Modular Algorithms for Large–Scale Total Variation Image Deblurring

Jesper Pedersen

LYNGBY 2005
EKSAMENSPROJEKT
NR. 06

IMM

Trykt af IMM, DTU

Preface

This master thesis has been carried out in the period 2nd of August 2004 to 18th of February 2005 at the department of Informatics and Mathematical Modelling (IMM) at the Technical University of Denmark. It is worth 30 ECTS points.

I wish to thank my supervisors Professor Per Christian Hansen and Associate Professor Hans Bruun Nielsen for many very useful and inspiring discussions, and for their priceless support.

I also wish to thank M.Sc., Ph.D student Toke Koldborg Jensen for interesting discussions, for proofreading and for always offering his help.

Furthermore, I would like to thank all my office colleagues always taking their time to discuss and for making me feel well at the office.

Finally, I would like to thank my family for always supporting me during hard times.

Kgs. Lyngby, February 18th, 2005

Jesper Pedersen

Abstract

We present the theory behind inverse problems and illustrate that regularization is needed to obtain useful solutions. The most frequently used solution techniques for solving ill–posed problems are based on the use of 2–norms, which are known not to be suitable when edges are desired in the regularized solution. The thesis covers the development of an algorithm for solving ill–posed problems, where edges and steep gradients are allowed. The key idea is to make use of Total Variation, thus the algorithm solves a Tikhonov based optimization problem with the Total Variation functional as penalty term. The algorithm is based on Newton's method with a line search procedure.

The Total Variation functional is the essential subject for allowing edges in the solutions. We explain the theory connected with Total Variation, and we discuss how important it is to use an appropriate discretization, and propose a way of doing this. Furthermore, we discuss how to deal with the non–linearity of the functional by introducing an extra set of variables. Introducing these variables we obtain an augmented system, which is globally "more" linear and thus easier to solve. The purpose of this is to obtain a faster convergence.

The implementation of the algorithm is carried out with focus on efficiency. It is implemented in the framework of the modular MATLAB toolbox M$\mathcal{OO}$RE TOOLS, which is designed for solving large–scale inverse problems. This results in a modular and object–oriented structure, which together with the interior use of iterative methods makes the implementation suitable for solving large–scale problems. All implementation aspects are explained in detail. Finally, some experiments using the algorithm are carried out.

Keywords *Total Variation, Tikhonov, ill–posed problems, regularization, steep gradients, MOORe Tools, large–scale inversion, iterative methods, object–oriented implementation, non–linear optimization, Newton's method, augmented system.*

Resumé

Teorien bag inverse problemer præsenteres, og det vises, at regularisering er nødvendig, hvis man vil opnå brugbare løsninger. De mest benyttede løsningsteknikker til at løse "ill–posed" problemer er baseret på brug af 2–normer, som er kendt for ikke at være særlig gode til at rekonstruere kanter i regulariserede løsninger. Denne afhandling dækker udviklingen af en algoritme til løsning af "ill–posed" problemer, hvor kanter og stejle gradienter tillades. Hovedidéen er at gøre brug af "Total Variation", således løser algoritmen et Tikhonov baseret optimeringsproblem med Total Variation funktionalen som strafled. Algoritmen er baseret på Newton's metode med liniesøgning.

Total Variation funktionalen er det essentielle emne, der gør, at kanter i løsningerne tillades. Teorien forbundet med Total Variation forklares, og det diskuteres, hvor vigtigt det er at benytte en passende diskretisering, og i afhandlingen gives der et bud på, hvordan dette kan gøres. Ydermere diskuteres, hvordan vi håndterer ikke–lineariteten af funktionalen ved at introducere ekstra variable. Ved at introducere disse variable opnås et større system, der er "mere" lineært globalt set og dermed lettere at løse. Således opnås en hurtigere konvergens.

Implementeringen af algoritmen er udført med fokus på effektivitet. Den er implementeret ved brug af M$\mathcal{OO}$Re Tools der er en modulær Matlab–pakke designet til at løse inverse problemer. Dette medfører en modulær og objekt–orienteret stuktur, der sammen med den indre brug af iterative metoder gør, at implementeringen kan benyttes til at løse storskala problemer. Alle aspekter forbundet med implementeringen er forklaret i detaljer. Sidst er algoritmen benyttet til at udføre eksperimenter.

Notation and Table of Symbols

The general notation has been chosen according to the following principle. All scalars are small, mostly Greek, characters. The situations where this is not the case, are easily detected from the context. Vectors are with small characters, e.g. x, and matrices are denoted with capital letters, e.g. A. If a vector and a matrix are denoted with the same symbol, represented as lower–case and upper–case respectively, the upper–case symbol means a diagonal matrix containing the elements of the vector. For example the matrix W is a diagonal matrix with elements $W_{ii} = w_i$ and the same holds for the vector ψ versus the diagonal matrix Ψ.

A_{ij} denotes the element in the ith row and jth column of A.

Whenever we use the symbol δ, it is connected with the following symbol, i.e., δw means a connected symbol and *not* a product between δ and w.

When bracket parenthesis are used $a[b+c]$ it means a product between a and $[b+c]$. Soft parenthesis, i.e., $a(b+c)$ means the function a evaluated in $b+c$. Sometimes the soft parenthesis are also used for products in cases where there are no risk of misreading.

MATLAB code and commands are represented with typewriter font, e.g., `D*X*Is'`.

The table beneath explains the symbols used in the thesis. The symbols have been placed in order of appearance.

Ω	Domain of integration
$K(s,t)$	Kernel in the Fredholm integral equation
$f(t)$	Unknown function in the Fredholm integral equation
$g(s)$	Right hand side function in the Fredholm integral equation
ϖ	Quadrature weights
A	The matrix that describes the properties of the system in $Ax = b$
x	The sought solution in the equation $Ax = b$
b	The right hand side in the equation $Ax = b$
$\otimes$	Kronecker product
vec / vec^{-1}	Functions for stacking and unstacking columns of a matrix
$A_{:j}$	The jth column in the matrix A
$A_{i:}$	The ith row in the matrix A
A_1, A_2	Matrices, e.g., in system $A_1XA_2^T = B$
B	Matrix representing a blurred image in the system $A_1XA_2^T = B$
$\sigma,\ \bar{\sigma}$	Standard deviation
$\mathcal{T}$	Toeplitz matrix
$\nabla(\cdot)$	Gradient operator
$I,\ I_n$	Identity matrix with dimensions $n \times n$
T	Objective function
$T'_x = \frac{\partial T}{\partial x}$	Partial differentiation of T with respect to x
$(Dx)_i$, $[Dx]_i$ or x_i	Reference to ith element in vector Dx or x
a_{ij}	Reference to element (i,j) in matrix A
$T(Dx)$	Function evaluation of T in Dx
$T[Dx]$	Product between T and Dx
$D^{(s)}$	Approximate first order derivative operator in the s–direction
$D^{(t)}$	Approximate first order derivative operator in the t–direction
$D^{(\mathrm{for})}$	Discrete forward derivative approximation
$D^{(\mathrm{cen})}$	Discrete central derivative approximation
λ	Regularization parameter in Tikhonov regularization method
$\frac{\mathrm{d}}{\mathrm{dt}}(\cdot)$	Derivative with respect to t
$\lvert\cdot\rvert$	Length, absolute value
$\lVert\cdot\rVert_1$	1–norm
$\lVert\cdot\rVert_p$	p–norm
$\lVert\cdot\rVert_2$	2–norm
$\mathrm{diag}(x)$	Diagonal matrix with diagonal elements x_i at location (i,i)
$\mathrm{cond}(A)$	Condition number of $A = \lVert A\rVert/\lVert A^{-1}\rVert$
x^*	Optimal value for x
$:=$	Assignment operator in pseudo code
$(\cdot)^{[k]}$	Value at the kth iteration
$\tau^{[k]}$	Length of step at iteration k
$\odot$	Hadamard product - elementwise multiplication

$\oslash$	Element wise division
$\hat{n}, \hat{m}$	Upper limit of sum indices - dependent on D
$\hat{I}(i,j)$	Indices relating the indices of a stacked vector and the original matrix indices
$H(f(t))$	The Hessian matrix, $H(f(t))_{ij} = \frac{\partial^2 f(t)}{\partial t_i \partial t_j}$
$A^\dagger$	The pseudo–inverse of A
$h,\ h^{(s)},\ h^{(t)}$	Grid spacing
$\Omega(f)/\Omega(x)$	Penalty function
e, E	Vector/matrix containing elements of white noise
$\tilde{e}, \tilde{E}$	Vector/matrix of all ones
$\tilde{e}_n$	Vector of zeros and a 1 in the nth element, e.g., $\tilde{e}_2 = [0, 1, 0, \ldots, 0]$
$\mathcal{J}_{\mathrm{TV}}(f)$	Continuous Total Variation functional, $\|\nabla f\|_1$
$J_{\mathrm{TV}}(x)$	Discrete Total Variation functional
x_{naive}	Naive solution, least–squares solution
x_{exact}	Exact solution
b_{noise}	Noisy b, $b_{noise} = b + e$
$\tilde{I}_m$	Modified “identity” matrix, where the last row has been removed
$\mathcal{D}^{(s)}$	Discrete derivative operator that operates on a stacked vector
$\tilde{D}^{(s)}$	Modified discrete derivative operator that ensures dimensions match
$\delta x,\ \delta w$	Steps in x– and w–direction respectively.
$\mathcal{G}$	Residual function for introduced extra variables
$\mathcal{N}(\cdot)$	Nullspace
$\mathbb{R}$	Set of real numbers
$\mathbb{R}^n$	Real n–space
$\mathcal{O}$	Order of magnitude
r	Residual function $r = Ax - b$
$\mathrm{sign}(\cdot)$	Sign function – returns the sign of the argument.
$\mathcal{A}$	Blurring matrix with Kronecker and Toeplitz structure
$\mathrm{span}(\cdot)$	The spanning of a space
ψ, 1D	$\psi_i = \sqrt{(Dx)_i^2 + \beta^2}$
ψ, 2D	$\psi_{\hat{I}} = \sqrt{(D^{(s)}x)_{\hat{I}}^2 + (D^{(t)}x)_{\hat{I}}^2 + \beta^2}$

Contents

List of Figures

Chapter 1

Introduction

The title of this project is composed of four terms: Modular algorithms, large–scale, total variation and image deblurring. Another term which is important for this work is inverse problems.

1.1 Explaining the Terms

We explain the four terms in reverse order.

The last term, image deblurring, is an example of an inverse problem. Inverse problems are the subject of the next chapter. In image deblurring one wants to obtain a clear (i.e. deblurred) image from a blurred and noisy image using a model for the blurring.

In some cases the true image contains edges. To reconstruct these edges is more or less impossible with standard solution techniques. We look at a subject, which makes it possible, namely Total Variation, which is the third term.

If the problem to solve is large, e.g., if the dimensions of the image are large, we are speaking of a large–scale problem, the second term. When this is the case it is important how implementation is carried out. This leads us to the final term, or rather the first term, namely modular algorithms.

The main focus of this work has been to develop and implement an algorithm, which is modular. The modularity means that the algorithm is also able to solve problems, which can not be represented with standard matrices and vectors. The specific problem may only have a modular object–oriented representation.

All in all the thesis covers the development of a modular algorithm using Total Variation, which can be used for solving large–scale problems. The algorithm is tested on image deblurring problems.

1.2 Thesis Outline

This thesis is constructed in the following way.

Chapter 2 gives an introduction to inverse problems, discrete ill–posed problems and regularization.

Chapter 3 focus on the definition and discretization of the Total Variation functional.

Chapter 4 describes some important subjects of *optimization*. We are dealing with a minimization problem, thus the chapter presents important definitions and theorems and it discusses different solution techniques.

Chapter 5 includes the theory behind the algorithm. It covers the derivation of the first and second derivative of the objective function and it deals with the derivation of more complicated systems involving extra variables and it deals with different solution approaches to the problem.

Chapter 6 deals with the implementation issues of the implemented algorithm in detail. This implementation builds on M$\mathcal{OO}$Re Tools, which is explained as well.

Chapter 7 contains test of the algorithm and some few experiments.

Chapter 8 concludes the thesis and makes suggestions for further work.

Finally, relevant Matlab code have been put in an appendix.

CHAPTER 2

Inverse Problems

2.1 The Generic Model

In many applications one has a model of a given system relating the *input* and the *output* of the system. Schematically this is visualized in Fig. 2.1, [11].

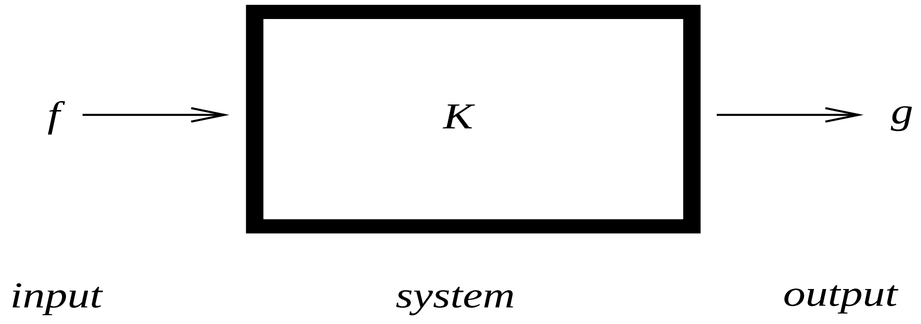


Figure 2.1: *The generic model.*

Here we look at linear inverse problems that can be formulated as integral equations in the form

$$\int_{\overline{\Omega}} \text{System} \times \text{Input } \mathrm{d}\overline{\Omega} = \text{Output} \ . \tag{2.1}$$

The form (2.1) is quite general. One problem is to determine the *output* given a mathematical model of the system and the *input* – this is often referred to as a *direct* or *forward* problem. The opposite, i.e., to determine the input of a system given exterior measurements (output) and a mathematical model relating the interior of the system and the measurements, is called an *indirect* or *inverse* problem.

2.2 The Fredholm Integral Equation

One special case of Eq. (2.1) is the Fredholm integral equation of the first kind, which has the following generic form

$$\int_0^1 K(s,t)f(t)dt = g(s)\ ,\ 0 \leq s \leq 1\ , \tag{2.2}$$

where $K(s,t)$ describes the system, $g(s)$ is a known function and $f(t)$ is the sought unknown function. Here the integration intervals are chosen, such that $0 \leq s \leq 1$ and $0 \leq t \leq 1$. In general this is not the case, but any problem can be transformed, such that s and t lie in these intervals.

Notice that $f(t)$ depends on one variable only, meaning that we are treating one dimensional problems. Later we will discuss how the equation looks in the two–dimensional case.

The difficulty in determining $f(t)$ is that the combination of the integration and the multiplication with $K(s,t)$ has a smoothing effect on $f(t)$, which means that $g(s)$ is smoother than $f(t)$. When integrating this means that information is inevitably lost, e.g., high frequencies or edges in $f(t)$ can not be reconstructed. This also means that the inverse problem to determine $f(t)$ given $K(s,t)$ and $g(s)$ is very hard. The reason is that even small errors in g can (and will) cause high frequencies in the calculated solution. We can not obtain the true solution to the problem. We are said to deal with an *ill–posed problem.*

Hadamard [13] was the first to define an ill–posed problem, as a problem where relative small perturbations of the output can lead to arbitrary large perturbations of the sought input.

Ill–posed problems arise in many practical applications such as geomagnetic inversion, tomography and image restoration. The latter we will explain in detail in Sec. 2.3.2.

2.2.1 Discretization of the Fredholm Integral Equation

To be able to solve the equation we need to discretize the equation. Using a quadrature method we obtain

$$\int_0^1 K(s,t)f(t)dt = \sum_{j=1}^{n} \varpi_j K(s,t_j)\tilde{f}(t_j) = g(s) \ ,$$

where $t_1, \dots, t_n$ are the abscissas for the particular quadrature rule, and ϖ_j the corresponding weights, see e.g. [7]. Note that f is substituted with $\tilde{f}$, since one can not expect to calculate the integral exactly.

By using collocation, i.e., demanding that the sum equals the right hand side in a series of points $s_1, \dots s_m$, we obtain

$$\sum_{j=1}^{n} \varpi_j K(s_i,t_j)\tilde{f}(t_j) = g(s_i) \qquad i = 1, \dots, m \ .$$

Note that m and n, in general, are different. This results in a linear system of equations on the form

$$\begin{pmatrix} \varpi_1 K(s_1,t_1) & \varpi_2 K(s_1,t_2) & \dots & \varpi_n K(s_1,t_n) \\ \varpi_1 K(s_2,t_1) & \varpi_2 K(s_2,t_2) & \dots & \varpi_n K(s_2,t_n) \\ \vdots & \vdots & \ddots & \vdots \\ \varpi_1 K(s_m,t_1) & \varpi_2 K(s_m,t_2) & \dots & \varpi_n K(s_m,t_n) \end{pmatrix} \begin{pmatrix} \tilde{f}(t_1) \\ \tilde{f}(t_2) \\ \vdots \\ \tilde{f}(t_n) \end{pmatrix} = \begin{pmatrix} g(s_1) \\ g(s_2) \\ \vdots \\ g(s_m) \end{pmatrix}$$

or just

$$Ax = b \ , \tag{2.3}$$

where $a_{ij} = \varpi_j K(s_i,t_j)$, $b_i = g(s_i)$ and $x_j = \tilde{f}(t_j)$, i.e., $A \in \mathbb{R}^{m\times n}$, $x \in \mathbb{R}^n$ and $b \in \mathbb{R}^m$.

We see that dealing with an integral equation like (2.2) is “just” a matter of solving a system of linear equations like (2.3). As we shall see in Sec. 2.4 there is more to it than just solving the system (2.3). But first we will turn the focus to how to deal with higher dimensional problems.

2.3 Dealing with Higher Dimensional Problems

When dealing with higher dimensional problems, the *Kronecker product* and the vec function become very applicable. The definitions follow.

Definition 2.3.1. *The Kronecker product* $\otimes$ *of two matrices* $\bar{A} \in \mathbb{R}^{m\times n}$ *and* $\breve{A} \in \mathbb{R}^{p\times r}$ *is defined in the following way*

$$\bar{A} \otimes \breve{A} = \begin{pmatrix} \bar{a}_{1,1}\breve{A} & \bar{a}_{1,2}\breve{A} & \dots & \bar{a}_{1,n}\breve{A} \\ \bar{a}_{2,1}\breve{A} & \bar{a}_{2,2}\breve{A} & \dots & \bar{a}_{2,n}\breve{A} \\ \vdots & \vdots & \ddots & \vdots \\ \bar{a}_{m,1}\breve{A} & \bar{a}_{m,2}\breve{A} & \dots & \bar{a}_{m,n}\breve{A} \end{pmatrix},$$

where $\bar{a}_{i,j}$ *denotes the elements in the matrix* $\bar{A}$. *The dimensions of* $\bar{A} \otimes \breve{A}$ *are* $mp \times nr$.

In general the Kronecker product $\bar{A} \otimes \breve{A}$ will be an very large matrix. We will later get back to aspects of how to store a Kronecker product in Matlab in an economic way.

Definition 2.3.2. *The function* vec *returns a stacking of the columns in the matrix* X

$$\hat{x} = \mathrm{vec}(X) = \begin{pmatrix} X_{:,1} \\ X_{:,2} \\ \vdots \\ X_{:,m} \end{pmatrix},$$

where $X_{:,i}$ *denotes the* ith *column of* X.

We will denote the stacked matrix X as $\hat{x}$. The inverse function of vec, vec^{-1}, is an “unstacking” defined in the following way

Definition 2.3.3. *The function* vec^{-1} *returns an unstacking of the vector* $\hat{x}$

$$X = \mathrm{vec}^{-1}(\hat{x}) = \begin{bmatrix} X_{:,1} & X_{:,2} & \dots & X_{:,m} \end{bmatrix},$$

where $X_{:,i}$ *again denotes the* ith *column of* X.

Note an important difference between vec and vec^{-1}, which is that vec^{-1} requires information on the dimensions of X. Thus the function is not unique unless this information is provided.

There exist some important relations between the Kronecker product and the function vec, see e.g. [19]. The relation

$$\mathrm{vec}(\bar{A}X\breve{A}^T) = [\breve{A} \otimes \bar{A}]\mathrm{vec}(X) \tag{2.4}$$

is useful in two–dimensional problems. It will be used in the following section.

2.3.1 The Fredholm Integral Equation in Two Dimensions

In two dimensions the Fredholm integral equation takes the following form

$$\int_0^1 \int_0^1 K(s,t,s',t')f(s,t)\, dsdt = g(s',t') \ . \tag{2.5}$$

When discretizing (2.5) by means of a quadrature method we end up with a system on the standard form

$$Ax = b \ , \tag{2.6}$$

where x and b are vectors. The dimensions of A match the dimensions of x and b respectively. This means that the row index of A will depend on the discretization of the parameters s' and t', and the column index of A will depend on the discretization of s and t.

The matrix A maps each point in the space spanned by (s,t) on to the space spanned by (s',t'). In practice the matrix is often dense, but there exist special cases, where it is sparse, has a special structure or a combination of both.

2.3.1.1 Separable Kernel

One of these cases is when the variables separate, then (2.5) can be written

$$\int_0^1 \int_0^1 \kappa_1(s,s')\kappa_2(t,t')f(s,t)\, dsdt = g(s',t') \ . \tag{2.7}$$

When discretizing (2.7) by means of a quadrature method we end up with a system of the form

$$A_1XA_2^T = B \ , \tag{2.8}$$

where $A_1 \in \mathbb{R}^{m\times n}$, $X \in \mathbb{R}^{n\times p}$, $A_2 \in \mathbb{R}^{r\times p}$ and $B \in \mathbb{R}^{m\times r}$.

We see that we can use (2.4) on (2.8) to obtain a linear system of equations

$$(A_2 \otimes A_1)\text{vec}(X) = \text{vec}(B) \tag{2.9}$$

$$\Leftrightarrow \qquad Ax = b \ , \tag{2.10}$$

where $A = (A_2 \otimes A_1)$, $x = \text{vec}(X)$ and $b = \text{vec}(B)$. From this we see that we end up with a "standard" linear system of equations.

One can use the theory of the Kronecker product and vec to understand that the theory behind two–dimensional problems is not different from the simpler one dimensional problem. However, Kronecker products can only be applied if the variables separate.

When dealing with large–scale problems one can not store the huge matrix $A_2 \otimes A_1$ explicitly. We will later get back to the use of a Matlab toolbox M$\mathcal{OO}$Re Tools, in which one actually can form the system using Kronecker products (which are stored implicitly) like in (2.9).

2.3.1.2 Convolution Form

Another special case is when the kernel only depends on the difference of the variables, i.e., when the Fredholm equation can be written as

$$\int_0^1 \int_0^1 K(s-s', t-t')f(s,t)dsdt = g(s',t') \ . \tag{2.11}$$

When the variables separate it is referred to as a *convolution.* In the one dimensional case this results in that A has *Toeplitz* structure. A Toeplitz matrix $\mathcal{T}$ has the form

$$\mathcal{T} = \begin{pmatrix} c_0 & c_{-1} & c_{-2} & \cdots & c_{-n+1} \\ c_1 & c_0 & c_{-1} & \ddots & \vdots \\ c_2 & c_1 & c_0 & \ddots & c_{-2} \\ \vdots & \ddots & \ddots & \ddots & c_{-1} \\ c_{n-1} & \cdots & c_2 & c_1 & c_0 \end{pmatrix} ,$$

i.e., $\mathcal{T}$ is a matrix with constant values along each diagonal. In general a Toeplitz matrix can be non–quadratic. The advantage of the Toeplitz structure is that the matrix can be represented by $2n-1$ coefficients only. This is a major advantage, when dealing with large–scale problems.

In the two–dimensional case the coefficient matrix has a structure called $BTTB$ (block Toeplitz with Toeplitz blocks).

When the problem is on convolution form (2.11) as well, we end up with a system on the form (2.6), where the coefficient matrix A has block Toeplitz structure. In this case the inverse problem of determining x is often referred to as *deconvolution*.

Note that the cases of the convolution form and the separable variables are not mutually exclusive. Both cases may occur in the same problem. When this happen, A can be written as a Kronecker product of Toeplitz matrices.

2.3.2 An Example – Image Deblurring

A two–dimensional example of an inverse problem is deblurring of digital images. In some cases one is given a blurred image, which one wants to deblur, i.e., obtain the underlying clear and unblurred image.

A gray–scale image can be represented as a matrix, where each element of the matrix represents a pixel in the image. Thus, the value of the element represents the light intensity of the corresponding pixel.

To be able to remove the blurring, we need a mathematical model for the blurring. This model must consider the type and strength of the blurring. In many practical applications one does not know the exact blurring and hence it must be estimated.

Atmospheric turbulence blur arises, e.g., in astronomical imaging. One way of modelling this blurring is to use a two–dimensional Gaussian point spread function [14], i.e.

$$\begin{aligned} K(s,t,s',t') &= \frac{1}{2\pi\sigma\bar{\sigma}}\exp\left(-\frac{1}{2}\left(\frac{s-s'}{\bar{\sigma}}\right)^2-\frac{1}{2}\left(\frac{t-t'}{\sigma}\right)^2\right) \\ &= \eta_{\bar{\sigma}}(s-s')\eta_{\sigma}(t-t')\ , \end{aligned} \tag{2.12}$$

where

$$\eta_\sigma(z) = \frac{1}{\sqrt{2\pi}\sigma} \exp\left(-\frac{1}{2}\left(\frac{z}{\sigma}\right)^2\right) . \tag{2.13}$$

Eq. (2.13) can be discretized to a Toeplitz matrix $\mathcal{T}$ with elements

$$(\mathcal{T}_\sigma)_{ij} = \frac{1}{\sqrt{2\pi}\sigma} \exp\left(-\frac{1}{2}\left(\frac{i-j}{\sigma}\right)^2\right) . \tag{2.14}$$

The atmospheric turbulence point spread function uses the Gaussian distribution (2.13) which goes towards zero for $z \to \pm\infty$. Thus we can represent the Toeplitz matrix with the υ largest elements. In this way we get a banded matrix, with bandwidth $2\upsilon - 1$, thus we can save memory and computation time in the deblurring process.

We see from (2.12) that the variables separate as described in Sec. 2.3.1.1. This means that the blurring in the two space directions are independent, and that the atmospheric turbulence point spread function $\mathcal{A}$ can be modelled with a Kronecker product

$$\mathcal{A} = \mathcal{T}_\sigma \otimes \mathcal{T}_{\bar{\sigma}} . \tag{2.15}$$

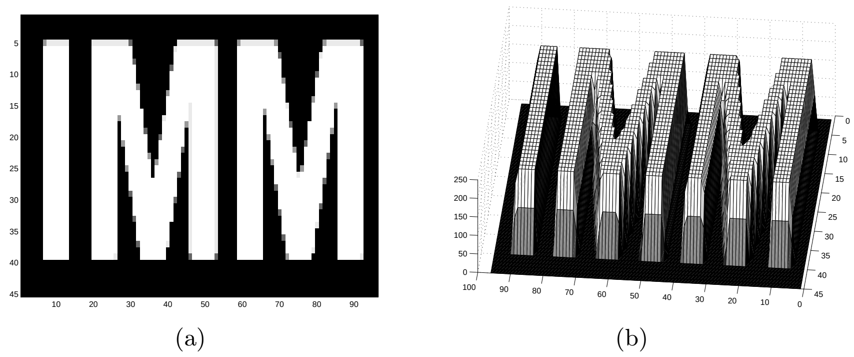
This is an example, where both the Kronecker products and Toeplitz structure arise. The atmospheric turbulence point spread function can be applied to blur an image X, directly

$$B = \mathcal{T}_{\bar{\sigma}} X \mathcal{T}_\sigma^T ,$$

where B denotes the blurred image.

Computing $\mathcal{A}$ by means of (2.14) and (2.15) with $\sigma = \bar{\sigma} = 2$ and apply it to a test image, we get a blurred image. An example of this is illustrated in Fig. 2.2, where we have blurred an image containing the text IMM.

In Chap. 7 we experiment and try to deblur, i.e., reconstruct true and sharp images from blurred ones.



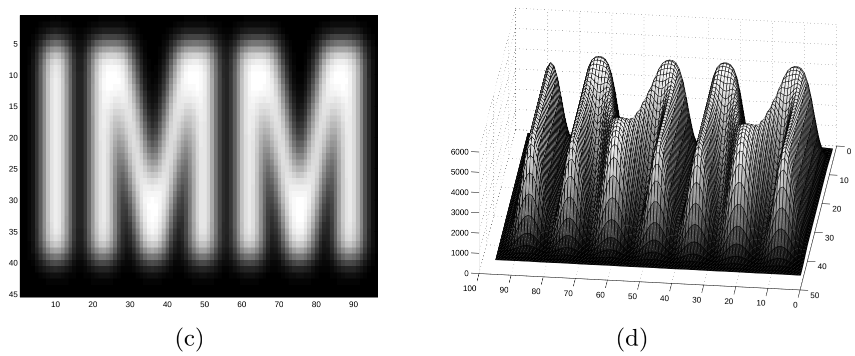


Figure 2.2: *(a) and (b): The test image with dimensions* 45×96*. (c) and (d): The blurred test image.*

2.4 Numerical Issues Related to Inverse Problems

In real–life applications the right hand side b of the linear system of equations (2.3) is often an outcome of measurements. It is well known that it is impossible to measure exactly, i.e., the measurements are contaminated with errors. Thus we can write $b = b_{\text{exact}} + e$, where e denotes the errors.

In many cases the matrix A is only an approximation to the underlying continuous system, thus we can interpret A as being a sum of the *exact* A and some noise, i.e., $A = A_{exact} + E$. In this work again we assume that A_{exact} is known. However if this is not the case, one can easily show that these errors can be mapped onto the right hand side b.

By now this means that we have two types of errors, namely the errors caused by accurate measurements and the discretization errors. In this work we assume that the errors arising in b (measured with a certain degree of accuracy) dominate the discretization error.

Hence it is trivial to show, see e.g. [3], that the upper bound on the relative error on the computed solution $\bar{x}$ is given by:

$$\frac{\|x_{\text{exact}} - \bar{x}\|}{\|x_{\text{exact}}\|} \leq \text{cond}(A) \frac{\|e\|}{\|b_{\text{exact}}\|} \tag{2.16}$$

where cond(A) denotes the condition number of A. The combination of even relative small errors in the right hand side and the fact that the condition number of A is large, makes it impossible to retrieve a usable solution with standard solution techniques, such as Gaussian elimination.

Since A is an discretization of the kernel in a Fredholm integral equation the condition number of the matrix will often be large in practical applications. This we can illustrate with a small experiment. We use the test problem "Wing" [12], with $n = 128$. The coefficient matrix A is ill–conditioned $\big(\text{cond}(A) = 3.4 \cdot 10^{20}\big)$, x is a known function and the output b is calculated by means of the forward problem $b = Ax$. To the right hand side b is added some white noise, such that $\|e\|/\|b_{\text{exact}}\| = 10^{-6}$ and we try to calculate the naive least–squares solution $x_{\text{naive}} = A^{\dagger}b$, where $A^{\dagger}$ is the Moore–Penrose matrix inverse.

The naive solution is very different from the exact solution. The reason for this is that it is completely dominated by inverted noise. The naive solution can be written

$$\begin{aligned} x_{\text{naive}} &= A^{\dagger} b \\ &= A^{\dagger} b_{\text{exact}} + A^{\dagger} e \\ &= x_{\text{exact}} + x_{\text{inv,noise}} \ , \end{aligned}$$

thus the high frequencies must be a consequence of inverted noise. This is a result of A having a smoothing effect on the solution x_{exact}. As explained in Sec. 2.2 even relatively small errors in b can results in large perturbations of the calculated solution. The exact solution and the right hand side b are visualized in Fig. 2.3.

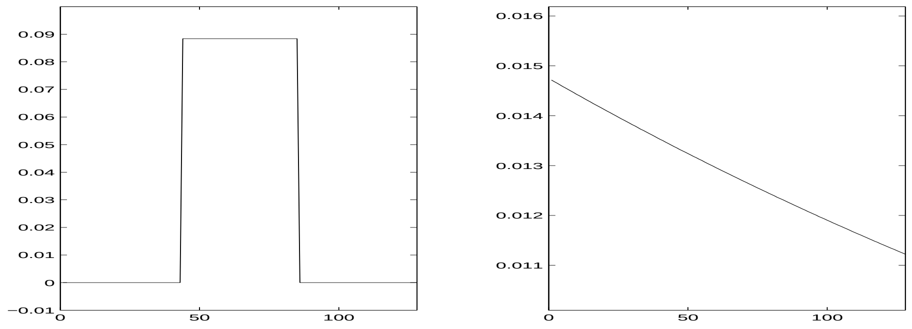


Figure 2.3: *The exact solution to the left and the right hand side b to the right. Note how smooth b is, and that one can not see the noise.*

The naive (least–squares) solution is visualized in Fig. 2.4. We see that the naive solution is completely unusable because it is dominated by inverted noise.

Algorithms dealing with how to suppress the influence from the errors on the solution are called *regularization* methods. This is the subject of the following section.

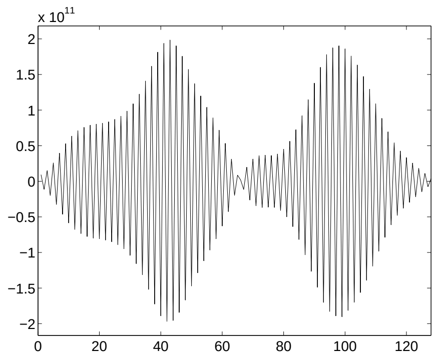


Figure 2.4: *The naive solution. We see that the naive solution is completely unusable because it is dominated by inverted noise. Notice the magnitude on the axes.*

2.5 Regularization

As stated, information is inevitably lost in the forward problem. If we want to retrieve a stable solution, we have to add extra information that compensates for the lost one. This information is called *a priori knowledge*, a knowledge which is known beforehand. The choice of which type of information is imposed on the solution has a high impact on the computed solutions.

The first thing we notice is that the incorporated a priori knowledge should ensure that the influence of the errors on the calculated solution is suppressed. One often used way of suppressing the errors is simply by setting up a new problem, which is called the Tikhonov regularization method.

2.5.1 Tikhonov Regularization

One of the most widely used methods is the Tikhonov regularization method [13]. It is defined in the following way

$$\min_x T(x) \ , \tag{2.17}$$

where

$$T(x) = \tfrac{1}{2}\|Ax - b\|_2^2 + \lambda\Omega(x) \tag{2.18}$$

is the objective function. The function $\Omega(x)$ is a given penalty function and λ is the regularization parameter. The reason for the constant factor $1/2$ in front of the residual term will become apparent in Chap. 5.

The idea is to find a stable solution by minimizing the sum of some norm of the residual $r = Ax - b$ and some measure of the solution x. The naive solution $x_{\text{naive}} = A^{\dagger}b$, i.e, a solution computed by means of e.g. Gaussian elimination, does not have a small solution norm. The reason to this is that in an ill–posed problem the naive solution will be very high–frequent as shown in Fig. 2.4. The reason for the high frequencies is that the inverted noise dominates, and the high frequencies results in that the norm of the computed solution will be large.

Tikhonov is more a formulation than an actual method. Tikhonov's method states what we expect of a stable solution of an inverse problem, but it does not say anything about how to actually solve the problem. Eq. (2.17) is an optimization problem, which can be solved using a variety of routines.

One way of finding the solution is to determine the SVD or GSVD (see e.g. [13]) and expand the solution in terms of the singular or generalized singular vectors. However, if the problem is large one can not compute an SVD, because it is too expensive and memory consuming. Dealing with large–scale problems one is forced to use iterative methods.

Which methods we will use in this project will come apparent in Chap. 4.

2.5.1.1 The Regularization Parameter

The regularization parameter λ is a trade–off parameter compromising fitting the data and ensuring that the penalty function is observed. Note that we will *not* in this work deal with how to choose the regularization parameter. Instead we assume that the optimal regularization parameter is known, which of course is not the case in real–life applications. Finding a regularization parameter close to the optimal one is in general a very difficult problem.

From the definition of Tikhonov regularization we can explain the meaning of the parameter λ. For $\lambda \to 0$ the computed solution approaches the naive least–squares solution x_{naive}. For $\lambda \to \infty$ the data are not fitted at all, and the weight is put on the penalty term only. The obtained solution depends on how $\Omega(x)$ is chosen, but in general this solution for $\lambda \to \infty$ will be just as unusable as the naive solution.

2.5.1.2 The Penalty Term

There are several ways to choose the penalty function, $\Omega(x)$, see e.g. [5]. It is often referred to as a *discrete smoothing norm*, which means that high frequencies in the computed solution x are penalized – only low frequencies may "go through" unregularized. In the standard form of Tikhonov $\Omega(x) = \|Dx\|_2^2$, with $D = I$. The matrix D can also be chosen as a weighting matrix or as a discrete derivative operator. In the latter case $\|D \cdot\|_2^2$ is called a semi–norm, since D has a nullspace such that $\|Dz\|_2$ can be zero, for a non–zero z.

Here we will show how various Tikhonov solutions look like for different choices of $\Omega(x)$.

2.5.1.3 Choosing $\boldsymbol{\Omega}(\mathbf{x}) = \|\mathbf{x}\|_2^2$

We still consider the test problem described in Sec. 2.4. In the left side of Fig. 2.5 it is illustrated that the standard choice $\Omega(x) = \|x\|_2^2$ yields smooth solutions. As explained in the previous section too large values of λ will result in unusable solutions. In this case the solution approaches the zero solution for $\lambda \to \infty$. Too small values of λ will result in high frequent solutions. Thus λ must be chosen as a compromise. The optimal solution, i.e., the solution computed by means of the optimal choice of λ, is visualized to the right in Fig. 2.5.

In test problems we compute the optimal regularization parameter, λ^*, by means of the optimization problem, i.e.,

$$\lambda^* = \underset{\lambda}{\operatorname{argmin}} \|x_\lambda - x_{\text{exact}}\|_2 \ , \tag{2.19}$$

where x_λ denotes the solution to the Tikhonov formulation for at given λ.

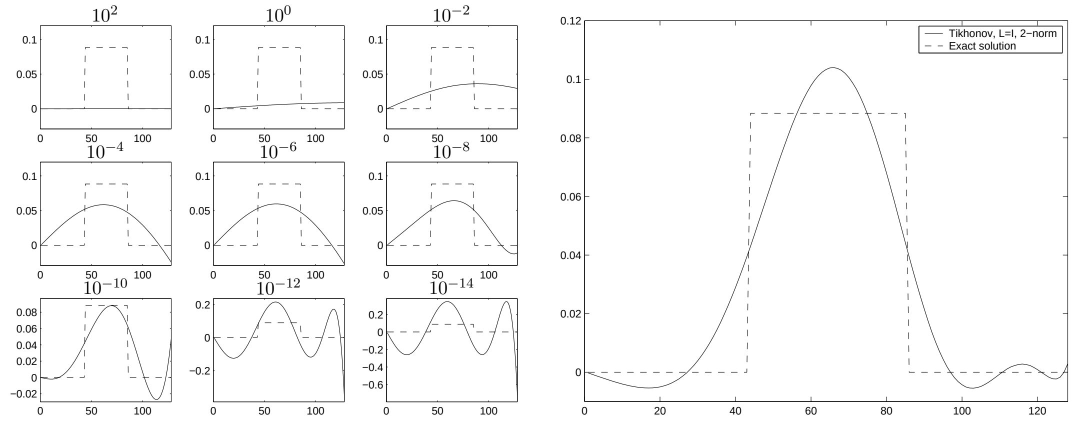


Figure 2.5: *To the left: Nine solution for different values of the regularization parameter. The values of λ are 10^2, 10^0,10^{-2},...,10^{-14}. The Tikhonov solutions are plotted with solid lines and the exact solution is dashed. Notice the different axis on the plots. To the right: The optimal Tikhonov solution (for $\lambda = 1.78 \cdot 10^{-11}$) and the exact solution.*

We notice that using regularization we are actually able to reconstruct solutions which are much better than the naive solution. However, we see that the solutions are very smooth and that the method has problems reconstructing the sharp edges in the solution.

2.5.1.4 Choosing $\mathbf{\Omega(x) = \|Dx\|_2^2}$

In some cases one can have a priori knowledge about the solution. As an example the knowledge could be that the solution has a small first order derivative. This is the case in the given test problem, see Fig. 2.6. We see that the gradient is zero everywhere except for two points, i.e., the gradient contains points, which are often referred to as *wild points* or *outliers.*

The information of the function having a small first order derivative can be imposed on the solution by choosing $\Omega(x) = \|Dx\|_2^2$, where D is a first order derivative operator. How to mathematically choose this operator, we will discuss in detail in the next chapter. The solutions obtained using the $\Omega(x) = \|Dx\|_2^2$ can be seen in Fig. 2.7.

We experience that the 2–norm is not appropriate when the given function has jumps.

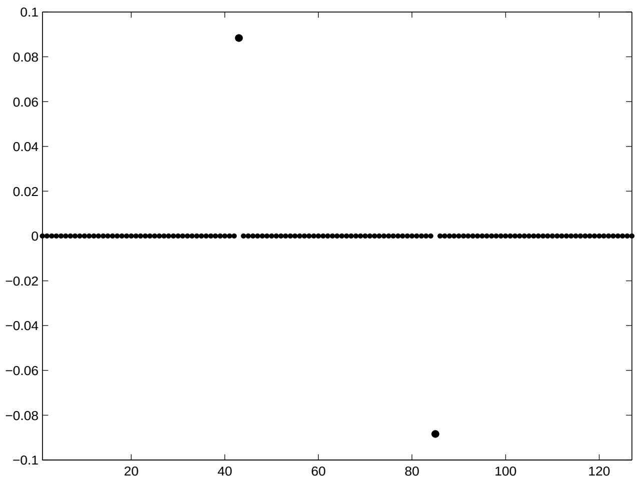


Figure 2.6: *The derivative of the solution to the wing test problem. We see that two outliers occur.*

Actually using the first derivative does not change the solution very much. We still obtain too smooth solutions. The reason to this is that the derivative includes outliers.

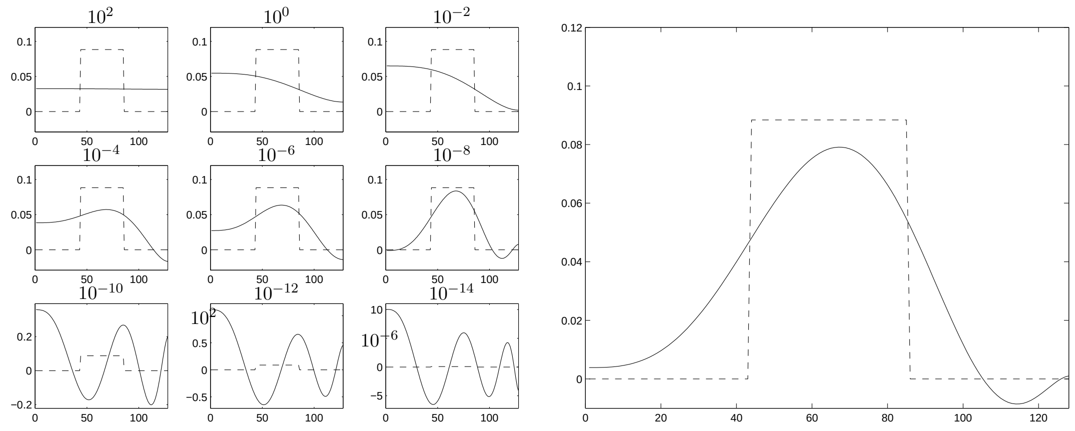


Figure 2.7: *To the left: Nine solution for different values of the regularization parameter. The values of λ are 10^2, 10^0,10^{-2},...,10^{-14}. The Tikhonov solutions are plotted with solid lines and the exact solution is dashed. Notice the different axes on the subplots. To the right: The optimal solution computed with $\lambda = 7.3 \cdot 10^{-9}$.*

It is well known from the theory of data fitting that the 2–norm is inappropriate, when the data to fit, includes these outliers, see e.g. [17]. The derivative of the best calculated solution can be seen in Fig. 2.8. Compared to Fig. 2.6 we see that the use of the 2–norm results in a too smooth solution.

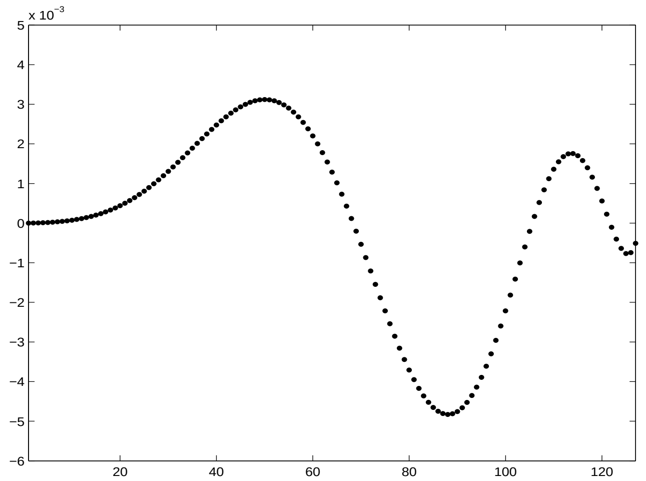


Figure 2.8: *The derivative of the solution calculated by means of optimal choice of λ.*

Instead one should consider using a 1–norm, which is known to avoid these difficulties.

Chapter Summary

In this chapter we have introduced ill–posed problems and shown how to discretize them. We have discussed how to deal with higher dimensional problems and discussed the special cases where the kernel is separable and on the convolution form. An example of a problem is given, and the numerical difficulties behind the problem are discussed. The Tikhonov regularization method is described with different penalty terms – all including a 2–norm, which is shown to result in too smooth solutions. In the following chapter we will introduce a more sophisticated penalty term called *Total Variation*, which includes the use of a 1–norm.

Chapter 3

Total Variation

3.1 Definition

Total Variation (TV) for a given continuous and differentiable function f of one variable t is defined in the following way

$$\mathcal{J}_{\mathrm{TV}}(f) \equiv \int \left|\frac{\mathrm{d}f}{\mathrm{dt}}\right| \mathrm{dt} = \left\|\frac{\mathrm{d}f}{\mathrm{dt}}\right\|_1 \ .$$

The one dimensional expression is a special case of the n–dimensional case, which is defined

Definition 3.1.1. *For a continuous and differentiable function $f(t) : \mathbb{R}^n \to \mathbb{R}$, Total Variation is defined as*

$$\mathcal{J}_{\mathrm{TV}}(f) \equiv \int_{\bar{\Omega}} |\nabla f(t)| \, \mathrm{d}t \ , \tag{3.1}$$

where $\overline{\Omega}$ is the domain and

$$|\nabla f(t)| = \left(\left(\frac{\partial f}{\partial t_1}\right)^2 + \ldots + \left(\frac{\partial f}{\partial t_n}\right)^2\right)^{(1/2)} \ . \tag{3.2}$$

The quantity (3.2) is in the image processing literature often referred to as *gradient magnitude*. Notice that the $|\cdot|$ is short notation for the 2–norm $\|\cdot\|_2$. This is what makes the TV functional a non–linear expression, e.g., in two dimensions the square root of a sum of squared quantities appears.

From the definition given in Def. 3.1.1 it is seen that TV is an integration of the length of all the gradients in any point in the domain $\bar{\Omega}$. As the gradient in a point is a measure of the variation of the function in this point, an integration over the entire domain must result in the total variation – hence the name.

One should notice that TV is non–linear, i.e., there exists no linear operator that, applied to a function f, returns the TV-measure. In a later chapter we will show that this non–linearity is the root of many problems and as well show how to deal with these problems.

3.2 Tikhonov and Total Variation

Since we want to use the Total Variation functional as the penalty term in Tikhonov's method, we need to discretize $\mathcal{J}_{\mathrm{TV}}$. This is done in the next section – here we will just denote J_{TV} as the discretized Total Variation functional.

The reason why we set the connection to Tikhonov before we show how to discretize J_{TV} is the following. In order to choose a good way of discretizing J_{TV} we have to know the context in which it is to be used. This will become apparent in the next section.

Combining Tikhonov's method with the discretized J_{TV} as the penalty term, i.e., $\Omega(x) = J_{\mathrm{TV}}(x)$, we get in the discrete case

$$\min_x T(x) \ , \tag{3.3}$$

where

$$T(x) = \tfrac{1}{2}\|Ax - b\|_2^2 + \lambda J_{\mathrm{TV}}(x) \ . \tag{3.4}$$

For a fixed λ this is a non–linear optimization problem in x. We discuss how to solve this in Chap. 4.

3.3 Why Total Variation?

We have seen in the previous chapter that using the 2–norm leads to smooth solutions, and here we study the impact of switching to the 1–norm.

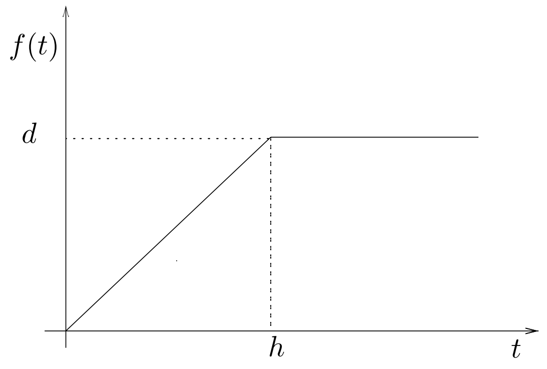


Figure 3.1: *A piecewise linear function illustrating the choice of the 1–norm.*

Inspecting the piecewise linear function illustrated in Fig. 3.1, we can calculate the p–norm (raised to the power p) of the gradient of $f(t)$ as follows

$$\begin{aligned}\|f'(t)\|_p^p &= \int_{-\infty}^{\infty} |f'(t)|^p dt \\ &= \int_0^h \left(\frac{d}{h}\right)^p dt \\ &= d^p h^{1-p} \ ,\end{aligned}$$

and hence

$$\|f'(t)\|_p = dh^{\frac{1-p}{p}} \ .$$

I.e., for $p = 1$ and $p = 2$ we have

$$\|f'(t)\|_1 = d$$

and

$$\|f'(t)\|_2 = d/\sqrt{h} \ .$$

This means that if $p = 1$, we see that the width of the interval h has no influence on $\|f'(t)\|_1$. Inspecting $\|f'(t)\|_2$ we see that the smaller h is, the larger the quantity

$\|f'(t)\|_2$ becomes. The conclusion of this little example is that, when using the 1–norm the variation is penalized. Using the 2–norm the variation is penalized as well, but rapid variations are penalized more than a steady variation. This shows that the 2–norm penalizes steep gradients.

From this we can see as well that the semi–norm $\|f'(t)\|_2$ has a unique minimizer, namely for $h \to \infty$. On the other hand $\|f'(t)\|_1$ is minimized by an infinite number of functions, because h can be chosen arbitrarily. Actually any monotone, possibly discontinuous functions that connects the origin and the point (h, d) will minimize $\|f'(t)\|_1$. From this we can conclude that the use of TV allows a wider class of functions, which can be a great advantage.

To illustrate which solutions we can obtain using TV instead of the two different penalty terms used in Sec. 2.5.1.2, we calculate solutions using the objective function (3.4). In Chap. 5 and Chap. 6 we describe the implementation of an algorithm that calculates these solutions – here we will just show that we are able to obtain much better solutions. In Fig. 3.2 solutions for different values of λ are visualized. Notice that the computed solutions are piecewise linear. We see that using a 1–norm instead of the 2–norm is much better in this case.

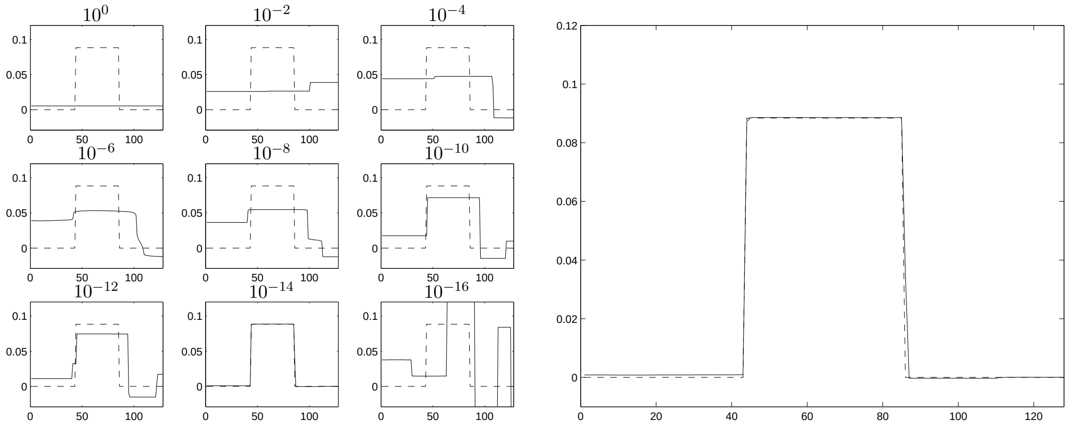


Figure 3.2: *To the left: Nine solution for different values of the regularization parameter. The values of λ are 10^0,10^{-2},...,10^{-16}. The Tikhonov solutions are plotted with solid lines and the exact solution is dashed. To the right: The optimal Tikhonov solution (for $\lambda = 1.95 \cdot 10^{-14}$) and the exact solution.*

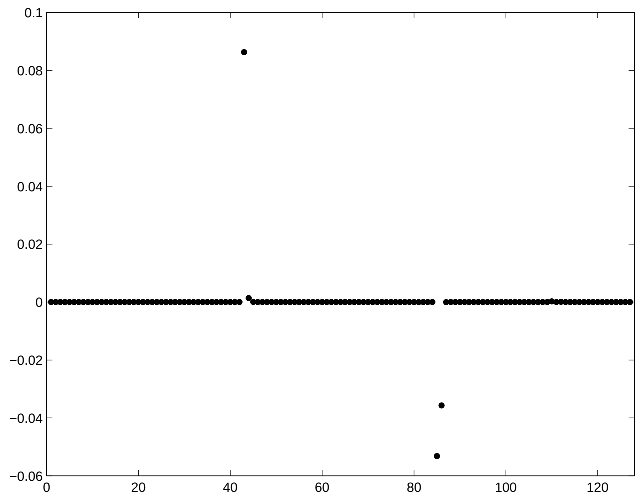


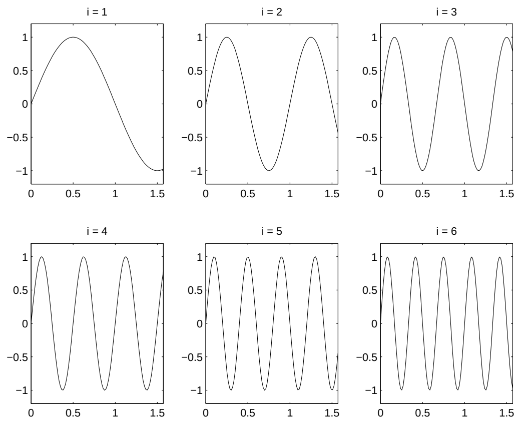
Figure 3.3: *The derivative of the solution computed by means optimal choice of λ.*

Furthermore we see from the derivative of the best calculated solution using the 1–norm in Fig. 3.3 that it has a more "spiky" nature, i.e., this penalty function allows jumps in the function. Notice also that with TV regularization we are able to produce solutions with localized steep gradients. Furthermore we do not need any a prior knowledge of the positions of these. Compared to Fig. 2.6 and Fig. 2.8 we see that this approach is very good.

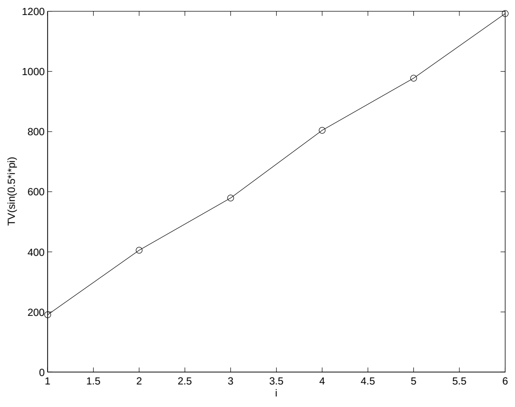
3.4 Examples of Various Functions and Their TV

To get a further understanding of TV, we illustrate the definition by two small examples. The first example illustrates how TV varies when the frequency of a function varies. See Fig. 3.4.

It is no surprise that TV is a measure of the total variation, hence the more a function varies the larger is the TV measure. Thus, a constant function will have a TV measure equal to zero. Another illustrative example shows how totally different functions can have the same TV measure. In Fig. 3.5 we have illustrated four functions with the same TV, showing that smooth and non–smooth functions can have the same TV, i.e., that this measure does not differ between these functions.



(a) $f_i(t) = \sin(i\pi t)$, $0 \leq t \leq \frac{\pi}{2}$



(b) $J_{\mathrm{TV}}(f_i(t))$

Figure 3.4: *(a) The six subfigures illustrate $f_i(t) = \sin(i\pi t)$, $0 \leq t \leq \frac{\pi}{2}$ for $i = 1, \ldots, 6$. (b) Illustration of how $J_{\mathrm{TV}}(f_i(t))$ varies for the 6 functions.*

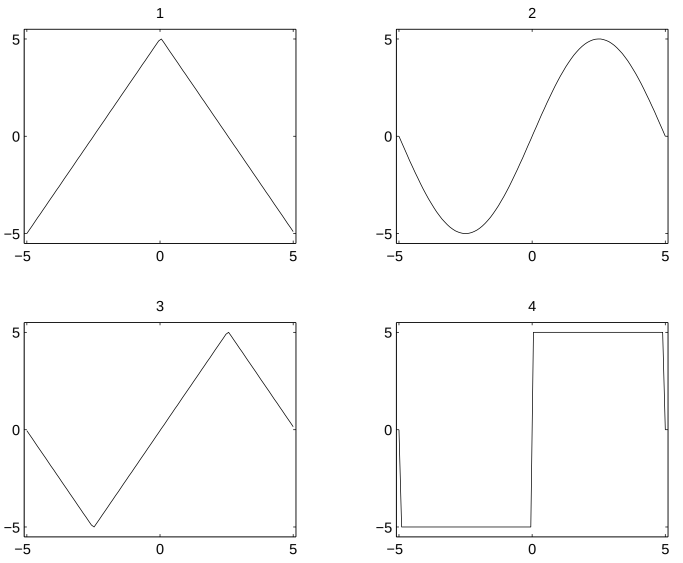


Figure 3.5: *Four functions with the same TV measure.*

This means that in the Tikhonov setting we can experience that the fitting term $\|Ax-b\|_2^2$ "prefers" a solution with steep gradients. If the penalty term is $\Omega(x) = \|x\|_2^2$ or $\Omega(x) = \|Dx\|_2^2$, we see that the steep gradients are not allowed by this penalty term, thus yielding too smooth solutions as seen in Fig. 2.5 and Fig. 2.7. This experiment shows that if the fitting term prefers a discontinuous solution, the discontinuities are not penalized using TV as the penalty term.

3.5 Discretization of the TV–Functional

In order to study total variation numerically, we use discretization. However there are several ways to do this. First we will describe an important definition, namely the null–space of a matrix.

Definition 3.5.1. *The nullspace $\mathcal{N}(\cdot)$ of a matrix $A \in \mathbb{R}^{m\times n}$ is defined as*

$$\mathcal{N}(A) = \{x \in \mathbb{R}^n \,|\, Ax = 0\}$$

i.e., $\mathcal{N}(A)$ is the span of all vectors x that fulfills $Ax = 0$.

In general, it is very important that one chooses the discrete approximation to the derivative operator appropriately. A necessary condition for the operator being well–suited for regularization is that the nullspace of this operator should be low–frequent. If the nullspace contains high–frequent components, one can not be sure to get a stable regularized solution. In the following sections we present several methods and study which regularizing effect the various discretizations have.

3.5.1 The One Dimensional Case

In the one dimensional case we will look at three different ways of discretizing the TV functional. The three discretizations are of first, second and fourth order, respectively.

3.5.1.1 Forward Derivative Approximation, First Order

One way is to use a forward approximation, which is a first order approximation. This means that the derivative of a function f in a given point t_k is calculated by means of the slope of the linear function that interpolates the points $\big(t_k, f(t_k)\big)$ and $\big(t_{k+1}, f(t_{k+1})\big)$. It is common to define the derivative to exist at the midpoint between the two points, i.e., at the point $\frac{1}{2}\big(t_{k+1}+t_k\big)$. See Fig. 3.6. When moving the forward approximation to the midpoint, in principle we make it a central estimate between the coordinates t_k and t_{k+1}. However, we will still refer to this as a forward approximation.

The one dimensional continuous formulation of the Total Variation functional can be approximated using a discrete forward approximation

$$J_{TV}(x) \approx \sum_{i=1}^{n-1} \frac{|x_{i+1}-x_i|}{h_i} = \frac{1}{h}\sum_{i=1}^{n-1} |x_{i+1}-x_i| \ , \tag{3.5}$$

where the last equality assumes that the sampling points are equidistant, i.e., $h_i = h$ for $\forall i$. We can rewrite Eq. (3.5) using matrix–vector notation

$$\frac{1}{h}\sum_{i=1}^{n-1} |x_{i+1}-x_i| = \|D^{(\text{for})}x\|_1 \ , \tag{3.6}$$

where

$$D^{(\text{for})} = \frac{1}{h}\begin{bmatrix} -1 & 1 & & & \\ & -1 & 1 & & \\ & & \ddots & \ddots & \\ & & & -1 & 1 \end{bmatrix} \tag{3.7}$$

is an $(n-1)\times n$–matrix and h is the grid spacing.

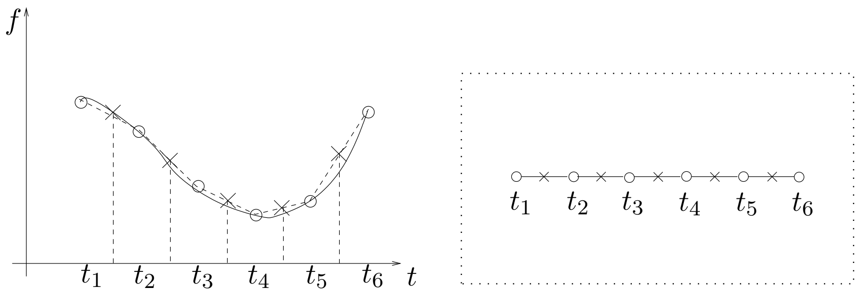


Figure 3.6: *The forward difference. The circles illustrate the abscissas of f and the crosses the abscissas of the forward approximation. Notice that the derivative of f lies between the points, where f is defined.*

In the Tikhonov setting the constant factor $1/h$ can be skipped and absorbed into the regularization parameter.

The nullspace of $D^{(\text{for})}$ is spanned by a constant vector, i.e., $\mathcal{N}(D^{(\text{for})}) = c_1 \cdot e$, where $c_1 \in \mathbb{R}$ and $e \in \mathbb{R}^n$ is a vector of all ones.

A consequence of using a forward approximation is that the derivative is defined in $n-1$ points only, i.e., the dimensions of the grids will not be the same. This problem we have to deal with. Essentially this is not very important in the one dimensional case. However, it will become apparent that it is very important in the two–dimensional case.

3.5.1.2 Central Derivative Approximation, Second Order

Another way of choosing the discrete derivative operator is to use a second order and central estimate. This means that we use the three points $\big(t_{k-1}, f(t_{k-1})\big)$, $\big(t_k, f(t_k)\big)$ and $\big(t_{k+1}, f(t_{k+1})\big)$ and find the interpolating second order polynomial through these points. The polynomial is differentiated and the value is calculated in the point t_k. The advantage is that the derivative is associated with the same grid as the original function, see Fig. 3.7. This approach also has some disadvantages. The first disadvantage is that the gradient is not defined in either of the endpoints, i.e., it is defined in $n-2$ points only. The second disadvantage is that this approach can not be used to recover really sharp edges, i.e., steep gradients, because three points on the abscissa are included. However, the last problem is reduced for a finer discretization.

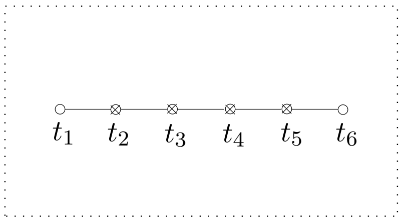


Figure 3.7: *The second order central difference. Notice that the derivative is associated with the same grid as the original function and that the derivative is not defined in the two endpoints.*

Substituting the forward approximation with a central approximation in (3.6), we get

$$J_{TV}(x) \approx \|D^{(\text{cen})}x\|_1 \ , \tag{3.8}$$

where

$$D^{(\text{cen})} = \frac{1}{h}\begin{bmatrix} -1 & 0 & 1 & & & & \\ & -1 & 0 & 1 & & & \\ & & & \ddots & \ddots & \ddots & \\ & & & & -1 & 0 & 1 \end{bmatrix} \tag{3.9}$$

is a $(n-2) \times n$ matrix.

Examining Fig. 3.8, which shows two vectors that span the nullspace of $D^{(\text{cen})}$, we see that there are some problems using this way of discretizing.

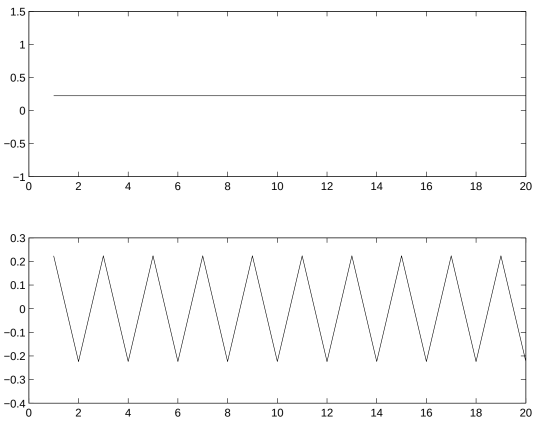


Figure 3.8: *The two vectors spanning the nullspace of $D^{(cen)}$.*

These vectors are a constant vector and a high frequent vector respectively. In terms of regularization this means that the central approximation can not be used, because we do not allow high frequencies in the regularized solution. In other words, the discrete derivative operator does not have the smoothing property. The reason for this high frequent nullspace can be explained with that $D^{(\text{cen})}$ only correlates every second point in the grid. This means that the points x_1, x_3, x_5, ... are correlated and x_2, x_4, x_6, ... are correlated. Since there exist no bond between two neighboring grid points x_i and x_{i+1}, this allows two sets of constant values, namely one value for the odd numbered grid points and one value for the even numbered. This extra degree of freedom is a clear drawback. An idea is to use approximations of higher order. In this work 3rd and 4th order approximations have been examined, and neither of these are suitable. The reason is that the higher order approximation one examines, the higher becomes the dimension of the given operator's nullspace. This means that for orders higher than one, we have to modify the operator to obtain a suitable nullspace. Doing the modification we have to ensure that the operator still is a meaningful gradient approximation.

Which modifications to use we will discuss in the two–dimensional case.

3.5.2 The Two–Dimensional Case

Moving to two dimensions we will represent the two–dimensional solution with a capital X, a quantity which we will denote a *two–dimensional vector*. However, it is represented and can be understood as being a matrix. When referring to subparts of a two–dimensional vector, we use the well known terminology from matrices. Examples are rows and columns of a two–dimensional vector, have the same meaning as if we were referring to a matrix. The two space directions we will denote s, corresponding to columns, and t, corresponding to rows. These directions are sketched in Fig. 3.12.

A discretization of (3.1) will take the form

$$J_{\mathrm{TV}}(X) \approx \sum_{i}^{\hat{m}} \sum_{j}^{\hat{n}} \sqrt{U_{ij}^2 + V_{ij}^2}\ , \tag{3.10}$$

where U_{ij} is the derivative of X in the s direction and V_{ij} the derivative in the t direction. Formally we can write

$$U_{ij} = (D^{(s)}X)_{ij} \quad \text{and} \quad V_{ij} = (XD^{(t)^T})_{ij}\ , \tag{3.11}$$

where $D^{(s)}$ and $D^{(t)}$ are discrete derivative operators in the two directions. The constants $\hat{m}$ and $\hat{n}$ depends on how one chooses the operators $D^{(s)}$ and $D^{(t)}$.

Eq. (3.11) can be written in terms of matrices and Kronecker products in the following way

$$U_{ij} = \left[\mathrm{vec}^{-1}\left(\left[I_m \otimes D^{(s)}\right]\mathrm{vec}(X)\right)\right]_{ij} \tag{3.12}$$

and

$$V_{ij} = \left[\mathrm{vec}^{-1}\left(\left[D^{(t)} \otimes I_n\right]\mathrm{vec}(X)\right)\right]_{ij}\ . \tag{3.13}$$

It is convenient to introduce the following notation for the Kronecker products $\mathcal{D}^{(s)} = D^{(s)} \otimes I_m$ and $\mathcal{D}^{(t)} = I_n \otimes D^{(t)}$. These are operators that operate on the stacked quantity $\operatorname{vec}(X)$.

Choosing $D^{(s)} = D^{(\text{for})}$ given in (3.7), i.e., as a forward approximation in the s–direction, the elements of U can be written as

$$U_{ij} = \frac{X_{i+1,j} - X_{i,j}}{h^{(s)}} \ , \text{ for } i = 1, \dots, n-1 \ , \ j = 1, \dots, m \ , \tag{3.14}$$

and in a similar way choosing $D^{(t)} = D^{(\text{for})}$, the elements of V are given by

$$V_{ij} = \frac{X_{i,j+1} - X_{i,j}}{h^{(t)}} \ , \text{ for } i = 1, \dots, n \ , \ j = 1, \dots, m-1 \ . \tag{3.15}$$

Using the forward approximation we can not add the two matrices U and V in Eq. (3.10) because their dimensions are incompatible. The dimensions are $(m-1) \times n$ and $m \times (n-1)$, respectively. Basically this means that the gradient is defined in the grid spanned by the first $(m-1) \times (n-1)$ points only, on the boundary of the domain only one of the components of the gradient exists. In [15] and [20] different ways to deal with this problem are described.

Eq. (3.10) can be evaluated if and only if the dimensions of U and V are the same. How we take care of this is very important, because there are several ways of doing it – and *not* all methods are equally good. Fundamentally, one can distinguish between two cases, namely removing or adding data. The scheme of adding splits up in different cases based on the type of data, which is added. Basically, we will cover the following three cases, where U and V are changed by

1. Removing data
2. Adding zero data
3. Adding non–zero data .

In the following sections these three cases are studied in detail.

3.5.2.1 Removing data from U and V

The first idea is to remove a column from U and a row from V. We choose to remove the last of both as illustrated in Fig. 3.9.

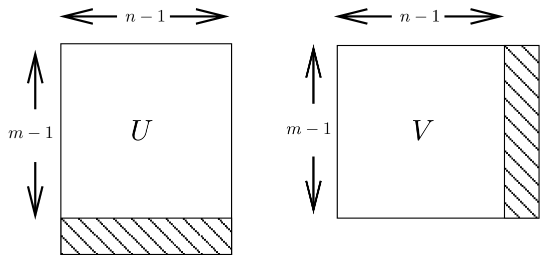


Figure 3.9: *The hatched areas illustrate the last row of U and the last column of V which are removed.*

Removing columns from a matrix, is a matter of multiplying this matrix with a modified "identity" matrix from the right. Removing rows is done in a similar way multiplying the matrix from the left.

We define $\tilde{I}_m$ to be a matrix with dimensions $(m-1)\times m$, which is an identity matrix, where the last row has been removed

$$\tilde{I}_m = \begin{bmatrix} 1 & & & 0 \\ & \ddots & & \vdots \\ & & \ddots & \vdots \\ & & 1 & 0 \end{bmatrix} .$$

Thus the product $X\tilde{I}_n^T$ will result in removing the last column of X. Similar we have in the t direction that the product $\tilde{I}_m X$ results in removing the last row of X.

In this way we approximate the functional (3.10) with

$$J_{\mathrm{TV}}(X) \approx \sum_{ij} \sqrt{\left(D^{(s)}X\tilde{I}_n^T\right)_{ij}^2 + \left(\tilde{I}_m X D^{(t)^T}\right)_{ij}^2} \quad ,$$

which can be rewritten in terms of Kronecker products

$$\begin{aligned} J_{\mathrm{TV}}(X) &\approx \sum_{\hat{I}(i,j)} \sqrt{\left(\left(\tilde{I}_m \otimes D^{(s)}\right)\mathrm{vec}(X)\right)_{\hat{I}}^2 + \left(\left(D^{(t)} \otimes \tilde{I}_n\right)\mathrm{vec}(X)\right)_{\hat{I}}^2} \\ &= \sum_{\hat{I}} \sqrt{\left(\tilde{\mathcal{D}}^{(s)}\hat{x}\right)_{\hat{I}}^2 + \left(\tilde{\mathcal{D}}^{(t)}\hat{x}\right)_{\hat{I}}^2}\ , \end{aligned}$$

where $\hat{I} = \hat{I}(i,j)$ is the index corresponding to the stacked vector $\hat{x}$. Adding a tilde in the symbol $\tilde{\mathcal{D}}^{(s)}$ means that the corresponding operator has been corrected to ensure match of the dimensions of U and V.

If one examines the nullspaces of the operators $\tilde{\mathcal{D}}^{(s)}$ and $\tilde{\mathcal{D}}^{(t)}$ it is revealed that the idea of removing data is a very poor idea. In Fig. 3.10 the two–dimensional vectors spanning the nullspaces are illustrated. They are computed by means of a small test problem where $X \in \mathbb{R}^{4\times 4}$. In this case the dimensions of the Kronecker products $\tilde{\mathcal{D}}^{(s)}$ and $\tilde{\mathcal{D}}^{(t)}$ are 16×9. Thus we expect a seven–dimensional nullspaces of these operators.

Using some mathematics we can examine the nullspace of a Kronecker product by examining the nullspaces of the individual parts of the product. Recall the important relation given in (2.4), i.e., $\mathrm{vec}(\bar{A}X\breve{A}^T) = [\breve{A}\otimes\bar{A}]\mathrm{vec}(X)$. The modified operator $\tilde{\mathcal{D}}^{(s)}$ we examine by studying the two terms in the Kronecker product, namely $D^{(s)}$ and $\tilde{I}_n$. We have that

$$\mathcal{N}(D^{(s)}) = \mathrm{span}(\tilde{e})\ ,\ \tilde{e} = [1,\dots,1]^T$$

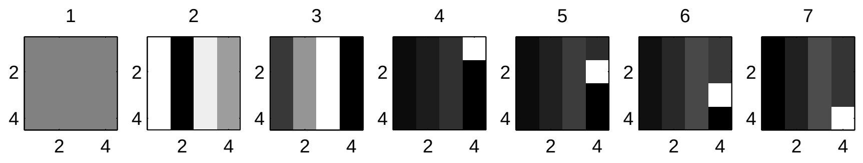
and

$$\mathcal{N}(\tilde{I}_n) = \mathrm{span}(\tilde{e}_n)\ ,\ \tilde{e}_n = [0,\dots,0,1]^T\ .$$

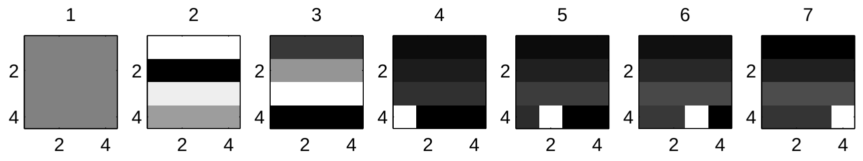
From these nullspaces we can describe the nullspace of the operator $\mathcal{N}(\tilde{\mathcal{D}}^{(s)})$. Denote an arbitrary vector q, and denote the stacked outer product $\mathcal{Q} = \tilde{e}q^T$ will lie in $\mathcal{N}(\tilde{\mathcal{D}}^{(s)})$, because

$$\begin{aligned} \left(\tilde{I}_n \otimes D^{(s)}\right)\mathrm{vec}(\mathcal{Q}) &= \mathrm{vec}\left(D^{(s)}\mathcal{Q}\tilde{I}_n^T\right) \\ &= \mathrm{vec}\left(D^{(s)}\tilde{e}q^T\tilde{I}_n^T\right) \\ &= \mathrm{vec}\left((D^{(s)}\tilde{e})(\tilde{I}_n q^T)\right) \\ &= 0\ , \end{aligned}$$

where the last relation holds since $D^{(s)}\tilde{e} = 0$. In the same way introducing yet another arbitrary vector $\tilde{q}$ we see that the stacked outer product of $\tilde{e}_n\tilde{q}^T$ also belongs to the nullspace of the operator.



(a) *Nullspace of* $\tilde{\mathcal{D}}^{(s)}$



(b) *Nullspace of* $\tilde{\mathcal{D}}^{(t)}$

Figure 3.10: *The seven 2D vectors spanning the nullspace of (a)* $\tilde{\mathcal{D}}^{(s)}$ *and (b)* $\tilde{\mathcal{D}}^{(t)}$.

This means that the nullspace is spanned by

$$\text{span}(q_1\tilde{e}^T, q_2\tilde{e}^T, \dots, q_n\tilde{e}^T, \tilde{e}_n\tilde{q}_1^T, \tilde{e}_n\tilde{q}_2^T, \dots, \tilde{e}_n\tilde{q}_m^T) \ .$$

Since q and $\tilde{q}$ are arbitrary vectors, we see that if we choose $q_1 = \tilde{e}_n$ and $\tilde{q}_1^T = \tilde{e}^T$ the products $q_1\tilde{e}^T$ and $\tilde{e}_n\tilde{q}_1^T$ are the same. This means that one vector is linearly dependent on the others, and that there are $m+n-1$ vectors spanning the nullspace. This does not prove that the nullspace can be of even lower dimension. One can for example use Gram–Schmidt orthonormalization method to show this.

In relation to the example that the nullspace in fact is seven–dimensional. We see that the nullspace of $\tilde{\mathcal{D}}^{(s)}$ is spanned by two–dimensional vectors, which are constant in the s–direction, except for the last column, where a high frequent behavior is discovered. Similar behavior is seen for the $\tilde{\mathcal{D}}^{(t)}$ operator in the t–direction.

Examining the nullspaces of $\tilde{D}^{(s)}$ and $\tilde{D}^{(t)}$, we can make conclusion on the set of arguments that make the TV–operator return zero. Since the TV–operator is non–linear we can not speak about a nullspace, but we can define clearly we have

$$\tilde{\mathcal{N}}(J_{\text{TV}}) = \mathcal{N}(\tilde{\mathcal{D}}^{(s)}) \cap \mathcal{N}(\tilde{\mathcal{D}}^{(t)}) \ ,$$

i.e., by $\tilde{\mathcal{N}}$ we denote all entries that makes J_{TV} return zero. Thus removing data as illustrated is not a good idea, because the nullspace of the derivative operators is spanned by vectors with high frequent components. Thus $\tilde{\mathcal{N}}(J_{\text{TV}})$ includes a constant vector and a “spike” in the corner.

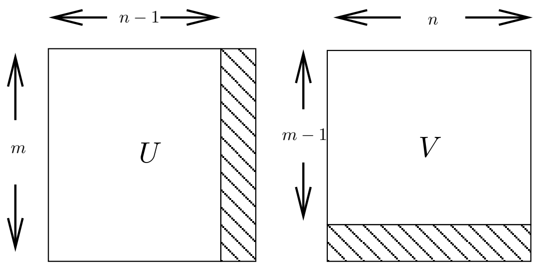


Figure 3.11: *The hatched areas illustrate the extra data which are appended to U and V.*

The approach of removing data is the same as changing the upper limits on the indices in (3.14) and (3.15) to $m-1$ and $n-1$, respectively. Thus we can see directly that the element X_{mn} is not represented in the Total Variation measure and hence not penalized at all. The result is that the variable can be any value predicted by means of the residual term in the Tikhonov formulation only. In practice this means that the given element is not penalized at all, because the residual term often allows a high frequent behavior and thus an unbounded value for X_{mn}. An example of a regularized solution computed by means of the above mentioned approach is shown in Chap. 7.

3.5.2.2 Introducing zero data to U and V

Instead of removing data from U and V we will investigate the influence of introducing extra data, such that the dimensions of U and V match. This is illustrated in Fig. 3.11.

Here we will investigate the case of adding zero data.

Modifying the forward approximation $D^{(\mathrm{for})}$ by appending a zero row after the last row, we obtain

$$\tilde{D}^{(\mathrm{for})} = \frac{1}{h} \begin{bmatrix} -1 & 1 & & & \\ & -1 & 1 & & \\ & & \ddots & \ddots & \\ & & & -1 & 1 \\ 0 & & \dots & & 0 \end{bmatrix}.$$

This approach is the same as changing the upper limit on the indices in (3.14) and

(3.15) to m and n respectively and defining that U_{ij} and V_{ij} should be zero, whenever i and j is "out of range", i.e.,

$$U_{ij} = 0 \ , \ \text{for } i > n-1 \quad \text{and} \quad V_{ij} = 0 \ , \ \text{for } j > m-1 \ .$$

This is similar to imposing a homogenous Neumann boundary condition on each part of the boundary. Adding a zero row does not change the nullspace of the operator, i.e., $\mathcal{N}(D^{(\text{for})}) = \mathcal{N}(\tilde{D}^{(\text{for})})$. This means that the operator $\mathcal{N}(\tilde{D}^{(\text{for})})$ fulfills the requirements having a low frequent nullspace and ensuring dimension match.

We could be satisfied with this derivative operator, but for completeness we will investigate the meaning of adding non–zero data also. Instead of looking at the forward approximation again, we will use the central estimate and add data to this to ensure match of dimensions as well as for obtaining a low frequent nullspace.

3.5.2.3 Adding non–zero data to U and V

A good idea is to introduce two extra rows in the operator $D^{(\text{cen})}$ given in (3.9). The problem with $D^{(\text{cen})}$ being suitable is that the nullspace contains high frequency components. Adding e.g., two zero rows in the operator would not change the nullspace, we should add something that changes the nullspace. Adding two rows from a forward approximation in the first and last row, we obtain

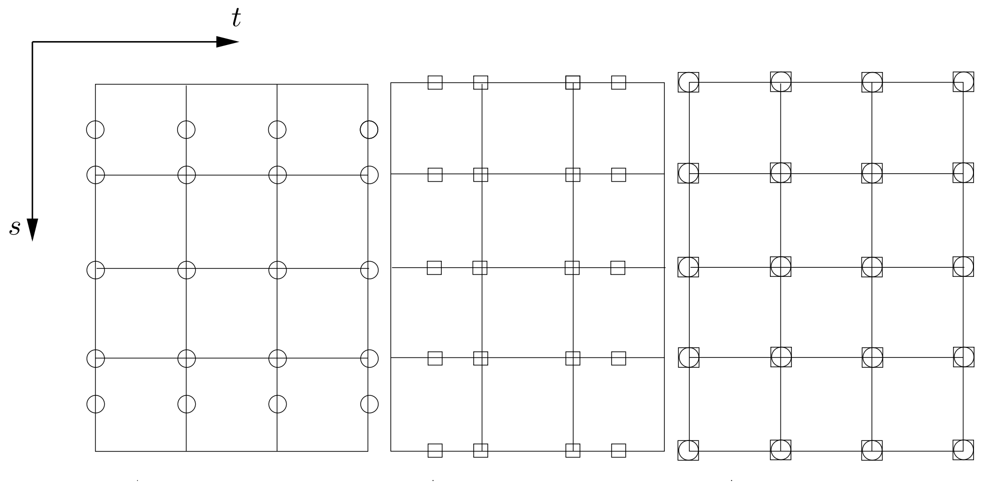
$$\tilde{D}^{(\text{cen})} = \frac{1}{2h} \begin{bmatrix} -2 & 2 & & & & \\ -1 & 0 & 1 & & & \\ & -1 & 0 & 1 & & \\ & & \ddots & \ddots & \ddots & \\ & & & -1 & 0 & 1 \\ & & & & -2 & 2 \end{bmatrix} . \tag{3.16}$$

Notice the difference in magnitude of the quantities. The first order approximation in the ends are twice the size of the rest of the elements, since the forward approximation uses one interval where the central approximation uses two intervals.

Investigating $\mathcal{N}(\tilde{D}^{(\text{cen})})$ we get the same result as visualized in the topmost plot in Fig. 3.8. The nullspace is spanned by a constant vector, i.e., it is low frequent.

In Fig. 3.12 the two–dimensional operator is illustrated, i.e., it illustrates at which points the operator derives.

This approach has all the properties we desire. It is a central estimate in the greater part of the domain, and on the boundary it is a forward approximation. Furthermore



(a) U (b) V (c) *The connected operator*

Figure 3.12: *The modified derivative operator. Note that the forward approximations at the ends are moved to the boundary to represent the derivative there.*

the nullspace is low frequent, thus this derivative operator is suitable for use in regularization. This approach is also the one, which is used in the implementation of the algorithm.

In [20, Sec. 8.2.2] the approach of removing data is used, with a minor change: The first row of U and the first column of V is removed. In [15] two different approaches are described. One can be considered as a variation of the method of removing data, the other ensures dimension match by multiplying with and "smoothing" matrix. Neither [15] nor [20] considered the aspects of a possible high frequent null–space.

3.6 Linear Approximation to the Total Variation Functional

As mentioned the non–linearity of the TV functional gives rise to troubles. The two–dimensional functional can be approximated with a linear expression in the following way

$$J_{\mathrm{TV}}(X) \approx \sum_{i}^{\hat{m}} \sum_{j}^{\hat{n}} \sqrt{U_{ij}^2 + V_{ij}^2} \approx \sum_{i}^{\hat{m}} \sum_{j}^{\hat{n}} |U_{ij}| + |V_{ij}| \ ,$$

where U and V are given in (3.11). Hence the discrete functional can be written in terms of Kronecker products

$$J_{\mathrm{TV}}(X) \approx \left\| \begin{pmatrix} I_m \otimes D^{(s)} \\ D^{(t)} \otimes I_n \end{pmatrix} \mathrm{vec}(X) \right\|_1 \ ,$$

where $D^{(s)}$ and $D^{(t)}$ can be any discretizations of first order derivatives (with low–frequent nullspaces), e.g., the forward approximation given in Eq. (3.7) can be used. This approach is used successfully in [15]. Beside the advantage of the expression being linear, we see that we do not have to deal with how to ensure that the dimensions of the derivative operators in the two directions are the same. However, in this work we focus on the non–linear "true" TV.

Chapter Summary

In this chapter we have introduced the Total Variation functional. We have described how to use it as a penalty term in Tikhonov's regularization method. We have concluded that one reason to use TV is that it allows a wider class of solutions. We have discussed how to discretize the TV functional and shown that in terms of regularization, it is very important how one chooses this discretization. The approach of removing data to ensure match of dimensions of the derivatives is a bad idea, because high frequencies are allowed in the regularized solution. On the other hand it is shown that different variants of the approach of adding data can successfully be applied to the stated problem. Finally, we have given our suggestion on how to choose a good discretization.

Chapter 4

Optimization Theory

In the previous chapter we saw that the problem to solve is an optimization problem, i.e., we want to find the minimizer x^* of a continuous and non–linear function $T : \mathbb{R}^n \rightarrow \mathbb{R}$. Optimization is a very large field, and this chapter will only cover the subjects needed for an understanding of the problem and the development of the algorithm, which we deal with in the next chapter.

The chapter is divided into two major parts. The first part covers the basic theory, e.g., necessary and sufficient conditions for a local minimizer. The second part deals with descent methods, such as the steepest descent and Newton's method.

The definitions and theorems described in the following sections are covered in [6], [8] and [9].

4.1 Theory and Definitions

We first introduce some needed information such as definition of the gradient and the Hessian.

The gradient of a continuous function $T : \mathbb{R}^n \rightarrow \mathbb{R}$ is defined as

$$\nabla T(x) = T'(x) \equiv \begin{bmatrix} \frac{\partial T}{\partial x_1} \\ \vdots \\ \frac{\partial T}{\partial x_n} \end{bmatrix} \tag{4.1}$$

and the elements of the Hessian matrix of the same function T are defined by

$$T''_{ij}(x) \equiv \frac{\partial^2 T}{\partial x_i \partial x_j} \tag{4.2}$$

Note that the Hessian matrix is symmetric.

An indispensable technique which is very often used in approximation of functions is the *Taylor series* or *Taylor expansion*. Using Taylor series one can approximate a given smooth function $T(x)$ with arbitrary precision around an expansion point x_0 [8]

$$T(x_0 + h) = T(x_0) + h^T T'(x_0) + \frac{1}{2} h^T T''(x_0) h + \mathcal{O}(\|h\|^3) \ ,$$

where the term $\mathcal{O}(\|h\|^3)$ represents all terms higher than second order. First or second order Taylor expansions are often used to model arbitrary continuous and differentiable functions of higher order. As long as we are relatively close to the expansion point x_0, i.e., h is "small", then the expansion will be a good approximation to the modelled function. The idea is that it is easier to deal with the simpler model. How large h can be chosen and still ensuring that the Taylor expansion is a good approximation depends on the behavior of the given function.

In Sec. 4.2 it will become apparent that many algorithms are based on approximation of the original function in a given iteration.

4.1.1 Conditions for a Local Minimizer

First we explain the definition of a local minimizer.

Definition 4.1.1.

$$\begin{gathered} x^* \text{ is a local minimizer for } T : \mathbb{R}^n \to \mathbb{R} \\ \text{if } T(x^*) \leq T(x) \text{ for } \|x^* - x\| \leq \epsilon \ (\epsilon > 0) \end{gathered}$$

This definition illustrates that $T(x)$ has minimum in x^* in a region defined by ϵ. If T is continuously differentiable, one can check if a point is a local minimizer, by using the following theorems.

Theorem 4.1.2. *A necessary (but not sufficient) condition for x^* being a local minimizer is that*

$$T'(x^*) = 0 \ .$$

If a point fulfills the condition, it is said to be a stationary point.

The reason why condition in Theorem 4.1.2 is not sufficient is that we can experience three different ways of behavior from T in x^*. The stationary point x^* can either be a local minimizer, a local maximizer or a saddle point.

To be able to illustrate a sufficient condition for the stationary point to be a local minimizer, we need to make an important definition, namely definition of positive definiteness.

Definition 4.1.3. *A matrix $A \in \mathbb{R}^{n\times n}$ is said to be positive definite if*

$$v^T A v > 0 \ ,$$

holds for any non–zero $v \in \mathbb{R}^n$. If the term is not strictly larger than zero, i.e., if $v^T A v \geq 0$, A is said to be semi–positive definite [6]. If $v^T A v < 0$ is A said to be negative definite. If A is neither positive definite nor negative definite, it is indefinite.

The sufficient condition for a point being a local minimizer is given as

Theorem 4.1.4. *Assume that x_{stat} is a stationary point. A sufficient condition for x_{stat} being a local minimizer is that $T''(x_{\text{stat}})$ is positive definite.*

The sufficient condition for a point being a global unique minimizer is given in Theorem 4.1.5.

Theorem 4.1.5. *If $T''(x)$ is strictly positive definite for all x, then $T(x)$ is a convex function, i.e., it has a global and unique minimizer.*

4.2 Methods

Here we will present some algorithms, which are frequently used in optimization problems.

4.2.1 Iterative Descent Methods

An iterative method is a method that produces a series of iterates

$$x^{[1]}, x^{[2]}, x^{[3]}, \ldots$$

such that $x^{[k]}$ converges to the minimizer x^* of $T(x)$ for $k \to \infty$.

Here we will focus on *descent methods*, i.e., methods that have the descent property. The property is that the function value decreases in every iteration, i.e.,

$$T\big(x^{[i+1]}\big) < T\big(x^{[i]}\big),\ \forall\ i\ .$$

Many iterative algorithms for solving non–linear optimization problems use a model in each iteration based on a Taylor series expansion. The *steepest descent* is a simple method, which in each iteration models the non–linear function to minimize with a first order Taylor series expansion, neglecting terms higher than first order.

The idea is to follow a path of descent directions predicted by gradient information only. The direction of the gradient, is that direction where we get the greatest decrease in the function value. Thus in each step the gradient is determined, and a step in the opposite direction of this is taken; a direction which is simply called as steepest descent.

The steepest descent is globally convergent (to a local minimum), but it only has linear convergence. Other methods are faster converging. One of them is Newton's method, which we will present in the following section.

4.2.2 Newton's Method

The Newton method has over the years been widely used for solving unconstrained optimization problems. In principle Newton's method (originally Newton–Raphson) is meant for finding roots of a function. However, a given minimization problem can easily be transformed into a problem of finding a root of the function by using the condition for stationarity, namely Theorem 4.1.2.

It is just like steepest descent an iterative method. Compared to steepest descent it is often better, because it uses second order information and it has quadratic convergence "close" to a minimizer. Using Taylor series expansion we obtain

$$T(x_0 + h) \approx T(x_0) + h^T T'(x_0) + \tfrac{1}{2} h^T T''(x) h = q(h) \; .$$

I.e., $q(h)$ is the quadratic model that approximates the objective function at a given iterate. If $T''(x)$ is non–singular the (unique) minimizer of $q(h)$ is obtained for $q'(h) = 0$, i.e.,

$$\begin{aligned} & T'(x_0) + T''(x_0)h = 0 \\ \Leftrightarrow \quad & T''(x_0)h = -T'(x_0) \; . \end{aligned} \tag{4.3}$$

Thus the minimizer of the quadratic model is obtained by solving the linear system in (4.3).

Theorem 4.2.1. *If $T''(x)$ is strictly positive definite, the direction δx determined from*

$$T''(x)\delta x = -T'(x)$$

is a descent direction.

Theorem 4.2.1 is very important. It illustrates what condition is needed for the Hessian matrix to ensure that the calculated direction is downhill.

The pseudo code for Newton's method is given in Tab. 4.1.

One problem with the Newton method is that it may have a poor convergence in the beginning of the iterations, if the starting guess is "far" from the solution. To ensure that the Newton method converges one has to add a step controlling mechanism. One of these mechanisms is called a *line search procedure*, and it is the subject of Sec. 4.2.3.

How to choose the stopping criteria for Newton's method, we will discuss in Chap. 6.

4.2.3 Line Search Procedure

To ensure convergence for the Newton method one has to incorporate a mechanism that controls step length. Basically there are three ways of doing this, namely by

```
Initial guess $x^{[0]}$
Set $k := 0$
while not STOP do
  Solve $T''(x^{[k]})\delta x^{[k]} = -T'(x^{[k]})$ for $\delta x^{[k]}$ (Calculate Newton direction)
  $\tau^{[k]} := \underset{\tau>0}{\operatorname{argmin}} \{T(x^{[k]} + \tau\delta x^{[k]})\}$(Line search)
  $x^{[k+1]} := x^{[k]} + \tau^{[k]}\delta x^{[k]}$ (Take step)
  $k := k + 1$
end
```

Table 4.1: Pseudo code for Newton's method

using a hybrid method that, e.g., combines steepest descent and Newton's methods, a *trust region* strategy or a *line search procedure*.

The first approach is for general optimization known as *Damped Newton*. The step direction is a combination of the steepest descent and the Newton direction. The damping parameter (i.e., indirectly the length of the step) is determined using a *gain–factor*, which is a number that describes how good a approximation the model is to the underlying function.

The second strategy defines a radius in which the model is assumed to fit the underlying objective function (again using the gain–factor). This strategy we will not describe further, in this work the third approach, the line search, is used.

Again there exist many different types of line search procedures. The basic idea behind the approach is to minimize the objective function in the determined search direction $\delta x^{[k]}$, i.e., to find the τ

$$\tau := \underset{\tau>0}{\operatorname{argmin}} \left\{T(x^{[k]} + \tau\delta x^{[k]})\right\} \ .$$

Line search procedures split up into two major classes, namely exact and soft line search. In exact line search one computes a minimizer in the search direction, which is close to the exact one. In most cases this is too expensive and inefficient, [9]. In soft line search, the criterion for an acceptable step length is more relaxed. A good and efficient implementation of line search procedures is very important to ensure the fastest possible convergence of the algorithm.

We will not go further into the subject of line search procedures here. For more on this subject see e.g., [8] and [9].

Chapter Summary

In this chapter we presented necessary theory by describing important definitions and theorems. We have touched the theory behind Newton's method and line search procedures.

Chapter 5

Algorithm - Theory

One of the aims of this work is to develop and implement an algorithm that efficiently computes solutions to the minimization problem (3.4). For the purpose we will use Newton's method. The problems may be large–scale, so the focus is partly to retrieve a system which can solved efficiently and stably with respect to numerical issues.

In this chapter we will deduce the theory behind the algorithm to solve the proposed problem (3.4).

We want to minimize the non–linear objective function $T(x) : \mathbb{R}^n \rightarrow \mathbb{R}$, i.e.,

$$\min_x T(x) \ ,$$

where

$$T(x) = \tfrac{1}{2}\|Ax - b\|_2^2 + \lambda J_{\mathrm{TV}}(x) \ . \tag{5.1}$$

According to Theorem 4.1.2 a necessary, but not sufficient, condition for a local minimum is to find where the gradient of the objective function is zero, i.e.,

$$T'(x) = 0 \ . \tag{5.2}$$

In this way the minimization problem, has been changed to be a problem of finding one (or more) roots of $T'(x)$. The stated condition is sufficient as well, if the objective function is convex. That the objective function in fact is convex, we will turn our focus on and prove in a later section.

There exist many different methods for finding the roots of a function. One of the most widely used methods is the Newton–method combined with a line search procedure, which is described in Sec. 4.2.2.

In the following sections we will present different ways of solving the given problem. First we present the "standard" Newton approach, hence we require a derivation of the the first and the second derivative of the objective function $T(x)$.

5.1 Standard Newton – The One Dimensional Case

One problem arises, when we want to calculate the first derivative of the objective function. The TV penalty functional given in Eq. (3.1) is not differentiable at locations where $|\nabla f(t)| = 0$. We cope with this problem by perturbing the functional with a small (positive) constant β^2. Hence the perturbed functional is for a continuous function $f : \mathbb{R}^n \to \mathbb{R}$ given in the following way

$$\mathcal{J}_{\mathrm{TV},\beta}(f) \equiv \left\| \sqrt{\|\nabla f\|_2^2 + \beta^2} \right\|_1 , \tag{5.3}$$

i.e., in one dimension

$$\mathcal{J}_{\mathrm{TV},\beta}(f) \equiv \left\| \sqrt{\left(\frac{\mathrm{d}f}{\mathrm{dt}}\right)^2 + \beta^2} \right\|_1 . \tag{5.4}$$

The discrete version of Eq. (5.4) is given directly by

$$J_{\mathrm{TV},\beta}(x) \equiv \left\| \left((Dx)^{\odot 2} + \beta^2 \tilde{e} \right)^{\odot 1/2} \right\|_1 , \tag{5.5}$$

where $(\cdot)^{\odot}$ denotes elementwise raising to a power. The quantity $\tilde{e}$ is a vector of all ones and D is a approximate discrete derivative operator as described in Sec. 3.5.1. We will later focus on the parameter β.

5.1.1 The Gradient of the Objective Function

The objective function given in (5.1) is a sum of two terms. Hence the gradient of the objective is the sum of the gradients of the two terms. Here we will derive the gradient of these two terms separately. From standard least–square calculations it is known that $\nabla(\frac{1}{2}\|Ax - b\|_2^2) = A^T(Ax - b)$. Here we see the reason for the constant $1/2$, namely that we avoid any constant in the gradient and later in the expression for the Hessian.

The last term we can rewrite as a sum

$$\Omega(x) = J_{TV,\beta}(x) = \sum_{i=1}^{\hat{n}} \psi_i,$$

with

$$\psi_i = \sqrt{u_i^2 + \beta^2}\ ,\ u_i = (Dx)_i$$

If D is a standard forward approximation, i.e., as in Eq. (3.7), then $u_i = x_{i+1} - x_i$ and $\hat{n} = n-1$. In general $\hat{n}$ depends on how the discrete approximation to the derivative is chosen. Here we will deduce the gradient and the Hessian without choosing the discrete approximation to the derivative explicitly.

Let $\nabla_j\big(\Omega(x)\big)$ denote the jth element of the gradient of Ω. The calculations involve use of the chain rule and simple calculus.

$$\begin{aligned}
\nabla_j\big(\Omega(x)\big) &= \frac{\partial \Omega(x)}{\partial x_j} \\
&= \sum_{i=1}^{\hat{n}} \frac{\partial \psi_i}{\partial x_j} \\
&= \sum_{i=1}^{\hat{n}} \frac{\partial \psi_i}{\partial u_i} \frac{\partial u_i}{\partial x_j} \\
&= \sum_{i=1}^{\hat{n}} \frac{2u_i}{2\sqrt{u_i^2 + \beta^2}} \cdot \frac{\partial u_i}{\partial x_j} \\
&= \sum_{i=1}^{\hat{n}} \frac{1}{\psi_i} [Dx]_i \cdot D_{ij} \\
&= D_{:j}^T \Psi^{-1} Dx\ ,
\end{aligned}$$

where

$$\Psi = \mathrm{diag}(\psi_i) = \mathrm{diag}\left(\left[[Dx]^{\odot 2} + \beta^2\right]^{\odot 1/2}\right)\ .$$

We see that the gradient can be formulated as a matrix/vector product in the following form

$$\nabla\big(\Omega(x)\big) = \breve{L}x\ ,$$

where

$$\breve{L} = D^T \Psi^{-1} D \tag{5.6}$$

is a symmetric matrix. D can in principle be an arbitrary matrix, but in order to represent the gradient of the TV functional, D should be an approximate discrete first order derivative operator.

To summarize the gradient of the objective function is given by

$$\begin{aligned} T'(x) &= A^T[Ax - b] + \lambda \breve{L} x \ , & (5.7) \\ &= A^T[Ax - b] + \lambda D^T \Psi^{-1} D x \ . & (5.8) \end{aligned}$$

5.1.2 The Hessian of the Objective function

In standard least–squares calculations one can easily derive $H(\frac{1}{2}\|Ax - b\|_2^2) = A^T A$. The Hessian of the last term in the objective function is more complicated. The calculations are written out in detail here.

$$\begin{aligned} (\Omega''(x)_{kj}) =& \frac{\partial^2 \Omega(x)}{\partial x_k \partial x_j} \\ =& \frac{\partial}{\partial x_k} \left(\sum_{i=1}^{\hat{m}} \frac{1}{\psi_i} [Dx]_i \cdot D_{ij} \right) \\ =& \sum_{i=1}^{\hat{m}} D_{ij} \frac{\partial}{\partial x_k} \left(\frac{1}{\psi_i} [Dx]_i \right) \\ =& \sum_{i=1}^{\hat{m}} D_{ij} \left[\frac{\partial \psi_i^{-1}}{\partial x_k} [Dx]_i + \frac{1}{\psi_i} \frac{\partial [Dx]_i}{\partial x_k} \right] \\ =& \breve{L} + \sum_{i=1}^{\hat{m}} D_{ij} \left[-\frac{1}{\psi_i^2} \frac{\partial \psi_i}{\partial u_i} \frac{\partial u_i}{\partial x_k} [Dx]_i \right] \\ =& \breve{L} + \sum_{i=1}^{\hat{m}} D_{ij} \left[-\frac{[Dx]_i^2}{\psi_i^3} D_{ik} \right] \end{aligned}$$

This information can be rewritten in matrix form in the following way

$$H\big(\Omega(x)\big) = \breve{L} + \tilde{L}$$

where $\breve{L}$ is given in (5.6) and

$$\begin{aligned}\tilde{L} &= D^T \text{diag}\left(-(Dx)^{\odot 2} \oslash \psi^{\odot 3}\right) D & (5.9)\\ &= -D^T \text{diag}(Dx)^2 \Psi^{-3} D\ . & (5.10)\end{aligned}$$

where $\Psi = \text{diag}(\psi_1, \dots, \psi_n)$. Hence the Hessian of the objective function T is given by

$$\begin{aligned}T''(x) &= A^T A + \lambda \left[\breve{L} + \tilde{L}\right]\\ &= A^T A + \lambda \left[D^T \Psi^{-1} D - D^T \text{diag}(Dx)^2 \Psi^{-3} D\right]\ .\end{aligned}$$

These results substantiate the results from [20]. Inserting the deduced expressions for the gradient and Hessian, the Newton method for solving the stated problem will take the form presented in Tab. 5.1.

```
Initial guess $x^{[0]}$
Set $k := 0$
*while not* STOP *do*
  calculate $\breve{L}$ and $\tilde{L}$ by means of (5.6) and (5.9) respectively
  $T' := A^T(Ax^{[k]} - b) + \lambda \breve{L} x^{[k]}$ (Calculate gradient $T'$ at $x^{[k]}$)
  $T'' := A^T A + \lambda(\breve{L} + \tilde{L})$ (Calculate Hessian at $x^{[k]}$)
  Solve $T'' \delta x^{[k]} = -T'$ for $\delta x^{[k]}$ (Calculate Newton direction)
  $\tau^{[k]} := \underset{\tau > 0}{\text{argmin}} \left\{T(x^{[k]} + \tau \delta x^{[k]})\right\}$(Line search)
  $x^{[k+1]} := x^{[k]} + \tau^{[k]} \delta x^{[k]}$ (Take step)
  $k := k + 1$
*end*
```

Table 5.1: *Pseudo code for Newton's method on the specific Total Variation problem.*

5.2 Dealing with the Non–Linearity of the System

The problem is to find a root for the system given in Eq. (5.1). Due to the presence of the highly nonlinear term $D^T \Psi^{-1} Dx$, Newton's method does not work satisfactorily in the sense that its domain of convergence is very small.

In the following sections we discuss different methods that deal with this problem.

5.2.1 The Influence of The Parameter β

The parameter β which is introduced to ensure that the TV functional is continuously differentiable plays another role too. It can have a great influence on the behavior of the objective function, which can be very non–linear. This is to put it mildly not advantageous for Newton's method.

We can study the reasons for the non–linearity by examining the gradient of the objective function, which is given in Eq. (5.8). The gradient of the penalty term depends on two terms, but in order to study the non–linearity, we examine the gradient of the TV term only. The reason for this is simply that the non–linearity is caused by the TV term only.

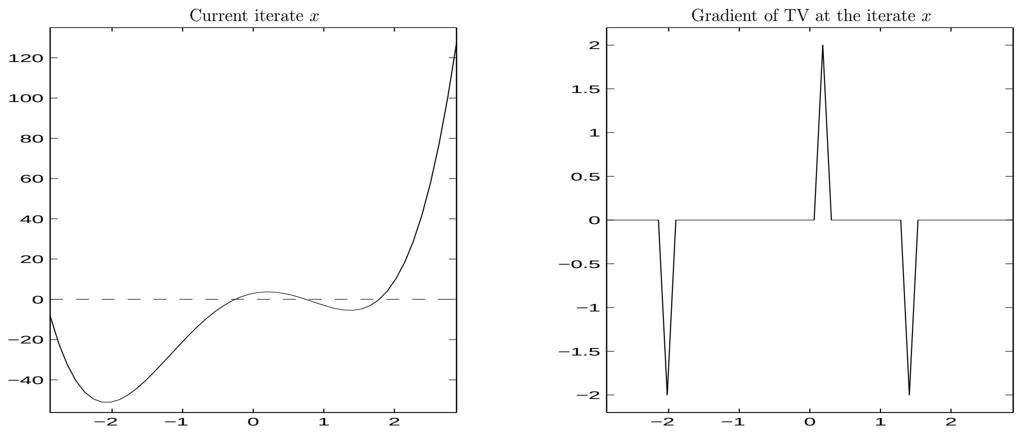


Figure 5.1: *To the left: A current iterate x. To the right: The gradient of the TV term $D^T \Psi^{-1} Dx$ of the iterate to the left.*

In Fig. 5.1 the gradient of the TV term $D^T \Psi^{-1} Dx$ is visualized for a given iterate x. Comparing the current iterate visualized to the left and the behavior of the term

$D^T\Psi^{-1}Dx$ visualized to the right, we see that the gradient of the TV term has spikes. These three spikes appear, where the current iterate has extrema. Notice that the spikes have the same magnitude, i.e., the term does not distinguish between the magnitude of the corresponding values of the iterate. However, the location of the spikes emphasizes the fact that the TV term prefers a constant solution/iterate. The spikes indicate, at which coordinates the current iterate should be changed to obtain a next iterate which approaches a constant solution.

However, this spiky nature, i.e., non–linearity of the gradient is the root of all evil for Newton's method. The reason to this is that Newton's method uses a linear model in each iteration to approximate the underlying function.

There are several ways to make the system more linear. We notice that the constant β which is added to make the TV–functional differentiable, has a huge influence on the linearity of the Total Variation functional. This is illustrated in Fig. 5.2.

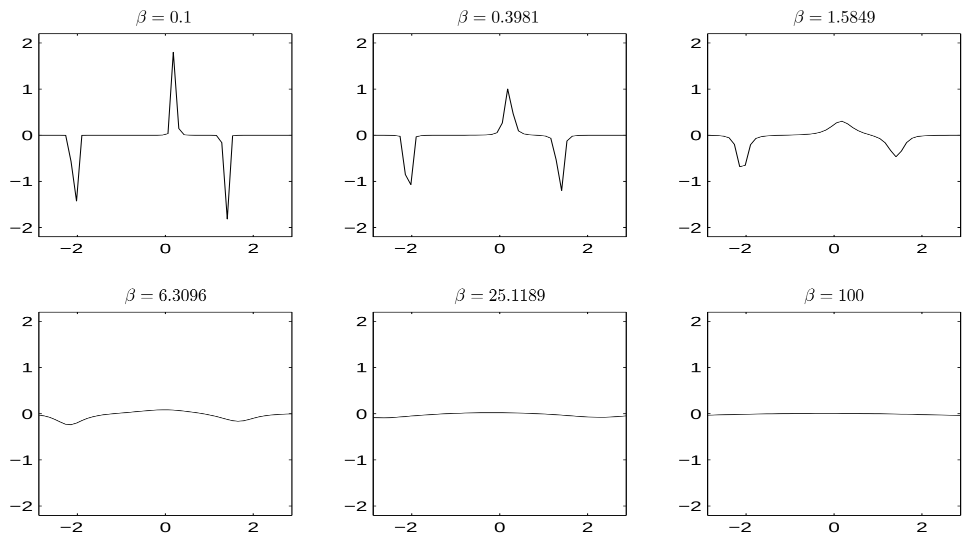


Figure 5.2: *Plots of the gradient of the TV functional $D^T\Psi^{-1}Dx$ for six values of β.*

We see that for small values of β the spikes remain, but for larger values of β the spikes are smoothed out. This means that the gradient becomes more well–behaved. This behavior can be explained by examining the underlying mathematical theory. As a result of the non–linearity of the TV–functional one can not isolate the quantity β and from this make direct conclusions on the behavior. However, we can use a Taylor series expansion and thus linearize the functional.

First we inspect the term

$$\begin{aligned}
\Psi_i^{-1}[Dx]_i &= \frac{[Dx]_i}{\sqrt{[Dx]_i^2+\beta^2}} \\
&= \frac{[Dx]_i}{\big|[Dx]_i\big|\left[1+\left[\frac{\beta}{[Dx]_i}\right]^2\right]} \\
&= \operatorname{sign}([Dx]_i)\left[1-\frac{1}{2}\left[\frac{\beta}{[Dx]_i}\right]^2+\mathcal{O}\left(\left[\frac{\beta}{[Dx]_i}\right]^4\right)\right] \;, \qquad (5.11)
\end{aligned}$$

and thus

$$\begin{aligned}
[D^T\Psi^{-1}Dx]_i &= \operatorname{sign}\big([Dx]_i\big)\left[1-\frac{1}{2}\left[\frac{\beta}{[Dx]_i}\right]^2\right]- \\
&\qquad \operatorname{sign}\big([Dx]_{i-1}\big)\left[1-\frac{1}{2}\left[\frac{\beta}{[Dx]_{i-1}}\right]^2\right]+\mathcal{O}\left(\left[\frac{\beta}{[Dx]_i}\right]^4\right) \\
&= \operatorname{sign}\big([Dx]_i\big)-\operatorname{sign}([Dx]_{i-1}) \\
&\qquad -\frac{1}{2}\operatorname{sign}\big([Dx]_i\big)\left[\frac{\beta}{[Dx]_i}\right]^2 \\
&\qquad +\frac{1}{2}\operatorname{sign}\big([Dx]_{i-1}\big)\left[\frac{\beta}{[Dx]_{i-1}}\right]^2+\mathcal{O}\left(\left[\frac{\beta}{[Dx]_i}\right]^4\right) \;.
\end{aligned}$$

We see from this derivation that the sum of the two first terms are typically zero. The case where it is not zero, is when the iterate has an extremum, i.e., there is a change of sign in the derivative.

The following two terms have the property that when the signs of two neighboring elements in the derivative Dx have the same sign, they more or less cancel out. If this is not the case, i.e., we have a sign change for two neighboring elements in Dx, then the spike effect occurs. In these cases the two terms will counteract this effect.

The conclusion of this experiment is that β has a smoothing effect on the gradient, and thus making it more well behaved.

If we turn the focus on the underlying function Eq. (5.1), where $J_{\mathrm{TV}}(x)$ is the perturbed TV–functional as written in (5.5), namely

$$J_{\mathrm{TV},\beta}(x) \equiv \left\|\big((Dx)^{\odot 2}+\beta^2\tilde{e}\big)^{\odot(1/2)}\right\|_1 \;. \qquad (5.12)$$

we can understand the direct meaning of β in a special case. If we choose β^2 to be "large" corresponding to the derivative term $(Dx)^{\odot 2}$, it will dominate the TV–term and in this case we have

$$J_{TV,\beta}(x) \approx \|\beta\|_1 \ . \tag{5.13}$$

In this way the TV–term is just a constant independent of x, which means that solution to the minimization problem Eq. (5.1) will be the unregularized (naive) least–squares solution. Of course the value of the minimum has been displaced, but the minimizer stays the same as for the least–squares solution.

Notice that when β is changed the objective function changes as well. In [4] a *continuation procedure* is described, where one starts out with a large β and through the iterations decreases the value of the parameter. In [5] it is proved that $\mathcal{J}_{\mathrm{TV},\beta}(x) \rightarrow \mathcal{J}_{\mathrm{TV}}(x)$ for $\beta \rightarrow 0$. Thus the solution goes towards the wanted TV solution for $\beta \rightarrow 0$. However, the authors have not found a successful heuristic to determine the value for β through the iterations.

Hence for now the parameter β should be chosen large enough to ensure differentiability and small enough to ensure correct results. Another way of dealing with the non–linearity is discussed in the following section.

5.3 A New Variable and an Augmented System

In this section we will describe how to benefit of introducing a set of new variables. People working in this field, e.g., the authors of [5] and [20], introduce the variables and connects them to *primal–dual* theory, because the extra set of variables can be interpreted as *dual* variables. However, in the following sections it is shown that an algorithm can be deduced without any use of primal–dual theory.

The focus of this work is to reconstruct solutions with jumps and steep gradients. When a given function has a jump, the gradient is unbounded, i.e., infinite. This fact we must deal with, because this singularity makes the Newton algorithm have an extremely small domain of convergence in each iteration.

A very good idea is to introduce a set of new variables w, which substitute the "most non–linear" part of the system $T'(x)$. This gives rise to a larger, but in general more globally linear, system to solve. This means that instead of (5.2) we solve the larger system

$$\begin{bmatrix} T'(x,w) \\ \mathcal{G}(x,w) \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \end{bmatrix} . \tag{5.14}$$

The basic idea of introducing a new variable, which deals with this behavior, originates from [5] and [20]. The elements of the introduced variable w are given by

$$w_i = \frac{\partial \psi_i}{\partial u_i} = \frac{u_i}{\psi_i} = \psi_i^{-1}[Dx]_i . \tag{5.15}$$

The new variable normalizes the components of the gradient, in fact in one dimension the new variable is by definition the normalized gradient, i.e.,

$$w_i = \frac{[Dx]_i}{\psi_i} = \frac{[Dx]_i}{\sqrt{[Dx]_i^2 + \beta^2}} \simeq \frac{[Dx]_i}{|[Dx]_i|} , \tag{5.16}$$

where the last relation holds if $\beta \ll [Dx]_i$, see Eq. (5.11). Clearly by definition the variable w is bounded. However, from this point we will treat w as a true separate variable.

Treating (5.15) for all i, we define a function $\mathcal{G}(x,w)$

$$\mathcal{G}(x,w) = \Psi w - Dx , \tag{5.17}$$

where $\Psi = \mathrm{diag}(\psi_1, \dots, \psi_n)$.

The gradient of the objective function T has been found in (5.7), i.e., substituting $\Psi^{-1}Dx$ by the variable w, we get

$$T'(x,w) \quad = \quad A^T[Ax - b] + \lambda D^T w . \tag{5.18}$$

We see that the gradient is not directly dependant on β – indirectly because the other part of the system does. The use of Newton's method requires a linearization of the system (5.14). Introducing small perturbations δx and δw of the of the function x and the variable w respectively, we obtain using Taylor series expansion

$$T'(x + \delta x, w + \delta w) \simeq T'(x,w) + A^T A \delta x + \lambda D^T \delta w . \tag{5.19}$$

Note that this is an approximation, where second order terms have been neglected. Linearizing $\mathcal{G}(x, w)$ where we only look at the ith element of $\mathcal{G}(x, w)$, we obtain

$$\begin{aligned}\mathcal{G}_i(x+\delta x, w+\delta w) &= \psi_i(x+\delta x, w+\delta w)[w_i+\delta w_i] - (D[x+\delta x])_i \\ &\simeq [\psi_i(x) + w_i[D\delta x]_i][w_i+\delta w_i] - (D[x+\delta x])_i \\ &\simeq \mathcal{G}_i(x,w) - [1-w_i^2][D\delta x]_i + \psi_i(x)\delta w_i \ .\end{aligned}$$

Again second order terms have been omitted. For sufficiently small values of δx and δw this approximation is valid in a region around the expansion points x and w. For all i we have

$$\mathcal{G}(x+\delta x, w+\delta w) \simeq \mathcal{G}(x,w) - [I-W^2]D\delta x + \Psi\delta w \ , \tag{5.20}$$

where $W = \mathrm{diag}(w_1, \ldots, w_n)$. As $\mathcal{G}(x, w)$ plays the role of residual it is zero at the optimal solution pair (x^*, w^*) – in general during the Newton iterations $\mathcal{G}(x, w)$ will differ from zero. Setting $\mathcal{G}(x + \delta x, w + \delta w) = 0$, equation (5.20) can be rewritten to

$$\lambda D\delta x - \lambda\left[I-W^2\right]^{-1}\Psi\delta w = \lambda\left[I-W^2\right]^{-1}\mathcal{G}(x,w) \ . \tag{5.21}$$

Combining (5.19) and (5.21) we obtain the following system to solve

$$\begin{bmatrix} A^TA & \lambda D^T \\ \lambda D & -\lambda\left[I-W^2\right]^{-1}\Psi \end{bmatrix}\begin{bmatrix}\delta x \\ \delta w\end{bmatrix} = \begin{bmatrix} -T'(x,w) \\ \lambda\left[I-W^2\right]^{-1}\mathcal{G}(x,w)\end{bmatrix} \ , \tag{5.22}$$

where the coefficient matrix is the Jacobian of the coefficient matrix in the system (5.14).

Using block elimination we obtain

$$\left[A^TA + \lambda D^T\Psi^{-1}\left[I-W^2\right]D\right]\delta x = -T'(x) \ . \tag{5.23}$$

The corresponding δw must be calculated afterwards. This is done by inserting δx in one of the two block equations, e.g.,

$$\delta w = -w + \Psi^{-1}Dx + [I-W^2]\Psi^{-1}D\delta x \ . \tag{5.24}$$

According to Theorem 4.2.1 the step δx calculated in (5.23) will be in a descent direction, if the coefficient matrix $A^TA + \lambda D^T\Psi^{-1}\left[I-W^2\right]D$ is positive definite.

In [1] it is proved that $\mathcal{J}_{TV,\beta}$ from (5.4) is a convex function for $\beta > 0$. This proves that the objective function in general (for $\lambda > 0$) is convex, thus having a unique minimizer. However, the authors do not prove the convexity after the introduction of the new variable and hence they do not conclude anything about the existence of a possible unique minimizer of the function. This is the reason why, we will investigate, if the augmented system has a unique minimizer.

The investigation can be carried out by determining, if the coefficient matrix in the system (5.23) is positive definite. To test for positive definiteness we use Def. 4.1.3. Let v be a arbitrary non–zero vector, then

$$v^T A^T A v = \bar{u}^T \bar{u} = \|\bar{u}\|_2^2 \geq 0 \ , \tag{5.25}$$

where $\bar{u} = Av$. The term $A^T A$ will (at least) be semi–positive definite. The case where $v^T A^T A v = 0$ is when the matrix is rank–deficient, namely when $Av = 0$, i.e., when v lies in the null–space of A. If A has full rank, $\bar{u}^T \bar{u} > 0$, i.e., $A^T A$ is positive definite.

Looking at the second term, we neglect the positive constant λ. The term $\Psi^{-1}\left[I - W^2\right]$ is a diagonal matrix with non–negative elements only, when $|w_i| \leq 1$. Letting $\Psi^{-1}\left[I - W^2\right] = \bar{\Upsilon}^{T^{1/2}} \bar{\Upsilon}^{1/2}$, we get

$$\begin{aligned} & v^T \left[\lambda D^T \Psi^{-1}\left[I - W^2\right] D\right] v \\ = \ & v^T D^T \bar{\Upsilon}^{T^{1/2}} \bar{\Upsilon}^{1/2} D v \\ = \ & \hat{u}^T \hat{u} \\ = \ & \|\hat{u}\|_2^2 \geq 0 \ , \end{aligned}$$

where $\hat{u} = \bar{\Upsilon}^{1/2} D v$. The matrix is positive semi–definite, because the product Dv is zero, when v lies in the null–space of D. By a proper choice of the first order derivative operator D, this is the case, when v is constant.

Thus the coefficient matrix is positive semi–definite in general if the nullspace of D is a subset of the nullspace of A, i.e., if $\mathcal{N}(D) \subseteq \mathcal{N}(A)$ – a phenomenon that practically never appears. If this is not the case, the coefficient matrix is positive definite and hence the step δx is indeed in a descent direction.

This basically proves the convexity of the objective function as well. This is the case since it holds for arbitrary x and when $|w_i| \leq 1 \ \ \forall i$. The explanation why we limit $|w_i| \leq 1 \ \ \forall \ i$ is given in Sec. 6.5.

Since the coefficient matrix is at least positive semi–definite it is known that if there exist more minimizers, they will all have the same minimum, i.e., corresponding objective value.

5.4 Rewriting the Augmented System

When we solve for the step in (5.22), we see that the quantity A^TA arises. Calculating this matrix product, we will square the condition number of A, we may loose some information because of rounding errors, and thus loosing some digits when solving the problem, see e.g., [7, p. 256]. Avoiding this is especially important, when A has a fairly large condition number. However, these numerical uncertainties can be avoided using a little trick, which is to introduce an auxiliary variable $z = A\delta x$. This gives rise to an extra equation $A\delta x - z = 0$, and the system (5.22) can we rewritten to the following – still symmetric – system

$$\begin{bmatrix} -I & A & 0 \\ A^T & 0 & \lambda D^T \\ 0 & \lambda D & -\lambda \left[I - W^2\right]^{-1} \Psi \end{bmatrix} \begin{bmatrix} z \\ \delta x \\ \delta w \end{bmatrix} = \begin{bmatrix} 0 \\ -T'(x,w) \\ \lambda \left[I - W^2\right]^{-1} \mathcal{G}(x,w) \end{bmatrix}. \tag{5.26}$$

The auxiliary variable z, is often referred to as a *dummy* variable; it is introduced and used in the calculations, but the final value of z is never used.

However, experiments show that both systems have some difficulties, when some of the quantities $|w_i|$ are close to 1. In these cases the block matrix $\lambda \left[W^2 - I\right]^{-1} \Psi$ will be close to singular. This behavior is to put it mildly not appreciated, since the basic idea is that jumps and edges should be allowed.

This we can deal with by forming the Schur–complement of $\begin{bmatrix} -I & A \\ A^T & 0 \end{bmatrix}$ in the following way

$$\begin{aligned} A^{\langle \text{aug} \rangle} &= \begin{bmatrix} -I & A \\ A^T & 0 \end{bmatrix} - \begin{bmatrix} 0 \\ \lambda D^T \end{bmatrix} \left[-\frac{1}{\lambda}[I - W^2]\Psi^{-1} \right] \begin{bmatrix} 0 & \lambda D \end{bmatrix} \\ &= \begin{bmatrix} -I & A \\ A^T & \lambda D^T [I - W^2] \Psi^{-1} D \end{bmatrix}. \end{aligned} \tag{5.27}$$

The right hand side is given by

$$\begin{aligned}
b^{\langle\mathrm{aug}\rangle} &= \begin{bmatrix} 0 \\ -T'(x,w) \end{bmatrix} + \begin{bmatrix} 0 \\ \lambda D^T \end{bmatrix} \left[-\frac{1}{\lambda}[I-W^2]\Psi^{-1}\right] \left[\ \lambda\left[I-W^2\right]^{-1}\mathcal{G}(x,w)\ \right] \\
&= \begin{bmatrix} 0 \\ -T'(x,w) + \lambda D^T\Psi^{-1}\mathcal{G}(x,w) \end{bmatrix} \\
&= \begin{bmatrix} 0 \\ -T'(x) \end{bmatrix} , \qquad (5.28)
\end{aligned}$$

where

$$T'(x) = A^T(Ax-b) + \lambda D^T\Psi^{-1}Dx \ .$$

I.e., the right hand side reduces, such that it depends on x only. The system to solve to find the step is thus given by

$$\begin{aligned}
&\begin{bmatrix} -I & A \\ A^T & \lambda D^T[I-W^2]\Psi^{-1}D \end{bmatrix} \begin{bmatrix} z \\ \delta x \end{bmatrix} = \begin{bmatrix} 0 \\ -T'(x) \end{bmatrix} \qquad (5.29) \\
\Leftrightarrow\quad & A^{\langle\mathrm{aug}\rangle}x^{\langle\mathrm{aug}\rangle} = b^{\langle\mathrm{aug}\rangle} \ . \qquad (5.30)
\end{aligned}$$

The system (5.30) will not have the same numerical problems as the system (5.22), and at the same time we get rid of the problems connected to the inversion $[I-W^2]^{-1}$ in Eq. (5.26).

After a calculation of the unknown variables $x^{\langle\mathrm{aug}\rangle} = [\delta x^T \ z^T]^T$, these are inserted in one of the two block equations two find the value of δw, i.e., we obtain the same updating formulae as written in Eq. (5.24), $\delta w = -w + \Psi^{-1}Dx + [I-W^2]\Psi^{-1}D\delta x$.

Note that the calculation of δw as expected is independent of z. Hence we have derived the system for a determination of the next direction for the step in the Newton method in the one dimensional case.

In the following section we will focus on the derivation of the similar and more complicated system in two dimensions.

5.5 The Augmented System in Two Dimensions

Dealing with the problem in two dimensions, we still want to express the two–dimensional solution as a vector. This can be done using the vec–function from Def. 2.3.2.

Let the vector $x = \mathrm{vec}(X)$ and A a large, maybe dense matrix mapping each element in x to the stacked right hand side vector $b = \mathrm{vec}(B)$.

Looking at the stacked problem it is useful to define new indices $\hat{I}(i,j) = i\cdot(n-1)+j$, which relates the coordinates of the original to the coordinates of the stacked problem. This means basically that $x_{\hat{I}} = X_{ij}$. Notice that the connection between $\hat{I}$ and (i,j) is unique – for a given $\hat{I}$ there exists one and only one pair of (i,j) and vice versa.

In two dimensions the objective function takes the following form

$$\begin{aligned} T(x) &= \tfrac{1}{2}\|Ax-b\|_2^2 + \lambda J_{\mathrm{TV}}(x) \\ &= \tfrac{1}{2}\|Ax-b\|_2^2 + \lambda \sum_{\hat{I}} \psi_{\hat{I}} \qquad (5.31) \\ &= \tfrac{1}{2}\|Ax-b\|_2^2 + \lambda \sum_{\hat{I}} \sqrt{u_{\hat{I}}^2 + v_{\hat{I}}^2 + \beta^2} \qquad (5.32) \end{aligned}$$

where

$$u_{\hat{I}} = (\mathcal{D}^{(s)}x)_{\hat{I}} \quad \text{and} \quad v_{\hat{I}} = (\mathcal{D}^{(t)}x)_{\hat{I}} \ . \qquad (5.33)$$

The operators $\mathcal{D}^{(s)}$ and $\mathcal{D}^{(t)}$ approximates the derivatives of the stacked vector x in the respective directions. It is assumed that these operators are modified to ensure match of dimensions as discussed in Chap. 3.

Calculating the partial derivative of $T(x)$ with respect to $x_{\hat{J}}$ we get

$$\frac{\partial T(x)}{\partial x_{\hat{J}}} = [A^T[Ax-b]]_{\hat{J}} + \lambda \sum_{\hat{I}} \frac{\partial \psi_{\hat{I}}}{\partial x_{\hat{J}}} \ . \qquad (5.34)$$

By applying the chain rule on the last term we obtain

$$\frac{\partial \psi_{\hat{I}}}{\partial x_{\hat{J}}} = \frac{\partial \psi_{\hat{I}}}{\partial u_{\hat{I}}}\frac{\partial u_{\hat{I}}}{\partial x_{\hat{J}}} + \frac{\partial \psi_{\hat{I}}}{\partial v_{\hat{I}}}\frac{\partial v_{\hat{I}}}{\partial x_{\hat{J}}} \ . \qquad (5.35)$$

By analogy with the derivation in the one dimensional case we now introduce two extra variables; one for each dimension, namely

$$w_{\hat{I}}^{(s)} = \frac{\partial \psi_{\hat{I}}}{\partial u_{\hat{I}}} = \frac{u_{\hat{I}}}{\psi_{\hat{I}}} = \frac{(\mathcal{D}^{(s)}x)_{\hat{I}}}{\psi_{\hat{I}}} \qquad (5.36)$$

and

$$w_{\hat{I}}^{(t)} = \frac{\partial \psi_{\hat{I}}}{\partial v_{\hat{I}}} = \frac{v_{\hat{I}}}{\psi_{\hat{I}}} = \frac{(\mathcal{D}^{(t)}x)_{\hat{I}}}{\psi_{\hat{I}}} \ . \qquad (5.37)$$

Inserting in (5.35) we get

$$\frac{\partial \psi_{\hat{I}}}{\partial x_{\hat{J}}} = \mathcal{D}^{(s)}_{\hat{I},\hat{J}} w^{(s)}_{\hat{I}} + \mathcal{D}^{(t)}_{\hat{I},\hat{J}} w^{(t)}_{\hat{I}} \ . \tag{5.38}$$

Inserting this in (5.34) we obtain

$$\frac{\partial T(x, w^{(s)}, w^{(t)})}{\partial x_{\hat{J}}} = [A^T[Ax-b]]_{\hat{J}} + \lambda \sum_{\hat{I}} \left[\mathcal{D}^{(s)}_{\hat{I},\hat{J}} w^{(s)}_{\hat{I}} + \mathcal{D}^{(t)}_{\hat{I},\hat{J}} w^{(t)}_{\hat{I}} \right] \ ,$$

which for all $\hat{J}$ becomes

$$\begin{aligned} T'(x, w^{(s)}, w^{(t)}) &= A^T[Ax-b] + \lambda \sum_{\hat{I}} \left[\mathcal{D}^{(s)}_{\hat{I},:} w^{(s)}_{\hat{I}} + \mathcal{D}^{(t)}_{\hat{I},:} w^{(t)}_{\hat{I}} \right] \\ &= A^T[Ax-b] + \lambda \mathcal{D}^{(s)^T} w^{(s)} + \lambda \mathcal{D}^{(t)^T} w^{(t)} \ . \end{aligned}$$

For each of the two equations (5.36) and (5.37), we define a residual function $\mathcal{G}^{(s)}(x, w^{(s)})$ and $\mathcal{G}^{(t)}(x, w^{(t)})$ respectively

$$\mathcal{G}^{(s)}_{\hat{I}}\big(x, w^{(s)}\big) = \psi_{\hat{I}} w^{(s)}_{\hat{I}} - \big(\mathcal{D}^{(s)} x\big)_{\hat{I}} \tag{5.39}$$

i.e., for all $\hat{I}$

$$\mathcal{G}^{(s)}\big(x, w^{(s)}\big) = \Psi w^{(s)} - \mathcal{D}^{(s)} x \tag{5.40}$$

and in the same way we obtain for the t–direction

$$\mathcal{G}^{(t)}\big(x, w^{(t)}\big) = \Psi w^{(t)} - \mathcal{D}^{(t)} x \ . \tag{5.41}$$

Again we want to solve a system, which is similar to the one seen in the one dimensional case, but larger, namely

$$\begin{bmatrix} T'(x, w^{(s)}, w^{(t)}) \\ \mathcal{G}^{(s)}(x, w^{(s)}) \\ \mathcal{G}^{(t)}(x, w^{(t)}) \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix} \ . \tag{5.42}$$

For this system we will find the Jacobian by applying Taylor series expansion on each part.

$$\begin{aligned} & T'(x+\delta x, w^{(s)}+\delta w^{(s)}, w^{(t)}+\delta w^{(t)}) \simeq \\ &\qquad A^T\big[A[x+\delta x]-b\big] + \lambda \mathcal{D}^{(s)^T}[w^{(s)}+\delta w^{(s)}] + \lambda \mathcal{D}^{(t)^T}[w^{(t)}+\delta w^{(t)}] \\ &\qquad = T'(x, w^{(s)}, w^{(t)}) + A^T A \delta x + \lambda \mathcal{D}^{(s)^T} \delta w^{(s)} + \lambda \mathcal{D}^{(t)^T} \delta w^{(t)} \ . \end{aligned}$$

When applying the linearization to the residual function $\mathcal{G}^{(s)}(x, w^{(s)})$ we obtain

$$\begin{aligned}
\mathcal{G}^{(s)}\left(x+\delta x, w^{(s)}+\delta w^{(s)}\right) &\simeq \Psi(x+\delta x)[w^{(s)}+\delta w^{(s)}] - \mathcal{D}^{(s)}[x+\delta x] \\
&\simeq \left[\Psi\left(x\right) + \operatorname{diag}\left(\frac{\partial \Psi}{\partial x}\delta x\right)\right][w^{(s)}+\delta w^{(s)}] - \mathcal{D}^{(s)}[x+\delta x] \\
&\simeq \Psi\left(x\right) w^{(s)} + \operatorname{diag}\left(\frac{\partial \Psi}{\partial x}\delta x\right) w^{(s)} + \Psi\left(x\right)\delta w^{(s)} - \mathcal{D}^{(s)}[x+\delta x] \\
&\simeq \Psi\left(x\right) w^{(s)} + W^{(s)}\frac{\partial \Psi}{\partial x}\delta x + \Psi\left(x\right)\delta w^{(s)} - \mathcal{D}^{(s)}[x+\delta x] \\
&= \mathcal{G}^{(s)}(x, w^{(s)}) + [W^{(s)}[W^{(s)}\mathcal{D}^{(s)} + W^{(t)}\mathcal{D}^{(t)}] - \mathcal{D}^{(s)}]\delta x + \Psi(x)\delta w^{(s)} \\
&= \mathcal{G}^{(s)}(x, w^{(s)}) + [W^{(s)}W^{(t)}\mathcal{D}^{(t)} - [I - W^{(s)^2}]\mathcal{D}^{(s)}]\delta x + \Psi(x)\delta w^{(s)}\ .
\end{aligned}$$

Note that $W^{(s)}$ and $W^{(t)}$ are diagonal matrices. The term $W^{(s)}W^{(t)}\mathcal{D}^{(t)}$ is not present in the one–dimensional case. The term connects the two space directions.

Note as well that setting all quantities relating the t–direction to zero, we obtain a system which resembles the one obtained in the one–dimensional case.

The function $\mathcal{G}^{(t)}(x, w^{(t)})$ is similar, namely

$$= \mathcal{G}^{(t)}(x, w^{(t)}) + [W^{(t)}W^{(s)}\mathcal{D}^{(s)} - [I - W^{(t)^2}]\mathcal{D}^{(t)}]\delta x + \Psi(x)\delta w^{(t)}\ .$$

The linearized system becomes

$$\begin{bmatrix} A^TA & \lambda \mathcal{D}^{(s)^T} & \lambda \mathcal{D}^{(t)^T} \\ \Theta^{(s)} & \Psi & 0 \\ \Theta^{(t)} & 0 & \Psi \end{bmatrix} \begin{bmatrix} \delta x \\ \delta w^{(s)} \\ \delta w^{(t)} \end{bmatrix} = - \begin{bmatrix} T'(x, w^{(s)}, w^{(t)}) \\ \mathcal{G}^{(s)}(x, w^{(s)}) \\ \mathcal{G}^{(t)}(x, w^{(t)}) \end{bmatrix}, \tag{5.43}$$

where

$$\Theta^{(s)} = -\left(I - W^{(s)^2}\right)\mathcal{D}^{(s)} + W^{(s)}W^{(t)}\mathcal{D}^{(t)}$$

and

$$\Theta^{(t)} = -\left(I - W^{(t)^2}\right)\mathcal{D}^{(t)} + W^{(t)}W^{(s)}\mathcal{D}^{(s)}\ .$$

Again we want to avoid doing the matrix–matrix multiplication A^TA explicitly, so this system is rewritten in exactly the same way, as (5.22) was rewritten to (5.26).

Again we introduce $z = A\delta x$ and obtain the system

$$\begin{bmatrix} -I & A & 0 & 0 \\ A^T & 0 & \lambda \mathcal{D}^{(s)^T} & \lambda \mathcal{D}^{(t)^T} \\ 0 & \Theta^{(s)} & \Psi & 0 \\ 0 & \Theta^{(t)} & 0 & \Psi \end{bmatrix} \begin{bmatrix} z \\ \delta x \\ \delta w^{(s)} \\ \delta w^{(t)} \end{bmatrix} = - \begin{bmatrix} 0 \\ T'(x, w^{(s)}, w^{(t)}) \\ \mathcal{G}^{(s)}(x, w^{(s)}) \\ \mathcal{G}^{(t)}(x, w^{(t)}) \end{bmatrix}. \tag{5.44}$$

This coefficient matrix is non–symmetric. By rewriting this we may obtain another system with a symmetric coefficient matrix. How to do this, we will show how in the following.

Denoting the coefficient matrix in the above system Φ, we can divide the matrix into four sub blocks, where

$$\Phi_{11} = \begin{bmatrix} -I & A \\ A^T & 0 \end{bmatrix}, \Phi_{12} = \begin{bmatrix} 0 & 0 \\ \lambda \mathcal{D}^{(s)^T} & \lambda \mathcal{D}^{(t)^T} \end{bmatrix},$$

$$\Phi_{21} = \begin{bmatrix} 0 & \Theta^{(s)} \\ 0 & \Theta^{(t)} \end{bmatrix} \text{ and } \Phi_{22} = \begin{bmatrix} \Psi & 0 \\ 0 & \Psi \end{bmatrix}.$$

Then we form the Schur–complement $A^{\langle \text{aug} \rangle}$ of Φ_{11} in Φ

$$\begin{aligned} A^{\langle \text{aug} \rangle} &= \Phi_{11} - \Phi_{12}\Phi_{22}^{-1}\Phi_{21} \\ &= \begin{bmatrix} -I & A \\ A^T & 0 \end{bmatrix} - \begin{bmatrix} 0 & 0 \\ \lambda \mathcal{D}^{(s)^T} & \lambda \mathcal{D}^{(t)^T} \end{bmatrix} \begin{bmatrix} \Psi^{-1} & 0 \\ 0 & \Psi^{-1} \end{bmatrix} \begin{bmatrix} 0 & \Theta^{(t)} \\ 0 & \Theta^{(s)} \end{bmatrix} \\ &= \begin{bmatrix} -I & A \\ A^T & -\lambda \mathcal{D}^{(s)^T}\Psi^{-1}\Theta^{(s)} - \lambda \mathcal{D}^{(t)^T}\Psi^{-1}\Theta^{(t)} \end{bmatrix}. \end{aligned}$$

It is seen that if $A_{22}^{\langle \text{aug} \rangle}$ is a symmetric block, then the complete Schur–complement $A^{\langle \text{aug} \rangle}$ is a symmetric matrix. Inserting the values of $\Theta^{(s)}$ and $\Theta^{(t)}$ we obtain

$$\begin{aligned} A_{22}^{\langle \text{aug} \rangle} &= -\lambda \mathcal{D}^{(s)^T}\Psi^{-1}\Theta^{(s)} - \lambda \mathcal{D}^{(t)^T}\Psi^{-1}\Theta^{(t)} \\ &= -\lambda \mathcal{D}^{(s)^T}\Psi^{-1}\left[-\left(I - W^{(s)^2}\right)\mathcal{D}^{(s)} + W^{(s)}W^{(t)}\mathcal{D}^{(t)}\right] \\ &\quad -\lambda \mathcal{D}^{(t)^T}\Psi^{-1}\left[-\left(I - W^{(t)^2}\right)\mathcal{D}^{(t)} + W^{(t)}W^{(s)}\mathcal{D}^{(s)}\right] \\ &= \lambda \mathcal{D}^{(s)^T}\Psi^{-1}\left[I - W^{(s)^2}\right]\mathcal{D}^{(s)} - \lambda \mathcal{D}^{(s)^T}\Psi^{-1}W^{(s)}W^{(t)}\mathcal{D}^{(t)} \\ &\quad +\lambda \mathcal{D}^{(t)^T}\Psi^{-1}\left[I - W^{(t)^2}\right]\mathcal{D}^{(t)} - \lambda \mathcal{D}^{(t)^T}\Psi^{-1}W^{(t)}W^{(s)}\mathcal{D}^{(s)} \qquad (5.45) \\ &= \lambda \begin{bmatrix} \mathcal{D}^{(s)^T} & \mathcal{D}^{(t)^T} \end{bmatrix} \begin{bmatrix} \Psi^{-1} & 0 \\ 0 & \Psi^{-1} \end{bmatrix} \begin{bmatrix} I - W^{(s)^2} & W^{(s)}W^{(t)} \\ I - W^{(t)^2} & W^{(t)}W^{(s)} \end{bmatrix} \begin{bmatrix} \mathcal{D}^{(s)} \\ \mathcal{D}^{(t)} \end{bmatrix}. \end{aligned}$$

In (5.45) the first and the third terms are symmetric matrices, since they are products of diagonal matrices between a matrix and the same matrix transposed. The sum of the second and the fourth terms form a symmetric term as well. The reason for this is that the sum of a matrix C and the same matrix transposed C^T, will be a symmetric matrix. Hence we just have to show, that the second term actually is the forth term transposed, i.e., (we neglect the constant λ)

$$\begin{aligned} \mathcal{D}^{(s)^T}\Psi^{-1}W^{(s)}W^{(t)}\mathcal{D}^{(t)} &= \mathcal{D}^{(s)^T}\Psi^{-1}W^{(t)}W^{(s)}\mathcal{D}^{(t)} \\ &= \left(\mathcal{D}^{(t)^T}\Psi^{-1}W^{(t)}W^{(s)}\mathcal{D}^{(s)}\right)^T . \end{aligned}$$

Notice that we can allow the interchanging of the inner matrices since these are diagonal. The result is that $A^{\langle\text{aug}\rangle}$ indeed is a symmetric matrix, which means that the update of the step can be determined by means of an iterative method taking advantage of the symmetry, e.g. Minres.

In order to set up the total system after eliminating the variables $\delta w^{(s)}$ and $\delta w^{(t)}$ we calculate the new right hand side, which is given by

$$\begin{aligned} b^{\langle\text{aug}\rangle} &= \begin{bmatrix} 0 \\ -T'(x,w^{(s)},w^{(t)}) \end{bmatrix} + \begin{bmatrix} 0 & 0 \\ \lambda\mathcal{D}^{(s)^T} & \lambda\mathcal{D}^{(t)^T} \end{bmatrix} \begin{bmatrix} \Psi^{-1} & 0 \\ 0 & \Psi^{-1} \end{bmatrix} \begin{bmatrix} \mathcal{G}^{(s)}(x,w^{(s)}) \\ \mathcal{G}^{(t)}(x,w^{(t)}) \end{bmatrix} \\ &= \begin{bmatrix} 0 \\ -T'(x,w^{(s)},w^{(t)}) + \lambda\mathcal{D}^{(s)^T}\Psi^{-1}\mathcal{G}^{(s)}(x,w^{(s)}) + \lambda\mathcal{D}^{(t)^T}\Psi^{-1}\mathcal{G}^{(t)}(x,w^{(t)}) \end{bmatrix} \\ &= \begin{bmatrix} 0 \\ -T'(x) \end{bmatrix} , \end{aligned} \tag{5.46}$$

where

$$T'(x) = A^T(Ax-b) + \lambda\mathcal{D}^{(s)^T}\Psi^{-1}\mathcal{D}^{(s)}x + \lambda\mathcal{D}^{(t)^T}\Psi^{-1}\mathcal{D}^{(t)}x .$$

The system to solve to find the step is thus given by

$$\begin{aligned} &\begin{bmatrix} -I & A \\ A^T & A_{22}^{\langle\text{aug}\rangle} \end{bmatrix} \begin{bmatrix} z \\ \delta x \end{bmatrix} = \begin{bmatrix} 0 \\ -T'(x) \end{bmatrix} \\ \Leftrightarrow \quad & A^{\langle\text{aug}\rangle}x^{\langle\text{aug}\rangle} = b^{\langle\text{aug}\rangle} \end{aligned} \tag{5.47}$$

where $A_{22}^{\langle\text{aug}\rangle}$ is given in (5.45). Note the similarity with the one–dimensional case (5.29) – the only difference is that $A_{22}^{\langle\text{aug}\rangle}$ consists of more terms.

Having determined the unknown variables $x^{\langle\mathrm{aug}\rangle}$, these are inserted in one of the two block equations to calculate the values of the two last variables $\delta w^{(s)}$ and $\delta w^{(t)}$. These are given by

$$\begin{bmatrix} \delta w^{(s)} \\ \delta w^{(t)} \end{bmatrix} = \begin{bmatrix} \Psi^{-1} & 0 \\ 0 & \Psi^{-1} \end{bmatrix} \left(- \begin{bmatrix} \mathcal{G}^{(s)}(x, w^{(s)}) \\ \mathcal{G}^{(t)}(x, w^{(t)}) \end{bmatrix} - \begin{bmatrix} \Theta^{(s)}\delta x \\ \Theta^{(t)}\delta x \end{bmatrix} \right) ,$$

i.e.,

$$\begin{aligned} \delta w^{(s)} &= -\Psi^{-1}[\mathcal{G}^{(s)}(x, w^{(s)}) + \Theta^{(s)}\delta x] \\ &= -w^{(s)} + \Psi^{-1}\left[D^{(s)}x - \left[W^{(s)}W^{(t)}\mathcal{D}^{(t)} - \left(I - W^{(s)^2}\right)\mathcal{D}^{(s)}\right]\delta x\right] \end{aligned} \tag{5.48}$$

and

$$\begin{aligned} \delta w^{(t)} &= -\Psi^{-1}[\mathcal{G}^{(t)}(x, w^{(t)}) + \Theta^{(t)}\delta x] \\ &= -w^{(t)} + \Psi^{-1}\left[D^{(t)}x - \left[W^{(t)}W^{(s)}\mathcal{D}^{(s)} - \left(I - W^{(t)^2}\right)\mathcal{D}^{(t)}\right]\delta x\right] \end{aligned} \tag{5.49}$$

In the implementation of the algorithm we solve the system (5.29) instead of the more numerically unstable system (5.43).

The conclusion of this section is that we have been able to derive a system with a symmetric coefficient matrix. In general the symmetric matrices are far to prefer from non–symmetric, since the methods we use to solve the problem are much more reliable than methods that works on general matrices.

Neither [5] nor [20] discover the possibility of rewriting the non–symmetric system to be symmetric. Instead they overcome the difficulty by using a symmetrization, i.e., they replace the existing coefficient matrix, say $A^{\langle\mathrm{aug,nonsym}\rangle}$, with $\bar{A} = A^{\langle\mathrm{aug,nonsym}\rangle} + A^{\langle\mathrm{aug,nonsym}\rangle^T}$. In this way they become capable of applying the conjugate gradient method. The symmetrization may, however, influence the convergence of the algorithm.

5.5.1 Proof of Convexity

Like in the one dimensional case we like to verify that the Hessian matrix for the augmented system is positive (semi–)definite, and hereby prove that the objective function is convex.

Eliminating in the system (5.47), we obtain

$$\left[A^T A + A_{22}^{\langle\text{aug}\rangle}\right] \delta x = -T'(x) \; . \tag{5.50}$$

We solve the numerical stable system (5.29), in the sense that $A^T A$ is not computed explicitly, but to verify that the objective function is convex, we will use the equivalent system (5.50).

The positive (semi)–definiteness was shown in the one dimensional case in (5.25). Here we will show that the coefficient matrix in (5.50) is positive semi–definite as well. Since we have already shown that $A^T A$ is positive semi–definite, we need to show that this is also the case for $A_{22}^{\langle\text{aug}\rangle}$.

Again we use the definition of positive definiteness Def. 4.1.3 and multiply with a non–zero vector v and the same vector transposed from either side respectively.

Introducing $\bar{u}^{(s)} = \Phi^{-1/2}\mathcal{D}^{(s)}v$ and $\bar{u}^{(t)} = \Phi^{-1/2}\mathcal{D}^{(t)}v$, equation (5.45) is rewritten to

$$\begin{aligned} v^T A_{22}^{\langle\text{aug}\rangle} v &= \bar{u}^{(s)^T}\left[I - W^{(s)^2}\right]\bar{u}^{(s)} + \bar{u}^{(t)^T}\left[I - W^{(t)^2}\right]\bar{u}^{(t)} \\ &\quad -\bar{u}^{(s)^T} W^{(s)} W^{(t)} \bar{u}^{(t)} - \bar{u}^{(t)^T} W^{(s)} W^{(t)} \bar{u}^{(s)} \\ &= \bar{u}^{(s)^T}\bar{u}^{(s)} + \bar{u}^{(t)^T}\bar{u}^{(t)} - \left[\bar{u}^{(s)^T} W^{(s)^2}\bar{u}^{(s)} + \bar{u}^{(t)^T} W^{(t)^2}\bar{u}^{(t)}\right. \\ &\quad \left. +\bar{u}^{(s)^T} W^{(s)} W^{(t)} \bar{u}^{(t)} + \bar{u}^{(t)^T} W^{(s)} W^{(t)} \bar{u}^{(s)}\right] \end{aligned} \tag{5.51}$$

Setting $\bar{z}^{(s)} = W^{(s)}\bar{u}^{(s)}$ and $\bar{z}^{(t)} = W^{(t)}\bar{u}^{(t)}$ we can rewrite (5.51) to

$$\begin{aligned} & \|\bar{u}^{(s)}\|_2^2 + \|\bar{u}^{(t)}\|_2^2 - \left[\bar{z}^{(s)^T}\bar{z}^{(s)} + \bar{z}^{(t)^T}\bar{z}^{(t)} + \bar{z}^{(s)^T}\bar{z}^{(t)} + \bar{z}^{(t)^T}\bar{z}^{(s)}\right] \\ = \; & \|\bar{u}^{(s)}\|_2^2 + \|\bar{u}^{(t)}\|_2^2 - \|\bar{z}^{(s)} + \bar{z}^{(t)}\|_2^2 \\ = \; & \|\bar{u}^{(s)}\|_2^2 + \|\bar{u}^{(t)}\|_2^2 - \|W^{(s)}\bar{u}^{(s)} + W^{(t)}\bar{u}^{(t)}\|_2^2 \end{aligned} \tag{5.52}$$

The last term is limited upward in the following way (Cauchy–Schwarz)

$$\begin{aligned} \|W^{(s)}\bar{u}^{(s)} + W^{(t)}\bar{u}^{(t)}\|_2^2 &\leq \|W^{(s)}\bar{u}^{(s)}\|_2^2 + \|W^{(t)}\bar{u}^{(t)}\|_2^2 \\ &\leq \|\bar{u}^{(s)}\|_2^2 + \|\bar{u}^{(t)}\|_2^2 \; , \end{aligned}$$

where the last inequality is a consequence of that all the elements of $W^{(s)}$ and $W^{(t)}$ are limited to be in the interval $[-1; 1]$. Again for an explanation of this limitation see Sec. 6.5.

Thus in equation (5.52) we see that the last term is smaller than or equal to the two first terms we subtract it from, hence we can conclude that $v^T A_{22}^{\langle \mathrm{aug} \rangle} v \geq 0$, i.e., that $A_{22}^{\langle \mathrm{aug} \rangle}$ is positive semi–definite.

In a larger perspective this means that the coefficient matrix given in (5.50) is positive definite for arbitrary x. Thus when we solve the equivalent indefinite system (5.29), we determine a Newton direction, which according to Theorem 4.2.1 is in a descent direction.

Chapter Summary

In this chapter we have derived the formulaes needed in order to implement an algorithm that can solve the Tikhonov based optimization problem with a Total Variation functional as penalty term. We have derived the gradient and Hessian in the standard formulation as well as for the augmented system including the extra variable. The latter in both the one and two–dimensional cases. The outcome was rewritten to a symmetric and numerically stable system. Finally, we proved that the convexity of the objective function remains after introducing the extra variables.

Chapter 6

Algorithm – Implementation Aspects

In this chapter we describe and discuss implementation aspects of the algorithm. Here we describe all implementation aspects of the algorithm designed for solving two–dimensional problems. A one–dimensional version is implemented as well, but all implementation aspects of this are included in the two–dimensional implementation.

6.1 The Algorithm

The algorithm we will propose here is in many ways similar to the approaches given in [5] and [20]. Though the skeleton is more or less the same, i.e., the introduction of the new variable and the use of Newton's method, parts of the algorithm differ from the above mentioned approaches in several ways. The differences will be explained as they appear.

First we present the pseudo code, which is given in Tab. 6.1.

```
Initial guess $x^{[0]}$
Set $k := 0$
Calculate initial guesses on $w^{(s)}$ and $w^{(t)}$ by means of (5.36) and (5.37)
while not STOP do
  Solve symmetric Newton system (5.29) (for $\delta x$) by means of Minres
  Determine $\delta w^{(s)}$ and $\delta w^{(t)}$ by means of (5.48) and (5.49)
  Find $\tau$ from (6.5)
  $w^{(s),[k+1]} := w^{(s),[k]} + \tau\delta w^{(s),[k]}$ (Take step in $w^{(s)}$–space)
  $w^{(t),[k+1]} := w^{(t),[k]} + \tau\delta w^{(t),[k]}$ (Take step in $w^{(t)}$–space)
  Determine $\tau^{(x)}$ such that $T(x^{[k]} + \tau^{(x)}\delta x^{[k]}) < T(x^{[k]})$
  $x^{[k+1]} := x^{[k]} + \tau^{(x)}\delta x^{[k]}$ (Take step in original variable)
  $k := k + 1$
end
```

Table 6.1: *Pseudo code for Newton's method on the augmented system*

The notation $\delta w^{(s),[k]}$ means $\delta w^{(s)}$ at the kth iteration.

In the following sections we will explain the different aspects in the order as they appear in the pseudo code. This means we will first discuss input arguments and starting guesses and then focus on the determination of the step by means of Minres. Then we will discuss line search procedures in the different variables and finally discuss how to choose stopping criteria.

However, the implementation is carried out in the framework of M$\mathcal{OO}$Re Tools, which means that we need to present the basics of M$\mathcal{OO}$Re Tools as a first thing.

6.1.1 M$\mathcal{OO}$Re Tools

M$\mathcal{OO}$Re Tools is a product of a PhD–project by M. Jacobsen [16]. It is a revised and object–oriented version of the Matlab package Regularization Tools by P. C. Hansen [12]. M$\mathcal{OO}$Re Tools is meant for solving and investigating inverse problems and it is designed for solving large–scale problems.

During extensive use of M$\mathcal{OO}$Re Tools bugs have appeared. These bugs have been corrected in the official version of M$\mathcal{OO}$Re Tools.

6.1.2 The Basics of M𝒪𝒪Re Tools

M𝒪𝒪Re Tools is a large toolbox, and here we will only cover the basics.
M𝒪𝒪Re Tools implements a lot of classes each representing abstract versions of known quantities, such as matrices and vectors. The classes are build up in a hierarchy such that already implemented and tested code can be reused in subclasses. Examples of these classes are LinearOperator, Matrix, Vector, KroneckerProduct, Vector2D, IdentityOperator and NullOperator. Two basic classes are the Matrix and Vector classes. Objects of these classes acts as if they were standard matrices and vectors. An example is that the Matrix class implements an overloaded method that takes care of multiplication with a Vector object.

Another important class we will make heavily use of in the algorithm for the two–dimensional case is the KroneckerProduct2D. A KroneckerProduct2D object stores two LinearOperator objects (e.g. Matrix objects) instead of storing the explicit Kronecker product of the two operators. An overloaded method in the class takes care of multiplication with another object, e.g., a Vector2D. An example of this is given Sec. 6.2. This makes it possible to deal with much larger problems than with standard Matlab.

6.1.3 Input Arguments and Starting Guesses

The code for the implemented algorithm can be seen in Appendix A. In general a two–dimensional problem will be given in one of the two forms (2.6) (general dense matrix) or (2.9) (i.e., Kronecker product). The algorithm is implemented such that it deals with both forms in the most efficient way. The implementation given here assumes that the solution lies in a two dimensional grid with equidistant spacing. The algorithm can easily be modified to be used in cases, where this is not the case. A more general form than this is discussed in Sec. 6.8.

The parameter list of input arguments from the code is the following.

```
[X,extra] = TVMT2D(A,b,lambda,beta,x,opts,dim)

A       A Matrix or a KroneckerProduct2D
b       A Vector or a Vector2D
lambda  A double -- Regularization parameter,
beta    Parameter ensuring differentiability of J_TV
            (default: beta = 1e-6)
x0      Starting guess, a Vector or a Vector2D
            (default: X0 = Vector( (||b||_F)/prod(size(b))*ones(dim))
             i.e., with same "energy" as b.
dim     A standard vector [Nx Ny] that describes the dimensions
        of the solution domain. If X0 is a Vector2D, dim can be
        left out.
opts    A standard vector [MaxIter Tol]
```

When `A` is a KroneckerProduct2D the dimensions of the problem is revealed for the program. This information is not included if `A` is a Matrix the user must provide the dimensions of the problem as well.

The classes KroneckerProduct2D and Vector2D are closely related, just like Matrix and Vector makes a match. A KroneckerProduct2D can be multiplied onto a Vector2D and a Matrix onto a Vector, if the dimensions are correct. On the other hand these types can not be mixed, a Matrix can not be multiplied onto a Vector2D and as well as a KroneckerProduct2D can not be multiplied onto a Vector. This makes a problem, when one wants to make a flexible code, where no assumptions are made on the type of input objects.

In the two–dimensional case we want to represent e.g. the input image with a Vector2D, but if the kernel does not separate in the variables, it can not be represented with a KroneckerProduct2D. In this case the kernel is represented by a Matrix. In this case the VectorReshape comes in handy. This type of object is in principle a type casting object for vector objects, i.e., when multiplied onto a Vector2D it can return a Vector, a Vector3D or VectorND and vice versa. A VectorReshape object resembles the built–in Matlab command `reshape`, just on operator form.

In the pseudo code given in [20] the user can give a starting guess on the extra variables as well. An essential question is, if it makes sense to start the algorithm in an iterate where the connection between the original and the extra variables are observed. A reason to allow starting guesses could be for the extra variables is, e.g., if one should be able to restart the algorithm after a number of iterations. However, in the implementation this has been left out.

In the implemented algorithm in this work the user may only give a starting guess on the original variable, `X0`. From this the initial values of the extra variables are calculated from their definition, i.e., in the first iteration we set both $\mathcal{G}^{(s)}$ from (5.40) and $\mathcal{G}^{(s)}$ from (5.41) equal to zero, and hence compute the corresponding value of the extra variables.

This means that if the user gives a starting image that contains edges these edges will immediately be reflected in the extra variables.

6.2 The Derivative Operators

As described in Sec. 3.5 the derivative operators $D^{(s)}$ and $D^{(t)}$ can be formed using Kronecker products. Since the unknown quantity, the image, is a two–dimensional structure, we represent this with an object of type Vector2D. Thus the derivative operators that should be applied to the image, should be of type KroneckerProduct2D.

The function `getL` is a M$\mathcal{OO}$Re Tools built–in function that generates matrices representing forward approximations to derivatives on matrix form. Since we intend to use the central estimate described in Sec. 3.5.2.3, a new function `getcentralL` has been implemented. It returns a matrix on the form (3.16) as a SparseMatrix object with dimensions specified by the user. The code for `getcentralL` can be seen in App. A.1.

Assume that an image with dimensions $m \times n$ is stored in a Matlab variable `dataimage`, then we can represent the operator $D^{(s)}$ with an object of type KroneckerProduct2D in the following way.

```
>>X = Vector2D(dataimage);
>>[m n] = size(X);
>>Is = IdentityOperator(n);
>>D = getcentralL(m);
>>Ds = KroneckerProduct2D(Is,D);
```

Storing the objects as separated objects in the KroneckerProduct2D we save a lot of memory, instead of saving the explicit Kronecker product.

The outcome of the product `Ds*X` is again a Vector2D object. An important issue is, what M$\mathcal{OO}$Re Tools does behind the back of the user, when the multiplication `Ds*X` is carried out. Instead of forming the Kronecker product explicitly and apply this to the stacked vector, it retrieves the two terms `Is` and `D` and apply them using two matrix products, i.e., `D*X*Is'`.

This is just one of many advantages of using M$\mathcal{OO}$Re Tools. In the following section we will discuss the use of an iterative method and how fabulous the object–oriented toolbox is for treating structured problems.

6.3 Determination of the Step Using Minres

The greater part of the work of solving large–scale problems lies in solving the linear system of equations that gives the Newton direction in each iteration. Since the system can (and will) be very large in real–life applications, we have to solve the linear system of equations using an iterative method.

The reason for using an iterative method for this purpose is that it is often faster, less memory consuming and that the solution can be computed to a prescribed tolerance. This is not the case for the direct methods (e.g. methods based on LU–factorization).

The *conjugate gradient* method (CG) is an iterative method, which can be used for solving symmetric, positive definite systems. In our case the coefficient matrix is indefinite, thus we can not use CG. However, we can use Minres, which is a variant of the CG. Minres can be be applied successfully to symmetric, indefinite systems. Both CG and Minres belong to a class called Krylov subspace methods, which have the property that the calculated solutions lie in a Krylov subspace [2].

In general iterative methods only "touch" the coefficient matrix A via matrix–vector multiplication, and these methods are therefore well suited for sparse and structured matrices [14].

Note that we use the M$\mathcal{OO}$Re Tools version of Minres, since it must be able to treat objects.

6.4 M$\mathcal{OO}$Re Tools Implementation Issues

Here we will present how to represent the system we solve with objects from M$\mathcal{OO}$Re Tools.

The system to solve is given in (5.29), namely

λ	Double
I	IdentityOperator
$\mathcal{D}^{(s)}$	KroneckerProduct2D
$\mathcal{D}^{(t)}$	KroneckerProduct2D
$W^{(s)}$	DiagonalOperator
$W^{(t)}$	DiagonalOperator
Ψ	DiagonalOperator

Table 6.2: *Object types for the the quantities in $A_{22}^{\langle\text{aug}\rangle}$.*

$$\begin{bmatrix} -I & A \\ A^T & A_{22}^{\langle\text{aug}\rangle} \end{bmatrix} \begin{bmatrix} z \\ \delta x \end{bmatrix} = \begin{bmatrix} 0 \\ -T'(x) \end{bmatrix} ,$$

where

$$T'(x) = A^T(Ax - b) + \lambda \mathcal{D}^{(s)^T} \Psi^{-1} \mathcal{D}^{(s)} x + +\lambda \mathcal{D}^{(t)^T} \Psi^{-1} \mathcal{D}^{(t)} x$$

and

$$\begin{aligned} A_{22}^{\langle\text{aug}\rangle} &= \lambda \Big(\mathcal{D}^{(s)^T} \Psi^{-1} \left(I - W^{(s)^2} \right) \mathcal{D}^{(s)} - \mathcal{D}^{(s)^T} \Psi^{-1} W^{(s)} W^{(t)} \mathcal{D}^{(t)} \\ &\quad + \mathcal{D}^{(t)^T} \Psi^{-1} \left(I - W^{(t)^2} \right) \mathcal{D}^{(t)} - \mathcal{D}^{(t)^T} \Psi^{-1} W^{(t)} W^{(s)} \mathcal{D}^{(s)} \Big) \ . \end{aligned} \tag{6.1}$$

This system we want to represent by MOORE TOOLS objects. We start out by explaining how we can represent the complicated sum $A_{22}^{\langle\text{aug}\rangle}$ with objects. In Tab. 6.2 it is described which quantities will be represented by which type of objects.

Note that $W^{(s)}$, $W^{(t)}$ and Ψ are all DiagonalOperator objects, each encapsulating a Vector2D object.

In $A_{22}^{\langle\text{aug}\rangle}$ we have a sum of four terms, which all are products of different types of objects. When multiplying objects one can use another object to store the different parts of the product. This type of object is called an OperatorProduct, which can store the different factors of a product without doing the multiplication explicitly. This is especially smart, when each of the factors are sparse or have a special structure and the explicit product will be dense or unstructured. This we can illustrate using the first term of (6.1) as an example.

The first term is given by $\mathcal{D}^{(s)^T} \Psi^{-1}(I - W^{(s)^2})\mathcal{D}^{(s)}$. Since the three objects in the middle $\Psi^{-1}(I - W^{(s)^2})$ are either objects of type DiagonalOperator or IdentityOperator, these types of objects are compatible. This means that the product of these will be a DiagonalOperator, and we can store the product as one DiagonalOperator object. On the other hand the complete product (inclusive the KroneckerProduct2D's) will be stored as a dense Matrix object, since this special type of product does not have a sparse or structured representation. This is the reason why it is optimal to store the three objects separately.

Another container object is OperatorSum, which is similar to an OperatorProduct, but instead it stores the different terms of a sum of objects.

How we represent $A_{22}^{\langle\mathrm{aug}\rangle}$ in terms of M$\mathcal{OO}$Re Tools objects is visualized in Fig. 6.1.

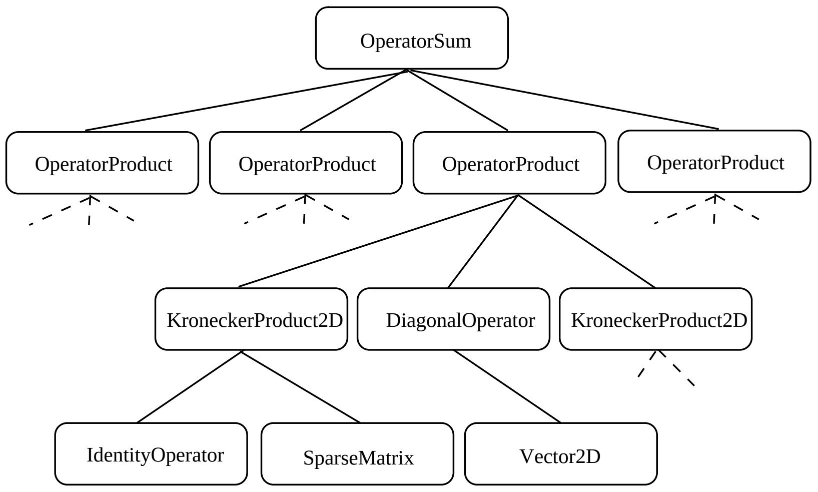


Figure 6.1: *Object representation of $A_{22}^{\langle\mathrm{aug}\rangle}$. All OperatorProducts in the second level are similar, which is also the case for the KroneckerProduct2D objects in the third level.*

The last two important object types are the OperatorArray and VectorStack. OperatorArray is, as the name hints at, a container for storing operators (objects from classes inheriting from the LinearOperator class) in an array. In principle it acts as a matrix with objects as elements. VectorStack is similar to an OperatorArray, but instead of LinearOperator objects it contains different types of Vectors (i.e. objects from classes that inheriting from the Vector class). These object types are needed to represent the augmented system, which is visualized in Fig. 6.2. The coefficient matrix is represented using an OperatorArray and the solution and the right hand side with VectorStack objects.

As earlier stated the Minres algorithm needs to multiply the coefficient matrix onto the right hand side. The question is how M$\mathcal{OO}$Re Tools takes care of the multipli-

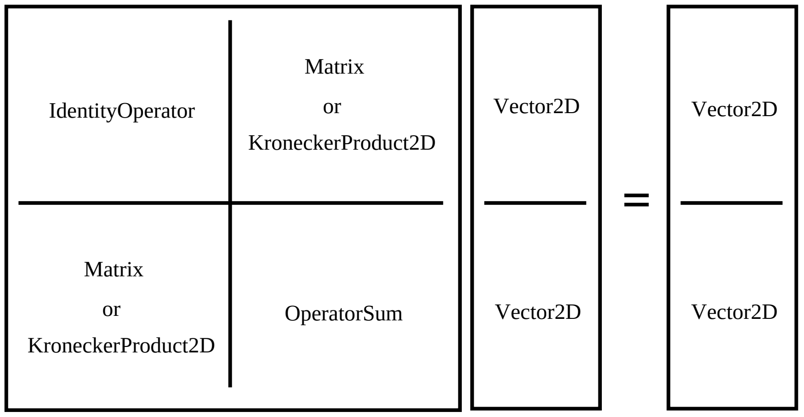


Figure 6.2: *The augmented $A^{\langle\mathrm{aug}\rangle}x^{\langle\mathrm{aug}\rangle} = b^{\langle\mathrm{aug}\rangle}$ system represented with an OperatorArray and VectorStack objects. Both types contain different types of objects.*

cation of the different objects in the OperatorArray with other objects.

An event that happens in each iteration of Minres, is when $A_{22}^{\langle\mathrm{aug}\rangle}$ is applied to a Vector2D from the right hand side. Caused by the linearity the multiplication of $A_{22}^{\langle\mathrm{aug}\rangle}$ with a Vector2D can be done by multiplying each term and factor in $A_{22}^{\langle\mathrm{aug}\rangle}$ onto the Vector2D and then afterwards, the summation can be carried out. This means that instead of multiplying with a huge, dense object once, the equivalent multiplication can be carried out using a number of much cheaper and less memory consuming operations.

A closer look at Fig. 6.1 reveals that $A_{22}^{\langle\mathrm{aug}\rangle}$ contains 20 objects at the bottom level (they are not all drawn). All these objects must be applied to the Vector2D. However, many of these multiplications are very cheap. Eight of the multiplications are with IdentityOperators, operations which are (almost) for free. The reason is that the member method taking care of multiplication in the IdentityOperator class of M𝒪𝒪Re Tools is executed. This method simply returns the multiplied object it self – no multiplication is carried out. Another eight of the products are with very sparse matrices and the last four are with DiagonalOperators. In each of the operations the corresponding multiplication method in the given class is executed, and thus the cheapest way of doing the multiplication is obtained.

As a nice side effect of the object–oriented approach the Minres returns an object of type VectorStack, which contains two Vector2Ds each representing z and δx. This means that when we apply the algorithm for image deblurring, the step it self is an image with same dimensions as the blurred image.

6.4.1 The Cost of The Augmented System

An essential question is, if there is an extra cost associated with the introduction of the extra variables. We discuss this looking at the one–dimensional case only, because it is the simplest case, and because the conclusion, we can draw, will hold for higher dimensional problems as well.

In standard Newton, the step is determined by solving the system from Sec. 5.1

$$\left[A^T A + \lambda \left[D^T \Psi^{-1} D - D^T \tilde{U}^2 \Psi^{-3} D\right]\right] \delta x = -T'(x) \ .$$

The system we compare to the system obtained in (5.23), namely

$$\left[A^T A + \lambda [D^T \Psi^{-1} D - D^T W^2 \Psi^{-1} D]\right] \delta x = -T'(x) \ .$$

Thus we see that the systems are very similar, and the work for solving these two distinct forms is more or less the same. Note that the original system can be obtained from the augmented system by inserting the definition of $W = \Psi^{-1}\tilde{U}$. An extra cost connected to the new formulation is that we must determine the step direction for the extra variable, i.e., $\delta w = -w + \Psi^{-1}Dx + [I - W^2]\Psi^{-1}D\delta x$. This calculation is quite inexpensive compared to the solving process of the above system.

However, the small amount of extra work in each iteration we gladly pay to obtain a faster convergence.

6.5 Step Controlling for the Extra Variables

To ensure the global convergence of the algorithm we need to impose a step–controlling mechanism on the extra variables $w^{(s)}$ and $w^{(t)}$.

At the solution we have that $w_i^{(s)} = \psi_i^{-1}[D^{(s)}x]_i$ and $w_i^{(t)} = \psi_i^{-1}[D^{(t)}x]_i$ and thus

$$\begin{aligned} w_i^{(s)^2} + w_i^{(t)^2} &= \frac{[D^{(s)}x]_i^2}{\psi_i^2} + \frac{[D^{(t)}x]_i^2}{\psi_i^2} \\ &= \frac{[D^{(s)}x]_i^2 + [D^{(t)}x]_i^2}{\psi_i^2} \\ &= \frac{\psi_i^2 - \beta^2}{\psi_i^2} \\ &= 1 - \frac{\beta^2}{\psi_i^2} \ , \end{aligned}$$

where the second–last equality follows from the definition of ψ; recall that ψ_i is the length of the gradient in the ith coordinate. Clearly in the limit for $\beta \to 0$, we have that the upper bound should be 1.

We see that if a temporary solution at a given iterate should be a possible solution, we must demand that the necessary conditions

$$w_i^{(s)^2} \leq 1 \ \forall \ i \ \text{ and } \ w_i^{(t)^2} \leq 1 \ \forall \ i \ .$$

are fulfilled. These conditions are equivalent to

$$|w_i^{(s)}| \leq 1 \ \forall \ i \ \text{ and } \ |w_i^{(t)}| \leq 1 \ \forall \ i \ . \tag{6.2}$$

This condition we impose on the extra variables. In principle $w^{(s)}$ and $w^{(t)}$ are free variables, but the condition illustrates that the algorithm is allowed only to iterate in "domains" where the solution may exist. There exist methods that allows an iterate, which is not a feasible solution. In some cases there exist a path through non–feasible iterates that leads to the solution faster. But this is out of the scope of this work to discuss, see e.g. [18, p. 47].

Another very important reason to make sure that Eq. (6.2) is fulfilled is that the block matrix $A_{22}^{\langle\text{aug}\rangle}$ in Eq. (5.29) is guaranteed to be non–singular and positive definite (when also $\lambda > 0$).

Different approaches are given in the literature for determining the step length. In [5] it is proposed to use the following approach to control the steps. For the s–direction we have

$$\tau^{(s)} = \rho \sup_{\tau} \bigl\{ |w_i^{(s)} + \tau \delta w_i^{(s)}| < 1 \forall \ i \bigr\} \quad , \quad \rho \in [0;1] \tag{6.3}$$

and similar in the t–driection

$$\tau^{(t)} = \rho \sup_{\tau} \bigl\{ |w_i^{(t)} + \tau \delta w_i^{(t)}| < 1 \forall \ i \bigr\} \quad , \quad \rho \in [0;1] \ , \tag{6.4}$$

where it is assumed that $|w_i^{(t)}| < 1 \forall\ i$ and $|w_i^{(s)}| < 1 \forall\ i$. The size of $\tau^{(s)}$ and $\tau^{(t)}$ is limited by ρ only. The authors of [5] propose a heuristic, where they update β and ρ. The basic idea is that they allow steps larger than the full Newton step, because the model is more well–behaved for a large β.

The approach in [20] is different in several ways. First of all it is proposed to use the same τ in the two directions, i.e., $\tau = \tau^{(s)} = \tau^{(t)}$. This means basically that we take Newton step in the extra variables. Secondly there is a limit on τ, namely that $\tau \leq 1$. This means that a full Newton step in the extra variables is only taken if the variables stays in the limited interval. Thus the proposed approach in [20] is

$$\tau = \max\left\{\sup_{\tau}\{|w_i^{(s)} + \tau\delta w_i^{(s)}| < 1, |w_i^{(t)} + \tau\delta w_i^{(t)}| < 1 \forall\ i\}, 1\right\} \ . \tag{6.5}$$

In the implemented algorithm we will use this approach.

6.6 Line search Procedure for the Original Variable

Using the formulation with the extra variables, [20] concludes that a line search procedure in the objective function T is in general not needed. Experiments show that a line search in the original variable often returns unit step length. The reason to this is the global more linear behavior of the original function. In [5] the authors make use of line search and associate the line search with a controlling of the β parameter as well.

However, to ensure convergence, we introduce a line search procedure in the original variable also. This is inexpensive, because the line search procedure will only be triggered if necessary. We check that the function value in the new iterate is less than in the old iterate, thus the cost of the line search is not needed "only" one function evaluation is wasted.

In the implementation we use a very primitive approach of a line search. The simple and naive approach is to simply to continue to halve the step until a lower object value is obtained. If a step is not taken within a relatively few number of iterations, say 30, we stop the line search and force a new outer iteration. Because we have taken steps in the variables the global system has changed, which means that we might be able to retrieve a better search direction in the next iteration. If the line search in general is needed one should consider a better approach using gradient information as well. This is easily incorporated, and the gradient is already determined as a part of the algorithm.

An important thing to notice is that the implementation builds on Newton's method, and the linear system we solve results in a Newton direction. However, because we use different length of steps in the respective variables, the overall step is not a Newton step. In Fig. 6.3 the taken step is illustrated for a one–dimensional problem. We see that the step taken is *not* a Newton step, because the length of the step taken in the respective variables are not the same. Another thing the figure illustrates is that the extra variable expands the solution space.

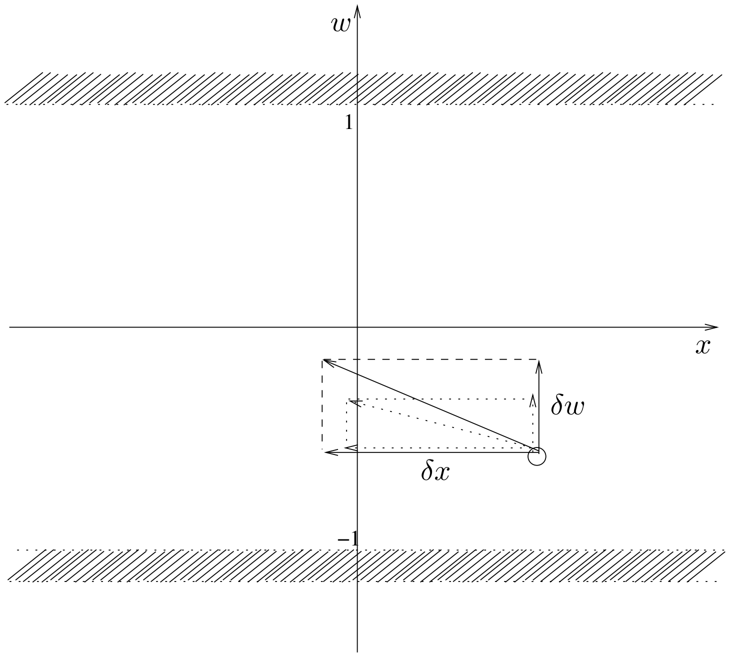


Figure 6.3: *The step is, in general, not a Newton step.*

Since it is not a Newton step, can we be sure that the step is downhill. In fact we can, because the objective function is convex.

6.7 Stopping Criteria

In this section we will discuss which stopping criteria is used in the implementation of the algorithm. In any algorithm it is very important to use and implement relevant stopping criteria, which ensures that the algorithm stops, when an iterate in some sense is close to the solution. However, no stopping criteria can guarantee that a given iterate is close to the minimizer.

We will discuss the relation between the stopping criteria on the outer Newton iterations and the interior stopping criteria, which must be given to Minres.

6.7.1 Newton Stopping Criteria

As outer stopping criteria we have chosen the following:

1. The classical safety break. The algorithm stops after a number of iterations chosen by the user.

2. A combination of a small step and a small decrease in the objective value

The second item we can illustrate mathematically

$$\|x^{[k]} - x^{[k-1]}\| < \epsilon_1$$

and

$$\|f\left(x^{[k]}\right) - f\left(x^{[k-1]}\right)\| < \epsilon_2 \ .$$

These criteria are very general and often used in descent methods. We will not discuss these stopping criteria further here. For more on this subject see e.g. [9].

6.7.2 Minres Stopping Criteria

The system (5.29) is solved by means of Minres an approximate direction for the step is computed. Clearly how close it is to the correct Newton direction depends on how the stopping criteria for Minres are chosen. We want that Minres calculates the direction with a "suitable" precision. The question is, what suitable means in this context.

If the system is solved using a direct method, we are guaranteed that the direction is downhill. This is a results of the proof in Sec. 5.5.1. This is not the case when the system with respect to a given tolerance. E.g., we might choose to solve the system with a stopping criterion, such that the relative error on the computed direction is at most 10^{-6}. The question is if we can we be sure that the direction is downhill when it is determined by an iterative method in this way.

The subject of determining an approximate Newton direction is known in the literature as *truncated Newton* or *inexact Newton*, see e.g.[10].

The higher precision, the more it will cost to compute the given direction. On the other hand having a good approximation on the Newton step, we can hope that we need fewer outer Newton iterations to get to minimum. When we are discussing the work, the algorithm does, we have to count the total number of inner iterations.

We have chosen that the stopping criterion for Minres is an order of magnitude lower than the outer Newton stopping criteria. This means that if the user chooses the stopping criteria for the outer Newton iteration associated with a small step equal to 10^{-6}, then the stopping criterion for corresponding Minres criterion is set to 10^{-7}. Note that even if the criteria are not directly related, we can still use this approach. Experiments should clarify if this is good approach.

6.8 Modular Implementation

A very nice feature of the object–oriented implementation of the algorithm is that it can used for a wider class of problems. The algorithm is not limited to solve problems represented by matrices and vectors only. This, however, requires a few modifications, which are that the operators that differentiates must be input arguments supplied by the user. Hence the lines in the code where the derivative operators are computed must be removed.

One can imagine problems, where A, x and b are abstract quantities (objects) that represent the underlying continuous system in some sense. An example of this could be, if the system is represented by finite element model (FEM). This basically means that the solution does not live in an equidistant grid, but maybe as scattered points on a surface. The A object should then map x onto b.

Besides the classes of A, x and b one need to implement new classes for the derivative operators. In these classes `sub_applytovector` method should be implemented to define the operation of differentiation. This means that whenever an object of the the new derivative operator class is multiplied with an object of, say, x then the product should return a representation of the derivative of x in the given direction.

Chapter Summary

In this chapter we discussed how the algorithm is implemented using objects form M𝒪𝒪Re Tools. We have described step controlling of the extra variable, line search for the original variable and stopping criteria for the algorithm. Finally, we have described that the algorithm because of the modular structure can be used for a wider class of problems.

CHAPTER 7

Experiments in 2D

In this chapter we make some experiments with the implemented algorithm. First of all we want to verify that it works properly. Secondly we look at interesting aspects of the implementation.

7.1 General Remarks About the Tests

All the tests are based on reconstructing images, where the exact solution is known. The blurred image is calculated by means of a forward model. As blurring model we use the atmospheric turbulence, which is described in Sec. 2.3.2. In all test examples we have added white noise, such that $\|e\|/\|b_{\text{exact}}\| = 10^{-6}$.

The parameter β is kept fixed equal to 10^{-6}.

One should notice that it is hard to verify that the algorithm returns the correct result(s). The reason is that the calculated solution is not equal to the true solution. The algorithm converges to the Tikhonov Total Variation solution. Another reason is that we have had no method or implementation to compare with.

7.2 A Simple Box

Here we will try to reconstruct a blurred image using the algorithm. The true image is a box centered in the image, thus it contains sharp edges. Furthermore the image is constructed of piecewise constant planes. In the two–dimensional case the TV in case is simply the length of the edges of the box multiplied with the height. Thus we expect the box to have a relative small TV measure.

The box image and the blurred image are visualized in Fig. 7.1.

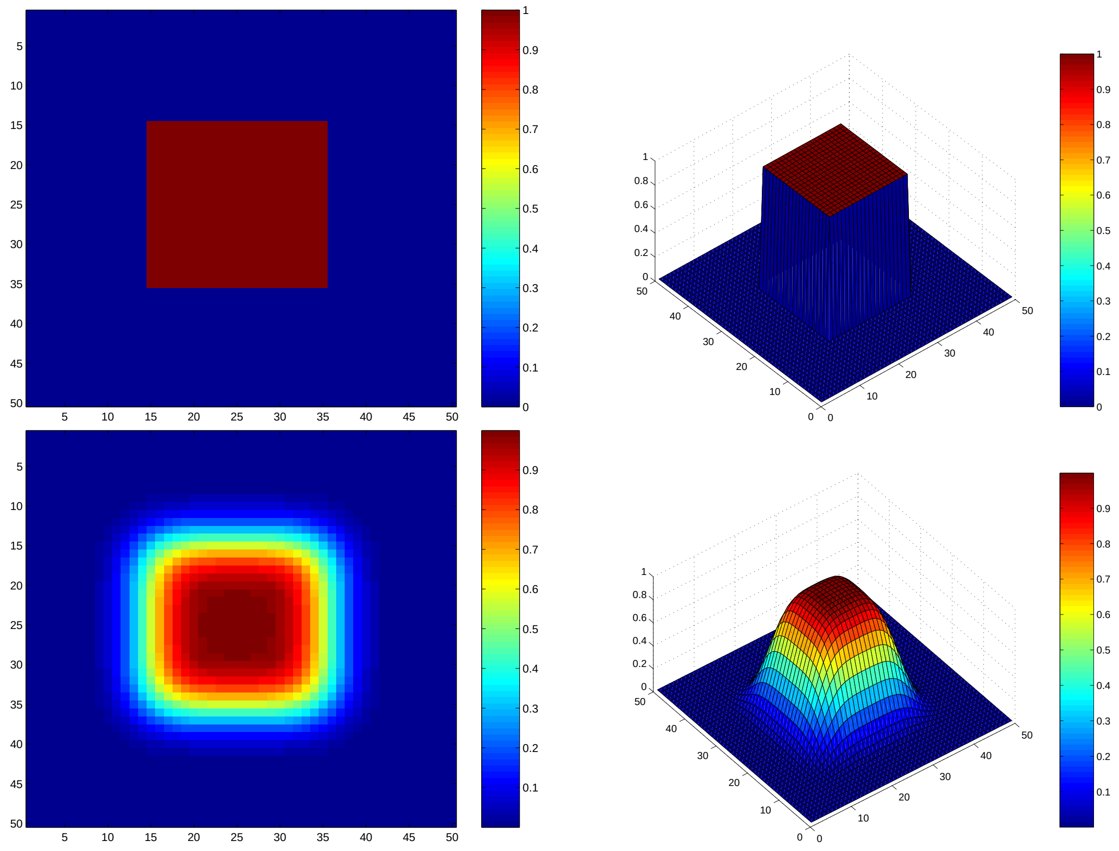
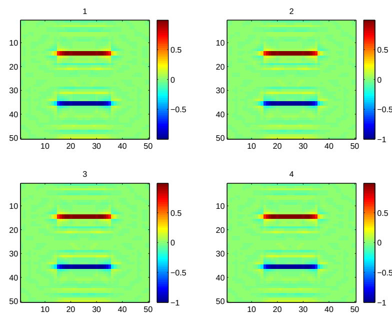


Figure 7.1: *The box test image and the blurred version*

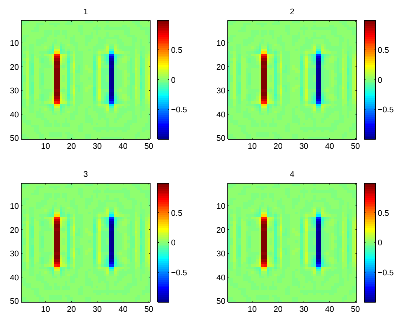
In the blurring process we have used $\sigma = \bar{\sigma} = 3$ (i.e., the standard deviation of the Gaussian point–spread function) and the bandwidth of the blurring matrices equal to 14. This results in that the blurring operator has a condition number equal to

$5.6 \cdot 10^{18}$. Here we try to reconstruct the true image from the blurred image.

The starting guess is chosen as the zero solution. During the first four iterations we examine what the extra variables look like. These are visualized in Fig. 7.2.



(a) The extra variable $w^{(s)}$



(b) The extra variable $w^{(t)}$

Figure 7.2: *The extra variables for the first four iterations.*

We see that the extra variables already in the first iteration have catched the edges of the image. The best calculated solution is visualized in Fig. 7.3.

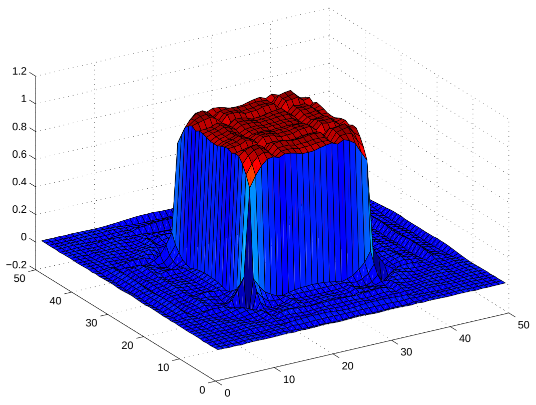


Figure 7.3: *Best reconstructed solution.*

We see that the edges really are allowed. This experiment just substantiates the experiments done in the one–dimensional case.

7.3 Two Boxes with Different Values

Another test image is shown together with the same image blurred in Fig. 7.4. It resembles the first test image, but it contains two boxes with different values.

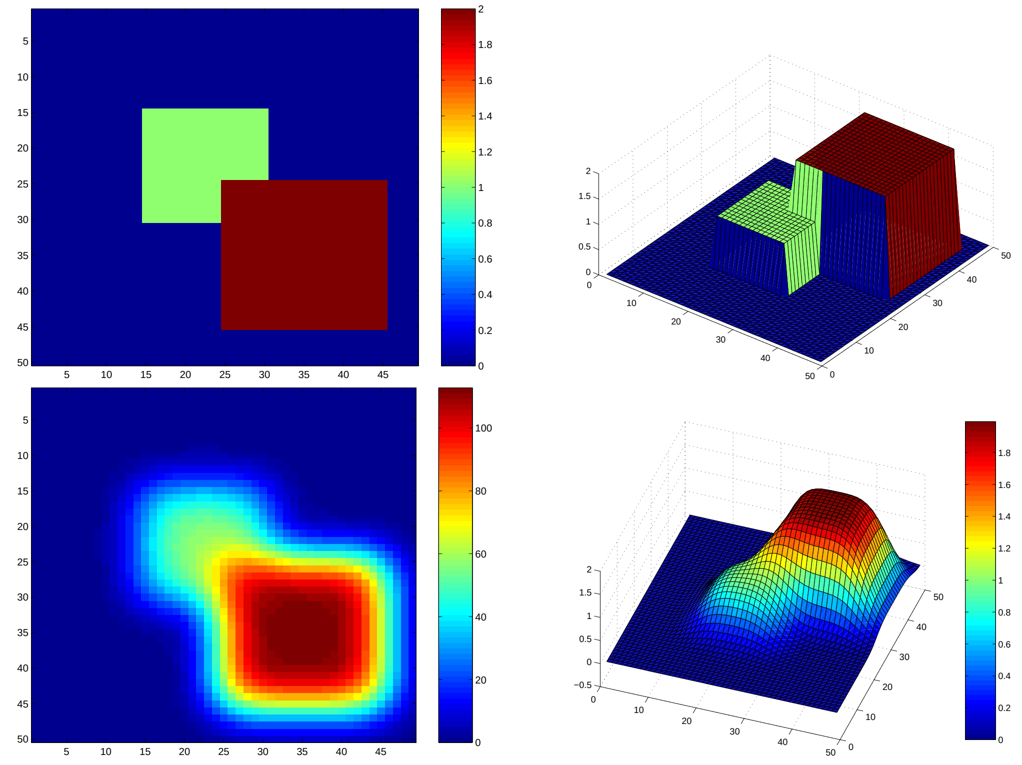
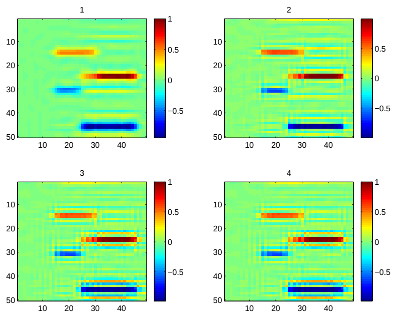


Figure 7.4: *The test image with the two boxes and the same blurred image.*

Again we inspect the extra variables in the first four iterations. They are visualized in Fig. 7.5, and again we see that already in the first iterations the edges are captured.


(a) The extra variable $w^{(s)}$

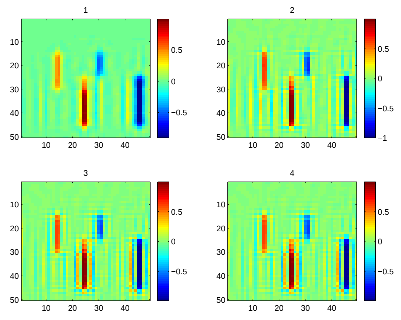

(b) The extra variable $w^{(t)}$

Figure 7.5: *The extra variables in the first four iterations.*

What happens if we use one of the discretizations of the Total Variation functional, where data were removed? We concluded that a high frequent behavior is expected in one corner point, if the approach given in Sec. 3.5.2.1 is used. A relevant question is how much influence this single point can have on the solution. In Fig. 7.6 a solution using the described approach is visualized. We see that a giant negative spike actually occurs in the corner.

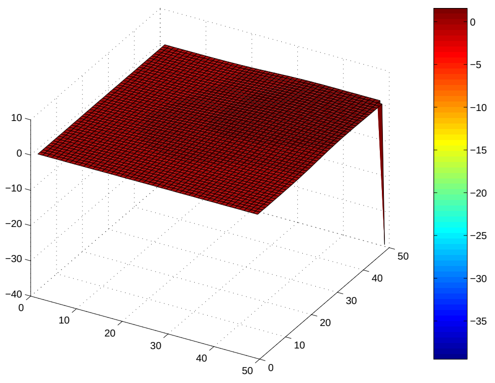


Figure 7.6: *A solution calculated by means of a bad discretization of TV.*

The problem is that this single number is significantly different from the other data, and hence it changes the norm of the given solution dramatically. Setting the value of the pixel with the spike equal to zero, we obtain the solution, which can be seen in Fig. 7.7.

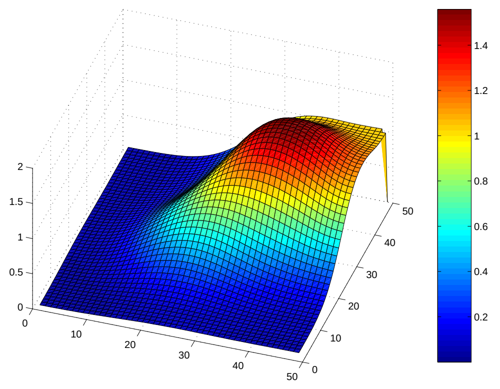


Figure 7.7: *A TV solution where the pixel in the corner is set to zero.*

We see that by canceling the spike out, we are actually able to obtain a solution, which is not that bad. However, the value of one pixel can change the value of the norm and hence it influences the solution dramatically. In this case we believe is a struck of pure luck that we obtain an acceptable solution.

However, if we use the discretization proposed in Chap. 3, we obtain a much better solution. The solution is not shown here.

7.4 A "Real" Test Image

A final experiment is to see how the algorithm works when the true image contains edges as well as smooth sections. The test image is the well–known "Lena" image, which is of dimensions are 222×208. It is visualized in Fig. 7.8.

From the theory we deduced on Total Variation, we expect that it allows smooth solutions as well as solutions with steep gradients. So in this case TV should be very good.

Again we blur the image, this time with $\sigma = \bar{\sigma} = 1.7$ and try to reconstruct this blurred image. The condition is fairly large with an order of magnitude 10^{13}. This is an example of large–scale problem. The coefficient matrix, i.e., the Kronecker product has the dimensions 46176×46176 and if we really tried to form this as a dense matrix, we could not do it. It would take up about 16 GB of memory. It gets

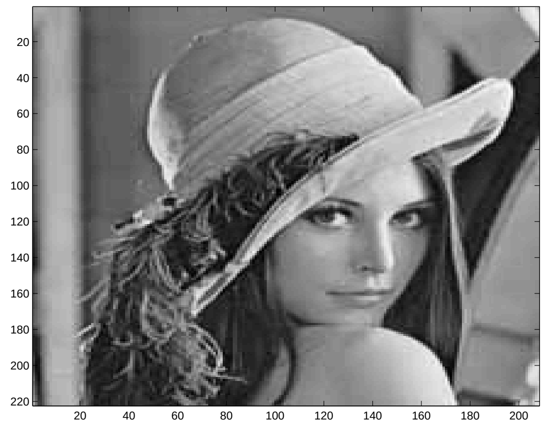


Figure 7.8: *The Lena test image.*

even worse, if we consider to solve such a problem. When using the object–oriented approach the same matrix takes us less than 100 Kb of memory.

Again we have visualized the extra variables in Fig. 7.9, this time for the two first iterations.

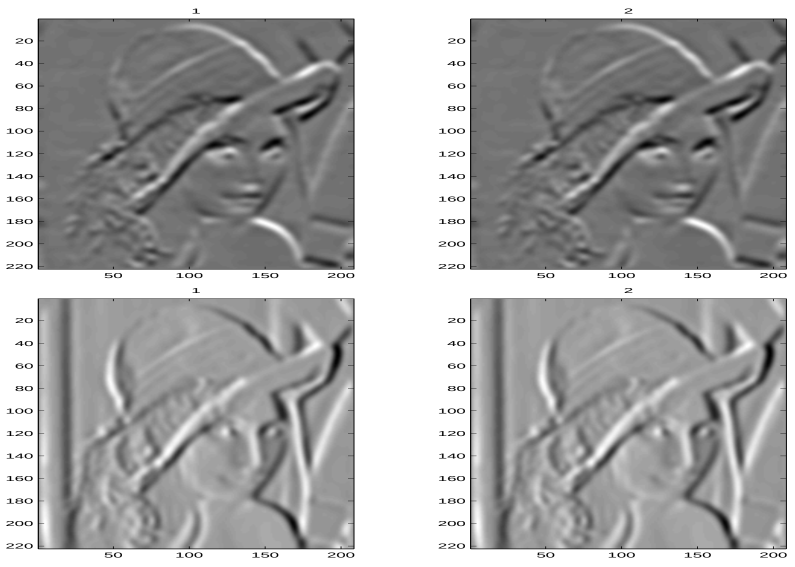


Figure 7.9: *The extra variable associated with the Lena image in the first two iterations. The two images at the top corresponds to the derivative in the s–direction, and the bottom to the t–direction.*

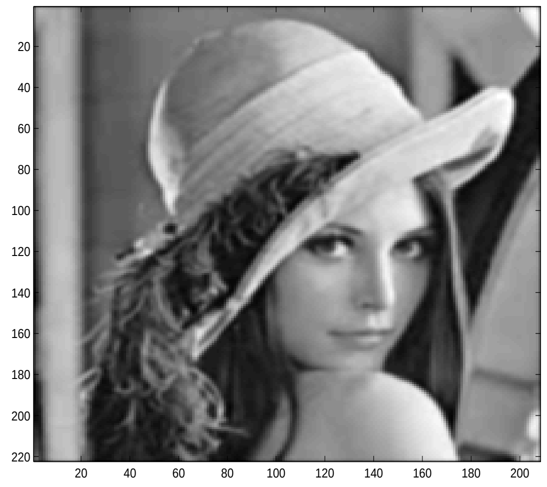


Figure 7.10: *The best calculated solution for the Lena test image.*

The best reconstructed solution is visualized in Fig. 7.10.

It is hard to clarify from this single test, whether TV is a good approach or not when it comes to smooth images with few edges. More test should be carried out in this field.

7.5 Further Experiments

Caused by the time limitation of this project, we have not been able to carry out that many experiments. The focus of this project have more been on the development of the algorithm.

A lot of interesting experiments can be carried out to examine how good the algorithm is, as well as doing experiments that shows that Total Variation regularization actually is very applicable in many problems. This is possible now with the object–oriented implementation.

In [5] it is concluded that the new linearization technique introducing the extra variables makes a much faster convergence. However, it would be nice to see which influence it has that the system we solve is symmetric. And what difference it does to use CG on the symmetrized system compared to Minres on the symmetric system.

A further investigation of the influence of the β parameter would also also be a good idea.

Chapter Summary

In this chapter we have made some experiments using the algorithm. We have illustrated how extra variable represents the edges in the images, and we can conclude that it seems that the algorithm works properly.

Chapter 8

Conclusion

In this thesis we have covered the basic theory of inverse problems, and we have discussed that the use of two–norms in some solution techniques results in too smooth solutions. To obtain solutions with edges we have introduced the Total Variation functional, which we have shown allows a wider class of solutions. The thesis has gained insight of Total Variation in general, and the nonlinear behavior of the functional.

The thesis has covered how important it is to choose an appropriate discretization for the two–dimensional Total Variation functional. We have shown that in terms of regularization the discretized functional should not allow high frequencies, and that this can be the case, if the discretization is not done carefully. We have proposed a new way of discretizing the functional.

The main focus of this work has been to develop an algorithm for solving large–scale inversion problems using a Tikhonov based algorithm using Total Variation. To be able to do this implementation we have derived the important theory needed.

The implemented algorithm is based on Newton's method. The TV functional is very nonlinear, we have shown how to introduce extra variables and thus how to obtain an augmented system. The new system gives a better linearization for Newton's method. Another contribution from this work is that we have derived a symmetric and numerically stable system to compute the Newton step.

We have proved that when introducing the extra variables the "landscape" which is minimized under certain limitations is convex, and thus that there exists a unique minimizer. The limitations which are associated with the extra variables are explained as well.

The implementation issues of the object–oriented Total Variation algorithm using M𝒪𝒪Re Tools has been explained in detail. The advantages of the object–oriented approach have been described as well.

Finally, the implemented algorithm has been tested with experiments.

8.1 Future Work

To speed up the convergence it might be an idea to find a good preconditioner for the augmented system for Minres. This might also help in situations where the system have a relatively large condition number.

Experiments with stopping criteria should be carried out in order to clarify how the outer Newton criteria and the inner Minres stopping criteria should be chosen. Again the connection may also depend on the issue of finding a good preconditioner.

Experiments should also clarify if the linear approximation to the discretized Total Variation functional can be used successfully. If this is the case, one should consider if TV is worth all the troubles that it is associated with.

Appendix A

Code

A.1 getcentralL.m

```
function L = getcentralL(n)
% getcentralL Create central modified discrete first order derivative operator
%
% <Synopsis>
%   L = getcentralL(n)
%
% <Description>
%   Creates a central discrete approximation to a first order derivative
%   operator on a regular grid with n points,
%\[
% \begin{bmatrix}
%       -2 &  2 &    &        &        &         &   \\
%       -1 &  0 &  1 &        &        &         &   \\
%          & -1 &  0 &   1    &        &         &   \\
%          &    &    & \ddots & \ddots & \ddots  &   \\
%          &    &    &        &   -1   &     0   & 1 \\
%          &    &    &        &        &    -2   & 2
% \end{bmatrix}
%\]
%
%The dimensions of the function and the resulting derivative are the same.
%The nullspace is spanned by a constant vector.
%
% Jesper Pedersen, IMM, DTU, 2005.

% Compute L.
c = [-1,0,1];
```

```
L = sparse(n,n);
L(1,1:2)=[-2 2];
L(n,n-1:n)=[-2 2];
for i=1:3
    L = L + sparse(2:n-1,[1:n-2]+i-1,c(i)*ones(1,n-2),n,n);
end
L = SparseMatrix(L);
```

A.2 Tvmt2d.m

```
function [Xa,extra] = TV2DMT(A,b,lambda,beta,x,opts,dim)

% Tihonov solution with total-varation penalty term.
% Solving procedure is based on Newton's method.
%
% Implementation builds on MOOReTools, which is needed for
% this implementation to work properly. MOOReTools is available
% at www.imm.dtu.dk/~tkj/mooretools/
%
% CALL:
%    [X,extra] = TV2DMT(A,b,lambda,beta,x,dim,opts)
%
% Solves the system
%
%    x_lambda = argmin {||Ax-b||_2^2 + lambda J_TV(x) },
%
% where J_TV is a perturbed Total-variation functional given by
%
%  J_TV(x) = sum_{i,j}( sqrt ( (dx/ds)_ij^2 + (dx/dt)_ij^2 + beta^2 ) ),
%
%  s and t are space coordinates corresponding to columns and rows
%  repectively.
%
%Input
% A       A Matrix or a KroneckerProduct2D
% b       A Vector or a Vector2D
% lambda  Regularization parameter
% beta    Parameter ensuring differentiability of J_TV
%             (default: beta = 1e-6)
% x0      Starting guess, a Vector or a Vector2D
%             (default: X0 = Vector( (||b||_F)/prod(size(b))*ones(dim))
%              i.e., with same "energy" as b.
% dim     A standard vector [Nx Ny] that describes the dimensions
%         of the solution domain. If X0 is a Vector2D, dim can be
%         left out.
% opts    A standard vector [MaxIter Tol]
```

```
%
%Output
% Xa      A VectorCollection that contains the iterates
% Extra   A struct that contains information on the iterates

%     Jesper Pedersen, February 2005, IMM, DTU

%Test imput arguments, and set defaults if not set

if nargin==7
    m = dim(1); n = dim(2);
    if isa(A,'Matrix')
        if size(A,1)~=m*n
            error('Dimension mismatch');
        end
    end
end
if nargin<6 | isempty(opts), opts = [50 1e-6]; end
if nargin<5
    if isa(A,'KroneckerProduct2D')
        m = size(getterm(A,2),2);
        n = size(getterm(A,1),1);
        x = Vector2D(zeros(m,n));
    else
        error('Since A is not a KroneckerProduct2D, dim must be defined');
    end
end

if exist('x')
    if isempty(x)
       if isa(A,'kroneckerproduct2d')==1
           x = Vector2D(zeros(m,n));
       else
           x = Vector(zeros(m*n,1));
       end
    end
end

if nargin<4, beta=1e-6; end
if nargin<3, error('Too few input arguments. A, b and lambda must be specified!'); end

%Initialization
iter = 1;
stop = 0;

%Initialize extra variable
ws = Vector2d(zeros(m,n));
wt = Vector2d(ws);
```

```
if isa(b,'Vector2d')
    %Discrete approximation to first order derivative
    Ds = KroneckerProduct2D(IdentityOperator(n),getcentralL(m));
    Dt = KroneckerProduct2D(getcentralL(n),IdentityOperator(m));
else
    %Use VectorReshape to be able to multiply KroneckerProduct2D onto a Vector
    Vr = VectorReshape(m,n);

    %Discrete approximation to first order derivative
    Ds = OperatorProduct({Vr',KroneckerProduct2D(IdentityOperator(n),getcentralL(m)),Vr});
    Dt = OperatorProduct({Vr',KroneckerProduct2D(getcentralL(n),IdentityOperator(m)),Vr});
end

%Empty VectorCollection to store iterands, starting guess as first element
Xa = VectorCollection;
Xa(1) = x;

%Calculate initial values
Dsx = Ds*x;
Dtx = Dt*x;

psi = sqrt( Dsx.^2 + Dtx.^2 + beta^2 );
psi_inv = 1./psi;
ws = Dsx.*psi_inv;
wt = Dtx.*psi_inv;

%Begin Newton iterations
while ~stop
    psi = sqrt( (Ds*x).^2 + (Dt*x).^2 + beta^2 );
    Tmx = A'*(A*x-b)+lambda*(Ds'*DiagonalOperator(psi_inv)*Ds*x + ...
          (Dt'*DiagonalOperator(psi_inv)*Dt*x))
    wswt = ws.*wt;
    ws2 = ws.^2;
    wt2 = wt.^2;

    schur = -lambda*OperatorSum( ...
         {OperatorProduct({Ds',DiagonalOperator((ws2-1).*psi_inv),Ds}),...
          OperatorProduct({Ds',DiagonalOperator(wswt.*psi_inv),Dt}), ...
          OperatorProduct({Dt',DiagonalOperator((wt2-1).*psi_inv),Dt}),...
          OperatorProduct({Dt',DiagonalOperator(wswt.*psi_inv),Ds})});

    A_aug = [-IdentityOperator(size(A)) A; A' schur];
    b_aug = [zeros(b); -Tmx];

    %Find the direction of the next step
    [s,flag] = minres(A_aug,b_aug,regset('Iter',500,'TolRes',0.1*opts(2)));
    dx = s{2};

    %Update the extra variables
    dws = -ws+(Ds*x + ((DiagonalOperator(1-ws2)*Ds - DiagonalOperator(wswt)*Dt)*dx)).*psi_inv;
    dwt = -wt+(Dt*x + ((DiagonalOperator(1-wt2)*Dt - DiagonalOperator(wswt)*Ds)*dx)).*psi_inv;

    %Avoiding divide by zero
    epsilon = 1e-16;
    p_ws = find(dws>epsilon); %positive
    n_ws = find(dws<-epsilon); %negative

    tau_ws1 = (1-getdata(ws(p_ws)))./getdata(dws(p_ws));
    tau_ws2 = (1+getdata(ws(n_ws)))./(-getdata(dws(n_ws)));
    tau_ws = [tau_ws1;tau_ws2];

    p_wt = find(dwt>epsilon); n_wt = find(dwt<-epsilon);
    tau_wt1 = (1-getdata(wt(p_wt)))./getdata(dwt(p_wt));
    tau_wt2 = (1+getdata(wt(n_wt)))./(-getdata(dwt(n_wt)));
    tau_wt = [tau_wt1;tau_wt2];

    %Find the minimum of all the tau values
    %The step in dw is limited to be maximum 0.99 (insert a 0.99 ind the end)
    tauw = min([tau_ws1(:); tau_ws2(:); tau_wt1(:); tau_wt2(:); 0.99]);

    %Soft line search
    if iter==1, Tvalue = objectfun2d(x,A,b,lambda,beta,m,n); end
    [tau,Tvalue,Tnew] = lines(x,A,b,lambda,beta,m,n,Tvalue,dx);

    x = x + tau*dx;
    ws = ws + tauw*dws;
    wt = wt + tauw*dwt;

    %Collect relavant information on the iterates
%    tic; extra.condAug(iter)=condest(getmatrix(A_aug));toc
    extra.minresflag(iter) = flag;
    extra.tauw(iter,:) = tauw;
    extra.taux(iter,:) = tau;
    extra.dx(iter) = norm(dx);
    extra.dws(iter) = norm(dws);
    extra.dwt(iter) = norm(dwt);

    Xa(:,iter) = x;

    %Check stopping criteria
    if opts(1)==iter
        extra.stop='STOP:iter';
        break;
    end
    if opts(2)>=extra.dx(iter);
        extra.stop='STOP:small x-step';
        break;
    end
    iter = iter+1
end
```

A.3 lines.m

```
function [tau,Tvalue,Tnew] = lines(x,A,b,lambda,beta,m,n,Tvalue,dx)

%Primitive line search procedure. Continues to halve the step until it is downhill.
%After 30 iterations, the line search is stoppet.

xx=x+dx;
Tnew = kridtfun2d(xx,A,b,lambda,beta,m,n);
tau = 1;
if Tnew < Tvalue
    Tvalue = Tnew;
else
    k=1;
    while Tnew > Tvalue
        tau = tau/2;
        xx = x + tau*dx;
        Tnew = objectfun2d(xx,A,b,lambda,beta,m,n);
        if k>30
            warning('Error: Line search stopped - too many iterations')
            Tnew=0;
        end
        k=k+1;
    end
end
```

A.4 objectfun2d.m

```
function [T,TV] = objectfun2d(x,A,b,lambda,beta0,m,n)

%Use VectorReshape to be able to multiply KroneckerProduct2D onto a Vector
Vr = VectorReshape(m,n);

%Discrete approximation to first order derivative
Ds = OperatorProduct({Vr',KroneckerProduct2D(IdentityOperator(n),getcentralL(m)),Vr});
Dt = OperatorProduct({Vr',KroneckerProduct2D(getcentralL(n),IdentityOperator(m)),Vr});

TV = sqrt( (Ds*x).^2 + (Dt*x).^2 + beta0^2 );
T = norm(A*x-b)^2 + lambda*sum( TV );
```

Bibliography

[1] R. Acar and C. Vogel. Analysis of bounded variation penalty method for ill-posed problems. *Inverse Problems*, Vol. 10, No. 6.:1217–1229, 1994.

[2] R. Barrett, M. Berry, T. F. Chan, J. Demmel, J. Donato, J. Dongarra, V. Eijkhout, R. Pozo, C. Romine, and H. Van der Vorst. *Templates for the Solution of Linear Systems: Building Blocks for Iterative Methods.* SIAM, Philadelphia, PA, 1994.

[3] R. L. Burden, J. D. Faires, and A. C. Reynolds. *Numerical Analysis.* Prindle, Weber and Schmidt, 1978. Wadsworth International Student Edition.

[4] R. H. Chan, T. F. Chan, and H. M. Zhou. Advanced signal processing algorithms. *Journal of Computational and Applied Mathematics, in Proceedings of the International Society of Photo-Optical Instrumentation Engineers, F. T. Luk, ed., SPIE*, pages 314–325, 1995.

[5] T. F. Chan, G. H. Golub, and P. Mulet. A nonlinear primal–dual method for total variation–based image restoration. *SIAM*, 20, No. 6:1964–1977, 1999.

[6] J. Eising. *Lineær Algebra.* Institut for Matematik, Danmarks Tekniske Universitet, 1997.

[7] L. Eldén, L. Wittmeyer-Koch, and H. B. Nielsen. *Introduction to Numerical Computation.* Studentlitteratur, 2004.

[8] R. Fletcher. *Practical Methods of Optimization.* Wiley, 1987. Second Edition.

[9] P. E. Frandsen, K. Jonasson, H. B. Nielsen, and O. Tingleff. Unconstrained optimization, 3rd edition, 2004.

[10] P. E. Giil, W. Murray, and Margaret H. Wright. *Practical Optimization.* Academic Press, Harcourt Brace Jovanovich, Publishers, Stanford University, 1981.

[11] C. H. Groetsch. *Inverse Problems in the Mathematical Sciences.* Vieweg Mathematics for Scientists, 1993.

[12] P. C. Hansen. Regularization tools – a MATLAB package for analysis and solution of discrete ill-posed problems. *Numerical Algorithms*, 6:1–35, 1994.

[13] P. C. Hansen. *Rank–Deficient and Discrete Ill–Posed Problems: Numerical Aspects of Linear Inversion.* SIAM, 1997.

[14] P. C. Hansen. Deconvolution and regularization with Toeplitz matrices. *Numer. Algo.*, 29:323–378, 2002.

[15] P.C. Hansen, M. Jacobsen, H. Sørensen, and J.M. Rasmussen. The pp–tsvd algorithm for image restoration problems; in p.c. hansen, b.h. jacobsen, and k. mosegaard (eds.), methods and applications of inversion, 2000.

[16] M. Jacobsen. *Modular Regularization Algorithms.* Ph.D. Thesis, Informatics and Mathematical Modelling, Technical University of Denmark, DK-2800, Lyngby, 2004.

[17] K. Madsen and H. B. Nielsen. Supplementary notes for 02611 optimization and data fitting, 2nd edition, 2004.

[18] H. B. Nielsen. Algorithms for linear optimization – an introduction, 1998.

[19] C. F. van Loan. The ubiquitous kronecker product. *Journal of Computational and Applied Mathematics*, 123:85–100, 2000.

[20] C. R. Vogel. *Computational Methods for Inverse Problems.* SIAM, 2002.