On the separation of sounds of musical instruments

Greevenbosch, B.

On the separation of sounds of musical instruments

A thesis submitted in partial fulfilment of the requirements for the degree of Master of Science at the University of Leiden by

Bert Greevenbosch

Mathematical Institute
Leiden, The Netherlands

Contents

Introduction

When listening to music, we are able to concentrate on one particular instrument. Somewhere between the perceiving of a sound by the ear and our experience of the sound a selection is made, which component of the sound is interesting, and which is not. Until now, it has not been achieved to model this phenomenon in a computer in a satisfactory way. In this thesis we approach this problem by studying a simpler version of it: we create a filter that filters one particular instrument, a clarinet, from a piece of music.

An instrument filter can be useful for studio purposes. For example, with an instrument filter it will be possible to change the volume of one of the instruments, or replace it by another. Also, it can be convenient if one can remove an erroneous note during the mastering process.

The research of natural sounds has a big resemblance to speech processing. In fact, speech is a natural sound. Many of the techniques used for natural sound processing are therefore taken from the field of speech processing. The field of instrument recognition has been studied by several researchers, see for example [1], [6], [16]. The recognition is done by measuring different characteristics from an already isolated sound, and applying some artificial intelligence technique for the recognition. These researches mainly focus on the possibility to categorise the instruments so that they can be found from a database of recordings.

The separation of natural sound sources has also been studied, see for example [29], [30]. These studies concentrate on the separation rather than the recognition: they split the sound into components, but the components are not classified. The recordings used to test the algorithms where mixes of two or three recordings of single sounds, and to test the algorithm the results where compared to the original sounds.

An approach to build a filter that filters one instrument from a recording could be the

combination of these two main areas: first split the sound into components and then classify the components to determine to which category they belong. The filtered signals can then be synthesised by adding the different components.

In this thesis we describe the results of our research in the period March 2004 - March 2005. We have managed to split a mixture of a clarinet tone and a viola tone by filtering the harmonics. We have built a phase vocoder and a tracking phase vocoder, which enable us to time-stretch signals and to decompose a signal into spectral lines. With these spectral lines the clarinet part can be separated from various pieces of music. A great deal of the research has been put in developing and studying methods to measure spectral lines.

This master thesis consists of four sections. A mathematical basis for methods that have been used is given in Chapter 1. In Chapter 2 we describe our experiments and their results; a method to separate the most pronounced instrument from a recording of music is developed. Final conclusions and topics for further research are given in Chapter 3. The MATLAB listings can be found in Chapter 4.

Audible examples are indicated by the term *Fragment*. They are available at the following internet address: `http://www.bertgreevenbosch.nl/msc`.

Chapter 1
A mathematical way to look at sound

1.1 The representation of sound in a computer

Sound is vibration in air pressure. It can be seen as a traveling longitudinal wave: along the direction in which the wave travels through the air, there are places where the air pressure is more than average, and places where it is less than average. In time, the difference between the air pressure of a certain point and the average air pressure changes, and if we make a plot of this difference against time we will see a periodic signal. The shape of this signal determines our experience of the sound.

1.1.1 Sampling

Let $x(t)$ be the function which returns the difference between the air pressure at a certain point and the average air pressure at time t. In physical reality, the function x takes real input values, and returns real values as well. Since we will analyse the sound in the digital domain, we need to discretise the signal. This process is called *sampling*. First we need some discrete version of $\mathbb{R}$, which we shall denote by $\mathcal{R}$. For example, if we normalise the maximum amplitude that our computer can handle to one, and if we want $\mathcal{R}$ to consist of M elements, we can choose $\mathcal{R} = \left\{-\frac{M-1}{M}, -\frac{M-3}{M}, ..., \frac{M-3}{M}, \frac{M-1}{M}\right\}$. Then we sample x by creating a vector $\mathbf{x} = (x[0], x[1], ..., x[N-1]) \in \mathcal{R}^N$, where $x[n]$ contains the value in $\mathcal{R}$ closest to $x(nT_s)$, with the time interval between two samples, $T_s > 0$, constant. The

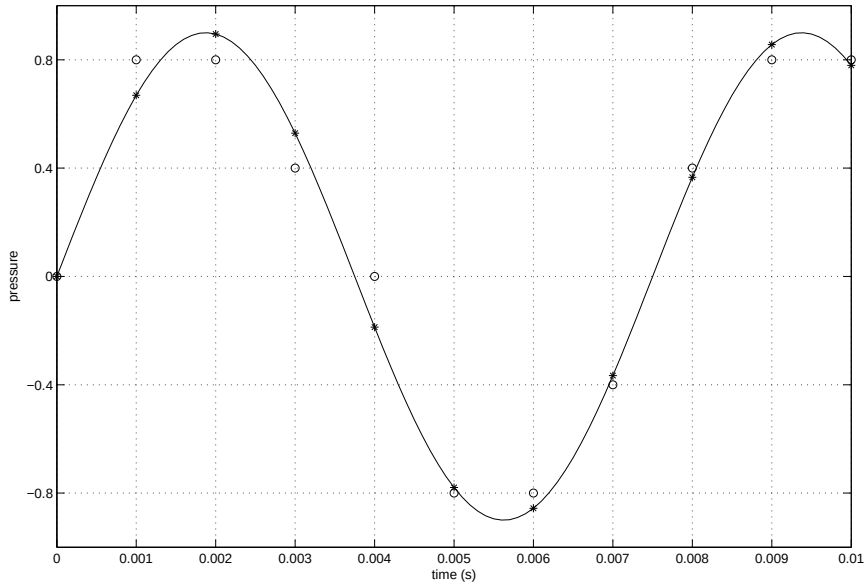


Figure 1.1.1: Sampling a sine (at a very low quality). The frequency of the sine is 133 Hz, and we sample it with a sampling rate of 1000 Hz. We have chosen $\mathcal{R} = \{-0.8, -0.4, 0.0, 0.4, 0.8\}$. The time points at which the sine is sampled are indicated by the vertical lines. The values of the sine at these time points are indicated by the stars in the figure. During the sampling process these values are quantised to the nearest values in $\mathcal{R}$, which are indicated by the circles.

reciprocal of T_s is called the *sample rate* and we shall denote it by ϕ_s.

Both the number of elements in $\mathcal{R}$ and the sample rate are factors in the quality of the digital sound. For example, a compact disk (CD) has a sample rate of 44.1 kHz, and uses 16-bit sampling, which means that $\mathcal{R}$ has $2^{16} = 65536$ elements.

The transition of elements in $\mathbb{R}$ to $\mathcal{R}$ is called *quantising*. Although this transition has some effect on the data, in this thesis we will usually consider $\mathbb{R}$ or, if it is mathematically more convenient, $\mathbb{C}$ instead. When we have enough bits and quantise wisely this transition is not or scarcely audible. Figure 1.1.1 illustrates sampling and quantising. For the clearness of the picture we have chosen a very low sample rate and an $\mathcal{R}$ with an extremely small number of elements.

In the next section we will see the influence of the sample rate on the quality of the sound. To do this, we first need to introduce the concept of a *phasor*.

1.1.2 Phasors and frequency units

Consider a continuous signal $x(t) = \cos \omega t$ (where t is measured in seconds). We can regard this signal as a projection of the complex signal $y(t) = \mathrm{e}^{\mathbf{i}\omega t}$ on the real axis. The function $y(t)$ is called a *phasor* with a frequency of $\frac{\omega}{2\pi}$ Hertz.

In the continuous case time is usually measured in seconds, and frequency in Hertz (Hz). In the discrete case we prefer to measure time in samples, and frequency in radians per sample. For example, a discrete signal $x[n] = \mathrm{e}^{\mathbf{i}\omega n}$ needs $\frac{2\pi}{\omega}$ samples to do one cycle. If we consider a phasor as a point moving along the unit circle, that means that in $\frac{2\pi}{\omega}$ samples the point has moved 2π radians along the unit circle. This implies that the point moves with a frequency of $2\pi / \left(\frac{2\pi}{\omega}\right) = \omega$ radians per sample.

1.1.3 Aliasing and the Nyquist frequency

Consider the continuous-time phasor $x(t) = \mathrm{e}^{\mathbf{i}2\pi t\phi}$, which has frequency ϕ. When we sample it at a sample rate of ϕ_s, we retrieve the discretised signal $\mathbf{x} \in \mathbb{C}^N$ where

$$x[n] = \mathrm{e}^{\mathbf{i}2\pi n T_s \phi}.$$

Since $x[n] = \mathrm{e}^{\mathbf{i}2\pi n T_s(\phi + k\phi_s)}$ for arbitrary $k \in \mathbb{Z}$, we obtain a sampled version of a phasor with frequency $\phi + k\phi_s$. Thus the signal $y_k(t) = \mathrm{e}^{\mathbf{i}2\pi(\phi + k\phi_s)t}$ and $x(t)$ give the same result when sampled. They are called *aliases* of one another.

Converting a discrete time signal back to a continuous time signal is generally done by a simple interpolation technique, in our case either by the sound driver or hardware of a computer. This means that if a computer plays the sampled version of any $y_k(t)$, it will play $y_l(t)$ with l such that $\phi + l\phi_s =: \omega$ is the nearest to zero. This implies that $-\frac{1}{2}\phi_s \leq \omega \leq \frac{1}{2}\phi_s$. The frequency $\frac{1}{2}\phi_s$ or π rad/sample is called the *Nyquist frequency*; it is the highest frequency that a sampled signal can contain.

In Figure 1.1.2 we see an example: the original cosine has a frequency of 3 Hz. It can be seen as the sum of two phasors: $y(t) = \frac{1}{2}\left\{\mathrm{e}^{-\mathbf{i}6\pi t} + \mathrm{e}^{\mathbf{i}6\pi t}\right\}$. When we sample this signal at a rate of 4 Hz, we see that these phasors have alias frequencies of respectively $\mathrm{e}^{\mathbf{i}2\pi t}$ and $-\mathrm{e}^{\mathbf{i}2\pi t}$, which result in the reconstructed signal $\cos(2\pi t)$ of frequency 1 Hz. Notice that this is the same as $\cos(-2\pi t)$ and therefore both cosines are also aliases of one another

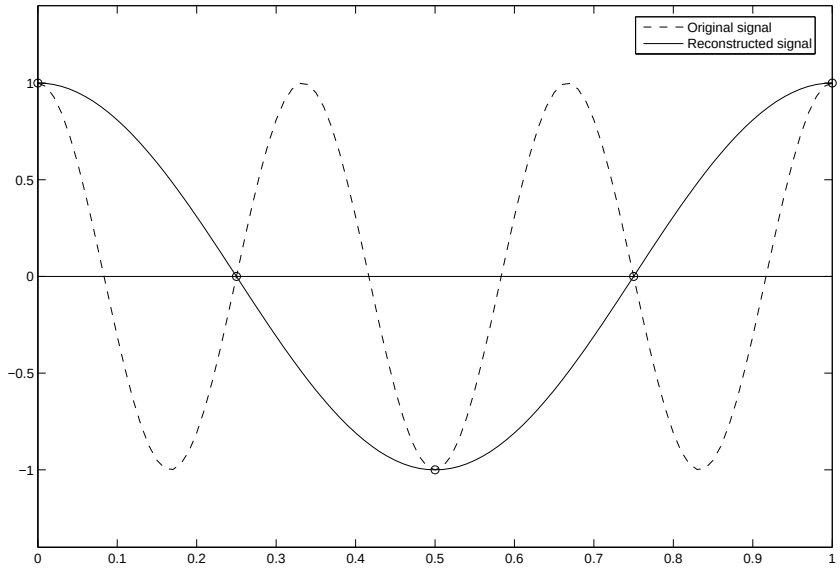


Figure 1.1.2: The aliasing of a cosine function. The original signal $x(t) = \cos 6\pi t$ (broken line) is sampled at the time points $t = 0, 0.25, 0.5, 0.75, 1$ (circles), and reconstructed to its alias $y(t) = \cos 2\pi t$ (unbroken line).

(although their frequencies differ by only 2 Hz).

As another example we look at a CD. The range of frequencies that the human ear can perceive is about 20 Hz to 22 kHz. This means that a CD only needs to be able to record frequencies up to 22 kHz, thus $\frac{1}{2}\phi_s = 22$ kHz, which means that a sample rate of 44 kHz should be enough. To be sure a little is added and we see why a CD has a sample rate of 44.1 kHz.

1.1.4 The instantaneous frequency

A natural sound is often the result of an event that can be described by some version of a wave equation. Therefore the resulting signal can be expressed as a summation of sinusoids with certain frequencies. These sinusoids are called the *partials* of the signal. We will use the following model for a sound $x(t)$:

$$x(t) = \sum_{k=1}^{K} a_k(t) \cos \phi_k(t)$$

where locally $a_k(t)$ and $\phi_k'(t)$ vary slowly. This implies that

$$x(t) \simeq \sum_{k=1}^{K} a_k(t_0) \cos \left(\phi_k(t_0) + \phi_k'(t_0)(t - t_0) \right)$$

locally. The derivative $\phi_k'(t)$ is called the *instantaneous frequency* of the *partial* $a_k(t)\cos(\phi_k(t))$ and $\phi_k(t_0)$ its *phase* at t_0.

1.2 Fourier analysis

Let us consider a sampled signal $\mathbf{x} \in \mathbb{C}^N$. There are two classical ways to analyse its frequency content: the *Discrete Fourier Transform* and the *Discrete-Time Fourier Transform*, which is also called the *z-transform*.

1.2.1 Discrete Fourier Transform

The Discrete Fourier Transform (DFT) is used to transform a finite time discrete signal $\mathbf{x} = (x[0], x[1], ..., x[N-1]) \in \mathbb{C}^N$ from the time domain into the discrete frequency domain. In $\mathbb{C}^N$ we will use the standard Euclidian inner product

$$< \mathbf{x}, \mathbf{y} >= \mathbf{x}\mathbf{y}^* = \sum_{n=0}^{N-1} x[n]\overline{y[n]},$$

where $\mathbf{x} = (x[0], x[1], ..., x[N-1])$ and $\mathbf{y} = (y[0], y[1], ..., y[N-1])$.

Lemma 1.2.1 *Let* $\mathbf{e_k} = \left(1, \mathrm{e}^{\mathbf{i}2\pi 1k/N}, \mathrm{e}^{\mathbf{i}2\pi 2k/N}, ..., \mathrm{e}^{\mathbf{i}2\pi nk/N}, ..., \mathrm{e}^{\mathbf{i}2\pi(N-1)k/N}\right)$. *The system* $\mathcal{S}_N = \{\mathbf{e_k}\}_{k=0}^{N-1}$ *forms an orthogonal basis of* $\mathbb{C}^N$.

Proof

Choose $p, q \in \{0, 1, ..., N-1\}$. If $p \neq q$ we find:

$$< \mathbf{e_p}, \mathbf{e_q} >= \sum_{n=0}^{N-1} \mathrm{e}^{\mathbf{i}2\pi np/N} \mathrm{e}^{-\mathbf{i}2\pi nq/N} = \sum_{n=0}^{N-1} \mathrm{e}^{\mathbf{i}2\pi n(p-q)/N} = \frac{\mathrm{e}^{\mathbf{i}2\pi N(p-q)/N} - 1}{\mathrm{e}^{\mathbf{i}2\pi(p-q)} - 1} = 0.$$

For $p = q$ we get

$$< \mathbf{e_p}, \mathbf{e_q} >= N.$$

By a straightforward argument the N orthogonal vectors constitute a basis for $\mathbb{C}^N$. $\square$

The orthogonality of $\mathcal{S}_N$ implies that

$$\mathbf{x} = \frac{1}{N} \sum_{k=0}^{N-1} < \mathbf{x}, \mathbf{e_k} > \mathbf{e_k}, \tag{1.2.1}$$

and the vector $\mathbf{x}^{\#} = \left(x^{\#}[0], x^{\#}[1], ..., x^{\#}[N-1]\right)$, where

$$x^{\#}[k] =< \mathbf{x}, \mathbf{e_k} >, \tag{1.2.2}$$

is called the *Discrete Fourier Transform* of $\mathbf{x}$. Formula (1.2.1) is called the *inverse DFT*. Notice that if the vector $\mathbf{x}$ consists of real entries only, then $x^{\#}[k] = \overline{x^{\#}[N-k]}$ for all $k \in \{1, 2, ..., N-1\}$. This implies that if we analyse such $\mathbf{x}$ in the frequency domain, it is enough to look at $\left\{x^{\#}[0], x^{\#}[1], ..., x^{\#}[\lfloor \frac{N}{2} \rfloor]\right\}$ only.

1.2.2 Fast Fourier transform

For the processing of sound which we will encounter in the sequel, it is vital to be able to compute Fourier transforms as fast as possible. The Fast Fourier Transform is a classical way to enhance computation speed considerably. The Fast Fourier Transform was first introduced in 1965 in [2].

Let us have a closer look at Formula (1.2.2). If $N > 1$ is not a prime number then there exist a positive integer $1 < p < N$ such that $p|N$. We rewrite (1.2.2) as follows:

$$x^{\#}[k] = \sum_{n=0}^{N-1} x[n] e^{-\mathbf{i}kn2\pi/N} \tag{1.2.3}$$

$$= \sum_{m=0}^{\frac{N}{p}-1} \sum_{l=0}^{p-1} x[mp+l] \mathrm{e}^{-\mathbf{i}k(mp+l)2\pi/N} = \sum_{l=0}^{p-1} \mathrm{e}^{-\mathbf{i}kl2\pi/N} \sum_{m=0}^{\frac{N}{p}-1} x[mp+l] \mathrm{e}^{-\mathbf{i}km2\pi/\frac{N}{p}}$$

$$= \sum_{l=0}^{p-1} \mathrm{e}^{-\mathbf{i}kl2\pi/N} x_{(l)}^{\#}[k] \tag{1.2.4}$$

where $\mathbf{x}_{(l)}^{\#}$ is the DFT of $\mathbf{x}_{(l)} = \left(x_{(l)}[m]\right)_{m=0}^{\frac{N}{p}-1}$ with $x_{(l)}[m] = x[mp+l]$.

The Identity (1.2.4) is the key to a recursive algorithm that computes $\mathbf{x}^{\#}$ faster than by directly evaluating (1.2.3). To illustrate this, let us consider what happens when $N = p^{\nu}$ where $\nu, p \in \mathbb{N}$, $p > 1$.

If we calculate $\mathbf{x}^{\#}$ using (1.2.3), we need $N(N-1)$ additions and N^2 multiplications.

If we use (1.2.4), we see that for the calculation of the DFT of a vector of size N, we need to calculate the DFT of p vectors of size $\frac{N}{p}$, and do an additional $N(p-1)$ additions and Np multiplications. When the vector has only one element it is equal to its DFT, which means that no additions or multiplications are needed. The number $A(N)$ of additions needed to calculate the DFT for a vector of size N equals

$$A(N) = \begin{cases} pA\left(\frac{N}{p}\right) + N(p-1) & \text{if} \quad N > 1 \\ 0 & \text{if} \quad N = 1 \end{cases} \tag{1.2.5}$$

and the number of multiplications $M(N)$ equals

$$M(N) = \begin{cases} pM\left(\frac{N}{p}\right) + Np & \text{if} \quad N > 1 \\ 0 & \text{if} \quad N = 1. \end{cases}$$

We can solve (1.2.5) by dividing it on both sides by N, and retrieve

$$A(N) = N(p-1)\log_p N.$$

Analogously we find $M(N) = Np\log_p N$. We see that our algorithm is of order $N\log N$, whilst direct evaluation is of order N^2. This significant improvement has become widely

used, and is commonly known as the *Fast Fourier Transform (FFT).*

When we implement the FFT on a computer, we see that the DFT of $\mathbf{x}_{(l)}$ has only $\frac{N}{p}$ elements, whilst we need N elements. This can be solved easily by observing that the DFT transform of a signal with length $\frac{N}{p}$ is periodic with period $\frac{N}{p}$, and therefore $x^{\#}_{(l)}[k] = x^{\#}_{(l)}[k - \nu\frac{N}{p}]$ with $\nu \in \mathbb{Z}$ chosen wisely.

1.2.3 The Discrete-Time Fourier Transform and the Continuous-Time Fourier Transform

The DFT is a powerful tool since it can be calculated swiftly using the FFT. However it has the disadvantage that it can only properly detect the frequencies $\left\{\frac{2\pi k}{N}\right\}_{k=0}^{N-1}$ whilst for all other frequencies *spectral smearing* occurs: a number of elements $x^{\#}[k]$ around the real frequency is assigned to a nonzero value. We can reduce the effect of spectral smearing by increasing N, but the increase of N also decreases the time localisation of a frequency. We will have a closer look at this phenomenon in Section 1.3.

A tool to analyse the frequency content of a sampled signal $\mathbf{x}$ very similar to the Discrete Fourier Transform (DFT) is the *Discrete-Time Fourier Transform (DTFT)*:

$$x^{\#}(\omega) = \sum_{n=-\infty}^{\infty} x[n]\mathrm{e}^{-\mathbf{i}\omega n},$$

which holds (for instance) for an absolute summable sequence $\mathbf{x} = (x[n])_{n=-\infty}^{\infty}$. Notice that $x^{\#}(\omega)$ is periodic with period 2π, which agrees with the aliasing phenomenon.

The vector elements $x[n]$ can be retrieved from $x^{\#}$ through the integral

$$x[n] = \frac{1}{2\pi}\int_{-\pi}^{\pi} x^{\#}(\omega)\mathrm{e}^{\mathbf{i}\omega n}\, d\omega,$$

which is called the *inverse formula* of the DTFT.

Notice that the DFT of a vector $\mathbf{x} = (x[0], x[1], ..., x[N-1])$ can be seen as a DTFT of a vector $\mathbf{y} = (y[n])_{n=-\infty}^{\infty}$, where $y[n] = x[n]$ for $0 \leq n < N$ and $y[n] = 0$ otherwise, evaluated at the frequencies $\omega = 0, \frac{2\pi}{N}, 2\frac{2\pi}{N}, ..., (N-1)\frac{2\pi}{N}$, thus $x^{\#}[k] = y^{\#}\left(k\frac{2\pi}{N}\right)$.

Finally, if we want to analyse the frequency content of an analogous signal $x(t)$, we can

use the *(continuous time) Fourier Transform*, which is defined as

$$\widehat{x}(\omega) = \int_{-\infty}^{\infty} x(t)\mathrm{e}^{-\mathbf{i}\omega t}\, dt.$$

It is used to measure how much oscillations of the frequency ω there are in x. The following theorem is classical, and tells us how to restore a function from its frequency content.

Theorem 1.2.1 *Let $x \in \mathbf{L}^1(\mathbb{R})$ and $\widehat{x} \in \mathbf{L}^1(\mathbb{R})$. Then*

$$x(t) = \frac{1}{2\pi}\int_{-\infty}^{\infty} \widehat{x}(\omega)\mathrm{e}^{\mathbf{i}\omega t}\, d\omega. \tag{1.2.6}$$

for almost every t.

Formula (1.2.6) is called the *inverse Fourier Transform*.
There exist many good introductions to the theory of Fourier Transformations, such as [20], [27] and [22] (Dutch).

1.3 Windowing

In general it is impossible to locate both the frequency content of a sound and the time position where the oscillations occur with arbitrary precision. There always has to be made a trade-off between time and frequency precision. This property of a periodic signal is commonly known as the *Heisenberg uncertainty principle* (see Section 1.5). When analysing a signal through cutting it in small parts, we should always bear this principle in mind. We cut the signal into parts by the use of windows: a *window* is a real function with finite support. Instead of considering the whole signal we consider only a part of it by multiplying the signal with a window. In Section 1.3.1 we will illustrate the problems by an example, and in the subsequent sections we shall have a closer look at the theory of *windowing*.

1.3.1 Frequency and time localisation using the DFT

We have seen that the DFT can be used to determine the frequency content of a sound. As an example we will have a look at what happens when we apply the DFT of size N to

the discrete signal

$$x[n] = \begin{cases} \mathrm{e}^{\mathbf{i}n\phi 2\pi/N} & \text{if } a \leq n < b \\ 0 & \text{otherwise,} \end{cases}$$

where a, b are positive integers with $a+2 \leq b \leq N$ and ϕ is a positive real number strictly smaller than N. Notice that for every frequency below the Nyquist frequency we can find a ϕ in the interval $[0, N)$ that yields a phasor of that frequency. When we apply the DFT to $\mathbf{x}$ we find for $k \neq \phi$:

$$\begin{aligned} x^{\#}[k] &= <\mathbf{x}, \mathbf{e_k}> = \sum_{n=a}^{b-1} \mathrm{e}^{-\mathbf{i}n(k-\phi)2\pi/N} \\ &= \frac{1}{\mathrm{e}^{\frac{1}{2}\mathbf{i}(k-\phi)2\pi/N} - \mathrm{e}^{-\frac{1}{2}\mathbf{i}(k-\phi)2\pi/N}} \sum_{n=a}^{b-1} \left\{ \mathrm{e}^{-\mathbf{i}(n-\frac{1}{2})(k-\phi)2\pi/N} - \mathrm{e}^{-\mathbf{i}(n+\frac{1}{2})(k-\phi)2\pi/N} \right\} \\ &= \frac{1}{2\mathbf{i}\sin((k-\phi)\pi/N)} \left\{ \mathrm{e}^{-\mathbf{i}(a-\frac{1}{2})(k-\phi)2\pi/N} - \mathrm{e}^{-\mathbf{i}(b-\frac{1}{2})(k-\phi)2\pi/N} \right\}, \end{aligned} \tag{1.3.1}$$

and if $k = \phi$ we obtain $x^{\#}[k] = b - a$. Taking absolute values yields:

$$|x^{\#}[k]| = \left| \frac{\sin((b-a)(k-\phi)\pi/N)}{\sin((k-\phi)\pi/N)} \right|$$

for $k \neq \phi$ and $x^{\#}[k] = b - a$ otherwise.

This absolute value depends only on the difference $b-a$, and not on the particular values of a and b. This means that if we look at the absolute values of the DFT we cannot determine where in the part of the signal that is considered the phasor with frequency $\phi\frac{2\pi}{N}$ begins or ends to play. Moreover, the larger N, the more time we cover in the discrete signal and the worse our *time localisation* becomes. On the other hand, our frequency localisation improves when we increase N. To see why, we need the following theorem:

Theorem 1.3.1 *If* $|x^{\#}[\overline{k}]| = \max_k |x^{\#}[k]|$ *then* $\overline{k} = \min_k(|k-\phi| \mod N)$

Proof

Consider the function

$$f(\omega) = \begin{cases} \left| \frac{\sin((b-a)(\omega-\phi)\pi/N)}{\sin((\omega-\phi)\pi/N)} \right| & \text{if} \quad 0 \leq \omega < N, \omega \neq \phi \\ b-a & \text{if} \quad \omega = \phi. \end{cases} \tag{1.3.2}$$

We extend this function periodically by defining $f(\phi + \nu N) = f(\phi)$ for $\nu \in \mathbb{Z}$. It then is symmetric around $\omega = \phi$. Moreover, we have $|x^{\#}[k]| = f(k)$ for all k. We observe that

$$f(\omega) \leq \frac{1}{|\sin((\omega - \phi)\pi/N)|} =: q(\omega)$$

for all $\omega \neq \phi$. On $(\phi - \frac{1}{2}N, \phi)$ the function q increases monotonously, whilst it decreases monotonously on $(\phi, \phi + \frac{1}{2}N)$. The last time that $f(\omega)$ equals $q(\omega)$ before ω passes ϕ is located at $\phi - \frac{N}{2(b-a)} =: \omega_1$. Analogously is the first time that $f(\omega)$ equals $q(\omega)$ after passing ϕ located at $\phi + \frac{N}{2(b-a)} =: \omega_2$. This implies that $f(\omega) < f(\omega_1)$ for all $\omega \in [\phi - \frac{1}{2}N, \omega_1)$ and $f(\omega) < f(\omega_2)$ for all $\omega \in (\omega_2, \phi + \frac{1}{2}N)$. Since $f(t)$ is symmetric around ϕ, $f(\omega_1) = f(\omega_2)$ and the maximum lies somewhere in the interval $[\omega_1, \omega_2]$. We shall assert that in fact the maximum is attained at ϕ.

Since $f(\omega)$ is symmetric around ϕ, it is enough to show that $f(\omega)$ increases monotonously on $(\omega_1, \phi) =: J$. First we notice that $g(\omega) = \sin((b-a)(\omega - \phi)\pi/N) < 0$, and $h(\omega) = \sin((\omega - \phi)\pi/N) < 0$ on J. Therefore

$$f(\omega) = \frac{g(\omega)}{h(\omega)}$$

is well defined.

The derivative of f is greater than zero, if for all $\omega \in J$

$$g'(\omega)h(\omega) - g(\omega)h'(\omega) > 0$$

which is the case, when

$$(b-a)\cos\left((b-a)(\omega-\phi)\frac{\pi}{N}\right)\sin\left((\omega-\phi)\frac{\pi}{N}\right) > \sin\left((b-a)(\omega-\phi)\frac{\pi}{N}\right)\cos\left((\omega-\phi)\frac{\pi}{N}\right).$$

Notice that on J both $\cos\left((b-a)(\omega-\phi)\frac{\pi}{N}\right) > 0$ and $\cos\left((\omega-\phi)\frac{\pi}{N}\right) > 0$. Thus it is equivalent to show that

$$(b-a)\tan\left((\omega-\phi)\frac{\pi}{N}\right) > \tan\left((b-a)(\omega-\phi)\frac{\pi}{N}\right) \tag{1.3.3}$$

for each $\omega \in J$. Both sides of (1.3.3) have a tangent of $(b-a)\frac{\pi}{N}$ at ϕ. The derivative of

the left side is

$$(b-a)\frac{\pi}{N}\left(\tan^2\left((\omega-\phi)\frac{\pi}{N}\right)+1\right), \tag{1.3.4}$$

whilst the derivative of the right side is

$$(b-a)\frac{\pi}{N}\left(\tan^2\left((b-a)(\omega-\phi)\frac{\pi}{N}\right)+1\right). \tag{1.3.5}$$

The period of (1.3.4) is greater than the period of (1.3.5). Therefore (1.3.4) is strictly less than (1.3.5) on $J\backslash\{\phi\}$, which implies that the left side of (1.3.3) increases slower than its right side. Because both converge to $(b-a)\frac{\pi}{N}$ at ϕ, this means that on J the left side of (1.3.3) is greater than the right side of (1.3.3), which is what we wanted to show.
The last remark we have to make is that $\omega_2-\omega_1\geq 1$, which means that there is at least one $k\in\mathbb{Z}$ such that $k\in[\omega_1,\omega_2]$. Because on $f(k)$ increases monotonously on (ω_1,ϕ), and decreases monotonously on (ϕ,ω_2) this implies that $f(k)$ is maximal when k is nearest to ϕ. □

1.3.2 DTFT and windowing

If we take $a=0$ and $b=N$ in (1.3.2), we can also consider $x[n]$ as the result of a infinite phasor $\mathbf{y}=\{e^{\mathbf{i}\xi n}|n\in\mathbb{Z}\}$, componentwise multiplied by a *rectangular window* $\mathbf{w}=(w[n])_{n=-\infty}^{\infty}$ of size N:

$$w[n]=\begin{cases}1 & \text{if } 0\leq n<N\\ 0 & \text{otherwise.}\end{cases} \tag{1.3.6}$$

If we calculate its DTFT we find

$$w^{\#}(\omega)=\frac{\sin\frac{1}{2}\omega N}{\sin\frac{1}{2}\omega}e^{-\mathbf{i}\frac{1}{2}\omega(N-1)}$$

and

$$y^{\#}(\omega)=2\pi\delta(\omega-\xi).$$

One can verify that the DTFT of two componentwise multiplied signals equals the con-

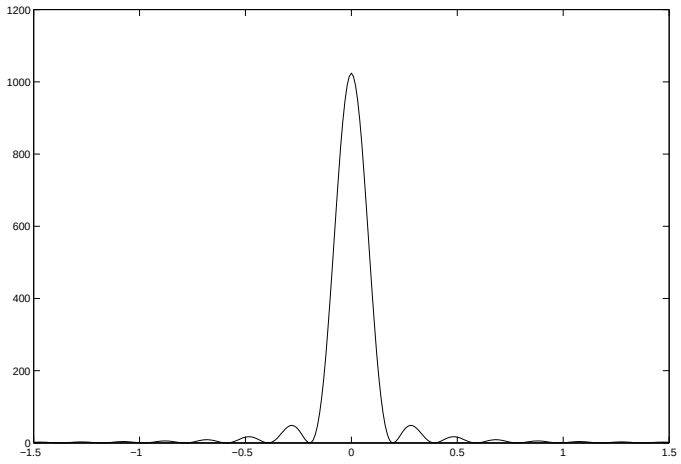


Figure 1.3.1: The graph of $|w^{\#}(\omega)|^2$ with $N = 32$, where $w[n]$ is the rectangular window as is given in (1.3.6).

volution product of the DTFTs of both signals, divided by 2π. This implies that if $x[n] = w[n]y[n]$ then

$$\begin{aligned} x^{\#}(\omega) &= \frac{1}{2\pi} w^{\#} * y^{\#}(\omega) \\ &= \frac{1}{2\pi} \int_{-\infty}^{\infty} w^{\#}(\zeta) y^{\#}(\omega - \zeta)\, d\zeta \\ &= \frac{\sin(\frac{1}{2}(\omega - \xi)N)}{\sin(\frac{1}{2}(\omega - \xi))} \mathrm{e}^{-\frac{1}{2}\mathbf{i}(\omega - \xi)(N-1)}, \end{aligned}$$

which agrees with (1.3.1), if we take $\phi = \frac{N\xi}{2\pi}$ and $k = \frac{N\omega}{2\pi}$.

A window is an important tool for analysing the frequency content of a digital sound: to measure the frequency content of the sound precisely we need to apply the DTFT to an infinitely number of samples. Obviously this is not possible in practice and therefore we need a window $\mathbf{w}$ of finite support and multiply this componentwise with the signal $\mathbf{y}$. If we apply the DTFT to the resulting windowed signal $\mathbf{x} = (x[n])_n$, where $x[n] = w[n]y[n]$, we find the frequency content of the windowed signal which is $x^{\#}(\omega) = \frac{1}{2\pi} w^{\#} * y^{\#}(\omega)$.

Let us now have a closer look at our rectangular window given in (1.3.6). We have plotted the square of the modulus of its transform $w^{\#}(\omega)$ in Figure 1.3.1 and observe that $|w^{\#}(\omega)|^2$

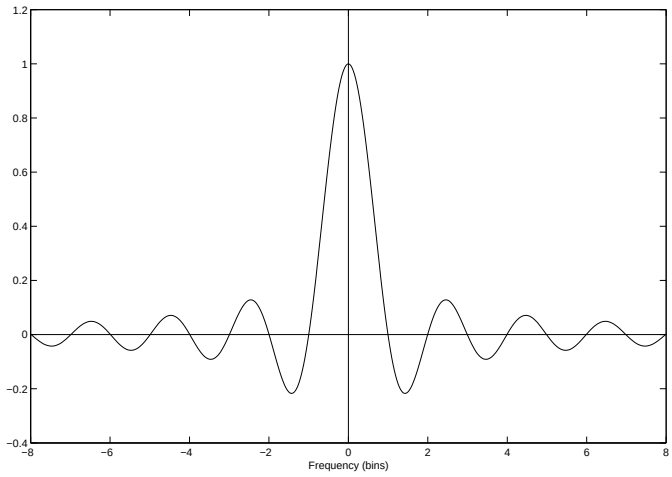


Figure 1.3.2: The DTFT of a rectangular window $\mathbf{w} = (w[n])_{n=-\infty}^{\infty}$, where $w[n] = 1$ if $-256 \leq n \leq 256$ and zero otherwise. We have normalised its DTFT $w^{\#}$ such that $w^{\#}(0) = \sum_n w[n] = 1$.

is an even function centred around 0. We have already seen that applying this window to the phasor $y[n] = \mathrm{e}^{-\mathbf{i}\xi n}$ yields $|x^{\#}(\omega)|^2 = |w^{\#}(\omega - \xi)|^2$. We can therefore find ξ by searching for the maximum value of $|x^{\#}(\omega)|^2$. This procedure is called *peak detection* and we will have a closer look at this in Section 2.5.5.

1.3.3 Choice of window

There are many possible functions $w(t)$ to choose as a window. It is convenient when w is a real function which is symmetric around $t = 0$, since it then has a real Fourier Transform which is symmetric around $\omega = 0$. The most common windows are real and symmetric, and have a Fourier Transform with a shape similar to Figure 1.3.2. In fact, this figure contains the DTFT of a symmetric rectangular window, which can be recognised as a Dirichlet kernel. When we measure the sum of two phasors $x[n] = x_1[n] + x_2[n]$ where $x_1[n] = \mathrm{e}^{\mathbf{i}\xi_1 n}$ and $x_2[n] = \mathrm{e}^{\mathbf{i}\xi_2 n}$ using a window $\mathbf{w} = (w[n])_n$, with DTFT $w^{\#}$, we see that in the frequency domain $y^{\#}(\omega) = w^{\#}(\omega - \xi_1) + w^{\#}(\omega - \xi_2)$. It is the sum of the DTFT of $\mathbf{w}$ translated by respectively ξ_1 and ξ_2. It is important that ξ_1 and ξ_2 are located far enough from each other, such that $w^{\#}(\xi_2 - \xi_1)$ is negligible and we can find ξ_1 and ξ_2 by

Name	$w[n]$	$\Delta\omega$ (bins)	**A** (dB)
Rectangular	1	1.21	-13
Hamming	$0.54 + 0.46\cos\left(2\pi\frac{n}{N}\right)$	1.81	-43
Gaussian	$\exp\left(-18\left(\frac{n}{N}\right)^2\right)$	2.18	-55
Hanning	$\cos(\pi\frac{n}{N})^2$	2.00	-32

Table 1.1: Properties of windows. The number n takes integer values in $\left[-\frac{1}{2}N, \frac{1}{2}N\right]$. In the third column we find the root mean square bandwidth, which is calculated by (1.3.7) and given in bins, where one bin is $\frac{2\pi}{N+1}$ radians/sample. In the last column the amplitude of the first side lobe, which is calculated by (1.3.8).

searching for the peaks in $y^{\#}$.

Following [15] we introduce the number $\Delta\omega$, the root-mean square bandwidth $\Delta\omega$ of the -6dB points, defined by the equation

$$\frac{|w^{\#}(\frac{1}{2}\Delta\omega)|^2}{|w^{\#}(0)|^2} = \frac{1}{4}. \tag{1.3.7}$$

Recall that the difference in decibel (dB) between two values A_1 and A_2 is equal to $20\log_{10}\left|\frac{A_1}{A_2}\right|$.

The number $\Delta\omega$ is useful for evaluating a window: if we measure two phasors with the same amplitude whose frequencies ξ_1 and ξ_2 are more than $\Delta\omega$ apart, the Fourier Transform of the windowed signal at $\omega = \frac{1}{2}(\xi_1 + \xi_2)$ is smaller than the Fourier Transform at ξ_1 and ξ_2. It ensures that each frequency has its own peak.

Because we try to measure frequencies by looking at the peaks in the spectrum of the windowed signal, it is important that the amplitudes of the side lobes are not to large, lest they are interpreted as peaks. A measure for the amplitude of the side lobes is the amplitude of the first side lobe, which is located at $\omega = \pm\omega_0$ and measured in decibels:

$$A = 10\log_{10}\frac{|w^{\#}(\omega_0)|^2}{|w^{\#}(0)|^2}. \tag{1.3.8}$$

In Table 1.1 we have collected some properties of various windows from [11], and in Figure 1.3.3 we have made a plot of the last three windows and their DTFTs. We have restricted the support of the windows to $\left[-\frac{1}{2}N, \frac{1}{2}N\right]$ and the values for A and $\Delta\omega$ are limits to which the respective properties converge when N is increased. Notice that $\Delta\omega$ is given in

frequency bins: a *frequency bin* is one element of the DFT. The width of a frequency bin is defined by $\frac{2\pi}{N}$, where N is the number of samples used in the DFT. The frequency bin $x^{\#}[k]$ is said to have a frequency of $k\frac{2\pi}{N}$ rad/sample associated to it. It is useful to express the bandwidth in terms of frequency bins because the bandwidth is proportional to the support of the window.

1.4 The Gibbs phenomenon

In the continuous time domain, let us consider a signal $f(t) \in \mathbf{L}^2(\mathbb{R})$ with Fourier Transform $\widehat{f}(\omega)$. If f is continuous, then

$$f_\xi(t) = \frac{1}{2\pi}\int_{-\xi}^{\xi} \widehat{f}(\omega)\mathrm{e}^{\mathbf{i}\omega t}\, d\omega \tag{1.4.1}$$

converges to $f(t)$ for all t when ξ goes to infinity. However, when f has a discontinuity at t_0, *Gibbs oscillations* occur: these are oscillations that occur in a neighbourhood of t_0, and have a maximum amplitude that does not vanish when ξ goes to infinity, but goes to a constant instead. Fortunately the time support in which these oscillations occur goes to zero as well as the energy of the oscillations, implying that $\|f - f_\xi\|_2^2$ goes to zero when ξ goes to infinity. This phenomenon was first explained by J.W. Gibbs [10]. For a proof of the above statements we refer to [15], page 34 ff..

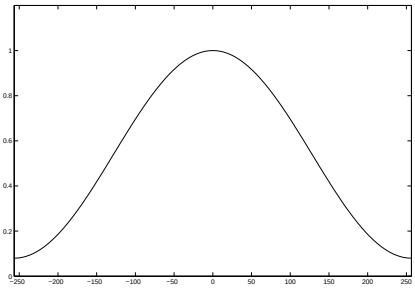
In the discrete time domain, although we cannot speak of continuous or discontinuous signals, a similar problem arises. To see how it works, let us first consider (1.4.1) as the result of an *ideal low pass filter* $h_\xi(t)$, which has a Fourier Transform (also called *transfer function*)

$$\widehat{h}_\xi(\omega) = \begin{cases} 1 & \text{if } |\omega| < \xi \\ 0 & \text{otherwise,} \end{cases}$$

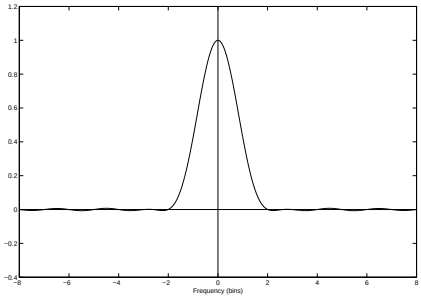
applied to the signal f.

We can translate this in the discrete domain by creating a discrete filter $\mathbf{g}_\xi = (g_\xi[0], g_\xi[1], ..., g_\xi[N-1])$ which has a DTFT (which we will also call transfer function) equal to

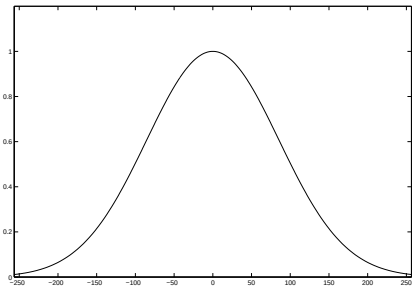
$$g_\xi^{\#}(\omega) = \begin{cases} 1 & \text{if } |\omega| < \xi \\ 0 & \text{otherwise.} \end{cases} \tag{1.4.2}$$



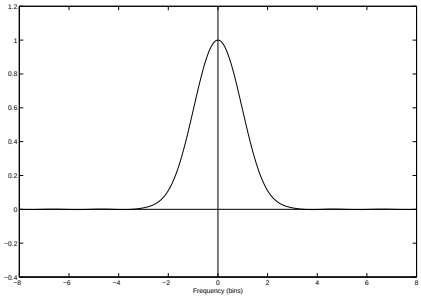
Hamming window



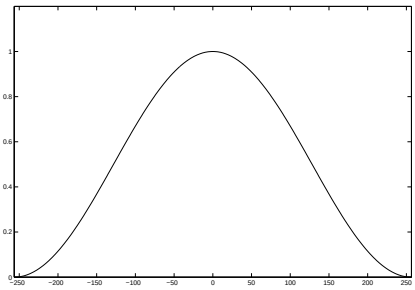
DTFT of a Hamming window



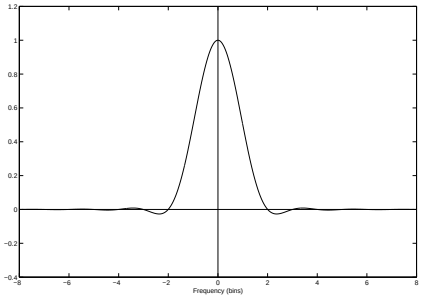
Gaussian window



DTFT of a Gaussian window



Hanning window



DTFT of a Hanning window

Figure 1.3.3: Various windows and their DTFTs. We have taken $N = 1023$ and normalised the DTFTs $w^{\#}$ such that $w^{\#}(0) = 1$.

If we denote the result of applying this filter to a vector $\mathbf{x}$ by $\mathbf{y}_\xi$, we have in the frequency domain: $y_\xi^\#(\omega) = x^\#(\omega)g_\xi^\#(\omega)$, which means that in the time domain we have $y_\xi[n] = \mathbf{x} * \mathbf{g}_\xi[n]$, where $\mathbf{g}_\xi$ is the inverse DTFT of $g_\xi^\#$, also called the filter's *impluse response*. We see that for nonzero n we have $g_\xi[n] = \frac{\sin(\xi n)}{\pi n}$, whilst for $n = 0$ we have $g_\xi[0] = \frac{\xi}{\pi}$.
If we take $\xi = \pi$ we see that $g_\xi[n] = 0$ for $n \neq 0$ whilst $g_\xi[0] = 1$, and therefore $y[n] = x[n]$, which makes sense since $x[n]$ can only contain frequencies in $[-\pi, \pi]$. If however we decrease ξ, we see that the time support of $\mathbf{g}_\xi$ is stretched and $g_\xi[n]$ can become nonzero for nonzero n as well. This means that $y_\xi[n]$ becomes dependent on other samples besides $x[n]$. If $\mathbf{x}$ is slowly varying, this is not much of a problem, since then the convolution $\mathbf{x} * \mathbf{g}_\xi[n]$ will be approximately equal to $Cx[n]$ for all n, where C is a constant. If however $\mathbf{x}$ is slowly varying except for a big jump between $x[n_0]$ and $x[n_0 + 1]$, we see that around n_0 the balance is distorted and Gibbs oscillations occur.
If Figure 1.4.1 we illustrate what happens when we apply the filter $\mathbf{g}_{\frac{\pi}{8}}$ to the signal $\mathbf{x}$ defined as

$$x[n] = \begin{cases} 1 & \text{if } n \geq 0 \\ 0 & \text{if } n < 0. \end{cases}$$

We see that around $n = 0$ Gibbs oscillations occur.

1.5 The Heisenberg uncertainty principle

Let us consider in the continuous domain a window $w(t)$ with finite support, which has a Fourier Transform $\widehat{w}(\omega)$. If we stretch this window by a factor s, we acquire a new window $g(t) = \frac{1}{\sqrt{s}} w(\frac{t}{s})$, with Fourier Transform $\widehat{g}(\omega) = \sqrt{s}\widehat{w}(\omega s)$. We see that the Fourier Transform is shrunk by the factor s. The energy $\|g\|_2^2$ is equal to $\|w\|_2^2$, and likewise is the energy $\|\widehat{g}\|_2^2$ equal to $\|\widehat{w}\|_2^2$. This implies that the energy of the window w is spread in g over a bigger time support whilst the energy of $\widehat{w}$ is spread over a smaller frequency support in $\widehat{g}$, which means that a decrease in time resolution equals an increase in frequency resolution.
Following [15], let us interpret $\frac{|w(t)|^2}{\|w\|_2^2}$ as a probability density, which has an average

$$u = \frac{1}{\|w\|_2^2} \int_{-\infty}^{\infty} t|w(t)|^2 \, dt$$

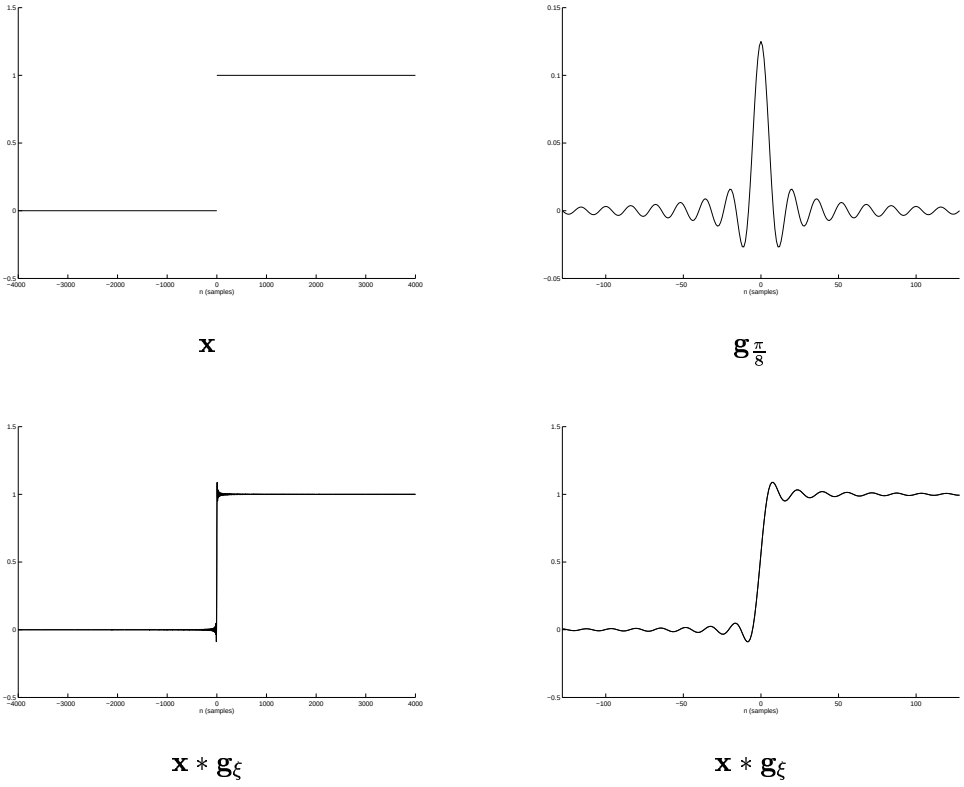


Figure 1.4.1: An illustration of the Gibbs phenomenon. We apply a discrete ideal low pass filter $g_{\frac{\pi}{8}}^{\#}$ (as defined in equation (1.4.2)), which has an impulse response $\mathbf{g}_{\frac{\pi}{8}}$, to the signal $\mathbf{x}$, which is constant everywhere except for a jump from 0 to 1 between $n = -1$ and $n = 0$. We see that Gibbs oscillations occur around the jump.

and a variance

$$\sigma_t^2 = \frac{1}{\|w\|_2^2} \int_{-\infty}^{\infty} (t-u)^2 |w(t)|^2 \, dt.$$

The same quantities can also be computed for $\frac{|\widehat{w}(\omega)|^2}{\|\widehat{w}\|_2^2}$, resulting in the mean value

$$\xi = \frac{1}{2\pi \|w\|_2^2} \int_{-\infty}^{\infty} t |\widehat{w}(\omega)|^2 \, d\omega$$

and a variance

$$\sigma_\omega^2 = \frac{1}{2\pi \|w\|_2^2} \int_{-\infty}^{\infty} (\xi - \omega)^2 |\widehat{w}(\omega)|^2 \, d\omega.$$

The *Heisenberg uncertainty principle* states that if $w \in \mathbf{L}^2(\mathbb{R})$ then $\sigma_t^2 \sigma_\omega^2 \geq \frac{1}{4}$, which

means that we cannot reach an arbitrary precision in both time resolution and frequency resolution.

The Heisenberg uncertainty principle originates from quantum mechanics [12]. It was restated in a statistical setting in [24] and has known many generalisations in operator theory and Fourier theory since ([9], [14]). For a proof of the above identities we refer to [15], p. 31.

1.6 The relationship between the DTFT and the Fourier Transform

The Heisenberg uncertainty principle also applies to the DTFT. To see why, let us study the relationship between the DTFT and the Fourier Transform. We will do this by two different approaches: first we consider the DTFT as an approximation of the continuous time Fourier Transform, and secondly we will have a look at the *Whittaker Sampling Theorem.*

Theorem 1.6.1 *Let $w(t)$ be a continuous differentiable signal with support $[-\frac{1}{2}, \frac{1}{2})$. We sample this signal by N samples and normalise, such that $W_N[n] = \frac{1}{\sqrt{N}} w\left(\frac{n}{N}\right)$. Let $W_N^{\#}(\omega)$ the DTFT of W_N, and $\widehat{w}(\omega)$ the Fourier Transform of w. Using the notation $f_N \simeq f$ if $\lim_{N\to\infty} |f_N - f| = 0$, the following estimates hold:*

$$w(t) \simeq \sqrt{N} W_N[Nt], \tag{1.6.1}$$

$$\|w\|_2 \simeq \|W_N\|_N, \tag{1.6.2}$$

$$\widehat{w}(\omega) \simeq \frac{1}{\sqrt{N}} W_N^{\#}(\zeta_N), \tag{1.6.3}$$

$$\|\widehat{w}\|_2 \simeq \|W_N^{\#}\|_2, \tag{1.6.4}$$

where $\zeta_N = \frac{\omega}{N}$, $\|W_N\|_N^2 = \sum_{n=-\frac{1}{2}N}^{\frac{1}{2}N-1} |W_N[n]|^2$, $\|\widehat{w}\|_2^2 = \int_{-\infty}^{\infty} |\widehat{w}(\zeta)|^2 \, d\zeta$ and $\|W_N^{\#}\|_2^2 = \int_{-\pi}^{\pi} |W_N^{\#}(\zeta)|^2 \, d\zeta$.

For the proof of Theorem 1.6.1 we need the following lemma:

Lemma 1.6.1 *Let $f(t)$ be a continuous differentiable function which has a support of $[-\frac{1}{2}, \frac{1}{2})$. Then*

$$\left| \int_{-\frac{1}{2}}^{\frac{1}{2}} f(t)\, dt - \frac{1}{N} \sum_{n=-\frac{1}{2}N}^{\frac{1}{2}N-1} f\left(\frac{n}{N}\right) \right| \leq \frac{1}{2N} \|f'\|_\infty$$

Proof

$$\begin{aligned}
&\left| \int_{-\frac{1}{2}}^{\frac{1}{2}} f(t)\, dt - \frac{1}{N} \sum_{n=-\frac{1}{2}N}^{\frac{1}{2}N-1} f\left(\frac{n}{N}\right) \right| \\
&= \left| \sum_{n=-\frac{1}{2}N}^{\frac{1}{2}N-1} \left\{ \int_0^{\frac{1}{N}} f\left(\frac{n}{N} + t\right)\, dt - \frac{1}{N} f\left(\frac{n}{N}\right) \right\} \right| \\
&= \left| \sum_{n=-\frac{1}{2}N}^{\frac{1}{2}N-1} \left\{ \int_0^{\frac{1}{N}} \left(f\left(\frac{n}{N}\right) + \int_0^t f'\left(\frac{n}{N} + \sigma\right)\, d\sigma \right) dt - \frac{1}{N} f\left(\frac{n}{N}\right) \right\} \right| \\
&= \left| \sum_{n=-\frac{1}{2}N}^{\frac{1}{2}N-1} \left\{ \int_0^{\frac{1}{N}} \int_0^t f'\left(\frac{n}{N} + \sigma\right)\, d\sigma\, dt \right\} \right| \\
&\leq \sum_{n=-\frac{1}{2}N}^{\frac{1}{2}N-1} \int_0^{\frac{1}{N}} \int_0^t \|f'\|_\infty\, d\sigma\, dt = \frac{1}{2N} \|f'\|_\infty.
\end{aligned}$$

□

Notice that the bound of Lemma 1.6.1 only is sharp when f equals 0 on $[-\frac{1}{2}, \frac{1}{2})$. If we allow $f(t)$ to be discontinuous at $t = -\frac{1}{2}$ and $t = \frac{1}{2}$ and define $f'(-\frac{1}{2}) = \lim_{h \downarrow 0} \frac{f(-\frac{1}{2}+h) - f(-\frac{1}{2})}{h}$, the bound is sharp when $f(t)$ equals a line on $[-\frac{1}{2}, \frac{1}{2})$.

Proof of Theorem 1.6.1

(1.6.1) $w(t) \simeq \sqrt{N} W_N[Nt]$: true for t=$\frac{n}{N}$, n integer such that $-\frac{1}{2} \leq \frac{n}{N} < \frac{1}{2}$. For other t the result follows by the continuity of w.

(1.6.2) $\|w\|_2 \simeq \|W_N\|_2$: The application of Lemma 1.6.1 to $f(t) = |w(t)|^2$ yields:

$$\left| \|w\|_2^2 - \|W_N\|_2^2 \right| \leq \frac{1}{N} \|w'\|_\infty \|w\|_\infty$$

(1.6.3) $\widehat{w}(\omega) \simeq \frac{1}{\sqrt{N}} W_N^{\#}(\zeta_N)$: Taking $f(t) = w(t)\mathrm{e}^{-\mathbf{i}\omega t}$ in Lemma 1.6.1 yields:

$$\begin{aligned}\left| \int_{-\frac{1}{2}}^{\frac{1}{2}} w(t)\mathrm{e}^{-\mathbf{i}\omega n}\, dt - \frac{1}{N} \sum_{n=-\frac{1}{2}N}^{\frac{1}{2}N-1} w\left(\frac{n}{N}\right) \mathrm{e}^{-\mathbf{i}\omega \frac{n}{N}} \right| &= \left| \int_{-\frac{1}{2}}^{\frac{1}{2}} w(t)\mathrm{e}^{-\mathbf{i}\omega t}\, dt - \frac{1}{\sqrt{N}} W_N^{\#}\left(\frac{\omega}{N}\right) \right| \\ &\leq \frac{\|w'\|_\infty + |\omega|\|w\|_\infty}{2N},\end{aligned}$$

(1.6.4) $\|\widehat{w}\|_2 \simeq \|W_N^{\#}\|_2$: since both w and W_N have finite support, we can apply the *continuous-time Plancherel formula*

$$\|\widehat{w}\|_2^2 = 2\pi \|w\|_2^2$$

and the *discrete-time Plancherel formula*

$$\|W_N^{\#}\|_2^2 = 2\pi \|W_N\|_2^2$$

to find

$$\left| \|\widehat{w}\|_2^2 - \|W_N^{\#}\|_2^2 \right| = 2\pi \left| \|w\|_2^2 - \|W_N\|_2^2 \right| \leq \frac{2\pi}{N} \|w'\|_\infty \|w\|_\infty.$$

□

Notice that in Theorem 1.6.1 the frequency variable ω is in radians, whilst normally we measure the frequency of discrete signals in radians per sample. Conversion of the frequency ω to radians per sample is done by dividing ω by N, resulting in $\zeta_N = \frac{\omega}{N}$, such that $w_N^{\#}(\zeta_N) \to \sqrt{N}\widehat{w}(\omega)$ when $N \to \infty$. This means that an increase of N squeezes the frequency support of $W_N^{\#}$, whilst it stretches the (discrete) time support of W_N.
For finite N, we can conclude from Theorem 1.6.1 that around the frequency 0 the DTFT behaves similar to the Fourier Transform. For higher frequencies however, the theorem does not give much information about the connection between both transforms. To gain more insight in what happens at higher frequencies, let us follow [15] and have a look at the Whittaker Sampling Theorem.

First we consider the function

$$f_d(t) = \sum_{n=-\infty}^{\infty} f(nT_s)\delta(t - nT_s) = \sum_{n=-\infty}^{\infty} x[n]\delta(t - nT_s),$$

where $\delta(t)$ is the Dirac function and $\mathbf{x} = (x[n])_{n=-\infty}^{\infty}$ a sampled version of f, such that $x[n] = f(nT_s)$. Taking the Fourier Transform of f_d yields

$$\widehat{f_d}(\omega) = \sum_{n=-\infty}^{\infty} x[n]\mathrm{e}^{-\mathbf{i}\omega T_s n} = x^{\#}(\zeta),$$

where $x^{\#}$ is the DTFT of $\mathbf{x}$ and $\zeta = \omega T_s$. Notice again the difference in units. For example, when ω is measured in radians per second and T_s is measured in seconds per sample, then ζ is measured in radians per sample.

Theorem 1.6.2 *The Fourier Transform of the discrete signal obtained by sampling f at intervals T_s is*

$$\widehat{f_d}(\omega) = \frac{1}{T_s} \sum_{k=-\infty}^{\infty} \widehat{f}\left(\omega - \frac{2k\pi}{T_s}\right). \tag{1.6.5}$$

We see that $x^{\#}(\zeta)$ equals $\widehat{f_d}\left(\frac{\zeta}{T_s}\right)$. Once again we see the aliasing phenomenon in formula (1.6.5): the DTFT of $\mathbf{x}$ evaluates in ζ to the sum of the Fourier Transform of f evaluated in $\frac{\zeta}{T_s}$ and its aliases. Also, we see that if we take the sample rate $\phi_s = \frac{1}{T_s}$ such that the support of $\widehat{f}$ is included in $[-\pi\phi_s, \pi\phi_s]$, we have all the information of $\widehat{f}$ in $x^{\#}$. This means that we can recover f from $x^{\#}$ completely. This is done by the use of the *Whittaker Sampling Theorem*:

Theorem 1.6.3 *If the support of $\widehat{f}$ is included in $\left[-\frac{\pi}{T_s}, \frac{\pi}{T_s}\right]$ then*

$$f(t) = \sum_{n=-\infty}^{\infty} f(nT_s)h_{T_s}(t - nT_s),$$

with

$$h_{T_s}(t) = \frac{\sin(\pi t/T_s)}{\pi t/T_s}.$$

For proofs of Theorem 1.6.2 and Theorem 1.6.3 we refer to [15], p. 43 ff.. In this book it is also proved that it is impossible for f to have both finite support in the frequency domain

and in the time domain. This means that in practical situations the requirement on the support of f in Theorem 1.6.3 cannot be met. However there are finite functions whose Fourier Transforms decrease sufficiently fast such that they become negligible outside a certain finite interval D, and Theorem 1.6.3 holds approximately. This is the case for most popular windows, which means if we take N big enough, the DTFTs of these windows resemble their Fourier Transforms very well.

As an example, let us look at the rectangular window r. For even N we define in the continuous domain:

$$R(t) = \begin{cases} 1 & \text{if } -\frac{1}{2}N - \frac{1}{2} < t < \frac{1}{2}N + \frac{1}{2} \\ 0 & \text{otherwise.} \end{cases}$$

We sample this window with a sample rate $T_s = 1$, such that the continuous frequency in radians is equal to the discrete frequency in radians per sample. This yields the result $\mathbf{r} = (r[n])_{n=-\infty}^{\infty}$, which is given by:

$$r[n] = \begin{cases} 1 & \text{if } -\frac{1}{2}N \leq n \leq \frac{1}{2}N \\ 0 & \text{otherwise.} \end{cases}$$

The DTFT of $\mathbf{r}$ equals

$$r^{\#}(\omega) = \frac{\sin \frac{1}{2}(N+1)\omega}{\sin \frac{1}{2}\omega},$$

whilst the Fourier Transform of R equals

$$\widehat{R}(\omega) = \frac{\sin \frac{1}{2}(N+1)\omega}{\frac{1}{2}\omega}.$$

We can estimate the difference between both transforms by

$$\left| r^{\#}(\omega) - \widehat{R}(\omega) \right| \leq \left| \frac{1}{\sin \frac{1}{2}\omega} - \frac{1}{\frac{1}{2}\omega} \right|, \tag{1.6.6}$$

which takes the biggest value on $[-\pi, \pi]$ when $\omega = \pm\pi$, and then it equals $1 - \frac{2}{\pi}$. When we compare this to the value $\widehat{R}(0) = N + 1$, we see that it is a very small disturbance.

As another example, let us have a look at the Hanning window. In the continuous domain:

$$W(t) = \begin{cases} \cos^2\left(\frac{t\pi}{N+1}\right) & \text{if } -\frac{1}{2}N - \frac{1}{2} < t < \frac{1}{2}N + \frac{1}{2} \\ 0 & \text{otherwise.} \end{cases}$$

Notice that we can rewrite

$$\cos^2\left(\frac{t\pi}{N+1}\right) = \frac{1}{4}\mathrm{e}^{-\mathbf{i}\frac{2\pi}{N+1}t} + \frac{1}{4}\mathrm{e}^{\mathbf{i}\frac{2\pi}{N+1}t} + \frac{1}{2},$$

and therefore

$$W(t) = \frac{1}{4}\mathrm{e}^{-\mathbf{i}\frac{2\pi}{N+1}t}R(t) + \frac{1}{4}\mathrm{e}^{\mathbf{i}\frac{2\pi}{N+1}t}R(t) + \frac{1}{2}R(t).$$

Since the Fourier Transform of $\mathrm{e}^{-\mathbf{i}\xi t}R(t)$ equals $\widehat{R}(\omega+\xi)$, we see that the Fourier Transform of W equals

$$\widehat{W}(\omega) = \frac{1}{4}\widehat{R}\left(\omega + \frac{2\pi}{N+1}\right) + \frac{1}{4}\widehat{R}\left(\omega - \frac{2\pi}{N+1}\right) + \frac{1}{2}\widehat{R}(\omega).$$

For the DTFT we have the same rule: the DTFT of $\mathrm{e}^{-\mathbf{i}\xi t}r(t)$ equals $r^{\#}(\omega+\xi)$. This means that if we sample $W(t)$ into $\mathbf{w} \in \mathbb{C}^{\infty}$ in the same manner as in the previous example, we find that the DTFT of $\mathbf{w}$ equals

$$w^{\#}(\omega) = \frac{1}{4}r^{\#}\left(\omega + \frac{2\pi}{N+1}\right) + \frac{1}{4}r^{\#}\left(\omega - \frac{2\pi}{N+1}\right) + \frac{1}{2}r^{\#}(\omega).$$

If we write $\xi_N = \frac{2\pi}{N+1}$ we can use (1.6.6) to estimate $\left|r^{\#}(\omega+\xi_N) - \widehat{R}(\omega+\xi_N)\right|$ and see that the right side of this estimate is maximal when $\omega = \pi$, and then equals $\frac{1}{\sin\pi\left(\frac{1}{2}+\frac{1}{N+1}\right)} - \frac{1}{\pi\left(\frac{1}{2}+\frac{1}{N+1}\right)}$. For the other two terms we have similar estimates, and we see that

$$\left|w^{\#}(\omega+\xi_N) - \widehat{W}(\omega+\xi_N)\right| \leq \frac{1}{2\sin\pi\left(\frac{1}{2}+\frac{1}{N+1}\right)} - \frac{1}{2\left(\frac{1}{2}+\frac{1}{N+1}\right)\pi} + \frac{1}{2} - \frac{1}{\pi} \to 1 - \frac{2}{\pi}$$

when $N \to \infty$. Compared to $w^{\#}(0) = \frac{1}{2}(N+1)$ this difference is negligible.

1.7 Wavelets

The result of the DTFT applied to a windowed signal is called the *Short-Time Fourier Transform* (STFT). In formula, if $\mathbf{x}$ is the signal and $\mathbf{w}$ the window,

$$\mathcal{S}\mathbf{x}(m,\xi) = \sum_{n=-\infty}^{\infty} x[n]w[n-m]\mathrm{e}^{-\mathbf{i}\xi(n-m)} = \sum_{n=-\infty}^{\infty} x[n+m]w[n]\mathrm{e}^{-\mathbf{i}\xi n}. \tag{1.7.1}$$

It can be used to measure the energy of $\mathbf{x}$ in a neighbourhood of m in the time domain, and in a neighbourhood of ξ in the frequency domain (see Section 2.1). Because $\mathbf{w}$ is chosen independently of frequency, for all frequencies the same trade-off is made between time resolution and frequency resolution.

One could also choose to make $\mathbf{w}$ dependent on ξ. This could be useful since periodic signals with low frequencies have a longer period than signals with higher frequencies, and if we want to measure using a constant number of periods we need a window with more time support for lower frequencies than we need for higher frequencies.

Notice that we can rewrite (1.7.1) to the infinite inner product

$$\mathcal{S}\mathbf{x}(m,\xi) = < \mathbf{x}, \mathbf{W}_{m,\xi} >,$$

where $\mathbf{W}_{m,\xi} = (W_{m,\xi}[n])_{N=-\infty}^{\infty}$ with $W_{m,\xi}[n] = w[n-m]\mathrm{e}^{-\mathbf{i}\xi(n-m)}$. $\mathbf{W}_{m,\xi}$ is called a *discrete Fourier atom* centred around (m,ξ).

For analysing sound signals, or more generally, signals where both phase and frequency are significant, Gabor wavelets are feasible. A *Gabor wavelet* $\psi_{u,\xi}$, is defined by

$$\psi_{u,\xi}(t) = \sqrt{\xi} g\left(\xi(t-u)\right) \mathrm{e}^{-\mathbf{i}\xi(t-u)}.$$

where $g(t)$ is a Gaussian window

$$g(t) = \frac{1}{(\sigma^2\pi)^{\frac{1}{4}}} \exp\left(-\frac{t^2}{2\sigma^2}\right),$$

with σ a real constant. We can now define a wavelet transform applied on a continuous

time signal f by

$$Wf(u,\xi) =< f, \psi_{u,\xi} >=< f(t), \sqrt{\xi}\psi\left(\xi(t-u)\right) >, \tag{1.7.2}$$

where

$$\psi(t) = g(t)\mathrm{e}^{-\mathbf{i}t}. \tag{1.7.3}$$

Formula (1.7.2) is called the *continuous wavelet transform* with *mother wavelet* ψ. The value u is called the *translation parameter* and the value $\frac{1}{\xi}$ the *dilatation parameter*. Formula (1.7.2) is a general formula for all continuous wavelets, whilst the choice (1.7.3) makes it a Gabor wavelet transform.
Discretising (1.7.2) yields the *Discrete Wavelet Transform* (DWT):

$$\mathcal{W}\mathbf{x}(m,\xi) =< \mathbf{x}, \mathbf{\Psi}_{m,\xi} >,$$

where $\mathbf{x}$ is a discrete signal and $\mathbf{\Psi}_{m,\xi}[n] = \sqrt{\xi}\psi(\xi(n-m))$. We see that the DWT has the desired properties: it is the STFT with a window that has a support which is proportional to the wavelength of the frequency ξ.

Chapter 2
Splitting sound

In this chapter we describe our experiments and their results. We will develop a method that follows the most pronounced instrument in a piece of music, and that separates it from the rest. In our research we have focussed on the clarinet, but the proposed method can be applied for the extraction of many other melody instruments as well.

2.1 Spectrograms and scalograms

To split a sound signal into two meaningful components (opposed to splitting a sound signal in one meaningful component and a noise signal), it is important to know what the sound looks like. A way to do this is by looking at the sound's frequency content. This can be done using a variety of tools, of which we shall apply two: the Short-Time Fourier Transform and a Discrete Wavelet Transform.
Using the STFT, we can visualise the frequency content of a signal $\mathbf{x}$ by the use of a spectrogram: The *spectrogram* $\mathcal{P}_{\mathcal{S}}\mathbf{x}$ is defined by

$$\mathcal{P}_{\mathcal{S}}\mathbf{x}(m,\xi) = |\mathcal{S}\mathbf{x}(m,\xi)|^2$$

The spectrogram measures the energy of $\mathbf{x}$ around the time-frequency neighbourhood (m,ξ).
In Figure 2.1.1 there is a graphical representation of the spectrograms of a clarinet, a bass flute, a piano and a viola. The values of $\mathcal{P}_{\mathcal{S}}\mathbf{x}(m,\xi)$ are determined by the colour in the picture; black indicating a high value and white a low value. Notice that the spectrograms

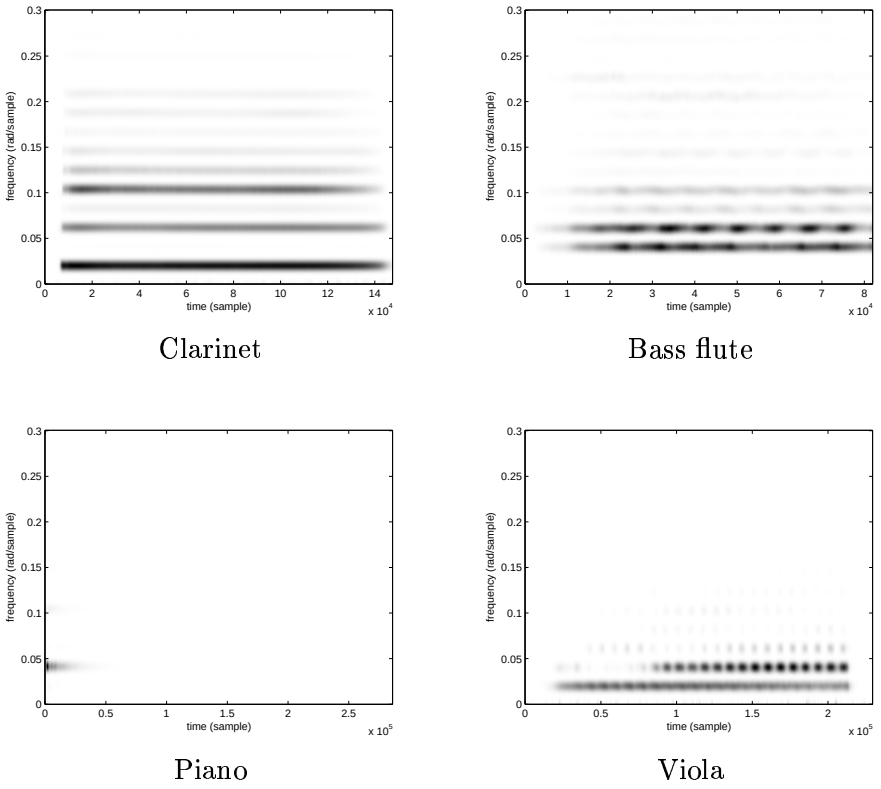


Figure 2.1.1: The spectrograms of a D3 note by four instruments, using the STFT with a Hanning window with a support of 1024 samples. Notice the darker spots in the bass flute and viola spectra, which are caused by the natural vibrato of the musicians.

of the instruments consist of a number of lines, which are called *spectral lines*. Most natural instruments produce sounds that consist of spectral lines. The lowest spectral line is called the *fundamental frequency*, whilst the other spectral lines are called *harmonics*, which have frequencies that are (approximately) an integer multiple of the fundamental frequency. We say that a harmonic is the n^{th} harmonic if its frequency is approximately equal to n times the fundamental frequency. Notice that the clarinet lacks the second and fourth harmonic, which is in agreement with the physical structure of a clarinet.

An approach, which looks much the same as a spectrogram at the first glance, is doing analysis by wavelets. Instead of looking at the spectrogram, we look at the *scalogram* $\mathcal{P}_{\mathcal{W}}\mathbf{x}$, which is defined by

$$\mathcal{P}_{\mathcal{W}}\mathbf{x}(m,\xi) = |\mathcal{W}\mathbf{x}(m,\xi)|^2$$

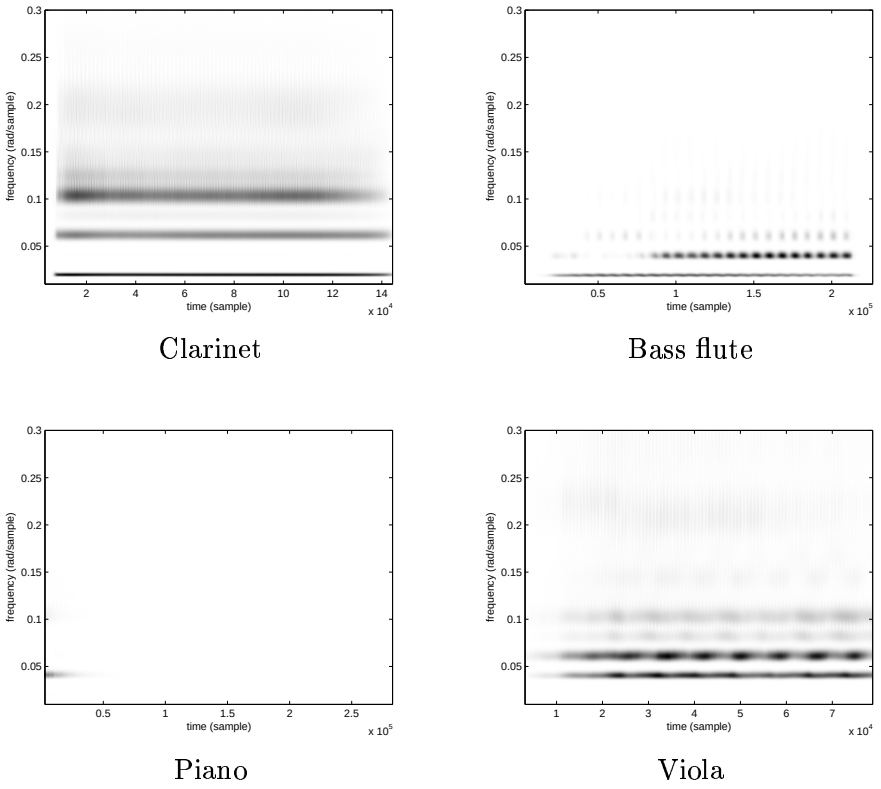


Figure 2.1.2: The scalograms of four instruments, using the mother wavelet $\psi(t) = \mathrm{e}^{-\mathbf{i}0.064t}\cos^2\left(\frac{\pi}{1024}t\right)$, which has a support of $(-512, 512)$ and is centred around 0 in the time domain, and around 0.064 in the frequency domain.

As is the case with a spectrogram, a scalogram measures the energy of $\mathbf{x}$ in a time-frequency neighbourhood of (u, ξ), but uses different trade-offs between time and frequency resolutions at different frequencies. In Figure 2.1.2 we have made visible the scalograms of the four instruments. We have chosen the mother wavelet $\psi(t) = \mathrm{e}^{-\mathbf{i}0.064t}\cos^2\left(\frac{\pi}{1024}t\right)$, such that the support of the wavelet at the frequency 0.064 radians/sample (which is the approximate frequency of the third harmonic) is 1024, and the shape of the third harmonic of in the scalogram resembles the shape of the third harmonic in the spectrogram in Figure 2.1.1. Notice that the spectral lines of the frequencies below 0.064 have a smaller width than their counterparts in Figure 2.1.1, whilst the width of the spectral lines of higher frequencies is bigger.

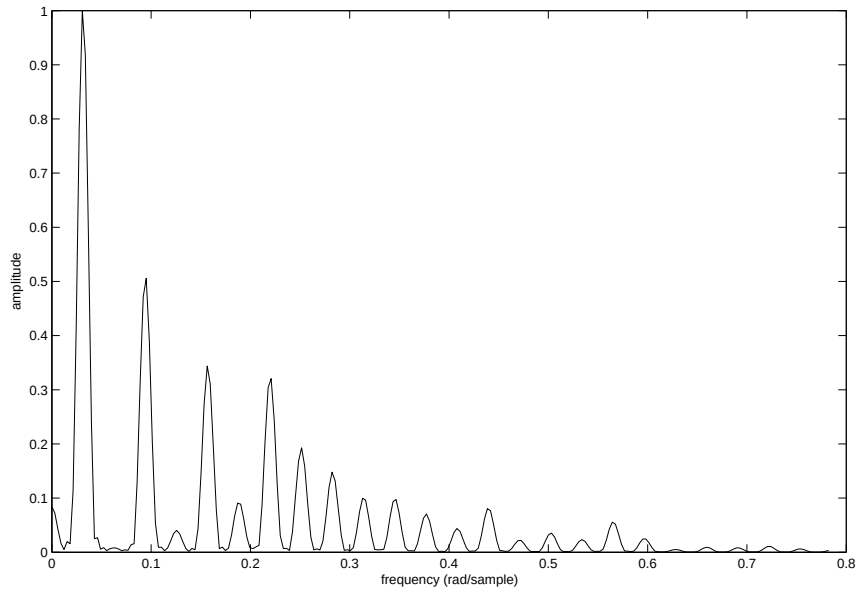


Figure 2.2.1: The frequency content of a clarinet tone A3. Notice the absence of the lower even multiples of the fundamental frequency, which is characteristic for instruments which are closed at one side, such as the clarinet.

2.2 An experiment in separating a clarinet and a viola

From the *McGill university master samples collection* [18][1], we have sampled two signals: one of a clarinet playing the note A3, which we shall denote by $\mathbf{c} = (c[0], c[1], ..., c[L-1])$, and one of a viola playing $A^{\#}3$ which we shall denote by $\mathbf{v} = (v[0], v[1], ..., v[L-1])$. Since the length of the sampled signal is influenced by the duration of the original analogue version, and original signals did not have the same length, it was necessary to clip the sampled clarinet signal to reduce its length to L samples. We have created a new signal $\mathbf{x}$ by adding $\mathbf{c}$ and $\mathbf{v}$ and tried to split it again. The splitting has been done by looking at the various harmonics: in Figure 2.2.1 we see the average value of the coefficients of a 1024 step windowed Discrete Fourier Transform of the clarinet tone. We see peaks at the frequencies 0.031, 0.092, 0.160 and 0.221 rad/sample. A way to split the signal $\mathbf{x}$ is by selecting the frequencies that have a high coefficient in the Discrete Fourier Transform of the clarinet

[1] The McGill university master samples are a universally obtainable set of recordings of all conventional musical instruments from western musical culture. These serve as a benchmark collection.

tone, and creating a signal $\mathbf{y}$ with only these frequencies, leaving the rest to be $\mathbf{y^C} = \mathbf{x} - \mathbf{y}$. We expect the clarinet to have high Fourier coefficients at the uneven harmonics only, but for general purpose we filter all harmonics above the fundamental frequency of the clarinet into the signal $\mathbf{y}$ (Figure 2.2.1 also motivates this choice). To do this, we have created a MATLAB function `harmdat` (see Section 4.2.1), which takes three parameters: ϕ_{low}, ϕ_{high} and N. It creates data for a filter that filters the frequencies that have a distance which is less than $\frac{1}{2}(\phi_{\text{high}} - \phi_{\text{low}}) =: \frac{1}{2}\Delta\phi$ to the frequency $\phi = \frac{1}{2}(\phi_{\text{low}} + \phi_{\text{high}})$ and its harmonics. More precisely, if we denote the sample rate by ϕ_s (in Hz), `harmdat` creates vector $\mathbf{w}^{\#} = (w^{\#}[0], w^{\#}[1], ..., w^{\#}[N-1]) \in \{0,1\}^N$, where $w^{\#}[k] = 1$, if $k\frac{\phi_s}{2\pi} \in \left[q\phi - \frac{1}{2}\Delta\phi, q\phi + \frac{1}{2}\Delta\phi\right]$ where $q \geq 1$ integer, and $w^{\#}[k] = 0$ otherwise.

The vector $\mathbf{w}^{\#}$ is used in the MATLAB function `filterclar`, which goes through the signal with steps of $\frac{6}{8}N$ samples, calculates at position n the Discrete Fourier Transform of $(x[n], x[n+1], ..., x[n+N-1])$, multiplies this Discrete Fourier Transform componentwise with $\mathbf{w}^{\#}$ and applies the inverse DFT resulting in $\mathbf{z} = (z[0], z[1], ..., z[N-1])$. From $\mathbf{z}$ it copies $\left(z[\frac{1}{8}N], z[\frac{1}{8}N+1], ..., z[\frac{7}{8}N-1]\right)$ to $\left(y[n+\frac{1}{8}N], y[n+\frac{1}{8}N+1], ..., y[n+\frac{7}{8}N-1]\right)$. Using only a part of $\mathbf{z}$ is necessary to prevent clicks caused by the *Gibbs phenomenon*: we have seen in Section 1.4 that for an ideal low pass filter Gibbs oscillations occur at places where a jump between the values of successive samples occurs. Our filter $\mathbf{w}$ can be seen as a sum of ideal low pass filters as follows: we can describe the transfer function as a sum of blocks, which have edges $\xi_0, \xi_1, ..., \xi_{2M+1}$ and $-\xi_0, -\xi_1, ..., -\xi_{2M+1}$ for a certain integer M. If we consider a filter that has only the two blocks with edges ξ_{2m}, ξ_{2m+1} and $-\xi_{2m}$, $-\xi_{2m+1}$, we see that its transfer function equals

$$w_m^{\#}(\omega) = \begin{cases} 1 & \text{if } \xi_{2m} \leq |\omega| < \xi_{2m+1} \\ 0 & \text{otherwise.} \end{cases}$$

which can be rewritten to $w_m^{\#}(\omega) = g_{\xi_{2m+1}}^{\#}(\omega) - g_{\xi_{2m}}^{\#}(\omega)$. We see that for the filter $w^{\#}(\omega) = \sum_{m=0}^{M} w_m^{\#}(\omega)$ we may expect oscillations around time points where the samples vary much. This is typically the case at the window edges, since we used a rectangular window. The oscillations can be heard as clicks, and therefore we cannot use the left and the right part of the vector $\mathbf{z}$.

Fragment 2.2.1 is the mixture of a clarinet and a viola sound. Applying the algorithm to

this fragment with $N = 16384$, $\phi_{\text{low}} = 210Hz$ and $\phi_{\text{high}} = 230Hz$ renders Fragments 2.2.2 and 2.2.3. We hear that Fragment 2.2.2 sounds more as a clarinet, whilst Fragment 2.2.3 sounds more as a viola. Thus we have as a first result a splitting of the signal into a clarinet like signal and a viola like signal. This also works when we focus on the viola and choose $\phi_{\text{low}} = 222Hz$ and $\phi_{\text{high}} = 242Hz$ (Fragments 2.2.4 and 2.2.5).

2.3 Note recognition

The data that are calculated by the MATLAB function `harmdat` are specific for one certain note. This means that when we want to filter an A, we cannot use the same filter as when we want to filter a B. If we want to filter the sound of a clarinet from the signal, we need somehow to determine which filter to apply. For this purpose we constructed a note recognition program.

A straight-forward way to determine the note that is being played is by applying the STFT and then checking which coefficient has the highest modulus. This is what the MATLAB function `detfreq` (see Section 4.3.1) does. It takes as input a piece of N samples of the signal `x`, multiplies this by a N-point Hanning Window and calculates the DFT. It then returns the frequency of the bin that contains the element with the biggest modulus. The higher N, the better the frequencies are localised in the frequency domain. `detfreq` returns 0 if the largest coefficient in the DFT is less than $0.001N$. Depending on the loudness of the signal, this minimum value could be altered and is introduced to prevent small noise to be interpreted as a note.

To determine the different notes of a piece of music, we have created the MATLAB function `notes` (see Section 4.3.2). There is a loop variable n that increases with a step size of Δn samples, and lets `detfreq` measure the main frequency ξ of the part $(x[n], x[n+1], ..., x[n+N-1])$, where $N > 0$ constant. We shall denote the result of this approximation by $\phi_{\text{fc}} \simeq \xi$.

The function `notes` has the exact frequencies of the notes C3 to C4 in its database, therefore it needs to normalise the measured frequency to the third octave. This is done by searching for an $q \in \mathbb{Z}$ such that $131 \leq 2^{3-q}\phi_{\text{fc}} < 262$. It then takes the note from the database that has a frequency nearest to the calculated frequency and multiplies its frequency by 2^{q-3} to restore it to its original octave.

We have applied the function `notes` to a piece of piano music[2] (Fragment 2.3.1), using several parameters. We have taken a step size $\Delta n = 1024$, and several precisions N. It turns out that when we take N low (4096, Fragment 2.3.2), the rhythm is good but frequencies are not localised well enough. Low frequencies are not localised at all, since the smallest frequency that can be localised is $\frac{2\pi}{1024}$ radians per sample, which is $\frac{44100}{1024} \approx 43$ Hz. If however we enlarge N to 16384 (Fragment 2.3.3), the frequency localisation becomes better, but the rhythm becomes distorted. This agrees with what we expect from the inverse proportionality of time and frequency resolution.

2.4 Splitting a real piece of music

To filter the sound of a clarinet from a real piece of music, we combine the note recognition algorithm with the clarinet filter algorithm. We have implemented this in the MATLAB function `filterpiece` (see Section 4.4.1). It uses a rectangular window of size N, and goes linearly through the signal $\mathbf{x}$ with counter n, which starts at $n = 1$ and takes steps of $\frac{6}{8}N$. The function creates two signals, $\mathbf{y_1}$ and $\mathbf{y_2}$, where $\mathbf{y_1} = (y_1[n])_n$ contains the clarinet part and $\mathbf{y_2} = (y_2[n])_n$ the rest.

One step of the iteration process is done as follows: the DFT of $(x[n], x[n+1], ..., x[n+N-1])$ is calculated, the result of which shall be denoted by $\mathbf{z}^\# = \big(z^\#[0], z^\#[1], ..., z^\#[N-1]\big)$. It then takes the index j that belongs to the item in $\mathbf{z}^\#$ with biggest absolute value, and considers it as the fundamental frequency of the clarinet tone. For for each $l \in \mathbb{Z}$ it takes the eleven elements with indices centred around jl, and puts them into $\mathbf{z_1^\#} \in \mathbb{R}^N$, and the rest into $\mathbf{z_2^\#} \in \mathbb{R}^N$, such that if an element $z_1^\#[k]$ of $\mathbf{z}^\#$ is nonzero, then $z_1^\#[k] = z^\#[k]$, and likewise for nonzero elements in $\mathbf{z_2^\#}$.

The inverse DFT is then calculated of $\mathbf{z_1^\#}$ and $\mathbf{z_1^\#}$, resulting respectively in $\mathbf{z_1} \in \mathbb{R}^N$ and $\mathbf{z_2} \in \mathbb{R}^N$. The values $\big\{z_1[m] \mid m = \frac{1}{8}N, ..., \frac{7}{8}N - 1\big\}$ are then copied into the result signals $\mathbf{y_1}$ and $\mathbf{y_2}$, such that $y_1[n+m] = z_1[m]$ and likewise for $\mathbf{z_2}$.

We have applied the algorithm to a piece of music[3] (Fragment 2.4.1). Using a small window ($N = 4096$) renders Fragment 2.4.2 as the clarinet part, and Fragment 2.4.3 as the piano

[2] Fragment 2.3.1 is a fragment of a musical piece called 'Canto Ostinato', which is composed by Simeon ten Holt and performed by Kees Wieringa and Polo de Haas.

[3] Fragment 2.4.1 is a fragment of the 'Hillandale Waltzes' by Victor Babin. The performers are Murray Khouri (clarinet) and Rosemary Barnes (piano).

part. Using a bigger window ($N = 16384$) renders Fragments 2.4.4 and 2.4.5. We hear that the bigger window renders a less distorted sound, but is to big for the roulades. This brought us to the idea to slow down the sound, such that the roulades are played less fast.

2.5 Time-stretching

To enhance the analysis of a sound signal further, we stretch the duration of the sound, without altering its pitch. This procedure is called *time-stretching* and can be done through the use of a *phase vocoder*, which was first introduced in [7]. A *tracking phase vocoder* is a special instance of a phase vocoder.
Although the phase vocoder is widely used in audio processing, it is hard to find a good description of its implementation (see also [4]). We have found a good description in [31], p. 237 ff., which we shall mainly follow. Our implementation of the tracking phase vocoder has been inspired by [21].
Fragments 2.5.1 to 2.5.6 demonstrate the application of the phase vocoders.

2.5.1 The phase vocoder

The phase vocoder is a program that goes through the input signal $\mathbf{x}$ with counter n_x and steps of size Δn_x, creating the output signal $\mathbf{y}$ on the run, with counter n_y and using steps of size Δn_y. The ratio $\frac{\Delta n_y}{\Delta n_x}$ determines the amount of time-stretching.
When the phase vocoder is at position n_x, it takes a number of subsequent samples centred around n_x and multiplied componentwise by a window $\mathbf{w}$ which is symmetric and real. Such piece of the signal is called a *grain*. The phase vocoder then calculates the DFT from this grain, and stores the amplitude information in $A_1[k]$ and the phase information in $\phi_1[k]$. The phase vocoder does the same with a grain centred around $n_x + \Delta n_x$, resulting in the amplitudes $A_2[k]$ and phases $\phi_2[k]$, where the phases are *unwrapped*, an operation that we will describe in Section 2.5.2.
The phase vocoder can now construct a part of the resulting signal

$$(y[n_y], y[n_y + 1], ..., y[n_y + \Delta n_y - 1]) .$$

It does this by linear interpolating (or extrapolating) the amplitudes and phases, and

constructing a cosine with these amplitudes and phases.
Thus, let $\Delta\phi[k] = \frac{\phi_2[k]-\phi_1[k]}{n_x}$ and $\Delta A[k] = \frac{A_2[k]-A_1[k]}{n_y}$, $A[k]$ and $\phi[k]$ the phases at position n_y, which are calculated earlier or taken arbitrary initially. The phase vocoder calculates

$$y[n_y + n] = \frac{1}{2}\sum_{k=0}^{\frac{1}{2}N-1} (A[k] + n\Delta A[k])\cos(\phi[k] + n\Delta\phi[k])$$

for $n = 0, 1, ..., \Delta n_y - 1$. Now it can calculate $A[k]$ and $\phi[k]$ for the next cycle by $A^*[k] = A[k] + \Delta n_y \Delta A[k]$, $\phi_y[k]^* = \phi_y[k] + \Delta n_y \Delta\phi[k]$, and start the next cycle.

2.5.2 Technical details

Window requirements There are three important details to consider for the phase vocoder. First we want the window $\mathbf{w}$ to be real and symmetric around 0. This is important because then the window $\mathbf{w}$ has a DTFT that is also real and symmetric around 0. This implies that if we apply the window to a phasor $\mathbf{y} = \left(\mathrm{e}^{\mathbf{i}(\xi n+\xi_0)}\right)_{n=-\infty}^{\infty}$, which has a phase ξ_0 at $n = 0$, we have

$$(yw)^{\#}(\omega) = \frac{1}{2\pi} y^{\#} * w^{\#}(\omega) = \mathrm{e}^{\mathbf{i}\xi_0}\int_{-\pi}^{\pi} \delta(\zeta - \xi) w^{\#}(\omega - \zeta)\, d\zeta = \mathrm{e}^{\mathbf{i}\xi_0} w^{\#}(\omega - \xi).$$

If we require that $w^{\#}(0) = \sum_n w[n] > 0$ (which is true for all popular windows), and if we take ω sufficiently close to ξ, the phase of $w^{\#}(\omega - \xi)$ equals 0 and we can measure the phase of $y[0]$ by the phase of $(yw)^{\#}(\omega)$.
FFT shifting If we calculate the DFT of a grain $g[n] = x[n]w[n]$ of size N, where $n = -\frac{1}{2}N, -\frac{1}{2}N + 1, ..., \frac{1}{2}N - 1$, we want it to be

$$g^{\#}[k] = \sum_{n=-\frac{N}{2}}^{\frac{N}{2}-1} g[n]\mathrm{e}^{-\mathbf{i}nk2\pi/N}, \tag{2.5.1}$$

whilst the DFT is normally implemented as

$$g^{\#}[k] = \sum_{n=0}^{N-1} g[n]\mathrm{e}^{-\mathbf{i}nk2\pi/N}.$$

This can be solved by considering g as a periodic signal, and rewriting (2.5.1) to

$$g^{\#}[k] = \sum_{n=0}^{\frac{N}{2}-1} g[n]\mathrm{e}^{-\mathbf{i}nk2\pi/N} + \sum_{n=\frac{N}{2}}^{N-1} g[n-N]\mathrm{e}^{-\mathbf{i}(n-N)k2\pi/N} = \sum_{n=0}^{N-1} g[n]\mathrm{e}^{-\mathbf{i}nk2\pi/N}.$$

On implementation level, this comes down to swapping the left half of the grain and the right half before applying the DFT. This operation is called *FFT shifting* and is implemented in MATLAB by the function `fftshift`.

Phase unwrapping When we calculate the phase ϕ of a complex number, we get a value modulo 2π. If we require that $\phi \in (-\pi, \pi]$, it is called the *principal argument* of the phase (notation: princargϕ). When we measure the phases $\phi_1[k]$ and $\phi_2[k]$, we only get the arguments modulo 2π, which means that the exact phase difference between the two is lost. We can regain this difference by *phase unwrapping*. An illustration of the phenomenon is given in Figure 2.5.1.

Let $\tilde{\phi}_1[k]$ be the principle argument of $\phi_1[k]$, and $\tilde{\phi}_2[k]$ be the principle argument of $\phi_2[k]$. We are interested in the precise difference $\phi_2[k] - \phi_1[k]$. Assume that $\phi_1[k] = \tilde{\phi}_1[k]$. Let $\psi[k]$ be the expected phase at t_2 in this bin, which is $\psi[k] = \phi_1[k] + \frac{k2\pi}{N}(t_2 - t_1)$, and let us assume that $|\psi[k] - \phi_2[k]| < \pi$. Because we have $\phi_2[k] = \tilde{\phi}_2[k] + l2\pi$ for a certain integer l, we have

$$\psi[k] - \phi_2[k] = \psi[k] - \tilde{\phi}_2[k] - l2\pi = \text{princarg}(\psi[k] - \tilde{\phi}_2[k]),$$

which implies that

$$l2\pi = \psi[k] - \tilde{\phi}_2[k] - \text{princarg}(\psi[k] - \tilde{\phi}_2[k])$$

and therefore

$$\phi_2[k] = \psi[k] - \text{princarg}(\psi[k] - \tilde{\phi}_2[k]).$$

Notice that since $\text{princarg}(-\theta) = -\text{princarg}(\theta)$ for all $\theta \neq \pi \mod 2\pi$, this can be rewritten to

$$\phi_2[k] = \psi[k] + \text{princarg}(\tilde{\phi}_2[k] - \psi[k]). \tag{2.5.2}$$

With (2.5.2) we can unwrap the phases $\phi_2[k]$ as long as $|\psi[k] - \phi_2[k]| < \pi$. If we measure a phasor with frequency ξ, we have $\phi_2[k] = \phi_1[k] + (t_2 - t_1)\xi$ and require that $|(\psi - \phi_2)[k]| = (t_2 - t_1)\left|\frac{k2\pi}{N} - \xi\right| < \pi$ or

$$\left|\frac{k2\pi}{N} - \xi\right| < \frac{\pi}{t_2 - t_1}$$

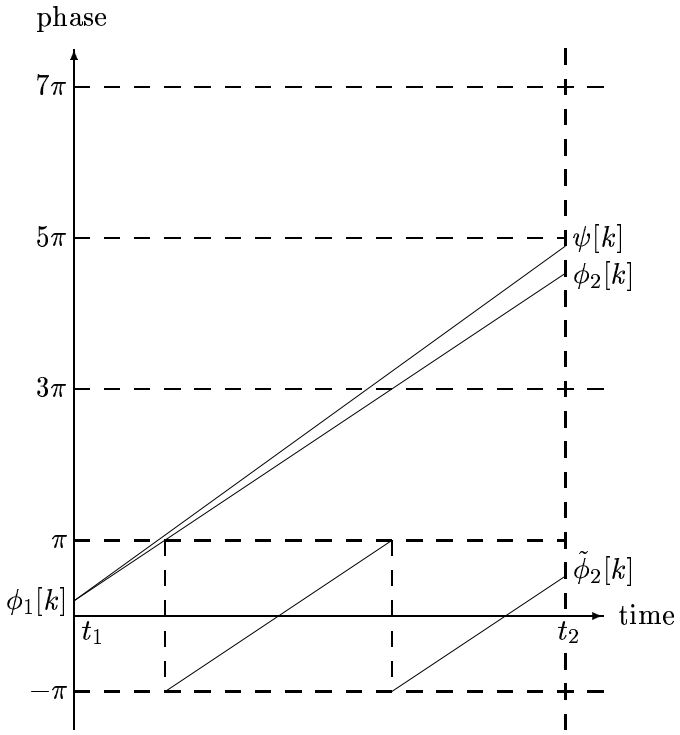


Figure 2.5.1: Phase unwrapping

to ensure the correct unwrapping of $\tilde{\phi}_2$. We find that this is true for $\frac{N}{t_2 - t_1}$ bins, and the smaller $t_2 - t_1$ the more phases are unwrapped correctly.

2.5.3 Implementation

In Section 4.5.1 the listing of the MATLAB function `pv`, which is our implementation of the phase vocoder, is given. Next to the input signal **x** it needs the following arguments:

1. N: the window size. A Hanning window is used. A typical value for N is 2048 at a sampling rate of 44100 Hz.

2. `overlap`: the overlap variable, which takes values between 0 and 1. This determines the overlap, and thereby the distance, between two successive grains. The distance between two successive grains is given by $\Delta n_x = (1 - \texttt{overlap})N$. A typical value

is 0.75.

3. `stretch`: the stretch factor. A stretch factor of 2 doubles the length of the signal.

2.5.4 The tracking phase vocoder

The traditional implementation of the phase vocoder does not consider the nature of the input signal too much. For example, very often adjacent bins have the same phase difference, because the phase differences are induced by the same partial (which is a result of spectral smearing). A tracking phase vocoder takes this in account and tries to dissect the signal in its partials, by matching the DFT bins that belong to the same partial of the sound.

The tracking phase vocoder goes through the signal, and measures the partials of successive grains. This is done by applying a *peak detection* algorithm, which determines the frequencies, amplitudes and phases of the main partials. Peak detection will be further explained in Section 2.5.5.

The old partials are kept in memory, and the new partials are matched to the old ones. To match a new partial to an old one, the following algorithm is applied: for each newly found partial an old partial is searched. The old partial that has a frequency that is (within certain boundaries) nearest to the frequency of the new partial is assigned to the new partial. When the old partial has already been assigned to another new partial, it becomes assigned to the one that has a frequency nearest to it, whilst no other old partial is assigned to the other new partial.

The assignment between an old partial and a new one is called a *match*. When a partial is not matched, it is matched virtually to a partial that has the same frequency but zero amplitude. The reconstruction is done by interpolation of the amplitudes and phases. The phases are interpolated linearly, as is the case by the phase vocoder. For the amplitudes however we apply a slightly different interpolation method: when we reconstruct a partial between time points n_1 and n_2, with respective amplitudes A_1 and A_2, we calculate for $n_1 \leq n < n_2$:

$$A(n) = A_1 \left(1 - \sin^2\left(\frac{\pi(n-n_1)}{2(n_2-n_1)}\right)\right) + A_2 \sin^2\left(\frac{\pi(n-n_1)}{2(n_2-n_1)}\right).$$

This to prevent discontinuities of the derivative of $A(n)$ in A_1 and A_2, which can be heard as artifacts.

2.5.5 Peak detection

In the windowing section we have seen that the peaks in the frequency spectrum of a windowed signal correspond to the frequencies and amplitudes of the phasors that make up the signal. This means that we can find these properties by looking for the peaks in the DTFT. This procedure is called *peak detection*, and we have implemented two versions of it.

In both versions we have applied *zero-padding*: we append Z zeros to a grain $\mathbf{x}$ of size N, resulting in a grain $\mathbf{x}_Z$ of size $N+Z$. To maintain window symmetry, $\frac{1}{2}Z$ zeros are inserted in front of $\mathbf{x}$ and $\frac{1}{2}Z$ zeros are appended at the end of $\mathbf{x}$. We see that the DFT of $\mathbf{x}_Z$ equals the DTFT of $\mathbf{x}$ calculated at frequency points $\left\{\frac{2\pi k}{N+Z}\right\}_{k=0}^{N+Z-1}$. Thus we can use the FFT algorithm to compute the DTFT at $N+Z$ different frequency points instead of only N frequency points.

Zero-padding allows us to consider the spectrum more precise. Notice that it does not increase the frequency resolution, since the size of the window and therefore its bandwidth is not changed. However, if we have a peak at frequency ξ, we can expect to locate it better, since we have a bigger chance of having calculated the DTFT of a frequency nearer to ξ. To make sure that this is true for all possibilities of ξ, we should take $Z = \nu N$ with ν integer, because then $x^{\#}[k] = x_Z^{\#}[(\nu+1)k]$ for $k = 0, 1, ..., N-1$.

Let us denote a phasor in $\mathbf{x}$ by $Ae^{\mathbf{i}(\xi n+\xi_0)}$. To find A, ξ and ξ_0, we look for a k such that $|x_Z^{\#}[k]| \geq |x_Z^{\#}[k+1]|$ and $|x_Z^{\#}[k]| > |x_Z^{\#}[k-1]|$. Because we use a real and symmetric window we can measure ξ_0 by the phase of $x_Z^{\#}[k]$. We compute an initial estimate $\widetilde{\zeta} = k\frac{2\pi}{N+Z}$ of ξ and refine this estimate in the same way as by the phase vocoder: let $\mathbf{y}_Z^{\#}$ be the DFT of the previous zero-padded grain. We compute the phase difference between $x_Z^{\#}[k]$ and $y_Z^{\#}[k]$ and divide by the number of samples between the grains, resulting in a new estimate $\widetilde{\xi}$. If we choose Z big enough we can approximate the amplitude A by $|x_Z^{\#}[k]|$. In Section 4.6.3 we see our MATLAB implementation `trpeak` of this algorithm.

Another way to refine the estimate $\widetilde{\zeta} =: \zeta_1$ is by the use of a second estimate ζ_2. We choose $\zeta_2 = \kappa\frac{2\pi}{N+Z}$, with κ equal to either $k-1$ or $k+1$, such that $|x_Z^{\#}[\kappa]|$ maximal. This ensures

that ξ is between ζ_1 and ζ_2. We measure ξ_0 by the phase of $x^{\#}(\zeta_1)$. If we denote the DTFT of the window that we use by $w^{\#}$, we acquire the following system of equations for A and ξ :

$$\begin{cases} |x^{\#}(\zeta_1)| &= Aw^{\#}(\zeta_1 - \xi) \\ |x^{\#}(\zeta_2)| &= Aw^{\#}(\zeta_2 - \xi). \end{cases} \tag{2.5.3}$$

We use a Hanning window

$$w[n] = \begin{cases} \cos^2\left(\frac{n\pi}{N}\right) & \text{if } -\frac{1}{2}N + 1 \leq n \leq \frac{1}{2}N - 1 \\ 0 & \text{otherwise,} \end{cases}$$

which has a DTFT equal to $w^{\#}(\omega) = \frac{1}{4}r^{\#}\left(\omega - \frac{2\pi}{N}\right) + \frac{1}{4}r^{\#}\left(\omega + \frac{2\pi}{N}\right) + \frac{1}{2}r^{\#}(\omega)$, where

$$r^{\#}(\omega) = \frac{\sin\frac{1}{2}(N-1)\omega}{\sin\frac{1}{2}\omega}$$

(see also Section 1.6.1). Because the root mean square bandwidth of a Hanning window is two bins, both $x^{\#}(\zeta_1)$ and $x^{\#}(\zeta_2)$ are unequal to zero. We can therefore simplify (2.5.3) to

$$|x^{\#}(\zeta_1)|w^{\#}(\zeta_2 - \xi) - |x^{\#}(\zeta_2)|w^{\#}(\zeta_1 - \xi) = 0.$$

This equation can be solved numerically, which we have done in the MATLAB function `trbisectpeak` (see Section 4.7.2).

Fragment 2.5.2 and Fragment 2.5.3 are results of the application of the tracking phase vocoder to Fragment 2.4.1, with respectively the phase unwrapping peak detection algorithm and the bisection peak detection algorithm. Similarly, Fragment 2.5.5 and 2.5.6 are results of applying the tracking phase vocoder to Fragment[4] 2.5.4.

2.5.6 Guides

Now that the signal is split into partials at each time point, and matched to the partials at the previous time point, we can follow the partials over time. This approach was first introduced in [17] and is done by maintaining *guides* that contain the evolution of the partials' frequencies and amplitudes over time. For a non-matched new partial a guide

[4]Fragment 2.5.4 is a fragment of W.A. Mozart's *Requiem*, performed by the Slovak Philharmonic Orchestra and Chorus, directed by Zdeněk Košler.

is created, whilst matched partials are assigned to the guide belonging to the old partial that the new partial was matched with. When a guide is not matched, it is finished and its contents are written to the output signal **y**.

2.5.7 Technical details

The partials from the current measurement and the one before are maintained in a matrix `tracks`, which has $\frac{1}{2}N + 1$ rows and seven columns. When the current time pointer is t_3, the previous one t_2 and the one before that t_1, the structure of one row in the matrix `tracks` is as follows:

Column	Contents
1	frequency between t_1 and t_2
2	amplitude at t_2
3	phase at t_2
4	frequency between t_2 and t_3
5	amplitude at t_3
6	phase at t_3
7	matched bin

A separate matrix `lines` that has $\frac{1}{2}N + 1$ rows is maintained to track the partials in time. Each row contains the following columns:

Column	Contents
1	start point in the time domain
2	number of partials
3	finished flag
4	phase of partial 1
5	frequency between partial 0 and partial 1
6	amplitude of partial 1
7	frequency between partial 1 and partial 2
8	amplitude of partial 2
9	frequency between partial 2 and partial 3
10	amplitude of partial 3
⋮	⋮

The start point in samples can be found by multiplying column one by Δn_y. A row in `lines` is available when column two has value zero. To make the connection between the matches in `tracks` and the spectral lines in `lines` a vector `xlat` with $\frac{1}{2}N + 1$ elements is maintained that links the row number in the `tracks` matrix to the row number in the `lines` matrix.

2.6 Splitting using the phase vocoders

We have modified the tracking phase vocoder to meet our purpose: the splitting of a real piece of music into a clarinet part and the rest. In these sections we will describe how. Fragments 2.6.1 to 2.6.11 demonstrate the application of the techniques.

2.6.1 A modification of the tracking phase vocoder

As a first approach, we modify the tracking phase vocoder without the guides to create two output signals $\mathbf{y_1}$, $\mathbf{y_2}$, where $\mathbf{y_1}$ contains the principle tones, and $\mathbf{y_2}$ the rest. A selection is made in the reconstruction phase, where the principal partial and its harmonics are reconstructed into $\mathbf{y_1}$, whilst the other partials are reconstructed into $\mathbf{y_2}$.
Fragment 2.6.1 ($\mathbf{y_1}$) and Fragment 2.6.2 ($\mathbf{y_2}$) are results of the application of the modified tracking phase vocoder to Fragment 2.4.1. We hear that $\mathbf{y_1}$ mainly follows the clarinet line, whilst $\mathbf{y_2}$ contains the rest. However, we also notice that $\mathbf{y_1}$ has disturbances when the piano attacks, which can be explained by the fact that during a piano attack the piano is louder than the clarinet for a short while, and therefore the principal tone is a piano tone instead of a clarinet tone.

2.6.2 Following the guides

To reduce the effect of the disturbances created by the piano attacks, we change the selection process of the partials. First we insert a column in the `lines` matrix that contains a *clarinet counter*. We create this column at position 4 and shift the rest of the columns one position to the right. For every grain we still make a selection which partial is considered as a clarinet tone, and which is not, but instead of directly writing the partial to either $\mathbf{y_1}$ or $\mathbf{y_2}$, we increase the clarinet counter if the partial is considered to be the principal partial

or one of its harmonics. When a guide ends the spectral line it contains is reconstructed either into $\mathbf{y_1}$ if the clarinet counter is more than a (user defined) fraction of the line's length, and in $\mathbf{y_2}$ otherwise.

Fragments[5] 2.6.3 to 2.6.11 demonstrate the modified tracking phase vocoder with a clarinet counter. We hear that the disturbances of $\mathbf{y_1}$ by the piano are reduced and that it mostly follows the clarinet line of the piece of music.

[5]Fragment 2.6.7 is a piece of the 'Jamaican Rumba' by Arthur Benjamin. It is performed by Murray Khouri (clarinet) and Rosemary Barnes (piano).

Chapter 3
Conclusions and topics for further research

In this small concluding chapter we will have a critical look at our modification of the tracking phase vocoder with guides. We denote this modified phase vocoder by 'the proposed method', and restate it in the following section:

3.1 The proposed method

In the Chapter 2 we have seen how during our research a method to separate musical signals arose. We can now state this method as follows:

The proposed method

Input:
$\mathbf{x}$: input signal
Δn: step size in the time domain

Output:
$\mathbf{y_1}$: clarinet signal
$\mathbf{y_2}$: non-clarinet signal

for k=0, 1, 2, ... **do**
measure amplitudes, frequencies and phases at time point $n_k = k\Delta n$
determine the fundamental frequency ξ of the clarinet tone
mark the partials with frequencies that are (approximately) an integer multiple of ξ as clarinet partials
match the partials with the guides
if a guide G is finished **then**
if more than a fraction p of the partials in G are marked as clarinet partials **then**
synthesise the content of G into the clarinet output signal $\mathbf{y_1}$
else
synthesise the content of G into the non-clarinet output signal $\mathbf{y_2}$
end if
remove G from memory
end if
end for
synthesise the content of the remaining guides into the output signals

We see that the algorithm is composed of four components:

- the measurement of the parameters of the partials that make up the signal;
- the identification of which partials are clarinet partials;
- the book-keeping of the guides;
- the reconstruction of the content of the guides.

With this algorithm we have been able to separate the clarinet line from real pieces of music as can be heard in Fragments 2.6.3 to 2.6.11. We think that our algorithm is a step in the right direction, and it can be refined by further research.

3.2 Spectral lines

The proposed method is based on the measurements of spectral lines. As long as the spectral lines are situated far enough in the frequency domain, they can be separated and

the method can be expected to work. In pieces where the frequencies of the spectral lines are too close to each other, large windows are needed which means too much decrease in time resolution. This happens mostly in pieces where many musical instruments are playing. On the other hand, applying the tracking phase vocoder to a record of an orchestra gives an acceptable result, although artifacts can be heard. This means that the tracking phase vocoder is still able to distinguish most spectral lines, the rest resulting in the artifacts.

A problem arises when the harmonics of different tones coincide. Our method will not be able to distinguish between these frequencies and therefore is bound to take away parts of other instruments. This problem will often occur, since tones with coinciding harmonics sound very well, and therefore many compositions contain tones that have such a relation.

3.3 Wavelets

During our research, we have often modified our programs such that they use the Discrete Wavelet Transform instead of the Short-Time Fourier Transform. However, these experiments rarely improved the methods. When the wavelet implementation resulted in an improvement, often the same improvement could be accomplished by a slight modification in the STFT version of the program.

A clue to the reason that wavelets have not shown to be useful for our methods can be seen in the scalograms in Section 2.1. The widths of the spectral lines of the different harmonics are different, and for high frequencies too much interference occurs and the harmonics become invisible. This has not to be a problem in all cases; for example a note recognition algorithm that implements a harmonic sieve (see [13]) with Gabor wavelets can exploit the bigger bandwidth at the higher frequencies.

3.4 Topics for further research

It would be convenient when the selection of the main instrument in the proposed method was enhanced: our method considers the spectral line with highest amplitude as the fundamental frequency of the main instrument, which is often the case, but not always. There even exist instruments that lack a fundamental frequency.

Also, selecting the spectral lines means that we only look at the harmonic component of a sound: a sound generally consists of a *harmonic component*, which can be seen as the spectral lines, and a *transient component*, which occurs mostly during the attack of the instrument. We expect that Fourier analysis will not be useful for the analysis of the transient component, because such a signal consists for a greater part of (structured) noise. Wavelets could be useful, in particular when they do not have a direct connection to an STFT of some sort.

The proposed method can be enhanced by applying a more intelligent instrument recognition algorithm. We think of measuring different properties of a tone, such as whether it falls within a certain frequency range, its attack time and the amount of energy in the harmonics. There are known instrument recognition algorithms that are based on pattern recognition in different characteristics of a sound that work with high accuracy. We expect that a similar algorithm can be implemented in the proposed method.

The transient component can also be used to determine which instrument is playing. When the transient component is used to determine the instrument, it should also be used in the construction of the output signals, by adding it either to the filtered signal, or to the residue.

3.5 Conclusion

In this thesis we propose a method to separate the clarinet line from a recording of music. We have seen how the method was developed and how it works. The method is a step into the direction of the solution of the problem to extract one particular instrument from a recording of multiple instruments. As we have pointed out, there are various possibilities to enhance our algorithm, both to obtain a better note recognition algorithm and to improve the quality of the separation. Suggestions for further research have been given.

Chapter 4
Listings

4.1 Spectrograms and Scalograms

4.1.1 spectrogram.m

```
% function spectrogram(x, N, T, zpad, maxfreq)
%
% creates a spectrogram plot of signal x using Fourier analysis and
% a Hanning window.
%
% x       : input signal
% N       : windowsize (must be even)
% T       : amount of time points
% zpad    : zero-padding argument: number of zeros to pad
% maxfreq : maximal frequency to show (rad/sample)
%
% copyright (C) 2005 Bert Greevenbosch

function spectrogram(x, N, T, zpad, maxfreq)

  x=torow(x);
  w=sin((0:N-1).*pi/N).^2; % hanning window
  ds=(length(x)-N-1)/T;
  s=N/2+1;
  mf=ceil((N+zpad)*maxfreq/(2*pi));
  im=zeros(N/2+1, mf+1);

  for q=1:T
    t=round(s);
    f=fft([x(t-N/2:t+N/2-1).*w, zeros(1, zpad)]);
    f=f(1:mf+1)';
    f=f.*conj(f);
    y(:, q)=f;
    s=s+ds;
  end

  setigpal
  imagesc([0,length(x)],[0, mf*2*pi/(N+zpad)],y);
  axis('xy');
  xlabel('time (sample)');
  ylabel('frequency (rad/sample)');
```

4.1.2 scalogram.m

```
% function scalogram(x, nu, minfreq, maxfreq, norows, nocols)
%
% plots the scalogram of the signal x, using wavelets based on the hanning
% window.
%
% input:
%
% x       : input signal
% nu      : number of waves in one wavelet
% minfreq : lowest frequency to show
% maxfreq : highest frequency to show
% norows  : number of frequencies to show
% nocols  : number of time points to do the measurements on
%
% copyright (C) 2005 Bert Greevenbosch

function scalogram(x, nu, minfreq, maxfreq, norows, nocols)

  x=tocolumn(x);
  im=zeros(norows, nocols);
  qbanks=minfreq+(1:norows)*(maxfreq-minfreq)/norows;
  Q=Qmatrix(qbanks, nu);
  sz=size(Q);
  N=sz(2);
  step=(length(x)-N)/nocols;
  left =-round(N/2);
  right=N-1+left;
  k=1;

  for n=-left+1:step:length(x)-right
    ni=round(n);
    xw=Q*fftshift(x(ni+left:ni+right));
    im(:, k)=xw.*conj(xw);
    k=k+1;
    last=n;
    disp(sprintf('%1.5f', n/(length(x)-right)));
  end

  hold off
  setigpal
  imagesc([-left+1, last],[minfreq, maxfreq], im);
  axis('xy');
  xlabel('time (sample)');
  ylabel('frequency (rad/sample)');
  size(Q)
```

4.1.3 setigpal.m

```
% function setigpal
%
% creates a gray scale colourmap.
%
% copyright (C) 2005 Bert Greevenbosch

function setigpal

  x=1:-1/255:0;
  x=[x;x;x];
  x=transpose(x);
  colormap(x);
```

4.1.4 Qmatrix.m

```
% function Q=Qmatrix(qbanks, nu)
%
% qbanks : frequencies
% nu     : number of waves
%
% each row k in Q contains a windowed Fourier atom, with a frequency
% qbanks[k], and a window with a size such that nu waves are considered:
% size nu*2*pi/qbanks[k].
%
% copyright (C) 2005 Bert Greevenbosch

function Q=Qmatrix(qbanks, nu)

  minfreq=min(qbanks);
  norows=length(qbanks);
  nocols=round(2*nu*pi/minfreq);

  Q=zeros(norows, nocols);
  disp(sprintf('Creating %d x %d Q matrix', norows, nocols));
  totmx=0;
  for k=1:norows
    f=qbanks(k);
    mx=round(nu*pi/f);
    Q(k, 1:mx)=exp(i*f*(0:mx-1)).*cos((0:mx-1)*pi/(2*mx)).^2/sqrt(2*mx);
    Q(k, nocols-mx+1:nocols)=exp(i*f*(-mx:-1)).*cos((-mx:-1)*pi/(2*mx)).^2/sqrt(2*mx);
    disp(sprintf('%1.5f', k/norows));
    totmx=totmx+mx;
  end
  totmx=totmx*2;
  disp(sprintf('Average length: %d', round(totmx/norows)));

  Q=sparse(Q);
```

4.2 A simple clarinet filter

4.2.1 harmdat.m

```
% function w=harmdat(lfreq, hfreq, N)
%
% creates a  filter  on the basis  of  harmonics.
%
% input:
%
% lfreq  : lower base frequency
% hfreq  : higher base frequency
% N      : size  of the  filter
%
% output:
%
% w      :  filter
%
% copyright (C) 2005 Bert Greevenbosch

function w=harmdat(lfreq, hfreq, N)

  w=zeros(1, N);
  clfreq=lfreq*N/44100;
  chfreq=hfreq*N/44100;
  fq=(clfreq+chfreq)/2
```

```
  dfreq=ceil((chfreq-clfreq)/2)
  k=1;
  while fq*k<N/2+1
    fl =floor(fq*k)-dfreq+1;
    fh=ceil(fq*k)+dfreq+1;
    if(fh>N/2)
      fh=N/2;
    end
    if( fl <1)
      fl =1;
    end
    for f=fl:fh
      if(f<N/2+1)
        w(f)=1;
        w(N-f+1)=1;
      end
    end
    k=k+1;
  end
```

4.2.2 filterclar.m

```
% function [y1, y2]= filterclar (x, w)
%
% clarinet  filter
%
% input:
%
% x  : input signal
% w :  filter
%
% output:
%
% y1 : clarinet  part
% y2 : non-clarinet part
%
% copyright (C) 2005 Bert Greevenbosch

function [y1, y2]= filterclar (x, w)

  N=length(w);
  lw=N/8;
  hg=7*lw;
  y1=zeros(1, length(x));

  if mod(length(x), N)~=0
    t=mod(length(x), N);
    x=[x, zeros(1, N-t)];
  end

  for n=1:hg-lw:length(x)-N+1
    z=fft(x(n:n+N-1));
    a1=z.*w;
    x1=real(ifft(a1));
    y1(n+lw:n+hg-1)=x1(lw+1:hg);
    disp(sprintf('%1.5f', n/length(x)));
  end

  y2=x-y1;
```

4.3 Note recognition

4.3.1 detfreq.m

```
% function f=detfreq(x)
%
% determines the main frequency f of the signal x
% returns frequency in Hz (when x sampled at a rate of 44100 Hz)
%
% copyright (C) 2005 Bert Greevenbosch

function f=detfreq(x);

    N=length(x);
    window=sin((0:N-1)*pi/N)'.^2;

    y=abs(fft(x(1:N).*window));
    y(1)=0;
    [mx, idx]=max(y);
    if(mx>.001*N)
      f=44100*(idx-1)/N;
    else
      f=0;
    end
```

4.3.2 notes.m

```
% function y=notes(x, N, dn)
%
% calculates notes using the discrete Fourier transform (DFT)
%
% input:
%
% x  : input signal
% N  : size of DFT
% dn : stepsize between succesive measurements
%
% output:
%
% y  : notes played as sines
%
% copyright (C) 2005 Bert Greevenbosch


function y=notes(x, N, dn)

   names=['c__'; 'cis'; 'd__'; 'dis'; 'e__'; 'f__'; 'fis'; 'g__'; 'gis'; 'a__'; 'ais'; 'b__'; 'c__']
   freq =[ 131,  139,  147,  156,  165,  175,  185,  196,  208,  220,  233,  247,  262]
   y=zeros(1, length(x));
   maxdif=0;
   phi0=0;

   for n=1:dn:length(x)-N
     fc=detfreq(x(n:n+N-1));
     if fc~=0
       ofc=fc;
       o=3;
       while fc<131
         fc=fc*2;
         o=o-1;
       end
```

```
      while fc>262
        fc=fc/2;
        o=o+1;
      end
      mn=abs(fc-freq(1));
      nt=1;
      for j=2:13
        if(abs(fc-freq(j))<mn)
          mn=abs(fc-freq(j));
          nt=j;
        end
      end
      if nt==13
        o=o+1;
        nt=1;
      end
      fcalc=freq(nt);
      l=o;
      while l>3
        fcalc=fcalc*2;
        l=l-1;
      end
      while l<3
        fcalc=fcalc/2;
        l=l+1;
      end
      if(abs(ofc-fcalc)>maxdif)
        maxdif=abs(ofc-fcalc);
      end
      y(n:n+dn-1)=sin(phi0+2*pi*[0:dn-1]*fcalc/44100)/3;
      phi0=phi0+2*pi*dn*fcalc/44100;
      phi0=mod(phi0, 2*pi);
      disp(sprintf('%s%d', names(nt, :), o));
    else
      y(n:n+dn-1)=sin(phi0)/3;
      disp('rest');
    end
  end

  maxdif
```

4.4 Filtering a clarinet from a real piece of music

4.4.1 filterpiece.m

```
% function [y1, y2]= filterpiece (x, N)
%
% filters clarinet from x into y1 and rest into y2
%
% input:
%
% x : orignal signal
% N : windowsize
%
% output:
%
% y1 : clarinet signal
% y2 : non-clarinet signal
%
% copyright (C) 2005 Bert Greevenbosch
```

```
function [y1, y2]= filterpiece (x , N)

  x=tocolumn(x);

  lw=N/8;
  hg=7*lw;

  y1=zeros(1, length(x));
  y2=y1;

  for n=1:hg-lw:length(x)-N+1
    z=fft(x(n:n+N-1));
    q=abs(z);
    q(1)=0;
    [mx, idx]=max(q);
    if(idx>N/2)
      idx=N+2-idx;
    end
    ener=zeros(1, N);
    a1=zeros(1, N);
    a2=z;
    l=1;
    while(l*idx<N/2+1)
      j1=l*(idx-1)-10+1;
      if(j1<2)
        j1=2;
      end
      j2=l*(idx-1)+10+1;
      if(j2>N/2+1)
        j2=N/2+1;
      end
      for j=j1:j2
        a1(j)=z(j);
        a2(j)=0;
        a1(N+2-j)=z(N+2-j);
        a2(N+2-j)=0;
      end
      l=l+1;
    end
    disp(sprintf('%1.4f', n/length(x)));
    x1=real(ifft(a1));
    x2=real(ifft(a2));
    y1(n+lw:n+hg-1)=x1(lw+1:hg);
    y2(n+lw:n+hg-1)=x2(lw+1:hg);
  end
```

4.5 The phase vocoder

4.5.1 pv.m

```
% function y=pv(x, N, overlap, stretch)
%
% an implementation of the phase vocoder
%
% x       : input signal
% N       : window size
% overlap : overlap factor  between two successive windows (for example .75)
% stretch : amount of time stretching
%
% copyright (C) 2005 Bert Greevenbosch
```

```
function y=pv(x, N, overlap, stretch)

    x=torow(x);
    window=sin((0:N-1)*pi/N).^2;
    dx=round(N*(1-overlap));
    dy=round(dx*stretch);
    stretch=dy/dx;
    disp(sprintf('Real stretching factor: %1.5f' , stretch ));
    y=zeros(round(length(x)*stretch), 1);
    nx=N/2+1;
    ny=round(nx*stretch);
    grain=x(nx-N/2:nx+N/2-1).*window;
    f=fft(fftshift(grain));
    f=f(1:N/2+1);
    phi1=angle(f);
    A1=abs(f);
    A=A1;
    nx=nx+dx;
    dpsi=(0:N/2)*dx*2*pi/N;
    phi=phi1;
    while(nx<length(x)-N/2)
        phi1=princarg(phi1);
        grain=x(nx-N/2:nx+N/2-1).*window;
        f=fft(fftshift(grain));
        f=f(1:N/2+1);
        phi2=princarg(angle(f));
        A2=abs(f);
        psi=phi1+dpsi;
        phi2=psi+princarg(phi2-psi);
        dphi=(phi2-phi1)/dx;
        dA=(A2-A1)/dy;
        for k=0:dy-1
            y(ny)=A*cos(phi)';
            A=A+dA;
            phi=phi+dphi;
            ny=ny+1;
        end
        phi1=phi2;
        A1=A2;
        nx=nx+dx;
        disp(sprintf('%1.5f', nx/length(x)));
    end
    y=y*max(abs(x))/max(abs(y));
```

4.5.2 princarg.m

```
% function phase=princarg(phase_in)
%
% calculates the principal argument of phase_in
%
% taken from Udo Zolzer's book 'DAFX - Digital Audio Effects';
% more detailed information can be found in the bibliography.

function phase=princarg(phase_in)

  phase=mod(phase_in+pi,-2*pi)+pi;
```

4.6 The tracking phase vocoder

4.6.1 trvoc.m

```
% y=trvoc(x, N, Z, overlap, stretch)
%
% input:
%
% x       : input signal
% N       : window size
% Z       : number of zeros for zero padding
% overlap : overlap factor
% stretch : time-stretch factor
%
% output:
%
% y       : time-stretched signal
%
% copyright (C) 2005 Bert Greevenbosch

function y=trvoc(x, N, Z, overlap, stretch)

  global fadein;
  global fadeout;
  global tracks;

  x=torow(x);
  window=sin((0:N-1)*pi/N).^2;
  dnx=round((1-overlap)*N);
  nx=N/2+1+dnx;
  dny=round(dnx*stretch);
  ny=dny;
  dpsi=(0:(N+Z)/2)*dnx*2*pi/(N+Z);
  [A, phi1]=calcAphi(x(1:N).*window, Z);
  tracks=zeros(N/2+1, 6);
  y=zeros(1, length(x)*stretch);
  fadein=-.5*cos((0:dny-1)*pi/dny)+.5;
  fadeout=.5*cos((0:dny-1)*pi/dny)+.5;
  while(nx<length(x)-N/2-dnx)
    [ampl2, dphi, phi2, phases]=trpeak(x(nx-N/2:nx+N/2-1).*window, phi1, dpsi, dnx, Z);
    trmatch(dphi, ampl2, phases, N);
    y(ny-dny+1:ny)=trreconstruct(N);
    nx=nx+dnx;
    ny=ny+dny;
    phi1=princarg(phi2);
    disp(sprintf('%1.5f; %d lines', nx/length(x), length(phases)));
  end;

  y=y*max(abs(x))/max(abs(y));
```

4.6.2 calcAphi.m

```
% function [A, phi]=calcAphi(x, Z)
%
% calculates amplitudes A and frequencies phi of x,
% such that (dft(x))[k+1] = A[k+1] exp(i phi[k+1])
% where the signal x is zero-padded with Z zeros
% Z should be even
%
% copyright (C) 2005 Bert Greevenbosch
```

```
function [A, phi]=calcAphi(x, Z)

    Zh=Z/2;
    N=length(x);
    y=fft(fftshift([zeros(1, Zh), x, zeros(1, Zh)]));
    y=y(1:(N+Z)/2+1);
    A=abs(y);
    phi=princarg(angle(y));
```

4.6.3 trpeak.m

```
% function [ampl2, dphi, phi2, phases]=trpeak(x, phi1, dpsi, dn, Z);
%
% applies peak detection to the signal x, by looking at the phase
% difference
%
% input:
%
% x       : input signal
% phi1    : phases of previous measurements
% dpsi    : frequency between previous measurements and current measurement
% dn      : number of samples between successive measurements
% Z       : number of zeros for zero-padding
%
% output:
%
% ampl2 : measured amplitudes belonging to the peaks
% dphi  : measured frequency between the peaks of the current measurement
%         and the previous measurement
% phi2  : phases of all frequency bins
% phases : phases belongint to the peaks
%
% copyright (C) 2005 Bert Greevenbosch

function [ampl2, dphi, phi2, phases]=trpeak(x, phi1, dpsi, dn, Z);

  N=length(x);
  [A, phi2]=calcAphi(x, Z);
  psi=phi1+dpsi;
  phi2=psi+princarg(phi2-psi);
  dfreq=(phi2-phi1)/dn;
  m=1;
  ampl2=0;
  phases=0;
  for k=2:(N+Z)/2
    if A(k-1)<A(k) & A(k+1)<=A(k) & A(k)>1
      ampl2(m)=A(k);
      dphi(m)=dfreq(k);
      phases(m)=phi1(k);
      m=m+1;
    end
  end
  m=m-1;
  ampl2=ampl2(1:m);
  dphi=dphi(1:m);
  phases=phases(1:m);
```

4.6.4 trmatch.m

```
% function trmatch(dphi, ampl2, phases, N)
%
% matches new partials to old partials
%
% global:
%
% tracks : data structure that contains the old and new partials
%
% input:
%
% dphi   : phase difference between current and last measurement
% ampl2 : amplitudes of current measurement
% phases : phases of current measurement
% N      : grain size
%
% copyright (C) 2005 Bert Greevenbosch

function trmatch(dphi, ampl2, phases, N);

  global tracks;
  binsz=2*pi/N;
  tracks(:, 4:7)=zeros(N/2+1, 4);
  for n=1:length(dphi)
    k=floor(abs(dphi(n)/binsz))+1;
    if(k<=N/2+1)
      tracks(k, 4)=dphi(n);
      tracks(k, 5)=ampl2(n);
      tracks(k, 6)=phases(n);
    end
  end
  for k=2:N/2
    if(tracks(k, 4)~=0)
      dm=abs(tracks(k, 4)-tracks(k-1, 1));
      d0=abs(tracks(k, 4)-tracks(k,   1));
      dp=abs(tracks(k, 4)-tracks(k+1, 1));
      if dm < d0 & dm < dp
        tracks(k, 7)=k-1;
        if tracks(k-1, 7)==k-1
          if abs(tracks(k-1, 4)-tracks(k-1, 1))>abs(tracks(k, 4)-tracks(k-1, 1))
            tracks(k-1, 7)=0;
          else
            tracks(k, 7)=0;
          end
        end
        if k>2
          if tracks(k-2, 7)==k-1
            if abs(tracks(k-2, 4)-tracks(k-1, 1))>abs(tracks(k, 4)-tracks(k-1, 4))
              tracks(k-2, 7)=0;
            else
              tracks(k, 7)=0;
            end
          end
        end
      end
      if d0 <= dm & d0 <= dp
        tracks(k, 7)=k;
        if tracks(k-1, 7)==k
          if abs(tracks(k-1, 4)-tracks(k, 1))>abs(tracks(k, 4)-tracks(k, 1))
            tracks(k-1, 7)=0;
          else
```

```
            tracks(k, 7)=0;
          end
        end
      end
      if dp < dm & dp < d0
        tracks(k, 7)=k+1;
      end
    end
  end
  for k=1:N/2
    if tracks(k, 7)~=0 & tracks(k, 4)~=0 % matched?
      bin=tracks(k, 7);
      if tracks(bin, 1)==0 | tracks(bin, 2)==0 % matched to zero?
        tracks(k, 7)=0;    % not matched
      end
    end
  end
```

4.6.5 trreconstruct.m

```
% function y=trreconstruct(N)
%
% reconstructs the current partials
%
% global: tracks, fadein, fadeout
%
% input:
%
% N : grain size
%
% output:
%
% y : reconstructed grain
%
% copyright (C) 2005 Bert Greevenbosch

function y=trreconstruct(N)

  global tracks;
  global fadein;
  global fadeout;
  dny=length(fadein);
  y=zeros(1, dny);
  phases=zeros(N/2+1, 1);
  for k=1:N/2+1
    bin=tracks(k, 7);
    if bin~=0
      if tracks(k, 5)~=0 | tracks(bin, 2)~=0
        A0=tracks(bin, 2);
        A1=tracks(k, 5);
        phi=(0:dny-1)*tracks(k, 4)+tracks(bin, 3);
        y=y+(A0*fadeout+A1*fadein).*cos(phi);
        tracks(bin, 2)=0; % this bin is done
      end
      phases(k)=dny*tracks(k, 4)+tracks(bin, 3);
    else
      if tracks(k, 5)~=0 % fade in
        A1=tracks(k, 5);
        phi=(-dny:-1)*tracks(k, 4)+tracks(k, 6);
        y=y+A1*fadein.*cos(phi);
        phases(k)=tracks(k, 6); % keep this phase
      end
```

```
    end
  end
  for k=1:N/2+1
    if tracks(k, 2)~=0 % frequency not done yet; fade out
      A0=tracks(k, 2);
      phi=(0:dny-1)*tracks(k, 1)+tracks(k, 3);
      y=y+A0*fadeout.*cos(phi);
    end
  end
  tracks(:, 1:2)= tracks (:, 4:5);
  tracks(:, 3)= princarg(phases);
  for k=1:N/2+1
    if tracks(k, 5)==0
      tracks(k, 1:3)=zeros(1, 3);
    end
  end
```

4.7 The tracking phase vocoder using the bisection method

4.7.1 trbisect.m

```
% y=trbisect(x, N, Z, overlap, stretch)
%
% implementation of the tracking vocoder using the bisection method
%
% input:
%
% x       : input signal
% N       : window size
% Z       : number of zeros for zero padding
% overlap : overlap factor
% stretch : time-stretch factor
%
% output:
%
% y       : time-stretched signal
%
% copyright (C) 2005 Bert Greevenbosch

function y=trbisect(x, N, Z, overlap, stretch)

  global fadein;
  global fadeout;
  global tracks;

  x=torow(x);
  window=cos((-N/2:N/2-1)*pi/N).^2;
  dnx=round((1-overlap)*N);
  nx=N/2+1+dnx;
  dny=round(dnx*stretch);
  ny=dny;
  tracks=zeros(N/2+1, 6);
  y=zeros(1, length(x)*stretch);
  fadein=-.5*cos((0:dny-1)*pi/dny)+.5;
  fadeout=.5*cos((0:dny-1)*pi/dny)+.5;
  while(nx<length(x)-N/2-dnx)
    [ampl2, dphi, phases]=trbisectpeak(x(nx-N/2:nx+N/2-1).*window, Z);
    trmatch(dphi, ampl2, phases, N);
    y(ny-dny+1:ny)=trreconstruct(N);
    nx=nx+dnx;
    ny=ny+dny;
```

```
    disp(sprintf('%1.5f; %d lines', nx/length(x), length(phases)));
  end;

  y=y*max(abs(x))/max(abs(y));
```

4.7.2 trbisectpeak.m

```
% function [ampl2, dphi, phases]=trbisectpeak(x, Z);
%
% applies peak detection to the signal x.
% uses the bisection method
%
% input:
%
% x       : input signal
% Z       : number of zeros for zero-padding
%
% output:
%
% ampl2 : measured amplitudes belonging to the peaks
% dphi   : measured frequency between the peaks of the current measurement
%          and the previous measurement
% phases : phases belongint to the peaks
%
% copyright (C) 2005 Bert Greevenbosch

function [ampl2, dphi, phases]=trbisectpeak(x, Z);

  x=torow(x);
  N=length(x);
  z=fft(fftshift([zeros(1, 0.5*Z), x, zeros(1, 0.5*Z)]));
  y=abs(z);
  phi2=princarg(angle(z));
  k=2;
  n=0;
  ampl2=0;
  phases=0;
  dphi=0;
  for k=2:(N+Z)/2

    if y(k-1)<y(k) & y(k+1)<=y(k) & y(k)>1
      if y(k-1)>y(k+1)
        zeta1=k-1;
        zeta2=k;
      else
        zeta1=k;
        zeta2=k+1;
      end
      y1=abs(y(zeta1));
      y2=abs(y(zeta2));
      zeta1=(zeta1-1)*2*pi/(N+Z);
      zeta2=(zeta2-1)*2*pi/(N+Z);

      omega1=zeta1;
      omega2=zeta2;

      sg=sign(y1*ws(zeta2-omega1, N)-y2*ws(zeta1-omega1, N));

      for m=1:10
        xi=(omega1+omega2)/2;
        res=y1*ws(zeta2-xi, N)-y2*ws(zeta1-xi, N);
        if res==0
```

```
        break;
      end
      if res*sg>0
        omega1=xi;
      else
        omega2=xi;
      end

    end
    xi=(omega1+omega2)/2;
    A = y1/ws(zeta1-xi, N);

    n=n+1;
    ampl2(n)=A;
    dphi(n)=xi;
    phases(n)=phi2(k);

  end

  k=k+1;

end
```

4.7.3 rs.m

```
% function f=rs(omega, N);
%
% calculates the DTFT in omega of the rectangular window
%
%   r[n] = 1   if -N/2 < n < N/2
%          0   otherwise
%
% copyright (C) 2005 Bert Greevenbosch

function f=rs(omega, N)

  if(omega==0)
    f=(N-1);
  else
    f=sin((N-1)*omega/2)/sin(omega/2);
  end
```

4.7.4 ws.m

```
% function f=ws(omega, N);
%
% calculates the DTFT in omega of the N point Hanning window, defined by
%
%   w[n] = cos([-N/2+1:-N/2-1]*pi/N]).^2
%
% copyright (C) 2005 Bert Greevenbosch

function f=ws(omega, N);

  f=.25*rs(omega-2*pi/N, N)+.25*rs(omega+2*pi/N, N)+.5*rs(omega, N);
```

4.8 The tracking phase vocoder with guides

4.8.1 trguides.m

```
% y=trguides(x, N, Z, overlap, stretch)
%
% Applies the tracking phase vocoder with guides to the signal x
%
% input:
%
% x       : input signal
% N       : window size
% Z       : number of zeros to append for zero-padding
% overlap : amount of overlap between succesive measurements
% stretch : time-stretch factor
%
% output:
%
% y       : time-stretched signal
%
% copyright (C) 2005 Bert Greevenbosch

function y=trguides(x, N, Z, overlap, stretch)

  global fadein;
  global fadeout;
  global lines;
  global tracks;
  global xlat;
  global y;

  x=torow(x);
  window=sin((0:N-1)*pi/N).^2;
  dnx=round((1-overlap)*N);
  dny=round(dnx*stretch);
  nx=N/2+1+dnx;
  dpsi=(0:(N+Z)/2)*dnx*2*pi/(N+Z);
  [A, phi1]=calcAphi(x(1:N).*window, Z);
  tracks=zeros(N/2+1, 6);
  lines=zeros(N/2+1, 3);
  xlat=zeros(N/2+1, 1);
  y=zeros(1, ceil(length(x)/dnx)*dny+dny+1);
  c=1;
  fadein=-.5*cos((0:dny-1)*pi/dny)+.5;
  fadeout=.5*cos((0:dny-1)*pi/dny)+.5;
  while(nx<=length(x)-N/2+1)
    [ampl2, dphi, phi2, phases]=trpeak(x(nx-N/2:nx+N/2-1).*window, phi1, dpsi, dnx, Z);
    trmatch(dphi, ampl2, phases, N);
    trguidesmatch(c, dny, N);
    trguidesreconstruct(N);
    nx=nx+dnx;
    c=c+1;
    phi1=princarg(phi2);
    disp(sprintf('%1.5f; %d lines', nx/length(x), length(phases)));
  end;
  trguidesreconstructlast(N);

  y=y*max(abs(x))/max(abs(y));
```

4.8.2 trguidesmatch.m

```
% function trguidesmatch(c, n, N)
%
% matches the partials to the guides
%
% global: tracks, lines, xlat
%
% input:
%
% c = number of the grain (first grain: 0, second grain: 1, ...)
% n = number of samples between successive grains in the reconstruction
% N = grain size of the measurements
%
% copyright (C) 2005 Bert Greevenbosch

function trguidesmatch(c, n, N)

  global tracks;
  global lines;
  global xlat;

  xlatnew=zeros(N/2+1, 1);
  phases=zeros(N/2+1, 1);
  oldlen=lines(:, 2);
  for k=1:N/2+1
    bin=tracks(k, 7);
    if bin~=0                                % matched
      addr=xlat(bin);
      l=lines(addr, 2)+1;
      lines(addr, 2)=l;                      % length
      lines(addr, 3)=0;                      % sleep counter;
      lines(addr, l*2+3)=tracks(k, 4);       % frequency
      lines(addr, l*2+4)=tracks(k, 5);       % amplitude
      tracks(bin, 2)=0;                      % this bin is done
      xlatnew(k)=addr;
      phases(k)=tracks(bin, 3)+n*tracks(k, 4);
    else
      if tracks(k, 5)~=0                     % fadein, birth of track
        addr=trguidesgetline(k);
        xlatnew(k)=addr;
        lines(addr, 1)=c;
        lines(addr, 2)=1;
        lines(addr, 3)=0;
        lines(addr, 4)=tracks(k, 6);         % phase
        lines(addr, 5)=tracks(k, 4);         % frequency
        lines(addr, 6)=tracks(k, 5);         % amplitude
        phases(k)=tracks(k, 6);
      end
    end
  end
  tracks(:, 1:2)=tracks(:, 4:5);
  tracks(:, 3)=princarg(phases);
  dlen=lines(:, 2)-oldlen;
  for addr=1:N/2+1
    if dlen(addr)==0 & lines(addr, 2)~=0 % no match and length nonzero
      lines(addr, 3)=1;                      % line done
    end
  end
  xlat=xlatnew;
  for k=1:N/2+1
    if tracks(k, 5)==0
```

```
      tracks(k, 1:3)=zeros(1, 3);
    end
  end
```

4.8.3 trguidesreconstruct.m

```
% function trguidesreconstruct(N)
%
% reconstructs finished lines into the target signals
%
% global: fadein, fadeout, y, lines, xlat, tracks
%
% input:
%
% N : size of the grain used for the measurements
%
% copyright (C) 2005 Bert Greevenbosch

function trguidesreconstruct(N)

  global fadein;
  global fadeout;
  global y;
  global lines;
  global xlat;
  global tracks;

  dny=length(fadein);
  for k=1:N/2+1
    if lines(k, 3) & lines(k, 2)>0
      x=0;
      phase=lines(k, 4);
      dphase=lines(k, 5);
      totfreq=dphase;
      A1=lines(k, 6);
      x(1:dny)=(fadein*A1).*cos(phase+(-dny:-1)*dphase);
      for m=1:lines(k, 2)-1
        A0=lines(k, m*2+4);
        A1=lines(k, m*2+6);
        dphase=lines(k, m*2+5);
        totfreq=totfreq+dphase;
        x(m*dny+1:(m+1)*dny)=(fadeout*A0+fadein*A1).*cos(phase+(0:dny-1)*dphase);
        phase=phase+dny*dphase;
      end
      m=lines(k, 2);
      A0=lines(k, m*2+4);
      dphase=lines(k, m*2+3);
      totfreq=totfreq+dphase;
      totfreq=totfreq/(lines(k, 2)+1);
      x(m*dny+1:(m+1)*dny)=(fadeout*A0).*cos(phase+(0:dny-1)*dphase);
      disp(sprintf('line ended: start %d; length %d; address: %d; average frequency: %1.5f', ...
        lines(k, 1), lines(k, 2), k, totfreq));
      y((lines(k, 1)-1)*dny+1:lines(k, 1)*dny+m*dny)=y((lines(k, 1)-1)*dny+1:lines(k, 1)*dny+m*dny)+x;
      lines(k, 1:3)=zeros(1, 3);
    end
  end
```

4.8.4 trguidesreconstructlast.m

```
% function trguidesreconstructlast(N)
%
% reconstructs the unfinished guides at the end of the process
%
% global: lines
%
% input:
%
% N = grain size used for measurements
%
% copyright (C) 2005 Bert Greevenbosch

function trguidesreconstructlast(N)

  global lines;
  for k=1:N/2+1
    if lines(k, 2)~=0
      lines(k, 3)=1;
    end
  end
  trguidesreconstruct(N);
```

4.8.5 trguidesgetline.m

```
% function addr=trguidesgetline(k);
%
% finds free space in lines structure
%
% global: lines
%
% input:
%
% k    : frequency bin
%
% output:
%
% addr : index of free space in lines
%
% copyright (C) 2005 Bert Greevenbosch

function addr=trguidesgetline(k);

  global lines;
  sz=size(lines);
  N=sz(1);
  addr=k;
  nu=0;
  while lines(addr, 2)~=0 & nu~=2
    addr=addr+1;
    if addr>N/2+1
      addr=1;
      nu=nu+1;
    end
  end
  if nu==2
    disp('[ERROR] expansion of lines neccessary');
    addr=N/2+2;
  end
```

4.9 The tracking phase vocoder used as a clarinet filter

4.9.1 trclar.m

```
% function [y1, y2]=trclar(x, N, Z, overlap)
%
% input:
%
% x       : input signal
% N       : window size
% Z       : number of zeros for zero-padding
% overlap : overlap factor
%
% output:
%
% y1      : clarinet signal
% y2      : non-clarinet signal
%
% copyright (C) 2005 Bert Greevenbosch

function [y1, y2]=trvoc(x, N, Z, overlap)

  global fadein;
  global fadeout;
  global tracks;

  x=torow(x);
  window=sin((0:N-1)*pi/N).^2;
  dn=round((1-overlap)*N);
  nx=N/2+1+dn;
  ny=dn;
  dpsi=(0:(N+Z)/2)*dn*2*pi/(N+Z);
  [A, phi1]=calcAphi(x(1:N).*window, Z);
  tracks=zeros(N/2+1, 6);
  y1=zeros(1, length(x));
  y2=y1;
  fadein=-.5*cos((0:dn-1)*pi/dn)+.5;
  fadeout=.5*cos((0:dn-1)*pi/dn)+.5;
  while(nx<length(x)-N/2-dn)
    [ampl2, dphi, phi2, phases]=trpeak(x(nx-N/2:nx+N/2-1).*window, phi1, dpsi, dn, Z);
    trmatch(dphi, ampl2, phases, N);
    clar=trdetclar(N);
    [y1(ny-dn+1:ny), y2(ny-dn+1:ny)]=trclarreconstruct(clar, N);
    nx=nx+dn;
    ny=ny+dn;
    phi1=princarg(phi2);
    disp(sprintf('%1.5f %d lines', nx/length(x), length(phases)));
  end;

  mul=max(abs(x))/max(abs(y1+y2));
  y1=y1*mul;
  y2=y2*mul;
```

4.9.2 trdetclar.m

```
% function clar=trdetclar(N)
%
% marks partials that are part of a clarinet tone
%
% global: tracks
%
```

```
% input:
%
% N    : grain size
%
% output:
%
% clar : vector of zeros and ones; clar(k)==1 if the bin k is part of a
%        clarinet tone.
%
% copyright (C) 2005 Bert Greevenbosch

function clar=trdetclar(N);

  global tracks;

  clar=zeros(N/2+1, 1);
  [maxamp, idx]=max(tracks(:, 5));
  basefreq=tracks(idx, 4);
  if(basefreq>0)
    freq=basefreq;
    while freq<pi
      bin=floor(N*freq/(2*pi));
      if(bin>1)
        clar(bin-1)=1;
      end
      if(bin>0)
        clar(bin  )=1;
      end
      if(bin<N/2+1)
        clar(bin+1)=1;
      end
      freq=freq+basefreq;
    end
  end
```

4.9.3 trclarreconstruct.m

```
% function [y1, y2]=trclarreconstruct(clar, N)
%
% reconstructs the current partials into either the clarinet signal or the
% non-clarinet signal
%
% global: tracks, fadein, fadeout
%
% input:
%
% clar : vector containing information about which bins are clarinet bins
% N    : grain size
%
% output:
%
% y1   : clarinet grain
% y2   : non-clarinet grain
%
% copyright (C) 2005 Bert Greevenbosch


function [y1, y2]=trclarreconstruct(clar, N)

  global tracks;
  global fadein;
  global fadeout;
```

```
dny=length(fadein);
y1=zeros(1, dny);
y2=y1;
phases=zeros(N/2+1, 1);
for k=1:N/2+1
    z=zeros(1, dny);
    bin=tracks(k, 7);
    if bin~=0
        if tracks(k, 5)~=0 | tracks(bin, 2)~=0
          A0=tracks(bin, 2);
          A1=tracks(k, 5);
          phi=(0:dny-1)*tracks(k, 4)+tracks(bin, 3);
          z=(A0*fadeout+A1*fadein).*cos(phi);
          tracks(bin, 2)=0; % this bin is done
        end
        phases(k)=dny*tracks(k, 4)+tracks(bin, 3);
    else
        if tracks(k, 5)~=0 % fade in
           A1=tracks(k, 5);
           phi=(-dny:-1)*tracks(k, 4)+tracks(k, 6);
           z=A1*fadein.*cos(phi);
           phases(k)=tracks(k, 6); % keep this phase
        end
    end
    if clar(k)==1
      y1=y1+z;
    else
      y2=y2+z;
    end
end
for k=1:N/2+1
    if tracks(k, 2)~=0 % frequency not done yet; fade out
        A0=tracks(k, 2);
        phi=(0:dny-1)*tracks(k, 1)+tracks(k, 3);
        z=A0*fadeout.*cos(phi);
        if clar(k)==1
          y1=y1+z;
        else
          y2=y2+z;
        end
    end
end
tracks(:, 1:2)= tracks (:, 4:5);
tracks(:, 3)= princarg(phases);
for k=1:N/2+1
  if tracks(k, 5)==0
      tracks(k, 1:3)=zeros(1, 3);
  end
end
```

4.10 The tracking phase vocoder with guides used as a clarinet filter

4.10.1 trclarguides.m

```
% [y1, y2]=trclarguides(x, N, Z, overlap, frac)
%
% Modification of the tracking phase vocoder with guides to extract the
% clarinet from a signal x
%
% input:
%
% x       : input signal
% N       : window size
% Z       : number of zeros to append for zero-padding
% overlap : amount of overlap between succesive measurements
% frac    : required amount of partials marked as clarinet for a line to be
%           marked as clarinet
%
% output:
%
% y1      : clarinet signal
% y2      : non-clarinet signal
%
% copyright (C) 2005 Bert Greevenbosch

function [y1, y2]=trclarguides(x, N, Z, overlap, frac)

  global fadein;
  global fadeout;
  global lines;
  global tracks;
  global xlat;
  global y1;
  global y2;

  x=torow(x);
  window=sin((0:N-1)*pi/N).^2;
  dn=round((1-overlap)*N);
  nx=N/2+1+dn;
  dpsi=(0:(N+Z)/2)*dn*2*pi/(N+Z);
  [A, phi1]=calcAphi(x(1:N).*window, Z);
  tracks=zeros(N/2+1, 6);
  lines=zeros(N/2+1, 3);
  xlat=zeros(N/2+1, 1);
  y1=zeros(1, length(x)+dn+1);
  y2=y1;
  c=1;
  fadein=-.5*cos((0:dn-1)*pi/dn)+.5;
  fadeout=.5*cos((0:dn-1)*pi/dn)+.5;
  while(nx<=length(x)-N/2+1)
    [ampl2, dphi, phi2, phases]=trpeak(x(nx-N/2:nx+N/2-1).*window, phi1, dpsi, dn, Z);
    trmatch(dphi, ampl2, phases, N);
    clar=trdetclar(N);
    trclarguidesmatch(c, dn, N, clar);
    trclarguidesreconstruct (N, frac);
    nx=nx+dn;
    c=c+1;
    phi1=princarg(phi2);
    disp(sprintf('%1.5f; %d lines', nx/length(x), length(phases)));
  end;
```

```
 trclarguidesreconstructlast (N, frac);

mul=max(abs(x))/(max(abs(y1+y2)));
y1=y1*mul;
y2=y2*mul;
```

4.10.2 trclarguidesmatch.m

```
% function trclarguidesmatch(c, n, N, clar)
%
% matches the partials to the guides
%
% global: tracks, lines, xlat
%
% input:
%
% c    : number of the grain ( first grain : 0, second grain : 1, ...)
% n    : number of samples between successive grains in the reconstruction
% N    : grain size of the measurements
% clar : vector containing data about which partials are part of a clarinet
%        tone.
%
% copyright (C) 2005 Bert Greevenbosch

function trclarguidesmatch(c, n, N, clar)

  global tracks;
  global lines;
  global xlat;

  xlatnew=zeros(N/2+1, 1);
  phases=zeros(N/2+1, 1);
  oldlen=lines(:, 2);
  for k=1:N/2+1
    bin=tracks(k, 7);
    if bin~=0                              % matched
      addr=xlat(bin);
      l=lines(addr, 2)+1;
      lines(addr, 2)=l;                    % length
      lines(addr, 3)=0;                    % sleep counter;
      lines(addr, l*2+4)=tracks(k, 4);     % frequency
      lines(addr, l*2+5)=tracks(k, 5);     % amplitude
      tracks(bin, 2)=0;                    % this bin is done
      xlatnew(k)=addr;
      phases(k)=tracks(bin, 3)+n*tracks(k, 4);
      lines(addr, 4)=lines(addr, 4)+clar(k);
    else
      if tracks(k, 5)~=0                   % fadein, birth of track
        addr=trguidesgetline(k);
        xlatnew(k)=addr;
        lines(addr, 1)=c;
        lines(addr, 2)=1;
        lines(addr, 3)=0;
        lines(addr, 5)=tracks(k, 6);       % phase
        lines(addr, 6)=tracks(k, 4);       % frequency
        lines(addr, 7)=tracks(k, 5);       % amplitude
        phases(k)=tracks(k, 6);
        lines(addr, 4)=lines(addr, 4)+clar(k);
      end
    end
  end
  tracks(:, 1:2)= tracks (:, 4:5);
```

```
  tracks(:, 3)=princarg(phases);
  dlen=lines(:, 2)-oldlen;
  for addr=1:N/2+1
    if dlen(addr)==0 & lines(addr, 2)~=0 % no match and length nonzero
      lines(addr, 3)=1;                  % line done
    end
  end
  xlat=xlatnew;
  for k=1:N/2+1
    if tracks(k, 5)==0
      tracks(k, 1:3)=zeros(1, 3);
    end
  end
```

4.10.3 trclarguidesreconstruct.m

```
% function trclarguidesreconstruct(N, frac)
%
% reconstructs finished lines into the target signals
%
% global: fadein, fadeout, y1, y2, lines, xlat, tracks
%
% input:
%
% N    : size of the grain used for the measurements
% frac : minimum fraction of partials that should be considered to be a
%        part of a clarinet tone to consider the guide to contain a
%        clarinet tone
%
% copyright (C) 2005 Bert Greevenbosch

function trclarguidesreconstruct(N, frac)

  global fadein;
  global fadeout;
  global y1;
  global y2;
  global lines;
  global xlat;
  global tracks;

  dny=length(fadein);
  for k=1:N/2+1
    if lines(k, 3) & lines(k, 2)>0
      x=0;
      phase=lines(k, 5);
      dphase=lines(k, 6);
      totfreq=dphase;
      A1=lines(k, 7);
      x(1:dny)=(fadein*A1).*cos(phase+(-dny:-1)*dphase);
      for m=1:lines(k, 2)-1
        A0=lines(k, m*2+5);
        A1=lines(k, m*2+7);
        dphase=lines(k, m*2+6);
        totfreq=totfreq+dphase;
        x(m*dny+1:(m+1)*dny)=(fadeout*A0+fadein*A1).*cos(phase+(0:dny-1)*dphase);
        phase=phase+dny*dphase;
      end
      m=lines(k, 2);
      A0=lines(k, m*2+5);
      dphase=lines(k, m*2+4);
      totfreq=totfreq+dphase;
```

```
      totfreq=totfreq/(lines(k, 2)+1);
      x(m*dny+1:(m+1)*dny)=(fadeout*A0).*cos(phase+(0:dny-1)*dphase);
      if( lines(k, 4)>=m*frac) % enough marked as clarinet
        disp(sprintf('line ended: start %d; length %d; address: %d; average frequency: %1.5f (clarinet)', ...
          lines(k , 1),  lines(k , 2),  k ,  totfreq ));
        y1((lines(k, 1)-1)*dny+1:lines(k, 1)*dny+m*dny)=y1((lines(k, 1)-1)*dny+1:lines(k, 1)*dny+m*dny)+x;
      else
        disp(sprintf('line ended: start %d; length %d; address: %d; average frequency: %1.5f', ...
          lines(k , 1),  lines(k , 2),  k ,  totfreq ));
        y2((lines(k, 1)-1)*dny+1:lines(k, 1)*dny+m*dny)=y2((lines(k, 1)-1)*dny+1:lines(k, 1)*dny+m*dny)+x;
      end
      lines(k, 1:4)=zeros(1, 4);
    end
  end
```

4.10.4 trclarguidesreconstructlast.m

```
% function trlinesreconstructlast(N, frac)
%
% reconstructs the unfinished guides at the end of the process
%
% global: lines
%
% input:
%
% N    : grain size used for measurements
% frac : minimum fraction of partials that should be considered to be a
%        part of a clarinet tone to consider the guide to contain a
%        clarinet tone
%
% copyright (C) 2005 Bert Greevenbosch

function trlinesreconstructlast(N, frac)

  global lines;
  for k=1:N/2+1
    if lines(k, 2)~=0
      lines(k, 3)=1;
    end
  end
  trclarguidesreconstruct(N, frac);
```

4.11 Auxiliary functions

4.11.1 play.m

```
% function play(x)
%
% plays the signal x at 44100 Hz
%
% copyright (C) 2005 Bert Greevenbosch

function play(x)

  sound(x, 44100);
```

4.11.2 tocolumn.m

```
% function y=tocolumn(x)
%
% converges a vector x to a column vector y
%
% copyright (C) 2005 Bert Greevenbosch

function y=tocolumn(x)

  sz=size(x);
  if sz(1)>sz(2)
     y=x;
  else
     y=x';
  end
```

4.11.3 torow.m

```
% function y=torow(x)
%
% converts the vector x to a column vector y
%
% copyright (C) 2005 Bert Greevenbosch

function y=torow(x)

  sz=size(x);
  if sz(1)<sz(2)
    y=x;
  else
    y=x';
  end
```

Bibliography

[1] Judith C. Brown, 'Computer identification of musical instruments using pattern recognition with cepstral coefficients as features', Physics department, Wellesley college, Wellesley, Massachusetts, *Journal of the acoustical society of America*, 105(3), March 1999, pages 1933–41

[2] J.W. Cooley and O.W. Tukey, 'An algorithm for the machine calculation of complex fourier series', *Mathematics of computation*, **19**, 1965, pages 297–301

[3] Ingrid Daubechies, *Ten lectures on wavelets*, Capital City Press, Montpelier, Vermont, 1992, ISBN 0 89871 274 2

[4] Amalia De Götzen, Nicola Bernardini and Daniel Arfib, 'Traditional (?) implementations of a phase-vocoder: the tricks of the trade', *Proceedings of the COST G-6 Converence on Digital Audio Effects (DAFX-00)*, Verona, Italy, 7–9 December 2000

[5] Giovanni De Poli, Aldo Piccialli and Curtis Roads, *Representations of musical signals*, The MIT Press, 1991

[6] Antti Eronen and Anssi Klapuri, 'Musical instrument recognition using cepstral coefficients and temporal features', Signal processing laboratory, Tampere university of technology *Proceedings 2000 IEEE international conference on acoustics, speech and signal processing*, ICASSP '00, 2:753–6

[7] J.L. Flanagan and R.M. Golden, 'Phase vocoder', *Bell systems techonological journal*, volume 45, November 1966, pages 1493–1509

[8] Neville H. Fletcher and Thomas D. Rossing, *The physics of musical instruments*, Springer-Verlag, New York, 1998, ISBN 0 387 98374 0

[9] G. Folland and A. Sitaram, 'The uncertainty principle: a mathematical survey', *Journal of Fourier analysis and applications*, volume 3, 1997, pages 207–238

[10] J.W. Gibbs, 'Fourier Series', *Nature* **59**, 1899, pages 200 and 606

[11] Fredric J. Harris, 'On the use of windows for harmonic analysis with the discrete Fourier transform', *Proceedings of the IEEE*, volume 66, no. 1, January 1978, pages 51–83

[12] W. Heisenberg, 'Über den anschaulichen Inhalt der quantentheoretischen Kinematik und Mechanik', *Zeitschrift für Physik*, volume 43, 1927, pages 172–198

[13] D.J. Hermes, 'Measurement of pitch by subharmonic summation', *Journal of the acoustical society of America*, volume 83, 1988, pages 257–264

[14] I. Hirschman, 'A note on entropy', *American journal of mathematics*, volume 79, 1957, pages 152–156

[15] Stéphane Mallat, *A wavelet tour of signal processing*, Academic Press, 1998, ISBN 0 21 466606 X

[16] Keith D. Martin and Youngmoo E. Kim, 'Musical instrument identification: A pattern-recognition approach', *Journal of the acoustical society of America*, 136th meeting, 1998, 103 (3 pt 2): 1768

[17] R.J. McAulay and T.F. Quatieri, 'Speech analysis/synthesis based on a sinusoidal representation', *IEEE transactions on acoustics, speech and signal processing*, volume 34, no. 4, 1986, pages 744–754

[18] McGill university master samples, `http://www.music.mcgill.ca/resources/mums/html`

[19] Sanjit K. Mitra and James F. Kaiser, *Handbook for digital signal processing*, John Wiley & sons, 1993, ISBN 0 471 61995 7

[20] E. Oran Brigham, *The fast Fourier transform*, Prentice-Hall, 1974, ISBN 0 13 307496 X

[21] Curtis Roads, *The computer music tutorial*, The MIT press, 1996, ISBN 0 252 18158 4

[22] A. van Rooij, *Fouriertheorie*, Epsilon uitgaven, Utrecht, 1988, ISBN 90 5041 014 6

[23] Stephen. W. Smith, *The scientist and engineer's guide to digital signal processing*, California Technical Publishing, 1999, `http://www.dspguide.com`

[24] A.J. Stam, 'Some inequalities satisfied by the quantities of information of Fisher and Shannon', *Information and control*, volume 2, 1959, pages 101–112

[25] Ken Steiglitz, *A digital signal processing primer*, Addison-Wesley publishing company, 1996, ISBN 0 8053 1684 1

[26] Stan Tempelaars, *Signal processing, speech and music*, Swets & Zeitlinger publishers, 1996, ISBN 90 265 1481 6

[27] E.C. Titchmarsh, *Theory of Fourier integrals*, Oxford university press, 1937

[28] C.R. Traas, H.G. ter Morsche and R.M.J. van Damme, *Splines en wavelets*, Epsilon uitgaven, Utrecht, 2000, ISBN 90 5041 057 X

[29] Tuamas Virtanen and Anssi Klapuri, 'Separation of harmonic sounds using multipitch analysis and iterative parameter estimation', Signal processing laboratory, Tampere university of technology, 2001, `http://www.cs.tut.fi/ tuomasv/demopage.html`

[30] Tuamas Virtanen, 'Separation of sound sources by convolutive sparse coding', Institute of Signal Processing, Tampere university of techonology, 2004, `http://www.cs.tut.fi/ tuomasv/demopage.html`

[31] Udo Zölzer et al, *DAFX – Digital Audio Effects*, John Wiley & sons, 2002, ISBN 0 471 49078 4

Acknowledgements

I would like to take this opportunity to thank the people that have supported me during my studies. I thank my parents for providing me with the means to have a good education. I thank the teachers that have guided me into the fields of mathematics. I thank dr. D.R. Pik for his advice and guidance on this project, and the pleasant cooperation that we had.

Index