

Edgar Daniel Chaguaro Aldaz

Digital Signal Processing Filtering Algorithm

Audio Equalization Using Matlab

Helsinki Metropolia University of Applied Sciences

Bachelor of Engineering

Degree Programme in Electronics

Thesis

7 May 2015

Author Title	Edgar Daniel Chaguaro Aldaz Digital Signal Processing Filtering Algorithm
Number of Pages Date	40 pages + 3 appendices 07 May 2015
Degree	Bachelor of Engineering
Degree Programme	Electronics
Instructor	Thierry Baills, Senior Lecturer
<p>The contemporary domain of Digital Signal Processing is in constant influx and trying to find new applications that will benefit the everyday life of ordinary people. In modern technology, most of the electronic processes use DSP algorithms in order to collect analogue information that is continually present all around us and convert it into a digital form. The need of understanding the basics of how these processes occur, has inspired to implement a DSP application for educational and testing purposes.</p> <p>The purpose of this thesis is to analyse the different filtering algorithms for DSP and how are they implemented using Matlab. Furthermore, the study attempts to apply one of these algorithms in a convenient way to implement a basic, user friendly, graphic equalizer. At the same time, the study attempts to focus on comprehensive and practical ways of learning and understanding the use of the DSP Filtering.</p> <p>The equalizer is implemented using Matlab and the tools that this software brings in concerning the DSP area. The use of Matlab on DSP has become a standard method for students due to its relative easiness to use and understand.</p> <p>The equalizer designed consists of a set of filters working together in its different own domain of frequencies to create the effect equivalent of an analogue equalizer. This application allows processing input audio data, control the parameters of the filters and displaying the graphical representation of the filter channels on the screen of a computer running Matlab.</p> <p>The students are now able to introduce an audio signal to the application. They are also able to manipulate this audio signal using the equalizer functions, as well as change the output result of this signal audible. This will help to understand the practical way to use the filters and their particular behaviour.</p>	
Keywords	Digital Signal Processing, Matlab

Tekijä Otsikko	Edgar Daniel Chaguaro Aldaz Digitaalisignaalin käsittely ja suodatus algoritmi
Sivumäärä Aika	40 sivua + 3 liitettä 7.5.2015
Tutkinto	Insinööri (AMK)
Koulutusohjelma	Elektroniikka
Ohjaaja	Lehtori Thierry Baills
<p>Nykyaikainen digitaalisen signaalinkäsittelyn ala laajenee jatkuvasti ja pyrkii löytämään uusia sovelluksia, jotka hyödyttävät tavallisten ihmisten arkea. Suurin osa modernin teknologian sähköisistä prosesseista käyttää DSP-algoritmeja saadakseen analogista tietoa, joka on jatkuvasti läsnä kaikkialla, muuntaen sen digitaaliseen muotoon. Tarve ymmärtää miten nämä prosessit esiintyvät, on innoittanut kehittämään DSP-sovelluksen opetus- ja testitarkoituksiin.</p> <p>Tämän työn tarkoituksena oli analysoida eri DSP:n suodatus-algoritmeja sekä sitä, miten ne on toteutettu käyttäen Matlabia. Lisäksi tutkimus pyrki käyttämään yhtä näistä algoritmeista käytännöllisellä tavalla, jotta voitaisiin toteuttaa käyttäjäystävällinen ja graafinen taajuuskorjain. Tutkimuksessa keskityttiin kattavasti ja käytännöllisesti tapoihin oppia ja ymmärtää DSP-suodatuksen käyttöä.</p> <p>Taajuuskorjain toteutettiin Matlabin ja sen ohjelmiston DSP-työkalujen avulla. Matlabin käytöstä DSP:ssa on tullut vakiomenetelmä opiskelijoille johtuen sen suhteellisen helpposta käytettävyydestä ja ymmärtävyydestä.</p> <p>Suunniteltu taajuuskorjain koostuu suodatin-valikoimasta, jotka työskentelevät yhdessä niiden eri taajuuksien toimialueilla, tarkoituksenaan luoda vaikutus vastaavaan analogiseen taajuuskorjaimeen.</p> <p>Tämä sovellus mahdollistaa sisään tulevien äänisignaalien käsittelyn, suodattimien parametrien hallinnan ja vastaavien graafisten suodatinkanavien esittämisen Matlab-tietokoneen näytöllä.</p> <p>Opiskelijat voivat nyt ladata äänisignaalin sovellukseen. Käyttäjät pystyvät myös käsittelemään äänisignaalia taajuuskorjaimen toiminnoilla sekä muuttamaan signaalien ulostuloa. Tämä auttaa ymmärtämään käytännöllistä tapaa suodattimien käytössä sekä niiden yksityiskohtaista käyttäytymistä.</p>	
Avainsanat	Tietokoneavusteinen signaalinkäsittely, Matlab

Contents

1	Introduction	1
2	Theory about Digital Signal Processing Applied to Audio Equalization	2
2.1	Simple Signal Processing Operations	4
2.2	Digital Filters	4
2.2.1	FIR and IIR Digital Filters	6
2.3	Dealing with DSP in the Matlab world	9
2.3.1	Digital Filtering on Matlab	12
2.3.2	DSP Algorithm Implementation	22
2.4	Audio Equalization (EQ)	27
2.4.1	Types of Equalizers	29
2.4.2	Graphic Equalization	30
2.4.3	EQ Analogue vs. EQ Digital	31
3	Audio Equalization Using Matlab	33
4	Results	37
5	Conclusions and Recommendations	39
	References	40
	Appendices	
	Appendix 1. Fourier and DFT on Matlab	
	Appendix 2. Example of filtering on Matlab	
	Appendix 3. EQ-ator equalizer implementation in Matlab	

1 Introduction

The signals found in the surrounding environment, and world around us in general, are essentially presented in an analogue form. This means that they are the representation of variations in physical quantities in any given time. These parameters include electricity, temperature, light, sound and pressure among others.

For these “natural” signals to be suitable for microprocessors they must be converted into a digital form through a process called Digital Signal Processing (DSP). In order to accurately represent these signals they need to go through certain types of pre-designed sampling and quantisation techniques based on different mathematical techniques.

Using DSP methods has numerous advantages over the traditional analogue methods, including greater accuracy, performance, flexibility, speed and cost. One of the common tools used together with the DSP is called Matlab.

Matlab is software that is easy to learn and understand allowing students to do most of the operations used on DSP, with the option of plotting and seeing on screen the graphical representation of processed signals. Another important characteristic about Matlab is that it includes an option for creating specific applications that will create shortcuts for solving certain fixed calculations in a simpler and faster manner. Finally interaction with and data processing from any external device is made possible by the basic serial over USB communication method used with Matlab.

The facilities that Matlab brings in its extensive area of engineering includes the capability of graphical implementation of functions with the GUI (Graphical User Interface) that make an easy, faster and more friendly way to interaction.

This is very useful for studying the way we can use algorithms instead of external circuits based of discrete components and operational amplifiers on e.g. filtering process. The different ways in which we can apply an algorithm, can be used to define how the information must be processed. Also, as the result can be displayed in graphical form, it is possible to visualize how the signal has been changed during the process. This, in turn, is helpful in understanding how signals behave in the so called natural world.

2 Theory about Digital Signal Processing Applied to Audio Equalization

In order to understand the extensive area of the Digital Signal Processing (DSP), it is necessary to have at least a basic understanding how it works. The core of the process deals with transforming the analogue signals into their digital equivalents. This is done with the help of Analogue to Digital (A/D) converters.

An A/D converter is a piece of hardware that converts an analogue signal into a digital one. Analogue signals pass through an input and through the processing are transformed into a form consisting of a combination of numbers that come through the output. Another similar procedure is the Digital to Analogue (D/A) converter, which often takes places at the end of DSP, these two are practically impossible to achieve through virtual systems or computer software. Sampling, quantization and coding steps are required in order to make the process of conversion possible. [1,281.]

Below is an example of how process of digitalization occurs when taking a small fragment of LF signal.

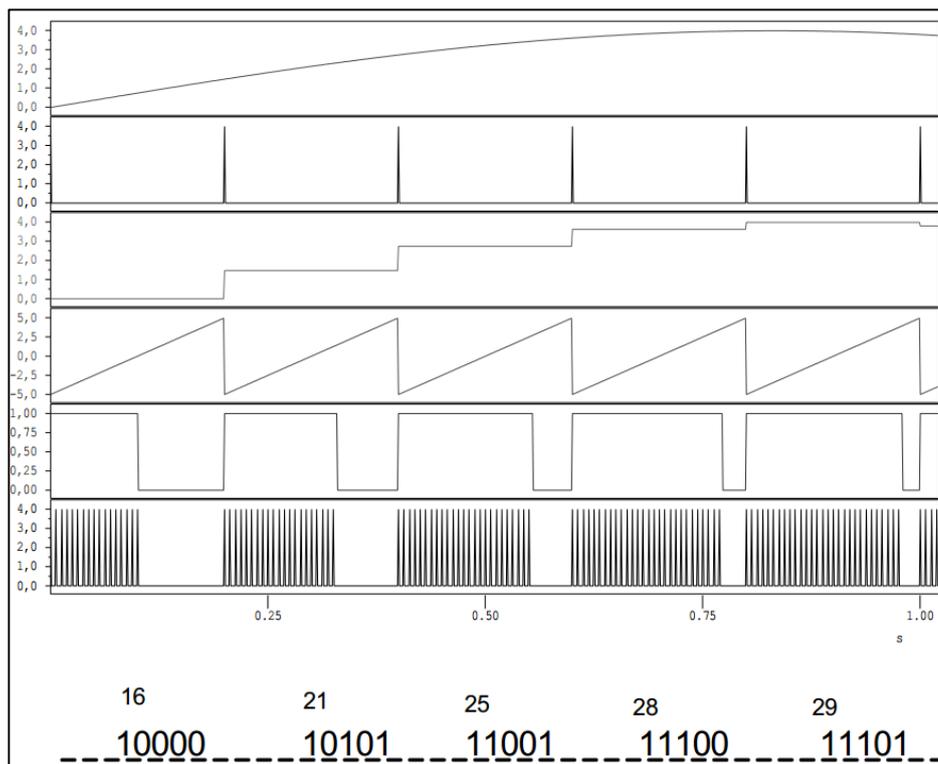


Figure 1 Process of sampling and quantization, reprinted from Ulrich (2006) [1,282]

As Figure 1 shows, sampling and quantization of the signal has transformed it into a string of numbers which appear at the bottom of the figure. It is important to note that

the digital expression in the output used a 5 bit code, meaning that the clock pulse frequency is 5 times the sampling.

The analogue signal which is continuous in time and amplitude must go through a basic process in order to be converted in to a digital one. This is acquired by putting it into a binary expression using zeros (0) and ones (1), by performing an analysis or making the results discrete valued. A discrete signal, which means that the signal is discrete in time but continuous in amplitude, is obtained through sampling. For the signal to be called digital it must be discrete in time and in amplitude; this is obtained through quantization, as shown in the Figure 2. Quantization is a process in which one sets specific values into a limit range of amplitude parameters of processed discrete signals. When comparing the original signal with the resulting after the process of quantization, we realize that the signals are not strictly identical because the presence of error of quantization. Error of quantization depends of the number of bits used on the conversion to the signal from analogue to digital and it is generally one-half of a low significant bit. The quantization error could be quantified by evaluating the Signal to Quantization Noise Ratio (SQNR) that it is in few word equal to: $SQNR = (1.76 + 6.02B)$ where B is the data representation in binary bits.

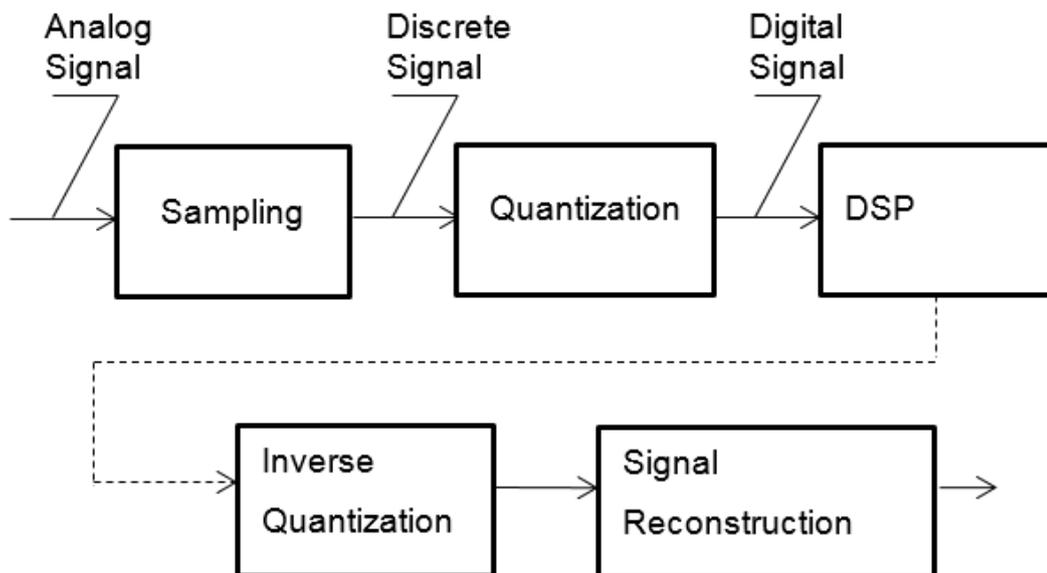


Figure 2. Diagram block of digitization process modified from Priyabrata (2010) [2,12].

It is important to consider how to avoid distortion due to the overlapping of the frequency bands. This is also known as aliasing. One way to avoid it is to limit the bandwidth of the spectrum with the help of an antialiasing filter. The theorem exposed by Nyquist-Shanon, consist of a low pass filter (antialiasing filter) that confirms that the signal bandwidth is smaller than the half of the sampling frequency (F_s). It would be considerably more convenient to verify this before the signal has gone through the process of sampling.

The Shanon theorem has the following mathematical notation:

$$x(t) = \sum_{k \in \mathbb{Z}} x(kT) \operatorname{sinc} \left(\frac{\pi}{T} (t - kT) \right). \quad (1)$$

Where k is an integer and T is the sampling period.

2.1 Simple Signal Processing Operations

There are three basic operations when dealing with signals that apply to digital filters: delay, scaling and addition. These operations are combined and implemented in order to obtain the desired result signal at the output of the process. The delay operation, which generates a duplicate of the original signal but delayed in time t_0 , is implemented first. So, if $x(t)$, then $y(t) = x(t - t_0)$.

Scaling on the other hand, is a simple multiplication of the given signal by a constant "a", so $y(t) = ax(t)$.

Finally the addition is the operation in which two or more signals join together forming a new signal: $y(t) = x_1(t) + x_2(t)$. [4, 4.]

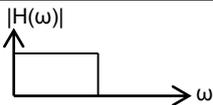
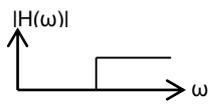
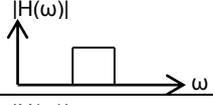
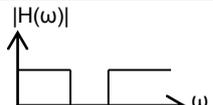
2.2 Digital Filters

According to Priyabrata, the process of selecting frequencies or range of frequencies in a given signal and the attenuation of the frequency components outside the chosen range is called filtering. [2, 25]. In general the signals we receive are mixed with other kinds of undesired signals called noise, so the filter is a discrete processor in which the signal desired at output is the result of the signal with the noise attenuated [3,103]. The frequency range allowed to pass through the filter is called *Pass Band*, and the

range where the signal is cancelled is the *Stop Band*. The separation area between passband and stop band is called *Cut-off frequency*.

Digital filtering is one of the most powerful tools in DSP. The characteristics of a digital filter can easily be changed under software control and this characteristic brings flexibility and stability to its use. DSP filters are easy to simulate and design and they don't drift due to component modifications. There is available a wide array of adaptive filters applications, including echo and noise cancellation, speak recognition and etc. There are four elementary filters that will be considered here, and a brief description of each of them is available on the table below.

Table 1. Elementary filters description

<u>Filter Name</u>	<u>Simplest Math. Expression</u>	<u>Description</u>	<u>Graph. representation</u>
Low Pass Filter (LPF)	$\frac{1}{2}(1 + z^{-1})$	allow low-frequency components of f_p .	
High Pass Filter (HPF)	$\frac{1}{2}(1 - z^{-1})$	allow high-frequency components of f_p .	
Band Pass Filter (BPF)	$\frac{1}{2}(1 - z^{-2})$	pass frequencies inside a specific range.	
Band Stop Filter (BSF)	$\frac{1}{2}(1 + z^{-2})$	attenuate frequencies inside a specific range	
*All Pass Filter (APF)	z^{-N}	pass all frequencies uniformly in gain and change the phase relationship.	

In the Table 1, f_p represents the passband edge frequency. In the band pass filter, there is two band pass edge frequencies where $f_{p1} < f_{p2}$. All the frequencies between this ranges are allowed and all band stop frequencies (f_s) above f_{s1} and below f_{s2} . are blocked. The band stop filter, on the other hand, allows all the frequencies above f_{p2} and below f_{p1} and blocks all those which are inside f_{s1} and f_{p2} .

2.2.1 FIR and IIR Digital Filters

Digital signal filters are divided into two fundamental types based on their impulse response.

The Finite Impulse Response (FIR) filter has an impulse response in which the number of non-zero samples are finite. It is represented in terms of the phase response, also linear and time invariance. It is a filter of easy design and considerable computational intensity. It is a feed-forward filter without feedback. Each output sample is a sum of weighted input samples. The mathematical representation of a FIR is the convolution of:

$$y(k) = h(k) * x(k); \quad (2)$$

$$y(k) = \sum_{m=0}^{N-1} h(m)x(k - m); \quad (3)$$

So, the transfer function is given by:

$$H(z) = \frac{Y(z)}{X(z)} = h_0 + h_1z^{-1} + \dots + h_{N-1}z^{N-1}. \quad (4)$$

The FIR filter has a finite impulse response, and due to this it is considered stable. The stability is guaranteed because the poles of the system are located inside of the unit circle on the z-domain. In other words, the poles of the z-domain transfer function are located in the origin [5,124].

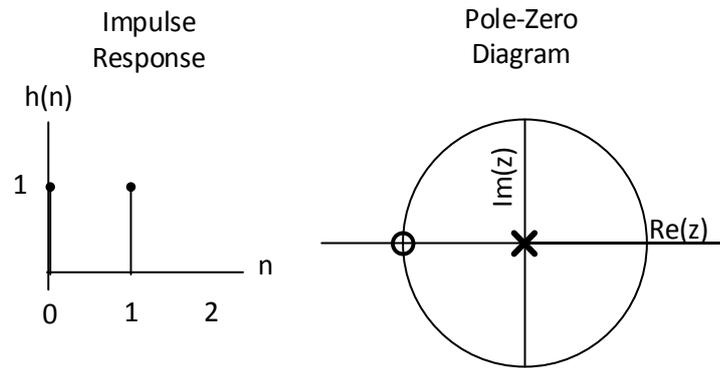


Figure 3. Impulse Response and stability characteristics of an FIR filter

In the left side of the Figure 3 is the basic given finite impulse response and on the right side the unit circle with the real and imaginary axes with the poles (X) and zeros (O) inside.

The Figure 4 gathered from Markovic and Brodersen (2012) [5,117] shows an example of the architecture implemented for an FIR filter, where the output signal requires N number of multiply-accumulates for each output.

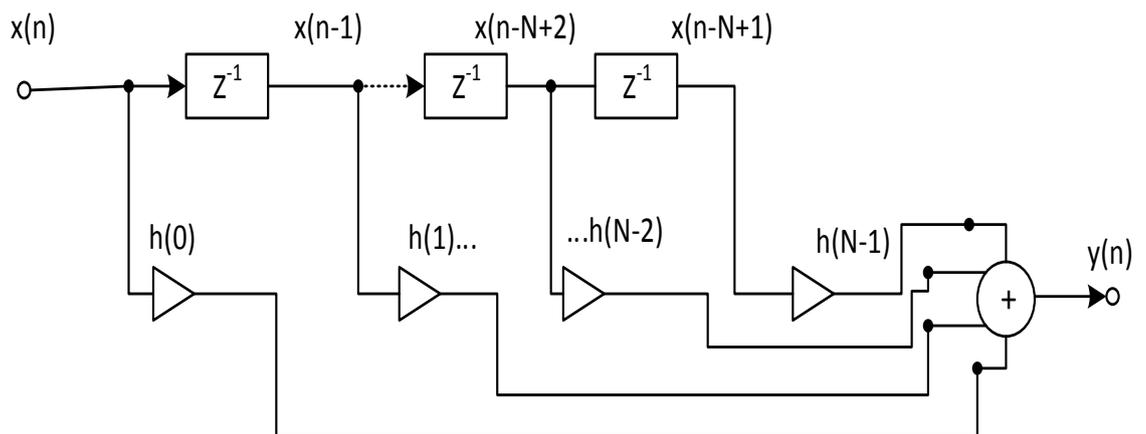


Figure 4. Example of an FIR filter structure, gathered from Markovic and Brodersen (2012) [5,117]

A series of additions and multiplications represent the N-tap finite impulse response function.

The Infinite Impulse Response (IIR) filter is generally based on the idea of classical analogue filters. It has an impulse response in which the number of non-zero samples are infinite. IIR filter is recursive: this means that it presents feedback and that the output is based on the combination of feed-forward inputs and feedback outputs.

$$y(k) = \sum_{m=0}^{N-1} h(m)x(k-m); \quad (5)$$

$$y(k) = \sum_{m=0}^N a_m x(k-m) - \sum_{m=1}^M b_m y(k-m); \quad (6)$$

$$H(z) = \frac{\sum_{m=0}^N a_m z^{-m}}{1 + \sum_{m=1}^M b_m z^{-m}}; \quad (7)$$

The z-transform of IIR is:

$$H(z) = \frac{Y(z)}{X(z)} = \frac{a_0 + a_1 z^{-1} + \dots + a_N z^{-N}}{1 + b_1 z^{-1} - \dots - b_M z^{-M}}; \quad (8)$$

The architecture of the IIR transfer function in the z-domain could be represented as shown in the Figure 5 below.

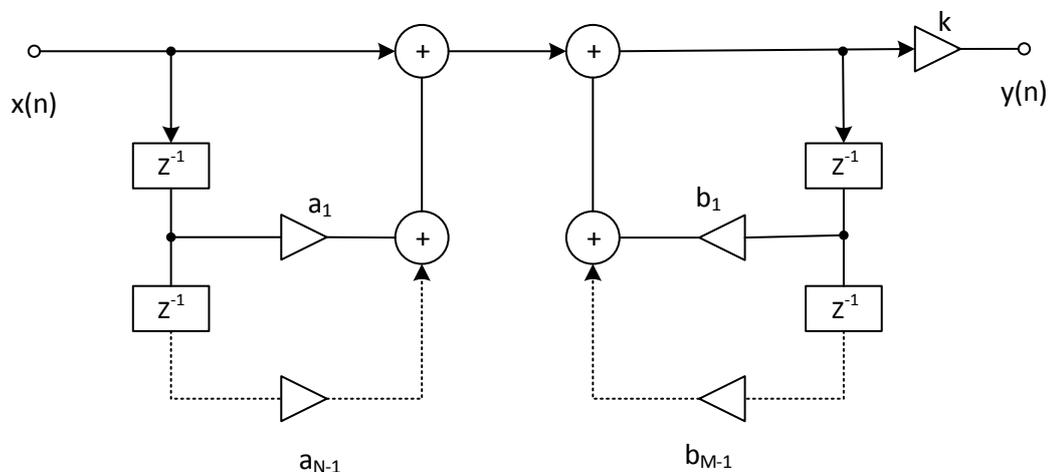


Figure 5. An IIR Architecture, modified from Markovic and Brodersen [5,125]

As shown in the Figure 5 modified from Markovic and Brodersen (2012) [5,125], the left side represents the feed-forward part of the IIR filter characteristic. It is also the numerator represented in $H(z)$. The right section on the other hand represents the feedback property of IIR. This is also the denominator in the z-domain transfer function [6, 91-108].

IIR filters are considered computationally efficient and non-linear. The stability can be checked by applying the principle of stability on the equation (9) below:

$$\sum_{k=0}^N h(k) = \frac{1 - a^{N+1}}{1 - a}; \text{ and } \frac{N \uparrow \infty}{|a| < 1} \rightarrow \begin{cases} \text{if } a < 1 \rightarrow \text{stable} \\ \text{if } a > 1 \rightarrow \text{unstable} \end{cases}; \quad (9)$$

2.3 Dealing with DSP in the Matlab world

Matlab has become an important tool in the DSP area. This is due to the fact that it can involve an extensive number of operations without a deep knowledge of programming or high elaborated commands on the command window. A vast data base provides a wide range of information, just with the help of “help” –command. For example when working with signals it is important to have a background knowledge about a process called the Fourier Transform. The Fourier Transforms are easily executed with the help of `fft` command in order to calculate the Discrete Fourier Transform (DFT) on the analysed signals. An example of the use of `fft` command line is on the Appendix 1, modified from Engelberg’s (2008) [7, 48]. The respective graphical results elaborated on Matlab and transformed with the command “`print -djpeg comp.jpg`” are shown in the Figure 6.

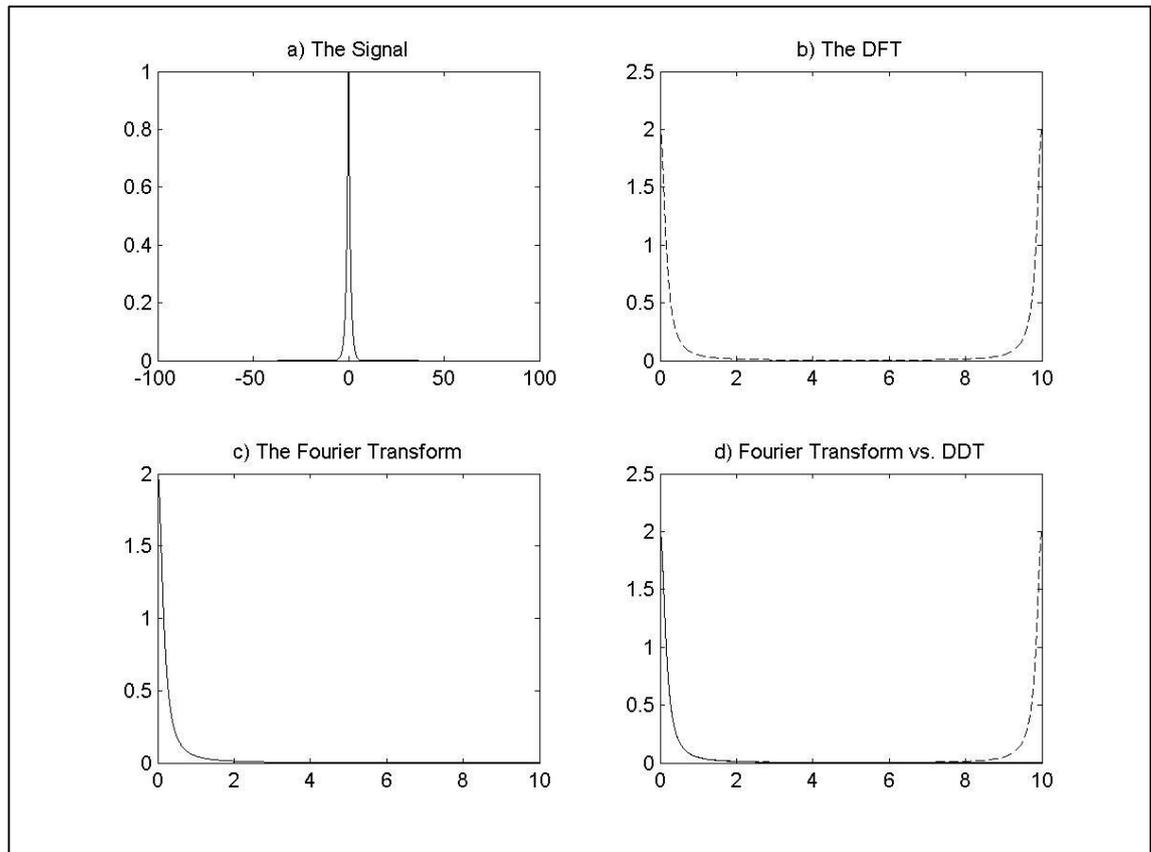


Figure 6 Construction of a signal, Fourier analysis and Discrete Fourier Transform

We can observe the comparison of the signal $y(t)=e^{-|t|}$ and the Fourier transform $Y(f)=2/[(2\pi F)^2+2]$.

With Matlab it is possible to check the stability of a digital system by analysing the z-plane and the relation between poles and zeros inside the unit circle, where coordinates are in the real and imaginary planes corresponding to x- and y axes.

A normal way to find the zeros and poles of a linear system is demonstrated by the following mathematical expression:

$$H(z) = \frac{b_0 + b_1 z^{-1} + \dots + b_{N-1} z^{-(N-1)}}{1 + a_1 z^{-1} + \dots + a_M z^{-(M-1)}}; \quad (10)$$

$$H(z) = z^{M-N} \frac{b_0 z^{N-1} + b_1 z^{N-2} + \dots + b_{N-1}}{z^{M-1} + a_1 z^{M-2} + \dots + a_{M-1}}. \quad (11)$$

For example for represent graphically the zeros and poles on the z-plane is possible with:

```
b = [0.2  0.3  0.4  0.5  0.6];
a = [1  -0.3  0.5  0.07  0.9];
zeros = roots(b)
poles = roots(a)
```

The result is represented as shown in listing 1.

```
zeros =                                poles =
    0.3369 + 1.2587i                    0.6757 + 0.8080i
    0.3369 - 1.2587i                    0.6757 - 0.8080i
   -1.0869 + 0.7653i                   -0.5257 + 0.7314i
   -1.0869 - 0.7653i                   -0.5257 - 0.7314i
```

Listing 1. Poles and zeros values on Matlab command prompt for an unstable filtering

In the figure 7, on the left side of the figure we observe the corresponding position of the zeros and two poles in the region of convergence; as well as two more poles inside the unite circle on the pole-zero plot. This represents an unstable system. But on the right side of the same figure, numerator and denominator are given different values for the transfer function: the graphic shows that the poles and zeros inside the unit circle. This kind of a system is considered stable.

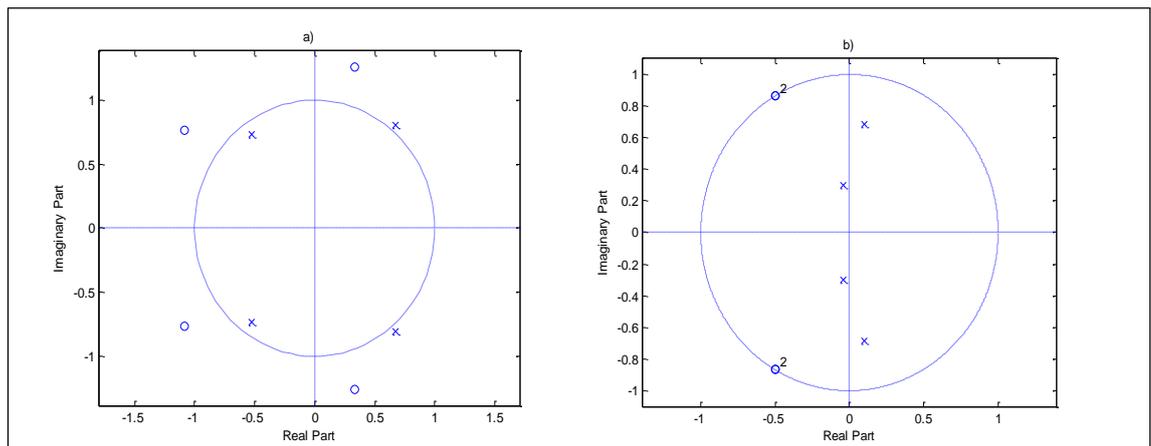


Figure 7. The pole-zero plot

Listing 2 below portrays the values of the transfer function corresponding to the b part of the figure 7.

```
d = [0.1 0.2 0.3 0.2 0.1];
c = [0.9 -0.1 0.5 0.02 0.04];
zeros = roots(d)
poles = roots(c)
zplane (d,c)
```

zeros =

```
-0.5000 + 0.8660i
-0.5000 - 0.8660i
-0.5000 + 0.8660i
-0.5000 - 0.8660i
```

poles =

```
0.0976 + 0.6854i
0.0976 - 0.6854i
-0.0421 + 0.3016i
-0.0421 - 0.3016i
```

Listing 2. Poles and zeros values on Matlab command prompt for an stable filtering

2.3.1 Digital Filtering on Matlab

The theoretical part concerning digital signal filtering is very broad and rather complicated in many parts and cases. For this reason some parts are occasionally supported with practical examples.

As mentioned before, there are four elementary filters: Low-pass, high-pass, band pass and band stop. These four can be used inside the two groups of the fundamental filters according to their impulse response.

The filter order is designed with “N” and it determines if the filter is a low-order or a high-order filter and in Matlab this value place a cutoff frequency value.

Let’s take some examples from the Matlab documentation. If we want to implement a low pass and high pass FIR filter of order 10 with a -6dB frequency of 9.6 kHz and a sampling frequency of 48 kHz, the command line comes as shown below in linting 3:

```
d=fdesign.lowpass('N,Fc',10,9600,48000);  
designmethods(d)  
Hd = design(d);  
% Display filter magnitude response  
fvtool(Hd);
```

```
d=fdesign.highpass('N,Fc',10,9600,48000);  
designmethods(d)  
Hd = design(d);  
% Display filter magnitude response  
fvtool(Hd);
```

Listing 3. FIR low-pass and high pass filter implementation in Matlab

Executing the line commands on Listing above, we get the plot in the frequency domain as shown in the figure 8 below.

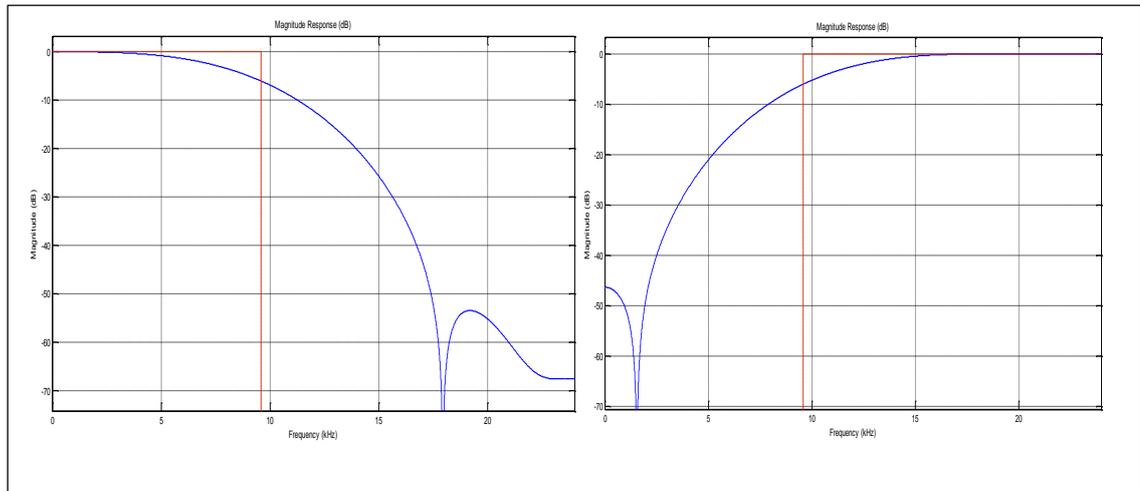


Figure 8. Low pass and high pass filter designed in FIR window method

In the next example in listing 4, we consider band pass and band stop filters using sampling frequency at 10 kHz, and two -3dB frequencies of 1 kHz and 2 kHz. There are a 10 order filter.

```
d = fdesign.bandpass('N,F3dB1,F3dB2',10,1e3,2e3,1e4);
Hd = design(d,'butter');
fvtool(Hd)
```

```
d = fdesign.bandstop('N,F3dB1,F3dB2',10,1e3,2e3,1e4);
Hd = design(d,'butter');
fvtool(Hd);
Hd = design(d,'butter');
```

Listing 4. IIR band pass and band stop filter implementation on Matlab.

Again, executing the line commands on Listing 4 above, we get the plot in the frequency domain as shown in the figure below.

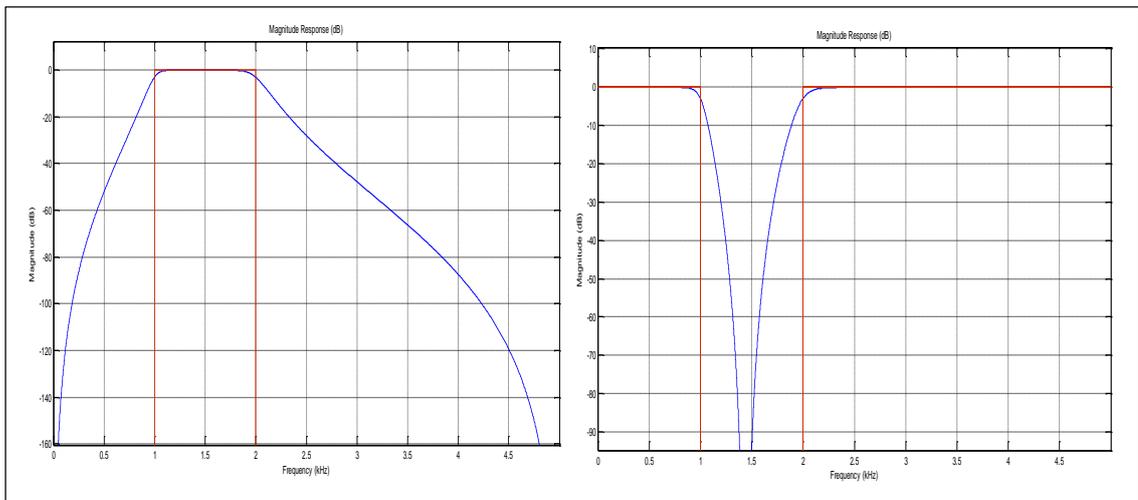


Figure 9. Band pass and band stop filter representation.

Matlab also allows to implement different filters including butterworth (`butter`), chebychev (`cheby1`, `cheby2`) as well as elliptic (`ellip`) filters, that can be both, analogue or digital [6,57].

Through applying these concepts we can give an example about filtering a sinusoidal signal with noise injected in it as illustrated in the Figure 10 below. The signal was generated following the instructions series below in listing 5:

```
x = cos(2*pi*12*[0:0.001:1.23]);
x(end) = [];
[b a] = butter(2, [0.6 0.7], 'bandpass');
filtered_noise = filter(b,a,randn(1, length(x)*2));

x = (x + 0.5*filtered_noise(500:500+length(x)-1))/length(x)*2;
```

Listing 5. Implementation of a noisy signal

As we can observe in the Figure 10, result of the execution of the commands above, is the representation of a sinusoidal wave form with a clearly presence of interference.

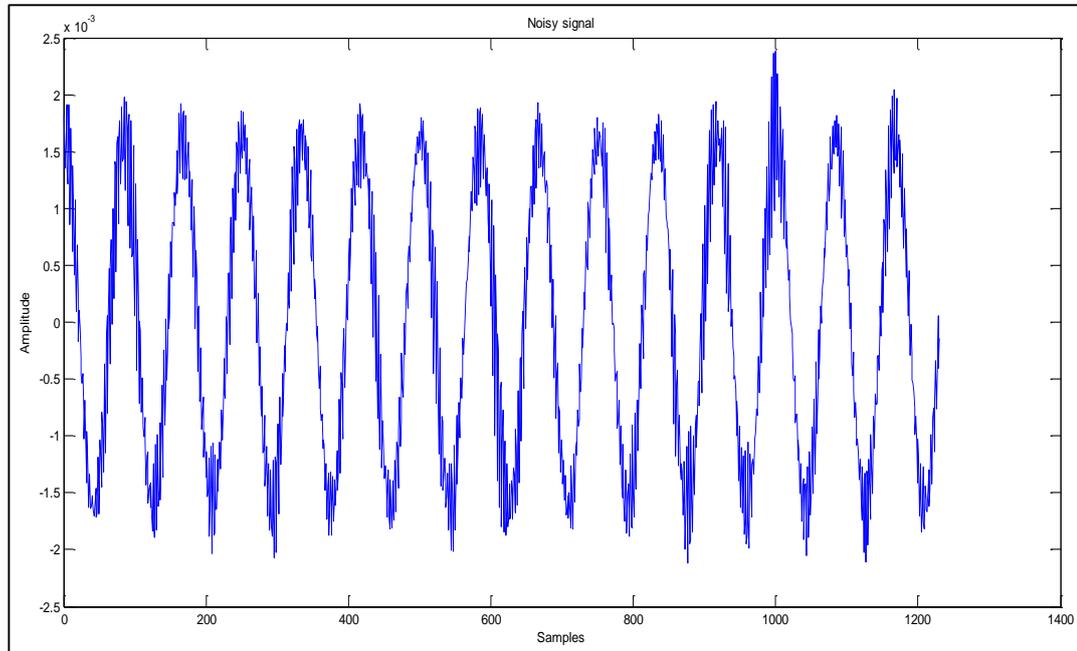


Figure 10. Noisy signal to be filtered

In order to have a better perspective on how the noise affects the signal in the figure 10 above, we could plot the magnitude spectrum of the signal considering its frequency domain.

```
X_mags = abs(fft(x));  
figure(10)  
plot(X_mags)  
xlabel('DFT Bins')  
ylabel('Magnitude')
```

Listing 6. Magnitude spectrum implementation of the noisy signal

As the result of applying the command lines above in listing 6, Matlab plots the graphic of the magnitude spectrum of the corrupted signal, in which it is possible to distinguish the sinusoidal signal and the noise in a separate range in the DFT scale. In the figure 11 below there is two larger peaks that presents the main signal and two different signals between 300 and 900 approximately that represents the noise.

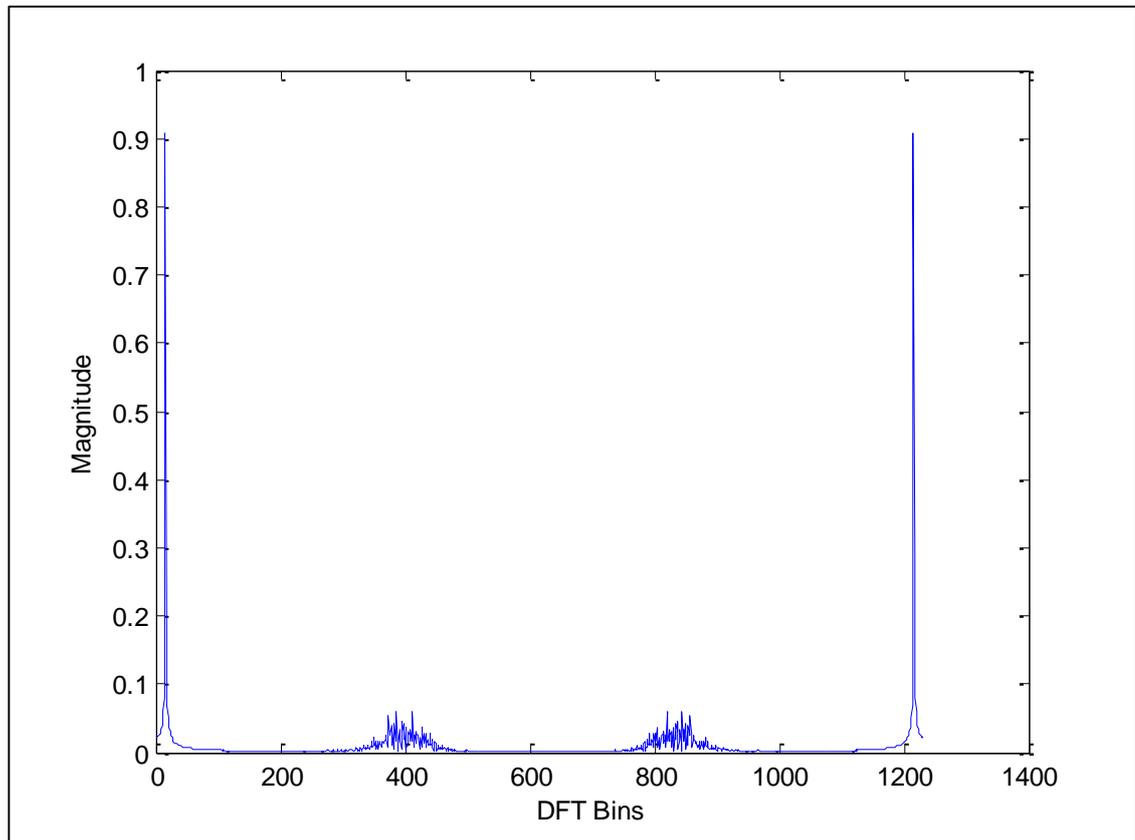


Figure 11. Magnitude spectrum of a noisy signal

The first peak from 0 to 0.1 in the figure 11 illustrates the sinusoidal part of the signal while the part between 300 to 500 (DFT Bins) represents the noise we want to cancel or attenuate. Next let's plot the first half of the DFT in which the normalised frequency axes is represented.

```
num_bins = length(X_mags);
plot([0:1/(num_bins/2 -1):1], X_mags(1:num_bins/2))
xlabel('Normalised frequency (\pi rads/sample)')
ylabel('Magnitude')
```

Listing 7. DFT first half plotting sentence

Applying the commands cited in the listing 7, Matlab plots the results as shown in figure 12 in which just an specific area of the DBT between zero to approximately 600 is taking in to consideration.

Figure 12 is a representation of the first peak and the noise in a closer form, so this way we can appreciate this two main parameters for the next steps on filtering.

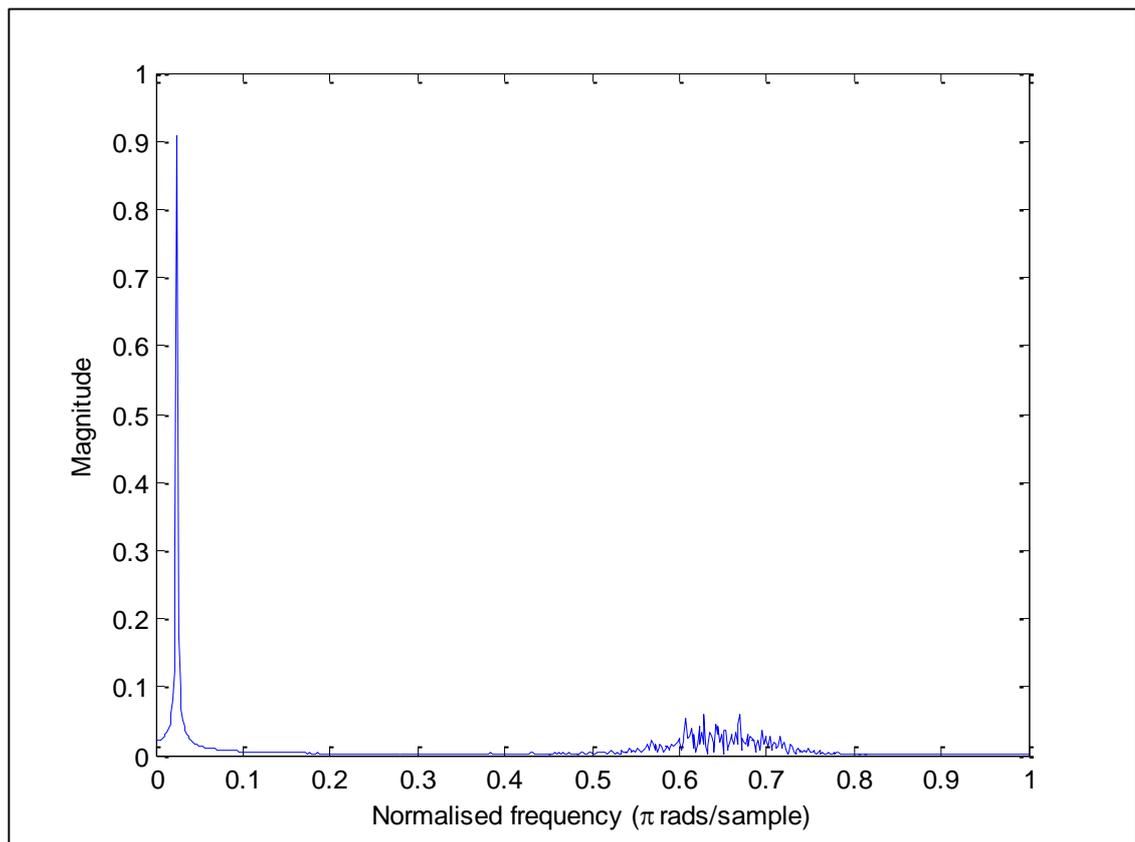


Figure 12. Normalized frequency plot. Signal and noise representation.

The units for normalized frequency in Matlab is given by π radians per sample (π rads/sample). Here the noise occupies the range between 0.5 to 0.8 units of the normalized frequency. The information these values gives us help us to design the filter that is ideal for reducing the unwanted noise on the signal.

While there are many different filtering techniques, here it is shown how to design a filter using the butterworth design. Firstly, we have to introduce the line:

```
[b a] = butter(2, 0.3, 'low');
```

Where the first value corresponds with the order of the filter, the second parameter is the cutoff frequency and the last indicates the type of filter to be implemented. As a result we receive back three coefficients for:

$$b = 0.1311 \quad 0.2622 \quad 0.1311;$$

$$a = 1.0000 \quad -0.7478 \quad 0.2722;$$

With these values it is possible to make the diagram of the filter following the common architecture of a IIR filter.

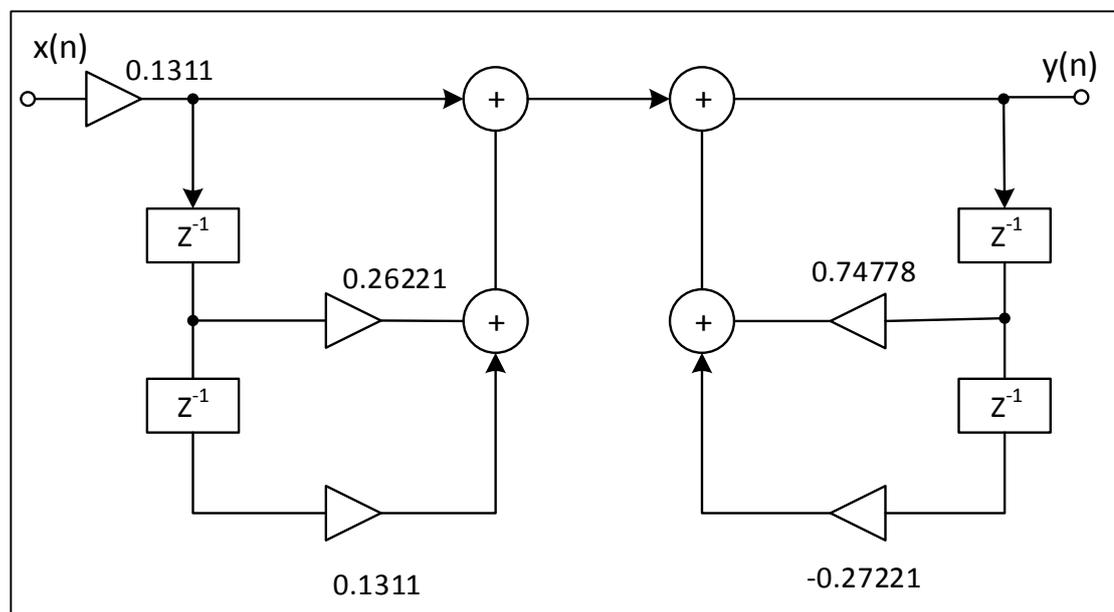


Figure 13. IIR Low pass filter diagram

The figure 13 above illustrates the recursive implementation of a low-pass filter with the respective coefficients of “b” and “a”.

In the magnitude spectrum graphic below, the red curve represents a second order low pass filter where a 0.3 cutoff value corresponds to ~ 0.7 in the magnitude axes which in turn is the correspondent to -3dB .

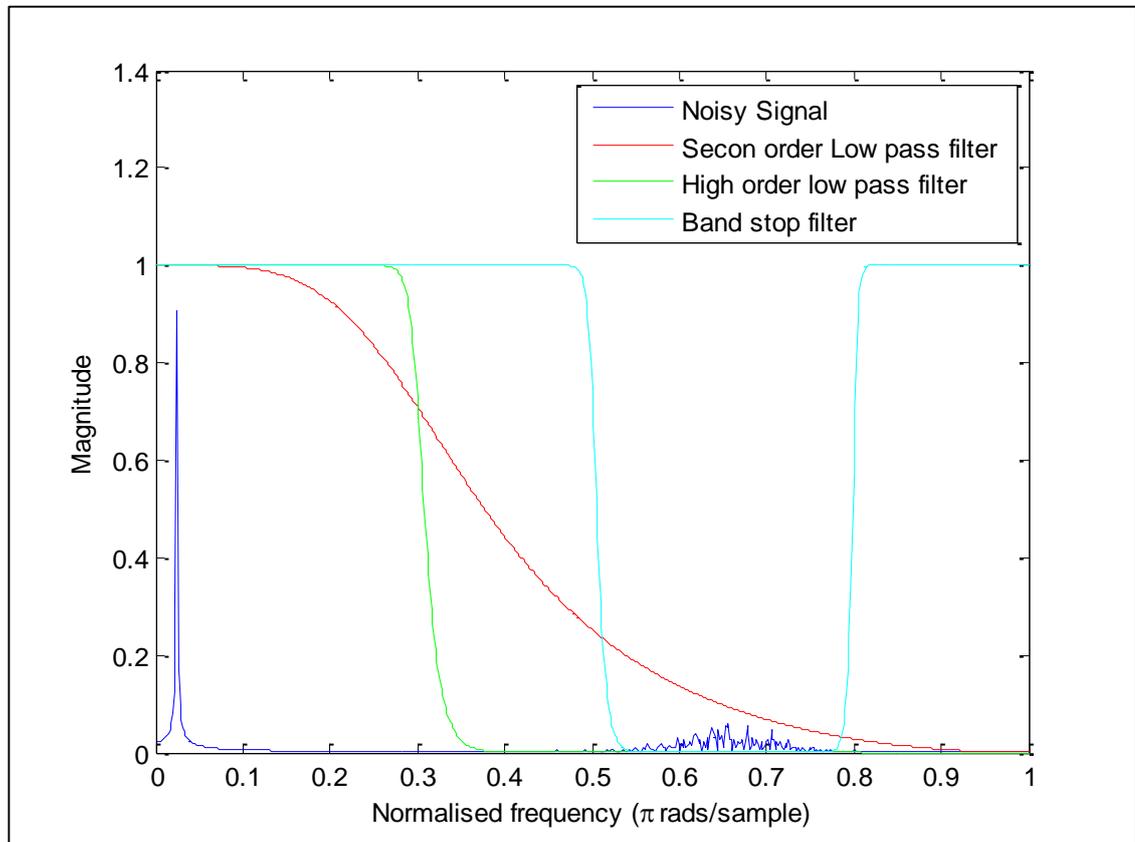


Figure 14. A comparison of three different filters.

As the figure 14 shows, this kind of filter leads only to a small decrease because the range of noise is within its admitted signal region.

The next example is also a low pass filter but with a higher order of 20. Here we can see that the frequency drop at the same cutoff frequency is sharper. It thus leaves the noise region outside of the admissible signal zone.

A band stop filter is designed to cancel the region between 0.5 and 0.8 so that we can denominate the rejected band region. In this case a small amount of noise is still present in the output. The reason for this is that there is a small amount of noise still present before and after the rejected band region that can not be totally eliminated.

A more accurate filter could be designed by giving even higher order value (e.g. 30). This would, however, involve a compromise in its efficiency as the processor needs more resources in order to make sampling calculations and/or to assign two different normalized values for frequencies.

The figure 15 below shows how the signal with noise looks like after the different steps of filtering mentioned above.

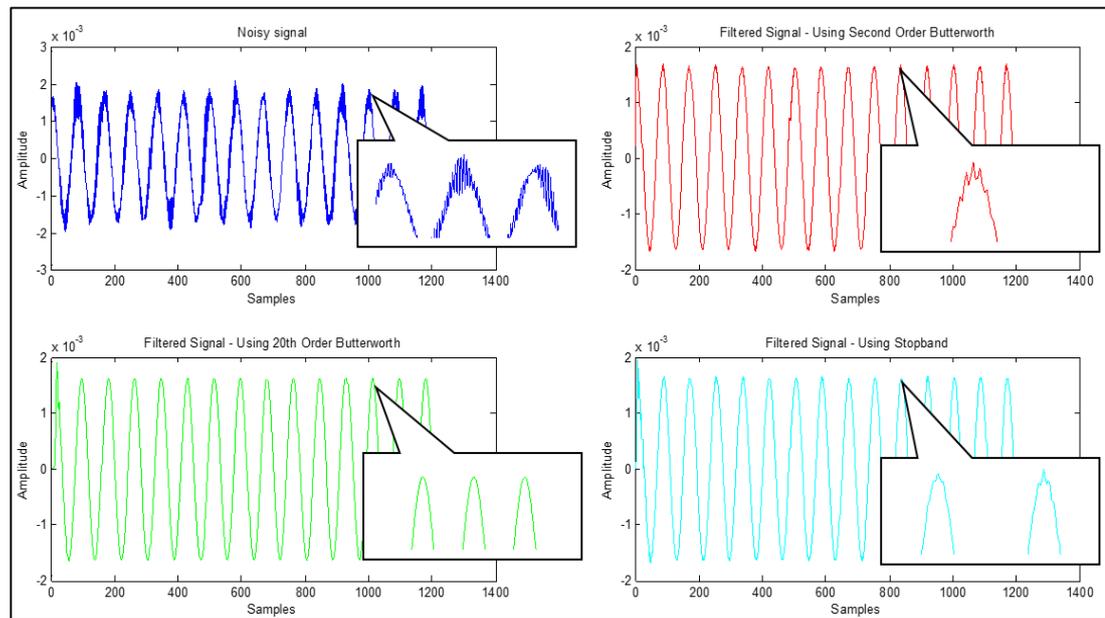


Figure 15. A comparison of the filtered signals using low pass and band stop filters

As mentioned earlier, the figure 15 demonstrates that the output signal resulting from a high order low pass filter is much cleaner in comparison with the second order one.

The other two filtering design techniques are called chebychev and elliptical. Their graphical representations for magnitude spectrum are shown in the figure below.

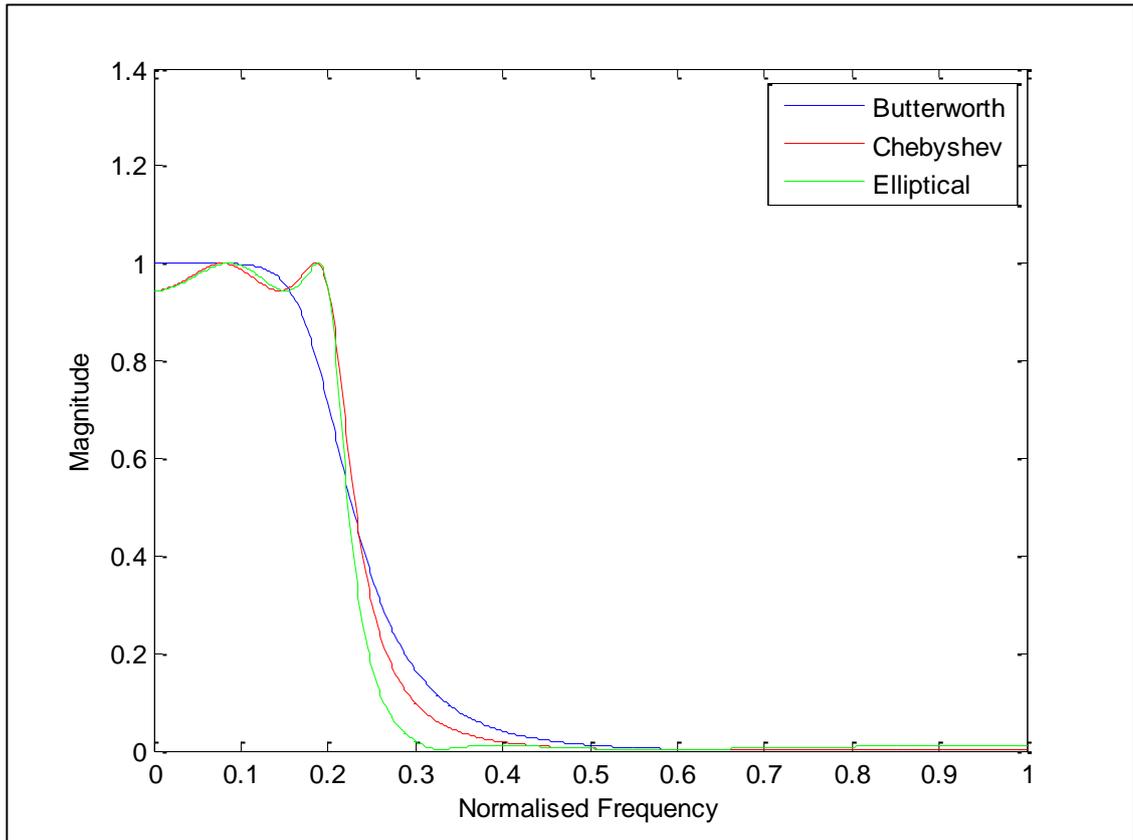


Figure 16. Comparison of three different filtering design techniques.

All the instructions for Matlab regarding the design techniques and the analysis before are included in the Appendix 2. [7.]

2.3.2 DSP Algorithm Implementation

In both recursive and non-recursive DSP, the algorithm implementation is based on DFT or differential equation. These algorithms would be used in two specific areas: in filtering algorithms and in signal analysis algorithms. Their implementation is possible by using hardware, firmware or software. This thesis focuses on the analysis of filtering algorithms implemented in software (Matlab).

In order to avoid certain problems in non-computable structures on a given transfer function, some matrix representations must be considered. As is the case mentioned by Mitra [4,600], if analysing a cascaded digital filter structure by giving the values directly in the order they appear, the result will not appear to be computable to the sys-

tem. Thus, it will be necessary to make some variations into the order in which the coefficients are implemented.

The figure 17 below represent the diagram of a cascaded digital filter structure modified from Mitra. It will help to understand how the computability condition works.

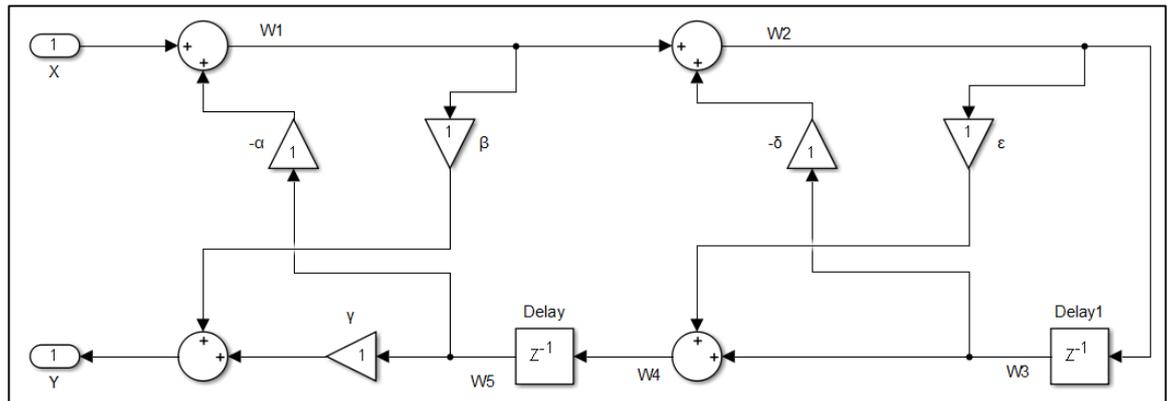


Figure 17. Cascaded filter structure

As the figure 17 above shows, there are certain values from W1 to W5 and corresponding equations are shown in the listing 8.

Eq. in frequency domain

$$\begin{aligned} W1(z) &= X(z) - \alpha W5(z), \\ W2(z) &= W1(z) - \delta W3(z), \\ W3(z) &= (z)^{-1} W2(z), \\ W4(z) &= W3(z) + \epsilon W2(z), \\ W5(z) &= (z)^{-1} W4(z), \\ Y(z) &= \beta W5(z) + \gamma W5(z), \end{aligned}$$

Eq. in Time Domain

$$\begin{aligned} w1[n] &= x[n] - \alpha w5[n], \\ w2[n] &= w1[n] - \delta w3[n], \\ w3[n] &= w2[n - 1], \\ w4[n] &= w3[n] + \epsilon w2[n], \\ w5[n] &= w4[n - 1], \\ y[n] &= \beta w5[n] + \gamma w5[n], \end{aligned}$$

Listing 8. Description of a non-computational algorithm.

$$\begin{bmatrix} w1[n] \\ w2[n] \\ w3[n] \\ w4[n] \\ w5[n] \\ y[n] \end{bmatrix} = \begin{bmatrix} x[n] \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix} + \begin{bmatrix} 0 & 0 & 0 & 0 & -\alpha & 0 \\ 1 & 0 & -\delta & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & \varepsilon & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ \beta & 0 & 0 & 0 & \gamma & 0 \end{bmatrix} \begin{bmatrix} w1[n] \\ w2[n] \\ w3[n] \\ w4[n] \\ w5[n] \\ y[n] \end{bmatrix} + \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} w1[n-1] \\ w2[n-1] \\ w3[n-1] \\ w4[n-1] \\ w5[n-1] \\ y[n-1] \end{bmatrix}$$

As can be seen from the example above, whether algorithm is computational or not, can be checked by taking into consideration the diagonal in the matrix. If the values of the diagonal – and also above the diagonal – are not zeros, the presence of delay-free loops makes the structure non computable.

Following this pattern, re-ordering results in a matrix representation, in which the diagonal, and the elements above the diagonal, are zeros :

$w3[n]; w5[n]; w1[n]; w2[n]; w4[n]; y[n]$.

$$\begin{bmatrix} w3[n] \\ w5[n] \\ w1[n] \\ w2[n] \\ w4[n] \\ y[n] \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ x[n] \\ 0 \\ 0 \\ 0 \end{bmatrix} + \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & -\alpha & 0 & 0 & 0 & 0 \\ -\delta & 0 & 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & \varepsilon & 0 & 0 \\ 0 & \gamma & \beta & 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} w1[n] \\ w2[n] \\ w3[n] \\ w4[n] \\ w5[n] \\ y[n] \end{bmatrix} + \begin{bmatrix} 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} w3[n-1] \\ w5[n-1] \\ w1[n-1] \\ w2[n-1] \\ w4[n-1] \\ y[n-1] \end{bmatrix}$$

Before the transfer function process, it is also important to check the structure in order to avoid errors.

This is possible through a filter analysis with the support of Simulink library and the signal processing toolbox. The instruction can be implemented in command window or by designing a piece of program in the command editor.

Through Mitra's [4,612] example, the impulse response samples implemented on Simulink will be displayed and the input and output figure sequence of the given transfer function using instructions on command window.

$$H(z) = \frac{0.0662272(1+3z^{-1}+3z^{-2}+z^{-3})}{1-0.93561423z^{-1}-0.10159107}; \quad (12)$$

For a filter implementation based on the Simulink technique, it is necessary to insert the values of the above equation to the corresponding places in the window as shown in the figure 18 below. After the process it is possible to choose between different plotting options and display extra information about the implemented filter.

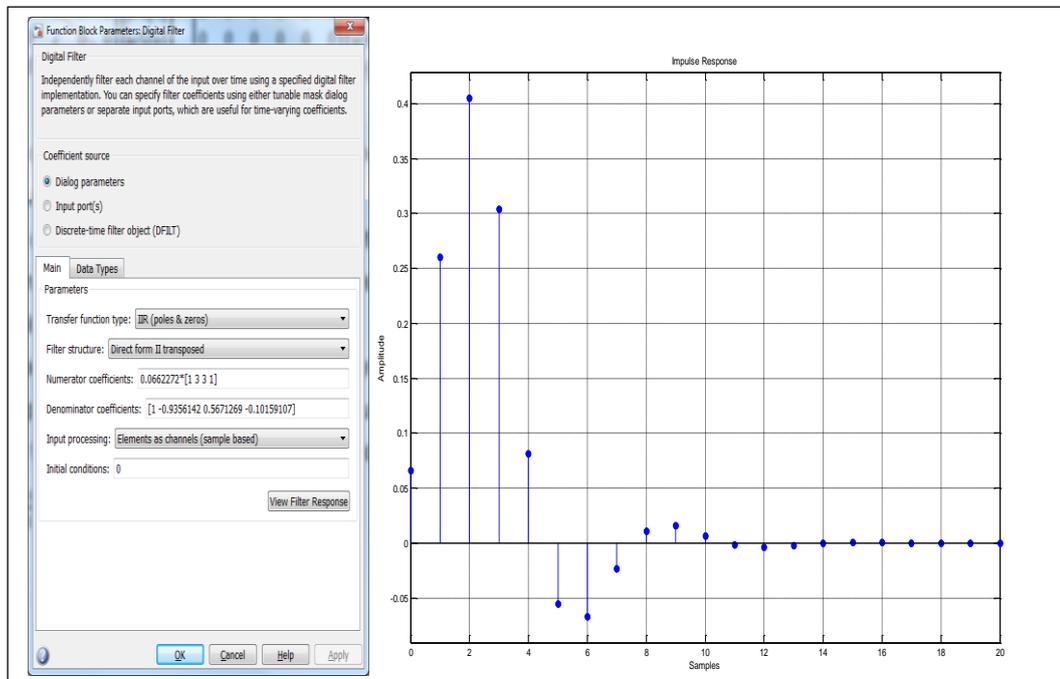


Figure 18. Filter implementation on Simulink and their Impulse response, Matlab plot.

The listing 10 below illustrates the structure of a lowpass IIR filter implemented on command line in which the first part corresponds to the random signal and the second part is the filtered signal.

```
% Illustration of Filtering by a Lowpass IIR Filter
% Generate the input sequence
k = 1:51;
w1 = 0.8*pi; w2 = 0.1*pi;
A = 1.5; B = 2.0;
x1 = A*cos(w1*(k-1)); x2 = B*cos(w2*(k-1));
x = x1+x2;
% Generate the output sequence by filtering the input
si = [0 0 0];
num = 0.0662272*[1 3 3 1];
```

```
den = [1 -0.9356142 0.5671269 -0.10159107];  
y = filter(num,den,x,si);  
% Plot the input and the output sequences  
subplot(2,1,1);  
stem(k-1,x); axis([0 50 -4 4]);  
xlabel('Time index n'); ylabel('Amplitude');  
title('Input sequence');  
subplot(2,1,2);  
stem(k-1,y); axis([0 50 -4 4]);  
xlabel('Time index n'); ylabel('Amplitude');  
title('Output sequence');
```

Listing 10. IIR Low pass filter

The behaviour of the low pass IIR filter implemented following the command line technique on the listing 10 gathered from Matlab help documentation, can be interpreted by observing the figure 19 below. Here the output signal is the result of filtering process of an “imperfect” input signal.

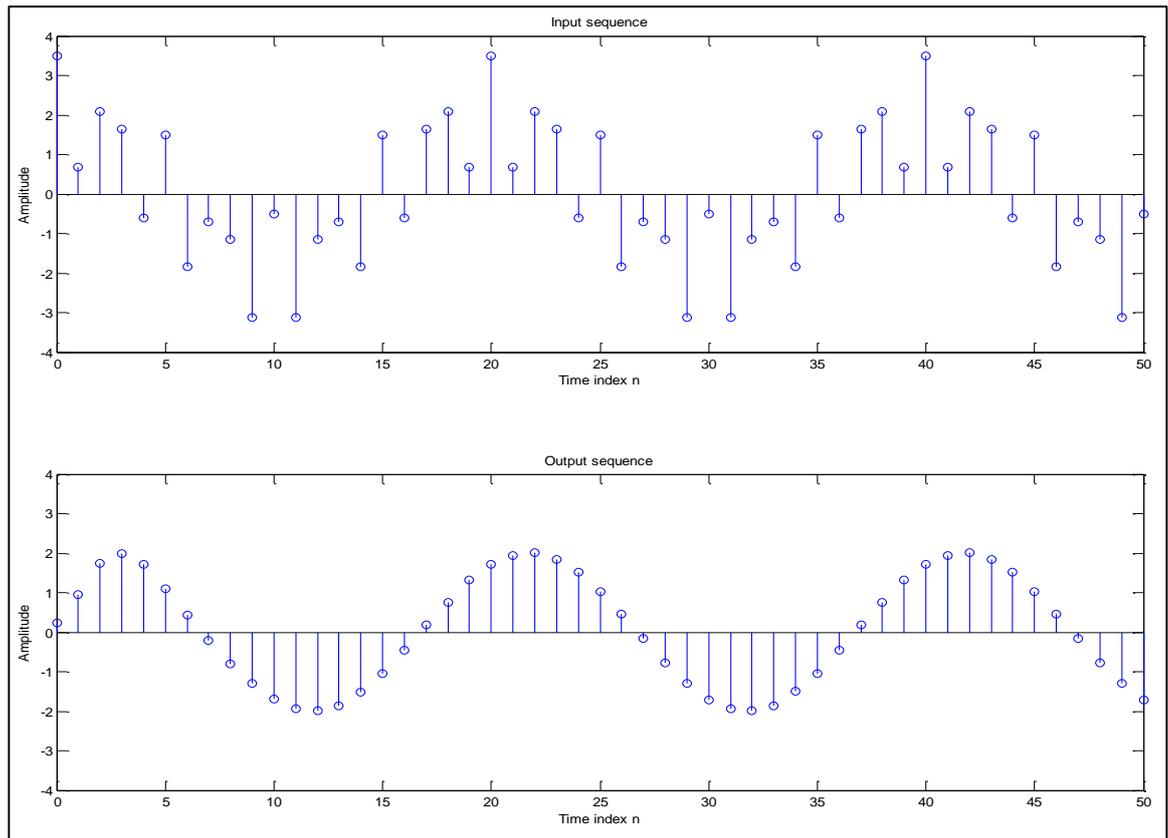


Figure 19. Input and output IIR filtering sequence, Matlab plot.

2.4 Audio Equalization (EQ)

Audio equalizers (EQ) are commonly used to adding, removing or modifying frequencies from specific audio signals. An equalizer is a system designed to change the characteristics of the tone of a given input audio signal on purpose by modifying the frequency response through an assortment of filters until the desired frequency spectrum is achieved. Two most important applications are corrective and enhancive EQ's.

Corrective EQ is commonly used for restoring sound, improving frequency response in halls or closed rooms, compensating bad recording and so on. Enhancive EQ is used for making the sound different from the original by making some “effects” or just improving the sound quality according to the situational conditions.

In technical terms, a “curve” is an element capable of assigning gain to a certain frequency ranges in both directions: referring as a boost when the gain is in positive direction or as a cut or attenuation when the gain is in the negative direction. This thesis

considers the study of two types of curves that both determine the type of equalizer: shelving-curve and bell-curve that will be presented below.

In the figure 20 below, a shelving-curve which attenuates or increases frequencies below or above a specific frequency can be observed.

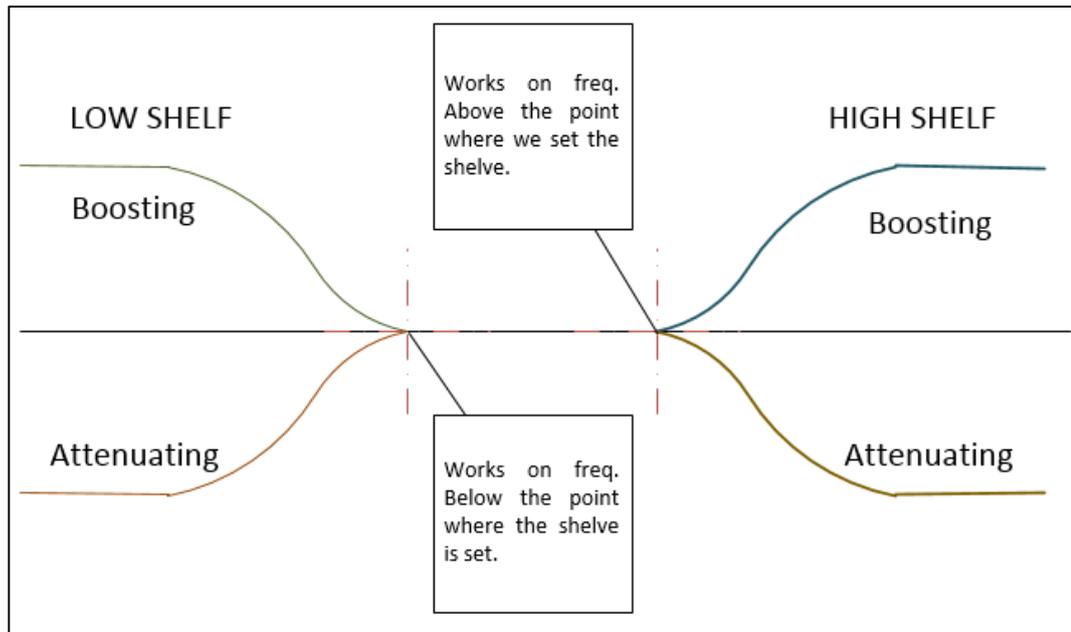


Figure 20. Low and High Shelving Curve

The figure 20 shows the different possibilities of defining a shelving curve. The curves on the left hand side of the figure 20 boost or attenuate the frequencies below a given point in the frequency-range and are thus referred as Low Shelf curve. On the contrary, the High Shelf curve boosts or attenuates frequencies above the given reference point and can be observed on the right hand side of the figure 20.

The second type of curve that also determines the type of equalizer is called Bell curve. Characteristic to this kind of curve is that it will boost or cut a frequency range around a pointed center frequency as illustrate the figure 21, where the center frequency has the most effect. It is called Bell curve because it decreases towards the outer frequencies and thus gives the EQ curve a form of a bell.

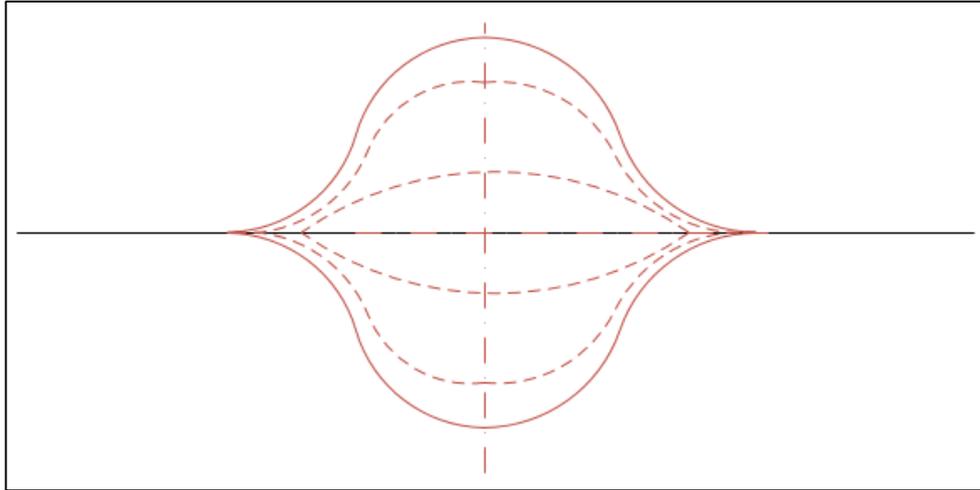


Figure 21. Representation of a Bell curve

2.4.1 Types of Equalizers

The Q represents the bandwidth of the frequency-band. This is the relation between the center frequency (cf) and the bandwidth (bw), and some manufacturers also call it “the Quality”. The Q parameter is present in every EQ types but it could be equal to one, in this case the Q parameter does not taking in to consideration.

$$Q = \frac{cf}{bw}. \quad (13)$$

A low value of Q means a wide bandwidth and a high value of Q indicates the presence of a narrow bandwidth.

The Gain parameter (boost/cut) amplifies or filters the selected frequencies in the band. The gain can cut or attenuate frequencies to -24dB and/or amplify or boost frequencies up to +24dB. Digital Equalizers offer higher boost/cut parameters.

Here is a first a brief description of different types of equalizers based on their curves and later a deeper analysis on the graphic equalizer.

- Fixed Frequency EQ: consists on a set of filters with only one control on the gain parameter. This can be either shelving or bell.

- Sweepable EQ: This can be also shelving or bell with controls of two parameters, the frequency and the gain.
- Parametric EQ: This is one of the more flexible equalizers. It has three control parameters that allows it to 1) sweep some specific frequencies on the audio band, 2) control the gain of the audio on a range of frequencies; and 3) control the bandwidth (Q). The frequency parameter sets the frequency for the EQ-band often between 20 – 20000 Hz, at a center point of a required bell-curve and/or at a start point frequency of the shelve curve.
- Graphical EQ: This is an audio control that allows the user to change the gain of the specific selected frequency bands based on the visual corresponding frequency diagram representation.

There are three types of filters that are more used included on the section of parametric equalizers: Bell filter, shelf filter and pass filter. The Bell filter consist of a bandpass filter with variable parameters on center frequency (20 – 20000 Hz), gain (± 15 dB) and Q (0.5 – 3). Shelf filters include for example treble and bass controls, and like was mentioned earlier in section 2.4, with the help of this filter it is possible to control the gain and cutoff frequency over a range of frequencies (high or low). Pass filters on the other hand are used for noise reduction: they are basically low or high pass filters with fixed cutoff frequencies. [8, 734.]

2.4.2 Graphic Equalization

The graphic equalizers are mainly formed of a series of bell shaped equalization bands fixed in frequency. This kind of EQ is controlled with a fader for boost or cut the fixed frequencies.

The fixed frequency selection of the graphic EQ is often regulated by a standard (SI-standard) for example either 8, 10, 15 or 31 bands.

Table 2. Possible 10 – Bands Graphic Equalizer frequencies

10 – Bands Graphical Equalizer frequencies in Hertz:									
31	63	125	250	500	1000	2000	4000	8000	16000

The ten bands equalizer mentioned on table 2 above will be considered for the implementation of an equalizer with less bands. As table 2 shows there are ten possible bands with different center frequencies. For the implementation of a 5-bands EQ, it is possible to consider the values of frequencies between the center frequencies of the mentioned 10-bands. Another variation could be taking the first and last frequencies for shelving filter controls (bass and treble controls) and 8-bands for bell filtering.

The figure 22 below illustrates the common structure of filtering with a set of five bell filters controlling random fixed frequency bands.

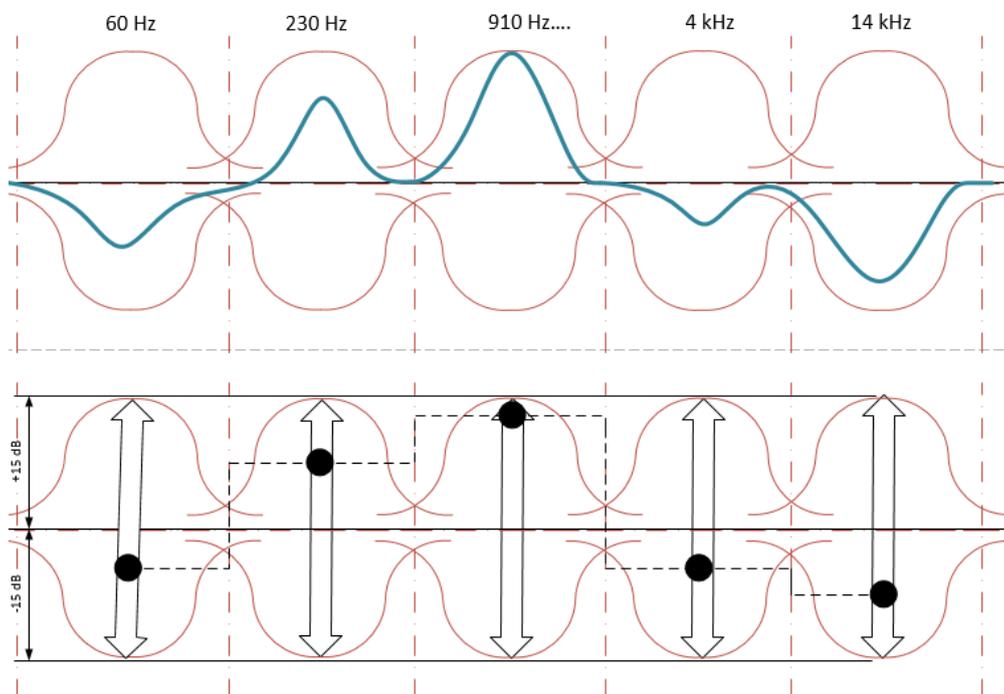


Figure 22. 5-Bands Graphic EQ representation.

2.4.3 EQ Analogue vs. EQ Digital

In the analogue frame the characteristics of frequency, gain and Q-factor are controlled interactively by the user with the help of variable resistors. Analogue EQ's are divided into passive and active ones based on their component types. The Passive equalizers use passive electronic components such as capacitors and inductors, and basically works by attenuating all the frequencies by a certain amount, except for those frequencies needing to be boosted which are less attenuated. The boost process done by a

second amplification step that adds gain evenly to the signal. The selected frequency is now boosted because it was not evenly attenuated. This kind of equalizers produce a pleasant sound with very low noise levels. The drawback is, that it is heavy and often expensive.

Active equalizers use active electronic components and the boosting process is done by a feedback loop: sending the selected frequency back to the amplifier and creating phasing. Active EQ is relatively low cost, lighter in weight but still creates an acceptable levels of noise.

The digital Equalizer parameters are controlled by the modification of the characteristics of the digital filters coefficients. They present high precision and have the capability of achieving more boost and cut. In other words, they generate less noise and phase-shift than the analogue ones.

According to Ifeachor and Jervis [8, 735], the process of equalization can be done by using basic filters described earlier. Generally their use and take into account the theory concerning Butterworth and Chebyshev filters. Their variations in the s-plane transfer function are described below:

$$H(s) = \frac{s^2 + As + \omega_n^2}{s^2 + Bs + \omega_n^2}; \quad (14)$$

$$H(s) = \frac{s^2 + 2As + A^2}{s^2 + 2Bs + B^2}; \quad (15)$$

$$H(s) = \frac{A^2 s^2 + 2A\omega_n^2 s + \omega_n^4}{B^2 s^2 + 2B\omega_n^2 s + \omega_n^4}; \quad (16)$$

Where

$$A = \frac{1}{(R+r)C} \left(3 - \frac{1}{K} \right);$$

$$B = \frac{1}{(R+r)C} \left(3 + \frac{1}{K} \right);$$

In the above equations the following parameters are described: R and C correspond to constant values, r is the frequency shifting, and K is the boost/cut parameter.

In order to implement the digital equivalent equalization, the s-planes transform function must be changed to the corresponding z-transfer function.

$$H(z) = \frac{az^2+bz+c}{dz^2+ez+f}; \quad (16)$$

Where

$$a = P^2 + AP + \omega_n^2;$$

$$b = 2\omega_n^2 - 2P^2;$$

$$c = P^2 - AP + \omega_n^2;$$

$$d = P^2 + BP + \omega_n^2;$$

$$e = 2\omega_n^2 - 2P^2;$$

$$f = P^2 - BP + \omega_n^2;$$

$$P = \frac{\omega_n}{\omega_p}; \quad \omega_p = \frac{2}{T} \tan\left(\frac{\omega_n T}{2}\right);$$

3 Audio Equalization Using Matlab

The implementation of the EQ for this thesis is based on the idea to create a tool to support the theory of DSP. Thus, simplicity is a primary factor. Another important issue to take into consideration is that the EQ is focused on educational purposes, therefore some modifications are done to the behaviour that a "normal" EQ has.

The number of bands selected for the EQ, is based on some of the most used audio applications found on the common smart telephones and/or tablets on the market. These common audio applications use a 5-band EQ, with range of frequencies from 60

– 14000 Hz and gain control of ± 15 dB. These will be taken as a reference for the design and implementation of the EQ.

The interface of the EQ was designed with the GUI Matlab tool in order to get a friendly and more comprehensive interaction with the EQ controls, as well as trying to introduce a user friendly way to make changes in the filter parameters. The EQ window shows five controllers which enable to set the EQ filter values and through them to modify the characteristics of the sound. The changes in the quality will be appreciated and clearly heard when playing the modified sound through the speakers.

The values of the analogue transfer function $H(s)$ and its corresponding z-domain transfer function $H(z)$ are based on the Chebyshev filter specifications. The Chebyshev II is recommended for use due to the practical characteristics it has in terms getting the correspondent analogue transfer functions $H(s)$ [8, 395.] and primordially because the filter is without a presence of passband ripple.

The Chebyshev II filter in Matlab generates an IIR digital filter with extra features depending of the arguments given on the syntax as cited below. The Matlab documentation describes in detail the use of this filter:

```
hd = design(d, 'cheby2')
hd = design(d, 'cheby2', designoption, value, designoption, value, ...)
```

The syntax used for the equalizer needs the specification of the behaviour of the curve given by the correct filtering process. Therefore the command used is:

```
[b, a] = cheby2(n, R, Wst, 'ftype')
```

The n is the order of the filter, R is the stopband ripple and Wst or Wn is the normalized cutoff frequency. The 'ftype' parameters used for the EQ are: 1) One lowpass filter ('low') for the first step called C1, creating a low-shelf curve in the beginning of the spectrum analysis. 2) Three band pass filters ('bandpass') in C2, C3, and C4 with bell-curve in the middle and 3) One high pass filter ('high') with a high-shelf curve at the end for C5 .

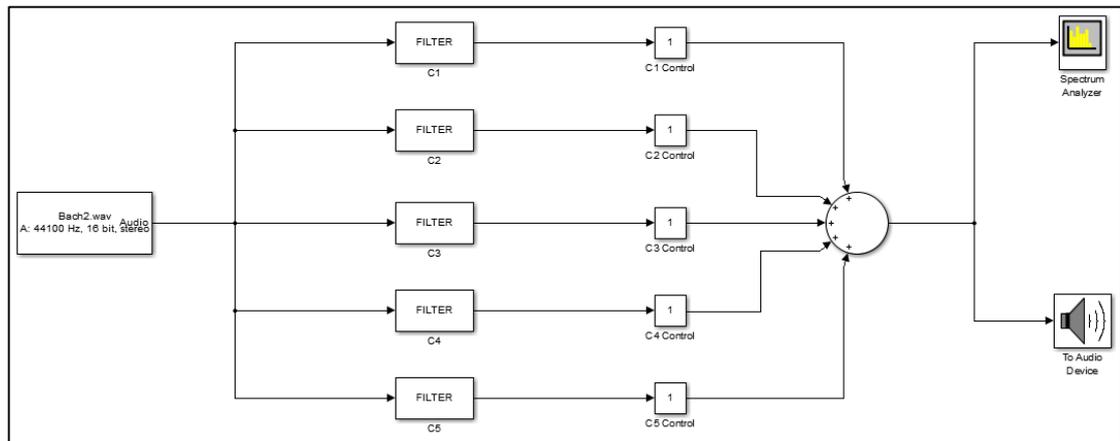


Figure 23. EQ-ator block diagram implementation using Simulink

The figure 23 above shows the block diagram for the 5-bands EQ, called EQ-ator. The left side illustrates the first block representing the input audio signal from a *.wav sound file. This is the process of loading the audio data located on the local disc or media files. The parameters used for this step are: $F_s = 44100$ Hz, and 1024 samples per audio channel. This audio signal goes to each one of the five different filters in the next five blocks.

The blocks C_n are the filters C1-C5. They have a frequency range based on the example of the figure 22, and the parameters described in the table 3 below.

Table 3. Reference frequencies and filter type for the 5-Band EQ

C_n	Freq. range [Hz]	Filter type
C1	60	Chevy2(n,R,W_n , 'low')
C2	230	Chevy2(n,R,W_n , 'bandpass')
C3	910	Chevy2(n,R,W_n , 'bandpass')
C4	4000	Chevy2(n,R,W_n , 'bandpass')
C5	14000	Chevy2(n,R,W_n , 'high')

The gain control for boosting or attenuation of the frequency ranges are described as "Cn controls". This scalar value of gain is controlled through a slider (fader) on the screen during a simulation. Another way to set the values for Cn controls is writing the desired value between -15 to +15 on the correspond space below the slider.

The audio signals of each of the filters mentioned come together in the next step of the block diagram with the sum parameter. The result signal is send through the last process step to the computer's audio device.

It is possible to see the curve representing the position of sliders in a spectrum analyser which displays the fixed number of the frequency in Hz over magnitude in dB of the respective channels C1-C5.

After executing the EQator.m file on Matlab, the window showed in the figure 24 appears on screen.

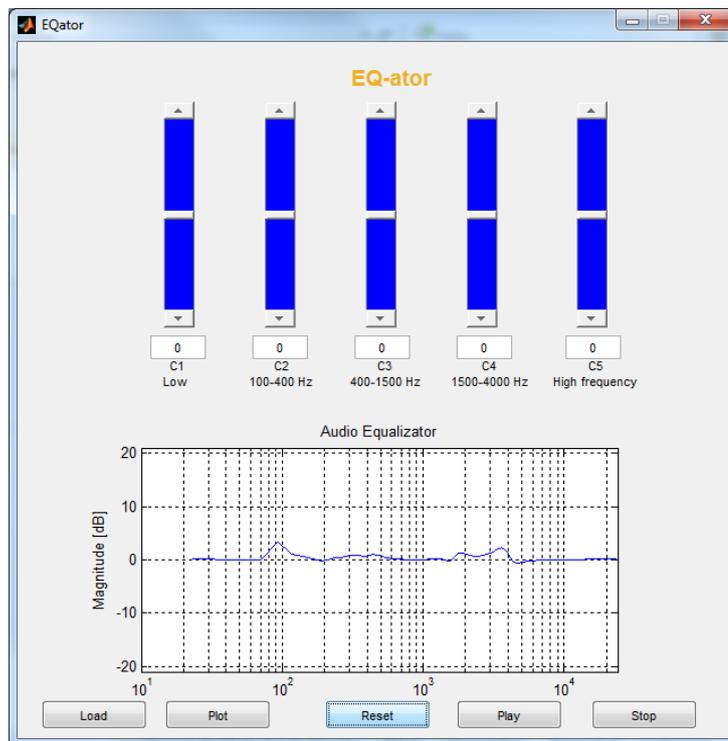


Figure 24. EQ-ator equalizer implemented on Matlab

The above figure 24 illustrates a 5-bands equalizer window implemented on GUI Matlab called EQ-ator. The five sliders control the respective range of frequencies described below each slider. After the sliders are set to their chosen positions, the graphical representation is executed by clicking the “plot” button. The audio file is loaded after clicking on the “load” button, and window of the figure 25 below appears on screen.

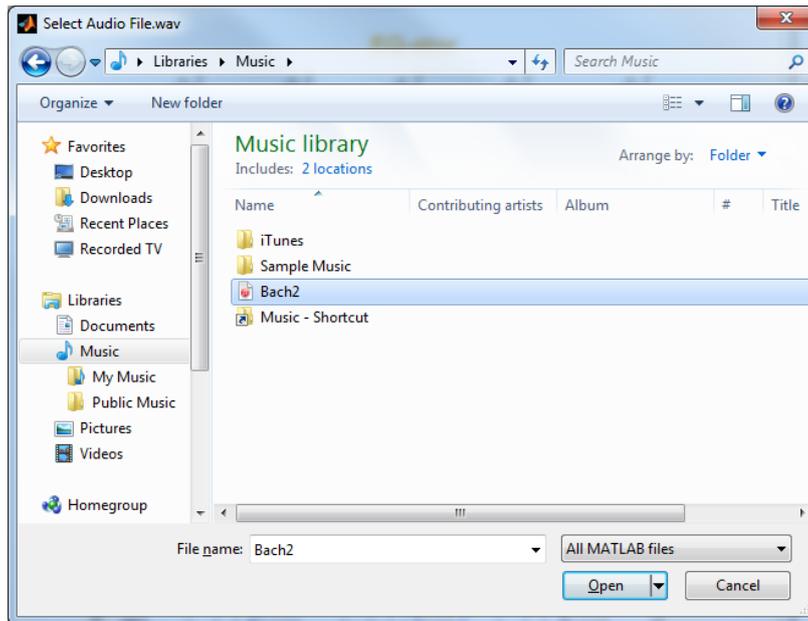


Figure 25. Audio file loading window for the EQ-ator

The permitted format for the audio files is *.wav. This is because the Matlab uses “wavread” audio input command. It is recommended to change to “audioread” for future references.

4 Results

The goal of this thesis, that is to analyse different filters and apply them to an audio equalizer implemented on Matlab, to understand how DSP theory works on a practical way, which is achieved at the end of this project.

The equalizer called EQ-ator brings the option to load a .wav audio file and listens this audio for a few seconds. This short reproduction of the audio file characteristic comes from the need to, first, avoid an excessive noise at the laboratory for a longer time reproduction when dealing with this application on the DSP class, and secondly because there is a defect on the implementation on Matlab commands, that only allows to play the audio sound without distortion caused by a possible overlapping, by using the command “`player.playblocking`”, and this command blocks all the functions of the equalizer (including the stop button), until the entire song was completed. This “defect”, on the other hand, suits well with the purpose of the equalizer.

The figure 26 below shows the behaviour of the EQ working with the gain parameters switched at ± 14.25 with the purpose of appreciation of how the sliders works at different positions and its respective curve at the spectrum analyser.

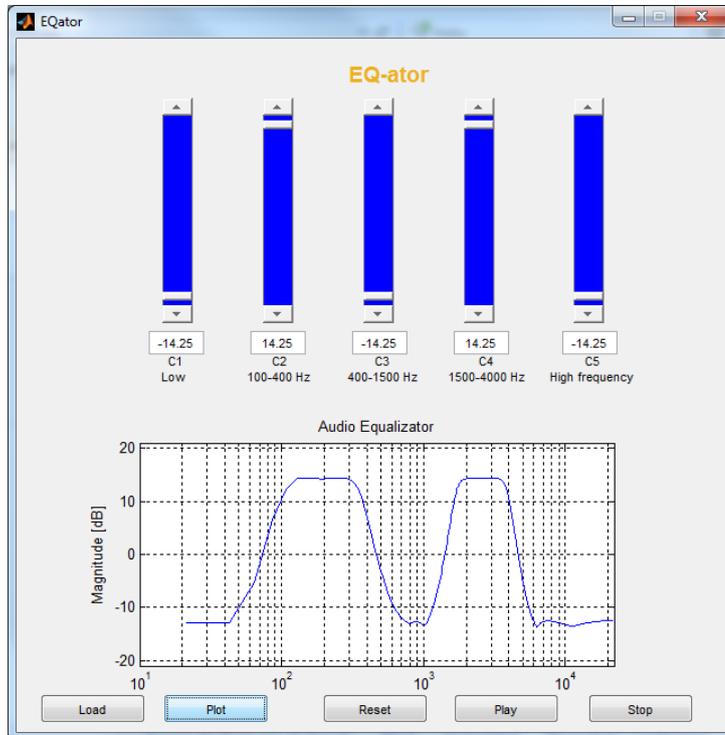


Figure 26. EQ parameters alternating at ± 14.25 dB

The following text shows up after executing the EQator.m file on the command window:

```
>> EQator
file name:
Bach2.wav
Warning: WAVREAD will be removed in a future release. Use AUDIOREAD
instead.
> In wavread at 62
   In EQador>equalizer_play at 516
   In EQador>play_Callback at 510
   In gui_mainfcn at 95
   In EQador at 42
   In
   @(hObject,eventdata)EQator('play_Callback',hObject,eventdata, GUIDATA(h
Object))
processing
playing finished
```

5 Conclusions and Recommendations

The first challenge faced in EQ implementation is to find suitable filters that will work well together in the required frequency ranges. The choice of correct filters and the ranges in which these filters give the desired results was achieved by trying each of them during the implementation process, as well as making variations on the ranges of frequency in the command editor until the curve seemed regular in the spectrum analyzer.

One particular feature on this EQ is found in the controller C5: the characteristics of the high pass filter was difficult to fit into the correct range. Due to this problem the curve of the filter invaded the previous frequency range. Nevertheless, the results are between the desired parameters as the curve in the figure 27 below illustrates.

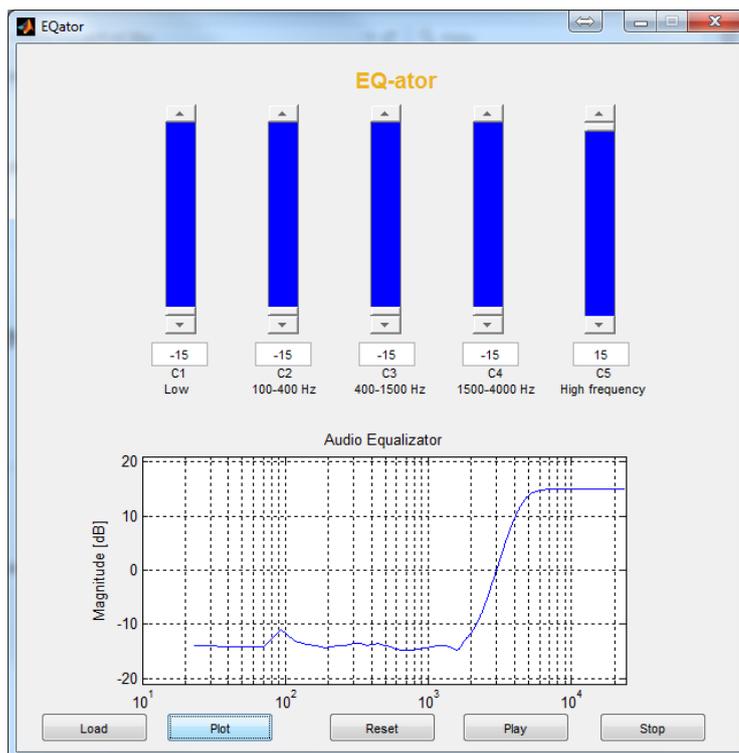


Figure 27. Behaviour of the C5 filter

In the future, it is good to know that there are some websites giving examples regarding the equalizer implementation using Matlab, but unfortunately they do not always work unless several changes are made. But, regardless of these issues, the examples can be used as practical reference points in understanding how to start with the different tools that Matlab brings in concerning the DSP area.

References

- 1 Karrenberg Ulrich. An Interactive Multimedia Introduction to Signal Processing. Second Edition. Düsseldorf, German: Springer; 2006.
- 2 Sinha Priyabrata. Speech Processing in Embedded Systems. USA: Springer; 2010.
- 3 Stearm Samuel D. Digital Signal Processing with Examples in Matlab. USA, Florida: CRC Press; 2003.
- 4 Mitra Sanjit K. Digital Signal Processing: a Computer-Based Approach; Fourth Edition. USA, New York: McGraw-Hill; 2011.
- 5 Markovic Dejan and Brodersen Robert. DSP Architecture Design Essentials. USA, New York: Springer; 2012.
- 6 Hussain Zahir, Sadik Amin, O'Shea Peter. Digital Signal Processing: An Introduction with Matlab and Applications. Berlin Heidelberg: Springer-Verlag; 2011.
- 7 David Dorran [online]. October 2013. URL: <https://dadorran.wordpress.com>. Accessed February 2015.
- 8 Ifeachor Emmanuel, Jervis Barrie. Digital Signal Processing: A Practical Approach. United States: Addison-Wesley, 1993.

Fourier and DFT on Matlab, gathered from Engelber's (2008) [7, 48]

```
>> t=[-1000:1000]/10;
>> y=exp(-abs(t));
>> subplot(2,2,1)
>> plot(t,y,'k')
>> title('a) The Signal')
>> z=fft(y);
>> f=[0:2000]/(2001/10);
>> subplot(2,2,2)
>> plot(f,(200.1/2001)*abs(z),'--k')
>> title('b) The DFT')
>> subplot(2,2,3)
>> plot(f,2./((2*pi*f).^2+1),'-k')
>> title('The Fourier Transform')
>> title('c) The Fourier Transform')
>> subplot(2,2,4)
>> plot(f,(200.1/2001)*abs(z),'--k',f,2./((2*pi*f).^2+1),'-k')
>> title('d) Fourier Transform vs. DDT')
>>print -djpeg comp.jpg
```

Table 1. Example of Array Operations on Matlab

Operation	Symbol	Example	Remarks
Sum or difference	+ or -	$\begin{bmatrix} 1 & 3 \\ 3 & 4 \end{bmatrix} + \begin{bmatrix} 0 & -2 \\ 1 & 3 \end{bmatrix} = \begin{bmatrix} 1 & 1 \\ 4 & 7 \end{bmatrix}$ $[1 \ 2 \ 3] - 2 = [-1 \ 0 \ 1]$	Dimensions must be the same
Matrix product	*	$\begin{bmatrix} 1 & 2 & 0 \\ 3 & 4 & 2 \end{bmatrix} * \begin{bmatrix} 0 & -1 \\ 1 & 2 \\ -1 & 1 \end{bmatrix} = \begin{bmatrix} 2 & 3 \\ 2 & 7 \end{bmatrix}$ $3 * [1 \ 2 \ 3] = [3 \ 6 \ 9]$	Inner Dimensions must be the same
Array Product	.*	$\begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix} .* \begin{bmatrix} 0 & -1 \\ 3 & 2 \end{bmatrix} = \begin{bmatrix} 0 & -2 \\ 9 & 8 \end{bmatrix}$ $3 .* [1 \ 2 \ 3] = [3 \ 6 \ 9]$	Dimensions must be same
Transposes	'	$\begin{bmatrix} 2 & 3 & 4 \\ 5 & 6 & 7 \end{bmatrix}' = \begin{bmatrix} 2 & 5 \\ 3 & 6 \\ 4 & 7 \end{bmatrix};$ $[1 + 2j \ 3 - 4j]' = \begin{bmatrix} 1 - 2j \\ 3 + 4j \end{bmatrix}$	j=(-1) ^{1/2} . Rows become columns.
	.'	$\begin{bmatrix} 2 & 3 & 4 \\ 5 & 6 & 7 \end{bmatrix}.' = \begin{bmatrix} 2 & 5 \\ 3 & 6 \\ 4 & 7 \end{bmatrix};$ $[1 + 2j \ 3 - 4j].' = \begin{bmatrix} 1 + 2j \\ 3 - 4j \end{bmatrix}$	
Exponentiation	.^	$2.^[1 \ 2 \ 3] = [2 \ 4 \ 8]$ $[1 \ 2 \ 3].^2 = [1 \ 4 \ 9]$ $\begin{bmatrix} 0 & 1 & 2 \\ 3 & 4 & 5 \end{bmatrix} .* \begin{bmatrix} 0 & 1 & 2 \\ 2 & 1 & 0 \end{bmatrix} = \begin{bmatrix} 0 & 1 & 4 \\ 9 & 4 & 1 \end{bmatrix}$	Dimensions must be the same
Backlash	\	$a * b = c, \text{ where } a = \begin{bmatrix} 5 & 3 \\ 2 & 2 \end{bmatrix} \text{ and } c = \begin{bmatrix} 1 \\ 2 \end{bmatrix}$ $b = a \setminus c = \begin{bmatrix} 5 & 3 \\ 2 & 2 \end{bmatrix} \setminus \begin{bmatrix} 1 \\ 2 \end{bmatrix} = \begin{bmatrix} -1 \\ 2 \end{bmatrix}$	Solves sets of linear equations

Table modified from Stearn (2003) [3, 6]

Example of filtering on Matlab

```
%% Example of filtering on Matlab
%generate the signal with noise
x= cos(2*pi*12*[0:0.001:1.23]);
x(end) = [];
[b a] = butter(2,[0.6 0.7], 'bandpass');
filtered_noise = filter(b,a,randn(1, length(x)*2));
x = (x + 0.5*filtered_noise(500:500+length(x)-1))/length(x)*2;

% Plotting the noise signal
subplot (2,2,1)
plot(x)
title('Noisy signal')
xlabel('Samples');
ylabel('Amplitude')

%Implement a second order filter (butterworth design technique)
[b a] = butter(2, 0.3, 'low')

%filter the signal
x_filtered = filter(b,a,x);

%plot the filtered signal
subplot (2,2,2)
plot(x_filtered,'r')
title('Filtered Signal - Using Second Order Butterworth')
xlabel('Samples');
ylabel('Amplitude')

%higher order filter
[b2 a2] = butter(20, 0.3, 'low')

%filter and plot the result
x_filtered2 = filter(b2,a2,x);
subplot(2,2,3)
plot(x_filtered2,'g')
title('Filtered Signal - Using 20th Order Butterworth')
xlabel('Samples');
ylabel('Amplitude')

%band stop filter
[b_stop a_stop] = butter(20, [0.5 0.8], 'stop');

%plot filtered signal
x_filtered_stop = filter(b_stop,a_stop,x);
subplot(2,2,4)
plot(x_filtered_stop,'c')
title('Filtered Signal - Using Stopband')
xlabel('Samples');
ylabel('Amplitude')

%plot magnitude spectrum of the signal
X_mags = abs(fft(x));
plot(X_mags)
xlabel('DFT Bins')
```

```

ylabel('Magnitude')

%plot first half of DFT (normalised frequency)
num_bins = length(X_mags);
plot([0:1/(num_bins/2 -1):1], X_mags(1:num_bins/2))
xlabel('Normalised frequency (\pi rads/sample)')
ylabel('Magnitude')

%plot the frequency response (normalised frequency)
H = freqz(b,a, floor(num_bins/2));
hold on
plot([0:1/(num_bins/2 -1):1], abs(H), 'r');

%Plot the magnitude spectrum and compare with lower order filter
H2 = freqz(b2,a2, floor(num_bins/2));
figure(10)
hold on
plot([0:1/(num_bins/2 -1):1], abs(H2), 'g');

%plot the band stop filter magnitude spectrum
H_stopband = freqz(b_stop,a_stop, floor(num_bins/2));
figure(10)
hold on
plot([0:1/(num_bins/2 -1):1], abs(H_stopband), 'c');

% comparison with other filter design techniques (chebyshev and elliptical)
[b_butter a_butter] = butter(4, 0.2, 'low');
H_butter = freqz(b_butter, a_butter);

[b_cheby a_cheby] = cheby1(4, 0.5, 0.2, 'low');
H_cheby = freqz(b_cheby, a_cheby);

[b_ellip a_ellip] = ellip(4, 0.5, 40, 0.2, 'low');
H_ellip = freqz(b_ellip, a_ellip);

%plot each filter to compare
figure(11)
norm_freq_axis = [0:1/(512 -1):1];
plot(norm_freq_axis, abs(H_butter))
hold on
plot(norm_freq_axis, abs(H_cheby), 'r')
plot(norm_freq_axis, abs(H_ellip), 'g')
legend('Butterworth', 'Chebyshev', 'Elliptical')
xlabel('Normalised Frequency');
ylabel('Magnitude')

%plot in dB
figure(12);
plot(norm_freq_axis, 20*log10(abs(H_butter)))
hold on
plot(norm_freq_axis, 20*log10(abs(H_cheby)), 'r')
plot(norm_freq_axis, 20*log10(abs(H_ellip)), 'g')
legend('Butterworth', 'Chebyshev', 'Elliptical')
xlabel('Normalised Frequency ');
ylabel('Magnitude (dB)')

```

EQ-ator equalizer implementation in Matlab:

Command editor

```

function varargout = EQator(varargin)
% EQator M-file for EQator.fig
%   EQator, by itself, creates a new EQator or raises the existing
%   singleton*.
%
%   H = EQator returns the handle to a new EQator or the handle to
%   the existing singleton*.
%
%   EQator('CALLBACK',hObject,eventData,handles,...) calls the lo-
%   cal
%   function named CALLBACK in EQator.M with the given input argu-
%   ments.
%
%   EQator('Property','Value',...) creates a new EQator or raises
%   the
%   existing singleton*. Starting from the left, property value
%   pairs are
%   applied to the GUI before EQator_OpeningFcn gets called. An
%   unrecognized property name or invalid value makes property ap-
%   plication
%   stop. All inputs are passed to EQator_OpeningFcn via varargin.
%
%   *See GUI Options on GUIDE's Tools menu. Choose "GUI allows on-
%   ly one
%   instance to run (singleton)".
%
% See also: GUIDE, GUIDATA, GUIHANDLES

% Edit the above text to modify the response to help EQator

% Last Modified by GUIDE v2.5 24-Apr-2015 02:00:48

% Begin initialization code - DO NOT EDIT
gui_Singleton = 1;
gui_State = struct('gui_Name',       mfilename, ...
                  'gui_Singleton',  gui_Singleton, ...
                  'gui_OpeningFcn', @EQator_OpeningFcn, ...
                  'gui_OutputFcn',  @EQator_OutputFcn, ...
                  'gui_LayoutFcn',  [], ...
                  'gui_Callback',   []);
if nargin && ischar(varargin{1})
    gui_State.gui_Callback = str2func(varargin{1});
end

if nargin
    [varargout{1:nargout}] = gui_mainfcn(gui_State, varargin{:});
else
    gui_mainfcn(gui_State, varargin{:});
end
% End initialization code - DO NOT EDIT

% --- Executes just before EQator is made visible.

```

```
function EQator_OpeningFcn(hObject, eventdata, handles, varargin)
% This function has no output args, see OutputFcn.
% hObject    handle to figure
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
% varargin   command line arguments to EQator (see VARARGIN)
global stop C Fs;
stop=1;
Fs=44100;

C=zeros(1,5);
set(handles.C1_var, 'min', -15);
set(handles.C1_var, 'max', 15);
set(handles.C1_var, 'value', 0);
set(handles.C1_var, 'SliderStep', [0.025,0.05]);
set(handles.C1_val, 'string', num2str(0));

set(handles.C2_var, 'min', -15);
set(handles.C2_var, 'max', 15);
set(handles.C2_var, 'value', 0);
set(handles.C2_var, 'SliderStep', [0.025,0.05]);
set(handles.C2_val, 'string', num2str(0));

set(handles.C3_var, 'min', -15);
set(handles.C3_var, 'max', 15);
set(handles.C3_var, 'value', 0);
set(handles.C3_var, 'SliderStep', [0.025,0.05]);
set(handles.C3_val, 'string', num2str(0));

set(handles.C4_var, 'min', -15);
set(handles.C4_var, 'max', 15);
set(handles.C4_var, 'value', 0);
set(handles.C4_var, 'SliderStep', [0.025,0.05]);
set(handles.C4_val, 'string', num2str(0));

set(handles.C5_var, 'min', -15);
set(handles.C5_var, 'max', 15);
set(handles.C5_var, 'value', 0);
set(handles.C5_var, 'SliderStep', [0.025,0.05]);
set(handles.C5_val, 'string', num2str(0));

equalizer_plot();
% Choose default command line output for EQator
handles.output = hObject;

% Update handles structure
guidata(hObject, handles);

% UIWAIT makes EQator wait for user response (see UIRESUME)
% uiwait(handles.figure1);

% --- Outputs from this function are returned to the command line.
function varargout = EQator_OutputFcn(hObject, eventdata, handles)
% varargout  cell array for returning output args (see VARARGOUT);
% hObject    handle to figure
```

```
% eventdata reserved - to be defined in a future version of MATLAB
% handles structure with handles and user data (see GUIDATA)

% Get default command line output from handles structure
varargout{1} = handles.output;

% --- Executes on slider movement.
function C1_var_Callback(hObject, eventdata, handles)
% hObject handle to C1_var (see GCBO)
% eventdata reserved - to be defined in a future version of MATLAB
% handles structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'Value') returns position of slider
% get(hObject,'Min') and get(hObject,'Max') to determine range
of slider
global C;
C(1)=get(hObject,'value');
set(handles.C1_val,'string',num2str(C(1)));

% --- Executes during object creation, after setting all properties.
function C1_var_CreateFcn(hObject, eventdata, handles)
% hObject handle to C1_var (see GCBO)
% eventdata reserved - to be defined in a future version of MATLAB
% handles empty - handles not created until after all CreateFcns
called

% Hint: slider controls usually have a light gray background.
if isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUiControlBackgroundColor'))
    set(hObject,'BackgroundColor',[.9 .9 .9]);
end

% --- Executes on slider movement.
function C2_var_Callback(hObject, eventdata, handles)
% hObject handle to C2_var (see GCBO)
% eventdata reserved - to be defined in a future version of MATLAB
% handles structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'Value') returns position of slider
% get(hObject,'Min') and get(hObject,'Max') to determine range
of slider
global C;
C(2)=get(hObject,'value');
set(handles.C2_val,'string',num2str(C(2)));

% --- Executes during object creation, after setting all properties.
function C2_var_CreateFcn(hObject, eventdata, handles)
% hObject handle to C2_var (see GCBO)
% eventdata reserved - to be defined in a future version of MATLAB
% handles empty - handles not created until after all CreateFcns
called

% Hint: slider controls usually have a light gray background.
```

```

if isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUiControlBackgroundColor'))
    set(hObject,'BackgroundColor',[.9 .9 .9]);
end

% --- Executes on slider movement.
function C3_var_Callback(hObject, eventdata, handles)
% hObject    handle to C3_var (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'Value') returns position of slider
%         get(hObject,'Min') and get(hObject,'Max') to determine range
of slider
global C;
C(3)=get(hObject,'value');
set(handles.C3_val,'string',num2str(C(3)));

% --- Executes during object creation, after setting all properties.
function C3_var_CreateFcn(hObject, eventdata, handles)
% hObject    handle to C3_var (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns
called

% Hint: slider controls usually have a light gray background.
if isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUiControlBackgroundColor'))
    set(hObject,'BackgroundColor',[.9 .9 .9]);
end

% --- Executes on slider movement.
function C4_var_Callback(hObject, eventdata, handles)
% hObject    handle to C4_var (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'Value') returns position of slider
%         get(hObject,'Min') and get(hObject,'Max') to determine range
of slider
global C;
C(4)=get(hObject,'value');
set(handles.C4_val,'string',num2str(C(4)));

% --- Executes during object creation, after setting all properties.
function C4_var_CreateFcn(hObject, eventdata, handles)
% hObject    handle to C4_var (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns
called

% Hint: slider controls usually have a light gray background.
if isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUiControlBackgroundColor'))

```

```

        set(hObject, 'BackgroundColor', [.9 .9 .9]);
    end

% --- Executes on slider movement.
function C5_var_Callback(hObject, eventdata, handles)
% hObject    handle to C5_var (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject, 'Value') returns position of slider
%        get(hObject, 'Min') and get(hObject, 'Max') to determine range
of slider
global C;
C(5)=get(hObject, 'value');
set(handles.C5_val, 'string', num2str(C(5)));

% --- Executes during object creation, after setting all properties.
function C5_var_CreateFcn(hObject, eventdata, handles)
% hObject    handle to C5_var (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns
called

% Hint: slider controls usually have a light gray background.
if isequal(get(hObject, 'BackgroundColor'),
get(0, 'defaultUiControlBackgroundColor'))
    set(hObject, 'BackgroundColor', [.9 .9 .9]);
end

% --- Executes on button press in plot_H.
function plot_H_Callback(hObject, eventdata, handles)
% hObject    handle to plot_H (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
global C Fs;
equalizer_plot();

function equalizer_plot()
global C Fs;
[a,b]=coef();
H=0;
for i=1:5
    H=H+10^(C(i)/20)*abs(freqz(b{i},a{i},1024));
end
f = Fs*[0:1023]/2048;
semilogx(f,20*log10(H));
xlabel('Frequency [Hz]');
ylabel('Magnitude [dB]');
title('Audio Equalizator');
axis([10 Fs/2 -21 21]);
grid on;

function [a,b]=coef()
global Fs;

```

```

%1.Filter
Fp1=198/(Fs/2);
[b1,a1]=cheby2(4,48,Fp1,'low');

%2.Filter
Fp2=[46,747]/(Fs/2);
[b2,a2]=cheby2(4,51,Fp2,'bandpass');

%3.Filter
Fp3=[200 3000]/(Fs/2);
[b3,a3]=cheby2(4,50,Fp3,'bandpass');

%4.Filter
Rp4=0.5;
Rs4=30;
Fp4=[1070 6000]/(Fs/2);
[b4,a4]=cheby2(4,45,Fp4,'bandpass');

%5.Filter
Rp5=0.5;
Rs5=30;
Fp5=1800/(Fs/2);
[b5,a5]=cheby2(4,49,Fp5,'high');
a={a1,a2,a3,a4,a5};
b={b1,b2,b3,b4,b5};

% --- Executes on button press in reset_H.
function reset_H_Callback(hObject, eventdata, handles)
% hObject    handle to reset_H (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
global Fs C;
C=zeros(1,5);
%Fs=44100;
Fs=48000;
set(handles.C1_val,'string',num2str(0));
set(handles.C2_val,'string',num2str(0));
set(handles.C3_val,'string',num2str(0));
set(handles.C4_val,'string',num2str(0));
set(handles.C5_val,'string',num2str(0));

set(handles.C1_var,'value',0);
set(handles.C2_var,'value',0);
set(handles.C3_var,'value',0);
set(handles.C4_var,'value',0);
set(handles.C5_var,'value',0);

equalizer_plot();

function C1_val_Callback(hObject, eventdata, handles)
% hObject    handle to C1_val (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
global C;

```

```

C(1)=str2num(get(hObject,'string'));
minn=get(handles.C1_var,'min');
maxx=get(handles.C1_var,'max');
if(C(1)<minn || C(1)>maxx)
    C(1)=get(handles.C1_var,'value');
    set(hObject,'string',num2str(0));
else
    set(handles.C1_var,'value',C(1));
end
% Hints: get(hObject,'String') returns contents of C1_val as text
%        str2double(get(hObject,'String')) returns contents of C1_val
as a double

% --- Executes during object creation, after setting all properties.
function C1_val_CreateFcn(hObject, eventdata, handles)
% hObject    handle to C1_val (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns
called

% Hint: edit controls usually have a white background on Windows.
%        See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end

function C2_val_Callback(hObject, eventdata, handles)
% hObject    handle to C2_val (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of C2_val as text
%        str2double(get(hObject,'String')) returns contents of C2_val
as a double
global C;
C(2)=str2num(get(hObject,'string'));
minn=get(handles.C2_var,'min');
maxx=get(handles.C2_var,'max');
if(C(2)<minn || C(2)>maxx)
    C(2)=get(handles.C2_var,'value');
    set(hObject,'string',num2str(0));
else
    set(handles.C2_var,'value',C(2));
end

% --- Executes during object creation, after setting all properties.
function C2_val_CreateFcn(hObject, eventdata, handles)
% hObject    handle to C2_val (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns
called

```

```

% Hint: edit controls usually have a white background on Windows.
% See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUiControlBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end

function C3_val_Callback(hObject, eventdata, handles)
% hObject handle to C3_val (see GCBO)
% eventdata reserved - to be defined in a future version of MATLAB
% handles structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of C3_val as text
% str2double(get(hObject,'String')) returns contents of C3_val
as a double
global C;
C(3)=str2num(get(hObject,'string'));
minn=get(handles.C3_var,'min');
maxx=get(handles.C3_var,'max');
if(C(3)<minn || C(3)>maxx)
    C(3)=get(handles.C3_var,'value');
    set(hObject,'string',num2str(0));
else
    set(handles.C3_var,'value',C(3));
end

% --- Executes during object creation, after setting all properties.
function C3_val_CreateFcn(hObject, eventdata, handles)
% hObject handle to C3_val (see GCBO)
% eventdata reserved - to be defined in a future version of MATLAB
% handles empty - handles not created until after all CreateFcns
called

% Hint: edit controls usually have a white background on Windows.
% See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUiControlBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end

function C4_val_Callback(hObject, eventdata, handles)
% hObject handle to C4_val (see GCBO)
% eventdata reserved - to be defined in a future version of MATLAB
% handles structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of C4_val as text
% str2double(get(hObject,'String')) returns contents of C4_val
as a double
global C;
C(4)=str2num(get(hObject,'string'));
minn=get(handles.C4_var,'min');
maxx=get(handles.C4_var,'max');

```

```

if(C(4)<minn || C(4)>maxx)
    C(4)=get(handles.C4_var, 'value');
    set(hObject, 'string', num2str(0));
else
    set(handles.C4_var, 'value', C(4));
end

% --- Executes during object creation, after setting all properties.
function C4_val_CreateFcn(hObject, eventdata, handles)
% hObject    handle to C4_val (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns
called

% Hint: edit controls usually have a white background on Windows.
%         See ISPC and COMPUTER.
if ispc && isequal(get(hObject, 'BackgroundColor'),
get(0, 'defaultUiControlBackgroundColor'))
    set(hObject, 'BackgroundColor', 'white');
end

function C5_val_Callback(hObject, eventdata, handles)
% hObject    handle to C5_val (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject, 'String') returns contents of C5_val as text
%         str2double(get(hObject, 'String')) returns contents of C5_val
as a double
global C;
C(5)=str2num(get(hObject, 'string'));
minn=get(handles.C5_var, 'min');
maxx=get(handles.C5_var, 'max');
if(C(5)<minn || C(5)>maxx)
    C(5)=get(handles.C5_var, 'value');
    set(hObject, 'string', num2str(0));
else
    set(handles.C5_var, 'value', C(5));
end

% --- Executes during object creation, after setting all properties.
function C5_val_CreateFcn(hObject, eventdata, handles)
% hObject    handle to C5_val (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns
called

% Hint: edit controls usually have a white background on Windows.
%         See ISPC and COMPUTER.
if ispc && isequal(get(hObject, 'BackgroundColor'),
get(0, 'defaultUiControlBackgroundColor'))
    set(hObject, 'BackgroundColor', 'white');
end

% --- Executes on button press in load.
function load_Callback(hObject, eventdata, handles)

```

```

% hObject     handle to load (see GCBO)
% eventdata   reserved - to be defined in a future version of MATLAB
% handles     structure with handles and user data (see GUIDATA)

global file_name;
%file_name=uigetfile('*wav','Choise a .wav file');
file_name=uigetfile('*wav','Select Audio File.wav');
% --- Executes on button press in play.
function play_Callback(hObject, eventdata, handles)
% hObject     handle to play (see GCBO)
% eventdata   reserved - to be defined in a future version of MATLAB
% handles     structure with handles and user data (see GUIDATA)

global stop file_name C;
stop=1;
equalizer_play();

function equalizer_play()
global stop file_name C;
disp('file name:');
disp(file_name);
[x,Fs]=wavread(file_name);
%[x,Fs]=audioread(file_name);

[a,b]=coef();
%l_sector=2*Fs;
%Nb=round(length(x)/l_sector);
l_sector=length(x);
Nb = 1;

y=0;

for i=1:floor(Nb)
    disp('processing');
    sector=x((i-1)*l_sector+1:i*l_sector);
    for k=1:5
        y=y+filter(10^(C(k)/20)*b{k},a{k},sector);
        if(stop==0)
            break;
        end
    end
    %wavplay(y,Fs,'async');

    player=audioplayer(y,Fs);

    player.playblocking;

    y=0;
    if(stop==0)
        break;
    end
end

disp('playing finished');

% --- Executes on button press in stop.

```

```
function stop_Callback(hObject, eventdata, handles)
% hObject      handle to stop (see GCBO)
% eventdata    reserved - to be defined in a future version of MATLAB
% handles      structure with handles and user data (see GUIDATA)
global stop;
stop=0;

% --- Executes on slider movement.
function slider6_Callback(hObject, eventdata, handles)
% hObject      handle to slider6 (see GCBO)
% eventdata    reserved - to be defined in a future version of MATLAB
% handles      structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'Value') returns position of slider
%         get(hObject,'Min') and get(hObject,'Max') to determine range
of slider

% --- Executes during object creation, after setting all properties.
function slider6_CreateFcn(hObject, eventdata, handles)
% hObject      handle to slider6 (see GCBO)
% eventdata    reserved - to be defined in a future version of MATLAB
% handles      empty - handles not created until after all CreateFcns
called

% Hint: slider controls usually have a light gray background.
if isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUiControlBackgroundColor'))
    set(hObject,'BackgroundColor',[.9 .9 .9]);
end
```