Digital Recursive Filters. A Tutorial for Filter Designers with Examples Implemented in Csound and SuperCollider

Themis G. Katsianos
Faculty of Music, McGill University, Montreal

September 1997

A thesis submitted to the Faculty of Graduate Studies and Research in partial fulfilment of the requirements of the degree of Master of Arts in Computer Applications in Music.

0-612-43893-7

Στους γονεις μου, στην μνημη της γιαγιας μου

και την Αννουλα.

Acknowledgements

My gratitude goes first and foremost to my supervisor, professor Bruce Pennycook. His trust in me, and his precise observations and suggestions were valuable and greatly appreciated. This work wouldn't have been possible without his guidance. As an undergraduate student in Music at Concordia University, I had the good fortune to be a student of professor Yaron Ross for three years. His passion and dedication have inspired me to a great degree. My parents have stood beside me and supported me all my life. Their love and faith in me have made me a stronger and better person. *Σας ευχαριστω Αντωνη και Μαρια.* Finally, I wish to thank Anne. Her love, understanding, and great smile made this happen.

Abstract

Filters constitute an essential tool for manipulating the spectral content of a signal. While there is a plethora of filtering tools, both in the hardware and software domain, the majority of them are geared towards engineers and scientists, rather than sound designers and electroacoustic composers. The "common-practice" approach is to consider filters as post-production tools. This can be restrictive if filters are to be used as artistic tools, dynamically involved in the shaping of the sound. This thesis was written with this approach in mind. Its aim is a) to provide a survey of the various digital recursive filters, enabling a filter designer to choose the one that suits his needs, b) to teach filter designers, such as electroacoustic composers and sound designers how to calculate digital filter coefficients, and c) implement filter algorithms using the familiar syntax of computer music languages such as *Csound* and *SuperCollider*.

Résumé

Les filtres constituent un outil essentiel pour la manipulation du contenu spectral d'un signal. Bien qu'il y ait abondance d'outils de filtrage, tant logiciels qu'au niveau "hardware", la plupart sont destinés aux ingénieurs et scientifiques plutôt qu'aux compositeurs et musiciens en électroacoustique. L'approche communément adoptée consiste à considérer les filtres comme des outils de post-production. Ceci peut être restrictif si une utilisation artistique des filtres est souhaitée durant le processus de création du son. C'est dans cette optique qu'a été écrite cette thèse. Ses objectifs sont a) de proposer une vue d'ensemble des divers filtres numériques récursifs pour permettre au compositeur de choisir le mieux adapté à ses besoins, b) d'enseigner au musicien et/ou ingênieur du son comment calculer les coefficients du filtre numérique, et c) d'implémenter les algorithmes de filtrage en utilisant la syntaxe de langages classiques en Informatique Musicale tels que *Csound* et *SuperCollider*.

Table of Contents

1. **Digital Filters. An Electroacoustic Approach** 1

1.1 Introduction 1
1.2 Available Digital Filtering Tools 2

2. **Filters in the Analog Domain** 6

2.1 Filters: An Introduction 6
2.2 Analog Transfer Functions 9
2.3 The Butterworth Response 12
2.4 The Chebyshev Response 13
2.5 The Inverse Chebyshev Response 15
2.6 The Elliptic Response 15
2.7 The Bessel Response 17
2.8 Comparison and Applications..19

3. **Filters in the Digital Domain** 20

3.1 Digital Filters 20
3.2 Digital Low-Pass Filter Design. The Bilinear Transformation Method 25
3.3 High-Pass Digital Filter Design 28
3.4 Band-Pass Digital Filter Design 29
3.5 Band-Rejection Digital Filter Design 31

4. **Digital Filter Design in Csound and SuperCollider** 33

4.1 A Complete Example Implemented in Csound 33
4.2 SuperCollider: A General Overview 37
4.3 A Complete Example Implemented in SuperCollider 42

5. Conclusion 51

Appendix A
Complete Analog Low-Pass Transfer Functions 54

A.1 Bessel Transfer Function 54
A.2 Butterworth Transfer Function 55
A.3 Elliptic or Cauer Transfer Function 56
A.4 Chebyshev Transfer Function 58
A.5 Inverse Chebyshev Transfer Function 59

Appendix B
Analog Bessel, Butterworth, Chebyshev, Inverse Chebyshev, and Elliptic Filter Data 62

Bibliography 66

References 66
Further Reading 67

1.

Digital Filters. A Sound Designer's Approach

1.1 Introduction

The concept of filters in music has been around since the invention of musical instruments. Although musicians may not realize it, a musical instrument such as a guitar or a drum can be considered a filter. The shape of the guitar body, for example, greatly affects the spectrum of the sound produced by the instrument. That is exactly what a filter does; it alters the characteristics of a sound as we shall see later in more detail. Our ears and our mouth are other familiar filter examples: they process and alter the characteristics of any sound we hear or produce.

Computer music composers and sound designers have undoubtedly come across filters in their effort to make their sounds realistic or even unrealistic. Who hasn't used reverb for example, even on the level of a basic sound synthesis course? The pioneers of computer music learned to process sounds using analog systems, since only analog circuits existed during the first stages of electronic music. Moog synthesizers included octave or 3rd-octave filters, as well as basic low-pass and high-pass filters, which were activated by plugging in cables to the right inputs. Later, in the late seventies and early eighties, the development of digital circuits led to the explosion of digital signal processing. At first, due to memory and speed limitations, multi-purpose systems capable of implementing various digital filter algorithms were limited. Later on, the introduction of micro-processors and the growth of personal computers changed the field drastically. Today, most personal computers are capable of real-time audio production and manipulation of any digital filter algorithm.

Unfortunately, the majority of tools for digital filtering today, is geared towards sound engineers rather than sound designers. The reason for this is that they are usually created with engineering applications in mind. On the other hand, there are applications for electroacoustic and computer music composers, but these are mostly editing and mixing tools which offer inadequate control mechanisms for precise filter manipulations. It was the author's observation and belief that electroacoustic and computer music composers should be able to use digital filtering tools based on their needs, and more important, use tools that they are familiar with. They should not have to be expert C programmers, or masters of complex analysis in order to be able to build digital filters.

This thesis is written with this premise in mind. As a result, I have chosen to gather the basic information necessary for a tutorial for filter designers on how to build digital recursive filters. Complex analysis was avoided and only basic algebra knowledge is needed to follow the text. The reader will find an analytical step by step method for building digital recursive filters using five different responses. The method used is that of the bilinear transform. It then becomes his choice as to what filter his work needs. Moreover, complete examples are given on how to implement the various digital filter algorithms, using computer music languages such as Csound and SuperCollider. As I stated previously, my rationale was that the filter, an instrument in itself, should be part of the code that produces the sound.

1.2 Available Digital Filtering Tools

Equalizers

Equalizers are devices whose primary function is to modify the characteristics of the signal passing through them. They do so by dividing the spectrum of the signal in three (low, mid, high) or more bands, for each one of which a fixed number of parameters such as gain, bandwidth, center frequency, etc., can be adjusted. Equalizers are found in recording-studio audio consoles, as separate hardware devices, and as computer software units (for example, as part of the effects of a software mixing/editing application). Equalizers can be classified in the following groups:

a) *Rotary-Knob Equalizers*. The equalizer's control mechanism is a rotary knob. Rotary potentiometers vary the amount of boost or cut, while rotary switches are usually used for frequency selection.

b) *Parametric Equalizers*. They are called parametric because they offer separate control of the center frequency, the bandwidth, and the amount of boost or cut for each frequency band.

c) *Graphic Equalizers*. They use a series of slide potentiometers to achieve the desired amount of boost or attenuation. The relative positions of the faders provide a graphic description of the output response.

d) *Digital Equalizers*. Digital equalizers are often equipped with both analog and digital signal inputs and outputs. Their advantage is that they are programable: a set of desired parameters can be stored in the memory for later use.

The problem with all types of equalizers is that they are more or less "black boxes" to an electroacoustic composer or sound designer. Information such as which type of response has been implemented, or what order of filter has been used is not readily available. With an equalizer you are basically "stuck" with the company's specifications.

Software Tools

There are numerous software tools available for digital filtering. Among them are:

Kyma [1] a graphical design language of real-time sounds and effects, that runs on both Macintosh and Windows platforms. The user can design sounds by connecting modules chosen from a palette of prototypes. Among its capabilities are waveshaping, sampling, spectral analysis/resynthesis, and filtering. The "catch" with *Kyma* is that you need *Capybara* to run it. *Capybara* is a multiprocessor DSP box used to realize the sounds you design using *Kyma*.

Matlab [2] a commercial interactive system and programming language for general scientific and technical computation and visualization. It provides an extensive number of "toolboxes", which are prepackaged functions and visual tools. Among them is the *Signal Processing Toolbox*, which can be used for the design of any analog or digital filter. Leading edge signal processing technologies, such as wavelets and advanced spectral analysis are included, as well as graphical demonstrations that allow for exploration of the various filter parameters.

Filter Designer [3] created to serve as a tool for electroacoustic composers. It provides a graphical environment for digital filter design and implementation, giving the user control over the various filter parameters. The program can process sound files either as a stand-alone application, or in conjunction with *Csound*. Among the responses used are Butterworth, Chebyshev, and Inverse Chebyshev.

Music-oriented software applications also include digital signal processing units. An example is *Deck II* [4]. *Deck II* is a digital mixing and editing program, which replaces the traditional multitrack tape recorder, the tape editing block, and the mixing board. Instead of recording sound to tape, it is stored in a digital format on a hard drive. Its effects menu includes delays, normalization, parametric EQs, etc. As in the case of equalizers, the user doesn't know and cannot choose the type of response or the order of filter used. As well, *Deck II* cannot be used as a compositional tool (at least not in the sense of creating audio signals as in the case of computer music languages).

Computer Music Languages

Sound synthesis and signal processing languages are used widely by electroacoustic and computer music composers for composition and manipulation of sound. It is of no surprise then that such languages offer filtering tools. Examples of such languages are *Cmix* [5], *Csound* [6], and *SuperCollider* [7].

Cmix was developed by Paul Lansky at Princeton University as a tool for processing sound files. It contains both independent programs for operating on soundfiles, as well as a collection of routines which aid in writing sound-processing code in C. It offers filters such as all-pass, all-pole, comb, as well as *ellipse*. *Ellipse* is a stand-alone *Cmix* program which processes signals through an elliptical filter, whose coefficients are generated by *filter*, a Fortran77 program which provides a set of coefficients based on the user's input of design values.

Csound was developed by Barry Vercoe at the Media Lab of M.I.T. Audio signals can be routed through various processing units. Among them are filters or signal modifiers such as *tone*, a 1st-order recursive low-pass filter, a series of Butterworth 2nd-order filters (low, high, band-pass, band-reject), etc. As in the case of *Cmix*, the user can write his own *Cmodules*, i.e. create his own signal generators and modifiers, and import them to *Csound*.

Both *Csound* and *Cmix* are ASCII based environments. Furthermore, the user has to define the parameters controlling his "instrument" before execution of the program, and can only change them between runs. This is not the case with

SuperCollider. *SuperCollider*, developed by James McCartney at the University of Texas Austin, is a sound synthesis programming language that offers real-time manipulation of the various parameters controlling the signals. This is achieved by means of a *Graphical User Interface* (GUI), which offers sliders, buttons, and other easily edited control tools. While the current SuperCollider version does not support C-imports, the author has announced that this capability will be included in the next version. Basic filters are available such as one-pole high-pass filters, band-pass filters, etc.

Languages such as *Csound* and *SuperCollider* are the ideal environment for creating filtering tools. Their syntax is well understood by most electroacoustic composers, and they can choose to implement filters either as a C-imported module, or by just using the signal processing tools built in the languages, which is the approach of this thesis.

2.

Filters in the Analog Domain

2.1 Filters: An Introduction

Filters are used to modify the amplitude and phase characteristics of the spectral components of a signal, without altering any of its frequency constituents. The characteristics of a filter are described by its *frequency response*, which can be derived experimentally. A test sine wave, with given amplitude and phase characteristics, can be used as an input to the filter. The ratio of the amplitude of the output signal to the amplitude of the input signal gives the *amplitude response* of the filter, and the ratio of the phase of the output signal to the phase of the input signal gives its *phase response*. The combination of amplitude and phase response gives us the response of the filter in the frequency domain. The characteristics of a filter can be described in the time domain as well, by what is known as its *impulse response*. The impulse response of a filter describes the signal emitted by the filter in response to an impulse, which is a unit sample input.

Filters can be classified into four basic groups according to the shape of their amplitude response. *Low-Pass* filters let all frequencies below a chosen cutoff frequency f_c to pass through unaffected,while attenuating all frequencies above it. Figure 2.1 shows an example of the amplitude response of a low-pass filter. *High-Pass* filters on the other hand, will let all frequencies above a chosen cutoff frequency f_c to pass through unaffected, while attenuating all frequencies below it. Figure 2.2 is an example of the amplitude response of a high-pass filter. As figures 2.1 and 2.2 illustrate, the point of transition from the stop-band to the pass-band and vice versa, defines the position of the cutoff frequency f_c, and is usually taken at the point where the amplitude response changes by a factor of .707 (or -3 dB).

Band-Pass filters will let all frequencies within a selected band to pass through unaffected, while attenuating all frequencies outside the band. The band is defined by its *bandwidth* BW, which is the difference between an upper and a lower cutoff frequency $f_u - f_l$. A band-pass filter is also characterized by its *center frequency*, $f_0 = \frac{f_u + f_l}{2}$. A *quality factor Q* can be defined to describe the characteristics of band-pass filters, as $Q = (f_0 / BW_{3dB})$. Figure 2.3 shows the amplitude response of a band-pass filter. *Band-reject* filters will attenuate all frequencies within a selected band, while letting through unaffected all frequencies outside the bandwidth. They are also characterized by a center frequency f_0, and a bandwidth BW. Figure 2.4 shows the amplitude response of a band-reject filter.

This thesis studies five different types of amplitude responses: a) the *Butterworth* response, b) the *Chebyshev* response, c) the *Inverse Chebyshev* response, d) the *Bessel* response, and e) the *Elliptic* response. Before we examine these responses in more detail, we have to present the concept of *analog transfer functions*.

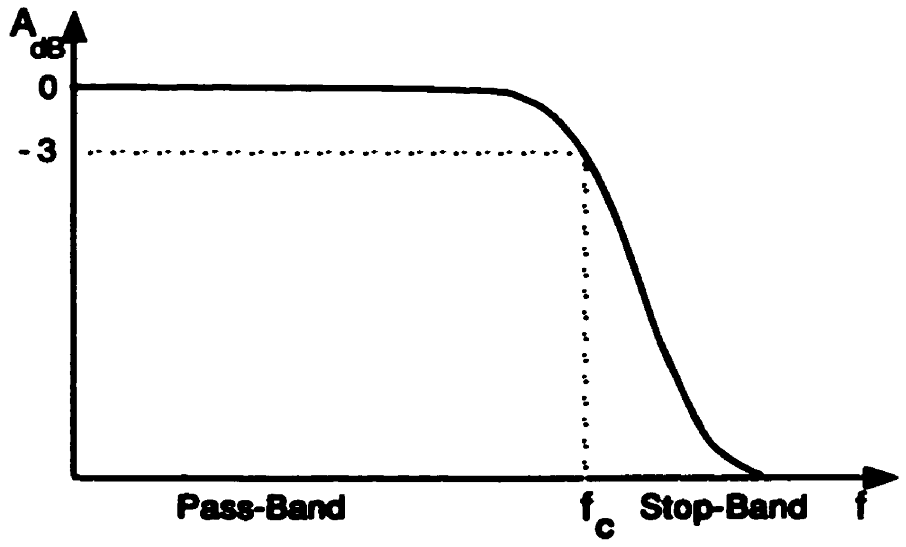

Figure 2.1: Amplitude response of a low-pass filter.

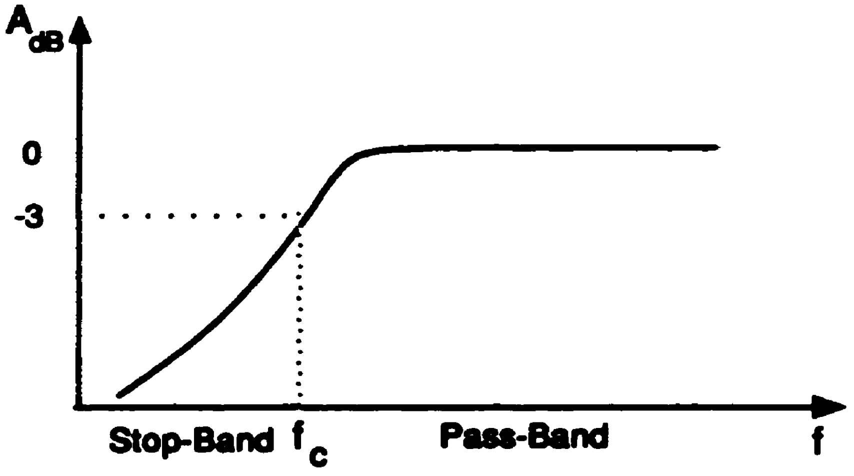

Figure 2.2: Amplitude response of a high-pass filter

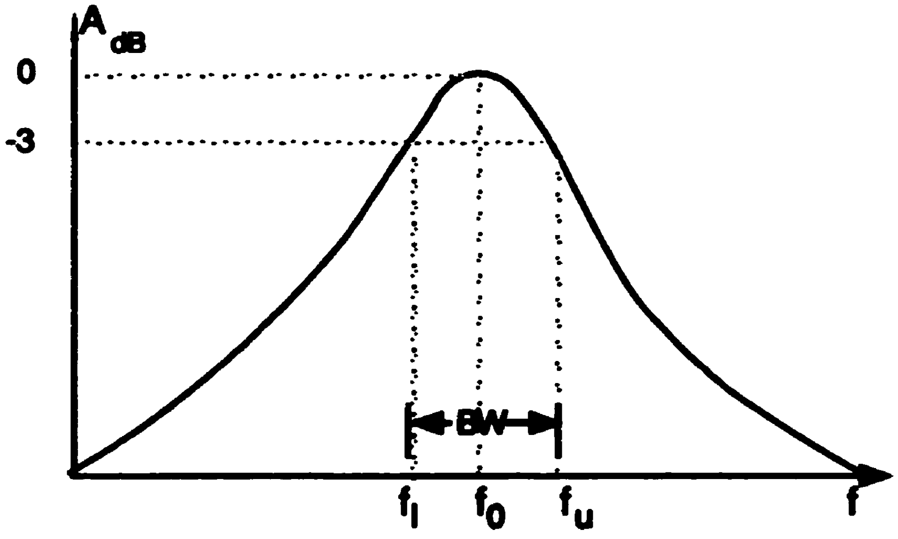

Figure 2.3: Amplitude response of a band-pass filter.

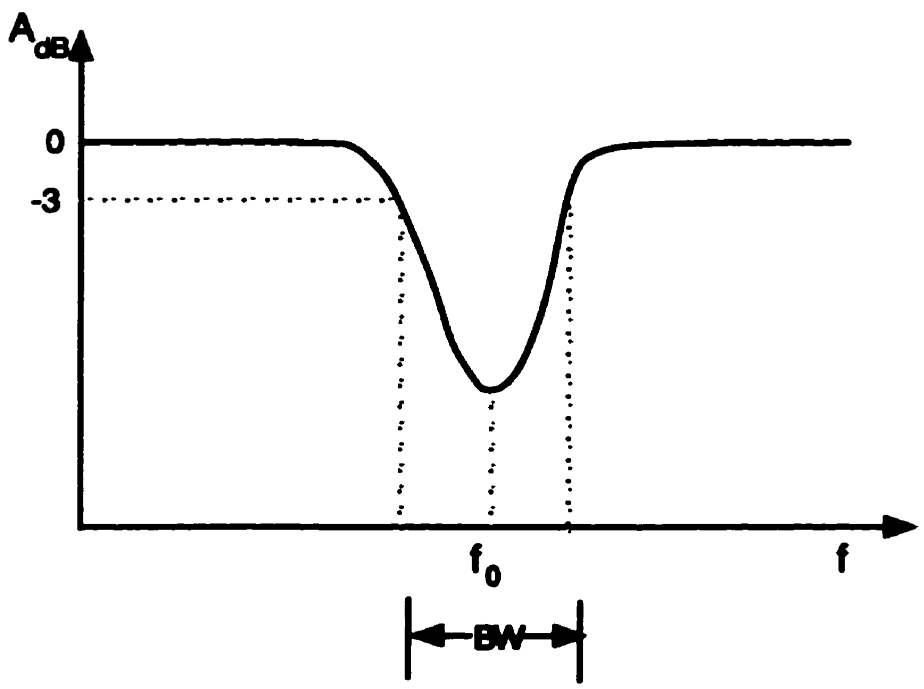

Figure 2.4: Amplitude response of a band-reject filter.

2.2 Analog Transfer Functions

Let us start with some definitions. A system is said to be *linear* if it obeys the *superposition principle*: if the system's responses are $y_1(t)$, $y_2(t)$, for $x_1(t)$, $x_2(t)$ inputs respectively, then the system's response to the input $ax_1(t)+bx_2(t)$ is $ay_1(t)+by_2(t)$.

A system can also be *time-invariant*, if its response does not change with time, and *causal*, if its output depends on present and past inputs, as well as past outputs. Such a system, which is also continuous in time, i.e. ***analog***, can be described by a function of the form:

$$G(s)=\frac{a_j s^j+a_{j-1}s^{j-1}+\ldots+a_0}{b_l s^l+b_{l-1}s^{l-1}+\ldots+b_0} \qquad (2.1)$$

which is known as the ***transfer function*** of the system. The variable s is related to the frequency ω of the system by a relation of the form $s=\sigma+i\omega$, which represents a mathematical model of analysis (complex analysis), that can be used for finding the ***zeros*** and ***poles*** of eq. (2.1), or, in other words, the values for which the numerator and denominator of eq. (2.1) become zero respectively. For example, let us consider a simple transfer function of the type:

$$G(s)=\frac{1}{s^2+\sqrt{2}\,s+1} \qquad (2.2)$$

If we try to find the roots of the denominator of (2.2), i.e. the values that make it zero, we are bound to come across the value $\sqrt{-1}$. Complex analysis overcomes that obstacle by simply defining an imaginary unit i, such that $i^2=-1$. While it has no direct physical meaning, it allows us to express the roots of eq. (2.2). These values can be depicted on the s-plane, as we can see in Figure 2.5.

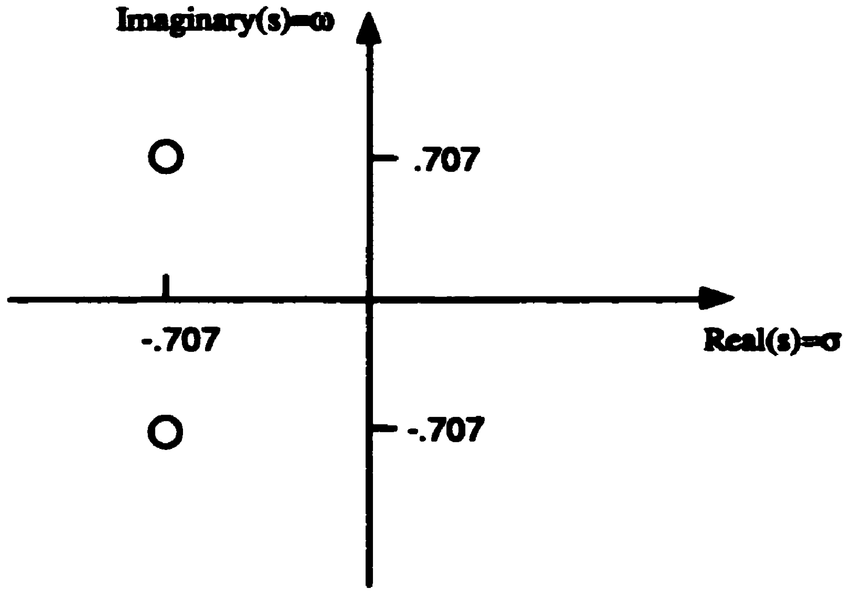

Figure 2.5: Transfer function poles on the s-plane

It can be shown [8] that the analog transfer function describing a system is related to its amplitude response *A(f)*, by an equation of the form:

$$A^2(f)=G(s)G(-s) \tag{2.3}$$

The reason (2.3) is very important is that for a given amplitude response *A(f)*, the design of the corresponding digital filter will be based on the analog prototype function *G(s)*, as we will see later in the text.

2.3 The Butterworth Response

The Butterworth low-pass amplitude-squared function is defined by:

$$A^2(f)=\frac{1}{1+\varepsilon^2(\frac{\omega}{\omega_c})^{2k}}=\frac{1}{1+\varepsilon^2(\frac{f}{f_c})^{2k}} \qquad (2.4)$$

where k represents the order of the corresponding transfer function, and

$$\omega_c=2\pi f_c$$
$$\varepsilon=(10^{-.1a_{pass}}-1)^{\frac{1}{2}} \qquad (2.5)$$

In eq. (2.5) ω_c (or f_c) is the radian (or cyclic) pass-band edge frequency of the filter, and a_{pass} its pass-band amplitude. The Butterworth response is often called the maximally flat response, because no other response has a smoother transition through the pass-band to the stop-band. The phase response is also very smooth, making it the ideal candidate when low phase distortion is required. Equation (2.4) shows that when $\omega=0$, the response will have unity gain, whereas for $\omega=\omega_c=1$, the gain at the normalized pass-band edge will depend on the value of ε, which, as eq. (2.5) shows, is related to the pass-band ripple. In many developments, it is convenient to normalize the frequency scale by selecting $\omega_c=1\ rad/s$. This allows for considerable freedom in designing filters, since a normalized transfer function can easily be unnormalized to any other frequency. The order of a Butterworth filter depends on the specifications set by the user. For given amplitudes and frequencies at the pass-band and stop-band respectively, the order is given by:

$$n_B=\frac{\log[(10^{-.1a_{stop}}-1)/(10^{-.1a_{pass}}-1)]}{2\log(\omega_{stop}/\omega_{pass})} \qquad (2.6)$$

Figure 2.5 shows a typical Butterworth low-pass amplitude response.

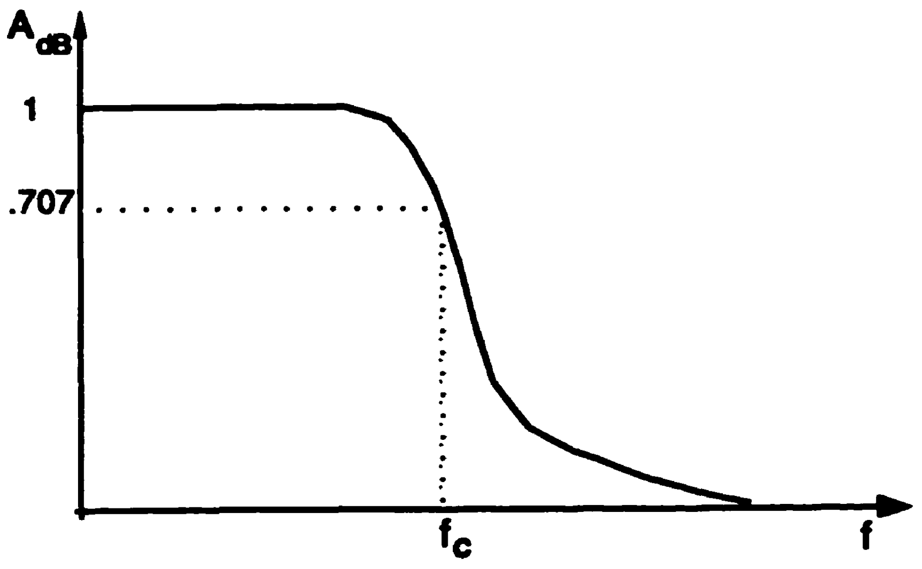

Figure 2.5: Amplitude response of a low-pass Butterworth filter, for n=6.

2.4 The Chebyshev Response

The basic Chebyshev amplitude response is defined by:

$$A^2(f)=\frac{\alpha}{1+\varepsilon^2 C_k^2(\omega/\omega_c)}=\frac{\alpha}{1+\varepsilon^2 C_k^2(f/f_c)} \tag{2.7}$$

where k represents both the order of the Chebyshev polynomial and the order of the corresponding transfer function, and ε is given by eq. (2.5). Chebyshev polynomials can be calculated by:

$$\begin{aligned} C_k(\omega)&=\cos[k\cos^{-1}(\omega)],\ \omega\leq 0\\ C_k(\omega)&=\cosh[k\cosh^{-1}(\omega)],\ \omega>0 \end{aligned} \tag{2.8}$$

The constant a determines the proper dc gain level. When the desired maximum gain is unity, $a = 1$.

Chebyshev filters allow for variation or ripple in the pass-band of the filter. Their transition characteristics are steeper in comparison to the Butterworth ones. This means that certain user specifications can be satisfied with lower order Chebyshev than with Butterworth filters. However, their phase response is not as linear as the phase response of Butterworth filters. For user-selected pass-band and stop-band frequencies and amplitudes, the order of a Chebyshev filter is given by:

$$n_c = \frac{\cosh^{-1}\{[(10^{-.1\alpha_{stop}} - 1)/(10^{-.1\alpha_{pass}} - 1)]^{\frac{1}{2}}\}}{\cosh^{-1}(\omega_{stop}/\omega_{pass})} \tag{2.9}$$

Figure 2.6 shows the form of the Chebyshev amplitude response.

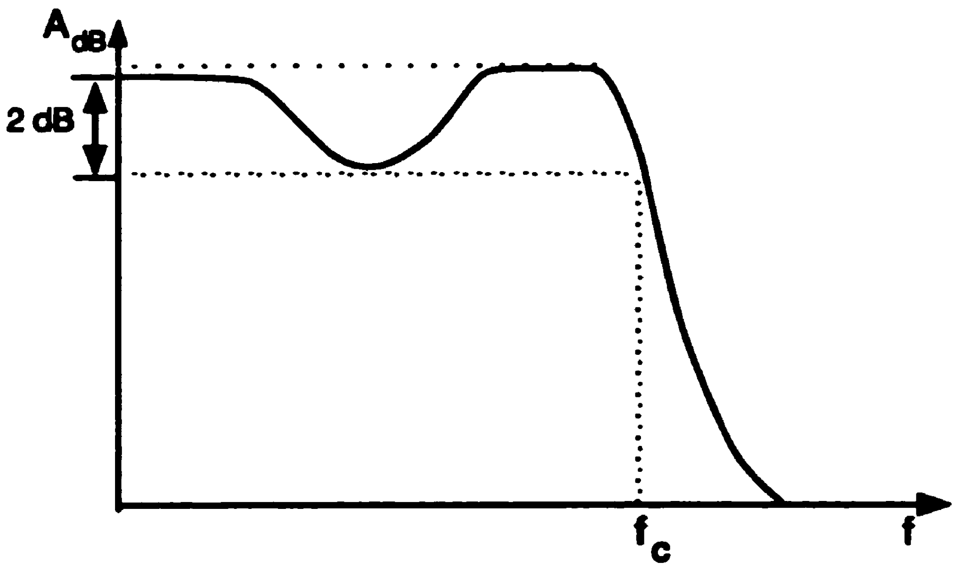

Figure 2.6: Chebyshev amplitude response for $k = 3$ and 2 dB pass-band ripple.

2.5 The Inverse Chebyshev Response

The amplitude-squared Inverse Chebyshev response is defined by:

$$A^2(f)=\frac{\varepsilon^2 C_N^2(\omega_s/\omega)}{1+\varepsilon^2 C_N^2(\omega_s/\omega)}=\frac{\varepsilon^2 C_N^2(f_s/f)}{1+\varepsilon^2 C_N^2(f_s/f)} \tag{2.10}$$

with ε given by eq. (2.5) and the Chebyshev polynomials by eq. (2.8).

Whereas the response of ordinary Chebyshev filters is equiripple in the pass-band and maximally flat in the stop-band, the Inverse Chebyshev response has a maximally flat magnitude in the pass-band and equal ripples in the stop-band. The frequency ω_s (or f_s) now defines the beginning of the stop-band. The Inverse Chebyshev response provides better transition characteristics than the Butterworth response, and better phase response than the standard Chebyshev response. For user-specified frequencies and amplitudes at the pass-band and the stop-band respectively, the order of an Inverse Chebyshev filter is given by:

$$n_{IC}=\frac{\cosh^{-1}\{[(10^{-.1\alpha_{stop}}-1)/(10^{-.1\alpha_{pass}}-1)]^{\frac{1}{2}}\}}{\cosh^{-1}(\omega_{stop}/\omega_{pass})} \tag{2.11}$$

We can observe that the order of an Inverse Chebyshev filter is given by exactly the same formula as the standard Chebyshev filter. Figure 2.7 shows the form of the Inverse Chebyshev amplitude response.

2.6 The Elliptic Response

The Cauer or Elliptic low-pass amplitude-squared function is defined by:

$$A^2(\omega)=\frac{1}{1+\varepsilon^2 R_N^2(\omega)} \tag{2.12}$$

where ε is given by eq. (2.5), ω is the radian frequency, and $R_n(f)$ are the ***Chebyshev Rational Functions***, given by:

$$\begin{aligned} R_n(\omega) &= \prod_{i=1}^{k} \frac{\omega^2-\Omega_i^2}{1-\omega^2\Omega_i^2}, \text{ if } n \text{ is even} \\ R_n(\omega) &= \omega \prod_{i=1}^{k} \frac{\omega^2-\Omega_i^2}{1-\omega^2\Omega_i^2}, \text{ if } n \text{ is odd} \end{aligned} \tag{2.13}$$

In equation (2.13), Ω_i represents the zeros of the functions $R_n(\omega)$. Elliptic filters are characterized by equiripple response in both the pass-band and the stop-band. Figure 2.8 shows the form of the elliptic function response for $k = 5$. With elliptic filters is possible to specify both the maximum ripple level in the pass-band, and the minimum attenuation level in the stop-band.

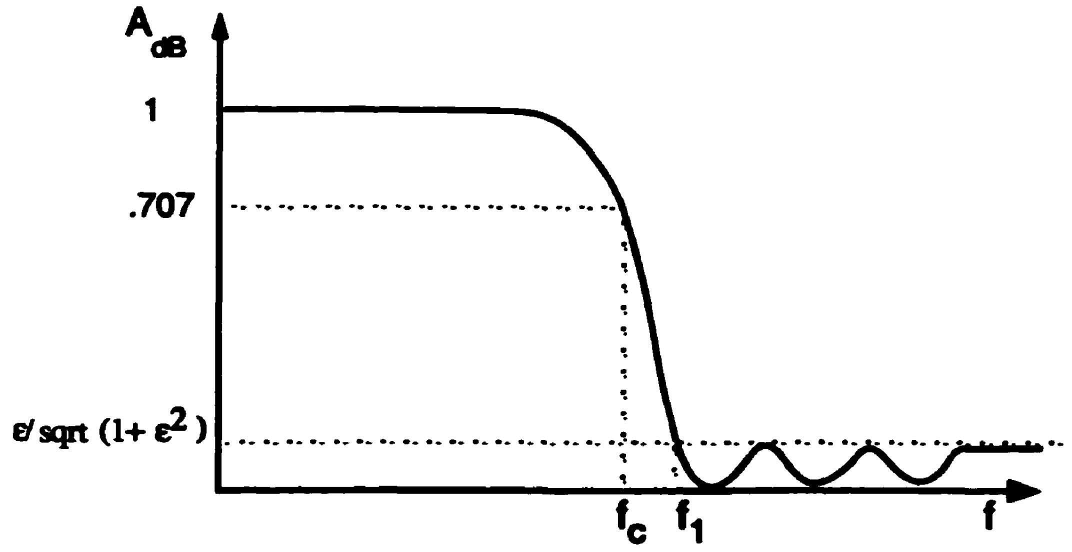

Figure 2.7: Inverted Chebyshev amplitude response for $k = 6$.

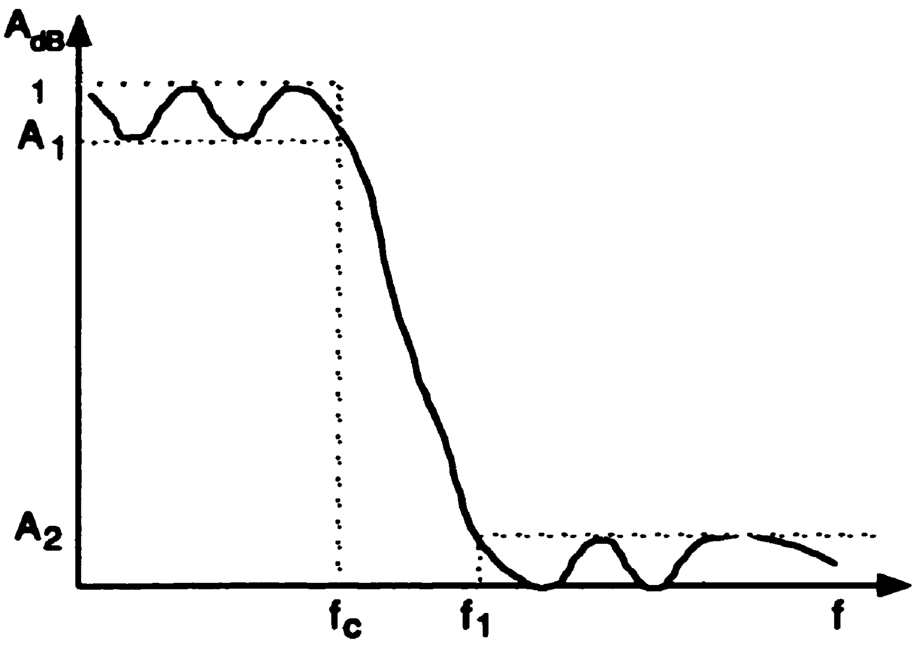

Figure 2.8: Elliptic function amplitude response with k = 5.

2.7 The Bessel Response

The amplitude-squared Bessel response is given by:

$$A^2(f)=\frac{2b_0^2}{\pi\omega^{2n+1}[J_{+v}^2(\omega/\omega_c)+J_{-v}^2(\omega/\omega_c)]}=\frac{2b_0^2}{\pi\omega^{2n+1}[J_{+v}^2(f/f_c)+J_{-v}^2(f/f_c)]} \tag{2.14}$$

where $J_{\pm v}(\omega)$ are ***Bessel polynomials*** [Appendix A]. The constant b_0 of the numerator is given by:

$$b_0 = \frac{(2n)!}{2^n n!} \qquad (2.15)$$

Eq. (2.15) shows that, for a given order of filter n, the value of b_0 and therefore the numerator of (2.14) is constant. Therefore, only the denominator can become zero for certain frequencies, which makes Bessel filters all-pole filters. The important parameter of these filters is their linearity of phase. Linearity of phase assures constant time delay, which means that time delay is independent of frequency. Time delays occur naturally in the transmission of a signal through space, such as on a coaxial cable, but can also be provided by delay filters. Because of their property, Bessel filters are also known as MFTD or *Maximally Flat Time-Delay* filters. Their pass-band response is not as flat as for the Butterworth response, and their stop-band attenuation is not as great at a given frequency as for either the Butterworth, or the Chebyshev response of the same order. Figure 2.9 shows the form of the amplitude response for a Bessel type filter.

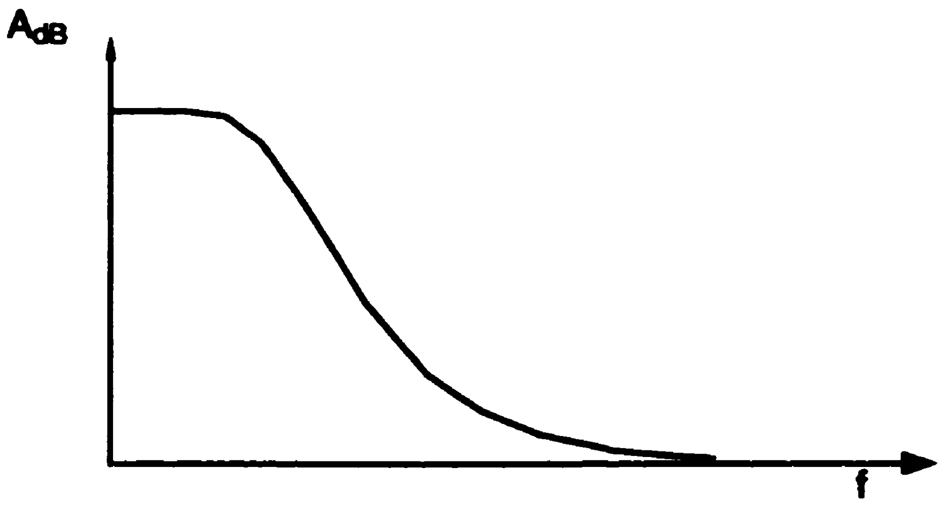

Figure 2.9: Form of the amplitude response of an MFTD filter.

2.8 Comparison and Applications

The responses we have examined can be classified into two groups. The Butterworth, Chebyshev, Inverse Chebyshev, and Elliptic responses constitute the first group. They are all characterized by a non-linear phase response. The Bessel response belongs to the second group, and is characterized by a fairly linear phase response. The choice of a response from one of the two groups depends on the type of application. Applications such as data transmission, where the signal is to be interpreted by digital hardware, and image processing, where the signal is used to reconstruct an image which is to be interpreted by the human eye, require the delay characteristic to be fairly flat. Bessel filters are more suitable for such applications.

On the other hand, the human ear is quite tolerable to phase distortion. Therefore, applications such as speech, or musical signal processing can use a type of response from the first group. The choice of a filter response from the first group depends again on the application. In terms of phase characteristics, Butterworth filters have the smoothest phase curve, followed by Inverse Chebyshev filters, Chebyshev filters, with the last choice being the Elliptic filters. In other applications, where time-domain characteristics are important, ripples in the frequency response cause irregularities, such as echoes in the time response. For such applications, Butterworth and Inverse Chebyshev filters are more desirable.

Butterworth filters are the ones most commonly used in the industry. They are mathematically simple, and they offer smooth gain response, as well as reasonably good phase response. However, for some applications they are unsuitable. Such applications require a more uniform transmission of frequencies in the pass-band, as well as a sharper rate of cutoff. Instead of achieving this by increasing the order of the Butterworth filter, Chebyshev filters can be used. Chebyshev filters are very useful in applications where the magnitude of the transfer function is of primary concern. If the equal ripples are desired in the stop-band, and the pass-band is flat, then Inverse Chebyshev filters can be used. If the equal-ripple property is required in both pass-band and stop-band, then Elliptic filters can be used. Elliptic filters offer excellent attenuation versus filter order but very poor phase response, and should be used only when the phase response is not critical.

3.

Filters in the Digital Domain

3.1 Digital Filters

Digital filters are used to modify the spectrum of a digital signal. We can think of them as "black boxes", which have a signal input, an in-the-box method used to process the signal, and a signal output. The way the "black box" or digital filter operates is described by its complex transfer function $H(z)$, where z is the complex frequency variable related to the frequency of the signal by an equation of the form $z = e^{i\omega}$. The amplitude $|H(z)|$ of the digital transfer function describes the frequency response of the filter, whereas its phase, phase($H(z)$), describes the filter's phase response. The z *transform* provides a means of describing a waveform in the z domain. If $x(n)$ is a discrete waveform, its z transform is given by:

$$X(z) = \sum_{n=-\infty}^{\infty} x(n) z^{-n} \qquad (3.1)$$

The term z^{-n} denotes an n_{th} sample delay operator. For example, z^{-1} will delay the signal one sample, i.e. $x(n)z^{-1} = x(n-1)$.

The usefulness of $H(z)$ is depicted in Figure 3.1:

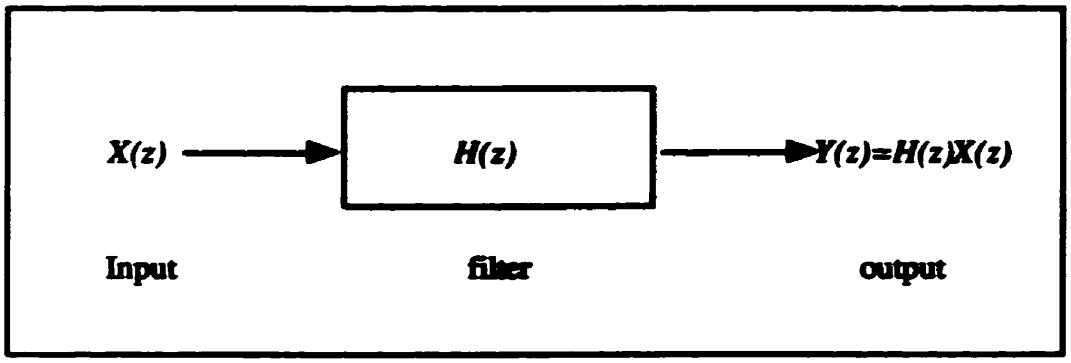

Figure 3.1: Operation of a digital filter in the z domain.

If we know *H(z)* for a filter, we can calculate the *z* transform of its response to a signal by multiplying *H(z)* by the *z* transform of the input signal. In other words,

$$H(z)=\frac{Y(z)}{X(z)} \tag{3.2}$$

If a digital filter is linear, causal, and time-invariant, its output can be described by an equation of the form [9]:

$$y(n)=\sum_{i=0}^{M} a_i x(n-i)-\sum_{i=1}^{N} b_i y(n-i) \tag{3.3}$$

If the second part of eq. (3.3) is equal to zero, i.e. all coefficients b_i are 0, the filter's output depends only on present and past inputs. Such filters are called *finite impulse response* filters or FIR, since zero input will eventually cause zero output. If, on the other hand, one or more of the b_i coefficients are non-zero, the filter's response depends on previous outputs as well, and in theory can last for ever. Such filters are called *infinite impulse response* filters, or IIR. They are also called recursive, since some part of the output is fed back into the filter's input. Digital recursive or IIR filters are said to be *unstable*, because their output can grow

infinitely. The focus of this thesis is on such IIR filters. More specifically, analytical methods are presented for building digital recursive filters whose transfer function follows responses such as *Butterworth*, *Chebyshev*, *Inverse Chebyshev*, *Bessel*, and *Elliptic*. Figure 3.2 shows a typical realization example of an IIR filter:

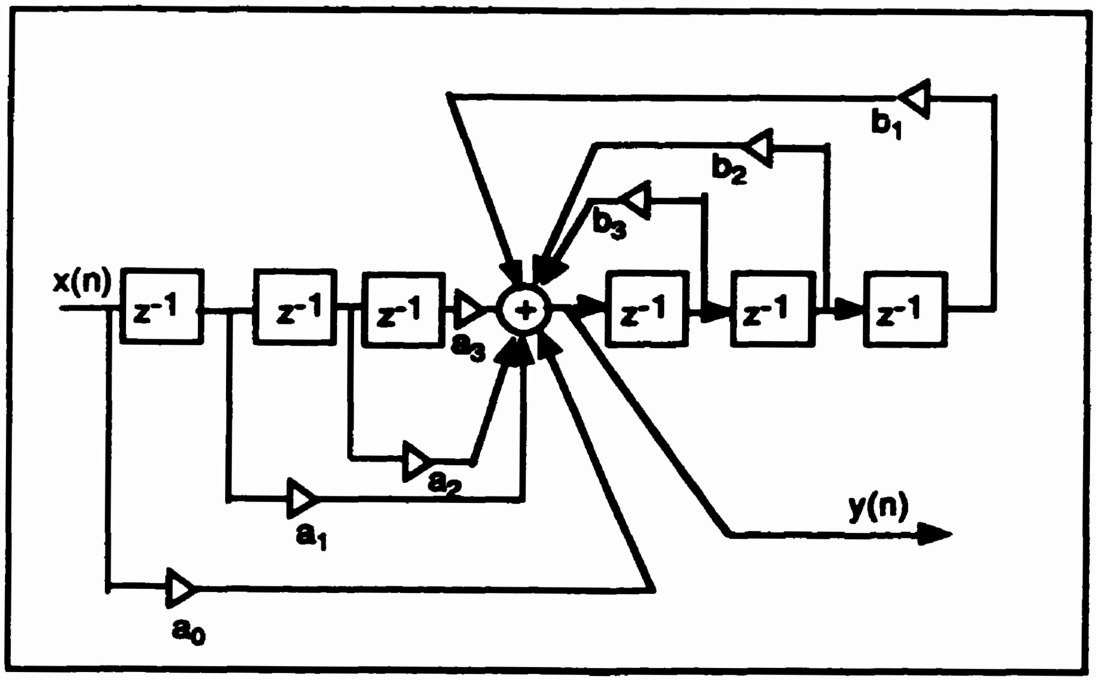

Figure 3.2: A typical realization of a 3rd-order IIR filter.

Let us now find an analytical expression for the digital transfer function *H(z)*, by applying the transformation of eq. (3.1) to eq. (3.3). We get:

$$Y(z)=\sum_{i=0}^{M} a_i z^{-i} X(z)-\sum_{i=1}^{N} b_i z^{-i} Y(z) \tag{3.4}$$

Combining now (3.2) and (3.4) we get:

$$H(z)=\frac{a_0+a_1z^{-1}+\ldots+a_Mz^{-M}}{1+b_1z^{-1}+\ldots+b_Nz^{-N}} \tag{3.5}$$

Both polynomials of the numerator and denominator of (3.5) can be factored, i.e. written as products of first order terms:

$$H(z)=a_0\frac{\prod_{i=1}^{M}(1-Z_iz^{-1})}{\prod_{i=1}^{N}(1-P_iz^{-1})} \tag{3.6}$$

Equation (3.6) is very useful because it gives what is called the *zeros* and *poles* of the filter. Zeros are all the values Z_i that will cause the numerator to become zero, and poles are all the values P_i, that will cause the denominator to become zero. Zeros cause a valley, whereas poles place peaks in the amplitude response of the filter. Figure 3.3 shows examples of zeros and poles.

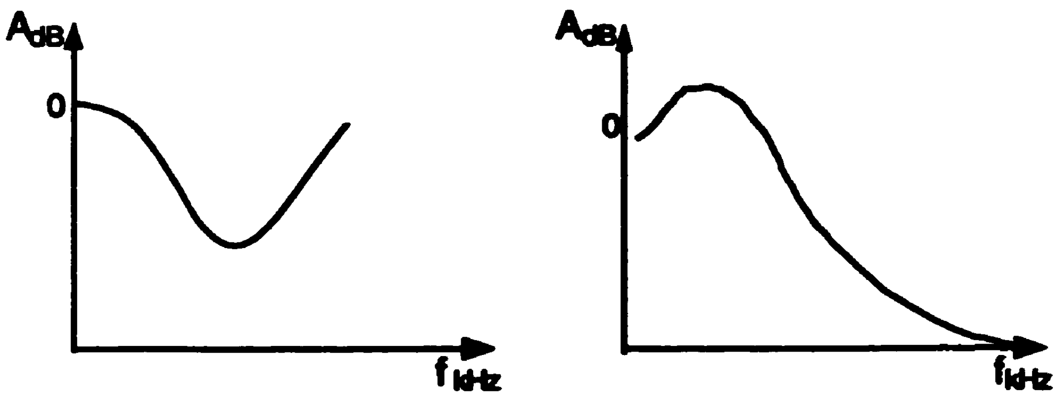

Figure 3.3: Amplitude responses of an all-zero, and an all-pole filter.

Similarly to the representation of the poles and zeros of an analog transfer function on the s-plane (section 2.2), the poles and zeros of a digital transfer function can be depicted on the z-plane, as shown in Figure 3.4:

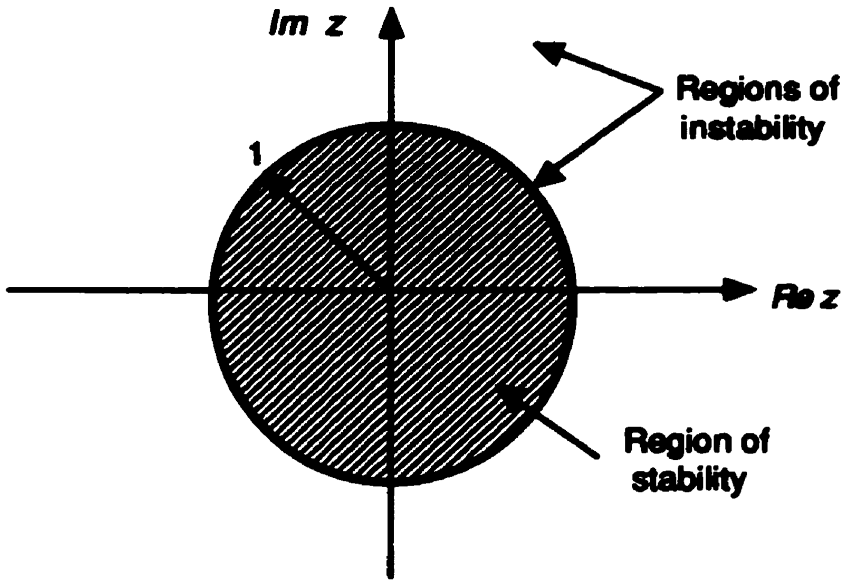

Figure 3.4: Permissible areas for the poles of H(z)

The unit circle of Figure 3.4 represents the condition for stability of a digital filter. The radius of the circle represents the permissible frequencies, ranging from zero to the Nyquist frequency. If a digital filter has poles that reside on, or outside the unit circle, the filter will be unstable resulting in output that may grow out of range and not return to zero when the input signal is removed.

3.2 Digital Low-Pass Filter Design. The Bilinear Transformation Method

The bilinear transformation method is an approximation method used to derive digital filters that, for any excitation, have approximately the same time-domain response as the original analog filter. If the analog transfer function is given by :

$$H(s)=\frac{A_0+A_1s+A_2s^2+\ldots+A_ks^k}{B_0+B_1s+B_2s^2+\ldots+B_ks^k} \tag{3.7}$$

then, the corresponding digital transfer function can be derived using a transformation of the form [10]:

$$s=C\frac{1-z^{-1}}{1+z^{-1}} \tag{3.8}$$

where the constant C is chosen so that the frequency scale of the desired filter is scaled to the proper range for the desired application. In the case of a low-pass filter, is given by:

$$C=\lambda_r cot(\frac{\pi f_c}{f_s}) \tag{3.9}$$

where λ_r is a low-pass reference radian frequency, f_c is the cutoff frequency, and f_s is the sampling rate frequency. Substitution of (3.8) to (3.7) yields the equivalent digital filter transfer function:

$$H(z)=\frac{a_0+a_1z^{-1}+a_2z^{-2}+\ldots+a_kz^{-k}}{1+b_1z^{-1}+b_2z^{-2}+\ldots+b_kz^{-k}} \tag{3.10}$$

The problem then of calculating the digital filter coefficients of (3.10) becomes a matter of expressing them in terms of their analog counterparts of eq. (3.7), using the transformation of eq. (3.8). The results of these calculations are tabulated for filter orders from 1 to 6 [11], though any desired order can be calculated using the above described method. The analog coefficients A_i and B_i can be calculated for various types of responses. Appendix A provides the reader with the complete analog low-pass transfer functions for the Butterworth, Chebyshev, Inverse Chebyshev, Bessel, and Elliptic responses. Based on the desired response and filter order, the reader can derive the analog transfer function using the equations of Appendix A, and then use the formulas of the following tables to calculate the desired digital filter coefficients. For the less "mathematically inclined" reader, Appendix B provides a different method of analog, low-pass filter coefficients derivation, based on data found in the literature.

Table 3.1

Digital Filter Coefficients in Terms of Analog Coefficients	
k=1, 1st order	k=2, 2nd order
$A=B_0+B_1C$ $a_0=(A_0+A_1C)/A$ $a_1=(A_0-A_1C)/A$ $b_1=(B_0-B_1C)/A$	$A=B_0+B_1C+B_2C^2$ $a_0=(A_0+A_1C+A_2C^2)/A$ $a_1=(2A_0-2A_2C^2)/A$ $a_2=(A_0-A_1C+A_2C^2)/A$ $b_1=(2B_0-2B_2C^2)/A$ $b_2=(B_0-B_1C+B_2C^2)/A$

Table 3.2

Digital Filter Coefficients in Terms of Analog Coefficients	
k=3, 3rd order	k=4, 4th order
$A_0=B_0+B_1C+B_2C^2+B_3C^3$	$A=B_0+B_1C+B_2C^2+B_3C^3+B_4C^4$
$a_0=(A_0+A_1C+A_2C^2+A_3C^3)/A$	$a_0=(A_0+A_1C+A_2C^2+A_3C^3+A_4C^4)/A$
$a_1=(3A_0+A_1C-A_2C^2-3A_3C^3)/A$	$a_1=(4A_0+2A_1C-2A_3C^3-4A_4C^4)/A$
$a_2=(3A_0-A_1C-A_2C^2+3A_3C^3)/A$	$a_2=(6A_0-2A_2C^2+6A_4C^4)/A$
$a_3=(A_0-A_1C+A_2C^2-A_3C^3)/A$	$a_3=(4A_0-2A_1C+2A_3C^3-4A_4C^4)/A$
$b_1=(3B_0+B_1C-B_2C^2-3B_3C^3)/A$	$a_4=(A_0-A_1C+A_2C^2-A_3C^3+A_4C^4)/A$
$b_2=(3B_0-B_1C-B_2C^2+3B_3C^3)/A$	$b_1=(4B_0+2B_1C-2B_3C^3-4B_4C^4)/A$
$b_3=(B_0-B_1C+B_2C^2-B_3C^3)/A$	$b_2=(6B_0-2B_2C^2+6B_4C^4)/A$
	$b_3=(4B_0-2B_1C+2B_3C^3-4B_4C^4)/A$
	$b_4=(B_0-B_1C+B_2C^2-B_3C^3+B_4C^4)/A$

Table 3.3

Digital Filter Coefficients in Terms of Analog Coefficients	
k=5, 5th order	k=6, 6th order
$A=B_0+B_1C+B_2C^2+B_3C^3+B_4C^4+B_5C^5$	$A=B_0+B_1C+B_2C^2+B_3C^3+B_4C^4+B_5C^5+B_6C^6$
$a_0=(A_0+A_1C+A_2C^2+A_3C^3+A_4C^4+A_5C^5)/A$	$a_0=(A_0+A_1C+A_2C^2+A_3C^3+A_4C^4+A_5C^5+A_6C^6)/A$
$a_1=(5A_0+3A_1C+A_2C^2-A_3C^3-3A_4C^4-5A_5C^5)/A$	$a_1=(6A_0+4A_1C+2A_2C^2-2A_4C^4-4A_5C^5-6A_6C^6)/A$
$a_2=(10A_0+2A_1C-2A_2C^2-2A_3C^3+2A_4C^4+10A_5C^5)/A$	$a_2=(14A_0+5A_1C-2A_2C^2-3A_3C^3-A_4C^4+5A_5C^5+14A_6C^6)/A$
$a_3=(10A_0-2A_1C-2A_2C^2+2A_3C^3+2A_4C^4-10A_5C^5)/A$	$a_3=(20A_0-4A_2C^2+4A_4C^4-20A_6C^6)/A$
$a_4=(5A_0-3A_1C+A_2C^2+A_3C^3-3A_4C^4+5A_5C^5)/A$	$a_4=(15A_0-5A_1C-A_2C^2+3A_3C^3-A_4C^4-5A_5C^5+15A_6C^6)/A$
$a_5=(A_0-A_1C+A_2C^2-A_3C^3+A_4C^4-A_5C^5)/A$	$a_5=(6A_0-4A_1C+2A_2C^2-2A_4C^4+4A_5C^5-6A_6C^6)/A$
$b_1=(5B_0+3B_1C+B_2C^2-B_3C^3-3B_4C^4-5B_5C^5)/A$	$a_6=(A_0-A_1C+A_2C^2-A_3C^3+A_4C^4-A_5C^5+A_6C^6)/A$
$b_2=(10B_0+2B_1C-2B_2C^2-2B_3C^3+2B_4C^4+10B_5C^5)/A$	$b_1=(6B_0+4B_1C+2B_2C^2-2B_4C^4-4B_5C^5-6B_6C^6)/A$
$b_3=(10B_0-2B_1C-2B_2C^2+2B_3C^3+2B_4C^4-10B_5C^5)/A$	$b_2=(14B_0+5B_1C-2B_2C^2-3B_3C^3-B_4C^4+5B_5C^5+14B_6C^6)/A$
$b_4=(5B_0-3B_1C+B_2C^2+B_3C^3-3B_4C^4+5B_5C^5)/A$	$b_3=(20B_0-4B_2C^2+4B_4C^4-20B_6C^6)/A$
$b_5=(B_0-B_1C+B_2C^2-B_3C^3+B_4C^4-B_5C^5)/A$	$b_4=(15B_0-5B_1C-B_2C^2+3B_3C^3-B_4C^4-5B_5C^5+15B_6C^6)/A$
	$b_5=(6B_0-4B_1C+2B_2C^2-2B_4C^4+4B_5C^5-6B_6C^6)/A$
	$b_6=(B_0-B_1C+B_2C^2-B_3C^3+B_4C^4-B_5C^5+B_6C^6)/A$

3.3 High-Pass Digital Filter Design

Digital high-pass transfer functions can be derived from an analog low-pass transfer function. The transformation is given by [12]:

$$s = C\frac{1+z^{-1}}{1-z^{-1}} \tag{3.11}$$

where the constant *C* is now given by:

$$C = \lambda_s tan(\frac{\pi f_c}{f_s}) \tag{3.12}$$

Therefore, if *H(s)* is the analog low-pass transfer function, the digital high-pass transfer function will be given by:

$$H(z) = H(s = \frac{C(1+z^{-1})}{1-z^{-1}}) \tag{3.13}$$

Another way in which the digital high-pass transfer function may be derived is directly from the digital low-pass transfer function. If $H_{hp}(z)$ and $H_{lp}(z)$ are the high- and low-pass digital transfer functions respectively, their relationship is given by:

$$H_{hp}(z) = H_{lp}(-z) \tag{3.14}$$

Let us now see how equation (3.14) affects the calculation of digital high-pass filter coefficients. The general form of the digital transfer function (3.10) can be rewritten as:

$$H(-z)=\frac{a_0+(-a_1)z^{-1}+a_2z^{-2}+\ldots+(-1)^kz^{-k}}{1+(-b_1)z^{-1}+b_2z^{-2}+\ldots+(-1)^kz^{-k}}=\frac{\sum_{n=0}^{k}(-1)^na_nz^{-n}}{1+\sum_{n=1}^{k}(-1)^nb_nz^{-n}} \tag{3.15}$$

What equation (3.15) means is that the digital high-pass filter coefficients will be exactly the same as the digital low-pass filter coefficients, with the only difference being the change of sign of the odd-numbered coefficients.

3.4 Band-Pass Digital Filter Design

Digital recursive band-pass filters can also be designed directly from a low-pass analog transfer function, using a form of the bilinear transformation. The transformation that has to be used is given by [13]:

$$s=C\left(\frac{1-Dz^{-1}+z^{-2}}{1-z^{-2}}\right) \tag{3.16}$$

where

$$C=\lambda_c\cot\left(\frac{\pi BW}{f_s}\right) \tag{3.17}$$

and

$$D=2\cos\left(\frac{2\pi f_0}{f_s}\right) \tag{3.18}$$

In the above formulas, BW is the bandwidth of the filter, and f_0 its center frequency. Therefore, if $H(s)$ is the analog transfer function of a low-pass filter, the digital transfer function of the corresponding band-pass filter will be given by:

$$H(z)=H(s=C\frac{1-Dz^{-1}+z^{-2}}{1-z^{-2}}) \tag{3.19}$$

From equation (3.16) we note that if the low-pass prototype transfer function is of order k, the resulting order of the digital band-pass transfer function will be $2k$. Let us, for example, derive the 2nd-order band-pass digital transfer function, and the corresponding digital coefficients. Using (3.7) and (3.16), we have:

$$\frac{A_0+A_1s}{B_0+B_1s}\to\frac{A_0+A_1C(\frac{1-Dz^{-1}+z^{-2}}{1-z^{-2}})}{B_0+B_1C(\frac{1-Dz^{-1}+z^{-2}}{1-z^{-2}})} \tag{3.20}$$

which results in:

$$H(z)=\frac{(B_0+B_1C)^{-1}[A_0+A_1C-A_1CDz^{-1}-(A_0-A_1C)z^{-2}]}{1-\frac{B_1CD}{B_0+B_1C}z^{-1}-\frac{B_0-B_1C}{B_0+B_1C}z^{-2}} \tag{3.21}$$

Comparison of (3.21) and equation (3.10) gives the digital 2nd-order band-pass filter coefficients:

$$\begin{aligned}
&a_0=\frac{A_0+A_1C}{B_0+B_1C},\ a_1=-\frac{A_1CD}{B_0+B_1C}\\
&a_2=-\frac{A_0-A_1C}{B_0+B_1C}\\
&b_1=-\frac{B_1CD}{B_0+B_1C},\ b_2=-\frac{B_0-B_1C}{B_0+B_1C}
\end{aligned} \tag{3.22}$$

The same method may be used for any other desired order.

3.5 Band-Rejection Digital Filter Design

Finally, the bilinear transformation method can be used for band-rejection filter design. The transformation now is given by [14]:

$$s=C(\frac{1-z^{-2}}{1-Dz^{-1}+z^{-2}}) \tag{3.23}$$

where

$$C=\lambda_f tan(\frac{\pi BW}{f_s}) \tag{3.24}$$

and D is given by (3.18). Then, the digital transfer function for the band-rejection case will be given by:

$$H(z)=H(s=C\frac{1-z^{-2}}{1-Dz^{-1}+z^{-2}}) \tag{3.25}$$

Using (3.25) and the method described in the previous section, we can get the digital transfer band-rejection functions for any desired order. Again, if k is the order of the prototype transfer function, the order of the resulting digital band-rejection function will be $2k$. As an example, we give the 2nd-order digital transfer band-rejection function and the corresponding digital coefficients. We get:

$$H(z)=\frac{(B_0+B_1C)^{-1}[A_0+A_1C-A_0Dz^{-1}+(A_0-A_1C)z^{-2}]}{1-\frac{B_0D}{B_0+B_1C}z^{-1}+\frac{B_0-B_1C}{B_0+B_1C}z^{-2}} \tag{3.26}$$

Comparing (3.26) and (3.10), we get the 2nd-order band-rejection digital filter coefficients:

$$
\begin{aligned}
a_0 &= \frac{A_0 + A_1 C}{B_0 + B_1 C} \\
a_1 &= -\frac{A_0 D}{B_0 + B_1 C} \\
a_2 &= \frac{A_0 - A_1 C}{B_0 + B_1 C} \qquad (3.27) \\
b_1 &= -\frac{B_0 D}{B_0 + B_1 C} \\
b_2 &= \frac{B_0 - B_1 C}{B_0 + B_1 C}
\end{aligned}
$$

4. Digital Filter Design in Csound and SuperCollider

4.1 A Complete Example Implemented in Csound

Let us now apply the method described earlier, to build a 3rd-order digital Butterworth high-pass filter in Csound. For a 3rd-order digital filter, eq. (3.3) gives the following input-output difference equation:

$$\begin{aligned} y(n) = {} & a_0 x(n) + a_1 x(n-1) + a_2 x(n-2) + a_3 x(n-3) \\ & - b_1 y(n-1) - b_2 y(n-2) - b_3 y(n-3) \end{aligned} \tag{4.1}$$

with corresponding digital transfer function (from 3.5):

$$H(z) = \frac{a_0 + a_1 z^{-1} + a_2 z^{-2} + a_3 z^{-3}}{1 + b_1 z^{-1} + b_2 z^{-2} + b_3 z^{-3}} \tag{4.2}$$

First, we have to derive the analog low-pass filter coefficients for a Butterworth 3rd-order low-pass filter. According to the method presented in Appendix B, the transfer function will consist of one 1st-order stage function term and one 2nd-order stage function term as follows:

$$H_a(s) = \frac{C_1}{s+C_1} \cdot \frac{C_2}{s^2+Bs+C_2} = \frac{1}{s+1} \cdot \frac{1}{s^2+s+1} = \frac{1}{1+2s+2s^2+s^3} \tag{4.3}$$

where we have assumed the normalized case of $\omega_c = 1\,rad/s$, and substituted the values of the coefficients C_i and B, using Table B.1 of the same appendix.

Comparing (3.7) with (4.3) we get the analog coefficients for a 3rd-order Butterworth low-pass filter:

$$\begin{gathered} A_0=1,\ A_1=0,\ A_2=0,\ A_3=0 \\ B_0=1,\ B_1=2,\ B_2=2,\ B_3=1 \end{gathered} \tag{4.4}$$

Now, we can use the formulas of Table 3.2 to derive the digital coefficients for a Butterworth 3rd-order low-pass filter. We get:

$$\begin{gathered} A=1+2C+2C^2+C^3 \\ a_0=\frac{1}{A},\ a_1=\frac{3}{A},\ a_2=\frac{3}{A},\ a_3=\frac{1}{A} \\ b_1=3+2C-2C^2-3C^3 \\ b_2=3-2C-2C^2+3C^3 \\ b_3=1-2C+2C^2-C^3 \end{gathered} \tag{4.5}$$

The next step is to convert the coefficients of (4.5) to high-pass digital filter coefficients. Using (3.13) or (3.14) we get:

$$\begin{gathered} A=1+2C+2C^2+C^3 \\ a_0=\frac{1}{A},\ a_1=-\frac{3}{A},\ a_2=\frac{3}{A},\ a_3=-\frac{1}{A} \\ b_1=-(3+2C-2C^2-3C^3) \\ b_2=3-2C-2C^2+3C^3 \\ b_3=-(1-2C+2C^2-C^3) \end{gathered} \tag{4.6}$$

The value of the constant C will be calculated using eq. (3.12). Now we are ready to build a 3rd-order high-pass Butterworth filter in Csound. The header of the orchestra will be:

```
sr = 44100
kr = 44100
ksmps = 1
nchnls = 1
```

It is important to notice the choice of the control-rate value *kr*. Since the output of IIR filters is calculated on a per sample basis (eq. 4.1), Csound must assure that one sample is calculated per control period, which explains the choice of *sr = kr*. Remember that the number of samples calculated in a control period is given by *ksmps=sr/kr*. By choosing *sr=kr*, we assure that *ksmps=1*. Let us now look at the first part of equation (4.1). We see that the input signal has to be delayed by one, two, and three samples respectively. We can do that by using Csound's signal modifier *delay1* which delays signals by one sample. A second-order delay can be created using *delay1* twice, and similarly, a third-order delay can be created using *delay1* three times. Let us now consider the second part of eq. (4.1). The order of the filter is 3, therefore, three intermediate output variables *aout1*, *aout2*, and *aout3* will be used, which correspond to the values of *y(n-1)*, *y(n-2)*, and *y(n-3)* of eq. (4.1). The current sample output value will be *aout*. During the calculation of the first sample, only the 1st term of (4.1) gives output. Then, during the calculation of the 2nd sample, the output *aout* becomes the value of *aout1*, which is used, along with the current and previous input values (created by using *delay1*), for the calculation of the new value of the output *aout*. It is not until the calculation of the 4th sample that all terms of eq. (4.1) begin contributing to the final output *aout*. Therefore, the procedure is to always have the last three values of outputs, along with the current and the last three values of inputs, giving the value of the current output.

The body of the orchestra follows:

```
instr 1
idur = p3                              ;duration of the test soundfile
icut = p4                              ;user provided cutoff frequency
iscale = p5                            ;scaling factor for the final output

;Calculate the value of the constant C using eq. 3.12
```

```
ivar = sin(3.14159*icut/sr)/cos(3.14159*icut/sr)

;Calculate the filter coefficients of eq. (4.6), using the power function ipow
ivarsqrd ipow ivar,2                    ;the term C^2 of eq 4.6
ivarcbd ipow ivar,3                     ;the term C^3 of eq. 4.6

ia0 = 1/(1+2*ivar+2*ivarsqrd+ivarcbd)
ia1 = -3*ia0
ia2 = 3*ia0
ia3 = =-ia0
ib1 = -(3+2*ivar-2*ivarsqrd-3*ivarcbd)*ia0
ib2 = (3-2*ivar-2*ivarsqrd+3*ivarcbd)*ia0
ib3 = -(1-2*ivar+2*ivarsqrd-ivarcbd)*ia0

;Initialize the intermediate output variables used
aout1 init 0
aout2 init 0
aout3 init 0

;The sound sample that will be filtered is imported in Csound using soundin.
ainput soundin "test_sound"             ;the input signal file

;Create a 3rd order delay
adel1 delay1 ainput
adel2 delay1 adel1
adel3 delay1 adel2

;Calculate the output signal
aout = ia0*ainput+ia1*adel1+ia2*adel2+ia3*adel3-ib1*aout1-ib2*aout2-ib3*aout3

;Apply recursion formula: y(n-1) = y(n), y(n-2) = y(n-1), etc.
aout3 = aout2
aout2 = aout1
aout1 = aout
```

```
;Provide a simple exponential segment envelope for the output signal
aenv expseg .1,.1*idur,1,idur*.8,1,idur*.1,.1

out aout*aenv*iscale                    ;output the signal
endin
```

A sample score file which controls duration, cutoff frequency, and scaling of the final output would be of the form:

```
;instr  start  dur   cutoff scale
i1      10     10    5000   4.3
i1      20     10    10000  8.5
e
```

4.2 SuperCollider: A General Overview

SuperCollider is a real-time sound synthesis programming language, currently running on the Power Macintosh platform. The brainchild of James McCartney, it was first published in 1996 as a combination of *Pyrite* and *Synth-O-Matic*. Pyrite is a MAX [15] object with powerful list processing capabilities, and Synth-O-Matic was a non-real-time synthesis program.

The architecture of the audio engine of the program is based on the concepts of samples, subframes, frames, audio rate, and control rate. The number of samples per second is the audio rate, and audio rate values change each sample period. On the other hand, control rate values change each subframe, which is a multiple of a number of samples. Finally, a frame is a multiple of the subframe size. It determines the latency between computation and performance of events, since it is the size of the audio buffer sent to the Sound Manager.

In order for the user to produce sound, his functions must be registered with the Digital Signal Processing or DSP engine of the program. The registered functions, or DSP tasks, are called every subframe until they are removed from their DSP stage. Four stages are supported, ranging from 0 to 3, and are executed in order. This allows the user to schedule various tasks accordingly.

SuperCollider provides the user with:

• *A higher level language.*

Object-oriented programming with single inheritance is supported. The user can define *classes*, as extensions of functions, and *methods*, that overload any built-in functions. Also, the "archaic" distinction of Music-N type languages between an orchestra and a score has not been implemented. Instead, the user can create and control the performance of his instrument from the same file.

• *A Graphical User Interface.*

The GUI menu of the program allows the user to create and manipulate Sliders, Range Sliders, Buttons, Radio Buttons, Check Boxes, Strings, and Graphics. These can be used during the execution of the program to offer real-time control of the various parameters of the code. Figure 4.1 shows an example of a SuperCollider GUI.

Along with the GUI menu, the user is provided with menus for Table, Audio, and Envelope windows. While the current version does not support editing of the contents of Audio windows, the Envelope and Table windows can be modified either by using the mouse or within the program.

• *MIDI control.*

Using the rich palette of MIDI functions available, the user can control incoming MIDI data, such as notes on and off, pitch bend, program changes, and aftertouch. Also, *MIDI Map Faders*, found in the GUI menu lets the user map screen faders to MIDI controllers.

• *Real time garbage collection.*

Real-time garbage collection simplifies dynamic instantiation of synthesis processes. The user is guaranteed an upper bound on the collection time and does not have to worry about the deletion procedure.

• *Routing of audio.*

Sound input can come from either a user-specified file or the Sound Manager. Sound output can go to an output file, or the Sound Manager, or both.

• *Task scheduling.*

Scheduling of events can be relative to a real-time clock, or a beat clock. Real-time clock scheduling allows the user to add and remove events from a real-time clock scheduler, as well as implement priority queues. On the other hand, tempo-based scheduling is relative to a beat clock. This allows the user to keep separate musical lines synchronised, while the tempo changes.

• *A large library of functions.*

The functions for processing and controlling audio are divided into four groups. *Audio rate* functions start with a capital *A*, and output a signal buffer of samples each control period. The control period is defined as the sampling rate (usually 44.1 KHz) divided by the subframe, which is a multiple of samples that can be selected in the "Set Globals" dialogue of the Synth menu.

Control rate functions start with a capital *K*, and output a single float value each control period. This makes them many times faster than an audio rate function for control purposes.

Polled functions start with a capital *P*, and can be called at intermittent times after they are created, for example, at the beginnings of notes.

The above three groups of functions all come in pairs. The first function of the pair can be thought of as the definition of the process, and the second function of the pair as the generation of the process defined by the first function. The last group of functions can be used directly, without the use of the pair scheme. These include:

◊ *Bitwise Operations* :	functions that operate on the bit level.
◊ *Boolean Operations* :	functions that enable Boolean logic calculations.
◊ *DSP Miscellaneous* :	functions that control the Digital Signal Processing engine of the program.
◊ *Functions and Frames* :	functions which can refer to arguments and variables that were defined in a lexically (i.e. textually) enclosing function, even if that enclosing function has completed execution.
◊ *Graphics Functions* :	functions that make use of the graphics capabilities of SuperCollider.
◊ *General List* :	functions that process lists .
◊ *List Iterators* :	functions that iteratively process lists.
◊ *List Ordering* :	functions that manipulate the ordering of the elements of lists.
◊ *List Sets* :	functions that process sets of lists.

◊ *Math Complex* :	functions that perform operations with complex numbers.
◊ *Math Transcendental* :	functions such as sqrt, log, exp, as well as trigonometric functions.
◊ *Math Infix* :	operators such as *, /, +, -, as well as various math functions.
◊ *Math Unary* :	number processing functions such as abs, floor, sign, etc.
◊ *More Math* :	another yet group of math functions which perform operations such as round, truncate, clip, etc.
◊ *MIDI Functions*:	functions that control the interaction of the program with MIDI.
◊ *Misc Functions* :	includes functions for printing messages, getting the time, etc.
◊ *Numerical List* :	functions that process lists of numbers.
◊ *Pattern Streams* :	pattern stream creation functions allow the user to generate complex nested sequences of data.
◊ *Pattern Threads* :	pattern thread functions are similar to Pattern Streams, except that threads manage the order of execution of functions in time instead of the ordering of data.
◊ *Random Numbers* :	functions used for random numbers generation, as well as generation of numbers according to distributions such as bernoulli, gaussian, etc.
◊ *Scheduling* :	functions that control the scheduling of events.
◊ *Tempo Scheduling* :	functions that allow the user to schedule events relative to a beat clock rather than a real-time clock.
◊ *Text I/O* :	functions that perform text and files related tasks.

◊ *Wave Functions* : **functions that allow the user to define and manipulate waves, as well as import audio files.**

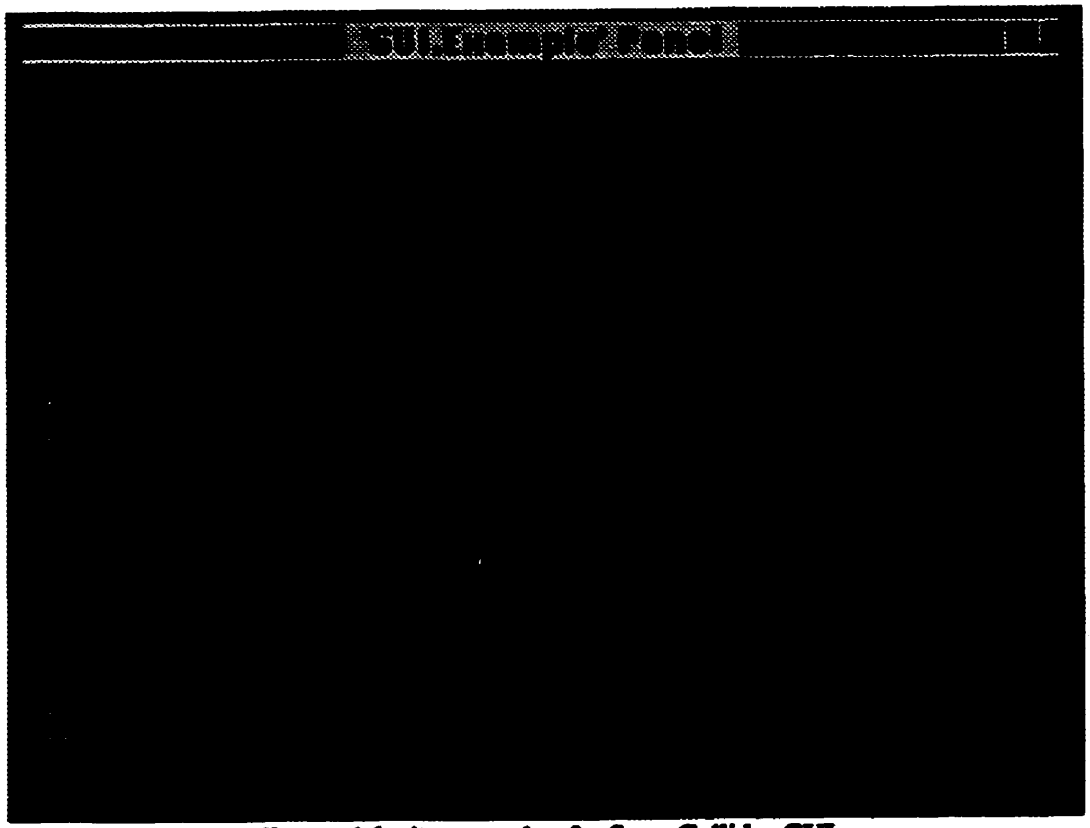

Figure 4.1: An example of a SuperCollider GUI.

4.3 A Complete Example Implemented in SuperCollider

Let us now build a 4th-order 3 dB digital band-pass Chebyshev filter in SuperCollider. For a 4th-order digital filter, eq. (3.3) gives the following input-output difference equation:

$$\begin{aligned} y(n) &= a_0x(n)+a_1x(n-1)+a_2x(n-2)+a_3x(n-3)+a_4x(n-4) \\ &\quad -b_1y(n-1)-b_2y(n-2)-b_3y(n-3)-b_4y(n-4) \end{aligned} \tag{4.7}$$

From the discussion on band-pass digital filter design, we know that a 4th-order digital band-pass transfer function can be derived from the corresponding 2nd-order low-pass analog transfer function. Therefore, we first have to derive the 2nd-order low-pass Chebyshev analog transfer function. Using either equations (A4.1)-(A4.5) of Appendix A, or Table B.3 of Appendix B we get:

$$H_c(s)=\frac{.7079478}{.7079478+.6448996s+s^2} \tag{4.8}$$

Then, comparison of (4.8) with eq. (3.7) gives:

$$\begin{aligned} &A_0=.7079478,\, B_0=.7079478 \\ &B_1=.6448996,\, B_2=1 \end{aligned} \tag{4.9}$$

Now we can derive the 4th-order digital band-pass coefficients by applying the transformation of eq. (3.16) to (4.8), and compare our results with (3.10). By doing so we get:

$$\begin{aligned}
a_0 &= \frac{B_0}{B_0+B_1C+B_2C^2}, \\
a_1 &= 0,\ a_2=-2a_0,\ a_3=0, a_4=a_0 \\
b_1 &= -\frac{B_1CD+2B_2C^2D}{B_0+B_1C+B_2C^2} \\
b_2 &= -\frac{2B_0-B_2C^2D-2B_2C^2}{B_0+B_1C+B_2C^2} \\
b_3 &= \frac{B_1CD-2B_2C^2D}{B_0+B_1C+B_2C^2} \\
b_4 &= \frac{B_0-B_1C+B_2C^2}{B_0+B_1C+B_2C^2}
\end{aligned} \tag{4.10}$$

Now, let us substitute the values of (4.9) to (4.10). We get:

$$\begin{aligned}
a_0 &= \frac{1}{1+.91094C+1.41253C^2} \\
a_1 &= 0,\ a_2=-2a_0,\ a_3=0,\ a_4=a_0 \\
b_1 &= -(.6448996CD+2C^2D)\frac{a_0}{.7079478} \\
b_2 &= -(2*.7079478-C^2D^2-2C^2)\frac{a_0}{.7079478} \\
b_3 &= (.6448996CD-2C^2D)\frac{a_0}{.7079478} \\
b_4 &= (.7079478-.6448996C+C^2)\frac{a_0}{.7079478}
\end{aligned} \tag{4.11}$$

The coefficients *C* and *D* will be calculated by equations (3.17) and (3.18) respectively. The code for the filter-instrument, fully commented, follows:

defaudioout L R;

Defaudioout defines the audio output buffers for the left and right channels. *L* and *R* represent the left and right outputs of the Sound Manager, but output can also be directed to a file.

defaudioin Lin Rin;

Defaudioin declares a buffer for reading input audio. The input audio can come from either a sound file or the Sound Manager.

defaudiobuf the_sound;

Defaudiobuf defines an audio buffer which is needed for instantaneous access to audio.

deftable env;

Deftable is used for defining tables. The defined tables can be selected from the Table menu and can be modified using the mouse.

```
defdelay c1( 0.1);
defdelay c2(0.1);
defdelay c3(0.1);
defdelay c4(0.1);
```

Defdelay is used to declare buffers for delay lines. The number in parenthesis specifies the maximum delay time.

```
init{
 --load the test sound file
 loadAudio(the_sound, "soundfile.aiff");
}
```

The *init* function is called after successful compilation of the code. In our case, it is used to load the sound file to be processed.

```
start {
--Call the filter instrument
instr1;
   }
```

The *start* function is executed first when the execution of the code begins. In our case it just calls the filter instrument *instr1*, which activates the DSP engine of the system.

instr1{

--Define variables used in the code

var s1, s2, s3, s4;

var x1, x2, x3, x4;

var ia0, ia1, ia2, ia3, ia4;

var ib1, ib2, ib3, ib4;

--Define the delay times (1 - 4 sample delays)

s1 = 0.000022675;

s2 = 0.000045351;

s3 = 0.000068027;

s4 = 0.000090702;

Since we are building a 4th-order band-pass filter the delay times will be: *1/sr*, *2/sr*, *3/sr*, and *4/sr*, with *sr=44100 Hz*.

--Initialize signal variables

asig = 0;

asig1 = 0;

asig2 = 0;

asig3 = 0;

asig4 = 0;

x = Abufrd (the_sound, 0.0, 1.0);   *--Read the signal from the buffer*

Abufrd creates an audio buffer reader. In our case there is no offset, and the speed of scanning through the file is normal.

--Create a 4th order delay

x1 = Adelay(c1, s1);

x2 = Adelay(c2, s2);

x3 = Adelay(c3, s3);

x4 = Adelay(c4, s4);

Adelay is used to create simple delay lines. The first parameter in the parenthesis is the buffer that was declared using *defdelay*, and the second parameter specifies the initial delay time in seconds. Note that the values of the parameters s_i have to be calculated and provided by the user, since SuperCollider doesn't support a one-sample delay operator.

```
--Provide an envelope function
amp = Atransient(env, 30.5, -9.dbamp, 0, `dspRemove);
```

Atransient creates an audio rate transient generator function for the amplitude envelope which uses the wave table *env* as the amplitude curve. Note that after the specified duration, the function *dspRemove* is called. *DspRemove* removes the envelope task from the DSP engine, which assures that the engine won't get filled with DSP tasks.

```
{
t=x.value;      --Read from the input audio buffer

--Get the GUI controlled values for center frequency and bandwidth
icenter = getItemValue(1);
ibandwidth = getItemValue(3);
```

The user can change the values of *center frequency* and *bandwidth* during execution, which are then passed to the DSP engine by the function *getItemValue*.

```
--Calculate the filter coefficients
ivar = 1 / tan(3.14159 * ibandwidth / sr);
iscale = 2 * cos(6.28318 * icenter / sr);

ia0 = 1/(1 + (0.91094 * ivar) + (1.41253 * (ivar**2)));
ia1 = 0;
ia2 = neg(2) * ia0;
ia3 = 0;
ia4 = ia0;
ib1 = neg(1) * ((0.6448996 * ivar * iscale) + (2 * (ivar ** 2) * iscale)) * ia0 /
```

```
0.7079478 ;
ib2 = neg(1) * ((2 * 0.7079478) - ((ivar ** 2) * (iscale ** 2)) - (2 * (ivar ** 2))) *
ia0 / 0.7079478;
ib3 = ((0.6448996 * ivar * iscale) - (2 * (ivar ** 2) * iscale)) * ia0 / 0.7079478;
ib4 = (0.7079478 - (0.6448996 * ivar) + (ivar ** 2)) * ia0 / 0.7079478;
```

The filter coefficients are calculated by the DSP engine at a rate defined by the subframe size, which is set by the user in the *Set Globals* of the *Synth Menu*. Note that for digital recursive filters, as it was explained in the Csound instrument earlier, one sample must be calculated per control period. Therefore, the sub-frame size must be equal to 1.

```
--Calculate the output signal according to eq. 4.7
asig = (ia0*t) + (ia1*x1.(t)) + (ia2*x2.(t)) + (ia3*x3.(t)) + (ia4*x4.(t)) -
(ib1*asig1) - (ib2*asig2) - (ib3*asig3) - (ib4*asig4);

--Apply recursion formula
asig4 = asig3;
asig3 = asig2;
asig2 = asig1;
asig1 = asig;

(asig *! amp.value).out(L).out(R);     --output the scaled signal

--Stop the engine when sound file is over
if now. > 30.5 then
  dspKill(1);
end.if
}.dspAdd(1);}
```

One of the great features of SuperCollider is its ability to provide real-time GUI controls for the instrument.

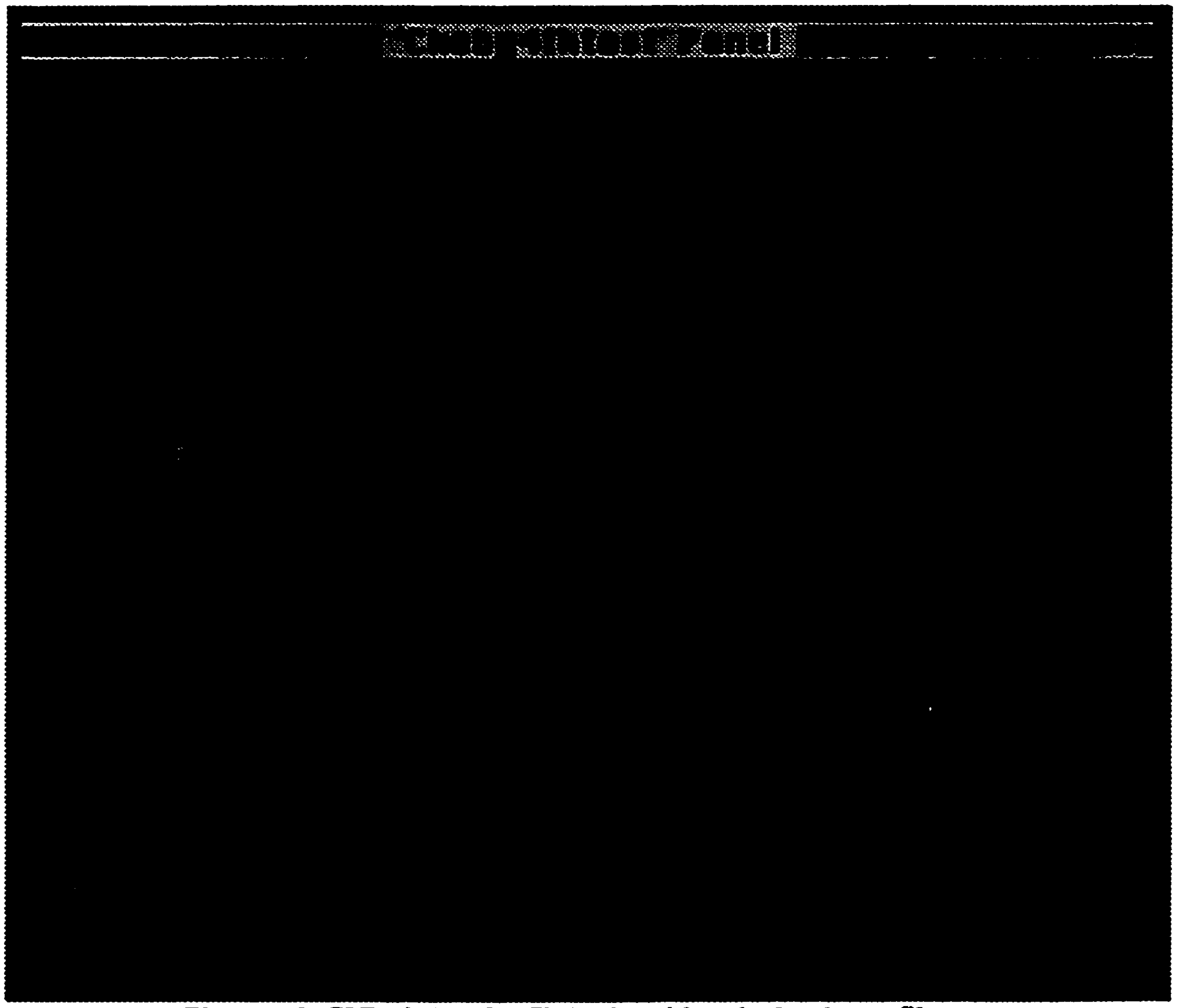
Figure 4.2: GUI window for Chebyshev 4th-order band-pass filter.

Figure 4.2 shows the interface for the center frequency and bandwidth of the filter, which can be controlled by the user during execution time. Figure 4.3 shows the envelope applied to the signal before its output.

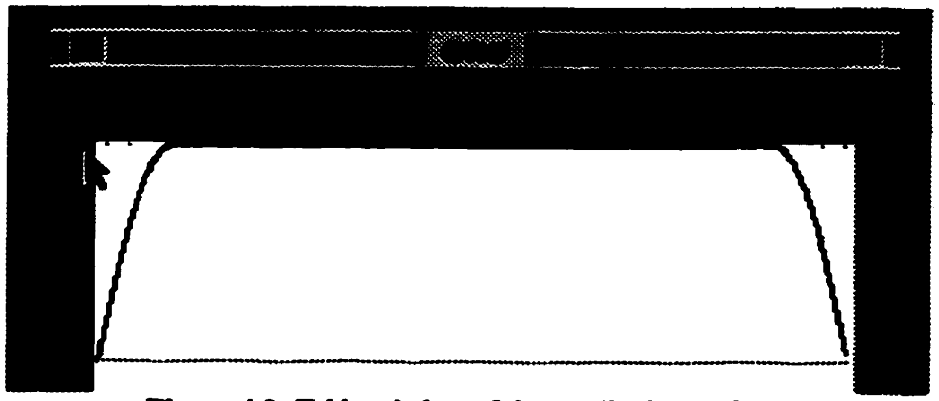

Figure 4.3: Table window of the amplitude envelope.

Figure 4.4 shows the original signal being processed.

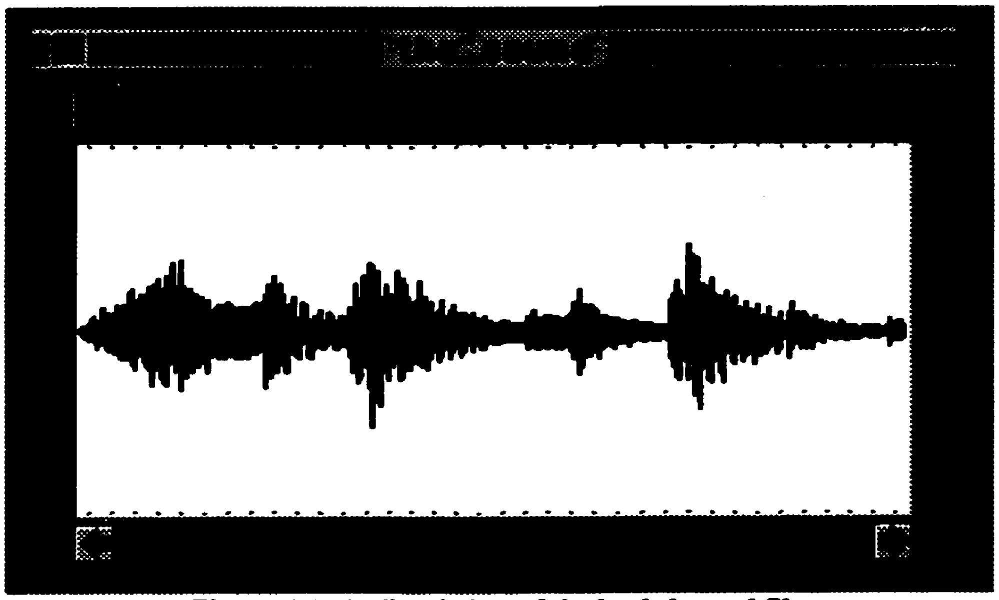
Figure 4.4: Audio window of the loaded sound file.

5. Conclusion

This thesis has studied digital recursive or IIR filters. Five different responses have been examined; Butterworth, Chebyshev, Inverse Chebyshev, Bessel, and Elliptic. The reader is given enough information on how to choose the response matching the needs of his applications. A method was presented for calculating the corresponding analog filter coefficients first, and then, using the bilinear transformation method, convert them to the digital domain. Complete examples implementing filter algorithms were given in two computer music languages, Csound and SuperCollider. Csound is currently more popular since it has been available for a longer time and on various platforms. SuperCollider is currently running only on the Power Macintosh. It offers real-time processing capabilities as well as GUI editing tools. These features, along with a series of improvements that the new version is expected to introduce, make it the author's first choice as a powerful real-time processing sound synthesis tool. As described in the introduction, the author's intention was to provide filter designers, such as sound designers and electroacoustic and computer music composers, with a tutorial on how to build digital recursive filters using tools that are familiar to them. As well, the strategy was based on viewing filters as another instrument in the designer's/composer's pallet, not as a post-production editing tool. This explains the choice of implementing the filters using computer music languages instead of a programming language such as *C*, for example.

Any order of a desired filter can be calculated using the presented method. For higher orders of filters, however, the method involves tedious calculations. Appendix B addresses this problem by realizing higher order systems as a cascade or product of lower order functions. If the analog function can be represented as:

$$G(s)=G_1(s)G_2(s)...G_n(s) \qquad (5.1)$$

then, the resulting discrete transfer function will be [16]:

$$H(z)=H_1'(z)H_2'(z)...H_m'(z) \quad (5.2)$$

where, again, the transformations of (3.8, 3.11, 3.16, 3.23) are used.

The choice of the bilinear transform as the method of conversion from the analog to the digital domain is justified by its relative simplicity and accuracy. Two other methods that can be used are the *Impulse-Invariance* method, and the *Step-Invariance* method. According to the Impulse-Invariance method [17], the impulse response of the digital system must be the same, or proportional to the corresponding impulse response of the prototype analog filter at sampling points. Figure 5.1 illustrates that.

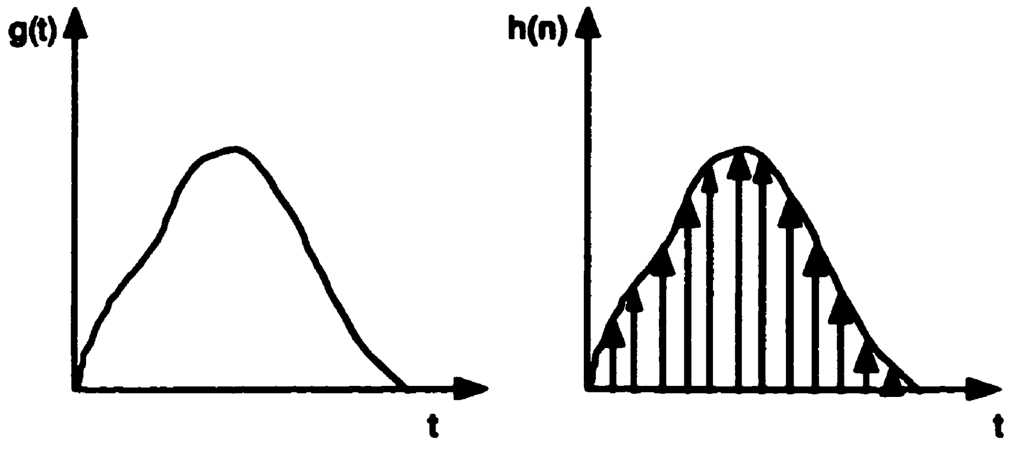

Figure 5.1: The Impulse-Invariance method.

Designing IIR filters using the impulse-invariance method is more difficult than using the bilinear transform method. Determining the impulse response corresponding to the reference analog function can be non-trivial, depending on the complexity of the analog function. Also, aliasing errors can occur, especially if the filter doesn't have sharp cutoff characteristics.

The Step-Invariance method [18] consists of defining unit step sequence response values of the digital filter as being equal to the sampled values of the unit step response of the reference analog filter. Figure 5.2 illustrates the step-invariance method.

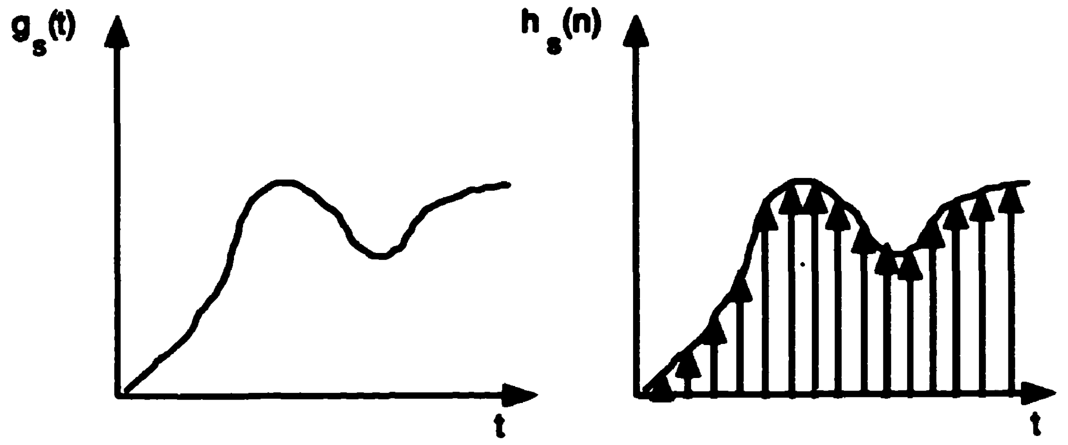

Figure 5.2: The Step-Invariance method.

Similar to the impulse-invariance method, aliasing problems will occur if the amplitude response of the filter is not sharply band-limited.

Finally, a few words about the non-mathematical nature of the thesis. It was the author's intention to provide sound designers with a tutorial, i.e. a how-to guide on digital recursive filters. His main concern was not to present an in-depth mathematical analysis of the subject; there is vast literature on that, a representative sample of which is presented in the bibliography. However, there is enough mathematical information presented in the text. An effort has been made though not to make it crucial for the understanding of the techniques presented.

Appendix A

Complete Analog Low-Pass Transfer Functions

A.1 Bessel Transfer Function

The complete, analog, *n*th-order low-pass transfer function of the Bessel response is given by the following formula:

$$H_{Bn}(S)=\frac{b_0}{\sum_{i=0}^{n} b_i S^i}=\frac{b_0}{S^n B(1/s)} \qquad \text{(A1.1)}$$

where

$$b_i=\frac{(2n-i)!}{2^{n-i}\, i!(n-i)!} \qquad \text{(A1.2)}$$

The Bessel polynomial $B(1/S)$ can be expressed in terms of Bessel functions as follows:

$$B(\frac{1}{j\omega})=\frac{1}{j^n}\sqrt{\frac{\pi\omega}{2}}[(-1)^n J_{-v}(\omega)-jJ_v(\omega)]e^{j\omega} \qquad \text{(A1.3)}$$

where

$$J_v(\omega)=\omega^v\sum_{i=0}^{\infty}\frac{(-1)^i\omega^{2i}}{2^{2i+v}\, i!\,\Gamma(v+i+1)} \qquad \text{(A1.4)}$$

$J_v(\omega)$ are the Bessel functions, and

$$\nu=n+\frac{1}{2} \tag{A1.5}$$

$\Gamma(x)$ is the gamma function given by:

$$\Gamma(x)=\int_0^\infty t^{x-1}e^{-t}dt \tag{A1.6}$$

For the purposes of Bessel filters x is an integer n = 1, 2, 3, ..., which transforms (A1.6) to:

$$\Gamma(n)=(n-1)! \tag{A1.7}$$

A.2 Butterworth Transfer Function

The complete, analog, low-pass nth-order transfer function of the Butterworth response is given by the following formulas:

$$H_{B,n}(S)=\frac{\prod_m (B_{2m})}{\prod_m (S^2+B_{1m}+B_{2m})},\ m=0,1,\ldots,\frac{n}{2}-1,\ if\ n\ even \tag{A2.1}$$

$$H_{B,n}(S)=\frac{R\prod_m (B_{2m})}{(S+R)\prod_m (S^2+B_{1m}+B_{2m})},\ m=0,1,\ldots,\frac{n-1}{2}-1,\ if\ n\ odd \tag{A2.2}$$

In the above formulas we have:

$$\begin{aligned} B_{1m} &= -\ 2R\cos(\theta_m) \\ B_{2m} &= (R\cos(\theta_m))^2+(R\sin(\theta_m))^2 \end{aligned} \tag{A2.3}$$

with

$$\theta_m=\frac{\pi(2m+n+1)}{2n},\ m=0,1,\ldots,\frac{n}{2}-1,\ \text{if } n \text{ even}$$
$$\theta_m=\frac{\pi(2m+n+1)}{2n},\ m=0,1,\ldots,\frac{n-1}{2}-1,\ \text{if } n \text{ odd} \qquad \text{(A2.4)}$$

and

$$R=(10^{0.1a_{pass}}-1)^{-\frac{1}{2n}} \qquad \text{(A2.5)}$$

A.3 Elliptic or Cauer Transfer Function

The complete, low-pass, analog, *n*th-order transfer function of the Elliptic or Cauer response is given by the following formulas:

$$H_{E,n}(S)=\frac{(10^{-0.05a_{pass}})\prod_m(B_{2m})\prod_m(S^2+A_{1m}S+A_{2m})}{\prod_m(A_{2m})\prod_m(S^2+B_{1m}S+B_{2m})},\ m=0,1,\ldots,\frac{n}{2}-1,\ \text{if } n \text{ even} \qquad \text{(A3.1)}$$

$$H_{E,n}(S)=\frac{\sigma_E\prod_m(B_{2m})\prod_m(S^2+A_{1m}S+A_{2m})}{(S+\sigma_E)\prod_m(A_{2m})\prod_m(S^2+B_{1m}S+B_{2m})},\ m=0,1,\ldots,\frac{n-1}{2}-1,\ \text{if } n \text{ odd} \qquad \text{(A3.2)}$$

where

$$
\begin{aligned}
&B_{1m}=-2\sigma_m\\
&B_{2m}=\sigma_m^2+\omega_m^2\\
&A_{1m}=0.0\\
&A_{2m}=\omega_m^2\\
&\sigma_R=\frac{sn(v_0,\sqrt{1-rt^2})cn(v_0,\sqrt{1-rt^2})}{1-sn^2(v_0,\sqrt{1-rt^2})}
\end{aligned}
\tag{A3.3}
$$

with

$$
\begin{aligned}
&\sigma_m=-\frac{cn[f(m),rt]dn[f(m),rt]sn(v_0,\sqrt{1-rt^2})cn(v_0,\sqrt{1-rt^2})}{1-dn^2[f(m),rt]sn^2(v_0,\sqrt{1-rt^2})}\\
&\omega_m=\frac{sn[f(m),rt]dn(v_0,\sqrt{1-rt^2})}{1-dn^2[f(m),rt]sn^2(v_0,\sqrt{1-rt^2})}\\
&\omega_m=\frac{1}{rt\,sn[f(m),rt]}
\end{aligned}
\tag{A3.4}
$$

where

$$
\begin{aligned}
&v_0=\frac{CEI(rt)sc^{-1}(\varepsilon^{-1},kn)}{nCEI(kn)}\\
&f(m)=\frac{CEI(rt)(2m+1)}{n},\ m=0,1,\ldots,\frac{n}{2}-1,\ \text{if } n \text{ even}\\
&f(m)=\frac{CEI(rt)(2m+2)}{n},\ m=0,1,\ldots,\frac{n-1}{2}-1,\ \text{if } n \text{ odd}
\end{aligned}
\tag{A3.5}
$$

and

$$CEI(k)=u(\frac{\pi}{2},k)=\int_{0}^{\frac{\pi}{2}}(1-k^2\sin^2x)^{-\frac{1}{2}}dx$$
$$rt=\frac{\omega_{pass}}{\omega_{stop}}$$
$$kn=(\frac{10^{-.1a_{pass}}-1}{10^{-.1a_{stop}}-1})^{\frac{1}{2}} \qquad \text{(A3.6)}$$
$$sn(u,k)=\sin(\phi)$$
$$cn(u,k)=\cos(\phi)$$
$$sc(u,k)=\tan(\phi)$$
$$dn(u,k)=\frac{d\phi}{du}$$

with

$$u(\phi,k)=\int_{0}^{\phi}(1-k^2\sin^2x)^{-\frac{1}{2}}dx \qquad \text{(A3.7)}$$

A.4 Chebyshev Transfer Function

The complete, analog, *n*th-order low-pass transfer function of the Chebyshev response is given by the following formulas:

$$H_{C,n}(S)=\frac{(10^{[illegible]})\prod_{m}(B_{2m})}{\prod_{m}(S^2+B_{1m}+B_{2m})},\ m=0,1,\ldots,\frac{n}{2}-1,\ \text{if } n \text{ even} \qquad \text{(A4.1)}$$

$$H_{C,n}(S)=\frac{\sinh(D)\prod_{m}(B_{2m})}{(S+\sinh(D))\prod_{m}(S^2+B_{1m}+B_{2m})},\ m=0,1,\ldots,\frac{n-1}{2},\ \text{if } n \text{ odd} \tag{A4.2}$$

In the above formulas we have:

$$\begin{aligned} B_{1m}&=2\sinh(D)\sin(\phi_m)\\ B_{2m}&=(-\sinh(D)\sin(\phi_m))^2+(\cosh(D)\cos(\phi_m))^2 \end{aligned} \tag{A4.3}$$

with

$$\begin{aligned} \phi_m&=\frac{\pi(2m+1)}{2n},\ m=0,1\ldots,\frac{n}{2}-1,\ \text{if } n \text{ even}\\ \phi_m&=\frac{\pi(2m+1)}{2n},\ m=0,1\ldots,\frac{n-1}{2}-1,\ \text{if } n \text{ odd} \end{aligned} \tag{A4.4}$$

and

$$D=\frac{\sinh^{-1}[(10^{-.1a_{pass}}-1)^{-\frac{1}{2}}]}{n} \tag{A4.5}$$

A.5 Inverse Chebyshev Transfer Function

The complete, analog, *n*th-order low-pass transfer function of the Inverse Chebyshev response is given by the following formulas:

$$H_{I,n}(S)=\frac{\prod_{m}(B_{2m})\prod_{m}(S^2+A_{1m}S+A_{2m})}{\prod_{m}(A_{2m})\prod_{m}(S^2+B_{1m}S+B_{2m})},\ m=0,1,\ldots,\frac{n}{2}-1,\ \text{if } n \text{ even} \tag{A5.1}$$

$$H_{I,n}(S)=\frac{[\sinh(D_I)]^{-1}\prod_m(B_{2m})\prod_m(S^2+A_{1m}S+A_{2m})}{(S+[\sinh(D_I)]^{-1})\prod_m(A_{2m})\prod_m(S^2+B_{1m}S+B_{2m})},\ m=0,1,\ldots,\frac{n-1}{2}-1,\ \text{if } n \text{ odd} \qquad \text{(A5.2)}$$

In the above formulas we have:

$$\begin{aligned} B_{1m}&=-2\sigma_m\\ B_{2m}&=\sigma_m^2+\omega_m^2\\ A_{1m}&=0.0\\ A_{2m}&=\omega_{sm}^2 \end{aligned} \qquad \text{(A5.3)}$$

with

$$\begin{aligned} \sigma_m&=\frac{\sigma_m'}{\sigma_m'^2+\omega_m'^2}\\ \omega_m&=\frac{-\omega_m'}{\sigma_m'^2+\omega_m'^2} \end{aligned} \qquad \text{(A5.4)}$$

where

$$\begin{aligned} \sigma_m'&=-\sinh(D_I)\sin(\phi_m)\\ \omega_m'&=\cosh(D_I)\cos(\phi_m)\\ \omega_{sm}&=\sec(\phi_m) \end{aligned} \qquad \text{(A5.5)}$$

and

$$\begin{aligned} \phi_m&=\frac{\pi(2m+1)}{2n},\ m=0,1\ldots,\frac{n}{2}-1,\ \text{if } n \text{ even}\\ \phi_m&=\frac{\pi(2m+1)}{2n},\ m=0,1\ldots,\frac{n-1}{2}-1,\ \text{if } n \text{ odd} \end{aligned} \qquad \text{(A5.6)}$$

and

$$D_i = \frac{\sinh^{-1}[(10^{.1a_{min}} - 1)^{-\frac{1}{2}}]}{n} \qquad \text{(A5.7)}$$

Appendix B

Analog Bessel, Butterworth, Chebyshev, Inverse Chebyshev, and Elliptic Filter Data

While Appendix A offers a direct way to calculate analog filter coefficients for any chosen response of any order, another shorter method will be presented here based on filter data found in the literature [19]. The low-pass transfer functions of the responses we examine can be factored in terms of *2nd-order stage functions*, if the order of the filter is even. If the order of the filter is odd, a single *1st-order stage function* must be included. In the case of Butterworth, Chebyshev, and Bessel filters, the 2nd-order stage function is of the form:

$$G(s)=\frac{C\omega_c^2}{s^2+B\omega_c s+C\omega_c^2} \tag{B.1}$$

In the case of Inverse Chebyshev and Elliptic filters, the 2nd-order stage function is of the form:

$$G(s)=\frac{\frac{C}{A}(s^2+A\omega_c^2)}{s^2+B\omega_c s+C\omega_c^2} \tag{B.2}$$

The 1st-order stage function is the same for all five responses:

$$G(s)=\frac{C\omega_c}{s+C\omega_c} \tag{B.3}$$

If, for example, the order of the filter is 3, then the corresponding transfer function would be the product of one term of the form (B.1) or (B.2), and one term of the form (B.3). If the order of the filter is 4, then two terms of the form (B.1) or (B.2) will be used. We now give coefficients *A, B, C,* for orders 2 to 5 for all five responses:

Table B.1

Butterworth Low-Pass Filter Data		
Order n	B	C
2	1.414214	1.000000
3	1.000000 -	1.000000 1.000000
4	.765367 1.847759	1.000000 1.000000
5	.618034 1.618034 -	1.000000 1.000000 1.000000

Table B.2

Bessel Low-Pass Filter Data		
Order n	B	C
2	3.000000	3.000000
3	- 3.677815	2.322185 6.459433
4	5.792421 4.207579	9.140131 11.487800
5	- 6.703913 4.649349	3.646739 14.272481 18.156315

Table B.3

Chebyshev 3 dB Pass-Band Ripple Width Low-Pass Filter Data		
Order n	B	C
2	.644900	.707948
3	.298620 -	.839174 .298620
4	.170341 .411239	.903087 .195980
5	.109720 .287250 -	.936025 .377009 .177530

Table B.4

Inverse Chebyshev 30 dB Minimum Stop-Band Loss Low-Pass Filter Data			
Order n	A	B	C
2	32.606961	1.413164	1.031123
3	5.976366	.933370	1.058740 1.134320
4	2.951050 17.199978	.630988 2.169970	1.061509 1.512100
5	2.056891 5.385010	.443052 1.697486	1.053952 1.542401 1.470208

Table B.5

Elliptic 3 dB Pass-Band Ripple Width			
Order n	A	B	C
2	16.341050	.802210	.825047
3	2.638395 -	.234092 -	.888381 .352928
4	1.416828 5.211857	.480516 .086865	.344290 .958043
5	1.775450 1.133895 -	.233483 .031739 -	.677491 .984644 .301724

Bibliography

References

[1] Kyma v4.5 (Symbolic Sound Corporation, 1996)

[2] Matlab v5 (Mathworks, 1997).

[3] Kennedy, Paul B. Filter Designer: *An Intuitive Digital Filter Design Environment*. M.A. Thesis, Faculty of Music, McGill University, 1996.

[4] Deck II v2.6 (Macromedia, 1996).

[5] Ferguson, Sean. *A Guide to Cmix*. Faculty of Music, McGill University, 1994.

[6] Vercoe, Barry. *Csound. A Manual for the Audio Processing System and Supporting Programs with Tutorials*. Media Lab, M.I.T., 1993.

[7] McCartney, James. *SuperCollider. A Real Time Sound Synthesis Programming Language*. Austin, Texas, 1996.

[8], [10], [11], [12], [13], [14]
Stanley,William D., and Gary R. Dougherty, and Ray R. Dougherty. *Digital Signal Processing*. 2nd ed. Reston, Virginia: Reston Publishing Company, Inc., A Prentice-Hall Company, 1984.

[9] Moore, Richard F. *Elements of Computer Music*. Englewood Cliffs, N.J.: Prentice- Hall, A Division of Simon & Schuster, 1990.

[15] Max v3.5 (Opcode Systems/IRCAM, 1996).

[16], [17], [18]
Antoniou, Andreas. *Digital Filters. Analysis, Design, and Applications*. 2nd ed. New York: McGraw-Hill, Inc., 1993.

[19] Johnson, D. E., and J. R. Johnson, and H.P. Moore. *A Handbook of Active Filters*. Englewood Cliffs, N.J.: Prentice-Hall, Inc., 1980.

Further Reading

Bateman, Andrew, & Warren Yates. *Digital Signal Processing Design*. London: Pitman Publishing, 1988.

Beauchamp, K. G. *Signal Processing. Using Analog and Digital Techniques*. London: William Clowes & Sons Ltd., 1973.

Candy, James V. *Signal Processing. The Modern Approach*. New York: McGraw-Hill Book Company, 1988.

Dodge, Charles & T.A. Jerse. *Computer Music. Synthesis, Composition, and Performance*. New York: Schirmer Books, A Division of Macmillan, Inc., 1985.

Jackson, Leland B. *Digital Filters and Signal Processing. Third Edition, with MATLAB® Exercises*. Kluwer Academic Publishers, 1996.

Johnson, David E. *Filter Theory*. Englewood Cliffs, New Jersey: Prentice-Hall, Inc., 1976.

Karl, John H. *An Introduction to Digital Signal Processing*. San Diego, California: Academic Press, Inc., 1989.

Kuc, Roman. *Introduction to Digital Signal Processing*. New York: McGraw-Hill, Inc., 1988.

Lam, Harry Y-F. *Analog and Digital Filters Design and Realization*. Englewood Cliffs, N.J.: Prentice-Hall, Inc., 1979.

Ludeman, Lonnie C. *Fundamentals of Digital Signal Processing*. New York: Harper & Row Publishers, 1986.

Mitra, Sanjit K., and James F. Kaiser, edited by. *Handbook for Digital Signal Processing*. John Wiley & Sons, 1993.

McGillem, Clare D., and George R. Cooper. *Continuous and Discrete Signal and System Analysis*. 2nd ed. New York: Holt, Rinehart, and Winston, 1984.

Oppenheim, Alan V., and Ronald W. Schafer. *Digital Signal Processing*. Englewood Cliffs, N.J.: Prentice-Hall, Inc., 1975.

Papoulis, Athanasios. *Signal Analysis*. New York: McGraw-Hill Book Company, 1977.

Peled, Abraham, and Bede Liu. *Digital Signal Processing. Theory, Design, and Implementation*. New York: John Wiley & Sons, 1976.

Pope, Stephen T. *Sound and Music Processing in SuperCollider*. Center for Research in Electronic Art Technology, University of California, CA, 1997.

Poularikas, Alexander D., and Samuel Seely. *Signals and Systems*. Boston: PWS Publishers, 1985.

Proakis, John G., and Dimitris G. Manolakis. *Introduction to Digital Signal Processing*. New York: Macmillan Publishing Company, 1988.

Rabiner, Laurence R. *Theory and Applications of Digital Signal Processing*. Englewood Cliffs, N.J.: Prentice-Hall Inc., 1975.

Rabiner, Lawrence R. and Charles M. Rader, edited by. *Digital Signal Processing*.

New York: IEEE Press, 1972.

Roberts, Richard A., and Clifford T. Mullis. *Digital Signal Processing*. Reading, Massachusetts: Addison-Wesley Publishing Company, 1987.

Rorabaugh, Britton C. *Digital Filter Designer's Handbook Featuring C Routines*. New York: McGraw-Hill, Inc., 1993.

Strawn, John, edited by. *Digital Audio Signal Processing. An Anthology*. Los Altos, California: William Kaufman, Inc., 1985.

Strum, Robert D., and Donald E. Kirk. *First Principles of Discrete Systems and Digital Signal Processing*. Reading, Massachusetts: Addison-Wesley Publishing Company, Inc., 1988.

Terrel, Trevor J. *Introduction to Digital Filters*. London: The Macmillan Press Ltd., 1980.

Thede, Les. *Analog and Digital Filter Design Using C*. Upper Sadle River, N.J.: Prentice-Hall PTR, 1996.

Weinberg, Louis. *Network Analysis and Synthesis*. New York: McGraw-Hill Book Company, Inc., 1962.

Woram, John M. *Sound Recording Handbook*. Indianapolis, Indiana: Howard W. Sams & Company, 1989.