Proseminar: Computer and Music

Fourier Transform

Andreas Brüggemann, Lukas Westhofen

November 17, 2017

Contents

1 Time and Frequency Domain of Audio Signals

The Fourier analysis offers a variety of tools for the purpose of analysing functions by dividing them into a finite or infinite sum of trigonometric functions with different frequencies. An audio signal can be represented by a real function in the time domain which describes the amplitude dependent on time. The Fourier transform can be used to get the frequency domain where the amplitude depends on the frequency of a specific trigonometric function instead of the time. The frequency domain has a high importance in audio processing. This domain is easier to analyse than the time domain in many cases as chord recognition for instance. It can also be used to implement digital frequency filters and noise reduction. In conclusion, the Fourier transform is a powerful mathematical tool in audio processing as in other purposes which provides many possibilities that have shaped today's interaction between computers and music.

2 Different Signal Spaces

There are four Hilbert spaces which represent different types of signals. The exact properties of Hilbert spaces are important to prove the existence of a basis for each space for instance. Each space has its own Fourier transform. The Hilbert spaces and corresponding signals are:[1]

- $\mathcal{L}^2([-T/2, T/2))\, T > 0$, contains continuous periodic signals
- $\mathcal{L}^2(\mathbb{R})$, contains continuous aperiodic signals
- $\ell^2(\mathbb{Z}/N)\, N \in \mathbb{N}$, contains discrete periodic signals
- $\ell^2(\mathbb{Z})$, contains discrete aperiodic signals

The periodic cases are addressed here.

3 Continuous, periodic Fourier transform

3.1 Real Fourier Series with $T = 2\pi$

The continuous, periodic Fourier transform is used to analyse continuous, periodic signals. Therefore, the Hilbert space $\mathcal{L}^2([-\pi, \pi))$ is used as a set of functions representing those signals with the period $T = 2\pi$. $\{\frac{1}{\sqrt{2}}, \cos(nt), \sin(nt) \mid n \in \mathbb{N}\}$ is an orthonormal basis of $\mathcal{L}^2([-\pi, \pi))$ using the inner product defined as:[2]

$$\langle f, g \rangle = \frac{1}{\pi} \int_{-\pi}^{\pi} f(t) \cdot g(t) dt \tag{1}$$

The Hilbert space only contains functions for signals with a finite amount of energy.

$$f \in \mathcal{L}^2([-\pi, \pi)) \Rightarrow \int_{-\pi}^{\pi} f(t) dt < \infty \tag{2}$$

[1] We use versions of the spaces in [2]

[2] We use $\mathbb{N}$ with $0 \notin \mathbb{N}$

$f \in L^2([-\pi, \pi))$ can be written as a linear combination using the given basis.

$$\forall n \in \mathbb{N} : \exists a_0, a_n, b_n \in \mathbb{R} : f(t) = \frac{a_0}{2} + \sum_{k=1}^{\infty} (a_k \cdot \cos(kt) + b_k \cdot \sin(kt)) \tag{3}$$

The given orthonormal basis can be used to express a linear combination using the inner product. The result leads to the coefficients given in Equation 3.

$$\begin{aligned}
&\forall\, f \in L^2([-\pi, \pi)) : f(t) = \langle f, \frac{1}{\sqrt{2}} \rangle \frac{1}{\sqrt{2}} + \sum_{k=1}^{\infty} (\langle f, \cos(kt) \rangle \cos(kt) + \langle f, \sin(kt) \rangle \sin(kt)) \\
&= \frac{1}{\pi} \int_{-\pi}^{\pi} f(t) \cdot \frac{1}{\sqrt{2}} dt \cdot \frac{1}{\sqrt{2}} \\
&+ \sum_{k=1}^{\infty} \left(\frac{1}{\pi} \int_{-\pi}^{\pi} f(t) \cdot \cos(kt) dt \cdot \cos(kt) + \frac{1}{\pi} \int_{-\pi}^{\pi} f(t) \cdot \sin(kt) dt \cdot \sin(kt) \right) \\
&= \frac{1}{2\pi} \int_{-\pi}^{\pi} f(t) dt \\
&+ \sum_{k=1}^{\infty} \left(\frac{1}{\pi} \int_{-\pi}^{\pi} f(t) \cdot \cos(kt) dt \cdot \cos(kt) + \frac{1}{\pi} \int_{-\pi}^{\pi} f(t) \cdot \sin(kt) dt \cdot \sin(kt) \right)
\end{aligned} \tag{4}$$

Comparing the result to Equation 3 gives the coefficients as follows:

$$\frac{a_0}{2} = \frac{1}{2\pi} \int_{-\pi}^{\pi} f(t) dt \Rightarrow a_0 = \frac{1}{\pi} \int_{-\pi}^{\pi} f(t) dt \tag{5}$$

$$a_k = \frac{1}{\pi} \int_{-\pi}^{\pi} f(t) \cdot \cos(kt) dt \text{ for } k \in \mathbb{N} \tag{6}$$

$$b_k = \frac{1}{\pi} \int_{-\pi}^{\pi} f(t) \cdot \sin(kt) dt \text{ for } k \in \mathbb{N} \tag{7}$$

3.2 Example

A popular example of using Fourier transforms is the square wave. The odd function shown in Figure 1 represents a square wave in $L^2([-\pi, \pi))$. This example demonstrates

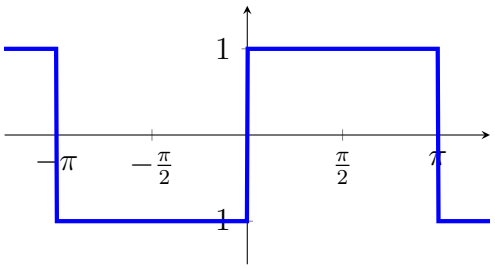

Figure 1: Odd 2π-periodical function

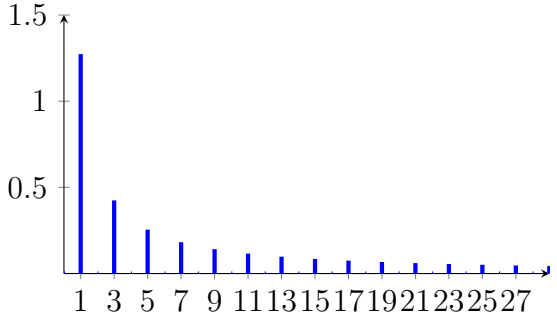

Figure 2: Frequency-domain of the square wave in Figure 1

that it is even possible to divide a square wave into sine and cosine waves only. The different frequencies of the sine waves are distributed as shown in Figure 2, while the amplitudes of all cosine waves and the offset equal zero. An infinite summation of the given sine waves produces the square wave. Parts of this summation are shown in Figure 3.

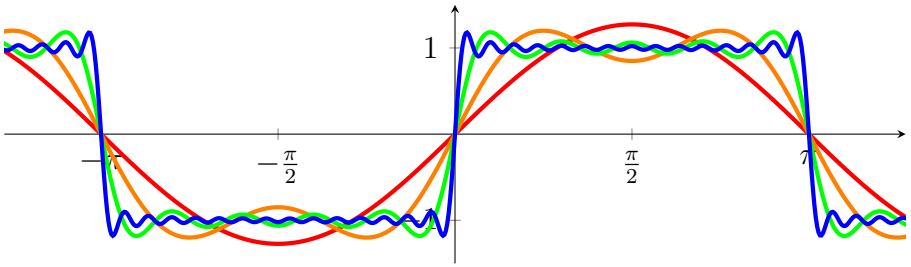

Figure 3: $f_2(t) = \sum_{k=1}^{n} b_k \cdot \sin(kt)$ for $n = 1$ (red), $n = 3$ (orange), $n = 9$ (green), $n = 29$ (blue),

3.3 Complex Fourier Series with $T = 2\pi$

For further steps and easier usage of the transform it is recommended to use the exponential function instead of sine and cosine waves. This transform can be achieved using the correlation between the exponential function and the trigonometric functions.

$$\begin{aligned} f(t) &= \frac{a_0}{2} + \sum_{k=1}^{\infty}(a_k \cdot \cos(kt) + b_k \cdot \sin(kt)) \\ &= c_0 \cdot e^{i \cdot 0t} + \sum_{k\in[1,\infty)} e^{ikt} c_k + \sum_{k\in(-\infty,-1]} e^{ikt} c_k = \sum_{k\in(-\infty,\infty)} c_k \cdot e^{ikt} \end{aligned}$$

with

$$\begin{aligned} c_0 &:= \frac{a_0}{2} \\ \forall k > 0 : c_k &:= \frac{1}{2} a_k + \frac{1}{2i} b_k = \frac{1}{2}(a_k - i \cdot b_k) \\ \forall k < 0 : c_k &:= \frac{1}{2} a_{-k} - \frac{1}{2i} b_{-k} = \frac{1}{2}(a_{-k} + i \cdot b_{-k}) \\ \Rightarrow c_0 &= \frac{1}{2\pi} \int_{-\pi}^{\pi} f(t) \cdot e^{-i \cdot 0t} dt \\ \forall k > 0 : c_k &= \frac{1}{2\pi} \int_{-\pi}^{\pi} f(t) \cdot e^{-ikt} dt \\ \forall k < 0 : c_k &= \frac{1}{2\pi} \int_{-\pi}^{\pi} f(t) \cdot e^{-ikt} dt \\ \Rightarrow \forall k \in \mathbb{Z} : c_k &= \frac{1}{2\pi} \int_{-\pi}^{\pi} f(t) \cdot e^{-ikt} dt \end{aligned}$$

In conclusion the transformation pair for periodic signals with $T = 2\pi$ is given with:

$$f(t) = \sum_{k \in (-\infty, \infty)} c_k \cdot e^{ikt} \tag{8}$$

$$c_k = \frac{1}{2\pi} \int_{-\pi}^{\pi} f(t) \cdot e^{-ikt} dt \, \forall k \in \mathbb{Z} \tag{9}$$

3.4 Complex Fourier Series with $T \in (0, \infty)$

If f with $T \neq 2\pi$ is given, it is possible to use the function h with:

$$f(t) = h(\frac{t \cdot 2\pi}{T}) = h(\omega_0 t) \, \forall t \in \mathbb{R}$$

This works for $T = 2\pi \Rightarrow f = h$ too. As $\{\frac{1}{\sqrt{2}}, \cos(nt), \sin(nt) \mid n \in \mathbb{N}\}$ is a basis[3] of the signals where Equation 2 is given with $T = 2\pi$ including h, $\{\frac{1}{\sqrt{2}}, \cos(\omega_0 nt), \sin(\omega_0 nt) \mid n \in \mathbb{N}\}$ is a basis of the corresponding signals with other values of T including f. The bounds of integration given in Equation 2 have to be changed to $-T/2$ and $T/2$ for this case. The new transformation pair is given with:

$$f(t) = h(\omega_0 t) = \sum_{k \in (-\infty, \infty)} c_k \cdot e^{ik\omega_0 t} \tag{10}$$

$$c_k = \frac{1}{2\pi} \int_{-\pi}^{\pi} h(u) \cdot e^{-iku} du = \frac{1}{T} \int_{-T/2}^{T/2} f(t) \cdot e^{-ik\omega_0 t} dt \, \forall k \in \mathbb{Z} \tag{11}$$

3.5 Conclusion

The transformation pair including Equations 10 and 11 can be used to change between time and frequency domain for all signals in $\mathcal{L}^2([-T/2, T/2))$ for any $T > 0$. The time domain is periodic and continuous while the frequency domain is aperiodic and discrete. The coefficients with index k describe the amplitude and phase of a trigonometric wave with $\omega = \omega_k = k\frac{2\pi}{T} = k \cdot \omega_0$. The real coefficients represent the amplitudes of sine and cosine waves while the addition of sine and cosine is used to achieve the phase while the complex coefficients hold both parameters in a more compact form.

4 Discrete Fourier Transform (DFT)

The signals for digital audio processing are discrete in their time domain instead of being continuous. A computer can only save a finite amount of values. Therefore the time domain should be periodic in order to make the frequency domain discrete too. The Discrete Fourier transform is used to transform discrete time domains into discrete frequency domains. To transform a signal $f(t)$ it should have a finite length T. If this signal is part of an infinite periodic signal, it should represent one period. If the origin is aperiodic, the part represented by $f(t)$ should be seen as a period of a periodic signal for

[3]With another definition of the inner product using $2/T$ instead of $1/\pi$ as factor and changing the bounds of the integral

a discrete frequency domain. A signal is represented by $f(t)$ which equals zero outside of the bounds $[0, T)$. The intervals are not chosen symmetrical to zero any more as the processed audio signal should start at $t = 0$. For further operations we define

$$f'(t) = \sum_{k=-\infty}^{\infty} f(t + kT) \text{ with}$$
$$\forall t \in [0, T) : f(t) = f'(t)$$

The continuous periodic case of the Fourier transform is used to analyse the periodic signal $f'(t)$. The bounds of integration are changed because the signal's interval starts at $t = 0$.

$$\begin{aligned} c_k &= \frac{1}{T} \int_0^T f'(t) \cdot e^{-ik\omega_0 t} dt = \frac{1}{T} \int_0^T f(t) \cdot e^{-ik\omega_0 t} dt \\ &= \frac{1}{T} \int_{-\infty}^{\infty} f(t) \cdot e^{-ik\omega_0 t} dt \, \forall k \in \mathbb{Z} \end{aligned}$$

Assuming there are N sampling points, $f(t)$ is only known for $t \in \{\frac{kT}{N} \mid k \in \mathbb{Z}, 0 \leq k < N\}$.

$$\int_{-\infty}^{\infty} f(t) \cdot e^{-ik\omega_0 t} dt \approx \sum_{k=0}^{N-1} f(\frac{kT}{N}) \cdot e^{-ik\omega_0 \frac{kT}{N}} \cdot \frac{T}{N}$$

For $m \in \mathbb{Z}$, $0 \leq m < N$:

$$\begin{aligned} F(m\omega_0) &= \frac{1}{\sqrt{2\pi}} \int_{-\infty}^{\infty} f(t) \cdot e^{-im\omega_0 t} dt \approx \frac{1}{\sqrt{2\pi}} \sum_{k=0}^{N-1} f\left(\frac{kT}{N}\right) \cdot e^{-im\omega_0 \frac{kT}{N}} \cdot \frac{T}{N} \\ &= \frac{T}{\sqrt{2\pi} N} \sum_{k=0}^{N-1} f\left(\frac{kT}{N}\right) \cdot e^{-2im\pi \frac{k}{N}} \end{aligned} \tag{12}$$

To transform the frequency domain into the time domain it is possible to solve the system of linear equations given with Equation 12 for every valid m according to [3]. For $k \in \mathbb{Z}$, $0 \leq k < N$:

$$f\left(\frac{kT}{N}\right) = \frac{\sqrt{2\pi}}{T} \sum_{m=0}^{N-1} F(m\omega_0) \cdot e^{2ik\pi \frac{m}{N}} \tag{13}$$

For the discrete Fourier transform time and frequency domain are discrete and periodic. Periodicity in one domain leads to a discretisation of the other domain. This is shown by using a Dirac comb to make a domain discrete which leads to convolution with the transformed Dirac comb in the other domain. That leads to periodicity in the opposing domain.

5 Optimisation

5.1 Symmetry of the Frequency Domain

Since the signal functions of an audio signal are restricted to real values as function values, a neat property occurs for the Fourier transforms discussed so far. Its spectrum contains

redundant information, because the negative coefficients resemble the positive ones. The source of this behaviour can be found in the complex conjugation of real numbers.

$$\sum_{k=\infty}^{\infty} c_k e^{-ikt} = f(t) = \overline{f(t)} = \sum_{k=\infty}^{\infty} \overline{c_k} e^{ikt} \;\Rightarrow\; \overline{c_k} = c_{-k} \tag{14}$$

The proof above shows only the property for $f \in \mathcal{L}^2([-\pi, \pi))$, but the missing ones follow the same principle. Now one can show that the absolute values of the positive and negative Fourier coefficients are the same. $\forall k \in \mathbb{Z} \setminus \{0\}$:

$$|c_k| = \sqrt{\mathrm{Re}(c_k) + \mathrm{Im}(c_k)} = \sqrt{\mathrm{Re}(\overline{c_k}) + \mathrm{Im}(\overline{c_k})} = \sqrt{\mathrm{Re}(-c_k) + \mathrm{Im}(-c_k)} = |c_{-k}|$$

After analysing the magnitude, one may be interested how the phase shift is affected. Other than the magnitude, the results vary with different functions f. Without going too deep mathematically the spectrum of all *even* (axis-symmetric) functions is mirrored across the y-axis and the spectrum of all *odd* (point-symmetric) functions is rotated around 180 degree around the origin. If the function possesses neither of these characteristics solely, one can show that every function can be divided into an even and an odd part.

In conclusion this means, that this property can be used to save memory space when storing a spectrum of a music signal by up to half of the original used space, if the analysed function is real and either only even or only odd.

5.2 Fast Fourier Transform (FFT)

The DFT's time complexity is given with $t(N) \in \Theta(N^2)$ because there are N sampling points for time and frequency domain and a summation over N indices is necessary to calculate one coefficient like in Equation 12 while the summations' bounds are the same for the inverse transform back to the time domain. Because of the high importance of DFT for audio processing it is important to switch between the domains faster. The fast Fourier transform calculates the missing domain recursively using a divide and conquer algorithm. It splits an array of sampling points in the odd and even ones depending on their index, calculates their Fourier transform and unites the results of the recursive calculations which takes time in $\Theta(N)$ together with the process of dividing. The complexity is given as $t(N) \in \Theta(N \cdot \log(N))$. The algorithm only works for $N = 2^m$ for an $m \in \mathbb{N}$. An implementation in Java can be found in the appendix.

6 Discrete Short-Time Fourier Transform

6.1 Motivation

The Fourier transform of a signal has one serious problem: It contains limited time information. Consider the following situation:

You are interested in the chords played in a time interval $[a, b], a, b \in [0, T]$ of a composition, which lasts T seconds. Therefore you use the Fourier transform of the whole

signal to analyse the frequencies to find the corresponding chords. But since the Fourier transform of this signal hides its time information, one can't recognise which tone was played at which time.

One solution to this problem is the **short-time Fourier transform**(**STFT**). It allows to create multiple Fourier transforms of one signal, which are assigned to certain time intervals of the signal. Since one wants to perform this task on a computer, one usually uses the **discrete short-time Fourier transform**.

6.2 Definition

As it was mentioned in 4, since digital processed music has always discrete values. To isolate certain time windows from the original signal to afterwards use multiple Fourier transforms, one introduces two new parameters: the *window function* the reference time stamp t. They both interact with each other, since g determines how much and how strong the time stamp t surrounding signal should be taken into consideration. A prototype formula of the discrete STFT would now be defined like this:

$$\tilde{f}_g(t,\omega) = \sum_{n=0}^{N-1} f(n)\overline{g}(n-t)e^{-2\pi ink/N} \tag{15}$$

But since there are situations like the computing of a visual representation of the spectrum in real-time, that need to process the information quickly, the computational efficiency of the Equation 15 has to be enhanced by limiting the amount of time windows and reference time stamps. This is achieved by adding the **hop size** $H \in \mathbb{N}$ into the formula.
This parameter is used to define a finite set of $m \in \mathbb{N}$ time windows with a constant offset H. Since a higher hop size decreases the memory consumption, but also decreases the resolution of the analysed time period, "[...] one often chooses $H = N/2$, which constitutes a good trade-off [...]" ([2], p.55).
Now the **discrete short-time Fourier transform** of a discrete signal x is given by:

$$\tilde{x}_g(m,k/N) := \sum_{n=0}^{N-1} x(n+mH)\overline{g}(n)e^{-2\pi ink/N} \tag{16}$$

6.3 Spectrogram

Recall the problem given in Chapter 6.1. If one not only wants to analyse one chord at a certain time, but the whole composition, there is a graphical display of a signal in a two or three dimensional graph. Such graph is called a **spectrogram**.

The x-axis represents the time spectrum and the y-axis represents the frequency spectrum of the signal. The magnitude of a frequency ω at the time t is given by a colour representation or by an additional z-axis. The formula to compute a spectogram with discrete values is given by:

$$\mathrm{Spec}_{\mathrm{disc}}(m,k/N) := |\tilde{x}_g(m,k/N)|^2 \tag{17}$$

To receive a meaningful spectrogram, one has to chose an appropriate window function and hop size. We want to emphasise this by the following example.

6.4 Properties of the STFT and Spectrograms

Example: You have a music recording and you want to find out which tone was played at which time, starting with the first ten seconds. Thus you use a STFT with a sampling rate of 1 kHz and a quantified frequency range from 0 Hz to 500 Hz with a step size of 0.5 Hz. Now the missing informations to compute a spectrogram are the window function[4] and the hop size H. The following spectrograms differ just in these parameters.

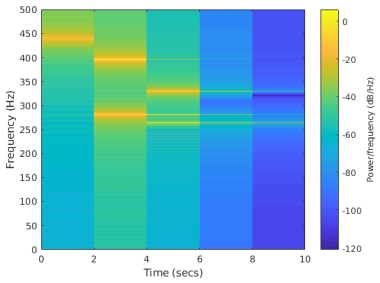

Figure 4: Spectrogram 1
$l = 2$ s, $H = 2$ s

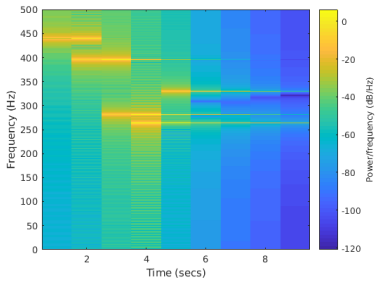

Figure 5: Spectrogram 2
$l = 2$ s, $H = 1$ s

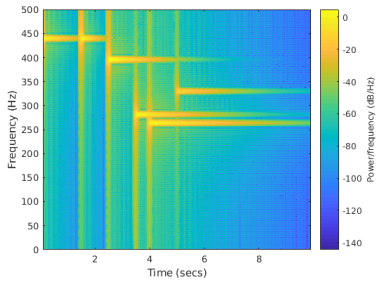

Figure 6: Spectrogram 3
$l = 256$ ms, $H = 6$ ms

Although these spectrograms have the same signal function, one can remark errors in the first two ones. For example the third and fourth tones are played in a quick succession with a time offset of 0.5 seconds. Due to the time window and hop size in Figure 4 the two tones start at the same reference point. The same applies for the second and third tone played in Figure 5. Also the first two strikes gets merged in Figure 4. However, in Figure 5 one can tell a slight difference between them, but an algorithm with a low sensitivity might not distinguish between the two.

In conclusion one has to find an appropriate window length l and hop size H to not analyse to big portions around the reference point and to not exceed the memory usage and time demand of the algorithms running on the target device. Furthermore, the computational processing of music using the STFT and spectrograms will always create errors due to the restricion to finite values. Thus the resulting sound files and played music will always possess a lower quality than the unprocessed ones. Furthermore, sound samples can differ in just tiny nuances, so the correctness of such algortihms is correlated to the memory usage and resoulution of the transformed signals. However, if one knows and compensates the properties of the STFT, it is an essential tool, which allows the digital processing of signals and espacially music of today's scale.

[4]We only analyse the effect of the window length l in seconds here

7 Fourier transform as Filter

Since any Fourier transform enables the access to the frequency domain of a signal, one can modulate the spectrum very easily. The strategy of a filter is to transform the signal into its spectrum, multiply a function with the amplitudes of frequencies and then use the inverse transform to get a filtered signal. In the following section we want to further explain this method by describing the implementation of low-,high- and band-pass filter. Furthermore one can achieve any filtering and modulation of the spectrum as long as the product of the filter function and the spectrum converges. This technique can also be used to implement an equalizer.

7.1 Low-, High- and Band-Pass Filter

Like the name implies, a low-/high-pass filter cuts off any frequency above/under a certain cut-off frequency. Usually they're achieved through resistors, capacitors and inductors as a hardware implementation. Along with the Fourier transform one can accomplish a similar result through a software filter. The filter function f as an ideal low-/high-pass is defined by:

$$f_{\text{low}}(\omega) := \begin{cases} 1 & \text{, if } \omega \leq \omega_{\text{cut-off}} \\ 0 & \text{, if } \omega > \omega_{\text{cut-off}} \end{cases}, \quad f_{\text{high}}(\omega) := \begin{cases} 1 & \text{, if } \omega \geq \omega_{\text{cut-off}} \\ 0 & \text{, if } \omega < \omega_{\text{cut-off}} \end{cases}$$

By combining the two filter, one receives the ideal band-pass filter:

$$f_{\text{band}}(\omega) := \begin{cases} 1 & \text{, if } \omega \in [\omega_l, \omega_u] \\ 0 & \text{, otherwise} \end{cases},$$

with ω_l and ω_u as the lower and upper cut-off frequencies. The desired filtering is now achieved by multiplying f with the spectrum.

The advantage of using software filters is, that every frequency above or below the threshold gets negated completely, which makes them ideal filter instead, which is not the case with the hardware way. But since this filter is a software implementation, — regarding 6.4 — one has to expect a loss in sound quality after retransforming the signal due to the sampling of the time domain and quantisation of the frequency domain. On the contrary, using hardware filter avoids quality loss through discretisation, but due to the physical properties of inductors and capacitors one can't achieve a perfect filter, since a capacitor/inductor needs a certain amount of time to (dis)charge its electric/magnetic field.

8 Occurring Effects

8.1 Nyquist-Shannon sampling theorem

Analysing signals using DFT limits the detectable frequencies. Equation 12 gives the coefficient for a frequency $m\omega_0$ where $m < N$. 5.1 shows the symmetry of the frequency

domain for DFT. This domain also is periodic. If it contains a frequency for an $m, N > m > \frac{N}{2}$, it also appears for $m' = m - N$ with $-\frac{N}{2} < m' < 0$. According to the symmetry the frequency is also represented by $m'' = -m'$ with $0 < m'' < \frac{N}{2}$ where there is the complex conjugation of the former coefficient. m'' also represents the frequency $m'' \cdot \omega_0$. This leads to the former frequency being interpreted as another frequency in the frequency domain. To prefent this m should be limited in a way that only allows N frequencies symmetric to zero: $m \leq \frac{N}{2}$.

$$f_{max} = \frac{1}{2\pi}\omega_{max} = \frac{1}{2\pi}\frac{N}{2} \cdot \omega_0 = \frac{1}{2\pi}\frac{N \cdot 2\pi}{2T} = \frac{1}{2}\frac{N}{T} = \frac{1}{2} \cdot f_{sample}$$

To conclude for a given signal the sampling frequency should be at least twice as high as the signal's highest contained frequency in order to detect and interpret the signal's frequencies correctly.

8.2 Leakage

Using time windows for the discrete short-time Fourier transform can lead to peaks in the frequency domain even if their frequency is not present in the signal. These peaks generally are positioned very close to each other. Their amplitudes decrease if the distance to the main peak increases. Figure 4 does not only show a single line representing a frequency of the source signal but also frequencies directly next to it not being zero. This leakage is caused by several factors as jumps in the time domain which can appear if the time window does not hit a period of the signal's current behaviour exactly.

The peaks can be reduced using a bell curve as window function. This prevents the jumps because the transformed function equals zero in its time domain at its borders but it also leads to peaks for frequencies absent in the source signal. The Hann window is one frequently used option for the needed window function ([2], page 98).

9 Appendix

```
1  private Complex[] recursiveFFT(Complex[] x) {
2     if (x.length == 1) {
3        return x;
4     } else {
5        Complex[] xEven = new Complex[x.length / 2];
6        Complex[] xOdd = new Complex[x.length / 2];
7        for (int i = 0; i < x.length / 2; i++) {
8           xEven[i] = x[2 * i];
9           xOdd[i] = x[2 * i + 1];
10       }
11       Complex[] ftXEven = recursiveFFT(xEven);
12       Complex[] ftXOdd = recursiveFFT(xOdd);
13       Complex[] ftX = new Complex[x.length];
14       for (int i = 0; i < x.length / 2; i++) {
15          Complex factor = new Complex(Math.cos(-2 * Math.PI * i / x.length),
16                Math.sin(-2 * Math.PI * i / x.length));
17          ftX[i] = Complex.sum(ftXEven[i], Complex.product(factor, ftXOdd[i]));
18          ftX[i + x.length / 2] = Complex.difference(ftXEven[i],
                Complex.product(factor, ftXOdd[i]));
19       }
20       return ftX;
21    }
22 }
```

Figure 7: Fast Fourier transform's implementation in Java, time to frequency domain

References

[1] Truong Nguyen Gilbert Strang. *Wavelets and Filter Banks*. Wellesley-Cambridge Press, 1 edition, 1996.

[2] Meinard Müller. *Fundamentals of Music Processing*. Springer Verlag, 1 edition, 2015.

[3] Thomas Westermann. Diskrete Fourier-Transformation und Anwendungen. Part of "Ergänzende Kapitel" of the book "Mathematik für Ingenieure": `http://www.home.hs-karlsruhe.de/~weth0002/buecher/mathe/downloads/kap18ext1059.pdf`. Accessed: 2017-11-13.