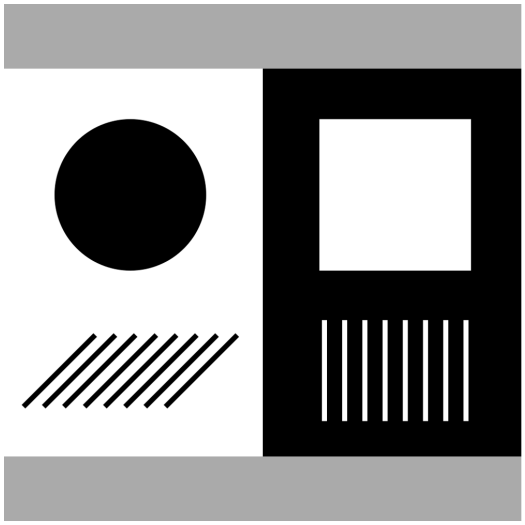
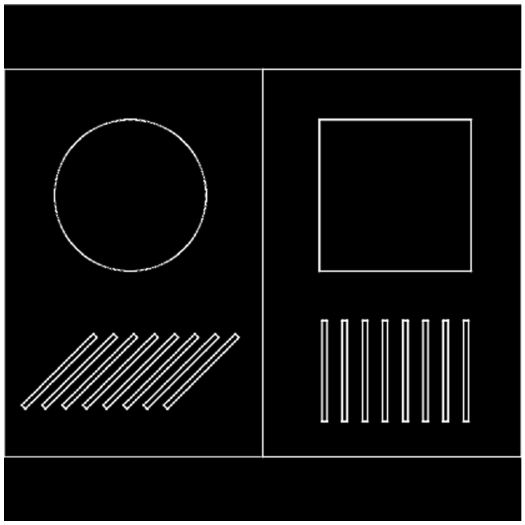
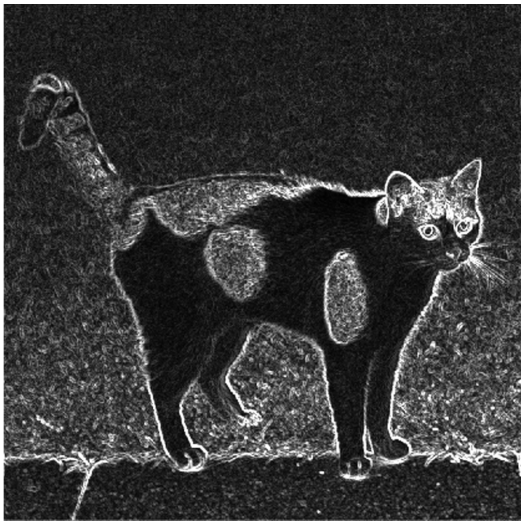
University of Basel

Gabor Transform and Vision

Master thesis in Mathematics



Date June 5, 2018
Author Andreas Aeschlimann
Supervisors Prof. Dr. Helmut Harbrecht
Prof. Dr. Lukas Rosenthaler

I would like to thank Prof. Dr. Helmut Harbrecht and
Prof. Dr. Lukas Rosenthaler for their excellent assistance
in the course of the writing of this thesis.

Special thanks also go to my girlfriend and my family
for supporting me in all these years of my studies.

Contents

1 Introduction

Fourier analysis has proved to be a useful tool since the early 19th century. Applying the Fourier transform to an audio signal or an image reveals the frequency spectrum of the whole space domain. However, input data is often non-stationary in real-world applications. For instance, an audio file with music has changing frequencies over time. Hence, new approaches have been searched during the last few decades. One of the newer methods is the Gabor transform, which has been first introduced by Dennis Gabor in 1946 (cf. [Gab46]).

In this thesis, we introduce the reader to the Gabor transform in one and two dimensions. In summary, the Gabor transform is a windowed Fourier transform with a Gaussian window function. In comparison to the Fourier transform, it provides a dependency on both space domain and frequency domain. Thus, the Gabor transform is a useful tool to study time-frequency dependencies for audio signals—or pixel-frequency dependencies for images. As of today, two-dimensional Gabor filters are indeed being widely used for image analysis, image compression, object recognition, medical diagnostics, and many more fields.

In order to make the reader familiar with all the necessary theory, we first present basics of the Fourier and then the Gabor analysis in one dimension. The main focus of this thesis lies on understanding the theory and on developing own implementations. To begin, the continuous Fourier transform will be presented in Chapter 2. Afterwards, we shall introduce the discrete Fourier transform in Chapter 3, at which end some MATLAB code is provided. Chapter 4 contains the Gabor transform and completes the study of one-dimensional signals.

Having seen the Gabor transform in one dimension, we are going to develop the same techniques in two dimensions. Chapter 5 first deals with the two-dimensional Fourier transform, which is again basis for the Gabor transform discussed in Chapter 6. To conclude the two-dimensional Gabor transform, we are going to use our own implementation to discuss some typical images.

At the end of the thesis, the reader may find two extensive MATLAB classes to get started on the discrete Fourier and Gabor transform. Based on this code, we also provide a small web application as a playground for the Gabor transform in two dimensions. This app may be found on `https://andreas-aeschlimann.github.io/gabor/`.

2 Fourier transform in one dimension

2.1 Motivation

Before implementing the discrete Gabor transform in one or two dimensions, we require to study the continuous Gabor transform. The Gabor transform is strongly related to the Fourier transform, which motivates the study of Fourier analysis.

For better understanding, we start with basic definitions required for the Fourier transform. After that, we dive into the Fourier transform for L^1 and L^2 functions on the real line. Especially the construction and study of the Fourier transform for L^2 functions will prove useful for the Gabor transform.

Throughout this chapter, we are using various results from [DS15, Chapters 2 and 3], [EG92, Chapter 4.3] and [Föl11, Chapter 10].

2.2 Preliminaries

We are going to work with functions defined in $\mathbb{R}^d$ for $d \in \{1, 2\}$ in this thesis. We henceforth reserve the variable $d \in \mathbb{N}$ for the dimension of the space.

This section contains the most important definitions needed for the introduction of the Fourier transform in one dimension. Most of these definitions can be directly applied to more than one dimension; we thus provide the general forms where possible.

In order to tell apart a one-dimensional from a multidimensional object, we will always write the latter in bold face. For example, we may sometimes write $\boldsymbol{x} = (x, y) \in \mathbb{R}^2$, which means that we treat $\boldsymbol{x} \in \mathbb{R}^2$ as a different variable as $x \in \mathbb{R}$.

Definition 2.1 (L^p spaces)**.** *Let $p \in \mathbb{R}$ with $1 \leq p < \infty$ and let $\Omega \subset \mathbb{R}^d$ be a Lebesgue-measurable set. Additionally, let $f : \Omega \to \mathbb{C}$ be a Lebesgue-measurable function. We say that $f \in L^p(\Omega)$ if*

$$\|f\|_{L^p(\Omega)} = \left(\int_\Omega |f(\boldsymbol{x})|^p d\boldsymbol{x} \right)^{1/p} < \infty.$$

Remark 2.2. *(i) One can prove that such $L^p(\Omega)$ are Banach spaces.*

(ii) $L^2(\Omega)$ is particularly interesting because it is a Hilbert space with the inner product $\langle f, g \rangle_{L^2(\Omega)} = \int_\Omega f(\boldsymbol{x}) \overline{g(\boldsymbol{x})} d\boldsymbol{x}$.

(iii) If a function is in $L^1(\mathbb{R}^d)$, we say it is Lebesgue-integrable.

We are primarily interested in the cases $p = 1$ and $p = 2$. As we often use the $L^2(\mathbb{R}^d)$ norms, we use the common notations

$$\|f\|_p = \|f\|_{L^p(\mathbb{R}^d)} \quad \text{and} \quad \|f\| = \|f\|_{L^2(\mathbb{R}^d)}.$$

The same applies for the inner product space $L^2(\mathbb{R}^d)$ where we will simply write

$$\langle f, g\rangle = \langle f, g\rangle_{L^2(\mathbb{R}^d)}.$$

Definition 2.3 (Convolution)**.** *Let $f, g : \mathbb{R}^d \to \mathbb{C}$ be two functions in $L^1(\mathbb{R}^d)$. The convolution of f and g, written $f * g$, is defined by*

$$(f * g)(\boldsymbol{x}) = \int_{\mathbb{R}^d} f(\boldsymbol{x} - \boldsymbol{y})g(\boldsymbol{y})d\boldsymbol{y}.$$

Remark 2.4. *(i) The convolution is well-defined because $f(\boldsymbol{x} - \boldsymbol{y})g(\boldsymbol{y})$ is integrable for almost all $\boldsymbol{x} \in \mathbb{R}^d$.*

(ii) The resulting $f * g$ exists for almost every $\boldsymbol{x} \in \mathbb{R}^d$ and is integrable.

(iii) Using a change of variables and the theorem of Fubini, it is easy to check that the convolution is commutative, associative and distributive.

Definition 2.5 (Continuous function spaces)**.** *We define the following function spaces:*

(1) We say $f \in C_c(\mathbb{R}^d)$ if $f : \mathbb{R}^d \to \mathbb{C}$ is a continuous function with compact support.

(2) We say $f \in C_0(\mathbb{R}^d)$ if $f : \mathbb{R}^d \to \mathbb{C}$ is a continuous function that vanishes at infinity, that means $f(\boldsymbol{x}) \to 0$ for $|\boldsymbol{x}| \to \infty$.

Remark 2.6. *(i) Note that $C_c(\mathbb{R}^d) \subset C_0(\mathbb{R}^d) \subset L^p(\mathbb{R}^d)$ for all $1 \leq p < \infty$.*

(ii) Additionally, $C_c(\mathbb{R}^d)$ is dense in $L^p(\mathbb{R}^d)$. This means that every function in $L^p(\mathbb{R}^d)$ can be approximated by a sequence of continuous functions with compact support.

2.3 Fourier transform in $L^1(\mathbb{R})$

Assume that we have a function $f \in L^1(\mathbb{R})$. By Definition 2.1, $\|f\|_1 < \infty$. We deduce that

$$\int_{-\infty}^{\infty} |f(t)e^{-i\omega t}|dt \leq \int_{-\infty}^{\infty} |f(t)| \underbrace{|e^{-i\omega t}|}_{=1} dt = \int_{-\infty}^{\infty} |f(t)| = \|f\|_1 < \infty$$

and hence the expression $f(t)e^{-i\omega t}$ is in $L^1(\mathbb{R})$. This result enables the following definition:

Definition 2.7 (Fourier transform in $L^1(\mathbb{R})$)**.** *Let $f \in L^1(\mathbb{R})$. The Fourier transform of f is defined by*

$$\mathcal{F}\{f(t)\} = \hat{f}(\omega) = \int_{-\infty}^{\infty} f(t)e^{-i\omega t}dt,$$

where $t, \omega \in \mathbb{R}$.

Remark 2.8. *The resulting function $\hat{f}(\omega)$ is also called frequency spectrum. Physically, it measures oscillations of f at the frequencies ω.*

Example 2.9. *Consider $f(t) = e^{-\beta^2 t^2}$ with $\beta > 0$. Clearly, f is in $L^1(\mathbb{R})$, because*

$$\|f\|_1^2 = \int_{-\infty}^{\infty} |f(t)|dt = \int_{-\infty}^{\infty} |e^{-\beta^2 t^2}|dt = \frac{1}{\beta}\int_{-\infty}^{\infty} |e^{-s^2}|ds = \tfrac{\sqrt{\pi}}{\beta} < \infty.$$

Applying Definition 2.7 results in

$$\begin{aligned}
\hat{f}(\omega) &= \int_{-\infty}^{\infty} f(t)e^{-i\omega t}dt = \int_{-\infty}^{\infty} e^{-\beta^2 t^2}e^{-i\omega t}dt \\
&= \int_{-\infty}^{\infty} \exp\left(-\beta^2\left[t + \tfrac{\omega}{2\beta^2}i\right]^2 - \tfrac{\omega^2}{4\beta^2}\right)dt \\
&= \exp\left(-\tfrac{\omega^2}{4\beta^2}\right)\int_{-\infty}^{\infty} \exp\left(-\beta^2\left[t + \tfrac{\omega}{2\beta^2}i\right]^2\right)dt \\
&= \exp\left(-\tfrac{\omega^2}{4\beta^2}\right)\int_{-\infty}^{\infty} e^{-\beta^2 x^2}dx \\
&= \exp\left(-\tfrac{\omega^2}{4\beta^2}\right)\tfrac{\sqrt{\pi}}{\beta},
\end{aligned}$$

with the change of variables $x = t + \frac{\omega}{2a^2}i$.

Remark 2.10. *Example 2.9 shows that $f(t) = e^{-t^2/2}$ is an eigenfunction of the Fourier transform operator $\mathcal{F}$ to the eigenvalue $\sqrt{2\pi}$.*

As the Fourier transform only consists of integration and multiplication, it retains many useful properties.

Theorem 2.11 (Basic properties of the Fourier transform)**.** *Let f, g be functions in $L^1(\mathbb{R})$ and let the scalar values $a, b \in \mathbb{C}$, $r \in \mathbb{R}$. Then the following statements hold:*

(1) $\mathcal{F}\{a\, f(t) + b\, g(t)\} = a\, \hat{f}(\omega) + b\, \hat{g}(\omega)$ ($\mathcal{F}$ is a linear operator),

(2) $\mathcal{F}\{f(t-r)\} = e^{-i\omega r}\hat{f}(\omega)$ (translation results in modulation),

(3) $\mathcal{F}\{e^{irt}f(t)\} = \hat{f}(\omega - r)$ *(modulation results in translation),*

(4) $\mathcal{F}\{\overline{f(t)}\} = \overline{\hat{f}(-\omega)}$ *(conjugation),*

(5) $\mathcal{F}\{f(rt)\} = \frac{1}{r}\,\hat{f}(\frac{\omega}{r})$, $r \neq 0$ *(scaling).*

Proof. (1) Follows directly from Definition 2.7.

(2) A change of variables by $x = t - r$ gives the result:

$$\begin{aligned}\mathcal{F}\{f(t-r)\} =& \int_{-\infty}^{\infty} f(t-r)e^{-i\omega t}dt = \int_{-\infty}^{\infty} f(x)e^{-i\omega(x+r)}dx \\ =& e^{-i\omega r}\int_{-\infty}^{\infty} f(x)e^{-i\omega x}dx = e^{-i\omega r}\hat{f}(\omega).\end{aligned}$$

(3) By applying Definition 2.7, we obtain

$$\begin{aligned}\mathcal{F}\{e^{irt}f(t)\} =& \int_{-\infty}^{\infty} e^{irt}f(t)e^{-i\omega t}dt = \int_{-\infty}^{\infty} f(t)e^{-i(\omega-r)t}dt \\ =& \hat{f}(\omega - r).\end{aligned}$$

(4) Applying conjugation multiple times results in

$$\begin{aligned}\mathcal{F}\{\overline{f(t)}\} =& \int_{-\infty}^{\infty} \overline{f(t)}e^{-i\omega t}dt = \int_{-\infty}^{\infty} \overline{f(t)e^{-i(-\omega)t}}dt \\ =& \overline{\int_{-\infty}^{\infty} f(t)e^{-i(-\omega)t}dt} = \overline{\hat{f}(-\omega)}.\end{aligned}$$

(5) The scaling property follows by change of variables $x = rt$:

$$\begin{aligned}\mathcal{F}\{f(rt)\} =& \int_{-\infty}^{\infty} f(rt)e^{-i\omega t}dt = \int_{-\infty}^{\infty} f(x)e^{-i\omega x/r}\frac{1}{r}dx \\ =& \frac{1}{r}\int_{-\infty}^{\infty} f(x)e^{-i(\omega/r)x}dx = \frac{1}{r}\hat{f}\left(\frac{\omega}{r}\right).\end{aligned}$$

□

Remark 2.12. *One could assume that* $\bar{\hat{f}} = \hat{\bar{f}}$*, but this is wrong in general. As shown in Theorem 2.11 in the fourth item, we have* $\hat{\bar{f}}(w) = \bar{\hat{f}}(-w)$ *instead.*

Theorem 2.13 (Continuity of the Fourier transform)**.** *If* $f \in L^1(\mathbb{R})$*, then* $\hat{f}(\omega)$ *is continuous on* $\mathbb{R}$*.*

Proof. Consider the following expression for $w, h \in \mathbb{R}$:

$$\begin{aligned}
|\hat{f}(\omega+h) - \hat{f}(w)| &= \left| \int_{-\infty}^{\infty} f(t) e^{-i(\omega+h)t} dt - \int_{-\infty}^{\infty} f(t) e^{-i\omega t} dt \right| \\
&= \left| \int_{-\infty}^{\infty} f(t)(e^{-i\omega t} e^{-iht} - e^{-i\omega t}) dt \right| \\
&= \left| \int_{-\infty}^{\infty} f(t) e^{-i\omega t}(e^{-iht} - 1) dt \right| \\
&\leq \int_{-\infty}^{\infty} |f(t)| \underbrace{|e^{-i\omega t}|}_{=1} |e^{-iht} - 1| dt = \int_{-\infty}^{\infty} |f(t)||e^{-iht} - 1| dt.
\end{aligned}$$

First, we discover that

$$|f(t)||e^{-iht} - 1| \leq 2|f(t)|, \qquad \lim_{h \to 0} |f(t)||e^{-iht} - 1| \to 0$$

for any fixed $t \in \mathbb{R}$. By the dominated convergence theorem (cf. [EG92, Chapter 1.3, Theorem 3]), we can deduce

$$\lim_{h \to 0} \left| \hat{f}(\omega+h) - \hat{f}(w) \right| \leq \lim_{h \to 0} \int_{-\infty}^{\infty} |f(t)||e^{-iht} - 1| dt = 0.$$

This proves that $\hat{f}$ is indeed continuous on $\mathbb{R}$. □

Another interesting fact about the Fourier transform is stated in the Riemann-Lebesgue lemma. It says that the Fourier transform of an L^1 function vanishes at infinity:

Theorem 2.14 (Riemann-Lebesgue lemma)**.** *Let $f \in L^1(\mathbb{R})$. Then*

$$\lim_{|\omega| \to \infty} |\hat{f}(\omega)| = 0.$$

Proof. We are using the trick that $e^{-i\omega t} = -e^{-i\omega t - i\pi} = -e^{-i\omega\left(t + \frac{\pi}{\omega}\right)}$ for $\omega \neq 0$. By Definition 2.7 and the change of variables $x = t + \frac{\pi}{\omega}$, we obtain

$$\hat{f}(w) = -\int_{-\infty}^{\infty} f(t) e^{-i\omega\left(t+\frac{\pi}{\omega}\right)} dt = -\int_{-\infty}^{\infty} f\left(x - \frac{\pi}{w}\right) e^{-i\omega x} dx.$$

We then write $\hat{f}(w) = \frac{1}{2}(\hat{f}(w) + \hat{f}(w))$ by using the above expression once. Hence

$$\begin{aligned}
\hat{f}(w) &= \frac{1}{2}\left(\int_{-\infty}^{\infty} f(t) e^{-i\omega t} dt - \int_{-\infty}^{\infty} f\left(x - \frac{\pi}{w}\right) e^{-i\omega x} dx \right) \\
&= \frac{1}{2}\left(\int_{-\infty}^{\infty} \left[f(t) - f\left(t - \frac{\pi}{\omega}\right) \right] e^{-i\omega t} dt \right).
\end{aligned}$$

Again, by applying the dominated convergence theorem, we deduce

$$\begin{aligned}\lim_{|\omega|\to\infty} |\hat{f}(\omega)| &\leq \frac{1}{2} \lim_{|\omega|\to\infty} \int_{-\infty}^{\infty} \left| f(t) - f\left(t - \frac{\pi}{\omega}\right)\right| \underbrace{\left|e^{-i\omega t}\right|}_{=1} dt \\ &= \frac{1}{2} \lim_{|\omega|\to\infty} \int_{-\infty}^{\infty} \left| f(t) - f\left(t - \frac{\pi}{\omega}\right)\right| dt \\ &= 0.\end{aligned}$$

This finishes the proof. □

Remark 2.15. *The Riemann-Lebesgue lemma also holds in higher dimensions, i.e. when $f \in L^1(\mathbb{R}^n)$ for $n = 2, 3, \ldots$. The proof in higher dimensions is similar.*

We now have shown: If a function f is in $L^1(\mathbb{R})$, then its Fourier transform $\hat{f}$ is continuous in $\mathbb{R}$ and decays at infinity, that is $\hat{f} \in C_0(\mathbb{R})$.

Having studied the most important properties of the Fourier transform, we may now have a look at its inverse.

Definition 2.16 (Inverse Fourier transform)**.** *If $f \in L^1(\mathbb{R})$, then we define the inverse Fourier transform as*

$$\check{f}(t) = \mathcal{F}^{-1}\{f(\omega)\} = \frac{1}{2\pi} \int_{-\infty}^{\infty} f(\omega) e^{i\omega t} d\omega,$$

where $t, \omega \in \mathbb{R}$.

Remark 2.17. *(i) Similarly, as mentioned before Definition 2.7, the integral converges for every $t \in \mathbb{R}$.*

(ii) Assume that we have $f \in L^1(\mathbb{R})$ and its Fourier transform $\hat{f}$ is in $L^1(\mathbb{R})$ as well. Applying the inverse Fourier transform to $\hat{f}$ will only result in the original f almost everywhere. If f is additionally continuous, then we get the identity $\mathcal{F}^{-1}\{\mathcal{F}\{f(t)\}\} = f(t)$ for all $t \in \mathbb{R}$.

Theorem 2.18 (Uniqueness)**.** *If $f, g \in L^1(\mathbb{R})$ such that $\hat{f} = \hat{g}$, then $f = g$ almost everywhere.*

Remark 2.19. *A proof may be delivered by the construction of summability kernels. Details can be found in [DS15, Chapter 3.3].*

To conclude our studies of the Fourier transform in L^1, we are going to look at the correlation between multiplication and convolution in L^1 and in the Fourier space.

Theorem 2.20 (Convolution theorem)**.** *Let $f, g \in L^1(\mathbb{R})$, then*

$$\mathcal{F}\{(f * g)(t)\} = \hat{f}(\omega)\hat{g}(\omega).$$

Proof. By the remark to Definition 2.3, $f * g \in L^1(\mathbb{R})$. We may simply apply Definition 2.7 and use Fubini's theorem (cf. [EG92, Chapter 1.4, Theorem 1]) to deduce

$$\begin{aligned}
\mathcal{F}\{(f * g)(t)\} &= \int_{-\infty}^{\infty} e^{-i\omega t} \int_{-\infty}^{\infty} f(t-u)g(u)du dt \\
&= \int_{-\infty}^{\infty} \int_{-\infty}^{\infty} f(t-u)g(u)e^{-i\omega t} du dt \\
&\overset{\text{(Fubini)}}{=} \int_{-\infty}^{\infty} \int_{-\infty}^{\infty} f(t-u)g(u)e^{-i\omega t} dt du \\
&= \int_{-\infty}^{\infty} g(u) \underbrace{\int_{-\infty}^{\infty} f(t-u)e^{-i\omega t} dt}_{=I} du.
\end{aligned}$$

Finally, we may set $v = t - u$ for the inner integral I to obtain

$$I = \int_{-\infty}^{\infty} f(v)e^{-i\omega(u+v)} dv = e^{-i\omega u} \int_{-\infty}^{\infty} f(v)e^{-i\omega v} dv = e^{-i\omega u}\hat{f}(w).$$

We conclude,

$$\begin{aligned}
\mathcal{F}\{(f * g)(t)\} &= \int_{-\infty}^{\infty} g(u)e^{-i\omega u}\hat{f}(w) du \\
&= \hat{f}(w) \int_{-\infty}^{\infty} g(u)e^{-i\omega u} du \\
&= \hat{f}(w)\hat{g}(w).
\end{aligned}$$

□

Theorem 2.21 (General modulation)**.** *Let $\hat{f}, \hat{g} \in L^1(\mathbb{R})$, then*

$$\mathcal{F}\{f(t)g(t)\} = \frac{1}{2\pi}(\hat{f} * \hat{g})(w).$$

Proof. We may use Definition 2.16 and again Fubini's theorem to get

$$\begin{aligned}
\mathcal{F}\{f(t)g(t)\} &= \int_{-\infty}^{\infty} f(t)g(t)e^{-i\omega t}dt \\
&= \int_{-\infty}^{\infty} f(t)\left(\frac{1}{2\pi}\int_{-\infty}^{\infty}\hat{g}(x)e^{ixt}dx\right)e^{-i\omega t}dt \\
&= \frac{1}{2\pi}\int_{-\infty}^{\infty}\int_{-\infty}^{\infty} f(t)\hat{g}(x)e^{-i\omega t}e^{ixt}dxdt \\
\overset{\text{(Fubini)}}{} &= \frac{1}{2\pi}\int_{-\infty}^{\infty}\int_{-\infty}^{\infty} f(t)\hat{g}(x)e^{-i\omega t}e^{ixt}dtdx \\
&= \frac{1}{2\pi}\int_{-\infty}^{\infty}\underbrace{\left(\int_{-\infty}^{\infty} f(t)e^{-i(\omega-x)t}dt\right)}_{=\hat{f}(w-x)}\hat{g}(x)dx \\
&= \frac{1}{2\pi}\int_{-\infty}^{\infty}\hat{f}(w-x)\hat{g}(x)dx = \frac{1}{2\pi}(\hat{f}*\hat{g})(\omega).
\end{aligned}$$

Note that all integrals exist because $\hat{f}*\hat{g} \in L^1(\mathbb{R})$. This concludes the proof. □

In this section, we have seen two limitations so far: We have only worked on a domain of a single dimension ($\mathbb{R}$) and considered functions in $L^1(\mathbb{R})$. While many of the properties can be easily extended to $L^1(\mathbb{R}^n)$, there are many simple functions as $f(t) = \sin(t)$ or $g(t) = 1$ that are not even in $L^1(\mathbb{R})$. As such functions appear regularly in applications, we are inspired to use more general function classes to develop the Fourier transform.

For the theory of the Gabor transform, we require Fourier analysis for functions in $L^2(\mathbb{R})$. We hereby conclude the section about Fourier transform in $L^1(\mathbb{R})$ and thus proceed with the Fourier transform in $L^2(\mathbb{R})$.

2.4 Fourier transform in $L^2(\mathbb{R})$

In this section, we will see that the Fourier transform of a function in $L^2(\mathbb{R})$ is again in $L^2(\mathbb{R})$. Even better, Plancherel's theorem will show that the Fourier transform is a Hilbert space isomorphism of $L^2(\mathbb{R})$ onto $L^2(\mathbb{R})$.

In general, we cannot provide a formula for the Fourier transform in $L^2(\mathbb{R})$. We are going to construct the definition by the limit of sequences of $C_c(\mathbb{R})$ functions instead. It is important to note that the definition of the Fourier transform in $L^1(\mathbb{R})$ and $L^2(\mathbb{R})$ only coincide almost everywhere for functions living in both spaces.

Remark 2.22. *(i) For the following theorem, we need Parseval's formula for complete orthonormal sequences in a Hilbert space. It says that if $(f_n)_{n\in\mathbb{N}}$ is a complete orthonormal sequence in a Hilbert space H, then*

$$\|f\|^2 = \sum_{n=1}^{\infty} |\langle f, f_n\rangle|^2,$$

where $\|.\|$ is a norm in H.

(ii) Note that $(f_n)_{n\in\mathbb{Z}}$ defined by

$$f_n(t) = \frac{1}{\sqrt{2\pi}} e^{int}$$

is a complete orthonormal sequence in the Hilbert space $H = L^2([-\pi, \pi])$. As this is not easy to prove, we refer to [DS15, Chapters 2.10 and 2.11] for further details.

Theorem 2.23. *Let $f : \mathbb{R} \to \mathbb{R}$ be a continuous function that is compactly supported. Then, $\hat{f} \in L^2(\mathbb{R})$ and*

$$\|\hat{f}\| = \sqrt{2\pi}\, \|f\|.$$

Proof. (i) To start, assume that f vanishes outside the interval $\Omega := [-\pi, \pi]$. Define f_n as in the remark above, hence

$$\begin{aligned}\|f\|^2 = \sum_{n=-\infty}^{\infty} |\langle f, f_n\rangle|^2 &= \sum_{n=-\infty}^{\infty} \left| \frac{1}{\sqrt{2\pi}} \int_{-\infty}^{\infty} f(t) e^{-int} dt \right|^2 \\ &= \frac{1}{2\pi} \sum_{n=-\infty}^{\infty} \left| \hat{f}(n) \right|^2.\end{aligned}$$

The last equality follows because $f \in C_c(\mathbb{R}) \subset L^1(\mathbb{R})$ and Definition 2.7.

(ii) But this result could also be applied to a function $g(t) = f(t)e^{-ixt}$. We deduce that

$$\begin{aligned}\|g\|^2 = \sum_{n=-\infty}^{\infty} |\langle g, f_n\rangle|^2 &= \sum_{n=-\infty}^{\infty} \left| \frac{1}{\sqrt{2\pi}} \int_{-\infty}^{\infty} f(t) e^{-ixt} e^{-int} dt \right|^2 \\ &= \frac{1}{\sqrt{2\pi}} \sum_{n=-\infty}^{\infty} \left| \int_{-\infty}^{\infty} f(t) e^{-i(n+x)t} dt \right|^2 = \frac{1}{2\pi} \sum_{n=-\infty}^{\infty} \left| \hat{f}(n+x) \right|^2.\end{aligned}$$

Additionally, the norms of f and g coincide, since

$$\|g\|^2 = \int_{-\infty}^{\infty} |g(t)|^2 dt = \int_{-\infty}^{\infty} |f(t)e^{-ixt}|^2 dt = \int_{-\infty}^{\infty} |f(t)|^2 dt = \|f\|^2.$$

(iii) Putting this together, we get

$$\|f\|^2 = \frac{1}{2\pi} \sum_{n=-\infty}^{\infty} \left|\hat{f}(n+x)\right|^2$$

and integrate this term with respect to x from 0 to 1. Thus,

$$\begin{aligned}\|f\|^2 &= \int_0^1 \|f\|^2 \, dx = \frac{1}{2\pi} \sum_{n=-\infty}^{\infty} \int_0^1 |\hat{f}(n+x)|^2 \, dx \\ &= \frac{1}{2\pi} \sum_{n=-\infty}^{\infty} \int_n^{n+1} |\hat{f}(\omega)|^2 \, d\omega = \frac{1}{2\pi} \int_{-\infty}^{\infty} |\hat{f}(\omega)|^2 d\omega \\ &= \frac{1}{2\pi} \|\hat{f}\|^2.\end{aligned}$$

(iv) Now assume that f does not vanish outside Ω. As f is compactly supported, we find $a > \pi$, such that f vanishes outside $[-a, a]$. Define $b := a/\pi$ and $g(t) = f(bt)$, hence g has its support in Ω.

Note that by the change of variables $t = bx$, we get

$$\|f\|^2 = \int_{-\infty}^{\infty} |f(t)|^2 dt \overset{(t=bx)}{=} b \int_{-\infty}^{\infty} |f(bx)|^2 dx = b \int_{-\infty}^{\infty} |g(x)|^2 dx = b \, \|g\|^2.$$

Remember that $f, g \in C_c(\mathbb{R}) \subset L^1(\mathbb{R})$, so we can apply Theorem 2.11:

$$\hat{g}(\omega) = \mathcal{F}\{g(t)\} = \mathcal{F}\{f(bt)\} = \frac{1}{b} \hat{f}\left(\frac{\omega}{b}\right), b > 0.$$

Using that and because g has its support in Ω, we shall apply the third item to g to obtain

$$\begin{aligned}\|f\|^2 = b \, \|g\|^2 &= \frac{b}{2\pi} \, \|\hat{g}\|^2 = \frac{b}{2\pi} \int_{-\infty}^{\infty} |\hat{g}(\omega)|^2 d\omega \\ &= \frac{b}{2\pi} \int_{-\infty}^{\infty} \left|\frac{1}{b} \hat{f}\left(\frac{\omega}{b}\right)\right|^2 d\omega \overset{(\omega=bx)}{=} \frac{b}{2\pi} \int_{-\infty}^{\infty} \left|\frac{1}{b} \hat{f}(x)\right|^2 b dx \\ &= \frac{1}{2\pi} \, \|\hat{f}\|^2,\end{aligned}$$

where another change of variables $\omega = bx$ has been applied.

(v) Clearly, $\hat{f} \in L^2(\mathbb{R})$ follows directly from the construction above. □

We have shown that the Fourier transform $\mathcal{F} : C_c(\mathbb{R}) \to L^2(\mathbb{R})$ is a continuous mapping. As $C_c(\mathbb{R})$ is dense in $L^2(\mathbb{R})$ and the Fourier transform is linear, we may extend it uniquely to a linear mapping $L^2(\mathbb{R}) \to L^2(\mathbb{R})$. This extension will be called Fourier transform on $L^2(\mathbb{R})$.

Definition 2.24 (Fourier transform in $L^2(\mathbb{R})$)**.** *Let $f \in L^2(\mathbb{R})$ and let $(f_n)_{n\in\mathbb{N}}$ be a sequence in $C_c(\mathbb{R})$ convergent to f in $L^2(\mathbb{R})$. Then the Fourier transform of f is defined by*

$$\hat{f} = \lim_{n\to\infty} \hat{f}_n.$$

Remark 2.25. *(i) Remember that the convergence of f_n and $\hat{f}_n$ is not pointwise, but in the L^2 norm, i.e. $\lim_{n\to\infty} \|f_n - f\| = 0$ and $\lim_{n\to\infty} \|\hat{f}_n - \hat{f}\| = 0$.*

(ii) The existence of the sequence f_n is given as $C_c(\mathbb{R})$ is dense in $L^2(\mathbb{R})$.

(iii) The existence of the Fourier transform of f_n is given because $C_c(\mathbb{R}) \subset L^1(\mathbb{R})$, so the theory of the last section applies.

(iv) The limit $\hat{f}$ exists because of Theorem 2.23. Indeed, we can verify that $\|\hat{f} - \hat{f}_n\| = \sqrt{2\pi}\,\|f - f_n\| \to 0$ as $n \to \infty$.

(v) This also shows that the limit $\hat{f}$ is independent of the choice of the approximating sequence f_n.

(vi) The Fourier transforms defined in $L^1(\mathbb{R})$ and $L^2(\mathbb{R})$ are not equal. But we can say that if $f \in L^1(\mathbb{R}) \cap L^2(\mathbb{R})$, then the Fourier transform defined by Definition 2.7 is in the equivalence class of the Fourier transform defined by Definition 2.24.

(vii) The Fourier transform of $f \in L^2(\mathbb{R})$ is not defined at a specific point. But we can assert that $\hat{f} \in L^2(\mathbb{R})$ and $\hat{f}$ is defined almost everywhere on $\mathbb{R}$.

Using Theorem 2.23, Definition 2.24 and Remark 2.25, we have proved the following theorem:

Theorem 2.26 (Parseval's identity)**.** *Let $f \in L^2(\mathbb{R})$, then*

$$\|\hat{f}\| = \sqrt{2\pi}\,\|f\|.$$

Theorem 2.27 (Fourier transform in $L^2(\mathbb{R})$)**.** *If $f \in L^2(\mathbb{R})$, then*

$$\hat{f}(\omega) = \lim_{n\to\infty} \int_{-n}^{n} f(t) e^{-i\omega t} dt.$$

Proof. Let $(f_n)_{n\in\mathbb{N}}$ a sequence such that

$$f_n(t) = \begin{cases} f(t), & -n < t < n, \\ 0, & \text{otherwise.} \end{cases} \tag{2.1}$$

Clearly, $f_n \in L^2(\mathbb{R})$ and from

$$\begin{aligned}\|f - f_n\|^2 &= \int_{-\infty}^{\infty} |f(t) - f_n(t)|^2 dt \\ &= \int_{-\infty}^{-n} |f(t)|^2 dt + \int_{-n}^{n} |f(t) - f(t)|^2 dt + \int_{n}^{\infty} |f(t)|^2 dt \\ &= \int_{-\infty}^{-n} |f(t)|^2 dt + \int_{n}^{\infty} |f(t)|^2 dt\end{aligned}$$

we obtain that $\|f - f_n\| \to 0$, hence $f_n \to f$ for $n \to \infty$. Using Theorem 2.26 we conclude

$$\|\hat{f} - \hat{f}_n\| = \sqrt{2\pi}\, \|f - f_n\| \to 0$$

as $n \to \infty$. It follows that

$$\hat{f}(\omega) = \lim_{n\to\infty} \hat{f}_n(\omega) = \lim_{n\to\infty} \int_{-\infty}^{\infty} f_n(t) e^{-i\omega t} dt = \lim_{n\to\infty} \int_{-n}^{n} f(t) e^{-i\omega t} dt.$$

□

Remark 2.28. *Many basic properties from the Fourier transform in $L^1(\mathbb{R})$—as shown in Theorem 2.11—also hold in $L^2(\mathbb{R})$.*

Theorem 2.29. *Let $f, g \in L^2(\mathbb{R})$. Then*

$$\langle \hat{f}, \bar{g} \rangle = \langle f, \bar{\hat{g}} \rangle.$$

Proof. Let $f, g \in L^2(\mathbb{R})$ and define f_m, g_n as in (2.1). Note that we have $f_m,\, g_n \in L^1(\mathbb{R})$ by construction. Clearly, the product $h(x, y) = e^{-ixy} f_m(y) g_n(x) \in L^1(\mathbb{R}^2)$. This allows us to use Fubini's theorem to obtain

$$\begin{aligned}\langle \hat{f_m}, \bar{g_n} \rangle &= \int_{-\infty}^{\infty} \hat{f}_m(x) g_n(x) dx = \int_{-\infty}^{\infty} \left(\int_{-\infty}^{\infty} f_m(y) e^{-ixy} dy \right) g_n(x) dx \\ &\overset{\text{(Fubini)}}{=} \int_{-\infty}^{\infty} \left(\int_{-\infty}^{\infty} g_n(x) e^{-ixy} dx \right) f_m(y) dy = \int_{-\infty}^{\infty} \hat{g}_n(y) f_m(y) dy \\ &= \langle f_m, \bar{\hat{g}_n} \rangle.\end{aligned}$$

We observe that the inner product is a continuous function, so we can swap the inner product with the limes. Hence, letting $m \to \infty$ leads to $\langle \hat{f}, \bar{g_n} \rangle = \langle f, \bar{\hat{g}_n} \rangle$. Similarly, letting $n \to \infty$ results in

$$\langle \hat{f}, \bar{g} \rangle = \langle f, \bar{\hat{g}} \rangle.$$

□

Theorem 2.30. *If $f \in L^2(\mathbb{R})$ and $g = \bar{\hat{f}}$, then $2\pi f = \bar{\hat{g}}$.*

Proof. From the definition of the inner product, Theorem 2.26 and Theorem 2.29, we deduce the following results:

(i) $\langle f, \bar{\hat{g}} \rangle = \langle \hat{f}, \bar{g} \rangle = \langle \hat{f}, \hat{f} \rangle = \|\hat{f}\|^2 = 2\pi \|f\|^2$,

(ii) $\langle \bar{\hat{g}}, f \rangle = \overline{\langle f, \bar{\hat{g}} \rangle} = \overline{\langle \hat{f}, \bar{g} \rangle} = \overline{\langle \hat{f}, \hat{f} \rangle} = \overline{\|\hat{f}\|^2} = \|\hat{f}\|^2 = 2\pi \|f\|^2$,

(iii) $\|\bar{\hat{g}}\|^2 = \|\hat{g}\|^2 = 2\pi \|g\|^2 = 2\pi \|\bar{\hat{f}}\|^2 = 2\pi \|\hat{f}\|^2 = (2\pi)^2 \|f\|^2$.

Putting this together gives us

$$\begin{aligned}
\|2\pi f - \bar{\hat{g}}\|^2 &= \langle 2\pi f - \bar{\hat{g}}, 2\pi f - \bar{\hat{g}} \rangle \\
&= \langle 2\pi f, 2\pi f \rangle + \langle 2\pi f, -\bar{\hat{g}} \rangle + \langle -\bar{\hat{g}}, 2\pi f \rangle + \langle -\bar{\hat{g}}, -\bar{\hat{g}} \rangle \\
&= (2\pi)^2 \|f\|^2 - 2\pi \langle f, \bar{\hat{g}} \rangle - 2\pi \langle \bar{\hat{g}}, f \rangle + \|\bar{\hat{g}}\|^2 \\
&\overset{\text{(item (i)–(iii))}}{=} (2\pi)^2 \|f\|^2 - (2\pi)^2 \|f\|^2 - (2\pi)^2 \|f\|^2 + (2\pi)^2 \|f\|^2 \\
&= 0.
\end{aligned}$$

This proves that $2\pi f = \bar{\hat{g}}$ in $L^2(\mathbb{R})$. □

Theorem 2.31 (Inverse Fourier transform in $L^2(\mathbb{R})$)**.** *Let $f \in L^2(\mathbb{R})$, then the inverse Fourier transform is defined by*

$$f(t) = \mathcal{F}^{-1}\{\hat{f}(\omega)\} = \lim_{n\to\infty} \frac{1}{2\pi} \int_{-n}^{n} \hat{f}(\omega) e^{i\omega t} d\omega,$$

where the limit is taken with respect to the $L^2(\mathbb{R})$ norm.

Proof. Let $f \in L^2(\mathbb{R})$ and $g = \bar{\hat{f}}$. Note that it follows directly from this definition that $\bar{g} = \hat{f}$. Using this, Theorem 2.27 and Theorem 2.30 ($2\pi f = \bar{\hat{g}}$), we deduce that

$$\begin{aligned}
2\pi f(t) = \overline{\hat{g}(t)} &= \overline{\lim_{n\to\infty} \int_{-n}^{n} g(\omega) e^{-i\omega t} d\omega} = \lim_{n\to\infty} \int_{-n}^{n} \overline{g(\omega) e^{-i\omega t} d\omega} \\
&= \lim_{n\to\infty} \int_{-n}^{n} \overline{g(\omega)} e^{i\omega t} d\omega = \lim_{n\to\infty} \int_{-n}^{n} \hat{f}(\omega) e^{i\omega t} d\omega.
\end{aligned}$$

Dividing by 2π delivers the statement of the theorem. □

Theorem 2.32 (General Parseval's relation)**.** *If $f, g \in L^2(\mathbb{R})$, then*

$$\langle \hat{f}, \hat{g} \rangle = 2\pi \langle f, g \rangle.$$

Proof. (i) Let $f, g \in L^2(\mathbb{R})$. As we know from Theorem 2.26 and by the linearity of the Fourier transform,

$$\|\hat{f} + \hat{g}\|^2 = \|\widehat{f+g}\|^2 = 2\pi \|f+g\|^2$$

holds.

(ii) Using the fact that for every $z \in \mathbb{C}$ we have $|z|^2 = z\bar{z}$, we obtain for the left-hand side

$$\begin{aligned}\|\hat{f} + \hat{g}\|^2 &= \int_{-\infty}^{\infty} |\hat{f}(\omega) + \hat{g}(\omega)|^2 d\omega = \int_{-\infty}^{\infty} \left(\hat{f}(\omega) + \hat{g}(\omega)\right)\left(\overline{\hat{f}(\omega)} + \overline{\hat{g}(\omega)}\right) d\omega \\ &= \|\hat{f}\|^2 + \int_{-\infty}^{\infty} \hat{f}(\omega)\overline{\hat{g}(\omega)} d\omega + \int_{-\infty}^{\infty} \overline{\hat{f}(\omega)}\hat{g}(\omega) d\omega + \|\hat{g}\|^2\end{aligned}$$

and for the right-hand side

$$\begin{aligned}\|f+g\|^2 &= \int_{-\infty}^{\infty} |f(t)+g(t)|^2 dt = \int_{-\infty}^{\infty} (f(t)+g(t))\left(\overline{f(t)} + \overline{g(t)}\right) dt \\ &= \|f\|^2 + \int_{-\infty}^{\infty} f(t)\overline{g(t)} dt + \int_{-\infty}^{\infty} \overline{f(t)} g(t) dt + \|g\|^2.\end{aligned}$$

Due to Theorem 2.26, we have $\|\hat{f}\|^2 = 2\pi \|f\|^2$ and $\|\hat{g}\|^2 = 2\pi \|g\|^2$, so we deduce from above that

$$\|\hat{f} + \hat{g}\|^2 = 2\pi \|f+g\|^2$$

is equivalent to

$$\int_{-\infty}^{\infty} \hat{f}(\omega)\overline{\hat{g}(\omega)} d\omega + \int_{-\infty}^{\infty} \overline{\hat{f}(\omega)}\hat{g}(\omega) d\omega = 2\pi \left(\int_{-\infty}^{\infty} f(t)\overline{g(t)} dt + \int_{-\infty}^{\infty} \overline{f(t)} g(t) dt\right).$$

(iii) As this equation holds for all $f, g \in L^2(\mathbb{R})$, we may also just replace g with $h = ig$. Clearly, $h \in L^2(\mathbb{R})$ and $\hat{h} = i\hat{g}$. Then the equation above yields

$$\begin{aligned}&\int_{-\infty}^{\infty} \hat{f}(\omega)\overline{(i\hat{g}(\omega))} d\omega + \int_{-\infty}^{\infty} \overline{\hat{f}(\omega)}(i\hat{g}(\omega)) d\omega \\ &\qquad = 2\pi \left(\int_{-\infty}^{\infty} f(t)\overline{(ig(t))} dt + \int_{-\infty}^{\infty} \overline{f(t)}(ig(t)) dt\right).\end{aligned}$$

Extracting i from the integrals and multiplying both sides with i results in the equivalent equation

$$\int_{-\infty}^{\infty} \hat{f}(\omega)\overline{\hat{g}(\omega)} d\omega - \int_{-\infty}^{\infty} \overline{\hat{f}(\omega)}\hat{g}(\omega) d\omega = 2\pi \left(\int_{-\infty}^{\infty} f(t)\overline{g(t)} dt - \int_{-\infty}^{\infty} \overline{f(t)} g(t) dt\right).$$

(iv) Finally, we may add the results from the second and third item to obtain that the following equation holds

$$\int_{-\infty}^{\infty} \hat{f}(\omega)\overline{\hat{g}(\omega)}d\omega = 2\pi \left(\int_{-\infty}^{\infty} f(t)\overline{g(t)}dt \right),$$

which is exactly what we had to prove. □

With the previous theorems we are now able to make some additional statements about the Fourier transform in $L^2(\mathbb{R})$.

Theorem 2.33 (Plancherel's theorem)**.** *For every $f \in L^2(\mathbb{R})$ there exists a $\hat{f} \in L^2(\mathbb{R})$ such that*

(1) $\hat{f}(\omega) = \lim_{n\to\infty} \int_{-n}^{n} f(t)e^{-i\omega t}dt$,

(2) $f(t) = \frac{1}{2\pi} \lim_{n\to\infty} \int_{-n}^{n} \hat{f}(\omega)e^{i\omega t}d\omega$,

(3) If $f \in L^1(\mathbb{R}) \cap L^2(\mathbb{R})$, then $\hat{f}(\omega) = \int_{-\infty}^{\infty} f(t)e^{-i\omega t}dt$,

(4) $\langle \hat{f}, \hat{g} \rangle = 2\pi \langle f, g \rangle$,

(5) $\|\hat{f}\| = \sqrt{2\pi}\, \|f\|$,

(6) The mapping $\mathcal{F} : L^2(\mathbb{R}) \to L^2(\mathbb{R})$ is a Hilbert space isomorphism.

Proof. (1) See Theorem 2.27—the Fourier transform in $L^2(\mathbb{R})$.

(2) See Theorem 2.31—the inverse Fourier transform in $L^2(\mathbb{R})$.

(3) See Definition 2.7, Definition 2.24 and Remark 2.25.

(4) See Theorem 2.32—general Parseval's relation.

(5) See Theorem 2.26—Parseval's identity.

(6) The mapping is a Hilbert space homomorphism by construction in Definition 2.24. It is clearly injective because of the fifth item. The surjectivity is the last thing to prove:

Fix $g \in L^2(\mathbb{R})$ in the image space. Further, let $h = \bar{g}$ and $f = \frac{1}{2\pi}\bar{\hat{h}}$. By Theorem 2.30, we obtain $h = \bar{\hat{f}}$, hence $g = \bar{h} = \hat{f}$. This means, for every g we have found a function $f \in L^2(\mathbb{R})$ such that $\hat{f} = g$. □

We now have established the basics of the Fourier analysis in $L^2(\mathbb{R})$. As a last step, we are going to investigate the result of multiple applications of the Fourier transform.

Theorem 2.34 (Duality of the Fourier transform)**.** *Let $f \in L^2(\mathbb{R})$, then*

$$\mathcal{F}\{\mathcal{F}\{f\}\}(x) = 2\pi f(-x)$$

for almost any $x \in \mathbb{R}$.

Proof. By Theorem 2.27 and Theorem 2.31, we have

$$\begin{aligned}\mathcal{F}\{\mathcal{F}\{f\}\}(x) &= \lim_{n\to\infty} \int_{-n}^{n} \hat{f}(\omega) e^{-ix\omega} d\omega \\ &= \lim_{n\to\infty} \int_{-n}^{n} \hat{f}(\omega) e^{i\omega(-x)} d\omega \\ &= 2\pi f(-x).\end{aligned}$$

As the Fourier transform is defined almost everywhere, we get the desired result. □

Remark 2.35. *The theorem above immediately implies that*

$$\mathcal{F}^4\{f\}(x) = (2\pi)^2 f(x)$$

for almost any $x \in \mathbb{R}$.

To conclude the theory of the continuous Fourier transform, we take note that its definition varies in different books. For example, some authors use the so-called "symmetrical" form, so that we have

$$\hat{f}(\omega) = \frac{1}{\sqrt{2\pi}} \int_{-\infty}^{\infty} f(t) e^{-i\omega t} dt, \qquad f(t) = \frac{1}{\sqrt{2\pi}} \int_{-\infty}^{\infty} \hat{f}(\omega) e^{i\omega t} d\omega.$$

Fortunately, this change will not invalidate any of the theory that we have seen above, but only change some factors in our theorems. The necessary modifications become quite obvious when following the proofs from this chapter: Quite often, one can cancel the term 2π or $\sqrt{2\pi}$.

3 Discrete Fourier transform in one dimension

3.1 Motivation

In the last chapter, we have discussed the Fourier transform for functions in $L^1(\mathbb{R})$ and $L^2(\mathbb{R})$. The use of these well-known spaces allowed us to make statements about the resulting transformed function $\hat{f}$. But the restriction to these $L^p(\mathbb{R})$ spaces does not imply that we cannot define the Fourier transform for other spaces.

Example 3.1. *The functions $f(x) = \sin(x)$, $g(x) = \cos(x)$ are not in $L^p(\mathbb{R})$ for $p = 1, 2$. By generalizing the Fourier analysis, we may still find the Fourier transforms*

$$\hat{f}(\omega) = \pi i \left[\delta(\omega + 1) - \delta(\omega - 1)\right],$$
$$\hat{g}(\omega) = \pi \left[\delta(\omega + 1) + \delta(\omega - 1)\right],$$

where δ is the Dirac distribution—sometimes called Dirac delta function. It can be loosely thought as a "function" defined as

$$\delta(x) = \begin{cases} \infty, & x = 0, \\ 0, & x \neq 0. \end{cases}$$

Instead of seeing the function $f(x) = \sin(x)$ as a function in $\mathbb{R}$, we could also restrict the domain to a finite interval $[a, b]$. Clearly, this new function $f_1(x) = \sin(x)\chi_{[a,b]}(x) \in L^p(\mathbb{R})$ for $p = 1, 2$ and we may find the Fourier transform in the classical way.

In real-world applications, we mostly deal with finite domains anyway, and we work with discrete functions. In one dimension, we can think of a digital audio signal measured over a specific amount of time. A two-dimensional example could be constructed from an image which defines a color or grayscale for each pixel (x, y).

There are different approaches to explain the transition from the continuous Fourier transform to the discrete Fourier transform. In this thesis, we will present the definition and try to motivate the construction.

This chapter follows the concepts of [DS15, Chapter 3] and [Har17, Chapter 4.2]. Some remarks have also been taken from [Bra00, Chapters 10 and 11].

3.2 Preliminaries

In order to reduce our time and effort, we are going to maintain our own notation throughout the next chapters. To avoid confusion with notation of other authors,

we shall provide our definitions for some well-known sets.

Definition 3.2. *We define the sets* $\mathbb{N} = \{1, 2, 3, \dots\}$ *and* $\mathbb{N}_0 = \{0, 1, 2, \dots\}$.

Definition 3.3. *As usual, we define* $\mathbb{Z} = \{\dots, -2, -1, 0, 1, 2, \dots\}$. *Additionally, we define the set* $\mathbb{Z}_n$ *for* $n \in \mathbb{N}$ *as*

$$\mathbb{Z}_n = \mathbb{Z}/n\mathbb{Z} = \{[0], [1], [2], \dots, [n-1]\},$$

where $[\,.\,]$ *is the equivalence class of a given number.*

Remark 3.4. *(i) It is easy to show that* $(\mathbb{Z}_n, +)$ *is a group.*

(ii) The elements of an equivalence class all share the same remainders when divided by $n - 1$. *For example,* $[0] = \{\dots, -n, 0, n, \dots\}$ *and* $[1] = \{\dots, -n + 1, 1, n + 1, \dots\}$.

(iii) We shall not use the notation of equivalence classes. Instead, we will identify any number with its equivalence class and remember that $m \equiv m + nk$ *for any* $m \in \mathbb{Z}_n$, $k \in \mathbb{Z}$.

Notation 3.5. *We are going to use the following shorthand notation in the two-dimensional case:* $\mathbb{Z}_{n_1 \times n_2} = \mathbb{Z}_{n_1} \times \mathbb{Z}_{n_2}$.

It is possible to identify discrete functions with vectors or matrices. In order to avoid issues, we always start their indexing by 0 and not by 1. We are going to use the same notation throughout this thesis:

Notation 3.6. *(1) If* $f : \mathbb{Z}_n \to \mathbb{C}$ *is a function, we will often identify it with a vector* $f \in \mathbb{C}^n$. *We may equally write* $f(0) = f_0, \dots, f(n-1) = f_{n-1}$.

(2) If $g : \mathbb{Z}_n^2 \to \mathbb{C}$ *is another function, we can identify it with a matrix* $g \in \mathbb{C}^{n \times n}$ *and equally write* $g(j_1, j_2) = g_{j_1, j_2}$.

Definition 3.7 (Discrete convolution)**.** *Let* $f, g : \mathbb{Z}_n^d \to \mathbb{C}$ *be two discrete functions for* $d \in \{1, 2\}$.

(1) If $d = 1$, *then the discrete convolution of* f *and* g *is given by*

$$(f * g)(j) = \sum_{k=0}^{n-1} f(k) g(j - k).$$

(2) If $d = 2$, *then the discrete convolution of* f *and* g *is given by*

$$(f * g)(j_1, j_2) = \sum_{k_1=0}^{n-1} \sum_{k_2=0}^{n-1} f(k_1, k_2) g(j_1 - k_1, j_2 - k_2).$$

Remark 3.8. *Due to the periodicity of* f *and* g, *the definition above is sometimes called circular convolution.*

3.3 Derivation of the discrete Fourier transform

Consider a function f with the Fourier transform

$$\hat{f}(\omega) = \int_{-\infty}^{\infty} f(t)e^{-i\omega t}dt.$$

In many cases, we will not have the tools to find the Fourier transform in an analytical way. Instead, we now try to approximate the integral.

First, we truncate the range of integration to an interval of $[a, b]$ and divide this interval in $n \in \mathbb{N}$ equidistant pieces, so that each piece has the width of $h = \frac{b-a}{n}$:

$$a = t_0 < t_1 < t_2 < \ldots < t_{n-1} < t_n = b.$$

Thus,

$$t_k = a + hk = a + \tfrac{b-a}{n}\, k, \quad k = 0, 1, \ldots, n. \tag{3.1}$$

Assuming that n is a large integer and $[a, b]$ is a large interval, the following function g is a good approximation of $\hat{f}$:

$$\hat{f}(\omega) = \int_{-\infty}^{\infty} f(t)e^{-i\omega t}dt \approx \int_{a}^{b} f(t)e^{-i\omega t}dt \approx \sum_{k=0}^{n-1} hf(t_k)e^{-i\omega t_k} = g(\omega).$$

We also require to discretize the domain of $\hat{f}$, so let us set

$$\omega_m = \frac{2\pi m}{b-a}, \quad m = 0, 1, \ldots, n. \tag{3.2}$$

Using (3.1) and (3.2), we obtain

$$\begin{aligned} g(\omega_m) &= \sum_{k=0}^{n-1} hf(t_k)e^{-i\omega_m t_k} = h\sum_{k=0}^{n-1} f(t_k)e^{-i\omega_m\left(a+\frac{b-a}{n}\,k\right)} \\ &= he^{-ia\omega_m}\sum_{k=0}^{n-1} f(t_k)e^{-i\,\frac{2\pi m}{b-a}\,\frac{b-a}{n}\,k} = he^{-ia\omega_m}\sum_{k=0}^{n-1} f(t_k)e^{-\frac{2\pi i}{n}mk}. \end{aligned}$$

Finally—by neglecting the term $he^{-ia\omega_m}$—we obtain the common form of the discrete Fourier transform as in Definition 3.11 below. Fortunately, the discrete Fourier transform is still comparable to its analytic counterpart. In "perfect" conditions, the absolute part of both transforms only differ by this factor $|he^{-ia\omega_m}| = h$.

It is an automatic consequence from above that the the discrete versions of f and $\hat{f}$ must be $[a, b]$-periodic. We note that a continuous function is in general

not in $L^2(\mathbb{R})$ if it is $[a,b]$-periodic. Nevertheless, we may find a discrete Fourier transform that corresponds to the continuous Fourier transform if f is in $L^2([a,b])$.

Looking at the computations, we note that the term $e^{2\pi i/n}$ plays an important role in the discrete Fourier transform. To make our lives easier, we are going to provide the following definition:

Definition 3.9 (Discrete Fourier transform matrix)**.** *From now on, we will reserve the variables w_n and W_n. We define them as*

$$w_n = e^{-2\pi i/n}$$

and

$$W_n = \begin{bmatrix} w_n^0 & w_n^0 & w_n^0 & \dots & w_n^{0(n-1)} \\ w_n^0 & w_n^1 & w_n^2 & \dots & w_n^{1(n-1)} \\ w_n^0 & w_n^2 & w_n^4 & \dots & w_n^{2(n-1)} \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ w_n^0 & w_n^{n-1} & w_n^{2(n-1)} & \dots & w_n^{(n-1)^2} \end{bmatrix}.$$

The matrix W_n is also called the nth order discrete Fourier transform matrix.

Remark 3.10. *(i) Some authors define w_n and W_n in a slightly different way. Take special note of the minus in the exponent of w_n.*

(ii) For the implementation of the fast Fourier transform, we will only need the case $n = 2$, so

$$W_2 = \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix}.$$

The reason will be clear once we have seen the algorithm for the fast Fourier transform.

We are now ready to define the discrete Fourier transform. An analogous form for the inverse Fourier transform will be stated later below. In all statements in this chapter, the variable $n \in \mathbb{N}$ should be linked to the discretization defined above.

Definition 3.11 (Discrete Fourier transform)**.** *Let $f : \mathbb{Z}_n \to \mathbb{C}$ be a function. Then, we define the discrete Fourier transform by*

$$\mathcal{F}_d\{f\}(m) = \hat{f}(m) = \sum_{k=0}^{n-1} f(k) w_n^{mk}$$

for $m \in \mathbb{Z}_n$.

Remark 3.12. *(i) As mentioned before, the function f can be seen as a vector in $\mathbb{C}^n$. We will thus identify the notation $f(m)$ and f_m.*

(ii) We are using the same notation $\hat{f}$ for the discrete Fourier transform as in the last chapter for the continuous Fourier transform.

Theorem 3.13 (Basic properties of the discrete Fourier transform)**.** *Assume we have the functions $f, g : \mathbb{Z}_n \to \mathbb{C}$ and their corresponding discrete Fourier transforms $\hat{f}, \hat{g}$. Additionally, set the scalar values $a, b \in \mathbb{C}$. Then the following properties hold for $j, m \in \mathbb{Z}_n$:*

(1) $\mathcal{F}_d\{a\, f(m) + b\, g(m)\} = a\, \hat{f}(m) + b\, \hat{g}(m)$ *($\mathcal{F}_d$ is a linear operator),*

(2) $\mathcal{F}_d\{f(m-j)\} = w_n^{jm} \hat{f}(m)$ *(time shift),*

(3) $\mathcal{F}_d\{f(m) w_n^{-jm}\} = \hat{f}(m-j)$ *(frequency shift),*

(4) $\mathcal{F}_d\{f(m)\cos\left(\frac{2\pi m j}{n}\right)\} = \frac{1}{2}\left(\hat{f}(m-j) + \hat{f}(m+j)\right)$ *(modulation).*

Proof. (1) The linearity follows directly from Definition 3.11.

(2) Let $g(m) = f(m-j)$. Note that we have $f(m+kn) = f(m)$ for all $m \in \mathbb{Z}_n$ and $k \in \mathbb{Z}$. With that in mind we obtain by Definition 3.11

$$\begin{aligned}
\mathcal{F}_d\{f(m-j)\} = \hat{g}(m) &= \sum_{k=0}^{n-1} g(k) w_n^{mk} \\
&= \sum_{k=0}^{n-1} f(k-j) w_n^{mk} = \sum_{k=-j}^{n-1-j} f(k) w_n^{m(k+j)} \\
&= \sum_{k=-j}^{-1} f(k) w_n^{m(k+j)} + \sum_{k=0}^{n-1-j} f(k) w_n^{m(k+j)} \\
\overset{(n\text{-periodicity})}{} &= \sum_{k=-j+n}^{-1+n} f(k) w_n^{m(k+j)} + \sum_{k=0}^{n-1-j} f(k) w_n^{m(k+j)} \\
&= \sum_{k=0}^{n-1} f(k) w_n^{m(k+j)} = w_n^{jm} \sum_{k=0}^{n-1} f(k) w_n^{mk} \\
&= w_n^{jm} \hat{f}(m).
\end{aligned}$$

(3) Let $g(m) = f(m)w_n^{-jm}$. Then, we get by Definition 3.11

$$\begin{aligned}\mathcal{F}_d\{f(m)w_n^{-jm}\} = \hat{g}(m) &= \sum_{k=0}^{n-1} g(k)w_n^{mk} = \sum_{k=0}^{n-1} f(k)w_n^{-jk}w_n^{mk} \\ &= \sum_{k=0}^{n-1} f(k)w_n^{(m-j)k} = \hat{f}(m-j).\end{aligned}$$

(4) Since

$$\cos\left(\frac{2\pi mj}{n}\right) = \frac{1}{2}\left(e^{-2\pi mji/n} + e^{2\pi mji/n}\right) = \frac{1}{2}\left(w_n^{jm} + w_n^{-jm}\right),$$

we may use the first and third item to get

$$\begin{aligned}\mathcal{F}_d\left\{f(m)\cos\left(\frac{2\pi mj}{n}\right)\right\} &= \mathcal{F}_d\left\{f(m)\frac{1}{2}\left(w_n^{jm} + w_n^{-jm}\right)\right\} \\ &= \frac{1}{2}\left(\mathcal{F}_d\left\{f(m)w_n^{jm}\right\} + \mathcal{F}_d\left\{f(m)w_n^{-jm}\right\}\right) \\ &= \frac{1}{2}\left(\hat{f}(m+j) + \hat{f}(m-j)\right).\end{aligned}$$

□

Remark 3.14. *See also Theorem 2.11 to compare to the properties of the continuous version.*

Theorem 3.15 (Hermitian symmetry)**.** *Let $f : \mathbb{Z}_n \to \mathbb{R}$ be a real discrete function and $\hat{f}$ is its corresponding discrete Fourier transform. Then*

$$\hat{f}(n-m) = \overline{\hat{f}(m)}$$

for all $m \in \mathbb{Z}_n$.

Proof. First we note that

$$\begin{aligned}w_n^{(n-m)k} &= e^{-2\pi i(n-m)k/n} = e^{-2\pi ikn/n}e^{2\pi imk/n} \\ &= e^{2\pi imk/n} = \overline{e^{-2\pi imk/n}} = \overline{w_n^{mk}}\end{aligned}$$

for $k, m \in \mathbb{Z}_n$. Using this and Definition 3.11, we obtain

$$\begin{aligned}\hat{f}(n-m) &= \sum_{k=0}^{n-1} f(k)w_n^{(n-m)k} = \sum_{k=0}^{n-1} f(k)\overline{w_n^{mk}} \\ &= \sum_{k=0}^{n-1} \overline{f(k)w_n^{mk}} = \overline{\hat{f}(m)}\end{aligned}$$

for all $m \in \mathbb{Z}_n$. □

Remark 3.16. *(i) Note that the Hermitian symmetry is only valid for real valued discrete functions f.*

(ii) The theorem states a useful property for the calculation of the discrete Fourier transform. It shows that computing $\hat{f}(m)$ for the first half of indices m is enough to construct the whole transform.

(iii) Additionally, it shows that $\hat{f}(0) = \hat{f}(n-0) = \overline{\hat{f}(0)}$, hence $\hat{f}(0)$ is always real.

(iv) If n is even, then $\hat{f}\left(\frac{n}{2}\right) = \hat{f}\left(n - \frac{n}{2}\right) = \overline{\hat{f}\left(\frac{n}{2}\right)}$ is also real.

We have already proved the convolution theorem for functions in $L^1(\mathbb{R})$, Theorem 2.20. Now, we are going to state its discrete counterpart:

Theorem 3.17 (Discrete convolution theorem)**.** *Let $f, g : \mathbb{Z}_n \to \mathbb{C}$ be discrete functions with their discrete Fourier transforms $\hat{f}, \hat{g}$. Then,*

$$\mathcal{F}\{f * g\}(m) = \hat{f}(m)\hat{g}(m)$$

for all $m \in \mathbb{Z}_n$.

Proof. Let $f, g : \mathbb{Z}_n \to \mathbb{C}$ be discrete functions with their discrete Fourier transforms

$$\hat{f}(m) = \sum_{k=0}^{n-1} f(k)w^{mk}, \qquad \hat{g}(m) = \sum_{k=0}^{n-1} g(k)w^{mk}$$

for all $m \in \mathbb{Z}_n$. Then,

$$\begin{aligned}
\mathcal{F}\{f * g\}(m) &= \mathcal{F}\left\{\sum_{k=0}^{n-1} f(k)g(j-k)\right\} \\
&= \sum_{j=0}^{n-1}\sum_{k=0}^{n-1} f(k)g(j-k)w^{mj} \\
&= \sum_{k=0}^{n-1} f(k)w^{mk}\sum_{j=0}^{n-1} g(j-k)w^{m(j-k)} \\
&= \sum_{k=0}^{n-1} f(k)w^{mk}\hat{g}(m) \\
&= \hat{f}(m)\hat{g}(m).
\end{aligned}$$

□

Remark 3.18. *From the theorem above we deduce that*

$$(f * g)(j) = \mathcal{F}^{-1}\{\hat{f}(m)\hat{g}(m)\}(j).$$

We will see in Theorem 3.27 that the complexity for the calculation of the discrete (inverse) Fourier transform can be reduced to $\mathcal{O}(n\log_2(n))$. On the other hand, the discrete convolution has a complexity of $\mathcal{O}(n^2)$. Thus, it is faster to calculate the convolution as a multiplication in the Fourier space.

Theorem 3.19 (Discrete inverse Fourier transform)**.** *Let $f : \mathbb{Z}_n \to \mathbb{C}$ be such that $\hat{f}$ is its discrete Fourier transform as defined in Definition 3.11. Then, the discrete inverse Fourier transform is*

$$\mathcal{F}_d^{-1}\{\hat{f}\}(m) = f(m) = \frac{1}{n}\sum_{k=0}^{n-1}\hat{f}(k)w_n^{-mk}$$

for $m \in \mathbb{Z}_n$.

Proof. (i) Let $j, k, m \in \mathbb{Z}_n$. Define $x = e^{2\pi i(m-j)/n}$. We discover that $x^n = 1$, so we may deduce that

$$\begin{aligned}\sum_{k=0}^{n-1} w_n^{-k(m-j)} &= \sum_{k=0}^{n-1} e^{2\pi ik(m-j)/n} = \sum_{k=0}^{n-1}\left(e^{2\pi i(m-j)/n}\right)^k \\ &= \sum_{k=0}^{n-1} x^k = \begin{cases} n, & x = 1, \\ \frac{1-x^n}{1-x} = 0, & x \neq 1. \end{cases}\end{aligned}$$

Note that $x = 1 \Leftrightarrow m = j$, hence

$$\sum_{k=0}^{n-1} w_n^{-k(m-j)} = n\delta_{j,m}.$$

(not to be confused with the Dirac distribution).

(ii) Using the first item, we get for $m \in \mathbb{Z}_n$

$$\begin{aligned}
\frac{1}{n}\sum_{k=0}^{n-1} \hat{f}(k) w_n^{-mk} &= \frac{1}{n}\sum_{k=0}^{n-1}\left(\sum_{j=0}^{n-1} f(j) w_n^{jk}\right) w_n^{-mk} \\
&= \frac{1}{n}\sum_{k=0}^{n-1}\left(\sum_{j=0}^{n-1} f(j) w_n^{-k(m-j)}\right) \\
&= \frac{1}{n}\sum_{j=0}^{n-1} f(j)\left(\sum_{k=0}^{n-1} w_n^{-k(m-j)}\right) \\
&= \frac{1}{n}\sum_{j=0}^{n-1} f(j) n \delta_{j,m} = \frac{1}{n} f(m) n \\
&= f(m).
\end{aligned}$$

□

Remark 3.20. *There is no need to directly implement the discrete inverse Fourier transform. It can be derived directly from the discrete Fourier transform because*

$$y = \frac{1}{n}\overline{W_n}\hat{y} \;\Leftrightarrow\; \bar{y} = \frac{1}{n}\overline{\overline{W_n}\hat{y}} = \frac{1}{n} W_n \bar{\hat{y}} \;\Leftrightarrow\; y = \frac{1}{n}\overline{W_n \bar{\hat{y}}}.$$

To finish the section on the discrete Fourier transform, we would like to look at an implementation in MATLAB. For this purpose, assume that we have a given vector $y \in \mathbb{C}^n$ and we would like to find the resulting discrete Fourier transform $\hat{y}$. The definition of the discrete Fourier transform yields

$$\hat{y_k} = \sum_{k=0}^{n-1} y_k w_n^{km}$$

for all $k \in \mathbb{Z}_n$. It is easy to see that this is nothing else than

$$\begin{bmatrix} \hat{y}_0 \\ \hat{y}_1 \\ \vdots \\ \hat{y}_{n-1} \end{bmatrix} = \underbrace{\begin{bmatrix} w_n^0 & w_n^0 & \dots & w_n^0 \\ w_n^0 & w_n^1 & \dots & w_n^{n-1} \\ \vdots & \vdots & \vdots & \vdots \\ w_n^0 & w_n^{n-1} & \dots & w_n^{(n-1)^2} \end{bmatrix}}_{=W_n} \begin{bmatrix} y_0 \\ y_1 \\ \vdots \\ y_{n-1} \end{bmatrix}.$$

Hence, our program only has to define the discrete Fourier transform matrix W_n and then solve the equation $\hat{y} = W_n y$. Below is a simple implementation without any further comments and error handling.

Listing 1: Basic discrete Fourier transform.

```
function fHat = fouriertransform(f)
    n = max(size(f));
    W = fouriermatrix(n);
    fHat = W*f;
end
```

Listing 2: Calculation of Fourier transform matrix.

```
function W = fouriermatrix(n)
    W = zeros(n, n);
    for i=1:n
        for j=1:n
            W(i,j) = exp(-2*pi*1i*(i-1)*(j-1)/n);
        end
    end
end
```

We should note that this way of implementing the discrete Fourier transform is not very efficient. Thus, the use of the code above is discouraged in real-world applications. The next section deals with improving this algorithm in terms of computer performance.

3.4 Implementation of the fast Fourier transform

So far, we have learned how to obtain information about the frequency of a signal on a discrete domain. By Definition 3.11, we need to execute $(n-1)n$ additions and n^2 multiplications to calculate the resulting vector from given data. Assuming we had a common music CD with a three minute track sampling at 44,100 Hertz, we would need at least 8,000,000 sample points in order to apply the discrete Fourier transform. Obviously, even modern computers would need their time to finish the occuring calculations. Applying the Fourier transform in more dimensions—for example in two-dimensional image analysis—increases the amount of operations even more.

For this reason, it was inevitable to find a more efficient way to calculate the discrete Fourier transform. Fortunately, the mathematicians Cooley and Tukey discovered an algorithm that drastically decreased the amount of necessary multiplications. In their paper from 1965, they described their findings how to reduce the amount of multiplications from $\mathcal{O}(n^2)$ to $\mathcal{O}(n \log_2(n))$ (cf. [CT65]). It is assumed that their method dates back to Gauss' research around 1805, but his

concepts lied idle until the invention of the digital computer.

After the ground-breaking paper of Cooley and Tukey, many other algorithms with similar techniques appeared. All these algorithms belong to the family of the so-called fast Fourier transforms, but as of today, the Cooley–Tukey algorithm remains the most used algorithm. Therefore, we are going to introduce this algorithm and simply call it fast Fourier transform.

By Definition 3.11—the discrete Fourier transform—we know that we need to calculate $W_n y$, where the matrix W_n consists of the entries $w_n^k = e^{-2\pi ik/n} \in \mathbb{C}$. The fact that the entries of the matrix W_n are periodic in n is the foundation of every fast Fourier transform algorithm. How to make use of the periodicity can be best observed in a simple example:

Example 3.21. *Let $f = [0, 1, 1, 0]^T \in \mathbb{C}^4$, hence $n = 4$. If we calculate $\hat{f}$ directly from Definition 3.11, we get*

$$\hat{f} = W_4 f = \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & -i & -1 & i \\ 1 & -1 & 1 & -1 \\ 1 & i & -1 & -i \end{bmatrix} \begin{bmatrix} 0 \\ 1 \\ 1 \\ 0 \end{bmatrix} = \begin{bmatrix} 2 \\ -1-i \\ 0 \\ -1+i \end{bmatrix}.$$

In order to compute the result, we had to perform $n^2 = 16$ multiplications. Instead of doing that, we could take a closer look at the matrix W_4 and the initial multiplication instead:

$$\begin{bmatrix} \hat{f}_0 \\ \hat{f}_1 \\ \hat{f}_2 \\ \hat{f}_3 \end{bmatrix} = \begin{bmatrix} w_4^0 & w_4^0 & w_4^0 & w_4^0 \\ w_4^0 & w_4^1 & w_4^2 & w_4^3 \\ w_4^0 & w_4^2 & w_4^0 & w_4^2 \\ w_4^0 & w_4^3 & w_4^2 & w_4^1 \end{bmatrix} \begin{bmatrix} f_0 \\ f_1 \\ f_2 \\ f_3 \end{bmatrix}.$$

Sorting the data on the right side of the equation by even and odd indices, we obtain

$$\begin{bmatrix} \hat{f}_0 \\ \hat{f}_1 \\ \hat{f}_2 \\ \hat{f}_3 \end{bmatrix} = \left[\begin{array}{cc|cc} w_4^0 & w_4^0 & w_4^0 & w_4^0 \\ w_4^0 & w_4^2 & w_4^1 & w_4^3 \\ \hline w_4^0 & w_4^0 & w_4^2 & w_4^2 \\ w_4^0 & w_4^2 & w_4^3 & w_4^1 \end{array}\right] \left[\begin{array}{c} f_0 \\ f_2 \\ \hline f_1 \\ f_3 \end{array}\right].$$

Knowing that W_2 is defined as

$$W_2 = \begin{bmatrix} w_2^0 & w_2^0 \\ w_2^0 & w_2^1 \end{bmatrix} = \begin{bmatrix} w_4^0 & w_4^0 \\ w_4^0 & w_4^2 \end{bmatrix} = \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix},$$

we can deduce that the above equation is equivalent to

$$\begin{bmatrix} \hat{f}_0 \\ \hat{f}_1 \\ \hat{f}_2 \\ \hat{f}_3 \end{bmatrix} = \left[\begin{array}{c|c} W_2 & D_2W_2 \\ \hline W_2 & -D_2W_2 \end{array}\right] \begin{bmatrix} f_0 \\ f_2 \\ \hline f_1 \\ f_3 \end{bmatrix}$$

with $D_2 = \operatorname{diag}(w_4^0, w_4^1) = \operatorname{diag}(1, -i)$. *So we see that our original problem reduces to finding* $\hat{a}$ *and* $\hat{b}$ *in*

$$\begin{bmatrix} \hat{f}_0 \\ \hat{f}_1 \end{bmatrix} = \underbrace{W_2 \begin{bmatrix} f_0 \\ f_2 \end{bmatrix}}_{=\hat{a}} + D_2 \underbrace{W_2 \begin{bmatrix} f_1 \\ f_3 \end{bmatrix}}_{=\hat{b}}, \qquad \begin{bmatrix} \hat{f}_2 \\ \hat{f}_3 \end{bmatrix} = \underbrace{W_2 \begin{bmatrix} f_0 \\ f_2 \end{bmatrix}}_{=\hat{a}} - D_2 \underbrace{W_2 \begin{bmatrix} f_1 \\ f_3 \end{bmatrix}}_{=\hat{b}}. \tag{3.3}$$

Finishing the calculations, we get

$$\begin{bmatrix} \hat{f}_0 \\ \hat{f}_1 \end{bmatrix} = \begin{bmatrix} f_0 + f_2 \\ f_0 - f_2 \end{bmatrix} + \begin{bmatrix} 1 & 0 \\ 0 & -i \end{bmatrix} \begin{bmatrix} f_1 + f_3 \\ f_1 - f_3 \end{bmatrix} = \begin{bmatrix} 2 \\ -1 - i \end{bmatrix},$$
$$\begin{bmatrix} \hat{f}_2 \\ \hat{f}_3 \end{bmatrix} = \begin{bmatrix} f_0 + f_2 \\ f_0 - f_2 \end{bmatrix} - \begin{bmatrix} 1 & 0 \\ 0 & -i \end{bmatrix} \begin{bmatrix} f_1 + f_3 \\ f_1 - f_3 \end{bmatrix} = \begin{bmatrix} 0 \\ -1 + i \end{bmatrix},$$

as expected.

Remark 3.22. *If we analyze equation* (3.3)*, we see that we only need one multiplication to calculate* $\hat{a}$ *and one to calculate* $\hat{b}$*. This is due to the fact that* W_2 *only contains one entry that is not equal to* 1*. On top of that, we need at most two additional multiplications to compute* $D_2\hat{b}$*, as* D_2 *has two non-zero entries. Hence, we need a total of* 4 *instead of* 16 *multiplications to find the discrete Fourier transform of* f*.*

From Example 3.21 we learned that we can reduce the amount of necessary multiplications to compute the discrete Fourier transform. In the theorem below, we are going to generalize this idea.

Theorem 3.23. *Let* $f : \mathbb{Z}_{2n} \to \mathbb{C}$ *be a function and* $\hat{f} : \mathbb{Z}_{2n} \to \mathbb{C}$ *its discrete Fourier transform defined by Definition 3.11. Further, define* $a, b : \mathbb{Z}_n \to \mathbb{C}$ *so that*

$$a(k) = f(2k), \qquad b(k) = f(2k+1)$$

for $k \in \mathbb{Z}_n$ *and let* $\hat{a}, \hat{b} : \mathbb{Z}_n \to \mathbb{C}$ *their discrete Fourier transforms. Then, the formulas*

$$\hat{f}(m) = \hat{a}(m) + w_{2n}^m\, \hat{b}(m), \qquad \hat{f}(n+m) = \hat{a}(m) - w_{2n}^m\, \hat{b}(m)$$

hold for all $m \in \mathbb{Z}_n$*.*

Proof. (i) Using Definition 3.11 and the identity $w_n = w_{2n}^2$, we split up the even and odd parts of the sum to obtain

$$\begin{aligned}
\hat{f}(m) &= \sum_{k=0}^{2n-1} f(k) w_{2n}^{mk} \\
&= \sum_{k=0}^{n-1} f(2k) w_{2n}^{m(2k)} + \sum_{k=0}^{n-1} f(2k+1) w_{2n}^{m(2k+1)} \\
&= \sum_{k=0}^{n-1} a(k) (w_{2n}^2)^{mk} + w_{2n}^m \sum_{k=0}^{n-1} b(k) (w_{2n}^2)^{mk} \\
&= \sum_{k=0}^{n-1} a(k) w_n^{mk} + w_{2n}^m \sum_{k=0}^{n-1} b(k) w_n^{mk}
\end{aligned}$$

for all $m \in \mathbb{Z}_{2n-1}$.

(ii) Let $m \in \mathbb{Z}_n$, then it follows directly from the first item that

$$\hat{f}(m) = \hat{a}(m) + w_{2n}^m \, \hat{b}(m).$$

(iii) Otherwise, first check that the following two statements hold:

$$\begin{aligned}
w_n^{(n+m)k} &= e^{-2\pi i (n+m)k/n} = e^{-2\pi i k} e^{-2\pi i m k/n} = w_n^{mk}, \\
w_{2n}^{n+m} &= e^{-2\pi i (n+m)/(2n)} = e^{-\pi i k} e^{-2\pi i m/(2n)} = -w_{2n}^m.
\end{aligned}$$

We deduce from the first item that

$$\begin{aligned}
\hat{f}(n+m) &= \sum_{k=0}^{n-1} a(k) w_n^{(n+m)k} + w_{2n}^{n+m} \sum_{k=0}^{n-1} b(k) w_n^{(n+m)k} \\
&= \sum_{k=0}^{n-1} a(k) w_n^{mk} - w_{2n}^m \sum_{k=0}^{n-1} b(k) w_n^{mk} \\
&= \hat{a}(n) - w_{2n}^n \, \hat{b}(n),
\end{aligned}$$

where again $m \in \mathbb{Z}_n$. □

Generally, implementations in programs like MATLAB are more efficient if we use a matrix/vector version of the theorem above. An equivalent formulation is the following corollary which resembles the notation of Example 3.21:

Corollary 3.24. *Let $y \in \mathbb{C}^{2n}$ be a function and $\hat{y} \in \mathbb{C}^{2n}$ its discrete Fourier transform defined by Definition 3.11. Further, define $c, d \in \mathbb{C}^n$ so that*

$$c = W_n \begin{bmatrix} y_0 \\ y_2 \\ \vdots \\ y_{2n-4} \\ y_{2n-2} \end{bmatrix}, \qquad d = D_n W_n \begin{bmatrix} y_1 \\ y_3 \\ \vdots \\ y_{2n-3} \\ y_{2n-1} \end{bmatrix}$$

with the diagonal matrix

$$D_n = \begin{bmatrix} w_{2n}^0 & & & & \\ & w_{2n}^1 & & & \\ & & w_{2n}^2 & & \\ & & & \ddots & \\ & & & & w_{2n}^{n-1} \end{bmatrix}.$$

Then, the formula

$$\hat{y} = \left[\begin{array}{c} \hat{y}_0 \\ \vdots \\ \hat{y}_{n-1} \\ \hline \hat{y}_n \\ \vdots \\ \hat{y}_{2n-1} \end{array}\right] = \left[\begin{array}{c} c + d \\ \hline c - d \end{array}\right]$$

holds.

Recursively applying Theorem 3.23 directly yields the Cooley–Tukey algorithm. In order to be able to repeat the recursion many times, we just have to make sure that the discretization is chosen such that $n = 2^m$ for $m \in \mathbb{N}$. The recursion ends at the moment where $n = 2$, as seen in Example 3.21.

In real-world discrete examples, we could encounter the case that n is not a power of 2. There are different ways to approach that issue: For instance, we could either use an algorithm that uses another base than 2, or we could simply extend the existing data to the next power of 2 by adding zeros.

Definition 3.25 (Cooley–Tukey algorithm)**.** *The Cooley–Tukey (or fast Fourier*

transform) algorithm is defined by the following instructions:

Algorithm 3.1: Cooley–Tukey algorithm

Input: $y = [y_0, y_1, \ldots, y_{n-1}]^T \in \mathbb{C}^n$ *where* $n = 2^m$ *for* $m \in \mathbb{N}$.
Output: $\hat{y} = [\hat{y}_0, \hat{y}_1, \ldots, \hat{y}_{n-1}]^T \in \mathbb{C}^n$, *the discrete Fourier transform of* y.

`// Calculate the discrete Fourier transform` $W_2 y$ `if` $n = 2$.
if $n = 2$ **then**
  $\hat{y} = \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix} y$
  return $\hat{y}$
end

`// Otherwise, halve the number of n and overwrite it.`
Set $n = n/2$.

`// Split the vector` y `in even and odd parts.`
Set $a = [y_0, y_2, \ldots, y_{2n-2}]^T$, $b = [y_1, y_3, \ldots, y_{2n-1}]^T \in \mathbb{C}^n$.

`// Calculate the discrete Fourier transforms of` a `and` b `with this algorithm recursively.`
Set $c = CooleyTukey(a)$ *and* $d = CooleyTukey(b)$.

`// Adjust the values of vector` $d = [d_0, d_1, \ldots, d_{n-1}]^T$.
for $k = 0, 1, \ldots, n-1$ **do**
  $d_k = e^{-i\pi k/n}\, d_k$
end

`// Save the discrete Fourier transform into` $\hat{y} \in \mathbb{C}^{2n}$.
Set $\begin{bmatrix} \hat{y}_0 \\ \hat{y}_2 \\ \vdots \\ \hat{y}_{2n-2} \end{bmatrix} = c + d$ *and* $\begin{bmatrix} \hat{y}_1 \\ \hat{y}_3 \\ \vdots \\ \hat{y}_{2n-1} \end{bmatrix} = c - d.$

`// Return the discrete Fourier transform.`
return $\hat{y}$

As noted in Remark 3.20, we do not need to make an implementation for the inverse version. Instead, we can use the algorithm above to compute the discrete inverse Fourier transform as well. To finish the theoretical part of the fast Fourier

transform, we would like to specify the improvement of cost for the multiplications.

Remark 3.26. *If we denote by $\phi(n)$ the maximal amount of needed multiplications for the n-point discrete Fourier transform, then Theorem 3.23 shows the relationship*

$$\phi(n) = 2\phi\left(\frac{n}{2}\right) + \frac{n}{2}. \tag{3.4}$$

Theorem 3.27. *Let $m \in \mathbb{N}$ be a number such that $n = 2^m$. Then, the maximal amount of necessary multiplications for the n-point fast Fourier transform is*

$$\phi(n) = \frac{n \log_2(n)}{2}.$$

Proof. We prove the theorem by induction on $m \in \mathbb{N}$.

(i) Let $m = 1$, that means $n = 2$. Then the fast Fourier transform needs only one multiplication (see Example 3.21). The formula yields

$$\phi(2) = \frac{2 \log_2(2)}{2} = 1,$$

which means we need at most one multiplication. Obviously, this is true.

(ii) Assume now that the formula for $m = \tilde{m} - 1$ holds, which means we have

$$\phi(2^{\tilde{m}-1}) = \frac{2^{\tilde{m}-1} \log_2(2^{\tilde{m}-1})}{2}.$$

Using the recursion formula (3.4), we obtain

$$\begin{aligned}
\phi(2^{\tilde{m}}) &= 2\phi(2^{\tilde{m}-1}) + 2^{\tilde{m}-1} = 2\left(\frac{2^{\tilde{m}-1} \log_2(2^{\tilde{m}-1})}{2}\right) + 2^{\tilde{m}-1} \\
&= 2^{\tilde{m}-1}(\tilde{m}-1) + 2^{\tilde{m}-1} = 2^{\tilde{m}-1}\tilde{m} = \frac{2^{\tilde{m}}\tilde{m}}{2} \\
&= \frac{2^{\tilde{m}} \log_2(2^{\tilde{m}})}{2},
\end{aligned}$$

hence the formula holds for $m = \tilde{m}$.

The proposition follows by induction. □

Remark 3.28. *One could argue that for $n = 2$, we do not need any multiplications and the recursion formula is not precise. Therefore, the function ϕ should be considered as upper limit for the amount of multiplications and the wording of the theorem contains "maximal amount".*

Theorem 3.27 showed us that the cost caused by the multiplications can be reduced from n^2 to $n \log_2(n)/2$ when using the fast Fourier transform instead of the discrete Fourier transform. Going back to our example with the music CD track ($n = 8{,}000{,}000$), we would need only 91 million instead of 64 trillion multiplications. In other words, one can reduce the amount of multiplications by a factor of 700,000.

To finish the section about the fast Fourier transform, we would like once more look at a possible implementation in MATLAB. Below is a simple example program of the Cooley–Tukey algorithm without comments and error handling. A full working example is shown at the end of the thesis in Listing 11.

Listing 3: Fast Fourier transform with Cooley–Tukey algorithm.

```
function fHat = fastfouriertransform(f)

    n = max(size(y));
    if n == 2
        fHat = [1, 1; 1 -1]*f;
        return;
    end

    n = n/2;

    c = fastfouriertransform(f(1:2:2*n));
    d = fastfouriertransform(f(2:2:2*n));

    for k=1:1:n
        d(k) = exp(-1i*pi*(k-1)/n)*d(k);
    end

    fHat = zeros(2*n, 1);
    fHat(1:n) = c+d;
    fHat(n+1:2*n) = c-d;

end
```

Listing 4: Inverse fast Fourier transform.

```
function f = inversefastfouriertransform(fHat)
    n = length(fHat);
    fHat = conj(fHat);
    f = 1/n * conj(fastfouriertransform(fHat));
end
```

The following examples use the MATLAB implementation for the fast Fourier transform:

Example 3.29. *Let $n = 2^7 = 128$ and the interval $[a, b] = [-1, 1] \subset \mathbb{R}$.*

(1) Define the function

$$f(t) = \begin{cases} 1, & -0.5 < t < 0.5, \\ 0, & otherwise, \end{cases}$$

which is clearly in $L^1(\mathbb{R}) \cap L^2(\mathbb{R})$ and we may calculate the continuous Fourier transform by

$$\begin{aligned} \hat{f}(\omega) &= \int_{-\infty}^{\infty} f(t) e^{-i\omega t} dt = \int_{-0.5}^{0.5} e^{-i\omega t} dt = \frac{1}{-i\omega} e^{-i\omega t} \Big|_{-0.5}^{+0.5} \\ &= \frac{1}{-i\omega} (e^{-0.5i\omega} - e^{0.5i\omega}) = \frac{2}{\omega} \frac{1}{2i} (e^{0.5i\omega} - e^{-0.5i\omega}) \\ &= \frac{2}{\omega} \sin(0.5\omega) \end{aligned}$$

for $\omega \neq 0$.

For the discrete Fourier transform, we remember that we only looked at values for ω equal to

$$\omega_n = \frac{2\pi n}{b - a} = \pi n.$$

We must not forget that we also dropped the factor $he^{-ia\omega_n}$ when deriving the discrete Fourier transform, hence the absolute values of the discrete and continuous versions will not match. That is why it makes sense to look at a normed value of $|\hat{f}(\omega)|$ in the discrete case.

Figure 1 shows the original function $f(t)$ against the normed discrete Fourier transform $|\hat{f}(\omega)|$ in a MATLAB plot, using the Cooley-Tukey algorithm from Definition 3.25 above. The resulting discrete Fourier transform has the form that we expected from the analytic result.

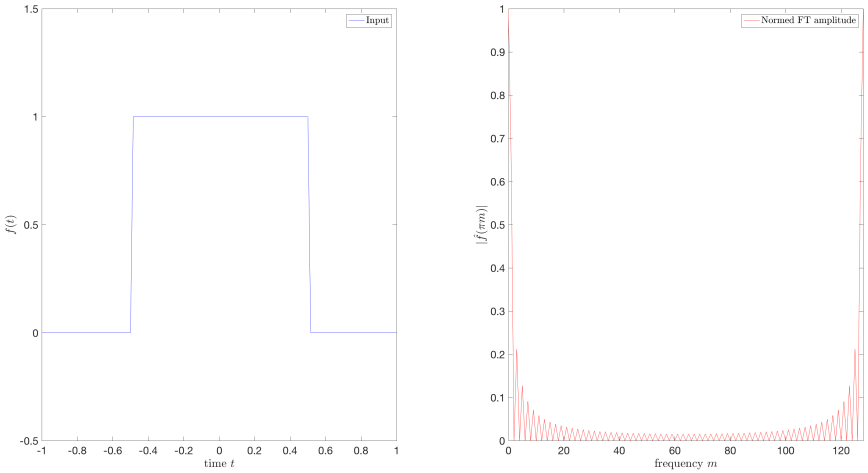


Figure 1: On the left $f(t)$, on the right the normed $|\hat{f}(\pi m)|$.

(2) Now, consider the function on the left side in Figure 2, that is

$$g(t) = 2\sin(2t\pi) + 3\sin(15t\pi) + 5\sin(7t\pi).$$

Obviously, this function is not in $L^2(\mathbb{R})$ and we cannot apply our theory from the continuous Fourier transform. But it is still possible to calculate the discrete Fourier transform.

Looking at the function $g(t)$, we note that it is composed from three sinuses with different amplitudes and frequences. To be more precise, we have the following on the interval $[-1, 1]$:

Function	***Amplitude***	***Frequency***
$2\sin(2t\pi)$	2	2
$3\sin(15t\pi)$	3	15
$5\sin(7t\pi)$	5	7

This table shows an important application of the discrete Fourier transform. Looking at the right graph of Figure 2, we discover that every frequency has a peak depending on its original value. For example, the peak of the sinusoid with frequency 7 is the largest because its amplitude 5 is the highest.

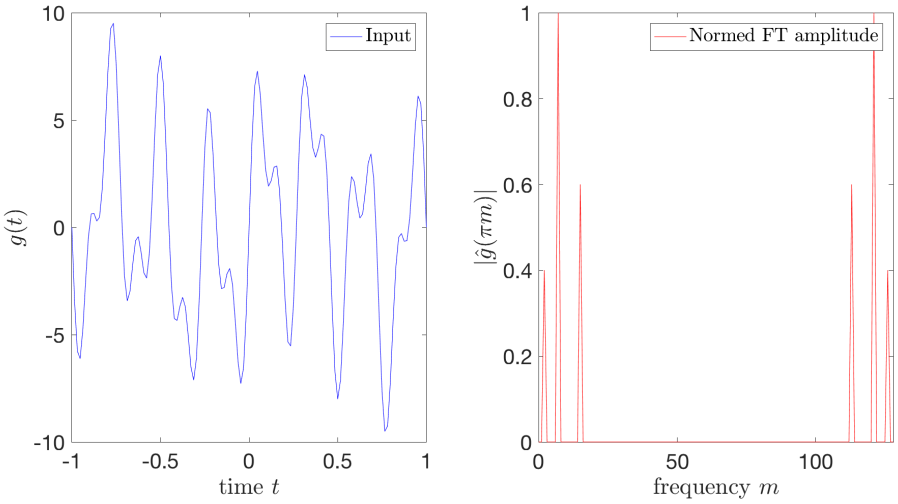


Figure 2: On the left $g(t)$, on the right the normed $|\hat{g}(\pi m)|$.

We hereby conclude the discrete Fourier transform in one dimension and start looking into the Gabor transform.

4 Gabor transform in one dimension

4.1 Motivation

The Fourier transform has proven to be a great tool when analyzing stationary signals. But as soon as we need to perform frequency analysis that is local in time, we cannot make use of the Fourier transform. In the last decades, new techniques have been presented, for example the Gabor or the wavelet transforms.

In this chapter, we would like to take a closer look at the Gabor transform. First, we are going to derive the one-dimensional Gabor transform for functions in $L^2(\mathbb{R})$ and later on for discrete data.

Throughout this chapter, we follow the concepts of [DS15, Chapter 4].

4.2 Gabor transform in $L^2(\mathbb{R})$

Looking at the Fourier transform presented in the previous chapters, we know that

$$\hat{f}(\omega) = \int_{-\infty}^{\infty} f(t) e^{-i\omega t} dt$$

gives us information about the frequency of a signal f: Analyzing $\hat{f}$, we can always determine which frequencies were present in the signal over the whole time domain. Unfortunately, it is not possible to say at which time the frequencies appeared or when they changed.

In 1946, the Hungarian-born electrical engineer Dennis Gabor had the idea to modify the Fourier transform by adding a shifted window function g into the integral (cf. [Gab46]). A window function is zero (or negligible) outside a given interval—the so-called window. This new transform then reads

$$\mathbf{WFT}\{f\}(t,\omega) = \int_{-\infty}^{\infty} f(\tau) g(\tau - t) e^{-i\omega\tau} d\tau \tag{4.1}$$

and is clearly depending on the time and on the frequency. It is sometimes called the windowed Fourier transform or the short-time Fourier transform and has many advantages over the regular Fourier transform. For example, as f is in $L^2(\mathbb{R})$ and the function g is negligible outside a given window, the integral can be approximated by adjusting the interval bounds to the window.

In practice, Gaussian functions proved to be a good choice as window functions in the windowed Fourier transform:

Definition 4.1 (Gaussian function)**.** *A Gaussian function is a function of the form*

$$g(t) = \alpha e^{-\beta^2(t-t_c)^2}, \quad t \in \mathbb{R}$$

with fixed $\alpha, \beta > 0$ *and* $t_c \in \mathbb{R}$*.*

Gaussian functions have many properties that are useful when applying the Fourier transform. As hinted in Example 2.9, any Gaussian function remains Gaussian after the application of the Fourier transform. Also, Gaussian functions comply with the L^2 theory of the Fourier transform, because any product of a function $f \in L^2(\mathbb{R})$ with a Gaussian g remains in $L^2(\mathbb{R})$.

Theorem 4.2 (Properties of Gaussian functions)**.** *Let* $f \in L^2(\mathbb{R})$*, let* g *be a Gaussian function and fix* $\tau \in \mathbb{R}$*. Then the following statements hold:*

(1) $g \in L^p(\mathbb{R})$ *for* $1 \leq p < \infty$*,*

(2) $fg \in L^1(\mathbb{R})$*,*

(3) $fg \in L^2(\mathbb{R})$*.*

Proof. Let $f \in L^2(\mathbb{R})$ and g be any Gaussian function as given in Definition 4.1.

(1) Let $1 \leq p < \infty$. Note that the value of the Gaussian integral is given by

$$\int_{-\infty}^{\infty} e^{-x^2} dx = \sqrt{\pi}.$$

Hence, we obtain by change of variables $z = \beta(t - t_c)\sqrt{p}$ that

$$\begin{aligned} \|g\|^p &= \int_{-\infty}^{\infty} |g(t)|^p dt = \int_{-\infty}^{\infty} \left|\alpha e^{-\beta^2(t-t_c)^2}\right|^p dt \\ &= \alpha^p \int_{-\infty}^{\infty} \left|e^{-\beta^2(t-t_c)^2}\right|^p dt = \alpha^p \int_{-\infty}^{\infty} e^{-\left(\beta(t-t_c)\sqrt{p}\right)^2} dt \\ \left(z=\beta(t-t_c)\sqrt{p}\right) &= \frac{\alpha^p}{\beta\sqrt{p}} \underbrace{\int_{-\infty}^{\infty} e^{-z^2} dz}_{=\sqrt{\pi}} < \infty. \end{aligned}$$

(2) Using Hölder's inequality, we obtain

$$\|fg\|_1 \leq \|f\| \, \|g\| < \infty$$

because f, g are both in $L^2(\mathbb{R})$ by the first item.

(3) Clearly, $g(t) \leq \alpha$ for all $t \in \mathbb{R}$. Thus,

$$\|fg\|^2 = \int_{-\infty}^{\infty} |f(t)g(t)|^2 dt \leq \int_{-\infty}^{\infty} |\alpha f(t)|^2 dt \leq \alpha^2 \|f\|^2 < \infty$$

as $f \in L^2(\mathbb{R})$. □

Using these results enable the formal definition of the Gabor transform and its properties:

Definition 4.3 (Gabor transform in $L^2(\mathbb{R})$)**.** *Let $f \in L^2(\mathbb{R})$ and g be a Gaussian function. The Gabor transform of f with respect to g is defined by*

$$\mathcal{G}_g\{f(t)\} = \tilde{f}_g(t, \omega) = \int_{-\infty}^{\infty} f(\tau) g(\tau - t) e^{-i\omega\tau} d\tau.$$

Remark 4.4. *(i) Some authors define g to be a general window function instead of a Gaussian function. In that case, the Gabor transform is the same as the windowed Fourier transform described in* (4.1)*.*

(ii) If the choice of the function g is clear from context, we sometimes omit the index g in the notation.

(iii) If we consider $t \in \mathbb{R}$ fixed, then we get

$$\tilde{f}_g(t, \omega) = \int_{-\infty}^{\infty} f(\tau) g(\tau - t) e^{-i\omega\tau} d\tau = \int_{-\infty}^{\infty} h(\tau) e^{-i\omega\tau} d\tau = \hat{h}(\omega)$$

for $h(\tau) = f(\tau)g(\tau - t)$.

(iv) The Gabor transform can be seen as inner product defined by

$$\tilde{f}_g(t, \omega) = \int_{-\infty}^{\infty} f(\tau) g(\tau - t) e^{-i\omega\tau} d\tau = \langle f, g_{t,\omega} \rangle,$$

where $g_{t,\omega}(\tau) = g(\tau - t)e^{i\omega t}$.

(v) It can also be seen as a convolution of two functions. Assume $g(\tau) = g(-\tau)$ for all $\tau \in \mathbb{R}$, which means $t_c = 0$ in Definition 4.1. If we consider $\omega \in \mathbb{R}$ fixed, then we get

$$\begin{aligned}
\tilde{f}_g(t, \omega) &= \int_{-\infty}^{\infty} f(\tau) g(\tau - t) e^{-i\omega\tau} d\tau \\
&= e^{-i\omega t} \int_{-\infty}^{\infty} f(\tau) g(\tau - t) e^{-i\omega(\tau - t)} d\tau \\
&\overset{g(\tau)=g(-\tau)}{=} e^{-i\omega t} \int_{-\infty}^{\infty} f(\tau) g(t - \tau) e^{i\omega(t - \tau)} d\tau \\
&= e^{-i\omega t} (f * g_\omega)(t)
\end{aligned}$$

for $g_\omega(t) = g(t)e^{i\omega t}$.

We have seen in the remark above that the Gabor transform given by Definition 4.3 can be computed in different ways. The following important functions naturally appeared:

Definition 4.5. *Let g be a Gaussian function and let $t, \omega \in \mathbb{R}$.*

(1) The function $g_{t,\omega}$ defined by

$$g_{t,\omega}(\tau) = g(\tau - t)e^{i\omega\tau}$$

for all $\tau \in \mathbb{R}$ is called Gabor function or Gabor wavelet.

(2) The function g_ω defined by

$$g_\omega(t) = g(t)e^{i\omega t}$$

for all $t \in \mathbb{R}$ is called Gabor filter.

Theorem 4.6 (Basic properties of the Gabor transform)**.** *Let f, h be functions in $L^2(\mathbb{R})$, let g be a Gaussian function and define the scalar values $a, b \in \mathbb{C}$, $r \in \mathbb{R}$. Then the following statements hold:*

(1) $\mathcal{G}_g\{a\, f(t) + b\, h(t)\} = a\, \tilde{f}_g(t,\omega) + b\, \tilde{h}_g(t,\omega)$ *($\mathcal{G}$ is a linear operator),*

(2) $\mathcal{G}_g\{f(t-r)\} = e^{-i\omega r}\tilde{f}_g(t-r,\omega)$ *(translation),*

(3) $\mathcal{G}_g\{e^{irt}f(t)\} = \tilde{f}_g(t,\omega-r)$ *(modulation),*

(4) $\mathcal{G}_g\{\overline{f(t)}\} = \overline{\tilde{f}_g(t,-\omega)}$ *(conjugation).*

Proof. (1) Follows directly from Definition 4.3.

(2) A change of variables by $x = \tau - r$ gives the result:

$$\begin{aligned}
\mathcal{G}_g\{f(t-r)\} &= \int_{-\infty}^{\infty} f(\tau - r)g(\tau - t)e^{-i\omega\tau}d\tau \\
&= \int_{-\infty}^{\infty} f(x)g(x-(t-r))e^{-i\omega(x+r)}dx \\
&= e^{-i\omega r}\int_{-\infty}^{\infty} f(x)g(x-(t-r))e^{-i\omega x}dx \\
&= e^{-i\omega r}\tilde{f}_g(t-r,\omega).
\end{aligned}$$

(3) By applying Definition 4.3, we obtain

$$\begin{aligned}\mathcal{G}_g\{e^{irt}f(t)\} &= \int_{-\infty}^{\infty} e^{ir\tau}f(\tau)g(\tau-t)e^{-i\omega\tau}d\tau \\ &= \int_{-\infty}^{\infty} f(\tau)g(\tau-t)e^{-i(\omega-r)\tau}dx \\ &= \tilde{f}_g(t,\omega-r).\end{aligned}$$

(4) Note that the Gaussian function g is real. Applying the conjugate multiple times gives us

$$\begin{aligned}\mathcal{G}_g\{\overline{f(t)}\} &= \int_{-\infty}^{\infty} \overline{f(\tau)}g(\tau-t)e^{-i\omega\tau}d\tau \\ &= \int_{-\infty}^{\infty} \overline{f(\tau)g(\tau-t)e^{i\omega\tau}}d\tau \\ &= \overline{\int_{-\infty}^{\infty} f(\tau)g(\tau-t)e^{i\omega\tau}d\tau} \\ &= \overline{\tilde{f}_g(t,-\omega)}.\end{aligned}$$

□

Theorem 4.7 (Parseval's formula)**.** *Let $f, h \in L^2(\mathbb{R})$ and g a Gaussian function. Additionally, let $\tilde{f}$ and $\tilde{h}$ be the Gabor transforms of f and h with respect to g. Then*

$$\langle \tilde{f}, \tilde{h} \rangle_{L^2(\mathbb{R}^2)} = 2\pi \|g\|^2 \langle f, h \rangle_{L^2(\mathbb{R})}.$$

Proof. (i) Let $f, h \in L^2(\mathbb{R})$ and g a Gaussian function. Now assume that $t \in \mathbb{R}$ is fixed and define

$$\phi(\tau) = f(\tau)g(\tau-t), \qquad \psi(\tau) = h(\tau)g(\tau-t).$$

Then, we get by Definition 2.7 and Definition 4.3 that

$$\begin{aligned}\tilde{f}(t,\omega) &= \int_{-\infty}^{\infty} f(\tau)g(\tau-t)e^{-i\omega\tau}d\tau = \int_{-\infty}^{\infty} \phi(\tau)e^{-i\omega\tau}d\tau = \hat{\phi}(\omega), \\ \tilde{h}(t,\omega) &= \int_{-\infty}^{\infty} h(\tau)g(\tau-t)e^{-i\omega\tau}d\tau = \int_{-\infty}^{\infty} \psi(\tau)e^{-i\omega\tau}d\tau = \hat{\psi}(\omega).\end{aligned}$$

(ii) Using this and Theorem 2.32, we obtain

$$\begin{aligned}\int_{-\infty}^{\infty} \tilde{f}(t,\omega)\overline{\tilde{h}(t,\omega)}d\omega &= \int_{-\infty}^{\infty} \hat{\phi}(\omega)\overline{\hat{\psi}(\omega)}d\omega \\ \overset{\text{(Theorem 2.32)}}{} &= 2\pi \int_{-\infty}^{\infty} \phi(\tau)\overline{\psi(\tau)}d\tau \\ &= 2\pi \int_{-\infty}^{\infty} f(\tau)g(\tau-t)\overline{h(\tau)g(\tau-t)}d\tau \\ &= 2\pi \int_{-\infty}^{\infty} f(\tau)\overline{h(\tau)}\,|g(\tau-t)|^2 d\tau.\end{aligned}$$

(iii) It follows by the second item, Fubini's theorem and the change of variables $x = \tau - t$ that

$$\begin{aligned}\langle \tilde{f}, \tilde{h}\rangle_{L^2(\mathbb{R}^2)} &= \int_{-\infty}^{\infty}\int_{-\infty}^{\infty} \tilde{f}(t,\omega)\overline{\tilde{h}(t,\omega)}d\omega dt \\ \overset{\text{(second item)}}{} &= 2\pi \int_{-\infty}^{\infty}\int_{-\infty}^{\infty} f(\tau)\overline{h(\tau)}|g(\tau-t)|^2 d\tau dt \\ \overset{\text{(Fubini)}}{} &= 2\pi \int_{-\infty}^{\infty}\int_{-\infty}^{\infty} f(\tau)\overline{h(\tau)}|g(\tau-t)|^2 dt d\tau \\ &= 2\pi \int_{-\infty}^{\infty} f(\tau)\overline{h(\tau)}\left(\int_{-\infty}^{\infty} |g(\tau-t)|^2 dt\right) d\tau \\ \overset{(x=\tau-t)}{} &= 2\pi \int_{-\infty}^{\infty} f(\tau)\overline{h(\tau)}\left(\int_{-\infty}^{\infty} |g(x)|^2 dx\right) d\tau \\ &= 2\pi \|g\|^2 \int_{-\infty}^{\infty} f(\tau)\overline{h(\tau)}d\tau \\ &= 2\pi \|g\|^2 \langle f, h\rangle_{L^2(\mathbb{R})}.\end{aligned}$$

□

Remark 4.8. *It is easy to see that for $f \in L^2(\mathbb{R})$, the resulting Gabor transform $\tilde{f}(t,\omega)$ is in $L^2(\mathbb{R}^2)$ because*

$$\|\tilde{f}\| = \langle \tilde{f}, \tilde{f}\rangle_{L^2(\mathbb{R}^2)} = 2\pi \|g\|^2 \langle f, f\rangle_{L^2(\mathbb{R})} = 2\pi \|g\|^2 \|f\|^2 < \infty.$$

Thus, it follows from the previous theorem that the Gabor transform is an operator from $L^2(\mathbb{R}) \to L^2(\mathbb{R}^2)$.

Theorem 4.9 (Inverse Gabor transform in $L^2(\mathbb{R})$)**.** *If $f \in L^2(\mathbb{R})$ and g is a Gaussian function, then*

$$f(t) = \mathcal{G}^{-1}\{\tilde{f}_g(\tau,\omega)\} = \frac{1}{2\pi}\frac{1}{\|g\|^2}\int_{-\infty}^{\infty}\int_{-\infty}^{\infty} \tilde{f}_g(\tau,\omega)g(t-\tau)e^{i\omega t}d\omega d\tau$$

is the inverse Gabor transform.

Proof. (i) Let $f \in L^2(\mathbb{R})$ and let g be a Gaussian function. By construction of the Gabor transform, $\tilde{f}$ exists and is in $L^2(\mathbb{R}^2)$.

(ii) Fix $\tau \in \mathbb{R}$. Then $f(t)g(t-\tau) \in L^1(\mathbb{R})$ as shown in Theorem 4.2. Furthermore, we have

$$\mathcal{F}\{f(t)g(t-\tau)\} = \tilde{f}_g(\tau, \omega)$$

for the Fourier transform with respect to t. Thus, we may apply Definition 2.16—the inverse Fourier transform for a function in $L^1(\mathbb{R})$—to obtain

$$f(t)g(t-\tau) = \mathcal{F}^{-1}\{\tilde{f}_g(\tau, \omega)\} = \frac{1}{2\pi}\int_{-\infty}^{\infty} \tilde{f}_g(\tau, \omega)e^{i\omega t}d\omega. \tag{4.2}$$

(iii) Using the result from the second item, we get

$$\begin{aligned} f(t)\,\|g\|^2 &= f(t)\int_{-\infty}^{\infty} |g(t-\tau)|^2 d\tau \\ &= \int_{-\infty}^{\infty} \Big(f(t)g(t-\tau)\Big)g(t-\tau)d\tau \\ &\overset{(4.2)}{=} \int_{-\infty}^{\infty} \left(\frac{1}{2\pi}\int_{-\infty}^{\infty} \tilde{f}_g(\tau, \omega)e^{i\omega t}d\omega\right) g(t-\tau)d\tau \\ &= \frac{1}{2\pi}\int_{-\infty}^{\infty}\int_{-\infty}^{\infty} \tilde{f}_g(\tau, \omega)g(t-\tau)e^{i\omega t}d\omega d\tau. \end{aligned}$$

This is equivalent to the proposition. □

Theorem 4.10 (Conservation of energy)**.** *If $f \in L^2(\mathbb{R})$ and g is a Gaussian function with $g(t) = g(-t)$ for all $t \in \mathbb{R}$, then*

$$\|f\|^2 = \frac{1}{2\pi}\frac{1}{\|g\|^2}\,\|\tilde{f}_g\|_{L^2(\mathbb{R}^2)}.$$

Proof. (i) We first calculate the Fourier transform of $\tilde{f}_g(t, \omega)$ with respect to t for a fixed $\omega \in \mathbb{R}$. By the symmetry of g, Fubini's theorem and the change of

variables $u = t - \tau$ we deduce

$$\begin{aligned}
\mathcal{F}\{\tilde{f}_g(t,\omega)\}(\nu) &= \int_{-\infty}^{\infty}\left(\int_{-\infty}^{\infty} f(\tau)g(\tau-t)e^{-i\omega\tau}d\tau\right)e^{-i\nu t}dt \\
\text{(symmetry of } g) &= \int_{-\infty}^{\infty}\left(\int_{-\infty}^{\infty} f(\tau)g(t-\tau)e^{-i\omega\tau}d\tau\right)e^{-i\nu t}dt \\
\text{(Fubini)} &= \int_{-\infty}^{\infty}\left(\int_{-\infty}^{\infty} g(t-\tau)e^{-i\nu t}dt\right)f(\tau)e^{-i\omega\tau}d\tau \\
(u=t-\tau) &= \int_{-\infty}^{\infty}\left(\int_{-\infty}^{\infty} g(u)e^{-i\nu(u+\tau)}du\right)f(\tau)e^{-i\omega\tau}d\tau \\
&= \int_{-\infty}^{\infty}\left(\int_{-\infty}^{\infty} g(u)e^{-i\nu u}du\right)f(\tau)e^{-i(\nu+\omega)\tau}d\tau \\
&= \int_{-\infty}^{\infty}\hat{g}(\nu)f(\tau)e^{-i(\nu+\omega)\tau}d\tau \\
&= \hat{f}(\nu+\omega)\hat{g}(\nu).
\end{aligned}$$

(ii) We remember from Theorem 2.26 that

$$\|\hat{f}\|^2 = 2\pi\,\|f\|^2, \qquad \|\hat{g}\|^2 = 2\pi\,\|g\|^2.$$

Thus,

$$\|\mathcal{F}\{\tilde{f}_g(t,\omega)\}\|^2 = 2\pi\,\|\tilde{f}_g(\cdot,\omega)\|^2$$

for any fixed $\omega \in \mathbb{R}$.

(iii) By using the first two items, applying Fubini's theorem and the change of

variables $u = \nu + \omega$ we obtain

$$\begin{aligned}
\frac{1}{2\pi}\frac{1}{\|g\|^2}\,\|\tilde{f}_g\|_{L^2(\mathbb{R}^2)} &= \frac{1}{2\pi}\frac{1}{\|g\|^2}\int_{-\infty}^{\infty}\int_{-\infty}^{\infty}|\tilde{f}_g(t,\omega)|^2 dt d\omega \\
&= \frac{1}{2\pi}\frac{1}{\|g\|^2}\int_{-\infty}^{\infty}\|\tilde{f}_g(\cdot,\omega)\|^2 d\omega \\
&\overset{\text{(second item)}}{=} \frac{1}{2\pi}\frac{1}{\|g\|^2}\int_{-\infty}^{\infty}\frac{1}{2\pi}\,\|\mathcal{F}\{\tilde{f}_g(t,\omega)\}\|^2 d\omega \\
&\overset{\text{(first item)}}{=} \frac{1}{(2\pi)^2}\frac{1}{\|g\|^2}\int_{-\infty}^{\infty}\|\hat{f}(\cdot+\omega)\hat{g}(\cdot)\|^2 d\omega \\
&= \frac{1}{(2\pi)^2}\frac{1}{\|g\|^2}\int_{-\infty}^{\infty}\int_{-\infty}^{\infty}|\hat{f}(\nu+\omega)\hat{g}(\nu)|^2 d\nu d\omega \\
&\overset{\text{(Fubini)}}{=} \frac{1}{(2\pi)^2}\frac{1}{\|g\|^2}\int_{-\infty}^{\infty}\left(\int_{-\infty}^{\infty}|\hat{f}(\nu+\omega)|^2\;d\omega\right)|\hat{g}(\nu)|^2 d\nu \\
&\overset{(u=\nu+\omega)}{=} \frac{1}{(2\pi)^2}\frac{1}{\|g\|^2}\int_{-\infty}^{\infty}\left(\int_{-\infty}^{\infty}|\hat{f}(u)|^2\;du\right)|\hat{g}(\nu)|^2 d\nu \\
&= \frac{1}{(2\pi)^2}\frac{1}{\|g\|^2}\,\|\hat{f}\|^2\;\|\hat{g}\|^2 \\
&= \left(\frac{1}{2\pi}\,\|\hat{f}\|^2\right)\left(\;\frac{1}{2\pi}\;\frac{1}{\|g\|^2}\|\hat{g}\|^2\right) \\
&\overset{\text{(second item)}}{=} \|f\|^2.
\end{aligned}$$

□

Phyiscally speaking, the Gabor transform converts a signal f (of one variable) into a function $\tilde{f}$ (of two variables) without changing its total energy.

4.3 Discrete Gabor transform

In the last section, we have learned about the Gabor transform and its similarities to the Fourier transform. For this reason, we may use the chapter about the discrete Fourier transform as basis for this section and follow the same path.

Consider now a Gaussian function g and a function f with its Gabor transform

$$\tilde{f}_g(t,\omega) = \int_{-\infty}^{\infty} f(\tau)g(\tau - t)e^{-i\omega\tau}d\tau.$$

In real-world applications, f is a signal that is given or measured over a limited amount of time. Due to the nature of the Gaussian function g, the Gabor transform

$\tilde{f}_g$ is negligible outside some interval, say $[a, b]$. The strong decay of g motivates the truncation of the integral in the discrete case. To start, let us divide this interval it into $n \in \mathbb{N}$ equidistant pieces of width $h = \frac{b-a}{n}$. We define

$$a = t_0 < t_1 < t_2 < \ldots < t_{n-1} < t_n = b,$$
$$a = \tau_0 < \tau_1 < \tau_2 < \ldots < \tau_{n-1} < \tau_n = b.$$

Hence,

$$t_k = \tau_k = a + hk = a + \tfrac{b-a}{n}\, k, \quad k = 0, 1, \ldots, n. \tag{4.3}$$

The discetization of the frequencies can again be done with

$$\omega_m = \frac{2\pi m}{b-a}, \quad m = 0, 1, \ldots, n. \tag{4.4}$$

The following function Ψ is now a good approximation of $\tilde{f}_g$:

$$\begin{aligned}
\tilde{f}_g(t, \omega) &= \int_{-\infty}^{\infty} f(\tau) g(\tau - t) e^{-i\omega\tau} d\tau \\
&\approx \int_a^b f(\tau) g(\tau - t) e^{-i\omega\tau} d\tau \\
&\approx \sum_{k=0}^{n-1} h f(\tau_k) g(\tau_k - t) e^{-i\omega\tau_k} = \Psi(t, \omega).
\end{aligned}$$

Using (4.3) and (4.4), we obtain

$$\begin{aligned}
\Psi(t_j, \omega_m) &= \sum_{k=0}^{n-1} h f(\tau_k) g(\tau_k - t_j) e^{-i\omega_m \tau_k} \\
&= h \sum_{k=0}^{n-1} f(\tau_k) g(\tau_k - t_j) e^{-i\omega_m \left(a + \frac{b-a}{n} k\right)} \\
&= h e^{-ia\omega_m} \sum_{k=0}^{n-1} f(\tau_k) g(\tau_k - t_j) e^{-i \frac{2\pi m}{b-a} \frac{b-a}{n} k} \\
&= h e^{-ia\omega_m} \sum_{k=0}^{n-1} f(\tau_k) g(\tau_k - t_j) e^{-\frac{2\pi i}{n} mk}.
\end{aligned}$$

Once again, we neglect the term $he^{-ia\omega_m}$. In order to get the discrete Gabor transform, we also require to discretize the function g. This can be achieved by simple evaluation of the continuous Gaussian function, but we discover that $\tau_k - t_j$ could be outside of $[a, b]$. We may resolve this issue by assuming that the discrete Gaussian g is restricted to $[a, b]$ and then periodically extended. To avoid discontinuities of g at the interval ends, we also make sure it is centered in the interval.

Definition 4.11 (Discrete Gaussian function)**.** *A discrete Gaussian $g : \mathbb{Z}_n \to \mathbb{R}$ is a function of the form*

$$g(k) = \alpha e^{-\beta^2(k-t_c)^2}, \quad k \in \mathbb{Z}_n$$

with fixed $\alpha, \beta > 0$ and $t_c \in \mathbb{Z}_n$. We say that g is centered, if furthermore $t_c = \frac{n}{2}$.

Remark 4.12. *If $g : \mathbb{Z}_n \to \mathbb{R}$ is a centered discrete Gaussian function, then*

$$g(k) = g(n-k) = g(-k)$$

for all $k \in \mathbb{Z}_n$.

In the same way we have seen it in the derivation of the discrete Fourier transform, we obtain that the the discrete versions of f and $\tilde{f}_g$ will be $[a, b]$-periodic as well.

Definition 4.13 (Discrete Gabor transform)**.** *Let $f : \mathbb{Z}_n \to \mathbb{C}$ be any function and let $g : \mathbb{Z}_n \to \mathbb{R}$ be a centered discrete Gaussian function. We define the discrete Gabor transform by*

$$\mathcal{G}_{d,g}\{f\}(j,m) = \tilde{f}_g(j,m) = \sum_{k=0}^{n-1} f(k)g(k-j)w_n^{mk}$$

for $j, m \in \mathbb{Z}_n$.

Remark 4.14. *(i) The resulting discrete Gabor transform of f with respect to the Gaussian function g is defined as $\tilde{f}_g : \mathbb{Z}_n \times \mathbb{Z}_n \to \mathbb{C}$.*

(ii) The argument $k - j$ of g has to be calculated modulo n, as g only operates on $\mathbb{Z}_n$.

(iii) If f is seen as a vector in $\mathbb{C}^n$, then $\tilde{f}_g$ can be seen as a complex square matrix of size $n \times n$, thus $\tilde{f}_g \in \mathbb{C}^{n\times n}$.

(iv) We are using the same notation $\tilde{f}_g$ for the discrete Gabor transform as in the last section for the continuous Gabor transform.

Similar to its continuous counterpart, the discrete Gabor transform can be seen as a discrete Fourier transform or discrete convolution:

Remark 4.15. *(i) If we fix the time variable $j \in \mathbb{Z}_n$, then we can write*

$$\tilde{f}_g(j,m) = \sum_{k=0}^{n-1} \phi_j(k) w_n^{mk} = \hat{\phi}_j(m) \tag{4.5}$$

for $\phi_j(k) = f(k)g(k-j)$.

(ii) Fix the frequency variable $m \in \mathbb{Z}_n$. Then, we obtain

$$\begin{aligned}\tilde{f}_g(j,m) &= \sum_{k=0}^{n-1} f(k)g(k-j)w_n^{mk} \\ &= w_n^{mj} \sum_{k=0}^{n-1} f(k)g(k-j)w_n^{m(k-j)} \\ &= w_n^{mj} \sum_{k=0}^{n-1} f(k)g(j-k)w_n^{-m(j-k)} \\ &= w_n^{mj}(f * g_m)\end{aligned}$$

for $g_m(j) = g(j)w_n^{-mj}$.

From Theorem 3.27 we know that we are able to compute the discrete Fourier transform with complexity $\mathcal{O}(n\log_2(n))$. Hence, the discrete Gabor transform will have complexity $\mathcal{O}(n^2\log_2(n))$ if we use the formula from the first item in the remark above.

Theorem 4.16 (Basic properties of the discrete Gabor transform)**.** *Let g be a centered discrete Gaussian function. Assume we have the functions $f, h : \mathbb{Z}_n \to \mathbb{C}$ and their corresponding discrete Gabor transforms $\tilde{f}_g$, $\tilde{h}_g$ with respect to g. Additionally, set the scalar values $a, b \in \mathbb{C}$. Then the following properties hold for $j, k, l, m \in \mathbb{Z}_n$:*

(1) $\mathcal{G}_{d,g}\{a\, f(j) + b\, g(j)\}(j,m) = a\, \tilde{f}_g(j,m) + b\, \tilde{h}_g(j,m)$ *(linearity),*

(2) $\mathcal{G}_{d,g}\{f(j-l)\}(j,m) = w_n^{ml}\tilde{f}_g(j-l,m)$ *(translation),*

(3) $\mathcal{G}_{d,g}\{f(j)w_n^{-jl}\}(j,m) = \tilde{f}(j,m-l)$ *(modulation),*

(4) $\mathcal{G}_{d,g}\{\overline{f(j)}\}(j,m) = \overline{\tilde{f}_g(j,-m)}$ *(conjugation).*

Proof. (1) The linearity follows directly from Definition 4.13.

(2) Let $h(j) = f(j-l)$. We obtain by Definition 4.13

$$\begin{aligned}
\mathcal{G}_{d,g}\{f(j-l)\}(j,m) = \tilde{h}_g(j,m) &= \sum_{k=0}^{n-1} h(k)g(k-j)w^{mk} \\
&= \sum_{k=0}^{n-1} f(k-l)g(k-j)w^{mk} \\
&= \sum_{k=-l}^{n-1-l} f(k)g((k+l)-j)w^{m(k+l)} \\
&= \sum_{k=0}^{n-1} f(k)g(k-(j-l))w^{m(k+l)} \\
&= w^{ml}\sum_{k=0}^{n-1} f(k)g(k-(j-l))w^{mk} \\
&= w^{ml}\tilde{f}_g(j-l,m).
\end{aligned}$$

(3) By applying Definition 4.13, we obtain

$$\begin{aligned}
\mathcal{G}_{d,g}\{f(j)w_n^{-jl}\}(j,m) &= \sum_{k=0}^{n-1} f(k)w_n^{-kl}g(k-j)w_n^{mk} \\
&= \sum_{k=0}^{n-1} f(k)g(k-j)w_n^{(m-l)k} \\
&= \tilde{f}_g(j,m-l).
\end{aligned}$$

(4) Using that g is a real function, we get by application of the conjugate

$$\begin{aligned}
\mathcal{G}_{d,g}\{\overline{f(j)}\}(j,m) &= \sum_{k=0}^{n-1} \overline{f(k)}g(k-j)w_n^{mk} \\
&= \sum_{k=0}^{n-1} \overline{f(k)g(k-j)w_n^{(-m)k}} \\
&= \overline{\tilde{f}_g(j,-m)}.
\end{aligned}$$

□

Theorem 4.17 (Hermitian symmetry)**.** *Let $f : \mathbb{Z}_n \to \mathbb{R}$ be a real discrete function and g a centered discrete Gaussian. Further, let $\tilde{f}_g$ be its corresponding discrete Gabor transform with respect to g. Then*

$$\tilde{f}_g(j,n-m) = \overline{\tilde{f}_g(j,m)}$$

for all $j, m \in \mathbb{Z}_n$.

Proof. Let f, g and $\tilde{f}_g$ as stated in the theorem. If we fix $j \in \mathbb{Z}_n$, then the discrete Gabor transform can be expressed as the discrete Fourier transform as shown in (4.5). Using Theorem 3.15, we obtain

$$\begin{aligned}\tilde{f}_g(j, n-m) &= \sum_{k=0}^{n-1} \phi_j(k) w_n^{(n-m)k} = \hat{\phi}_j(n-m) \\ &\overset{\text{(Theorem 3.15)}}{=} \overline{\hat{\phi}_j(m)} = \sum_{k=0}^{n-1} \overline{\phi_j(k) w_n^{mk}} = \overline{\tilde{f}_g(j, m)}\end{aligned}$$

for every $m \in \mathbb{Z}_n$, where $\phi_j(k) = f(k)g(k-j)$. □

After having seen the most important properties of the discrete Gabor transform, we are now ready to discuss its inverse.

For our discretization in (4.3) and (4.4), we assumed the same amount of samples n in time and frequency domain. Most authors choose a more general form of the discrete Gabor transform because they subsample the time variable j in Definition 4.13. Then, we would get

$$\tilde{f}_g(j, m) = \sum_{k=0}^{n-1} f(k) g(k - cj) w_n^{mk},$$

where $c \in \mathbb{N}$. In that case, the resulting matrix is not square. Throughout this thesis, we will forgo using this general form and stick with our original definition.

Even by using our version of the discrete Gabor transform, it is not easy to prove the invertibility. Finding the conditions on the discretization and on the Gaussian under which we achieve invertibility is also known as the Gabor representation problem and requires the study of the so-called Gabor frames. For details, we refer to [Chr08] and [DS15].

Theorem 4.18 (Discrete inverse Gabor transform)**.** *Let* $g : \mathbb{Z}_n \to \mathbb{R}$ *be a centered discrete Gaussian function and let* $f : \mathbb{Z}_n \to \mathbb{C}$ *be such that* $\tilde{f}_g$ *is its discrete Gabor transform as defined in Definition 4.13. Then, the discrete inverse Gabor transform is*

$$\mathcal{G}_d^{-1}\{\tilde{f}_g\}(m) = f(m) = \frac{1}{n} \frac{1}{\|g\|^2} \sum_{j=0}^{n-1} \sum_{k=0}^{n-1} \tilde{f}_g(j, k) g(m-j) w_n^{-mk}$$

for $m \in \mathbb{Z}_n$*, where the norm* $\|\cdot\|$ *denotes the Euclidean norm in* $\mathbb{R}^n$.

Proof. (i) From the first item in the proof of Theorem 3.19 we know that

$$\sum_{k=0}^{n-1} w_n^{-k(m-l)} = n\delta_{l,m}$$

for appropriate $k, l, m \in \mathbb{Z}_n$.

(ii) Let f, g and $\tilde{f}_g$ be as given in the proposition. Using the first item, we get for $m \in \mathbb{Z}_n$

$$\begin{aligned}
&\frac{1}{n}\frac{1}{\|g\|^2}\sum_{j=0}^{n-1}\sum_{k=0}^{n-1}\tilde{f}_g(j,k)g(m-j)w_n^{-mk} \\
&\quad= \frac{1}{n}\frac{1}{\|g\|^2}\sum_{j=0}^{n-1}\sum_{k=0}^{n-1}\sum_{l=0}^{n-1} f(l)g(l-j)w_n^{kl}g(m-j)w_n^{-mk} \\
&\quad= \frac{1}{n}\frac{1}{\|g\|^2}\sum_{j=0}^{n-1}\sum_{l=0}^{n-1} f(l)g(l-j)g(m-j)\sum_{k=0}^{n-1} w_n^{-k(m-l)} \\
&\quad= \frac{1}{n}\frac{1}{\|g\|^2}\sum_{j=0}^{n-1}\sum_{l=0}^{n-1} f(l)g(l-j)g(m-j)n\delta_{l,m} \\
&\quad= \frac{1}{\|g\|^2}\sum_{j=0}^{n-1} f(m)g(m-j)g(m-j) \\
&\quad= f(m)\frac{1}{\|g\|^2}\sum_{j=0}^{n-1} g(j)^2 = f(m).
\end{aligned}$$

□

Remark 4.19. *The discrete inverse Gabor transform can be expressed using the discrete inverse Fourier transform. Hence,*

$$\begin{aligned}
f(m) &= \frac{1}{n}\frac{1}{\|g\|^2}\sum_{j=0}^{n-1}\sum_{k=0}^{n-1}\tilde{f}_g(j,k)g(m-j)w_n^{-mk} \\
&= \frac{1}{\|g\|^2}\sum_{j=0}^{n-1} g(m-j)\left(\frac{1}{n}\sum_{k=0}^{n-1}\tilde{f}_g(j,k)w_n^{-mk}\right) \\
&= \frac{1}{\|g\|^2}\sum_{j=0}^{n-1} g(m-j)\mathcal{F}_d^{-1}\{\tilde{f}_g(j,k)\}(j,m).
\end{aligned}$$

4.4 Implementation of the discrete Gabor transform

To finish this chapter, we are going to provide an implementation of the discrete Gabor transform in MATLAB. Using the code provided, we will discuss a few examples that appeared in the previous chapters in this thesis.

We only provide simple working examples for algorithms in MATLAB in this section. A full and more elegant implementation—including error handling and comments—is shown in Listing 12 at the end of this thesis.

First, we would like to implement the discrete Gabor transform. We remember from (4.5), that we may calculate the discrete Gabor transform using the fast Fourier transform. Employing this idea, we are able to write down a simple algorithm in MATLAB as shown in Listing 5.

Listing 5: Discrete Gabor transform .

```
function fTilde = fastgabortransform(f, g)

    n = max(size(f));
    fTilde = zeros(n,n);
    tInd = 0:n-1;

    for j = 0:n-1
        tShift = mod(tInd - j, n) + 1;
        phi = f .* g(tShift);
        fTilde(j+1, :) = fastfouriertransform(phi);
    end

end
```

In comparison to the inverse of the fast Fourier transform, it is slightly more difficult to implement the discrete inverse Gabor transform. We may use the form derived in Remark 4.19 in order to get an efficient algorithm as shown in Listing 6.

Listing 6: Discrete inverse Gabor transform .

```
function f = inversefastgabortransform(fTilde, g)

    n = max(size(fTilde));
    f = zeros(n, 1);
    tInd = 0:n-1;

    for j = 0:n-1
```

```
        tShift = mod(tInd - j, n) + 1;
        phiHat = fTilde(j+1, :).';
        phi = inversefastfouriertransform(phiHat);
        f = f + g(tShift) .* phi;
    end

    f = 1/norm(g, 2).^2 * f;

end
```

We are now ready to use the algorithm of the fast Gabor transform in some examples and plot the results in MATLAB. For the following examples, we set $n = 2^7 = 128$ and consider the interval $[a, b] = [-1, 1] \subset \mathbb{R}$. In the appearing figures, we plot the original input function against the discrete Gabor transform with respect to the discretized Gaussian function $g(t) = e^{-\beta^2 t^2}$. The resulting transforms have been shifted in time by one unit to enable the comparability to the original functions. Additionally, the logarithm has been taken in order to decrease the dominance of large values. As the graph of the discrete Gabor transform is three-dimensional, we chose to show the surface plot from above.

To start our series of examples, we look at three different discrete impulse functions.

Example 4.20. *In this example, we fix $\beta = 10$, thus the Gaussian window function is defined as $g(t) = e^{-(10t)^2}$.*

(1) Define the function

$$u(t) = \begin{cases} 1, & -0.5 \leq t \leq +0.5, \\ 0, & \textit{otherwise}. \end{cases}$$

Figure 3 shows the original function u against the discrete Gabor transform $\tilde{u}_g$.

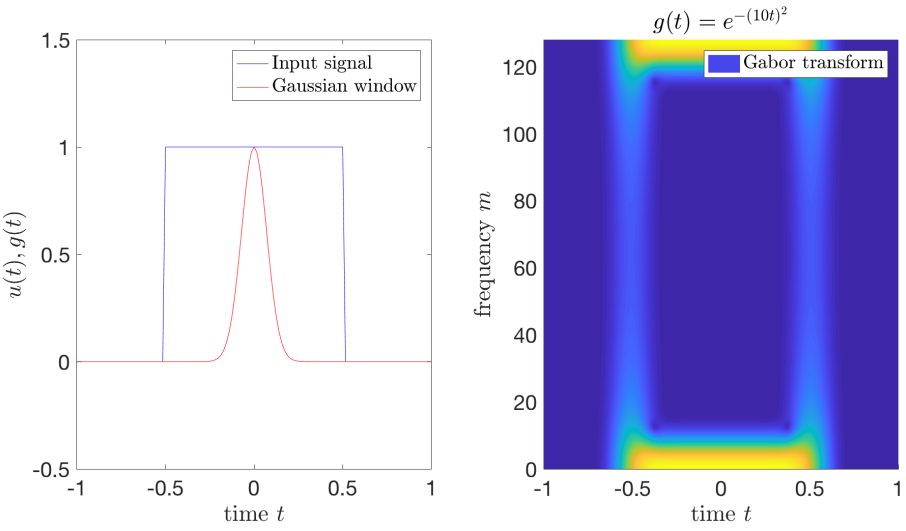


Figure 3: On the left $u(t)$ and $g(t)$, on the right $|\tilde{u}_g(t, \pi m)|$.

We can see that the resulting Gabor transform is basically zero (blue) with a small peak (yellow) around times -0.5 *to* 0.5 *and frequency* 0. *More interestingly, we are able to see the discontinuity at* $t = \pm 0.5$ *for all frequencies (light blue).*

(2) Consider the function

$$v(t) = \begin{cases} -0.5, & t < 0, \\ +0.5, & t \geq 0. \end{cases}$$

Again, we compute the discrete Gabor transform and obtain the graphs as shown in Figure 4.

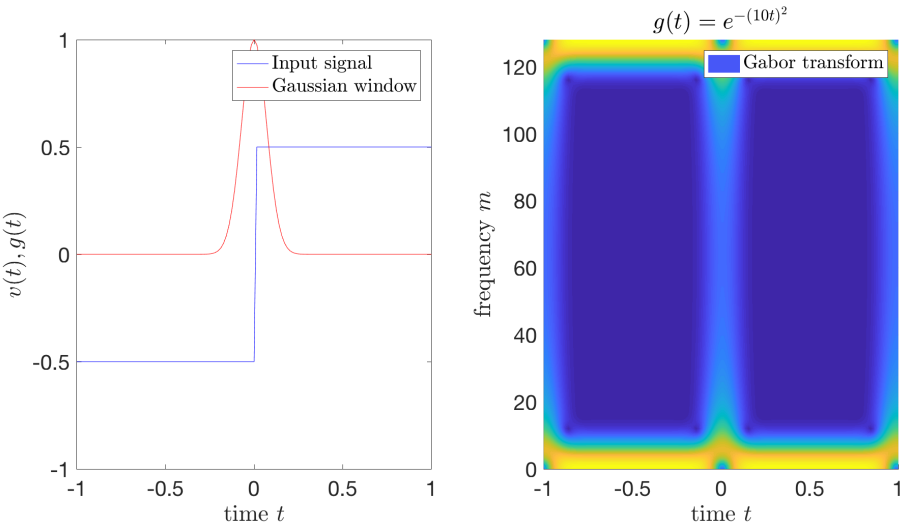


Figure 4: On the left $v(t)$ and $g(t)$, on the right $|\tilde{v}_g(t, \pi m)|$.

The resulting Gabor transform looks similar as in the first example, but it has its peaks at frequency 0 *and at times when the original function was horizontal. The discontinuity for* $t = 0$ *is clearly visible in the Gabor transform for all frequencies. As the function* v *is* $[-1, 1]$*-periodic, we are able to see the discontinuity at* $t = \pm 1$ *as well.*

(3) Consider the function

$$w(t) = \begin{cases} -0.5, & t < -0.5, \\ t, & -0.5 \leq t \leq +0.5, \\ +0.5, & t > +0.5. \end{cases}$$

Figure 5 shows the function w *and its Gabor transform* $\tilde{w}_g$*.*

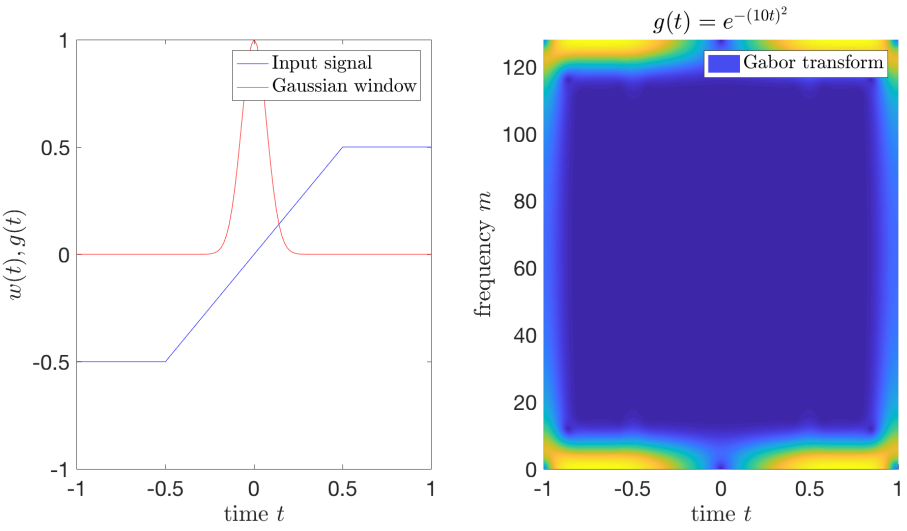


Figure 5: On the left $w(t)$ and $g(t)$, on the right $|\tilde{w}_g(t, \pi m)|$.

As in the second example, peaks are visible at frequency 0 *and at times when the original function was horizontal. At time of the value increase for* $t \in [-0.5, +0.5]$*, the resulting transform is close to zero for all frequencies. While it shows the discontinuity at times* $t = \pm 1$ *for all frequencies, it does nearly not reveal the kink at* $t = \pm 0.5$*. Zooming in at the points* $(t, m) = (\pm 0.5, 15)$*, we can see a tiny bulge which indicates the kink.*

In the last chapter, we have seen a function that was composed of three different frequencies in Example 3.29. We now apply the discrete Gabor transform to the very same function and look at the resulting plot.

Example 4.21. *Consider the periodic function*

$$f(t) = 2\sin(2t\pi) + 3\sin(15t\pi) + 5\sin(7t\pi).$$

As noted, this function has frequencies 2, 7 *and* 15 *on the interval* $[-1, 1]$*. For this example, we choose to set* $\beta = 2$*; thus the Gaussian window function is* $g(t) = e^{(-2t)^2}$*. If we look at the Gabor transform in Figure 6, we may see that these three frequencies do not change during the given interval.*

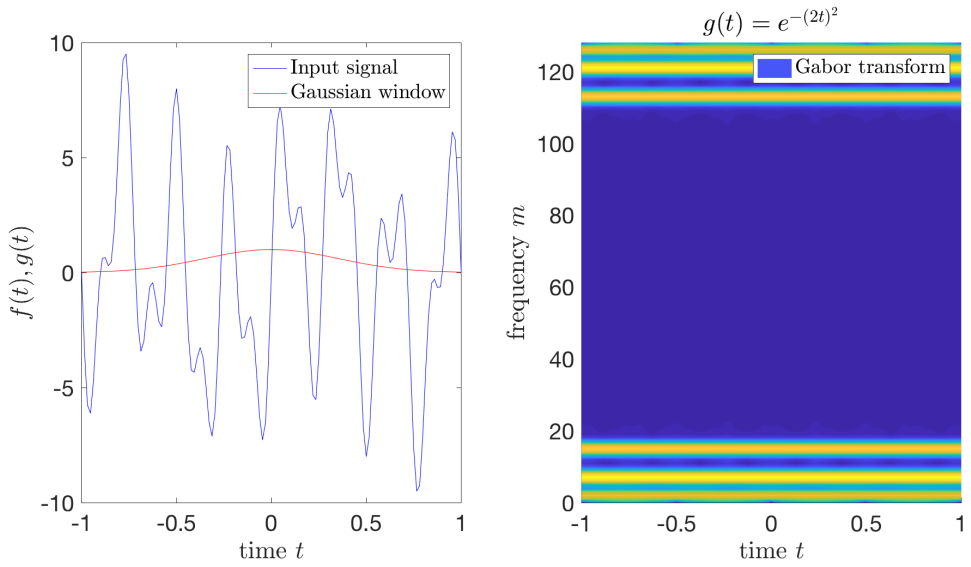


Figure 6: On the left $f(t)$ and $g(t)$, on the right $|\tilde{f}_g(t, \pi m)|$.

It is obvious to the eye that the Gabor transform does not have any advantages over the Fourier transform in this case: The information in time is simply redundant.

So far, we have looked at local bumping functions and a function with constant frequencies over time. We already suspect that the true power of the Gabor transform shows when considering examples with varying frequencies. In real-world applications, this is a common scenario: We may think of an audio recording of music with different notes. The following example shows how changing frequencies affects the graph of the discrete Gabor transform:

Example 4.22. *Let the function*

$$f(t) = \begin{cases} 10\sin(30\pi t) + 13\sin(20\pi t), & t < 0, \\ 15\sin(10\pi t^2), & t \geq 0. \end{cases}$$

For $t \in [-1, 0[$, f consists of the frequencies 20 and 30. For $t \in [0, 1]$, the frequency increases due to the square in the sine. Thus, we expect the Gabor transform to reveal these frequencies within the appropriate time frames. Figure 7 shows the original function f on the top and four Gabor transforms with respect to different Gaussian window functions $g(t) = e^{-\beta^2 t^2}$ on the bottom.

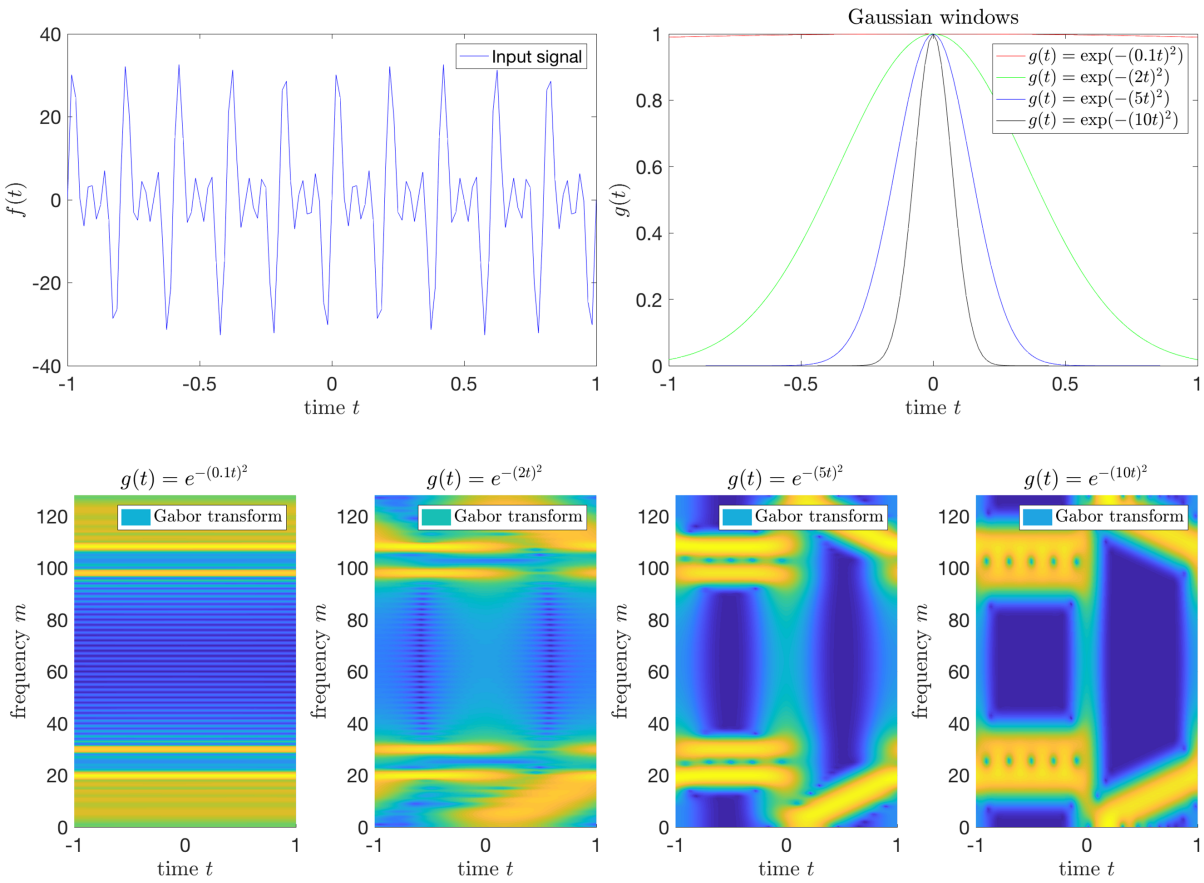


Figure 7: On the top $f(t)$ and $g(t)$, on the bottom $|\tilde{f}_g(t, \pi m)|$.

These graphs are a good illustration of the importance of choosing an appropriate Gaussian window function. It is obvious that β must not be chosen too small or too large in order to get a meaningful result.

If β is too small, then $g \approx 1$ on our interval. But from $g = 1$ we may deduce from Definition 4.13 that

$$\tilde{f}_g(j, m) = \sum_{k=0}^{n-1} f(k)g(k-j)w_n^{mk} = \sum_{k=0}^{n-1} f(k)w_n^{mk} = \hat{f}(m)$$

for all $j, m \in \mathbb{Z}_n$. This is why the first transform graph ($\beta = 0.1$) from Figure 7 yields no useful time-frequency information.

On the other hand, β should not be too large neither. If the sampling rate is not large enough, then the resulting graph looks blurry because we cannot deal with the

locality of the Gaussian. In that case, the discrete Gaussian is just 1 *in the origin and* 0 *otherwise. In Figure 7, the fourth transform graph visualizes this problem for* $\beta = 10$.

Looking at our set of transforms, we get the best result with $\beta = 5$ *as shown in the third transform graph. The time-frequency dependence can be easily used to analyze the original function.*

Following these examples, we summarize our general findings in a quick remark:

Remark 4.23. *(i) Due to the the Hermitian symmetry shown in Theorem 4.17, we can see a repetition of the absolute graph of the discrete Gabor transform above* $m > n/2$.

(ii) The choice of β *in the Gaussian window function is critical in order to get a readable time-frequency graph. From the examples above we find that* β *has to be quite large to detect sharp edges. On the other hand, for smooth and precise detection of specific frequencies of periodic functions, the resulting Gabor transform becomes blurry if* β *is large.*

(iii) The Gabor transform proved to be useful for detecting discontinuities in functions, but it was not able to show small kinks.

With the examples above, we conclude the chapter about the Gabor transform in one dimension.

5 Fourier transform in two dimensions

5.1 Motivation

Having seen the Fourier and Gabor transform in one dimension yields a good basis for their extension to more dimensions. This chapter provides a compact overview of the extension of the Fourier transform to two dimensions.

In one dimension, we imagined a function to be a known audio signal over a time period. The Fourier transform then showed the frequency spectrum of this function over the whole time period. In two dimensions, we can assume a function to define color values of an image. The same way as we used $t \in \mathbb{R}$ to describe the time domain in one dimension, we may use the pair $(x, y) \in \mathbb{R}^2$ to describe the location of a pixel in two dimensions. When looking at the Fourier transform of an image, we will obtain the frequency spectrum of the whole image.

We do not require any new notation for the two-dimensional Fourier transform and refer to Chapter 2 and Chapter 3 for preliminaries.

5.2 Fourier transform in $L^2(\mathbb{R}^2)$

The Fourier theory of one dimension can be easily extended to two (or more) dimensions. Again, the basic definition shall be given for functions in $L^1(\mathbb{R}^2)$. In order to tell apart a one-dimensional from a two-dimensional variable, we will write the latter in bold face:

Definition 5.1 (Fourier transform in $L^1(\mathbb{R}^2)$)**.** *Let $f \in L^1(\mathbb{R}^2)$ and $\boldsymbol{x}, \boldsymbol{\omega} \in \mathbb{R}^2$.*

(1) The Fourier transform of f is defined by

$$\mathcal{F}\{f(\boldsymbol{x})\} = \hat{f}(\boldsymbol{\omega}) = \int_{\mathbb{R}^2} f(\boldsymbol{x}) e^{-i\boldsymbol{\omega}\cdot\boldsymbol{x}} d\boldsymbol{x}.$$

(2) The inverse Fourier transform is defined as

$$\check{f}(\boldsymbol{x}) = \mathcal{F}^{-1}\{f(\boldsymbol{\omega})\} = \frac{1}{(2\pi)^2} \int_{\mathbb{R}^2} f(\boldsymbol{\omega}) e^{i\boldsymbol{\omega}\cdot\boldsymbol{x}} d\boldsymbol{\omega}.$$

Remark 5.2. *(1) We are now dealing with $\boldsymbol{x}, \boldsymbol{\omega} \in \mathbb{R}^2$. The $\cdot$ in the exponent denotes the standard inner product on $\mathbb{R}^2$.*

(2) We may follow the whole theory from Chapter 2 to extend the Fourier transform to $L^2(\mathbb{R}^2)$ in the same way as we have seen in Definition 2.24.

(3) All the theorems of Chapter 2 may be appropriately expanded to two dimensions, but some appearing constant factors need to be adjusted. Comparing Definition 5.1 with Definition 2.7, it is obvious that this factor usually is $(2\pi)^2$ in two dimensions instead of 2π in one dimension.

In Theorem 2.20, we have stated and proved the one-dimensional case of the convolution theorem. As it is one of the most important theorems of Fourier analysis, we hereby repeat the statement for two dimensions:

Theorem 5.3 (Convolution theorem)**.** *Let $f, g \in L^1(\mathbb{R}^2)$, then*

$$\mathcal{F}\{(f * g)(\boldsymbol{x})\} = \hat{f}(\boldsymbol{\omega})\hat{g}(\boldsymbol{\omega}).$$

We omit the proof, as it is just an extension of the existing proof to two dimensions.

5.3 Discrete Fourier transform

We have motivated the construction of the discrete Fourier transform in one dimension in Chapter 3.3. Diving into two dimensions, we may discretize the set $[a_1, b_1] \times [a_2, b_2] \subset \mathbb{R}^2$ by

$$\begin{aligned} a_1 = x_0 < x_1 < x_2 < \dots x_{n_1-1} < x_{n_1} = b_1, \\ a_2 = y_0 < y_1 < y_2 < \dots y_{n_2-1} < y_{n_2} = b_2, \end{aligned}$$

where $n_1, n_2 \in \mathbb{N}$ are the amount of equidistant pieces for each axis. Discretizing the domain of $\hat{f}$ can be done by

$$\begin{aligned} \omega_{m_1} &= \frac{2\pi m_1}{b_1 - a_1}, \quad m_1 = 0, 1, \dots, n_1, \\ \nu_{m_2} &= \frac{2\pi m_2}{b_2 - a_2}, \quad m_2 = 0, 1, \dots, n_2. \end{aligned}$$

Following the same derivation as in Chapter 3.3, we will obtain the definition of the Fourier transform in two dimensions.

Definition 5.4 (Discrete Fourier transform)**.** *Let $f : \mathbb{Z}_{n_1 \times n_2} \to \mathbb{C}$ be a function and $(m_1, m_2) \in \mathbb{Z}_{n_1 \times n_2}$.*

(1) We define the discrete Fourier transform by

$$\mathcal{F}_d\{f\}(m_1, m_2) = \hat{f}(m_1, m_2) = \sum_{k_1=0}^{n_1-1} \sum_{k_2=0}^{n_2-1} f(k_1, k_2) w_{n_1}^{m_1 k_1} w_{n_2}^{m_2 k_2}.$$

(2) If $\hat{f}$ is the discrete Fourier transform as defined above, then the discrete inverse Fourier transform is defined as

$$\mathcal{F}_d^{-1}\{\hat{f}\}(m_1, m_2) = f(m_1, m_2) = \frac{1}{n^2} \sum_{k_1=0}^{n_1-1} \sum_{k_2=0}^{n_2-1} \hat{f}(k_1, k_2) w_{n_1}^{-m_1 k_1} w_{n_2}^{-m_2 k_2}.$$

Remark 5.5. *The discrete Fourier transform in two dimensions is separable in the sense that we may write*

$$\begin{aligned} \hat{f}(m_1, m_2) &= \sum_{k_1=0}^{n_1-1} \sum_{k_2=0}^{n_2-1} f(k_1, k_2) w_{n_1}^{m_1 k_1} w_{n_2}^{m_2 k_2} \\ &= \sum_{k_1=0}^{n_1-1} \left(\sum_{k_2=0}^{n_2-1} f(k_1, k_2) w_{n_2}^{m_2 k_2} \right) w_{n_1}^{m_1 k_1}. \end{aligned}$$

In other words, we have to calculate two one-dimensional discrete Fourier transforms for each pair (m_1, m_2) in order to get the two-dimensional version.

The convolution theorem for the discrete Fourier transform in one dimension has been given in Theorem 3.17. Below-mentioned is its counterpart in two dimensions:

Theorem 5.6 (Discrete convolution theorem)**.** *Let $f, g : \mathbb{Z}_{n_1 \times n_2} \to \mathbb{C}$ be discrete functions with their discrete Fourier transforms $\hat{f}, \hat{g}$. Then,*

$$\mathcal{F}\{f * g\}(m_1, m_2) = \hat{f}(m_1, m_2)\hat{g}(m_1, m_2)$$

for all $(m_1, m_2) \in \mathbb{Z}_{n_1 \times n_2}$.

The previous theorem is specifically important because we are going to use it in our implementation for the Gabor transform in two dimensions in Chapter 6. We omit the proof once again, as it is a direct extension of the one-dimensional case.

5.4 Implementation of the fast Fourier transform

As for the one-dimensional case, we know from Chapter 3 that the fast Fourier transform reduces the complexity from $\mathcal{O}(n^2)$ to $\mathcal{O}(n \log_2(n))$. Due to the separability of the discrete Fourier transform in two dimensions, we may use the algorithms from the one-dimensional case.

Our version of the fast Fourier transform uses the Cooley–Tukey algorithm, where $n = 2^s$ for $s \in \mathbb{N}$. Hence, we require to assume the same in two dimensions. In order to simplify the notation, we shall further use $n = n_1 = n_2 = 2^s$, even though this is not required in general.

Remark 5.5 directly motivates a simple MATLAB implementation. As mentioned before, full working examples with comments are given in Listing 11.

Listing 7: Fast Fourier transform in two dimensions.

```
function fHat = fastfouriertransform2d(f)

    [n, ~] = size(f);

    fHatTemp = zeros(n, n);
    fHat = zeros(n, n);

    for k=1:1:n
       fHatTemp(k,:) = fastfouriertransform(f(k, :).').';
    end

    for k=1:1:n
       fHat(:,k) = fastfouriertransform(fHatTemp(:, k));
    end

end
```

Listing 8: Inverse fast Fourier transform in two dimensions.

```
function f = inversefastfouriertransform2d(fHat)
    n = length(fHat);
    fHat = conj(fHat);
    f = 1/n * 1/n * conj(fastfouriertransform2d(fHat));
end
```

We may deduce from the snippets above that the complexity of the fast Fourier transform in two dimensions is $\mathcal{O}(n^2 \log_2(n))$.

6 Gabor transform in two dimensions

6.1 Motivation

We have dedicated the last chapter to the two-dimensional Fourier transform, which can reveal the whole frequency spectrum of an image. If we would like to learn more about the original location $\boldsymbol{x} = (x, y) \in \mathbb{R}^2$ of a specific frequency $\boldsymbol{\omega} = (\omega, \nu) \in \mathbb{R}^2$, we will have to find other means of analysis.

Similar to the one-dimensional case, the two-dimensional Gabor transform solves this issue because it makes a transformed function depend on both location and frequency. Thus, its mapping is given as $\tilde{f} : \mathbb{R}^4 \to \mathbb{R}$ and has a five-dimensional graph. In the first part of this chapter, we are going to construct the transform in the same way as we have done it in Chapter 4 for one dimension.

While the graph of the Gabor transform provides the desired domain-frequency dependency, it raises other issues. On one hand, even for modern computers it is time-consuming to calculate the discrete transform. On the other hand, it is not easy to visualize all dimensions of the graph at once. It is therefore common to fix a specific frequency $\boldsymbol{\omega}$ and then show the three-dimensional graph in a contour plot. We are going to use that approach and show an implementation in the second part of this chapter.

This chapter contains some remarks from [IKK05], although most results have been built upon the previous chapters of this thesis.

6.2 Gabor transform in $L^2(\mathbb{R}^2)$

We have discussed the one-dimensional Gabor transform in Chapter 4. As with the Fourier transform, we may easily extend the Gabor to two dimensions. The definition of the Gabor transform makes use of a Gaussian function, hence we first need to understand the form of the latter in two dimensions.

A general form of any Gaussian function in one dimension has been given in Definition 4.1. It contains variables for the height (α), width (β), and translation (t_c) of the graph.

In the two-dimensional case, we require the dilation in height as well. Then again, we must consider the width and translation of the Gaussian in directions of both axes of the spatial domain. We additionally require an angle parameter so that the graph of the Gaussian may be rotated.

Definition 6.1 (Gaussian function)**.** *A Gaussian function in two dimensions is a*

function of the form

$$g(x,y) = \alpha e^{-\beta^2 x'^2 - \gamma^2 y'^2}, \quad (x,y) \in \mathbb{R}^2$$

with

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} \cos(\theta) & \sin(\theta) \\ -\sin(\theta) & \cos(\theta) \end{bmatrix} \begin{bmatrix} x - x_c \\ y - y_c \end{bmatrix}$$

such that $\alpha, \beta, \gamma > 0$, $(x_c, y_c) \in \mathbb{R}^2$ *and* $\theta \in [0, 2\pi[$.

Remark 6.2. *Whereas the parameter* α *is the dilation in height, the parameters* β *and* γ *define the width of the Gaussian. The pair* (x_c, y_c) *determines the translation and* θ *the rotation in the* xy*-plane.*

Example 6.3 (Gaussian functions)**.** *Consider the Gaussian functions*

$$\begin{aligned} g_1(x,y) &= e^{-x^2-y^2}, \\ g_2(x,y) &= e^{-x^2-4y^2}, \\ g_3(x,y) &= e^{-(x-1)^2-4(y-2)^2}, \\ g_4(x,y) &= e^{-\left[(x-1)\cos\left(\frac{\pi}{4}\right)+(y-2)\sin\left(\frac{\pi}{4}\right)\right]^2-4\left[-(x-1)\sin\left(\frac{\pi}{4}\right)+(y-2)\cos\left(\frac{\pi}{4}\right)\right]^2}, \end{aligned}$$

where $(x,y) \in \mathbb{R}^2$*. Figure 8 shows a contour plot of all four Gaussian functions in the* xy*-plane. The peaks are indicated by a bright yellow, values close to zero are dark blue.*

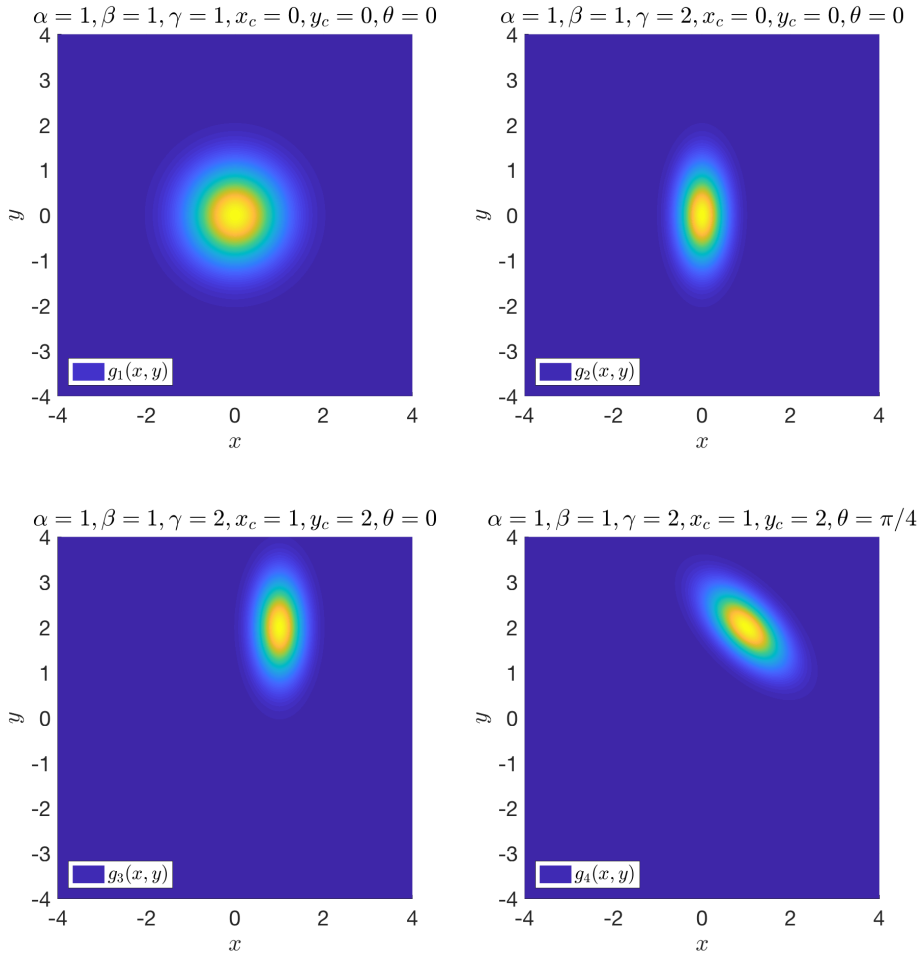


Figure 8: Contour plot of all four Gaussian functions.

The Gaussian functions in two dimensions retain the properties from one dimension as discussed in Chapter 4. Hence, we are now ready to provide the Gabor transform in two dimensions:

Definition 6.4 (Gabor transform in $L^2(\mathbb{R}^2)$)**.** *Let $f \in L^2(\mathbb{R}^2)$ and g be a Gaussian function. The Gabor transform of f with respect to g is defined by*

$$\mathcal{G}_g\{f(\boldsymbol{x})\} = \tilde{f}_g(\boldsymbol{x}, \boldsymbol{\omega}) = \int_{\mathbb{R}^2} f(\boldsymbol{\tau}) g(\boldsymbol{\tau} - \boldsymbol{x}) e^{-i\boldsymbol{\omega}\cdot\boldsymbol{\tau}} d\boldsymbol{\tau},$$

where $\boldsymbol{x}, \boldsymbol{\tau}, \boldsymbol{\omega} \in \mathbb{R}^2$.

Remark 6.5. *(i) The resulting Gabor transform $\tilde{f}_g$ is a mapping $\mathbb{R}^4 \to \mathbb{R}$. Hence, its graph is five-dimensional.*

(ii) In real-world applications, it is impractical to use the form from the definition. Instead, we may assume $g(\boldsymbol{\tau}) = g(-\boldsymbol{\tau})$ for all $\boldsymbol{\tau} \in \mathbb{R}^2$ and fix a frequency $\boldsymbol{\omega} \in \mathbb{R}^2$. We then obtain

$$\begin{aligned}\tilde{f}_g(\boldsymbol{x}, \boldsymbol{\omega}) &= \int_{\mathbb{R}^2} f(\boldsymbol{\tau}) g(\boldsymbol{\tau} - \boldsymbol{x}) e^{-i\boldsymbol{\omega}\cdot\boldsymbol{\tau}} d\boldsymbol{\tau} \\ &= e^{-i\boldsymbol{\omega}\cdot\boldsymbol{x}} \int_{\mathbb{R}^2} f(\boldsymbol{\tau}) g(\boldsymbol{\tau} - \boldsymbol{x}) e^{-i\boldsymbol{\omega}\cdot(\boldsymbol{\tau}-\boldsymbol{x})} d\boldsymbol{\tau} \\ &= e^{-i\boldsymbol{\omega}\cdot\boldsymbol{x}} \int_{\mathbb{R}^2} f(\boldsymbol{\tau}) g(\boldsymbol{x} - \boldsymbol{\tau}) e^{i\boldsymbol{\omega}\cdot(\boldsymbol{x}-\boldsymbol{\tau})} d\boldsymbol{\tau} \\ &= e^{-i\boldsymbol{\omega}\cdot\boldsymbol{x}} (f * g_{\boldsymbol{\omega}})(\boldsymbol{x})\end{aligned}$$

for $g_{\boldsymbol{\omega}}(\boldsymbol{x}) = g(\boldsymbol{x}) e^{i\boldsymbol{\omega}\cdot\boldsymbol{x}}$.

As seen in one dimension, we may define the Gabor filter that naturally appears in the Gabor transform:

Definition 6.6. *Let g be a Gaussian function and let $\boldsymbol{\omega} \in \mathbb{R}^2$. The function $g_{\boldsymbol{\omega}}$ defined by*

$$g_{\boldsymbol{\omega}}(\boldsymbol{x}) = g(\boldsymbol{x}) e^{i\boldsymbol{\omega}\cdot\boldsymbol{x}}$$

for all $\boldsymbol{x} \in \mathbb{R}^2$ is called Gabor filter.

It is indeed a common approach to calculate the two-dimensional Gabor transform of a function by the method presented in the remark above. From Theorem 5.3, we deduce that the appearing convolution may be evaluated by

$$(f * g_{\boldsymbol{\omega}})(\boldsymbol{x}) = \mathcal{F}^{-1}\{\, \mathcal{F}\{f(\boldsymbol{x})\}\, \mathcal{F}\{g_{\boldsymbol{\omega}}(\boldsymbol{x})\}\, \}(\boldsymbol{x})$$

for all $\boldsymbol{x} \in \mathbb{R}^2$. It is therefore of interest to look at the Fourier transform of the Gabor filter.

Theorem 6.7. *Let $\boldsymbol{\omega} = (\omega, \nu) \in \mathbb{R}^2$ and let g be a Gaussian function defined by*

$$g(x, y) = \alpha e^{-\beta^2 x^2 - \gamma^2 y^2}, \quad (x, y) \in \mathbb{R}^2,$$

where $\alpha, \beta, \gamma > 0$ are real numbers. Additionally, fix $\boldsymbol{\omega} = (\omega, \nu) \in \mathbb{R}^2$. Then, the Gabor filter

$$g_{\boldsymbol{\omega}}(x, y) = \alpha e^{-\beta^2 x^2 - \gamma^2 y^2} e^{i(\omega x + \nu y)}, \quad (x, y) \in \mathbb{R}^2$$

has the Fourier transform

$$\hat{g}_{\boldsymbol{\omega}}(u, v) = \pi \tfrac{\alpha}{\beta\gamma} \exp\left(-\tfrac{(u-\omega)^2}{4\beta^2} - \tfrac{(v-\nu)^2}{4\gamma^2} \right), \quad (u, v) \in \mathbb{R}^2.$$

Proof. (i) We first calculate the Fourier transform of the one-dimensional Gabor filter $g_\omega(x) = e^{-\beta^2 x^2} e^{i\omega x}$. We obtain

$$\begin{aligned}
\hat{g}_\omega(u) &= \int_{-\infty}^{\infty} e^{-\beta^2 x^2} e^{i\omega x} e^{-iux} dx \\
&= \int_{-\infty}^{\infty} \exp\left(-\beta^2 \left[x + \tfrac{\omega-u}{2\beta^2} i\right]^2 - \tfrac{(\omega-u)^2}{4\beta^2}\right) dx \\
&= \exp\left(-\tfrac{(\omega-u)^2}{4\beta^2}\right) \int_{-\infty}^{\infty} \exp\left(-\beta^2 \left[x + \tfrac{\omega-u}{2\beta^2} i\right]^2\right) dx \\
&= \exp\left(-\tfrac{(\omega-u)^2}{4\beta^2}\right) \int_{-\infty}^{\infty} e^{-\beta^2 s^2} dx \\
&= \exp\left(-\tfrac{(u-\omega)^2}{4\beta^2}\right) \tfrac{\sqrt{\pi}}{\beta}
\end{aligned}$$

for all $u \in \mathbb{R}$.

(ii) Now, let $g_{\boldsymbol{\omega}}$ be a Gabor filter as defined in the statement of the theorem. We deduce from the first item and Fubini's theorem that

$$\begin{aligned}
\hat{g}_{\boldsymbol{\omega}}(u,v) &= \int_{-\infty}^{\infty}\int_{-\infty}^{\infty} \alpha e^{-\beta^2 x^2 - \gamma^2 y^2} e^{i(\omega x + \nu y)} e^{-i(ux+vy)} dx dy \\
\text{(Fubini)} &= \alpha \left[\int_{-\infty}^{\infty} e^{-\beta^2 x^2} e^{i\omega x} e^{-iux} dx\right] \left[\int_{-\infty}^{\infty} e^{-\gamma^2 y^2} e^{i\nu y} e^{-ivy} dy\right] \\
\text{(item (i))} &= \alpha \left[\tfrac{\sqrt{\pi}}{\beta} \exp\left(-\tfrac{(u-\omega)^2}{4\beta^2}\right)\right] \left[\tfrac{\sqrt{\pi}}{\gamma} \exp\left(-\tfrac{(v-\nu)^2}{4\gamma^2}\right)\right] \\
&= \pi \tfrac{\alpha}{\beta\gamma} \exp\left(-\tfrac{(u-\omega)^2}{4\beta^2} - \tfrac{(v-\nu)^2}{4\gamma^2}\right)
\end{aligned}$$

for all $(u,v) \in \mathbb{R}^2$. □

Remark 6.8. *We have chosen $x_c = y_c = \theta = 0$ in view of Definition 6.1. Using the result from the theorem, it is not hard to generalize the statement for arbitrary x_c, y_c and θ.*

6.3 Discrete Gabor transform

In order to derive the discrete version of the Gabor transform in two dimensions, we may use the same discretization as already introduced in previous chapters. Since we would like to keep the formulas simple, we once again use the same amount of samples for both original and frequency domain for each dimension. If we say the variables $(x,y) \in \mathbb{R}^2$ describe the space domain and $(\omega,\nu) \in \mathbb{R}^2$ the frequency domain, we shall have n_1 sample points in direction of x and ω, while we shall have

n_2 sample points in direction of y and ν. Details about the discretization may be found in Chapter 5.3.

The discrete Gabor transform may be derived in the same way as in the one-dimensional case in Chapter 4.3. Again, we have to note that the discrete Gaussian function $g : \mathbb{Z}_{n_1 \times n_2} \to \mathbb{R}$ is a simple evaluation of the continuous Gaussian function implementing (n_1, n_2)-periodicity:

Definition 6.9 (Discrete Gaussian function)**.** *A discrete Gaussian $g : \mathbb{Z}_{n_1 \times n_2} \to \mathbb{R}$ is a function of the form*

$$g(k_1, k_2) = \alpha e^{-\beta^2 k_1'^2 - \gamma^2 k_2'^2}, \quad (k_1, k_2) \in \mathbb{Z}_{n_1 \times n_2}$$

with

$$\begin{bmatrix} k_1' \\ k_2' \end{bmatrix} = \begin{bmatrix} \cos(\theta) & \sin(\theta) \\ -\sin(\theta) & \cos(\theta) \end{bmatrix} \begin{bmatrix} k_1 - x_c \\ k_2 - y_c \end{bmatrix}$$

such that $\alpha, \beta, \gamma > 0$, $(x_c, y_c) \in \mathbb{Z}_{n_1 \times n_2}$ and $\theta \in [0, 2\pi[$. We say that g is centered, if furthermore $x_c = \frac{n_1}{2}, y_c = \frac{n_2}{2}$.

Remark 6.10. *If $g : \mathbb{Z}_{n_1 \times n_2} \to \mathbb{R}$ is a centered discrete Gaussian function, then*

$$g(k_1, k_2) = g(n_1 - k_1, n_2 - k_2) = g(-k_1, -k_2)$$

for all $(k_1, k_2) \in \mathbb{Z}_{n_1 \times n_2}$.

Using the derivation and remark from above, we may obtain the desired definition:

Definition 6.11 (Discrete Gabor transform)**.** *Let $f : \mathbb{Z}_{n_1 \times n_2} \to \mathbb{C}$ be any function and let $g : \mathbb{Z}_{n_1 \times n_2} \to \mathbb{R}$ be a centered discrete Gaussian function. We define the discrete Gabor transform by*

$$\begin{aligned} \mathcal{G}_{d,g}\{f\}(j_1, j_2, m_1, m_2) &= \tilde{f}_g(j_1, j_2, m_1, m_2) \\ &= \sum_{k_1=0}^{n_1-1} \sum_{k_2=0}^{n_2-1} f(k_1, k_2) g(k_1 - j_1, k_2 - j_2) w_n^{m_1 k_1 + m_2 k_2} \end{aligned}$$

for $(j_1, j_2), (m_1, m_2) \in \mathbb{Z}_{n_1 \times n_2}$.

Remark 6.12. *Fix the frequency variable $(m_1, m_2) \in \mathbb{Z}_{n_1 \times n_2}$. Then,*

$$\begin{aligned}
\tilde{f}_g(j_1, j_2, m_1, m_2) &= \sum_{k_1=0}^{n_1-1} \sum_{k_2=0}^{n_2-1} f(k_1, k_2) g(k_1 - j_1, k_2 - j_2) w_n^{m_1 k_1 + m_2 k_2} \\
&= w_n^{m_1 j_1 + m_2 j_2} \sum_{k_1=0}^{n_1-1} \sum_{k_2=0}^{n_2-1} f(k_1, k_2) g(k_1 - j_1, k_2 - j_2) w_n^{m_1(k_1 - j_1) + m_2(k_2 - j_2)} \\
&= w_n^{m_1 j_1 + m_2 j_2} \sum_{k_1=0}^{n_1-1} \sum_{k_2=0}^{n_2-1} f(k_1, k_2) g(j_1 - k_1, j_2 - k_2) w_n^{-m_1(j_1 - k_1) - m_2(j_2 - k_2)} \\
&= w_n^{m_1 j_1 + m_2 j_2} (f * g_{m_1, m_2})
\end{aligned}$$

for $g_{m_1,m_2}(j_1, j_2) = g(j_1, j_2) w_n^{-m_1 j_1 - m_2 j_2}$ and for all $(j_1, j_2) \in \mathbb{Z}_{n_1 \times n_2}$.

We may extract the two-dimensional Gabor filter from the remark above:

Definition 6.13 (Discrete Gabor filter)**.** *Let $g : \mathbb{Z}_{n_1 \times n_2} \to \mathbb{R}$ be a centered discrete Gaussian and fix $(m_1, m_2) \in \mathbb{Z}_{n_1 \times n_2}$. Then, we define the discrete Gabor filter by*

$$g_{m_1,m_2}(k_1, k_2) = g(k_1, k_2) w_n^{-m_1 k_1 - m_2 k_2}$$

for all $(k_1, k_2) \in \mathbb{Z}_{n_1 \times n_2}$.

Even with modern computers in 2018, calculating the complete discrete Gabor transform would be slow due to the complexity of $\mathcal{O}(n^4 \log_2(n))$. If we fix a frequency $(m_1, m_2) \in \mathbb{Z}_{n_1 \times n_2}$, we may break down the complexity to $\mathcal{O}(n^2 \log_2(n))$. For this reason, most authors do not introduce the Gabor transform in two dimensions; they directly work with Gabor filters instead.

6.4 Semi-discrete Gabor filter

While the discrete Gabor filter from the last section descends directly from the discrete Gabor transform, we will not use it in image analysis. Looking at Definition 6.13, we observe that the discretization of the frequency is not necessary for calculating the Gabor transform. The goal of this section is to derive the semi-discrete Gabor filter and then modify it to a more practical version for use in image analysis.

Therefore, let us now fix a frequency $\boldsymbol{\omega} = (\omega, \nu) \in \mathbb{R}^2$ and go back to the continuous Gabor transform. We remember from Remark 6.5 that

$$\tilde{f}_g(\boldsymbol{x}, \boldsymbol{\omega}) = e^{-i\boldsymbol{\omega} \cdot \boldsymbol{x}} (f * g_{\boldsymbol{\omega}})(\boldsymbol{x}), \quad \boldsymbol{x} \in \mathbb{R}^2,$$

with $g_{\boldsymbol{\omega}}(\boldsymbol{x}) = g(\boldsymbol{x})e^{i\boldsymbol{\omega}\cdot\boldsymbol{x}}$. Discretizing only the space domain in the same way as in the previous section, we may derive the semi-discrete version

$$\tilde{f}_g(j_1, j_2, \omega, \nu) = e^{-i(\omega j_1 + \nu j_2)}(f * g_{\boldsymbol{\omega}})(j_1, j_2), \quad (j_1, j_2, \omega, \nu) \in \mathbb{Z}_{n_1 \times n_2} \times \mathbb{R}^2,$$

where $g_{\boldsymbol{\omega}}(k_1, k_2) = g_{\omega,\nu}(k_1, k_2) = g(k_1, k_2)e^{i(\omega k_1 + \nu k_2)}$.

We always assumed g to be a centered discrete Gaussian function. Thus, it makes sense to center the sinusoidal part of $g_{\boldsymbol{\omega}}$ as well. We may achieve this by the following adjustment that allows for a translation by $(x_c, y_c) \in \mathbb{Z}_{n_1 \times n_2}$—the same translation that we used for the discrete Gaussian function in Definition 6.9:

$$\begin{aligned}(f * g_{\boldsymbol{\omega}})(j_1, j_2) &= \sum_{k_1=0}^{n-1}\sum_{k_2=0}^{n-1} f(k_1, k_2)g(k_1 - j_1, k_2 - j_1)e^{-i[\omega(k_1 - j_1) + \nu(k_2 - j_2)]} \\ &= e^{-i(\omega x_c + \nu y_c)} \sum_{k_1=0}^{n-1}\sum_{k_2=0}^{n-1} f(k_1, k_2)g(k_1 - j_1, k_2 - j_1)e^{-i[\omega(k_1 - x_c - j_1) + \nu(k_2 - y_c - j_2)]} \\ &= e^{-i(\omega x_c + \nu y_c)}(f * g_{\boldsymbol{\omega},c})(j_1, j_2)\end{aligned}$$

with $g_{\boldsymbol{\omega},c}(k_1, k_2) = g_{\omega,\nu,c}(k_1, k_2) = g(k_1, k_2)e^{i[\omega(k_1 - x_c) + \nu(k_2 - y_c)]}$. Putting the last two results together, we obtain

$$\tilde{f}_g(j_1, j_2, \omega, \nu) = e^{-i[\omega(j_1 + x_c) + \nu(j_2 + y_c)]}(f * g_{\boldsymbol{\omega},c})(j_1, j_2).$$

In real-world applications, we will not care about the prepending factor and just calculate the convolution $f * g_{\boldsymbol{\omega},c}$. We henceforth call the function $g_{\boldsymbol{\omega},c}$ semi-discrete Gabor filter:

Definition 6.14 (Semi-discrete Gabor filter)**.** *Let $g : \mathbb{Z}_{n_1 \times n_2} \to \mathbb{R}$ be a discrete Gaussian and fix $\boldsymbol{\omega} = (\omega, \nu) \in \mathbb{R}^2$. Then, we define the semi-discrete Gabor filter by*

$$\begin{aligned}g_{\boldsymbol{\omega},c}(k_1, k_2) &= g_{\omega,\nu,c}(k_1, k_2) \\ &= g(k_1, k_2)e^{i[\omega(k_1 - x_c) + \nu(k_2 - y_c)]}\end{aligned}$$

for all $(k_1, k_2) \in \mathbb{Z}_{n_1 \times n_2}$. If furthermore $x_c = \frac{n_1}{2}$ and $y_c = \frac{n_2}{2}$, we say that the Gabor filter is centered.

Remark 6.15. *Including the parameters from the Gaussian function, the semi-discrete Gabor filter reads*

$$g_{\omega,\nu,c}(k_1, k_2) = \alpha e^{-\beta^2 k_1'^2 - \gamma^2 k_2'^2} e^{i[\omega(k_1 - x_c) + \nu(k_2 - y_c)]},$$

where

$$\begin{bmatrix} k_1' \\ k_2' \end{bmatrix} = \begin{bmatrix} \cos(\theta) & \sin(\theta) \\ -\sin(\theta) & \cos(\theta) \end{bmatrix} \begin{bmatrix} k_1 - x_c \\ k_2 - y_c \end{bmatrix}.$$

This form of a semi-discrete Gabor filter is widely used in image analysis. Interestingly, authors like John Daugman (cf. [Dau85]) suggest that simple cells in the visual cortex of mammalian brains can be modeled by Gabor filters. This assumption led to many implementations of the Gabor filters: As of today, they have been used in image analysis, image compression, object recognition, medical diagnostics, and many more fields.

However, most authors use a parametrization different from the one given in Definition 6.14. Based on findings in biological experiments, Nikolay Petkov (cf. [Pet95]) suggests an alternative form of Gabor filters which can be directly derived from our definition: If we set

$$\alpha = 1, \quad \beta = \frac{1}{\sqrt{2}\,\sigma}, \quad \gamma = \frac{\xi}{\sqrt{2}\,\sigma}, \quad \omega = \frac{2\pi\cos(\theta)}{\lambda}, \quad \nu = \frac{2\pi\sin(\theta)}{\lambda},$$

for $\sigma, \xi, \lambda > 0$ and $\theta \in [0, 2\pi[$, we obtain

$$g_{\omega,\nu,c}(k_1, k_2) = \exp\left(-\frac{k_1'^2 + \xi^2 k_2'^2}{2\sigma^2}\right) \exp\left(2\pi i \frac{k_1'}{\lambda}\right).$$

We are going to see in the next section that Gabor filters help to extract image features which are aligned orthogonally to the direction (angle θ) of the Gaussian function. The role of θ in the Gaussian function g of a Gabor filter is thus fundamental in discovering features. It is therefore common to use an array of Gabor filters—a Gabor filter bank—to extract features from multiple directions.

Definition 6.16 (Gabor filter bank)**.** *A Gabor filter bank is a finite collection of arbitrary Gabor filters.*

6.5 Implementation of the discrete Gabor transform

To begin the last section of the Gabor transform in two dimensions, we shall define a shorthand notation for the convolution of an input image and a semi-discrete Gabor filter:

Definition 6.17. *Let $f : \mathbb{Z}_{n_1 \times n_2} \to \mathbb{R}$ represent image color data and let $g_{\boldsymbol{\omega},c} : \mathbb{Z}_{n_1 \times n_2} \to \mathbb{C}$ be a semi-discrete Gabor filter. We define the shorthand notation*

$$f^*(j_1, j_2) = f^*_{g_{\boldsymbol{\omega},c}}(j_1, j_2) = (f * g_{\boldsymbol{\omega},c})(j_1, j_2)$$

for all $(j_1, j_2) \in \mathbb{Z}_{n_1 \times n_2}$.

For this whole section, we assume $n = n_1 = n_2 = 2^s$ for an $s \in \mathbb{N}$. Thus, we define the input image to be a function $f : \mathbb{Z}_n^2 \to \mathbb{C}$ and a semi-discrete Gabor filter to be a function $g_{\boldsymbol{\omega},c} : \mathbb{Z}_n^2 \to \mathbb{C}$. The resulting convolution is then a function $f^* : \mathbb{Z}_n^2 \to \mathbb{C}$.

Below you can see a possible MATLAB implementation of the two-dimensional semi-discrete Gabor filter. The code returns a matrix that contains the values of the filter function in relation to a meshgrid of x and y values.

Listing 9: Creation of a Gabor filter.

```
function gwc = gaborfilter2d(a, b, c, xc, yc, theta, omega, nu, x
    , y)

    gauss = a*exp(...
        -b^2*(+(x-xc)*cos(theta)+(y-yc)*sin(theta)).^2 ...
        -c^2*(-(x-xc)*sin(theta)+(y-yc)*cos(theta)).^2 ...
    );
    sinusoid = exp(1i*omega*(x-xc) + 1i*nu*(y-yc));

    gwc = gauss .* sinusoid;

end
```

Thanks to Theorem 5.6, we know that a discrete convolution may be conveniently calculated using the fast Fourier transform. The MATLAB code below uses previously defined functions of the (inverse) fast Fourier transform in Listing 7 and Listing 8. It returns a matrix with the values of the convolution of the image and the filter.

Listing 10: Discrete convolution of an image with a Gabor filter in two dimensions.

```
function fStar = fastgaborconvolution2d(f, gw)
    fHat = fastfouriertransform2d(f);
    gwHat = fastfouriertransform2d(gw);
    fStar = inversefastfouriertransform2d(fHat .* gwHat);
end
```

More details may be found in Listing 12 for all to Gabor analysis related codes.

Remark 6.18. *(i) It is common in image analysis to normalize a filter such that all positive and negative values sum up to zero. Thus, we would have to slightly adjust the function in Listing 9. We provide both the standard and normalized version of the filter in Listing 12.*

(ii) Listing 10 uses three functions with a complexity of $\mathcal{O}(n^2 \log_2(n))$ each. As seen in Theorem 6.7, the Gabor filter itself may be analytically transferred to the Fourier space—we could therefore reduce the implementation to two calls of complexity $\mathcal{O}(n^2 \log_2(n))$.

To finish this thesis, we would like to discuss the effect of Gabor filters on three input grayscale images represented by $f : \mathbb{Z}_n^2 \to \mathbb{Z}_{256}$. Thus, the images all have a width and a height of n pixels. Throughout this section, we use the common parametrization $(\xi, \sigma, \lambda, \theta) \in \mathbb{R}^4$ so that the centered semi-discrete Gabor filter reads

$$g_{\omega,\nu,c}(k_1, k_2) = \exp\left(-\frac{k_1'^2 + \xi^2 k_2'^2}{2\sigma^2}\right) \exp\left(2\pi i \frac{k_1'}{\lambda}\right)$$

with

$$\begin{bmatrix} k_1' \\ k_2' \end{bmatrix} = \begin{bmatrix} \cos(\theta) & \sin(\theta) \\ -\sin(\theta) & \cos(\theta) \end{bmatrix} \begin{bmatrix} k_1 - \frac{n}{2} \\ k_2 - \frac{n}{2} \end{bmatrix}.$$

for all $(k_1, k_2) \in \mathbb{Z}_n^2$.

For each example presented below, we shall use a filter bank containing filters with fixed $(\xi, \sigma, \lambda) \in \mathbb{R}^3$ and $\theta = \frac{k\pi}{4}$ for $k = 0, 1, 2, 3$. As in the one-dimensional case, we have to shift the resulting image by $-\frac{n}{2}$ in each dimension in order to allow comparability to the input image. Every example contains the following eight images:

(1) The input grayscale image f.

(2) The contour plot of the real (cosine) part of the Gabor filter with $\theta = 0$.

(3) The contour plot of the imaginary (sine) part of the Gabor filter with $\theta = 0$.

(4) The convolution f^* of the input image with the Gabor filter of angle $\theta = 0$.

(5) The convolution f^* of the input image with the Gabor filter of angle $\theta = \frac{\pi}{4}$.

(6) The convolution f^* of the input image with the Gabor filter of angle $\theta = \frac{\pi}{2}$.

(7) The convolution f^* of the input image with the Gabor filter of angle $\theta = \frac{3\pi}{4}$.

(8) The pointwise sum of all four convolutions in the images (4) to (7).

Example 6.19. *Set $\xi = 0.5$, $\sigma = 1$, $\lambda = 2$, and $\theta = \frac{k\pi}{4}$ for $k = 0, 1, 2, 3$. As first example, we choose a synthetic grayscale image with different shapes and colors.*

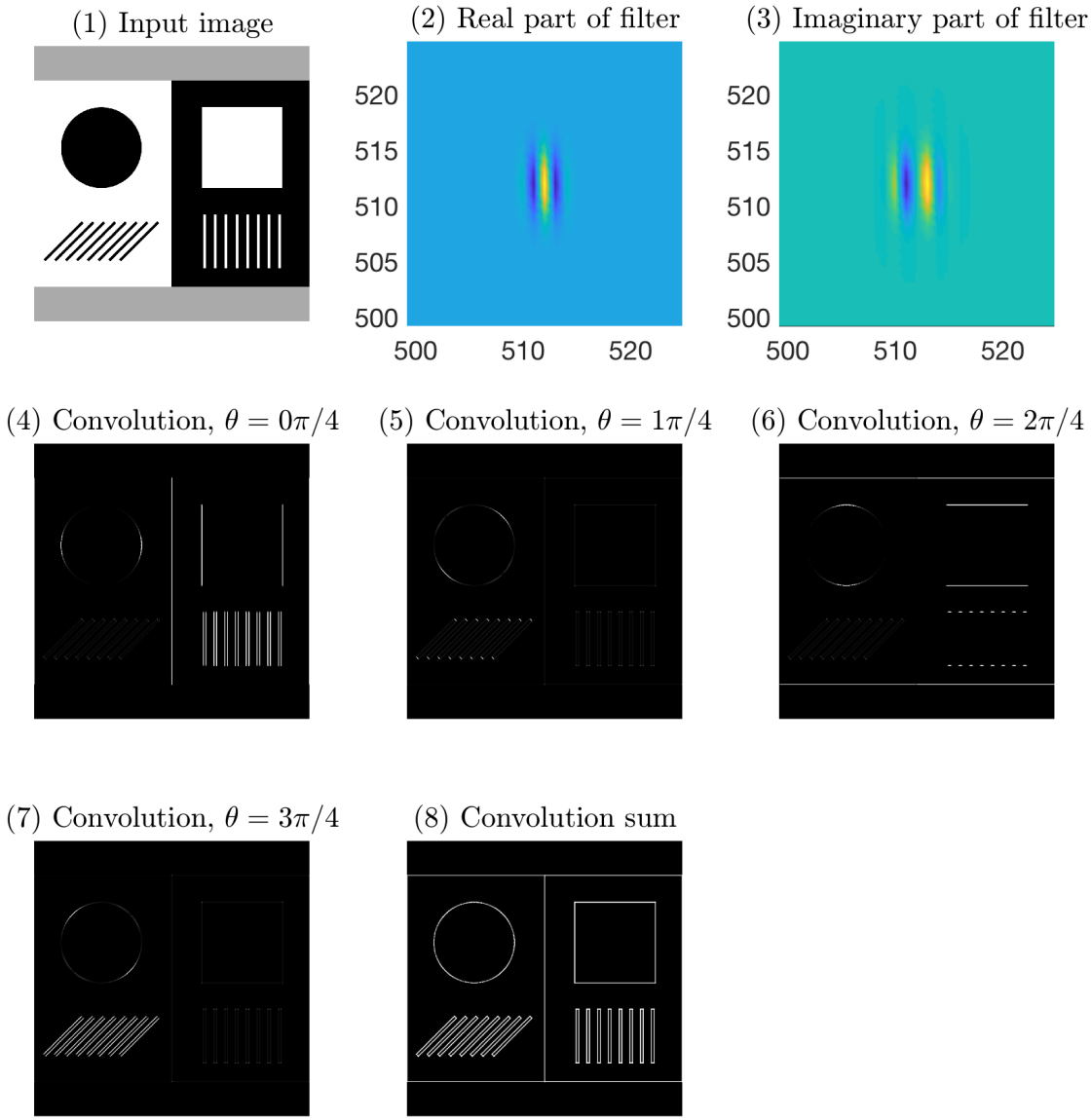


Figure 9: Input image, Gabor filter and convolutions.

Looking at the sum of all convolutions in image (8)*, we see that all edges of the shapes and color transitions have been captured from the input image* (1)*. Each*

filter detects those image features that are orthogonal to the direction of its sinusoid. We note the following for images (4) *to* (7)*:*

(i) $\theta = 0$*: Vertical edges, lines and color transitions are well detected. Due to the periodicity of our data, the left border of the white area and the right border of the black area are also apparent. The circle shows some glowing at its edges at the left and right side. The rectangle and all vertical lines reveal their left and right borders.*

(ii) $\theta = \frac{\pi}{4}$*: Edges of all shapes are hardly visible. Only the circle shows slight glowing at the points that are nearly orthogonal to the filter. Interestingly, the endpoints of the diagonal lines are glowing because their ends indicate a change of color as well.*

(iii) $\theta = \frac{\pi}{2}$*: The transitions of colors from gray to black/white are well visible. The upper and lower borders of the rectangle and circle are apparent as well. This time, the upper and lower end of the vertical lines are glowing.*

(iv) $\theta = \frac{3\pi}{4}$*: The diagonal lines in the lower left are best visible because they are aligned orthogonally to the filter's direction. The circle again is glowing at the points that are nearly orthogonal to the filter.*

Example 6.20. *Let* $\xi = 0.5$*,* $\sigma = 4$ *and* $\lambda = 1$*, and* $\theta = \frac{k\pi}{4}$ *for* $k = 0, 1, 2, 3$*. The input image shows a domestic cat in a garden.*

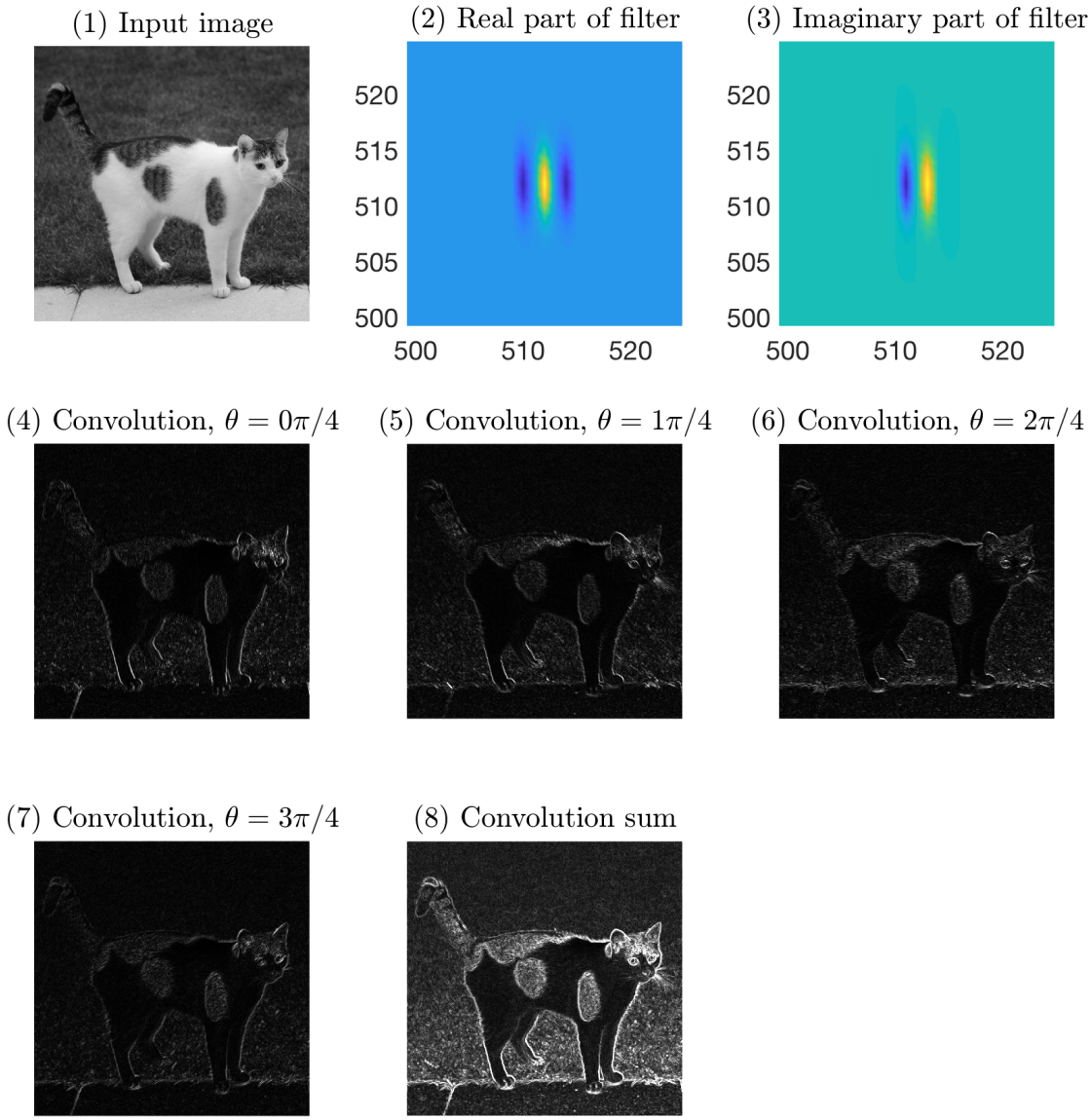


Figure 10: Input image, Gabor filter and convolutions.

The sum of the convolution manages to extract the silhouette of the cat from the background. We note that patterns in the fur are easily recognized—similar to

the shapes in the last example. The grass in the lower part of the image is sharp enough to be detected by the filters, while the upper part vanishes due to the bokeh effect. Looking closer at the convolution for each angle, we note:

(1) $\theta = 0$*: The joint of the floor is well visible because it is nearly orthogonal to the filter. The same applies to the four legs of the cat and its toes.*

(2) $\theta = \frac{\pi}{4}$*: It is interesting to see that even the small hairs at the cat's chest and the whiskers on the right side are very bright. Strong transitions like the top of the left ear or the hind legs are also glowing more than the rest.*

(3) $\theta = \frac{\pi}{2}$*: The joint of the floor is nearly invisible now, while the transition from grass to floor is now well visible.*

(4) $\theta = \frac{3\pi}{4}$*: Only ears and eyes are outstanding because most edges in the image are not orthogonal to this angle.*

Example 6.21. *Let* $\xi = 0.5$*,* $\sigma = 4$ *and* $\lambda = 1$*, and* $\theta = \frac{k\pi}{4}$ *for* $k = 0, 1, 2, 3$*. In this example, we are looking at an image with different textures.*

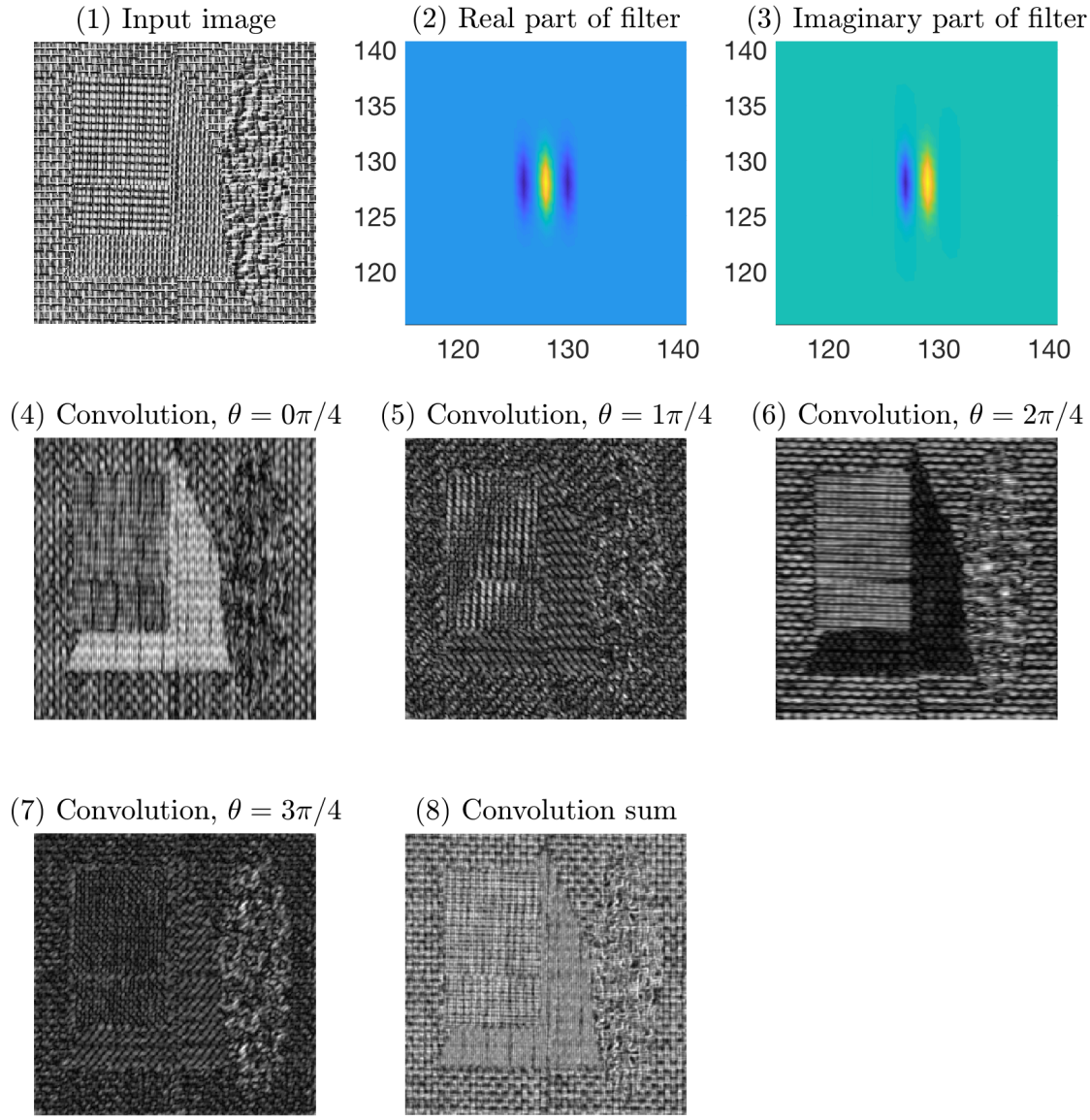


Figure 11: Input image, Gabor filter and convolutions.

This example shows the importance of an intelligent selection of angles. As the textures are mostly aligned along the axes, the convolutions at angles $\theta = \frac{\pi}{4}, \frac{3\pi}{4}$

do not contribute to reveal the shapes. On the other hand, the other convolutions extract the three major shapes from the input image, while they were nearly invisible to the naked eye.

Gabor filters proved to be excellent in detecting discontinuities in images, such as color transitions or edges of objects. It is possible to extract an object from a background that is homogenous or that has a periodic shape. The more random the image is, the more difficult it is to detect a specific shape.

One of the tasks in image analysis is to find appropriate parameters $(\xi, \sigma, \lambda, \theta) \in \mathbb{R}^4$ for the Gabor filters. Based on John Daugman's paper (cf. [Dau85]), many authors dedicated their work to this task. By experiments, Judson Jones and Larry Palmer confirmed that simple cells in the visual cortex of cats can be modeled by Gabor filters. Additionally, they found that the parameters should comply with $\xi \in [0.23, 0.92]$ and $\frac{\sigma}{\lambda} \in [0.4, 0.9]$ (cf. [JP87]).

Another question is how to vary the angle θ in order to find a good balance between computational cost and optimal result. In the paper of Ilonen, Kämäräinen and Kälviäinen, this particular question is discussed (cf. [IKK05]).

Our web application at GitHub pages lets the reader play with all parameters and try different images: `https://andreas-aeschlimann.github.io/gabor/`

7 Additional implementations

7.1 Online web application

We developed an online web application in order to test Gabor filters in two dimensions. It may be found at the following GitHub pages address: `https://andreas-aeschlimann.github.io/gabor/`.

The web application has been developed in TypeScript—a language built on top of JavaScript. In addition to that, we have used a new web standard called WebAssembly in order to speed up the mathematical calculations. WebAssembly allows us to implement time-consuming tasks in C and then compile the code to a format that the browser is able to read. Even though the whole application runs in the browser, experiments have shown that the calculations of the transforms are nearly as fast as with native C.

The source code of the whole application may be found on GitHub as well: `https://github.com/andreas-aeschlimann/gabor`

7.2 MATLAB classes

Using the (discrete) theory from this thesis, we have the necessary tools to implement several MATLAB methods for Fourier and Gabor analysis in one and two dimensions. As of 2018, many of these methods are already part of the standard MATLAB code base. In real-world applications, it might be faster to use these methods instead of our implementations.

The source code of these MATLAB classes is attached below, but may be as well found on GitHub: `https://github.com/andreas-aeschlimann/gabor`.

Fourier class

The Fourier class provides methods for the fast Fourier transform in one and two dimensions.

One can simply call **Fourier.fft**(f) for the fast Fourier transform and **Fourier.ifft**(f) for the inverse fast Fourier transform, where the variable f may be any vector or square matrix.

The method **Fourier.conv**(x, y) calculates the circular convolution of two vectors or square matrices x, y.

Listing 11: Fourier methods.

```
classdef Fourier < handle
    % Contains static methods for discrete Fourier analysis.

    %%%%%%%%%%%%%%%%%
    % Math methods %
    %%%%%%%%%%%%%%%%%

    methods (Static)

        function fHat = fft(f)
            % Calculates the 1d/2d fast Fourier transform of a vector/
            % matrix.
            %
            % Params:
            % - matrix f        any complex input matrix
            %
            % Returns:
            % - matrix fHat     the complex output matrix

            % Determine dimension of input vector
            [n1, n2] = size(f);

            if n1 > 1 && n2 > 1
                fHat = Fourier.fft2(f);
            elseif n1 == 1 || n2 == 1
                fHat = Fourier.fft1(f);
            else
                error('Input y has invalid dimensions.');
            end

        end

        function f = ifft(fHat)
```

```
34              % Calculates the 1d/2d inverse fast Fourier transform of a
35              % vector/matrix.
36              %
37              % Params:
38              % - matrix fHat     any complex input matrix
39              %
40              % Returns:
41              % - matrix f        the complex output matrix
42
43              % Determine dimension of input vector
44              [n1, n2] = size(fHat);
45
46              if n1 > 1 && n2 > 1
47                  f = Fourier.ifft2(fHat);
48              elseif n1 == 1 || n2 == 1
49                  f = Fourier.ifft1(fHat);
50              else
51                  error('Input y has invalid dimensions.');
52              end
53          end
54
55          function xy = conv(x, y)
56              % Calculates the (circular) convolution using the fast Fourier
57              % transform.
58              %
59              % Params:
60              % - matrix x    any complex input matrix
61              % - matrix y    any complex input matrix
62              %
63              % Returns:
64              % - matrix xy   the discrete convolution x*y
65
66              % Determine dimension of input data
67              [n1, n2] = size(x);
68
69              % Show error if necessary
70              if (isequal(size(x), size(y)) == 0)
71                  error("Input vectors/matrices must have same dimensions.");
72              end
73
74              % Calculate convolution depending on dimension
75              if (n1 > 1 && n2 > 1)
76                  xy = Fourier.ifft2(Fourier.fft2(x) .* Fourier.fft2(y));
77              elseif (n1 > 1 && n2 == 1) || (n1 == 1 && n2 > 1)
78                  xy = Fourier.ifft(Fourier.fft1(x) .* Fourier.fft1(y));
79              else
80                  error("Input data is invalid, matrices are of size 0.");
81              end
82
```

```
83          end
84
85      end
86
87      methods (Static, Access = private)
88
89          function fHat = fft1(f)
90              % Calculates the 1d fast Fourier transform of a vector.
91              %
92              % Params:
93              % - vector f        any complex input vector
94              %
95              % Returns:
96              % - vector fHat     the complex output vector
97
98              % Determine dimension of input vector
99              [n1, n2] = size(f);
100             m = min(n1, n2);
101             n = max(n1, n2);
102
103             % Show error if necessary
104             if m ~= 1
105                 error("Input must not be a matrix.");
106             end
107             if n < 2 || rem(log2(n), 1) > 0
108                error("Input vector must be of size 2^m for any positive
                       integer m.");
109             end
110
111             % If n is 2, apply the Fourier matrix
112             if n == 2
113                 fHat = zeros(n1, n2);
114                 fHat(1) = f(1)+f(2);
115                 fHat(2) = f(1)-f(2);
116                 return;
117             end
118
119             % Halve the value of n
120             n = n/2;
121
122             % Calculate c, d
123             c = Fourier.fft(f(1:2:2*n));
124             d = Fourier.fft(f(2:2:2*n));
125
126             % Correct d value
127             for k=1:1:n
128                d(k) = exp(-1i*pi*(k-1)/n)*d(k);
129             end
130
```

```
131             % Set the values and return
132             fHat = zeros(size(f));
133             fHat(1:n) = c+d;
134             fHat(n+1:2*n) = c-d;
135 
136         end
137 
138         function f = ifft1(fHat)
139             % Calculates the 1d inverse fast Fourier transform of a vector.
140             %
141             % Params:
142             % - vector fHat     any complex input vector
143             %
144             % Returns:
145             % - vector f        the complex output vector
146 
147             % Calculate the IFFT through the FFT by this equivalence:
148             % f = 1/n conj(W)*fHat <=> f = 1/n conj(W*conj(fHat))
149 
150             n = length(fHat);
151             fHat = conj(fHat);
152             f = 1/n * conj(Fourier.fft(fHat));
153 
154         end
155 
156         function fHat = fft2(f)
157             % Calculates the 2d fast Fourier transform of a matrix.
158             %
159             % Params:
160             % - matrix f        any complex input matrix
161             %
162             % Returns:
163             % - matrix fHat     the complex output matrix
164 
165             % Determine dimension of input matrix
166             [n1, n2] = size(f);
167             n = n1;
168 
169             % Show error if necessary
170             if abs(n2-n1) > 0
171                 error("Input matrix must be square.");
172             end
173             if n < 2 || rem(log2(n), 1) > 0
174                 error("Input matrix must be of size 2^m for any positive
                        integer m.");
175             end
176 
177             % Initialize zero matrices
178             fHatTemp = zeros(n, n);
```

```
179             fHat = zeros(n, n);
180 
181             % Apply the 1d fast Fourier transform for all rows
182             for k=1:1:n
183                fHatTemp(k,:) = Fourier.fft1(f(k, :).').';
184             end
185 
186             % Apply the 1d fast Fourier transform for all cols
187             for k=1:1:n
188                fHat(:,k) = Fourier.fft1(fHatTemp(:, k));
189             end
190 
191         end
192 
193         function f = ifft2(fHat)
194             % Calculates the 2d inverse fast Fourier transform of a matrix.
195             %
196             % Params:
197             % - matrix fHat     any complex input matrix
198             %
199             % Returns:
200             % - matrix f        the complex output matrix
201 
202             % Calculate the IFFT through the FFT by this equivalence:
203             % f = 1/n^2 conj(W)*fHat <=> y = 1/n^2 conj(W*conj(fHat))
204 
205             n = length(fHat);
206             fHat = conj(fHat);
207             f = 1/n^2 * conj(Fourier.fft2(fHat));
208 
209         end
210 
211     end
212 
213     %%%%%%%%%%%%%%%%
214     % Test methods %
215     %%%%%%%%%%%%%%%%
216 
217     methods (Static)
218 
219         function test()
220             % Tests the functionality of the class and prints the result.
221 
222             % Setup test variables
223             tol = 10^(-9);
224             amount = 5;
225 
226             fprintf('========================================\n\n');
227             fprintf('Starting Fourier class test...\n\n');
```

```
228             fprintf('– Error tolerance: %d\n', tol);
229             fprintf('– Tests per method: %d\n', amount);
230             fprintf('\n');
231 
232             % Test FFT
233             fprintf('————————\nTesting FFT:\n');
234             for k=1:amount
235                 Fourier.fftTest(tol)
236             end
237 
238             % Test IFFT
239             fprintf('————————\nTesting IFFT:\n');
240             for k=1:amount
241                 Fourier.ifftTest(tol)
242             end
243 
244             % Test FFT/IFFT
245             fprintf('————————\nTesting FFT&IFFT:\n');
246             for k=1:amount
247                 Fourier.fftifftTest(tol)
248             end
249 
250             % Test FFT2
251             fprintf('————————\nTesting FFT2:\n');
252             for k=1:amount
253                 Fourier.fft2Test(tol)
254             end
255 
256             % Test IFFT 2
257             fprintf('————————\nTesting IFFT2:\n');
258             for k=1:amount
259                 Fourier.ifft2Test(tol)
260             end
261 
262             % Test FFT2/IFFT2
263             fprintf('————————\nTesting FFT2&IFFT2:\n');
264             for k=1:amount
265                 Fourier.fft2ifft2Test(tol)
266             end
267 
268             % Test circular convolution
269             fprintf('————————\nTesting Convolution:\n');
270             for k=1:amount
271                 Fourier.convTest(tol)
272             end
273 
274             fprintf('————————\nTesting completed with 0 issues.\n');
275 
276         end
```

```
277
278     end
279
280     methods (Static, Access = private)
281
282         function fftTest(tol)
283             y = Fourier.randc(2^16, 1);
284             diff = max(abs(Fourier.fft(y) - fft(y)));
285             Fourier.printTestResult(diff, tol);
286         end
287
288         function ifftTest(tol)
289             y = Fourier.randc(2^16, 1);
290             diff = max(abs(Fourier.ifft(y) - ifft(y)));
291             Fourier.printTestResult(diff, tol);
292         end
293
294         function fftifftTest(tol)
295             y = Fourier.randc(2^16, 1);
296             diff = max(abs(Fourier.fft(Fourier.ifft(y)) - y));
297             Fourier.printTestResult(diff, tol);
298         end
299
300         function fft2Test(tol)
301             y = Fourier.randc(2^8, 2^8);
302             diff = max(max(abs(Fourier.fft(y) - fft2(y))));
303             Fourier.printTestResult(diff, tol);
304         end
305
306         function ifft2Test(tol)
307             y = Fourier.randc(2^8, 2^8);
308             diff = max(max(abs(Fourier.ifft(y) - ifft2(y))));
309             Fourier.printTestResult(diff, tol);
310         end
311
312         function fft2ifft2Test(tol)
313             y = Fourier.randc(2^8, 2^8);
314             diff = max(max(abs(Fourier.fft(Fourier.ifft(y)) - y)));
315             Fourier.printTestResult(diff, tol);
316         end
317
318         function convTest(tol)
319             x = Fourier.randc(2^10, 1);
320             y = Fourier.randc(2^10, 1);
321             diff = max(abs(Fourier.conv(x, y) - cconv(x, y, 2^10)));
322             Fourier.printTestResult(diff, tol);
323         end
324
325         function y = randc(m, n)
```

```
326             y = randi(100, m, n) .* rand(m, n) + 1i * randi(100, m, n) .*
                    rand(m, n);
327         end
328 
329         function printTestResult(diff, tol)
330             if (diff > tol)
331                 fprintf('[ ] Calculation error: %d\n', diff);
332                 error('Error too high or tolerance too low.');
333             else
334                 fprintf('[X] Calculation error: %d\n', diff);
335             end
336         end
337 
338     end
339 
340 end
```

Gabor class

The Gabor class contains functionality for the fast Gabor transform in one and two dimensions.

For a vector t of time samples, **Gabor.gaussian1**(a, b, t_c, t) evaluates the one-dimensional Gaussian in all these points and returns a vector. For the two-dimensional Gaussian we require a meshgrid with matrices x and y, then we may use **Gabor.gaussian2**(a, b, c, x_c, y_c, θ, x, y) in order to get a matrix with corresponding values.

For the one-dimensional Gabor transform, we provide the methods **Gabor.fgt1**(f, g) and **Gabor.ifgt1**($fTilde$, g), where g is a Gaussian vector or matrix generated with the provided functions.

Two-dimensional Gabor filters may be created by the pre-defined methods **Gabor.filter2**(a, b, c, x_c, y_c, θ, ω, nu, x, y) if no normalization is desired and **Gabor.normalizedFilter2**(a, b, c, x_c, y_c, θ, ω, ν, x, y) if the sum of all positive and negative values should be zero. **Gabor.fgc2**(f, g_w) then calculates the convolution of an input matrix f and the previously generated filter g_w.

The method **Gabor.translate**(A, $hShift$, $vShift$) allows the circular shift of any matrix or vector A in horizontal and vertical direction.

Most methods of the Gabor class depend on the Fourier class, we thus need both classes to be in the working directory of MATLAB when using the Gabor class.

Listing 12: Gabor methods.

```
classdef Gabor < handle
    % Contains static methods for discrete Gabor analysis.

    %%%%%%%%%%%%%%%%
    % Math methods %
    %%%%%%%%%%%%%%%%

    methods (Static)

        function g = gaussian1(a, b, tc, t)
            % Calculates a vector from a Gaussian function evaluated at
            % all points in the vector t.
            %
            % The resulting vector is equivalent to the evaluation of
            %   g(t) = a * exp(-b^2 * (t-tc).^2)
            %
```

```
17            % Params:
18            % - number a    max height of the Gaussian
19            % - number b    width/variance of the Gaussian
20            % - number tc   center of the Gaussian
21            % - vector t    any vector containing time samples
22            %
23            % Returns:
24            % - vector g    the output vector
25
26            if a <= 0
27                error('a has to be a real value bigger than 0.');
28            end
29
30            gauss = @(t) a * exp(-b^2 * (t-tc).^2);
31            g = gauss(t);
32
33        end
34
35        function g = gaussian2(a, b, c, xc, yc, theta, x, y)
36            % Calculates a matrix from a Gaussian function evaluated at
37            % all points in the meshgrid matrices x, y.
38            %
39            % The resulting vector is equivalent to
40            %   g(x, y) = a * exp(-b^2*x'.^2 - c^2*y'.^2)
41            % where
42            %   x' = + (x-xc)cos(theta) + (y-yc)sin(theta)
43            %   y' = - (x-xc)sin(theta) + (y-yc)cos(theta)
44            %
45            % Params:
46            % - number a        max height of the Gaussian
47            % - number b        width/variance of the Gaussian in x
48            % - number c        width/variance of the Gaussian in y
49            % - number xc       center of the Gaussian in x
50            % - number yc       center of the Gaussian in y
51            % - number theta    rotation angle of the Gaussian
52            % - vector x        any matrix containing meshgrid samples
53            % - vector y        any matrix containing meshgrid samples
54            %
55            % Returns:
56            % - matrix g        the output matrix
57
58            if a <= 0
59                error('a has to be a real value bigger than 0.');
60            end
61
62            gauss = @(x, y) a * exp(...
63                -b^2 * (+(x-xc)*cos(theta) + (y-yc)*sin(theta)).^2 ...
64                -c^2 * (-(x-xc)*sin(theta) + (y-yc)*cos(theta)).^2 ...
65            );
```

```
66              g = gauss(x, y);
67
68          end
69
70          function fTilde = fgt1(f, g)
71              % Calculates the one-dimensional fast Gabor transform of a
72              % vector.
73              %
74              % The length of the vectors y and g have to coincide.
75              %
76              % The returning matrix is fTilde(j, m), where j is time
77              % and m is frequency.
78              %
79              % Params:
80              % - vector f        any complex input vector
81              % - vector g        evaluated Gaussian function
82              %
83              % Returns:
84              % - matrix fTilde   the complex output matrix
85
86              % Determine dimension of input vectors
87              [n1, n2] = size(f);
88              [n3, n4] = size(g);
89              n = max(size(f));
90
91              % Transpose if y is 1xn vector
92              if n1 == 1
93                  f = f.';
94              end
95
96              % Transpose if g is 1xn vector
97              if n3 == 1
98                  g = g.';
99              end
100
101             % Show error if necessary
102             if max([n1, n2]) ~= max([n3, n4])
103                error("Length of vectors y and g must coincide.");
104             end
105             if norm(g, 2) == 0
106                 error("The Gaussian function g must not be 0.");
107             end
108
109             % Setup matrix
110             fTilde = zeros(n,n);
111
112             % Initial indices
113             tInd = 0:n-1;
114
```

```
115             % Loop through all times
116             for j = 0:n-1
117 
118                 % Calculate the time shift modulo n
119                 tShift = mod(tInd - j, n) + 1;
120 
121                 % Define phi(k) = y(k)*g(k-j)
122                 phi = f .* g(tShift);
123 
124                 % For each moment in time, calculate the FFT
125                 fTilde(j+1, :) = Fourier.fft(phi);
126 
127             end
128 
129         end
130 
131         function f = ifgt1(fTilde, g)
132             % Calculates the one-dimensional inverse fast Gabor transform
133             % of a square matrix.
134             %
135             % The size of fTilde and g have to coincide.
136             %
137             % The returning vector is f(j), where j is time.
138             %
139             % Params:
140             % - matrix fTilde   any complex input square matrix
141             % - vector g        evaluated Gaussian function
142             %
143             % Returns:
144             % - vector y        the complex output vector
145 
146             % Determine dimension of input vectors
147             [n1, n2] = size(fTilde);
148             [n3, n4] = size(g);
149             n = max(size(fTilde));
150 
151             % Transpose if g is 1xn vector
152             if n3 == 1
153                 g = g.';
154             end
155 
156             % Show error if necessary
157             if abs(n1-n2) > 0 || max([n1, n2]) ~= max([n3, n4])
158                error("Length of y and g must coincide.");
159             end
160             if norm(g, 2) == 0
161                 error("The Gaussian function g must not be 0.");
162             end
163 
```

```
164             % Setup resulting vector
165             f = zeros(n, 1);
166 
167             % Initial indices
168             tInd = 0:n-1;
169 
170             % Loop through all time samples (of the integral)
171             for j = 0:n-1
172 
173                 % Calculate the time shift modulo n
174                 tShift = mod(tInd - j, n) + 1;
175 
176                 % Get the frequency information of the current time
177                 phiHat = fTilde(j+1, :).';
178 
179                 % Calculate the IFFT of that vector
180                 phi = Fourier.ifft(phiHat);
181 
182                 % Sum
183                 f = f + g(tShift) .* phi;
184 
185             end
186 
187             % Make sure the resulting vector is normed properly
188             f = 1/norm(g, 2).^2 * f;
189 
190         end
191 
192         function gw = filter2(a, b, c, xc, yc, theta, omega, nu, x, y)
193             % Calculates a matrix with values of a Gabor filter evaluated
194             % at all points in the meshgrid matrices x, y.
195             %
196             % Params:
197             % - number a        max height of the Gaussian
198             % - number b        width/variance of the Gaussian in x
199             % - number c        width/variance of the Gaussian in y
200             % - number xc       center of the Gaussian in x
201             % - number yc       center of the Gaussian in y
202             % - number theta    rotation angle of the Gaussian
203             % - number omega
204             % - number nu
205             % - vector x        any matrix containing meshgrid samples
206             % - vector y        any matrix containing meshgrid samples
207             %
208             % Returns:
209             % - matrix g        the output matrix
210 
211             if a <= 0
212                 error('a has to be a real value bigger than 0.');
```

```
213             end
214
215             g1 = Gabor.gaussian2(a, b, c, xc, yc, theta, x, y);
216             g2 = exp(1i*omega*(x-xc) + 1i*nu*(y-yc));
217             gw = g1 .* g2;
218
219         end
220
221         function gw = normalizedFilter2(a, b, c, x0, y0, theta, w0, w1, x,
                y)
222
223             % Calculate the matrix of the filter without normalization
224             gw = Gabor.filter2(a, b, c, x0, y0, theta, w0, w1, x, y);
225
226             % Normalize both real and imaginary part
227             gwReal = real(gw);
228             gwImag = imag(gw);
229
230             % Find the indices with positive and negative values
231             gwRealPosInd = find(gwReal > 0);
232             gwRealNegInd = find(gwReal < 0);
233             gwImagPosInd = find(gwImag > 0);
234             gwImagNegInd = find(gwImag < 0);
235
236             % Get the averages
237             gwRealPosFact = sum(gwReal(gwRealPosInd));
238             gwRealNegFact = abs(sum(gwReal(gwRealNegInd)));
239             gwImagPosFact = sum(gwImag(gwImagPosInd));
240             gwImagNegFact = abs(sum(gwImag(gwImagNegInd)));
241             sumReal = (gwRealPosFact + gwRealNegFact)/2;
242             sumImag = (gwImagPosFact + gwImagNegFact)/2;
243
244             if (sumReal > 0)
245                 gwRealPosFact = gwRealPosFact / sumReal;
246                 gwRealNegFact = gwRealNegFact / sumReal;
247             end
248             if (sumImag > 0)
249                 gwImagPosFact = gwImagPosFact / sumReal;
250                 gwImagNegFact = gwImagNegFact / sumReal;
251             end
252
253             % Set the new values
254             gwReal(gwRealPosInd) = gwRealNegFact * gwReal(gwRealPosInd);
255             gwReal(gwRealNegInd) = gwRealPosFact * gwReal(gwRealNegInd);
256             gwImag(gwImagPosInd) = gwImagNegFact * gwImag(gwImagPosInd);
257             gwImag(gwImagNegInd) = gwImagPosFact * gwImag(gwImagNegInd);
258
259             % Put back the results
260             gw = gwReal + 1i * gwImag;
```

```
261
262         end
263
264         function fStar = fgc2(f, gw)
265             % Calculates the convolution of an input matrix and a Gabor
266             % filter.
267             %
268             % The size of the matrices y and gw have to coincide.
269             %
270             % Params:
271             % - matrix y        any complex input square matrix
272             % - matrix gw       evaluated Gabor filter function
273             %
274             % Returns:
275             % - matrix yTilde   the complex output matrix
276
277             % Determine dimension of input vectors
278             [n1, n2] = size(f);
279             [n3, n4] = size(gw);
280
281             % Show error if necessary
282             if abs(n1-n2) > 0 || abs(n3-n4) > 0 || abs(n1-n3) > 0
283                error("Matrices y and filter must be square and coincide in
                       size.");
284             end
285
286             fStar = Fourier.conv(f, gw);
287
288         end
289
290         function A = translate(A, hShift, vShift)
291             % Shifts any matrix.
292             %
293             % Params:
294             % - matrix A        any complex input matrix
295             % - number hShift   amount of indices to shift horizontally
296             % - number vShift   amount of indices to shift vertically
297             %
298             % Returns:
299             % - matrix A        the complex output matrix
300
301             A = circshift(A, [vShift, hShift]);
302
303         end
304
305     end
306
307     methods (Static, Access = private)
308     end
```

```
309
310     %%%%%%%%%%%%%%%%
311     % Test methods %
312     %%%%%%%%%%%%%%%%
313
314     methods (Static)
315
316         function test()
317             % Tests the functionality of the class and prints the result.
318
319             % Setup test variables
320             tol = 10^(-9);
321             amount = 5;
322
323             fprintf('========================================\n\n');
324             fprintf('Starting Gabor class test...\n\n');
325             fprintf('- Error tolerance: %d\n', tol);
326             fprintf('- Tests per method: %d\n', amount);
327             fprintf('\n');
328
329             % Test Gaussian1
330             fprintf('----------\nTesting Gaussian1:\n');
331             for k=1:amount
332                 Gabor.gaussian1Test();
333             end
334
335             % Test Gaussian2
336             fprintf('----------\nTesting Gaussian2:\n');
337             for k=1:amount
338                 Gabor.gaussian2Test();
339             end
340
341             % Test FGT
342             fprintf('----------\nTesting FGT:\n');
343             for k=1:amount
344                 Gabor.fgtTest(tol);
345             end
346
347             % Test FGT/IFGT
348             fprintf('----------\nTesting FGT&IFGT:\n');
349             for k=1:amount
350                 Gabor.fgtifgtTest(tol);
351             end
352
353             % Test NormalizedFilter
354             fprintf('----------\nTesting NormalizedFilter2:\n');
355             for k=1:amount
356                 Gabor.normalizedfilter2test(tol);
357             end
```

```
358
359             fprintf('----------\nTesting completed with 0 issues.\n');
360
361         end
362
363     end
364
365     methods (Static, Access = private)
366
367         function gaussian1Test()
368
369             tol = 100000;
370
371             a = Gabor.randr(1, 1);
372             b = Gabor.randr(1, 1);
373             tc = Gabor.randr(1, 1);
374             t = linspace(0, 100, tol);
375
376             g = Gabor.gaussian1(a, b, tc, t);
377
378             [j, ~] = min(g);
379             [l, m] = max(g);
380
381             if (j < 0 || l > a)
382                 fprintf('[ ] Gaussian a=%d, b=%d, tc=%d.\n', a, b, tc);
383                 error('Error with parameter a.');
384             end
385
386             if (abs(t(m)-tc) > 100/tol)
387                 fprintf('[ ] Gaussian a=%d, b=%d, tc=%d.\n', a, b, tc);
388                 error('Error with parameter tc.');
389             end
390
391             fprintf('[X] Gaussian a=%d, b=%d, tc=%d.\n', a, b, tc);
392
393         end
394
395         function gaussian2Test()
396
397             tol = 1000;
398
399             a = Gabor.randr(1, 1);
400             b = 0.5+randi(10);
401             c = 0.5+randi(10);
402             xc = Gabor.randr(1, 1);
403             yc = Gabor.randr(1, 1);
404             theta = Gabor.randr(1, 1);
405             [x, y] = meshgrid(linspace(0, 100, tol+1), linspace(0, 100, tol
                    +1));
```

```
406
407             g = Gabor.gaussian2(a, b, c, xc, yc, theta, x, y);
408
409             j = min(min(g));
410             maximum = max(max(g));
411             [l, m] = find(g == maximum);
412
413             if (j < 0 || maximum > a)
414                 fprintf('[ ] Gaussian a=%d, b=%d, c=%d, xc=%d, yc=%d, theta
                        =%d.\n', a, b, c, xc, yc, theta);
415                 error('Error with parameter a.');
416             end
417
418             if (abs(x(l,m)-xc) > 2*100/tol)
419                 fprintf('[ ] Gaussian a=%d, b=%d, c=%d, xc=%d, yc=%d, theta
                        =%d.\n', a, b, c, xc, yc, theta);
420                 error('Error with parameter xc.');
421             end
422
423             if (abs(y(l,m)-yc) > 2*100/tol)
424                 fprintf('[ ] Gaussian a=%d, b=%d, c=%d, xc=%d, yc=%d, theta
                        =%d.\n', a, b, c, xc, yc, theta);
425                 error('Error with parameter yc.');
426             end
427
428             fprintf('[X] Gaussian a=%d, b=%d, c=%d, xc=%d, yc=%d, theta=%d
                    .\n', a, b, c, xc, yc, theta);
429
430         end
431
432         function fgtTest(tol)
433             t = (1:1:2^8)';
434             y = Gabor.randc(2^8, 1);
435             g = Gabor.gaussian1(1, 0, 0, t);
436             yTilde = Gabor.fgt1(y, g);
437             diff = max(abs(yTilde(1, :).' - Fourier.fft(y)));
438             Gabor.printTestResult(diff, tol);
439         end
440
441         function fgtifgtTest(tol)
442             t = (1:1:2^8)';
443             y = Gabor.randc(2^8, 1);
444             g = Gabor.gaussian1(Gabor.randr(1, 1), Gabor.randr(1, 1), (2^8)
                    /2, t);
445             diff = max(abs(Gabor.ifgt1(Gabor.fgt1(y, g), g) - y));
446             Gabor.printTestResult(diff, tol);
447         end
448
449         function normalizedfilter2test(tol)
```

```
450
451             h = 1000;
452
453             a = Gabor.randr(1, 1);
454             b = Gabor.randr(1, 1);
455             c = Gabor.randr(1, 1);
456             xc = Gabor.randr(1, 1);
457             yc = Gabor.randr(1, 1);
458             theta = Gabor.randr(1, 1);
459             w0 = Gabor.randr(1, 1);
460             w1 = Gabor.randr(1, 1);
461             [x, y] = meshgrid(linspace(0, 100, h), linspace(0, 100, h));
462
463             gwc = Gabor.normalizedFilter2(a, b, c, xc, yc, theta, w0, w1, x
                    , y);
464
465             diff = sum(sum(real(gwc))) + sum(sum(imag(gwc)));
466
467             if (diff > tol)
468                 fprintf('[ ] Normalization error: %d\n', diff);
469                 error('Error too high or tolerance too low.');
470             else
471                 fprintf('[X] Normalization error: %d\n', diff);
472             end
473
474         end
475
476         function y = randr(m, n)
477             y = randi(100, m, n) .* rand(m, n);
478         end
479
480         function y = randc(m, n)
481             y = randi(100, m, n) .* rand(m, n) + 1i * randi(100, m, n) .*
                    rand(m, n);
482         end
483
484         function printTestResult(diff, tol)
485             if (diff > tol)
486                 fprintf('[ ] Calculation error: %d\n', diff);
487                 error('Error too high or tolerance too low.');
488             else
489                 fprintf('[X] Calculation error: %d\n', diff);
490             end
491         end
492
493     end
494
495 end
```

References

[Bra00] Ronald N. Bracewell. *The Fourier Transform and its Applications*. McGraw-Hill Book Co, Singapore, 3rd edition, 2000.

[Chr08] Ole Christensen. *Frames and Bases*. Birkhäuser Boston, Boston, 2008.

[CT65] James W. Cooley and John W. Tukey. An Algorithm for the Machine Calculation of Complex Fourier Series. *Mathematics of Computation*, 19:297–301, 1965.

[Dau85] John G. Daugman. Uncertainty relation for resolution in space, spatial frequency, and orientation optimized by two-dimensional visual cortical filters. *Journal of the Optical Society of America A*, 2(7):1160–1169, July 1985.

[DS15] Lokenath Debnath and Firdous Ahmad Shah. *Wavelet Ttransforms and Their Applications*. Springer Science+Business Media, New York, 2nd edition, 2015.

[EG92] Lawrence C. Evans and Ronald F. Gariepy. *Measure Theory and Fine Properties of Functions*. CRC Press, Boca Raton, 1st edition, 1992.

[Föl11] Otto Föllinger. *Laplace-, Fourier- und z-Transformation*. VDE Verlag GmbH, Berlin, 10th edition, 2011.

[Gab46] Dennis Gabor. Theory of Communication. Part 1: The Analysis of Information. *Journal of the Institution of Electrical Engineers*, 93:429–441, 1946.

[Har17] Helmut Harbrecht. Lecture notes to Einführung in die Numerik, 2017. `http://cm.dmi.unibas.ch/teaching/numerik/skript.pdf`, [2018-06-30].

[IKK05] Jarmo Ilonen, Joni-Kristian Kämäräinen, and Heikki Kälviäinen. Efficient Computation of Gabor Features. Technical report, Lappeenranta University of Technology, 2005.

[JP87] Judson P. Jones and Larry A. Palmer. An Evaluation of the Two-Dimensional Gabor Filter Model of Simple Receptive Fields in Cat Striate Cortex. *Journal of Neurophysiology*, 58(6):1233–1258, December 1987.

[Pet95] Nikolay Petkov. Biologically Motivated Computationally Intensive Approaches to Image Pattern Recognition. *Future Generation Computer Systems*, 11(4-5):451–465, August 1995.