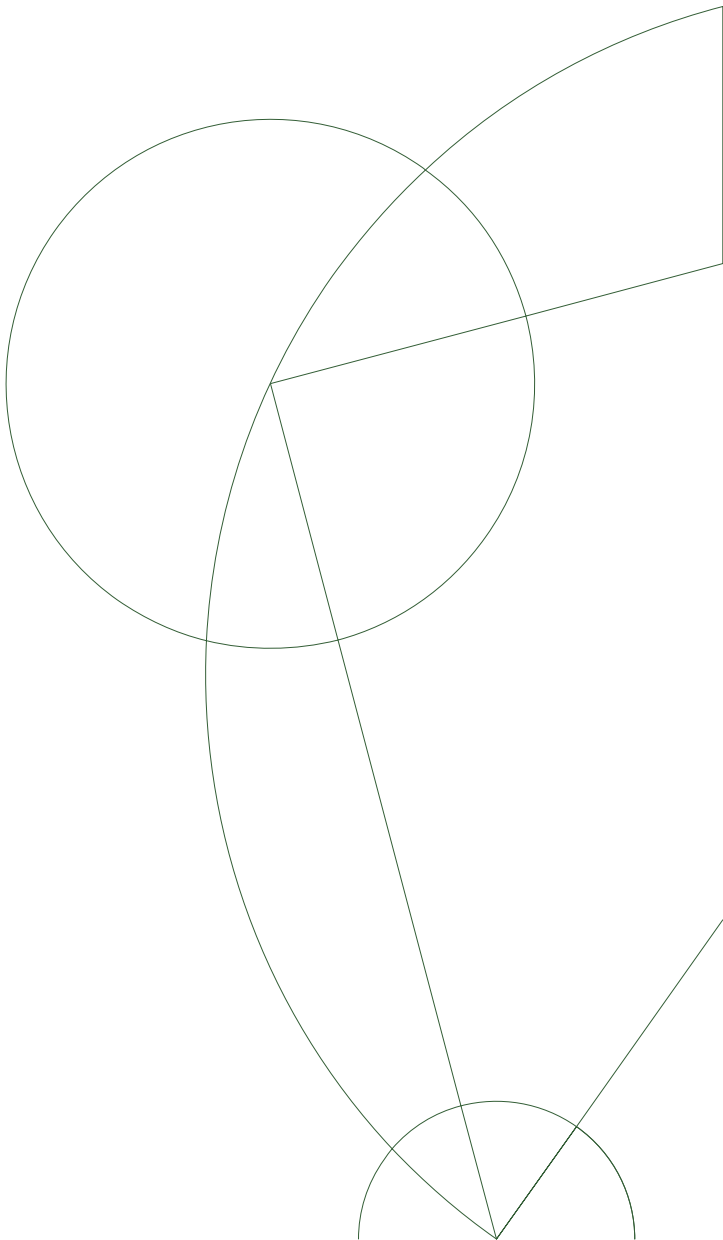
FACULTY OF SCIENCE
UNIVERSITY OF COPENHAGEN

B.Sc. Thesis

Casper Balslev Hansen

Real-time Fourier Analysis & Convolution



Supervisor: Jon Sporring

Januar 22, 2018

Contents

Abstract

In this project I will present the theory of Fourier analysis in relation to audio signals. In particular, the project will investigate the application of filtering under the constraints of real-time signals.

With a practical example in mind (a guitar tuner), I will compare the most common and competitive methods of filtering, and choose the most efficient for this application. Furthermore, the design of such filters is covered in some detail, with emphasis on the chosen method and filter type.

I will conclude theoretical content by analysing the resulting filters, showing that they behave as expected, as well as provide some insight on how to decide if a filter is suitable for a particular application.

Once the filters have been derived and shown to produce the desired results, I will explain the algorithm of the tuner and how it was implemented, as well as the filtering.

Lastly, I will provide some internal tests, showing that the tuner works as expected using a synthetised ideal signal. I will then test the system with real signals, external to the system, showing the filters' hypothetical positive effect on the system.

Resumé

I dette projekt vil jeg præsentere teorien bag Fourier analyse i relation til lydsignaler. Særligt vil projektet undersøge anvendelsen af filtrering under de begrænsninger der forekommer ved signaler i sand tid.

Med et praktisk eksempel i sinde (en guitartuner), vil jeg sammenligne de mest almindelige og konkurrencedygtige metoder af filtrering, og vælge den mest effektive til denne anvendelse. Ydermere er designprocessen af disse filtre forklaret tilnærmelsesvis detajleret, hvor der lægges vægt på den valgte metode og filter type.

Afslutningsvis for den teoretiske del, vil jeg udføre en analyse af de resulterende filtre, der efterviser at de virker som forventet, samt give nogen indsigt om, hvordan man kan afgøre om et filter er egnet til en given anvendelse.

Efter filtrene er blevet udledt og vist til at producere de ønskede resultater, vil jeg forklare tuner algoritmen og hvordan den blev implementeret, såvel som filtrene.

Til sidst vil jeg give nogle interne test, der viser at tuneren virker som forventet ved brug af et syntetisk ideelt signal. Derpå vil jeg teste systemet med ægte signaler, eksterne for systemet, der efterviser filtrenes hypotetiske positive effekt på systemet.

Introduction

Although the theory and techniques discussed may be applicable for any signal, albeit adapted thereto, for our purposes we will constrain ourselves to audio signals. We begin by examining what a signal is in this context.

1.1 Analog Signals

An analog audio signal is a variable electrical voltage obtained with a transducer[1]. A transducer produces a continuous function mapping its voltage $x(t)$ at time t, and has a voltage ceiling k, at which point the signal gets distorted.

Definition 1.1.1 (Audio signal) *Let k be a voltage ceiling of a transducer. Then the continuous function $x(t) : \mathbb{R} \to [-k, k]$ of time t describes an audio signal of a transducer with a ceiling of $\pm k$.*

For the time being, we will be looking at audio signals of finite length.

1.1.1 Analyzing signals

Jean-Baptiste Joseph Fourier (1768–1830) devised harmonic analysis as part of his study of thermodynamics. The theory states that *any periodic function can be described as an infinite sum of sinusoids*.

Definition 1.1.2 (Periodic function) *Let f be a function. If $f(x) = f(x + T), \forall x$, then f is periodic with period T.*

Theorem 1.1.1 (Fourier series) *Let f be a periodic continuous function of time t, with period T, then the infinite sum*

$$x(t) = \sum_{k=-\infty}^{\infty} a_k e^{ik\omega_0 t}, \text{ where } \omega_0 = \frac{2\pi}{T} \tag{1.1}$$

describes f completely.

The Fourier series can be intuitively thought of as a *synthesis* of a signal $x(t)$ from a set of complex exponential coefficients a_k, which modulate the sinusoids $e^{ik\omega_0 t}$.

Given that $x(t) \in \mathbb{R}$ for the signals we intend to process, we have that $\overline{x}(t) = x(t)$. Taking the complex conjugate and reversing the sum by a change of sign, we see that

$$x(t) = \sum_{k=-\infty}^{\infty} a_k e^{ik\omega_0 t} \implies \overline{x(t)} = \sum_{k=-\infty}^{\infty} \overline{a_k} e^{-ik\omega_0 t} = \sum_{k=-\infty}^{\infty} \overline{a_{-k}} e^{ik\omega_0 t} \tag{1.2}$$

[1] A device which converts one form of energy into another.

Therefore, if $x(t) \in \mathbb{R}$ then $a_k = \overline{a_{-k}}$. This observation allows us to save computational time, because given either one, we can predict the other, which is taken advantage of in the Fast Fourier Transform algorithm. Each $a_{\pm k}$ defines the amplitude and phase of the kth sinusoid $e^{ik\omega_0 t}$, which follows from the examining the exponential form, as follows

$$a_k e^{ik\omega_0 t} = |a_k| e^{i\theta} e^{ik\omega_0 t} = |a_k| e^{i(k\omega_0 t + \theta)} \tag{1.3}$$

Suppose we have such a periodic signal $x(t)$. We can then use the Fourier series to analyze the signal by extracting these complex coefficients. Let n be an integer, and multiply both sides of the Fourier series by $e^{-in\omega_0 t}$ and integrate both sides from 0 to T

$$x(t)e^{-in\omega_0 t} = \sum_{k=-\infty}^{\infty} a_k e^{i(k-n)\omega_0 t} \tag{1.4}$$

$$\int_0^T f(t)e^{-in\omega_0 t}dt = \int_0^T \sum_{k=-\infty}^{\infty} a_k e^{i(k-n)\omega_0 t}dt \tag{1.5}$$

$$= \sum_{k=-\infty}^{\infty} a_k \int_0^T e^{i(k-n)\omega_0 t}dt \tag{1.6}$$

$$= Ta_n, \quad \because e^{i(k-n)\omega_0 t} = \begin{cases} 1 & k = n \\ 0 & k \neq n \end{cases} \tag{1.7}$$

$$\frac{1}{T}\int_0^T x(t)e^{-in\omega_0 t}dt = a_n \tag{1.8}$$

These complex exponential coefficients are called *Fourier coefficients*.

Theorem 1.1.2 (Fourier series expansion) *Let x be a periodic continuous function (see definition 1.1.2, p. 3) of time t, with period T, then the integral*

$$a_k = \frac{1}{T}\int_0^T x(t)e^{-ik\omega_0 t}dt \tag{1.9}$$

defines the complex Fourier coefficients a_k of f.

For electrical signals, a_0 is often referred to as the *DC-offset*, or *DC term*. If we plug in $k = 0$ we see that $e^{-i0\omega_0 t} = 1$. Reducing the entire expression to a constant corresponding to the average value of the entire signal, which can be thought of as an offset on the y-axis.

$$a_0 = \frac{1}{T}\int_0^T f(t)e^{-i0\omega_0 t}dt = \frac{1}{T}\int_0^T f(t)dt \tag{1.10}$$

Now, let's consider a continuous function $x(t)$ of length T. The function isn't periodic, however. For convenience, we assume the that $x(t) = 0, \forall t \in [-\infty, 0) \cup (T, \infty]$.

Let $\tilde{x}(t)$ be a function which copies $x(t)$ and repeats every T units, such that $\tilde{x}(t + kT) = x(t)$, where $t \in [0, T]$ and $k \in \mathbb{Z}$. Then $\tilde{x}(t)$ is a continuous periodic function which has a Fourier series. By examining the Fourier coefficients, we see that substituting $\tilde{x}(t)$ with $x(t)$ doesn't affect the integral, as they are equal in the interval $[0, T]$. Subsequently, extending the integral from $-\infty$ to ∞ doesn't change the result, as $x(t)$ is zero outside of $[0, T]$.

$$a_k = \frac{1}{T}\int_0^T \tilde{x}(t)e^{-ik\omega_0 t}dt = \frac{1}{T}\int_0^T x(t)e^{-ik\omega_0 t}dt = \frac{1}{T}\int_{-\infty}^{\infty} x(t)e^{-ik\omega_0 t}dt \tag{1.11}$$

With the above in mind, let's define $X(\omega)$

$$X(\omega) = \int_{-\infty}^{\infty} x(t)e^{-ik\omega_0 t}dt \quad \text{such that} \quad a_k = \frac{1}{T}X(k\omega_0) \tag{1.12}$$

Then, we can reconstruct the signal $\tilde{x}(t)$ by

$$\tilde{x}(t) = \sum_{k=-\infty}^{\infty} \frac{1}{T} X(k\omega_0) e^{ik\omega_0 t}, \text{ where } \omega_0 = \frac{2\pi}{T} \tag{1.13}$$

$$= \frac{1}{2\pi} \sum_{k=-\infty}^{\infty} \left[X(k\omega_0) e^{ik\omega_0 t} \right] \omega_0 \tag{1.14}$$

Intuitively, we can think of this as a sum of rectangles under the curve of the function $X(\omega)e^{i\omega t}$, each of which is of width ω_0. If we let $\omega_0 \to 0$, then $T \to \infty$, and the sum then becomes an integral of ω, and as the copies 'move outward toward infinity', we see that $\tilde{x}(t) \to x(t)$. Thus,

$$x(t) = \frac{1}{2\pi} \int_{-\infty}^{\infty} X(\omega) e^{i\omega t} d\omega \tag{1.15}$$

Which is called the inverse Fourier transform.

Theorem 1.1.3 (Inverse Fourier transform) *Let $X(\omega)$ be a periodic continuous function of frequency ω, then*

$$x(t) = \frac{1}{2\pi} \int_{-\infty}^{\infty} X(\omega) e^{i\omega t} d\omega \tag{1.16}$$

The inverse of this was the defined as $X(\omega)$, which is called the Fourier transform.

Theorem 1.1.4 (Fourier transform) *Let $x(t)$ be a non-periodic continuous function of time t, then*

$$X(\omega) = \int_{-\infty}^{\infty} x(t) e^{-i\omega t} dt \tag{1.17}$$

In summary, if we have a periodic function, we can take the Fourier series, or the Fourier transform, because as shown the Fourier series is merely a special case of the Fourier series. If however, we have a non-periodic function, we can only take the Fourier transform. The Fourier transform itself is a special case of the Laplace transform.

1.2 Digital Signals

In order to easily distinguish between continuous and discrete signals, we will employ a notational convention using parenthesis $x(t)$ and brackets $x[n]$, respectively, and also substitute the continuous variable time t with a discrete integer number variable n.

1.2.1 Sampling

To continuous signal $x(t)$ into a discrete one we measure and sample the signal at, or over, evenly-spaced intervals T, called the *sampling period*. By sampling at intervals of T we get a discrete signal $x[n]$ which is sampled at $\frac{1}{T}$ Hz, or samples per second, called the *sampling frequency*. For mathematical convenience, we define $\omega_s = \frac{2\pi}{T}$ as the sampling frequency, which is measured in radians, as opposed to hertz.

Such a system is called a *sampler*, and the process is referred to as *sampling*, the most simple of which is given by

$$x[n] = x(nT), \text{ where } n \text{ is an integer and a sampling period of } T. \tag{1.18}$$

Theorem 1.2.1 (The sampling theorem) *Given a band-limited signal $x[n]$ with a maximum frequency ω_B, sampled at frequency ω_s. If $\omega_s > 2\omega_B$, then $x(t)$ can be perfectly reconstructed.*

In audio, a common sampling frequency ω_s is at 44.100 Hz. At this sample-rate the signal is bandlimited by ω_B, which is called the *Nyquist frequency*. In this case $\omega_B = 22.050$ Hz, which is just above the frequency range of human hearing.

Definition 1.2.1 (Nyquist frequency) *Let ω_s be a sample-rate of some signal $x[t]$. The maximum frequency we can represent in a discrete-time system is then $\omega_B = \frac{\omega_s}{2}$.*

This property of discrete-time systems becomes evident, when we look at the discrete-time Fourier transform (see section 1.2.2, p. 6). In particular, equation (1.21) makes this clear. The sampling theorem then intuitively tells us that we must sample above twice the desired maximum frequency of the signal in order to produce such a frequency.

1.2.2 Discrete-time analysis

In continuous-time, we had the forward and inverse Fourier transformation. In discrete-time, we can adapt the continuous-time equations with a few minor adjustments.

In the case of the Fourier transform, we simply substitute the integral over the continuous function $f(x)$ with a sum over $f[x]$, giving us the discrete-time fourier transform.

Theorem 1.2.2 (Discrete-time Fourier transform, DTFT) *Let $f[n]$ be a discrete non-periodic function. Then*

$$F(\omega) = \sum_{k=-\infty}^{\infty} f[n]e^{-i\omega n} \tag{1.19}$$

is the discrete-time equivalent of the Fourier transform.

If we look at $F(\omega + 2\pi)$, we see that the discrete-time fourier transform is periodic, with period 2π.

$$F(\omega + 2\pi) = \sum_{k=-\infty}^{\infty} f[n]e^{-i(\omega+2\pi)n} = \sum_{k=-\infty}^{\infty} f[n]e^{-i\omega n}e^{-i2\pi n} = \sum_{k=-\infty}^{\infty} f[n]e^{-i\omega n} = F(\omega) \tag{1.20}$$

We also see the symmetric property $\overline{F(\omega)} = F(-\omega)$, much like the coefficients of the Fourier series.

Given that ω is a frequency, this periodicity and symmetry tells us that such a signal is band-limited by π. We can verify this by producing the fastest oscillation possible

$$x[n] = e^{-i\pi n} = \cos \pi n + i \sin \pi n = (-1)^n \tag{1.21}$$

Intuitively, we can picture this on the unit-circle in the complex plane as a signal which oscillates between the extrema -1 and 1. If we plug in any $\omega = (\pi, 2\pi)$, we can use this intuition to visualize that the wave appears to 'move' in the opposite direction.

As pointed out, $F(\omega)$ is 2π periodic and symmetric. Therefore, we restrict the integral of the inverse discrete-time Fourier transform to this interval. Otherwise the integral would not converge.

Theorem 1.2.3 (Inverse discrete-time Fourier transform, IDTFT) *Let $F(\omega)$ be a function of frequency ω. Then*

$$f[n] = \frac{1}{2\pi} \int_{-\pi}^{\pi} F(\omega)e^{i\omega n} d\omega \tag{1.22}$$

defines the inverse discrete-time Fourier transform.

We see that the IDTFT is merely a Fourier series expansion of a signal with period 2π.

1.2.3 Discrete analysis

We would like to represent a digital signal as a sum of coefficients, as the Fourier series. To do so, we recognize the that we will only require a fixed amount of sinusoids.

For convenience, we will define $W_N = e^{i\frac{2\pi}{N}}$, which is called an Nth root of unity. Now, consider a periodic discrete signal of length N. Then the nth sinousiod reduces to

$$e^{ik\frac{2\pi}{N}(n+N)} = e^{ik\frac{2\pi}{N}n}e^{ik2\pi} = e^{ik\frac{2\pi}{N}n} = W_N^{kn} \tag{1.23}$$

Which means there are only a finite number of complex exponentials. If we apply this to the Fourier series, we get a sum from 0 to $N-1$.

Theorem 1.2.4 (Discrete Fourier Transform, DFT)

$$X[k] = \sum_{n=0}^{N-1} x[n]e^{-ik\frac{2\pi}{N}n} = \sum_{n=0}^{N-1} x[n]W_N^{-kn} \tag{1.24}$$

We can interpret this as evaluating the discrete-time Fourier transform at $n\frac{2\pi}{N}$, effectively sampling the continuous function $X(\omega)$ at intervals of $\frac{2\pi}{N}$. Thus, a signal of length N produces exactly N unique coefficients. We know from the Fourier series that these are complex conjugates of each other, and therefore only allows us to express $\frac{N}{2}$ real sinousoids.

An N-point DFT consists of N unique sinousoids, called *frequency bins*, each of which are centered at frequencies $k\frac{\omega_s}{N}$, for $k \in [0, N-1]$, and the k_0 bin is merely the average or *DC term*. The range of the bins collect contributions of frequencies at either side of the bin's center, spanning the range $\frac{\omega_s}{N}$.

Correspondingly, we get the inverse discrete Fourier transform,

Theorem 1.2.5 (Inverse discrete Fourier transform, IDFT)

$$x[n] = \frac{1}{N}\sum_{n=0}^{N-1} X[k]e^{ik\frac{2\pi}{N}n} = \frac{1}{N}\sum_{n=0}^{N-1} X[k]W_N^{kn} \tag{1.25}$$

Notice the discrete Fourier transform allows us to express the coefficients by Nth roots of unity, which is at the heart of the Fast Fourier Transform (see section 2.2, p. 10) algorithm.

1.2.4 Z-transform

As mentioned in passing the continuous-time Fourier transform is a special case of the Laplace transform. While we won't go into any depth about the Laplace transform itself, we will discuss its discrete equivalent, and some intuition of this is therefore granted.

Definition 1.2.2 (Laplace Transform) *Let $x(t)$ be a complex signal where $s \in \mathbb{C}$, then*

$$X(s) = \int_{-\infty}^{\infty} x(t)e^{-st}dt \tag{1.26}$$

defines the Laplace transform $X(s)$ of $x(t)$.

If we let $s = i\omega$, we get exactly the Fourier transform, which constrains the complex exponential in the Laplace transform to the unit circle in the complex plane. This key concept carries over into the discrete version of the Laplace transform, called the *Z-transform*.

Definition 1.2.3 (Z-Transform) *Let $x[n]$ be a complex signal where $z \in \mathbb{C}$, then*

$$X(z) = \sum_{n=-\infty}^{\infty} x[n] z^{-n} \tag{1.27}$$

defines the Z-transform $X(z)$ of the signal $x[n]$.

Similarly, if we let $z = e^{i\omega}$ we get exactly the DTFT. In other words if we constrain the magnitude of z, such that $|z| = 1$, then this is in fact the discrete-time Fourier transform.

These equations allow us to move from the time-domain (functions of time) to the frequency-domain (functions of frequency), in both the analog and digital realms.

1.3 Audio Processing

The term *mixing* is the process of balancing not only levels of each individual audio track, but more importantly controlling dynamics of- and frequency distribution among tracks.

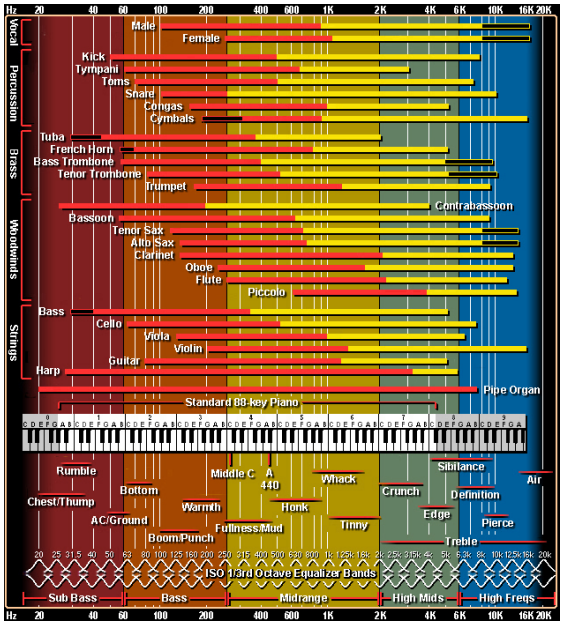


Figure 1.1: Frequency ranges of instruments (source: audio-issues.com)

1.3.1 Spectral Processing

Each audio track has its own frequency characteristics. By the additivity of the Fourier transform, when we sum all of the signals, we sum the frequency content. The signals compete for spectral room. I.e the frequency content of a kick drum and bass guitar are concentrated at 30–400Hz. This is why we 'clean up', or *filter*, signals. We may also want to emphasize certain frequencies of a signal, to make it stand out more in a mix.

1.3.2 Dynamics Processing

Controlling the dynamics of a track has several applications; evening out a signals that have quiet and load parts (i.e. vocals), emphasis on transients (i.e. drums), enhance sustain (i.e. guitar), etc. However, we will focus on spectral processing in this thesis.

1.3.3 Guitar tuner

For this project, I have chosen to implement a guitar tuner. I chose this for several reasons; 1) I hypothesize applying filtering to reduce noise will improve the tuner's ability to recognise the principal frequency, 2) for any reasonable input there is an expected output, and 3) its use is simple and easy to demonstrate.

A guitar tuner filters and analyzes an incoming signal, presumably a plugged guitar string, and determines the principal frequency in that signal. It then converts this into the corresponding musical note. If the guitar string is out of tune, it shows the deviation from the nearest note, by indicating *cents*, which is a ratio between two musical note frequencies — formally defined as $c = 1200 \log_2(\frac{f_1}{f_2})$. In simpler terms, it is merely a division into hundredths of the interval between musical notes — i.e. adding a 100 cents to E♭ yields E.

Because the theory is applicable for most of audio processing and analysis, I will present it in its general form, as the incentive of filtering the signal in the tuner will be the same as with any audio filtering; the reduction of noise and unwanted frequency content.

Filtering Methods

When we talk about processing, the signal is transformed by some mechanism or device, which is referred to as a *system*. In example, a sampler, which we have talked about, is a system. Here, we will talk about filters, which are also systems. The systems we will discuss will be linear time invariant (see appendix A.1, p. 37). From here on out, unless stated otherwise, when we refer to a system we will assume that it is LTI compliant.

A simple example of a filter system is the moving average (2.1) which averages the signal for each n over N units.

$$y[n] = \frac{1}{N} \sum_{k=-\lfloor \frac{N}{2} \rfloor}^{\lfloor \frac{N}{2} \rfloor} x[n-k] \tag{2.1}$$

2.1 Ideal Filters

As discussed (see section 1.3, p. 8), we are interested in filtering audio signals, so as to make room in the frequency spectrum for other signals, reduce noise or similar. In our case, we wish to remove any frequency content that doesn't constitute the range of the tuner.

The most important filters in this context are the *low-pass* (LP) and *high-pass* (HP) filters, which removes frequencies above or below, respectively, a so-called *cut-off* frequency ω_c. The preserved frequency range is called the *pass band*, while the frequency range being filtered out is called the *stop band*. The frequency range between the pass- and stop bands is called the *transition band*. Ideally, we would like the low- and high-pass filters to have as narrow a transition band as possible, as shown below.

$$H_{\text{lp}}(\omega) = \begin{cases} 1 & \text{iff. } \omega < |\omega_c| \\ 0 & \text{otherwise} \end{cases} \tag{2.2}$$

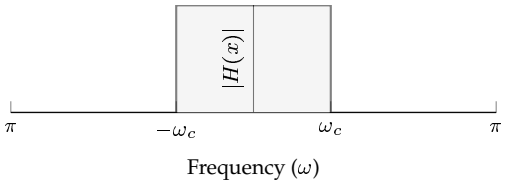


Figure 2.1: Ideal low-pass filter

$$H_{\text{hp}}(\omega) = \begin{cases} 0 & \text{iff. } \omega < |\omega_c| \\ 1 & \text{otherwise} \end{cases} \tag{2.3}$$

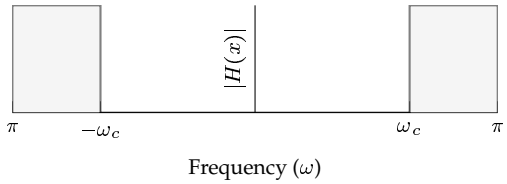


Figure 2.2: Ideal high-pass filter

The filters shown above have a so-called *brick-wall* transition band because of the discontinuity at ω_c, and this is the desired effect in the frequency domain. Let's examine the time domain of the low-pass filter. For convenience, let the height of the pass band be π,

$$h_{\text{lp}}(t) = \frac{1}{2\pi} \int_{-\infty}^{\infty} H_{\text{lp}}(\omega) e^{i\omega t} d\omega = \frac{1}{2\pi} \int_{-\omega_c}^{\omega_c} \pi e^{i\omega t} d\omega = \frac{\pi \left(e^{i\omega_c t} - e^{-i\omega_c t} \right)}{2\pi i t} = \frac{\sin \omega_c t}{t} \tag{2.4}$$

Notice we can write $h_{\text{lp}}(t) = \omega_c \frac{\sin \omega_c t}{\omega_c t}$, which is a sinc-function $\omega_c \operatorname{sinc} \omega_c t$, making the filter exhibit an undesirable property; it is *non-causal*. Meaning it depends on future samples.

Theorem 2.1.1 (Causality) *Let $H(x)$ be an LTI system, such that $y[n] = H(x[n])$. If $y[t_0]$, for any t_0 doesn't depend on $x[t_1]$, where $t_1 > t_0$, then the system $H(x)$ is causal.*

Now that we know how ideal filters behave in either domain, we may examine techniques for achieving the desired effect, and circumventing the undesirable ones.

2.2 Fast Fourier Transform

The easiest way of manipulating the frequency content of a signal is to treat it directly in the frequency-domain. In this case we use the *Fast Fourier Transform* (FFT) algorithm. While the algorithm itself is out of the scope of the thesis, we will briefly go over a few key ideas and properties. We will be looking at the *Radix-2 FFT*.

2.2.1 Complexity analysis

The regular DFT can be reduced to $2N^2$ operations in $\mathbb{R}$, giving the $O(N^2)$ running-time complexity, which is not a desirable in a real-time context.

The FFT is a divide-and-conquer approach by asserting that the length of $x[n]$ is a power of two, or simply zero-pad $x[n]$ if not, such that the $N = 2^L$ criterion is met, and then splits even and odd sums of the DFT, called *decimation in time*. To do so, we define $n = 2r$,

$$X[k] = \sum_{n=0}^{N-1} x[n] e^{-ik\frac{2\pi}{N}n} = \sum_{n=0}^{N-1} x[n] W_N^{kn} \tag{2.5}$$

$$= \sum_{r=0}^{N/2-1} x[2r] W_N^{2rk} + \sum_{r=0}^{N/2-1} x[2r+1] W_N^{(2r-1)k} \tag{2.6}$$

$$= \sum_{r=0}^{N/2-1} x[2r] (W_N^2)^{rk} + W_N^k \sum_{r=0}^{N/2-1} x[2r+1] (W_N^2)^{rk} \tag{2.7}$$

At this point, we recognize that we can leverage the periodicity of the complex exponential by substituting $W_N^2 = W_{N/2}$, allowing us to rewrite both sums such that they resemble DFT's of length $\frac{N}{2}$.

$$= \sum_{r=0}^{N/2-1} x[2r] W_{N/2}^{rk} + W_N^k \sum_{r=0}^{N/2-1} x[2r+1] W_{N/2}^{rk} \tag{2.8}$$

We end up with two DFT's of length $\frac{N}{2}$, where one is multiplied by a so-called *twiddle factor*, the sum of which yields the length N DFT. By repeating this procedure, we get a recursive algorithm that produces $\frac{N}{2}$ base case DFT's of length 2, which reduces to a simple addition and subtraction since $W_2 = -1$ and $W_1 = 1$.

Thus, the algorithm is a recursive tree-structure with a height of $O(\lg N)$ divide step. Each level of which has a linear conquer step $O(N)$. Giving the general complexity $O(N \lg N)$.

2.2.2 Circular convolution

The relationship between time- and frequency domain operations is given by the convolution theorem.

Theorem 2.2.1 (Convolution theorem) *Let $x(t)$ and $y(t)$ functions of time t, and let $X(\omega)$ and $Y(\omega)$ be their respective Fourier transforms. Then*

$$\mathcal{F}^{-1}\{X(\omega)Y(\omega)\} = (x \circledast y)(t) \quad \textit{and} \quad \mathcal{F}\{(x \circledast y)(t)\} = X(\omega)Y(\omega) \tag{2.9}$$

That is, multiplication in either domain is equivalent of circular convolution (2.10) in the other, and vice versa. While this property is probably the most important of all, allowing us to perform the same filter in either domain, in order to use it effectively, we must circumvent the circularity, which causes aliasing.

$$\begin{bmatrix} h[0] & h[n-1] & \dots & h[2] & h[1] \\ h[1] & h[0] & \ddots & \ddots & h[2] \\ \vdots & \ddots & \ddots & \ddots & \vdots \\ & \ddots & \ddots & h[0] & h[n-1] \\ h[n-2] & & & h[1] & h[0] \\ h[n-1] & h[n-2] & \dots & & \end{bmatrix} \begin{bmatrix} x[0] \\ x[1] \\ \vdots \\ \\ x[m-2] \\ x[m-1] \end{bmatrix} = \begin{bmatrix} y[0] \\ y[1] \\ \vdots \\ \\ y[m-2] \\ y[m-1] \end{bmatrix} \tag{2.10}$$

One solution is to recognise the length of the output of such a convolution in the time-domain. The length of the output of a signal and kernel of lengths m and n is $m + n - 1$. If we zero-pad such that $N \geq m + n - 1$ of the FFT, then we have eliminated the aliasing issue. However, we have implicitly multiplied by a rectangular window, which as shown (2.4) corresponds to a sinc in the other domain — this is known as spectral leakage.

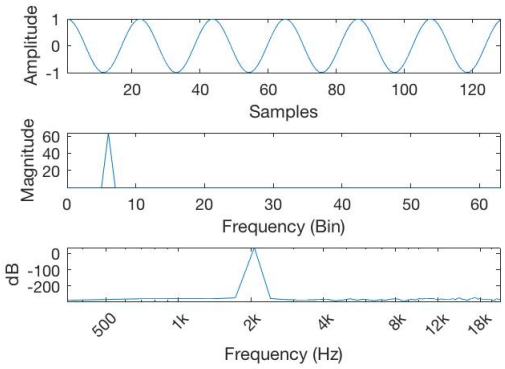


Figure 2.3: Non-padded cosine at 2kHz

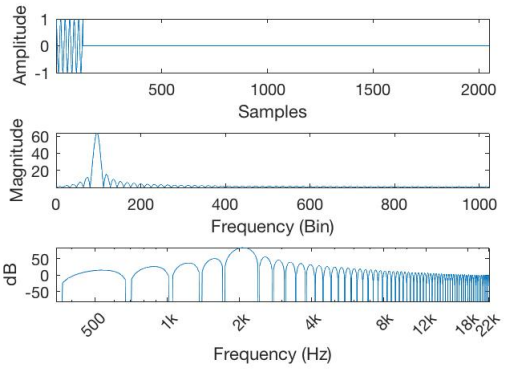


Figure 2.4: Zero-padded 2kHz cosine

As evident from the above figures, the single cosine leaks out into surrounding frequency bins. The padding applied above is exaggerated to show how the leakage spreads.

There are other reasons why zero-padding is needed in real-time applications. Particularly, in audio processing where we have to deliver an output with minimal latency, with no discernable delay. As such, each frame segment, discussed in the following section is usually very small — 4096 samples or less. But reducing N reduces the frequency resolution, therefore we zero-pad the signal in order to artificially produce a finer resolution.

2.2.3 Advantages

1. Easy to manipulate frequencies
2. Highly optimized algorithms

2.2.4 Disadvantages

1. Circular convolution in time
2. Spectral leakage

2.3 Short-time Fourier Transform

Knowing a signal in its entirety allows us to describe it completely as a sum of sinusoids. In a real-time setting, however, we only know a small part of the signal at any given moment.

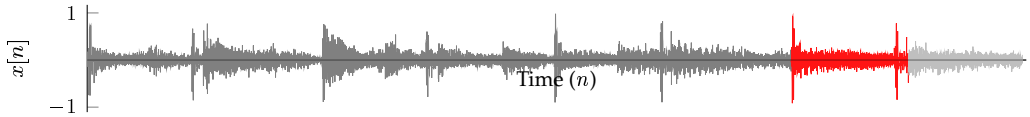


Figure 2.5: Signal, current (red), previous (dark grey), and future samples (light grey).

Each chunk is, as with zero-padding, implicitly multiplied by a rectangular window. We may account for this by adopting another window, which minimises the effect of spectral leakage, i.e. the *Hanning window*

$$w(x) = \frac{1}{2}\left(1 - \cos\left(\frac{2\pi x}{N-1}\right)\right) \tag{2.11}$$

and adapting our algorithm to accomodate the change, by employing the *Short-time Fourier Transform* (STFT). It takes the DFT of a windowed frame centered at the time mR,

$$X_m(\omega) = \sum_{n=-\infty}^{\infty} x[n]w[n - mR]e^{-i\omega n} \tag{2.12}$$

where R is the so-called *hop size*.

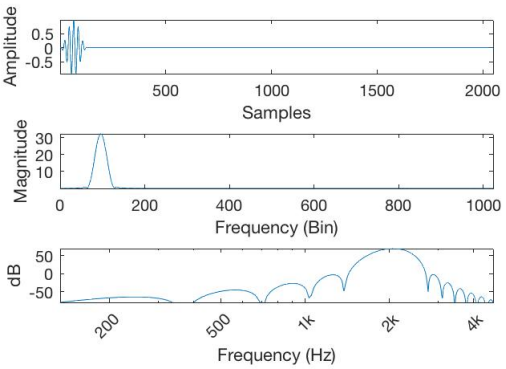


Figure 2.6: Zero-padded 2kHz cosine multiplied by Hanning window

Rather than moving ahead by the size of the frame, for consecutive frames, it moves ahead by R, and if

$$\sum_{m=-\infty}^{\infty} w[n - mR] = c, \forall n \in \mathbb{Z}, \text{ where } c \text{ is some constant.} \tag{2.13}$$

which is called the *constant overlap-add* property, then the sum of each intermediate DFT taken at intervals of R is equivalent to that of the entire DFT.

$$\sum_{n=-\infty}^{\infty} X(\omega) = \sum_{m=-\infty}^{\infty}\sum_{n=-\infty}^{\infty} x[n]w[n - mR]e^{-i\omega n} \tag{2.14}$$

$$= \sum_{n=-\infty}^{\infty} x[n]e^{-i\omega n} \sum_{m=-\infty}^{\infty} w[n - mR] = c\sum_{n=-\infty}^{\infty} x[n]e^{-i\omega n} \tag{2.15}$$

If we chose a window function where $c = 1$, we get exactly the regular DFT.

2.3.1 Advantages

1. Minimises spectral leakage
2. Effective reuse of twiddle factors
3. Circumvents circular convolution

2.3.2 Disadvantages

1. Recalculates each intermediate DFT
2. Each window is an $O(N)$-operation
3. High latency of at least N

2.4 Sliding Discrete Fourier Transform

The *Sliding DFT*[4] employs a more data-efficient approach by reusing the DFT bins of a frame for time t to calculate the bins of the following DFT at time $t+1$.

The DFT at time t is calculated as follows.

$$X_t(n) = \sum_{k=0}^{N-1} x[t+k]W_N^{-kn} \tag{2.16}$$

In a real-time context, we may assume that the initial frame is simply zeros, alleviating the need for an initial DFT. Sliding the window ahead from time t to $t+1$, we can then produce the DFT at time $t+1$ from the previous DFT.

$$X_{t+1}(n) = \sum_{k=0}^{N-1} x[t+k+1]W_N^{-kn} = \sum_{k=1}^{N-1} x[t+k]W_N^{-(k-1)n} \tag{2.17}$$

$$= \left(\sum_{k=0}^{N-1} x[t+k]W_N^{-kn} - f(t) + f(t+N) \right) W_N^n \tag{2.18}$$

$$= \left(X_t(n) - f(t) + f(t+N) \right) W_N^n \tag{2.19}$$

What happens when calculating the DFT at $t+1$ is a phase shift in the frequency domain, corresponding to the shift in the time domain. Effectively 'undoing' the impulse of the sample at t, and adding the impulse of the sample at $t+N$.

2.4.1 Advantages

1. Highly memory-efficient

2.4.2 Disadvantages

1. Moving by N samples requires $O(N^2)$

2.5 Convolution

If a system adheres to the properties of LTI, we can analyse its behaviour by putting an impulse (see definition A.2.1, p. 38) through it.

Let's suppose we have a system $\tau(x)$, then $h[n] = \tau(\delta[n])$ defines the *impulse response* of the filter. By linearity we have (2.21), and by time invariance we have (2.22).

$$y[n] = \tau(x[n]) = \tau\left(\sum_{k=-\infty}^{\infty} \delta[n-k]x[k] \right) \tag{2.20}$$

$$= \sum_{k=-\infty}^{\infty} \tau(\delta[n-k])x[k] \tag{2.21}$$

$$= \sum_{k=-\infty}^{\infty} h[n-k]x[n] \tag{2.22}$$

The latter of which is the discrete-time convolution of an arbitrary signal $x[n]$ by the impulse response $h[n]$. Thus, if we know the impulse response $h[n]$, we can produce $y[n]$ entirely by the time-domain convolution of $x[n]$ with $h[n]$.

Definition 2.5.1 (Discrete Convolution)

$$(x * y)[n] = \sum_{k=-\infty}^{\infty} x[k]y[n-k] \tag{2.23}$$

A problem becomes apparent in convolving ideal filters then; the impulse response extends out to infinity, making it impossible to realise in the time-domain.

In relation to convolution, if we don't 'flip' the kernel and conjugate the signal $x[n]$, we get the *correlation* of the signals $x[n]$ and $y[n]$, at some lag l.

Definition 2.5.2 (Discrete Correlation) *Let $x(t)$ and $y(t)$ be discrete functions of time n, then*

$$r_{xy}[l] = \sum_{n=-\infty}^{\infty} \overline{x}[n]y[n+l] \tag{2.24}$$

is the cross-correlation of signal x with y at lag l.

If the $x[n] \in \mathbb{R}$, then $\overline{x}[n] = x[n]$, as it is in our case, since we are dealing with real-valued signals. Further, if $x[n] = y[n]$, this is known as *auto-correlation*. The latter form of correlation is useful in determining periodicity within signals, which we will use it for later.

Correlation asserts that the signals don't deviate radically from each other. We can normalise the correlation formula to mend this.

Definition 2.5.3 (Normalised Correlation) *Let $x[n]$ and $y[n]$ be a discrete functions of time n, then*

$$\hat{r}_{yx}[l] = \frac{r_{xy}[l]}{\sqrt{r_{xx}[0]r_{xy}[0]}} \tag{2.25}$$

defines the normalised correlation of signals x and y at lag l.

This produces a more reliable correlation in the range $[-1; 1]$, regardless of deviation.

2.5.1 Complexity analysis

Given a signal $x[n]$ of length N and kernel $h[n]$ of length M. For each N entries in $x[n]$ we have M constant time operations, amounting to NM operations. In the worst case we have $N = M$, giving us a running-time complexity of $O(N^2)$.

However, in real-time, the kernel size M is usually very small. Further, the size of each frame is a common variable. Thus, the size M will determine whether convolution gives a lower running-time.

Let's fix N and examine at which M the running-times intersect. We have $NM = N \lg N \implies M = \lg N$, giving us the table in figure 2.7, to the right.

N	256	512	1024	2048	4096
M	8	9	10	11	12

Figure 2.7: Theoretical FFT and convolution running-time intersections

---

2.5.2 Advantages

1. Worst case almost never realised
2. Doesn't suffer spectral leakage
3. Minimal overhead and latency

2.5.3 Disadvantages

1. Filter design much more complicated
2. Must be designed with very low M

Filter Design

Now that we have discussed the various strategies of filtering, let's explore if we can design filters in the time-domain using convolution of adequate quality, while also keeping the size of such filtes low enough to compete with the FFT strategies.

3.1 Finite Impulse Response

Consider a running average filter. If we put an impulse through the filter, we get what is called a *finite impulse response* (FIR) filter. We obtain the generalized form of FIR filters by letting the averaging coefficient $\frac{1}{M}$ be a sequence of coefficients b_k.

$$y[n] = \sum_{k=0}^{M} b_k x[n-k] \tag{3.1}$$

Let's try to model an FIR approximation of h_{lp} of length N, where N is odd for convenience. We know from (2.4) that $h_{\text{lp}}(t) = h_{\text{lp}}(-t)$, so we let $h[n] = h[-n]$ and $M = \frac{N-1}{2}$, in which case we can simplify the filter by its symmetry

$$H(\omega) = \sum_{n=-M}^{M} h[n]e^{-i\omega n} = h[0] + \sum_{n=1}^{M} h[n]e^{-i\omega n} + \sum_{n=1}^{M} h[-n]e^{i\omega n} \tag{3.2}$$

$$= h[0] + \sum_{n=1}^{M} h[n](\cos\omega n - i\sin\omega n) + \sum_{n=1}^{M} h[-n](\cos\omega n + i\sin\omega n) \tag{3.3}$$

$$= h[0] + 2\sum_{n=1}^{M} h[n]\cos\omega n \tag{3.4}$$

This gives a finite length symmetric filter. Notice that the imaginary terms cancel. The filter violates the condition of causality, however. We can mend this by applying a delay of M units, giving us $h[n-M]$, which can be expressed as an equivalent filter, symmetric at M and $h[n] = h[N-n-1]$. Rearranging the sum to accomodate the change, we get

$$H(\omega) = e^{i\omega(-M)}\left(h[M] + \sum_{n=0}^{M-1} 2h[n]\cos(\omega(M-n))\right) \tag{3.5}$$

Equivalently, for a filter with an even length N, we have that

$$H(\omega) = e^{i\omega(-M)}\sum_{n=0}^{M} 2h[n]\cos(\omega(M-n)) \tag{3.6}$$

Let $A(\omega) = |H(\omega)|$ in the above, and consider the convolution of $X(\omega)$ for some signal $x[n]$ with the above filter $H(\omega)$.

$$X(\omega)H(\omega) = |X(\omega)||H(\omega)|e^{i(\angle H(\omega)+\angle X(\omega))} = |X(\omega)|A(\omega)e^{i(\angle X(\omega)-\omega M)} \tag{3.7}$$

Notice that the *phase response* $\angle H(\omega)$ is a linear function $-\omega M$, called *linear phase*. This is a consequence of the cancellation of $i \sin$-terms when a filter is symmetric, as derived above. It is a desirable effect because each frequency is shifted by a constant amount, which merely a delay in time, and not considered a distortion of the input signal.

Any symmetric and anti-symmetric FIR filter is *always* a *linear phase filter*. Below is a list of all four linear phase filter types, which phase and amplitude functions.

Type	N	Symmetry	θ	$A(\omega)$
I	Odd	Symmetric	$-M\omega$	$h[M] + 2\sum_{n=0}^{M-1} h[n]\cos((M-n)\omega)$
II	Even	Symmetric	$-M\omega$	$2\sum_{n=0}^{M} h[n]\cos((M-n)\omega)$
III	Odd	Anti-symmetric	$\pi/2 - M\omega$	$2\sum_{n=0}^{M-1} h[n]\sin((M-n)\omega)$
IV	Even	Anti-symmetric	$\pi/2 - M\omega$	$2\sum_{n=0}^{M} h[n]\sin((M-n)\omega)$

Figure 3.1: Types of linear phase FIR filters

FIR filters have many other applications in audio as well. Impulse responses can be sampled from reverberation, guitar cabinets, etc., to emulate rooms or pieces of gear.

3.1.1 Frequency sampling method

Given that we can express any of the aforementioned filters entirely by $A(\omega)$, we can sample N evenly spaced frequencies, which corresponds to the DFT coefficients, and take the IDFT to obtain the time-domain equivalent $h_{\text{lp}}[x]$.

Let's assume a type I linear phase FIR filter of length $N = 63$ and a cut-off frequency $\omega_c = 8\frac{2\pi}{N}$. The phase is then $-M\omega = -\left(\frac{N-1}{2}\right)\omega = -31\omega$ and the set of sampled DFT coefficients of $A_{lp}(\omega)$, sampled at $\frac{2\pi}{N}$ is denoted as A_{lp}. If we plug in, we have the following

$$h_{\text{lp}}[n] = \frac{1}{N}\sum_{n=0}^{N-1} H_{\text{lp}}(\omega)e^{ik\frac{2\pi}{N}n} = \frac{1}{N}\sum_{n=0}^{N-1} A_{lp}[n]e^{-M\omega}e^{ik\frac{2\pi}{N}n} \tag{3.8}$$

Which, by calculation, results in the corresponding $h[n]$ filter, shown below.

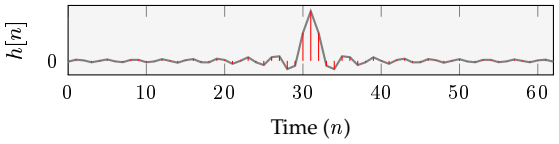


Figure 3.2: Example impulse response of type I linear phase FIR filter

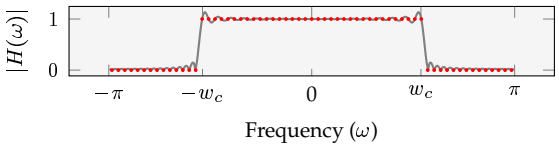


Figure 3.3: Example magnitude response of type I linear phase FIR filter

The delay in time is quite obvious from figure 3.2. Notice the resemblence with $\omega_c \operatorname{sinc} \omega_c t$ derived earlier. The corresponding magnitude response $|H(\omega)|$ is shown in figure 3.3.

While the method requires little effort to produce a filter, there is a significant error around ω_c. This *ringing* effect is known as the *Gibbs phenomenon*, and no matter how large we make N, the error is proven to remain at around 9% of the discontinuity[1].

3.1.2 Advantages

1. Minimal phase distortion
2. Always stabile (see section 3.2.2, p. 17)

3.1.3 Disadvantages

1. Delay is proportional with quality
2. Error at the cut-off frequency

3.2 Infinite Impulse Response

As we increase the quality of FIR filters, they get larger in size and introduce delays, resulting in latency. And, as explained, we want to keep the filter size small enough to compete with FFT. If we introduce feedback into the system, we get an *infinite impulse response*.

$$\sum_{k=0}^{N} a_k y[n-k] = \sum_{k=0}^{M} b_k x[n-k] \tag{3.9}$$

$$a_0 y[n] = \sum_{k=0}^{M} b_k x[n-k] - \sum_{k=1}^{N} a_k y[n-k] \tag{3.10}$$

The difference equation (3.9) introduces the feedback, and by isolating $y[n]$, we get the general form of an IIR filter (3.10). Usually we normalise the coefficients such that $a_0 = 1$.

3.2.1 Transfer function

Applying the Z-transform to the difference equation (3.9) gives us

$$Y(z)\sum_{k=0}^{N} a_k z^{-k} = X(z)\sum_{k=0}^{M} b_k z^{-k} \implies Y(z) = X(z)\frac{\sum_{k=0}^{M} b_k z^{-k}}{\sum_{k=0}^{N} a_k z^{-k}} \tag{3.11}$$

which by the convolution theorem implies that

$$H(z) = \frac{\sum_{k=0}^{M} b_k z^{-k}}{\sum_{k=0}^{N} a_k z^{-k}} = \frac{\sum_{k=0}^{M} b_k z^{-k}}{1 + \sum_{k=1}^{N} a_k z^{-k}} \tag{3.12}$$

Note that if $a_k = 0, \forall k \neq 0$, then it's an FIR filter, giving us the Z-transform of FIR filters.

$$H(z) = \sum_{k=0}^{M} b_k z^{-k} \tag{3.13}$$

3.2.2 Region of convergence

As alluded to, the Fourier transform is related to the Z-transform by the magnitude of z.

Let's be explicit about the magnitude, and substitute z with its polar form in the Z-transform.

$$X(z) = X(re^{i\omega}) = \sum_{n=-\infty}^{\infty} (x[n]r^{-n})e^{-i\omega n} \tag{3.14}$$

Let $\hat{x}[n] = x[n]r^{-n}$ be a signal, then the discrete-time Fourier transform of $\hat{x}[n]$ is exactly the Z-transform of $x[n]$. Or, vice versa, let $r = 1$, then the Z-transform is merely the DTFT.

The significance of r becomes apparent when we consider that the Z-transform convergences if $\sum_{n=-\infty}^{\infty} |x[n]r^{-n}| < \infty$, and if this convergence is true for $r = 1$, then so does the DTFT. The Z-transform may converge in places where the DTFT does not, and by analysing the range of r in this convergence, we can determine if the DTFT convergences as well. If the DTFT converges at every point in the unit circle of the complex plane, we say that the system is *stabile*, meaning a that is filter doesn't blow up and produce ∞-values.

Definition 3.2.1 (Stability) *Let $h[n]$ be an LTI-compliant system. If the Z-transform of $h[n]$ converges for $r = 1$, then the region of convergence includes the unit circle and the DTFT exists, and the system is considered stabile.*

3.2.3 Poles and zeros

Observing that $x[n]z^{-n} = x[n](z^{-1})^n$ makes it clear that $X(z)$ is a complex polynomial, and can usually be expressed as a rational function.

When this is true, as previously shown (3.12) to be the case for FIR and IIR filters, we have a transfer function of form $H(z) = \frac{N(z)}{D(z)}$, by which we can easily identify what is known as the *poles* (3.15) and *zeros* (3.16) of $H(z)$, as shown below.

$$D(z) = 0 \implies X(z) = \infty \tag{3.15}$$

$$N(z) = 0 \implies X(z) = 0 \tag{3.16}$$

Knowing the poles and zeros, we can write $H(z)$ in *pole/zero form*.

Definition 3.2.2 (Transfer function pole/zero form) *Let c_i the be poles and d_i the be zeros of some transfer function $H(z)$. Assuming $b_0 \neq 0$, then*

$$H(z) = g\frac{\prod_{i=1}^{M}(z - c_i)}{\prod_{i=1}^{N}(z - d_i)} = g\frac{\prod_{i=1}^{M}(1 - c_i z^{-1})}{\prod_{i=1}^{N}(1 - d_i z^{-1})}, \text{ where } g = \frac{b_0}{a_0} \tag{3.17}$$

is the pole/zero form of the transfer function. The factor g is called the 'gain factor'.

The poles and zeros of the Z-transform allows us to analyse the behaviour of a filter in the frequency-domain by expressing the magnitude response $|H(e^{i\omega})|$ by the pole/zero form of a transfer function

$$|H(e^{i\omega})| = g\frac{\prod_{i=1}^{M}|1 - c_i e^{-i\omega}|}{\prod_{i=1}^{N}|1 - d_i e^{-i\omega}|} = g\frac{\prod_{i=1}^{M}|e^{i\omega} - c_i|}{\prod_{i=1}^{N}|e^{i\omega} - d_i|} \tag{3.18}$$

In doing so, we get a ratio between the distance from each of the zeros and poles to a point on $e^{i\omega}$. Intuitively, the closer zeros are to some point on $e^{i\omega}$, the less the magnitude response becomes. Conversely, the closer poles are to some point on $e^{i\omega}$, the stronger the magnitude response becomes. The above (3.18) also makes it quite easy to see that, if a zero and a pole are equal to each other, they simply cancel out each other, and that there are M and N potential poles and zeros, respectively. Thus, order corresponds to poles and zeros.

3.2.4 Bilinear transformation

The most common, tried and trusted way of designing IIR filters is to design it as an analog filter, and then convert it into a digital approximation. The analog design process has been around for much longer than the digital, and closed-form solutions exists for well-known filter types. Analog filters are designed and expressed in the Laplace domain.

The bilinear transform is a mapping $H(s) \to H(z)$, which we get by expressing z in terms of s, and apply Euler's formulae to reduce the expression.

$$z = e^{sT} = \frac{e^{sT/2}}{e^{-sT/2}} \approx \frac{1 + sT/2}{1 - sT/2} \implies s = \frac{1}{T}\ln z, \text{ where } T \text{ is the sampling period} \tag{3.19}$$

The $\ln z$ term can be approximated using either Taylor- or Laurent series,

$$\ln z = 2\left[\frac{z-1}{z+1} + \frac{1}{3}\left(\frac{z-1}{z+1}\right)^3 + \frac{1}{5}\left(\frac{z-1}{z+1}\right)^5 + \dots\right] \approx 2\left(\frac{z-1}{z+1}\right) \tag{3.20}$$

which gives us the bilinear transform, named after its two linear functions in the numerator and denominator.

Definition 3.2.3 (Bilinear transform)

$$s \leftarrow \frac{2}{T}\left(\frac{z-1}{z+1}\right) = \frac{2}{T}\left(\frac{1-z^{-1}}{1+z^{-1}}\right) \quad \text{, and conversely} \quad z \leftarrow \frac{1+sT/2}{1-sT/2} \tag{3.21}$$

The above substitutes s in an analog filter, and effectively maps the complex s-plane of the Laplace transformation to the z-plane of the Z-transformation.

Recall that $z = re^{i\omega}$, and we may write $s = \sigma + i\Omega$. If we substitute into the formula and rearrange, we have that

$$\sigma + i\Omega = \frac{2}{T}\frac{re^{i\omega}-1}{re^{i\omega}+1} = \frac{2}{T}\left(\frac{r^2-1}{1+r^2+2r\cos(\omega)} + i\frac{2r\sin(\omega)}{1+r^2+2r\cos(\omega)}\right) \tag{3.22}$$

$$\therefore \sigma = \frac{2}{T}\frac{r^2-1}{1+r^2+2r\cos(\omega)}, \text{ and } \Omega = \frac{2}{T}\frac{2r\sin(\omega)}{1+r^2+2r\cos(\omega)} \tag{3.23}$$

It then follows that when $r = 1$, then $\sigma = 0$. Therefore, the bilinear transform maps the $i\Omega$-axis to the unit circle of the z-plane. Furthermore, if $r > 1$ then $\sigma > 0$. Therefore the right-hand side of the s-plane is mapped outside the unit circle of the z-plane — the unstable region. Conversely, the stable region inside the unit circle $0 \leq r \leq 1$, corresponds to the left-hand side of the s-plane.

Frequency warping

The distinction between analog Ω (radians per second) and digital ω (radians per sample) frequencies becomes apparent if we let $r = 1$ and hence $\sigma = 0$, and isolate either one,

$$s = i\Omega = \frac{2}{T}\frac{z-1}{z+1} = \frac{2}{T}\frac{z-1}{z+1} = \frac{2}{T}\frac{e^{i\omega}-1}{e^{i\omega}+1} = \frac{2}{T}\frac{e^{i\omega/2}e^{i\omega/2}-e^{i\omega/2}e^{-i\omega/2}}{e^{i\omega/2}e^{i\omega/2}+e^{i\omega/2}e^{-i\omega/2}} \tag{3.24}$$

$$= \frac{2}{T}\frac{e^{i\omega/2}(e^{i\omega/2}-e^{-i\omega/2})}{e^{i\omega/2}(e^{i\omega/2}+e^{-i\omega/2})} = \frac{2}{T}\frac{e^{i\omega/2}-e^{-i\omega/2}}{e^{i\omega/2}+e^{-i\omega/2}} = \frac{2}{T}\frac{\frac{2i}{2i}(e^{i\omega/2}-e^{-i\omega/2})}{\frac{2}{2}(e^{i\omega/2}+e^{-i\omega/2})} \tag{3.25}$$

$$\implies \Omega = i^{-1}\frac{2}{T}\frac{2i\sin(\frac{\omega}{2})}{2\cos(\frac{\omega}{2})} = \frac{2}{T}\frac{\sin(\frac{\omega}{2})}{\cos(\frac{\omega}{2})} = \frac{2}{T}\tan\left(\frac{\omega}{2}\right) \tag{3.26}$$

And conversely, isolating ω, we have that $\omega = 2\tan^{-1}\left(\Omega\frac{T}{2}\right)$.

The non-linear relationship between continuous-time frequencies and discrete-time frequencies is quite evident — this is called *frequency warping*.

As a consequence, the range $[-\pi;\pi]$ in ω —where we define discrete-time frequencies— the function is injective, eliminating frequency aliasing, which occur in other methods, like *impulse invariance*.

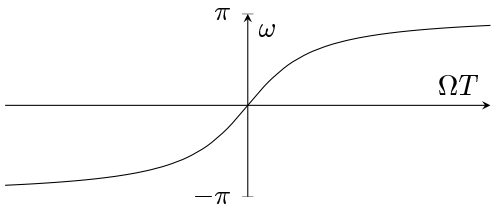


Figure 3.4: Frequency warping

Analog filters are usually designed from a low-pass model, and always with a *normalised critical frequency* $\Omega = 1$[3], which is later scaled in accordance with the specification. The so-called *prototype filter* is then transformed by *frequency transformation* to give other filter types, such that the high-pass filter. If we choose Ω accordingly, then T can be determined

$$1 = \frac{2}{T}\tan\left(\frac{\omega}{2}\right) \implies \frac{2}{T} = \cot\left(\frac{\omega}{2}\right) \tag{3.27}$$

Substituting into (3.21), we get the *normalised bilinear transform*.

Definition 3.2.4 (Normalised bilinear transform)

$$s \leftarrow \frac{1}{\tan\left(\frac{\omega}{2}\right)} \frac{1 - z^{-1}}{1 + z^{-1}} \tag{3.28}$$

We can substitute directly, and reduce the expression to match the Z-transfer function (3.12), or we can derive some substitution rules, making it easier to convert.

First, we recognise these identities[5],

$$\tan\left(\frac{\omega}{2}\right) = \frac{\sin\omega}{1 + \cos\omega} \tag{3.29}$$

$$\left(\tan\left(\frac{\omega}{2}\right)\right)^2 = \frac{1 - \cos\omega}{1 + \cos\omega} \tag{3.30}$$

and for each order of s (up to s^2 is sufficient for our purposes), we substitute and reduce;

$$1 = s^0 \leftarrow \frac{1 + \cos\omega}{1 + \cos\omega} \frac{1 + 2z^{-1} + z^{-2}}{1 + 2z^{-1} + z^{-2}} \tag{3.31}$$

$$s^1 \leftarrow \frac{1 + \cos\omega}{\sin\omega} \frac{1 - z^{-2}}{1 + 2z^{-1} + z^{-2}} \tag{3.32}$$

$$s^2 \leftarrow \frac{1 + \cos\omega}{1 - \cos\omega} \frac{1 - 2z^{-1} + z^{-2}}{1 + 2z^{-1} + z^{-2}} \tag{3.33}$$

Factoring out $\frac{1+\cos\omega}{1+2z^{-1}+z^{-2}}$, we get

$$1 \leftarrow \frac{1 + 2z^{-1} + z^{-2}}{1 + \cos\omega} \tag{3.34}$$

$$s \leftarrow \frac{1 - z^{-2}}{\sin\omega} \tag{3.35}$$

$$s^2 \leftarrow \frac{1 - 2z^{-1} + z^{-2}}{1 - \cos\omega} \tag{3.36}$$

Further, multiplying by $(\sin\omega)^2$, we can remove the fractions

$$1 \leftarrow (1 + 2z^{-1} + z^{-2})(1 - \cos\omega) \tag{3.37}$$

$$s \leftarrow (1 - z^{-2})\sin\omega \tag{3.38}$$

$$s^2 \leftarrow (1 - 2z^{-1} + z^{-2})(1 + \cos\omega) \tag{3.39}$$

Giving us an easy set of substitution rules for the bilinear transform.

3.2.5 Biquadratic IIR

Designing analog filters is a huge subject, and far beyond our scope. Butterworth filters are characterized by their flat passband, in contrast to Chebyshev and Elliptic filters. Which is why I chose this particular model.

I give the simplest normalised 2nd. order (Butterworth) low-pass model.

$$H(s) = \frac{1}{s^2 + \frac{s}{Q} + 1} \tag{3.40}$$

where the *quality factor* Q is the ratio between the energy loss in the transition band and the resonance in the near ω_c.

A Q of $\frac{1}{\sqrt{2}}$ is called the *half-power point*, as the energy loss at the cut-off frequency is $20\log\left(\frac{1}{\sqrt{2}}\right)$, which corresponds to half a drop in power $10\log\left(\frac{1}{2}\right)$.

Applying the bilinear transform by the substitutions (3.37), (3.38) and (3.39) to the filter (3.40), and reduce the expression to match the Z-transform transfer function (3.12), we get

$$H(z) = \frac{(1 + 2z^{-1} + z^{-2})(1 - \cos\omega)}{(1 - 2z^{-1} + z^{-2})(1 + \cos\omega) + (1 - z^{-2})\sin\omega\frac{1}{Q} + (1 + 2z^{-1} + z^{-2})(1 - \cos\omega)} \tag{3.41}$$

$$= \frac{((1 - \cos(\omega))/2) + (1\cos(\omega))z^{-1} + ((1 - \cos(\omega))/2)z^{-2}}{(1 + \sin(\omega)/2Q) - 2\cos(\omega)z^{-1} + (1 - \sin(\omega)/2Q)z^{-2}} \tag{3.42}$$

the digital equivalent, where the coefficients are expressed as formulae dependent on the quality factor Q and normalised discrete frequency $\omega = \omega_c\frac{2\pi}{\omega_s}$, where ω_c is in Hz. In this case an IIR filter with two quadratic equations in its difference equation, aptly called a biquadratic filter — often abbreviated *biquad*. We need only set the desired ω and Q. The same procedure can be repeated for any analog filter, yielding its digital equivalent.

We only need low- and high-pass filters, to remove frequencies outside the range of the tuner (see section 5.1.5, p. 28). The 2nd. order Butterworth analog high-pass filter is given by $H(s) = \frac{s^2}{s^2+\frac{s}{Q}+1}$, and their resulting responses are shown as part of the figure 3.8 and 3.9 in the following section.

3.2.6 Cascaded filters

For $Q > \frac{1}{\sqrt{2}}$ we get a resonance peak in the passband. If we wish to increase the slope in the transition band without affecting the passband, we could devise a higher order filter, but as order increases, so does the complexity of the design, and higher order filters are much more likely to become unstable due to numerical precision as poles get closer together. Instead, we apply filters in *cascade*, producing a higher order, while maintaining the stability.

Consider the poles in the s-domain as vectors. As previously shown (see section 3.2.3, p. 18), an Nth order filter produce N poles. Like the Z-domain, these are mirrored conjugates over the σ-axis. Unlike the Z-domain, however, the magnitude of the pole vector is the frequency and the angle corresponds to the Q. If we want to apply filters in cascade and maintain the Q, each pole must be adjusted accordingly.

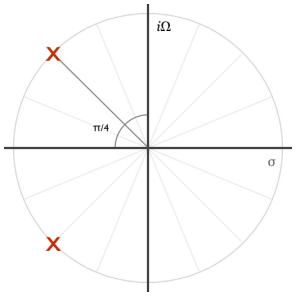


Figure 3.5: Poles in the s-plane of a 2nd order biquad

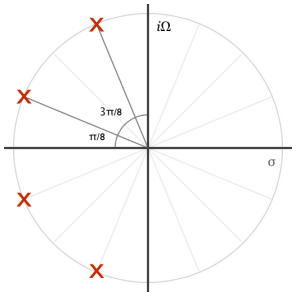


Figure 3.6: Poles in the s-plane of a 4th order biquad

n	Q
2	$1/\sqrt{2} = 0.7071$
3	0.5, 1.0
4	0.5412, 1.3067
5	0.5, 0.6180, 1.6180
6	0.5176, 0.7071, 1.9319
7	0.5, 0.5550, 0.8019, 2.2470
8	0.5098, 0.6013, 0.9, 2.5629

Figure 3.7: Table of (approximate) Q for nth order Butterworth filters (stage ordered)

An nth order filter will have an angle $\frac{\pi}{n}$ between each pole. This gives us the formula for the Q for a given pole angle θ; $1/2\cos(\theta)$. And, trivially the magnitude of the vectors, or frequency, stay the same. Since these are normalised, they are of unit length, as explained.

Choosing the half-power point Q, where Butterworth filters are maximally flat, we get the angle $\frac{\pi}{4}$, or $\frac{\pi}{2}$ between the poles of a biquad. A 2-staged biquad gives an angle of $\frac{\pi}{8}$, etc.

Note that we can have odd order filters, as suggested in the table above. This is merely a matter of adding a one-pole filter, and adjusting the Q angles of the biquads accordingly.

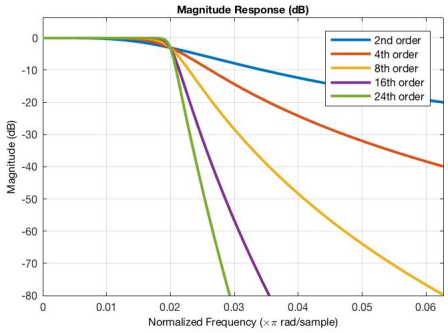


Figure 3.8: Magnitude response of cascaded low-pass biquads with cut-off at 440 Hz

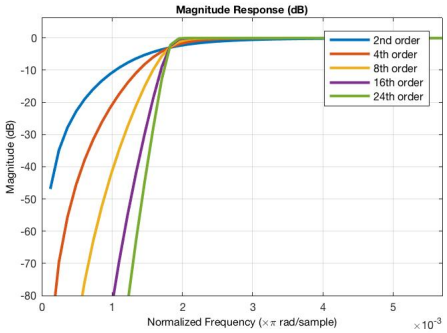


Figure 3.9: Magnitude response of cascaded hp-pass biquads with cut-off at 40 Hz

As evident from figure 3.8 and 3.9, cascaded filters drastically increase the slope in the transition band, without affecting the pass band. Furthermore, the magnitude responses clearly show the half-power attenuation at the cut-off frequency, no matter the filter order.

3.2.7 Advantages

1. Low delay penalty and efficient
2. Steeper attenuation slopes
3. Can applied in cascade

3.2.8 Disadvantages

1. Non-linear phase
2. Can be unstable
3. Difficult to design

Analysis

4.1 Phase Distortion

It is evident from figures 4.1 and 4.2, contrary to the FIR filters discussed, the IIR filters' phase response is not linear. As previously shown (see section 3.1, p. 15), linear phase is a direct consequence of the symmetry or anti-symmetry of the impulse response. Therefore, IIR filters can never achieve this property by definition. This non-linearity is constitutes distortion of the signal.

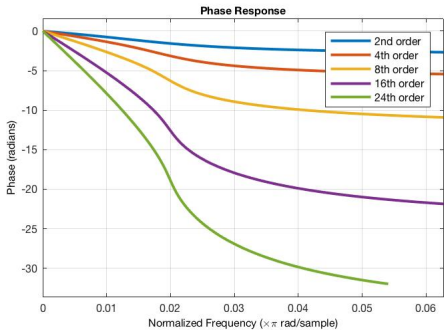


Figure 4.1: Phase response of cascaded low-pass biquads with cut-off at 440 Hz

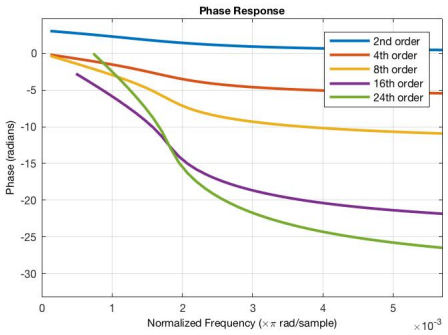


Figure 4.2: Phase response of cascaded high-pass biquads with cut-off at 40 Hz

While non-linear phase response indicates distortion, it does so in terms of radians added to each sinusiodal component of the signal. It is much more intuitive to analyse the severity of it in terms of samples, or delay, shown in figure 4.3 and 4.4.

Definition 4.1.1 (Phase delay) *Let $\theta(\omega)$ be a phase response, then*

$$P(\omega) = -\frac{\theta(\omega)}{\omega} \tag{4.1}$$

defines the delay of each sinusoidal component.

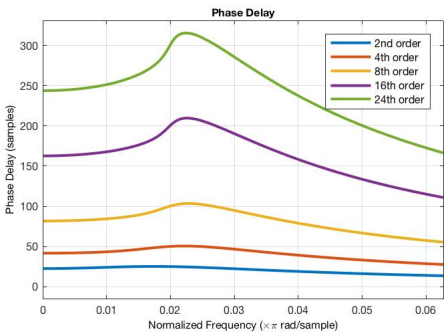


Figure 4.3: Phase delay of cascaded low-pass biquads with cut-off at 440 Hz

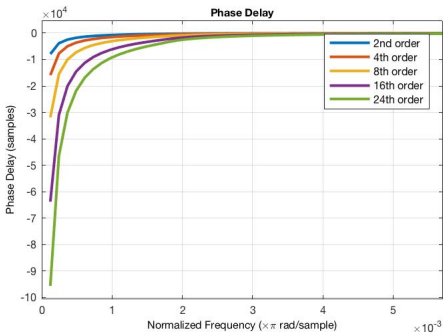


Figure 4.4: Phase delay of cascaded high-pass biquads with cut-off at 40 Hz

Further, if we take the derivative of the phase response, we can graph the rate of change in phase, called *group delay*, shown in figure 4.5 and 4.6.

Definition 4.1.2 (Group delay) *Let $\theta(\omega)$ be a phase response, then*

$$D(\omega) = -\frac{d}{d\omega}\theta(\omega) \tag{4.2}$$

defines the rate of change in phase.

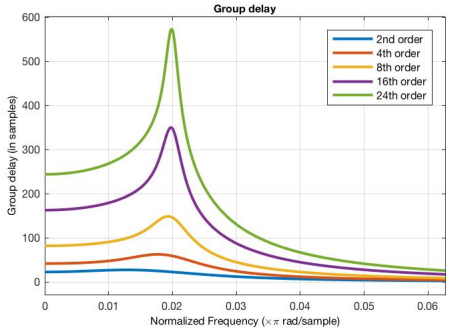


Figure 4.5: Group delay of cascaded low-pass biquads with cut-off at 440 Hz

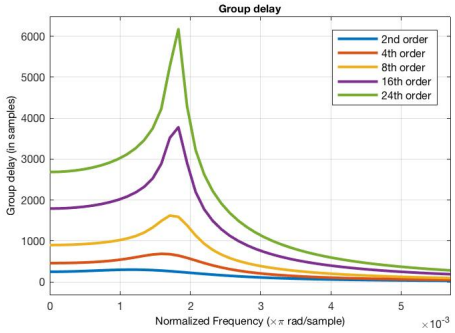


Figure 4.6: Group delay of cascaded high-pass biquads with cut-off at 40 Hz

Depending on the severity and application, we may want to reconsider the choice of filter. In our case, the effect isn't meant to be audible. Any delay in phase is therefore not of significant importance. If we were filtering audio for anything audible, we may want to either revise the filter, or compensate for the distortion in some way.

4.2 Stability

Plotting the poles and zeros (figure 4.7) of the filters allows us to examine the behavior of the filter. Crucially, we can visually inspect the stability of the filter. Recall, that stability in the Z-domain is dependent on poles being inside the unit circle.

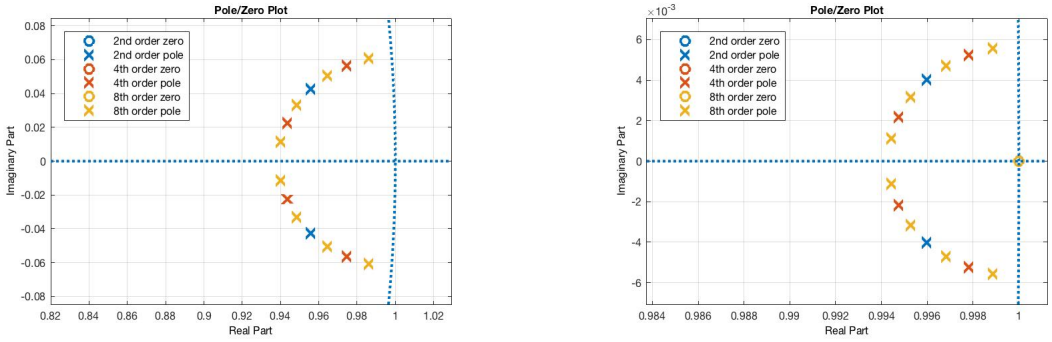


Figure 4.7: Pole-zero diagrams (zoomed) of low- (left) and highpass (right) filters

We can make use of the intuition given (see section 3.2.3, p. 18) to verify that they are indeed performing the desired filtering. The low-pass filter has all its zeros placed at -1, making the frequency response attenuate as we approach the maximum frequency. And the poles located around 1 amplifies the frequency response at low frequencies.

Similarly, the high-pass filter has all its poles in the low frequency range, to pull up the response immediately following the zero at 1 that attentuates the minimum frequency.

In both cases all poles are contained within the boundaries of the unit circle, and therefore both filters are by definition (see definition 3.2.1, p. 17) stabile.

Implementation

As a concrete example application of the techniques discussed, with simplicity in mind, I have chosen to implement a tuner. It relies on a single input, making it easy to demonstrate. Any frequencies outside the detection range of the tuner will be considered noise, and should be filtered out of the signal. Therefore, the tuner will employ a low-pass filter to limit the range and remove noise, and a high-pass filter to remove DC-offset.

While, as pointed out, FFT is an ideal choice for frequency spectrum analysis, I found that the FFT approach would not help in identifying the fundamental frequency of a guitar string — even for filtered signals.

As evident from figure 5.1, plugging an E♭ string, which has a fundamental frequency of 77,87 Hz, reveals overtones at multiples thereof. The overtones are seemingly more powerful, making it difficult to determine the fundamental frequency.

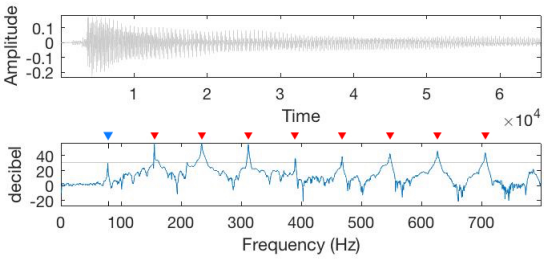


Figure 5.1: FFT of E♭ string. Fundamental frequency (blue arrow), and overtones (red arrows).

Recognising that the signal of a musical instrument is mostly periodic, aside from the transient, I reckon that a correlation (see definition 2.5.3, p. 14) might provide a more reliable musical tone detection algorithm. That is, we want to determine the lag l at which the normalised auto-correlation is maximal. Once found, the fundamental frequency f can be calculated by $f = \frac{\omega_s}{l}$, where ω_s is the sample rate.

I choose to view the signal starting at lag l as a different signal y. We then seek to maximise the normalised cross-correlation of x and y;

$$\hat{r}_{xy}[l] = \frac{r_{xy}[l]}{\sqrt{r_{xx}[0] r_{yy}[0]}} \tag{5.1}$$

As l grows large, so do the operations needed to perform the correlation. Finding candidates for l in advance allows us to minimise the number of correlations. By the definition of periodicity $x[n_0] = x[n_0 + l]$, therefore the algorithm uses peak-detection to provide candidates for l, since such a peak must exist at $x[n_0]$ and $x[n_0 + l]$. The first peak detected at n_0 is considered the reference signal, and following peaks are considered candidates for l.

5.1 Algorithm

The algorithm maintains a fixed-sized circular buffer A of n samples and two lists P and R of reference points into the buffer. The circular buffer is simply an array and an index, indicating the head and tail of the buffer.

A reference point is a data structure which holds information about its `index` into A and its sample `value`. It also contains `yy` and `xy` which holds running correlations of A, as

well as `nc` which is the normalised correlation. The `x` and `y` naming refers to references, or indices into A — `y` being the reference point itself, while `x` is the reference of the least index in the list, or zero-lag reference, corresponding to the first element of the list, as they are added sequentially.

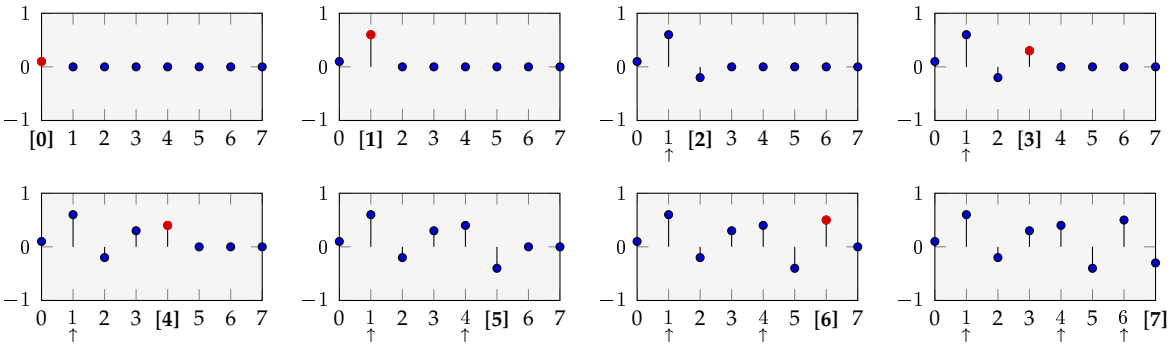
Reference points are found by keeping track of the highest positive DC-offset, or peak value, and a reference to the peak is added to P when the signal crosses the DC line. This is referred to as the *peak-detection* step. Its purpose is to find candidate offsets into A that are likely to produce high correlations.

5.1.1 Peak-detection Example

A visual illustration of a simple case where $n = 8$ is given to make it easier to understand how references are found and added to the P list.

The initial condition of each pass is that the buffer `index` (indicated by brackets) is 0. The buffer A always contain the samples of the previous frame, and is zeroed out on the first pass. Likewise the list P (elements indicated by ↑) is empty. The current tracked highest DC-value, or peak candidate, is shown in red.

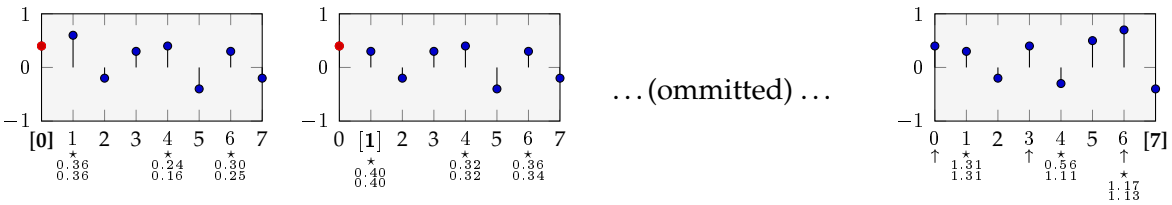
Given the sequence $\{0.1, 0.6, -0.2, 0.3, 0.4, -0.4, 0.5, -0.3\}$, we get the following,



This concludes the first pass, as `index` modulus the length n becomes 0, resetting `index` and meeting the initial condition of each pass. The buffer A is now full and P contains references to all peaks in the current frame. The lists are then swapped, such that $R = P$, and P is emptied to meet the initial condition of each pass.

5.1.2 Correlation Example

Now, the references in R (elements indicated by $\star$) represent lag candidates, that are likely to produce high correlations. On following pass, the first candidate, as explained, is used as the zero-lag reference and subsequent candidates are correlated with this, producing the `xy` correlation. Further, all references are correlated with themselves, giving the `yy` to later produce a normalised correlation `nc`. The former two correlations are shown underneath the corresponding $\star$ in order (`xy` on top and `yy` at the bottom). Note that correlations happen *before* adding the new sample to the buffer, making it possible to read and correlate the current frame whilst writing the subsequent frame, even when correlating at index 0. So, given the subsequent sequence $\{0.4, 0.3, -0.2, 0.4, -0.3, 0.5, 0.7, -0.4\}$, we get the following.



List R now contains correlations for each candidate reference point. Upon entering a new pass, before swapping the lists, we normalise the correlations, determine the maximum (excluding the zero-lag reference) — the result is the algorithm's best periodic match.

Let x and y denote the signal starting at the first reference and any of candidate reference, respectively. And let $\hat{r}_n$ denote the normalised correlation of r_{xy}. Following the examples above, we have that

$$\hat{r}_1 = \frac{0.56}{\sqrt{1.31 \cdot 1.11}} \approx 0.47 \qquad \hat{r}_2 = \frac{1.17}{\sqrt{1.31 \cdot 1.13}} \approx 0.96 \tag{5.2}$$

In this case, $\hat{r}_2$ is a pretty good match. The frequency is then chosen for this pass. The correlation is used to weight the frequency's influence on the tuner's output frequency to reduce erroneous detections, described later (see section 5.1.4, p. 28).

5.1.3 Pseudo code

Any reference-detection, such as zero-crossings, would work. I found that the greatest peak between zero-crossings produced few and reliable reference points — the emphasis here is on *few*, as each reference results in $O(n)$ operations, that is, $O(|R|n)$ operations per pass.

In the pseudo-code, I have taken the liberty of simplifying for readability which do not alter the running-time. For instance, the pseudo code assumes that there are always valid peaks in P, and no weighting is applied. I have also set an upper limit on $|P|$ in the actual implementation.

```
1  while (true) {
2      input = getNextSample()
3      if (index == 0) {
4          match = 0
5          z = R.pop() // pop first reference
6          for i, r in R {
7              r.nc = r.xy / sqrt(z.yy * r.yy)      // normalised
8              if (r.nc > R[match].nc) match = i    // find match
9          }
10
11         f = sr / (R[match].index - z.index)      // calculate frequency
12
13         // swap reference lists
14         R = P
15         P = []
16     }
17
18     // correlate
19     x = A[(R[0].index + index) % N]
20     for r in R {
21         y = A[(r.index + index) % N]
22         r.yy = y * y     // zero-lag correlation
23         r.xy = x * y     // lag correlation
24     }
25
26     // peak detection
27     if (input > threshold && input > peak.value)
28         peak = Peak(index=index, value=input)
29     else if (peak.index >= 0 && input < 0.0 && last > 0.0) {
30             P.append(peak)
31             peak.index = -1 // invalidate the peak
32         }
33     }
34
35     A[index] = input
36     index = (index + 1) % N
37 }
```

5.1.4 Error correction

Initial tests of the technique in Matlab using a single pass on finite-length signals yielded perfect frequency detection, every time. In the real-time implementation, I found it may erroneously detect frequencies at half of the actual frequency sometimes, even when requiring correlation-values be above a certain threshold q. I suspect the reason is that both correlations will be very close, as both should in theory match very well. This isn't a problem from a usage perspective, as half or double the frequency is merely an octave, and hence still the same tone — unless it is a design requirement to display the octave as well. However, the cent meter would jump sporadically between frequencies far apart, and this is a critical component that most be stabile.

To mend erroneous detections, in addition to averaging frequencies over time, smoothening the transitions, I created a weighting function $w(x)$ (see figure 5.2), based on nothing else than pure experimentation, that simply remaps correlations in $[q; 1]$ to an exponential function clamped to $[0; 1]$, diminishing the effect of less certain correlations. The averaging mechanism gives us a unit of measurement; we will define a confidence variable $c \in [0; 1]$ as the sum of the weights over the length of the weight vector.

The function $w(x)$ is also weighted by the deviation d from the current frequency, making it harder for the tuner to let go of its current conviction.

Let $a = \frac{1}{1-q}$ and $b = a - 1$, then the weighting function is given by

$$w(x, d) = de^{((ax-b)-1)\sqrt{2}} \tag{5.3}$$

The deviance d is calculated from the current ω_0 and the newly detected ω frequency by an interpolation of three functions, which are parameterised by a strength variable s.

Let the frequency ratio be $\Delta_\omega = \frac{|\omega_0 - \omega|}{\max(\omega, \omega_0)}$, then the deviance function (see figure 5.3) is interpolated based on the confidence c, at threshold t, as such

$$d(\Delta_\omega) = \begin{cases} \left(1 - \frac{c}{t}\right) e^{-\Delta_\omega^{(1+s)}} + \frac{c}{t}\left(1 - \Delta_\omega\right) & \text{if } 0 \leq c \leq t \\ \left(1 - \frac{c-t}{1-t}\right)\left(1 - \Delta_\omega\right) + \left(\frac{c-t}{1-t}\right) e^{-(1+2s)\Delta_\omega} & \text{if } t < c \leq 1 \end{cases} \tag{5.4}$$

Note that deviance is asymmetrical as it's a ratio of frequencies. This limits the deviance function, and gives some minimal chance for all frequencies to get through.

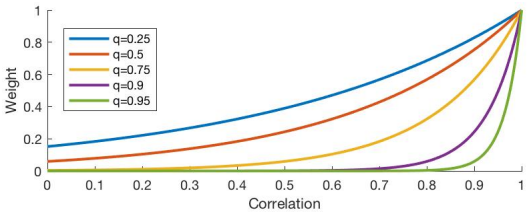


Figure 5.2: Exponential remapping weighting function $w(q, d)$ — assuming $d = 1$

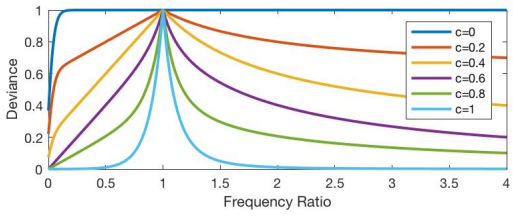


Figure 5.3: Deviance function with $s = 8$ and $t = \frac{1}{2}$ (values used in implementation)

5.1.5 Parameters

The algorithm's frequency range is obviously bounded by length of the buffer A and the sample rate. We must choose $n \geq \lceil \frac{\omega_s}{\omega_m} \rceil$, where ω_m is the lowest frequency we wish to detect. In the case of a guitar, we want to be able to detect frequencies in the range $[82.41; 329.63]$, corresponding to the low E_2 (82.41 Hz) and high E_4 (329.63 Hz) strings, for a standard tuning (E_2 A_2 D_3 G_3 B_3 E_4). However, it's quite common to down-tune, so we might want to leave a bit of room for lower range tunings — I've chosen the interval $[40; 440]$ Hz.

Furthermore, it will only analyse the incoming signal if its dB-level is above a listening threshold of -60dB.

5.2 Filter

The filters employed are biquads, which are quite easy to implement. The implementation utilitizes a class that calculates the coefficients based on enumerations of a filter type as well as the cut-off frequency parameter, and as an added feature it allows for higher order cascade application by specifying the order. As we've seen, biquads are merely a specific case of (3.12), where $M = N = 2$. Let S be the number of stages, and let `B` and `A` be arrays of length L, containing the biquad coefficients of each corresponding stage. The biquad filtering is then implemented according to direct-form I, shown in figure 5.4.

```
for x in samples {
    for s in [0:S-1] {
        b = B[s], a = A[s]
        xs = XS[s], ys = YS[s]

        N = b[0] * x
          + b[1] * xs[0]
          + b[2] * xs[1]
        D = a[1] * ys[0]
          + a[2] * ys[1]
        y = (N - D) / a[0]

        xs[1] = xs[0]; xs[0] = x;
        ys[1] = ys[0]; ys[0] = y;

        x = ys[0];
    }
}
```

While the running-time of the filter is linear, we can further minimise the number of operations. If we normalise the coefficients by a_0 at the time we calculate the coefficients, we can remove a division by a_0 for each sample.

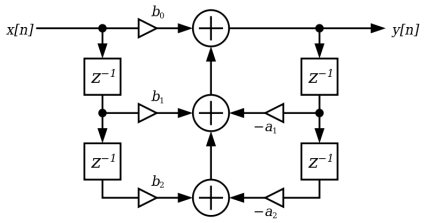


Figure 5.4: Flow graph of direct-form I (source: Wikipedia)

5.2.1 DC Removal

As the tuner relies on microphone input, it is important to ensure that we mend any DC-offset. Low-quality, broken or damaged microphones may have such an offset.

The DC-blocking filter may be implemented in many ways, and I have chosen the simplest; a bilinear filter of a single constant R. I won't derive the formula as this filter is simpler than the biquads, and can even be applied using the same implementation — in that case $b_2 = a_2 = 0$.

The actual implementation is much simpler; the coefficients are $b = \{1, -1\}$ and $a = \{1 - R\}$, and is implemented as follows;

```
y0 = x0 - x1 + y1 * R
x1 = x0;
y1 = y0;
return y0;
```

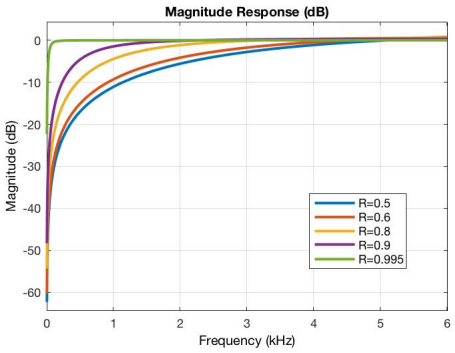


Figure 5.5: DC-Blocking filter magnitude responses ($R = 0,995$ used in implementation)

In our case, since the lower frequency range is quite low, it can be argued that we don't need a full-fledged high-pass biquad filter, and that the DC-filter would suffice. The only considerable noise occurs above the range, while only the DC-offset is of significant concern in this particular case.

The DC-blocker has a zero at 1 and a pole at R. So long as $R \leq 1$ the filter is stabile.

5.3 Interface

The interface of a tuner is quite simple. There are 2 primary indicators; 1) the note registered, if any, and 2) number of cents we're off from the correct tone. In my implementation the note registered is shown at the center, and around it are two concentric circles; the cent meter at the outer rim with an indicator moving around this rim, and a combined level- and tuning indicator. The latter is gray whenever input does not exceed the listening threshold, and the tuner is therefore not listening. If it is listening it is showing how well the current frequency matches the musical tone detected, and is red if the cent meter is at ± 50 and green if the cent meter is at 0 — any value in-between shows an interpolated color thereof.

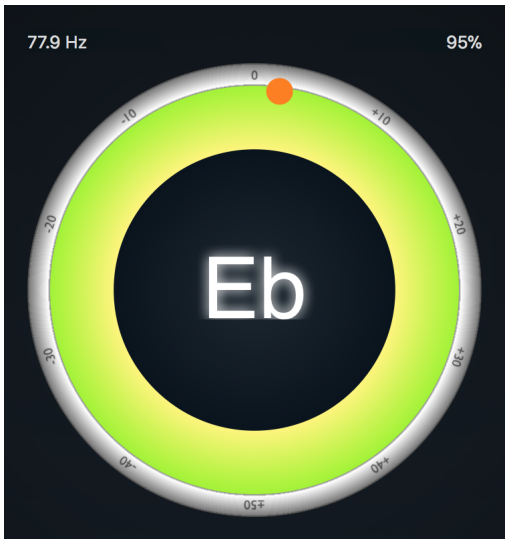


Figure 5.6: Screenshot of tuner (in tune)

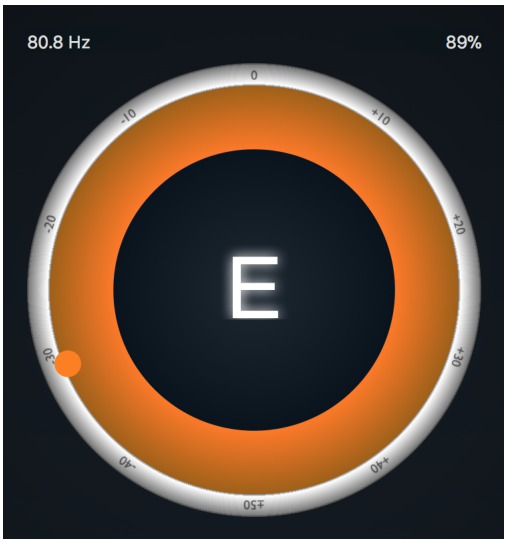


Figure 5.7: Screenshot of tuner (out of tune)

Additionally, the interface I have implemented shows the frequency registered of the current input in the upper left-hand corner, as well as the confidence in the upper right-hand corner.

5.3.1 Demonstration

Since the tuner is platform-dependent and the use of it requires a guitar, I have provided a demonstration of the tuner in the form of a video. The video is available on YouTube through this URL `http://www.casperbhansen.dk/tuner-test/`

Testing

A tuner is by nature a listening device. Testing it is therefore predominantly oriented around external tests. Before doing so, however, I will show that the internal mechanics work as expected by producing input with an expected result internally.

6.1 Internal Tests

6.1.1 Unit Testing

Creating optimal conditions for a tuner to provide tests is difficult using a a microphone or even direct line input, which it is intended to use. Instead, I have written some simple signal generators, whose output can be fed directly to the tuner, and as such provide the conditions necessary for unit testing the tuner.

As a tuner doesn't produce a definite output, but rather a variable state, I have constrained each test with a time limit and a confidence margin the test must fulfil in order to be considered succesful; Each test must reach the sought frequency within 1% error and a confidence in this frequency of at least 50%, and both criteria must be met within 2 seconds.

With pure primitive waveforms, as figure 6.1 shows, the tuner recognises every frequency in the tuners range as expected, and even some outside as well — which is good. The purpose of the filters isn't to restrict its ability to recognise frequencies outside the its range, but rather reduce noise.

In summary, for expected outcomes, it took an average of 0.89 seconds to reach the conditions, with an average percentage of the targeted frequency of 99,8% and an average confidence of 62,9%.

Hz				
20 Hz	0,0%	0,0%	0,0%	0,0%
25 Hz	1,9%	14,8%	0,0%	0,0%
30 Hz	38,6%	46,0%	0,0%	29,0%
40 Hz	61,6%	52,8%	54,1%	52,4%
60 Hz	51,6%	51,6%	51,7%	51,7%
80 Hz	63,9%	51,7%	92,1%	51,6%
120 Hz	51,6%	91,0%	51,7%	51,7%
150 Hz	95,1%	51,7%	51,7%	51,7%
200 Hz	93,7%	92,3%	52,5%	51,7%
250 Hz	81,4%	83,7%	51,7%	85,9%
320 Hz	51,7%	91,1%	51,7%	51,4%
380 Hz	51,7%	81,6%	77,9%	51,7%
400 Hz	74,9%	51,7%	51,7%	91,1%
440 Hz	52,4%	51,7%	75,2%	69,1%
460 Hz	67,4%	52,3%	51,7%	72,7%
480 Hz	70,7%	72,0%	68,8%	66,3%
Avg. Conf.	66,32%	68,24%	60,16%	59,99%
Avg. Pct.	99,71%	99,86%	99,81%	99,81%
Avg. Time	0,87 sec.	0,92 sec.	0,90 sec.	0,89 sec.

Figure 6.1: Primitive unit testing results (Unit: confidence, data source: `tests.cpp`)

6.1.2 Filtering

Finally, in order to show that the filters do in fact alter signals as expected, I have recorded some audio and applied the biquad filters of different orders, and taken the FFT of the resulting signal.

The implementation does not make use of the high-pass filter, since the DC filter was favorable to it, as pointed out. In figure 6.2 we see the DC filter flattening the magnitude response at the very lowest frequencies, as expected. The figure further shows that the low-pass filter does indeed dampen the frequencies above the cut-off frequency.

A 2nd order biquad low-pass was sufficient to produce good results, but in the implementation, by pure experimentation, I ended up using an 8th order cascaded biquad, since it produced more satisfactory results. Any order above this did not produce any significant change in the results.

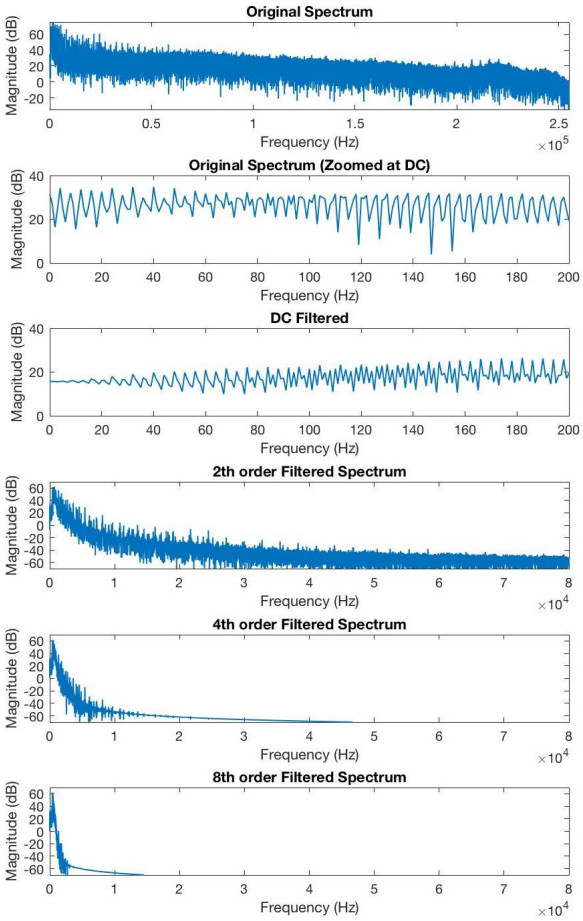


Figure 6.2: FFTs of DC- and biquad filtered signals with cut-off at 440 Hz

6.2 External Tests

As the system is a reference system, testing it requires definite knowledge of the principal frequency of the input to provide a baseline measurement. I employed a tone generator as reference source for testing on real input with a known frequency. Once a baseline has been established, it is critical to test the tuner with the input it is intended for, a guitar, and in various environments that may affect the tuners ability to give a stabile reading.

6.2.1 Experimental Set-up

All baseline measurements were conducted in a vocal booth (see figure 6.5, p. 33) in my own home studio. The room is built with sound proofing materials which minimises outside noise, and has been treated acoustically to minimise resonance. The measurements were recorded when the tuner reached a resting state, with minimal fluctuation. The experiment was done using a tone generator (*Tone Gen Pro*, for iOS) producing the primitive waveforms; sine, triangle, square and saw. Once established that the tuner works in a controlled environment, I will move the system out of the confines of a low-noise environment and try to break it by imposing noise on the source of the signal.

6.2.2 Baseline Measurements

Here, I give the results of the measurements taken in the controlled acoustic environment.

In order to fulfill the specification satisfactory, the tuner is required to 1) respond with minimal error using the tone generator within the frequency range $[40; 440]$ Hz, 2) achieve a stabile resting state for all open guitar tones in a standard tuning within less than 1 second.

Hz			
20 Hz	N/A	N/A	N/A
25 Hz	25,0 (32%)	25,0 (48%)	25,0 (30%)
30 Hz	30,0 (89%)	30,0 (58%)	30,0 (62%)
40 Hz	40,0 (98%)	40,0 (91%)	40,0 (92%)
80 Hz	80,0 (97%)	80,1 (95%)	80,0 (99%)
110 Hz	110,2 (76%)	110.0 (99%)	110,0 (98%)
220 Hz	220,5 (70%)	220.5 (80%)	220,5 (74%)
440 Hz	441,0 (75%)	441,0 (85%)	441,0 (85%)

Figure 6.3: Measurements of hertz and confidence in studio (source: tone generator)

Tone	Expected	Result
E_2	82,41 Hz	82,4 (99%)
A_2	110,0 Hz	110,0 (99%)
D_3	146,83 Hz	147,1 (95%)
G_3	196,0 Hz	196,0 (96%)
B_3	246,94 Hz	246,8 (88%)
E_4	329,63 Hz	330,2 (82%)

Figure 6.4: Measurements of hertz and confidence in studio (source: pre-tuned guitar)

The program log showed that a satisfactory conditions were produced within 1–18 passes. At a sample-rate of 44.100 Hz, and a pass being 2048 samples, this is 46–836 ms, which is satisfactory. The tuner began to fail at around 800-1000 Hz and above — again, well within the requirement. It did not recognise any sine wave, which i did suspect, since sine waves are low in power.

In summary, I find the results satisfactory, since all requirements are met. More over, I did not expect such high confidence readings for the plugged guitar strings at an average of 93,2% — these far exceeded my expectations.

Figure 6.5: Physical set-up; Shure SM7-B microphone in vocalbooth

6.2.3 Noise

In these tests, I will try to break the system. The experiment will be done with the filters on and off, and repeated in two different environments that may impact the clarity of the received signal. I place the tuner along with a controlled noise source and turn up the noise level until the system no longer produces a reliable reading. The noise source is not musical in nature, but one that is rich in all frequencies; I used a recording of a rain and thunder storm. The goal of this test is to show that the filters make a viable difference, and each frequency is therefore tested with and without the filters on. Particularly, I expect the low-pass filter to improve the measurements at high noise levels.

Before the test I took a measurement of the dB SPL (sound pressure level) in the room with no articifial noise. The test will record the average confidence readings of 6 measurements from the frequencies in standard tuning. The noise level is then increased until the confidence falls below 50% or if impossible to take a reading because of fluctuations.

Noise (dB SPL)	w/ Filter	w/o Filter
35	87,1%	79,7%
40	85,7%	70,5%
50	84,3%	67,5%
55	79,6%	62,5%
60	60,5%	44,3%
62	49,8%	N/A

Figure 6.6: Confidence measurements in neutral environment (living room)

Noise (dB SPL)	w/ Filter	w/o Filter
35	82,2%	82,6%
40	87,2%	69,7%
50	66,2%	64,8%
56	69,7%	53,0%
60	55,2%	42,0%
62	51,9%	N/A

Figure 6.7: Confidence measurements in a hard-surface environment (bathroom)

In both instances, the tuner fails to produce the required result at around 60dB SPL, without the filters on. With the filter on, the tuner clearly produces a higher confidence. While this is satisfactory for the test, since it shows an improvement by employing the filters, personally I am not satisfied with the results. Had no other test been carried out at this point, I would have tampered with the filter and weighting parameters until I was satisfied with the results.

6.2.4 Chord

In this test, although not required by the tuner, I will play simple triad chords and record how the tuner responds to such an input. While I can make no arguable prediction as to the result, as the tuner was not designed to recognise chords, I suspect that either the root tone will dominate the signal, or jump between the constituent tones that make up the chord.

Chord/Root	3rd.	5th.	Result
A♭m	B	Eb	A♭
B	E♭	F♯	B
C♯	F	A♭	C♯
E	G♯	B	E
E♭	G	B♭	E♭
F♯	B♭	C♯	F♯

Figure 6.8: Chord input measurements

The experiment was repeated across different octaves, and the results stayed the same. Although, the results tended toward the lowest octave in the range of the tuner of the tone, which makes sense, in retrospect, since harmonious tones (or chords) are periodic by definition.

While not a requirement or even designed to do so, the tuner does in fact seem to recognise the root of a chord.

Conclusion

From the theoretical foundations of Fourier Analysis, developed by Jean-Baptiste Joseph Fourier (1768–1830), the thesis has presented a multitude of techniques in applying the theory in practice under the constraints of real-time processing.

We have seen that ideal filters are theoretical mathematical constructs that can never be realised in practice. We can, however, get very good approximations using the cascaded filtering method. The equivalent filter order thereof must, however, stay small enough in order to remain faster than the Fast Fourier Transform.

I remark that while the Fast Fourier Transform is a highly tuned and efficient algorithm, in a real-time context, despite efforts to adapt it thereto, it is highly contested by infinite impulse response convolution. Similarly, finite impulse response convolution is only applicable if we desire a linear phase, or if the intent is not mere frequency manipulation. In conclusion I find that Fast Fourier transform methods —Short-Time Fourier Transform and Sliding Discrete Fourier Transform— are valuable tools for real-time spectral analysis. However, when we want to manipulate the spectral content of real-time signals, in most cases we would choose a time-domain based filter. As shown, however, there are trade-offs for each method — ie. delay, latency, efficiency, quality, etc. Therefore choosing a technique should reflect and conform to the requirements of the problem at hand.

In testing the tuner I found room for many improvements. An algorithm that solves this problem can be implemented in many ways, and my effort to design such an algorithm was based on simple assumptions about the incoming signal and used the theory of convolution and correlation, presented in theory of the thesis. The main objective of it was to provide a viable example, showing that the filters do remove noise and have a positive influence on the system's ability to recognise the principal frequency. I find this to be the case.

Definitions

A.1 Linear Time Invariant (LTI) systems

In order for a system to be linear it must exhibit the additive and homogeneous properties, as defined below.

Lemma A.1.1 (Additivity)
Let $x_k[n] \rightarrow y_k[n]$ be a system. If

$$x_1[n] + x_2[n] \rightarrow y_1[n] + y_2[n] \tag{A.1}$$

then the system exhibits the additive property.

Lemma A.1.2 (Homogeity)
Let $x[n] \rightarrow y[n]$ be a system and let α be any constant. If

$$\alpha x[n] \rightarrow \alpha y[n] \tag{A.2}$$

the system exhibits the homogeneous property.

Linearity then follows from systems that exhibit both of the aforementioned properties.

Theorem A.1.1 (Linearity) *If a system $H(x)$ exhibits both the additivity and homogeneous property, such that*

$$\alpha x_1[n] + \beta x_2[n] \rightarrow \alpha y_1[n] + \beta y_2[n] \tag{A.3}$$

then the system is linear.

Theorem A.1.2 (Time invariance) *Let $H(x)$ be a system, such that $y[n] = H(x[n])$ and let t_0 be a constant. If*

$$x[t - t_0] \rightarrow y[t - t_0] \tag{A.4}$$

then the system is time invariant.

A.2 Special functions

A.2.1 Dirac delta function

Definition A.2.1 (Dirac Delta Function) *The δ function is a special function, which has the following properties*

$$\delta(t) = \begin{cases} \infty & t = 0 \\ 0 & t \neq 0 \end{cases} \qquad \textit{and} \qquad \int_{-\infty}^{\infty} \delta(t)dt = 1 \tag{A.5}$$

in continuous time. And its discrete equivalent is given by

$$\delta[t] = \begin{cases} \infty & n = 0 \\ 0 & n \neq 0 \end{cases} \qquad \textit{and} \qquad \sum_{k=-\infty}^{\infty} \delta[n] = 1 \tag{A.6}$$

A.2.2 Step function

Definition A.2.2 (Step Function) *The step function is a special function, which has the following properties*

$$u(t) = \begin{cases} 1 & t \geq 0 \\ 0 & t < 0 \end{cases} \qquad \textit{and} \qquad u[n] = \begin{cases} 1 & n \geq 0 \\ 0 & n < 0 \end{cases} \tag{A.7}$$

A.2.3 sinc function

Definition A.2.3 (sinc function)

$$\mathrm{sinc}\, x = \frac{\sin x}{x} \tag{A.8}$$

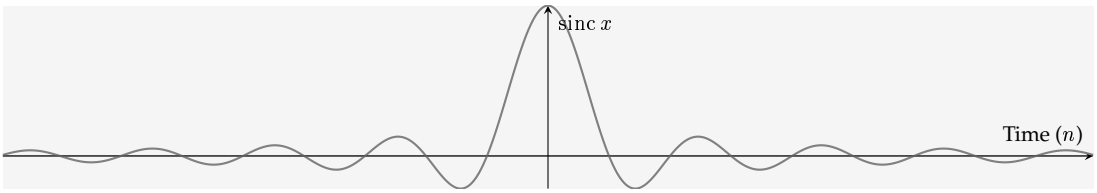


Figure A.1: Sinc function

Bibliography

[1] J. G. Proakis & D. G. Manolakis, *Digital Signal Processing*, Pearson Education, 4th. Edition, 2007

[2] J. H. McClellan, R. W. Schafer & M. A. Yoder, *Signal Processing First*,

[3] Steve Winder, *Analog and Digital Filter Design*, 2nd Edition, Newnes Press, 1997

[4] R. Bradford, R. Dobson & J. Ffitch, *Sliding is Smoother than Jumping*, Department of Computer Science, University of Bath England

[5] Robert Bristow-Johnson, *Cookbook formulae for audio EQ biquad filter coefficients*, http://www.musicdsp.org/files/Audio-EQ-Cookbook.txt

[6] C. Berg & J. P. Solovej, *Noter til Analyse 1: Fourierrækker og Metriske rum*, Department of Mathematics, University of Copenhagen, 1st. Edition, 2011