UPPSALA
UNIVERSITET

ISRN UTH-INGUTB-EX-E-2017/04-SE
Examensarbete 15 hp
Juni 2017

Real time digital signal processing using Matlab

Jesper Nordström

UPPSALA UNIVERSITET

Teknisk- naturvetenskaplig fakultet
UTH-enheten

Besöksadress:
Ångströmlaboratoriet
Lägerhyddsvägen 1
Hus 4, Plan 0

Postadress:
Box 536
751 21 Uppsala

Telefon:
018 – 471 30 03

Telefax:
018 – 471 30 00

Hemsida:
http://www.teknat.uu.se/student

Abstract

Real time digital signal processing using Matlab

Jesper Nordström

Increased usage of electronic devices and the fast development of microprocessors has increased the usage of digital filters ahead of analog filters. Digital filters offer great benefits over analog filters in that they are inexpensive, they can be reprogrammed easily and they open up whole new range of possibilities when it comes to Internet of things.

This thesis describes development of a program that can sample music from the computer's microphone input, filter it inside the program with user built filters and reconstruct the music to the computer's headphone output meaning that the music can be played from the speakers. All of this is to happen in real time. The program is developed for students studying at the department of ``Signals and Systems'' and the program is supposed the be one of the educational tools to make sense of signals and filtering.

The program works well and filters the sound with satisfying results. It is easy to create filters and filter the signal. Since it is music that is filtered constructing perfect filters with minimum ripple, minimum or linear phase is quite difficult to achieve. The program could be improved by improving the user interface, making the environment more interactive and less difficult to construct good filters. Some improvements could also be made to the implementation; as of now the program might run a bit slow on startup on slower computers.

Handledare: Ping Wu
Ämnesgranskare: Steffi Knorn
Examinator: Tomas Nyberg
ISRN UTH-INGUTB-EX-E-2017/04-SE

Acknowledgements

Jag har engagerat många vänner, klasskompisar och familj för stöd och input till det här projektet. Ett stort tack till mina vänner i klassen som alltid finns där för de små frågorna som hjälper en att friska upp minnet. Ett speciellt tack till Mattias Dahlstedt som helt obekymrat bara alltid finns där och som alltid vet. Tack Filip Östermark som stöttade och hjälpte med sin stora kunskap i ett område där jag var ny och obekväm. Tack till Rachel Northedge som tog sig tid som hon egentligen saknade för att ge mig värdefull input. Stort tack till Ping Wu, min handledare som alltid är stöttande, positiv och framför allt alltid gör sitt yttersta för att hjälpa en och ger en känsla av att han vill se en lyckas. Tack till Steffi Knorn min ämnesgranskare som gav mycket input i slutet av projektet. En person som inte går att tacka nog är min fantastiska flickvän Elvira Bergström. Utan henne hade jag inte orkat komma tillbaka från den tuffa tid som följde olyckan och som stöttar en med det som behövs, oavsett om det är värmande ord eller att göra snygga bilder till rapporten. Ord kan inte beskriva hur tacksam jag är.

Sammanfattning

Världen vi lever i blir allt mer digital. Snart kommer det att finnas en microprocessor i varje elektronisk pryl som finns i våra hem, inklusive all köksapparatur och andra prylar i våra hem som kan drivas med el. I framtiden kommer många av dessa prylar också vara uppkopplade på nätet, sammanlänkat i ett Internet of things. I all elektronisk apparatur finns det filter av olika slag, filter som filtrerar brus, filter som skyddar mer känslig elektronik och filter som ändrar signaler. Där det finns en mikroprocessor så finns också möjligheter att konstruera och använda digitala filter. Digitala filter har många fördelar gentemot analoga filter i det att de är väldigt billiga att tillverka och implementera, de är lätta att ändra egenskaper på bara genom att programmera om mjukvaran och att de inte ändrar egenskaper i takt med att elektroniken blir gammal och sliten. Nackdelar är att digitala filter inte får ett lika platt impulssvar och inte heller är lika precisa eller billiga som analoga filter i enkla fall. Digitala filter introducerar också en fördröjning i systemet, något som är marginellt och försumbart vad gäller analoga filter.

I detta examensarbete utvecklas ett program som körs i Matlab genom att man kopplar in en ljudspelande dator eller mobil i mic-uttaget på en dator, samplar det inkommande ljudet och filtrerar ljudet i programmet. Efter att ljudet blivit filtrerat så kan ljudet spelas upp genom hörlursuttaget på dator, allt i realtid. Programmet ska alltså fungera som en lite mer avancerad equalizer.

Programmet är utvecklat i Matlab med hjälp av Matlabs DSP toolbox. Syftet är att programmet ska kunna användas i utbildningen på avdelningen för ”Signaler och system”. Därav har mycket tanke vigts åt att programmet ska vara så pedagogiskt som möjligt med en så intuitiv GUI som möjligt. Så mycket som möjligt av funktionerna har försökt att vara självförklarande så långt det går och där förklaring behövs finns det info-knappar med enklare förklaringar. Studenter ska alltså kunna starta programmet, tillverka och implementera filter i programmets miljö samt kunna testa dessa filter på riktigt ljud i realtid.

Programmet fungerar väl och låter användaren på ett enkelt sätt implementera och tillverka filter. Filtrering av musik är svårt, perfekta filter krävs och det är inte enkelt att tillverka filter som inte ger någon som helst distortion. Att vidareutveckla och göra programmet enklare att implementera på musik genom att ha en kreativ men ändå vetenskaplig guide till att få filtret så precist som krävs för musik hade varit bra. Det hade kunnat öka kreativiteten och men samtidigt också lärandet.

Contents

Chapter 1

Introduction

1.1 Background

In this day and age, as every side of our daily lives shifts more and more into the digital world, the technology we use is developing fast. According to statistics conducted by the authority of Swedish statistics (SCB), 93 percent of the Swedish population had access to the internet in the first quarter of 2016 [page 4 15]. Today 50 percent of the population listens to music via the internet[page 6 15]. It is not only mobile phones, computers and tablets that use digital technology anymore. According to Intel's website more than 200 billion devices will have "smart"-technology with at least a microprocessor in 2020[2].

In almost every kitchen appliance, camera, mobile phone or music playing device there are filters. Some are simple analog ones that only work as protection for the more delicate electronics, while others include complicated dj music mixing board filters where one undesired frequency can destroy an entire song. In mp3-players and various other types of music-playback devices it is important that the listener can listen to a song without noise, also different genres of music require different sorts of filtering.

Digital filters will have a large role to play as electronic devices become increasingly "smart". Microprocessors open up a whole new range of possibilities when it comes to filtering. Digital filters are not as precise as analog filters, nor do they have equally as flat impulse responses. However they are very cost effective, especially if the device already has a microprocessor. They can also be reprogrammed easily and if the device has an internet connection, the impulse response can easily be changed with just a software update.

It is often not required for the filter to run in real-time, if the input exists in data files and can be computed beforehand. However often the input signal is not known, the signal has perhaps to be sampled, patterns must be recognized and signal operations performed while the application is running. One example of this is a telephone call with a noisy background. It will be sampled into the sender's telephone, filtered and then played in the ear of the receiver, with the noise filtered to the extent that it is easy to interpret what the sender is saying. There are other more advanced examples of this, like machine vision and smart cameras, Biometrics and Grid automation[3].

1.2 Aim of the project

The aim is to create a program that can sample sound from a mic input, perform Digital Signal Processing (DSP) on the signal and then play it using the computer's headphone output. The program is to be used as a supplement to laboratory exercises and lectures and thus be an educational tool. The program should not be too heavy on the computers and pedagogic to some extent to make it accessible.

1.3 Project description

This program should be created so that it can sample from the mic input of an ordinary personal computer, filter it through some variety of filters, either some of the built in filters that already exists in the program or by creating a filter with coefficients of the users preferences. The program is thought of as an educational tool to complete the already existing laboratations and homework studies in the course "Signals and Systems" at the department of Signals and Systems at Uppsala University, a syllabus can be found in Appendix A.

There are a variety of functions that this program should offer, among which the most important ones are listed below.

- Sample and perform operations on sound either from mic input or file input.
- Play audio back to the user from the computer's headphone output.
- Change the sampling rate of the sound that is sampled from mic.
- Read out the time plot of the sampled signal.
- Read out the frequency spectrum of the signal.
- Create a filter from a custom filter design by typing in the coefficients of the filter. This works for both IIR and FIR filters as will be seen in Section 2.3.
- Create a filter by specifying the type of filter, sampling frequency, cut off frequency etcetera and further on apply the filter to the input signal.
- Apply a function to the filter. This opens up the opportunity to create more advanced filters, cascaded filters or filters that invite to play with the sound signal, thereby encouraging interest in the subject from students.

The usage of the program is supposed to be straightforward and a block diagram describing the basics of the program is shown below in Figure 1.1.

1. Choose whether the source should be from a file or from mic input.
   (a) If file: type in the file name in the edit file name text box.
   (b) If mic: set the desired sample rate in the sample rate text box.
2. Construct your own filter by typing in the coefficients of the numerator and the denominator in their respective text box. The default transfer function should be (1/1), in other words no filtration at all.
3. Optional: construct a Matlab filter using the built in filter construction algorithm "designfilt" from Matlab.
4. Optional: choose one of the prefabricated filters from the pop-up menu.

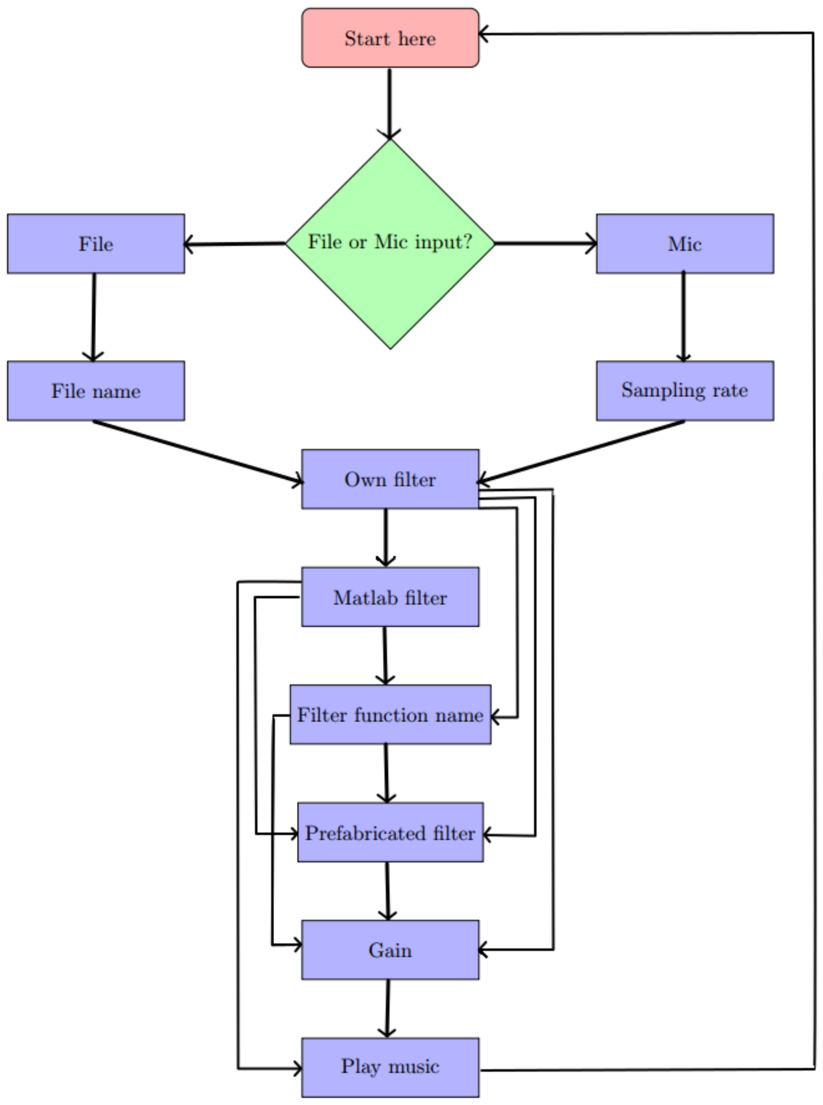


Figure 1.1: A flowchart of how the main GUI should be used.

5. Specify the gain to the output. The default value is 1 (no gain).
6. Press the on-button and play the music. Filter by switching between the different types of filtration and activate the filtration by pressing the apply filtration tick box.

It should also be possible to pause while the program is running. Either by pausing the entire program or by pausing the FFT plot or time plot alone. Pausing the FFT plot or time plot is merely to give the program some increased performance. Pausing the program should allow the user to freeze the song at a particular moment and acquire some information about the signal.

1.4 Project specifications

There are no precise technical numbers specifications for the project. However as the program is to be used for educational purposes, and with students that have different types of computers, it would be favourable if the program was not too computationally demanding.

The program should also include some prefabricated filters, at least one IIR-filter per type (bandpass, lowpass, highpass) as well as at least one FIR-filter per type.

Chapter 2

Theory

In this chapter the essential underlying theory of which the program is based on is discussed. The outline of the chapter is written in the same order as the theory is used in the program. Starting with some theory about analog signals, continuing on how the signal is sampled and quantized into a digital signal inside the computer's sound card and further on how the signal is processed inside the computer and played back to the user. A block diagram describing the system is shown below in Figure 2.1.

There are three types of signals; continuous-time analog signals, discrete-time signals, and digital signals[6]. The world we live in is analog, the physical signals such as speech and sound are continuous-time analog signals that propagate through some medium, typically air, and then affect our hearing sense through vibrations. The ear has a complicated system of ear canals and fluids which are manipulated by hairs that vibrate in the same frequency as the tone. There are nerves that send signals to the brain depending on the vibrations and the pressure of the fluids[10].

2.1 Analog to digital conversion

A computer can only store or handle binary data. The analog signal has to be sampled and be converted to digital data which is an operation by the Analog to Digital Converter (ADC) in the computer's sound card. This is done by taking samples of a continuous-time signal, see Figure 2.2a. This will be a discrete-time signal and could be, not always, a

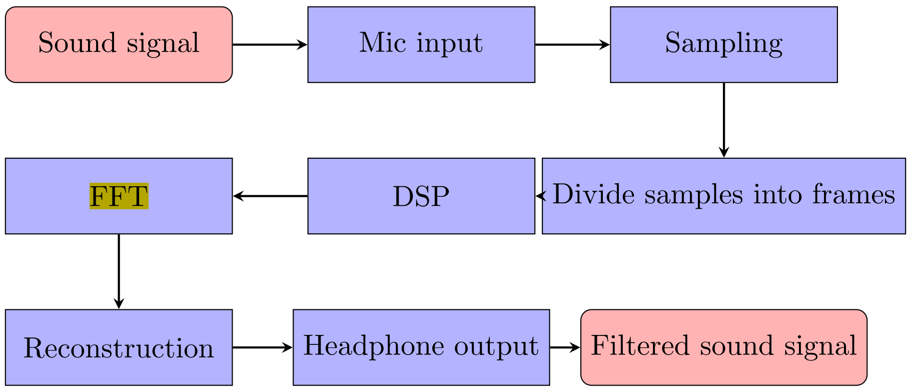


Figure 2.1: A block diagram describing the entire system including the Digital Signal Processing (DSP).

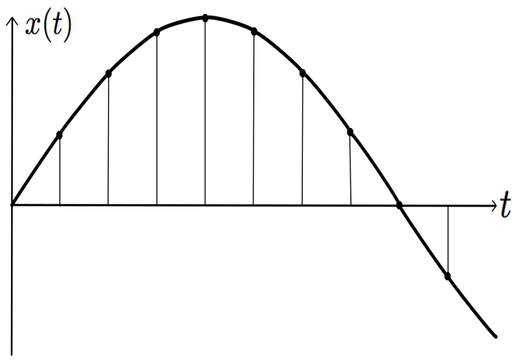
digital signal[6]. After the sampling process the signal has to be quantized, meaning that the signal at time $x_d(t_0)$ has to be converted into its closest digital representation. The sampling process is described by Equation 2.1[4].

$$x_d(t) = x_c(n\Delta t)\sum_n \delta(t - n\Delta t) \tag{2.1}$$

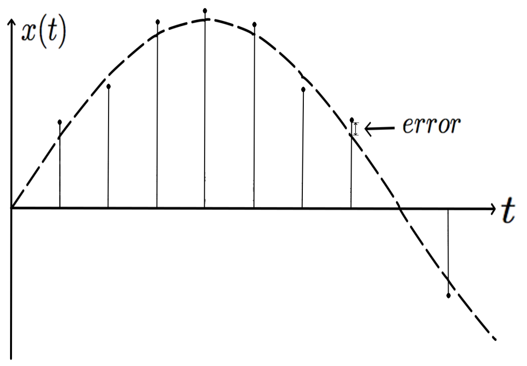
The Δt in Equation 2.1 is the time in between the impulses. x_d and x_c is the discrete-time and continuous-time signal respectively. From the convolution theorem a multiplication in the time domain as in Equation 2.1 corresponds to a convolution in the frequency domain[4].

$$X_d(\omega) = X_c(\omega) * \mathcal{F}\{\sum \delta(t - n\Delta t)\} \tag{2.2}$$

Sampling a signal using Equation 2.1 is the same as convolving the signal $X_c(w)$ (illustrated in Figure 2.3a) with the spectrum of the impulse train $\mathcal{F}\{\sum \delta(t - n\Delta t)\}$ (illustrated in Figure 2.3b). The result is illustrated i Figure 2.4.

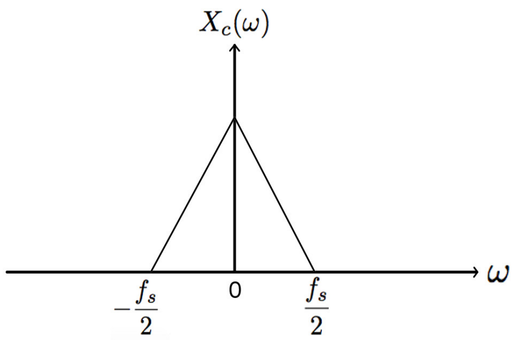


(a) Sampling process.

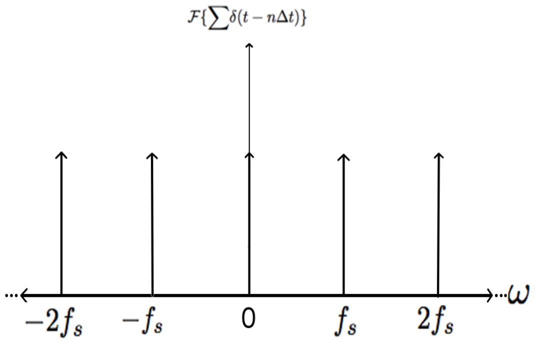


(b) Quantization process.

Figure 2.2: The figures illustrate the sampling process and the quantization process.



(a) Signal spectrum.



(b) Impulse train.

Figure 2.3: The figures illustrate the two signals that are being convolved.

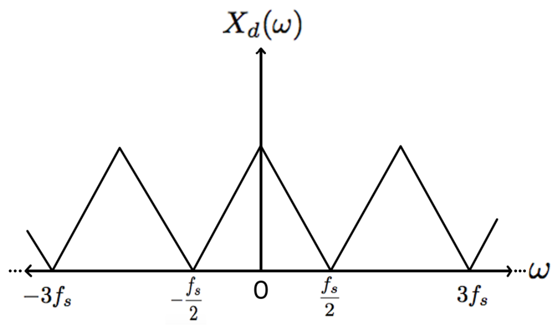


Figure 2.4: Fourier transform of the sampled signal.

2.2 Sampling and frames

For the Digital Signal Processing to be executed in real time there are some important criteria to be met, considering the rate at which samples are taken and the size of the frames that store these samples.

2.2.1 Sampling

The Δt constant in Equations 2.1 and 2.2 specifies at which rate the signal is sampled at. According to the Nyquist theorem, the sampling frequency f_s must be choosen at least twice as large as the bandwidth of the signal (f_M) to allow the original signal to be constructed perfectly from the samples. The Nyquist theorem is stated below in Equation 2.3[16].

$$f_s \geq 2f_M \tag{2.3}$$

If the sampled signal would have contained higher frequencies than $f_s/2$ then the signal would not be able to be constructed perfectly anymore. The signal would suffer from aliasing distortion meaning that the frequencies higher than $f_s/2$ would be interpreted as lower frequencies[13]. Equation 2.4 shows how this periodical property of a sampled signal can interpret a higher frequency $f_s/2+a$ as a low frequency[16]. It folds higher frequencies back into the range $[-f_s/2, f_s/2]$ [11].

$$f_s/2 + a \rightarrow f_s/2 - a \tag{2.4}$$

This is called aliasing distortion and as Figure 2.5 shows, makes the convolution between the signal and impulse train overlap itself thereby adding the overlapping parts. Taking samples more often compresses the impulse train in time domain, but in frequency domain the impulse train is expanded. Figure 2.5 shows the convolution between the signal $X_c(\omega)$ and the impulse train $\mathcal{F}\{\sum \delta(t - n\Delta t)\}$ when there is aliasing distortion[4].

2.2.2 Frames

After the ADC has taken the samples and converted them to be represented with a binary representation the samples have to be stored. This operation is not performed by the computer's sound card, it is performed in Matlab. A common method is to use frames

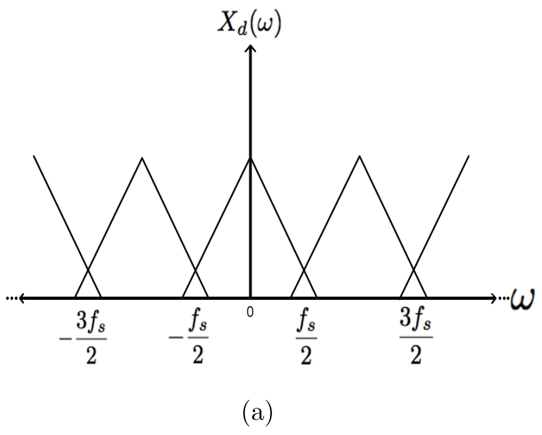


Figure 2.5: Illustration of overlapping and aliasing.

and let one frame be a finite number of samples. The frames are then processed one at a time[14] as illustrated in Figure 2.6.

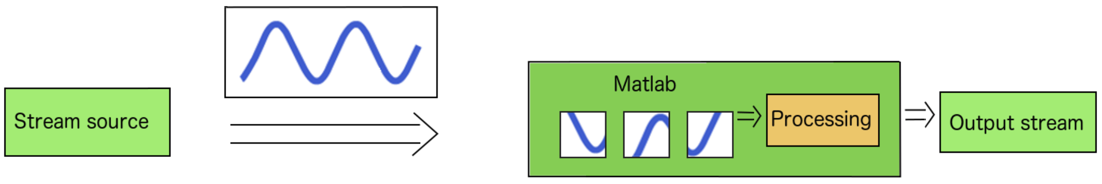


Figure 2.6: Illustration of frame processing.

Frame processing does three things that take time. They are illustrated in Figure 2.7. There is some time necessary to acquire the frame which is called overhead, some time necessary to process the frame and finally a buffer. As long as the overhead and the processing time is less than the frame time, meaning that there still is some buffer, the frame is processed in real time.

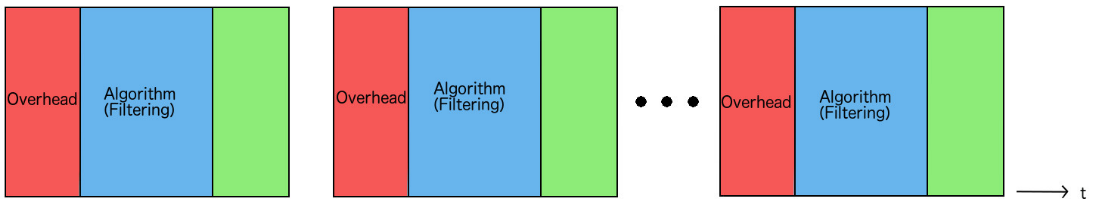
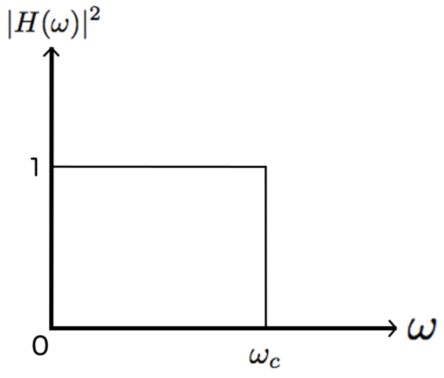


Figure 2.7: Illustration of frame processing in real time.

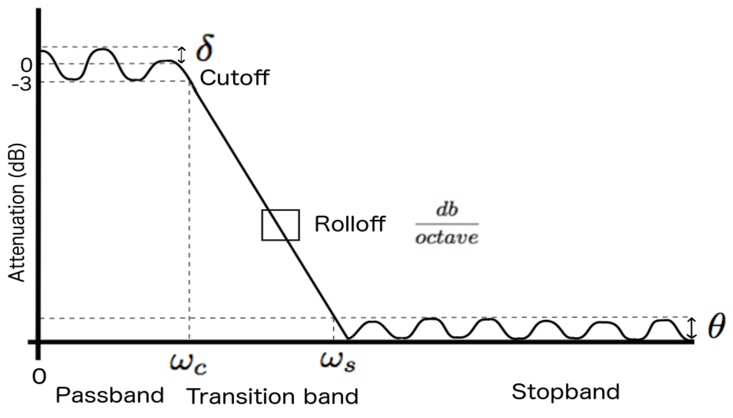
2.3 Digital filters

When discussing digital filters or filters in general, it is important to keep in mind how the ideal filter would behave.

$$H(\omega)H(-\omega) = |H(\omega)|^2 = \frac{1}{1 + F(\omega^2)} \tag{2.5}$$



(a) The brickwall lowpass filter.



(b) Specification of a lowpassfilter.

Figure 2.8

The ideal filter does not filter or affect the amplitude in the passband, but in the stopband the signal should be completely attenuated. An ideal filter also has no passband ripple ($\delta = 0$) and no stopband ripple ($\theta = 0$), with an infinite slope in the rolloff at the cutoff frequency ω_c[13].

A typical way of approximating this behavior is by applying a squared magnitude transfer function as described in Equation 2.5. This is the underlying theory behind the butterworth prototype[4].

Depending on the application of the filter, the phase response could also be made to have some desired specifications. When mastering sound, both linear phase and minimum phase filters are used, depending on the desired effect[7].

2.3.1 Finite impulse response filters

The main advantage of FIR filters is that it is easy to control the phase shift that the filter applies to the signal. In some applications this is crucial. For example, when it comes to music an irregular phase shift can lead to phase distortion[16][10].

An FIR filter is a non-recursive filter meaning that none of the outputs are used as an input. The general difference Equation is given in Equation 2.6 where N is the number of coefficients and N-1 is the filter order. The filter structure is seen in Figure 2.9 which is an illustration of the z-transform of Equation 2.7[4].

$$y(n) = \sum_{i=0}^{N-1} a_i x(n-i) \tag{2.6}$$

$$Y(z) = X(z) \sum_{i=0}^{N-1} a_i z^{-i} \tag{2.7}$$

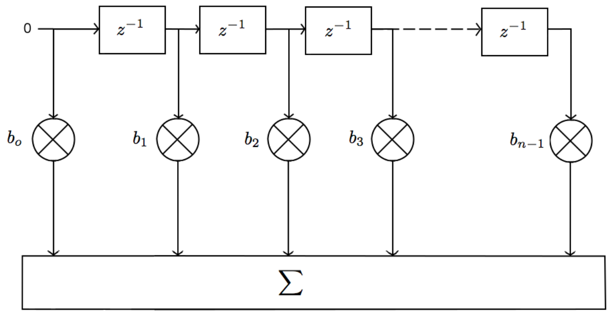


Figure 2.9: The structure of the non-recursive FIR filter.

To construct an FIR filter there are a number of methods, which all yield to find the coefficients $h(n)$. One of the more common methods is the windowing method. This is a straight forward method where the desired impulse response is chosen first. The stopband attenuation, filter class (lowpass, bandpass, etc) together with the cutoff frequencies is chosen. The analog cutoff frequencies have to be converted into digital frequencies by the use of Equation 2.8. The coefficients are then calculated by the help of Equation 2.9 which is the inverse DTFT[4].

$$w_c = \frac{2\pi f_c}{f_s} \tag{2.8}$$

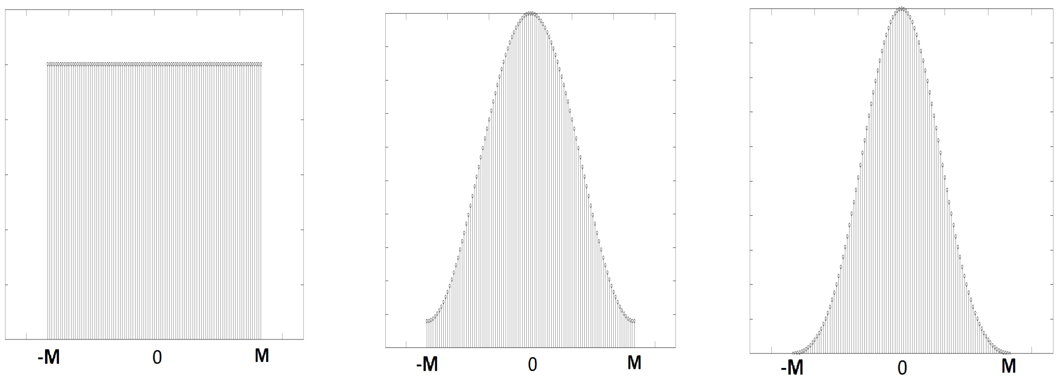
$$h_d(n) = \frac{f_s}{2\pi} \int_{-\pi}^{\pi} H_{ideal}(w) e^{j\omega n} d\omega \tag{2.9}$$

$$h_w(n) = w(n) h_d(n) \tag{2.10}$$

The result of Equation 2.9 is not a finite series. $h_d(n)$ has to be truncated by some window with the same length as the desired number of coefficients for it to be finite. Perhaps the easiest one is the simple rectangular window illustrated in Figure 2.10a. There are however other windows to choose from, as shown in 2.10[13].

The windows are always symmetric. These tapered window functions can decrease the stopband ripple but then there is a trade-off between some ripple and a small shift in the cutoff frequency[13].

Finally, for the filter to be a functioning FIR filter the filter has to be right shifted by half the length of the filter for it to become causal which is done by Equation 2.11. The h(n) coefficients is simply the impulse response of the filter[13].



(a) Rectanglar window. (b) Hamming window. (c) Blackman window.

Figure 2.10: The figures illustrate different windows.

$$h(n) = h_w(n - \frac{N}{2}) \tag{2.11}$$

Matlab can use the equiripple method that leans on the Chebyshev criterion which yields to a minimization of the maximum error to minimize the passband ripple[11]. The theory is quite complex and is also quite computationally demanding[13].

2.3.2 Infinite impulse response filters

The general difference equation of an IIR filter can be seen in Equation 2.12. And the general transfer function by Equation 2.13.

$$y(n) = \sum_{i=0}^{L-1} b_i x(n-l) - \sum_{m=1}^{M} a_m y(n-m) \tag{2.12}$$

$$H(z) = \frac{\sum_{i=0}^{L-1} b_i z^{-i}}{1 + \sum_{m=1}^{M} a_m z^{-m}} \tag{2.13}$$

Infinite impulse response filters have greater similarities to their analog equivalent. The basic thought of providing an ideal filter response and simply use inverse DTFT to acquire the time equivalent filter is simply not feasible since the algebra in the majority of cases is to difficult. The common method is to use prototypes of filters, such as Butterworth prototypes or Chebyshev prototypes. These are analog prototypes and are then converted into digital filters using some transformation from the S-plane for continuous time systems into the discrete Z-plane for discrete time systems.

Chebyshev prototype

These two prototypes both have advantages and disadvantages. The advantage considering Chebyshev filters is a fast rolloff rate thereby giving the filter a narrow transition band, much more narrow than the equivalent Butterworth filter. However the Chebyshev exhibits ripples either in the passband or in the stopband depending on whether it is a Type I or a Type II filter. Another name for Chebyshev filters is equiripple filters because of the fact that these ripples are of equal size[16]. The general transfer function is given by Equation

2.14 and the value of the ripples is controlled by the design parameter ε in Equation 2.15[4].

$$|H(\omega)| = \frac{1}{(1+\varepsilon^2 C_n^2(\omega))^{1/2}} \tag{2.14}$$

$$C_n(\omega) = \begin{cases} cos(ncos^{-1}\omega), & |w| \leq 1 \\ cosh(ncosh^{-1}\omega), & |w| > 1 \end{cases}$$

$$\delta = 1 - \frac{1}{(1+\varepsilon^2)^{1/2}} \tag{2.15}$$

Butterworth prototype

Butterworth filters have no passband ripple and no stopband ripple, and the magnitude response is close to constant at low frequencies, therefore often called "maximally flat"[4][13]. The Butterworth has a better phase response than the Chebyshev prototype, however the transition band is wider. Hence, with respect to the roll-off the Chebyshev performs better[16].

The Butterworth prototype is based on the brick wall ideal filter in Figure 2.8a and has the frequency response shown in Equation 2.16. The transfer function can be created by placing the poles in Equation 2.17 by the use of Equation 2.18. The ω_c constant is the desired cutoff frequency converted from analog to the periodic digital frequency by the use of Equation 2.8[4].

$$|H(\omega)| = \frac{G_0}{\sqrt{1+(w/w_c)^{2n}}} \tag{2.16}$$

$$H(s) = \frac{G_0}{\prod_k^n (s-p_k)/\omega_c} \tag{2.17}$$

$$p_k = \omega_c e^{(j\frac{2k+n-1}{2n}\pi)}, k = 1, 2, ..., n \tag{2.18}$$

Bilinear transformation

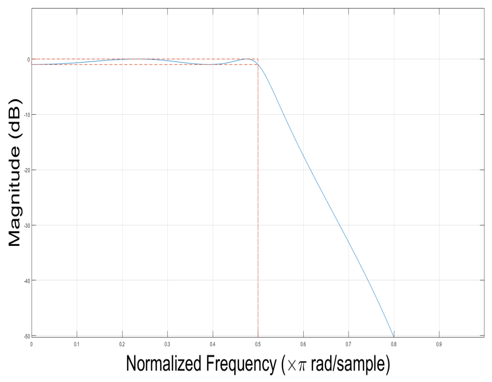
When the coefficients have been computed and the transfer function is complete the filter has to be transformed from an analog filter into a digital filter. This is done by the use of the bilinear transformation. In this method the imaginary axis in the s-plane between $\pm\infty$ is mapped into the unit circle of the z-plane . This is done by the use of Equation 2.19. However one must not forget to also transform the desired digital cutoff frequency ω_d to the analog ω_a with the help of equation 2.20 before applying Equation 2.19[4].

$$s = \frac{2(1-z^{-1})}{\Delta t(1+z^{-1})} \tag{2.19}$$

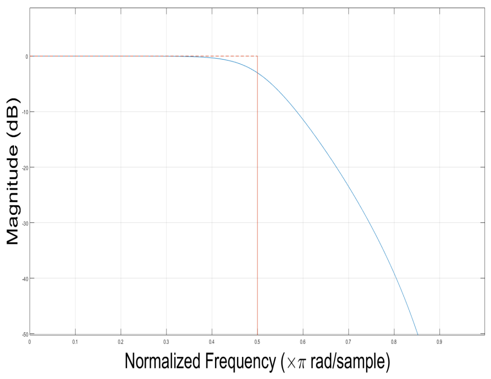
$$w_a = \frac{2}{\Delta t} tan\left(\frac{\omega_d \Delta t}{2}\right) \tag{2.20}$$

Comparison

As mentioned above there is a trade-off between passband ripples and a smaller transition band between the Butterworth and the Chebyshev prototype. Figure 2.11 compares the two methods graphically.



(a) Chebyshev prototype.



(b) Butterworth prototype.

Figure 2.11: The figures illustrate the trade-off between the two of the most common prototypes, both of the 4th order.

Conversion from lowpass to other types of filters

Using the Butterworth or Chebyshev prototype always constructs a lowpass filter. To construct a highpass or bandpass filter one of the following transformations are needed.

For lowpass to highpass:

$$z_{LP}^{-1} = -z^{-1} \tag{2.21}$$

The highpass has cutoff frequency:

$$w_{cHP} = \frac{\omega_s}{2} - \omega_{cLP} \tag{2.22}$$

For lowpass to highpass (ω_{cu} is the upper cutoff frequency and ω_{cl} is the lower):

$$z_{LP}^{-1} = \frac{-z^{-1}(z^{-1} - \alpha)}{(1 - \alpha z^{-1}}, \alpha = \frac{\frac{\pi(\omega_{cu}+\omega_{cl})}{\omega_s}}{\frac{\pi(\omega_{cu}-\omega_{cl})}{\omega_s}} \tag{2.23}$$

The bandpass cutoff frequency:

$$\begin{aligned} \omega_{cLP} &= \omega_{cu} - \omega_{cl} \\ \omega_{sLP} &= \omega_{cLP} + \Delta\omega \\ \Delta\omega &= \omega_{cl} - \omega_{sl} \end{aligned} \tag{2.24}$$

2.4 Discrete Fourier transform

One of the most important data that provides the most information about a signal is the Discrete Fourier transform DFT. The DFT gives information about which different frequencies the signal contains and their relative amplitudes.

The DFT can easily be derived from the analog Fourier transform FT in Equation 2.25 combined with the impulse train sampled signal (x(t) every Δt) seconds. After some mathematical calculations, one can reach the discrete-time Fourier transform in Equation 2.27[4].

$$X(\omega) = \int_{-\infty}^{\infty} x(t)e^{(-j\omega t)}dt \tag{2.25}$$

$$x_c(t) = \sum_{n=-\infty}^{\infty} x(n\Delta t)\delta(t - n\Delta t) \tag{2.26}$$

$$X_c(\omega) = \sum_{n=-\infty}^{\infty} x(n\Delta t)e^{(-j\omega n\Delta t)} \tag{2.27}$$

This transform works well but is not very practical since it involves a summation of an infinite number of samples. However if the data record only contains a finite number of samples an approximation of the DTFT called discrete Fourier transform (DFT) can be used. The DFT can be seen in Equation 2.28. And with the help of Equation 2.29 the definition of the DFT can be reached as of Equation 2.30. The integer k in Equation 2.30 is referred to as the “bin” number and with the use of the sampling frequency this can provide information about the amplitude at a specific frequency[4].

$$\hat{X}_c(\omega) = \sum_{n=0}^{N-1} x(n\Delta t)e^{(-j\omega n\Delta t)} \tag{2.28}$$

$$\Delta\omega = \frac{2\pi}{\Delta t N} \tag{2.29}$$

$$\hat{X}_c(\omega) = \sum_{n=0}^{N-1} x(n)e^{\left(\frac{-jnk2\pi}{N}\right)} \tag{2.30}$$

This algorithm is very inefficient and there is another faster, more efficient algorithm called the fast Fourier transform or FFT. Signal processing is more often than not performed on limited hardware and this DFT is very costly, the N-point sequence depend quadratically on N. A well implemented FFT with N points depends logarithmically on N, hence giving the FFT algorithm the notation $C_N = \mathcal{O}(N \log N)$ instead of the DFT's $C_N = \mathcal{O}(N^2)$[19]. The FFT command in Matlab uses a FFT from the FFTW library[8][9].

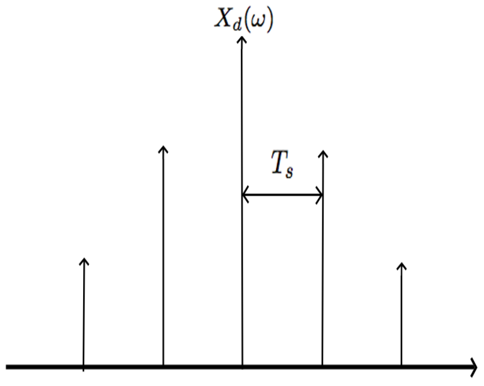
2.5 Digital to analog conversion

To convert a digital signal back to an analog signal is done by the use of a digital to analog converter (DAC). Most commercial DAC's are of type zero-order-hold or ZOH. If the output sampling rate is T seconds, then the ZOH holds the output value for T seconds again. Consult Figure 2.12b and the output in Figure 2.13. The ZOH is described in Equation 2.31 and the operation of converting the signal to analog is by convolving the desired digital signal with this ZOH. The result of this convolution is described in Equation 2.33[16][13].

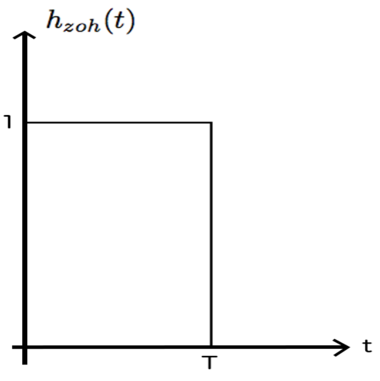
$$h_{ZOH} = \begin{cases} 1, & 0 \leq t < T \\ 0, & otherwise \end{cases} \tag{2.31}$$

$$H_{ZOH}(\omega) = \int_0^T e^{-j\omega t} dt = T sinc\left(\frac{\omega T}{2\pi}\right) e^{-\frac{j\omega T}{2}} \tag{2.32}$$

$$y(t) = h_{ZOH}(t) * y(n\Delta t) = \sum_n y(n\Delta t) h_{ZOH}(t - n\Delta t) \tag{2.33}$$



(a) Digital signal in time.



(b) ZOH in time.

Figure 2.12: The figures illustrate the ideal samples from the signal and the ZOH in time domain.

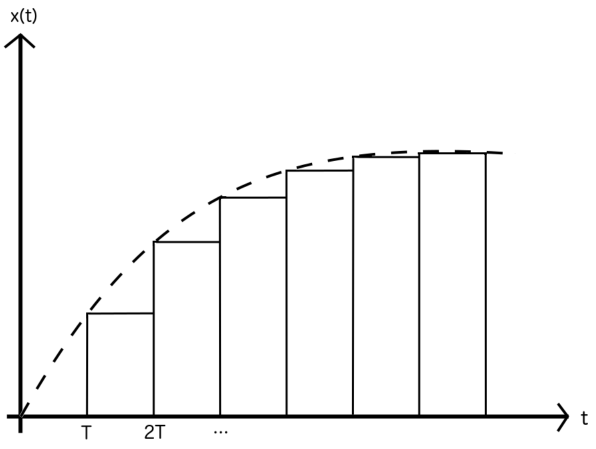


Figure 2.13: The output from the ZOH.

In the frequency domain the spectrum of the sampled signal can be seen in Figure 2.14. This spectrum is multiplied with the transform of the ZOH described in Equation 2.32 and illustrated in Figure 2.15. The ZOH can be seen as a lowpass filter filtering out all but the primary component of the signal. The result can be seen in Figure 2.16. To acquire the DAC's output, this is found by inverse Fourier transform of this primary component ($y(t) = \mathcal{F}^{-1}[Y(\omega)]$)[13]. The ideal ZOH filter has a flat magnitude response and a linear phase. As seen in Figure 2.13, the output of the DAC is not completely smooth. An analog lowpass filter is fitted after the ZOH to smooth out the signal[16].

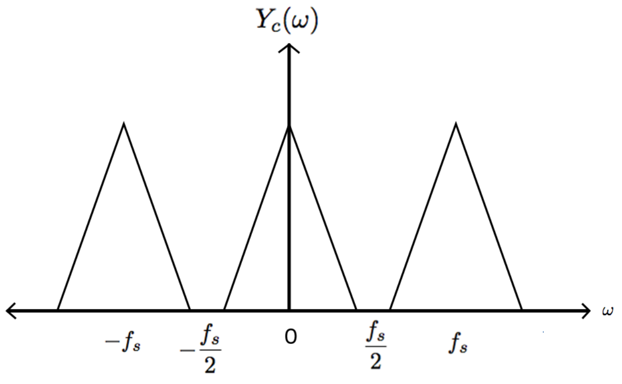


Figure 2.14: Train of spectral components.

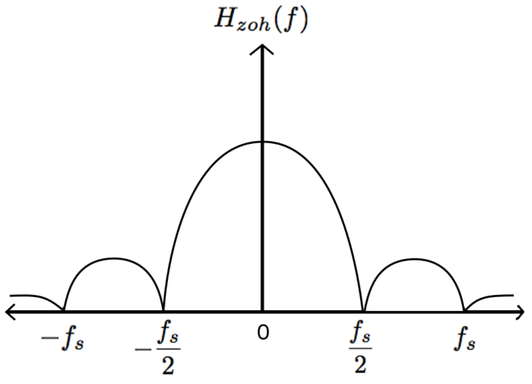


Figure 2.15: Spectra of the ZOH filter.

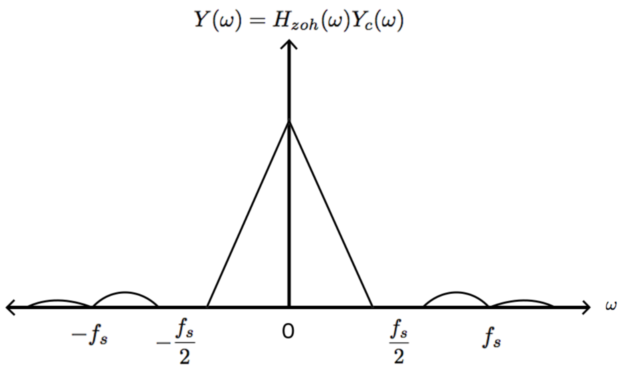


Figure 2.16: Spectral output of the DAC.

Chapter 3

Implementation

3.1 Overview

In this chapter we explain the practical implementation of the system. The chapter is divided into two main parts, first the part of the hardware implementation describes roughly how the computer's sound card works, as well as what has been done to adapt the program to different types of computers.

The second part describes the implementation using Matlab. Section 3.3.1 gives a brief introduction to Matlab, and mentions some of the benefits that was utilized in the program. Section 3.3.2 explains the foundation of the main loop and finally Section 3.3.4 thoroughly explains the graphical user interface.

3.2 Hardware implementation

3.2.1 Sound card

No additional hardware components are needed to run the program. The sampling and reconstruction is all carried out by the sound card of the computer. A sound card typically consist of four major components[20]:

- A DAC.
- An ADC.
- An interface to connect to the motherboard, typically a PCI but could be a USB for external sound cards.
- Input and output connectors such as RCA-inputs and outputs, 3.5mm microphone jacks and headphone outputs.

Depending on the type of computer, the sound card can the range from low end built-in sound cards in the motherboard to advanced external sound cards for enthusiasts and professionals. Typical features of a simple sound card could be a 2.1 stereo sound output, a 5.1 surround sound input together with a 44.1 & 48kHz 16-bit playback and a 44.1 & 48kHz 16-bit recording[1]. In Figure 3.1, a simple block diagram of a sound card can be seen[17].

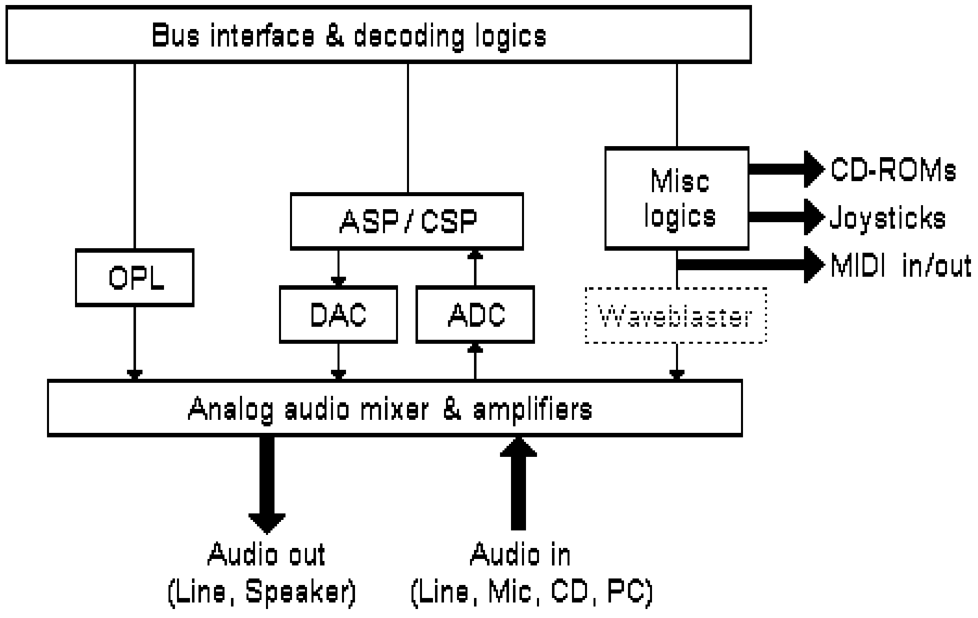

Bus interface & decoding logics
OPL
ASP / CSP
DAC
ADC
Misc logics
CD-ROMs
Joysticks
MIDI in/out
Waveblaster
Analog audio mixer & amplifiers
Audio out (Line, Speaker)
Audio in (Line, Mic, CD, PC)

Figure 3.1: A simplified block diagram of a sound card.

In Figure 3.1 the block ASP/CSP stands for Advanced/Creative Signal processor. Some sound cards have a built in dedicated signal processor that handles some operations instead of the computer's CPU[20].

3.2.2 Adaption to multiple types of computers

Several precautionary measures has been taken to ensure that as many types of computers as possible can handle the program. The most profound effect on performance is to draw the plots. The option "Reduced visual playback" reduces the frame update frequency of the plots by four times. The vectors that contain the samples of each frame are also reduced in size by factor four, reducing the algorithm's computations as described in section 2.2. Some vector's lengths are also limited. Both input and output time vectors are limited to contain information from the past two seconds to reduce in overhead.

Another limitation is that the prefabricated filters are loaded from a separate file once the program starts up. Even though loading from a file might not be the fastest operation, this was found to be faster than creating the filters from scratch every time the program started.

3.3 Software implementation

3.3.1 Matlab

All of the program and all of the code was written in Mathworks Matlab 2015b. Matlab is, in contrast to for example C or C++, a high level language based on matrices. With over 30 years since Matlab started taking shape, it has grown to one of the most used programs for engineering simulations and modelling. Matlab is complemented by Simulink which is a graphical programming language where instead of typing code the user selects boxes and elements representing different operations sorts and draws connections between these. Matlab and Simulink can also be combined to work together. Matlab also includes various toolboxes with functions and objects that can be created to get a project started. The toolboxes are related to various engineering fields such as aerospace, electronics, control theory and many more. This makes Matlab one of the most popular software programs for engineers and scientists[12][18].

In this project the many functions of sampling from a file and the mic, filtering and reconstructing the signal, and the DSP toolbox were used. The DSP toolbox provides analyzing tools such as Time Scope objects, Spectrum Analyzer objects and filter constructing algorithms. Only a small part of these functions are used in the finished project, but many of them were used in the development so that the focus could be put onto one problem at a time[5].

3.3.2 Generation of code

To construct the program, various tools in Matlab have been used. To create the Graphical User Interface (GUI), the Matlab environment "Guide" has been used. This is a tool to help create GUI:s more creatively by instead of typing code specifying where to put buttons and axes, they can be dragged and placed on a canvas which decreases the time and effort tremendously when working with a complicated GUI. The main GUI's construction in "Guide" is shown in Figure 3.2.

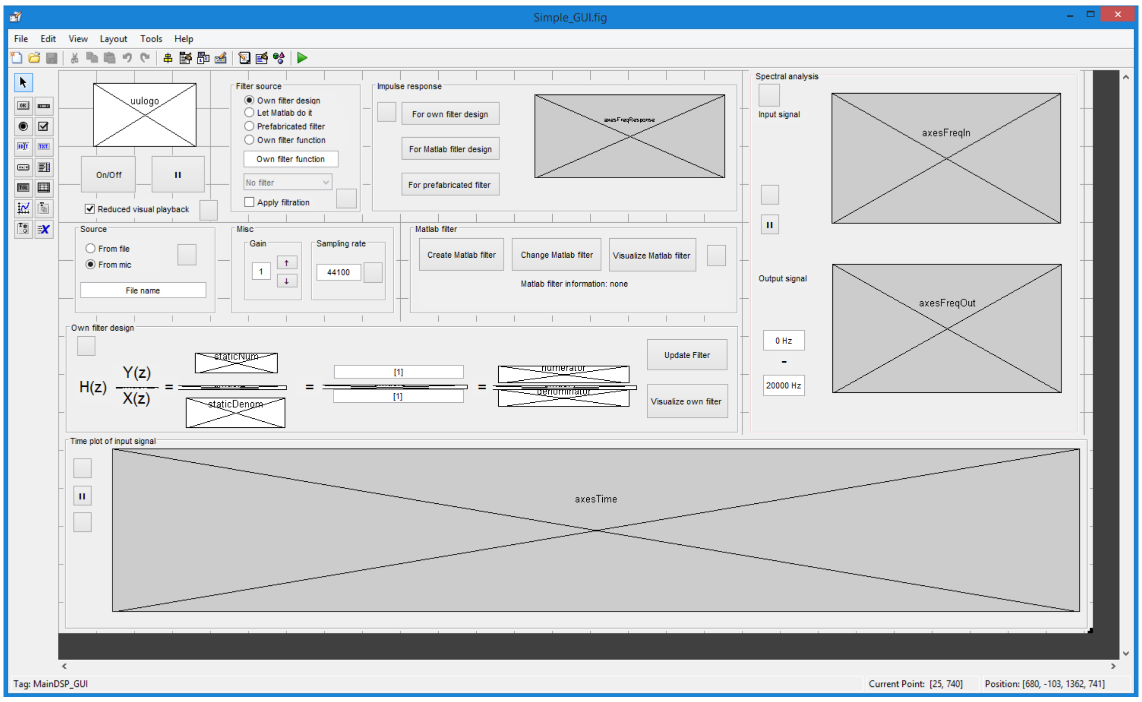


Figure 3.2: The main GUI in the creative setting "Guide".

The base of the program is the main loop. A flowchart of the main loop can be seen in Figure 3.3. The simplified version of the structure of the loop can be seen in Figure 3.4. This pseudocode describes the test bench upon which the main loop is built. The program clearly has three phases. The program is first initialized, the "Audiorecorder" object is created, variables are loaded as well as memory for vectors and arrays is preallocated. Then, in the main loop, the sound is sampled and divided into frames. At the end of the loop there is usually either some plots created or some sound is sent to tha audio output.

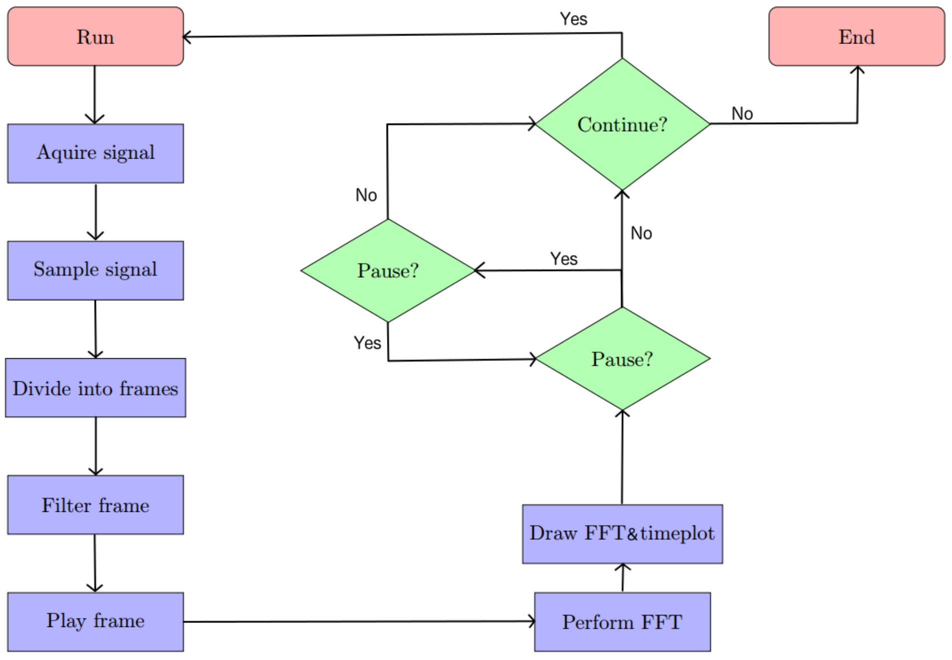


Figure 3.3: The loop that runs while the On/Off toggle button is pushed.

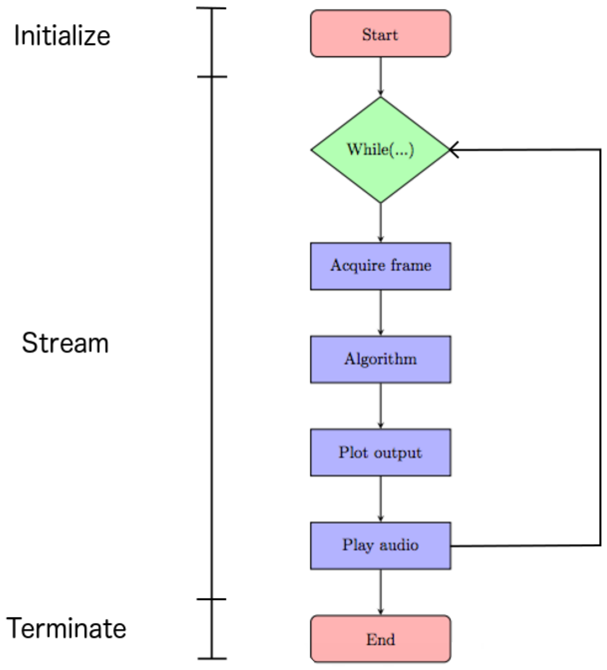


Figure 3.4: Pseudocode of the base of a testbench, the foundation of the main loop.

The middle of the loop is where the signal is processed, either by filtering or by computing the FFT is performed. At the end of the program there is a terminate phase, where in this case the “Audiorecorder” object is released.

3.3.3 Filter design algorithms

Since this program was used for educational purposes there should be different ways to construct and filter the audio. The different ways are listed below.

- The main way of constructing a filter is by typing in the coefficients of the filter a_n and b_n in the text boxes of the "Own filter design"-control panel.
- Create a Matlab filter in the "Matlab filter"-control panel.
- Create a Matlab function (see Section 3.3.4) and typing the name of the function in the "Own filter function"-text box in the "Filter source" control panel.

When it comes to designing the filter for the "Own filter design", it is up to the user to find the coefficients. Matlab then uses the function "y = filter(b,a,x)" to calculate the output y from input signal x with the help of coefficients b and a.

"Matlab filter" uses the "Filter Design assistant" to calculate a filter object which is also filtering with the help of "y = filter(filterobject, x)". This provided tool Matlab includes design methods such as Equiripple, Kaiser window, Butterworth, Chebyshev Type I and Type II and Elliptic.

The prefabricated filters have various design methods. The basic FIR and IIR filter types have been created with the help of Matlab's "filterbuilder", which is slightly more complicated than "Filter Design Assistant".

3.3.4 Graphical user interface

The most important part of the program is the graphical user interface. The main GUI can be seen in Figure 4.1, and the other two can be seen in Figure 4.2 and 4.3. In this section the different parts of the GUI will be reviewed to give an idea of how the program is used, in complement to the flowchart in Figure 1.1.

All of these separate control panels have an information button, which open a separate help dialog with a small text describing the functionality of the control panel as well as some of the limitations.

On/Off

The On/Off control unit consists of three main interaction sources. There is the On/Off toggle button which starts the main loop described by the flowchart in Figure 3.3. There is also a pause toggle button which can be used at any point to pause and restart. The pause button activates a spin loop in the main loop. While the pause button is pushed the user could enter either the magnified timeplot GUI or the magnified FFT plot GUI. Third there is the Reduced visual playback tick box which is ticked by default. This is described in more detail in Section 3.2.2.

Source

There is the option whether to sample from a file or whether to sample from the mic input of the computer. Both of them use the same algorithm and both of them process the music in real time. The mic input is the default and the file input is merely thought of as a complement to increase the flexibility of the program. The user chooses how to

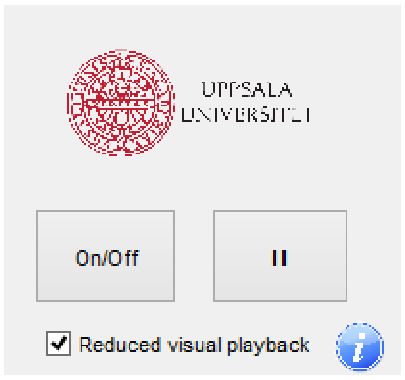


Figure 3.5: The main GUI's on/off control unit.

sample the music and if the file input is choosen, the file name has to be entered as text in the "File name"-text box. The entire file name has to be entered, including the file type extension.

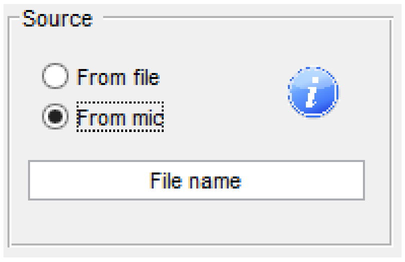


Figure 3.6: The source control panel.

Miscellaneous

Setting the sampling rate is only an option while the filter source is set to mic input. When the main loop is running, the sampling rate text box is blocked from usage but the user can still read out which sampling rate is used. It is the file's sampling rate that sets the sampling rate at which the program samples the sound. The default is 44100 Hz. The gain works in a similar manner as a volume knob. By pressing the arrows, the gain multiplier can be increased or decreased. The user can also choose to enter a non-integer value by typing the value into the textbox.

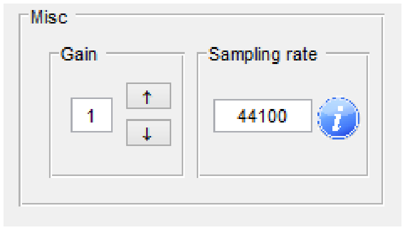


Figure 3.7: The misc control panel.

Own filter design

Since both IIR filters and FIR filters can be expressed as a fraction (see Section 2.3.1 and 2.3.2) the "Own filter design" panel is supposed to be self explanatory. The coefficients of the filter are entered in the text box separated by a space (ascii 32). This can be done at any time. However, the main loop has to be off to press the push button "Update filter". By pressing the "Update filter"-button the coefficients are saved as the "Own filter design" object, making it possible to use the filter to filter the sound. The idea is that the student has constructed a filter on paper and wants to apply it in a real world application. The "Own filter design" panel lets the student apply the filter easily on some signal. The "Visualize own filter" is another tool to make the student understand the properties of the filter (s)he has constructed by opening up the built in Matlab GUI "Filter Visualization Tool". In this environment the student can acquire information about the magnitude response, phase response and other important information about the structure, stability and type.

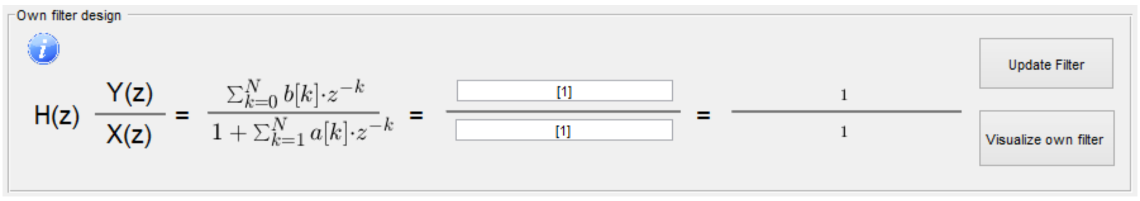

Own filter design
$H(z)\ \frac{Y(z)}{X(z)} = \frac{\Sigma_{k=0}^{N} b[k]\cdot z^{-k}}{1+\Sigma_{k=1}^{N} a[k]\cdot z^{-k}} = \frac{[1]}{[1]} = \frac{1}{1}$
Update Filter
Visualize own filter

Figure 3.8: The own filter design management control panel.

Matlab filter

When students will construct filters in the future, there is a great chance that companies or others will have prefabricated filter algorithms and programs for this purpose. Also, when discussing filter properties with coworkers and fellow students it is important that the terminology is correct. The "Matlab filter" panel lets the student construct a filter and apply it without the tedious works of calculating the coefficients by hand. The "Create Matlab filter"-push button allows the user to create a filter using Matlab's "Filter design assistant" by specifying parameters to meet requirements. This gives an incentive to learn the terminology used in filter specifications as well as common construction algorithm names. The "Change Matlab filter"-push button lets the user change the parameters of the already constructed filter and "Visualize Matlab filter" opens up the "Filter Visualization Tool" to acquire the information already mentioned in Section 3.3.4. The text below gives some simple information about the filter, but the text string mostly aims to indicate whether a filter has already been created or not.

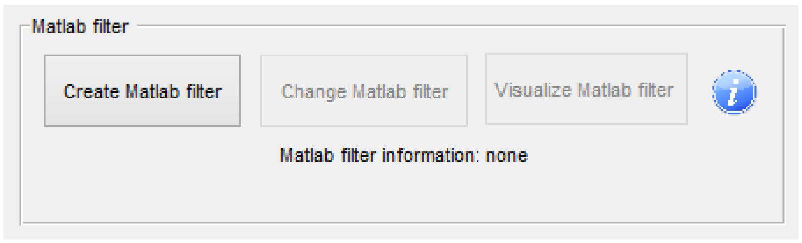


Figure 3.9: The matlab filter design panel.

Filter source

The filter source control panel allows the user to choose the filter source. The options are:

- Own filter design

- Matlab filter ("Let Matlab do it")
- Prefabricated filter
- Own filter function

"Own filter design" and "Let Matlab do it" uses the filter methods described in Section 3.3.4. "Prefabricated filter" lets the user choose between a selection of filters from a pop-up menu, some FIR filters, some IIR filters and some simpler filters with sound effects. "Own filter function" lets the user construct a function script in Matlab which can receive an input vector, perform filtering or whatever the user wishes to do with the samples, and return a vector of the same size. This option is suitable if the user wishes to use more complicated filters. This invites the user to play and create filters with sound effects, and cascaded filters with endless possibilities. The only criterion to create such a filter is as mentioned, that the size and shape are the same for input and ouput signal, and the function header must be as described in Equation 3.1 with a header that handles an indexation variable. The reason for this is that many more advanced filter such as a flanger for instance has a periodical feature to create these sound effects. When Matlab divides the samples into frames, these frames represent such a short period of time in a song that these periodical features do not have time to create an effect. A flanger has a periodical feature with a period of around 20-40ms[10], a frame of 1024 samples sampled at 44100Hz represents a time from $t = 0$ seconds to $t = \frac{1}{44100} \cdot 1024 = 0.232$ seconds. The flanger then needs to keep track of its own periodicity when the next frame arrives so that the effect can continue without interruptions that would worsen the sound.

The filtering method can be changed anytime while the program is running, even if the main loop is activated. To activate filtering, the tick box "Apply filter" needs to be ticked.

$$function\ y = testfilterfunction(audioIn, index) \tag{3.1}$$

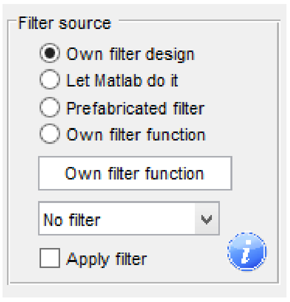


Figure 3.10: Filter source control panel.

Impulse response

The user can plot a simple plot in the axes inside the "Impulse response" panel. The interface is very limited and there are no possibilities for interaction. The panel is merely there to give the user a rough idea of how the impulse response of the filter might look like. The reason for this is that in Matlab's "Filter Visualization Tool" there already are possibilities of acquiring this information.

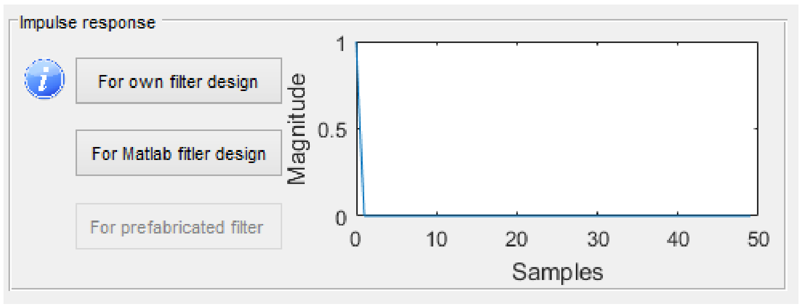


Figure 3.11: Impulse response response panel with axes.

Spectral analysis

The spectral analysis control panel is designed to communicate to the user the sufficient information needed to be able to know what type of filter is needed. The top axes plots the FFT of the input signal and the bottom the output signal. It is possible to change the visible range. However only the x-axis can be changed and only while pause is activated. So either the program has to be paused or the "Spectral analysis" pause toggle button needs to be pressed. If the user needs more information, while pause is active (s)he can press the magnifying glass, opening up the spectral analysis's GUI. The GUI is seen in Figure 4.2. With this interface open the user can change both the limits of the x-axis and the y-axis to have a closer look. However, only information from the present frame is available.

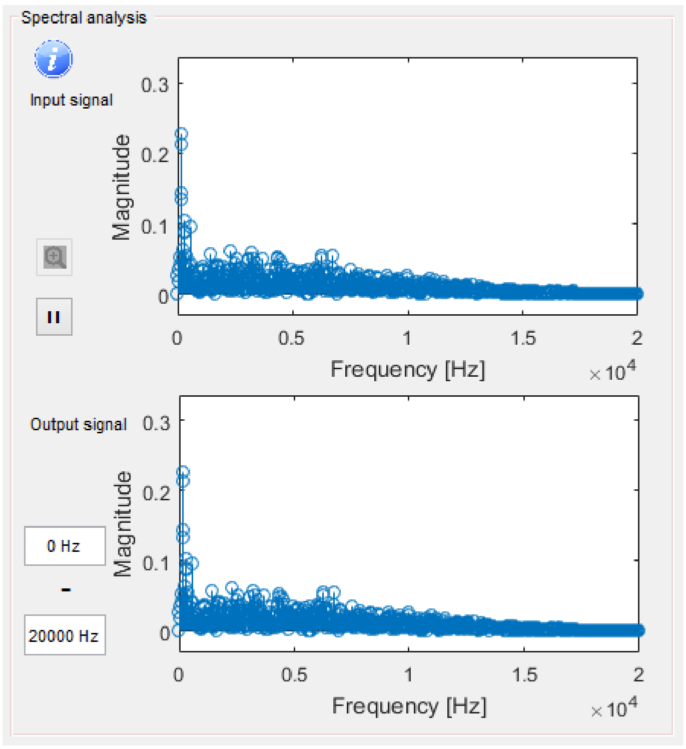


Figure 3.12: The spectral analysis control panel.

Time plot of input signal

The “Time plot”-panel works in a similar manner as the “Spectral analysis”-panel in Section 3.3.4. Though, in the main GUI only the input signal is available to look at. To be able to see the time plot of the output signal, the user has to use the magnifying glass push button to open the “Time plot”-GUI. This GUI looks very similar to the “Spectral analysis”-GUI, with possibilities to change the limits of both the x-axis and the y-axis, though only information from the past 2 seconds is available as described in Section 3.2.2.

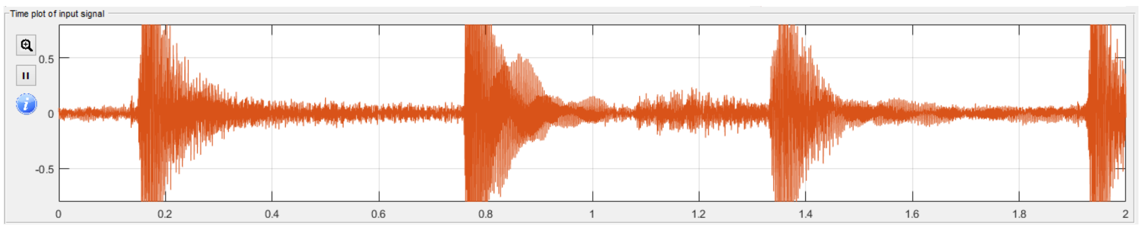


Figure 3.13: Time plot axes with controls.

Chapter 4

Results

4.1 Software

The main program's graphical user interface can be seen in Figure 4.1. On top of the main GUI, there are also two more simple GUIs that can be opened by pressing the magnifying glass next to some of the plots while the program is paused.

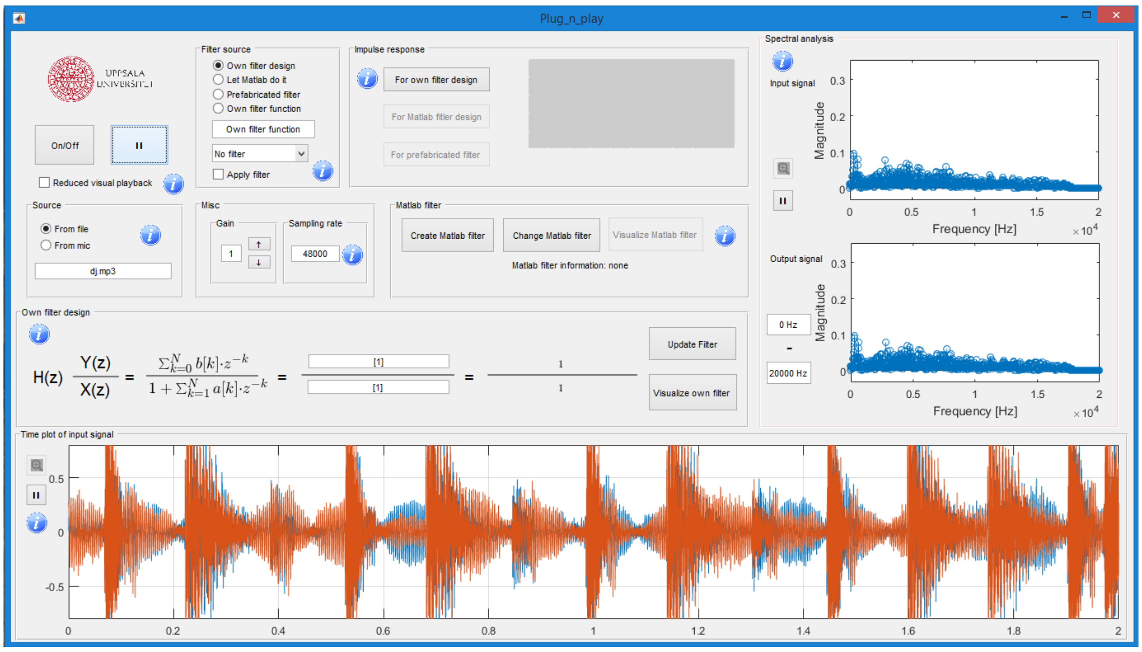


Figure 4.1: The main program's graphical user interface.

One problem that was encountered was the way that variables were passed between functions in the program. Matlab is based on Java but passing variables inside the GUI is different from having instance variables connected to objects. In this program, variables are passed by the use of the handles-structure, see Appendix B.1. This was also a problem when it came to filtering with the prefabricated filters due to passing the filters both fast and elegantly. Constructing six filters in the startup of the program would not be ideal either, this would at least double the startup time. These filters are preconstructed and loaded from a separate file instead.

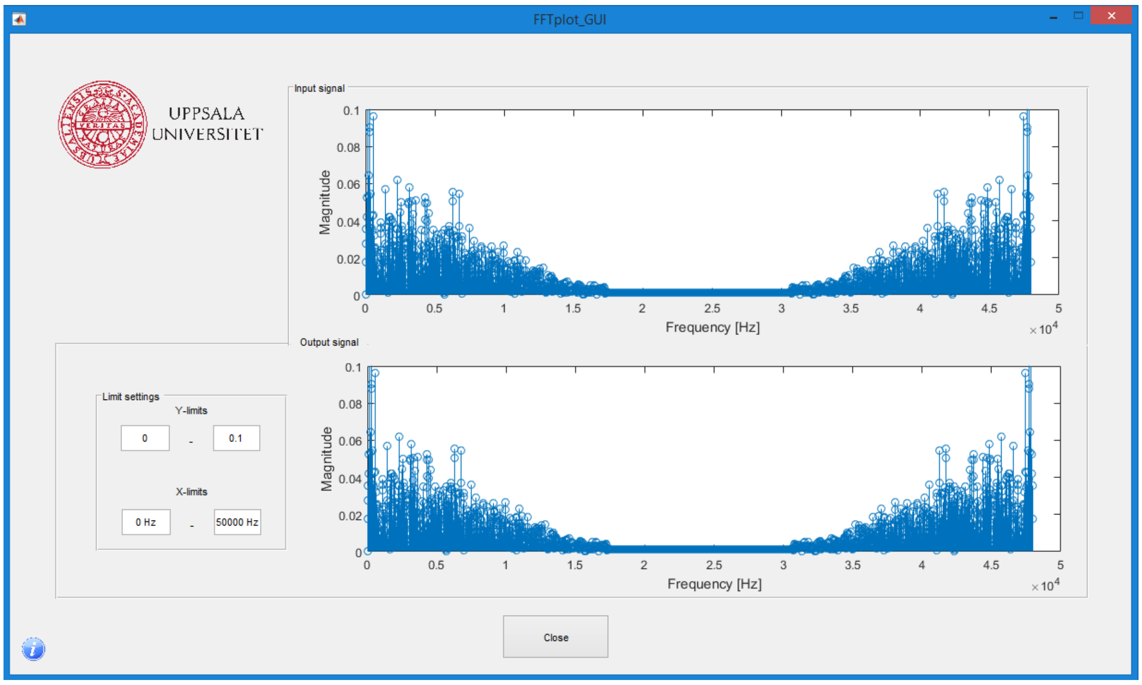


Figure 4.2: The GUI of the magnified FFT plot.

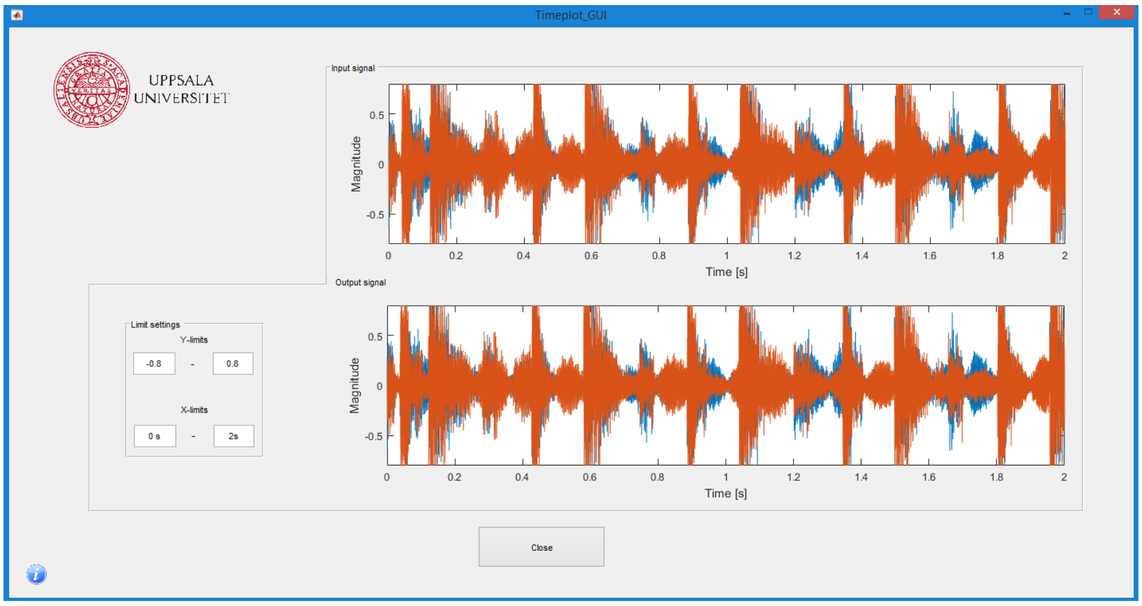


Figure 4.3: The GUI of the magnified time plot.

4.2 Test runs

4.2.1 Two sinuses

To test the program, another device was connected to the 3.5mm jack into the mic input of the computer. This device was another computer, which produced a sound containing two sinusoids of 1000 Hz and 10000 Hz respectively. The signal can be seen in Figure 4.4, with the spectrum in Figure 4.5. Since there are three major ways to filter the sound (four ways to apply own specifications) as described in Section 3.3.4, these will be tested each by two filters, one simple second order IIR filter and one more advanced FIR filter.

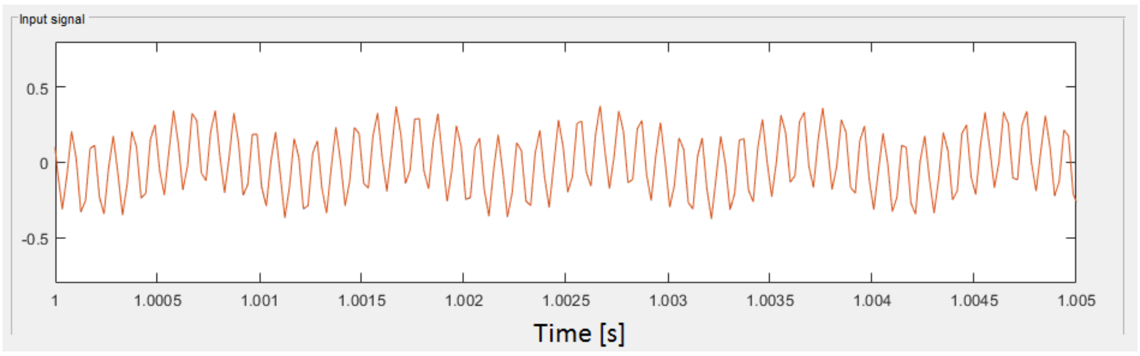


Figure 4.4: A zoomed image of the signal.

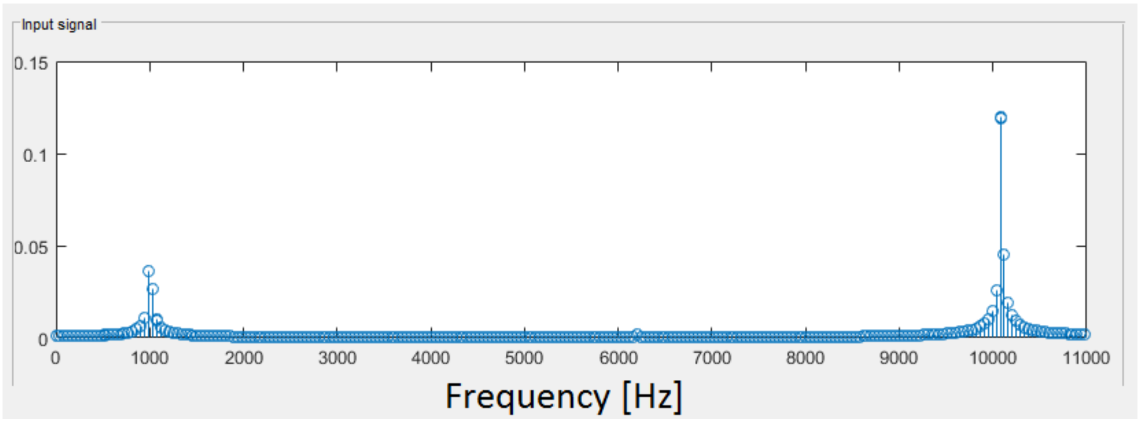


Figure 4.5: Spectra of the input signal.

Filtering by "Matlab filter" control panel

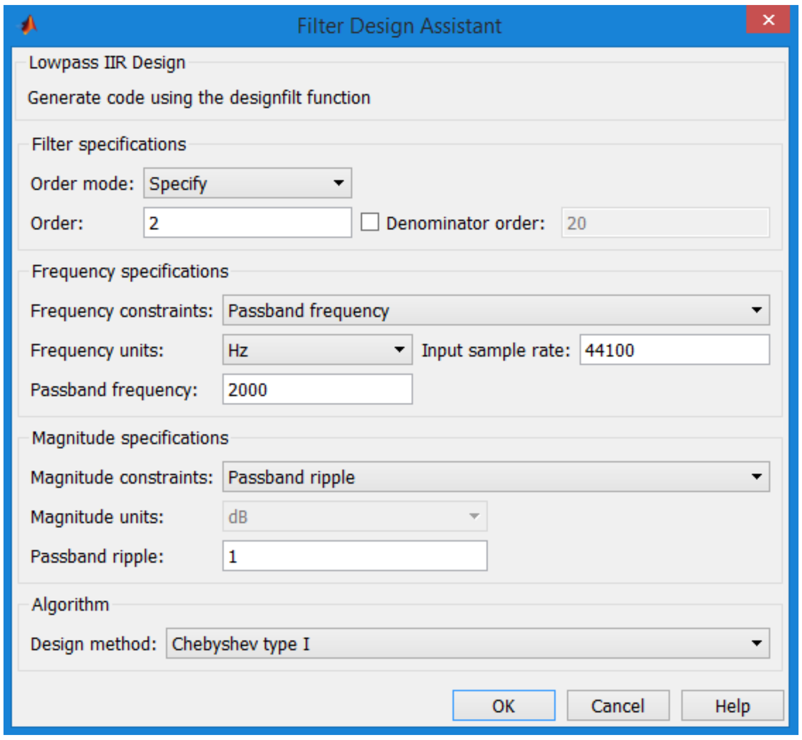
The 1000Hz sinus is the signal of interest, while the 10000Hz signal is noise. First a Matlab filter is created using the Matlab filter control panel, see Section 3.3.4. The specifications of Figure 4.6 are selected. The filter is visualized by pressing the "Visualize Matlab filter"-button, the result can be seen in Figure 4.7. The impulse response can also be seen by pressing the "For own filter design"-button in the "Impulse response"-panel.

Filtering by "Own filter design" control panel

The coefficients were now copied by creating the exact same filter in Matlab's command window, and pasted in the "Own filter design" control panel. Pressing the "Update filter"-button gave the following result shown in Figure 4.8. Pressing the "Visualize own filter"-button gave the magnitude and phase response shown in Figure 4.9. The impulse response looks similar, though the x-axis has a different scaling.

Filtering by "Own filter function"

The coefficients were also copied into a separate Matlab function. And the function was written so that it could be applied in the program according to Section 3.3.4. When using a separate function, there is not any built in interface to confirm that the filter is correct so that it is up to the user to check. This function can be seen in Appendix C.1.


Filter Design Assistant

Lowpass IIR Design

Generate code using the designfilt function

Filter specifications

Order mode: Specify

Order: 2

Denominator order: 20

Frequency specifications

Frequency constraints: Passband frequency

Frequency units: Hz

Input sample rate: 44100

Passband frequency: 2000

Magnitude specifications

Magnitude constraints: Passband ripple

Magnitude units: dB

Passband ripple: 1

Algorithm

Design method: Chebyshev type I

Figure 4.6: The specifications entered in “Filter Design Assistant”

Results from filtering the two sinus signals

The results from filtering using the “Own filter design” control panel is presented in Figure 4.10 and 4.11.

The results from using the “Own filter design” control panel and using the “Own filter function” produced the exact same result as in Figure 4.10 and 4.11.

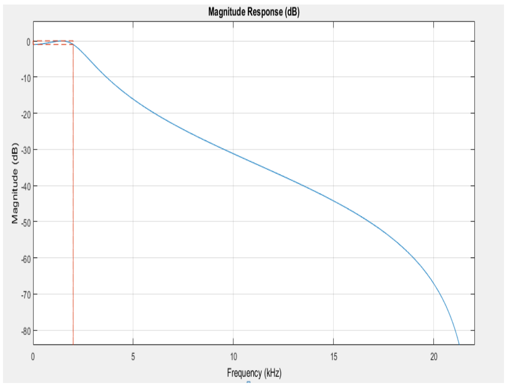
4.2.2 Filtering Bach - Orchestral Suite No. 3 (Air)

This is a 30 second segment of Bach with a single frequency noise that needs to be deleted and random noise. The signal is shown below in Figure 4.12 together with the spectrum of the signal in Figure 4.13.

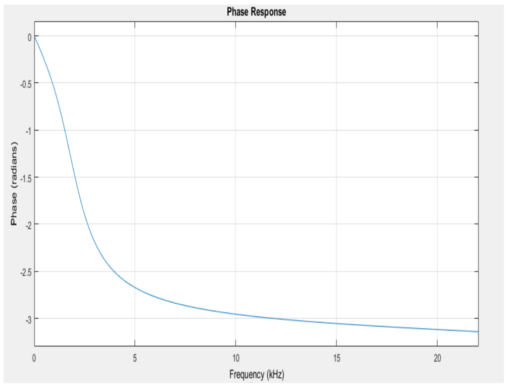
Figure 4.13 shows that the music has a tone placed at 2500Hz that needs filtering but also lots of noise from about 6000Hz. For such a delicate piece of music as little disturbance at other frequencies as possible is desired. A IIR filter with as little phase shift as possible in the other frequencies was calculated in the “Matlab filter” control panel. This filter is shown in Figure 4.14. To remove the higher frequencies the music is filtered by a lowpass FIR filter with a passband frequency of 6000Hz and a stopband frequency of 6700Hz, this is shown in Figure 4.15. A cascaded filter can only be built as a “Own filter function”. This filter function can be found in Appendix C.2.

As can be seen in Figure 4.14b the phase response is minimal at most frequencies except at around the filtered frequencies for the IIR bandsstop filter. Also the FIR filter has linear phase, the result of these cascaded filters should have minimum phase distortion. The result by applying the filter is shown below.

One can clearly see that most of the disturbances are gone in the output. However there

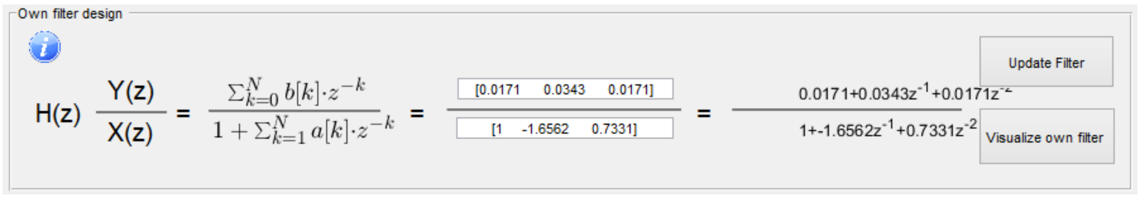


(a)



(b)

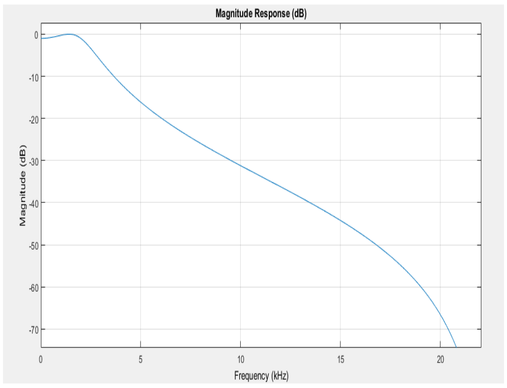
Figure 4.7: Magnitude and phase of IIR filter.



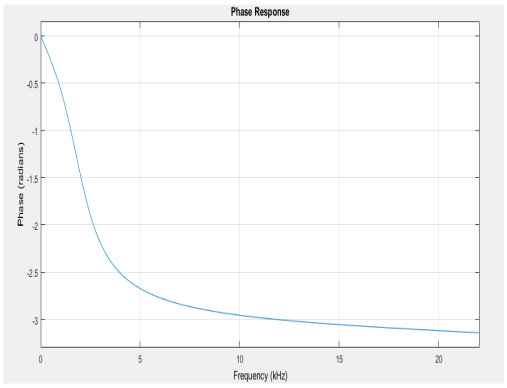
Own filter design

$$H(z)\ \frac{Y(z)}{X(z)} = \frac{\Sigma_{k=0}^{N} b[k]\cdot z^{-k}}{1+\Sigma_{k=1}^{N} a[k]\cdot z^{-k}} = \frac{[0.0171\quad 0.0343\quad 0.0171]}{[1\quad -1.6562\quad 0.7331]} = \frac{0.0171+0.0343z^{-1}+0.0171z^{-2}}{1+-1.6562z^{-1}+0.7331z^{-2}}$$

Figure 4.8: The resulting transfer function from entering the coefficients.



(a)



(b)

Figure 4.9: Magnitude and phase of the IIR filter built by entering coefficients.

are still some left, a warning sign is that the output time plot looks disturbed. The sound is not perfect either.

4.3 Realtime

All processing works in real time. A song can be played on the mobile phone and filtered using a computer having Matlab installed. Slower computers might struggle to keep up in real time if the “Reduced visual playback” tick box is not ticked.

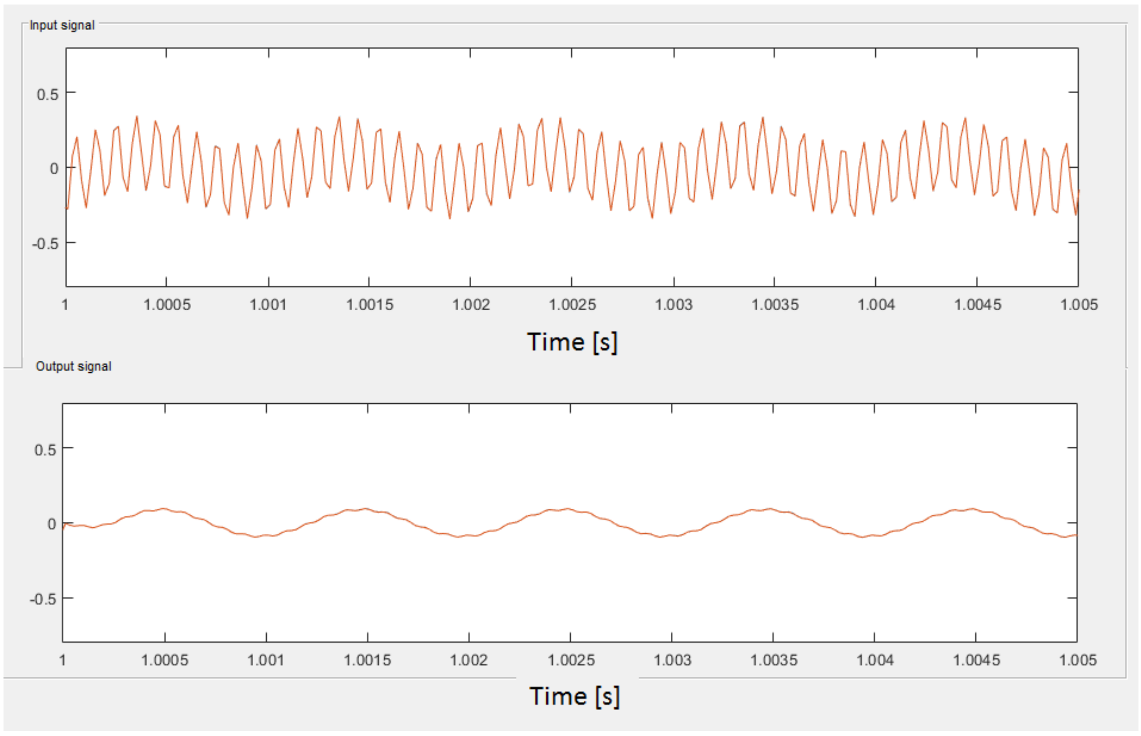


Figure 4.10: The input and output signal in time from the "Matlab filter".

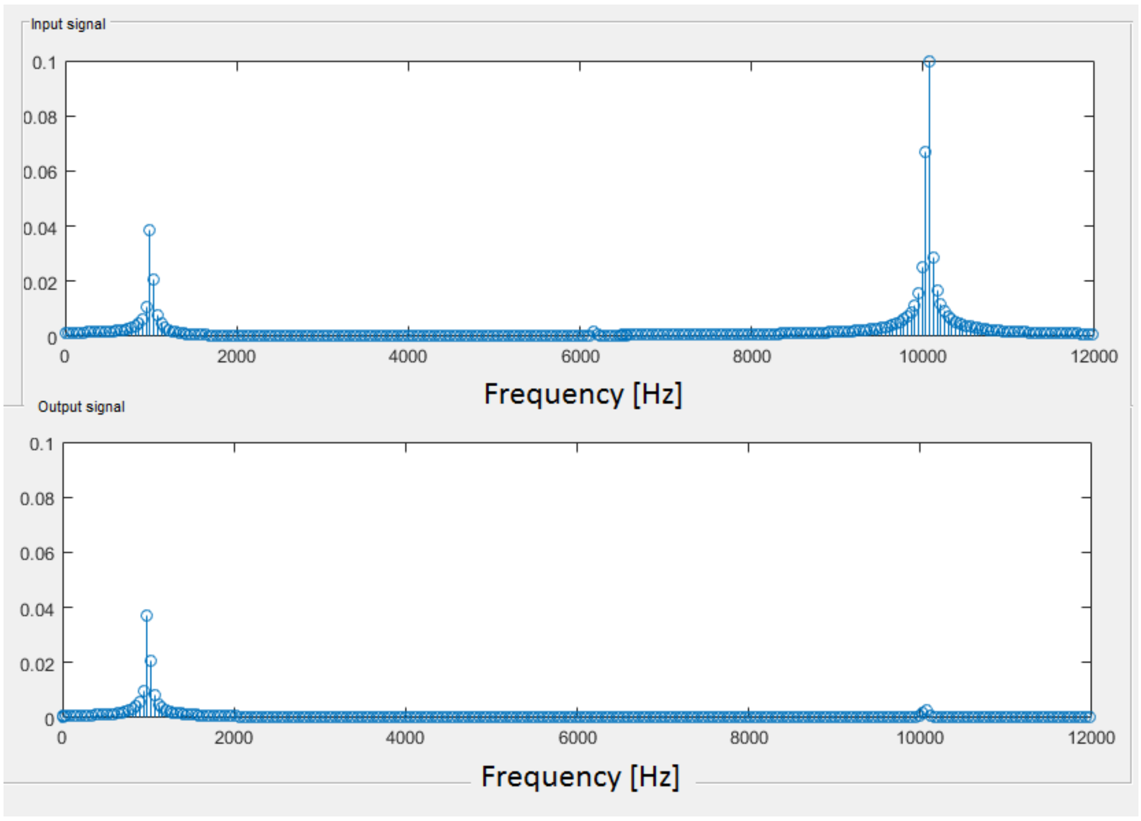


Figure 4.11: The input and output spectra from the "Matlab filter".

4.4 Adaption to multiple computer systems

The adaptions that have been made to include slower computer systems work well. Two test computers have been used, a desktop with a Intel Core i5-4570 processor clocked at

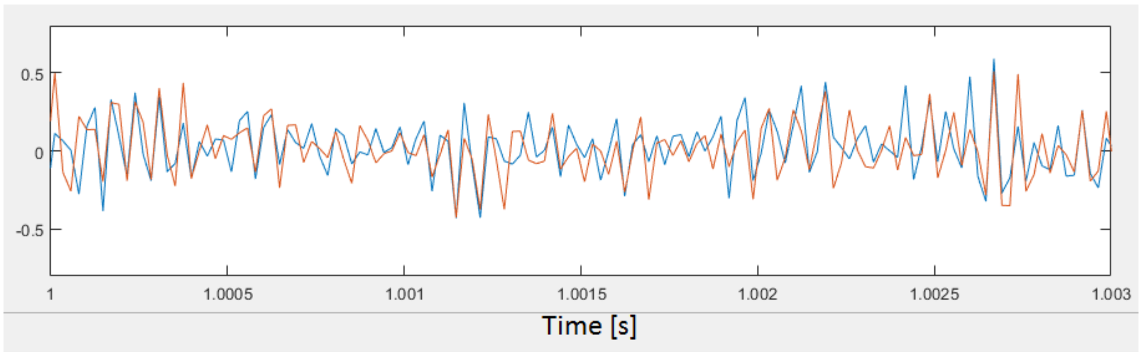


Figure 4.12: The noisy piece of music in time. Blue and red represents channel one and channel two.

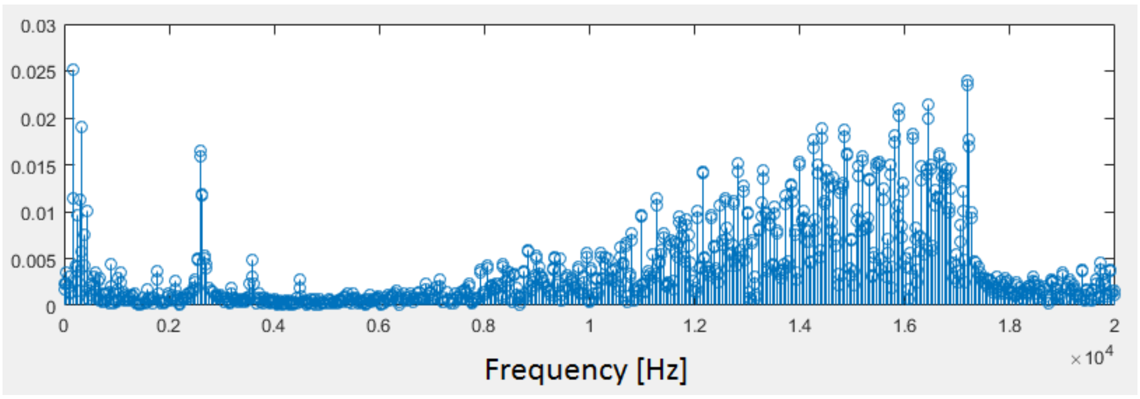
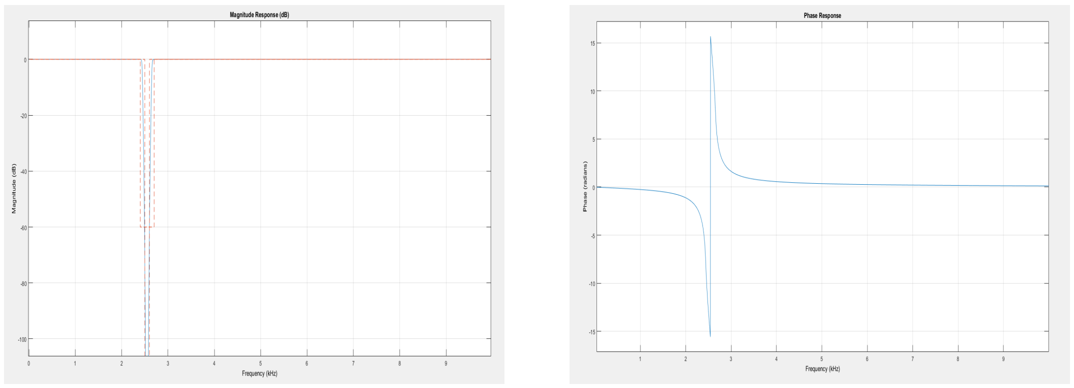


Figure 4.13: The spectra of the noisy piece of music.

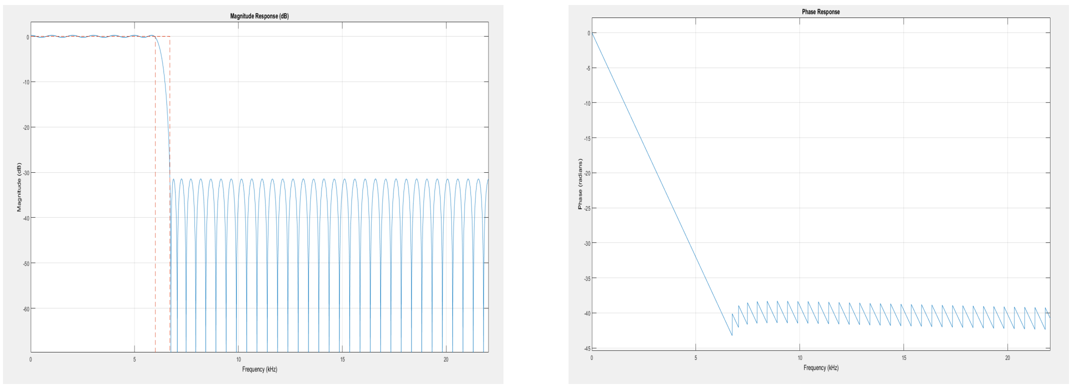


(a) (b)

Figure 4.14: Magnitude and phase of the IIR filter to filter out the single 2500Hz signal.

3.2GHz with a Nvidea GeForce GTX 760 graphics card and the other which is a laptop with a Intel Core i5-3337U at 1.8GHz and a built in graphics card.

Note: Both the sound and the plots lag when playing without "Reduced visual playback" on the laptop.



(a) (b)

Figure 4.15: Magnitude and phase of the FIR filter to filter out the high frequency noise.

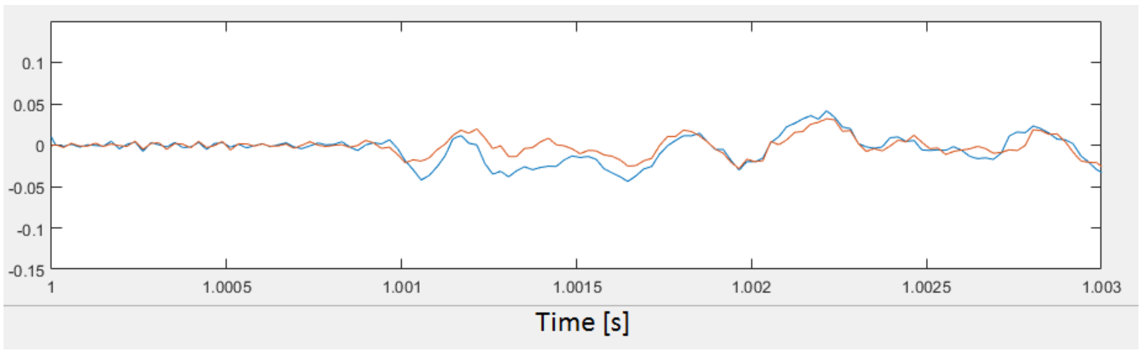


Figure 4.16: The filtered piece of music in time. Blue and red represents channel one and channel two.

Computer:	Desktop	Laptop
Startup:	2.6043 s	16.6710 s
Start loop:	0.0448 s	0.1800 s
Average loop turn(Reduced):	0.0198 s	0.0202 s
Average loop turn(Full):	0.0191 s	0.0691 s

Table 4.1: A simple comparison between a more powerful stationary computer and a less powerful somewhat typical laptop running the program.

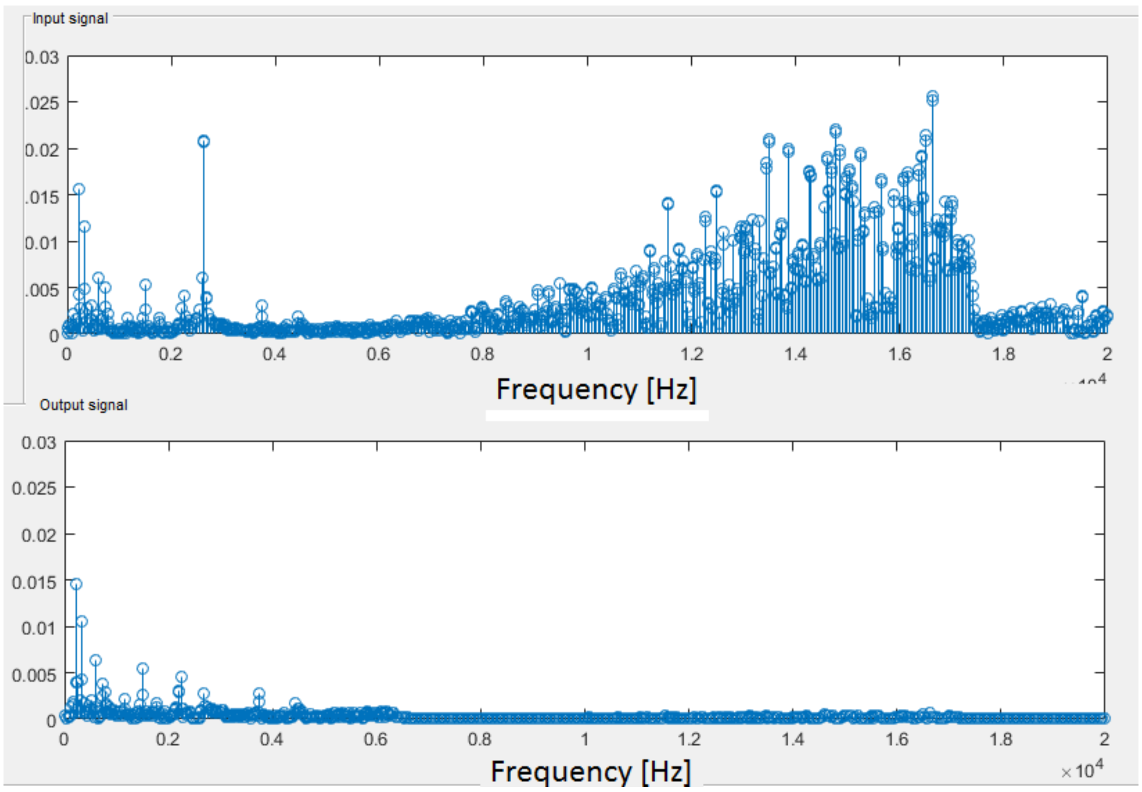


Figure 4.17: The spectra of the filtered piece of music.

Chapter 5

Discussion

The aim of the project was to create a program that can sample sound from a mic input, perform DSP on the signal and then play it using the computer's headphone outlet. This is a feature that works very well in the program. With the help of Matlab's DSP toolbox, the sampling and reconstruction of the signal is near perfect and when listening one cannot hear that the music has been sampled and reconstructed an additional time.

The result when filtering the classical piece by Bach in Section 4.2.2 was not perfect. The output sound still suffered from some noise and there was some phase distortion from the filtering even though a minimum phase IIR filter were used as well as a linear phase FIR filter. The problem were not the filtering but probably that the filters was not designed perfectly.

To use a high level language like Matlab for these types of operations has both advantages and disadvantages. The drawback is that Matlab is computationally expensive, and there are limitations for the user to influence how core functions operate. On the other hand, many operations are well adapted. The advantages are that it is easy to get going. This program is easy to start up with the help of the DSP toolbox and the simple syntax of the program. Another advantage is that Matlab is a familiar environment for the students. Many homework tasks are already done in Matlab and letting students use and familiarize themselves with a more complete program may help them get over the threshold more easily when it comes to constructing their own programs and algorithms in the future.

Another aim was to make the program somewhat pedagogic. The program has not been tested and the layout and design has not been based on any scientific studies that would have affected the design. However, the design was chosen by a student based on a personal experience when recently taking the course. The design has been implemented to be as self explanatory as possible, an example of this is the “Own filter design” control panel described in Section 3.3.4 where the transfer function is printed on the screen explicitly to avoid confusion. The fact that the design is not based on any scientific studies must however be stressed.

The adaption to multiple computer systems worked well on the two computers that the program was tested on. With the help of the “Reduced visual playback”-button described in Section 3.3.4 even the slower computers can run the program. The reduced update frequency is however somewhat noticeable.

Chapter 6

Further development

The program in its present form shows a lot of potential but could probably be improved even more after it has been tested in the field. Some improvements for further development are listed below. Either the direction of further development could be taken in a softer direction, concentrating on the student with more pedagogical and educational tools built in. Or the program could be taken to a more advanced level, with more advanced filter construction methods.

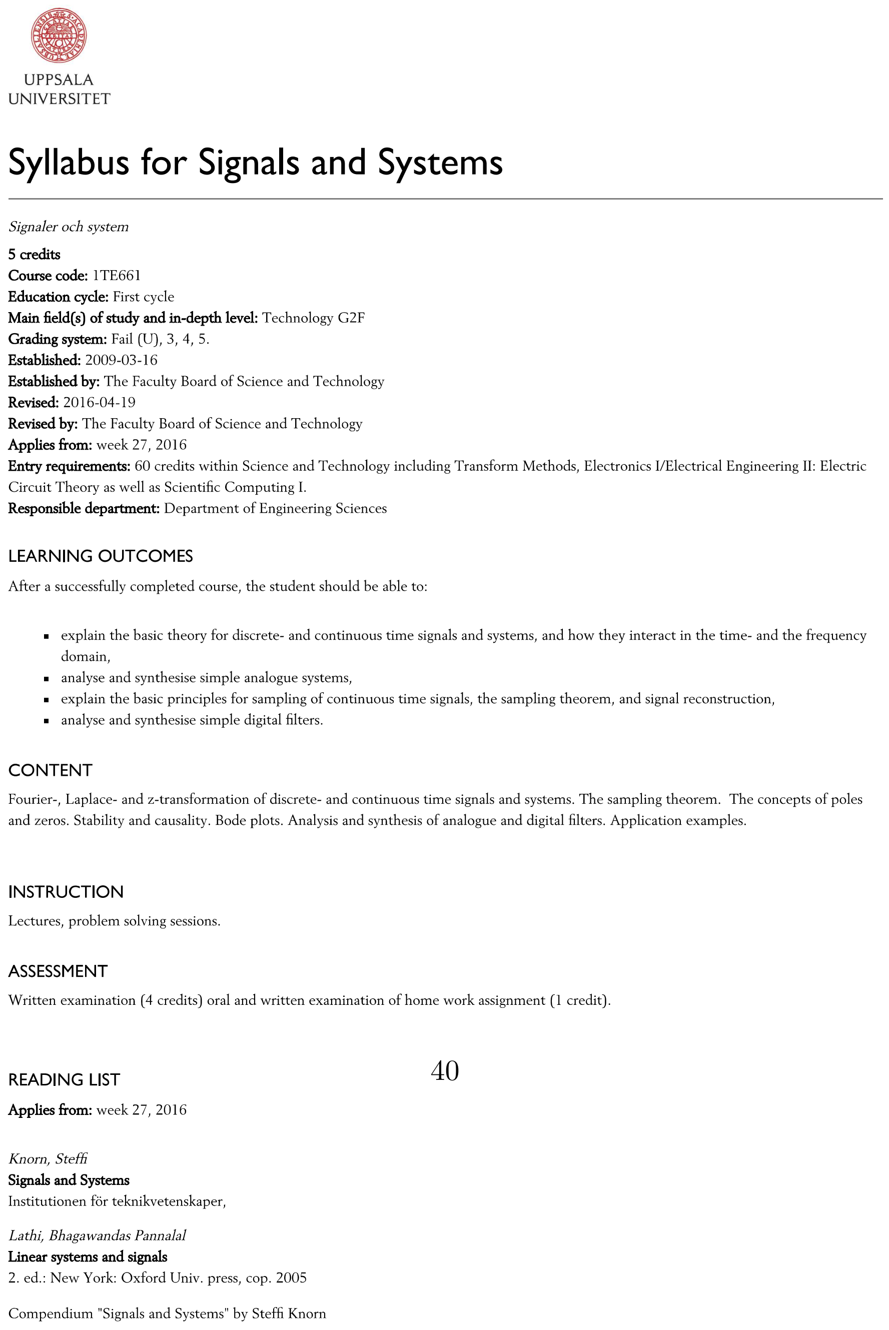
- The GUI could be redesigned based on scientific reports on how to reach and involve students.
- More tools for construction filters and more involving implementation algorithms could be considered.
  - For example when constructing filters, the student could manually place the poles and zeros in the z-plane with a plot visualizing the filters amplitude and phase response so that the student can see which changes gives which results.
  - Separate design GUI:s for IIR and FIR filters. The IIR design GUI could plot all the poles and show the equation for when the user tries to generate a Butterworth prototype. The FIR filter environment could show the equation for each step when using the window method.
  - Constructing filters for musical purposes is difficult. Looking at interactive music design programs for DJing and music production is very easy. Creating a GUI that can somehow combine the scientific process together with the creative process could be great for learning.
- The program could include more than just filtering with IIR and FIR filters from the digital signal processing world. More advanced filtering and random signal processing could also be implemented.
- Better preconstructed filters to give better examples.
- The program could be rewritten and reimplemented in a lower level programming language. This would increase the speed and decrease the processing power dramatically.
- The educational tools could be deleted and substituted by simpler yet powerful tools to create filters easily.
- A helping wizard for constructing more advanced filters could be implemented. A

tool that would make it easier to implement musical filters and help create distortionless filters.

- Add possibility to generate input signal with Matlab such as simpler sinusoid signals.

Appendix A

Syllabus for Signals and Systems course



UPPSALA
UNIVERSITET

Syllabus for Signals and Systems

Signaler och system

5 credits
Course code: 1TE661
Education cycle: First cycle
Main field(s) of study and in-depth level: Technology G2F
Grading system: Fail (U), 3, 4, 5.
Established: 2009-03-16
Established by: The Faculty Board of Science and Technology
Revised: 2016-04-19
Revised by: The Faculty Board of Science and Technology
Applies from: week 27, 2016
Entry requirements: 60 credits within Science and Technology including Transform Methods, Electronics I/Electrical Engineering II: Electric Circuit Theory as well as Scientific Computing I.
Responsible department: Department of Engineering Sciences

LEARNING OUTCOMES

After a successfully completed course, the student should be able to:

- explain the basic theory for discrete- and continuous time signals and systems, and how they interact in the time- and the frequency domain,
- analyse and synthesise simple analogue systems,
- explain the basic principles for sampling of continuous time signals, the sampling theorem, and signal reconstruction,
- analyse and synthesise simple digital filters.

CONTENT

Fourier-, Laplace- and z-transformation of discrete- and continuous time signals and systems. The sampling theorem. The concepts of poles and zeros. Stability and causality. Bode plots. Analysis and synthesis of analogue and digital filters. Application examples.

INSTRUCTION

Lectures, problem solving sessions.

ASSESSMENT

Written examination (4 credits) oral and written examination of home work assignment (1 credit).

READING LIST

Applies from: week 27, 2016

Knorn, Steffi
Signals and Systems
Institutionen för teknikvetenskaper,

Lathi, Bhagawandas Pannalal
Linear systems and signals
2. ed.: New York: Oxford Univ. press, cop. 2005

Compendium "Signals and Systems" by Steffi Knorn

Appendix B

Main program Matlab code

B.1 Main GUI

```
function varargout = Plug_n_play(varargin)
% PLUG_N_PLAY MATLAB code for Plug_n_play.fig
%      PLUG_N_PLAY, by itself, creates a new PLUG_N_PLAY or raises the existing
%      singleton*.
%
%      H = PLUG_N_PLAY returns the handle to a new PLUG_N_PLAY or the handle to
%      the existing singleton*.
%
%      PLUG_N_PLAY('CALLBACK',hObject,eventData,handles,...) calls the local
%      function named CALLBACK in PLUG_N_PLAY.M with the given input arguments.
%
%      PLUG_N_PLAY('Property','Value',...) creates a new PLUG_N_PLAY or raises the
%      existing singleton*.  Starting from the left, property value pairs are
%      applied to the GUI before Plug_n_play_OpeningFcn gets called.  An
%      unrecognized property name or invalid value makes property application
%      stop.  All inputs are passed to Plug_n_play_OpeningFcn via varargin.
%
%      *See GUI Options on GUIDE's Tools menu.  Choose "GUI allows only one
%      instance to run (singleton)".
%
% See also: GUIDE, GUIDATA, GUIHANDLES

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

% Last Modified by GUIDE v2.5 19-Feb-2017 16:24:24
% Begin initialization code - DO NOT EDIT
gui_Singleton = 1;
gui_State = struct('gui_Name',        mfilename, ...
                   'gui_Singleton',   gui_Singleton, ...
                   'gui_OpeningFcn', @Plug_n_play_OpeningFcn, ...
                   'gui_OutputFcn',  @Plug_n_play_OutputFcn, ...
                   'gui_LayoutFcn',  [] , ...
                   'gui_Callback',   []);
if nargin && ischar(varargin{1})
    gui_State.gui_Callback = str2func(varargin{1});
end

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


% --- Executes just before Plug_n_play is made visible.
function Plug_n_play_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 Plug_n_play (see VARARGIN)

% Choose default command line output for Plug_n_play
% Initial setup
handles.output = hObject;
movegui(gcf, 'center');
wait = waitbar(0/100, 'Loading GUI', 'windowstyle', 'modal');
frames = java.awt.Frame.getFrames();
frames(end).setAlwaysOnTop(1);   %Waitbar always on top
% Add path
addpath('Development','Functions','Images','Sound');
%% For the textfields ---------------------------------------------------
% Below is setup for the division-lines ----------------------------------
set(handles.divider1, 'Visible', 'off', 'Xlim', [0, 1], 'YLim', [0, 1]);
set(handles.divider2, 'Visible', 'off', 'Xlim', [0, 1], 'YLim', [0, 1]);
set(handles.divider3, 'Visible', 'off', 'Xlim', [0, 1], 'YLim', [0, 1]);
set(handles.divider4, 'Visible', 'off', 'Xlim', [0, 1], 'YLim', [0, 1]);
```

```
line([0.1, 1], [1, 1], 'Parent', handles.divider1, 'Color', 'black');
line([0.1, 1], [1, 1], 'Parent', handles.divider2, 'Color', 'black');
line([0.1, 1], [1, 1], 'Parent', handles.divider3, 'Color', 'black');
line([0.1, 1], [1, 1], 'Parent', handles.divider4, 'Color', 'black');

%% The general expression ---------------------------------------------
waitbar(25/100);
axes(handles.staticNum);
set(handles.staticNum, 'Visible', 'off');
text('position',[0 0 0],'interpreter','latex','string','\fontsize{15}{0}\selectfont$\su

axes(handles.staticDenom);
set(handles.staticDenom, 'Visible', 'off');
text('position',[0 1 0],'interpreter','latex','string','\fontsize{15}{0}\selectfont$1+\

%% The filter expression ----------------------------------------------
waitbar(40/100);
axes(handles.numerator);
set(handles.numerator, 'Visible', 'off', 'xlim', [0 1], 'ylim', [0 1]);
text('position',[0.5 0.5 0],'interpreter','latex','string','1');

axes(handles.denominator);
set(handles.denominator, 'Visible', 'off','xlim', [0 1], 'ylim', [0 1]);
text('position',[0.5 0.5 0],'interpreter','latex','string','1');

%% Setup filtration ---------------------------------------------------
waitbar(50/100);
handles.matlabFiltration = [];
set(handles.updatematlabfilter, 'Enable', 'off');
set(handles.visualmatlab, 'Enable', 'off');
set(handles.freqzMatlab, 'Enable', 'off');
set(handles.freqzPrefab, 'Enable', 'off');

set(handles.magntime, 'Enable', 'off');
set(handles.FFTmagn, 'Enable', 'off');

%% Setup plot initial -------------------------------------------------
waitbar(60/100);
set(handles.axesFreqIn,'xcolor',get(gcf,'color'));
set(handles.axesFreqIn,'ycolor',get(gcf,'color'));

set(handles.axesTime,'xcolor',get(gcf,'color'));
set(handles.axesTime,'ycolor',get(gcf,'color'));

set(handles.axesFreqOut,'xcolor',get(gcf,'color'));
set(handles.axesFreqOut,'ycolor',get(gcf,'color'));

set(handles.axesFreqResponse,'xcolor',get(gcf,'color'));
set(handles.axesFreqResponse,'ycolor',get(gcf,'color'));

%% Setup layout -----------------------------------------------------------
waitbar(70/100);
axes(handles.uulogo);
imshow('uulog.png');

[Info,map] = imread('Info.png','png');             %
[r,c,d] = size(Info);                              %
x = ceil(r/30);                                    %
y = ceil(c/30);                                    %
g = Info(1:x:end,1:y:end,:);                       %
g(g==255) = 5.5*255;                               %Handles size of image
set(handles.infoReduced,'CData',g);
set(handles.infoSamplingrate,'CData',g);
set(handles.infoSource,'CData',g);
set(handles.infoSpectralanalysis,'CData',g);
set(handles.infoOwnfilter,'CData',g);
set(handles.infoMatlabfilter,'CData',g);
set(handles.infoApplyfiltration,'CData',g);
set(handles.infoImpulseresponse,'CData',g);
set(handles.infoTime,'CData',g);

[Info,map] = imread('magn.png','png');
[r,c,d] = size(Info);
x = ceil(r/15);
y = ceil(c/15);
g = Info(1:x:end,1:y:end,:);
g(g==255) = 5.5*255;
set(handles.magntime,'CData',g);
set(handles.FFTmagn,'CData',g);

%% Initialize excessive handles -------------------------------------------
waitbar(80/100);
handles.handenominatorVector = [1];
handles.hannumeratorVector = [1];
handles.prefabFilter = 1;
handles.SampleVector = 0;
handles.SampleVectorOut = 0;
handles.FFTin = 0;
handles.FFTout = 0;
handles.freqXaxis = 0;
handles.TimeAxisVector = 0;
handles.samplingRate = 44100;
filters = matfile('filters.mat');          %Load from struct
handles.lowpassFIR = filters.lowpassFIR;
handles.lowpassIIR = filters.lowpassIIR;
handles.bandpassFIR = filters.bandpassFIR;
handles.bandpassIIR = filters.bandpassIIR;
handles.highpassFIR = filters.highpassFIR;
handles.highpassIIR = filters.highpassIIR;
handles.chosenFilter = 'nofilter';
handles.filterisobject = false;
handles.hasShownFilterError = false;
set(handles.ownfilter, 'Value', 1);
waitbar(99/100);
delete(wait);
set(handles.prefabFilterpop, 'Enable', 'on');
% Update handles structure
guidata(hObject, handles);


% UIWAIT makes Plug_n_play wait for user response (see UIRESUME)
% uiwait(handles.MainDSP_GUI);


% --- Outputs from this function are returned to the command line.
function varargout = Plug_n_play_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 button press in onbutton.
function onbutton_Callback(hObject, eventdata, handles)
% hObject    handle to onbutton (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

%% Turns off dials to prevent user input --------------------------------
handles = guidata(hObject);
if get(hObject, 'Value')
set(handles.updatebutton, 'Enable', 'off');
set(handles.samplingfreq, 'Enable', 'off');
set(handles.reduced, 'Enable', 'off');
set(handles.creatematlabfilter, 'Enable', 'off');
set(handles.updatematlabfilter, 'Enable', 'off');
set(handles.magntime, 'Enable', 'off');
set(handles.FFTmagn, 'Enable', 'off');
set(handles.prefabFilterpop, 'Enable', 'off');

%% Calls the function that starts the operation -------------------------
Simple_playGUI(hObject, handles);

%% Turns on dials again --------------------------------------------------
guidata(hObject,handles);
set(handles.updatebutton, 'Enable', 'on');
set(handles.samplingfreq, 'Enable', 'on');
set(handles.reduced, 'Enable', 'on');
set(handles.creatematlabfilter, 'Enable', 'on');
set(handles.updatematlabfilter, 'Enable', 'on');
set(handles.prefabFilterpop, 'Enable', 'on');
if ~exist(get(handles.filtername, 'String'))
    set(handles.filtername, 'String', 'Own filter function');
end
set(handles.ownfunction, 'Enable', 'on');
set(handles.filtername, 'Enable', 'on');
end


% Hint: get(hObject,'Value') returns toggle state of onbutton


% --- Executes on button press in micbutton.
function micbutton_Callback(hObject, eventdata, handles)
% hObject    handle to micbutton (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
% Hint: get(hObject,'Value') returns toggle state of micbutton


% --- Executes on button press in filebutton.
function filebutton_Callback(hObject, eventdata, handles)
% hObject    handle to filebutton (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hint: get(hObject,'Value') returns toggle state of filebutton



function filename_Callback(hObject, eventdata, handles)
% hObject    handle to filename (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 filename as text
% str2double(get(hObject,'String')) returns contents of filename as a double


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

%% Probably change in guide for white background
% 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','green');
end


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

```
% Hint: place code in OpeningFcn to populate axesFreqIn


% --- Executes on button press in togglebutton2.
function togglebutton2_Callback(hObject, eventdata, handles)
% hObject    handle to togglebutton2 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hint: get(hObject,'Value') returns toggle state of togglebutton2


% --- Executes on button press in pausebutton.
function pausebutton_Callback(hObject, eventdata, handles)
% hObject    handle to pausebutton (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

%------------------------------------------------------------------------
%% For the pause button
handles = guidata(hObject);
if (get(hObject,'Value'))
    set(hObject, 'String', char(9658));
    set(handles.magntime, 'Enable', 'on');
    set(handles.FFTmagn, 'Enable', 'on');
else
    set(hObject, 'String', [char(10074) char(10074)]);
    set(handles.magntime, 'Enable', 'off');
    set(handles.FFTmagn, 'Enable', 'off');
end
guidata(hObject,handles);
%------------------------------------------------------------------------

% Hint: get(hObject,'Value') returns toggle state of pausebutton



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

```
%% Enters the lower limit. If the string does not contain only numbers, the
% other letters will not be fed into outputString.
inputString = get(hObject, 'String'); % Retrieves from textbox
if isempty(inputString)          % If user has just erased value
    set(hObject, 'String', '0 Hz');  % Set default
else
    outputString = '';               % "Declare"
    for i = 1:length(inputString)    %Go through whole string
        if isnan(str2double(inputString(i))) % Really if not empty (is number)
        else       %Does not enter else if for example a letter is in string
            outputString = [outputString inputString(i)]; %Feed number output
        end
    end
    if isempty(outputString)   %If input never contained a number
        set(hObject, 'String', '0 Hz'); %Sets default
    else
        outputString = [outputString ' Hz']; %Adds Hz
        %set(hObject, 'String', '');
        set(hObject, 'String', outputString); %Sets input number
    end
end
% Hints: get(hObject,'String') returns contents of xlimitlow as text
% str2double(get(hObject,'String')) returns contents of xlimitlow as a double


% --- Executes during object creation, after setting all properties.
function xlimitlow_CreateFcn(hObject, eventdata, handles)
% hObject    handle to xlimitlow (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 xlimithigh_Callback(hObject, eventdata, handles)
% hObject    handle to xlimithigh (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
%% Enters the upper limit. If the string does not contain only numbers, the
% other letters will not be fed into outputString.
inputString = get(hObject, 'String'); % Retrieves from textbox
if isempty(inputString)      % If user has just erased value
    set(hObject, 'String', '20000 Hz');   % Set default
else
    outputString = '';                        % "Declare"
    for i = 1:length(inputString)          %Go through whole string
        if isnan(str2double(inputString(i))) % Really if not empty(is number)
        else         %Does not enter else if for example a letter is in string
            outputString = [outputString inputString(i)];  %Feed number output
        end
    end
    if isempty(outputString)  %If input never contained a number
        set(hObject, 'String', '20000 Hz');  %Sets default
    else
        outputString = [outputString ' Hz'];  %Adds Hz
        set(hObject, 'String', outputString);  %Sets input number
    end
end
% Hints: get(hObject,'String') returns contents of xlimithigh as text
% str2double(get(hObject,'String')) returns contents of xlimithigh as a double


% --- Executes during object creation, after setting all properties.
function xlimithigh_CreateFcn(hObject, eventdata, handles)
% hObject    handle to xlimithigh (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 numeratorVector_Callback(hObject, eventdata, handles)
% hObject    handle to numeratorVector (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
%% Determines what happens when enter is pushed after fed in coefficients
numeratorString = get(hObject, 'String');   % Retrieves string
numeratorVector = str2vec(numeratorString); % Converts into vector
handles.hannumeratorVector = numeratorVector;
set(hObject, 'String', [ '[' num2str(numeratorVector) ']' ]); % Displays
guidata(hObject, handles);
% Hints: get(hObject,'String') returns contents of numeratorVector as text
% str2double(get(hObject,'String')) returns contents of numeratorVector as a double


% --- Executes during object creation, after setting all properties.
function numeratorVector_CreateFcn(hObject, eventdata, handles)
% hObject    handle to numeratorVector (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 denominatorVector_Callback(hObject, eventdata, handles)
% hObject    handle to denominatorVector (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
denominatorString = get(hObject, 'String'); % Retrieves string
denominatorVector = str2vec(denominatorString);  % Converts into vector
handles.handenominatorVector = denominatorVector;
set(hObject, 'String', [ '[' num2str(denominatorVector) ']' ]); % Displays
guidata(hObject, handles);
% Hints: get(hObject,'String') returns contents of denominatorVector as text
% str2double(get(hObject,'String')) returns contents of denominatorVector as a double


% --- Executes during object creation, after setting all properties.
function denominatorVector_CreateFcn(hObject, eventdata, handles)
% hObject    handle to denominatorVector (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 updatebutton.
function updatebutton_Callback(hObject, eventdata, handles)
% hObject    handle to updatebutton (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

%% Numerator ----------------------------------------------------------------
% numStr = get(handles.numeratorVector, 'String'); %Retrieves string
% numVec = str2vec(numStr); % Converts into vector
numVec = handles.hannumeratorVector;
numStr = '1'; % Initializes
for i = 1:length(numVec)   % Go through vector
    if i == 1              % First iteration
        numStr = [num2str(numVec(i))]; % Add first coefficient
    else    % Next: create the rest of the string
        numStr = [numStr '+' num2str(numVec(i)) 'z^-^{' num2str(i-1) '}'];
    end
end
axes(handles.numerator);  % Specify axes
cla(handles.numerator);   % Clear axes
text((0.5-4.7*((length(numVec))/36)), 0.7, numStr); % Set string, varies

%% Denominator --------------------------------------------------------------
% denomStr = get(handles.denominatorVector, 'String'); %Retrieves string
% denomVec = str2vec(denomStr);  % Converts into vector
denomVec = handles.handenominatorVector;
denomStr = '1';       % Initializes
for i = 1:length(denomVec)   % Go through vector
    if i == 1                % First iteration
        denomStr = [num2str(denomVec(i))]; % Add first coefficient
    else          % Next: create the rest of the string
        denomStr = [denomStr '+' num2str(denomVec(i)) 'z^-^' num2str(i-1)];
    end
```

```
end
axes(handles.denominator);  % Specify axes
cla(handles.denominator);   % Clear axes
text((0.5-4.7*((length(denomVec))/36)), 0.7, denomStr); % Set string, varies


% --- Executes on button press in visualown.
function visualown_Callback(hObject, eventdata, handles)
% hObject    handle to visualown (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
%% Uses numerator and denominator to visualize the filter in separate window
% numStr = get(handles.numeratorVector, 'String');
% numVec = str2vec(numStr);
numVec = handles.hannumeratorVector;
% denomStr = get(handles.denominatorVector, 'String');
% denomVec = str2vec(denomStr);
denomVec = handles.handenominatorVector;
Fs = str2double(get(handles.samplingfreq, 'String'));
fvtool(numVec,denomVec, 'NormalizedFrequency', 'off','Fs',Fs);


% --- Executes on button press in visualmatlab.
function visualmatlab_Callback(hObject, eventdata, handles)
% hObject    handle to visualmatlab (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
fvtool(handles.matlabFiltration, 'NormalizedFrequency', 'off');


% --- Executes on button press in creatematlabfilter.
function creatematlabfilter_Callback(hObject, eventdata, handles)
% hObject    handle to creatematlabfilter (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
filter = designfilt;
if isempty(filter)
else
    handles.matlabFiltration = filter;
    matlabfilterinformation(handles, filter);
    set(handles.updatematlabfilter, 'Enable', 'on');
    set(handles.visualmatlab, 'Enable', 'on');
    set(handles.freqzMatlab, 'Enable', 'on');
```

```
end

guidata(hObject, handles);



% --- Executes on button press in applyfilter.
function applyfilter_Callback(hObject, eventdata, handles)
% hObject    handle to applyfilter (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
filter = handles.matlabFiltration;
if isempty(filter) && (get(handles.matlabfilter, 'Value') == 1)
    errordlg('Matlab filter must be created first','Filter Error');
    set(hObject, 'Value', 0);
end


% Hint: get(hObject,'Value') returns toggle state of applyfilter


% --- Executes on button press in updatematlabfilter.
function updatematlabfilter_Callback(hObject, eventdata, handles)
% hObject    handle to updatematlabfilter (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
filter = handles.matlabFiltration;
designfilt(filter);
handles.matlabFiltration = filter;
matlabfilterinformation(handles, filter);
guidata(hObject, handles);


% --- Executes on selection change in prefabFilterpop.
function prefabFilterpop_Callback(hObject, eventdata, handles)
% hObject    handle to prefabFilterpop (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
% Determine the selected data set.
str = get(hObject, 'String');
val = get(hObject,'Value');
% Set current data to the selected data set.
switch str{val};
```

```
case 'No filter'
    set(handles.freqzPrefab, 'Enable', 'off');
    handles.chosenFilter = 'nofilter';
case 'Own filter function'
    set(handles.freqzPrefab, 'Enable', 'on');
    handles.chosenFilter
case 'Lowpass 1000Hz FIR'
    set(handles.freqzPrefab, 'Enable', 'on');
    handles.chosenFilter = handles.lowpassFIR;
case 'Lowpass 1000Hz IIR'
    set(handles.freqzPrefab, 'Enable', 'on');
        handles.chosenFilter = handles.lowpassIIR;
case 'Bandpass 1000-1300Hz FIR'
    set(handles.freqzPrefab, 'Enable', 'on');
    handles.chosenFilter = handles.bandpassFIR;
case 'Bandpass 1000-1300Hz IIR'
    set(handles.freqzPrefab, 'Enable', 'on');
    handles.chosenFilter = handles.bandpassIIR;
case 'Highpass 1500Hz FIR'
    set(handles.freqzPrefab, 'Enable', 'on');
    handles.chosenFilter = handles.highpassFIR;
case 'Highpass 1500Hz IIR'
    set(handles.freqzPrefab, 'Enable', 'on');
    handles.chosenFilter = handles.highpassIIR;
case 'Bass filter 250 Hz Lowpass'
    set(handles.freqzPrefab, 'Enable', 'off');
    handles.chosenFilter = 'lowpass250Hz';
case 'Varying filter'
    set(handles.freqzPrefab, 'Enable', 'off');
    handles.chosenFilter = 'variablefilter';
case 'Flanger'
    set(handles.freqzPrefab, 'Enable', 'off');
    handles.chosenFilter = 'flanger';
end
% Save the handles structure.
guidata(hObject,handles)




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

% Hint: popupmenu 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 gain_Callback(hObject, eventdata, handles)
% hObject    handle to gain (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
gainstr = get(hObject, 'String');
if isnan(str2double(gainstr))
    errordlg('Gain must be of numeric type','Gain Error');
    set(hObject, 'String', '1');
elseif str2double(gainstr) < 0
     set(hObject, 'String', '0');
end

% Hints: get(hObject,'String') returns contents of gain as text
% str2double(get(hObject,'String')) returns contents of gain as a double


% --- Executes during object creation, after setting all properties.
function gain_CreateFcn(hObject, eventdata, handles)
% hObject    handle to gain (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 slider movement.
```

```
function slider3_Callback(hObject, eventdata, handles)
% hObject    handle to slider3 (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 slider3_CreateFcn(hObject, eventdata, handles)
% hObject    handle to slider3 (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 gainup.
function gainup_Callback(hObject, eventdata, handles)
% hObject    handle to gainup (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
current = get(handles.gain, 'String');
currentNumber = str2double(current);
new = num2str(floor(currentNumber + 1));
set(handles.gain, 'String', new);



% --- Executes on button press in gaindown.
function gaindown_Callback(hObject, eventdata, handles)
% hObject    handle to gaindown (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
current = get(handles.gain, 'String');
currentNumber = str2double(current);
new = (floor(currentNumber - 1));
if (new < 0)
```

```
    new = 0;
end
new = num2str(new);
set(handles.gain, 'String', new);



function samplingfreq_Callback(hObject, eventdata, handles)
% hObject    handle to samplingfreq (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
freqstr = get(hObject, 'String');
freq = str2double(freqstr);
if (isnan(freq) || (freq < 8000) || (192000 < freq) ||~(isreal(freq)) ...
        || (mod(freq,1) ~= 0))
    msg = ['Sampling frequency must be a positive integer between 8000 '...
        'and 192000 Hz'];
    errordlg(msg,'Sampling Error');
    set(hObject, 'String', '44100');
else
    handles.samplingRate = freq;
end
guidata(hObject, handles);

% Hints: get(hObject,'String') returns contents of samplingfreq as text
%str2double(get(hObject,'String')) returns contents of samplingfreq as a


% --- Executes during object creation, after setting all properties.
function samplingfreq_CreateFcn(hObject, eventdata, handles)
% hObject    handle to samplingfreq (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 prefabfilter.
```

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

% Hint: get(hObject,'Value') returns toggle state of prefabfilter


% --- Executes on button press in reduced.
function reduced_Callback(hObject, eventdata, handles)
% hObject    handle to reduced (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hint: get(hObject,'Value') returns toggle state of reduced


% --- Executes on button press in infoReduced.
function infoReduced_Callback(hObject, eventdata, handles)
% hObject    handle to infoReduced (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
p1 = ['Reduced visual playback reduces update frequency of the plots.'...
     'This should be used at all times except when using a more powerful '...
     'computer.\n\n'];

p2 = ['This does not make a substantial difference for the eye but is a'...
     'lot easier for our poor friend the computer.\n\n'];

p3 = ['You can also press the pause buttons in the "Spectral analysis" '...
     'and "Time plot of input signal" sections, this will pause the'...
     'visual playback which reduces ',...
       'overhead and thereby increases performance.\n\n'];
  string = [p1 p2 p3];
helpdlg(sprintf(string), 'Reduced visual playback information');


% --- Executes on button press in infoSamplingrate.
function infoSamplingrate_Callback(hObject, eventdata, handles)
% hObject    handle to infoSamplingrate (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
p1 = ['The sample rate text box lets you specify what samle rate you ',...
```

```
    'want when recording from the microphone jack. If the source is  ',...
    'from a file , the sample rate of that file will be the applied  ',...
    'sample rate.\n\n'];

p2 = ['The sample rate can not be changed while the program is  ',...
      'running.\n\n'];

p3 = ['All prefabricated filters are made for a sample rate of 44100  ',...
      'Hz.It is not possible to use the prefabricated filters when  ',...
      'using another sample frequency for mic or file input. This is  ',...
      'only to avoid confusion if the sample friquency is changed,  ',...
      'the cut off frequency will be dislocated from the displayed  ',...
      'in the filter name.\n\n'];


string = [p1 p2 p3];
helpdlg(sprintf(string), 'Sample rate info');

% --- Executes on button press in infoSource.
function infoSource_Callback(hObject, eventdata, handles)
% hObject    handle to infoSource (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
part1 = ['Choose between file or mic input.\n\n'];

part2 = ['The file must be located in the current Matlab directory.  ',...
    'The different file types accepted are .mp3, .wma, .flac, .wav,  ',...
    'etc. It is important to write the file with .filetype included,  ',...
    'like "filename.mp3" but without ". It is the sampling rate of  ',...
    'the file that control the playback sampling rate. If the  ',...
    'sampling rate is anything other than 44100 Hz, the files sampling  ',...
    'rate is shown in the sampling rate text box.\n\n'];

part3 =  ['Mic input uses the built in 3,5 mm mic input jack in the  ',...
          'computer. If the computer has a built in microphone, that  ',...
          'can also be used. When mic input is used, the sampling rate  ',...
          'can be set manually in the sampling rate text box.\n\n'];
string = [part1 part2 part3];
helpdlg(sprintf(string), 'Source info');


% --- Executes on button press in infoMatlabfilter.
function infoMatlabfilter_Callback(hObject, eventdata, handles)
```

```
% hObject    handle to infoMatlabfilter (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
p1 = [ 'Let Matlab do the hard work of constructing the filter ',...
        'for you.\n\n'];

p2 = ['By pressing "Create Matlab filter" Matlabs own "designfilt" ',...
    'manager opens up. You can then choose which type of filter you''d ',...
    ' like to construct and what properties you want the filter to ',...
    'have.\n\n'];

p3 = ['When specifying the frequency it is important to have in mind ',...
    'that if the frequency is set in Hz and the order mode is Minimum, ',...
    'the creation of the filter can take some time and can be tough ',...
    'for weaker computers. Also, make sure that the "Input sample ',...
    'rate" matches the sampling rate that is set in the sampling rate ',...
    'text box.\n\n'];

p4 = ['The "Change Matlab filter" button lets you change the filter ',...
    'design properties without creating a new filter.\n\n'];

p5 = ['The "Visualize Matlab filter" button opens Matlabs "fvtool". ',...
    'With "fvtool" you can acquire information about the magnitude ',...
    'response, phase response, filter coefficients and other ',...
    'important properties.\n\n'];

p6 = ['Apart from "designfilt" there is also the Matlab functions ',...
    '"filterbuilder" and "sptool" wich let you build filters at ',...
    'different levels of what Matlab does for you automatically.\n\n'];

string = [p1 p2 p3 p4 p5 p6];
helpdlg(sprintf(string), 'Matlab filter info');

% --- Executes on button press in infoOwnfilter.
function infoOwnfilter_Callback(hObject, eventdata, handles)
% hObject    handle to infoOwnfilter (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
p1 = ['This "Own filter design" section lets you construct your own ',...
       'filter by typing in the coefficients of the filters ',...
       'denominator and numerator, as both FIR and IIR filters can be ',...
       'constructed in this way. The coefficients are b[k] and a[k] ',...
       'that is shown in the general formula right next to the text ',...
```

```
      'boxes.\n\n'];

p2 = ['The "Update filter" button creates a filter based on the ',...
     'coefficients in the text boxes. The transfer function of the ',...
     'created filter will then be printed to the right of the ',...
     'text boxes.\n\n'];

p3 = ['The "Visualize own filter" button opens up "fvtool". With ',...
      '"fvtool" you can acquire information about the magnitude ',...
      'response, phase response, filter coefficients and other ',...
      'important properties.\n\n'];

string = [p1 p2 p3];
helpdlg(sprintf(string), 'Own filter design info');

% --- Executes on button press in infoSpectralanalysis.
function infoSpectralanalysis_Callback(hObject, eventdata, handles)
% hObject    handle to infoSpectralanalysis (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
p1 = ['Spectral analysis shows the Fast Fourier transform (fft/number ',...
      'of samples), of both the input signal at the top and the output ',...
      'signal at the bottom.\n\n'];

p2 = ['The y-axis adjusts in accordance to the amplitude of the ',...
      'frequencies. The x-axis is ajustable in the boxes on the left ',...
      'side of the output signal''s fft. The lower limit of the x-axis ',...
      'is decided by the number in the top box and the upper limit by ',...
      'the bottom box.\n\n'];

p3 = ['Use the pause button to freeze the plots either to reduce the ',...
      'overhead and thereby making it easier on the computer or to ',...
      'use the magnifying button. \n\n'];

p4 = ['The magnifying button opens up a new window and displays the ',...
      'same plots shown in spectral analysis but larger. To press this ',...
      'button, either pause the whole program or use the spectral ',...
      'analysis pause button.\n\n'];


p4 = ['Please note; the limit of the x-axis can only be changed ',...
      'while the program is turned off or paused.\n\n'];
```

```
  string = [p1 p2 p3 p4];
helpdlg(sprintf(string), 'Spectral analysis info');

% --- Executes on button press in infoApplyfiltration.
function infoApplyfiltration_Callback(hObject, eventdata, handles)
% hObject    handle to infoApplyfiltration (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
p1 = ['Choose between "Own filter design", "Let Matlab do it" and ',...
      '"Prefabricated filter". The default is set to "Own filter ',...
      'design" since H(z)=1 when the filter coefficients are ',...
      'unchanged.\n\n'];

p2 = ['To choose "Matlab filter design", there needs to be a Matlab ',...
      'filter designed first.\n\n'];

p3 = ['"Prefabricated filter" lets you choose between numerous ',...
      'prefabricated filters with different specifications. These ',...
      'will only work with a sampling rate of 44100 Hz.\n\n'];

p4 = ['After choosing the desired filter, mark the "Apply filtration" ',...
      'box. This can be done while the program is running.\n\n'];

p5 = ['As long as a Matlab filter has been created and a prefabricated ',...
      'filter has been chosen, it is possible to switch between the ',...
      'different filtering methods while the program is running and ',...
      'the "Apply filtration" box is ticked.\n\n'];

p6 = ['You can also use "Own filter function". Make sure that you ',...
      'have the filter function in the current matlab directory, and ',...
      'simply type in the filter function name. Make sure that your ',...
      'filer function looks similar to the below header with the same ',...
      'number of in- and output variables:\n\n'];

p7 = ['"function y = testfilterfunction(audioin,index)"\n\n'];


p8 = ['The above function is called "testfilterfunction" but the ',...
      'name can be anything as long as the name is called correctly ',...
      'in the GUI''s textbox. In this case if you had made a function ',...
      'called testfilterfunction, you would simply type in ',...
      '"testfilterfunction" without the ". audioin is a (1024,2) ',...
      'matrix and index is the value of the current itteration. If ',...
      'index is 1 audioin is the first 1024 samples of the song, ',...
      'Index 2 of audioin is the next 1024-2048 samples index must ',...
      'not be used but some filters has implementations that needs ',...
      'a variable that keep track of where in the song you are.\n\n'];


   string = [p1 p2 p3 p4 p5 p6 p7 p8];
helpdlg(sprintf(string), 'Filter source info');



% --- Executes on button press in freqzOwn.
function freqzOwn_Callback(hObject, eventdata, handles)
% hObject    handle to freqzOwn (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
denomVec = handles.handenominatorVector;
numVec = handles.hannumeratorVector;
%fvtool(numVec,denomVec,'Analysis','impulse');
imp = [1;zeros(49,1)];%
%imp = dirac(0:49);
response = filter(numVec, denomVec, imp);
plot(handles.axesFreqResponse, 0:49, response);
xlabel(handles.axesFreqResponse, 'Samples')
ylabel(handles.axesFreqResponse, 'Magnitude')

%[response,w] = freqz(numVec,denomVec,1000);
%plot(handles.axesFreqResponse, w/pi, 20*log10(abs(response)));
%xlabel(handles.axesFreqResponse,...
%'Normalized Frequency [\times\pi rad/sample]');
%ylabel(handles.axesFreqResponse, 'Magnitude [dB]')




% --- Executes on button press in freqzMatlab.
function freqzMatlab_Callback(hObject, eventdata, handles)
% hObject    handle to freqzMatlab (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
matlabfilt = handles.matlabFiltration;
imp = [1;zeros(1000,1)];%dirac(0:49);
response = filter(matlabfilt, imp);
plot(handles.axesFreqResponse, 0:1000, response);
```

```
xlabel(handles.axesFreqResponse, 'Samples')
ylabel(handles.axesFreqResponse, 'Magnitude')
%[response,w] = freqz(matlabfilt,1000);
%plot(handles.axesFreqResponse, w/pi, 20*log10(abs(response)));
%xlabel(handles.axesFreqResponse,...
%'Normalized Frequency [\times\pi rad/sample]')
%ylabel(handles.axesFreqResponse, 'Magnitude [dB]')



% --- Executes on button press in freqzPrefab.
function freqzPrefab_Callback(hObject, eventdata, handles)
% hObject    handle to freqzPrefab (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
g = handles.chosenFilter;
if ~ischar(g)
    imp = [1;zeros(1000,1)];%dirac(0:49);
    response = filter(g, imp);
    plot(handles.axesFreqResponse, 0:1000, response);
    xlabel(handles.axesFreqResponse, 'Samples')
    ylabel(handles.axesFreqResponse, 'Magnitude')
%     [response,w] = freqz(handles.chosenFilter,1000);
%     plot(handles.axesFreqResponse, w/pi, 20*log10(abs(response)));
%     set(handles.axesFreqResponse, 'YLim', [-60 10]);
%     xlabel(handles.axesFreqResponse,...
%     'Normalized Frequency [\times\pi rad/sample]')
%     ylabel(handles.axesFreqResponse, 'Magnitude [dB]')
end


% --- Executes on button press in infoImpulseresponse.
function infoImpulseresponse_Callback(hObject, eventdata, handles)
% hObject    handle to infoImpulseresponse (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
p1 = ['Impulse response shows a simplified plot of the impulse ',...
      'response also visible by "Visualize Matlab filter design" ',...
      'and "Visualize own filter".\n\n'];

p2 = ['Some prefabricated filters impulse responses can also be ',...
      'shown.\n\n'];
```

```
p3 = ['The x-axis is fixed by a fix number of samples. For own ',...
      'filter desingn, impulse response only sows for the fifty ',...
      'first samples. For the other two it shows the first thousand ',...
      'samples. This panel shall only give an indication for how ',...
      'the impulse response will look. \n\n'];

string = [p1 p2 p3];
helpdlg(sprintf(string), 'Impulse response info');

% --- Executes on button press in matlabfilter.
function ownfilter_Callback(hObject, eventdata, handles)
% hObject    handle to matlabfilter (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
set(handles.applyfilter, 'Enable', 'on');
% Hint: get(hObject,'Value') returns toggle state of matlabfilter


% --- Executes on button press in matlabfilter.
function matlabfilter_Callback(hObject, eventdata, handles)
% hObject    handle to matlabfilter (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hint: get(hObject,'Value') returns toggle state of matlabfilter


% --- Executes on button press in magntime.
function magntime_Callback(hObject, eventdata, handles)
% hObject    handle to magntime (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
Timeplot_GUI;


% --- Executes on button press in infoTime.
function infoTime_Callback(hObject, eventdata, handles)
% hObject    handle to infoTime (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

p1 = ['Use the pause button to freeze the plots either to reduce the ',...
      'overhead and thereby making it easier on the computer or to ',...
```

```
        'use the magnifying button. \n\n'];

p2 = ['Use the magnifying button when paused to wiew a magnification  ',...
        'of the input and output signal. There you can also adjust the  ',...
        'X- and Y- limits. To press this button, either pause the whole  ',...
        'program or use the Time plot pause button.\n\n'];


string = [p1 p2];

helpdlg(sprintf(string), 'Time plot info');



% --- Executes on button press in FFTmagn.
function FFTmagn_Callback(hObject, eventdata, handles)
% hObject    handle to FFTmagn (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
FFTplot_GUI;


% --- Executes on button press in pauseFFT.
function pauseFFT_Callback(hObject, eventdata, handles)
% hObject    handle to pauseFFT (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
if (get(hObject,'Value'))
    set(hObject, 'String', char(9658));
    set(handles.FFTmagn, 'Enable', 'on');
else
    set(hObject, 'String', [char(10074) char(10074)]);
    set(handles.FFTmagn, 'Enable', 'off');
end


% --- Executes on button press in pauseTime.
function pauseTime_Callback(hObject, eventdata, handles)
% hObject    handle to pauseTime (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
if (get(hObject,'Value'))
    set(hObject, 'String', char(9658));
    set(handles.magntime, 'Enable', 'on');
else
    set(hObject, 'String', [char(10074) char(10074)]);
    set(handles.magntime, 'Enable', 'off');
end



function filtername_Callback(hObject, eventdata, handles)
% hObject    handle to filtername (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 filtername as text
% str2double(get(hObject,'String')) returns contents of filtername as a


% --- Executes during object creation, after setting all properties.
function filtername_CreateFcn(hObject, eventdata, handles)
% hObject    handle to filtername (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
```

B.2 FFT plot GUI

```
function varargout = FFTplot_GUI(varargin)
% FFTPLOT_GUI MATLAB code for FFTplot_GUI.fig
%      FFTPLOT_GUI, by itself, creates a new FFTPLOT_GUI or raises the existing
%      singleton*.
%
%      H = FFTPLOT_GUI returns the handle to a new FFTPLOT_GUI or the handle to
%      the existing singleton*.
%
%      FFTPLOT_GUI('CALLBACK',hObject,eventData,handles,...) calls the local
%      function named CALLBACK in FFTPLOT_GUI.M with the given input arguments.
%
```

```
%      FFTPLOT_GUI('Property','Value',...) creates a new FFTPLOT_GUI or raises the
%      existing singleton*.  Starting from the left, property value pairs are
%      applied to the GUI before FFTplot_GUI_OpeningFcn gets called.  An
%      unrecognized property name or invalid value makes property application
%      stop.  All inputs are passed to FFTplot_GUI_OpeningFcn via varargin.
%
%      *See GUI Options on GUIDE's Tools menu.  Choose "GUI allows only one
%      instance to run (singleton)".
%
% See also: GUIDE, GUIDATA, GUIHANDLES

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

% Last Modified by GUIDE v2.5 19-Jan-2017 20:51:48

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

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


% --- Executes just before FFTplot_GUI is made visible.
function FFTplot_GUI_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 FFTplot_GUI (see VARARGIN)
% Choose default command line output for FFTplot_GUI
handles.output = hObject;
movegui(gcf, 'center');
axes(handles.axesUUlogo);
imshow('uulog.png');

[Info,map] = imread('Info.png','png');
[r,c,d] = size(Info);
x = ceil(r/30);
y = ceil(c/30);
g = Info(1:x:end,1:y:end,:);
g(g==255) = 5.5*255;
set(handles.infoFFTLarge,'CData',g);

h = findobj('Tag','MainDSP_GUI');
if ~isempty(h)
    mainData = guidata(h);
    FrequencyAxis = mainData.freqXaxis;
    AudioInSpectra = mainData.FFTin;
    AudioOutSpectra = mainData.FFTout;
    stem(handles.axesFFTin, [FrequencyAxis, FrequencyAxis],...
        [AudioInSpectra(1,:) AudioInSpectra(2,:)]);
    set(handles.axesFFTin, 'YLim', [0 0.01])
    stem(handles.axesFFTout, [FrequencyAxis, FrequencyAxis],...
        [AudioOutSpectra(1,:) AudioOutSpectra(2,:)]);
    set(handles.axesFFTout, 'YLim', [0 0.01])
end

% Update handles structure
guidata(hObject, handles);

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


% --- Outputs from this function are returned to the command line.
function varargout = FFTplot_GUI_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;



function XLimLow_Callback(hObject, eventdata, handles)
% hObject    handle to XLimLow (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
string = get(hObject, 'String');
vector = str2vec(string);
g = get(handles.axesFFTin, 'XLim');
if g(2) <= vector(1)
    errordlg('The lower limit must be lower than the upper limit',...
        'Limit error');
else
    if isempty(vector)
        set(handles.axesFFTin, 'XLim', [0 g(2)]);
        set(handles.axesFFTout, 'XLim', [0 g(2)]);
        set(handles.XLimLow, 'String', '0 Hz');
    else
        set(handles.axesFFTin, 'XLim', [vector(1) g(2)]);
        set(handles.axesFFTout, 'XLim', [vector(1) g(2)]);
        set(handles.XLimLow, 'String', [num2str(vector(1)) ' Hz']);
    end
end

% Hints: get(hObject,'String') returns contents of XLimLow as text
%        str2double(get(hObject,'String')) returns contents of XLimLow as a


% --- Executes during object creation, after setting all properties.
function XLimLow_CreateFcn(hObject, eventdata, handles)
% hObject    handle to XLimLow (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 XLimHigh_Callback(hObject, eventdata, handles)
% hObject    handle to XLimHigh (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
string = get(hObject, 'String');
vector = str2vec(string);
g = get(handles.axesFFTin, 'XLim');
if vector(1) <= g(1)
    errordlg('The upper limit must be larger than the lower limit',...
        'Limit error');
else
    if isempty(vector)
        set(handles.axesFFTin, 'XLim', [g(1) 2]);
        set(handles.axesFFTout, 'XLim', [g(1) 2]);
        set(handles.XLimHigh, 'String', '20000 Hz');
    else
        set(handles.axesFFTin, 'XLim', [g(1) vector(1)]);
        set(handles.axesFFTout, 'XLim', [g(1) vector(1)]);
        set(handles.XLimHigh, 'String', [num2str(vector(1)) ' Hz']);
    end
end

% Hints: get(hObject,'String') returns contents of XLimHigh as text
%        str2double(get(hObject,'String')) returns contents of XLimHigh as a


% --- Executes during object creation, after setting all properties.
function XLimHigh_CreateFcn(hObject, eventdata, handles)
% hObject    handle to XLimHigh (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 YLimLow_Callback(hObject, eventdata, handles)
% hObject    handle to YLimLow (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
string = get(hObject, 'String');
vector = str2vec(string);
g = get(handles.axesFFTin, 'YLim');
if g(2) <= vector(1)
    errordlg('The lower limit must be lower than the upper limit',...
        'Limit error');
else
    if isempty(vector)
        set(handles.axesFFTin, 'YLim', [0 g(2)]);
        set(handles.axesFFTout, 'YLim', [0 g(2)]);
        set(handles.YLimLow, 'String', '0');
    else
        set(handles.axesFFTin, 'YLim', [vector(1) g(2)]);
        set(handles.axesFFTout, 'YLim', [vector(1) g(2)]);
        set(handles.YLimLow, 'String', num2str(vector(1)));
    end
end

% Hints: get(hObject,'String') returns contents of YLimLow as text
%        str2double(get(hObject,'String')) returns contents of YLimLow as


% --- Executes during object creation, after setting all properties.
function YLimLow_CreateFcn(hObject, eventdata, handles)
% hObject    handle to YLimLow (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 YLimHigh_Callback(hObject, eventdata, handles)
% hObject    handle to YLimHigh (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
string = get(hObject, 'String');
vector = str2vec(string);
g = get(handles.axesFFTin, 'YLim');
if vector(1) <= g(1)
    errordlg('The upper limit must be larger than the lower limit',...
        'Limit error');
else
    if isempty(vector)
        set(handles.axesFFTin, 'YLim', [g(1) 2]);
        set(handles.axesFFTout, 'YLim', [g(1) 2]);
        set(handles.YLimHigh, 'String', '0.01');
    else
        set(handles.axesFFTin, 'YLim', [g(1) vector(1)]);
        set(handles.axesFFTout, 'YLim', [g(1) vector(1)]);
        set(handles.YLimHigh, 'String', num2str(vector(1)));
    end
end

% Hints: get(hObject,'String') returns contents of YLimHigh as text
%        str2double(get(hObject,'String')) returns contents of YLimHigh as


% --- Executes during object creation, after setting all properties.
function YLimHigh_CreateFcn(hObject, eventdata, handles)
% hObject    handle to YLimHigh (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 infoFFTLarge.
function infoFFTLarge_Callback(hObject, eventdata, handles)
% hObject    handle to infoFFTLarge (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
```

```
p1 = ['This GUI present the fast fourier transform plot for both input -',...
    'and output signals.\n\n'];

p2 = ['Use the textboxes to ajust the limits of the Y-axis and X-axis ',...
     'respectively. Please note that in the primary GUI the Y-limits ',...
     'are set automatically but here they have to be set manually.\n\n'];

  string = [p1 p2];

helpdlg(sprintf(string), 'Timeplot GUI info');


% --- Executes on button press in pushbutton2.
function pushbutton2_Callback(hObject, eventdata, handles)
% hObject    handle to pushbutton2 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
close(handles.FFTplot_GUI);
```

B.3 Time plot GUI

```
function varargout = Timeplot_GUI(varargin)
% TIMEPLOT_GUI MATLAB code for Timeplot_GUI.fig
%      TIMEPLOT_GUI, by itself, creates a new TIMEPLOT_GUI or raises the existing
%      singleton*.
%
%      H = TIMEPLOT_GUI returns the handle to a new TIMEPLOT_GUI or the handle to
%      the existing singleton*.
%
%      TIMEPLOT_GUI('CALLBACK',hObject,eventData,handles,...) calls the local
%      function named CALLBACK in TIMEPLOT_GUI.M with the given input arguments.
%
%      TIMEPLOT_GUI('Property','Value',...) creates a new TIMEPLOT_GUI or raises the
%      existing singleton*.  Starting from the left, property value pairs are
%      applied to the GUI before Timeplot_GUI_OpeningFcn gets called.  An
%      unrecognized property name or invalid value makes property application
%      stop.  All inputs are passed to Timeplot_GUI_OpeningFcn via varargin.
%
%      *See GUI Options on GUIDE's Tools menu.  Choose "GUI allows only one
%      instance to run (singleton)".
%
% See also: GUIDE, GUIDATA, GUIHANDLES
% Edit the above text to modify the response to help Timeplot_GUI

% Last Modified by GUIDE v2.5 13-Jan-2017 11:22:27

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

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


% --- Executes just before Timeplot_GUI is made visible.
function Timeplot_GUI_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 Timeplot_GUI (see VARARGIN)

% Choose default command line output for Timeplot_GUI
handles.output = hObject;
movegui(gcf, 'center');
axes(handles.axesuulogo);
imshow('uulog.png');

[Info,map] = imread('Info.png','png');
[r,c,d] = size(Info);
x = ceil(r/30);
y = ceil(c/30);
g = Info(1:x:end,1:y:end,:);
```

```
g(g==255) = 5.5*255;
set(handles.infoLargeTimePlot,'CData',g);

h = findobj('Tag','MainDSP_GUI');
if ~isempty(h)
    mainData = guidata(h);
    TimeAxisVector = mainData.TimeAxisVector;
    SampleVector = mainData.SampleVector;
    SampleVectorOut = mainData.SampleVectorOut;
    plot(handles.axesInputTime, TimeAxisVector, SampleVector);
    set(handles.axesInputTime, 'YLim', [-0.8 0.8])
    plot(handles.axesOutputTime, TimeAxisVector, SampleVectorOut);
    set(handles.axesOutputTime, 'YLim', [-0.8 0.8])
end


% Update handles structure
guidata(hObject, handles);

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


% --- Outputs from this function are returned to the command line.
function varargout = Timeplot_GUI_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 button press in closeButton.
function closeButton_Callback(hObject, eventdata, handles)
% hObject    handle to closeButton (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
close(handles.Timeplot_GUI);
```

```
function XLimitLow_Callback(hObject, eventdata, handles)
% hObject    handle to XLimitLow (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
string = get(hObject, 'String');
vector = str2vec(string);
g = get(handles.axesInputTime, 'XLim');
if g(2) <= vector(1)
    errordlg('The lower limit must be lower than the upper limit',...
        'Limit error');
elseif (vector(1) < 0) || (2 <= vector(1))
    errordlg('The limit value is limited to a value between [0 2).',...
        'Limit error');
else
    if isempty(vector)
        set(handles.axesInputTime, 'XLim', [0 g(2)]);
        set(handles.axesOutputTime, 'XLim', [0 g(2)]);
        set(handles.XLimitLow, 'String', '0 s');
    else
        set(handles.axesInputTime, 'XLim', [vector(1) g(2)]);
        set(handles.axesOutputTime, 'XLim', [vector(1) g(2)]);
        set(handles.XLimitLow, 'String', [num2str(vector(1)) ' s']);
    end
end
% Hints: get(hObject,'String') returns contents of XLimitLow as text
%        str2double(get(hObject,'String')) returns contents of XLimitLow a


% --- Executes during object creation, after setting all properties.
function XLimitLow_CreateFcn(hObject, eventdata, handles)
% hObject    handle to XLimitLow (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 XLimitHigh_Callback(hObject, eventdata, handles)
% hObject    handle to XLimitHigh (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
string = get(hObject, 'String');
vector = str2vec(string);
g = get(handles.axesInputTime, 'XLim');
if vector(1) <= g(1)
    errordlg('The upper limit must be larger than the lower limit',...
        'Limit error');
elseif (vector(1) <= 0) || (2 < vector(1))
    errordlg('The limit value is limited to a value between (0 2].',...
        'Limit error');
else
    if isempty(vector)
        set(handles.axesInputTime, 'XLim', [g(1) 2]);
        set(handles.axesOutputTime, 'XLim', [g(1) 2]);
        set(handles.XLimitHigh, 'String', '2 s');
    else
        set(handles.axesInputTime, 'XLim', [g(1) vector(1)]);
        set(handles.axesOutputTime, 'XLim', [g(1) vector(1)]);
        set(handles.XLimitHigh, 'String', [num2str(vector(1)) ' s']);
    end
end

% Hints: get(hObject,'String') returns contents of XLimitHigh as text
%        str2double(get(hObject,'String')) returns contents of XLimitHigh


% --- Executes during object creation, after setting all properties.
function XLimitHigh_CreateFcn(hObject, eventdata, handles)
% hObject    handle to XLimitHigh (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 YLimitLow_Callback(hObject, eventdata, handles)
% hObject    handle to YLimitLow (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
string = get(hObject, 'String');
vector = str2vec(string);
g = get(handles.axesInputTime, 'YLim');
if g(2) <= vector(1)
    errordlg('The upper limit must be larger than the lower limit',...
        'Limit error');
else
    if isempty(vector)
        set(handles.axesInputTime, 'YLim', [-0.8 g(2)]);
        set(handles.axesOutputTime, 'YLim', [-0.8 g(2)]);
        set(handles.YLimitLow, 'String', '-0.8');
    else
        set(handles.axesInputTime, 'YLim', [vector(1) g(2)]);
        set(handles.axesOutputTime, 'YLim', [vector(1) g(2)]);
        set(handles.YLimitLow, 'String', [num2str(vector(1))]);
    end
end

% Hints: get(hObject,'String') returns contents of YLimitLow as text
%        str2double(get(hObject,'String')) returns contents of YLimitLow


% --- Executes during object creation, after setting all properties.
function YLimitLow_CreateFcn(hObject, eventdata, handles)
% hObject    handle to YLimitLow (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 YLimitHigh_Callback(hObject, eventdata, handles)
```

```
% hObject    handle to YLimitHigh (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
string = get(hObject, 'String');
vector = str2vec(string);
g = get(handles.axesInputTime, 'YLim');
if vector(1) <= g(1)
    errordlg('The upper limit must be larger than the lower limit',...
        'Limit error');
else
    if isempty(vector)
        set(handles.axesInputTime, 'YLim', [g(1) 0.8]);
        set(handles.axesOutputTime, 'YLim', [g(1) 0.8]);
        set(handles.YLimitHigh, 'String', '0.8');
    else
        set(handles.axesInputTime, 'YLim', [g(1) vector(1)]);
        set(handles.axesOutputTime, 'YLim', [g(1) vector(1)]);
        set(handles.YLimitHigh, 'String', [num2str(vector(1))]);
    end
end

% Hints: get(hObject,'String') returns contents of YLimitHigh as text
%        str2double(get(hObject,'String')) returns contents of YLimitHigh


% --- Executes during object creation, after setting all properties.
function YLimitHigh_CreateFcn(hObject, eventdata, handles)
% hObject    handle to YLimitHigh (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 infoLargeTimePlot.
function infoLargeTimePlot_Callback(hObject, eventdata, handles)
% hObject    handle to infoLargeTimePlot (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
p1 = 'This GUI present timeplot for both input signal and output signal\n\n';

p2 = ['Use the textboxes to ajust the limits of the Y-axis and X-axis  ',...
      'respectively. Although, there is not possible to have an X-axis',...
      'that show more than two seconds. This is because the data that',...
      'is sent to this GUI only has data from the past two seconds\n\n'];

  string = [p1 p2];

helpdlg(sprintf(string), 'Timeplot GUI info');
```

B.4 Main loop

```
function Simple_playGUI(hObject, handles)
% A simple program that samples music from audiofile and plays it
%%Initailization
SamplesPerFrame = 1024;
WishedSampleRate = handles.samplingRate;

% Checks whether it's supposed to read from file or from mic
if get(handles.filebutton, 'Value')   %If file
    fileName = get(handles.filename, 'String');
    if (exist(fileName,'file') == 2)
        % Samplerate is set by the file, not the user!
        FReader = dsp.AudioFileReader(fileName,'SamplesPerFrame',...
            SamplesPerFrame);
        Fs = FReader.SampleRate;    %Fs cannot be chosen
        set(handles.samplingfreq, 'String', Fs); %Signals files sampling
        play(hObject, handles, SamplesPerFrame, Fs, FReader);
    else
        errordlg('File not found in current directory','File Error');
        set(handles.onbutton, 'Value', 0);
    end
else               %If mic
    mic = dsp.AudioRecorder('SamplesPerFrame', SamplesPerFrame,...
        'SampleRate', WishedSampleRate, 'OutputDataType','double');
    Fs = mic.SampleRate;
    play(hObject, handles, SamplesPerFrame, Fs, mic);
end
end

function play(hObject, handles, SamplesPerFrame, Fs, inputObject)
```

```
%% Initialization of objects dependent on Fs
Player = dsp.AudioPlayer('SampleRate',Fs);

%% Initialization of stuff outside the loop
% Filters
numVec = handles.hannumeratorVector;
denomVec = handles.handenominatorVector;

matlabFilt = handles.matlabFiltration;

filterfunction = str2func('nofilter');
chosenFilter = handles.chosenFilter;
filterisobject = ~(ischar(chosenFilter));
if (Fs == 44100)
    prefabfilter = handles.chosenFilter;
    if ~filterisobject
        filterfunction = str2func(prefabfilter);
    end
else
    set(handles.prefabFilterpop, 'Value', 1);
    set(handles.applyfilter, 'Value', 0);
    set(handles.ownfilter, 'Value', 1);
end

ownfilterfunction = get(handles.filtername, 'String');
if (exist(ownfilterfunction))
    ownfilterfunction = str2func(ownfilterfunction);
else
    set(handles.filtername, 'String', 'Function not found!');
    set(handles.ownfunction, 'Enable', 'off');
    set(handles.filtername, 'Enable', 'off');
    set(handles.applyfilter, 'Value', 0);
    set(handles.ownfilter, 'Value', 1);

end

%---------- For timeplot -----------------------------------------------
Index = 1;

FramesPerSecond = floor(Fs/SamplesPerFrame);
SecondsPerFrame = 1/FramesPerSecond;
TimePlotSeconds = 2;
TimePlotFrames = FramesPerSecond*TimePlotSeconds;
```

```
reducedFlag = get(handles.reduced, 'Value');
if  reducedFlag == 1
    everyNth = 4;
else
    everyNth = 1;
end

SampleVector = zeros((TimePlotFrames*SamplesPerFrame)/everyNth,2);
SampleVectorOut = zeros((TimePlotFrames*SamplesPerFrame)/everyNth,2);
TimeAxisVector = linspace(0,2,(TimePlotFrames*SamplesPerFrame)/everyNth)';

AudioInPlot = plot(handles.axesTime, TimeAxisVector, SampleVector);
grid(handles.axesTime, 'on');
ylim(handles.axesTime, [-0.8 0.8]);
xlim(handles.axesTime, [0 2]);

%----------For frequencyplot-----------------------------------------------
dt = 1/Fs;          % SamplesPerSecond
FrequencyAxis = (0:(SamplesPerFrame-1))*Fs/SamplesPerFrame;
AudioInSpectra = zeros(2, length(FrequencyAxis));
AudioOutSpectra = zeros(2, length(FrequencyAxis));

% Find limits
[low, high] = findLimits(handles);

% Initiate plots
SpectrumInPlot = stem(handles.axesFreqIn, [FrequencyAxis, FrequencyAxis],...
    [AudioInSpectra(1,:) AudioInSpectra(2,:)]);
ylim(handles.axesFreqIn, [-0.02 0.01]);
xlim(handles.axesFreqIn, [low high]);

SpectrumOutPlot = stem(handles.axesFreqOut, [FrequencyAxis,...
    FrequencyAxis], [AudioOutSpectra(1,:) AudioOutSpectra(2,:)]);
xlabel(handles.axesFreqOut, 'Frequency [Hz]');
ylim(handles.axesFreqOut, [-0.02 0.01]);
xlim(handles.axesFreqOut, [low high]);

ylimitvector = zeros(1,100);

%% Play while onbutton is pushed
while get(handles.onbutton, 'Value')
```

```
% Read frame from file or mic
AudioIn = step(inputObject);

% Read gain from gain textbox
gain = str2double(get(handles.gain, 'String'));

% Filter, either by Matlabfilter or own filter
if get(handles.applyfilter, 'Value')
    if get(handles.matlabfilter, 'Value')
        AudioOut = gain*(filter(matlabFilt, AudioIn));
    elseif get(handles.prefabfilter, 'Value')
        if filterisobject
            AudioOut = gain*(filter(prefabfilter, AudioIn));
        else
            AudioOut = gain*(filterfunction(AudioIn, Index));
        end
    elseif get(handles.ownfunction, 'Value')
        AudioOut = gain*(ownfilterfunction(AudioIn, Index));
    else
        AudioOut = gain*(filter(numVec, denomVec, AudioIn));
    end
else
    AudioOut = gain*AudioIn;
end

% Play resulting audio
step(Player,AudioOut);

%% Time plot ----------------------------------------------------------
% Input signal Vector
if get(handles.pauseTime, 'Value')
    handles.TimeAxisVector = TimeAxisVector;
    handles.SampleVector = SampleVector;
    handles.SampleVectorOut = SampleVectorOut;
    guidata(hObject, handles);
else
    SampleVector = createYtimeplot(SampleVector, AudioIn, Index,...
        TimePlotFrames, SamplesPerFrame, everyNth);

    % Output signal Vector
    SampleVectorOut = createYtimeplot(SampleVectorOut, AudioOut,...
        Index, TimePlotFrames, SamplesPerFrame, everyNth);
    %---- Time Plot Plot---------------------------------------------------
    set(AudioInPlot, {'ydata'}, {SampleVector(:,1); SampleVector(:,2)});
end

Index = Index + 1;

%% FFT-plot---------------------------------------------------------------
if get(handles.pauseFFT, 'Value')
    [low, high] = findLimits(handles);
    xlim(handles.axesFreqIn, [low high]);
    xlim(handles.axesFreqOut, [low high]);
    handles.FFTin = AudioInSpectra;
    handles.FFTout = AudioOutSpectra;
    handles.freqXaxis = FrequencyAxis;
    guidata(hObject, handles);
else
    AudioInSpectra = (abs(fft(AudioIn))/(SamplesPerFrame))';
    AudioOutSpectra = (abs(fft(AudioOut))/(SamplesPerFrame))';

    set(SpectrumInPlot, {'ydata'},...
        {[AudioInSpectra(1,:) AudioInSpectra(2,:)]});
    set(SpectrumOutPlot,...
        {'ydata'}, {[AudioOutSpectra(1,:) AudioOutSpectra(2,:)]});

    %Automatic y-axis
    maxIn = max(max(AudioInSpectra));
    ylimitvector = circshift(ylimitvector,1,2);
    ylimitvector(1,1) = maxIn;
    staticMax = max(ylimitvector);
    yLow = -(staticMax/10);
    yHigh = staticMax + (staticMax/10);
    ylim(handles.axesFreqOut, [yLow (yHigh + 0.0000001)]);
    ylim(handles.axesFreqIn, [yLow (yHigh + 0.0000001)]);
end

%% Pausebutton
while (get(handles.pausebutton, 'Value'))
    % Set limits off the FFT-plots
    [low, high] = findLimits(handles);
    xlim(handles.axesFreqIn, [low high]);
    xlim(handles.axesFreqOut, [low high]);
    handles.TimeAxisVector = TimeAxisVector;
    handles.SampleVector = SampleVector;
```

```
        handles.SampleVectorOut = SampleVectorOut;
        handles.freqXaxis = FrequencyAxis;
        handles.FFTin = AudioInSpectra;
        handles.FFTout = AudioOutSpectra;
        guidata(hObject, handles);
        pause(0.1);
    end

    %% Miscellaneous

    if reducedFlag == 0
        if mod(Index, 2) == 0
            drawnow limitrate;
            pause(0.001);
        end
    else
        if mod(Index, 8) == 0
            drawnow limitrate;
            pause(0.001);
        end
    end

end
%% Terminate
release(inputObject);
release(Player);

end

function y = nofilter(input, varargin)
y = input;
end
```

B.5 Convert string to vector

```
function y = str2vec(String)
% A function that converts a string to a vector containing all the numbers
% inside the string by deleting everything but the numbers, in case the
% user happens to enter a letter etc.
if isempty(String)    % If user has deleted
    String = '[1]';   % Set default
    y = 1;            % Return default
else
```

```
    String = [String ' ']; % To make sure that the last isn't a number
    numeratorVector = [];  % Declare
    numberString = '';     % Declare
    for i = 1:length(String)         % Go through String
        nextChar = str2double(String(i));  % Retrieve char
        if (String(i) == '-' || String(i) == '.') % Special case for - and .
            numberString = [numberString String(i)];  % Just add it
            continue;
            % Coz there are 2 types if not a number, either [] or NaN.
            % This indicates that numberstring can be added as element to
            % numeratorVector
        elseif (isempty(nextChar) || isnan(nextChar))
            nextCoefficient = str2double(numberString);
            if isnan(nextCoefficient)  % Checks that numberstring isn't NaN
                numberString = '';      % If NaN, delete.
            else   % Just add number to numeratorVector.
                numeratorVector = [numeratorVector str2double(numberString)];
                numberString = '';      % Clear numberString.
            end
        else  % If longer number, like 999. Add whole number.
            numberString = [numberString String(i)];
        end
    end
    y = numeratorVector;
end
```

B.6 Create y for timeplot

```
function SampleVector = createYtimeplot(SampleVector, audio, Index,...
TimePlotFrames, SamplesPerFrame, everyNth)
%
%A function called that outputs the timeplot y-values
%
%

    if (Index < TimePlotFrames) % To fill up the x-axis
        SampleVector(Index*(SamplesPerFrame/everyNth):(Index+1)*...
            (SamplesPerFrame/everyNth)-1,:) = ...
            audio(1:everyNth:length(audio),:);
    else  % Else it shifts back a frame and puts a new frame first
        SampleVector = circshift(SampleVector,...
            (-SamplesPerFrame/everyNth),1);
        SampleVector((TimePlotFrames-1)*(SamplesPerFrame/everyNth)+1:...
            TimePlotFrames*(SamplesPerFrame/everyNth),:)...
            = audio(1:everyNth:length(audio),:);
    end
end
```

B.7 Create y for timeplot

```
function SampleVector = createYtimeplot(SampleVector, audio, Index,...
TimePlotFrames, SamplesPerFrame, everyNth)
%
%A function called that outputs the timeplot y-values
%
%

    if (Index < TimePlotFrames) % To fill up the x-axis
        SampleVector(Index*(SamplesPerFrame/everyNth):(Index+1)*...
            (SamplesPerFrame/everyNth)-1,:) = ...
            audio(1:everyNth:length(audio),:);
    else  % Else it shifts back a frame and puts a new frame first
        SampleVector = circshift(SampleVector,...
            (-SamplesPerFrame/everyNth),1);
        SampleVector((TimePlotFrames-1)*(SamplesPerFrame/everyNth)+1:...
            TimePlotFrames*(SamplesPerFrame/everyNth),:)...
            = audio(1:everyNth:length(audio),:);
    end
end
```

B.8 Find limits to x-axis in spectrum plot

```
function [x1,x2] = findLimits(handles)
% Reads the textboxes that specifies the limits
% Converts from string to numbers and returns the numbers

stringHigh = get(handles.xlimithigh, 'String');
high = str2double(stringHigh(1:length(stringHigh)-3)); % Delete 'Hz'
stringLow = get(handles.xlimitlow, 'String');
low = str2double(stringLow(1:length(stringLow)-3));    % Delete 'Hz'
if high <= low    % Error
    errordlg('Upper limit must be higher than lower limit!','Limit error');
    set(handles.xlimitlow, 'String', '0 Hz');
    low = 0;
end
x1 = low;
```

```
x2 = high;
end
```

B.9 Print Matlab filter information

```
function matlabfilterinformation(handles, filter)
    % A function that updates the matlabfilterinformation
    % Is used by more than one button
    % Simply composes a string and sets the text in text field

    part0 = 'Matlab filter information:';
    part1 = ['FrequencyResponse: ' filter.FrequencyResponse];
    part2 = [' ImpulseResponse: ' filter.ImpulseResponse];
    part3 = [' SampleRate: ' filter.SampleRate];
    part4 = [' DesignMethod: ' filter.DesignMethod];
    str = [part0 char(10) part1 part2 part3 part4];

    set(handles.matlabfilterinformation, 'String', str);
end
```

Appendix C

Matlab code used in results

C.1 Filter function for filtering sinuses in section 4.2.1

```
function audioOut = twosinuses(audioIn, index)

%designfilt('lowpassiir', 'FilterOrder', 2, 'PassbandFrequency', 2000,...
%'PassbandRipple', 1, 'SampleRate', 44100);

coeffs = [0.0171 0.0343 0.0171 1 -1.6562 0.7331];
numerator = coeffs(1:3);
denominator = coeffs(4:6);

audioOut = filter(numerator, denominator, audioIn);
```

C.2 Filter function for filtering Bach Air Suite No.3 in section 4.2.2

```
function audioOut = pingsbach(audioIn, index)
%
%The function that filters Bach Air from Suite No.3, given by Ping.
%

%filtbandstop = designfilt('bandstopiir', 'PassbandFrequency1', 2400,...
%'StopbandFrequency1', 2500, 'StopbandFrequency2', 2600,...
%'PassbandFrequency2', 2700, 'PassbandRipple1', 0.01,...
%'StopbandAttenuation', 60, 'PassbandRipple2', 0.01, 'SampleRate', 44100);

%filtLowpass = designfilt('lowpassfir', 'FilterOrder', 90,...
%'PassbandFrequency', 6000, 'StopbandFrequency', 6700, 'SampleRate', 44100);
```

```
coeffsBandstop = [0.9975    -1.8652    0.9975    1.0000    -1.8541    0.9950
    0.9975    -1.8652    0.9975    1.0000    -1.8759    0.9954
    0.9930    -1.8568    0.9930    1.0000    -1.8466    0.9858
    0.9930    -1.8568    0.9930    1.0000    -1.8667    0.9867
    0.9893    -1.8498    0.9893    1.0000    -1.8416    0.9781
    0.9893    -1.8498    0.9893    1.0000    -1.8578    0.9793
    0.9866    -1.8447    0.9866    1.0000    -1.8394    0.9727
    0.9866    -1.8447    0.9866    1.0000    -1.8500    0.9737
    0.9851    -1.8421    0.9851    1.0000    -1.8403    0.9701
    0.9851    -1.8421    0.9851    1.0000    -1.8439    0.9705];


coeffsLowpass = [-0.0040    0.0134    0.0061    0.0018    -0.0021    -0.0042...
     -0.0030    0.0008    0.0045    0.0049    0.0014    -0.0037    -0.0064...
    -0.0042    0.0016    0.0068    0.0071    0.0017    -0.0058    -0.0095...
    -0.0059    0.0029    0.0105    0.0105    0.0019    -0.0094    -0.0146...
    -0.0086    0.0054    0.0171    0.0165    0.0021    -0.0166    -0.0249...
    -0.0139    0.0113    0.0329    0.0317    0.0022    -0.0395    -0.0620...
    -0.0363    0.0438    0.1544    0.2502    0.2880    0.2502    0.1544...
    0.0438    -0.0363    -0.0620    -0.0395    0.0022    0.0317    0.0329...
    0.0113    -0.0139    -0.0249    -0.0166    0.0021    0.0165    0.0171...
    0.0054    -0.0086    -0.0146    -0.0094    0.0019    0.0105    0.0105...
    0.0029    -0.0059    -0.0095    -0.0058    0.0017    0.0071    0.0068...
    0.0016    -0.0042    -0.0064    -0.0037    0.0014    0.0049    0.0045...
    0.0008    -0.0030    -0.0042    -0.0021    0.0018    0.0061    0.0134...
    -0.0040];

audioCascade = sosfilt(coeffsBandstop,audioIn);
audioOut = filter(coeffsLowpass,1,audioCascade);
```

Bibliography

[1] *5.1-ljudkort, SC012, Sweex Europe BV.* https://www.elfa.se/sv/ljudkort-sweex-europe-bv-sc012/p/11066507?q=ljudkort&page=5&origPos=5&origPageSize=50&simi=95.32. Accessed: 2016-02-17.

[2] *A Guide to the Internet of Things Infographic.* http://www.intel.com/content/www/us/en/internet-of-things/infographics/guide-to-iot.html. Accessed: 2016-01-21.

[3] *Applications for Digital Signal Processors.* http://www.ti.com/lsds/ti/processors/dsp/applications.page. Accessed: 2016-01-21.

[4] John Thompson Bernard Mulgrew Peter Grant. *Digital Signal Processing: Concepts and Applications.* Second. Palgrave Macmillan, 2003. ISBN: 333-96356-2.

[5] *DSP System Toolbox.* https://se.mathworks.com/products/dsp-system.html. Accessed: 2016-02-17.

[6] Bonnie S. Heck Edward W. Kamen. *Fundamentals of Signals and Systems using the Web and Matlab.* Third. Pearson Education, 2007. ISBN: 0-13-168737-9.

[7] *EQ: Linear Phase vs Minimum Phase.* https://www.youtube.com/watch?v=efKabAQQsPQ. Accessed: 2016-02-14.

[8] *fft.* https://se.mathworks.com/help/matlab/ref/fft.html. Accessed: 2016-02-15.

[9] *FFTW.* http://www.fftw.org. Accessed: 2016-02-15.

[10] Russ Haines. *Digitalt ljud.* First. Academic Press, Inc, 2002. ISBN: 91-636-0699-2.

[11] John H. Karl. *An Introduction to Digital Signal Processing.* First. Academic Press, Inc, 1989. ISBN: 0-12-398420-3.

[12] *Matlab.* https://se.mathworks.com/products/matlab.html. Accessed: 2016-02-17.

[13] Boaz Porat. *A Course in Digital Signal Processing.* First. John Wiley & Sons, Inc, 1997. ISBN: 0-471-14961-6.

[14] *Sample- and Frame-Based Concepts.* https://se.mathworks.com/help/dsp/ug/sample-and-frame-based-concepts.html. Accessed: 2016-02-13.

[15] SCB. *Privatpersoners användning av datorer och internet.* Sept. 2016.

[16] Bob H. Lee Sen M. Kuo. *Real-Time Digital Signal Processing: Implementations, Application and Experiments with the TMS320C55X.* John Wiley & Sons, Inc, 2001. ISBN: 0-470-84534-1.

[17] *Soundcard tips and facts.* http://www.epanorama.net/documents/pc/soundcard_tips.html. Accessed: 2016-02-17.

[18] *The Origins of MATLAB.* https://se.mathworks.com/company/newsletters/articles/the-origins-of-matlab.html. Accessed: 2016-02-17.

[19] John G. Proakis Vinay K. Ingle. *Digital Signal Processing Using MATLAB.* Third. Cengage Learning, 2010. ISBN: 978-1-111-42737-5.

[20] *What Is a Sound Card? - Definition, Function Types.* `http://study.com/academy/lesson/what-is-a-sound-card-definition-function-types.html`. Accessed: 2016-02-17.