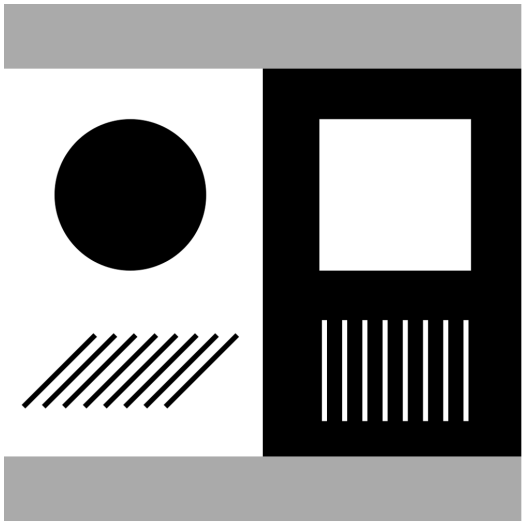
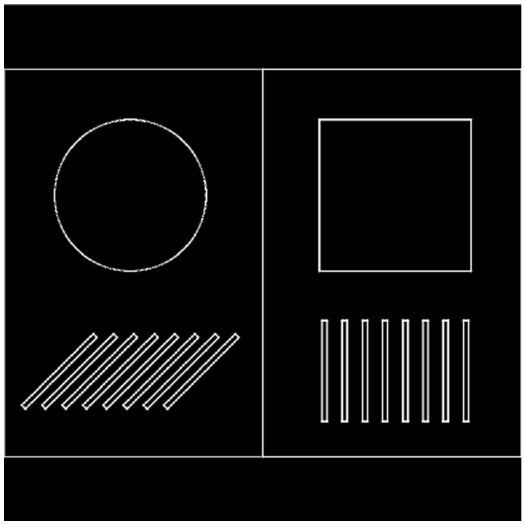
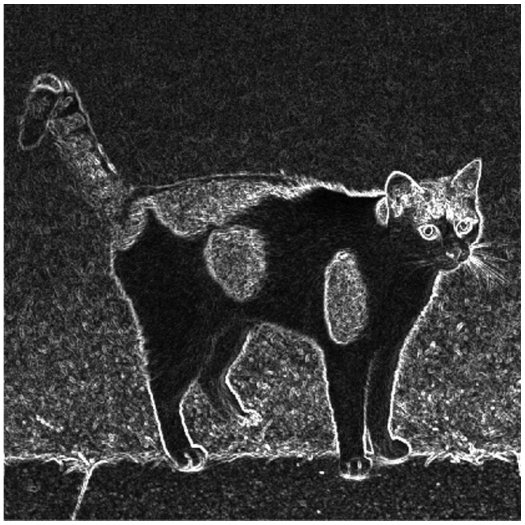
University of Basel

Gabor Transform and Vision

Master's Thesis in Mathematics



Date June 30, 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 writing this thesis.

Special thanks also to my girlfriend and my family
for supporting me in all these years during 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 was 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. To summarize, the Gabor transform is a windowed Fourier transform with a Gaussian window function. Compared 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 widely used for image analysis, image compression, object recognition, medical diagnostics, and in many other fields.

In order to make the reader familiar with the necessary theory, we first present the 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, the discrete Fourier transform is introduced 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 the basis for the Gabor transform shown in Chapter 6. To conclude the two-dimensional Gabor analysis, we are going to use our own implementation and discuss some typical images.

At the end of the thesis, the reader can 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 Gabor filters in two dimensions. This app may be found under `https://andreas-aeschlimann.github.io/gabor/`.

2 The Fourier transform in one dimension

2.1 Motivation

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

For a better understanding, we start with basic definitions needed 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 to be 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

In this thesis, we are going to work with functions defined in $\mathbb{R}^d$ for $d \in \{1, 2\}$. 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 distinguish between a one-dimensional and 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 than $x \in \mathbb{R}$.

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 the $L^2(\mathbb{R}^d)$ norms regularly appear, 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*

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

for almost every $\boldsymbol{x} \in \mathbb{R}^d$.

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

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

(2) We say that $f \in C_0(\mathbb{R}^d)$ if $f : \mathbb{R}^d \to \mathbb{C}$ is a continuous function vanishing 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 $L^p(\mathbb{R}^d)$ may be approximated by a sequence of continuous functions with compact support.

2.3 The 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 $L^1(\mathbb{R})$ for all $\omega \in \mathbb{R}$. 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*

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

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

Remark 2.8. *The resulting function $\hat{f}$ 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*

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

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 $x = t + \frac{\omega}{2\beta^2}i$.

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 $\sqrt{2\pi}$.*

The Fourier transform has many useful properties:

Theorem 2.11 (Basic properties of the Fourier transform)**.** *Let f, g be functions in $L^1(\mathbb{R})$ and let $\hat{f}, \hat{g}$ be their corresponding Fourier transforms with respect to the variable t. Additionally, set the scalar values $a, b \in \mathbb{C}$ and $r \in \mathbb{R}$. Then, the following statements hold for all $\omega \in \mathbb{R}$:*

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

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

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

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

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

Proof. (1) The assertion follows directly from Definition 2.7.

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

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

(3) By applying Definition 2.7, we obtain

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

(4) Applying conjugation multiple times results in

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

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

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

□

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* $\bar{\hat{f}}(w) = \bar{f}(-w)$ *instead.*

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

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

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 fact that $e^{-i\omega t} = -e^{-i\omega t - i\pi} = -e^{-i\omega\left(t+\frac{\pi}{\omega}\right)}$ for all $\omega \neq 0$. 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,

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

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. □

We now have shown the following: 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 will take a look at its inverse.

Definition 2.15 (Inverse Fourier transform)**.** *If $f \in L^1(\mathbb{R})$, then we define the inverse Fourier transform as*

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

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

Remark 2.16. *(i) Similarly, as mentioned prior to 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 2.17 (Uniqueness)**.** *If $f, g \in L^1(\mathbb{R})$ such that $\hat{f} = \hat{g}$, then $f = g$ (in the sense of the L^1 norm).*

Remark 2.18. *A proof may be established 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 the spatial domain and the frequency domain.

Theorem 2.19 (Convolution theorem)**.** *If $f, g \in L^1(\mathbb{R})$, then*

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

for almost every $\omega \in \mathbb{R}$.

Proof. We know from Remark 2.4 that $f * g \in L^1(\mathbb{R})$. We may simply apply Definition 2.7 and use Fubini's theorem (cf. [EG92, Chapter 1.4, Theorem 1]) to deduce

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

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,

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

□

Theorem 2.20 (General modulation)**.** *If $\hat{f}, \hat{g} \in L^1(\mathbb{R})$, then*

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

for almost every $\omega \in \mathbb{R}$.

Proof. We may use Definition 2.15 and once again Fubini's theorem to get

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

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 the properties can be easily extended to $L^1(\mathbb{R}^d)$, 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, the consideration of more general function classes is justified.

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

2.4 The 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 $L^2(\mathbb{R})$. Furthermore, 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 definitions of the Fourier transform in $L^1(\mathbb{R})$ and $L^2(\mathbb{R})$ only coincide almost everywhere for functions being part of both spaces.

Remark 2.21. *(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 $\|\cdot\|$ 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 2.22. *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 because of Definition 2.7.

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

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

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 $a > 0$ such that f vanishes outside $[-a, a]$. Define $b = a/\pi$ and $g(t) = f(bt)$, hence g has its support in Ω.

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:

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

Using this 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. $\square$

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 2.23 (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 2.24. *(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 2.22. 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 in Definition 2.7 is in the equivalence class of the Fourier transform defined in Definition 2.23.

(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 2.22, Definition 2.23, and Remark 2.24, we have proved the following theorem:

Theorem 2.25 (Parseval's identity)**.** *If $f \in L^2(\mathbb{R})$, then*

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

Theorem 2.26 (Fourier transform in $L^2(\mathbb{R})$)**.** *If $f \in L^2(\mathbb{R})$, then*

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

for almost every $\omega \in \mathbb{R}$.

Proof. Let $(f_n)_{n\in\mathbb{N}}$ be 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 2.25, 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 2.27. *Many basic properties of the Fourier transform in $L^1(\mathbb{R})$—as shown in Theorem 2.11—also hold in $L^2(\mathbb{R})$.*

Theorem 2.28. *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) for $m, n \in \mathbb{N}$. Note that we have $f_m,\ g_n \in L^1(\mathbb{R})$ by construction and thus $f_m(y) g_n(x) e^{-ixy} \in L^1(\mathbb{R}^2)$. This allows us to use Fubini's theorem to obtain

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

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 2.29. *If $f \in L^2(\mathbb{R})$ and $g = \bar{\hat{f}}$, then $2\pi f = \bar{\hat{g}}$ (in the sense of the L^2 norm).*

Proof. From the definition of the inner product, Theorem 2.25, and Theorem 2.28, 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 2.30 (Inverse Fourier transform in $L^2(\mathbb{R})$)**.** *Let $f \in L^2(\mathbb{R})$, then the inverse Fourier transform is defined by*

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

for almost every $t \in \mathbb{R}$.

Proof. Let $f \in L^2(\mathbb{R})$ and $g = \bar{\hat{f}}$. It follows directly from this definition that $\bar{g} = \hat{f}$. Using this, Theorem 2.26, and Theorem 2.29 ($2\pi f = \bar{\hat{g}}$), 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 yields the statement of the theorem. □

Theorem 2.31 (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 2.25 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 2.25, we have $\|\hat{f}\|^2 = 2\pi \|f\|^2$ and $\|\hat{g}\|^2 = 2\pi \|g\|^2$. Hence, 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:

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

This is exactly what we intended 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 2.32 (Plancherel's theorem)**.** *For every $f \in L^2(\mathbb{R})$, there exists an $\hat{f} \in L^2(\mathbb{R})$ 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 2.26—the Fourier transform in $L^2(\mathbb{R})$.

(2) See Theorem 2.30—the inverse Fourier transform in $L^2(\mathbb{R})$.

(3) See Definition 2.7, Definition 2.23, and Remark 2.24.

(4) See Theorem 2.31—general Parseval's relation.

(5) See Theorem 2.25—Parseval's identity.

(6) The mapping is a Hilbert space homomorphism by construction in Definition 2.23. 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 $f = \frac{1}{2\pi}\bar{\hat{h}}$. We obtain $h = \bar{\hat{f}}$ by Theorem 2.29, hence $g = \bar{h} = \hat{f}$. This means: For every g, we may find a function $f \in L^2(\mathbb{R})$ such that $\hat{f} = g$. □

We now have established the basics of 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 2.33 (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 every $x \in \mathbb{R}$.

Proof. By Theorem 2.26 and Theorem 2.30, 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 in L^2 is defined almost everywhere, we get the desired result. □

Remark 2.34. *The theorem above immediately implies that*

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

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

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, therefore

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

Fortunately, this change will not invalidate any of the theory that we have seen above, but only affect 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 The discrete Fourier transform in one dimension

3.1 Motivation

In the previous 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 $f(x) = \sin(x)$ and $g(x) = \cos(x)$ are not in $L^p(\mathbb{R})$ for $p \in \{1, 2\}$. By generalizing 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 $f_1(x) = \sin(x)\chi_{[a,b]}(x)$ is in $L^p(\mathbb{R})$ for $p \in \{1, 2\}$ and we can find the Fourier transform in the classical way.

Looking at real-world applications, we mostly deal with finite domains and 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 $[\,\cdot\,]$ *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. *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:* $\mathbb{Z}_{n_1 \times n_2} = \mathbb{Z}_{n_1} \times \mathbb{Z}_{n_2}$, *where* $n_1, n_2 \in \mathbb{N}$.

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 $g : \mathbb{Z}_{n_1 \times n_2} \to \mathbb{C}$ *is another function, we can identify it with a matrix* $g \in \mathbb{C}^{n_1 \times n_2}$ *and equally write* $g(j_1, j_2) = g_{j_1, j_2}$.

Definition 3.7 (Discrete convolution)**.** *Let* f *and* g *be discrete functions.*

(1) If $f, g : \mathbb{Z}_n \to \mathbb{C}$, *then the discrete convolution of* f *and* g *is given by*

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

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

(2) If $f, g : \mathbb{Z}_{n_1 \times n_2} \to \mathbb{C}$, then the discrete convolution of f and g is given by

$$(f * g)(j_1, j_2) = \sum_{k_1=0}^{n_1-1} \sum_{k_2=0}^{n_2-1} f(k_1, k_2) g(j_1 - k_1, j_2 - k_2)$$

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

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 will 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,

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

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

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

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

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

Using (3.1) and (3.2), we obtain

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

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 parts of both transforms only differ by this factor $|he^{-ia\omega_m}| = h$.

It is an automatic consequence from above that 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 simplify the notation, 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

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

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 in Theorem 3.19. 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 all $m \in \mathbb{Z}_n$.

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 $f(j)$ and f_j.*

(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 $\hat{f}, \hat{g}$ with respect to the variable j. Additionally, set the scalar values $a, b \in \mathbb{C}$ and $l \in \mathbb{Z}_n$. Then, the following properties hold for all $m \in \mathbb{Z}_n$:*

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

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

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

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

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

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

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

Hence,

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

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

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

(4) Since

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

we may use the first and third item to get

$$\begin{aligned}\mathcal{F}_d\left\{f(j)\cos\left(\frac{2\pi jl}{n}\right)\right\}(m) &= \mathcal{F}_d\left\{f(j)\frac{1}{2}\left(w_n^{lj} + w_n^{-lj}\right)\right\}(m) \\ &\overset{\text{(first item)}}{=} \frac{1}{2}\left(\mathcal{F}_d\left\{f(j)w_n^{lj}\right\}(m) + \mathcal{F}_d\left\{f(j)w_n^{-lj}\right\}(m)\right) \\ &\overset{\text{(third item)}}{=} \frac{1}{2}\left(\hat{f}(m+l) + \hat{f}(m-l)\right).\end{aligned}$$

□

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 $\hat{f}$ 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 all $k, m \in \mathbb{Z}_n$. 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 2.19. 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,*

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

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

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

for all $m \in \mathbb{Z}_n$. Then, we obtain by Definition 3.7

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

□

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

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

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

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

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

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

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

□

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 $f \in \mathbb{C}^n$ and we would like to find the resulting discrete Fourier transform $\hat{f}$. The definition of the discrete Fourier transform yields

$$\hat{f}_m = \sum_{k=0}^{n-1} f_k w_n^{mk}$$

for all $m \in \mathbb{Z}_n$. This expression is equivalent to

$$\begin{bmatrix} \hat{f}_0 \\ \hat{f}_1 \\ \vdots \\ \hat{f}_{n-1} \end{bmatrix} = \underbrace{\begin{bmatrix} w_n^0 & w_n^0 & \dots & w_n^0 \\ w_n^0 & w_n^1 & \dots & w_n^{n-1} \\ \vdots & \vdots & \vdots & \vdots \\ w_n^0 & w_n^{n-1} & \dots & w_n^{(n-1)^2} \end{bmatrix}}_{=W_n} \begin{bmatrix} f_0 \\ f_1 \\ \vdots \\ f_{n-1} \end{bmatrix}.$$

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

Listing 1: The Fourier transform.

```
function fHat = fourierTransform1(f)
    n = max(size(f));
    W = fourierMatrix(n);
    fHat = W*f;
end
```

Listing 2: Calculation of the 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 shows how to improve this algorithm in terms of computer performance.

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

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

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. According to Definition 3.11, we need to execute at most $(n-1)n$ 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. Even modern computers would require some time to finish the occurring 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 is inevitable to find a more efficient way to calculate the discrete Fourier transform. Fortunately, the mathematicians Cooley and Tukey discovered an algorithm that drastically decreases the amount of necessary multiplications. In their paper from 1965, they described their findings on 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's 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. 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 $w_n^k = e^{-2\pi i k/n} \in \mathbb{C}$. The fact that the entries of the matrix W_n are n-periodic 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

$$\begin{bmatrix} \hat{f}_0 \\ \hat{f}_1 \\ \hat{f}_2 \\ \hat{f}_3 \end{bmatrix} = \left[\begin{array}{c|c} W_2 & D_2 W_2 \\ \hline W_2 & -D_2 W_2 \end{array}\right] \left[\begin{array}{c} f_0 \\ f_2 \\ \hline f_1 \\ f_3 \end{array}\right]$$

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

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

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*

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

for all $k \in \mathbb{Z}_n$ 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 $m \in \mathbb{Z}_{2n}$.

(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, we note that the following two statements hold:

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

We deduce from the first item that

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

where again $m \in \mathbb{Z}_n$. □

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

$$c = W_n \begin{bmatrix} f_0 \\ f_2 \\ \vdots \\ f_{2n-4} \\ f_{2n-2} \end{bmatrix}, \quad d = D_n W_n \begin{bmatrix} f_1 \\ f_3 \\ \vdots \\ f_{2n-3} \\ f_{2n-1} \end{bmatrix}$$

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

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

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 $n = 2^s$ for $s \in \mathbb{N}$. 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:*

Input: $f = [f_0, f_1, \ldots, f_{n-1}]^T \in \mathbb{C}^n$, *where* $n = 2^s$ *for* $s \in \mathbb{N}$.
Output: $\hat{f} = [\hat{f}_0, \hat{f}_1, \ldots, \hat{f}_{n-1}]^T \in \mathbb{C}^n$, *the discrete Fourier transform of* f.

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

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

`// Split the vector` f `into even and odd parts.`
Set $a = [f_0, f_2, \ldots, f_{2n-2}]^T$, $b = [f_1, f_3, \ldots, f_{2n-1}]^T \in \mathbb{C}^n$.

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

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

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 yields the relationship*

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

Theorem 3.27. *Let $s \in \mathbb{N}$ be a number such that $n = 2^s$. 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 $s \in \mathbb{N}$.

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

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

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

(ii) Assume now that the formula holds for $s = \tilde{s} - 1$. We thus have

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

Using the recursion formula (3.4), we obtain

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

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

Remark 3.28. *One could argue that we do not need any multiplications for $n = 2$ 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 may be reduced from $\mathcal{O}(n^2)$ to $\mathcal{O}(n \log_2(n))$ 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 only need 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 this section, we would like to look at a possible implementation in MATLAB once more. 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: The fast Fourier transform using the Cooley–Tukey algorithm.

```
function fHat = fastFourierTransform1(f)

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

    n = n/2;

    c = fastFourierTransform1(f(1:2:2*n));
    d = fastFourierTransform1(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: The inverse fast Fourier transform.

```
function f = inverseFastFourierTransform1(fHat)
    n = length(fHat);
    fHat = conj(fHat);
    f = 1/n * conj(fastFourierTransform1(fHat));
end
```

3.5 Illustration of the fast Fourier transform

In this section, we are going to discuss two examples that use the MATLAB implementation for the fast Fourier transform. For this purpose, let $n = 2^7 = 128$ and consider the interval $[a, b] = [-1, 1] \subset \mathbb{R}$.

Example 3.29. *Define the function*

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

which is clearly in $(L^1 \cap L^2)([-1, 1])$.

We may calculate the continuous Fourier transform by

$$\hat{f}(\omega) = \int_{-1}^{1} f(t) e^{-i\omega t} dt = \int_{-0.5}^{0.5} e^{-i\omega t} dt.$$

This integral can be analytically evaluated, provided $\omega \in \mathbb{R} \setminus \{0\}$*:*

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

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

$$\omega_m = \frac{2\pi m}{b - a} = \pi m$$

for all $m \in \{0, 1, \ldots, n\}$*. We should not forget that we also dropped the factor* $he^{-ia\omega_m}$ *when deriving the discrete Fourier transform. Hence, the absolute values of the discrete and continuous versions will not match. It therefore makes sense to look at* $|\hat{f}(\omega_m)|$ *in the discrete case.*

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

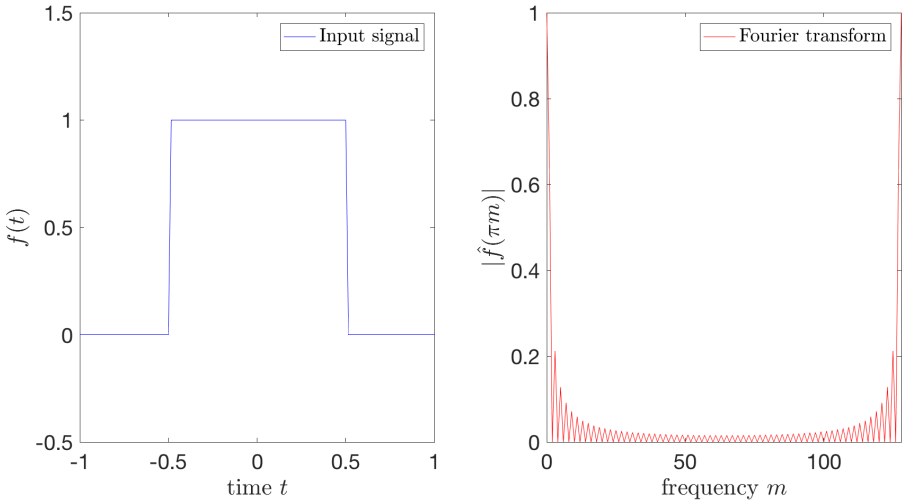


Figure 1: $f(t)$ on the left, the normed $|\hat{f}(\pi m)|$ on the right.

Example 3.30. *Consider the function on the left side in Figure 2, that is*

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

for all $t \in [-1, 1]$.

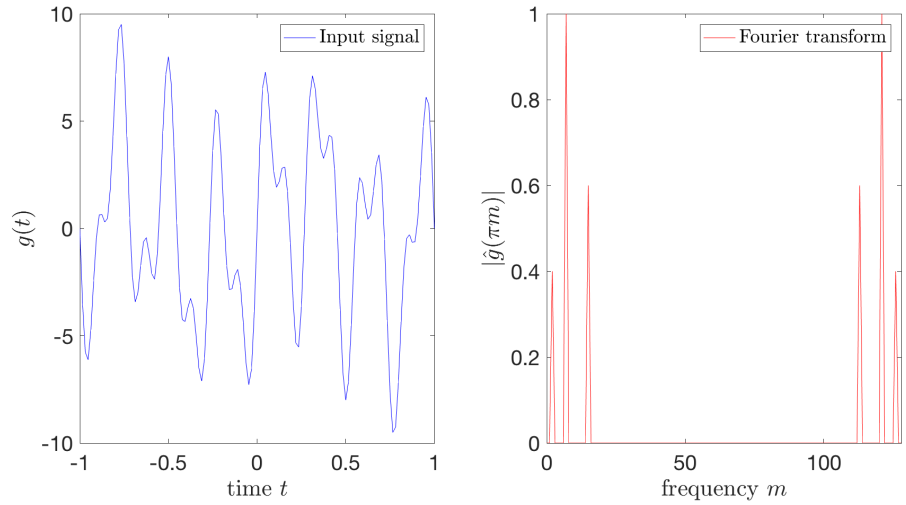


Figure 2: $g(t)$ on the left, the normed $|\hat{g}(\pi m)|$ on the right.

Obviously, this function is not in $L^2(\mathbb{R})$ *and we cannot apply our theory from*

the continuous Fourier transform. But as it is in $L^2([-1,1])$, we are still able to calculate the discrete Fourier transform.

Looking at the function $g(t)$, we note that it is composed of three sines with different amplitudes and frequencies. 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 can see that every frequency has a peak depending on the corresponding amplitude. For example, the peak of the sinusoid with frequency 7 is the largest, because its amplitude 5 is the highest.

Remark 3.31. *Due to the Hermitian symmetry shown in Theorem 3.15, we can see a repetition of the absolute graph of the discrete Fourier transform for $m > n/2$.*

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

4 The Gabor transform in one dimension

4.1 Motivation

The Fourier transform has proved 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 The 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 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 depends on time and 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 indicated 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* $f \in L^2(\mathbb{R})$ *and let* g *be a Gaussian function. 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 let 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 the 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 deduce

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

because f and 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 enables 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 let g be a Gaussian function. The Gabor transform of f with respect to g is defined by*

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

for all $(t,\omega) \in \mathbb{R}^2$.

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

(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

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

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

Theorem 4.6 (Basic properties of the Gabor transform)**.** *Let g be a Gaussian function. Assume we have the functions $f, h \in L^2(\mathbb{R})$ and their corresponding Gabor transforms $\tilde{f}_g$, $\tilde{h}_g$ with respect to g. Additionally, set the scalar values $a, b \in \mathbb{C}$ and $r \in \mathbb{R}$. Then, the following properties hold for all $(t, \omega) \in \mathbb{R}^2$:*

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

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

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

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

Proof. (1) The assertion follows directly from Definition 4.3.

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

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

(3) By applying Definition 4.3, we obtain

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

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

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

□

Theorem 4.7 (Parseval's formula)**.** *Let $f,h \in L^2(\mathbb{R})$ and let g be 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 let g be a Gaussian function. Now, assume that $t \in \mathbb{R}$ is fixed and define

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

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 2.31, we obtain

$$\begin{aligned}\int_{-\infty}^{\infty} \tilde{f}(t,\omega)\overline{\tilde{h}(t,\omega)}d\omega &= \int_{-\infty}^{\infty} \hat{\phi}(\omega)\overline{\hat{\psi}(\omega)}d\omega \\ &\overset{\text{(Theorem 2.31)}}{=} 2\pi \int_{-\infty}^{\infty} \phi(\tau)\overline{\psi(\tau)}d\tau \\ &= 2\pi \int_{-\infty}^{\infty} f(\tau)g(\tau-t)\overline{h(\tau)g(\tau-t)}d\tau \\ &= 2\pi \int_{-\infty}^{\infty} f(\tau)\overline{h(\tau)}\,|g(\tau-t)|^2 d\tau.\end{aligned}$$

(iii) It follows from 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. *If $f \in L^2(\mathbb{R})$, then the resulting Gabor transform $\tilde{f}(t,\omega)$ is in $L^2(\mathbb{R}^2)$ because*

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

It thus 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*

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

is the inverse Gabor transform for all $t \in \mathbb{R}$.

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

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

for the Fourier transform with respect to t. Thus, we may apply Definition 2.15—the inverse Fourier transform for a function in $L^1(\mathbb{R})$—to obtain

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

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

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

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

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

By rearranging the terms in the integrals, we obtain

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

(ii) We remember from Theorem 2.25 that

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

Thus,

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

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

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

□

Physically 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 The 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 will use Chapter 3 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 thus motivates the truncation of the integral in the discrete case.

To start, let us divide this interval into $n \in \mathbb{N}$ equidistant pieces of width $h = \frac{b-a}{n}$. We define

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

Hence,

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

The discretization of the frequencies can again be done with

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

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

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

Using (4.3) and (4.4), we obtain

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

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 a simple evaluation of the continuous Gaussian function, but we find 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 make sure it is centered in the interval. Also, we shall reserve the variable $n \in \mathbb{N}$ for the rest of this chapter and link it to the discretization above.

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 $t_c \in \mathbb{R}$. 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 as in the derivation of the discrete Fourier transform, we obtain that the discrete versions of f and $\tilde{f}_g$ will also be $[a,b]$-periodic.

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

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 written 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*

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

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

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

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

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

We know from Theorem 3.27 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 $a, b \in \mathbb{C}$ and $l \in \mathbb{Z}_n$. Then, the following properties hold for all $(j, m) \in \mathbb{Z}_n^2$:*

(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 that

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

(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) Knowing 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 let g be a centered discrete Gaussian. Furthermore, let $\tilde{f}_g$ be the discrete Gabor transform of f 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 $\tilde{f}_g$ be 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 the 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 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 all $m \in \mathbb{Z}_n$, where the norm $\|\cdot\|$ denotes the Euclidean norm in $\mathbb{R}^d$.

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

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

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

By the n-periodicity of g, we deduce

$$\frac{1}{\|g\|^2}\sum_{j=0}^{n-1} f(m)g(m-j)g(m-j) = f(m)\frac{1}{\|g\|^2}\sum_{j=0}^{n-1} g(j)^2 = f(m)$$

for all $m \in \mathbb{Z}_n$. □

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

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

4.4 Implementation of the discrete Gabor transform

Approaching the end of 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 sophisticated 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.

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 5: The Gabor transform.

```
function fTilde = gaborTransform1(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, :) = fastFourierTransform1(phi);
    end

end
```

Listing 6: The inverse Gabor transform.

```
function f = inverseGaborTransform1(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 = inverseFastFourierTransform1(phiHat);
        f = f + g(tShift) .* phi;
    end

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

end
```

4.5 Illustration of the discrete Gabor transform

We are now ready to use the algorithm of the discrete Gabor transform in some examples and plot the results in MATLAB. For these examples, we set $n = 2^7 = 128$ and consider the interval $[a, b] = [-1, 1] \subset \mathbb{R}$. In the following 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 applied 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 will look at three different discrete impulse functions.

Example 4.20. *Let us fix* $\beta = 10$. *Thus, the Gaussian window function is defined as* $g(t) = e^{-(10t)^2}$.

(1) Define the function

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

Figure 3 shows the original function u, *the Gaussian window function* g, *and the discrete Gabor transform* $\tilde{u}_g$. *We may compare the result with the discrete Fourier transform that was presented in Figure 1.*

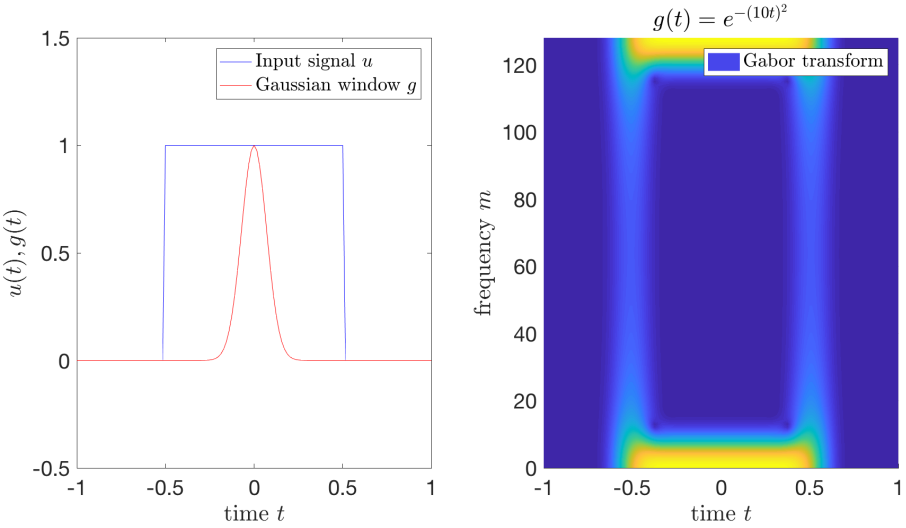


Figure 3: $u(t)$ and $g(t)$ on the left, $|\tilde{u}_g(t, \pi m)|$ on the right.

We can see that the resulting Gabor transform is practically 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 discontinuities 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.

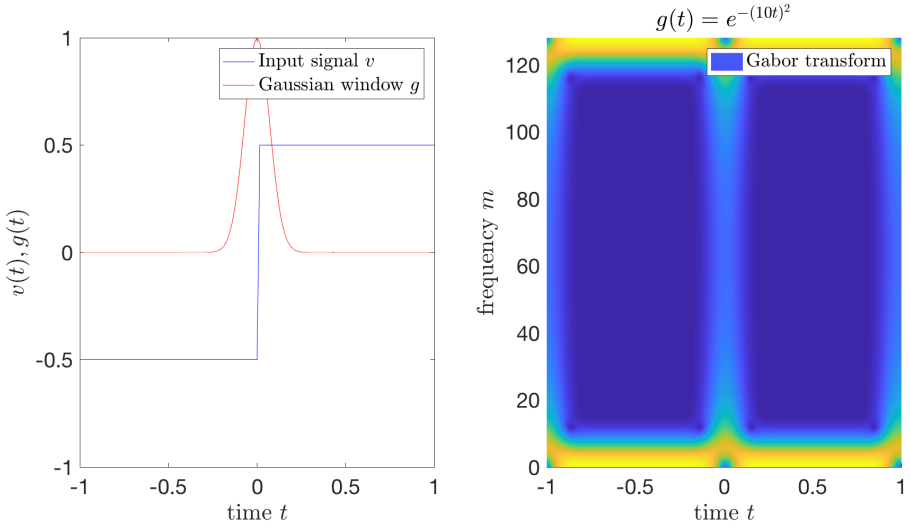


Figure 4: $v(t)$ and $g(t)$ on the left, $|\tilde{v}_g(t, \pi m)|$ on the right.

The resulting Gabor transform looks similar to the one in the first example, but it has its peaks at frequency 0 *and at times when the original function was horizontal. The discontinuity at* $t = 0$ *is clearly visible in the Gabor transform for all frequencies. As the function* v *is* $[-1, 1]$*-periodic, we are able to see the discontinuities 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*, the Gaussian window function* g *and the Gabor transform* $\tilde{w}_g$*. As in the second example, peaks are visible at frequency* 0 *and at times when the original function was horizontal. For* $t \in [-0.5, +0.5]$*, the resulting transform is close to zero for all frequencies. While it shows the discontinuities at times* $t = \pm 1$ *for all frequencies, it nearly does not reveal the kinks at* $t = \pm 0.5$*. Zooming in at the points* $(t, m) = (\pm 0.5, 15)$*, we can see a tiny bulge which indicates a kink.*

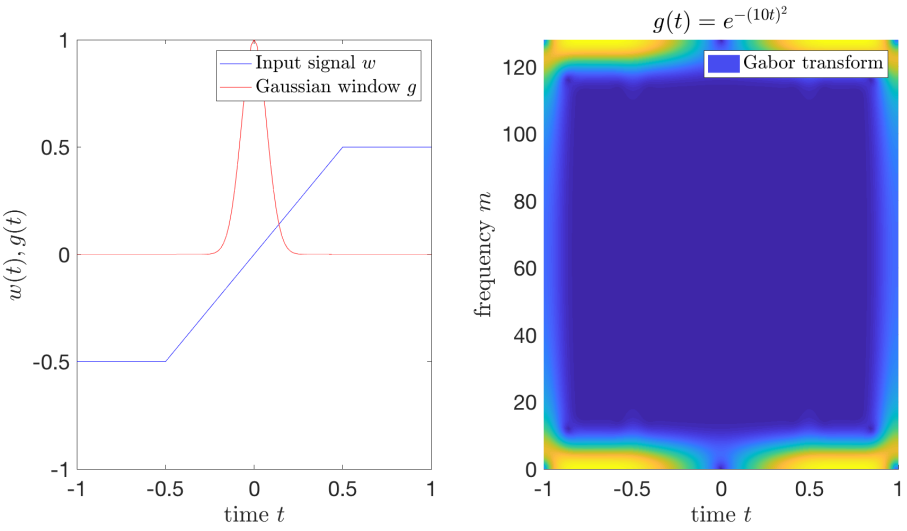


Figure 5: $w(t)$ and $g(t)$ on the left, $|\tilde{w}_g(t, \pi m)|$ on the right.

In the last chapter, we have seen a function that was composed of three different frequencies in Example 3.30. 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*

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

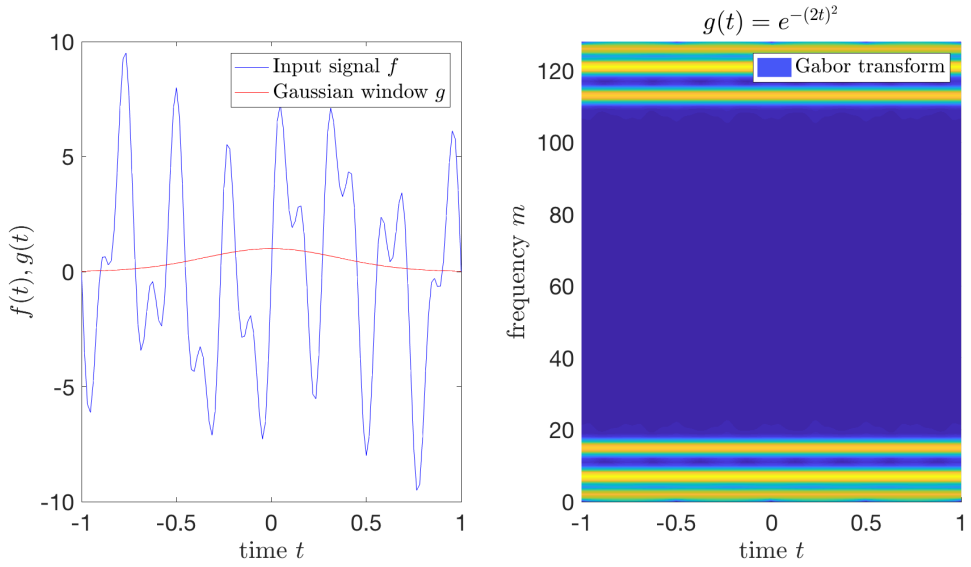


Figure 6: $f(t)$ and $g(t)$ on the left, $|\tilde{f}_g(t, \pi m)|$ on the right.

As noted, this function contains the 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 can see that these three frequencies do not change during the given interval.*

Here, the Gabor transform does not have any advantages over the Fourier transform from Figure 2: 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 is revealed when considering examples with varying frequencies. In real-world applications, this is a common scenario: Let us 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* $g(t) = e^{-\beta^2 t^2}$ *at the bottom.*

These graphs are a good illustration of the importance of choosing an appropriate Gaussian window function. It is obvious that β *should 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 $(j, m) \in \mathbb{Z}_n^2$. *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 either. 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.

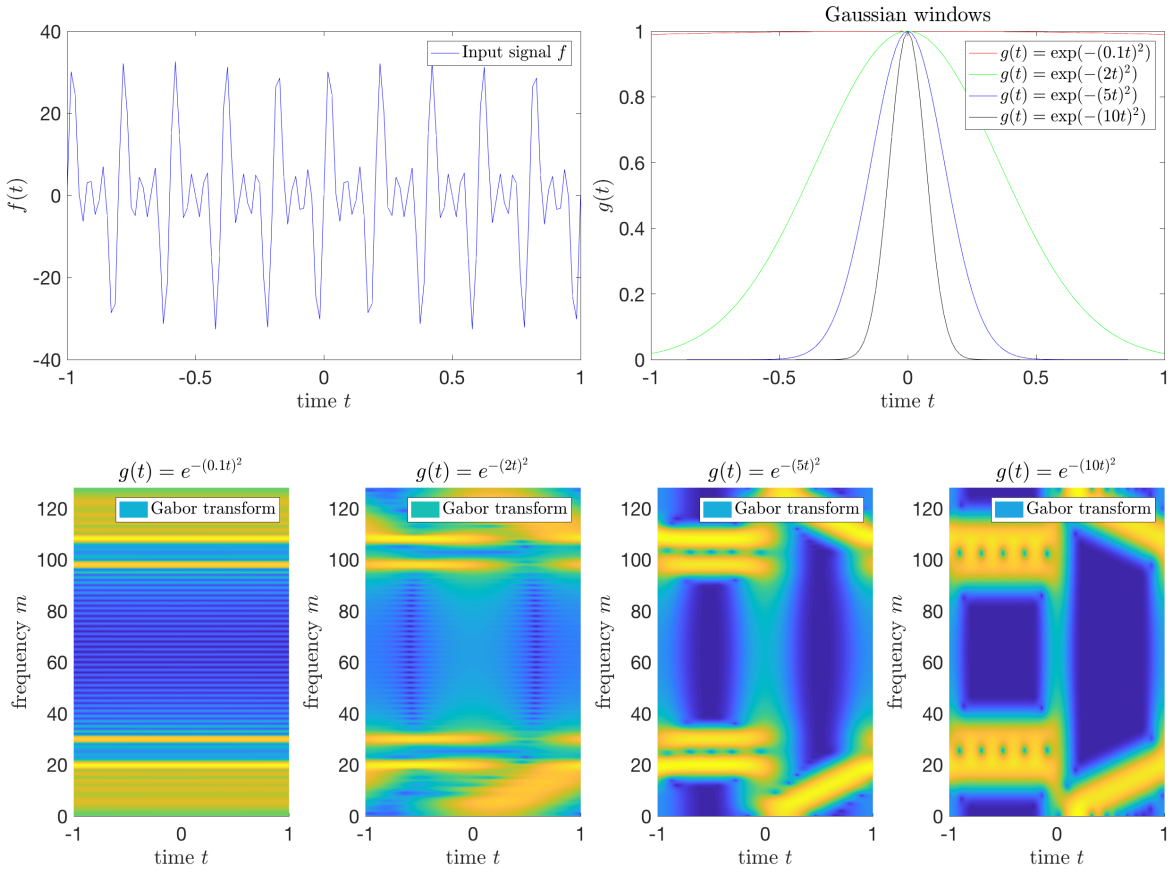


Figure 7: $f(t)$ and $g(t)$ on the top, $|\tilde{f}_g(t, \pi m)|$ on the bottom.

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

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

(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 too large.

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

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

5 The 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 2.2 and Chapter 3.2 for preliminaries.

5.2 The Fourier transform in $L^1(\mathbb{R}^2)$ and $L^2(\mathbb{R}^2)$

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 $f \in L^1(\mathbb{R}^2)$.*

(1) The Fourier transform of f is defined by

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

for all $\boldsymbol{\omega} \in \mathbb{R}^2$.

(2) The inverse Fourier transform is defined as

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

for all $\boldsymbol{x} \in \mathbb{R}^2$.

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 2.23.

(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 and Definition 2.15, it is obvious that this factor usually is $(2\pi)^2$ in two dimensions instead of 2π in one dimension.

In Theorem 2.19, 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*

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

for almost every $\boldsymbol{\omega} \in \mathbb{R}^2$.

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

5.3 The 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 $[a_x, b_x] \times [a_y, b_y] \subset \mathbb{R}^2$ by

$$\begin{aligned} a_x = x_0 < x_1 < x_2 < \dots x_{n_1-1} < x_{n_1} = b_x, \\ a_y = y_0 < y_1 < y_2 < \dots y_{n_2-1} < y_{n_2} = b_y, \end{aligned}$$

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

$$\begin{aligned} \omega_{m_1} &= \frac{2\pi m_1}{b_x - a_x}, \quad m_1 \in \{0, 1, \dots, n_1\}, \\ \nu_{m_2} &= \frac{2\pi m_2}{b_y - a_y}, \quad m_2 \in \{0, 1, \dots, n_2\}. \end{aligned}$$

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

We shall reserve the variables $n_1, n_2 \in \mathbb{N}$ for the rest of this thesis and link them to the discretization above.

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

(1) We define the discrete Fourier transform by

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

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

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

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

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

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 especially important because we are going to use it in our implementation for Gabor filters in two dimensions in Chapter 6.5. 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 3.4 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 are required to assume the same in two dimensions. In order to simplify the notation, we shall use $n = n_1 = n_2 = 2^s$ from now on, 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: The fast Fourier transform in two dimensions.

```
function fHat = fastFourierTransform2(f)

    [n, ~] = size(f);

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

    for k=1:1:n
       fHatTemp(k,:) = fastFourierTransform1(f(k, :).').';
    end

    for k=1:1:n
       fHat(:,k) = fastFourierTransform1(fHatTemp(:, k));
    end

end
```

Listing 8: The inverse fast Fourier transform in two dimensions.

```
function f = inverseFastFourierTransform2(fHat)
    n = length(fHat);
    fHat = conj(fHat);
    f = 1/n * 1/n * conj(fastFourierTransform2(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 The Gabor transform and Gabor filters 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 dependent 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 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 The 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 transform 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.

The general form of a 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 a parameter for 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 need 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.*

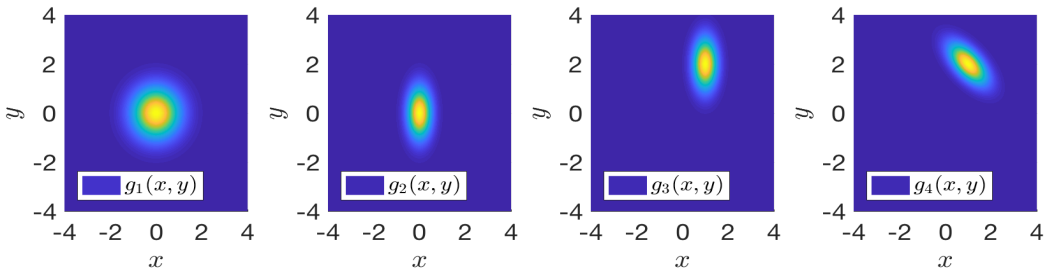


Figure 8: Contour plots of all four Gaussian functions.

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 let g be a Gaussian function. The Gabor transform of f with respect to g is defined by*

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

for all $\boldsymbol{x},\boldsymbol{\omega} \in \mathbb{R}^2$.

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 shall 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 spatial 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 a common approach to calculate the two-dimensional Gabor transform of a function by the method presented in the remark above. We deduce from Theorem 5.3 that the appearing convolution may be evaluated by

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

for almost every $\boldsymbol{x} \in \mathbb{R}^2$. 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. 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

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

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

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

for all $(u, v) \in \mathbb{R}^2$. □

Remark 6.8. *We have chosen $x_c = y_c = \theta = 0$ considering Definition 6.1. The statement of the theorem may be generalized for arbitrary x_c, y_c, and θ.*

6.3 The discrete Gabor transform

In order to derive the discrete version of the Gabor transform in two dimensions, we can use the same discretization as already introduced in the previous chapters. Since we would like to keep the formulas simple, we use the same amount of samples for both original and frequency domain for each dimension once again. If we say the variables $(x, y) \in \mathbb{R}^2$ describe the spatial 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 all $(j_1, j_2), (m_1, m_2) \in \mathbb{Z}_{n_1 \times n_2}$.

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

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

for $g_{m_1,m_2}(j_1, j_2) = g(j_1, j_2) w_n^{-m_1 j_1 - m_2 j_2}$.

We may extract the two-dimensional Gabor filter from Remark 6.12:

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 $(m_1, m_2) \in \mathbb{Z}_{n_1 \times n_2}$. The function g_{m_1,m_2} defined 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 $(k_1, k_2) \in \mathbb{Z}_{n_1 \times n_2}$ is called discrete Gabor filter.

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 work with Gabor filters from the beginning.

6.4 Semi-discrete Gabor filters

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

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

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

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

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

Definition 6.14 (General 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, a function of the form*

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

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

Remark 6.15. *Including the parameters from the Gaussian function, the general 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 has led to many implementations: As of today, Gabor filters have been used in image analysis, image compression, object recognition, medical diagnostics, and many more fields.

However, most authors use a parametrization that is a little less general as 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 derived directly from our definition: If we set

$$\alpha = 1, \quad \beta = \frac{1}{\sqrt{2}\,\sigma}, \quad \gamma = \frac{\xi}{\sqrt{2}\,\sigma}, \quad \omega = \frac{2\pi\cos(\theta)}{\lambda}, \quad \nu = \frac{2\pi\sin(\theta)}{\lambda},$$
$$x_c = \frac{n_1}{2}, \quad y_c = \frac{n_2}{2},$$

we obtain the following form:

Definition 6.16 (Semi-discrete Gabor filter)**.** *Let $\xi, \sigma, \lambda > 0$ and $\theta \in [0, 2\pi[$. A function of the form*

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

for all $(k_1, k_2) \in \mathbb{Z}_{n_1 \times n_2}$ is called semi-discrete Gabor filter, where

$$\begin{bmatrix} k_1' \\ k_2' \end{bmatrix} = \begin{bmatrix} \cos(\theta) & \sin(\theta) \\ -\sin(\theta) & \cos(\theta) \end{bmatrix} \begin{bmatrix} k_1 - \frac{n_1}{2} \\ k_2 - \frac{n_2}{2} \end{bmatrix}.$$

We are going to see in the next sections that these Gabor filters help to extract image features which are aligned orthogonally to the direction (angle θ) of the Gaussian function and the sinusoid. The role of θ 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 angles.

Definition 6.17 (Gabor filter bank)**.** *A Gabor filter bank is a finite collection of arbitrary Gabor filters.*

We may deduce the following helpful property from Definition 6.16:

Theorem 6.18. *Let $\xi, \sigma, \lambda > 0$ and $\theta_1 \in [0, 2\pi[$. If we set $\theta_2 = \theta_1 + \pi$, then*

$$g_{\xi,\sigma,\lambda,\theta_2}(k_1, k_2) = \overline{g_{\xi,\sigma,\lambda,\theta_1}}(k_1, k_2)$$

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

Proof. (i) First, we define for all $(k_1, k_2) \in \mathbb{Z}_{n_1 \times n_2}$

$$g_{\xi,\sigma,\lambda,\theta_1}(k_1, k_2) = \exp\left(-\frac{k_1'^2 + \xi^2 k_2'^2}{2\sigma^2}\right) \exp\left(2\pi i \frac{k_1'}{\lambda}\right),$$
$$g_{\xi,\sigma,\lambda,\theta_2}(k_1, k_2) = \exp\left(-\frac{k_1''^2 + \xi^2 k_2''^2}{2\sigma^2}\right) \exp\left(2\pi i \frac{k_1''}{\lambda}\right),$$

where

$$\begin{bmatrix} k_1' \\ k_2' \end{bmatrix} = \begin{bmatrix} \cos(\theta_1) & \sin(\theta_1) \\ -\sin(\theta_1) & \cos(\theta_1) \end{bmatrix} \begin{bmatrix} k_1 - \frac{n_1}{2} \\ k_2 - \frac{n_2}{2} \end{bmatrix},$$
$$\begin{bmatrix} k_1'' \\ k_2'' \end{bmatrix} = \begin{bmatrix} \cos(\theta_2) & \sin(\theta_2) \\ -\sin(\theta_2) & \cos(\theta_2) \end{bmatrix} \begin{bmatrix} k_1 - \frac{n_1}{2} \\ k_2 - \frac{n_2}{2} \end{bmatrix}.$$

(ii) We note that $\cos(\theta_2) = -\cos(\theta_1)$ and $\sin(\theta_2) = -\sin(\theta_1)$. Looking at the coordinate transforms from the first item, we deduce $k_l'' = -k_l'$ and $k_l''^2 = k_l'^2$ for $l \in \{1, 2\}$.

(iii) From the second item, we obtain

$$\begin{aligned} g_{\xi,\sigma,\lambda,\theta_2}(k_1, k_2) &= \exp\left(-\frac{k_1''^2 + \xi^2 k_2''^2}{2\sigma^2}\right) \exp\left(2\pi i \frac{k_1''}{\lambda}\right) \\ &= \exp\left(-\frac{k_1'^2 + \xi^2 k_2'^2}{2\sigma^2}\right) \exp\left(-2\pi i \frac{k_1'}{\lambda}\right) \\ &= \overline{g_{\xi,\sigma,\lambda,\theta_1}}(k_1, k_2). \end{aligned}$$

□

Theorem 6.19. *Let $f : \mathbb{Z}_{n_1 \times n_2} \to \mathbb{R}$ be a real discrete function. Additionally, let $\xi, \sigma, \lambda > 0$, $\theta_1 \in [0, 2\pi[$, and $\theta_2 = \theta_1 + \pi$. Then,*

$$(f * g_{\xi,\sigma,\lambda,\theta_2})(j_1, j_2) = (\overline{f * g_{\xi,\sigma,\lambda,\theta_1}})(j_1, j_2)$$

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

Proof. We first note that $f = \bar{f}$ because f is a real function. By Theorem 6.18, we also know that $g_{\xi,\sigma,\lambda,\theta_2} = \overline{g_{\xi,\sigma,\lambda,\theta_1}}$. Using Definition 3.7, we finally obtain

$$\begin{aligned} (f * g_{\xi,\sigma,\lambda,\theta_2})(j_1, j_2) &= \sum_{k_1=0}^{n_1-1} \sum_{k_2=0}^{n_2-1} f(k_1, k_2) g_{\xi,\sigma,\lambda,\theta_2}(j_1 - k_1, j_2 - k_2) \\ \overset{(f = \bar{f}, \text{ Theorem 6.18})}{=} &\sum_{k_1=0}^{n_1-1} \sum_{k_2=0}^{n_2-1} \overline{f(k_1, k_2) g_{\xi,\sigma,\lambda,\theta_1}(j_1 - k_1, j_2 - k_2)} \\ &= (\overline{f * g_{\xi,\sigma,\lambda,\theta_1}})(j_1, j_2). \end{aligned}$$

□

Remark 6.20. *The two previous theorems prove that we are only required to study semi-discrete Gabor filters with angles* $\theta \in [0, \pi[$.

6.5 Implementation of Gabor filters

For the rest of this chapter, we assume that $n = n_1 = n_2 = 2^s$ for an $s \in \mathbb{N}$. Thus, we define the input image, Gabor filter, and resulting convolution to be functions $\mathbb{Z}_n^2 \to \mathbb{C}$.

Below, we present a possible MATLAB implementation for the creation of two-dimensional general semi-discrete Gabor filters (cf. Definition 6.14). 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 in two dimensions.

```
function gwc = gaborFilter2(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
```

According to Theorem 5.6, we know that the 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: Convolution of an image with a Gabor filter in two dimensions.

```
function fStar = gaborConvolution2(f, gwc)
    fHat = fastFourierTransform2(f);
    gwHat = fastFourierTransform2(gwc);
    fStar = inverseFastFourierTransform2(fHat .* gwHat);
end
```

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

Remark 6.21. *(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 $\mathcal{O}(n^2 \log_2(n))$.

6.6 Illustration of Gabor filters

Concluding 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 semi-discrete Gabor filter parametrization $(\xi, \sigma, \lambda, \theta) \in \mathbb{R}^4$ from Definition 6.16. Hence, a filter reads

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

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 $(k_1, k_2) \in \mathbb{Z}_n^2$. When discussing the output image, we shall write $f^* = f * g_{\xi,\sigma,\lambda,\theta}$ in order to simplify notation.

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

(1) The input grayscale image f.

(2) The sum of all four convolutions in the images (3) to (6).

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

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

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

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

Example 6.22. *Set $\xi = 0.5$, $\sigma = 1$, $\lambda = 2$, and $\theta = \frac{k\pi}{4}$ for $k \in \{0, 1, 2, 3\}$. The basic Gabor filter with angle $\theta = 0$ is visualized in two contour plots in Figure 9 below. The left plot shows the real part of the filter, the right plot shows the imaginary part. Large values are shown in yellow, low values in dark blue.*

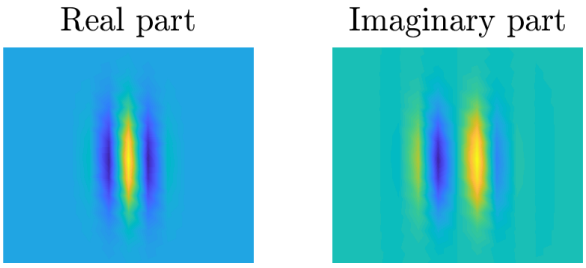


Figure 9: Real and imaginary part of the Gabor filter.

As a first example, we chose a synthetic grayscale image with different shapes and colors.

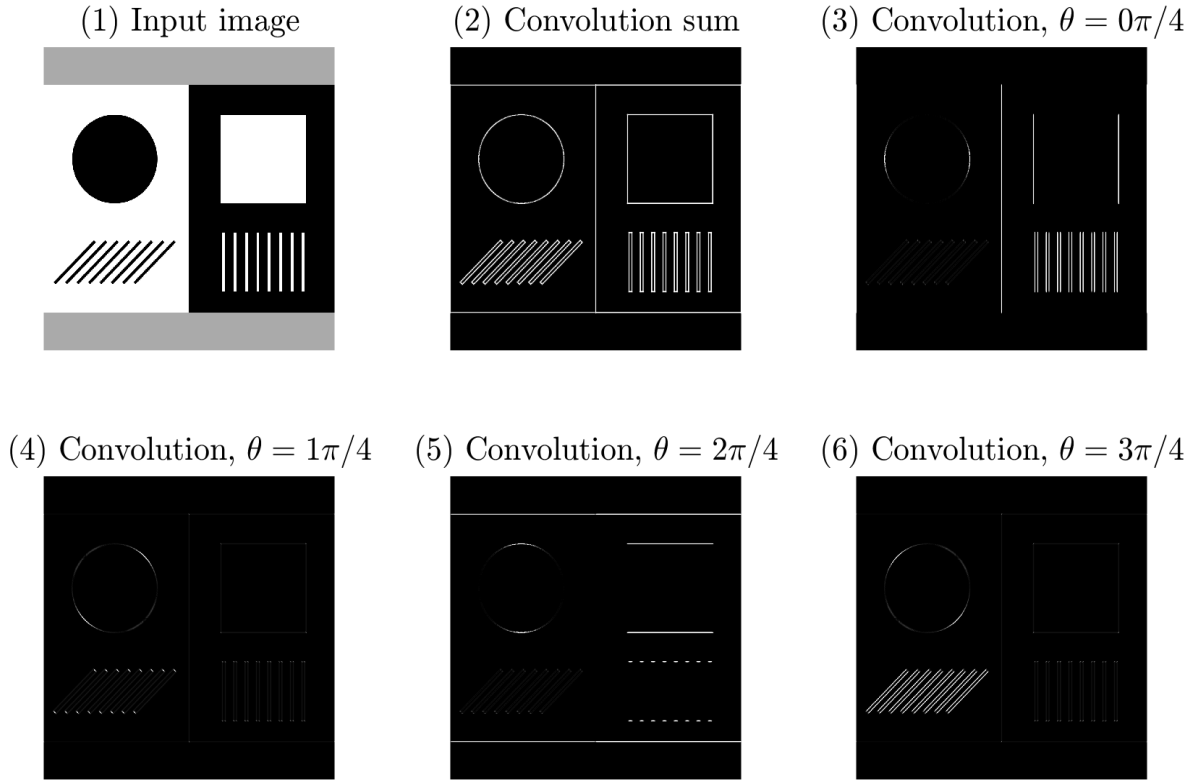


Figure 10: Input image and convolutions.

Looking at the sum of all convolutions in image (2)*, 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 the images* (3) *to* (6)*:*

(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 edge 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 clearly 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. Again, the circle is glowing at the points that are nearly orthogonal to the filter.*

Example 6.23. *Let* $\xi = 0.5$*,* $\sigma = 1$*,* $\lambda = 4$*, and* $\theta = \frac{k\pi}{4}$ *for* $k \in \{0, 1, 2, 3\}$*. Again, the basic Gabor filter is first shown in Figure 11.*

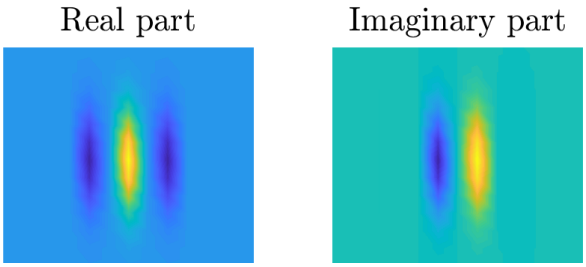
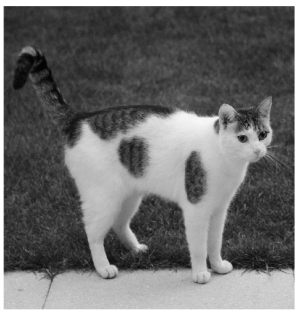


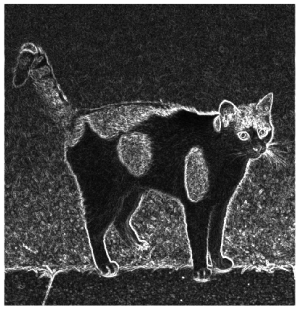
Figure 11: Real and imaginary part of the Gabor filter.

The input image shows a domestic cat in a garden. Our hope is that certain shapes may be detected in the same way as in the example before.

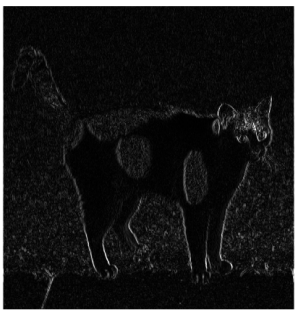
(1) Input image



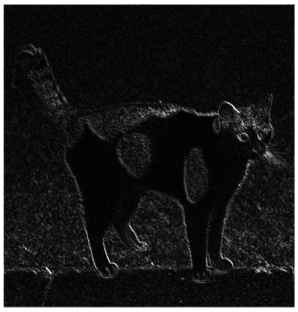
(2) Convolution sum



(3) Convolution, $\theta = 0\pi/4$



(4) Convolution, $\theta = 1\pi/4$



(5) Convolution, $\theta = 2\pi/4$

(6) Convolution, $\theta = 3\pi/4$

Figure 12: Input image 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 recognizable—similarly 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 of the image vanishes due to the bokeh effect. Looking closer at the convolution for each angle, we note:

(i) $\theta = 0$*: 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.*

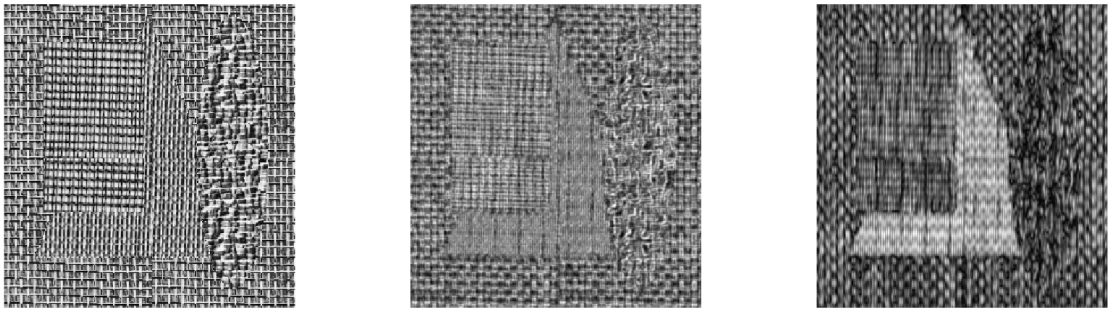
(ii) $\theta = \frac{\pi}{4}$*: It is interesting to see that even the small hairs at the cat's chest and the whiskers on the right side are very bright. Strong transitions like the top of the left ear or the hind legs are also glowing more than the rest.*

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

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

Example 6.24. *Let $\xi = 0.5$, $\sigma = 1$, $\lambda = 4$, and $\theta = \frac{k\pi}{4}$ for $k \in \{0, 1, 2, 3\}$, which means that we chose the same filters as in the example above. This time, we are looking at an image consisting of different textures.*

(1) Input image (2) Convolution sum (3) Convolution, $\theta = 0\pi/4$



(4) Convolution, $\theta = 1\pi/4$ (5) Convolution, $\theta = 2\pi/4$ (6) Convolution, $\theta = 3\pi/4$

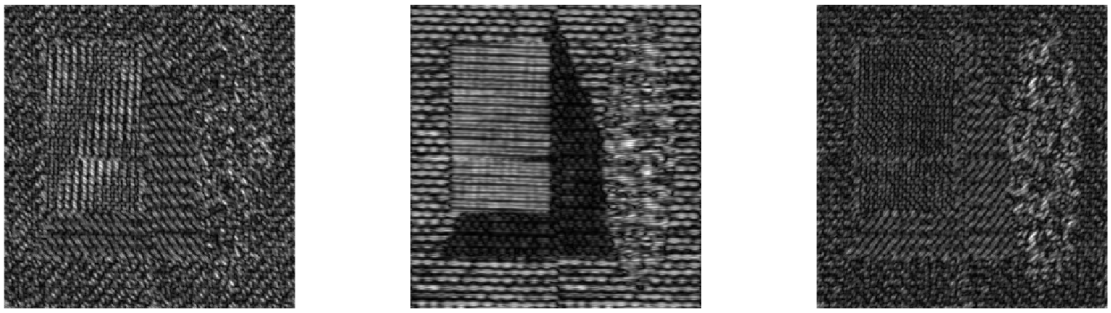


Figure 13: Input image and convolutions.

This example shows the importance of an intelligent selection of angles. As the textures are mostly aligned along the axes, the convolutions at the 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 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 allows the reader to 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, our 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 our MATLAB classes is attached below, but may also be found at 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.

Use the **help** function inside MATLAB to get information about all functions and their parameters.

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.
            %
            % The matrix f needs to be of size 2^s*1 or 2^s*2^s.
            %
            % Params:
            % - matrix f        any complex input matrix
            %
            % Returns:
            % - matrix fHat     the complex output matrix

            % Determine dimension of input vector/matrix
            [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 f has invalid dimensions.');
            end
```

```
32
33          end
34
35          function f = ifft(fHat)
36              % Calculates the 1d/2d inverse fast Fourier transform of a
37              % vector/matrix.
38              %
39              % The matrix fHat needs to be of size 2^s*1 or 2^s*2^s.
40              %
41              % Params:
42              % - matrix fHat     any complex input matrix
43              %
44              % Returns:
45              % - matrix f        the complex output matrix
46
47              % Determine dimension of input vector/matrix
48              [n1, n2] = size(fHat);
49
50              if n1 > 1 && n2 > 1
51                  f = Fourier.ifft2(fHat);
52              elseif n1 == 1 || n2 == 1
53                  f = Fourier.ifft1(fHat);
54              else
55                  error('Input fHat has invalid dimensions.');
56              end
57          end
58
59          function xy = conv(x, y)
60              % Calculates the (circular) convolution using the fast Fourier
61              % transform.
62              %
63              % The matrices x, y need to be of size 2^s*1 or 2^s*2^s.
64              %
65              % Params:
66              % - matrix x    any complex input matrix
67              % - matrix y    any complex input matrix
68              %
69              % Returns:
70              % - matrix xy   the discrete convolution x*y
71
72              % Determine dimension of input data
73              [n1, n2] = size(x);
74
75              % Show error if necessary
76              if (isequal(size(x), size(y)) == 0)
77                  error('Input vectors/matrices must have same dimensions.');
78              end
79
80              % Calculate convolution depending on dimension
```

```
81              if (n1 > 1 && n2 > 1)
82                  xy = Fourier.ifft2(Fourier.fft2(x) .* Fourier.fft2(y));
83              elseif (n1 > 1 && n2 == 1) || (n1 == 1 && n2 > 1)
84                  xy = Fourier.ifft(Fourier.fft1(x) .* Fourier.fft1(y));
85              else
86                  error('Input data is invalid, vectors/matrices are of size
                        0.');
87              end
88
89          end
90
91      end
92
93      methods (Static, Access = private)
94
95          function fHat = fft1(f)
96              % Calculates the 1d fast Fourier transform of a vector.
97              %
98              % Params:
99              % - vector f        any complex input vector
100             %
101             % Returns:
102             % - vector fHat     the complex output vector
103
104             % Determine dimension of input vector
105             [n1, n2] = size(f);
106             m = min(n1, n2);
107             n = max(n1, n2);
108
109             % Show error if necessary
110             if m ~= 1
111                 error('Input must not be a matrix.');
112             end
113             if n < 2 || rem(log2(n), 1) > 0
114                error('Input vector must be of length 2^s for any positive
                        integer s.');
115             end
116
117             % If n is 2, apply the Fourier matrix
118             if n == 2
119                 fHat = zeros(n1, n2);
120                 fHat(1) = f(1)+f(2);
121                 fHat(2) = f(1)-f(2);
122                 return;
123             end
124
125             % Halve the value of n
126             n = n/2;
127
```

```
128             % Calculate c, d
129             c = Fourier.fft(f(1:2:2*n));
130             d = Fourier.fft(f(2:2:2*n));
131 
132             % Correct d value
133             for k=1:1:n
134                d(k) = exp(-1i*pi*(k-1)/n)*d(k);
135             end
136 
137             % Set the values and return
138             fHat = zeros(size(f));
139             fHat(1:n) = c+d;
140             fHat(n+1:2*n) = c-d;
141 
142         end
143 
144         function f = ifft1(fHat)
145             % Calculates the 1d inverse fast Fourier transform of a vector.
146             %
147             % Params:
148             % - vector fHat     any complex input vector
149             %
150             % Returns:
151             % - vector f        the complex output vector
152 
153             % Calculate the IFFT through the FFT by this equivalence:
154             % f = 1/n conj(W)*fHat <=> f = 1/n conj(W*conj(fHat))
155 
156             n = length(fHat);
157             fHat = conj(fHat);
158             f = 1/n * conj(Fourier.fft(fHat));
159 
160         end
161 
162         function fHat = fft2(f)
163             % Calculates the 2d fast Fourier transform of a matrix.
164             %
165             % Params:
166             % - matrix f        any complex input matrix
167             %
168             % Returns:
169             % - matrix fHat     the complex output matrix
170 
171             % Determine dimension of input matrix
172             [n1, n2] = size(f);
173             n = n1;
174 
175             % Show error if necessary
176             if abs(n2-n1) > 0
```

```
177                 error('Input matrix must be square.');
178             end
179             if n < 2 || rem(log2(n), 1) > 0
180                 error('Input matrix must be of size 2^s*2^s for any
                        positive integer s.');
181             end
182 
183             % Initialize zero matrices
184             fHatTemp = zeros(n, n);
185             fHat = zeros(n, n);
186 
187             % Apply the 1d fast Fourier transform for all rows
188             for k=1:1:n
189                fHatTemp(k,:) = Fourier.fft1(f(k, :).').';
190             end
191 
192             % Apply the 1d fast Fourier transform for all cols
193             for k=1:1:n
194                fHat(:,k) = Fourier.fft1(fHatTemp(:, k));
195             end
196 
197         end
198 
199         function f = ifft2(fHat)
200             % Calculates the 2d inverse fast Fourier transform of a matrix.
201             %
202             % Params:
203             % - matrix fHat     any complex input matrix
204             %
205             % Returns:
206             % - matrix f        the complex output matrix
207 
208             % Calculate the IFFT through the FFT by this equivalence:
209             % f = 1/n^2 conj(W)*fHat <=> f = 1/n^2 conj(W*conj(fHat))
210 
211             n = length(fHat);
212             fHat = conj(fHat);
213             f = 1/n^2 * conj(Fourier.fft2(fHat));
214 
215         end
216 
217     end
218 
219     %%%%%%%%%%%%%%%%
220     % Test methods %
221     %%%%%%%%%%%%%%%%
222 
223     methods (Static)
224 
```

```
225         function test()
226             % Tests the functionality of the class and prints the result.
227 
228             % Setup test variables
229             tol = 10^(-9);
230             amount = 5;
231 
232             fprintf('========================================\n\n');
233             fprintf('Starting Fourier class test...\n\n');
234             fprintf('- Error tolerance: %d\n', tol);
235             fprintf('- Tests per method: %d\n', amount);
236             fprintf('\n');
237 
238             % Test FFT
239             fprintf('----------\nTesting FFT:\n');
240             for k=1:amount
241                 Fourier.fftTest(tol)
242             end
243 
244             % Test IFFT
245             fprintf('----------\nTesting IFFT:\n');
246             for k=1:amount
247                 Fourier.ifftTest(tol)
248             end
249 
250             % Test FFT/IFFT
251             fprintf('----------\nTesting FFT&IFFT:\n');
252             for k=1:amount
253                 Fourier.fftifftTest(tol)
254             end
255 
256             % Test FFT2
257             fprintf('----------\nTesting FFT2:\n');
258             for k=1:amount
259                 Fourier.fft2Test(tol)
260             end
261 
262             % Test IFFT2
263             fprintf('----------\nTesting IFFT2:\n');
264             for k=1:amount
265                 Fourier.ifft2Test(tol)
266             end
267 
268             % Test FFT2/IFFT2
269             fprintf('----------\nTesting FFT2&IFFT2:\n');
270             for k=1:amount
271                 Fourier.fft2ifft2Test(tol)
272             end
273 
```

```
            % Test circular convolution
            fprintf('------------\nTesting Convolution:\n');
            for k=1:amount
                Fourier.convTest(tol)
            end

            fprintf('------------\nTesting completed with 0 issues.\n');

        end

    end

    methods (Static, Access = private)

        function fftTest(tol)
            y = Fourier.randc(2^16, 1);
            diff = max(abs(Fourier.fft(y) - fft(y)));
            Fourier.printTestResult(diff, tol);
        end

        function ifftTest(tol)
            y = Fourier.randc(2^16, 1);
            diff = max(abs(Fourier.ifft(y) - ifft(y)));
            Fourier.printTestResult(diff, tol);
        end

        function fftifftTest(tol)
            y = Fourier.randc(2^16, 1);
            diff = max(abs(Fourier.fft(Fourier.ifft(y)) - y));
            Fourier.printTestResult(diff, tol);
        end

        function fft2Test(tol)
            y = Fourier.randc(2^8, 2^8);
            diff = max(max(abs(Fourier.fft(y) - fft2(y))));
            Fourier.printTestResult(diff, tol);
        end

        function ifft2Test(tol)
            y = Fourier.randc(2^8, 2^8);
            diff = max(max(abs(Fourier.ifft(y) - ifft2(y))));
            Fourier.printTestResult(diff, tol);
        end

        function fft2ifft2Test(tol)
            y = Fourier.randc(2^8, 2^8);
            diff = max(max(abs(Fourier.fft(Fourier.ifft(y)) - y)));
            Fourier.printTestResult(diff, tol);
        end
```

```
323
324         function convTest(tol)
325             x = Fourier.randc(2^10, 1);
326             y = Fourier.randc(2^10, 1);
327             diff = max(abs(Fourier.conv(x, y) - cconv(x, y, 2^10)));
328             Fourier.printTestResult(diff, tol);
329         end
330
331         function y = randc(m, n)
332             y = randi(100, m, n) .* rand(m, n) + 1i * randi(100, m, n) .*
                    rand(m, n);
333         end
334
335         function printTestResult(diff, tol)
336             if (diff > tol)
337                 fprintf('[ ] Calculation error: %d\n', diff);
338                 error('Error too high or tolerance too low.');
339             else
340                 fprintf('[X] Calculation error: %d\n', diff);
341             end
342         end
343
344     end
345
346 end
```

Gabor class

The Gabor class contains functionality for the discrete Gabor transform and Gabor filters 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 two methods **Gabor.fgt1**(f, g) and **Gabor.ifgt1**($fTilde$, g), where g is a Gaussian vector generated with the function above.

Two-dimensional Gabor filters may be created by the predefined methods **Gabor.filter2**(a, b, c, x_c, y_c, θ, ω, ν, 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 shall be zero. **Gabor.fgc2**(f, g_w) then calculates the convolution of a square input matrix f and a previously generated square filter matrix g_w.

The method **Gabor.translate**(A, $hShift$, $vShift$) allows the circular shift of any matrix or vector A in horizontal and vertical direction.

Use the **help** function inside MATLAB to get information about all functions and their parameters.

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.
```

```
13          %
14          % The resulting vector is equivalent to the evaluation of
15          %   g(t) = a * exp(-b^2 * (t-tc).^2).
16          %
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 matrix 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 and need
75            % to be of size 2^s*1.
76            %
77            % The returning matrix is fTilde(j, m), where j is time
78            % and m is frequency.
79            %
80            % Params:
81            % - vector f        any complex input vector
82            % - vector g        evaluated Gaussian function
83            %
84            % Returns:
85            % - matrix fTilde   the complex output square matrix
86
87            % Determine dimension of input vectors
88            [n1, n2] = size(f);
89            [n3, n4] = size(g);
90            n = max(size(f));
91
92            % Transpose if y is 1xn vector
93            if n1 == 1
94                f = f.';
95            end
96
97            % Transpose if g is 1xn vector
98            if n3 == 1
99                g = g.';
100           end
101
102           % Show error if necessary
103           if max([n1, n2]) ~= max([n3, n4])
104              error('Length of vectors y and g must coincide.');
105           end
106           if norm(g, 2) == 0
107               error('The Gaussian function g must not be 0.');
108           end
109
110           % Setup matrix
```

```
111             fTilde = zeros(n,n);
112 
113             % Initial indices
114             tInd = 0:n-1;
115 
116             % Loop through all times
117             for j = 0:n-1
118 
119                 % Calculate the time shift modulo n
120                 tShift = mod(tInd - j, n) + 1;
121 
122                 % Define phi(k) = y(k)*g(k-j)
123                 phi = f .* g(tShift);
124 
125                 % For each moment in time, calculate the FFT
126                 fTilde(j+1, :) = Fourier.fft(phi);
127 
128             end
129 
130         end
131 
132         function f = ifgt1(fTilde, g)
133             % Calculates the one-dimensional inverse fast Gabor transform
134             % of a square matrix.
135             %
136             % The size of fTilde needs to be 2^s*2^s, the size of g 2^s*1.
137             %
138             % The returning vector is f(j), where j is time.
139             %
140             % Params:
141             % - matrix fTilde   any complex input square matrix
142             % - vector g        evaluated Gaussian function
143             %
144             % Returns:
145             % - vector y        the complex output vector
146 
147             % Determine dimension of input vectors
148             [n1, n2] = size(fTilde);
149             [n3, n4] = size(g);
150             n = max(size(fTilde));
151 
152             % Transpose if g is 1xn vector
153             if n3 == 1
154                 g = g.';
155             end
156 
157             % Show error if necessary
158             if abs(n1-n2) > 0 || max([n1, n2]) ~= max([n3, n4])
159                error('Length of y and g must coincide.');
```

```
160             end
161             if norm(g, 2) == 0
162                 error('The Gaussian function g must not be 0.');
163             end
164 
165             % Setup resulting vector
166             f = zeros(n, 1);
167 
168             % Initial indices
169             tInd = 0:n-1;
170 
171             % Loop through all time samples (of the integral)
172             for j = 0:n-1
173 
174                 % Calculate the time shift modulo n
175                 tShift = mod(tInd - j, n) + 1;
176 
177                 % Get the frequency information of the current time
178                 phiHat = fTilde(j+1, :).';
179 
180                 % Calculate the IFFT of that vector
181                 phi = Fourier.ifft(phiHat);
182 
183                 % Sum
184                 f = f + g(tShift) .* phi;
185 
186             end
187 
188             % Make sure the resulting vector is normed properly
189             f = 1/norm(g, 2).^2 * f;
190 
191         end
192 
193         function gw = filter2(a, b, c, xc, yc, theta, omega, nu, x, y)
194             % Calculates a matrix with values of a Gabor filter evaluated
195             % at all points in the meshgrid matrices x, y.
196             %
197             % Params:
198             % - number a        max height of the Gaussian
199             % - number b        width/variance of the Gaussian in x
200             % - number c        width/variance of the Gaussian in y
201             % - number xc       center of the Gaussian in x
202             % - number yc       center of the Gaussian in y
203             % - number theta    rotation angle of the Gaussian
204             % - number omega    x-direction of the sinusoid
205             % - number nu       y-direction of the sinusoid
206             % - matrix x        any matrix containing meshgrid samples
207             % - matrix y        any matrix containing meshgrid samples
208             %
```

```
209             % Returns:
210             % - matrix g        the output matrix
211
212             if a <= 0
213                 error('a has to be a real value bigger than 0.');
214             end
215
216             g1 = Gabor.gaussian2(a, b, c, xc, yc, theta, x, y);
217             g2 = exp(1i*omega*(x-xc) + 1i*nu*(y-yc));
218             gw = g1 .* g2;
219
220         end
221
222         function gw = normalizedFilter2(a, b, c, x0, y0, theta, omega, nu,
                x, y)
223             % Calculates a matrix with values of a normalized Gabor filter
224             % evaluated at all points in the meshgrid matrices x, y.
225             %
226             % Params:
227             % - number a        max height of the Gaussian
228             % - number b        width/variance of the Gaussian in x
229             % - number c        width/variance of the Gaussian in y
230             % - number xc       center of the Gaussian in x
231             % - number yc       center of the Gaussian in y
232             % - number theta    rotation angle of the Gaussian
233             % - number omega    x-direction of the sinusoid
234             % - number nu       y-direction of the sinusoid
235             % - matrix x        any matrix containing meshgrid samples
236             % - matrix y        any matrix containing meshgrid samples
237             %
238             % Returns:
239             % - matrix g        the output matrix
240
241             % Calculate the matrix of the filter without normalization
242             gw = Gabor.filter2(a, b, c, x0, y0, theta, omega, nu, x, y);
243
244             % Normalize both real and imaginary part
245             gwReal = real(gw);
246             gwImag = imag(gw);
247
248             % Find the indices with positive and negative values
249             gwRealPosInd = find(gwReal > 0);
250             gwRealNegInd = find(gwReal < 0);
251             gwImagPosInd = find(gwImag > 0);
252             gwImagNegInd = find(gwImag < 0);
253
254             % Get the averages
255             gwRealPosFact = sum(gwReal(gwRealPosInd));
256             gwRealNegFact = abs(sum(gwReal(gwRealNegInd)));
```

```
257             gwImagPosFact = sum(gwImag(gwImagPosInd));
258             gwImagNegFact = abs(sum(gwImag(gwImagNegInd)));
259             sumReal = (gwRealPosFact + gwRealNegFact)/2;
260             sumImag = (gwImagPosFact + gwImagNegFact)/2;
261
262             if (sumReal > 0)
263                 gwRealPosFact = gwRealPosFact / sumReal;
264                 gwRealNegFact = gwRealNegFact / sumReal;
265             end
266             if (sumImag > 0)
267                 gwImagPosFact = gwImagPosFact / sumImag;
268                 gwImagNegFact = gwImagNegFact / sumImag;
269             end
270
271             % Set the new values
272             gwReal(gwRealPosInd) = gwRealNegFact * gwReal(gwRealPosInd);
273             gwReal(gwRealNegInd) = gwRealPosFact * gwReal(gwRealNegInd);
274             gwImag(gwImagPosInd) = gwImagNegFact * gwImag(gwImagPosInd);
275             gwImag(gwImagNegInd) = gwImagPosFact * gwImag(gwImagNegInd);
276
277             % Put back the results
278             gw = gwReal + 1i * gwImag;
279
280         end
281
282         function fStar = fgc2(f, gw)
283             % Calculates the convolution of an input matrix and a Gabor
284             % filter.
285             %
286             % The size of the square matrices f and gw have to coincide
287             % and need to be of 2^s*2^s.
288             %
289             % Params:
290             % - matrix f        any complex input square matrix
291             % - matrix gw       evaluated Gabor filter function
292             %
293             % Returns:
294             % - matrix fTilde   the complex output square matrix
295
296             % Determine dimension of input matrices
297             [n1, n2] = size(f);
298             [n3, n4] = size(gw);
299
300             % Show error if necessary
301             if abs(n1-n2) > 0 || abs(n3-n4) > 0 || abs(n1-n3) > 0
302                error('Matrices f and filter must be square and coincide in
                        size.');
303             end
304
```

```
305             fStar = Fourier.conv(f, gw);
306 
307         end
308 
309         function A = translate(A, hShift, vShift)
310             % Shifts any matrix.
311             %
312             % Params:
313             % - matrix A        any complex input matrix
314             % - number hShift   amount of indices to shift horizontally
315             % - number vShift   amount of indices to shift vertically
316             %
317             % Returns:
318             % - matrix A        the complex output matrix
319 
320             A = circshift(A, [vShift, hShift]);
321 
322         end
323 
324     end
325 
326     methods (Static, Access = private)
327     end
328 
329     %%%%%%%%%%%%%%%%
330     % Test methods %
331     %%%%%%%%%%%%%%%%
332 
333     methods (Static)
334 
335         function test()
336             % Tests the functionality of the class and prints the result.
337 
338             % Setup test variables
339             tol = 10^(-9);
340             amount = 5;
341 
342             fprintf('========================================\n\n');
343             fprintf('Starting Gabor class test...\n\n');
344             fprintf('- Error tolerance: %d\n', tol);
345             fprintf('- Tests per method: %d\n', amount);
346             fprintf('\n');
347 
348             % Test Gaussian1
349             fprintf('------------\nTesting Gaussian1:\n');
350             for k=1:amount
351                 Gabor.gaussian1Test();
352             end
353 
```

```
354             % Test Gaussian2
355             fprintf('------------\nTesting Gaussian2:\n');
356             for k=1:amount
357                 Gabor.gaussian2Test();
358             end
359 
360             % Test FGT
361             fprintf('------------\nTesting FGT:\n');
362             for k=1:amount
363                 Gabor.fgtTest(tol);
364             end
365 
366             % Test FGT/IFGT
367             fprintf('------------\nTesting FGT&IFGT:\n');
368             for k=1:amount
369                 Gabor.fgtifgtTest(tol);
370             end
371 
372             % Test NormalizedFilter
373             fprintf('------------\nTesting NormalizedFilter2:\n');
374             for k=1:amount
375                 Gabor.normalizedfilter2test(tol);
376             end
377 
378             fprintf('------------\nTesting completed with 0 issues.\n');
379 
380         end
381 
382     end
383 
384     methods (Static, Access = private)
385 
386         function gaussian1Test()
387 
388             tol = 100000;
389 
390             a = Gabor.randr(1, 1);
391             b = Gabor.randr(1, 1);
392             tc = Gabor.randr(1, 1);
393             t = linspace(0, 100, tol);
394 
395             g = Gabor.gaussian1(a, b, tc, t);
396 
397             [j, ~] = min(g);
398             [l, m] = max(g);
399 
400             if (j < 0 || l > a)
401                 fprintf('[ ] Gaussian a=%d, b=%d, tc=%d.\n', a, b, tc);
402                 error('Error with parameter a.');
```

```
403             end
404
405             if (abs(t(m)-tc) > 100/tol)
406                 fprintf('[ ] Gaussian a=%d, b=%d, tc=%d.\n', a, b, tc);
407                 error('Error with parameter tc.');
408             end
409
410             fprintf('[X] Gaussian a=%d, b=%d, tc=%d.\n', a, b, tc);
411
412         end
413
414         function gaussian2Test()
415
416             tol = 1000;
417
418             a = Gabor.randr(1, 1);
419             b = 0.5+randi(10);
420             c = 0.5+randi(10);
421             xc = Gabor.randr(1, 1);
422             yc = Gabor.randr(1, 1);
423             theta = Gabor.randr(1, 1);
424             [x, y] = meshgrid(linspace(0, 100, tol+1), linspace(0, 100, tol
                    +1));
425
426             g = Gabor.gaussian2(a, b, c, xc, yc, theta, x, y);
427
428             j = min(min(g));
429             maximum = max(max(g));
430             [l, m] = find(g == maximum);
431
432             if (j < 0 || maximum > a)
433                 fprintf('[ ] Gaussian a=%d, b=%d, c=%d, xc=%d, yc=%d, theta
                        =%d.\n', a, b, c, xc, yc, theta);
434                 error('Error with parameter a.');
435             end
436
437             if (abs(x(l,m)-xc) > 2*100/tol)
438                 fprintf('[ ] Gaussian a=%d, b=%d, c=%d, xc=%d, yc=%d, theta
                        =%d.\n', a, b, c, xc, yc, theta);
439                 error('Error with parameter xc.');
440             end
441
442             if (abs(y(l,m)-yc) > 2*100/tol)
443                 fprintf('[ ] Gaussian a=%d, b=%d, c=%d, xc=%d, yc=%d, theta
                        =%d.\n', a, b, c, xc, yc, theta);
444                 error('Error with parameter yc.');
445             end
446
447             fprintf('[X] Gaussian a=%d, b=%d, c=%d, xc=%d, yc=%d, theta=%d
```

```
                .\n', a, b, c, xc, yc, theta);
448
449         end
450
451         function fgtTest(tol)
452             t = (1:1:2^8)';
453             y = Gabor.randc(2^8, 1);
454             g = Gabor.gaussian1(1, 0, 0, t);
455             yTilde = Gabor.fgt1(y, g);
456             diff = max(abs(yTilde(1, :).' - Fourier.fft(y)));
457             Gabor.printTestResult(diff, tol);
458         end
459
460         function fgtifgtTest(tol)
461             t = (1:1:2^8)';
462             y = Gabor.randc(2^8, 1);
463             g = Gabor.gaussian1(Gabor.randr(1, 1), Gabor.randr(1, 1), (2^8)
                    /2, t);
464             diff = max(abs(Gabor.ifgt1(Gabor.fgt1(y, g), g) - y));
465             Gabor.printTestResult(diff, tol);
466         end
467
468         function normalizedfilter2test(tol)
469
470             h = 1000;
471
472             a = Gabor.randr(1, 1);
473             b = Gabor.randr(1, 1);
474             c = Gabor.randr(1, 1);
475             xc = Gabor.randr(1, 1);
476             yc = Gabor.randr(1, 1);
477             theta = Gabor.randr(1, 1);
478             w0 = Gabor.randr(1, 1);
479             w1 = Gabor.randr(1, 1);
480             [x, y] = meshgrid(linspace(0, 100, h), linspace(0, 100, h));
481
482             gwc = Gabor.normalizedFilter2(a, b, c, xc, yc, theta, w0, w1, x
                    , y);
483
484             diff = sum(sum(real(gwc))) + sum(sum(imag(gwc)));
485
486             if (diff > tol)
487                 fprintf('[ ] Normalization error: %d\n', diff);
488                 error('Error too high or tolerance too low.');
489             else
490                 fprintf('[X] Normalization error: %d\n', diff);
491             end
492
493         end
```

```
494
495         function y = randr(m, n)
496             y = randi(100, m, n) .* rand(m, n);
497         end
498
499         function y = randc(m, n)
500             y = randi(100, m, n) .* rand(m, n) + 1i * randi(100, m, n) .*
                    rand(m, n);
501         end
502
503         function printTestResult(diff, tol)
504             if (diff > tol)
505                 fprintf('[ ] Calculation error: %d\n', diff);
506                 error('Error too high or tolerance too low.');
507             else
508                 fprintf('[X] Calculation error: %d\n', diff);
509             end
510         end
511
512     end
513
514 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, 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, 1985.

[DS15] Lokenath Debnath and Firdous Ahmad Shah. *Wavelet Transforms 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, 1987.

[Pet95] Nikolay Petkov. Biologically Motivated Computationally Intensive Approaches to Image Pattern Recognition. *Future Generation Computer Systems*, 11(4-5):451–465, 1995.