Time-frequency analysis and its applications in denoising

Oddvar Christiansen

Department of Informatics
University of Bergen
Norway

Cand. Scient. Thesis
December 2002

Acknowledgments

First I would like to thank my excellent supervisor professor Hans Munthe-Kaas for his teaching, support and guidance for two years.

I would also like to thank Ingrid Honve for her support, and for reading the thesis.

Finally, thanks to all my friends for five interesting years at the University of Bergen.

Contents

Acknowledgments **3**

1 Introduction **7**
1.1 Fourier transform 10
1.1.1 Continuous Fourier transform 10
1.1.2 Discrete Fourier transform 11
1.1.3 Discrete 2d Fourier transform 11
1.2 Zak transform 11

2 The Gabor representation **13**
2.1 The Gabor uncertainty principle 13
2.2 Continuous Gabor representation 17
2.3 Discrete Gabor representation 18
2.3.1 Calculating the window function 20
2.3.2 Calculating the Gabor transform 23
2.3.3 Calculating the inverse Gabor transform 24
2.4 Oversampled Gabor representation 25
2.4.1 Calculating the Gabor transform 25
2.4.2 Calculating the inverse Gabor transform 28
2.4.3 Calculating the window function 28

3 Algorithms **31**
3.1 Fast Fourier transform and its inverse 31
3.2 Notation 32
3.3 Critical sampled Gabor representation 36
3.3.1 The 2d Fourier transform and its inverse 36
3.3.2 The Zak transform and its inverse 37
3.3.3 The periodization of the elementary function 39
3.3.4 The Gabor transform and its inverse 41
3.4 Oversampled Gabor representation 43
3.4.1 Left rotation of a vector 43
3.4.2 The Zak transform of the elementary function 44
3.4.3 The Zak transform of the window function 46

3.4.4 The oversampled Gabor transform and its inverse 47

4 Denoising by thresholding **51**
4.1 Introduction 51
4.2 Thresholding 52
4.2.1 Hard thresholding 52
4.2.2 Soft thresholding 53
4.2.3 Thresholding Gabor coefficients 54
4.2.4 Example 54
4.3 Selecting the threshold 58
4.3.1 Statistical method 58
4.3.2 Threshold selection by SURE 63
4.3.3 Unknown noise level 66
4.4 Simulation experiments 66
4.4.1 Bumps 67
4.4.2 HeaviSine 72
4.4.3 Doppler 77
4.4.4 Blocks 82
4.4.5 Quadchirp 87
4.4.6 Mishmash 92
4.4.7 Conclusion 97
4.5 Gabor versus Wavelet denoising 97
4.6 Denoising music 104

5 Concluding remarks and further work **107**

A **109**
A.1 Proof of theorem 2.3.2 109

Bibliography **111**

Chapter 1

Introduction

Time-frequency analysis plays a central role in signal analysis. Already long time ago it was recognized that a global Fourier transform of a signal is of little value to analyze the frequency spectrum it. Transient signals evolving in time in an unpredictable way necessitate the notion of frequency analysis that is local in time. Why this is so can easily be illustrated with an example. Take a recording of someone playing the piano. If we represent this piece of music as a function of time, see Figure 1.1, we are able to see the transition from one note to the next, but we get little insight about which notes are in play. On the other hand the Fourier representation, see Figure 1.2, gives us a clear indication about the notes played, but there is no information about the duration and the order of sequence of the notes. Neither of the representations are very satisfactory. We will prefer a representation which is local in both time and frequency, like music notation, which tells the musician which note to play at a given moment. This is exactly what we get if we use a time-frequency representation, see Figure 1.3. We are now able to see which notes are in play and the order of sequence they are played.

One approach to obtain a local time-frequency analysis, suggested by various scientists, is to cut the signal into slices and do a Fourier transform on these slices. But the functions obtained by this crude segmentation are not periodic. The jump at the boundaries is interpreted as a discontinuity, resulting in large Fourier coefficients at high frequencies.

In 1946 Dennis Gabor introduced the Gabor expansion or the inverse Gabor transform [15] to perform simultaneous time-frequency analysis of signals. He introduced a set of basis functions consisting of Gaussian windows modulated by complex exponentials. It was initially seen as a purely theoretical tool since there were no effective means by which it could be computed. Gabors work went almost unnoticed until the early 80's, when the work of Bastiaans and Janssen refreshed the interest of mathematicians and engineers in Gabor analysis. During the 90's the development of time-frequency analysis has benefited from the rise of wavelet theory, and for some time both theories grew in parallel. Today, time-frequency analysis presents itself as an interdisciplinary area of research.

The applications of time-frequency analysis are many. On the applied side time-frequency analysis deals with problems in signal analysis, communication theory and image processing. In our thesis we develop a method using time-frequency analysis for the removal of white noise from signals. Since the beginning of the 90's there has been considerably interest in the use of wavelet transforms for the removal of noise from signals and images. The wavelet transform decompose the signal using translation and dilation of a single basis function, and is a time-scale analysis, not a time-frequency analysis. We will not treat wavelets in our thesis, and the interested reader is refereed to the excellent textbook of Strange and Nguyen [23].

The most employed method for noise removal has been the "WaveShrink", developed by Donoho and Johnstone [10] [12] [13]. This method uses a transform-based filtering, working in three steps:

- Transform the noisy data into the wavelet domain, i.e. time-scale domain.
- Shrink the resulting coefficients, thereby suppressing those coefficients containing noise.
- Transform back into the original domain.

In our thesis we develop a similar method, transforming the signal into the time-frequency domain instead of the time-scale domain. To transform the signal into the time-frequency domain we use the Gabor transform. We show that for some signals our method performs better than the "WaveShrink".

The thesis is organized in the following way: In the remaining part of this chapter we give a short overview of the Fourier transform and the Zak transform. In Chapter 2 we present the mathematical techniques for the Gabor transform and the inverse Gabor transform. We treat both the critical sampled and the oversampled case. In Chapter 3 we develop algorithms for the discrete Gabor transform and the inverse discrete Gabor transform. We treat both the critical sampled and the oversampled case and analyze the computational cost of the algorithms. The algorithms are all based on the mathematical techniques presented in Chapter 2. In Chapter 4 we develop a method using time-frequency analysis for the removal of white noise from signals. We show that for some signals this method performs better than the "WaveShrink".

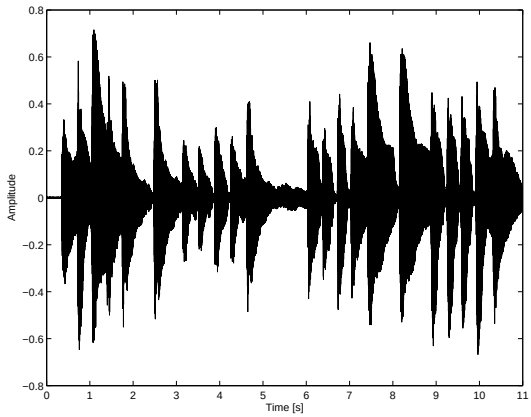

Figure 1.1: The recording of someone playing the piano represented as a function of time. We are able to see the transition from one note to the next, but we get little insight about which notes are in play.

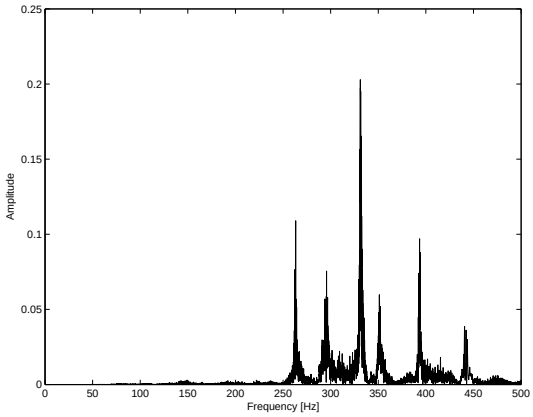

Figure 1.2: The Fourier representation of someone playing the piano. We get clear indication about the notes played, but there is no information about the duration and the order of sequence of the notes.

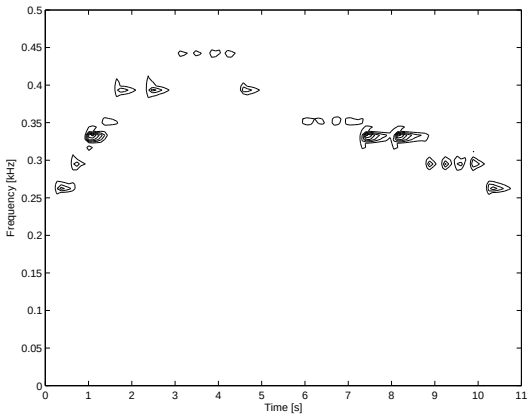

Figure 1.3: The time-frequency representation of someone playing the piano. We are able to see which notes are in play and the order of sequence they are played.

1.1 Fourier transform

It is sometimes convenient to describe a signal x(t) say, not in the time domain, but in the frequency domain by means of its frequency spectrum, i.e. the Fourier transform of x(t). This frequency spectrum shows us the global distribution of the energy of the signal as a function of frequency.

The Fourier transform originates from the work of the French mathematician J.B. J. Fourier in the early nineteenth century. While many contributed to the field, Fourier is honored for his mathematical discoveries and insight into the practical usefulness of the techniques.

1.1.1 Continuous Fourier transform

For a continuous function of one variable $x(t)$, the Fourier transform will be defined as

$$\overline{x}(w) = \int_{-\infty}^{\infty} x(t) e^{-jwt} dt \tag{1.1}$$

and the inverse transform as

$$x(t) = \frac{1}{2\pi} \int_{-\infty}^{\infty} \overline{x}(w) e^{jwt} dw, \tag{1.2}$$

where j is the square root of -1 and e denotes the exponential function

$$e^{j\theta} = \cos(\theta) + j \sin(\theta). \tag{1.3}$$

1.1.2 Discrete Fourier transform

To distinguish the discrete case from the continuous case, we will use square brackets [ ] to denote a discrete signal, whereas we use curved brackets () to denote a continuous signal. For a discrete function of one variable, $x[n]$, the Fourier transform will be defined as

$$\overline{x}[k] = \sum_{n=0}^{N-1} x[n] e^{-j2\pi kn/N} \tag{1.4}$$

and the inverse transform as

$$x[n] = \frac{1}{N} \sum_{k=0}^{N-1} \overline{x}[k] e^{j2\pi kn/N}. \tag{1.5}$$

1.1.3 Discrete 2d Fourier transform

We can extend the 1 dimensional Fourier transform to 2 dimensions. This can be done both in the discrete and continuous case, but we will restrict ourselves to the discrete case her.

The discrete 2d Fourier transform of the array $a_{m,k}$ is given by

$$\overline{a}[n,l;N,M] = \sum_{m=0}^{M-1} \sum_{k=0}^{N-1} a_{m,k} e^{-j(2\pi ml/M - 2\pi kn/N)}, \tag{1.6}$$

which is periodic in the discrete index n with period N and periodic in the discrete index l with period M.

The inverse discrete 2d Fourier transform of the array $\overline{a}[n,l;N,M]$ is given by

$$a_{m,k} = \frac{1}{MN} \sum_{n=0}^{N-1} \sum_{l=0}^{M-1} \overline{a}[n,l;N,M] e^{j(2\pi ml/M - 2\pi kn/N)} \tag{1.7}$$

which is periodic in the discrete index m with period M and periodic in the discrete index k with period N.

1.2 Zak transform

The Zak transform provides a way to represent a time function by a 2 dimensional time-frequency function. The transform exists in both the discrete and continuous case, but we will only treat the discrete case here. J. Zak was the first who systematically studied this transform in connection with solid state physics [30] [31] [32]. Some of its properties were known before Zak's work, however. The same transform is called Weil-Brezin map and it is claimed that the transform was already known to Gauss. The Zak transform has interesting applications to signal

analysis, and later we will show how the Zak transform can be applied to the Gabor representation.

The discrete Zak transform of the discrete array $x[n]$ is given by

$$\tilde{x}[n, l; N, M] = \sum_{m=-\infty}^{\infty} x[n + mN] e^{-j2\pi ml/M}, \tag{1.8}$$

which is periodic in the discrete-frequency variable l with period M.

If we define $X[n]$ as the summation of $x[n]$ over distances rMN, where $r \in \mathbf{Z}$, i.e.

$$X[n] = \sum_{r=-\infty}^{\infty} x[n + rMN], \tag{1.9}$$

we can rewrite the discrete Zak transform in the following way

$$\tilde{x}[n, l; N, M] = \sum_{m=0}^{M-1} X[n + mN] e^{-j2\pi ml/M}, \tag{1.10}$$

The inverse discrete Zak transform of the array $\tilde{x}[n, l; N, M]$ is given by

$$X[n + mN] = \frac{1}{M} \sum_{l=0}^{M-1} \tilde{x}[n, l; N, M] e^{j2\pi ml/M}, \tag{1.11}$$

which is a periodic sequence with period M.

Chapter 2

The Gabor representation

Introduction

The Gabor expansion or the inverse Gabor transform was introduced in 1946 by Dennis Gabor [15] to perform simultaneous time-frequency analysis of signals. He argued that the optimal representation for a signal is one which combines frequency and locality information. For this purpose he introduced a set of basis functions consisting of Gaussian windows modulated by complex exponentials. It was initially seen as a purely theoretical tool since there were no effective means by which it could be computed. During the 90's, however, there emerged several fast algorithms for the computation of the Gabor transform and its inverse.

In this chapter we present the Gabor uncertainty principle and the mathematical techniques for the Gabor transform and the inverse Gabor transform. We treat both the critical sampled and the oversampled case. The results in this chapter are mainly influenced by the work of Bastiaans [3] [4], but the work of Qian and Chen [19] and Wexler and Raz [28] are also used. However, we have tried to present the material in our own way.

2.1 The Gabor uncertainty principle

When doing a time-frequency analysis of a signal it would be desirable to have no limits on the resolution in the time and frequency domain. However, this is not possible because the resolution in the time and frequency domain are reciprocal. When increasing the resolution in the time domain, the resolution in the frequency domain reduces. This is what Gabor proved in his uncertainty principle [15]. He proved this by applying to arbitrary signals the same mathematical apparatus as used in the Heisenberg-Weyl derivation of the uncertainty principle in quantum mechanics. We will not present the complete proof here, our intention is to build intuition, so we present a couple of informal derivations [18].

Suppose we are trying to measure the frequency of a tone. Intuitively, the longer the sample we take, the more accurate our measurement will be. This sug-

gests that the error in measuring the frequency, Δf, is inversely related to the duration of the measurement, Δt. This intuition can be made a little more precise by considering a very basic kind of frequency measurement. Suppose we have a device that counts every time our signal reaches a maximum; then the number of maxima in an interval of time Δt will be the average frequency during that interval. How long must Δt be in order to guarantee that we can distinguish frequencies differing by Δf ? This will occur when counts for f and $f + \Delta f$ are guaranteed to differ by at least one. That is,

$$(f + \Delta f)\Delta t - f\Delta t \geq 1$$

or

$$\Delta f \Delta t \geq 1. \tag{2.1}$$

This is the basic Gabor uncertainty principle, it means that the product of the uncertainties in frequency and time must exceed a fixed constant, and so the accuracy with which one of them can be measured limits the best possible accuracy with which the other can be measured. Heisenbergs uncertainty principle is a simple corollary of (2.1), since according to quantum mechanics the energy of a photon is proportional to its frequency, $E = hf$. Multiplying both sides of (2.1) by h (Plancks constant) yields

$$\Delta E \Delta t \geq h,$$

which is one form of Heisenbergs principle. Of course, Heisenberg derived his principle first. Gabors accomplishment was to show that the same mathematical derivation applied to communication systems.

A more formal derivation of Gabors uncertainty principle is based on the observation that the "spread" of a signal and its Fourier transform are inversely proportional, see Figure 2.1. To accomplish this we must specify a way of measuring the spread of functions. Although there are many ways to define these measures, we define the nominal duration of a signal x to be the duration of a rectangular pulse of the same area and amplitude at the origin as the signal, see Figure 2.2. Thus the nominal duration Δt is defined by the equation

$$\Delta t |x(0)| = \int_{-\infty}^{\infty} |x(t)|\, dt.$$

Similar, the nominal bandwidth of the Fourier transform of x, $\overline{x} = \mathcal{F}(x)$, is defined

$$\Delta t |\overline{x}(0)| = \int_{-\infty}^{\infty} |\overline{x}(t)|\, dt.$$

Next write $|\overline{x}(0)|$ as the Fourier transform of x evaluated at $w = 0$

$$|\overline{x}(0)| = \left| \int_{-\infty}^{\infty} x(t) e^{-jwt} dt \Big|_{w=0} \right| = \left| \int_{-\infty}^{\infty} x(t) dt \right| \leq \int_{-\infty}^{\infty} |x(t)|\, dt = \Delta t |x(0)|.$$

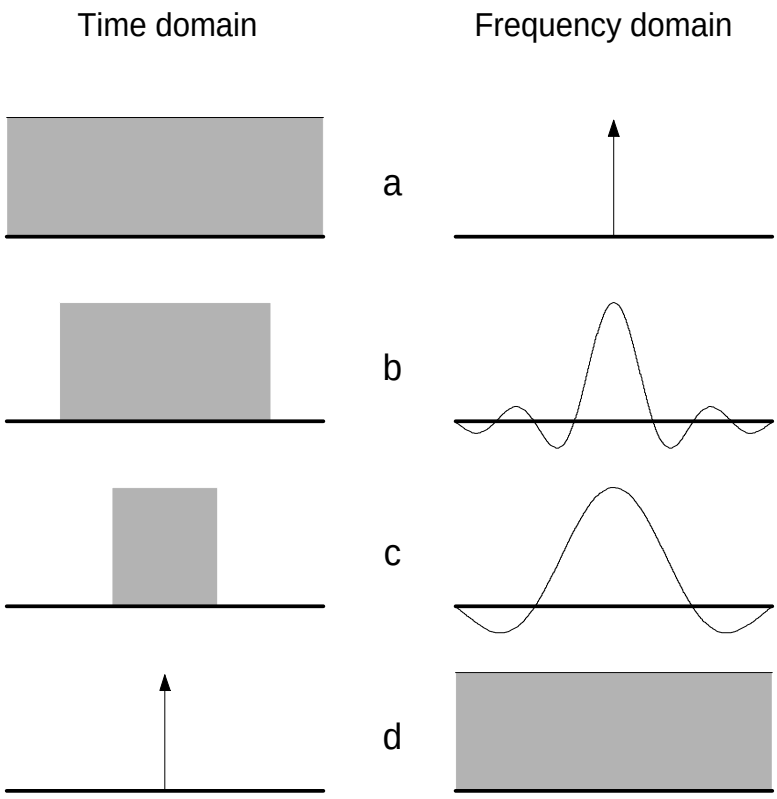

Figure 2.1: The “spread” of a signal and its Fourier transform are inversely proportional. (a) A constant function in the time domain corresponds to a unit impulse in the frequency domain. (b,c) As the width of a pulse in the time domain decreases, its spectrum in the frequency domain spreads. (d) A unit impulse in the time domain has a spectrum which is a constant function.

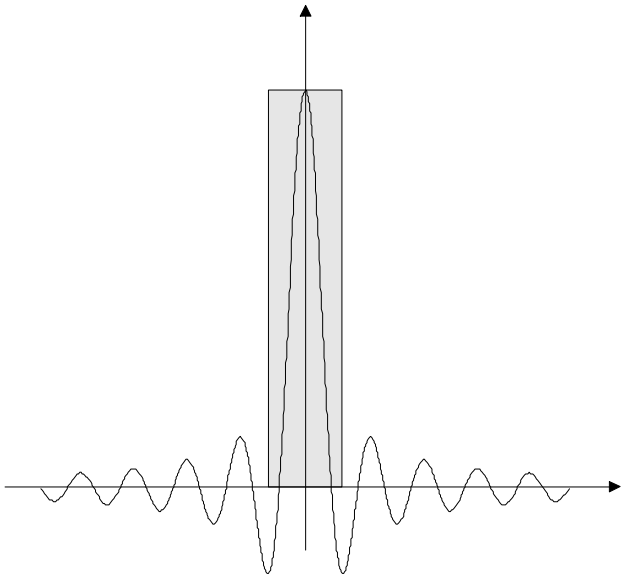

Figure 2.2: The nominal bandwidth of a spectrum is the width of a rectangular pulse (shaded) that has height equal to the amplitude of the spectrum at the origin, and that has the same area as the absolute value of the spectrum.

Therefore

$$\Delta t \geq \frac{|\overline{x}(0)|}{|x(0)|}. \tag{2.2}$$

Similarly, applying the inverse Fourier transform

$$|x(0)| = \left| \int_{-\infty}^{\infty} \overline{x}(t) e^{jwt} dw \Big|_{t=0} \right| = \left| \int_{-\infty}^{\infty} \overline{x}(t) dw \right| \leq \int_{-\infty}^{\infty} |\overline{x}(t)| \, dw = \Delta f |\overline{x}(0)|.$$

Therefore

$$\Delta f \geq \frac{|x(0)|}{|\overline{x}(0)|}. \tag{2.3}$$

Multiplying (2.2) and (2.3) we get the general Gabor uncertainty principle

$$\Delta f \Delta t \geq \frac{|x(0)|}{|\overline{x}(0)|} \frac{|\overline{x}(0)|}{|x(0)|} = 1. \tag{2.4}$$

Thus we see that the nominal duration and the nominal bandwidth are reciprocals of each other. In other words, there is a minimum possible simultaneous localization of the signal in the time and frequency domains. We can decrease Δt, and so localize the signal better in the time domain, or decrease Δf, and so localize it better in the frequency domain, but we can not localize it arbitrarily well in both domains simultaneously.

2.2 Continuous Gabor representation

Consider an elementary signal $g(t)$. The classical, and also Gabors original choice, is a Gaussian, but the signal may have a rather arbitrary shape. The Gaussian, however, have the advantage that each shifted and modulated version occupies the smallest possible area in the time-frequency domain [3]. For this reason we will choose the Gaussian shaped elementary signal

$$g(t) = 2^{1/4} e^{-\pi(t/T)^2}, \tag{2.5}$$

where the factor $2^{1/4}$ is added to normalize the energy to unity.

The Gabor expansion or the inverse Gabor transform of a continuous function is given by

$$x(t) = \sum_{m=-\infty}^{\infty} \sum_{k=-\infty}^{\infty} a_{m,k} g_{m,k}(t), \tag{2.6}$$

where

$$g_{m,k}(t) = g(t - mT) e^{j\Omega kt}$$

and the time shift T and the frequency shift Ω satisfy the relationship $\Omega T = 2\pi$, and where m and k may take all integer values. What we have done here is to decompose the signal $x(t)$ into a superposition of properly shifted (over discrete distances mT) and modulated (with discrete frequencies $k\Omega$) versions of the elementary signal $g(t)$.

The problem with this decomposition is that the basis functions $g_{m,k}$ produced from the shifted and modulated versions of the elementary signal $g(t)$ are not orthonormal, i.e. their inner product are not zero. This makes the calculations of the coefficients $a_{m,k}$ difficult. To overcome this problem we try to find a bi-orthonormal basis, i.e. we try to find a bi-orthonormal window function, denoted $w(t)$, such that

$$a_{m,k} = < x, w_{m,k} > = \int x(t) w^*_{m,k}(t) dt, \tag{2.7}$$

where

$$w_{m,k}(t) = w(t - mT) e^{j\Omega kt}$$

and the asterisk denotes complex conjugation. This is known as the Gabor transform.

Lemma 2.2.1. *For (2.7) to hold we must require the bi-orthonormality condition*

$$\int g_{p,q}(t) w^*_{m,k}(t) dt = \delta[p - m]\delta[q - k], \tag{2.8}$$

where

$$\delta[p - m] = \begin{cases} 1 & p{=}m, \\ 0 & p \neq m \end{cases}$$

Proof. Substituting (2.6) into (2.7) leads to

$$a_{m,k} = \int \Big(\sum_{p=-\infty}^{\infty} \sum_{q=-\infty}^{\infty} a_{p,q} g_{p,q}(t) \Big) w^*_{m,k}(t) dt.$$

Rearranging factors we get

$$a_{m,k} = \sum_{p=-\infty}^{\infty} \sum_{q=-\infty}^{\infty} a_{p,q} \int g_{p,q}(t) w^*_{m,k}(t) dt.$$

For the equality to hold we must require the bi-orthonormality condition (2.8). □

This bi-orthonormality condition guarantees that if we start with an array of Gabor coefficients $a_{m,k}$, construct a signal via (2.6) and subsequently substitute this signal into (2.7), we end up with the original coefficients array. It can be shown [3] that it also guarantees that if we start with a certain signal $x(t)$, construct its Gabor coefficients $a_{m,k}$ via (2.7) and subsequently substitute these coefficients into (2.6), we end up with the original signal. Hence the equations (2.6) and (1.6) form a transform pair.

Though of great mathematical interest, the continuous Gabor transform and its inverse are not suited for numerical computations. Computers require a finite number of time steps, and can not work with an infinite time space. To overcome this problem we need the discrete versions of the transforms. For those interested in the continuous case we refer to [3] for further reading.

2.3 Discrete Gabor representation

As already mentioned, to be able to use computers in the computation of the Gabor transform (2.7) and the inverse Gabor transform (2.6) we need discrete versions of the transforms.

Let us consider a discrete signal $x[n]$, on the analogy of the inverse Gabor transform for continuous signals (2.6) we define the inverse Gabor transform for discrete signals in the following way

$$x[n] = \sum_{m=-\infty}^{\infty} \sum_{k=0}^{N-1} a_{m,k} g_{m,k}[n], \tag{2.9}$$

where

$$g_{m,k}[n] = g(n - mN) e^{j\Theta kn}$$

and the time shift N and the frequency shift $\Theta = 2\pi/N$ are respective counterparts of T and $\Omega = 2\pi/T$ in the continuous case. It should be mentioned that $\Theta = 2\pi/N$ was Gabors originally choice and is called critical sampling. It is also possible to choose $\Theta < 2\pi/N$, this is called oversampling and possesses several advantages. We treat oversampling in Section 2.4.

The discrete elementary signal $g[n]$ is the sampled continuous elementary signal, with sampling distance T/N, i.e.

$$g[n] = g\Big(n\frac{T}{N}\Big) = 2^{1/4}e^{-\pi(n/N)^2} \quad (N \text{ odd}) \tag{2.10}$$

and

$$g[n] = g\Big([n+\tfrac{1}{2}]\frac{T}{N}\Big) = 2^{1/4}e^{-\pi([n+\frac{1}{2}]/N)^2} \quad (N \text{ even}). \tag{2.11}$$

As in the continuous case the Gabor coefficients $a_{m,k}$ can be evaluated if we can find a bi-orthonormal window function, denoted $w[n]$, such that

$$a_{m,k} = \sum_{n=-\infty}^{\infty} x[n]w^*_{m,k}[n], \tag{2.12}$$

where

$$w_{m,k}[n] = w(n - mN)e^{j\Theta kn}. \tag{2.13}$$

This is known as the Gabor transform for discrete signals. In the critically sampled form ($\Theta = 2\pi/N$) the Gabor transform for discrete signals is an array that is periodic in the frequency variable k with period N. This periodicity in the Gabor transform results from the discrete nature of the signal.

As in the continuous case we must require the following condition:

Lemma 2.3.1. *For (2.12) to hold we must require the bi-orthonormality condition*

$$\sum_{n=-\infty}^{\infty} g_{p,q}[n]w^*_{m,k}[n] = \delta[p-m]\delta[q-k], \tag{2.14}$$

where

$$\delta[p-m] = \begin{cases} 1 & p{=}m, \\ 0 & p \neq m \end{cases}.$$

Proof. Substituting (2.9) into (2.12) leads to

$$a_{m,k} = \sum_{n=-\infty}^{\infty}\Big(\sum_{p=-\infty}^{\infty}\sum_{q=0}^{N-1} a_{p,q}g_{p,q}[n]\Big)w^*_{m,k}[n].$$

Rearranging factors we get

$$a_{m,k} = \sum_{p=-\infty}^{\infty}\sum_{q=0}^{N-1} a_{p,q}\sum_{n=-\infty}^{\infty} g_{p,q}[n]w^*_{m,k}[n].$$

For the equality to hold we must require the bi-orthonormality condition (2.14). □

Thus by the same argumentation as in the continuous case the Gabor transform (2.12) and the inverse Gabor transform (2.9) form a transform pair.

The problem with this definition is the infinite sum over m in (2.9) and the infinite sum over n in (2.12). In order to have a proper discrete Gabor representation we need to truncate the summations. This can be achieved exactly only for finite signals. Thus in what follows we assume that the signal $x[n]$ is of finite length. If the signal is infinite we can split the signal into parts and treat each part separately. The length of the signal has to be a multiple of the time shift N, i.e. the length must be MN, where M is a positive integer. This can always be achieved by padding the end of the signal with zeros. In addition we define $W[n]$ and $G[n]$ as the summation of $w[n]$ and $g[n]$ respectively over distances rMN, where $r \in \mathbf{Z}$, i.e.

$$W[n] = \sum_{r=-\infty}^{\infty} w[n + rMN] \tag{2.15}$$

and

$$G[n] = \sum_{r=-\infty}^{\infty} g[n + rMN]. \tag{2.16}$$

$W[n]$ and $G[n]$ are periodic with period MN, and are called the periodization of $w[n]$ and $g[n]$ respectively. Using this we can make fully periodized versions of the Gabor representation [28]. The discrete inverse Gabor transform takes the form

$$x[n] = \sum_{m=0}^{M-1} \sum_{k=0}^{N-1} a_{m,k} G_{m,k}[n], \tag{2.17}$$

where

$$G_{m,k}[n] = G[n - mN] e^{j2\pi kn/N} \tag{2.18}$$

and the discrete Gabor transform takes the form

$$a_{m,k} = \sum_{n=0}^{MN-1} x[n] W^*_{m,k}[n], \tag{2.19}$$

where

$$W_{m,k}[n] = W[n - mN] e^{j2\pi kn/N}. \tag{2.20}$$

In the sequel we will discuss only discrete versions of the transforms. Hence Zak, Gabor and Fourier transforms will henceforth refer to the discrete versions.

2.3.1 Calculating the window function

It is possible to calculate the window function directly from the bi-orthonormal equation system (2.14). This is a deterministic equation system, and hence can be solved with direct methods [19][28]. However when the length of the window

function is large, i.e. when the length of $x[n]$ is large, this equation system will be huge and very time consuming to solve. Instead we use a different approach. Based on [3] we state the following theorem, giving us the Zak transform of the window function.

Theorem 2.3.1. *We have the following relation between the Zak transform of the elementary signal $g[n]$ and the Zak transform of the window function $w[n]$:*

$$\tilde{w}[n,l;N,M] = \frac{1}{N\tilde{g}^*[n,l;N,M]}. \tag{2.21}$$

Proof. We start with the bi-orthonormality condition (2.14) and choose $p = q = 0$, this gives

$$\delta[-m]\delta[-k] = \sum_{n'=-\infty}^{\infty} g[n']w^*_{m,k}[n'].$$

Multiplying both sides with $e^{-j[2\pi ml/M - 2\pi kn/N]}$ and taking the sum over $m \in \{0,...,M-1\}$ and $k \in \{0,...,N-1\}$, i.e. taking the 2 dimensional Fourier transform of both sides [cf. (1.6)], we get

$$\sum_{m=0}^{M-1}\sum_{k=0}^{N-1} \delta[-m]\delta[-k]e^{-j[2\pi ml/M-2\pi kn/N]} = \sum_{m=0}^{M-1}\sum_{k=0}^{N-1}\sum_{n'=-\infty}^{\infty} g[n']w^*_{m,k}[n']e^{-j[2\pi ml/M-2\pi kn/N]}. \tag{2.22}$$

The product of the delta functions $\delta[-m]\delta[-k]$ is 1 only in the point $m = k = 0$, and zero elsewhere. In this point the exponential is 1, resulting in that the left side of (2.22) equals 1. Using this and (2.13) we get

$$1 = \sum_{m=0}^{M-1}\sum_{k=0}^{N-1}\sum_{n'=-\infty}^{\infty} g[n']w^*[n'-mN]e^{-j2\pi kn'/N}e^{-j[2\pi ml/M-2\pi kn/N]}.$$

Rearranging factors we get

$$1 = \sum_{m=0}^{M-1}\sum_{n'=-\infty}^{\infty} g[n']w^*[n'-mN]\Big(\sum_{k=0}^{N-1} e^{-j2\pi k(n'-n)/N}\Big)e^{-j2\pi ml/M}.$$

The complex exponential is a circular function, thus each sum of length N equals 0 unless $n' = n + pN$, where $p \in \mathbf{Z}$, i.e.

$$\sum_{k=0}^{N-1} e^{-j2\pi k(n'-n)/N} = \begin{cases} 0 & n' \neq n+pN, \\ N & n' = n+pN. \end{cases}$$

Thus making the substitution $n' = n + pN$ we get

$$1 = N \sum_{m=0}^{M-1} \sum_{p=-\infty}^{\infty} g[n + pN] w^*[n + (p - m)N] e^{-j2\pi ml/M}.$$

Multiplying the right side with $e^{j2\pi pl/M} e^{-j2\pi pl/M} = 1$ and taking a final rearrangement we find that

$$1 = N \sum_{p=-\infty}^{\infty} g[n + pN] e^{-j2\pi pl/M} \sum_{m=0}^{M-1} w^*[n + (p - m)N] e^{j2\pi(p-m)l/M}.$$

Since the sum over p is infinite we can make the substitution $m' = p - m$, this gives

$$1 = N \Big[\sum_{p=-\infty}^{\infty} g[n + pN] e^{-j2\pi pl/M} \Big] \Big[\sum_{m'=-\infty}^{\infty} w[n + m'N] e^{-j2\pi m'l/M} \Big]^*.$$

Comparing with (1.8) we see that the first summation is the Zak transform of the elementary signal $g[n]$ and the second is the Zak transform of the window function $w[n]$, i.e.

$$1 = N \tilde{g}[n, l; N, M] \tilde{w}^*[n, l; N, M]$$

Solving for $\tilde{w}^*[n, l; N, M]$ and transposing both sides we get (2.21), which completes the proof. □

We are now able to calculate the periodized window function $W[n]$ in an easy way:

- Use (2.16) to calculate the MN periodic elementary function $G[n]$, $n \in \{0, ..., N - 1\}$.
- Use (1.10) to calculate the Zak transform $\tilde{g}[n, l; N, M]$, $n \in \{0, ..., N - 1\}$ and $l \in \{0, ..., M - 1\}$, of the elementary function.
- Use (2.21) to calculate the Zak transform $\tilde{w}[n, l; N, M]$, $n \in \{0, ..., N - 1\}$ and $l \in \{0, ..., M - 1\}$, of the window function.
- Use the inverse Zak transform (1.11) to calculate the periodized window function $W[n]$.

The algorithms are treated in detail in Chapter 3. A problem with this approach should be mentioned. If the Zak transform of $g[n]$ has zeros, the relation (2.21) has no solution. On the discrete grid we have no zeroes, but the method is numerically unstable. The solution to this problem is oversampling, which we present in the next section.

2.3.2 Calculating the Gabor transform

Having found the periodized window function $W[n]$ we can easily calculate $a_{m,k}$ using (2.19), however the computational cost of this is huge. Each coefficient requires $\mathcal{O}(MN)$ operations, resulting in a total cost of $\mathcal{O}(M^2N^2)$ operations to calculate all the MN coefficients. Remember that MN is the the length of the vector $x[n]$, and thus for large vectors this will be extremely time consuming.

One approach to reduce the computational cost of (2.19) is to use sampled FFT [19]. Defining

$$R_m[n] = x[n]W^*[n - mN]$$

we can rewrite (2.19) in the following way:

$$\begin{aligned} a_{m,k} &= \sum_{n=0}^{MN-1} x[n]W^*[n-mN]e^{-j2\pi kn/N} \\ &= \sum_{n=0}^{MN-1} R_m[n]e^{-j2\pi kn/N} \\ &= \sum_{p=0}^{M-1}\sum_{n'=0}^{N-1} R_m[pN+n']e^{-j2\pi k(pN+n')/N} \\ &= \sum_{p=0}^{M-1}\sum_{n'=0}^{N-1} R_m[pN+n']e^{-j2\pi kn'/N}. \end{aligned}$$

The second summation is a N-point FFT, and thus we can calculate all the coefficients using M^2 N-point FFTs, having a total cost of $\mathcal{O}(M^2N\log N)$ operations. This is the easiest algorithm to implement, but if the length of the vector $x[n]$ is large even this is a very slow algorithm.

To calculate the discrete Gabor transform for large vectors we need to reduce the computational cost even more. The next algorithm use the following important theorem based on [3]:

Theorem 2.3.2. *The Fourier transform of the coefficients $a_{m,k}$ can be found as the scaled product of the Zak transform of the signal $x[n]$ and the Zak transform of the window function $w[n]$, i.e.*

$$\bar{a}[n,l;N,M] = N\tilde{x}[n,l;N,M]\tilde{w}^*[n,l;N,M]. \tag{2.23}$$

Proof. The proof of this theorem can be found in the appendix. □

If we combine this theorem and theorem 2.3.1 we get the following relation

$$\bar{a}[n,l;N,M] = \frac{\tilde{x}[n,l;N,M]}{\tilde{g}[n,l;N,M]} \tag{2.24}$$

Using this relation we can find the Gabor coefficients $a_{m,k}$ in the following way:

- Use (2.16) to calculate the MN periodic elementary function $G[n]$, $n \in \{0, ..., N-1\}$.
- Use (1.10) to calculate the Zak transform $\tilde{g}[n, l; N, M]$, $n \in \{0, ..., N-1\}$ and $l \in \{0, ..., M-1\}$, of the elementary function.
- Use (1.10) to calculate the Zak transform $\tilde{x}[n, l; N, M]$, $n \in \{0, ..., N-1\}$ and $l \in \{0, ..., M-1\}$, of the signal $x[n]$.
- Use (2.24) to calculate the Fourier transform $\bar{a}[n, l; N, M]$, $n \in \{0, ..., N-1\}$ and $l \in \{0, ..., M-1\}$, of the coefficients $a_{m,k}$.
- Use the inverse Fourier transform (1.7) to calculate the coefficients $a_{m,k}$.

In Chapter 3 we present a fast implementation of this algorithm, and prove that this method reduces the computational cost to $\mathcal{O}(MN \log[\max\{M, N\}])$ operations.

2.3.3 Calculating the inverse Gabor transform

We often need to calculate the inverse operation, i.e. calculate the signal $x[n]$ from a given set of coefficients $a_{m,k}$. As for the calculation of the Gabor transform, using the Zak transform we can develop a fast algorithm for this. The algorithm uses the following corollary based on [3]:

Corollary 2.3.1. *The Zak transform of the signal x[n] can be found as the product of the Fourier transform of the coefficients $a_{m,k}$ and the Zak transform of the elementary function $g[n]$, i.e.*

$$\tilde{x}[n, l; N, M] = \bar{a}[n, l; N, M]\tilde{g}[n, l; N, M]. \tag{2.25}$$

Proof. The proof of this corollary follows directly from theorem 2.3.1 and theorem 2.3.2. □

Using this we can find the inverse Gabor transform in the following way:

- Use (2.16) to calculate the MN periodic elementary function $G[n]$, $n \in \{0, ..., N-1\}$.
- Use (1.10) to calculate the Zak transform $\tilde{g}[n, l; N, M]$, $n \in \{0, ..., N-1\}$ and $l \in \{0, ..., M-1\}$, of the elementary function.
- Use (1.6) to calculate the Fourier transform $\bar{a}[n, l; N, M]$, $n \in \{0, ..., N-1\}$ and $l \in \{0, ..., M-1\}$, of the coefficients $a_{m,k}$.
- Use (2.25) to calculate the Zak transform $\tilde{x}[n, l; N, M]$, $n \in \{0, ..., N-1\}$ and $l \in \{0, ..., M-1\}$, of the signal $x[n]$.
- Use the inverse Zak transform (1.11) to calculate the signal $x[n]$.

In Chapter 3 we present a fast implementation of this algorithm, and prove that this method reduces the computational cost to $\mathcal{O}(MN\log[\max\{M,N\}])$ operations.

2.4 Oversampled Gabor representation

In the last section we used frequency shift $\Theta = 2\pi/N$, which was Gabors originally choice. This led to a numerically unstable representation and thus we need to modify this choice. Using oversampling, i.e. modifying the frequency shift to $\Theta < 2\pi/N$, we can make stable algorithms [3].

The oversampled inverse Gabor transform takes the form [cf. (2.17)]

$$x[n] = \sum_{m=0}^{M-1}\sum_{k=0}^{K-1} a_{m,k}G_{m,k}[n], \tag{2.26}$$

where

$$G_{m,k}[n] = G[n-mN]e^{j2\pi kn/K} \tag{2.27}$$

and $K > N$. The oversampled Gabor transform takes the form [cf. (2.19)]

$$a_{m,k} = \sum_{n=0}^{MN-1} x[n]W^*_{m,k}[n], \tag{2.28}$$

where

$$W_{m,k}[n] = W[n-mN]e^{j2\pi kn/K}. \tag{2.29}$$

In general there are no restrictions on K other than it must be integer and larger than N [3]. We will, however, only treat integer oversampling, i.e. we require that N is a divisor of K. That is $K/N \in \mathbf{Z}$. For convenience we introduce an integer $p = K/N$.

As in the case of critical sampling we treat signals $x[n]$ of finite length MN. This can always be achieved by padding the end of the signal with zeroes. We present the material in a different order than we did for the critically sampled case. Instead of starting with the calculation of the window function we start with the calculation of the Gabor transform and the inverse Gabor transform. We then show how expressing the Gabor transform and the inverse Gabor transform as vector products can be used for the calculation of the window function.

2.4.1 Calculating the Gabor transform

As for the critically sampled case we use the Zak transform for the calculation of the Gabor transform. We have the following important theorem based on [3]:

Theorem 2.4.1. *The Fourier transform of the coefficients $a_{m,k}$ can be found as the scaled product of the Zak transform of the signal $x[n]$ and the Zak transform of the window function $w[n]$, i.e.*

$$\bar{a}[n, l + rM/p; K, M] = K\tilde{x}[n, l; K, M/p]\tilde{w}^*[n, l + rM/p; N, M], \qquad (2.30)$$

where $p = K/N$ *and* $r \in \{0, ..., p-1\}$.

Proof. We start with the oversampled Gabor transform (2.28)

$$a_{m,k} = \sum_{n'=0}^{MN-1} x[n']W^*[n' - mN]e^{-j2\pi kn'/K}.$$

Multiplying both sides with $e^{-j[2\pi ml/M - 2\pi kn/K]}$ and taking the sum over $m \in \{0, ..., M-1\}$ and $k \in \{0, ..., K-1\}$, i.e. taking the 2 dimensional Fourier transform of both sides [cf. (1.6)], we get

$$\overline{a}[n, l; M, K] = \sum_{m=0}^{M-1}\sum_{k=0}^{K-1}\Big(\sum_{n'=0}^{MN-1} x[n']W^*[n'-mN]e^{-j2\pi kn'/K}\Big)e^{-j(2\pi ml/M - 2\pi kn/K)}.$$

Rearranging factors we get

$$\overline{a}[n, l; M, K] = \sum_{m=0}^{M-1}\sum_{k=0}^{K-1} x[n']W^*[n'-mN]\Big(\sum_{n'=0}^{MN-1} e^{-j2\pi k(n'-n)/K}\Big)e^{-j2\pi ml/M}.$$

The complex exponential is a circular function, thus each sum of length K equals 0 unless $n' = n + qK$, where $q \in \mathbf{Z}$, i.e.

$$\sum_{k=0}^{K-1} e^{-j2\pi k(n'-n)/K} = \begin{cases} 0 & n' \neq n + qK, \\ K & n' = n + qK \end{cases}.$$

Thus making the substitution $n' = n + qK$ we get

$$\overline{a}[n, l; M, K] = K\sum_{m=0}^{M-1}\sum_{q=-\infty}^{\infty} x[n + qK]W^*[n + qK - mN]e^{-j2\pi ml/M}.$$

Multiplying the right side with $e^{j2\pi qlp/M}e^{-j2\pi qlp/M} = 1$, where $p = K/N$, and taking a final rearrangement we find

$$\overline{a}[n, l; M, K] = K\sum_{q=-\infty}^{\infty} x[n+qK]e^{-j2\pi qlp/M}\sum_{m=0}^{M-1} W^*[n+qK-mN]e^{j2\pi(qp-m)l/M}.$$

Using that $K = pN$ we get

$$\overline{a}[n, l; M, K] = K\sum_{q=-\infty}^{\infty} x[n+qK]e^{-j2\pi qlp/M}\sum_{m=0}^{M-1} W^*[n+(qp-m)N]e^{j2\pi(qp-m)l/M}.$$

Since the sum over q is infinite we can make the substitution $m' = qp - m$, this gives

$$\overline{a}[n,l;M,K] = K\Big[\sum_{q=-\infty}^{\infty} x[n+qK]e^{-j2\pi qlp/M}\Big]\Big[\sum_{m'=0}^{M-1} W[n+m'N]e^{-j2\pi m'l/M}\Big]^*.$$

Comparing with (1.8) and (1.10) we see that the first summation is the Zak transform of the signal $x[n]$ and the second is the Zak transform of the window function $w[n]$, i.e.

$$\bar{a}[n,l;K,M] = K\tilde{x}[n,l;K,M/p]\tilde{w}^*[n,l;N,M].$$

We finally replace l by $l + rM/p$ and use the periodicity property of the Zak transform $\tilde{x}[n,l;K,M/p]$, this gives

$$\bar{a}[n,l+rM/p;K,M] = K\tilde{x}[n,l;K,M/p]\tilde{w}^*[n,l+rM/p;N,M],$$

which completes the proof. □

Note that in Gabors original case of critical sampling ($p = 1, K = N$), (2.30) takes the simple product form (2.23):

$$\bar{a}[n,l;N,M] = N\tilde{x}[n,l;N,M]\tilde{w}^*[n,l;N,M].$$

Using this theorem we can find the oversampled Gabor transform in the following way:

- Calculate the Zak transform $\tilde{w}[n,l+rM/p;N,M]$,
  $n \in \{0,...,K-1\}$, $l \in \{0,...,M/p-1\}$ and $r \in \{0,...,p-1\}$,
  of the window function (see Section 2.4.3).

- Use (1.10) to calculate the Zak transform $\tilde{x}[n,l;K,M/p]$,
  $n \in \{0,...,K-1\}$ and $l \in \{0,...,M-1\}$, of the signal $x[n]$.

- Use (2.30) to calculate the Fourier transform $\bar{a}[n,l+rM/p;K,M]$,
  $n \in \{0,...,K-1\}$, $l \in \{0,...,M/p-1\}$ and $r \in \{0,...,p-1\}$,
  of the Gabor coefficients $a_{m,k}$.

- Use the inverse Fourier transform (1.7) to calculate the Gabor coefficients $a_{m,k}$.

In Chapter 3 we present detailed algorithms and analyze the computational cost of this method.

2.4.2 Calculating the inverse Gabor transform

As for the critically sampled case we use the Zak transform for the calculation of the inverse Gabor transform. We have the following important theorem based on [3]:

Theorem 2.4.2. *The Zak transform of the signal x[n] can be found as the scaled sum of the products of the Fourier transform of the coefficients $a_{m,k}$ and the Zak transform of the elementary function $g[n]$, i.e.*

$$\tilde{x}[n,l;K,M/p] = \frac{1}{p}\sum_{r=0}^{p-1} \bar{a}[n,l+rM/p;K,M]\tilde{g}[n,l+rM/p;N,M]. \quad (2.31)$$

Proof. The proof of this theorem can be found in [3]. □

Using this theorem we can find the inverse Gabor transform in the following way:

- Use (2.16) to calculate the MN periodic elementary function $G[n]$, $n \in \{0,...,K-1\}$.
- Use (1.10) to calculate the Zak transform $\tilde{g}[n,l+rM/p;N,M]$, $n \in \{0,...,K-1\}$, $l \in \{0,...,M/p-1\}$ and $r \in \{0,...,p-1\}$, of the elementary function.
- Use (1.6) to calculate the Fourier transform $\bar{a}[n,l+rM/p;K,M]$, $n \in \{0,...,K-1\}$, $l \in \{0,...,M/p-1\}$ and $r \in \{0,...,p-1\}$, of the Gabor coefficients $a_{m,k}$.
- Use (2.31) to calculate the Zak transform $\tilde{x}[n,l;K,M/p]$, $n \in \{0,...,K-1\}$ and $l \in \{0,...,M/p-1\}$, of the signal $x[n]$.
- Use the inverse Zak transform (1.11) to calculate the signal $x[n]$.

In Chapter 3 we present detailed algorithms and analyze the computational cost of this method.

2.4.3 Calculating the window function

We have shown how the Gabor transform and the inverse Gabor transform can be calculated given the window function $w[n]$. In this section we address the problem of calculating this window function. In the critically sampled case we used theorem 2.3.1 for the calculation of the window function. This theorem followed directly from the bi-orthonormality condition (2.14). In the oversampled case there are no known ways of using the bi-orthonormality condition in a similar way. Thus we have to choose a different approach. We express the Gabor transform and the inverse Gabor transform as vector products and show how this can be used for the calculation of the window function.

The Fourier transform $\bar{a}[n, l + rM/p; K, M]$ can be split into the r functions

$$\bar{a}_r[n, l] = \bar{a}[n, l + rM/p; K, M],$$

where $r \in \{0, .., p - 1\}$. Likewise we can split the Zak transforms $\tilde{g}[n, l + rM/p; N, M]$ into the r functions

$$\tilde{g}_r[n, l] = \tilde{g}[n, l + rM/p; N, M],$$

and the Zak transform $\tilde{w}[n, l + rM/p; N, M]$ into the r functions

$$\tilde{w}_r[n, l] = \tilde{w}[n, l + rM/p; N, M].$$

Using vector notation the p functions $\bar{a}_r[n, l]$ can be combined into a p-dimensional column vector

$$\bar{\mathbf{a}} = \begin{pmatrix} \bar{a}_1[n, l] \\ \bar{a}_2[n, l] \\ \vdots \\ \bar{a}_{p-1}[n, l] \end{pmatrix}.$$

Likewise the p functions $\tilde{g}_r[n, l]$ can be combined into the p-dimensional row vector

$$\tilde{\mathbf{g}} = \begin{pmatrix} \tilde{g}_1[n, l] & \tilde{g}_2[n, l] & \dots & \tilde{g}_{p-1}[n, l] \end{pmatrix}, \tag{2.32}$$

and the p functions $\tilde{w}_r[n, l]$ into the p-dimensional row vector

$$\tilde{\mathbf{w}} = \begin{pmatrix} \tilde{w}_1[n, l] & \tilde{w}_2[n, l] & \dots & \tilde{w}_{p-1}[n, l] \end{pmatrix}.$$

Using these vectors the Gabor transform (2.28) can be expressed as the product

$$\bar{\mathbf{a}} = K\tilde{\mathbf{w}}^*\tilde{x}, \tag{2.33}$$

where we have adopted the short hand notation $\tilde{x} = \tilde{x}[n, l; N, M]$. The inverse Gabor transform (2.26) takes the form

$$\tilde{x} = \frac{1}{p}\tilde{\mathbf{g}}\bar{\mathbf{a}}. \tag{2.34}$$

Substituting (2.33) into (2.34) we get the relation

$$\tilde{x} = \frac{K}{p}\tilde{\mathbf{g}}\tilde{\mathbf{w}}^*\tilde{x}, \tag{2.35}$$

which should hold for any arbitrary vector $\tilde{\mathbf{x}}$, i.e. for any arbitrary signal $x[n]$. Thus this condition leads immediately to the corollary

Corollary 2.4.1. *For the Gabor transform (2.33) and the inverse Gabor transform (2.34) to form a transform pair we must impose the following bi-orthonormality condition*

$$\frac{K}{p}\tilde{\mathbf{g}}\tilde{\mathbf{w}}^* = 1. \tag{2.36}$$

Using this we can find the window function as the solution to (2.36), however, there is one problem. We have p unknowns and only one equation, thus the system is underdetermined. This is due to the fact that in oversampling the set of shifted and modulated versions of the elementary signal $g[n]$ is overcomplete. Thus the window function $w[n]$ that corresponds to an elementary signal $g[n]$ is not unique.

What we do is to find the solution in the sense of the minimum L_2 norm. This can be achieved by the so-called generalized (Moore-Penrose) inverse [6] $\tilde{\mathbf{g}}^+$, defined by

$$\tilde{\mathbf{g}}^+ = \tilde{\mathbf{g}}^*(\tilde{\mathbf{g}}\tilde{\mathbf{g}}^*)^{-1}.$$

The optimum solution w_{opt} then reads

$$\tilde{\mathbf{w}}_{opt} = \frac{p}{K}(\tilde{\mathbf{g}}^+)^* \tag{2.37}$$

and the optimum solution $\bar{\mathbf{a}}_{opt}$ reads

$$\bar{\mathbf{a}}_{opt} = p(\tilde{\mathbf{g}}^+)\tilde{x}. \tag{2.38}$$

Using 2.37 we can calculate the Zak transform $\tilde{w}[n, l + rM/p; N, M]$, $n \in \{0, ..., K-1\}$, $l \in \{0, ..., M/p-1\}$ and $r \in \{0, ..., p-1\}$, of the window function in the following way:

- Use (2.16) to calculate the MN periodic elementary function $G[n]$, $n \in \{0, ..., K-1\}$.
- Use (2.37) to calculate the Zak transform $\tilde{w}[n, l + rM/p; N, M]$, $n \in \{0, ..., K-1\}$, $l \in \{0, ..., M/p-1\}$ and $r \in \{0, ..., p-1\}$, of the window function.

Chapter 3

Algorithms

In this chapter we develop algorithms for the Gabor transform and the inverse Gabor transform. We treat both the critical sampled and the oversampled case and analyze the computational cost of the algorithms. The algorithms are all based on the mathematical techniques presented in Chapter 2. We refer to the method as the Zak-Gabor method.

It should be mentioned that there exists methods based on relaxation networks [9], matrix inversion [14] [26], unitary matrix factorization [20] [25] and conjugate gradients [24]. However, a performance analysis [7] between relaxation networks, matrix inversion and the Zak-Gabor method in the critical sampled case shows that the Zak-Gabor approach is very stable, accurate and by far the most rapid of the three. The unitary matrix factorization method is essentially the same as the Zak-Gabor method. Both are based on the FFT and the main difference is how the methods are presented, unitary matrix factorization versus the Zak transform. The conjugate gradient (CG) method is iterative and the efficiency depends on the convergence rate of the CG. It is mainly used when it is important not to have any restrictions on the lattice parameters.

We start with a short introduction to the Fast Fourier transform and the inverse Fast Fourier transform. We then define some notation before we present algorithms for the 2d Fourier transform and its inverse, the Zak transform and its inverse and the periodization of the elementary function. Using these algorithms we present algorithms for the critical sampled Gabor transform and its inverse. At the end we extend these algorithms to handle integer oversampling.

3.1 Fast Fourier transform and its inverse

The ability to develop fast algorithms for the Gabor transform and the inverse Gabor transform relies on fast computation of the Fourier transform (1.6) and the inverse Fourier transform (1.7). This can be achieved using the Fast Fourier transform (FFT) and the inverse Fast Fourier transform (IFFT).

The idea behind the FFT and the IFFT is a divide and conquer approach to

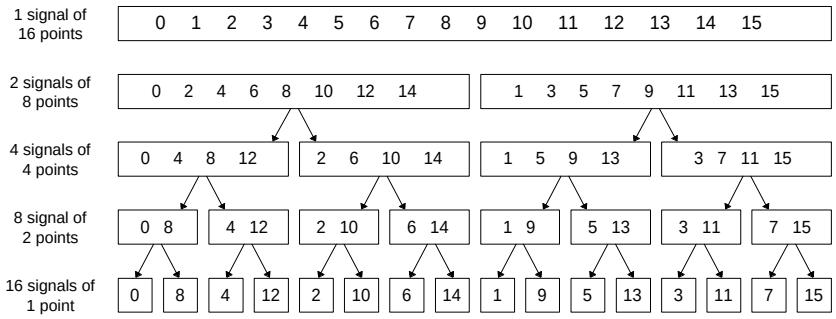

Figure 3.1: The FFT decomposition. A N-point signal is decomposed into N signals, each containing a single point. Each stage separates the even and odd numbered samples.

recursively break up the original N-point sample into two ($N/2$) sequences, see Figure 3.1. This is because a series of smaller problems are easier to solve than one large one. The Fourier transform and the inverse Fourier transform requires $(N-1)^2$ complex multiplications and $N(N-1)$ complex additions as opposed to the FFT and IFFT approach of breaking it down into series of 2 point samples which only require 1 multiplication and 2 additions and the recombination of the points which is minimal. The result is that the computational cost reduces from $\mathcal{O}(N^2)$ operations to $\mathcal{O}(N \log N)$ operations. Details of the FFT and the IFFT can be found in several textbooks [2] [21].

For most programming languages, such as C/C++, matlab and fortran, algorithms for calculating the FFT and IFFT can be downloaded from the internet.

3.2 Notation

Matrix indexing

Having a $M \times N$ matrix $\mathbf{X}$ we define the following index notations. $\mathbf{X}[i,j]$ returns the element at row number i and column number j. $\mathbf{X}[:,j]$ returns every element at column number j. $\mathbf{X}[i,:]$ returns every element at row number i. $\mathbf{X}[:,i:j:k]$ returns the sub matrix

$$\mathbf{X} = \begin{pmatrix} X[1,i] & X[1,i+j] & \dots & X[1,k] \\ X[2,i] & X[2,i+j] & \dots & X[2,k] \\ \vdots & \vdots & & \vdots \\ X[M,i] & X[M,i+j] & \dots & X[M,k] \end{pmatrix},$$

and $\mathbf{X}[i:j:k,:]$ returns the sub matrix

$$\mathbf{X} = \begin{pmatrix} X[i,1] & X[i,2] & \dots & X[i,N] \\ X[i+j,1] & X[i+j,2] & \dots & X[i+j,N] \\ \vdots & \vdots & & \vdots \\ X[k,1] & X[k,2] & \dots & X[k,N] \end{pmatrix}.$$

All indexes start at 1.

Example: Having the 4×3 matrix

$$\mathbf{X} = \begin{pmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \\ 10 & 11 & 12 \end{pmatrix},$$

$\mathbf{X}[4,3]$ returns $\{12\}$, $\mathbf{X}[:,3]$ returns $\{3,6,9,12\}$ and $\mathbf{X}[4,:]$ returns $\{10,11,12\}$. $\mathbf{X}[:,1:2:3]$ returns the sub matrix

$$\mathbf{X_s} = \begin{pmatrix} 1 & 3 \\ 4 & 6 \\ 7 & 9 \\ 10 & 12 \end{pmatrix},$$

and $\mathbf{X}[1:2:3,:]$ returns the sub matrix

$$\mathbf{X_s} = \begin{pmatrix} 1 & 2 & 3 \\ 7 & 8 & 9 \end{pmatrix}.$$

Matrix operations

- $\mathbf{X}$*$\mathbf{Y}$ denotes the product of $\mathbf{X}$ and $\mathbf{Y}$. The number of columns of $\mathbf{X}$ must equal the number of rows of $\mathbf{Y}$ unless one is scalar. A scalar can be multiplied into anything.
- $\mathbf{X}$.*$\mathbf{Y}$ denotes the element-by-element multiplication of $\mathbf{X}$ and $\mathbf{Y}$. $\mathbf{X}$ and $\mathbf{Y}$ must have the same dimensions unless one is scalar. A scalar can be multiplied into anything.
- $\mathbf{X}$./$\mathbf{Y}$ denotes the element-by-element division of $\mathbf{Y}$ into $\mathbf{X}$. $\mathbf{X}$ and $\mathbf{Y}$ must have the same dimensions unless one is scalar. A scalar can be divided with anything.
- $\mathbf{X}^T$ denotes the non-conjugate transpose of $\mathbf{X}$.
- $\mathbf{X}^*$ denotes the conjugate transpose of $\mathbf{X}$.

Sum of elements

For vectors, $SUM(\mathbf{x})$ is the sum of the elements of $\mathbf{x}$. For matrices, $SUM(\mathbf{X})$ is a vector with the sum over each column.

Example: Having the matrix

$$\mathbf{X} = \begin{pmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \\ 10 & 11 & 12 \end{pmatrix},$$

$SUM(\mathbf{X})$ gives the vector $\mathbf{x} = \{22, 26, 30\}$

Vector

The method $vector(J : K)$ returns the vector $\{J, J + 1, ..., K\}$.

Example: $vector(-1 : 3)$ returns {-1,0,1,2,3}.

Comments

A Line starting with a %-sign is a comment.

Length of vector

The method $length(\mathbf{x})$ returns the number of elements in the vector $\mathbf{x}$.

Vector concatination

The method $concat(\mathbf{x_1}, \mathbf{x_2})$ appends the vector $\mathbf{x_2}$ to the end of the vector $\mathbf{x_1}$.

Example: Having the vectors $\mathbf{x_1} = \{1, 2, 3\}$ and $\mathbf{x_2} = \{4, 5, 6\}$, $concat(\mathbf{x_1}, \mathbf{x_2})$ returns the vector $\{1, 2, 3, 4, 5, 6\}$.

Matrix replication

The method $repmat(\mathbf{X}, M, N)$ replicates the matrix $\mathbf{X}$ into a $M \times N$ block matrix.

Example: Having the matrix

$$\mathbf{X} = \begin{pmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \\ 10 & 11 & 12 \end{pmatrix},$$

$repmat(\mathbf{X}, 1, 2)$ returns the 1×2 block matrix

$$\mathbf{X}' = \begin{pmatrix} \mathbf{X} & \mathbf{X} \end{pmatrix} = \begin{pmatrix} 1 & 2 & 3 & 1 & 2 & 3 \\ 4 & 5 & 6 & 4 & 5 & 6 \\ 7 & 8 & 9 & 7 & 8 & 9 \\ 10 & 11 & 12 & 10 & 11 & 12 \end{pmatrix}.$$

Fast Fourier transform and its inverse

Having the vector $\mathbf{x} = \{x_0, x_1, ..., x_{N-1}\}$ of length N we define $FFT(\mathbf{x})$ as an algorithm calculating the discrete Fourier transform

$$\overline{x}[k] = \sum_{n=0}^{N-1} x[n] e^{-j2\pi kn/N} \tag{3.1}$$

using $\mathcal{O}(N \log N)$ operations.

Likewise we define $IFFT(\overline{\mathbf{x}})$ as an algorithm calculating the discrete inverse Fourier transform

$$x[n] = \frac{1}{N} \sum_{k=0}^{N-1} \overline{x}[k] e^{j2\pi kn/N} \tag{3.2}$$

using $\mathcal{O}(N \log N)$ operations.

Vector to matrix decomposition

Having the vector $\mathbf{x} = \{x_0, x_1, ..., x_{MN-1}\}$ of length MN we define $MAT_{M\times N}(\mathbf{x})$ as an algorithm decomposing the vector $\mathbf{x}$ into the $M \times N$ matrix

$$\mathbf{X} = \begin{pmatrix} x_0 & \dots & x_{N-1} \\ x_N & \dots & x_{2N-1} \\ \vdots & \ddots & \vdots \\ x_{M(N-1)} & \dots & x_{MN-1} \end{pmatrix}.$$

Example : Having the vector $\mathbf{x} = \{1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12\}$, $MAT_{4\times 3}(\mathbf{x})$ returns the matrix

$$\mathbf{X} = \begin{pmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \\ 10 & 11 & 12 \end{pmatrix}.$$

Matrix to vector decomposition

Having the $M \times N$ matrix

$$\mathbf{X} = \begin{pmatrix} a_{11} & \dots & a_{1N} \\ a_{21} & \dots & a_{2N} \\ \vdots & \ddots & \vdots \\ a_{M1} & \dots & a_{MN} \end{pmatrix}$$

we define $VEC(\mathbf{X})$ as an algorithm decomposing the matrix $\mathbf{X}$ into the vector $\mathbf{x} = \{a_{11}, ..., a_{1N}, a_{21}, ..., a_{2N}, ..., a_{M1}, ..., a_{MN}\}$ of length MN .

Example: Having the matrix

$$\mathbf{X} = \begin{pmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \\ 10 & 11 & 12 \end{pmatrix},$$

$VEC(\mathbf{X})$ returns the vector $\mathbf{x} = \{1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12\}$.

3.3 Critical sampled Gabor representation

We start this section presenting algorithms for the 2d Fourier transform and its inverse, the Zak transform and its inverse and the periodization of the elementary function. Using these algorithms we present algorithms for the critical sampled Gabor transform and its inverse.

3.3.1 The 2d Fourier transform and its inverse

Using the FFT and IFFT we develop an algorithm calculating the 2d Fourier transform (1.6) of a $M \times N$ matrix $\mathbf{A}$. Comparing (1.6) with (3.1) and (3.2) we see that this can be done by first taking the FFT of the columns of $\mathbf{A}$, and then take the IFFT of the rows. Since the IFFT is scaled by the factor $1/N$ we have to multiply the result by the factor N. To be consistent with the definition (1.6) we transpose the final result.

Algorithm 1 The 2d Fourier transform.

INPUT: $M \times N$ matrix $\mathbf{A}$.
OUTPUT: $N \times M$ matrix $\overline{\mathbf{A}}$ containing the 2d Fourier transform of $\mathbf{A}$.

% Fourier transform columns.
for $i = 0$ to $N - 1$ **do**
  $\mathbf{A_1}[:, i] \leftarrow FFT(\mathbf{A}[:, i])$
end for

% Inverse Fourier transform rows.
for $i = 0$ to $M - 1$ **do**
  $\overline{\mathbf{A}}[i, :] \leftarrow N \cdot IFFT(\mathbf{A_1}[i, :])$
end for

% Transpose matrix.
$\overline{\mathbf{A}} \leftarrow \overline{\mathbf{A}}^{\mathbf{T}}$

Likewise we calculate the inverse 2d Fourier transform (1.7) of a $N \times M$ matrix $\overline{\mathbf{A}}$ by first taking the IFFT of the rows of $\overline{\mathbf{A}}$, and then take the FFT of the columns.

We then scale the result with the factor $1/(MN)$ from (1.7), but since the IFFT is scaled by the factor $1/M$ we only scale the result by the factor $1/N$. In addition we transpose the final result to be consistent with the definition (1.7).

Algorithm 2 The 2d inverse Fourier transform.

INPUT: $N \times M$ matrix $\overline{\mathbf{A}}$.
OUTPUT: $M \times N$ matrix $\mathbf{A}$ containing the inverse 2d Fourier transform of $\overline{\mathbf{A}}$.

% Inverse Fourier transform rows.
for $i = 0$ to $N - 1$ **do**
  $\mathbf{A_1}[i, :] \leftarrow IFFT(\overline{\mathbf{A}}[i, :])$
end for

% Fourier transform columns.
for $i = 0$ to $M - 1$ **do**
  $\mathbf{A}[:, i] \leftarrow FFT(\mathbf{A_1}[:, i])$
end for

% Scale and transpose matrix.
$\mathbf{A} \leftarrow \frac{1}{N} \cdot \mathbf{A^T}$

Computational cost

The 2d Fourier transform requires N FFT operations and M IFFT operations. Each FFT operation requires $\mathcal{O}(M \log M)$ operations and each IFFT operation requires $\mathcal{O}(N \log N)$ operations. In addition the matrix transpose requires $\mathcal{O}(MN)$ operations. This gives a total cost of $\mathcal{O}(MN \log M) + \mathcal{O}(MN \log N)$ + $\mathcal{O}(MN)$ = $\mathcal{O}(MN \log[\max\{M, N\}])$.

The inverse operation requires N IFFT operations and M FFT operations. Each FFT operation requires $\mathcal{O}(N \log N)$ operations and each IFFT operation requires $\mathcal{O}(M \log M)$ operations. In addition the matrix transpose requires $\mathcal{O}(MN)$ operations. This gives a total cost of $\mathcal{O}(MN \log N) + \mathcal{O}(MN \log M)$ + $\mathcal{O}(MN)$ = $\mathcal{O}(MN \log[\max\{M, N\}])$.

3.3.2 The Zak transform and its inverse

Using the FFT we develop an efficient algorithm calculating the ZAK transform $\tilde{x}[n, l; N, M]$, $n \in \{0, ..., N - 1\}$ and $l \in \{0, ..., M - 1\}$, of a vector $\mathbf{x}$ of length MN.

We decompose the vector $\mathbf{x} = \{x_0, x_1, ..., x_{MN-1}\}$ into the $M \times N$ matrix

$$\mathbf{X} = \begin{pmatrix} x_0 & \cdots & x_{N-1} \\ x_N & \cdots & x_{2N-1} \\ \vdots & \ddots & \vdots \\ x_{N(M-1)} & \cdots & x_{MN-1} \end{pmatrix}.$$

What we have achieved is that column number i contains the elements $X[i + mN]$, where $m \in \{0, 1, ..., M - 1\}$. Comparing with (1.10) we see that the Zak transform now easily can be calculated taking the FFT of each column in $\mathbf{X}$. The resulting matrix is transposed into a $N \times M$ matrix $\tilde{\mathbf{X}}$ containing the Zak transform of the vector $\mathbf{x}$.

Algorithm 3 The Zak transform.

INPUT: Vector $\mathbf{x}$ of length MN, lattice parameters M and N.
OUTPUT: $N \times M$ matrix $\tilde{\mathbf{X}}$ containing the Zak transform of $\mathbf{x}$.

% Decompose vector into an $M \times N$ matrix.
$\mathbf{X_1} \leftarrow MAT_{M \times N}(\mathbf{x})$

% Fourier transform columns.
for $i = 0$ to $N - 1$ **do**
  $\mathbf{X}[:, i] \leftarrow FFT(\mathbf{X_1}[:, i])$
end for

% Transpose matrix.
$\tilde{\mathbf{X}} \leftarrow \mathbf{X}^T$

The inverse operation takes the $N \times M$ matrix $\tilde{\mathbf{X}}$ and returns the vector $\mathbf{x}$. We transpose the matrix $\tilde{\mathbf{X}}$ and take the IFFT of each column. This returns the matrix $\mathbf{X}$. Decomposing the matrix row by row into a vector, as described in Section 3.2, returns the vector $\mathbf{x}$.

Computational cost

The Zak transform requires N FFT operations, each requiring $\mathcal{O}(M \log M)$ operations. In addition the vector to matrix decomposition and the matrix transpose both require $\mathcal{O}(MN)$ operations. This gives a total cost of $\mathcal{O}(MN \log M)$ operations.

The inverse operation requires M IFFT operations, each requiring $\mathcal{O}(N \log N)$ operations. In addition the matrix to vector decomposition and the matrix transpose both require $\mathcal{O}(MN)$ operations. This gives a total cost of $\mathcal{O}(MN \log N)$ operations.

Algorithm 4 The inverse Zak transform.

INPUT: $N \times M$ matrix $\tilde{\mathbf{X}}$.
OUTPUT: Vector $\mathbf{x}$ of length MN containing the inverse Zak transform of $\tilde{\mathbf{X}}$.

% Transpose matrix.
$\mathbf{X_1} \leftarrow \tilde{\mathbf{X}}^{\mathbf{T}}$

% Inverse Fourier transform columns.
for $i = 0$ to $M - 1$ **do**
  $\mathbf{X}[:, i] \leftarrow IFFT(\mathbf{X_1}[:, i])$
end for

% Decompose the $M \times N$ matrix into a vector.
$\mathbf{x} \leftarrow VEC(\mathbf{X})$

3.3.3 The periodization of the elementary function

The implementation of the periodization (2.16) is not possible without truncating the infinite sum over r, i.e. we have to make r finite. The number of terms necessary depends on the desired accuracy and can either be calculated in advance or during the iteration. Since the first option requires less computation in addition to easy implementation we have chosen this one. Since we have restricted us to the Gaussian elementary function (2.10) and (2.11) the number of terms necessary can easily be calculated.

We show the calculation for the odd function, but the calculation for the even function is identical. The ratio between two succeeding terms is given by

$$\frac{g[n]}{g[n+MN]} = \frac{2^{1/4}e^{-\pi(n/N)^2}}{2^{1/4}e^{-\pi((n+MN)/N)^2}} = \frac{1}{e^{-\pi M^2}}$$

Since the lattice parameter M is at least 2 the ratio is at least 534. This means that the terms decrease at least by a factor of 534. Since the maximum value of the Gaussian function is $2^{1/4}$ the third term, $g[n+3MN]$, is not more than in the order of 10^{-9}. Thus by including the 7 terms from $g[n-3MN]$ to $g[n+3MN]$ we are ensured an accuracy of at least 10^{-9}.

The implementation can be done easily. We make a $7 \times MN$ matrix $\mathbf{X}$ containing the evaluation points $n + rMN$, $r \in \{-3, ..., 0, ..., 3\}$ and $n \in \{0, ..., MN - 1\}$:

$$\mathbf{X} = \begin{pmatrix} 0-3MN & 1-3MN & 2-3MN & \dots & MN-1-3MN \\ 0-2MN & 1-2MN & 2-2MN & \dots & MN-1-2MN \\ 0-MN & 1-MN & 2-MN & \dots & MN-1-MN \\ 0 & 1 & 2 & \dots & MN-1 \\ 0+MN & 1+MN & 2+MN & \dots & MN-1+MN \\ 0+2MN & 1+2MN & 2+2MN & \dots & MN-1+2MN \\ 0+3MN & 1+3MN & 2+3MN & \dots & MN-1+3MN \end{pmatrix}.$$

Each column i in the matrix $\mathbf{X}$ contains the evaluation points $(i-1)+rMN$, $r \in \{-3, ..., 0, ..., 3\}$. Where the -1 factor is included because the column indexing starts at 1.

Next calculate the function

$$g[\mathbf{X}] = \begin{pmatrix} g[0-3MN] & g[1-3MN] & g[2-3MN] & \dots & g[MN-1-3MN] \\ g[0-2MN] & g[1-2MN] & g[2-2MN] & \dots & g[MN-1-2MN] \\ g[0-MN] & g[1-MN] & g[2-MN] & \dots & g[MN-1-MN] \\ g[0] & g[1] & g[2] & \dots & g[MN-1] \\ g[0+MN] & g[1+MN] & g[2+MN] & \dots & g[MN-1+MN] \\ g[0+2MN] & g[1+2MN] & g[2+2MN] & \dots & g[MN-1+2MN] \\ g[0+3MN] & g[1+3MN] & g[2+3MN] & \dots & g[MN-1+3MN] \end{pmatrix}.$$

The periodization vector (2.16) can then be calculated taking the sum of each column.

Algorithm 5 The periodization of the elementary function.

INPUT: Lattice parameters M and N.
OUTPUT: Vector $\mathbf{g_p}$ of length MN containing the periodization of the Gaussian elementary function.

% Evaluation points.
$\mathbf{x} \leftarrow$ vector(-3*M*N : 4*M*N-1)

% $7 \times MN$ matrix containing the evaluation points.
$\mathbf{X} \leftarrow MAT_{7\times MN}(\mathbf{x})$

% Evaluate matrix.
$\mathbf{X_g} \leftarrow$ g[$\mathbf{X}$]

% Sum each column.
$\mathbf{g_p} \leftarrow$ SUM($\mathbf{X_g}$)

Computational cost

Each of the four stages in the algorithm requires $\mathcal{O}(MN)$ operations, giving a total cost of $\mathcal{O}(MN)$ operations.

3.3.4 The Gabor transform and its inverse

We now have the necessary algorithms for calculating the Gabor transform (2.19). Given the input vector $\mathbf{x}$ and the lattice parameters M and N we want to calculate the Gabor coefficient matrix $\mathbf{A}$. The implementation follows directly from the method presented in Subsection 2.3.2.

It's important to note that for a given lattice we only have to calculate the window function once. This means that the Gabor transform of a set of equally lengthened vectors can be calculated using the same window function. This reduces the computational cost considerably.

Algorithm 6 The Gabor transform.

INPUT: Vector $\mathbf{x}$, lattice parameters M and N.
OUTPUT: $M \times N$ matrix $\mathbf{A}$ containing the Gabor transform of $\mathbf{x}$.

Insert (MN-length($\mathbf{x}$)) zeroes at the end of $\mathbf{x}$ making it of length MN.

% Use Algorithm 5 to calculate the MN periodized elementary function
% of length MN.
$\mathbf{g_p} \leftarrow$ The periodization of the elementary function

% Use Algorithm 3 to calculate the Zak transform $\tilde{g}[n, l; N, M]$,
% $n \in \{0, ..., N-1\}$ and $l \in \{0, ..., M-1\}$, of the elementary function.
$\mathbf{G_{ZT}} \leftarrow \tilde{g}[n, l; N, M]$

% Use Algorithm 3 to calculate the Zak transform $\tilde{x}[n, l; N, M]$,
% $n \in \{0, ..., N-1\}$ and $l \in \{0, ..., M-1\}$, of the vector $\mathbf{x}$.
$\mathbf{X_{ZT}} \leftarrow \tilde{x}[n, l; N, M]$

%Use relation (2.24) to calculate the 2d Fourier transform $\bar{a}[n, l; N, M]$,
% $n \in \{0, ..., N-1\}$ and $l \in \{0, ..., M-1\}$, of the coefficients.
$\mathbf{A}_{FT} \leftarrow \mathbf{X_{ZT}}./\mathbf{G_{ZT}}$

%Use Algorithm 2 to calculate the 2d inverse Fourier transform
% of $\mathbf{A}_{FT}$.
$\mathbf{A} \leftarrow$ The 2d inverse Fourier transform of $\mathbf{A}_{FT}$

The inverse operation takes the Gabor coefficient matrix $\mathbf{A}$ and calculates the vector $\mathbf{x}$. The implementation follows directly from the method presented in Subsection 2.3.3.

Algorithm 7 The inverse Gabor transform.

INPUT: $M \times N$ matrix $\mathbf{A}$ containing the Gabor transform of $\mathbf{x}$.
OUTPUT: Vector $\mathbf{x}$.

% Find the lattice parameters M and N from the dimensions of $\mathbf{A}$.
$M \leftarrow length(A[:,1])$
$N \leftarrow length(A[1,:])$

% Use Algorithm 5 to calculate the MN periodized elementary function % of length MN.
$\mathbf{g}_p \leftarrow$ The periodization of the elementary function

% Use Algorithm 3 to calculate the Zak transform $\tilde{g}[n,l;N,M]$,
% $n \in \{0,...,N-1\}$ and $l \in \{0,...,M-1\}$, of the periodized elementary
% function.
$\mathbf{G}_{ZT} \leftarrow \tilde{g}[n,l;N,M]$

%Use Algorithm 1 to calculate the 2d Fourier transform $\bar{a}[n,l;N,M]$,
% $n \in \{0,...,N-1\}$ and $l \in \{0,...,M-1\}$, of the coefficient matrix.
$\mathbf{A}_{FT} \leftarrow \bar{a}[n,l;N,M]$

% Use the relation (2.25) to calculate the Zak transform $\tilde{x}[n,l;N,M]$,
% $n \in \{0,...,N-1\}$ and $l \in \{0,...,M-1\}$, of the vector $\mathbf{x}$.
$\mathbf{X}_{ZT} \leftarrow \mathbf{A}_{FT}.*\mathbf{G}_{ZT}$

%Use Algorithm 4 to calculate the inverse Zak transform of $\mathbf{X}_{ZT}$.
$\mathbf{x} \leftarrow$ The inverse Zak transform of $\mathbf{X}_{ZT}$

Computational cost

For the Gabor transform the periodization of the elementary function requires $\mathcal{O}(MN)$ operations. The Zak transform of the periodized elementary function and the Zak transform of the vector both require $\mathcal{O}(MN \log M)$ operations. The element-by-element division requires $\mathcal{O}(MN)$ operations and the 2d inverse Fourier transform requires $\mathcal{O}(MN \log[\max\{M, N\}])$ operations. This gives a total cost of $\mathcal{O}(MN \log[\max\{M, N\}])$ operations.

Likewise for the inverse Gabor transform the periodization of the elementary function requires $\mathcal{O}(MN)$ operations and the Zak transform of it requires $\mathcal{O}(MN \log M)$ operations. The 2d Fourier transform of the coefficient matrix requires $\mathcal{O}(MN \log[\max\{M, N\}])$ operations and the element-by-element multiplication requires $\mathcal{O}(MN)$ operations. Finally the inverse Zak transform requires $\mathcal{O}(MN \log N)$ operations, giving a total cost of
$\mathcal{O}(MN \log[\max\{M, N\}])$ operations.

3.4 Oversampled Gabor representation

In this section we extend the algorithms presented in the last section to handle integer oversampling. The algorithms are all based on the mathematical techniques presented in Section 2.4.

In the critical sampled case we could easily calculate the Zak transform of the window function as the reciprocal of the Zak transform of the elementary function (2.21). As explained in Subsection 2.4.3 this is not possible in the oversampled case. Instead we have to use relation (2.37) to calculate the Zak transform of the window function.

We start this section presenting an algorithm for rotating a vector left. This will help us implementing an efficient algorithm for the Zak transform of the elementary function, which we present next. Then an algorithm for calculating the Zak transform of the window function is presented, and finally algorithms for the oversampled Gabor transform and its inverse are presented.

3.4.1 Left rotation of a vector

Given the vector $\mathbf{x} = \{x_0, x_1, ..., x_{N-1}\}$ we implement an algorithm left rotating the vector i places into the vector $\mathbf{x_r} = \{x_i, x_{i+1}, ..., x_{N-1}, x_0, ..., x_{i-1}\}$.

The implementation is straight forward, moving the i first elements to the end of the sequence.

Computational cost

We have to move every element in the vector, giving a total cost of $\mathcal{O}(N)$ operations, where N is the number of elements in the vector.

Algorithm 8 Left rotation of a vector.

INPUT: Input vector $\mathbf{x}$, rotation i.
OUTPUT: Rotated vector $\mathbf{x_r}$.

$\mathbf{x_1} \leftarrow \mathbf{x}\Big((1+i) : length(\mathbf{x})\Big)$
$\mathbf{x_r} \leftarrow concat\Big(\mathbf{x_1}, \mathbf{x}(1:i)\Big)$

3.4.2 The Zak transform of the elementary function

In the oversampled case we need to calculate $\tilde{g}[n, l; N, M]$, $n \in \{0, ..., K-1\}$ and $l \in \{0, ..., M-1\}$, where $K = pN$ and p is the oversampling factor. That is, whereas we in the critically sampled case calculated the Zak transform of the periodized elementary function $G[n']$, $n' \in \{0, ..., MN-1\}$, of length MN, we in the oversampled case have to calculate the Zak transform of the periodized elementary function $G[n']$, $n' \in \{0, ..., MK-1\}$, of length MK. Since we use the same lattice parameters, M and N, we can not use Algorithm 3 directly. We have to split $G[n']$ into p functions of length MN, Zak transform each part and then combine them again.

The key observation is that the calculation of the Zak transform $\tilde{g}[n, l; N, M]$ can be divided into p intervals. The first interval is when $n \in \{0, ..., N-1\}$. In this interval we use function values $G[n']$, where $n' \in \{0, ..., MN-1\}$. This is a vector of length MN and thus we can use Algorithm 3 to calculate the Zak transform of it. In the next interval, when $n \in \{N, ..., 2N-1\}$, we use function values $G[n']$, where $n' \in \{N, ..., N(M+1)-1\}$. This is also a vector of length MN and we can use Algorithm 3 to calculate the Zak transform of it. Generally we have that in interval i, $i \in \{0, ..., p-1\}$, where $n \in \{i*N, ..., (i+1)N-1\}$ we use function values $G[n']$, where $n' \in \{i*N, ..., N(M+i)-1\}$. This is a vector of length MN, and Algorithm 3 can be used for calculating the Zak transform of it.

When we have calculated the Zak transform $\mathbf{G^i_{ZT}}$ from each interval $i, i \in \{0, ..., p-1\}$, we combine them into the $p \times 1$ block matrix

$$\mathbf{G_{ZT}} = \begin{pmatrix} \mathbf{G^0_{ZT}} \\ \mathbf{G^1_{ZT}} \\ \vdots \\ \mathbf{G^{p-1}_{ZT}} \end{pmatrix}.$$

The resulting $K \times M$ matrix is the Zak transform of periodized elementary function $G[n]$ of length MK. Since we only have available the method presented in Subsection 3.2 we have to combine them in the following equivalent way

$$\mathbf{G_{ZT}} = \begin{pmatrix} (\mathbf{G^0_{ZT}})^T & (\mathbf{G^1_{ZT}})^T & \dots & (\mathbf{G^{p-1}_{ZT}})^T \end{pmatrix}^T.$$

Another important observation is that even though $G[n']$ is of length KM it is still MN periodic, meaning that $G[n'] = G[n' + MN]$ and on the form $G[n] = \{\underbrace{g_0, ..., g_{MN-1}}_{0}, \underbrace{g_0, ..., g_{MN-1}}_{1}, ..., \underbrace{g_0, ..., g_{MN-1}}_{p-1}\}$. Because of this the vector used in interval i, $\mathbf{g_i} = \{g_{iN}, ..., g_{MN-1}, g_0, ..., g_{iN-1}\}$, is a $i * N$ places left rotation of the vector $\mathbf{g_0} = \{g_0, ..., g_{MN-1}\}$, used in the first interval. For example the vector used in the second interval, $\mathbf{g_1} = \{g_N, ..., g_{MN-1}, g_0, ..., g_{N-1}\}$, is a 5 places left rotation of the vector used in the first interval.

Thus by starting with the vector $\mathbf{g_0} = \{g_0, g_1, ..., g_{MN-1}\}$ and alternating between taking the Zak transform and rotating N places left, we can calculate the Zak transform of the periodized elementary function in an elegant way.

Algorithm 9 The Zak transform of the periodized elementary function.

INPUT: Lattice parameters M and N, oversampling factor p.
OUTPUT: $K \times M$ matrix $\mathbf{G}_{ZT}$ containing the Zak transform of the MN periodized elementary function of length MK.

% Use Algorithm 5 to calculate the MN periodized elementary function
% of length MN.
$\mathbf{g_p} \leftarrow$ The periodized elementary function

% Use Algorithm 3 to calculate the Zak transform $\tilde{g}_p[n, l; N, M]$,
% $n \in \{0, ..., N-1\}$ and $l \in \{0, ..., M-1\}$, of $\mathbf{g_p}$. Transpose the result.
$\mathbf{G_{ZT}} \leftarrow \tilde{g}_p[n, l; N, M]$
$\mathbf{G_{ZT}} \leftarrow (\mathbf{G_{ZT}})^T$

for $i = 1$ to $p - 1$ **do**
  % Use Algorithm 8 to rotate $\mathbf{g_p}$ $i * N$ places left.
  $\mathbf{g_r} \leftarrow$ Rotate $\mathbf{g_p}$ $i * N$ places left

  % Use Algorithm 3 to calculate the Zak transform $\tilde{g}_r[n, l; N, M]$,
  % $n \in \{0, ..., N-1\}$ $l \in \{0, ..., M-1\}$, of $\mathbf{g_r}$.
  $\mathbf{G'_{ZT}} \leftarrow \tilde{g}_r[n, l; N, M]$

  % Append $\mathbf{G'_{ZT}}$ to the end of $\mathbf{G_{ZT}}$.
  $\mathbf{G_{ZT}} \leftarrow concat\Big(\mathbf{G_{ZT}}, (\mathbf{G'_{ZT}})^T\Big)$
end for

% The final transpose.
$\mathbf{G_{ZT}} \leftarrow (\mathbf{G_{ZT}})^T$

Computational cost

The periodization of the elementary function requires $\mathcal{O}(MN)$ operations. The following Zak transform requires $\mathcal{O}(MN \log M)$ operations and the matrix transpose requires $\mathcal{O}(MN)$ operations. Inside the loop, which is repeated $p-1$ times, we have tree operations. The rotation, which requires $\mathcal{O}(MN)$ operations. The following Zak transform, which requires $\mathcal{O}(MN \log M)$ operations and the final appending, which requires $\mathcal{O}(MN)$ operations. Since $p = K/M$ this gives a total cost of $\mathcal{O}(KN \log M)$ operations.

3.4.3 The Zak transform of the window function

Having the Zak transform of the periodized elementary function we use the mathematical techniques presented in Subsection 2.4.3 to calculate the Zak transform of the window function. For each $n \in \{0, ..., K-1\}$ and $l \in \{0, ..., M/p - 1\}$ we set up the vector (2.32) and use relation (2.37) to calculate the Zak transform of the window function.

Algorithm 10 The Zak transform of the window function.

INPUT: $K \times M$ matrix $\mathbf{G}_{ZT}$ containing the Zak transform of $\mathbf{g}_p$.
OUTPUT: $K \times M$ matrix $\mathbf{W}_{ZT}$ containing the Zak transform of the window function.

% Find the lattice parameters K and M from the dimension of $\mathbf{G_{ZT}}$.
$K \leftarrow length(\mathbf{G}_{ZT}[:, 1])$
$M \leftarrow length(\mathbf{G}_{ZT}[1, :])$

for $n = 1$ to K **do**
  for $l = 1$ to M/p **do**

    % Set up the vector (2.32).
    $w \leftarrow \mathbf{G}_{ZT}(n, l : (M/p) : M)$

    % Use (2.37) to calculate the Zak transform of the window function.
    $W(n, l : (M/p) : M) \leftarrow \left(p/K * w^* * 1/(w * w^*)\right)^*$

  end for
end for

Computational cost

Inside the nested loop, which is repeated $K * M/p = MN$ times, we have two operations. The set up of the vector which requires $\mathcal{O}(K/N)$ operations. The

calculation of relation (2.37) which also uses $\mathcal{O}(K/N)$ operations. This gives a total cost of $\mathcal{O}(MK)$ operations.

3.4.4 The oversampled Gabor transform and its inverse

Given the input vector $\mathbf{x}$, the oversampling factor p and the lattice parameters M and N we want to calculate the oversampled Gabor transform of $\mathbf{x}$. The implementation follows directly from the method presented in Subsection 2.4.1.

Algorithm 11 The oversampled Gabor transform.

INPUT: Vector $\mathbf{x}$, oversampling factor p, lattice parameters M and N.
OUTPUT: $M \times K$ matrix $\mathbf{A_{GT}}$ containing the Gabor transform of $\mathbf{x}$.

$K \leftarrow p * N$

Insert (MN-length($\mathbf{x}$)) zeroes at the end of $\mathbf{x}$ making it of length MN

% Use Algorithm 9 to calculate the Zak transform $\tilde{g}[n, l; N, M]$,
% $n \in \{0, ..., K-1\}$ and $l \in \{0, ..., M-1\}$, of the elementary function.
$\mathbf{G_{ZT}} \leftarrow \tilde{g}[n, l; N, M]$

% Use Algorithm 10 to calculate the Zak transform $\tilde{w}[n, l; N, M]$,
% $n \in \{0, ..., K-1\}$ and $l \in \{0, ..., M-1\}$, of the window function.
$\mathbf{W_{ZT}} \leftarrow \tilde{w}[n, l; N, M]$

% Use Algorithm 3 to calculate the Zak transform $\tilde{x}[n, l; K, M/p]$,
% $n \in \{0, ..., K-1\}$ and $l \in \{0, ..., M/p-1\}$, of the signal $\mathbf{x}$.
$\mathbf{X'_{ZT}} \leftarrow x[n, l; K, M/p]$

% Replicate the matrix into a $1 \times p$ block matrix to calculate the Zak
% transform $\tilde{x}[n, l; K, M/p]$, $n \in \{0, ..., K-1\}$ and $l \in \{0, ..., M-1\}$,
% of the signal $\mathbf{x}$.
$\mathbf{X_{ZT}} \leftarrow repmat(\mathbf{X'_{ZT}}, 1, p)$

%Use relation (2.30) to calculate the 2d Fourier transform $\bar{a}[n, l; K, M]$,
% $n \in \{0, ..., K-1\}$ and $l \in \{0, ..., M-1\}$, of the coefficients.
$\mathbf{A}_{FT} \leftarrow K * \mathbf{X_{ZT}} .* \mathbf{W_{ZT}}$

%Use Algorithm 2 to calculate the 2d inverse Fourier transform
% of $\mathbf{A}_{FT}$.
$\mathbf{A_{GT}} \leftarrow$ The 2d inverse Fourier transform of $\mathbf{A}_{FT}$

The inverse operation takes the coefficient matrix $\mathbf{A_{GT}}$ and calculates the vector $\mathbf{x}$. The implementation follows directly from the method presented in Subsec-

tion 2.4.2.

Algorithm 12 The inverse Gabor transform.

INPUT: $M \times K$ matrix $\mathbf{A_{GT}}$ containing the Gabor transform of $\mathbf{x}$.
OUTPUT: Vector $\mathbf{x}$.

% Find the lattice parameters M and K from the dimension of $\mathbf{A_{GT}}$.
$M \leftarrow length(\mathbf{A_{GT}}[:, 1])$
$K \leftarrow length(\mathbf{A_{GT}}[1, :])$

% Use Algorithm 9 to calculate The Zak transform $\tilde{g}[n, l; N, M]$,
% $n \in \{0, ..., K-1\}$ and $l \in \{0, ..., M-1\}$, of the elementary function.
$\mathbf{G_{ZT}} \leftarrow \tilde{g}[n, l; N, M]$

% Use Algorithm 1 to calculate The 2d Fourier transform $\overline{a}[n, l; K, M]$,
% $n \in \{0, ..., K-1\}$ and $l \in \{0, ..., M-1\}$, of $\mathbf{A_{GT}}$.
$\mathbf{A_{FT}} \leftarrow \overline{a}[n, l; K, M]$

% Use relation (2.31) to calculate The Zak transform, $\tilde{x}[n, l; K, M/p]$
% $n \in \{0, ..., K-1\}$ and $l \in \{0, ..., M/p-1\}$ of the vector $\mathbf{x}$.
$l \leftarrow M/p$
$\mathbf{X_{ZT}} \leftarrow K \times M/p$ matrix containing zeroes
for $r = 0$ to $p-1$ **do**
  $\mathbf{X_{ZT}} \leftarrow \mathbf{X_{ZT}} + \mathbf{A_{FT}}(:, r*l+1 : 1 : (r+1)*l) .* \mathbf{G_{ZT}}(:, r*l+1 : 1 : (r+1)*l)$
end for
$\mathbf{X_{ZT}} \leftarrow 1/p * \mathbf{X_{ZT}}$

% Use Algorithm 4 to calculate the inverse Zak transform of $\mathbf{X_{ZT}}$.
$\mathbf{x} \leftarrow$ The inverse Zak transform of $\mathbf{X_{ZT}}$

Computational cost

For the Gabor transform the Zak transform of the elementary function requires $\mathcal{O}(KN \log M)$ operations and the Zak transform of the window function requires $\mathcal{O}(MK)$ operations. The Zak transform of the input vector requires $\mathcal{O}(MN)$ operations and the replication requires $\mathcal{O}(MK)$ operations. The element-by-element multiplication requires $\mathcal{O}(MK)$ operations and the 2d inverse Fourier transform requires $\mathcal{O}(MK \log[\max\{M, K\}])$ operations. This gives a total cost of $\mathcal{O}(MK \log[\max\{M, K\}])$ operations.

Likewise for the inverse Gabor transform the Zak transform of the elementary function requires $\mathcal{O}(KN \log M)$ operations. The 2d Fourier transform of the coefficient matrix requires $\mathcal{O}(MK \log[\max\{M, K\}])$ operations. Inside the for loop, which is repeated p times, we have the element-by-element multiplication which

requires $\mathcal{O}(MN)$ operations. This gives a total cost of $\mathcal{O}(MK)$ operations for the loop. The final inverse Zak transform requires $\mathcal{O}(MN \log N)$ operations, giving a total cost of $\mathcal{O}(MK \log[\max\{M, K\}])$ operations.

Chapter 4

Denoising by thresholding

During the 90's there have been considerably interest in the use of wavelet transforms for the removal of noise from signals and images. The most employed method has been the "WaveShrink" developed by Donoho and Johnstone [10] [12] [13]. This method uses a transform-based thresholding, working in three steps:

- Transform the noisy data into the wavelet domain, i.e. time-scale domain.
- Shrink the resulting coefficients, thereby suppressing those coefficients containing noise.
- Transform back into the original domain.

The method is used for denoising a wide class of signals corrupted by additive white Gaussian noise. It has been successfully applied to 1D and 2D data such as NMR spectra and geophysical data.

In this chapter we develop a denoising method using the Gabor transform instead of the wavelet transform, i.e. instead of transforming the noisy signal into the time-scale domain we use the time-frequency domain. This method is called "GaborShrink" and numerical tests we have done show that for some signals this method performs better than the "WaveShrink" method.

The implementation of the "GaborShrink" is done in Matlab from MathWorks.

4.1 Introduction

Suppose we are given a signal $\mathbf{x} = \{x_0, ..., x_{N-1}\}$ generated from

$$x_n = f_n + \sigma z_n,\ n = 0, ..., N-1, \tag{4.1}$$

where $f_n = f(t_n)$. The z_n's are independently distributed Gaussian random variables with zero mean and variance one, and σ is a known noise level. The z_n's are also referred to as white noise. Our goal is to denoise the signal $\mathbf{x}$, i.e. to find a good estimate $\hat{\mathbf{f}}$ of the underlying signal $\mathbf{f} = \{f(t_0), ..., f(t_{N-1})\}$. The hat on top

of a variable is the notation throughout this chapter to indicate the estimate of the corresponding variable.

The noise signal $\mathbf{x}$ can be represented by a linear combination of Gabor elementary functions $G_{m,k}$, i.e.

$$x[n] = \sum_{m=0}^{M-1} \sum_{k=0}^{K-1} a_{m,k} G_{m,k}[n]$$

Since the energy in white noise is uniformly distributed in the frequency domain it will be reasonable to assume that the noise energy also is uniformly distributed among the Gabor coefficients $a_{m,k}$. If in addition the noise energy is small compared to the signal energy, the magnitude of the noise coefficients will be at a lower level than the magnitude of the signal coefficients. Thus thresholding the coefficients with a well chosen threshold gives a good estimate $\hat{\mathbf{f}}$ of the the underlying signal $\mathbf{f}$, i.e.

$$\hat{f}[n] = \sum_{m=0}^{M-1} \sum_{k=0}^{K-1} a'_{m,k} G_{m,k}[n],$$

where $a'_{m,k}$ are the thresholded coefficients.

When thresholding our goal is to minimize the Mean Square Error

$$R(\mathbf{f}, \hat{\mathbf{f}}) = \frac{1}{N} \sum_{i=0}^{N-1} E(\hat{f}_i - f_i)^2, \tag{4.2}$$

also known as the L_2-RISK. Since we do not know $\mathbf{f}$, this is not a trivial task.

For measuring the "strength" of the noise in (4.1) we use the signal-to-noise ratio defined by

$$SNR = \frac{SD(f_n)}{\sigma},$$

where SD is the standard deviation.

4.2 Thresholding

There are several ways of performing thresholding, we use the two most common methods, namely hard thresholding and soft thresholding.

4.2.1 Hard thresholding

When hard thresholding a function $x(t)$ the function values with an absolute value below or equal to the threshold level δ is replaced by 0, see Figure 4.1. The output y_{hard} is

$$y_{hard} = \begin{cases} x(t) & |x(t)| > \delta \\ 0 & |x(t)| \leq \delta \end{cases} \tag{4.3}$$

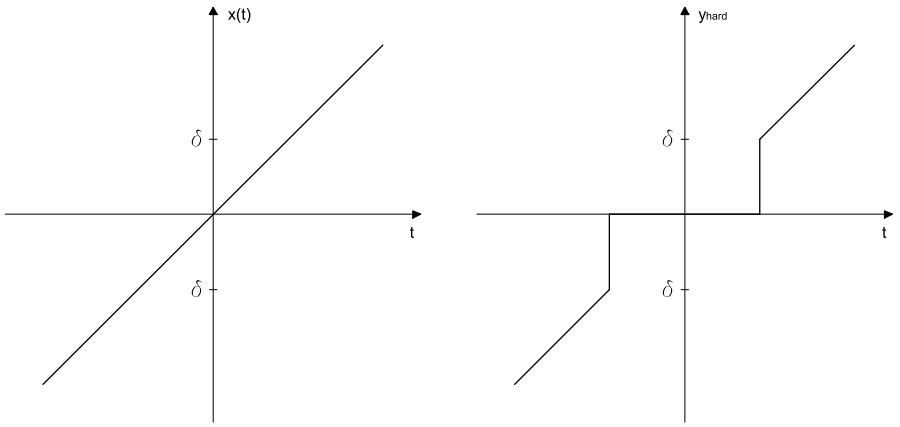

Figure 4.1: Hard thresholding the function $x(t)$.

4.2.2 Soft thresholding

When soft thresholding a function $x(t)$ the function values with an absolute value below or equal to the threshold level δ is replaced by 0, and the function values with an absolute value above the threshold level are shrunk with δ, see Figure 4.2. The output y_{soft} is

$$y_{soft} = \begin{cases} \text{sign}(x(t))(|x(t)| - \delta) & |x(t)| > \delta \\ 0 & |x(t)| \leq \delta \end{cases} \tag{4.4}$$

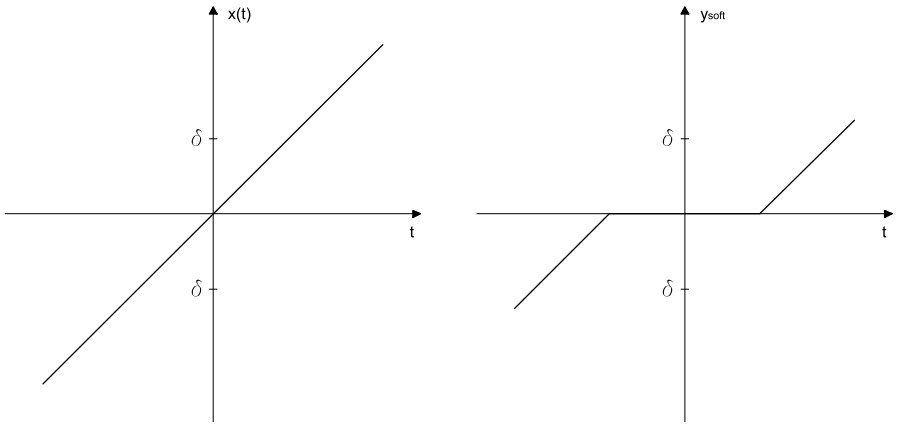

Figure 4.2: Soft thresholding the function $x(t)$.

4.2.3 Thresholding Gabor coefficients

In our case the function values $x(t)$ in (4.3) and (4.4) are the Gabor coefficients $a_{m,k}$. However, there is one problem, the coefficients are complex valued. Thus there are two ways of performing the thresholding. We can either threshold the real coefficients and the imaginary coefficients separately, or we can threshold the absolute values of the complex coefficients. Numerical tests we have done show that we achieve the best results thresholding the absolute values of the complex coefficients. The reason for this is probably that thresholding the real and imaginary coefficients separately alters the phase of the complex coefficients, and this again results in extra noise when the coefficients are transformed back into the time domain. Thus unless specified otherwise we use the absolute values of the complex coefficients for thresholding.

4.2.4 Example

Before we continue to the difficult task, namely selecting the threshold level δ, we present an example. The example is meant for motivation and illustrates the potential of the method.

We take a test signal, add white Gaussian noise and then try to use hard thresholding to reduce the noise. The test signal consists of two damped sinusoids with frequencies 200 Hz and 400 Hz:

$$y_n = \epsilon \cdot e^{-10 \cdot n \cdot \Delta t} \Big(\sin(2\pi \cdot 200 \cdot n \cdot \Delta t) + \sin(2\pi \cdot 400 \cdot n \cdot \Delta t) \Big),$$

where $n \in \{0, ..., 512\}$ and $\Delta t = 1$ms, yielding a sampling frequency of 1kHz. The constant ϵ is the signal level. Figure 4.3 shows the frequency response of the test signal.

We now add white Gaussian noise with mean zero and standard deviation one. The signal level ϵ is chosen so that the signal-to-noise ratio is one. Figure 4.4 shows the frequency response of the signal. We clearly see the noise, which is uniformly distributed in the frequency domain.

We then transform the noisy signal into the time-frequency domain, see the upper part of Figure 4.5. From the figure we see the signal components and the noise, which is uniformly distributed in the time-frequency domain. For a comparison we have added the time-frequency plot of the noise free signal in the lower part of Figure 4.5.

What we will do is to use hard thresholding to remove the noise coefficients. This is possible because the majority of the signal components are larger than the noise coefficients. Since we have available the noise free signal we can find the optimal threshold value from the L_2-RISK (4.2). Plotting the L_2-RISK for different thresholding levels, see Figure 4.6, we find that $\delta = 0.3$ is the optimal threshold value.

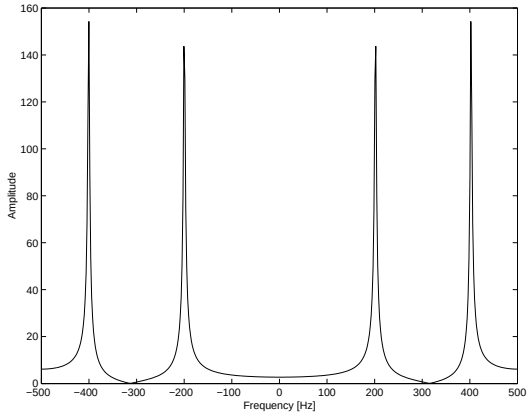

Figure 4.3: Frequency response of the test signal y_n.

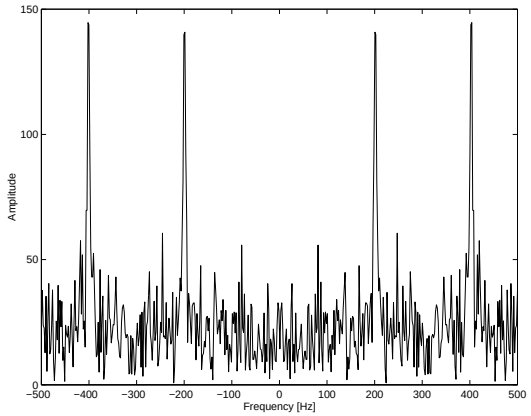

Figure 4.4: Frequency response of the noisy signal.

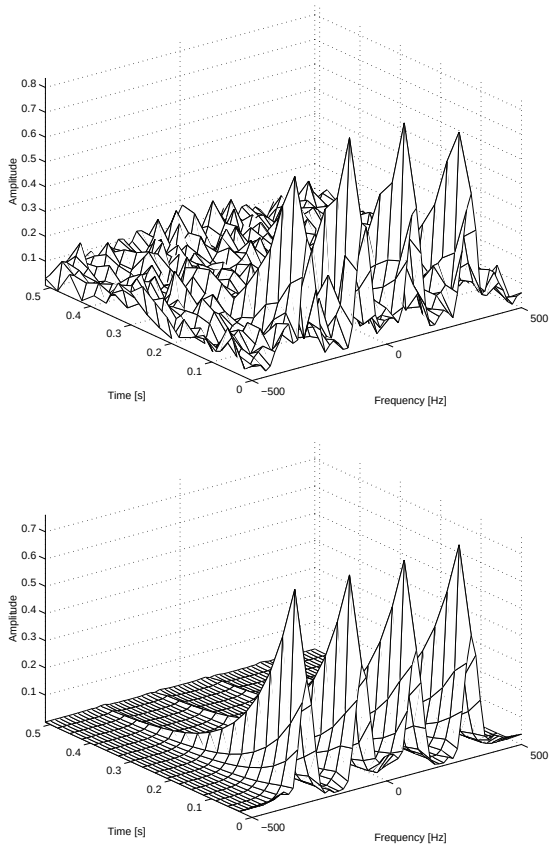

Figure 4.5: Upper: Time-frequency plot of the noisy signal. Lower: Time-frequency plot of the noise free signal.

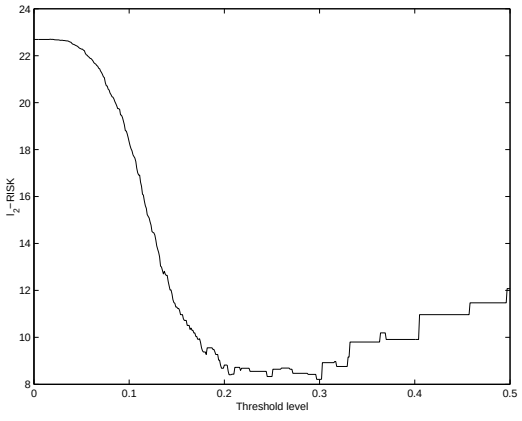

Figure 4.6: L_2-RISK versus threshold level.

Thresholding the coefficients at this level we get the time-frequency plot in Figure 4.7. The noise coefficients are removed, but we have also lost the smallest signal coefficients. This is not possible to avoid since these coefficients were "drowned" in the noise. However, the majority of the signal components still remains. Using the inverse transform on these coefficients we get the most optimal denoised signal using hard thresholding. Figure 4.8 shows the frequency response of the denoised signal. From the figure we see that the majority of the noise is removed.

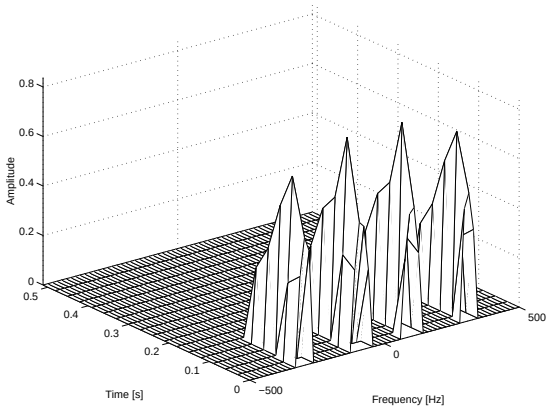

Figure 4.7: Time-frequency plot after the coefficients have been hard thresholded.

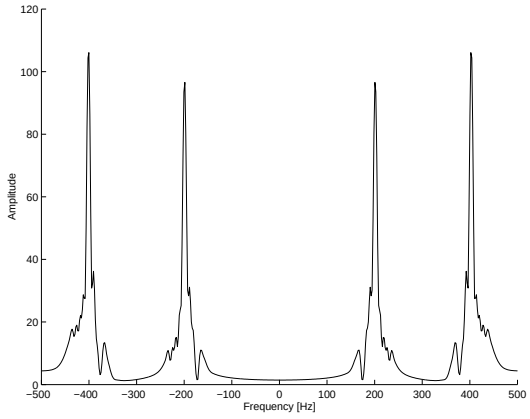

Figure 4.8: Frequency response of the optimal denoised signal.

4.3 Selecting the threshold

The central issue in a threshold procedure is the selection of an appropriate threshold. If this threshold is too small, the result is still noisy. On the other hand, a large threshold also removes signal coefficients. It is intuitively clear that higher noise level requires higher threshold. We distinguish between local threshold, which compute one threshold for a group of coefficients, and global threshold which compute one threshold for all the coefficients. In addition we classify the procedure as data-driven if the threshold selection also depends on the input signal.

We investigate two methods. The first one, which we have developed ourselves, is global and uses the statistical properties of the noise for selecting the threshold. The second one uses Steins unbiased RISK estimate. This method is global and data-driven.

4.3.1 Statistical method

The noise is assumed to be white, i.e. uniformly distributed in the frequency domain. This can be modeled using normal distributed random variables. The density function of the normal distributed random variable is

$$f_X(x) = \frac{1}{\sqrt{2\pi}\sigma} e^{-\frac{(x-\mu)^2}{2\sigma^2}},$$

where μ is the mean and σ is the standard deviation. The normal distribution is denoted $N(\mu, \sigma)$ and its graph, called the normal curve, is the bell-shaped curve of Figure 4.9. The normal distribution is often referred to as Gaussian distribution, in honor of Karl Friedrich Gauss (1777-1855). The particular normal distribution that has a mean of 0 and a standard deviation of 1 is called the standard normal distribution, and is denoted $N(0, 1)$. For modeling white noise we use scaled standard normal distributed random variables.

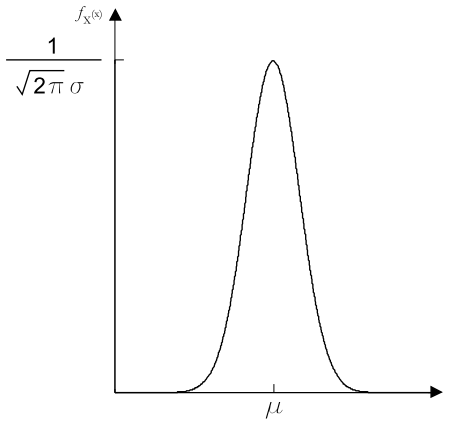

Figure 4.9: The normal distribution with mean μ and standard deviation σ.

The idea behind the method is fairly simple. We want to find the level where we statistically erase a given percentage of the noise coefficients. To do this we have to examine the standard normal distribution and how the noise energy is distributed in the time-frequency domain.

The curve of any continuous probability distribution is constructed such that the area under the curve bounded by two ordinates $x = x_1$ and $x = x_2$ equals the probability that the random variable X assumes a value between $x = x_1$ and $x = x_2$. Thus for the standard normal distribution the probability that the value of the random variable is between $x = x_1$ and $x = x_2 = -x_1$ is given by

$$P(|X| < x_1) = 2\int_0^{x_1} f_X(x)dx = \frac{2}{\sqrt{2\pi}}\int_0^{x_1} e^{-\frac{x^2}{2}}dx = \frac{1}{\sqrt{2}} \cdot erf(x_1), \quad (4.5)$$

where $erf(x_1)$ is the error function.

Since we are interested in finding the level where a given percentage of the noise coefficients are below we need the inverse relation of (4.5), i.e. given a probability p we want to find the ordinate $x = x_1$. Using the inverse error function this can be found using the relation

$$x_1 = \sqrt{2} \cdot inverf(p). \quad (4.6)$$

For example for a probability $p = 0.95$ the ordinate is $x_1 = 1.96$. This means that for a series of uncorrelated random variables (white noise) approximately 95% of them have an absolute value smaller than 1.96, see Figure 4.10. The calculation of the $inverf(p)$ function has to be done numerically.

If the random variable X does not have a standard normal distribution, but has zero mean and standard deviation σ, we have to scale (4.6) by σ. This gives the following important fact that we will use later:

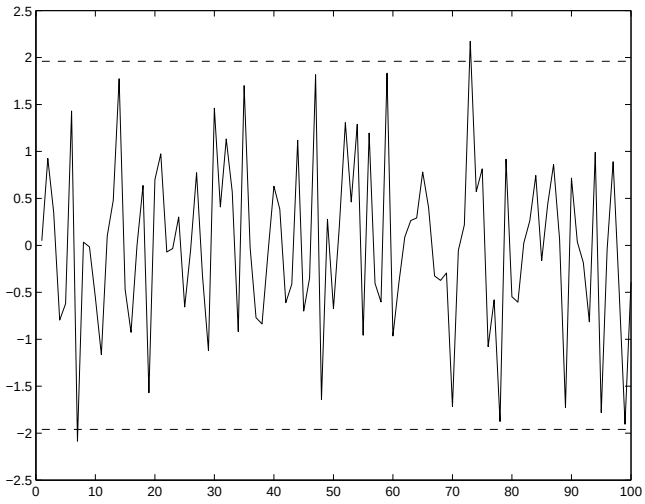

Figure 4.10: One hundred uncorrelated normal distributed random variables. Statistically 95% of them have an absolute value smaller than 1.96

- For a series of uncorrelated random variables with zero mean and standard deviation σ statistically $100 \cdot p$ % of them will be smaller than the level t_X defined by

$$t_X = \sigma \cdot \sqrt{2} \cdot inverf(p). \tag{4.7}$$

Next we have to examine how the noise energy is distributed in the time-frequency domain. Since the Gabor transform is not an orthogonal transform this is not a trivial task. It is hard to derive exact equations for the relation between the energy in the time domain and the energy in the time-frequency domain. We instead formulate approximations, which are enough accurate for our usage.

Since the energy in white noise is uniformly distributed in the frequency domain it will be reasonable to assume that the energy is almost equally distributed among the frequency bands. There are K frequency bands, where frequency band i is defined by

$$\beta_i = \{a_{0,i}, a_{1,i}, ..., a_{M-1,i}\}, \qquad i \in \{0, ..., K-1\},$$

and the coefficients $a_{m,k}$ is defined by (2.28).

Numerical tests shows that, with a small error, the assumption that the energy is equally distributed among the frequency bands is true. In fact, if we have the white noise signal

$$x_w = \sigma_w \cdot N(0, 1),$$

and transform the signal into the time-frequency domain, we have the following relation

$$||x_w||_2 \sim K||\beta_i||_2 \qquad i \in \{0, ..., K-1\}. \tag{4.8}$$

This means that the norm of each frequency band is approximately the norm of the white noise divided by K.

The energy in frequency band β_i is defined by

$$\mathsf{E}_{\beta_i} = ||\beta_i||_2^2.$$

Using this and (4.8) we have the following relation

$$||x_w||_2^2 \sim K^2\mathsf{E}_{\beta_i} \qquad i \in \{0, ..., K-1\}, \tag{4.9}$$

meaning that the energy in each frequency band is approximately the energy in the white noise divided by K^2.

From (4.9) we can draw the two following important relations:

- When the number of frequency bands are increased by a factor of two, the energy in each frequency band, E_{β_i}, is decreased by a factor of four.
- When the number of time steps are increased by any factor, the energy in each frequency band, E_{β_i}, remains the same.

We have to point out one more fact about the energy in each frequency band. Since the Gabor coefficients are complex the energy is divided between the real coefficients and the imaginary coefficients. In fact, approximately half of the energy is in the real coefficients and the other half is in the imaginary coefficients. Defining $\mathsf{E}_{\beta_i}^R$ as the energy in real coefficients and $\mathsf{E}_{\beta_i}^I$ as the energy in the imaginary coefficients we have the relation

$$\mathsf{E}_{\beta_i}^R \sim \mathsf{E}_{\beta_i}^I \sim \frac{1}{2}\mathsf{E}_{\beta_i} = \frac{||x_w||_2^2}{2K^2}. \tag{4.10}$$

It should be mentioned that this is not true for every frequency band. Because of the circularity of the complex exponential function we have p frequency bands, where p is the oversampling factor, which only consists of real coefficients. However, the error in the final calculated threshold level made by assuming that every frequency band contains complex coefficients is very small. Thus we assume that (4.10) holds for every frequency band.

Next we have to examine the relation between the noise level of the white noise, σ, and the noise level of transformed white noise. We have from statistics (see [29] sentence 10.5) that when $X_0, ..., X_{N-1}$ are independent random variables with mean μ and standard deviation σ the summation

$$\frac{1}{\sigma^2}\sum_{n=0}^{N-1}(X_n - \mu)^2$$

is a chi-squared distribution or χ^2 distribution with N degrees of freedom. Thus if the random variables have zero mean we have that

$$\frac{1}{\sigma^2}\sum_{n=0}^{N-1} X_n^2 \tag{4.11}$$

is a χ^2 distribution with N degrees of freedom. In addition we have that (see [29] eq. 10.5) if a random variable Z has a χ^2 distribution with v degrees of freedom the expected value of Z is

$$E(Z) = v. \tag{4.12}$$

Using (4.12) and the fact that (4.11) is a χ^2 distribution we have that

$$E(\frac{1}{\sigma^2}\sum_{n=0}^{N-1} X_n^2) = N,$$

or

$$\sigma^2 = \frac{E(\sum_{i=1}^{N-1} X_i^2)}{N}.$$

The summation of the squared random variables is the energy of the random variables. Using this and (4.10) we can find the variance of the real coefficients in each frequency band as

$$\sigma_R^2 = \frac{E(\mathsf{E}_{\beta_i}^R)}{M} = \frac{\frac{||x_w||_2^2}{2K^2}}{M},$$

where M is the number of real coefficients in each frequency band. The energy in the noise signal $||x_w||_2^2$ can equally be written as

$$||x_w||_2^2 = MN\sigma_w^2,$$

where MN is the length of the noise signal. Using this we get

$$\sigma_R^2 = \frac{\frac{MN\sigma_w^2}{2K^2}}{M} = \frac{N\sigma_w^2}{2K^2}.$$

Taking the square root of both sides we get the standard deviation of the the real coefficients in each frequency band

$$\sigma_R = \sqrt{\frac{N}{2}}\frac{\sigma_w}{K}.$$

Since the energy is equally divided between the real and imaginary coefficients we have that standard deviation of the imaginary coefficients equals the standard deviation of the real coefficients, thus

$$\sigma_R = \sigma_I = \sqrt{\frac{N}{2}}\frac{\sigma_w}{K}. \tag{4.13}$$

If we now combine (4.7) and (4.13) we have that approximately $100 \cdot p$ % of the absolute values of the real coefficients and $100 \cdot p$ % of the absolute values of the imaginary coefficients are smaller than the level t_{ri} defined by

$$t_{ri} = \frac{\sqrt{N}\sigma_w}{K} \cdot inverf(p). \tag{4.14}$$

As explained in Section 4.2.3, the most optimal result is achieved when using the absolute values of the complex coefficients for the thresholding. Using (4.14) we have that approximately $100 \cdot p$ % of the absolute values of the coefficients are smaller than

$$t_a = \sqrt{t_{ri}^2 + t_{ri}^2} = \sqrt{2}t_{ri} = \frac{\sqrt{2}\sqrt{N}}{K}\sigma_w \cdot inverf(p). \tag{4.15}$$

This is the relation we have been seeking. Using (4.15) we can find an appropriate threshold level for the coefficients. Numerical tests we have done show that we achieve the best results by choosing $p = 0.99$ for hard thresholding and $p = 0.75$ for soft thresholding. This gives the following threshold levels:

Hard thresholding

$$\delta_{hard} = \frac{\sqrt{2}\sqrt{N}}{K}\sigma_w \cdot inverf(0.99) = 2.5758 \cdot \frac{\sqrt{N}}{K}\sigma_w \tag{4.16}$$

Soft thresholding

$$\delta_{soft} = \frac{\sqrt{2}\sqrt{N}}{K}\sigma_w \cdot inverf(0.75) = 1.1503 \cdot \frac{\sqrt{N}}{K}\sigma_w \tag{4.17}$$

4.3.2 Threshold selection by SURE

A well known threshold selector used in wavelet denoising is Steins unbiased RISK estimator, also known as SURE. In this section we try to adapt this method to Gabor denoising. For an estimator to be unbiased it requires that on the average the estimates will yield the true value.

Given a signal $\mathbf{x} = \{x_0, ..., x_{N-1}\}$ generated from

$$x_n = f_n + \sigma z_n,\ n = 0, ..., N-1,$$

where $f_n = f(t_n)$. The z_n's are independently distributed Gaussian random variables with zero mean and variance one and σ is a known noise level. Let $\hat{\mathbf{f}} = \{\hat{f}_1, ..., \hat{f}_N\}$ be an estimator of $\mathbf{f} = \{f_0, ..., f_{N-1}\}$. Introduce the mean squared risk of $\hat{\mathbf{f}}$:

$$R = \sum_{n=0}^{N-1} E(\hat{f}_n - f_n)^2.$$

Assume that the estimators $\hat{f}_n$ have the form

$$\hat{f}_n = x_n + H_\delta(x_n),$$

where δ is a threshold level and $H_\delta(\cdot)$ is a weakly differentiable real valued function for any fixed δ.

Since the true parameters f_n are unknown we can not calculate R explicitly. Charles Stein [22] introduced a method for estimating the risk unbiased. Stein showed that when $H_\delta(\cdot)$ is weakly differentiable, then

$$R = R(\sigma, \mathbf{x}, \delta) = \sum_{n=0}^{N-1} \sigma^2 + 2\sigma^2 \frac{d}{dx} H_\delta(x)\Big|_{x=x_n} + H_\delta^2(x_n). \tag{4.18}$$

This is known as Steins unbiased risk estimate (SURE). The Stein principle is to minimize (4.18) with respect to δ and take the minimizer as a data driven estimator of the optimal δ, i.e.

$$\hat{\delta} = \min_{\delta \geq 0} R(\sigma, \mathbf{x}, \delta).$$

It was our intention to use Steins unbiased risk estimate for both hard and soft thresholding, however, the hard threshold function is not continuous, and therefore it does not have a bounded weak derivative in Steins sense [16]. Thus the SURE procedure for hard thresholding is not valid and we can only apply the SURE procedure for the soft thresholding.

Soft thresholding

For soft thresholding the threshold function $H_\delta(x_n)$ must be $-x_n$ when $|x_n| < \delta$, $-\delta$ when $x_n \geq \delta$ and δ when $x_n \leq \delta$, see Figure 4.11, this gives the function

$$H_\delta(x_n) = -x_n I(|x_n| < \delta) - \delta I(|x_n| \geq \delta)\text{sign}(x_n),$$

where

$$I(|x_n| < \delta) = \begin{cases} 1 & |x_n| < \delta \\ 0 & \text{else} \end{cases}$$

and

$$\text{sign}(x_n) = \begin{cases} 1 & x_n > 0 \\ 0 & x_n \leq 0 \end{cases}.$$

The derivative of $H_\delta(x_n)$ is

$$\frac{d}{dx} H_\delta(x)\Big|_{x=x_n} = -I(x_n| < \delta),$$

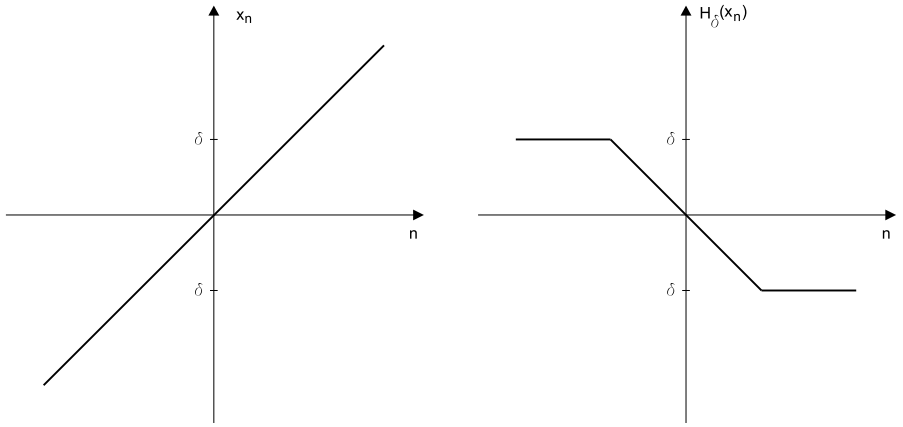

Figure 4.11: Soft thresholding.

which gives the SURE function

$$\begin{aligned} R(\sigma, x_n, \delta) &= \sum_{n=0}^{N-1} \sigma^2 - 2\sigma^2 I(|x_n| < \delta) + \delta^2 I(|x_n| \geq \delta) + x_n^2 I(|x_n| < \delta) \\ &= \sum_{n=0}^{N-1} \sigma^2 + (x_n^2 - 2\sigma^2) I(|x_n| < \delta) + \delta^2 I(|x_n| \geq \delta) \\ &= \sum_{n=0}^{N-1} (x_n^2 - \sigma^2) I(|x_n| < \delta) + (\sigma^2 + \delta^2) I(|x_n| \geq \delta) \end{aligned}$$

Minimizing this function with respect to δ yields the estimate of the optimal δ for soft thresholding.

Thresholding Gabor coefficients

As mentioned in Section 4.2.3 we achieve the best result when thresholding the absolute values of the complex coefficients. However, the SURE-function requires the noise to be Gaussian distributed with zero mean, something the absolute values of the complex coefficients do not fulfill. Thus to use Steins principle we have to threshold the real coefficients and imaginary coefficients separately. This works because separately the noise in the real and imaginary coefficients are Gaussian distributed with zero mean and variance σ given by (4.13).

This gives the following threshold levels:

Soft thresholding

Real coefficients:

$$\delta^R_{soft} = \min_{\delta \geq 0} \sum_{m=0}^{M-1} \sum_{k=0}^{K-1} \Big[\big((real(a_{m,k}))^2 - \sigma^2 \big) I(|real(a_{m,k})| < \delta) + (\sigma^2 + \delta^2) I(|real(a_{m,k})| \geq \delta) \Big]$$

Imaginary coefficients:

$$\delta^I_{soft} = \min_{\delta \geq 0} \sum_{m=0}^{M-1} \sum_{k=0}^{K-1} \Big[\big((imag(a_{m,k}))^2 - \sigma^2 \big) I(|imag(a_{m,k})| < \delta) + (\sigma^2 + \delta^2) I(|imag(a_{m,k})| \geq \delta) \Big]$$

4.3.3 Unknown noise level

What if the noise level is unknown? It is possible to estimate the noise level from the median of the coefficients in the highest frequency bands. We have done a few numerical tests with good results, however, more research is needed.

4.4 Simulation experiments

In this section we run simulation experiments to investigate the performance of the methods we have discussed. For the experiments we use six functions often used for testing wavelet denoising. The six functions are due to Donoho and Johnstone [11]: Bumps, HeaviSine, Doppler, Blocks, Quadchirp and Mishmash. The functions are normalized such that their standard deviation equals 7, and sampled with four different sample sizes. For each signal and sample size we add 100 different white noise signals $N(0, 1)$. This gives 100 test signals, with a signal-to-noise ratio of 7, for each signal and sample size. These test signals are denoised and (4.2) is used to calculate the Mean Square Error for each test signal. The frequency resolution M is set 16 and double oversampling is used. For the denoising we test both hard and soft thresholding. For hard thresholding the statistical method is used for estimating the threshold levels, and for soft thresholding both the statistical method and the SURE method are used for estimating the threshold levels. For both hard and soft thresholding the Mean Square Error for each test signal is compared against the optimal Mean Square Error, which is the Mean Square Error achieved using the optimal threshold level. This is possible to calculate since we have the noise free signals. It is important to be aware of that when speaking of the optimal Mean Square Error we do not mean the most optimal Mean Square Error we can achieve using Gabor denoising. It is only the most optimal Mean Square

Error for the given resolution when hard and soft thresholding are used, and all coefficients are thresholded using the same level. There exists other techniques like Garrote shrinkage [5], Firm shrinkage [17] and cycle spinning [8]. These methods should be tested in further work.

4.4.1 Bumps

The Bumps function is defined by the equation

$$f(t) = \sum_{j=0}^{10} h_j K((t - t_j)/w_j), \qquad K(t) = (1 + |t|^4)^{-1}.$$
$$t_j = \{.1, .13, .15, .23, .25, .40, .44, .65, .76, .78, .81\},$$
$$h_j = \{4, 5, 3, 4, 5, 4.2, 2.1, 4.3, 3.1, 5.1, 4.2\},$$
$$w_j = \{.005, .005, .006, .01, .01, .03, .01, .01, .005, .008, .005\}.$$

The Bumps function is normalized such that the standard deviation equals 7, and sampled with four different sample sizes, namely $N_s = 128$, $N_s = 512$, $N_s = 2048$ and $N_s = 8192$, see Figure 4.12. For each sampled signal we add 100 different white noise signals $N(0, 1)$, resulting in 100 test signals for each sample size, see Figure 4.13. For each sample size we denoise each of the 100 test signals as explained in the beginning of this section.

The Mean Square Errors for the denoised test signals using hard thresholding are plotted in Figure 4.14 and the Mean Square errors for the denoised test signals using soft thresholding are plotted in Figure 4.15. The average Mean Square Errors for the hard thresholding are presented in Table 4.1 and the average Mean Square Errors for the soft thresholding are presented in Table 4.2.

From Figure 4.14 and Figure 4.15 we see that the statistical method performs very well. For hard thresholding the Mean Square Errors are almost identical to the optimal Mean Square Errors. The results for soft thresholding are also good, but the Mean Square Errors differs a little more for $N_s = 8192$. The SURE method also performs well, but except for $N_s = 8192$ the statistical method performs better. Comparing Table 4.1 and Table 4.2 we see that the results for hard thresholding are better than the one for soft thresholding. Thus we achieve the best results using hard thresholding, and estimating the thresholds levels with the statistical method. In Figure 4.16 we have plotted the denoised signals using hard thresholding and the statistical method for estimating the threshold levels.

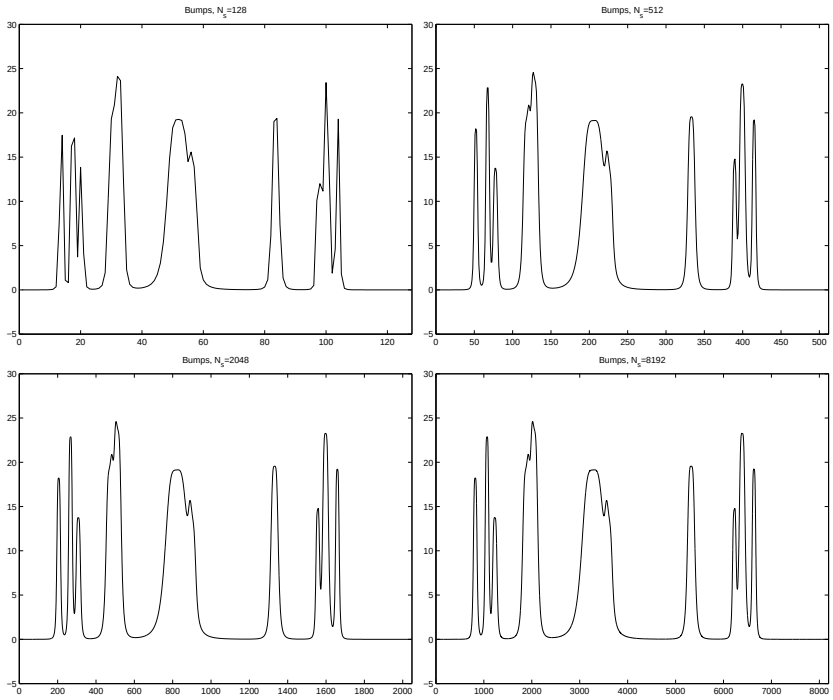

Figure 4.12: The Bumps function with sample size $N_s = 128$, $N_s = 512$, $N_s = 2048$ and $N_s = 8192$.

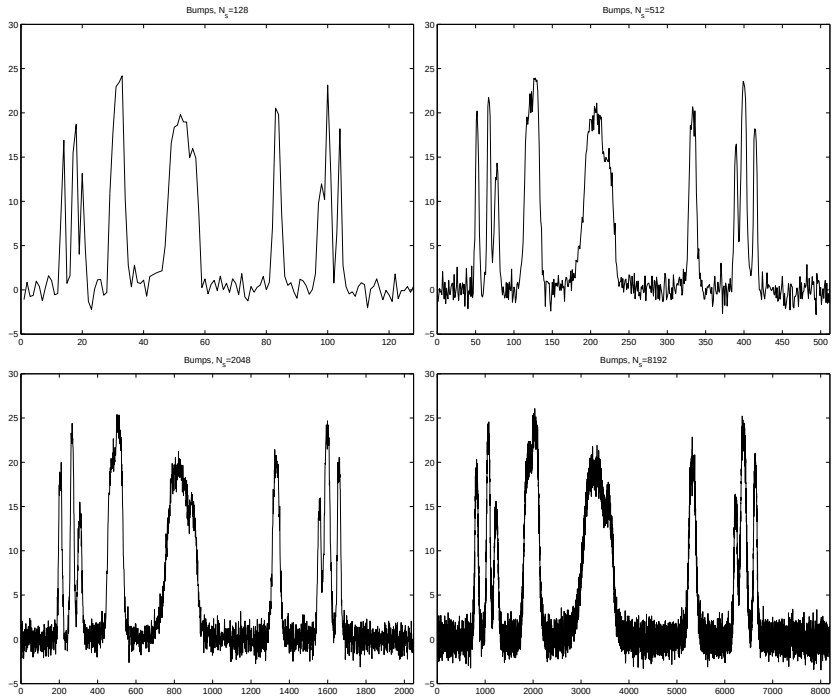

Figure 4.13: The Bumps function with sample size $N_s = 128$, $N_s = 512$, $N_s = 2048$ and $N_s = 8192$, and added white noise $N(0, 1)$. In the simulation 100 different white noise signals are used for each sample size.

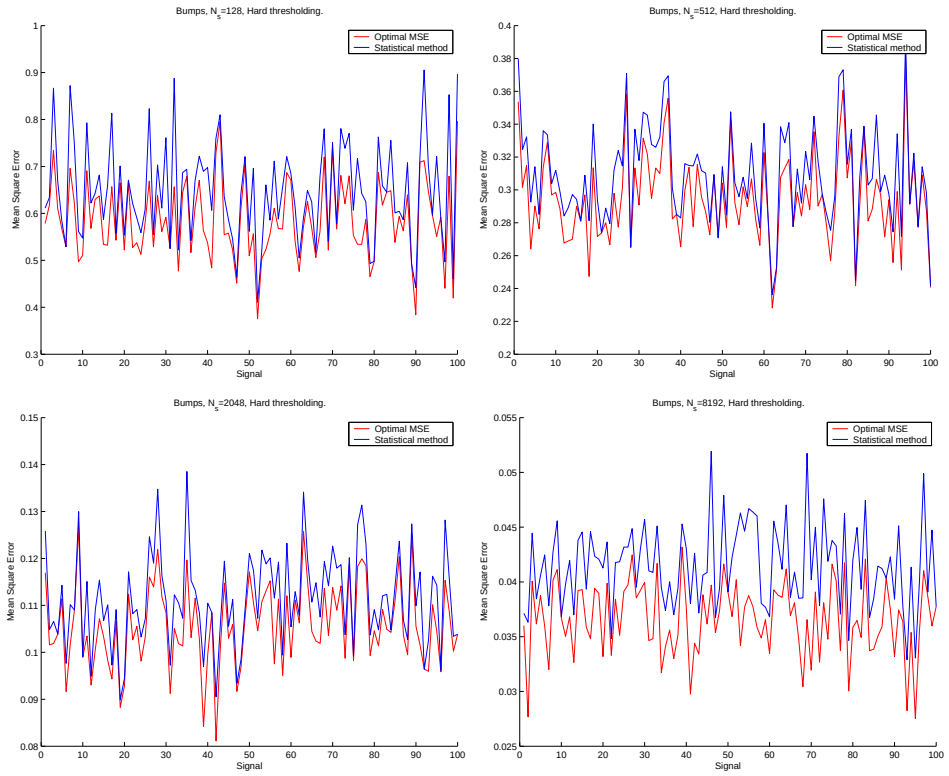

Figure 4.14: The Mean Square Errors using hard thresholding on the noisy Bumps signals. We have used four different sample sizes, and the Mean Square Errors for the statistical method are plotted against the optimal Mean Square Errors.

Sample size	Optimal MSE	Statistical method
$N_s = 128$	0.59	0.65
$N_s = 512$	0.30	0.31
$N_s = 2048$	0.11	0.11
$N_s = 8192$	0.037	0.042

Table 4.1: The average Mean Square Errors using hard thresholding on the noisy Bumps signals.

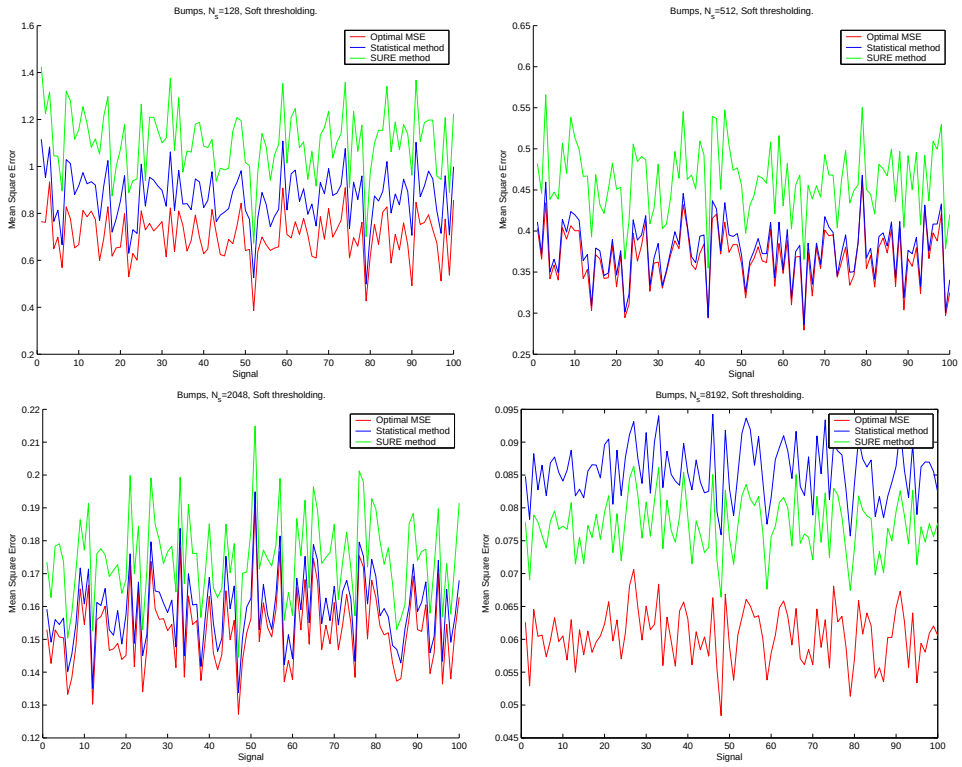

Figure 4.15: The Mean Square Errors using soft thresholding on the noisy Bumps signals. We have used four different sample sizes, and the Mean Square Errors for the statistical method and the SURE method are plotted against the optimal Mean Square Errors.

Sample size	Optimal MSE	Statistical method	SURE method
$N_s = 128$	0.71	0.87	1.1
$N_s = 512$	0.37	0.38	0.46
$N_s = 2048$	0.15	0.16	0.18
$N_s = 8192$	0.061	0.086	0.077

Table 4.2: The average Mean Square Errors using soft thresholding on the noisy Bumps signals.

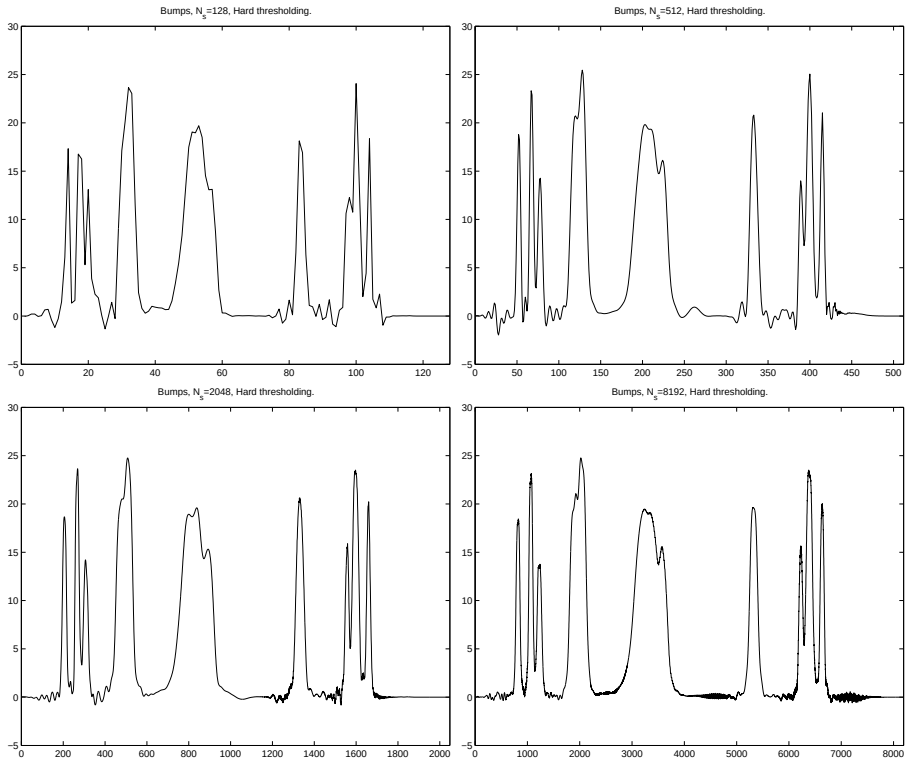

Figure 4.16: The denoised Bumps signals using hard thresholding and estimating the threshold levels using the statistical method.

4.4.2 HeaviSine

The Heavisine function is defined by the equation

$$f(t) = 4\sin(4\pi t) - \text{sign}(t - .3) - \text{sign}(.72 - t).$$

The HeaviSine function is normalized such that the standard deviation equals 7, and sampled with four different samplesizes, namely $N_s = 128$, $N_s = 512$, $N_s = 2048$ and $N_s = 8192$, see Figure 4.17. For each sampled signal we add 100 different white noise signals $N(0, 1)$, resulting in 100 test signals for each sample size, see Figure 4.18. For each sample size we denoise each of the 100 test signals as explained in the beginning of this section.

The Mean Square Errors for the denoised test signals using hard thresholding are plotted in Figure 4.19 and the Mean Square errors for the denoised test signals using soft thresholding are plotted in Figure 4.20. The average Mean Square Errors for the hard thresholding are presented in Table 4.3 and the average Mean Square Errors for the soft thresholding are presented in Table 4.4.

The results are almost identical to those for the Bumps function. From Figure 4.19 and Figure 4.20 we see that the statistical method performs very well. For

hard thresholding the Mean Square Errors are almost identical to the optimal Mean Square Errors. The results for soft thresholding are also good, but the Mean Square Errors differs a little more for $N_s = 8192$. The SURE method also performs well, but except for $N_s = 8192$ the statistical method performs better. Comparing Table 4.3 and Table 4.4 we see that the results for hard thresholding are better than the one for soft thresholding. Thus we achieve the best results using hard thresholding, and estimating the thresholds levels with the statistical method. In Figure 4.21 we have plotted the denoised signals using hard thresholding and the statistical method for estimating the threshold levels.

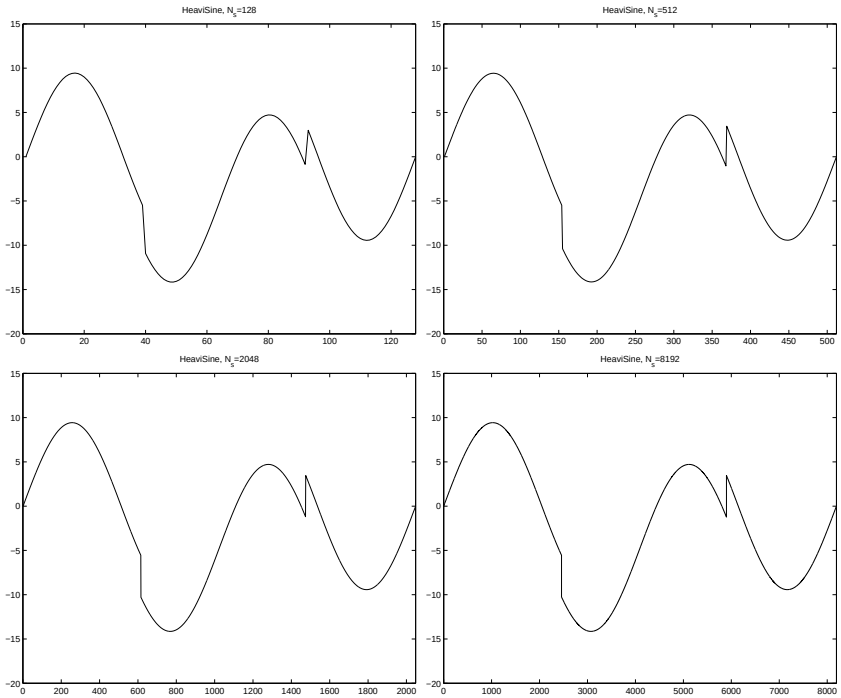

Figure 4.17: The HeaviSine function with sample size $N_s = 128$, $N_s = 512$, $N_s = 2048$ and $N_s = 8192$.

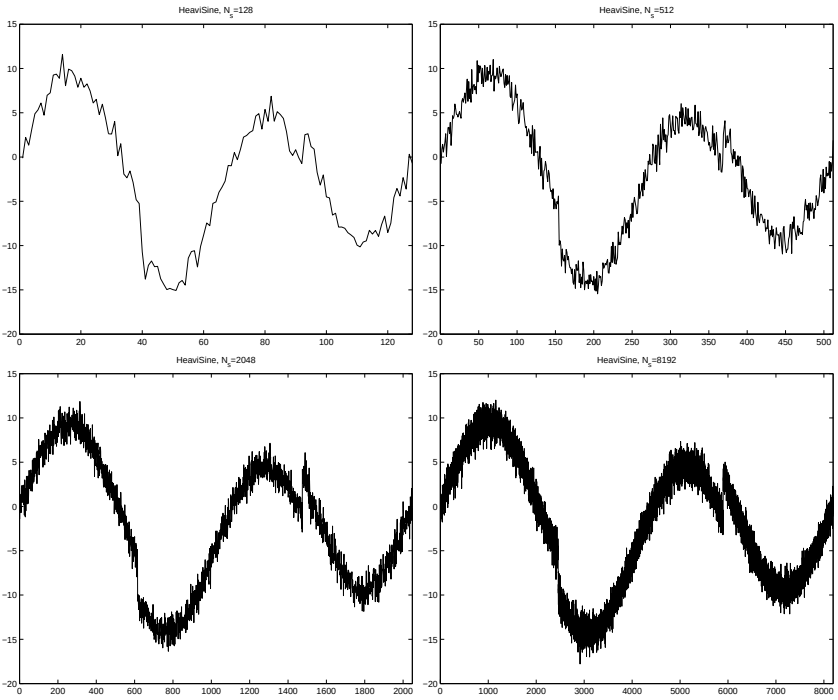

Figure 4.18: The HeaviSine function with sample size $N_s = 128$, $N_s = 512$, $N_s = 2048$ and $N_s = 8192$, and added white noise $N(0, 1)$. In the simulation 100 different white noise signals are used for each sample size.

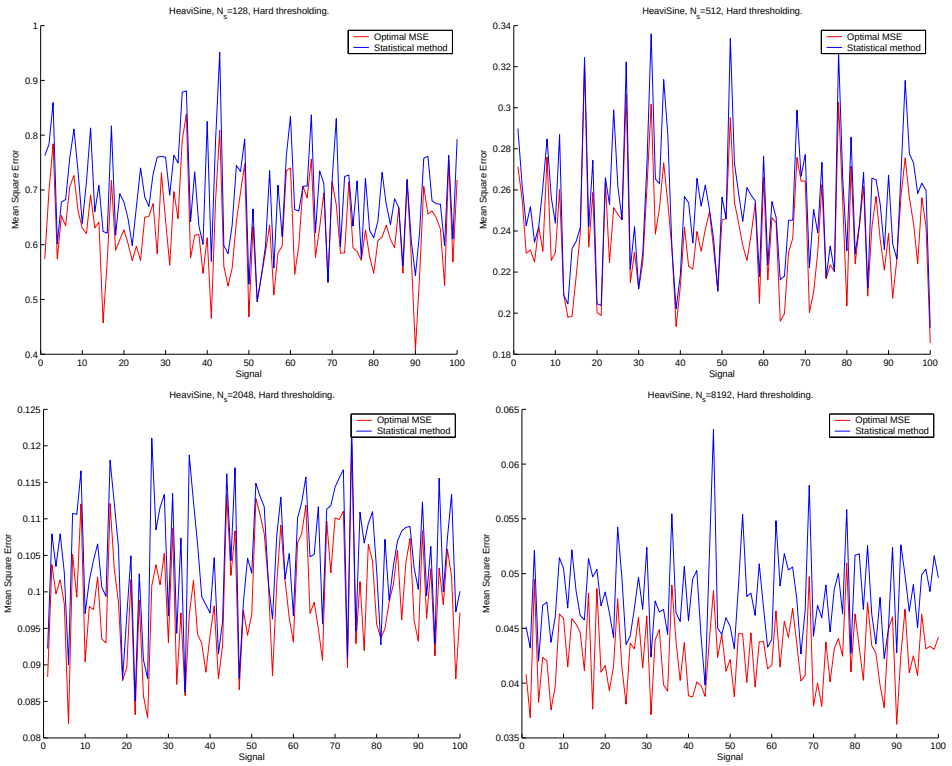

Figure 4.19: The Mean Square Errors using hard thresholding on the noisy HeaviSine signals. We have used four different sample sizes, and the Mean Square Errors for the statistical method are plotted against the optimal Mean Square Errors.

Sample size	Optimal MSE	Statistical method
$N_s = 128$	0.63	0.69
$N_s = 512$	0.24	0.25
$N_s = 2048$	0.10	0.10
$N_s = 8192$	0.043	0.048

Table 4.3: The average Mean Square Errors using hard thresholding on the noisy HeaviSine signals.

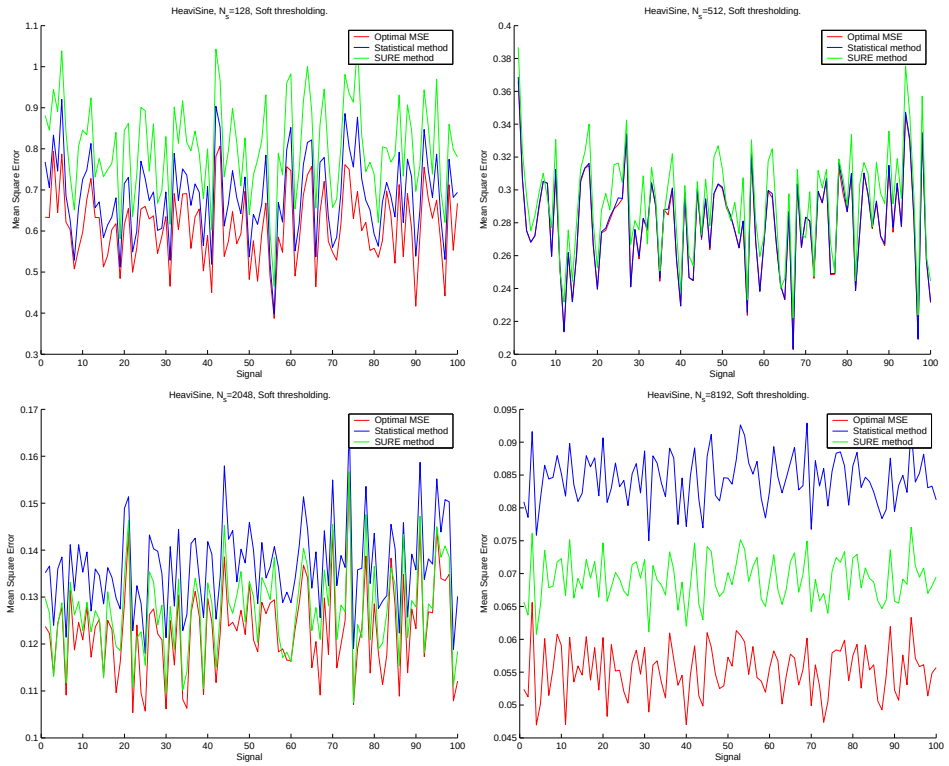

Figure 4.20: The Mean Square Errors using soft thresholding on the noisy HeaviSine signals. We have used four different sample sizes, and the Mean Square Errors for the statistical method and the SURE method are plotted against the optimal Mean Square Errors.

Sample size	Optimal MSE	Statistical method	SURE method
$N_s = 128$	0.61	0.69	0.80
$N_s = 512$	0.28	0.28	0.29
$N_s = 2048$	0.12	0.14	0.13
$N_s = 8192$	0.055	0.084	0.070

Table 4.4: The average Mean Square Errors using soft thresholding on the noisy HeaviSine signals.

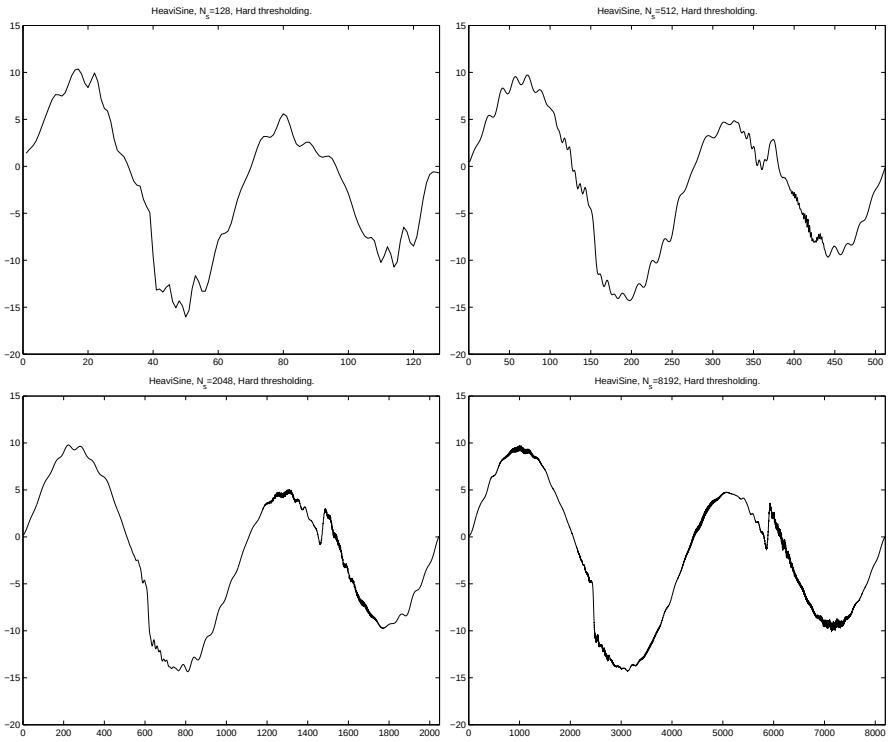

Figure 4.21: The denoised HeaviSine signals using hard thresholding and estimating the threshold levels using the statistical method.

4.4.3 Doppler

The Doppler function is defined by the equation

$$f(t) = \sqrt{t(1-t)}\sin(2\pi(1+\epsilon)/(t+\epsilon)), \qquad \epsilon = .05.$$

The Doppler function is normalized such that the standard deviation equals 7, and sampled with four different samplesizes, namely $N_s = 128$, $N_s = 512$, $N_s = 2048$ and $N_s = 8192$, see Figure 4.22. For each sampled signal we add 100 different white noise signals $N(0,1)$, resulting in 100 test signals for each sample size, see Figure 4.23. For each sample size we denoise each of the 100 test signals as explained in the beginning of this section.

The Mean Square Errors for the denoised test signals using hard thresholding are plotted in Figure 4.24 and the Mean Square errors for the denoised test signals using soft thresholding are plotted in Figure 4.25. The average Mean Square Errors for the hard thresholding are presented in Table 4.5 and the average Mean Square Errors for the soft thresholding are presented in Table 4.6.

The results are almost identical to those from the Bumps and HeaviSine functions. From Figure 4.24 and Figure 4.25 we see that the statistical method performs

very well. For hard thresholding the Mean Square Errors are almost identical to the optimal Mean Square Errors. The results for soft thresholding are also good, but the Mean Square Errors differs a little more for $N_s = 8192$. The SURE method also performs well, but except for $N_s = 8192$ the statistical method performs better. Comparing Table 4.5 and Table 4.6 we see that the results for hard thresholding are better than the one for soft thresholding. Thus we achieve the best results using hard thresholding, and estimating the thresholds levels with the statistical method. In Figure 4.26 we have plotted the denoised signals using hard thresholding and the statistical method for estimating the threshold levels.

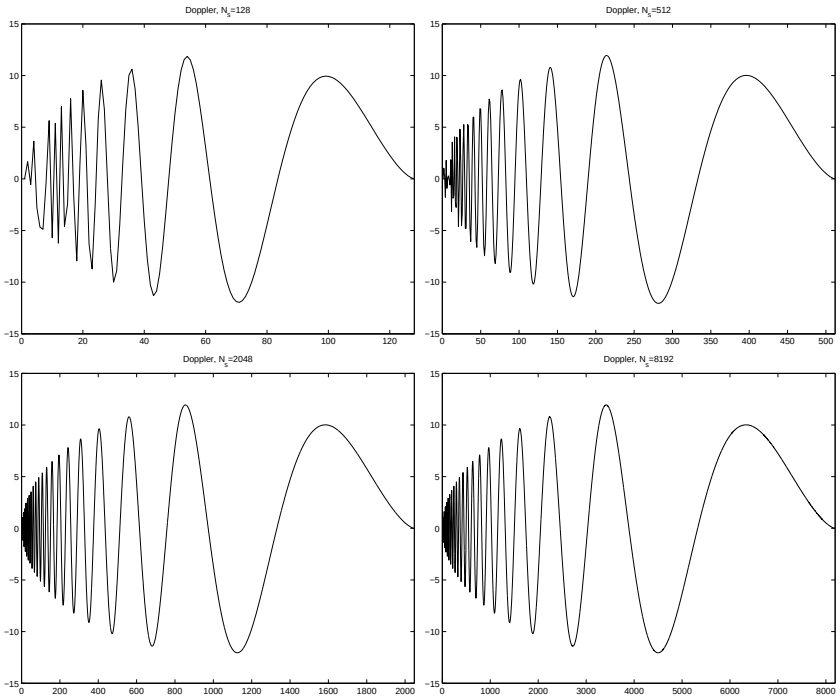

Figure 4.22: The Doppler function with sample size $N_s = 128$, $N_s = 512$, $N_s = 2048$ and $N_s = 8192$.

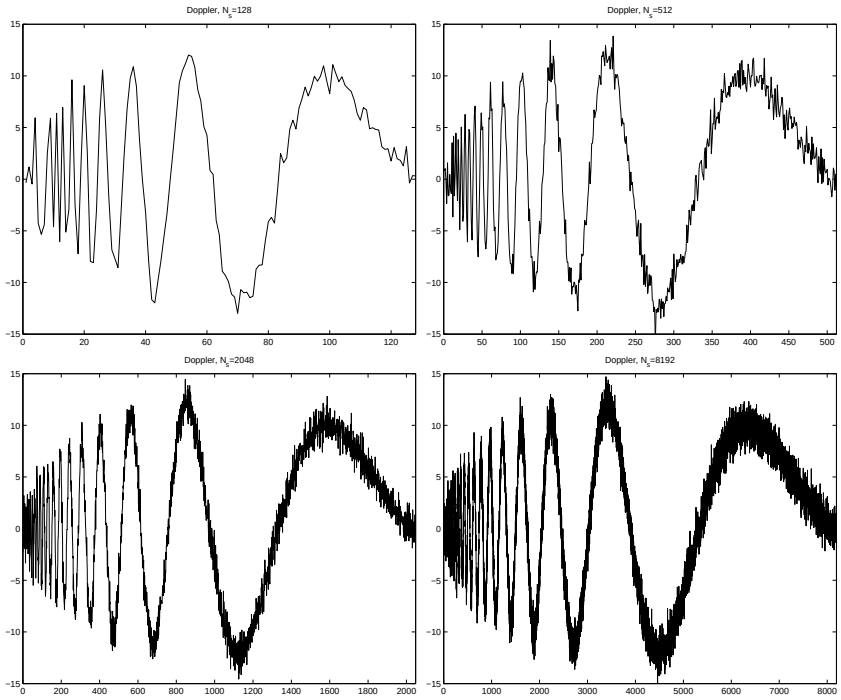

Figure 4.23: The Doppler function with sample size $N_s = 128$, $N_s = 512$, $N_s = 2048$ and $N_s = 8192$, and added white noise $N(0, 1)$. In the simulation 100 different white noise signals are used for each sample size.

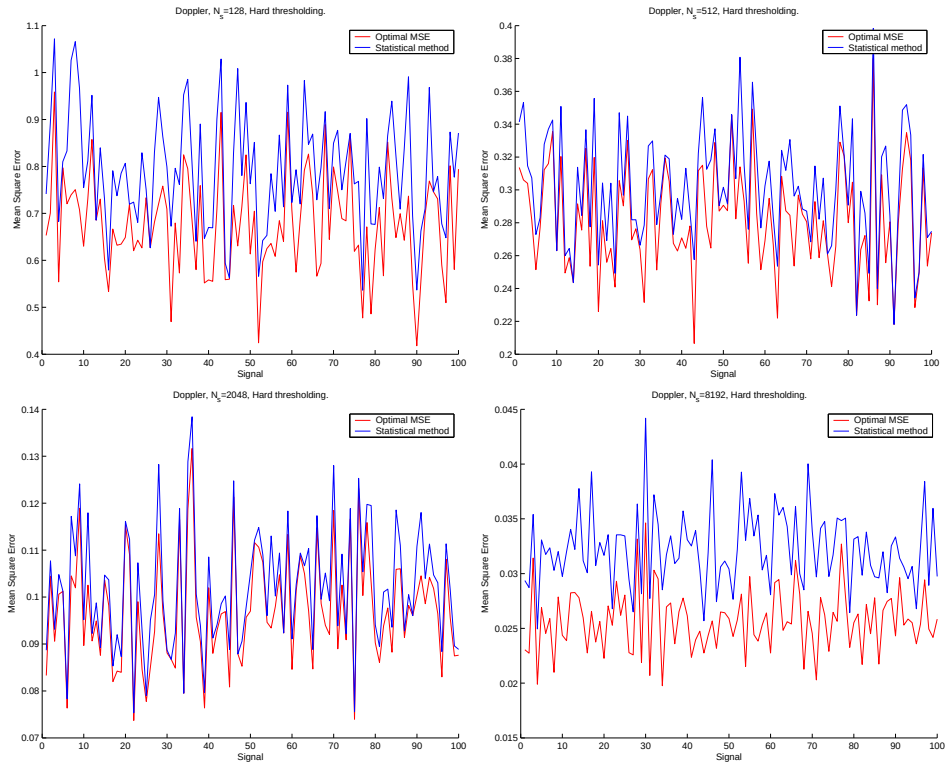

Figure 4.24: The Mean Square Errors using hard thresholding on the noisy Doppler signals. We have used four different sample sizes, and the Mean Square Errors for the statistical method are plotted against the optimal Mean Square Errors.

Sample size	Optimal MSE	Statistical method
$N_s = 128$	0.68	0.79
$N_s = 512$	0.28	0.30
$N_s = 2048$	0.097	0.10
$N_s = 8192$	0.026	0.032

Table 4.5: The average Mean Square Errors using hard thresholding on the noisy Doppler signals.

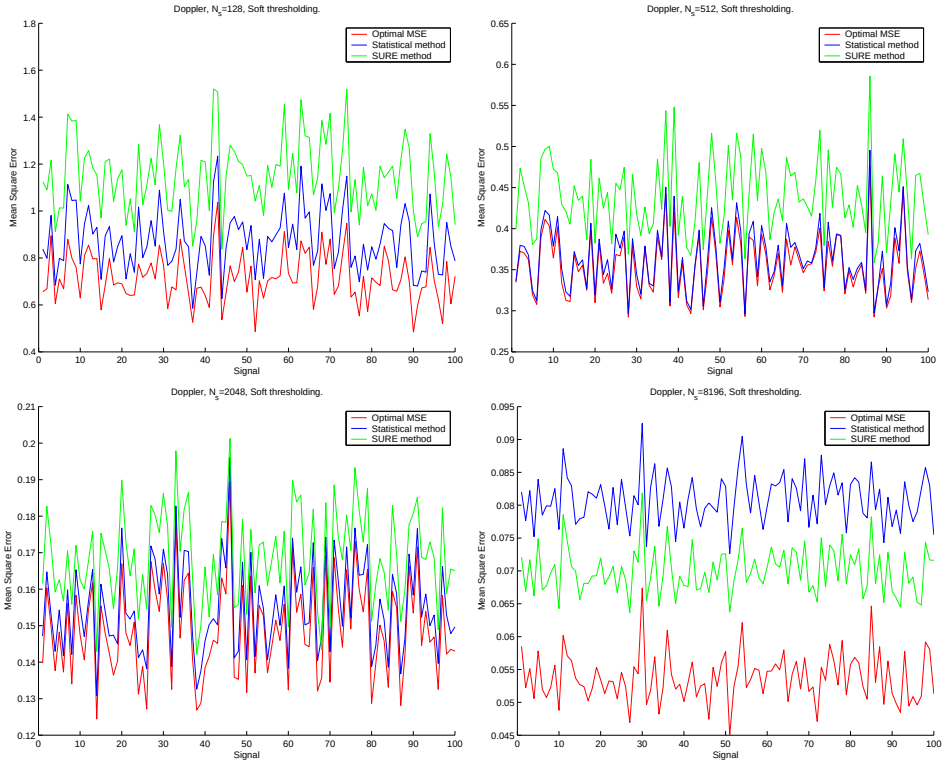

Figure 4.25: The Mean Square Errors using soft thresholding on the noisy Doppler signals. We have used four different sample sizes, and the Mean Square Errors for the statistical method and the SURE method are plotted against the optimal Mean Square Errors.

Sample size	Optimal MSE	Statistical method	SURE method
$N_s = 128$	0.72	0.88	1.15
$N_s = 512$	0.36	0.36	0.44
$N_s = 2048$	0.15	0.16	0.17
$N_s = 8192$	0.054	0.081	0.070

Table 4.6: The average Mean Square Errors using soft thresholding on the noisy Doppler signals.

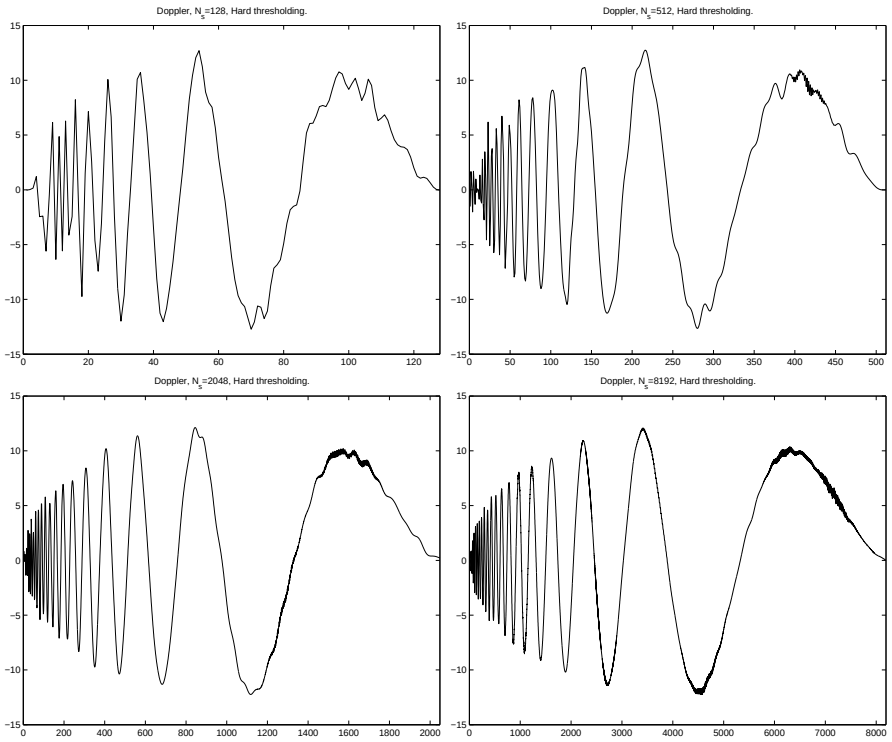

Figure 4.26: The denoised Doppler signals using hard thresholding and estimating the threshold levels using the statistical method.

4.4.4 Blocks

The Blocks function is defined by the equation

$$f(t) = \sum_{j=0}^{10} h_j K(t - t_j) \qquad K(t) = (1 + \text{sign}(t))/2.$$
$$t_j = \{.1, .13, .15, .23, .25, .40, .44, .65, .76, .78, .81\},$$
$$h_j = \{4, -5, 3, -4, 5, -4.2, 2.1, 4.3, -3.1, 5.1, -4.2\}.$$

The Blocks function is normalized such that the standard deviation equals 7, and sampled with four different samplesizes, namely $N_s = 128$, $N_s = 512$, $N_s = 2048$ and $N_s = 8192$, see Figure 4.27. For each sampled signal we add 100 different white noise signals $N(0, 1)$, resulting in 100 test signals for each sample size, see Figure 4.28. For each sample size we denoise each of the 100 test signals as explained in the beginning of this section.

The Mean Square Errors for the denoised test signals using hard thresholding are plotted in Figure 4.29 and the Mean Square errors for the denoised test signals using soft thresholding are plotted in Figure 4.30. The average Mean Square Errors

for the hard thresholding are presented in Table 4.7 and the average Mean Square Errors for the soft thresholding are presented in Table 4.8.

The results are very similar to the previous ones. From Figure 4.29 and Figure 4.30 we see that the statistical method performs very well. For hard thresholding the Mean Square Errors are almost identical with the optimal Mean Square Errors. The results for soft thresholding are also very good. The SURE method also performs well, but the statistical method performs better. Comparing Table 4.7 and Table 4.8 we see that the results for hard thresholding are better than the one for soft thresholding. Thus we achieve the best results using hard thresholding, and estimating the thresholds levels with the statistical method. In Figure 4.31 we have plotted the denoised signals using hard thresholding and the statistical method for estimating the threshold levels. From the Figure we see that we have removed some noise, however, there is a lot of noise left. Thus the denoising is not so good for the Blocks function. The reason for this is the many discontinuities in the Blocks function. In the time-frequency domain these discontinuities result in many small coefficients in the high frequency bands. These small coefficients are removed in the thresholding process and result in noise when the signal is transformed back into the time domain.

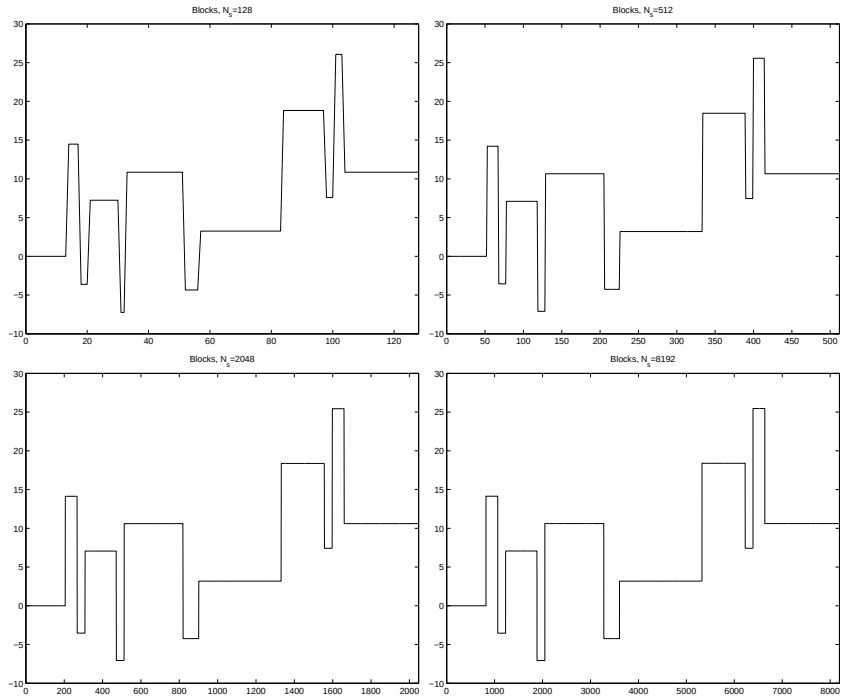

Figure 4.27: The Blocks function with sample size $N_s = 128$, $N_s = 512$, $N_s = 2048$ and $N_s = 8192$.

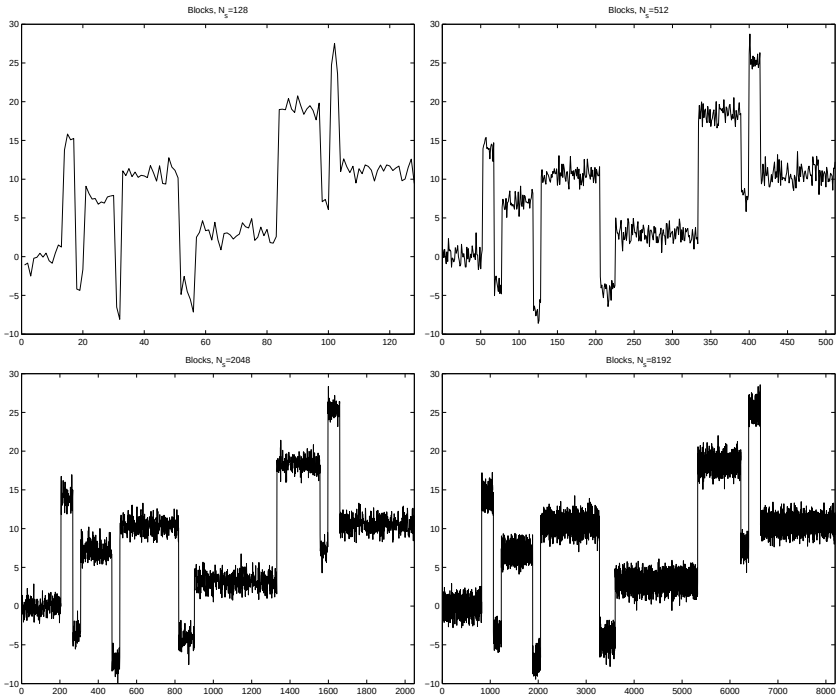

Figure 4.28: The Blocks function with sample size $N_s = 128$, $N_s = 512$, $N_s = 2048$ and $N_s = 8192$, and added white noise $N(0,1)$. In the simulation 100 different white noise signals are used for each sample size.

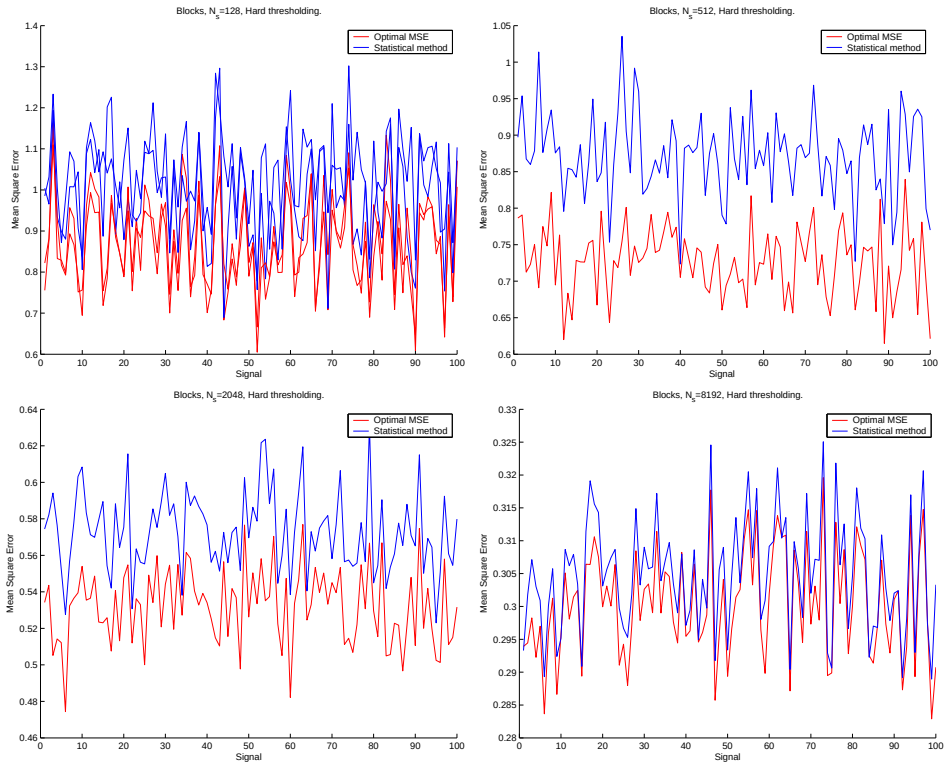

Figure 4.29: The Mean Square Errors using hard thresholding on the noisy Blocks signals. We have used four different sample sizes, and the Mean Square Errors for the statistical method are plotted against the optimal Mean Square Errors.

Sample size	Optimal MSE	Statistical method
$N_s = 128$	0.87	1.03
$N_s = 512$	0.73	0.87
$N_s = 2048$	0.53	0.57
$N_s = 8192$	0.30	0.30

Table 4.7: The average Mean Square Errors using hard thresholding on the noisy Blocks signals.

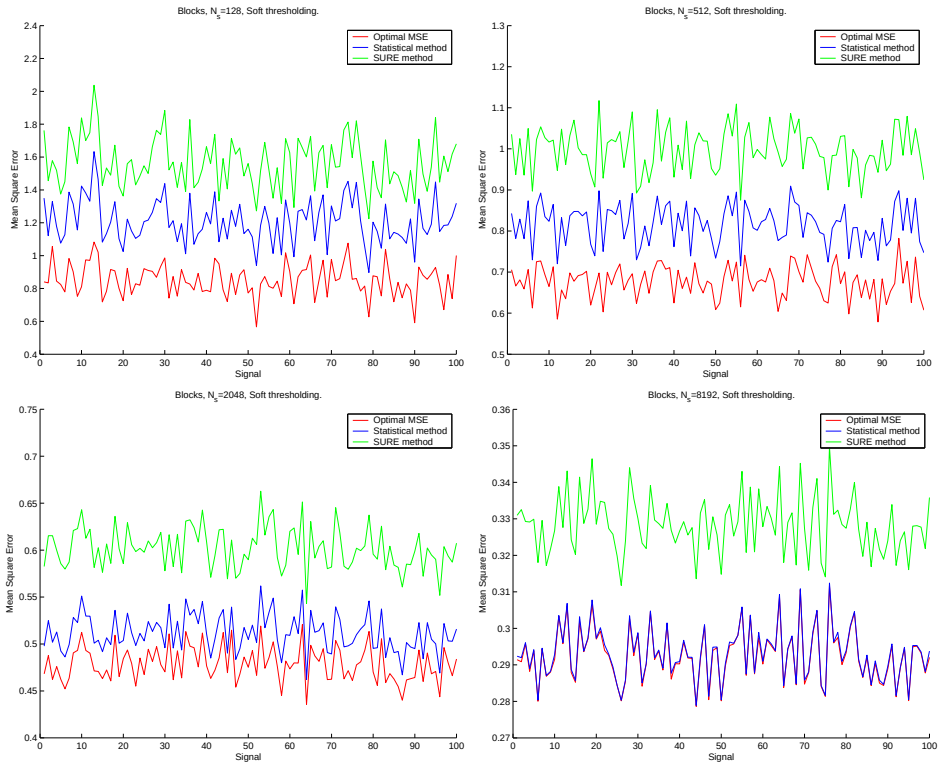

Figure 4.30: The Mean Square Errors using soft thresholding on the noisy Blocks signals. We have used four different sample sizes, and the Mean Square Errors for the statistical method and the SURE method are plotted against the optimal Mean Square Errors.

Sample size	Optimal MSE	Statistical method	SURE method
$N_s = 128$	0.85	1.21	1.56
$N_s = 512$	0.68	0.82	1.00
$N_s = 2048$	0.48	0.51	0.60
$N_s = 8192$	0.29	0.29	0.33

Table 4.8: The average Mean Square Errors using soft thresholding on the noisy Blocks signals.

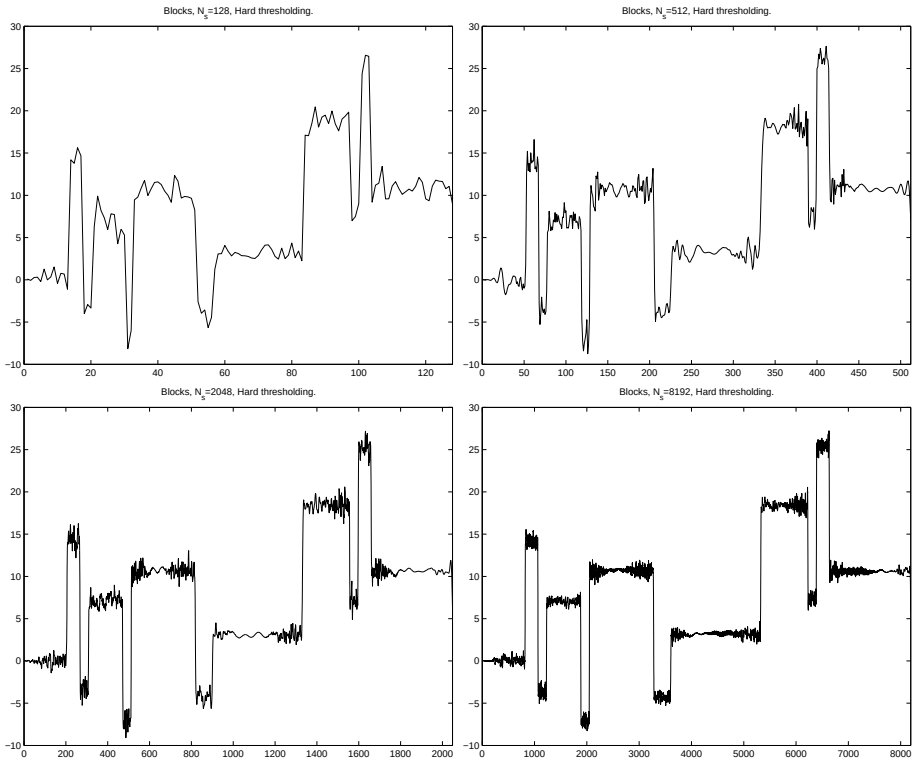

Figure 4.31: The denoised Blocks signals using hard thresholding and estimating the threshold levels using the statistical method.

4.4.5 Quadchirp

The Quadchirp function is defined by the equation

$$f(t) = \sin\Big(N_s(\pi/3)t^3\Big).$$

The Quadchirp function is normalized such that the standard deviation equals 7, and sampled with four different samplesizes, namely $N_s = 128$, $N_s = 512$, $N_s = 2048$ and $N_s = 8192$, see Figure 4.32. For each sampled signal we add 100 different white noise signals $N(0,1)$, resulting in 100 test signals for each sample size, see Figure 4.33. For each sample size we denoise each of the 100 test signals as explained in the beginning of this section.

The Mean Square Errors for the denoised test signals using hard thresholding are plotted in Figure 4.34 and the Mean Square errors for the denoised test signals using soft thresholding are plotted in Figure 4.35. The average Mean Square Errors for the hard thresholding are presented in Table 4.9 and the average Mean Square Errors for the soft thresholding are presented in Table 4.10.

The results are very similar to the previous ones. From Figure 4.34 and Figure 4.35 we see that the statistical method performs very well. For hard and soft thresh-

olding the Mean Square Errors are almost identical to the optimal Mean Square Errors. The SURE method also performs well, but the statistical method performs better. Comparing Table 4.9 and Table 4.10 we see that the results for hard thresholding are better than the one for soft thresholding. Thus we achieve the best results using hard thresholding, and estimating the thresholds levels with the statistical method. In Figure 4.36 we have plotted the denoised signals using hard thresholding and the statistical method for estimating the threshold levels.

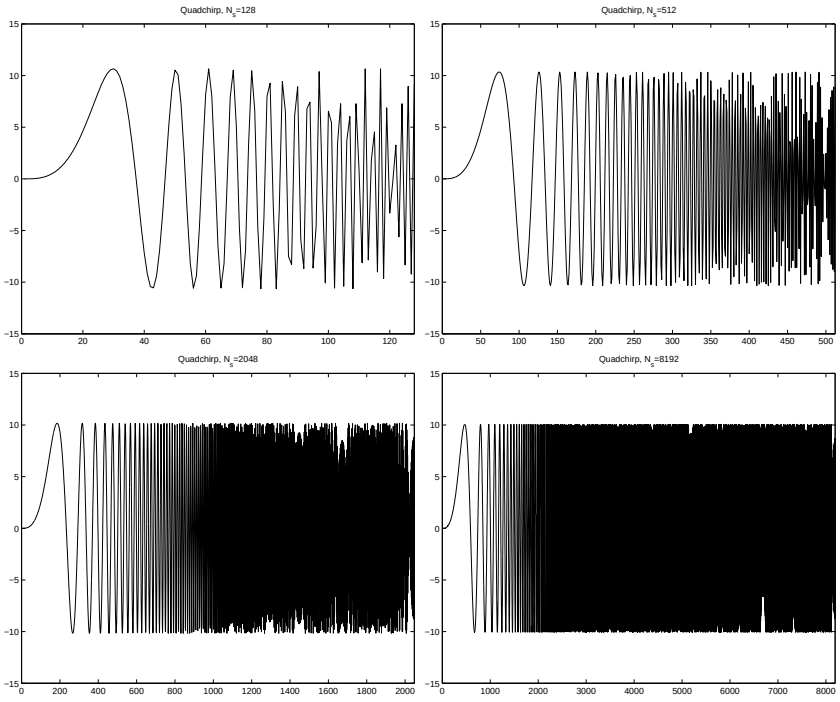

Figure 4.32: The Quadchirp function with sample size $N_s = 128$, $N_s = 512$, $N_s = 2048$ and $N_s = 8192$.

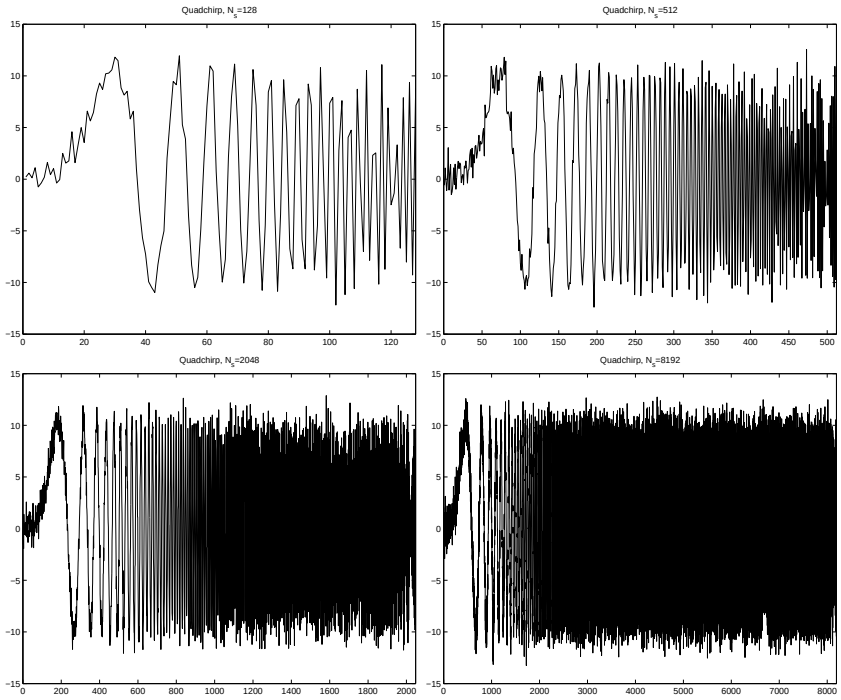

Figure 4.33: The Quadchirp function with sample size $N_s = 128$, $N_s = 512$, $N_s = 2048$ and $N_s = 8192$, and added white noise $N(0, 1)$. In the simulation 100 different white noise signals are used for each sample size.

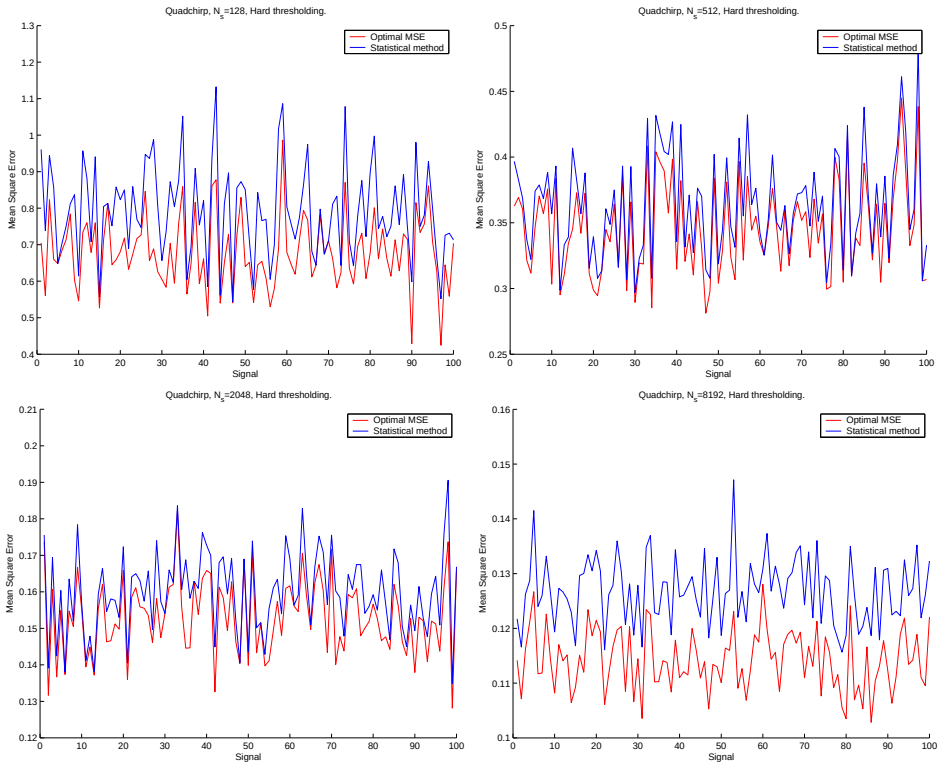

Figure 4.34: The Mean Square Errors using hard thresholding on the noisy Quadchirp signals. We have used four different sample sizes, and the Mean Square Errors for the statistical method are plotted against the optimal Mean Square Errors.

Sample size	Optimal MSE	Statistical method
$N_s = 128$	0.68	0.79
$N_s = 512$	0.35	0.36
$N_s = 2048$	0.15	0.16
$N_s = 8192$	0.11	0.13

Table 4.9: The average Mean Square Errors using hard thresholding on the noisy Quadchirp signals.

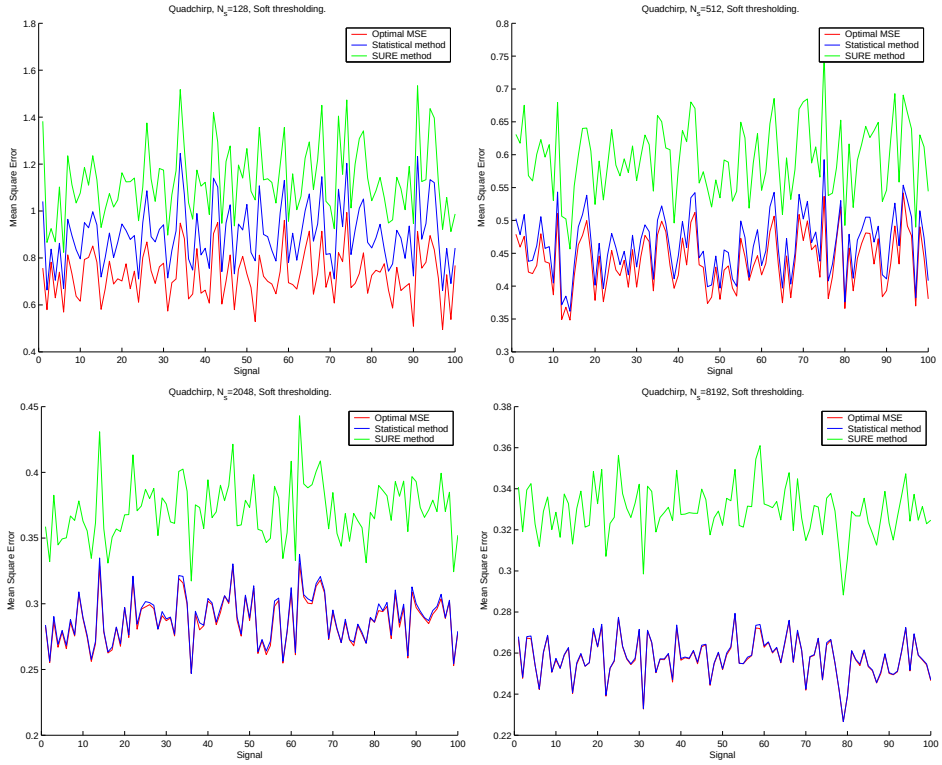

Figure 4.35: The Mean Square Errors using soft thresholding on the noisy Quadchirp signals. We have used four different sample sizes, and the Mean Square Errors for the statistical method and the SURE method are plotted against the optimal Mean Square Errors.

Sample size	Optimal MSE	Statistical method	SURE method
$N_s = 128$	0.73	0.90	1.13
$N_s = 512$	0.44	0.46	0.59
$N_s = 2048$	0.29	0.29	0.37
$N_s = 8192$	0.26	0.26	0.33

Table 4.10: The average Mean Square Errors using soft thresholding on the noisy Quadchirp signals.

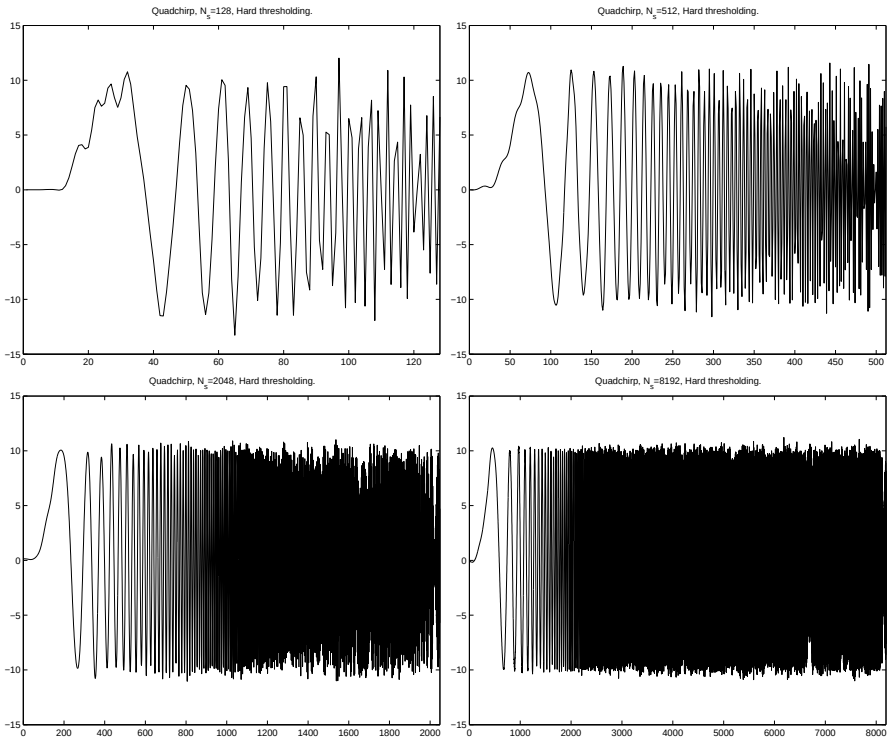

Figure 4.36: The denoised Quadchirp signals using hard thresholding and estimating the threshold levels using the statistical method.

4.4.6 Mishmash

The Mishmash function is defined by the equation

$$x = \sin\left(2^{N_s}(\pi/3)t^3\right) + \sin(0.6902 \cdot \pi \cdot 2^{N_s} \cdot t) + \sin(0.125 \cdot \pi \cdot t^2 \cdot 2^{N_s}).$$

The Quadchirp function is normalized such that the standard deviation equals 7, and sampled with four different samplesizes, namely $N_s = 128$, $N_s = 512$, $N_s = 2048$ and $N_s = 8192$, see Figure 4.37. For each sampled signal we add 100 different white noise signals $N(0, 1)$, resulting in 100 test signals for each sample size, see Figure 4.38. For each sample size we denoise each of the 100 test signals as explained in the beginning of this section.

The Mean Square Errors for the denoised test signals using hard thresholding are plotted in Figure 4.39 and the Mean Square errors for the denoised test signals using soft thresholding are plotted in Figure 4.40. The average Mean Square Errors for the hard thresholding are presented in Table 4.11 and the average Mean Square Errors for the soft thresholding are presented in Table 4.12.

The results are very similar to the previous ones. From Figure 4.39 and Figure 4.40 we see that the statistical method performs very well. For hard and soft

thresholding the Mean Square Errors are almost identical to the optimal Mean Square Errors. The SURE method also performs well, but the statistical method performs better. Comparing Table 4.11 and Table 4.12 we see that the results for hard thresholding are better than the one for soft thresholding. Thus we achieve the best results using hard thresholding, and estimating the thresholds levels with the statistical method. In Figure 4.41 we have plotted the denoised signals using hard thresholding and the statistical method for estimating the threshold levels.

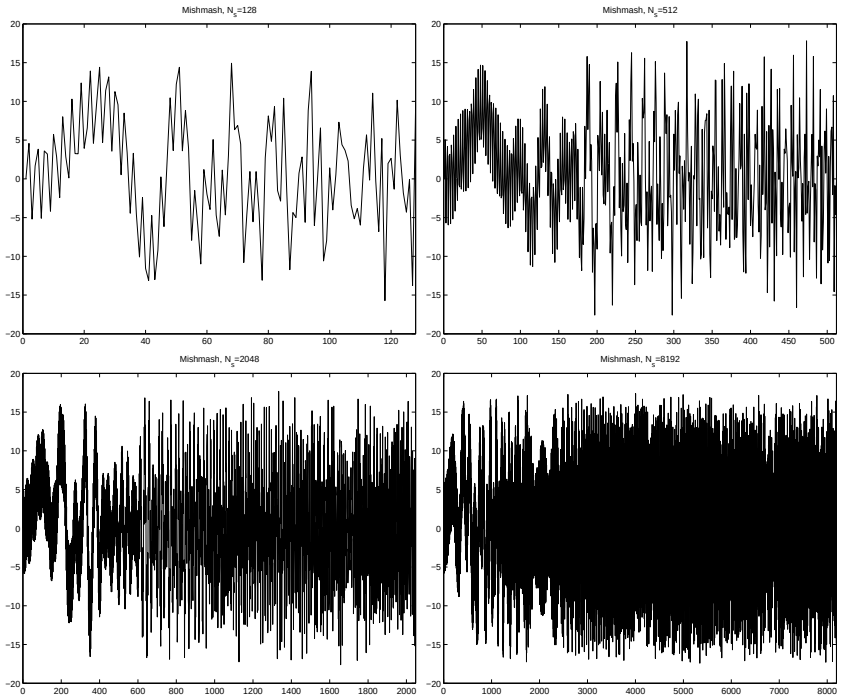

Figure 4.37: The Mishmash function with sample size $N_s = 128$, $N_s = 512$, $N_s = 2048$ and $N_s = 8192$.

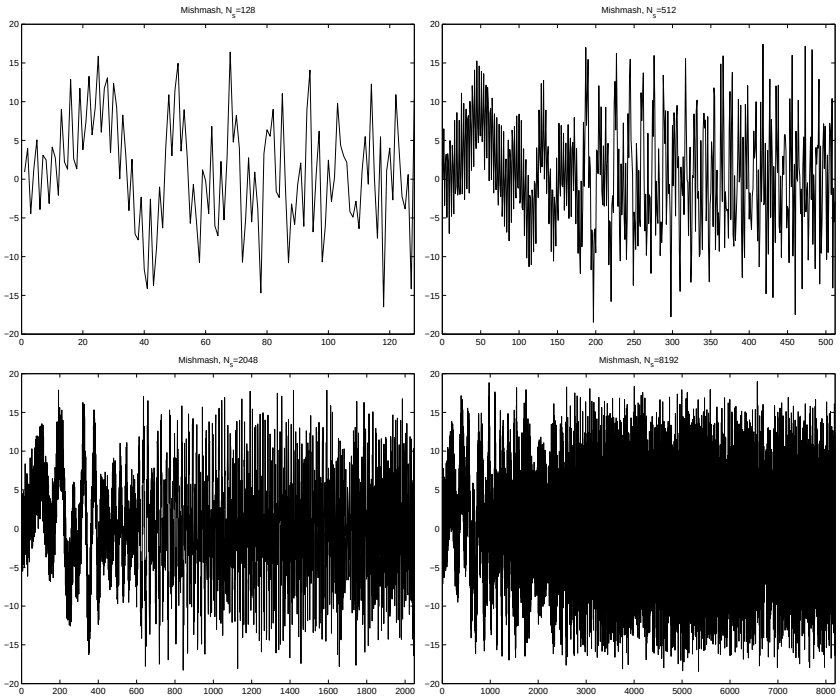

Figure 4.38: The Mishmash function with sample size $N_s = 128$, $N_s = 512$, $N_s = 2048$ and $N_s = 8192$, and added white noise $N(0, 1)$. In the simulation 100 different white noise signals are used for each sample size.

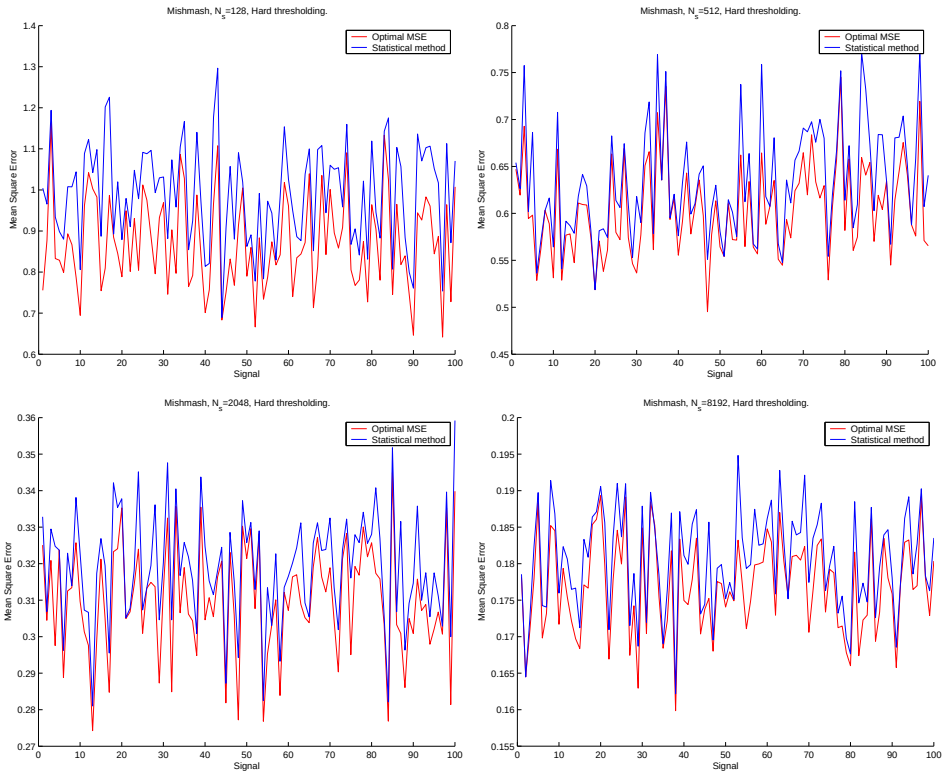

Figure 4.39: The Mean Square Errors using hard thresholding on the noisy Mishmash signals. We have used four different sample sizes, and the Mean Square Errors for the statistical method are plotted against the optimal Mean Square Errors.

Sample size	Optimal MSE	Statistical method
$N_s = 128$	0.87	0.99
$N_s = 512$	0.60	0.63
$N_s = 2048$	0.31	0.32
$N_s = 8192$	0.18	0.18

Table 4.11: The average Mean Square Errors using hard thresholding on the noisy Mishmash signals.

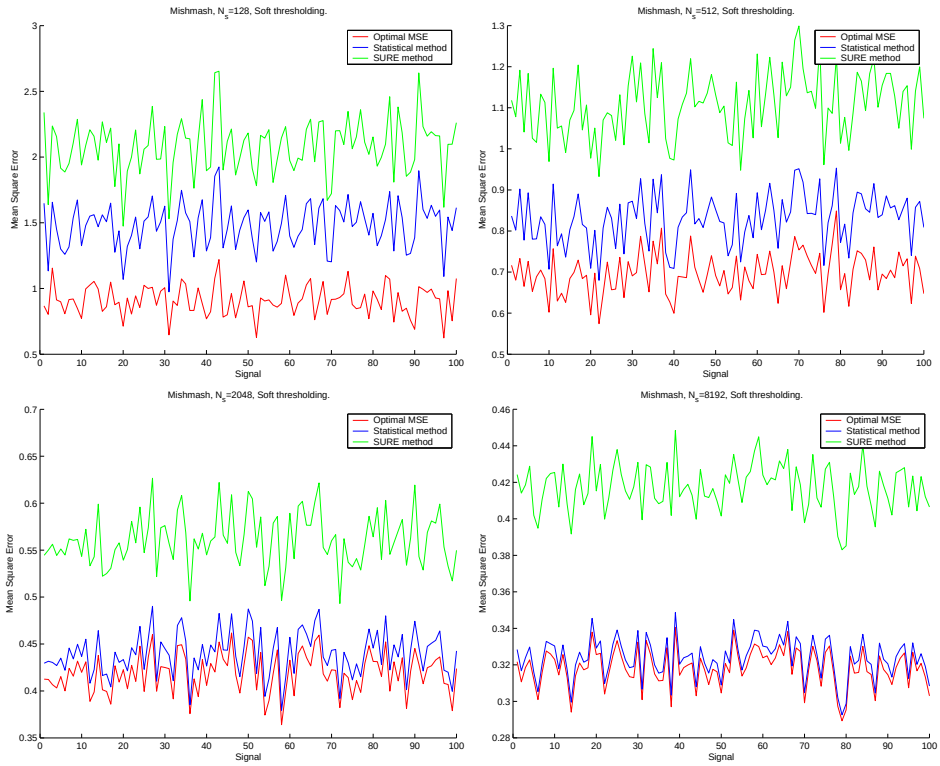

Figure 4.40: The Mean Square Errors using soft thresholding on the noisy Mishmash signals. We have used four different sample sizes, and the Mean Square Errors for the statistical method and the SURE method are plotted against the optimal Mean Square Errors.

Sample size	Optimal MSE	Statistical method	SURE method
$N_s = 128$	0.91	1.48	2.08
$N_s = 512$	0.70	0.83	1.11
$N_s = 2048$	0.42	0.44	0.56
$N_s = 8192$	0.32	0.32	0.42

Table 4.12: The average Mean Square Errors using soft thresholding on the noisy Mishmash signals.

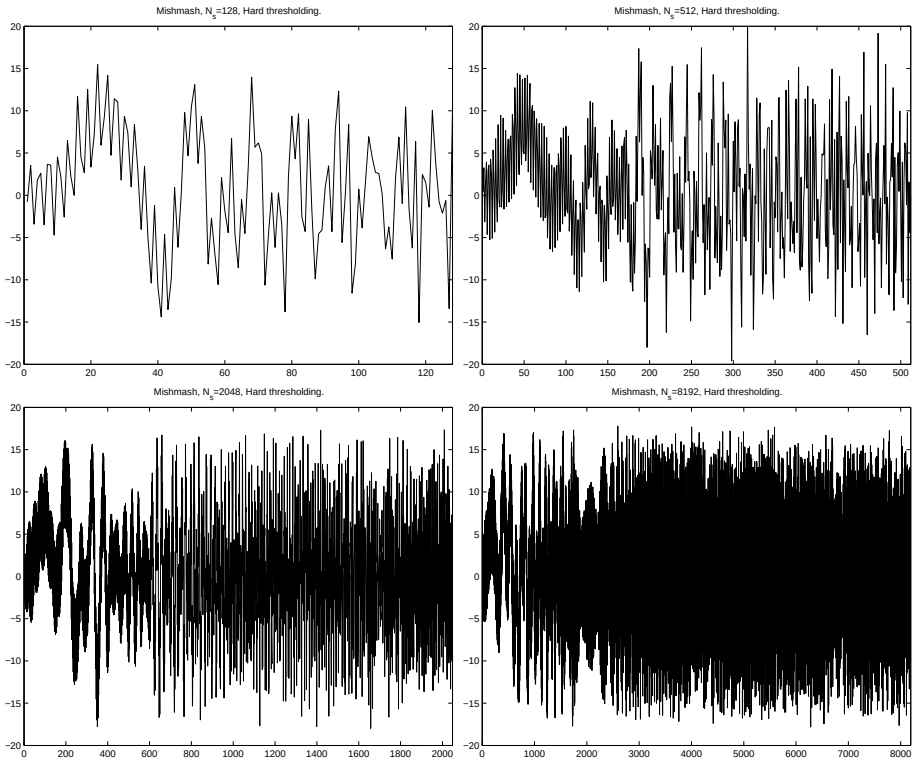

Figure 4.41: The denoised Mishmash signals using hard thresholding and estimating the threshold levels using the statistical method.

4.4.7 Conclusion

The numerical tests we have done clearly show that the statistical method performs very well. For both hard and soft thresholding the threshold levels estimated with the statistical method are very close to the optimal threshold levels for almost all signals and sample sizes. The SURE method also performs good, but the statistical method performs better.

We also notice that the Mean Square Errors using hard thresholding are lower than the Mean Square Errors using soft thresholding. From this we conclude that the best results are achieved using hard thresholding and estimating the threshold levels with the statistical method.

4.5 Gabor versus Wavelet denoising

In this section we compare the efficiency of the Gabor denoising against the efficiency of the wavelet denoising. For the experiments we again use the six functions due to Donoho and Johnstone [11]: Bumps, HeaviSine, Doppler, Blocks, Quad-

chirp and Mishmash. The functions are normalized such that their standard deviation equals 7, and sampled with four different sample sizes, namely $N_s = 128$, $N_s = 512$, $N_s = 2048$ and $N_s = 8192$. For each signal and sample size we add 100 different white noise signals $N(0, 1)$. This gives 100 test signals, with a signal-to-noise ratio of 7, for each signal and sample size. These test signals are denoised using hard Gabor denoising, soft wavelet denoising and hard wavelet denoising. The Mean Square Errors for the hard Gabor denoising are plotted against the Mean Square Errors for the soft wavelet denoising and the Mean Square Errors for the hard wavelet denoising. The results can be found in figures 4.42-4.47. The average Mean Square Errors are also calculated, and the results can be found in tables 4.13-4.18. Notice that whereas hard thresholding gives the best results when Gabor denoising is used, soft thresholding gives the best results when wavelet denoising is used.

For the hard Gabor denoising the statistical method is used for estimating the threshold levels, the frequency resolution M is set to 16 and double oversampling is used. For the hard and soft wavelet denoising the wavelet toolbox in Matlab is used. Each of the test signals are denoised using the "wden" function with the following settings: Symlet 8 wavelets, 5 levels decomposition, Steins unbiased risk estimate and no rescaling.

The results show that the "GaborShrink" method can compete with the "WaveShrink" method.

- For both the Quadchirp function and the Mishmash function the results for the Gabor denoising are much better than the ones for wavelet denoising, see Figure 4.46 and Figure 4.47.

- For the Bumps function the results for the Gabor denoising are a little better than the ones for wavelet denoising, see Figure 4.42.

- For the Doppler function the results for the Gabor denoising and the wavelet denoising are almost identical, see Figure 4.44. The Gabor denoising is a little better for $N_s = 8192$, but the wavelet denoising is a little better for $N_s = 128$.

- For the HeaviSine function the results for the wavelet denoising are a little better than the ones for the Gabor denoising, see Figure 4.43.

- For the Blocks function the results for the wavelet denoising are better than the ones for the Gabor denoising, see Figure 4.45. This is due to the reasons explained in section 4.4.4.

From this we conclude that the "GaborShrink" method is a very good alternative to the "WaveShrink" method, especially when denoising unregular signals like the Quadchirp, Mishmash and Bumps signals.

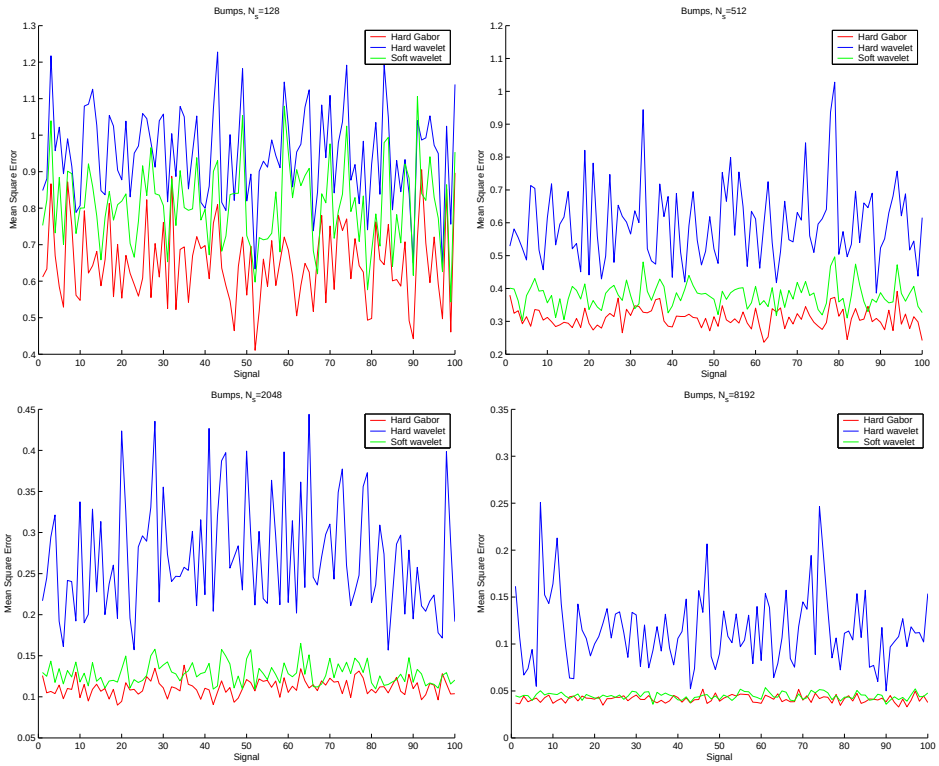

Figure 4.42: The Mean Square Errors denoising the noisy Bumps signals. We have used four different sample sizes, and the Mean Square Errors for the hard thresholded Gabor denoising are plotted against the Mean Square Errors for the hard thresholded and soft thresholded wavelet denoising.

Sample size	Hard Gabor den.	Hard wavelet den.	Soft wavelet.
$N_s = 128$	0.65	0.95	0.81
$N_s = 512$	0.31	0.60	0.38
$N_s = 2048$	0.11	0.27	0.13
$N_s = 8192$	0.04	0.11	0.04

Table 4.13: The average Mean Square Errors denoising the noisy Bumps signals.

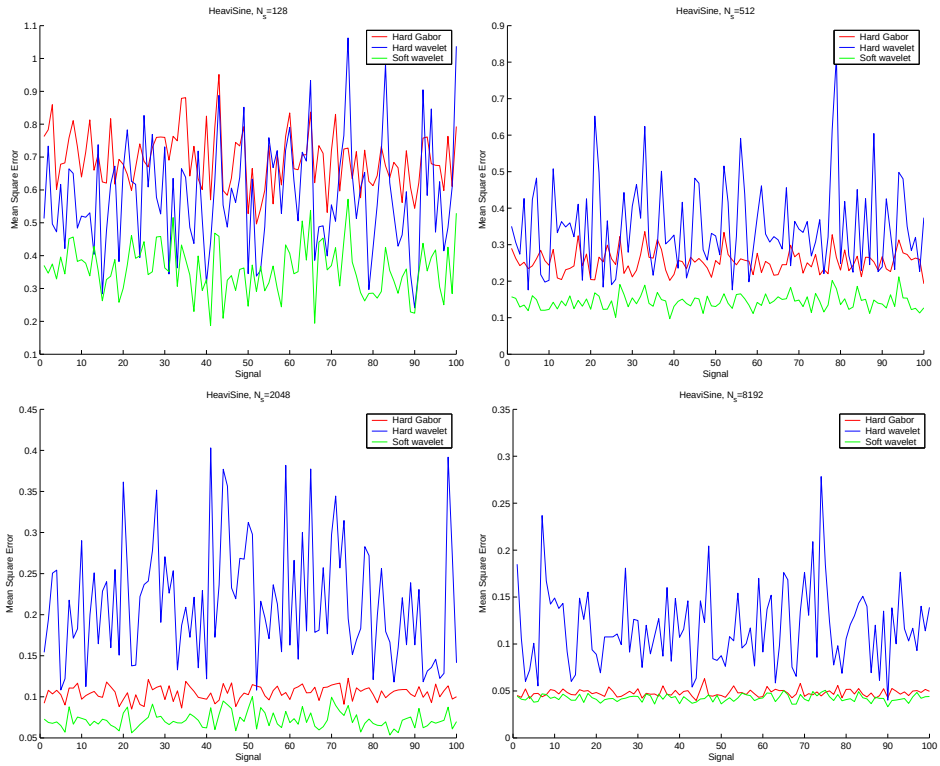

Figure 4.43: The Mean Square Errors denoising the noisy HeaviSine signals. We have used four different sample sizes, and the Mean Square Errors for the hard thresholded Gabor denoising are plotted against the Mean Square Errors for the hard thresholded and soft thresholded wavelet denoising.

Sample size	Hard Gabor den.	Hard wavelet den.	Soft wavelet.
$N_s = 128$	0.69	0.58	0.36
$N_s = 512$	0.25	0.35	0.14
$N_s = 2048$	0.10	0.22	0.07
$N_s = 8192$	0.05	0.12	0.04

Table 4.14: The average Mean Square Errors denoising the noisy HeaviSine signals.

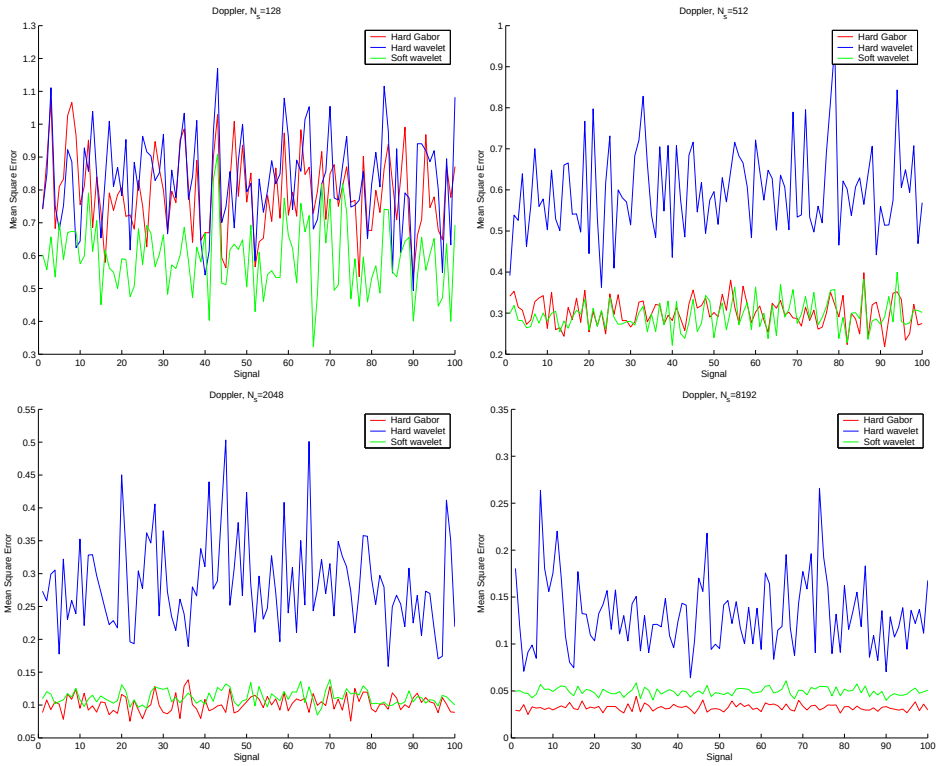

Figure 4.44: The Mean Square Errors denoising the noisy Doppler signals. We have used four different sample sizes, and the Mean Square Errors for the hard thresholded Gabor denoising are plotted against the Mean Square Errors for the hard thresholded and soft thresholded wavelet denoising.

Sample size	Hard Gabor den.	Hard wavelet den.	Soft wavelet.
$N_s = 128$	0.79	0.83	0.60
$N_s = 512$	0.30	0.60	0.29
$N_s = 2048$	0.10	0.29	0.11
$N_s = 8192$	0.032	0.13	0.050

Table 4.15: The average Mean Square Errors denoising the noisy Doppler signals.

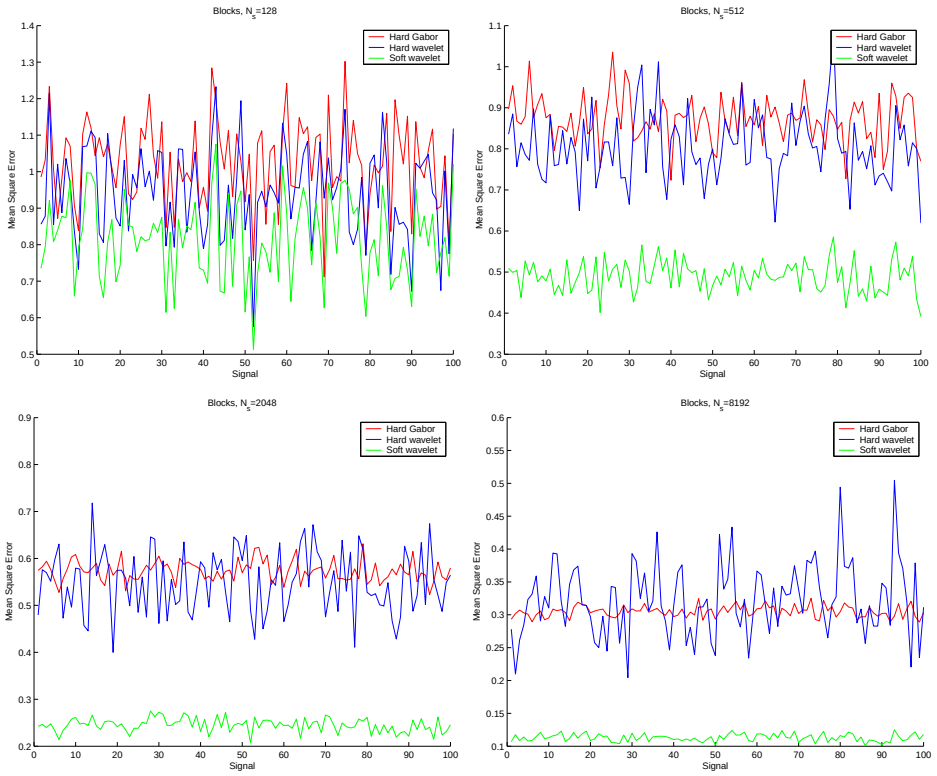

Figure 4.45: The Mean Square Errors denoising the noisy Blocks signals. We have used four different sample sizes, and the Mean Square Errors for the hard thresholded Gabor denoising are plotted against the Mean Square Errors for the hard thresholded and soft thresholded wavelet denoising.

Sample size	Hard Gabor den.	Hard wavelet den.	Soft wavelet.
$N_s = 128$	1.03	0.94	0.82
$N_s = 512$	0.87	0.80	0.49
$N_s = 2048$	0.57	0.55	0.25
$N_s = 8192$	0.30	0.32	0.11

Table 4.16: The average Mean Square Errors denoising the noisy Blocks signals.

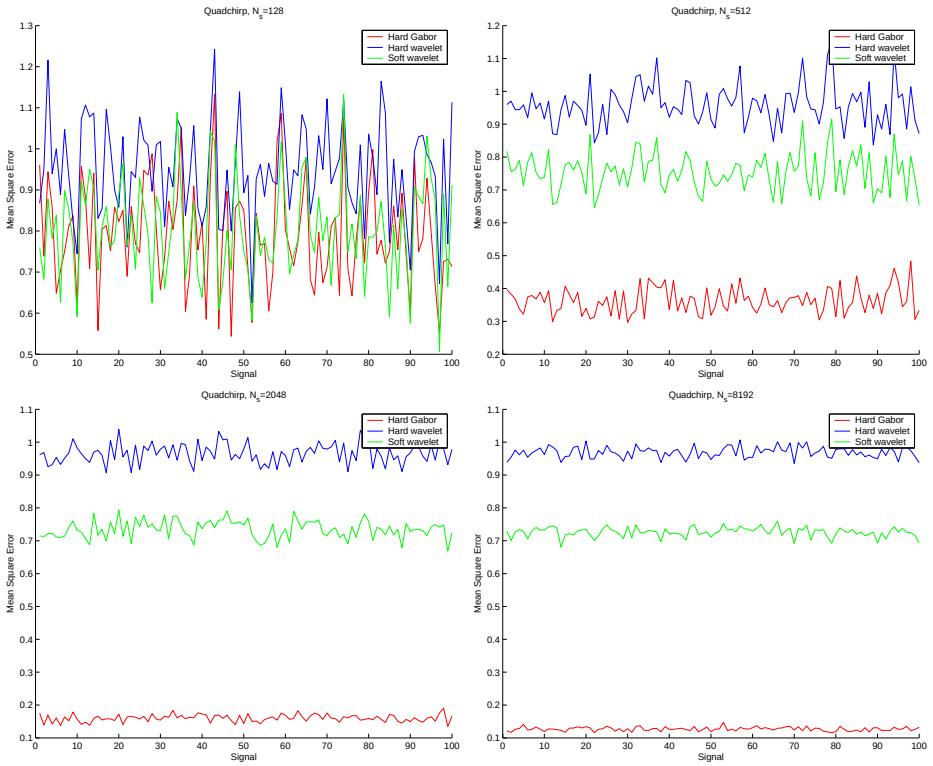

Figure 4.46: The Mean Square Errors denoising the noisy Quadchirp signals. We have used four different sample sizes, and the Mean Square Errors for the hard thresholded Gabor denoising are plotted against the Mean Square Errors for the hard thresholded and soft thresholded wavelet denoising.

Sample size	Hard Gabor den.	Hard wavelet den.	Soft wavelet.
$N_s = 128$	0.79	0.95	0.80
$N_s = 512$	0.36	0.96	0.75
$N_s = 2048$	0.16	0.97	0.73
$N_s = 8192$	0.13	0.97	0.79

Table 4.17: The average Mean Square Errors denoising the noisy Quadchirp signals.

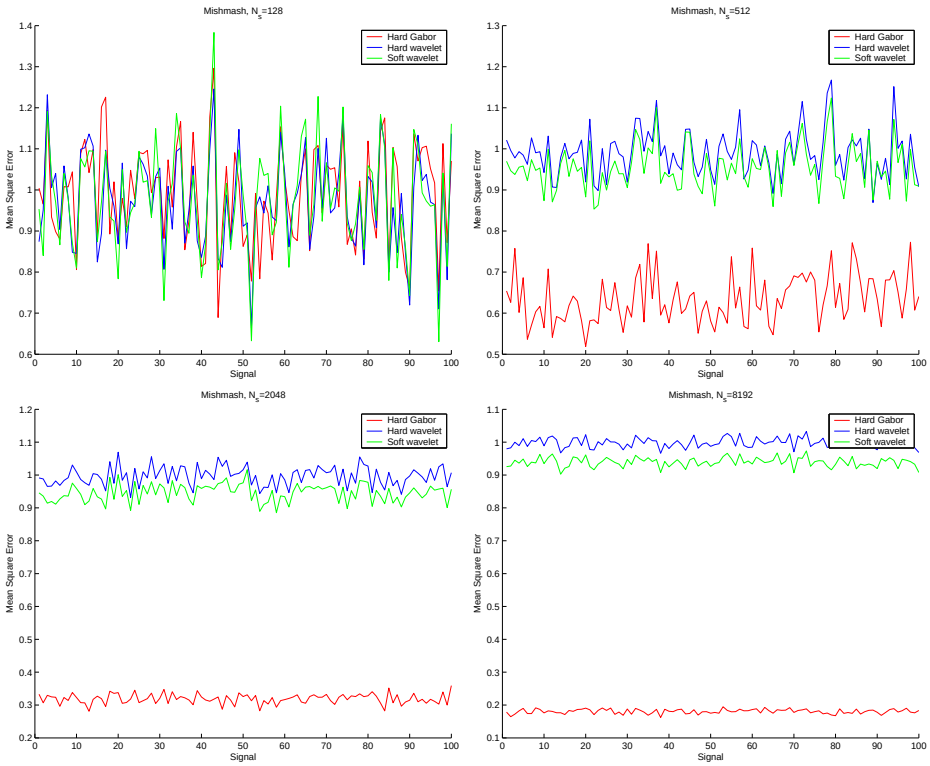

Figure 4.47: The Mean Square Errors denoising the noisy Mishmash signals. We have used four different sample sizes, and the Mean Square Errors for the hard thresholded Gabor denoising are plotted against the Mean Square Errors for the hard thresholded and soft thresholded wavelet denoising.

Sample size	Hard Gabor den.	Hard wavelet den.	Soft wavelet.
$N_s = 128$	0.99	0.97	0.98
$N_s = 512$	0.64	0.99	0.96
$N_s = 2048$	0.32	1.00	0.95
$N_s = 8192$	0.18	1.00	0.94

Table 4.18: The average Mean Square Errors denoising the noisy Mishmash signals.

4.6 Denoising music

In addition to the numerical tests in the last section we have denoised an old recording of Grieg playing the piano. The recording is from the beginning of the 20'th century and contains strong white noise. The recording is denoised using hard Gabor thresholding, but we have not used the statistical method for estimating the

noise level. Instead we tried several thresholding levels and chose the one that sounded best. We chose this approach because it is not always the most optimal denoised signal that sounds best. This is because our ears are complex and nonlinear instruments.

The original recording and the denoised version can be downloaded from http://www.ii.uib.no/∼oddvar/Grieg.html. It should be mentioned that the denoised version still contains some noise. This is because the noise in the original recording is not completely white, it contains spikes or outliers. These spikes have larger amplitude than the rest of the noise and are therefore not removed in the thresholding procedure. In wavelet denoising there is done research on this type of noise, and a method called "outlier resistant wavelet transform" is designed [1]. It is probably possible to design an outlier resistant method for Gabor denoising too, something that will give a better denoising of the recording.

Chapter 5

Concluding remarks and further work

In this thesis we have presented the mathematical techniques for the Gabor transform and the inverse Gabor transform. We have treated both the critical sampled and the oversampled case. For both cases we developed algorithms, and analyzed the computational cost of them.

We developed a method called "GaborShrink" for the removal of white noise from signals. The method uses transform-based filtering, working in three steps:

- Transform the noisy data into the time-frequency domain using the Gabor transform.
- Hard threshold the coefficients, thereby suppressing those coefficients containing noise.
- Transform back into the original domain using the inverse Gabor transform.

For the estimation of the threshold level we developed a method using the statistical properties of the white noise. We showed that this method performed very well compared to the optimal threshold level.

We denoised six different test functions using both the "GaborShrink" method and the "WaveShrink" method. We compared the results and showed that the "GaborShrink" method performed better for the Quadchirp, the Mishmash and the Bumps function. We concluded that the "GaborShrink" is a very good alternative to the "WaveShrink", especially when denoising unregular signals.

Further work

- We have only tested hard and soft thresholding in this thesis. There exists other techniques that can be tested, like Garrote shrinkage [5], Firm shrinkage [17] and cycle spinning [8].

- In Subsection 4.3.3 we mentioned that it is possible to estimate the noise level from the median of the coefficients in the highest frequency bands. We have done a few numerical tests with good results, however, more research is needed.
- Recently, a new sampling lattice - the quincunx lattice - has been introduced as a sampling geometry in the Gabor scheme, which geometry is different from the traditional rectangular sampling geometry [27]. The "GaborShrink" can be implemented with the quincunx sampling lattice and the efficiency can be compared against the regular sampling lattice.
- An outlier resistant method for the "GaborShrink" can be developed, see Section 4.6.
- The Gabor transform and its inverse can be extended to two dimensions, and the "GaborShrink' can then be tested on noisy images.

Appendix A

A.1 Proof of theorem 2.3.2

Proof. We start with the Gabor transform (2.19)

$$a_{m,k} = \sum_{n'=0}^{MN-1} x[n']W^*(n' - mN)e^{-j2\pi kn'/N}.$$

Multiplying both sides with $e^{-j[2\pi ml/M - 2\pi kn/N]}$ and taking the sum over $m \in \{0, ..., M-1\}$ and $k \in \{0, ..., N-1\}$, i.e. taking the 2 dimensional Fourier transform of both sides [cf. (1.6)], we get

$$\overline{a}[n, l; M, N] = \sum_{m=0}^{M-1}\sum_{k=0}^{N-1}\Big(\sum_{n'=0}^{MN-1} x[n']W^*(n'-mN)e^{-j2\pi kn'/N}\Big)e^{-j(2\pi ml/M - 2\pi kn/N)}.$$

Rearranging factors we get

$$\overline{a}[n, l; M, N] = \sum_{m=0}^{M-1}\sum_{k=0}^{N-1} x[n']W^*(n'-mN)\Big(\sum_{n'=0}^{MN-1} e^{-j2\pi k(n'-n)/N}\Big)e^{-j2\pi ml/M}.$$

The complex exponential is a circular function, thus each sum of length N equals 0 unless $n' = n + qN$, where $q \in \mathbf{Z}$, i.e.

$$\sum_{k=0}^{N-1} e^{-j2\pi k(n'-n)/N} = \begin{cases} 0 & n' \neq n + qN, \\ N & n' = n + qN \end{cases}.$$

Thus making the substitution $n' = n + qN$ we get

$$\overline{a}[n, l; M, N] = N\sum_{m=0}^{M-1}\sum_{p=-\infty}^{\infty} x[n + pN]W^*(n + (p-m)N)e^{-j2\pi ml/M}.$$

Multiplying the right side with $e^{j2\pi pl/M}e^{-j2\pi pl/M} = 1$ and taking a final rearrangement we find

$$\overline{a}[n, l; M, N] = N\sum_{p=-\infty}^{\infty} x[n+pN]e^{-j2\pi pl/M}\sum_{m=0}^{M-1} W^*(n+(p-m)N)e^{j2\pi(p-m)l/M}.$$

Since the sum over p is infinite we can make the substitution $m' = p - m$, this gives

$$\overline{a}[n,l;M,N] = N\Big[\sum_{p=-\infty}^{\infty} x[n+pN]e^{-j2\pi pl/M}\Big]\Big[\sum_{m'=0}^{M-1} W(n+m'N)e^{-j2\pi m'l/M}\Big]^*.$$

Comparing with (1.8) and (1.10) we see that the first summation is the Zak transform of the signal $x[n]$ and the second is the Zak transform of the window function $w[n]$, i.e.

$$\bar{a}[n,l;K,M] = N\tilde{x}[n,l;N,M]\tilde{w}^*[n,l;N,M], \tag{A.1}$$

which completes the proof. □

Bibliography

[1] H.-Y. Gao A. Bruce, D. Donoho and D. Martin. Denoising and robust nonlinear wavelet analysis. *SPIE Proceedings, Wavelet Applications*, 2242, 1994.

[2] R. W. Schafer A. V. Oppenheim and J. R. Buck. *Discrete-time signal processing 2nd ed.* Prentice Hall signal processing series, 1999.

[3] M. J. Bastiaans. Gabor's expansion and the Zak transform for continuous and discrete-time signals:critical sampling and rational oversampling. Technical report, Faculty of Electrical Engineering, Eindhoven University of Technology, Eindhoven, The Netherlands, December 1995.

[4] M. J. Bastiaans and M. C. W. Geilen. On the discrete Gabor transform and the discrete Zak transform. *Signal Process*, 49:151–166, 1996.

[5] L. Breiman. Better subset regression using the nonnegative garrote. *Technometrics*, 37:373–384, 1996.

[6] S.L. Campbell and C.D. Meyer. *Generalized inverses of linear transformations*. Dover Publications, 1991.

[7] T. T. Chinen and T. R. Reed. A performance analysis of fast Gabor transform methods. *Graphical models and image processing*, 59(3):117–127, may 1997.

[8] R. R. Coifman and D. L. Donoho. Translation-invariant de-noising. *Wavelets and statistics. New York: Springer-Verlag*, 74:125–150, 1995.

[9] J. G. Daugman. Complete discrete 2-d Gabor transform by neural networks for image analysis and compression. *IEEE Transactions on acoustics, speech and signal processing*, 36(7):1169–1179, July 1988.

[10] G. Kerkyacharian D.L Donoho, I.M. Jonstone and D. Picard. Wavelet shrinkage: Asymptopia? *J. R. Statist. Soc. B.*, 57:301–337, 1995.

[11] D. L. Donoho and I. M. Johnstone. Adapting to unknown smoothness via wavelet shrinkage. *Journal of the American Statistical Association*, 90:1200–1224, 1995.

[12] D.L Donoho. De-noising by soft-thresholding. *IEEE Trans. Inform. Theory*, 41:613–627, 1995.

[13] D.L Donoho and I.M. Jonstone. Ideal spatial adaption via wavelet shrinkage. *Biometrika*, 81:425–455, 1994.

[14] T. Ebrahimi and M. Kunt. Image compression by Gabor expansion. *Opt. Eng.*, 30(7):873–880, 1991.

[15] D. Gabor. Theory of communication. *Journal of the Institute for Electrical Engineers*, 93:429–439, 1946.

[16] H.-Y. Gao. Wavelet shrinkage denoising using the non-negative garrote. *J. Comput. Graph. Statist.*, 7:469–488, 1998.

[17] H.-Y. Gao and A. G. Bruce. Waveshrink with firm shrinkage. *Statistica Sinica*, 4:855–874, 1996.

[18] B. MacLennan. Gabor representations of spatiotemporal visual images. Technical report, Computer Science Department, University of Tennessee, Knoxville, TN 37996, October 1994.

[19] S. Qian and D. Chen. Discrete Gabor transform. *IEEE Trans on Signal Processing*, 41:2429–2438, jul 1993.

[20] F. Zhou S. Qiu and P. E. Crandall. Discrete Gabor transform with complexity $\mathcal{O}$(nlogn). *Signal Processing*, 77:159–170, 1999.

[21] S. W. Smith. *The Scientist and Engineer's Guide to Digital Signal Processing*. California Technical Publishing, 1997.

[22] C. Stein. Estimation of the mean of a multivariate normal distribution. *Annals of Statistics*, 9(6):1135–1151, 1981.

[23] G. Strand and T. Nguyen. *Wavelets and Filter Banks*. Wellesley - Cambridge Press, revised edition edition, 1997.

[24] T. Strohmer. Computational frameworks for discrete Gabor analysis. NUHAG - Numerical Harmonic Analysis Group.

[25] T. Strohmer. A unified approach to numerical algorithms for discrete Gabor expansions. NUHAG - Numerical Harmonic Analysis Group.

[26] T. R. Reed T. Ebrahimi and M. Kunt. Video coding using a pyramidal Gabor expansion. *SPIE Vis. Commun. Image Process.*, 1360:489–502, 1990.

[27] J. A. van Leest and M. J. Bastiaans. Gabor's signal expansion and the Gabor transform on a non-separable time-frequency lattice. *Journal of the Franklin Institute*, 337:291–301, 2000.

[28] J. Wexler and S. Raz. Discrete Gabor expansions. *Signal Processing*, 21:207–220, 1990.

[29] A. Høyland. *Sannsynlighetsregning og statistisk metodelæere*. Tapir, 3rd edition, 1979.

[30] J. Zak. Finite translations in solid-state physics. *Phys. Rev. Lett.*, 19:1385–1387, 1967.

[31] J. Zak. Dynamics of electrons in solid in external fields. *Phys. Rev.*, 168:686–695, 1968.

[32] J. Zak. The kq-representation in the dynamics of electrons in solids. *Solid State Physics*, 27:1–62, 1972.