NTNU
Norwegian University of
Science and Technology

A Phase Field Approach for a Modified Mumford-Shah Functional Based on Fractional Derivatives

Monica Kappelslåen
Plassen

Master of Science in Physics and Mathematics
Submission date: June 2016
Supervisor: Markus Grasmair, MATH

Norwegian University of Science and Technology
Department of Mathematical Sciences

Abstract

Recovering the true image, or true signal, from a corrupted one is by no means a trivial task, and we will in this paper study some approaches to finding good approximations to this unknown true image. The main focus will be on a minimization method based on wavelet decompositions of images and finding a solution which is sparse with respect to the wavelet basis. We then argue that knowledge of the position of edges between objects in the image could be utilized with the aim to improve the edge preserving capabilities of the method. Therefore, we present an edge detection approach and a subsequent edge dependent weighting of the coefficients in the minimization method. If this method does in fact improve the appearance of the solution, we shall seek a more sophisticated and efficient approach. Borrowing a phase field approach used by Ambrosio and Tortorelli to approximate a solution to the Mumford–Shah functional, we construct a modified functional where the edge dependent weighting is found based on the phase field. After developing an alternating minimization method, we test the different approaches on a blurred image, and study the differences in the results.

Sammendrag

Et kjent problem i signalbehandling er å rekonstruere det originale bildet, eller et signal av en annen type, fra et som er uskarpt eller på en annen måte ødelagt. Dette er er på ingen måte en triviell oppgave. Vi vil i denne oppgaven studere metoder for å finne gode approksimasjoner til dette ukjente, virkelige bildet. Hovedfokuset vil være på en minimeringsmetode basert på wavelet dekomponering av bilder, og å finne løsninger som har få koeffisienter som er større enn null med hensyn til wavelet basisen. Vi argumenterer deretter for at kunnskap om hvor kanter mellom objekter i bildet befinner seg kan brukes med det mål å forbedre metodens evne til å bevare disse. Derfor presenterer vi en metode for å detektere kanter, og innfører deretter kantavhenging vekting av koeffisientene i minimeringsmetoden. Dersom dette gir den ønskede effekten søker vi en mer elegant og effektiv metode. Vi låner en tilnærming brukt av Ambrosio og Tortorelli for å finne en approksimasjon til Mumford–Shah funksjonalen, og konstruerer en modifisert funksjonal hvor den kantavhengige vektingen finnes fra et fasefelt. Vi utvikler en alternerende minimeringsmetode, og tester den og de andre metodene på et uskarpt bilde, før vi studerer forskjellene i resultatene.

Preface

This thesis completes my studies in the Master's degree programme in Applied Physics and Mathematics, with specialization in Industrial Mathematics, at the Norwegian University of Science and Technology (NTNU).

Tremendous thanks are due to my supervisor, prof. Markus Grasmair. His guidance, support, patience and willingness to share his knowledge and expertise has been invaluable in the planning, execution and finalization of my work. I would also like to extend my gratitude to my friend Alexander Eide Silli for taking the time to proofread my thesis, saving me from some embarrassment and providing me with a few good laughs during my last night in the study hall.

Contents

List of Figures

Abbreviations

AT	=	Ambrosio and Tortorelli
DWT	=	Discrete wavelet transform
ITA	=	Iterative thresholding algorithm
MRA	=	Multiresolution analysis
MS	=	Mumford–Shah

Chapter 1

Introduction

A lot has happened since the first attempts at photography in the early 19^{th} century that required days of exposure, through it becoming available to the masses in the beginning of the 20^{th} century and to the fast pace, never ending production of digital images and videos happening today. Nowadays we use photography in many forms to study anything from micro structures to far away galaxies, and we expect the images to be of a certain quality. This, however, is often hindered by apparatus imperfections or other external factors. We therefore rely on image processing to help restore quality, and extract and otherwise process the information contained in the image.

In image processing, input images are processed using mathematical operations to produce a specific type of output, that being another image with a certain quality or other information extracted from the input. Two important image processing tasks are the processes of removing blur and noise from corrupted images. Noise and blur are unwanted errors naturally introduced in most images due to equipment imperfections or limitations and external disturbance such as camera motion, object motion or atmospheric disturbances.

In the problem of deblurring and denoising, the aim is to extract the true image from the blurred and noisy recorded image. If the nature of the blurring process is well known and defined, the process is called non-blind restoration, as opposed to blind restoration where such information is not available. In this paper, we will assume that this information is available, as blurring often is a result of a known process. As for noise, we will assume that it is additive, meaning it is independent of the input signal (the true image), and the recorded image is a sum of the true image and the noise. The deblurring and denoising problem now becomes the inverse problem of recovering the

true image from the corrupted recorded image. This is however an ill posed problem, so additional requirements on the solution has to be introduced.

In order to solve inverse problems, one can introduce mild assumptions on the solution through regularization. The assumptions could be on the size or smoothness of the solution, and solving a regularized problem should produce suitable approximations to the original problem. In 1989, Mumford and Shah proposed [15] the idea of letting such an approximation be a segmentation, or as they called it the "cartoon version" of the image. A segmented image is one that has been separated into smooth areas, and is of interest in many fields, such as medical imaging. The idea is to find areas that have approximately the same property, for example uniform color, and represent the image as a collection of such areas. Assuming some regularity of the imaged objects, Mumford and Shah claimed that an image can be segmented into piecewise smooth regions separated by a finite set of edges. The resulting approximation will look like a cartoon, with sharply drawn edges surrounding objects drawn without texture. Finding the approximation now becomes an optimization problem, minimizing a functional with respect to the piecewise smooth representation and the collection of edges. The functional they defined is, however, not easy to deal with in practice.Therefore, several approximations have been proposed, among them a phase field based approach by Ambrosio and Tortorelli.

In recent years, the idea of sparsity requirements on the solution has become popular. Instead of the previous Sobolev norm regularizations, this method require the solution to be sparse with respect to some orthonormal basis, that is: only a finite number of coefficients with respect to the orthonormal basis should be non zero. With this requirement, one could find solutions with some of the same characteristics as the segmented images described above. In order to achieve this, we will in this paper study problems regularized by a weighted ℓ^p-norm of the solutions with respect to a Haar wavelet basis. Wavelets provide an orthonormal basis well suited for decomposing images, and in addition, an equivalent norm with spaces well suited for describing smoothness properties of functions, namely Besov spaces. An iterative method to minimize a functional regularized by such a sparsity constraint has been proposed by Daubechies et al. [5], and we will study this method's reconstruction abilities on a blurred image. In addition we will try to improve the method's edge preservation abilities by incorporating an edge detector and a subsequent edge dependent weighting of the coefficients. We expect that this will in some sense allow us to favour nonzero coefficients in the areas of the image that constitute edges between objects.

Motivated by the similarities between the expected results of the Mumford–Shah approximation and the sparsity constrained approximation, we will propose a modified Mumford–Shah functional. Combining the phase field approach of the Ambrosio and Tortorelli approximation and the ℓ^p penalization of the wavelet coefficients, we hope to improve even further on the edge preservation capabilities of the reconstruction. We will in the following present, study and develop the methods needed to solve the mentioned problems. First, we will present some aspects of image processing, before we will introduce wavelet decomposition and some related topics. We then move on to the methods, introducing an iterative thresholding algorithm before proposing a modified Mumford–Shah idea. After developing the numerical methods, we test the implementations on a blurred image and study the results.

Chapter 2

Image processing

In signal processing, images are in some way processed to enhance or restore properties or to extract information not readily available, We will in this section introduce a few properties of images, two unwanted errors that can occur, and explain why the task of removing such errors can be challenging.

2.1 Images, blur and noise

Images are an important part of most peoples lives. We see live images through our eyes, capture images for the future with our cameras and rely on our dentist to assess our dental health from x-ray images. Micro organisms are studied through microscopes and far away galaxies through telescopes. With the technology we have today, we expect images to be of a certain quality.

What characterizes what we would call a "good" image, is among other things, that pictured objects are easily told apart from other objects or areas in the image with different characteristics, such as color. However, in the process of capturing and storing images, unwanted errors often occurs. Imperfections of devices, such as camera lenses or chips, or mistakes such as the camera moving can impact and diminish the quality of images. In image processing, images are analysed and manipulated with the goal to improve on such errors or otherwise extract information. In the following, a few types of these unwanted errors are presented.

2.1.1 Blur

We say that an image is blurred if part of, or the whole, image appears out of focus. Edges can be unclear, details are lost and the photographs can appear smudged. Blurring often occurs as a result of camera motion, defocus or lens or mirror defects, to mention a few. One famous example of a mirror defect causing blurry images comes from the Hubble Space Telescope [7].

In 1990 the Hubble telescope was launched into low Earth orbit. Its mission was to record images without some of the limiting factors that ground based telescopes face, such as background light and turbulence in the atmosphere. Within weeks of the launch it was clear that the telescope was not able to produce a sharp focus, and the image quality was much lower than expected. The fault was identified as being a result of the main mirror being ground to the wrong shape, so that parallel light rays reflected off the mirror near the center focus at a different point than light rays reflected off the mirror closer to its edges. This effect is known as spherical aberration, and the multiple focal points produce a blurred image. This blurring effect is well defined, and can be modelled by a convolution operator with a low pass filter. In the following, we will consider such blur modelled by convolution.

2.1.2 Noise

Noise is unwanted, random variation of the intensity in an image. In digital images, it appears as pixel level variation. Noise can occur during the process of taking an image, for example due to chip noise in poor lighting conditions or chip defects, or in digital post processing.

A common type of noise is Gaussian, additive and independent, meaning that each pixel value is a sum of the true value and a random fluctuation which is normally distributed and independent of the pixel value. This type is called Gaussian noise, and can for example be sensor noise caused by poor lighting conditions. In digital cameras, images are recorded by a chip that registers photons. By nature, the density of photons are subject to random fluctuation, so the recorded pixel value is always a sum of a true signal and a random error, the noise. So, even though such noise is always present, it is more noticeable in poor lighting conditions, meaning fewer photons, because the ratio between noise and the signal is higher that in good lighting conditions.

2.2 Ill posedness of the inverse problem

In this paper, we will consider an observed image g over a rectangular domain $\Omega \subset \mathbb{R}^2$. We assume that the image has been corrupted, or blurred, by some process, and possibly disturbed by some kind of noise. Mathematically, we can represent the observed image g as the sum of a convolution between a blur kernel h and a "perfect world" image u and an additive noise n, namely

$$g = h * u + n. \tag{2.1}$$

We will assume that the blur kernel h is known and symmetric. Also, we will assume that n is white Gaussian noise, meaning that the noise is drawn from an independent zero mean normal distribution with variance σ, so that $n \sim N(0, \sigma^2)$. Now, one could propose the idea of minimizing the functional

$$\int_\Omega (g - h * u)^2 dx$$

to find the "perfect world" image u. However, this problem is an inverse problem, and turns out to be ill posed.

First, let us recall a few known facts. We say that a function f in in the space L^2 if the following holds

$$\int f^2 dx < \infty,$$

and the sequence $\{c_k\}$ is in the space ℓ^2 if

$$\sum_k c_k^2 < \infty.$$

For simplicity, let us consider the one-dimensional setting, and try to solve the minimization problem in the Fourier domain. We know that for a function f the following holds

$$f \in L^2(\Omega) \Rightarrow \mathcal{F}(f) \in \ell^2(\mathbb{Z}). \tag{2.2}$$

Now if $f \in H^1(\Omega)$, this implies $f^{(1)} \in L^2(\Omega)$. Let f_m be the m-th Fourier coefficient of f, then we have

$$f_m(im) \in \ell^2(\mathbb{Z})$$

so

$$\sum_{m=-\infty}^{\infty} |f_m^2||m|^2 < \infty.$$

For our problem we are looking for a solution u in $H^1(\Omega)$, we know $h \in H^1(\Omega)$ but $n \notin H^1(\Omega)$. If we take h_m and n_m to be the m-th Fourier coefficients of h and n respectively, this means that $\sum_{m=-\infty}^{\infty} h_m^2 |m|^2 < \infty$ and $\sum_{m=-\infty}^{\infty} n_m^2 |m|^2 = \infty$. Taking the Fourier transform of Equation (2.1), we get

$$\mathcal{F}(g) = \mathcal{F}(h)\mathcal{F}(u) + \mathcal{F}(n).$$

Define y as the solution of $h * y = g$, so that we can write

$$\mathcal{F}(y)\mathcal{F}(h) = \mathcal{F}(g),$$

and

$$\mathcal{F}(y) = \frac{\mathcal{F}(g)}{\mathcal{F}(h)} = \mathcal{F}(u) + \frac{\mathcal{F}(n)}{\mathcal{F}(h)}.$$

Therefore

$$y_m = \frac{g_m}{h_m} = u_m + \frac{n_m}{h_m}.$$

Let us consider the last term in the equation above. Since h is in H^1 and n is not, the coefficients h_m must go to zero much faster than n_m, and $\frac{n_m}{h_m} \to \infty$. Thus, we are left with a solution that is not in $H^1(\Omega)$. To deal with this problem, we will need to introduce regularization terms.

Chapter 3

Wavelets

When working with signals it is sometimes desirable to extract information that is not readily available by studying the signal as is, compress it for more effortless storage or remove unwanted data, such as noise. One popular approach is the Fourier transform. This takes a signal from its original, often time or spatial, domain to the frequency domain. That is, after transforming a signal, we are left with information on exactly what frequencies occur in the signal. However, all information on when, or where, these frequencies occur are not available in the transformed signal. This can be a non-issue in certain cases, but quite problematic in others. To give an example, say we are studying a musical piece and want to reproduce it on an instrument of our choice. It would not help us much to know frequencies occurs in the piece, if we do not know when and in which order. Even though there are approaches to make such information available with Fourier transforms, like the Short Time Fourier Transform, this sacrifices the resolution of the solution. Another approach is to use the Wavelet transform [14].

3.1 Haar wavelets

To understand wavelets, we shall start with 1-dimensional signals and introduce the theory through Multiresolution Analysis. Then we will use this to extend the theory to the 2-dimensional case, which we need to study images.

3.1.1 The scaling function and wavelet

Wavelet analysis is similar to Fourier analysis in that it approximates a function as a sum of basis functions. Unlike the sin and cos base functions of the Fourier analysis however, which is only localized in frequency, the wavelet

basis functions are localized both in frequency and time. To generate the base functions, two prototype functions must be chosen. These are the scaling function, also called averaging function, ϕ and its corresponding wavelet ψ. The two functions must be compactly supported and satisfy the following requirements

$$\int_{-\infty}^{\infty} \phi(x)\, dx = 1,$$
$$\int_{-\infty}^{\infty} \psi(x)\, dx = 0,$$
$$\int_{-\infty}^{\infty} \phi(x)\, \psi(x)\, dx = 0.$$

Given these two functions we use translations and dilations of them to generate the basis functions. We will talk about the scale of a wavelet, referencing the size of its support, which is changed through the dilation of ψ. A large, or coarse, scale wavelet gives the main feature of the signal over an interval, while finer details will be showed by the small scale wavelets. There are different wavelet transforms, but as we in this paper study discrete signals in terms of images, we shall use the discrete wavelet transform (DWT). In order to understand the methods of the DWT, we will first introduce the Multiresolution Analysis (MRA).

3.1.2 Multiresolution analysis

The multiresolution analysis (MRA) is a framework suited to create basis of scaling functions and wavelets, and to decompose a signal into successively coarser approximations and details. We start with the space $L^2(\mathbb{R})$, that is, the space of all functions f for which

$$\int_{\mathbb{R}} f^2 dx < \infty.$$

The MRA consists of a sequence of nested subspaces V_j that approximates $L^2(\mathbb{R})$, and satisfies a few other criterions, all of which are summed up below.

1. The spaces are *nested*: $V_j \subset V_{j+1}$,
2. The union of the spaces is *dense* in $L^2(\mathbb{R})$, that is: $\overline{\cup_{j\in\mathbb{Z}} V_j} = L^2(\mathbb{R})$,
3. $\cap_{j\in\mathbb{Z}} V_j = \{0\}$,
4. The following scaling relation holds: $f(\cdot) \in V_j \Leftrightarrow f(2^{-j}\cdot) \in V_0$,

5. There is an $\phi \in V_0$ such that the set $\{\phi(x-k), k \in \mathbb{Z}\}$ is an orthonormal basis for V_0.

If the above holds, we say that the collection $\{V_j, j \in \mathbb{Z}\}$ is a MRA with scaling function ϕ.

The scaling function

We will now construct an orthonormal basis of the spaces V_j given the basis of V_0. The space V_0 has an orthonormal basis $\{\phi_{0k}, k \in \mathbb{Z}\}$, where

$$\phi_{0k}(x) = \phi(x-k).$$

In order to establish the orthonormal basis of V_j, we first use the scaling relation. If $f(2^{-j}x) \in V_j$, then we must have $f(x) \in V_0$. That is, it can be expressed as a sum of the basis functions $\{\phi(x-k), k \in \mathbb{Z}\}$. Let us substitute x with $2^j x$, which gives us that $f(x)$ can be expressed as a linear combination of $\{\phi(2^j x - k), k \in \mathbb{Z}\}$. To ensure orthonormality, we need

$$\int \phi\left(2^j x - k\right) \phi\left(2^j x - l\right) dx = \delta_{kl} = \begin{cases} 1 & \text{if } k = l \\ 0 & \text{if } k \neq l. \end{cases}$$

Let us calculate

$$\int \phi\left(2^j x - k\right) \phi\left(2^j x - l\right) dx = 2^{-j} \int \phi(x-k)\,\phi(x-l) = 2^{-j}\delta_{kl},$$

because the ϕ_{0k} are orthonormal. Thus, we have that if V_j is a multiresolution analysis with scaling function ϕ, the space V_j is spanned by the orthonormal basis

$$\left\{\phi_{jk}(x) = 2^{j/2}\phi\left(2^j x - k\right), k \in \mathbb{R}\right\}.$$

The spaces V_j are called approximation spaces because any $f \in L^2(\mathbb{R})$ can be approximated to a precision of our choice by a function in V_j, as long as we choose j large enough.

The wavelet

We have now established the basics of the scaling function, so let us bring the wavelet into the picture. Because $V_j \subset V_{j+1}$, we know we can express the scaling function $\phi(x) \in V_j$ as a sum

$$\phi(x) = \sum_{k \in \mathbb{Z}} h(k) \sqrt{2} \phi(2x - k) \tag{3.1}$$

where $h(k) = \langle \phi(x), \phi(2x-k) \rangle$. However, the opposite is not necessarily true, we are not guaranteed that a scaling function in V_{j+1} can be expressed as a sum of functions in V_j. With this in mind, we seek a space W_j which is the orthogonal complement of V_j to V_{j+1}, so that

$$V_{j+1} = V_j \oplus W_j.$$

If we can find basis functions for such a set, we can decompose a scaling function in V_{j+1} into a sum of coarser scale scaling functions and their orthogonal complements. We will therefore seek a set of basis functions for W_j in accordance with the MRA.

As for the scaling function, we start with the case $j = 0$. We know $\psi \in V_1$, meaning we can express it as a sum

$$\psi = \sum_k a_k \phi_{1k}.$$

Our goal now is to find the coefficients a_k such that ψ is orthogonal to the reference space V_0,

$$\psi \perp V_0,$$

and to its translates

$$\psi(x-k) \perp \psi(x) \quad \text{for } \{k \in \mathbb{Z} \backslash \{0\}\}.$$

It is possible to show, see for example [14], that these conditions are fulfilled if we define

$$a_k = (-1)^k h(1-k),$$

with $h(k)$ given by (3.1), so that

$$\psi(x) = \sum_{k \in \mathbb{Z}} (-1)^k h(1-k) \phi(2x-k).$$

Further, the set

$$\{\psi(x-k), k \in \mathbb{Z}\}$$

is an orthonormal basis for the space W_0. Similarly as for the scaling function, we can use the scaling relation of the MRA to find that a orthonormal basis for W_j is

$$\left\{\psi_{jk}(x) = 2^{j/2} \psi\left(2^j x - k\right), k \in \mathbb{Z}\right\}.$$

Decomposing a signal

Let us now see how we use these sets of basis functions to decompose a signal. Say we have a signal $f \in L^2(\mathbb{R})$ that we wish to decompose. Then, because of the density of $\overline{\cup_j V_j}$, one can find a N such that $\|f - f_N\|_{L^2} < \epsilon$ for $f_N \in V_N$ and a preassigned precision $\epsilon \geq 0$. Let us for ease of notation assume now that $f = f_N$. Then, we know that we can express the function as the sum

$$f = \sum_{k \in \mathbb{Z}} \langle f, \phi_{N,k} \rangle \, \phi_{N,k}.$$

Using the decomposition $V_N = V_{N-1} \oplus W_{N-1}$, we can write this as

$$f = \sum_{k \in \mathbb{Z}} \langle f, \phi_{N-1,k} \rangle \, \phi_{N-1,k} + \sum_{k \in \mathbb{Z}} \langle f, \psi_{N-1,k} \rangle \, \psi_{N-1,k}.$$

Continuing the decomposition of the ϕ's to a desired coarsest level, here chosen as $j = 0$, we finally arrive at the following wavelet expansion

$$f = \sum_{k \in \mathbb{Z}} \langle f, \phi_{0,k} \rangle \, \phi_{0,k} + \sum_{j=0}^{N} \sum_{k \in \mathbb{Z}} \langle f, \psi_{j,k} \rangle \, \psi_{j,k}.$$

We have now established the wavelet decomposition of a signal.

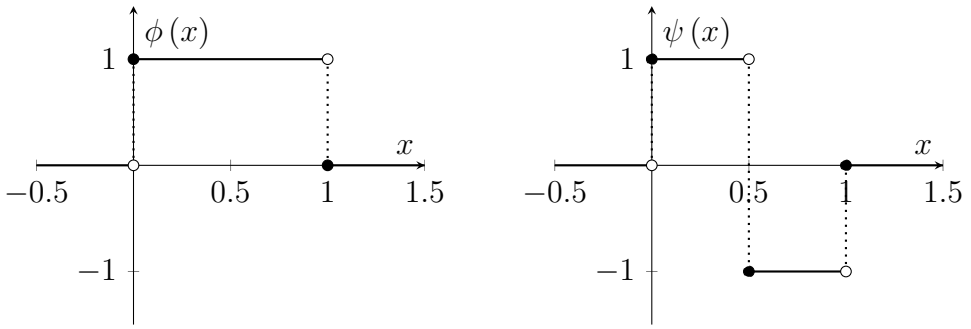
3.1.3 The Haar wavelet

There are many pairs of scaling functions ϕ and wavelets ψ, but we will use the oldest and simplest of them all, namely the Haar functions. The Haar scaling function is a box function defined by

$$\phi(x) = \begin{cases} 1 & \text{if } x \in [0,1), \\ 0 & \text{else,} \end{cases} \tag{3.2}$$

while the wavelet is defined as

$$\psi(x) = \begin{cases} 1 & \text{if } x \in \left[0, \frac{1}{2}\right) \\ -1 & \text{if } x \in \left[\frac{1}{2}, 1\right), \\ 0 & \text{else.} \end{cases} \tag{3.3}$$



(a) Haar scaling function. **(b)** Haar wavelet.

Figure 3.1: Haar functions.

3.1.4 Two dimensional wavelets

We have now established sets such that $L^2(\mathbb{R}) = \overline{\cup_j V_j}$. However, this theory can be extended to higher dimensions, and as we will work with images in $L^2(\mathbb{R}^2)$, we will introduce some theory for the 2-dimensional wavelet analysis.

The question now, is if we can approximate functions in $L^2(\mathbb{R}^2)$ as a sum of basis functions, and if yes, how do we find appropriate basis functions? Let's start with the relation

$$L^2\left(\mathbb{R}^2\right) = L^2(\mathbb{R}) \otimes L^2(\mathbb{R}),$$

which basically means that any function $u(x, y)$ in $L^2(\mathbb{R}^2)$ can be represented as a sum of products of two functions say f_k and g_k, in $L^2(\mathbb{R}^2)$, namely

$$u(x, y) = \sum_k f_k(x) g_k(y).$$

Using the Haar scaling functions defined in the previous section, this means that every function in $L^2(\mathbb{R}^2)$ can be approximated by characteristic functions of rectangles.

Now, again using the relation stated above, we shall construct the two-dimensional basis functions. We write

$$\begin{aligned} L^2\left(\mathbb{R}^2\right) &= L^2(\mathbb{R}) \otimes L^2(\mathbb{R}) \\ &= \overline{(\cup_j V_j) \otimes (\cup_j V_j)} \\ &= \overline{\cup_j V_j \otimes V_j}. \end{aligned}$$

Therefore, we define

$$\widehat{V_j} = V_j \otimes V_j$$

so that

$$L^2\left(\mathbb{R}^2\right) = \overline{\cup_j \widehat{V_j}}.$$

We then seek the orthogonal complement $\widehat{W_j}$ of $\widehat{V_j}$ in $\widehat{V_{j+1}}$, recalling the one dimensional relation

$$V_{j+1} = V_j \oplus W_j.$$

Using this and the definition of $\widehat{V_j}$, we find

$$\begin{aligned}\widehat{V_{j+1}} &= V_{j+1} \otimes V_{j+1} \\ &= (V_j \oplus W_j) \otimes (V_j \oplus W_j) \\ &= (V_j \otimes V_j) \oplus (V_j \otimes W_j) \oplus (W_j \otimes V_j) \oplus (W_j \otimes W_j)\,.\end{aligned}$$

We recognise the first tensor product as the already defined $\widehat{V_j}$, so we define

$$\widehat{W_j} = (V_j \otimes W_j) \oplus (W_j \otimes V_j) \oplus (W_j \otimes W_j)\,.$$

Now we recall the fact that if $u_k(x)$ is a basis of U and $v_k(y)$ is a basis of V, then $u_k(x)v_k(y)$ is a basis of $U \otimes V$. Using this relation, we can now express the 2 dimensional scaling function and wavelets in terms of the 1 dimensional definitions we already have in equations (3.2) and (3.3). The scaling function is defined as

$$\Phi(x, y) = \phi(x)\,\phi(y)\,.$$

As for the wavelets there are now three of them, all arising from the different tensor products in the definition of $\widehat{W_j}$, namely

$$\Psi(x, y) = \begin{cases} \phi(x)\,\psi(y) \\ \psi(x)\,\phi(y) \\ \psi(x)\,\psi(y)\,. \end{cases}$$

Haar wavelets

For the 2-dimensional Haar wavelets, this means that we have basis functions that are non-zero over rectangular domains,and we represent the functions in figure figure 3.2. We now have a framework we can use to decompose

(a) $\phi(x)\phi(x)$ **(b)** $\phi(x)\psi(y)$ **(c)** $\psi(x)\phi(y)$ **(d)** $\psi(x)\psi(y)$

Figure 3.2: 2-dimensional Haar functions.

2-dimensional signals, and we will put this into perspective with image processing in Section 3.2.

3.1.5 Notation

In the following, the notation would become cumbersome and confusing if we were to denote which of the tree versions of wavelets in the definition of Ψ we are dealing with, which of its translates and at which scale j. We shall therefore, for simplicity, abbreviate this full label by γ, and denote the full set of wavelet number, translates and scales by Γ. We shall also move away from the notations Φ and Ψ and simply use ϕ and ψ. Then we shall use the following shorthand notation for a specific wavelet coefficient of a function

$$f_\gamma = \langle f, \psi_\gamma \rangle .$$

3.2 Wavelets and image processing

In the world of image processing, two-dimensional wavelet transformations are widely used for compression, de-noising and other procedures. As an example the JPEG 2000 image coding system [9] is a wavelet based technology meant to supersede its discrete cosine based original JPEG standard. In this section we will have a quick look at how images are decomposed using two-dimensional wavelets.

3.2.1 Decomposing an image

Let us first study how a digital image is stored. For the sake of simplicity, let us assume that we are dealing with quadratic images of size 2^M by 2^M, that is, an image is represented by 2^{2M} pixels, and gray scale images only (extension into color images would simply be a matter of adding a dimension for the RGB values). Each pixel is assigned a value, in computing the value is commonly an integer in the range $[0, 255]$, describing its intensity where 0 is black and 255 is white. Instead of storing each pixel value, decomposition and compression methods are widely used to efficiently store and transmit images. We have already defined the 2 dimensional scaling function and wavelets, and below follows a simple description of how the decomposition is applied to the image.

In short terms, the wavelet decomposition process of an image is performed in the following manner: The process is first applied to the rows of the matrix, finding the average (the coefficient for the scaling function) and difference (the coefficient for the wavelet) between pairs of pixels that lie on the same row. This produces two new matrices of size $2^M \times 2^{M-1}$, an average matrix and a detail matrix. Then, the process is applied to the columns of

(a) Average. **(b)** Horizontal. **(c)** Vertical. **(d)** Diagonal.

Figure 3.3: 2-dimensional Haar functions.

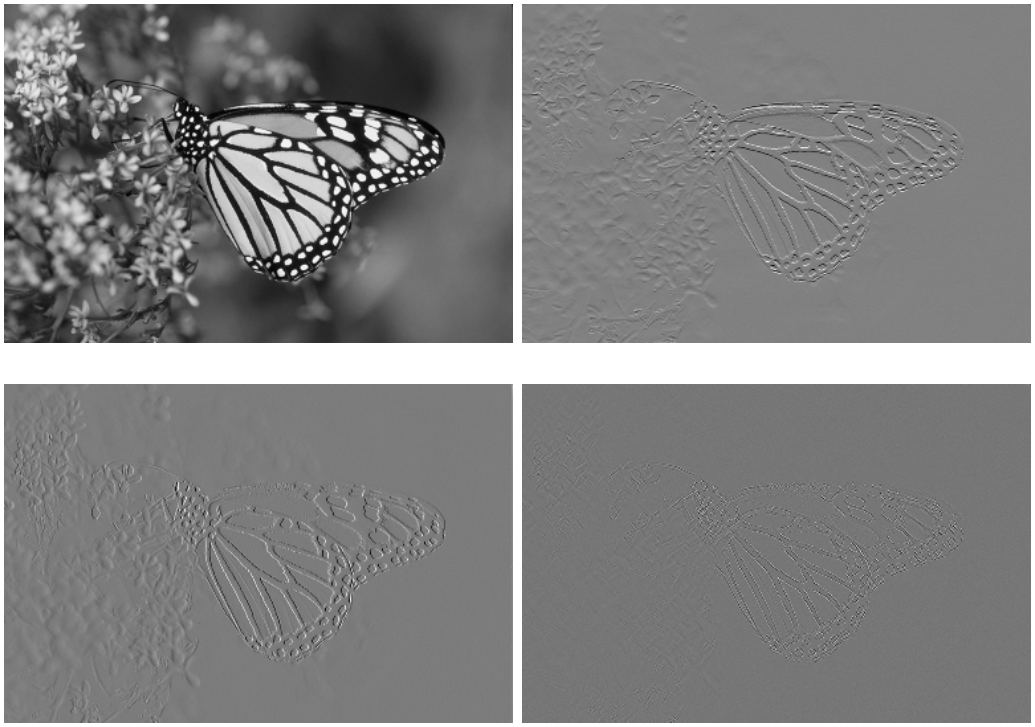
the two new matrices, producing four matrices of size $2^{M-1} \times 2^{M-1}$. The coefficients corresponding to the scaling function is now contained in one of the smaller matrices, while the other three contain the coefficients of the three different wavelets. We often say that the first wavelet, defined as $\phi(x)\psi(y)$ represent horizontal details, the second, namely $\psi(x)\phi(y)$, represent vertical details and the last one, $\psi(x)\psi(y)$, represent diagonal details, see figure 3.3. Figure 3.4 shows the averages and detail coefficients of the first step of a decomposition of an image. The scaling function coefficients are averages, and to continue the decomposition, the process can be applied again to the average matrix, producing four new matrices of size $2^{M-2} \times 2^{M-2}$. The process can be continued until we are left with a single average value and $2^{2M} - 1$ wavelet coefficients. If any of the coefficients are 0, we have to store fewer values, thus reducing the storage requirements of the image. We could also argue that some of the wavelet coefficients are so small that setting them equal to 0 would not change the appearance of the image visibly, and this would further reduce the storage requirements. This leads us to the concept of thresholding.

3.3 Thresholding

Have you ever uploaded an image to a social media website, and noticed that its quality seems to have been drastically reduced? This is an artefact due to a compression process the web site puts the file through, reducing its file size, sometimes at the cost of appearance of the image. A part of this compression process is a thresholding method. Despite the rather gloomy introduction, thresholding is actually quite useful when working with wavelet decompositions of signals.

After decomposing a signal into its scaling and wavelet coefficients, one could ask the question whether all the details (contained in the wavelet coefficients) are necessary, or if it could be sufficient to use just a few to reproduce an approximation of the original signal. Thresholding is a method where coefficients that are considered "small enough", in some chosen way, are removed. The idea is that the small coefficients can be removed without affecting the

(a) Approximation coefficients. **(b)** Horizontal detail coefficients.



(c) Vertical detail coefficients. **(d)** Diagonal detail coefficients.

Figure 3.4: First level wavelet decomposition of the monarch image, using Haar wavelets.

general picture noticeably. In the setting of an image, thresholding could be applied to the decomposed image with the idea that removing very small variations in the image would not be noticeable to the human eye. In that case, the file to be stored would be smaller than its original, a favourable feature for storage purposes. There exist however, as we briefly mentioned, a trade off between storage space and accuracy, in images manifesting itself as a possible decrease of image quality. One must therefore choose a thresholding approach suitable for the problem at hand. This is also the idea behind the methods we shall study later on; finding approximations where many of the coefficients are (near) zero, but the main features of the image preserved.

3.4 Besov spaces

From Fourier analysis we know that if ϕ_k is a Fourier basis, then we can represent a function $u \in L^2$ as

$$u = \sum_k c_k \phi_k.$$

We also know, from Parsevals identity, that the following relation holds

$$u = \sum_k c_k \phi_k \in L^2 \Leftrightarrow \sum |c_k|^2 < \infty,$$

that is

$$\{c_k\}_k \in \ell^2.$$

In addition, we can write the first derivative

$$u' = \sum_k ikc_k \phi_k,$$

so that, again using Parsevals identity,

$$u \in H^1 \Leftrightarrow \sum_k |kc_k|^2 < \infty$$

As we have already seen, the decomposition with respect to the Fourier bases is not always preferable, so it becomes natural that we ask ourselves if there exists a similar characterization as the ones above, just with wavelets? Also, instead of studying the properties of functions with, say, first or second derivatives, could is it possible to say something about functions with fractional derivatives, that is a derivative of order $p > 0$ of some function u with p not necessarily being an integer? The answer is yes!

Besov spaces $B^s_{p,q}\left(\mathbb{R}^d\right)$ are well suited for describing smoothness properties of functions, and consists of functions that have s derivatives in $L^p\left(\mathbb{R}^d\right)$, with the q providing some additional fine tuning. We shall focus our attention to the case when $p = q$. In short, functions that are mostly smooth, but have a few local irregularities, such as for example jump discontinuities, can still belong to a Besov space. As it turns out, wavelets provide an equivalence norm for the Besov spaces. As described in [17], this norm can be defined, for $s = 2$ and $d = 2$, as

$$\|u\|^p_{2,p} = \sum_{\gamma \in \Gamma} 2^{3pj-2j} |\langle u, \psi_\gamma \rangle|^p \tag{3.4}$$

Let $c_\gamma = 2^{3pj-2j}|u_\gamma|$, then we then have the relation

$$u \in B^2_{p,p}\left(\mathbb{R}^2\right) \Leftrightarrow \sum_\gamma |c_\gamma|^p < \infty$$

that is

$$\{c_\gamma\}_\gamma \in \ell^p.$$

Motivated by this, the term defined in equation (3.4) will be used as a penalization term for a minimization method.

Chapter 4

An iterative thresholding algorithm

In this chapter we will, as mentioned in the introduction, seek a approximate solution to the problem defined in (2.1) by imposing a regularizing term based on a weighted ℓ^p norm of the solution. In recent years, there has been a shift of focus from studying solutions with certain differentiability requirements, to sparsity requirement of the solution with respect to some basis. That is, to approximate the solution of the problem

$$\min \|Ku - g\|^2, \tag{4.1}$$

one imposes a regularizer such that only a finite number of coefficients of u with respect to some orthonormal basis $(\psi_\gamma)_{\gamma\in\Gamma}$ can be non zero. We will let the blurring kernel K be the convolution kernel studied previously, that is $Ku = h * u$. We will also assume that $\|K^*K\| < 1$, where K^* is the adjoint operator. In the following we will give the motivation behind the method, introduce the regularizer and present an algorithm for solving the problem.

4.1 Motivation

Recall that we are considering the problem of finding an object u from a blurred, noisy image g, in other words we wish to solve the problem

$$g = Ku + n.$$

As we already know, this is an ill-posed problem, and we therefore wish to seek an approximation thorough regularization techniques. We recall the discussion on images from section 2.1, and how non-blurred images differ from blurred ones. In a non-blurry image we expect rapid changes in intensity between objects in the image, so that they are easy to tell apart, which

usually is not the case for blurred images. This is where wavelets come in. The idea is to use a regularization term that promotes few but large wavelet coefficients, and we wish to ensure that the large wavelet coefficients appear where there should be edges between objects in the image.

We start by defining an orthonormal basis of $L^2(\mathbb{R}^2)$: we will consider a Haar-wavelet basis $(\psi_\gamma)_{\gamma\in\Gamma}$. Assume that we are given a set of strictly positive weights $\mathbf{w} = w_\gamma$. Now, we will define a functional where a penalization term is added to the discrepancy $(Ku - g)^2$. Nothing new so far, but instead of using a standard regularisation technique of, say, quadratic penalty, we will use weighted ℓ^p-norm of the coefficients of u with respect to the wavelet basis, letting $1 \leq p \leq 2$. We now recall the brief discussion of Besov spaces in section 3.4. As we already mentioned, mostly smooth functions with a few discontinuities can still belong to a Besov space. In a sense, this is what we expect from an image, slowly varying, that is mostly smooth, intensity within objects in the image, and rapid changes, possibly discontinuities, at the edges between them. This motivates the search for solutions u in the Besov space $B_p^2(\mathbb{R}^2)$. We use the wavelet based equivalence norm of the Besov space as the regularization term, and after incorporating the scaling factor 2^{3pj-2j} in the weights w_γ we can write this minimization problem as

$$\min_u \frac{1}{2}\|Ku - g\|^2 + \beta \sum_{\gamma\in\Gamma} \frac{w_\gamma}{2} |\langle u, \psi_\gamma\rangle|^p , \tag{4.2}$$

where $\beta > 0$ is a regularization parameter. Now, consider the case where the weights w_γ are constant, and let us briefly discuss how the penalisation depends on the parameter p. For $p = 2$, this is a standard quadratic penalty problem, where small coefficients, $|\langle u, \psi_\gamma\rangle| < 1$, are penalized less than larger ones, the ones for which $|\langle u, \psi_\gamma\rangle| > 1$. However, if we keep the weights fixed and decrease p from 2 to 1, we will increase the penalty on the small coefficients, and decrease the penalty on the larger ones, favouring solutions u with fewer but larger components with respect to $(\psi_\gamma)_{\gamma\in\Gamma}$. For $p < 2$, the minimization of (4.2) could promote a sparse representation of u in the wavelet basis. We should note that for $p > 1$, it is unlikely that any wavelet coefficients will be actually *equal* to zero, so we cannot expect real sparsity of the solution with this method, unless $p = 1$. Nonetheless, we shall consider also $p > 1$, hoping to find solutions with many near-zero coefficients.

4.2 The iterative algorithm

We now wish to solve the unconstrained minimization problem defined in (4.2). However, the variational equations that arise are coupled, and non-linear in the case $p \neq 2$, and so it is not immediately clear how to solve them. As proposed in [5], we shall therefore create a new optimization problem that we solve making iterative Forward and Backward Euler steps. To ease notation, let us define

$$\mathcal{R}_{w,p}(u) = \beta \sum_{\gamma \in \Gamma} \frac{w_\gamma}{2} |\langle u, \psi_\gamma \rangle|^p \tag{4.3}$$

We will now study the constrained optimization problem

$$\min \frac{1}{2} \|Ka - g\|^2 + \mathcal{R}_{w,p}(u), \quad \text{s.t. } a = u, \tag{4.4}$$

which is obviously equivalent to (4.2). To solve (4.4), we construct a function with a quadratic penalty term,

$$F(a, u) = \frac{1}{2} \|Ka - g\|^2 + \mathcal{R}_{w,p}(u) + \frac{1}{2} \|a - u\|^2 . \tag{4.5}$$

Let us assume that we have a pair of iterates (u^n, a^n). Then, to find a value for the next iterate a^{n+1}, we make a Forward Euler step in the direction of the negative gradient of F with respect to a, that is

$$\begin{aligned} a^{n+1} &= a^n - \nabla_a F(a^n, u^n) \\ &= a^n - [K^*(Ku^n - g) + (a^n - u^n)] \\ &= u^n - K^*(Ku^n - g) \end{aligned}$$

Next, we use a^{n+1} in (4.5), and minimize $F(a^{n+1}, u)$ with respect to u.

$$u^{n+1} = \arg\min F(a^{n+1}, u),$$

which means

$$0 \in \partial \left(\frac{1}{2} \left\| a^{n+1} - u \right\|^2 + \mathcal{R}_{w,p} \left(u^{n+1} \right) \right),$$

where ∂f is the *subgradient* of f, see for example Chapter 3 in [3] for a formal definition. Equivalently, we have

$$-u^{n+1} + a^{n+1} \in \partial \left(\mathcal{R}_{w,p} \left(u^{n+1} \right) \right).$$

This gives us a formula for updating u, namely

$$\begin{aligned} u^{n+1} &= (\mathbf{I} + \partial \mathcal{R}_{w,p})^{-1} \left(a^{n+1} \right) \\ &= (\mathbf{I} + \partial \mathcal{R}_{w,p})^{-1} \left(u^n - K^* \left(Ku^n - g \right) \right) \end{aligned}$$

where $\mathbf{I}$ is the identity.

4.2.1 The inverse map

Let us study the map $(\mathbf{I} + \partial\mathcal{R}_{w,p})^{-1}$. Define

$$\mathbf{S}_{w,p} = (\mathbf{I} + \partial\mathcal{R}_{w,p})^{-1},$$

so that $u^{n+1} = \mathbf{S}_{w,p}(u^n - K^*(Ku - g))$, and let us see if this inverse exists. There are two different cases that has to be treated separately due to the differentiability of the functional. We will see what happens when $p = 1$ or $p > 1$.

The case $p = 1$

Let us regroup. We want to find a formula for u^{n+1} so that it solves

$$(\mathbf{I} + \partial\mathcal{R}_{w,p})\left(u^{n+1}\right) = a^{n+1},$$

which can be expressed as

$$u^{n+1} + \beta\frac{wp}{2}\operatorname{sign}\left(u^{n+1}\right)\left|u^{n+1}\right|^{p-1} = a^{n+1}.$$

We note that the equations for the wavelet coefficients u_γ^{n+1} decouple, so that we can solve for each one separately. For $p = 1$, the regularization term is differentiable only for $u_\gamma \neq 0$. For the non-zero coefficients, the equation reduces to

$$u_\gamma^{n+1} + \beta\frac{w_\gamma}{2}\operatorname{sign}\left(u_\gamma^{n+1}\right) = a_\gamma^{n+1}.$$

Let us assume that $u_\gamma^{n+1} > 0$. Then we have the equation

$$\begin{aligned} u_\gamma^{n+1} + \beta\frac{w_\gamma}{2} &= a^{n+1} \\ u_\gamma^{n+1} &= a^{n+1} - \beta\frac{w_\gamma}{2}. \end{aligned}$$

Therefore, for $u_\gamma^{n+1} > 0$, we need to require $a_\gamma^{n+1} > \beta\frac{w_\gamma}{2}$. If, on the other hand we assume $u_\gamma^{n+1} < 0$, we solve

$$\begin{aligned} u_\gamma^{n+1} - \beta\frac{w_\gamma}{2} &= a^{n+1} \\ u_\gamma^{n+1} &= a^{n+1} + \beta\frac{w_\gamma}{2}, \end{aligned}$$

and must have $a_\gamma^{n+1} < -\beta\frac{w_\gamma}{2}$. Now we have two conditions on a_γ^{n+1}, and if it does not satisfy either of the two, we let $u_\gamma^{n+1} = 0$. To summarize, we have that

$$u_\gamma^{n+1} = S_{w_\gamma,1}\left(a_\gamma^{n+1}\right) = S_{w_\gamma,1}\left(u_\gamma^n - \left[K^*(Ku^n - g)\right]_\gamma\right)$$

where the function $S_{w,1}$ is defined as

$$S_{w_\gamma,1}(x) = \begin{cases} x - \beta\frac{w_\gamma}{2} & \text{if } x > \beta\frac{w_\gamma}{2}, \\ 0 & \text{if } |x| \leq \beta\frac{w_\gamma}{2}, \\ x + \beta\frac{w_\gamma}{2} & \text{if } x < -\beta\frac{w_\gamma}{2}. \end{cases} \tag{4.6}$$

The case $p > 1$

For $p > 1$, the functional is differentiable for all u_γ, and the map reduces to

$$S_{w_\gamma,p}(u_\gamma) = \left(u_\gamma + \beta\frac{w_\gamma p}{2}\operatorname{sign}(u_\gamma)|u_\gamma|^{p-1}\right)^{-1}.$$

For $w_\gamma \geq 0$, the function $u_\gamma + \beta\frac{w_\gamma p}{2}\operatorname{sign}(u_\gamma)|u_\gamma|^{p-1} = a_\gamma$ is a one-to-one map, so the inverse $S_{w_\gamma,p}(a)$ exists. However, it does not in general have an analytical expression. We must therefore use a numerical method to approximate the solution. We will study such a method in section 4.4.

4.2.2 Algorithm

We arrive at the following iterative scheme: for $u^0, g \in L^2(\Omega)$, K a bounded operator and $1 \leq p \leq 2$, given the orthonormal basis $(\psi_\gamma)_{\gamma\in\Gamma}$ and a set of positive weights $(w_\gamma)_{\gamma\in\Gamma}$, we can approximate the solution of (4.4) by iteratively updating

$$u^{n+1} = \mathbf{S}_{w,p}\left(u^n - K^*\left(Ku^n - g\right)\right). \tag{4.7}$$

In [5] they prove that $u^{n+1} = \mathbf{S}_{w,p}(a^{n+1})$ is in fact a minimizer of $F(u, a^{n+1})$ and that the iterative scheme in equation (4.7) approaches a minimizer of the original problem (4.2).

4.3 Edge detection and weights

Let us now consider how to choose the weights w_γ. The obvious initial choice of letting the weights be only decomposititon-level dependent should produce a result of smooth regions separated by jumps in intensity. We know however, that these jumps should happen at the edges of the objects in the image, and we could assume that not much important information would be lost if for example textural components were lost. Therefore, one could argue that if we could penalize the wavelets that correspond to edges in the image *less* than the ones that do not, and thus favouring these being large, we might enhance the algorithms edge preservation abilities. We will therefore present a method for edge detection.

4.3.1 Edge detection by wavelet size

Following an idea proposed in [6], we present a, as they call it, "not very sophisticated" edge detector algorithm. The algorithm is based on the following idea: one would expect wavelet coefficients u_γ of fine decomposition scales to be large in the presence of an edge, but also for oscillating components, like texture, which does not constitute edges. Texture components, however, typically only appear as large wavelet components for fine scales, while wavelet coefficients in a wider range of scales would be affected in the presence of an edge. Therefore, to avoid assuming that oscillating components are edges, the following approach is proposed.

First, we choose a set of decomposition scales $\{j_{\text{fine}}, ..., j_{\text{coarse}}\}$ in which we want to search for edges. For each decomposition level in this set, and all orientations of each level, compute the mean value of the absolute value of the corresponding wavelet coefficients. Set a threshold parameter t_γ proportional to this mean value. Then, for all positions x, we test if the corresponding wavelet coefficients $|u_\gamma|$ with x in the support of the wavelet for the decomposition scales $\{j_{\text{fine}}, ..., j_{\text{coarse}}\}$ are larger than the threshold parameters t_γ. That is, if

$$x \in \operatorname{supp} \psi_\gamma$$

we check if the following holds

$$|u_\gamma| > t_\gamma. \tag{4.8}$$

If (4.8) holds for x for all $j \in \{j_{\text{fine}}, ..., j_{\text{coarse}}\}$, then we say that x is an edge. Choosing level dependent weights $w_{\gamma,\text{e}}$ for edges and $w_{\gamma,\text{s}}$ for non-edge coefficients, we define

$$w_\gamma(x) = \begin{cases} w_{\gamma,\text{e}} & \text{if } j \in \{j_{\text{fine}}, ..., j_{\text{coarse}}\} \text{ and } x \text{ is an edge,} \\ w_{\gamma,\text{s}} & \text{else.} \end{cases}$$

Using this edge detector to assign weights, we aim to improve the edge preservation capabilities of the ITA.

4.4 Algorithmic details

To be able to implement the iterative algorithm (4.7), there are a few procedures that need to be studied. We must find a way to update $u^{n+1} = \mathbf{S}_{w,p}(a^{n+1})$ for $p > 1$, and also choose a proper stopping criterion for the ITA. We start by presenting Newtons method to find the inverse map.

4.4.1 Newtons method for finding the map $\mathbf{S}_{w,p}$

As we have seen, when $p = 1$, there is an analytical expression for the inverse map $\mathbf{S}_{w,p} = (\mathbf{I} + \partial\mathcal{R})^{-1}$, however this is in general not the case for $p > 1$. Therefore, we need to approximate

$$u^{n+1} = \mathbf{S}_{w,p}\left(a^{n+1}\right) \tag{4.9}$$

numerically in the case $p > 1$. That is, find an approximation to the u_γ^{n+1} that solves the equation

$$u_\gamma^{n+1} + \beta\frac{w_\gamma p}{2}\operatorname{sign}\left(u_\gamma^{n+1}\right)|u_\gamma^{n+1}|^{p-1} = a_\gamma^{n+1}. \tag{4.10}$$

A well known approach for finding such a root is called Newtons method (proper reference). Newtons method approximates the root of a function $f(x) = 0$ from a well chosen first guess x_0, by the iterative process

$$x_{i+1} = x_i - \frac{f(x_i)}{f'(x_i)}. \tag{4.11}$$

In short, what the method does, is to move along the tangent line to the curve in the point $(x_i, f(x_i))$ and choose as x_{i+1} the point where this line intersects with the x-axis.

Choosing the starting points

Let us for now omit the sub- and superscripts, and study the function in question, namely

$$f(u) = u + \beta\frac{wp}{2}\operatorname{sign}(u)|u|^{p-1},$$

and quickly recall that we want to approximate the u^* that solves $f(u) = a$. To be sure the method converge towards the solution, we need to choose a good starting point u_0. Figure 4.1 shows a sketch of the function $f(u)$. We note that if $a = 0$, the solution is simply $u^* = 0$. However, if $a > 0$, then u^* must also be positive, while if $a < 0$, so is u^*. Also, $f(u)$ is a concave, monotonically increasing function for $u < 0$ and a convex monotonically decreasing function for $u > 0$. We will in the following look at these two situations separately.

As mentioned, when $u > 0$ we $f(u)$ is a convex, monotonically increasing function, in fact it now reduces to

$$f(u) = u + \beta\frac{wp}{2}u^{p-1}.$$

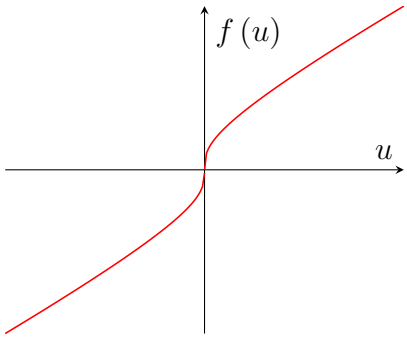


Figure 4.1: Sketch of the function $f(u)$.

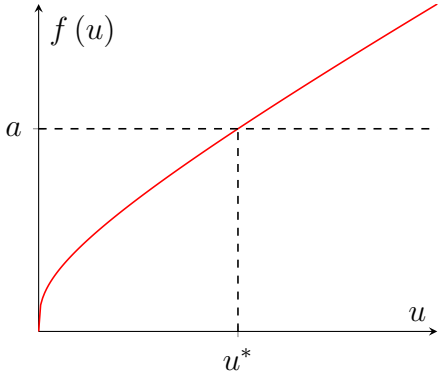


Figure 4.2: Sketch of the function $f(u)$ for $u > 0$.

Figure 4.2 shows a sketch of $f(u)$, and a positive a with the corresponding solution u^*. The newton method will, as described previously, move along the tangent line of the function in the current point, until it crosses the line a. Therefore, it seems reasonable to choose a starting value u_0 that is smaller than u^*, because the tangent line of the function lies above the function and crosses the line a before it passes u^*. Thus, all following iterates $u^n \leq u^*$.

To choose $u_0 < u^*$, we use the test value 1. If $f(1) = a$, all is good and we have found the exact solution $u^* = 1$. If $f(1) > a$ however, we know that $u^* < 1$, which in turn implies that u_0 should be smaller than 1. We want to find a u_0 such that $u_0 + \frac{wp}{2}u_0^{p-1} < a$, and observe that since $0 < u_0 < 1$, we have $u_0^{p-1} > u_0$. Then it holds that

$$u_0 + \beta\frac{wp}{2}u_0^{p-1} < u_0^{p-1} + \beta\frac{wp}{2}u_0^{p-1}.$$

Therefore, we choose the u_0 that satisfies

$$u_0^{p-1} + \beta \frac{wp}{2} u_0^{p-1} = a,$$

namely

$$u_0 = \left(\frac{a}{1 + \beta \frac{wp}{2}} \right)^{\frac{1}{p-1}}.$$

If $f(1) < a$ we know that $u^* > 1$. So, let's choose a starting point $1 < u^0 < u^*$. Now we have that $u_0^{p-1} < u_0$, so

$$u_0 + \beta \frac{wp}{2} u_0^{p-1} < u_0 + \beta \frac{wp}{2} u_0.$$

Therefore, we choose u_0 such that

$$u_0 + \beta \frac{wp}{2} u_0 = a$$

and find

$$u_0 = \frac{a}{1 + \beta \frac{wp}{2}}.$$

Now, let us look at the case when $a < 0$, and thus $u^* < 0$. Figure 4.3

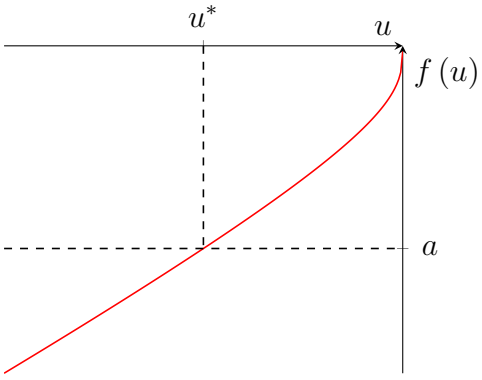


Figure 4.3: Sketch of the function $f(u)$ for $u < 0$.

shows a sketch of the function and a sample a and corresponding u^*. Following the same reasoning as for $u^* > 0$, we now deduce that we need to choose a starting point u_0 larger than u^*.We now use the test value $u = -1$, so that if $f(-1) = a$ we know $u^* = -1$, otherwise if $f(u) > a$ then $u^* > -1$ and if

$f(-1) < a$ we have $u^* < -1$. We repeat the process from before and arrive at the following initial guesses: if $f(u) > a$ then

$$u_0 = -\left(\frac{|a|}{1+\beta\frac{wp}{2}}\right)^{\frac{1}{p-1}},$$

if $f(u) < a$ then

$$u_0 = \frac{a}{1+\beta\frac{wp}{2}}.$$

All in all, we have the following scheme

$$\begin{aligned} &f(\operatorname{sign}(a)) = a, && u^* = \operatorname{sign}(a), \\ &|f(\operatorname{sign}(a))| > |a|, && u_0 = \operatorname{sign}(a)\left(\frac{|a|}{1+\beta\frac{wp}{2}}\right)^{\frac{1}{p-1}}, \\ &|f(\operatorname{sign}(a))| < |a|, && u_0 = \frac{a}{1+\beta\frac{wp}{2}}. \end{aligned} \tag{4.12}$$

The algorithm

All in all, for each wavelet coefficient u_γ^{n+1} we find the new value either directly from the first equation in (4.12), or approximate it by the Newton iterations presented as pseudocode in algorithm 1.

Algorithm 1 Newtons method

Given a, choose u_0 in accordance with (4.12). Choose a tolerance τ and maximum iterations $i_{\max}$.
$i = 0$
while $f(u_i) - a > \tau$ and $i < i_{\max}$ **do**
  $u_{i+1} = u_i - \left(u_i + \beta\frac{wp}{2}\operatorname{sign}(u_i)|u_i|^{p-1} - a\right) \cdot \left(1 + \beta\frac{wp}{2}(p-1)|u_i|^{p-2}\right)^{-1}$
  $i = i + 1$
end while
Set $u_\gamma^{n+1} = u_i$

4.4.2 Stopping criterion for the thresholding algorithm

Let us now try to find a proper stopping criterion for the iterative thresholding algorithm. Recall that we have the fixed point iteration

$$\mathbf{u}^{n+1} = \mathbf{S}_{wp}\left(\mathbf{a}^{n+1}\right) = \mathbf{G}\left(\mathbf{u}^n\right)$$

We would like to stop the iterations when we have an iterate u^n close enough to the regularized solution u_α. But what is "close enough"? To answer this, let us take a step back.

Exact, regularized and approximate solutions

Our goal is to find an approximate solution to the problem

$$Ku = g, \tag{4.13}$$

by solving a regularized problem

$$\min \|Ku - g\|^2 + \mathcal{R}_{w,p}(u). \tag{4.14}$$

However, we assume we are only given noisy data g^δ, and that this satisfies $\left\|g - g^\delta\right\| \leq \delta$. Let us call u_β^δ the solution of (4.14) with noisy data g^δ, and u^* the exact solution of $Ku - g = 0$. One idea could therefore be to stop the iterations when the distance between the iterate and the regularized solution is smaller than the distance between the exact solution and the regularized solution, that is when

$$\left\|u_\beta^\delta - u^n\right\| < \left\|u_\beta^\delta - u^*\right\|. \tag{4.15}$$

Let us therefore look for bounds for these two norms.

Bound for the norm $\left\|u_\alpha^\delta - u^n\right\|$

Since we are working with a fixed point iteration, we know from the Banach fixed point theorem, see for example Chapter 5 in [2], that u^n satisfies the following a-posteriori estimate

$$\left\|u^n - u_\alpha^\delta\right\| \leq \frac{C}{1-C}\left\|u^n - u^{n+1}\right\|,$$

where $C < 1$ is a contraction factor. As $C \approx \left\|D\mathbf{G}_{wp}\left(u_\alpha^\delta\right)\right\| \approx \left\|D\mathbf{G}_{wp}\left(u_\alpha^\delta\right)\right\|$, we need to calculate this. We start by finding an expression for the derivative

$$\begin{aligned} DS_{w,p}\left(a^{n+1}\right) &= D\left[\left(\mathbf{I} + \partial\left(\mathcal{R}_{w,p}\right)\right)^{-1}\left(u^n - K^*\left(Ku^n - g\right)\right)\right) \\ &= \left[D\left(\mathbf{I} + \partial\mathcal{R}_{w,p}\right) \circ \left(\mathbf{I} + \partial\mathcal{R}_{w,p}\right)^{-1}\left(a^{n+1}\right)\right]^{-1} \circ \left(\mathbf{I} - K^*K\right) \\ &= \left[D\left(\mathbf{I} + \partial\mathcal{R}_{w,p}\right)\left(u^{n+1}\right)\right]^{-1} \circ \left(\mathbf{I} - K^*K\right) \end{aligned}$$

where we have used the relation

$$D\left(F^{-1}\right) = \left[(DF) \circ F^{-1}\right]^{-1}.$$

Then, using the relation

$$\|AB\| \leq \|A\| \cdot \|B\|,$$

and the requirement we put on K,

$$\|\mathbf{I} - K^*K\| \leq 1,$$

we find an upper bound for the norm

$$\begin{aligned}\left\|DS_{w,p}\left(a^{n+1}\right)\right\| &= \left\|\left[D\left(\mathbf{I}+\partial\mathcal{R}_{w,p}\right)\left(u^{n+1}\right)\right]^{-1} \circ (\mathbf{I}-K^*K)\right\| \\ &\leq \left\|\left[D\left(\mathbf{I}+\partial\mathcal{R}_{w,p}\right)\left(u^{n+1}\right)\right]\right\| \|(\mathbf{I}-K^*K)\| \\ &\leq \left\|\left[D\left(\mathbf{I}+\partial\mathcal{R}_{w,p}\right)\left(u^{n+1}\right)\right]\right\|.\end{aligned}$$

So, let us try and find an expression for this norm. From the definition

$$\mathcal{R}_{w,p}(u) = \frac{\beta}{2}\sum_{\gamma} w_\gamma |u_\gamma|^p$$

we know that

$$\partial\mathcal{R}_{w,p}(u) = \frac{\beta}{2}\operatorname{diag}\left(pw_\gamma \operatorname{sign}(u_\gamma)|u_\gamma|^{p-1}\right)$$

where $\operatorname{diag}(x)$ is a diagonal matrix with x as entries. From there find the expression

$$D\left(\partial\mathcal{R}_{w,p}(u)\right) = \frac{\beta}{2}\operatorname{diag}\left(p(p-1)w_\gamma |u_\gamma|^{p-2}\right)$$

Thus, we know that

$$D\left(\mathbf{I}+\partial\mathcal{R}_{w,p}\right)\left(u^{n+1}\right) = \operatorname{diag}\left(1+\frac{\beta}{2}p(p-1)w_\gamma\left|u_\gamma^{n+1}\right|^{p-2}\right)$$

and must have

$$\left[D\left(\mathbf{I}+\partial\mathcal{R}_{w,p}\right)\left(u^{n+1}\right)\right]^{-1} = \operatorname{diag}\left(\left(1+\frac{\beta}{2}p(p-1)w_\gamma\left|u_\gamma^{n+1}\right|^{p-2}\right)^{-1}\right).$$

Finally, we arrive at an expression for the norm

$$\left\|\left[D\left(\mathbf{I}+\partial\mathcal{R}_{w,p}\right)\left(u^{n+1}\right)\right]^{-1}\right\| = \max_\gamma \operatorname{diag}\left(\left(1+\frac{\beta}{2}p(p-1)w_\gamma\left|u_\gamma^{n+1}\right|^{p-2}\right)^{-1}\right).$$

Bound for the norm $\left\|u_\alpha^\delta - u^*\right\|$

We will not treat this bound in detail here, but refer to the results in [8], most importantly Preposition 8 . The main result is that, assuming $\left\|g - g^\delta\right\| \leq \delta$ holds and u_β^δ is a regularized solution, then if β and δ are sufficiently small, the following bound exists

$$\frac{1}{2}\left\|u_\beta^\delta - u^*\right\|^2 \lesssim \left(2\beta + \frac{\delta^2}{2\beta}\right) \max_\gamma \left|\left\langle u_\beta^\delta, \psi_\gamma \right\rangle\right|^{2-p}.$$

So, if we assume that $u^{n+1} \sim u_\beta^\delta$, we have the bound

$$\left\|u_\beta^\delta - u^*\right\| \lesssim \left(\left(4\beta + \frac{\delta^2}{\beta}\right) \max_\gamma \left|u_\gamma^{n+1}\right|^{2-p}\right)^{1/2},$$

which is dependent on the current iterate.

Stopping criterion

All in all, choose to stop the ITA when the iterate is such that

$$\begin{aligned} \max_\gamma \operatorname{diag}\left(\left(1 + \frac{\beta}{2}p(p-1)w_\gamma \left|u_\gamma^{n+1}\right|^{p-2}\right)^{-1}\right) \\ \leq \left(\left(4\beta + \frac{\delta^2}{\beta}\right) \max_\gamma \left|u_\gamma^{n+1}\right|^{2-p}\right)^{1/2}. \end{aligned} \tag{4.16}$$

Chapter 5

A modified Mumford–Shah functional

In 1989 Mumford and Shah proposed a functional used for simultaneous image segmentation and restoration. Based on an idea of the relationship between real world objects and their 2-dimensional projections onto images, they constructed the functional, which measures the difference between an image and an approximation consistent of sets of smooth regions and jumps. In the following we shall introduce the original Mumford–Shah functional and an approximation by Ambrosio and Tortorelli [1]. Then, we will propose a modified functional based on the ℓ^p penalty approach we have already seen.

5.1 The Mumford–Shah functional

In their famous paper from 1989 [15], Mumford and Shah propose a functional that measures the degree of match between a (possibly blurred and noisy) input image g and a segmentation u. In the paper, they argue that since images are 2-dimensional projections of our three dimensional world we can make certain assumptions on the properties of the objects in the image. The assumptions are that an object will have approximately the same color, or gray scale, everywhere, and boundaries between objects will appear as rapid changes in the color or intensity. Thus, one should be able to find a piecewise smooth approximation of the image g as a collection of smooth regions u separated by a jump set S. This is known as image segmentation. Mumford and Shah described u as the cartoon version of the input g. Assuming some regularity of the objects in the image, the size of the jump set S is finite. With this in mind, we can introduce the Mumford–Shah energy.

5.1.1 The weak and strong Mumford–Shah energy

We define the strong Mumford Shah energy as

$$\mathcal{E}(u,S) = \int_\Omega (Ku-g)^2\,dx + \beta \int_{\Omega\backslash S} |\nabla u|^2\,dx + \alpha \cdot \text{length}(S\cap\Omega). \quad (5.1)$$

The first integral measures the difference between the blurry input image g and the blurred segmented approximation Ku, the second term measures the size of the gradient of u within the regions separated by S, while the third term measures the length of the edges. The idea is to find the best piecewise smooth approximation of the sharp image, u, by minimizing (5.1) with respect to u and S. The parameters α and β are positive and their value controls the level of segmentation of the image, and they need both be larger than 0. Otherwise, if $\beta = 0$, the size of the second integral becomes irrelevant, and we can choose $S = \{\emptyset\}$. This means that the third term is also equal to 0, and we are left with the problem of minimizing $\int_\Omega (Ku-g)^2\,dx$, which we have already seen is ill-posed. If we on the other hand choose $\alpha = 0$, the size of length$(S\cap\Omega)$ is irrelevant, so we can choose $S = \Omega$. This makes $\Omega\backslash S = \{\emptyset\}$, so the second integral is equal to 0, and we are again left with the ill posed problem of minimizing $\int_\Omega (Ku-g)^2\,dx$. Therefore, we need $\alpha, \beta > 0$, and note that if we let β become larger we require more smoothness of u, and if we let α become larger we require the edge set S to be smaller.

They continue with the definition of a weak functional, where a new edge set S_u dependent on u is defined, and its length measured by the one dimensional Hausdorff measure $\mathcal{H}^1(S_u)$.

$$\mathcal{E}(u) = \int_\Omega (Ku-g)^2\,dx + \beta \int_{\Omega\backslash S_u} |\nabla u(x)|^2\,dx + \alpha\mathcal{H}^1(S_u). \quad (5.2)$$

Neither the strong nor the weak functionals are practical, however, so several methods of dealing with the problem has been proposed. In 1990 Ambrosio and Tortorelli proposed an approximation based on a phase field approach [1].

5.1.2 The Ambrosio and Tortorelli approximation

In the core of the phase field approach lies the replacement of the edge set S_u by a function, phase field, that takes two distinct values for the different states: edge/not edge. Defining a function v that is close to 0 in the presence

of an edge, and 1 otherwise, Ambrosio and Tortorelli proposed replacing the last term of equation (5.2) by two new, competing terms

$$\alpha \int_{\Omega} \varepsilon \left|\nabla v\right|^2 + \frac{\left(v-1\right)^2}{4\varepsilon} dx.$$

The first term $\left|\nabla v\right|^2$ favours local homogeneity of the functional, while the second term favours $v \sim 1$. The scale parameter ε controls the "width" of the phase field. Then, they replace the middle term of equation (5.2) by

$$\beta \int_{\Omega} v^2 \left|\nabla u\right|^2 dx,$$

so that this term vanishes when $v \sim 0$. Thus, this term measures the size of the gradient of u away from the edges. We are left with the following functional

$$F_{\varepsilon}\left(u,v\right) = \int_{\Omega} \left(Ku-g\right)^2 dx + \beta \int_{\Omega} v^2 \left|\nabla u\right|^2 dx + \alpha \int_{\Omega} \varepsilon \left|\nabla v\right|^2 + \frac{\left(v-1\right)^2}{4\varepsilon} dx.$$

It can be shown, see for example Chapter 4 in [4], that this functional Γ-converges to the Mumford–Shah functional.

5.2 Modification

The iterative algorithm studied in chapter 4 favours some of the same properties of u as the Mumford–Shah approach; the ℓ^p regularization term can be utilized to produce approximations which are quite smooth in most regions, because the method favour solutions with few but large wavelet coefficients. This motivates the modified Mumford–Shah approach we will study in this section. We replace the second term in (5.1) by the ℓ^p penalization term $\mathcal{R}_{w,p}\left(u\right)$ defined in equation (4.3), and define a weight function dependent on the phase field, $w_{\gamma}\left(v\right)$.

5.2.1 From quadratic to ℓ^p penalty

As we have discussed, the second term in the MS functional penalizes the size of the gradient of u outside the edge set S with quadratic penalty. Let us attempt replacing this by a ℓ^p penalization with $1 \leq p \leq 2$. Assuming the Ambrosio and Tortorelli approximation is still applicable, we define a modified version

$$F_{\varepsilon}^{MOD}(u,v) = \int_{\Omega} \left(Ku-g\right)^2 dx + \beta \sum_{\gamma} w_{\gamma} \left|\langle u, \psi_{\gamma}\rangle\right|^p + \alpha \int_{\Omega} \varepsilon \left|\nabla v\right|^2 + \frac{\left(v-1\right)^2}{4\varepsilon} dx.$$

We here assume $v(x)$ to be the phase field as in the original AT approximation, so we need to be able to construct a weight function dependent on v to appropriately penalize the correct wavelets in the sum.

5.2.2 The weight function

To define the weight function, let us recall what we want to achieve: wavelet that constitutes an edge should be penalized less than wavelets in smooth regions of the image. In the phase field, edges are indicated by $v(x) = 0$. Thus, if there is an edge in a pixel, or several pixels, contained in the support of a wavelet, this wavelet should we weighted as an edge. Therefore, we propose the following weight function

$$w_\gamma(v)(x) = \min\left\{v(y)^2 + \kappa : y \in \operatorname{supp}\psi_\gamma\right\} \cdot s_\gamma, \tag{5.3}$$

where $s_\gamma = 2^{3pj-3j}$ is the level dependent scaling parameter defined in section 3.4, and κ is some small parameter $0 < \kappa < 1$ such that $w_\gamma \neq 0$. Thus, if any $v(x)$ for $x \in \operatorname{supp}\psi_\gamma$ is less than one, marking the presence of an edge, that wavelet coefficient will be weighted less and thus preferred over those whose wavelets have $v(x) = 1$ in all of its support.

5.3 Minimizing the modified functional

Having defined the modified functional, we shall now develop a procedure for minimizing it with respect to u and v. The functional is separately convex with respect to u and v, so this motivates us to develop an alternating minimization approach. Keeping one variable fixed, we shall approximate a minimizer of the functional in terms of the other variable. Having done this, we shall switch, fixing the second variable and minimizing with respect to the first. Continuing with this alternation, we wish to approximate minimizers of the functional $F_\varepsilon^{MOD}(u, v)$.

5.3.1 Minimizing with respect to u

Assuming we have an approximation of v, for example an initial guess of, say, $v \equiv 1$, we can minimize the functional with respect to u. In this case, the third term of $F_\varepsilon^{MOD}(u, v)$, namely $\alpha \int_\Omega \varepsilon |\nabla v|^2 + \frac{(v-1)^2}{4\varepsilon} dx$ is constant, so we only need to minimize the first two terms. That is, the functional defined by

$$\int_\Omega (Ku - g)^2 \, dx + \beta \sum_\gamma w_\gamma |\langle u, \psi_\gamma \rangle|^p .$$

Since this is exactly the functional we minimized using the iterative thresholding algorithm previously, we will apply this method here. Using the weight function defined in (5.3), we approximate the solution with the ITA.

5.3.2 Minimizing with respect to v

If we keep u fixed, the first term of $F_\varepsilon^{MOD}(u, v)$ is constant, so we only need to focus on the last two terms. To minimize F_ε^{MOD} with respect to v, we shall attempt a gradient descent algorithm.

Line search methods and gradient descent

Let us quickly introduce the gradient descent algorithm, which is a line search method. Given a function f that we wish to minimize with respect to some variable x, line search methods starts from a point x_k, choose a search direction p_k, decide how far along this direction to move, and updates the variable as $x_{k+1} = x_k + \lambda_k p_k$. The main difference between different line search methods is how to choose the search direction and step size to ensure convergence. We will in the following study the gradient descent and backtracking algorithms.

Gradient descent, also called steepest descent, is a line search method in which one searches for the minimum of a function f along the direction of its negative gradient. Hence the name *steepest* descent, as the negative gradient is the direction in which the function decreases most rapidly. Starting from a point x_k, we must be able to calculate the gradient $\nabla f(x_k) = \nabla f_k$, then choose $p_k = -\nabla f_k$. After choosing a step length λ_k in some appropriate manner, we update the value $x_{k+1} = x_k + \lambda_k p_k$.

To choose the step length λ_k, we face a tradeoff between accuracy and computation time. The ideal choice for λ_k would be the one which minimizes $f(x_k + \lambda_k p_k)$, however for most practical purposes, computing this minimizer would be too computationally expensive. Therefore, we search for a step size that is not too small and reduces f "sufficiently". To ensure that the process makes reasonable progress along the search direction, we can choose the step length λ_k using a so called backtracking approach, and make sure that the step length fulfils the sufficient decrease condition, as defined in Chapter 3 in [16],

$$f(x_k + \lambda_k p_k) \leq f(x_k) + c\lambda_k \nabla f_k^T p_k, \tag{5.4}$$

for some small parameter $c \in (0, 1)$.

Finding the descent direction

As we have already mentioned, the search direction of the gradient descent is the negative gradient of the function. We now assume u fixed, and define a new v dependent functional

$$\begin{aligned} H(v)(x) &= \beta \sum_{\gamma} w_\gamma \left|\langle u, \psi_\gamma \rangle\right|^p + \alpha \int_\Omega \varepsilon \left|\nabla v\right|^2 + \frac{(v-1)^2}{4\varepsilon} dx \\ &= \beta \sum_{\gamma} w_\gamma c_\gamma + \alpha \int_\Omega \varepsilon \left|\nabla v\right|^2 + \frac{(v-1)^2}{4\varepsilon} dx. \end{aligned} \tag{5.5}$$

Thus, we need to find the gradient of this function, so we calculate the functional derivative

$$\nabla H(v)(x) = \beta \sum_{\gamma} c_\gamma \nabla w_\gamma - \alpha\varepsilon\Delta v + \alpha \frac{v-1}{2\varepsilon}. \tag{5.6}$$

Let us study the derivative ∇w_γ of the weight function. To find an expression for this, we first define

$$e_x(y) = \begin{cases} 1 & \text{if } y = x \\ 0 & \text{else,} \end{cases}$$

then we formally have

$$\begin{aligned} \nabla w_\gamma &= \left.\frac{d}{dt}\right|_{t=0} w_\gamma (v + te_x) \\ &= \left.\frac{d}{dt}\right|_{t=0} \min\left\{(v(y) + te_x)^2 + \kappa : y \in \operatorname{supp}\psi_\gamma\right\} \cdot s_\gamma. \end{aligned}$$

Let us first assume that there is a unique minimizer x' of v in $\operatorname{supp}\psi_\gamma$. Then we have

$$\nabla w_\gamma = \left|_{t=0} 2\left(v(x') + te_x(x')\right) e_x(x') s_\gamma,\right.$$

so that

$$\nabla w_\gamma(v)(x) = \begin{cases} 2v(x) s_\gamma & \text{if } x' = x \\ 0 & \text{else.} \end{cases}$$

However, if a unique minimizer does not exist, the above argument does not work. It should be possible to compute directional derivatives which won't

be linear, but for simplicity we will use the formal gradient with a weighting factor $\mu \in (0,1)$. Thus, we express the gradient of the weight function as

$$\nabla w_\gamma(v)(x) = \begin{cases} 0 & \text{if } x \notin \operatorname{supp} \psi_\gamma, \\ 0 & \text{if } v(x) > w_\gamma(v)(x), \\ 2v(x) \cdot s_\gamma & \text{if } x \text{ is a unique minimizer of } v^2 \text{ in } \operatorname{supp} \psi_\gamma, \\ 2v(x)\mu \cdot s_\gamma & \text{if } x \text{ is a non-unique minimizer of } v^2 \text{ in } \operatorname{supp} \psi_\gamma. \end{cases} \tag{5.7}$$

Choosing the step length

The backtracking line search, see Chapter 3 in [16] for more details, searches along the direction p_k and iteratively decrease step size until the sufficient decrease condition is fulfilled. It does so by choosing an initial step size and iteratively decreasing this by a factor τ until the sufficient decrease condition (5.4) is fulfilled. In algorithm 2 we show the backtracking algorithm for our problem.

Algorithm 2 Backtracking

Choose $\lambda > 0,\ \tau \in (0,1),\ c \in (0,1)$
while $H(v_k - \lambda f_k) > H(v_k) - c\lambda \nabla f_k^T \nabla f_k$ **do**
    $\lambda \leftarrow \tau\lambda$
end while
Set $\lambda_k = \lambda$

Chapter 6

Implementations

For the implementation of the different numerical processes in this thesis, the programming language MATLAB [13] has been used. Mainly a language intended for numerical computing, MATLAB provides, for example, easy handling of multidimensional array operations and, in this case most importantly, the Image Processing Toolbox [11]. This toolbox contains efficient functions and algorithms for a wide range of different image processing processes, such as convolutions and visualizations. We will in this section study and present some elements of the implementations of the numerical methods we use to solve the minimization problems presented in the previous chapters.

6.1 The iterative thresholding algorithm

We start with the iterative thresholding algorithm. Being a simple iterative process, the main issues include the implementation of the blur kernel and how to impose this on u, how to do the wavelet decomposition and the implementation of the edge detector.

6.1.1 The blur kernel K

When implementing the ITA, we need to be able to define the operator K and apply it to u. In the Image Processing Toolbox, we find the functions `fspecial` [10] and `imfilter` [12]. The `fspecial` function creates a two dimensinal filter of a specific type, for example an averaging filter or a lowpass Gaussian filter, the latter of which we will use in the following. To create the Gaussian filter, we need to define the size and standard deviation and pass it to `fspecial`, which creates the Gaussian filter suitable to use with `imfilter`. The function `imfilter` takes the multidimensional array u and filters it with

the multidimensional array K, giving an output of the same size as u. In addition to the input signal and the filter, we can specify the boundary option and filtering type. Passing the flags '*conv*' and '*symmetric*', the function performs multidimensional filtering using convolusion, computing input array values outside the bounds of the array using reflection across the border.

6.1.2 Wavelet functions

We also need to calculate the wavelet coefficients of the image. MATLAB provides functions for multilevel 2-dimensional wavelet decomposition, reconstruction, and coefficient extractions. Using the function `wavedec2` we get the scaling and wavelet coefficients of an input signal with respect to a wavelet basis and to a level of our choice. We will use the Haar-wavelet and decompose the image as much as possible, that is to the maximum level. The deconstruction function produces two arrays, one containing the coefficients and the other is a bookkeeping matrix. Having these two matrices, the `detcoef2` function allows us to extract the detail coefficients for a specific orientation (horizontal, vertical, diagonal or all) at a certain scale. The decomposition counterpart is the function `waverec2`, which takes the coefficient array, the bookkeeping matrix and wavelet type as input, and reconstructs the signal.

6.1.3 The edge detector and assigning weights

To find the edges, we define a function `find_edges`. This takes the wavelet coefficients, the bookkeeping matrix, j_{fine} and j_{coarse}, and a threshold parameter as input. It returns a boolean array of the same size as the array with wavelet coefficients, indicating which wavelet coefficients should be weighted as edges. The function calculates the threshold value for each level and orientation, and checks which of the corresponding coefficients are larger. Then it checks if there, for a position x in the image, is a "path", through the scales $\{j_{\text{fine}}, ..., j_{\text{coarse}}\}$ and wavelets with x in their support, of coefficients larger than their threshold. If so, the positions corresponding to these coefficients are marked as *true* in the boolean output matrix. The implementation of the edge detector can be found in the Appendix.

Given the boolean output array of `find_edges`, we can create a function `assign_scaledep_weights` that takes this array as an input, together with two different values w_e and w_s that indicates weighting, and return a weight array to be used in the ITA. The function simply iterates through the scales and assigns weights appropriately, see the Appendix for the MATLAB code.

6.1.4 The algorithm

Having all these functions available in MATLAB makes the implementation straight forward, and what is left is to properly implement the function $S_{1,p}$, the Newton iterations and properly impose stopping criterions. We present the pseudocode for the ITA in algorithm 3. The Appendix contains a MATLAB code for the ITA and the Newton method.

Algorithm 3 ITA

Given g, K, u^0, β, w, p and maximum decomposition level.
$n = 0$
while not converged **do**
  $a^{n+1} = u^n - K^*(Ku^n - g)$
  if $p == 1$ **then**
    $u^{n+1} = S_{w,1}(a^{n+1})$ as defined in (4.6)
  else
    Approximate u^{n+1} with Newtons method as defined in algorithm 1
  end if
  Test convergence in accordance with equation (4.16)
  $n = n + 1$
end while
return u^n

6.2 The phase field approach of the modified functional

For the minimization of the modified MS functional, we will iteratively alternate between minimizing with respect to u and v. For the minimization with respect to u, we will use the ITA from the previous section, albeit with a different weighting function. Similarly as for the other functions, the implementation of the weight function defined in equation (5.3) can be found in the Appendix. For the minimization with respect to v, we will present a gradient descent algorithm. Note that we choose v to be half the size of u in both the x and y directions, so that one pixel in v corresponds to a square of four pixels in u, or rather, one coefficient of the finest decomposition scale of u.

6.2.1 Gradient descent for v

We recall the functional defined in equation (5.5)

$$H(v) = \beta \sum_{\gamma} w_\gamma c_\gamma + \alpha \int_\Omega \varepsilon \left|\nabla v\right|^2 + \frac{(v-1)^2}{4\varepsilon} dx.$$

and its gradient defined in (5.6)

$$\nabla H\left(v\right)\left(x\right) = \beta \sum_{\gamma} c_\gamma \nabla w_\gamma - \alpha\varepsilon\Delta v + \alpha\frac{v-1}{2\varepsilon},$$

which we need for the descent direction $p_v = -\nabla H(v)$. For the implementation of the gradient descent algorithm for v, we need to discretize the functions, so we first make some definitions. For a v of size $M \times N$, we let $v_{i,j}$ be the value of v at the position (i, j) and define the grid spacing as

$$h = 1/\max\{M, N\}.$$

For the derivatives we use forward finite differences

$$\delta_x^+ v_{i,j} = \frac{1}{h}\left(v_{i+1,j} - v_{i,j}\right)$$

and

$$\delta_y^+ v_{i,j} = \frac{1}{h}\left(v_{i,j+1} - v_{i,j}\right).$$

Using this approximations, we can find discretized function for both the $|\nabla v|^2$ in the expression for H and the Δv in the function for p_v. To calculate the function $H(v)$ we use a sum of the indices, and scale by h^2. However, to deal with the boundary conditions, we shall let the sum only go to $M-1$ and $N-1$. We use simple matrix multiplications to calculate the functional value and the descent direction p_v. The implementation can be seen in the MATLAB code `GD_v` in the Appendix.

6.2.2 Alternating minimization

Having the minimization approaches with respect to the two variables separately, we can implement the alternating approach. Given initial values, the method minimizes with respect to u and then v until some requirements are met. In the implementation of `modified_AT` included in the Appendix, we require the L^2 norm of the difference between the old and new u and the max norm of the difference between the old and new v to be smaller than given tolerances.

Chapter 7

Numerical Results

To run tests on the implementations, we will use a standard test image [18] of a monarch. The monarch image is of size 512×768, and will be decomposed into a 8 level Haar wavelet decomposition for the different procedures in this section. Figure 7.1a shows the greyscale version of the image, while figure 7.1b shows the same image blurred by a convolution with a 15×15 Gaussian blur kernel with variance $\sigma = 5$. This blurring is quite strong, so

(a) Original monarch image. **(b)** Blurred monarch image.

Figure 7.1: Test image.

the blurred image is very different from the original image, and the blurring operator is very ill-conditioned, but this makes it a good problem to test our implementations on. Thus, our goal is to find an approximation to figure 7.1a using the reconstruction algorithms on the input image in figure 7.1b.

7.1 The iterative thresholding algorithm

We will in this section study some results of reconstructing the monarch image using the ITA. We start with the constant (but level dependent) weighting approach, before studying the results when applying the edge dependent weighting of the wavelet coefficients.

7.1.1 Restoration with constant weights

Figure 7.2: Constant penalty reconstruction with $p = 1.1$.

We study the result u found using the iterative thresholding algorithm with scale dependent weights, that is, we set $w_\gamma = 2^{3pj-2j}$. We set $\beta = 1$, and $p = 1.1$ thus searching a solution with few, but large, wavelet coefficients. We note that the original image, seen in figure 7.1a consists of sections of near constant intensities, at least apart from the section of flowers on the left hand side of the image. That is, there is not much variation in intensity in the black regions of the monarchs wing, nor in the white spots. Thus, we could expect a small p value to yield a good solution for this image, or at least the monarch. Figure 7.2 shows the reconstructed image with $p = 1.1$. We note that the reconstructed image does in fact seem clearer than the blurred image in figure 7.1b, the change in intensities seem to be more rapid between sections of different values, details are in short easier to tell apart. We also note that the solution seems to be "blocky", meaning that for example the lines that should be straight, but diagonal, in the moths wing have jagged edges. The blockyness is also apparent in the white spots on the wings that should be round, see the section of the approximation in figure 7.4b.

The blockyness of the solution should however not come as a surprise, as we are trying to approximate the circles, and diagonal lines, using square shaped basis functions. For finer decomposition levels, there are more coefficients than for the coarser scales. So to achieve a sparse representation, it is not beneficial for the method to allow the fine decomposition wavelet coefficients to be large. Thus, the coarse scale coefficients are used to produce the change in intensity, and we get the blocky appearance. This is the main motivation behind the edge detection and edge dependent weighting we test in the following, that we can favour the fine scale wavelet coefficients on edges to improve on the appearance of the solution.

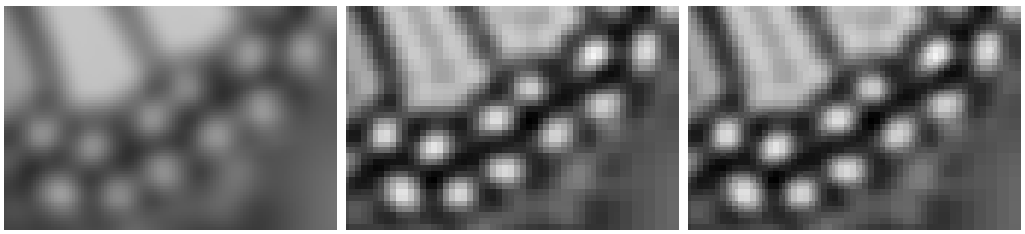
7.1.2 Restoration with edge-dependent weights

We now run the procedure with edge dependent weighting of the coefficients. After running the ITA once using constant weights, we use the edge detector on the solution u to attain the wavelet coefficients that constitute edges and assign edge dependent weights. Then we use these weights to run the iterative process again. For the tests in the following, we search for edges in the set of the 6 finest wavelet decomposition levels. We say that a wavelet coefficient is an edge if its absolute value is twice as large as the average of the absolute values of all coefficients of the same level and orientation (horizontal, vertical or diagonal, recall figure 3.3). We set the edge weighting as

$$w_\gamma = \begin{cases} 2^{3pj-2j} & \text{if } u_\gamma \text{ is part of an edge,} \\ 0.01 \cdot 2^{3pj-2j} & \text{else.} \end{cases}$$

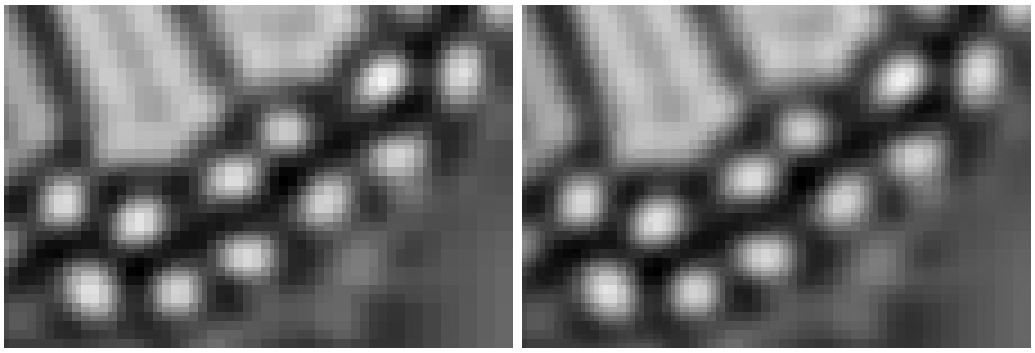
Running the procedure, we get the image shown in figure 7.3. At first glance, the difference between the images in figure 7.2 and figure 7.3 is not very apparent, so let us zoom in and have a closer look. Figure 7.4 show a section of the reconstructed image for both the constant and the weighted penalty methods. We notice that the weighted penalty increases the resolution of the edges. Moreover, this decreases the blocky appearance of the solution that we observed previously, which was our goal. We argued earlier that a value for p close to 1 should produce good results for the approximation. Let us see what happens if we increase the value to $p = 1.5$. We must zoom in on the picture for the differences to be visible, and in figure 7.5 we see sections of the reconstructions for $p = 1.1$ and $p = 1.5$. As the larger p promotes a smoother solution, we see that the edges are not as well defined for $p = 1.5$ as it is in the solution with $p = 1.1$.

Figure 7.3: Weighted penalty reconstruction with $p = 1.1$.



(a) Blurred image. **(b)** Constant penalty. **(c)** Weighted penalty.

Figure 7.4: Sections of the reconstructed monarch image.



(a) Weighted penalty with $p = 1.1$. **(b)** Weighted penalty with $p = 1.5$.

Figure 7.5: Sections of the reconstructed monarch image with different values for p.

7.1.3 Reconstructing a noisy image

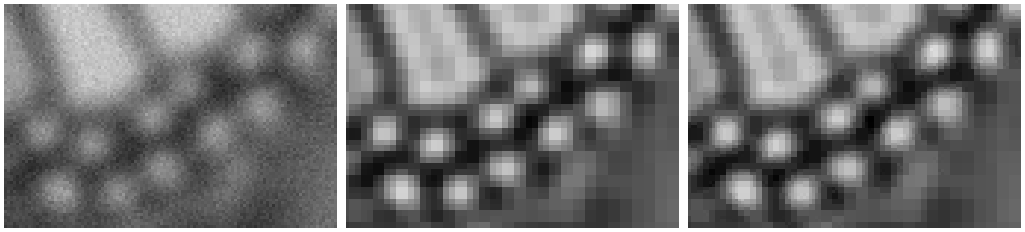
Let us now add some noise to the blurred image and study the reconstruction abilities of the ITA on such a noisy image. We add a Gaussian noise with mean 0 and a standard deviation $10^{-3}\,(\max(g) - \min(g))$ to the blurred image g, and the resulting image can be seen in figure 7.6. The blur and noise is more apparent in the section shown in figure 7.8a.

Figure 7.6: Blurred and noisy image.

Figure 7.7: Reconstruction of blurred and noisy image with scale dependent weights and $p = 1.1$.

In order to reduce the noise in the reconstruction, we require more smooth-

ness of the function by increasing the value of β. For the reconstruction shown in figure 7.7, we have used the values $p = 1.1$ and $\beta = 10$. We observe that the reconstructed image still appears to piecewise smooth. However, the noise has introduced some slow variations particularly visible in the sections of the approximation which should be near constant, see for example the background above and to the right of the monarch's wing. The finer details are however well preserved, as we observe in the section shown in figure 7.8b. We also notice that blocky wavelet artefacts in the solution are more prominent now, this is a result of the larger β which makes the weights of the wavelet coefficients larger.



(a) Noisy image. **(b)** Constant penalty. **(c)** Weighted penalty.

Figure 7.8: Sections of the reconstructions of the blurred and noisy image

A reconstruction with edge dependent weights are also computed. The result is not noticeably different from the scale dependent reconstruction in figure 7.7 before we zoom in and have a look at the details. Therefore, we have included a section of the reconstruction with edge dependent weights in figure 7.8c . We observe that, as before, this approach increases the resolution of the edges and therefore decreases the blocky appearance of the solution. As a result of the noise in the input image, the approximation is not as smooth as for the image which was only blurred.

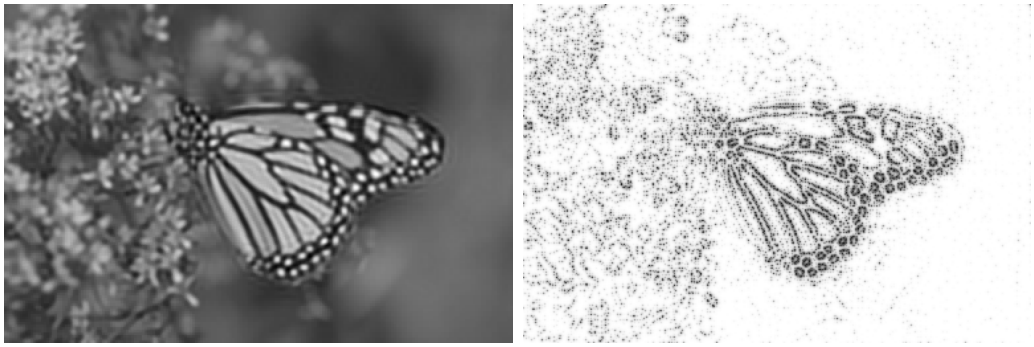
7.2 Modified Mumford–Shah

We now move to the modified functional. We first run the implementation with the parameters

$$\alpha = 7000, \quad \beta = 1, \quad p = 1.1, \quad \varepsilon = 10^{-3}$$

and the result can be seen in figure 7.9. The parameter choice for ε is due to the scaling of the image to area approximately equal to 1 in the numerical implementation. The ε represents the width of the edges, and with this scaling, 10^{-3} is approximately the pixel size. The α is chosen to

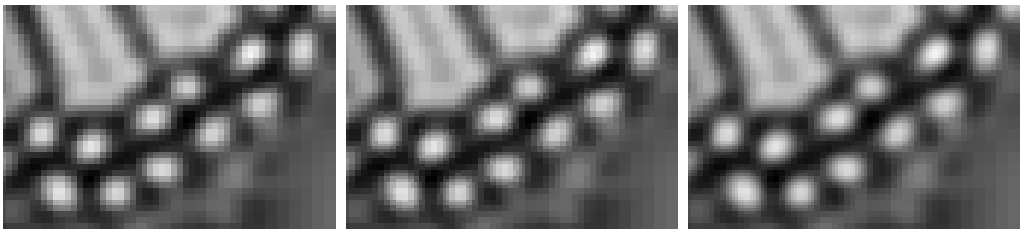
appropriately balance the wavelet term and phase field term in the modified functional. The reconstruction in figure 7.9a appear to be a mostly smooth



(a) Reconstructed image. **(b)** Phase field.

Figure 7.9: Deblurred image and phase field.

approximation of the original image. We see that the lines in the phase field in figure 7.9b clearly defines the edges separating areas of approximately the same grey scale in the reconstructed image. We can say that the phase field represents the segmentation of the image. Again, we zoom in to see details.



(a) Constant weight reconstruction. **(b)** Edge dep. weight reconstruction. **(c)** Phase field based reconstruction.

Figure 7.10: Section of reconstructed image with different weighting.

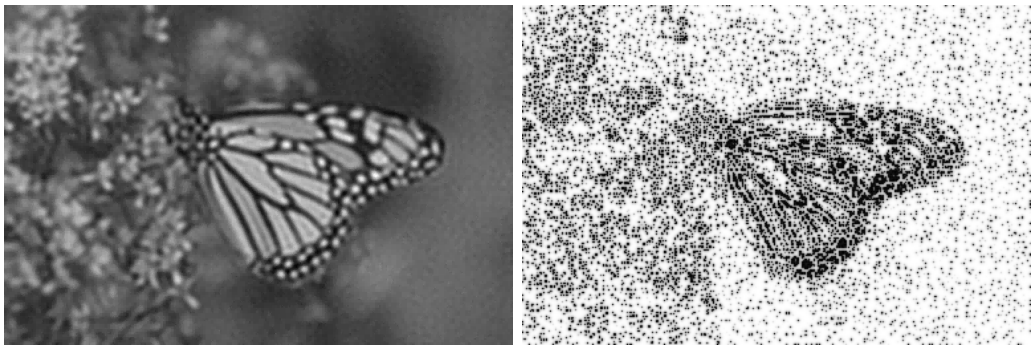
In figure 7.10, we see again the section of the reconstructed image with constant weight, with edge dependent weighting and with phase field based weighting. This time we observe that the increase in resolution of the edges for the phase field based reconstruction seem to be even more prominent than what we achieved with the edge detection based weighting. The blocking is also much less noticeable, again because we are increasing the resolution and not trying to approximate the circular shapes using only the coarse scale wavelets. We can conclude that we have been able to reduce some wavelet artefacts of the reconstruction by imposing the phase field based weighting of the solution.

7.2.1 Noisy image

Let us test the modified Ambrosio and Tortorelli approach on the blurred and noisy image. We use the parameters

$$\alpha = 7000, \quad \beta = 10, \quad p = 1.1, \quad \varepsilon = 10^{-3},$$

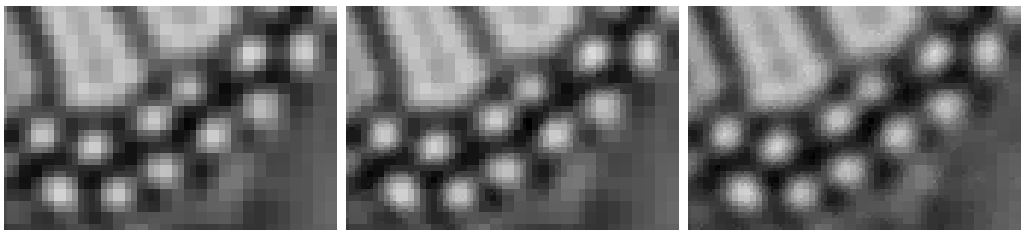
and run the alternating algorithm. Figure 7.11 shows the reconstructed im-



(a) Reconstruction. **(b)** Phase field.

Figure 7.11: Reconstruction and phase field of noisy and blurred image.

age and phase field. We observe that the phase field is 0 in some points where there shouldn't be any edges in the image. This is due to the noise in the input. Figure 7.12 shows a section of the reconstructed image both for the



(a) Constant weight reconstruction. **(b)** Edge dep. weight reconstruction. **(c)** Phase field based reconstruction.

Figure 7.12: Section of reconstructed image with different weighting.

constant weighted ITA approach and the phase field approach. As for the no-noise problem, the phase field approach removes most of the blocky appearance of the approximation, producing a solution with smoother edges.

Chapter 8

Conclusions

We have in this paper proposed a modified Mumford–Shah idea. This was done combining the segmentation idea and minimizing functional by Mumford and Shah with the regularization method of requiring sparse representation of the solution with respect to a orthonormal basis. Using the phase field approach by Ambrosio and Tortorelli and defining the weighting of the wavelet coefficient dependent on the phase field, we developed an alternating minimization process using the iterative thresholding algorithm for the minimization with respect to the wavelet coefficients, and a gradient descent method for the phase field. The resulting solution showed a substantial reduction of the blocking artefacts compared to the solution obtained with the ITA only.

For further work, the efficiency of the implementation could be of interest, also making it applicable to images of larger size than the (rather small) test image used in this thesis. Also, we used a gradient descent method for the minimization of the modified functional with respect to v, but alternative methods could be examined. Because of the improvement on wavelet artefacts and increase in resolution of the edges one could consider applications in other areas of imaging, for example computed tomography (CT).

Bibliography

[1] L. Ambrosio and V. Tortorelli. Approximation of functional depending on jumps by elliptic functional via γ-convergence. *Communications on Pure and Applied Mathematics*, 43(8):99–1036, 1990.

[2] K. Atkinson and W. Han. *Theoretical Numerical Analysis: A Functional Analysis Framework*. Texts in Applied Mathematics. Springer-Verlag, New York, 2005.

[3] J. M. Borwein and A. S. Lewis. *Convex Analysis and Nonlinear Optimization: Theory and Examples*. CMS Books in Mathematics. Springer-Verlag, New York, 2 edition, 2006.

[4] A. Chambolle. Inverse problems in image processing and image segmentation: some mathematical and numerical aspects. Technical report, Université de Paris-Dauphine, 2000.

[5] I. Daubechies, M. Defrise, and C. De Mol. An iterative thresholding algorithm for linear inverse problems with a sparsity constraint. *Communications on Pure and Applied Mathematics*, 57(11):1413–1457, 2004.

[6] I. Daubechies and G. Teschke. Variational image restoration by means of wavelets: Simultaneous decomposition, deblurring, and denoising. *Applied and Computational Harmonic Analysis*, 19(1):1 – 16, 2005.

[7] H. W. Engl, M. Hanke, and A. Neubauer. *Regularization of Inverse Problems*. Springer Science and Business Media, 1996.

[8] M. Grasmair, M. Haltmeier, and O. Scherzer. Sparse regularization with ℓ^q penalty term. *Inverse Problems*, 24(5):055020, 2008.

[9] Joint Photographic Experts Group. Jpeg 2000 overview. `https://jpeg.org/jpeg2000/index.html`. [Online; accessed 7-June-2016].

[10] The MathWorks Inc. fspecial. http://se.mathworks.com/help/images/ref/fspecial.html, 2016. [Online; accessed 31-May-2016].

[11] The MathWorks Inc. Image processing toolbox. http://se.mathworks.com/products/image/, 2016. [Online; accessed 27-May-2016].

[12] The MathWorks Inc. imfilter. http://se.mathworks.com/help/images/ref/imfilter.html, 2016. [Online; accessed 31-May-2016].

[13] The MathWorks Inc. Matlab. http://se.mathworks.com/products/matlab/, 2016. [Online; accessed 27-May-2016].

[14] S. Mallat. *A Wavelet Tour of Signal Processing*. Academic Press, Boston, third edition edition, 2009.

[15] D. Mumford and J. Shah. Optimal approximations by piecewise smooth functions and associated variational problems. *Communications on Pure and Applied Mathematics*, 42(5):577–685, 1989.

[16] J. Nocedal and S. J. Wright. *Numerical optimization*. Springer Series in Operations Research and Financial Engineering. Springer, Berlin, second edition, 2006.

[17] H. Triebel. *Bases in Function Spaces, Sampling, Discrepancy, Numerical integration*. EMS Tracts in Mathematics. European Mathematical Society Publishing House, Zürich, 2010.

[18] Wikipedia. Standard test image — wikipedia, the free encyclopedia. https://en.wikipedia.org/w/index.php?title=Standard_test_image&oldid=676864185, 2015. [Online; accessed 22-April-2016].

Appendix

ITA

```
1  function u = iterative_thresholding(g,u0,K,p,beta,weights,level,
       delta)
2  %ITERATIVE_THRESHOLDING
3  %   Input parameters:
4  %   g:              blurred/noisy input image
5  %   u0:             initial guess for solution
6  %   K:              blur kernel
7  %   p:              p = [1,2]
8  %   beta:           regularization parameter
9  %   weights:        penalty weights
10 %   level:          wavelet decomposition level
11 %   delta:          noise level. Default value of 0 if not given
12 %
13 %   Output parameters:
14 %   u:              reconstructed image
15
16     if p < 1 || p > 2
17         error('Error: p must be in the interval [1,2].')
18     end
19
20     if nargin == 7
21         delta = 0;
22     end
23
24     u = u0;
25     [u_wavelet, index] = wavedec2(u,level,'haar');
26
27     weights = weights*beta;
28
29     it = 0;
30     converged = false;
31     while ~converged
32         temp = imfilter(g-imfilter(u,K,'symmetric','conv'),K','
               symmetric','conv');
33         [temp_wavelet,~] = wavedec2(temp,level,'haar');
34         a = u_wavelet + temp_wavelet;
35         if p == 1
36             u_wavelet = S_w1(a,weights);
37             u_new = waverec2(u_wavelet,index,'haar');
38         else
```

```
39              u_wavelet = newton(a,weights,p);
40              u_new = waverec2(u_wavelet,index,'haar');
41          end
42
43          C = max(1./(1+p*(p-1)*((abs(u_wavelet)/max(size(u))).^(p
                -2))));
44          convergence_check = (C/(1-C))*norm(u_new-u,2);
45          C2 = (2/(p*(p-1)))*3^(2-p)*(max(abs(u_wavelet)))^(2-p);
46          convergence_val = sqrt(4*beta*C2 + ((delta^2)/beta)*C2);
47          if convergence_check< convergence_val
48              converged = true;
49          end
50          it = it+1;
51
52          u = u_new;
53          fprintf('Thresholding iteration nr. %d. Stopping value
                is %4e,current value is %4e.\n',it,convergence_val,
                convergence_check)
54      end
55  end
56
57  function u_new = S_w1(u, weights)
58      [M,N] = size(u);
59      u_new = zeros(M,N);
60      for i = 1:M
61          for j = 1:N
62              if u(i,j) >= weights(i,j)/2
63                  u_new(i,j) = u(i,j)-weights(i,j)/2;
64              elseif u(i,j) <= -weights(i,j)/2
65                  u_new(i,j) = u(i,j)+weights(i,j)/2;
66              else
67                  u_new(i,j) = 0;
68              end
69          end
70      end
71  end
72
73  function u_new = newton(a,weights,p)
74
75      %for matrix: loop through and find starting values
76      [M,N] = size(a);
77      u_new = zeros(M,N); %this is the solution matrix
78
79      nit = 5; %maximum number of iterations of newton method
80      tol =10^-3; %error tolerance for convergence check of
81                  %newton method
82
83
84      signum = sign(a); %the sign of the rhs values.
```

```
85      F = signum + (weights.*p./2).*signum.*abs(signum).^(p-1);
86
87      %loop through the elements, find soltion values
88      for i = 1:M
89          for j = 1:N
90              sign_val = signum(i,j);
91              if sign_val == 0 %f must also be 0
92                  u_new(i,j) = 0;
93              elseif F(i,j) == a(i,j)
94                  u_new(i,j) = sign_val;
95              else%find start point and run newton iterations
96                  b = a(i,j); %rhs for this element
97                  if abs(F(i,j)) > abs(b)
98                      un = sign_val*(abs(b)/(1+(weights(i,j)*p)/2)
                            )^(1/(p-1));
99                  else
100                     un = b/(1+(weights(i,j)*p)/2);
101                 end
102                 % Start of newton iterations
103                 it = 0;
104                 err = inf;
105                 while ((it < nit) && (err > tol))
106                     un = un - (un + ((p*weights(i,j))/2)*
                            sign_val*(abs(un)^(p-1))-b)/(1+((p*
                            weights(i,j))/2)*(p-1)*sign_val*un*(abs(
                            un)^(p-3)));
107                     err = abs(un + ((p*weights(i,j))/2)*sign_val
                            *(abs(un)^(p-1))-b);
108                     it = it+1;
109                 end
110                 u_new(i,j) = un;
111
112             end
113         end
114     end
115 end
```

Edge detector

```
function mask = find_edges(w_coeffs,w_index,j_coarse,...
                           j_fine, threshold_parameter)
%FIND_EDGES  function that finds and marks all wavelet
%            coefficients that corresponds to an edge in
%            the image.
%   input:  w_coeffs: array of wavelet coefficients
%           w_indices: corresponding blockkeeping matrix
%           j_coarse: coarsest level to check
%           j_fine: finest level
%           threshold_parameter: parameter to include
%           when establishing the threshold for wavelet
%           coefficients
%
%   output: mask: a boolean one dimensional array of the
%           same size as w_coeffs, indication whether the
%           corresponding wavelet coefficient is part of an
%           edge (=1) or not (=0);

    w_coeffs = abs(w_coeffs);
    %M is the level of wavelet decomp
    M = size(w_index,1)-2;
    mask = zeros(1,w_index(M+2,1)*w_index(M+2,2));

    for j = j_fine:j_coarse
        [cH, cV, cD] = detcoef2('all',w_coeffs,w_index,j);
        siz = w_index(M+2-j,1)*w_index(M+2-j,2);
        % Find threshold
        thresholdH = (threshold_parameter*sum(sum(cH)))/siz;
        thresholdV = (threshold_parameter*sum(sum(cV)))/siz;
        thresholdD = (threshold_parameter*sum(sum(cD)))/siz;
        % Check if the corresponding coefficients are larger
        % than the threshold
        mask(siz+1:2*siz) = (w_coeffs(siz+1:2*siz)>thresholdH);
        mask(2*siz+1:3*siz) = (w_coeffs(2*siz+1:3*siz)>
            thresholdV);
        mask(3*siz+1:4*siz) = (w_coeffs(3*siz+1:4*siz)>
            thresholdD);
    end

    for orientation = ['h','v','d']

        %first we go from the finest scale to the coarsest scale
        curr_mask = detcoef2(orientation,mask,w_index,j_fine);
        rows = 1:2:size(curr_mask,1)-1;
        cols = 1:2:size(curr_mask,2)-1;
```

```
44          downscaled = curr_mask(rows,cols)+curr_mask(rows+1,cols)
                +curr_mask(rows,cols+1)+curr_mask(rows+1,cols+1);
45          recursive_down(mask,j_fine+1,j_coarse,'h',downscaled,
                w_index);
46          recursive_down(mask,j_fine+1,j_coarse,'v',downscaled,
                w_index);
47          recursive_down(mask,j_fine+1,j_coarse,'d',downscaled,
                w_index);
48      end
49      for orientation = ['h','v','d']
50          %then, go from coarsest scale to finest scale
51          curr_mask = detcoef2(orientation,mask,w_index,j_coarse);
52          rows = 1:2:2*size(curr_mask,1)-1;
53          cols = 1:2:2*size(curr_mask,2)-1;
54          upscaled = zeros(size(curr_mask,1),size(curr_mask,2));
55          upscaled(rows,cols) = curr_mask;
56          upscaled(rows+1,cols) = curr_mask;
57          upscaled(rows,cols+1) = curr_mask;
58          upscaled(rows+1,cols+1) = curr_mask;
59          recursive_up(mask,j_coarse-1,j_fine,'h',upscaled,w_index
                );
60          recursive_up(mask,j_coarse-1,j_fine,'v',upscaled,w_index
                );
61          recursive_up(mask,j_coarse-1,j_fine,'d',upscaled,w_index
                );
62      end
63  end
64
65  function recursive_down(mask,j,j_coarse,orientation,downscaled,
        w_index)
66      M = size(w_index,1)-2;
67      if orientation == 'h'
68          o = 1;
69      elseif orientation == 'v'
70          o = 2;
71      else
72          o = 3;
73      end
74      curr_mask = detcoef2(orientation,mask,w_index,j);
75      curr_mask = ((downscaled.*curr_mask) >0);
76      siz = w_index(M+2-j,1)*w_index(M+2-j,2);
77      indices = (o)*siz+1:(o+1)*siz;
78      mask(indices) = curr_mask;
79          if j < j_coarse
80              rows = 1:2:size(curr_mask,1)-1;
81              cols = 1:2:size(curr_mask,2)-1;
82              downscaled = curr_mask(rows,cols)+curr_mask(rows+1,
                    cols)+curr_mask(rows,cols+1)+curr_mask(rows+1,
                    cols+1);
```

```
83              recursive_down(mask,j+1,j_coarse,'h',downscaled,
                    w_index);
84              recursive_down(mask,j+1,j_coarse,'v',downscaled,
                    w_index);
85              recursive_down(mask,j+1,j_coarse,'d',downscaled,
                    w_index);
86          end
87  end
88
89  function recursive_up(mask,j,j_fine,orientation,upscaled,w_index
        )
90      M = size(w_index,1)-2;
91      if orientation == 'h'
92          o = 1;
93      elseif orientation == 'v'
94          o = 2;
95      else
96          o = 3;
97      end
98              curr_mask = detcoef2(orientation,mask,w_index,j);
99              curr_mask = ((upscaled.*curr_mask) >0);
100             siz = w_index(M+2-j,1)*w_index(M+2-j,2);
101             indices = (o)*siz+1:(o+1)*siz;
102             mask(indices) = curr_mask;
103             if j > j_fine
104                 rows = 1:2:2*size(curr_mask,1)-1;
105                 cols = 1:2:2*size(curr_mask,2)-1;
106                 upscaled = zeros(size(curr_mask,1),size(
                        curr_mask,2));
107                 upscaled(rows,cols) = curr_mask;
108                 upscaled(rows+1,cols) = curr_mask;
109                 upscaled(rows,cols+1) = curr_mask;
110                 upscaled(rows+1,cols+1) = curr_mask;
111                 recursive_up(mask,j-1,j_fine,'h',upscaled,
                        w_index)
112                 recursive_up(mask,j-1,j_fine,'v',upscaled,
                        w_index)
113                 recursive_up(mask,j-1,j_fine,'d',upscaled,
                        w_index)
114             end
115
116 end
```

Assigning weights

```
function weights = assign_scaledep_weights(mask,mask_index,
    is_edge,is_not_edge,p)
    M = size(mask_index,1)-2;
    weights = is_not_edge*ones(size(mask));
    weights(mask==1) = is_edge;
    for j = 1:M
        scaling_factor = 2^(-3*p*j+2*j);
        siz = mask_index(M+2-j,1)*mask_index(M+2-j,2);
        indices = siz+1:4*siz;
        weights(indices) = scaling_factor*weights(indices);
    end
end
```

Minimization of the modified AT functional

```
function [u,v,u1] = modified_AT(g,K,p,alpha,beta,epsilon,it_max)
    max_level = wmaxlev(size(g),'haar')-1;
    [M,N] = size(g);
    %% initializations
    u0 = g;
    u = g;
    v_init = ones(M/2,N/2);
    v_new = v_init;
    [~,u_w_index] = wavedec2(u,max_level,'haar');
    %% iterate
    it_counter = 1;
    u_change = inf;
    v_change = inf;
    u_tol = 10;
    v_tol = 0.01;
    while (u_change > u_tol || v_change > v_tol) && it_counter
        <= it_max
        v = v_new;
        weights = find_AT_weights(v,u_w_index,max_level,p);%
            better choice for level
        u_old = u;
        u = iterative_thresholding(g,u0,K,p,beta,weights,
            max_level);
        if it_counter == 1
            u1 = u;
        end
        v_new = GD_v(v,u,max_level,v_tol,alpha,beta,epsilon,p);
        v_change =max(max(abs(v-v_new)))
        u_change = norm(u-u_old,2)
        it_counter = it_counter + 1
    end
    fprintf('Algorithm ended after %d iterations\n',it_counter);
end
```

Gradient descent

```
function v = GD_v(v,u,decomp_level,tol_v,alpha,beta,epsilon,p)

    [M,N]= size(v);
    if M > N
        h = 1/M;
    else
        h = 1/N;
    end

    Dx = -diag([ones(M-1,1);0]) + diag(ones(M-1,1),1);
    Dy = -diag([ones(N-1,1);0]) + diag(ones(N-1,1),-1);

    Lx = -2*diag([1/2;ones(M-2,1);1/2]) + diag(ones(M-1,1),1) +
       diag(ones(M-1,1),-1);
    Ly = -2*diag([1/2;ones(N-2,1);1/2]) + diag(ones(N-1,1),1) +
       diag(ones(N-1,1),-1);

    %This matrix is used in the computation of the functional,
       which only
    %sums up to N-1, M-1
    R = ones(M,N);
    R(:,N) = 0;
    R(M,:) = 0;
    [u_w_coeffs,u_w_index] = wavedec2(u,decomp_level,'haar');

    %trenger å finne funksjonalverdien
    change_v = inf;
    cg_it = 0;

    %decomp_level = decomp_level -2;
    while change_v > tol_v
        cg_it = cg_it + 1;
        FUNC_VAL = h^2*sum(sum(R.*(...
                (alpha*epsilon/(h^2))*((Dx*v).^2 ...
                + (v*Dy).^2)...
                + (alpha/(4*epsilon))*(v-1).^2 )))...
                + beta*find_AT_weights(v,u_w_index,decomp_level,
                   p)*(abs(u_w_coeffs).^p)'; %%ERR???

        pv = -(h^2*(-2*(alpha*epsilon/(h^2))*(Lx*v+v*Ly) + ...
            (alpha/(2*epsilon))*(v-1))+...
            beta*find_grad_J(v,u_w_coeffs,u_w_index,decomp_level
               ,p));
           lambda_v = 0.1; tau = 0.1; c = 0.01;
        %lambda_v = 1; tau = 0.1; c = 0.1;
```

```
41          t = c*sum(sum(pv.*pv));
42
43          v2 = v+lambda_v*pv;
44          FUNC_VAL_NEW = h^2*sum(sum(R.*(...
45                  (alpha*epsilon/(h^2))*((Dx*v2).^2 ...
46                  + (v2*Dy).^2)...
47                  + (alpha/(4*epsilon))*(v2-1).^2 )))...
48                  + beta*find_AT_weights(v2,u_w_index,decomp_level
                        ,p)*(abs(u_w_coeffs).^p)';
49
50          it = 0;
51          % backtracking until stepsize is small enough
52          while FUNC_VAL-FUNC_VAL_NEW < lambda_v*t && it < 10
53              lambda_v = lambda_v*tau;
54              v2 = v+lambda_v*pv;
55              FUNC_VAL_NEW = h^2*sum(sum(R.*(...
56                  (alpha*epsilon/(h^2))*((Dx*v2).^2 ...
57                  + (v2*Dy).^2)...
58                  + (alpha/(4*epsilon))*(v2-1).^2)))...
59                  + beta*find_AT_weights(v2,u_w_index,decomp_level
                        ,p)*(abs(u_w_coeffs).^p)';
60
61              it = it + 1;
62          end
63          v = v2;
64          change_v = max(max(lambda_v*abs(pv)));
65          fprintf('Gradient descent step nr. %d used %d
                backtracking steps. ||v-v_new|| = %f\n',cg_it,it,
                change_v);
66      end
67  end
68
69  function grad_J = find_grad_J(v,u_w_coeffs,u_w_index,j_coarse,p)
70  %UNTITLED2 Summary of this function goes here
71  %   Detailed explanation goes here
72      grad_J = zeros(size(v));
73      z = v.^2;
74      for j = 1:j_coarse
75          [cH, cV, cD] = detcoef2('all',u_w_coeffs,u_w_index,j);
76          u_coeffs = abs(cH).^p+abs(cV).^p+ abs(cD).^p;
77          supp_size = 2^(j-1); %size of supprt of wavelet in v
                scaling
78          scaling_factor = 2^(-3*p*j+2*j);
79          rows = u_w_index(end-j,1);
80          cols = u_w_index(end-j,2);
81          %% find matrix with minimum values, zj
82          if j == 1
83             grad_J = grad_J + scaling_factor*u_coeffs;
84          else
```

```
85
86              temp = zeros(rows,cols*supp_size);
87              for r = 1:rows %calculate the minimum in each row
88                  temp(r,:) = min(z(supp_size*(r-1)+1:supp_size*r
                        ,:));
89              end
90              zj = zeros(rows,cols);
91              for c = 1:cols
92                  zj(:,c) = min(temp(:,supp_size*(c-1)+1:supp_size
                        *c),[],2);
93              end
94              %% create a mask with 1 if z == the min, 0 else
95              maskj = (kron(zj,ones(supp_size)) == z);
96              %% TEST TO CHECK IF UNIQUE MINIMIZER
97              %temp = zeros(rows,cols*supp_size);
98              for r = 1:rows %calculate the minimum in each row
99                  temp(r,:) = sum(maskj(supp_size*(r-1)+1:
                        supp_size*r,:));
100             end
101             unique = zeros(rows,cols);
102             for c = 1:cols
103                 unique(:,c) = sum(temp(:,supp_size*(c-1)+1:
                        supp_size*c),2);
104             end
105             unique = (unique == 1);
106             maskj = 0.1*maskj + 0.9*kron(unique,ones(supp_size))
                    .*maskj;
107             %% update gradient
108             grad_J = grad_J + scaling_factor*(kron(u_coeffs,ones
                    (supp_size))).*maskj;
109         end
110     end
111     grad_J = 2*grad_J.*v;
112 end
```

Phase field weight function

```
function weights = find_AT_weights(v,u_w_index,j_coarse,p)
    kappa = 0.01;
    weights = zeros(1,u_w_index(end,1)*u_w_index(end,2));
    for j = 1:j_coarse
        rows = u_w_index(end-j,1);
        cols = u_w_index(end-j,2);
        supp_size = 2^(j-1); %size of the support of the
           wavelets in v scale
        scaling_factor = 2^(-3*p*j+2*j);
        siz = rows*cols;

        %% find matrix with minimum values,
        z = v.^2;
        if j == 1
            weights(siz+1:4*siz) =([z(:);z(:);z(:)]+kappa)*
               scaling_factor;
        else
            temp = zeros(rows,cols*supp_size);
            for r = 1:rows %calculate the minimum in each row
                temp(r,:) = min(z(supp_size*(r-1)+1:supp_size*r
                   ,:));
            end
            zj = zeros(rows,cols);

            for c = 1:cols
                zj(:,c) = min(temp(:,supp_size*(c-1)+1:supp_size
                   *c),[],2);
            end
            zj = zj(:);
            weights(siz+1:4*siz) = ([zj;zj;zj]+kappa)*
               scaling_factor;
        end
    end

end
```