code. To get $y[n]$ in time domain, we can use `ifft()`.





From the plots it can be seen how the linear convolution correctly reflects the filter characteristics, both in-band and out-of-band, while the circular convolution with DFT, even with correct zero-padding, introduces **circular aliasing** and unwanted interferences due to the periodicity imposed by the DFT. The spectrum of $X(f)$ and $H(f)$ is repeated periodically, and the out-of-band frequencies can interfere with the original ones causing irregularities. So, this is the reason why the **linear convolution** provides "**clean**" results, while the circular convolution can have out-of-band **irregularities**.