NTNU
Norwegian University of
Science and Technology

Lasso-Path and Taut String Algorithm for One-Dimensional Total Variation Regularization

Lene Hagen

Master of Science in Physics and Mathematics
Submission date: June 2017
Supervisor: Markus Grasmair, IMF

Norwegian University of Science and Technology
Department of Mathematical Sciences

Abstract

In this thesis, we consider the minimization problem arising from a linear operator equation when applying Tikhonov regularization. In particular, we study the one-dimensional case with total variation penalization. We present two numerical methods for the reconstruction of the original function, both of which give exact results. The first is the taut string algorithm, for which we assume no linear transformation of the data. The second is a modified lasso-path algorithm, derived for the total variation regularization. The thesis focuses on the derivation of these algorithms. The taut string algorithm is of lower complexity than the lasso-path algorithm, but the trade off is that it is only applicable on non-transformed data and only computes the solution for a single regularization parameter. The lasso-path is applicable on all linear transformations, and computes the solution simultaneously for all regularization parameters in a given range. Finally, we test both of the algorithms with a few numerical experiments. We test the taut string algorithm for a block signal, and for moderate amount of noise we are able to reconstruct the signal. We test the lasso-path algorithm on the same block signal for different linear operator equations, in particular for moving averages and for random measurements. In general, when not too much noise is added, we are able to reconstruct the significant jumps of the original signal, although not exactly. For the random matrices, with fewer measurements than unknowns, the problem is more difficult, but we were able to find a good reconstruction if the number of jumps in the original signal were small enough.

Sammendrag

I denne masteroppgaven ser vi på minimeringsproblemet vi får ved å legge til Tikhonov-regularisering til en lineær operatorligning. Vi tar for oss det endimensjonale problemet hvor vi straffer med totalvariasjon. Vi presenterer to numeriske metoder for å rekonstruere den originale funksjonen. Begge metodene gir eksakt løsning. Den første metoden er taut string-algoritmen. For denne metoden antar vi at det ikke er noen lineærtransformasjon av dataene. Den andre metoden er en lasso-algoritme som er modifisert ved å utlede for totalvariasjonsregularisering. Oppgaven fokuserer på utledningen av disse algoritmene. Taut string-algoritmen har lavere kjøretid enn lasso-algoritmen, men den fungerer kun på ikke-transformerte data og beregner løsningen for en enkelt regulariseringsparameter. Lasso-algoritmen kan brukes på alle lineærtransformasjoner, og beregner løsningen for alle regulariseringsparametre i et intervall samtidig. Til slutt tester vi begge algoritmene med noen numeriske eksperimenter. Vi tester taut string-algoritmen for et blokksignal, og vi klarer å rekonstruere signalet med moderat mengde støy til stede. Vi tester lasso-algoritmen på det samme blokksignalet med forskjellige lineære likningssystemer, spesielt for glidende/bevegde gjennomsnitt og for tilfeldige målinger. Generelt klarer vi å rekonstruere de signifikante hoppene, skjønt ikke helt eksakt, når det ikke er for mye støy. Problemet er mye vanskeligere for de tilfeldige matrisene, der vi brukte færre målinger enn antall ukjente. Allikevel klarte vi å finne en god rekontruksjon hvis antall hopp i signalet ikke var for stort.

Preface

This master thesis was written during my final four months at the Norwegian University of Science and Technology in Trondheim. Its submission marks the completion of my five year long integrated M.Sc. in Applied Physics and Mathematics, with specialization Industrial Mathematics.

I would like to thank my supervisor, Associate Professor Markus Grasmair, for proposing an interesting topic, and for guiding me the last year. Thank you for the countless hours you spent helping me and answering my questions. I would also like to thank Eirik and my family for their support.

Contents

Chapter 1

Introduction

The problem of reconstructing a signal u_0 from measurements f, also known as an inverse problem, arises in many applications. For all measurements there will always be some noise present, and in addition, the signal might also be transformed in some way. That is, we have to solve an equation of the form

$$f = Au_0 + \varepsilon \tag{1.1}$$

for u_0, where f is the given data, A is a transformation and ε some unknown noise.

For example, an audio signal might be subject to some form of damping in a microphone or speaker, or we have a frequency dependent damping in a integrated circuit.

Another example is that of compressive sampling. There are times the cost of sampling is so high, that we try to find a way to reconstruct the original signal by gathering fewer measurements than there are unknown values. That is, we try to solve an underdetermined system. An example is to limit the radiation exposure during a CT scan. The aim is to reconstruct the best image using few measurements. This kind of sampling is referred to as compressive sampling, or compressed sensing, which is expanded on in [1].

These are all continuous problems, and in this thesis we will try to solve a discretized version of such a problem. We assume that u_0 is the discretization of a univariate function $\hat{u_0}$.

In particular, we will consider the problem of reconstructing $u_0 \in \mathbb{R}^n$ from some noisy data $f \in \mathbb{R}^m$. We assume that u_0 is operated on by a linear operator $A \in \mathbb{R}^{m \times n}$. In practice this means that we have m measurements of an n-dimensional vector.

The transformation A can take many forms depending on the problem one is studying. For example, it can be a discretized convolution operator, or another integral operator, or it can be a random matrix. The latter is described in e.g. the article about compressive sampling, [1].

We will not assume anything about A, other than that it is a real $m \times n$ matrix. The goal of this thesis is to find a solution method to the general problem, and we will test our method for signals subject to a few different linear operators.

Generally, we have m equations and $m + n$ unknowns (u_0 and ε). With no prior knowledge of the true solution, it is difficult to find a good model to fit the data. Therefore it is necessary to assume something about the signal u_0. We assume that is it piecewise constant with a small number of jumps.

As proposed in [9], one can try to compute the least squares estimates with a total variation (TV) regularization term. Then for some regularization parameter $\alpha > 0$ we have the minimization problem

$$\min_{u \in \mathbb{R}^n} \quad \frac{1}{2}\|Au - f\|_2^2 + \alpha \|Du\|_1. \tag{1.2}$$

Here D is the forward difference matrix

$$D = \begin{bmatrix} -1 & 1 & & \\ & \ddots & \ddots & \\ & & -1 & 1 \end{bmatrix}. \tag{1.3}$$

In Chapter 4 and 5 we derive two different algorithms for solving (1.2). The first algorithm solves for the general case where A is any linear transformation, and the second solves (1.2) when A is the identity matrix. The algorithm for the latter is known as the *taut string algorithm*, see e.g. [10] and [8].

The algorithm for the general case is a variant of the *lasso path algorithm*, which is described in [7]. In the original lasso-path algorithm one minimizes the functional $\frac{1}{2}\|Au - f\|_2^2 + \alpha\|u\|_1$. In our case, we have the regularization term $\alpha\|Du\|_1$ instead of $\alpha\|u\|_1$, but up to some modifications the algorithm is still applicable.

The main outline of this thesis is as follows. In Chapter 2 we go through some theory of convex analysis, which we need to derive the algorithms. We discuss the existence of minimizers, and derive the optimality conditions for the problem in Chapter 3, before we in Chapter 4 and 5 derive the algorithms. Chapter 6 takes on some numerical experiments with the taut string algorithm, and Chapter 7 does the same for the lasso-path algorithm for total variation.

Chapter 2

Convex Analysis

We go through some of the theory of convex analysis needed in later chapters. A reader familiar with convex functions, subdifferentials and conditions for existence of optimizers of a convex optimization problem may skip this chapter.

Definition 2.1 (Convex function)**.** A function $f\colon \mathbb{R}^n \to \mathbb{R}$ is *convex* if

$$f(\alpha x + (1-\alpha)y) \leq \alpha f(x) + (1-\alpha)f(y) \tag{2.1}$$

for all points $x, y \in \mathbb{R}^n$ and $0 < \alpha < 1$. The function f is *strictly convex* if the inequality holds strictly for all distinct points x and y.

Similarly, a function f is *concave* if $f(\alpha x + (1-\alpha)y) \geq \alpha f(x) + (1-\alpha)f(y)$ for $x, y \in \mathbb{R}^n$ and $0 < \alpha < 1$. Equivalently, a function f is convex if and only if $-f$ is concave.

Example 2.2 (Convex function)**.** We show that the function $f\colon \mathbb{R} \to \mathbb{R}, x \mapsto |x|$ is convex. Take $0 < \alpha < 1$ and $x, y \in \mathbb{R}$. Then we have

$$\begin{aligned} f(\alpha x + (1-\alpha)y) &= |\alpha x + (1-\alpha)y| \\ &\leq |\alpha x| + |(1-\alpha)y| \\ &= |\alpha||x| + |(1-\alpha)||y| \\ &= \alpha|x| + (1-\alpha)|y| \\ &= \alpha f(x) + (1-\alpha)f(y). \end{aligned} \tag{2.2}$$

If we know that we are working with a differentiable function f it can be proved, cf. [5, p. 183], that the following relation holds:

Lemma 2.3 (Convexity by first order differentiation)**.** Let $f\colon \mathbb{R}^n \to \mathbb{R}$ be differentiable on $\mathbb{R}^n$. Then f is convex if and only if $f(y) \geq f(x) + \langle \nabla f(x), y - x\rangle$ for all points $x, y \in \mathbb{R}^n$. The function f is strictly convex if the inequality holds strictly for all distinct points x and y.

We introduce a generalization of a monotone function to higher dimensions:

Definition 2.4 (Monotone mapping)**.** A mapping $T\colon \mathbb{R}^n \to \mathbb{R}^n$ is said to be *monotonically increasing* if

$$\langle T(x) - T(y), x - y\rangle \geq 0 \tag{2.3}$$

for all $x, y \in \mathbb{R}^n$.

If $n = 1$, the condition in Definition 2.4 becomes $(T(x) - T(y)) \cdot (x - y) \geq 0$ for all $x, y \in \mathbb{R}$. Equivalently, for any x and y such that $x \leq y$ we have that $T(x) \leq T(y)$, which is the standard definition of a monotonically increasing scalar function. Similarly, for a monotonically decreasing function we define the generalization to higher dimensions by reversing the inequality in Definition 2.4. We can now define the relation between a differentiable function f and its gradient ∇f.

Theorem 2.5 (Monotone gradient)**.** *Let $f\colon \mathbb{R}^n \to \mathbb{R}$ be differentiable on $\mathbb{R}^n$. Then f is convex on $\mathbb{R}^n$ if and only if its gradient ∇f is monotonically increasing.*

Proof. Cf. [5, p. 185]. □

An analogue of Theorem 2.5 also holds for concave functions. The function f is concave if and only if its gradient is monotonically decreasing. If we know that we are working with a twice differentiable function f, then it can be proved, cf. [5, p. 190], that the following relation holds:

Lemma 2.6 (Convexity by second order differentiation)**.** Let $f\colon \mathbb{R}^n \to \mathbb{R}$ be a twice differentiable function on $\mathbb{R}^n$. Then f is convex at x if and only if the Hessian of f at x, denoted by $\nabla^2 f(x)$, is positive semi-definite for all $x \in \mathbb{R}^n$. The function f is strictly convex iff $\nabla^2 f(x)$ is positive definite for all x.

We have looked at several ways to characterize a convex function. Often we build functions using simpler ones, which is the motivation for presenting the following convexity preserving operation

Theorem 2.7 (Sum of convex functions)**.** *Let $f_1, f_2, \ldots, f_m$ be convex functions and $t_1, t_2, \ldots, t_m$ be positive numbers. Then $f := \sum\limits_{i=1}^{m} t_i f_i$ is a convex function. If at least one of the functions f_i is strictly convex, then f is strictly convex.*

Proof. For any f_i satisfying (2.1) the product $t_i f_i$ will also satisfy (2.1) when $t_i > 0$. Then,

$$\begin{aligned} f(\alpha x + (1-\alpha)y) &= \sum_{i=1}^{m} t_i f_i(\alpha x + (1-\alpha)y) \\ &\leq \sum_{i=1}^{m} (\alpha t_i f_i(x) + (1-\alpha)t_i f_i(y)) = \alpha f(x) + (1-\alpha) f(y), \end{aligned}$$

which by Definition 2.1 shows that f is convex. If at least one f_i is strictly convex, then the inequality will be strict and hence f will be strictly convex. □

Definition 2.8 (Subdifferential). The *subdifferential* $\partial f(x)$ of a convex function $f\colon \mathbb{R}^n \to \mathbb{R}$ at x is the set of all vectors $s \in \mathbb{R}^n$ such that

$$f(y) \geq f(x) + \langle s, y - x\rangle \tag{2.4}$$

for all $y \in \mathbb{R}^n$. Any such vector s is called a *subgradient* of f.

The subdifferential is set-valued. In the case where f is convex and differentiable at x, the subdifferential contains only one element, the vector $\nabla f(x)$. We look at a simple yet fitting example to illustrate this concept.

Example 2.9 (Subdifferential). Again we consider the convex function $f(x) = |x|$. For $x \neq 0$ we know that f is differentiable and hence $\partial f(x) = f'(x) = \operatorname{sgn}(x) = \frac{|x|}{x}$. We know that f is not differentiable at $x = 0$. Inserting $x = 0$ in (2.4) we obtain the condition $|y| \geq sy$, which implies that

$$\begin{cases} s \leq \dfrac{|y|}{y} = 1 & \text{if } y > 0, \\ s \geq \dfrac{|y|}{y} = -1 & \text{if } y < 0. \end{cases} \tag{2.5}$$

Hence we have $\partial f(0) = [-1, 1]$. Summarized we define the notation for the subdifferential of f as

$$\partial f(x) = \frac{|x|}{x} := \begin{cases} \operatorname{sgn}(x) & \text{if } x \neq 0 \\ [-1, 1] & \text{if } x = 0. \end{cases} \tag{2.6}$$

Theorem 2.10 (Optimality condition). *Let f be a convex function. Then $\bar{x}$ is a minimizer of f if and only if $0 \in \partial f(\bar{x})$.*

Proof. We easily see when inserting $s = 0$ into (2.4) that we obtain $f(y) \geq f(\bar{x})$ for all y, and hence $\bar{x}$ is a minimizer. In order to prove the converse, let $\bar{x}$ be a minimizer. Then $f(y) \geq f(\bar{x})$ for all y, and hence $s = 0$ is a subgradient of f. □

Theorem 2.11 (Linear mappings and subdifferentials). *Let $A \in \mathbb{R}^{m\times n}$, and let $g\colon \mathbb{R}^n \to \mathbb{R}$ be convex. Then $g \circ A$ is convex and $\partial(g \circ A)(x) = A^T \partial g(Ax)$.*

Proof. Cf. [5, p. 263] □

Theorem 2.12. *Let $f_1, f_2, \dots, f_m$ be convex functions and $t_1, t_2, \dots, t_m$ be positive numbers. Then*

$$\partial\left(\sum_{i=1}^{m} t_i f_i\right)(x) = \sum_{i=1}^{m} t_i \partial f_i(x). \tag{2.7}$$

Proof. Cf. [5, p. 262]. □

Theorem 2.13. *Let $f\colon \mathbb{R}^n \to \mathbb{R}$ be a strictly convex function. Then f has at most one global minimizer.*

Proof. Assume to the contrary that f has more than one global minimizer, say $\bar{x} \neq \bar{y}$. Then we have $f(\bar{x}) = f(\bar{y})$. Take the point $x = \frac{1}{2}\bar{x} + \frac{1}{2}\bar{y}$. Then by the strict convexity of f we have

$$f(x) = f\left(\frac{1}{2}\bar{x} + \frac{1}{2}\bar{y}\right) < \frac{1}{2}f(\bar{x}) + \frac{1}{2}f(\bar{y}) = f(\bar{x})$$

which contradicts the assumption that $\bar{x}$ and $\bar{y}$ are global minimizers. Hence we cannot have more than one minimizer when f is strictly convex. $\square$

Definition 2.14. A function $f\colon \mathbb{R}^n \to \mathbb{R}$ is said to be *coercive* if

$$f(x) \to \infty \quad \text{whenever} \quad \|x\| \to \infty. \tag{2.8}$$

This definition is equivalent to all sublevel sets of f being compact. In particular, f is coercive if and only if for every $r \in \mathbb{R}$ there exists an $M \in \mathbb{R}$ such that $\|x\| \leq M$ whenever $f(x) \leq r$.

Theorem 2.15. *Let $f\colon \mathbb{R}^n \to \mathbb{R}$ be coercive and continuous. Then f has at least one global minimizer.*

Proof. Cf. [4, p. 34]. $\square$

Chapter 3

Optimality Conditions

Recall that we want to solve the minimization problem

$$\min_{u \in \mathbb{R}^n} \quad \frac{1}{2}\|Au - f\|_2^2 + \alpha\|Du\|_1. \tag{3.1}$$

As we presented in the introduction, we will derive two different algorithms for solving the problem, the taut string algorithm and the lasso-path algorithm for total variation. The taut string algorithm solves the problem when $A \in \mathbb{R}^{n\times n}$ is the identity operator, the lasso-path algorithm solves the problem for any $A \in \mathbb{R}^{m\times n}$. Independent of which of these algorithms we shall derive, we can still draw several common conditions that must hold for both problems.

In Chapter 3.2, we apply the first order necessary optimality condition to the general problem where A can be any linear operator. For the taut string algorithm, we simply have to insert the identity matrix for A to have the same conditions. In Chapter 3.1 we explore the existence of minimizers.

Before we start, we want to emphasize that there are other ways of deriving these conditions and presenting them. For instance, in some cases the derivation of the dual problem gives additional insight. Here we have only included what is necessary for our goal.

3.1 Existence of Minimizers

We want to examine if, and when, there exists a solution to (3.1), and in which cases that minimizer is unique. We can draw conclusions about the existence and uniqueness by looking at the coerciveness and convexity, respectively, of the functional.

First, we examine the convexity of the functional. If we differentiate the first term of (3.1) twice, we get that the Hessian is $\nabla^2 \frac{1}{2}\|Au - f\|_2^2 = A^T A$, which is positive semi-definite. From Theorem 2.6 we then know that $\frac{1}{2}\|Au - f\|_2^2$ is a convex function.

For the second term, the function $\|\cdot\|_1$ is a sum of absolute values, which we know from Example 2.2 is convex. Since all the scalars are positive, we can from Theorem 2.7

draw the conclusion that the functional in (3.1) is convex. Hence any solution u of (3.1) is a global minimizer.

In the case where A is an invertible matrix, it follows that A^TA is positive definite. Hence, by the same theorems as above, $\|Au - f\|^2$, and consequently the whole functional in (3.1), is strictly convex. Then from Theorem 2.13 we know that the minimization problem (3.1) has at most one solution. That is, if A is invertible and we have found a solution, we know that this is the only solution.

Further, we examine the coercivity of the functional. Let $\mathbb{1}$ denote the $n \times 1$ vector where all elements are equal to 1. We propose the following:

Lemma 3.1.1. If $\mathbb{1} \notin \ker(A)$, the functional in (3.1) is coercive for all $\alpha > 0$. In particular (3.1) has a solution for all $\alpha > 0$.

Proof. Define the functional $F_\alpha(u) = \frac{1}{2}\|Au - f\|_2^2 + \alpha\|Du\|_1$. We prove that u can be bounded in terms of $F_\alpha(u)$ under the assumption $\mathbb{1} \notin \ker(A)$, which is equivalent to showing coercivity of F_α.

Choose $t > 0$ and u such that $F_\alpha(u) \leq t$. We want to show that we can always find an expression $g_\alpha(t)$ such that $\|u\|_1 \leq g_\alpha(t)$, i.e., show that u is bounded.

We decompose u into

$$u = \text{proj}_{\ker(D)}u + \text{proj}_{(\ker(D))^\perp}u, \tag{3.2}$$

and to shorten notation we define

$$T := \text{proj}_{\ker(D)} \quad \text{and} \quad Q := \text{proj}_{(\ker(D))^\perp}. \tag{3.3}$$

We have already specified that we are working with $D\colon \mathbb{R}^n \to \mathbb{R}^{n-1}$ as the finite difference operator given in (1.3). Then we know that the null space of D is the set of all constant functions:

$$\ker(D) = \{c \cdot \mathbb{1} : c \in \mathbb{R}\}. \tag{3.4}$$

The projection of u onto the kernel of D is the constant part of u. We have

$$u = c_u \cdot \mathbb{1} + (u - c_u \cdot \mathbb{1}) = c_u \cdot \mathbb{1} + Qu, \tag{3.5}$$

where $c_u = \frac{1}{n}\langle u, \mathbb{1}\rangle$. From this decomposition of u we have

$$\|u\|_1 = \|c_u \cdot \mathbb{1} + Qu\|_1 \leq |c_u|\|\mathbb{1}\|_1 + \|Qu\|_1. \tag{3.6}$$

If we restrict D to $(\ker(D))^\perp$, or $\text{ran}(Q)$, we have that $\ker(D|_{(\ker(D))^\perp})$ is trivial, and hence $D|_{(\ker(D))^\perp}$ is injective. Since $\dim(\ker(D)) = 1$ it follows that $\dim(\ker(D))^\perp = n - 1$, and hence $D|_{(\ker(D))^\perp} : (\ker(D))^\perp \to \mathbb{R}^{n-1}$ is invertible. We can write

$$Qu = (D|_{\text{ran}(Q)})^{-1} \circ DQu \tag{3.7}$$

which gives us

$$\begin{aligned} \|Qu\|_1 = \|(D_{\text{ran}(Q)})^{-1}DQu\|_1 &\leq \|(D|_{\text{ran}(Q)})^{-1}\|_1\|DQu\|_1 \\ &= \|(D|_{\text{ran}(Q)})^{-1}\|_1 \cdot \|Du\|_1. \end{aligned} \tag{3.8}$$

The last inequality comes from $Du = D(Tu + Qu)$ where Tu is a constant function, and hence $DTu = 0$. Since $F_\alpha(u) \le t$ we have from (3.1) that $\|Du\|_1 \le \frac{t}{\alpha}$, which gives

$$\|u\|_1 \le |c_u|\|\mathbb{1}\|_1 + \frac{t}{\alpha}\|(D|_{\mathrm{ran}(Q)})^{-1}\|_1. \tag{3.9}$$

To find an upper bound for $|c_u|\|\mathbb{1}\|_1$ we use the fact that

$$\|ATu\|_1 = \|A(c_u \cdot \mathbb{1})\|_1 = |c_u|\|A\mathbb{1}\|_1, \tag{3.10}$$

which gives us

$$|c_u|\|\mathbb{1}\|_1 = \|ATu\|_1 \frac{\|\mathbb{1}\|_1}{\|A\mathbb{1}\|_1}, \tag{3.11}$$

which holds because of the assumption $\mathbb{1} \notin \ker(A)$, which is equivalent to $\|A\mathbb{1}\| \neq 0$. We have

$$\begin{aligned}
\|ATu\|_1 &\le \|ATu + AQu - AQu\|_1 \\
&= \|A(Tu + Qu) - AQu\|_1 \\
&= \|Au - AQu\|_1 \\
&\le \|Au\|_1 + \|AQu\|_1 \\
&= \|Au + f - f\|_1 + \|AQu\|_1 \\
&\le \|Au - f\|_1 + \|f\|_1 + \|A\|_1\|Qu\|_1 \\
&\le \sqrt{m}\|Au - f\|_2 + \|f\|_1 + \|A\|_1\|(D|_{\mathrm{ran}(Q)})^{-1}\|_1\frac{t}{\alpha}.
\end{aligned} \tag{3.12}$$

From (3.1) we have $\|Au - f\|_2 \le \sqrt{2t}$, which gives us

$$\|ATu\|_1 \le \sqrt{2mt} + \|f\|_1 + \|A\|_1\|(D|_{\mathrm{ran}(Q)})^{-1}\|_1\frac{t}{\alpha}. \tag{3.13}$$

Then we have showed that if $F_\alpha(u) \le t$ we have

$$\begin{aligned}
\|u\|_1 \le \left(\sqrt{2mt} + \|f\|_1 + \|A\|_1\|(D|_{\mathrm{ran}(Q)})^{-1}\|\frac{t}{\alpha}\right)\frac{\|\mathbb{1}\|_1}{\|A\mathbb{1}\|_1} \\
+\frac{t}{\alpha}\|(D|_{\mathrm{ran}(Q)})^{-1}\|_1 =: g_\alpha(t),
\end{aligned} \tag{3.14}$$

and hence F_α is coercive and thus has a minimizer. In particular, say u minimizes $F_\alpha(u)$. Then

$$F_\alpha(u_\alpha) \le F_\alpha(0) = \frac{1}{2}\|f\|_2^2, \tag{3.15}$$

which implies that

$$\|u_\alpha\|_1 \le g_\alpha\left(\frac{1}{2}\|f\|_2^2\right). \tag{3.16}$$

□

In the case where A is invertible, we have that $\ker(A) = 0$, and the functional in (3.1) is therefore coercive. From theorem 2.15 we know that if the functional is coercive, we have at least one minimizer. Combining this with the strict convexity result, we have shown for invertible A that (3.1) has a unique, global minimizer. In the other cases when $A \neq I$, any minimizer u is still a global minimizer, and it exists as long as $\mathbb{1} \notin \ker(A)$.

We will in the rest of this thesis assume that $\mathbb{1} \notin \ker(A)$ is satisfied, which implies that there exists a minimizer. We will also assume that the minimizer is unique, even when A is not necessarily invertible. We will come back to why and in which cases we can assume this in the end of Chapter 4.3.

3.2 Optimality Condition

Before we apply the optimality condition to our problem we introduce the function

$$G\ : \mathbb{R}^n \to \mathbb{R}, x \mapsto \alpha \|x\|_1. \tag{3.17}$$

In this way the minimization problem becomes

$$\min_{u \in \mathbb{R}^n} \quad \frac{1}{2}\|Au - f\|_2^2 + G(Du). \tag{3.18}$$

For u to be a solution to (3.1), we know from Theorem 2.10 that 0 must be a subgradient of $I(u)$. That is, we must have

$$\partial \frac{1}{2}\|Au - f\|_2^2 + \partial(G \circ D)(u) = A^T(Au - f) + D^T \partial G(Du) \ni 0. \tag{3.19}$$

Here ∂ denotes the subdifferential, which we defined in Definition 2.8. This generalization of the derivative is necessary in the cases where the function we are working with is not differentiable in all points, as is the case for G when $(Du)_i = 0$.

The subdifferential is set valued, hence the inclusion sign in (3.19). We can rewrite the expression by introducing a new variable, say p, such that (3.19) becomes

$$0 = A^T(Au - f) + D^T p \quad \text{where} \quad p \in \partial G(Du). \tag{3.20}$$

Now we have a reformulated optimality condition which is perhaps easier to work with. In this way we have separated out the element of (3.19) that is set valued into its own problem.

To sum up we have now shown that the vector u is a minimizer of (3.18) if and only if

$$D^T p = A^T f - A^T A u \quad \text{where} \quad p \in \partial G(Du). \tag{3.21}$$

Now we take a closer look at the expression $\partial G(Du)$. By following the same argumentation as in Example 2.9, we see that the jth element of the vector $\partial G(Du)$ is $\alpha \frac{|(Du)_j|}{(Du)_j}$. Here we use the same definition of the notation as in the example:

$$\frac{|(Du)_j|}{(Du)_j} := \begin{cases} \operatorname{sgn}((Du)_j) & \text{if} \quad (Du)_j \neq 0, \\ [-1, 1] & \text{if} \quad (Du)_j = 0. \end{cases} \tag{3.22}$$

Hence

$$p \in \partial G(Du) \iff \begin{cases} p_i = \alpha \cdot \operatorname{sgn}((Du)_i) & \text{if} \quad (Du)_i \neq 0, \\ |p_i| \leq \alpha & \text{if} \quad (Du)_i = 0. \end{cases} \tag{3.23}$$

Now we look at the system of equations

$$D^T p = A^T f - A^T A u. \tag{3.24}$$

We know that for (3.24) to be satisfied, we must have

$$A^T f - A^T A u \in \operatorname{ran}\left(D^T\right) \tag{3.25}$$

which is equivalent to

$$A^T f - A^T A u \in \ker(D)^{\perp}. \tag{3.26}$$

Recall that the null space of D is the set of all constant functions:

$$\ker(D) = \{c \cdot \mathbb{1} : c \in \mathbb{R}\}. \tag{3.27}$$

The orthogonal complement of $\ker(D)$ is consequently

$$(\text{Ker } D)^{\perp} = \{x : c\langle x, \mathbb{1}\rangle = 0, x \in \mathbb{R}^n, c \in \mathbb{R}\} = \{x : \langle x, \mathbb{1}\rangle = 0, x \in \mathbb{R}^n\}. \tag{3.28}$$

Then, for (3.26) to be satisfied we must have that

$$\mathbb{1}^T \left(A^T f - A^T A u\right) = 0, \tag{3.29}$$

or, written using summation:

$$\sum_{i=1}^{n} \left(A^T f - A^T A u\right)_i = 0. \tag{3.30}$$

Hence for u to be a solution to (3.1) it is necessary that

$$\sum_{i=1}^{n} \left(A^T f\right)_i = \sum_{i=1}^{n} \left(A^T A u\right)_i. \tag{3.31}$$

Further, to illustrate the system in (3.24), we have in vector form

$$D^T p = \begin{bmatrix} -p_1 \\ p_1 - p_2 \\ \vdots \\ p_{j-1} - p_j \\ \vdots \\ p_{n-1} \end{bmatrix} \begin{matrix} \\ \\ \\ \leftarrow \text{row } j. \\ \\ \\ \end{matrix} \tag{3.32}$$

We see that if we take the cumulative sum of $D^T p$ we get a vector containing only one p_j in each element. This suggests the following. We take the cumulative sum of the system of equations in (3.24) and rearrange the terms. Then we get the new system

$$p_j = \sum_{i=1}^{j} \left(A^T A u - A^T f\right)_i, \quad \text{for} \quad j = 1, \ldots, n-1. \tag{3.33}$$

For the last element, all p_js cancel, and we are left with

$$\sum_{i=1}^{n} \left(A^T Au - A^T f\right)_i = 0, \tag{3.34}$$

which is the same restriction we arrived at in (3.31).

Taking the cumulative sums is an invertible operation. Hence the equations we derived here are still sufficient optimality conditions. We sum up the results in the following theorem:

Theorem 3.2.1. The vector u is a minimizer of (3.1) if and only if the following are satisfied:

$$p_j = \sum_{i=1}^{j} \left(A^T Au - A^T f\right)_i \quad \text{for} \quad j = 1, \ldots, n-1, \tag{3.35}$$

$$\sum_{i=1}^{n} \left(A^T Au - A^T f\right)_i = 0, \tag{3.36}$$

$$p_i = \alpha \cdot \text{sgn}((Du)_i) \quad \text{if} \quad (Du)_i \neq 0, \tag{3.37}$$

$$|p_i| \leq \alpha. \tag{3.38}$$

The results we arrived at here are what we will continue with in the derivation of the algorithms. We make a junction here, as the continued work with each of the algorithms differ from this point on.

Chapter 4

Lasso-Path Algorithm for TV Regularization

In this chapter we will present the lasso-path algorithm for total variation for solving the problem given in (3.1). The algorithm is named such, because it is a variant of the *lasso-path algorithm*, which is described in [7]. In the original lasso algorithm one minimizes the functional $\frac{1}{2}\|Au-f\|_2^2+\alpha\|u\|_1$. In our case, we have the regularization term $\alpha\|Du\|_1$ instead of $\alpha\|u\|_1$, but up to some modifications the algorithm is still applicable.

In Chapter 3.2 we derived the optimality condition for the problem, which we will use here to derive the algorithm. We repeat both the minimization problem and the optimality conditions here for readability. We want to solve

$$\min_u \quad \frac{1}{2}\|Au - f\|_2^2 + \alpha\|Du\|_1. \tag{4.1}$$

The optimality conditions states that (u,p) is the primal-dual solution to (4.1) if and only if the following are satisfied:

$$p_j = \sum_{i=1}^{j} \left(A^T Au - A^T f\right)_i \quad \text{for} \quad j = 1,\dots,n-1, \tag{4.2}$$

$$\sum_{i=1}^{n} \left(A^T Au - A^T f\right)_i = 0, \tag{4.3}$$

$$p_i = \alpha \cdot \operatorname{sgn}((Du)_i) \quad \text{if} \quad (Du)_i \neq 0, \tag{4.4}$$

$$|p_i| \leq \alpha. \tag{4.5}$$

For simplicity, we will often in the following chapters denote (4.1) as P_α. The primal-dual solutions (u,p) are dependent of α, therefore we use the notation (u_α, p_α) as the solution to P_α.

In Chapter 4.1 we show that both the primal solution u_α and the dual solution p_α are continuously dependent of α. We will need this result later in Chapter 4.3, where we derive the lasso-path algorithm for total variation. At the end of Chapter 4.3 we revisit the assumption of uniqueness of solutions and explain why the assumption we made in Chapter 3.1 holds. Chapter 4.2 presents the main idea of the algorithm. In Chapter 4.4 and 4.5 we discuss computational simplifications and modifications we made.

4.1 Continuous Dependence of α

Lemma 4.1.1. A sequence $(v_k) \in \mathbb{R}^n$ converges to $v^* \in \mathbb{R}^n$ if and only if every subsequence $v_{k'}$ has a subsequence $v_{k''}$ that converges to v^*.

Proof. If a sequence (v_k) converges to v^* , then clearly every subsequence has a subsequence that also converges to v^*, namely the subsequence itself. To prove the converse, we assume that the sequence (v_k) does not converge to v^*. Then there exists an $\epsilon > 0$ and a subsequence $v_{k'}$ such that $\|v_{k'} - v^*\| > \epsilon$ and then $v_{k'}$ does not have a subsequence that converges to v^*, and we have a contradiction. $\square$

Theorem 4.1.2. Denote the minimization problem in (3.1), with regularization parameter α, as P_α. Assume P_α has a unique solution, and let u_α, p_α be the unique primal-dual solution. Then the function $\alpha \mapsto (u_\alpha, p_\alpha)$ is continuous.

Proof. Denote $(u_{\alpha_k}, p_{\alpha_k})$ as the unique primal-dual solutions of P_{α_k}. Let $\hat{\alpha} > 0$ and let $\alpha_k \to \hat{\alpha}$. Then (3.35)–(3.38) is satisfied for $\alpha = \alpha_k$. As we proved in the proof for Lemma 3.1.1, any solution u_α to P_α is bounded. In particular, $\|u_\alpha\|_1 \leq g_\alpha(\frac{1}{2}\|f\|_2^2)$ where the function g is defined in (3.14). We have $g_{\alpha_k}(\frac{1}{2}\|f\|_2^2) \to g_{\hat{\alpha}}(\frac{1}{2}\|f\|_2^2) < \infty$, therefore, the sequence $(u_{\alpha_k})_k$ is bounded. The sequence $(p_{\alpha_k})_k$ is clearly also bounded, because is satisfies (4.5). The Bolzano-Weierstrass theorem states that any bounded sequence in $\mathbb{R}^n$ has a convergent subsequence. We denote the subsequence of $(u_{\alpha_k}, p_{\alpha_k})$ as $(u_{\alpha_{k'}}, p_{\alpha_{k'}})$ and its limit as (u^*, p^*).

Now we need to show that (u^*, p^*) satisfies (3.35)–(3.38) for $\alpha = \hat{\alpha}$. We see that (3.35), (3.36) and (3.38) are clearly satisfied. If $(Du^*)_j \neq 0$ then there exists a k' large enough such that $(Du_{\alpha_{k'}})_j \neq 0$ while $\text{sgn}((Du^*)_j) = \text{sgn}((Du_{\alpha_{k'}})_j)$.

Since $u_{\alpha_{k'}}$ is a solution to $P_{\alpha_{k'}}$, we have that $(p_{\alpha_{k'}})_j = \alpha_{k'}\text{sgn}((Du_{\alpha_{k'}})_j)$ when $(Du_{\alpha_{k'}})_j \neq 0$ which in the limit gives us $(p^*)_j = \hat{\alpha}\text{sgn}((Du^*)_j)$. Hence all the optimality conditions (3.35)–(3.38) are satisfied, and (u^*, p^*) is a solution to $P_{\hat{\alpha}}$. Since we have assumed uniqueness of solutions, we have that $(u^*, p^*) = (u_{\hat{\alpha}}, p_{\hat{\alpha}})$.

We want to show that $(u_\alpha, p_\alpha) \to (u_{\hat{\alpha}}, p_{\hat{\alpha}})$ when $\alpha \to \hat{\alpha}$. We have shown that every subsequence $(u_{\alpha_k}, p_{\alpha_k})$ has a subsequence $(u_{\alpha_{k'}}, p_{\alpha_{k'}})$ that converges to $(u_{\hat{\alpha}}, p_{\hat{\alpha}})$. Then, by Lemma 4.1.1 we have shown that $(u_\alpha, p_\alpha) \to (u_{\hat{\alpha}}, p_{\hat{\alpha}})$, and hence proven continuity. $\square$

4.2 Main Idea

The lasso-path algorithm for total variation is an iterative algorithm. In each step it solves P_α for a range of α-values. More exactly, in the range of α-values, the algorithm finds a general expression for the solution, dependent of α. This results in an algorithm that solves P_α for all parameter values in the given range.

The algorithm works its ways through decreasing values of α, and for each iteration the number of jumps in the solution changes. The main idea is to start with a constant solution u_α which one obtains for all α-values greater than some lower bound, say α_0. Then for α-values lower than α_0, we find a general solution with one jump that is valid until we reach some lower bound, say α_1. Again, we find a new solution, now with two jumps, that is valid in a new interval. We continue this process until we reach the desired number of jumps in the solution, or until we reach some predetermined value of α.

Finding the new α-value for where the characteristics of the solution changes is done by finding the smallest step in the direction of decreasing α, for which the optimality conditions no longer hold. Then, because the solutions are continuously dependent of α, we find where the new jump appears or disappears. Also, we do not have to make the distinction between the solutions at these α-knot-points. That is, say u^1_α is the solution that is valid for $\alpha \geq \alpha_0$, and u^2_α is the solution for $\alpha_1 \leq \alpha \leq \alpha_0$, then we have $u^1_{\alpha_0} = u^2_{\alpha_0}$.

4.3 Derivation of Algorithm

We start by considering if a constant function can be a solution to (4.1). (One would expect this to be the case for really large values of α.) We see what the optimality conditions yield, and for which values of α a constant function is a solution.

Now let u be a constant function, which means we can write

$$u = c \cdot \mathbb{1}, \tag{4.6}$$

for some $c \in \mathbb{R}$. Then u is a minimizer of (4.1) if and only if the conditions in (4.2)–(4.5) are satisfied. Of course we don't need to check condition (4.4) since we have assumed u is constant and hence $(Du)_i = 0$ for all $i = 1, \dots, n-1$. We start with condition (4.3): we need

$$\sum_{i=1}^{n} \left(A^T A(c\mathbb{1}) - A^T f\right)_i = 0, \tag{4.7}$$

which is the same as

$$\mathbb{1}^T \left(A^T A(c\mathbb{1}) - A^T f\right) = c\|A\mathbb{1}\|_2^2 - \langle \mathbb{1}, A^T f\rangle = 0. \tag{4.8}$$

Hence for a constant function $u = c \cdot \mathbb{1}$ to be a minimizer we must have

$$c = \frac{\langle \mathbb{1}, A^T f\rangle}{\|A\mathbb{1}\|_2^2}. \tag{4.9}$$

From condition (4.5) and (4.2) we have

$$|p_j| \leq \alpha \quad \text{if and only if} \quad \Big|\sum_{i=1}^{j} \left(A^T A u - A^T f\right)_i\Big| \leq \alpha, \tag{4.10}$$

which by inserting for u, and then c, leads to

$$\left|\sum_{i=1}^{j} \left(\frac{\langle \mathbb{1}, A^T f\rangle}{\|A\mathbb{1}\|_2^2} \cdot A^T A\mathbb{1} - A^T f\right)_j\right| \leq \alpha. \tag{4.11}$$

Since we assume uniqueness of solutions, we have that (4.11) holds for a certain α if and only if the solution u is given by

$$u = \frac{\langle \mathbb{1}, A^T f\rangle}{\|A\mathbb{1}\|_2^2} \cdot \mathbb{1}. \tag{4.12}$$

Then the dual solution p is given by

$$p_j = \sum_{i=1}^{j} \left(\frac{\langle \mathbb{1}, A^T f\rangle}{\|A\mathbb{1}\|_2^2} \cdot A^T A \mathbb{1} - A^T f \right)_j . \tag{4.13}$$

Now we define

$$\alpha_0 = \max_j \left| \sum_{i=1}^{j} \left(\frac{\langle \mathbb{1}, A^T f\rangle}{\|A\mathbb{1}\|_2^2} \cdot A^T A \mathbb{1} - A^T f \right)_j \right| = \max_j |p_j|. \tag{4.14}$$

Then if $\alpha_0 \leq \alpha < \infty$, we know that u and p as given in (4.12) and (4.13) are the solutions to P_α.

Now, if $\alpha < \alpha_0$ we know that a constant function can no longer be a solution to (4.1), and must hence contain at least one jump. We assume that the maximum in (4.14) is obtained in a single coefficient. From this assumption we can only have that one jump appear at the time. Then, when $\alpha = \alpha_0$, because the solution (u_α, p_α) is continuously dependent of α, as shown in Chapter 4.1, we know that u_{α_0} obtains a jump in the index $\arg\max_j |p_j|$.

As we have seen, for α-values in the interval $[\alpha_0, +\infty)$, the solution to P_α remains the same, and with zero jumps. Now, as we find the α-value where the solution obtains a jump, we will see that this property will characterize the solution in the range $\alpha_1 < \alpha < \alpha_0$, where α_1 represents a value where the numbers of jumps in the solution changes again. In other words, the solution u_α will always have *one* jump when $\alpha_1 \leq \alpha < \alpha_0$. Then, for a given α_2, when $\alpha_2 \leq \alpha < \alpha_1$, the solution will have *two* jumps.

In general, in the range of α-values where the solution u_α has a given number of jumps, we remark that the actual solutions still changes. This is because we find an expression for the solution u_α dependent of α, the number of jumps, the sign of the jumps and where the jumps appear, which holds for the whole interval. Therefore, the main focus is to determine for which values of α the number of jumps of the solution changes, where the jumps appear and what sign the jumps have.

As in (4.14), where we needed the constant solution to find the α-value where the solution obtains a jump, we will always need the solution in the previous step to find the α-value where the number of jumps changes. That is, we need to work iteratively through the decreasing values of α and ”solve” the problem in each step.

In summary, we start with α so large that the solution is constant. From this constant solution we find α_0, the α-value where the solution obtains a jump. We formulate this new one-jump solution, depending on α, and find for which α-value a new jump appears. We continue this until we reach the desired α-value, or until we obtain the desired number of jumps.

We introduce the following notation to show which iteration we are in. Let the constant solution u discussed previously be denoted u^0, and its corresponding dual p^0.

Then

$$\alpha_0 = \max_i |p_i^0|, \tag{4.15}$$

$$i_1 = \arg\max_i |p_i^0|. \tag{4.16}$$

We also need to keep track of the sign of the jump that appear at i_1. We define

$$s_{i_1} = \operatorname{sgn}(p_{i_1}^0). \tag{4.17}$$

Hence, we have that

$$p_{i_1}^0 = \alpha s_{i_1}. \tag{4.18}$$

Now, our goal is to find an expression for u_α^1 and find α_1, such that u_α^1 is a valid solution to P_α, when $\alpha_1 \le \alpha \le \alpha_0$. We know that for $\alpha \in [\alpha_1, \alpha_0]$, the solution u_α^1 has a jump at i_1.

Let $I_1^1 = [1, \dots, i_1]$ and $I_2^1 = [i_1 + 1, \dots, n]$ denote the intervals over which u_α^1 is constant. Also, let $\mathbb{1}_I$ denote the $n \times 1$ vector with ones on the interval I, and zeros otherwise:

$$(\mathbb{1}_I)_i = \begin{cases} 0 & \text{if} \quad i \notin I, \\ 1 & \text{if} \quad i \in I, \end{cases} \tag{4.19}$$

or, in vector form

$$\mathbb{1}_I = [0, \dots, 0, \underbrace{1, \dots, 1}_{I}, 0, \dots, 0]^T. \tag{4.20}$$

We use the notation $c_1^1(\alpha)$ and $c_2^1(\alpha)$ to denote the parameter dependent coefficients of u_α^1, such that we can write

$$u^1(\alpha) = c_1^1(\alpha)\mathbb{1}_{I_1^1} + c_2^1(\alpha)\mathbb{1}_{I_2^1} = \sum_{j=1}^{2} c_j^1(\alpha)\mathbb{1}_{I_j^1}. \tag{4.21}$$

Now if we multiply (3.24) by each of the vectors $\mathbb{1}_{I_1^1}, \mathbb{1}_{I_2^1}$ we get

$$\begin{aligned} \mathbb{1}_{I_1^1}^T D^T p = -p_{i_1} = -\alpha s_{i_1} = \mathbb{1}_{I_1^1}^T \left(A^T f - A^T A u\right), \\ \mathbb{1}_{I_2^1}^T D^T p = p_{i_1} = \alpha s_{i_1} = \mathbb{1}_{I_2^1}^T \left(A^T f - A^T A u\right). \end{aligned} \tag{4.22}$$

Inserting for (4.21), we get

$$\begin{aligned} -\alpha s_{i_1} = \mathbb{1}_{I_1^1}^T \left(A^T f - c_1^1(\alpha) A^T A \mathbb{1}_{I_1^1} - c_2^1(\alpha) A^T A \mathbb{1}_{I_2^1}\right), \\ \alpha s_{i_1} = \mathbb{1}_{I_2^1}^T \left(A^T f - c_1^1(\alpha) A^T A \mathbb{1}_{I_1^1} - c_2^1(\alpha) A^T A \mathbb{1}_{I_2^1}\right). \end{aligned} \tag{4.23}$$

Now we can solve for $c_1^1(\alpha), c_2^1(\alpha)$. We get

$$\begin{pmatrix} c_1^1(\alpha) \\ c_2^1(\alpha) \end{pmatrix} = S_1^{-1} \left[\begin{pmatrix} \mathbb{1}_{I_1^1}^T A^T f \\ \mathbb{1}_{I_2^1}^T A^T f \end{pmatrix} + \alpha \begin{pmatrix} s_{i_1} \\ -s_{i_1} \end{pmatrix} \right], \tag{4.24}$$

where

$$S_1 = \begin{bmatrix} \mathbb{1}_{I_1^1}^T A^T A \mathbb{1}_{I_1^1} & \mathbb{1}_{I_1^1}^T A^T A \mathbb{1}_{I_2^1} \\ \mathbb{1}_{I_2^1}^T A^T A \mathbb{1}_{I_1^1} & \mathbb{1}_{I_2^1}^T A^T A \mathbb{1}_{I_2^1} \end{bmatrix}. \tag{4.25}$$

(Here the subscript 1 in S_1 represents the iteration index.) To shorten the notation, we define the vectors

$$c^1(\alpha) = \begin{pmatrix} c_1^1(\alpha) \\ c_2^1(\alpha) \end{pmatrix}, \quad d^1 = S_1^{-1} \begin{pmatrix} \mathbb{1}_{I_1^1}^T A^T f \\ \mathbb{1}_{I_2^1}^T A^T f \end{pmatrix} \quad \text{and} \quad e^1 = S_1^{-1} \begin{pmatrix} s_{i_1} \\ -s_{i_1} \end{pmatrix}, \tag{4.26}$$

so we have

$$c^1(\alpha) = d^1 + \alpha e^1. \tag{4.27}$$

Now if we insert $\alpha = \alpha_0$ in (4.27), we get the constants c_1^1, c_2^1, which would give us the one-jump solution to P_{α_0}. But this is not our goal. We continue by finding the expression for u_α^1. By inserting (4.27) into (4.21), we get

$$u^1(\alpha) = \sum_{j=1}^{2} \left(d_j^1 + \alpha e_j^1\right) \mathbb{1}_{I_j^1} \tag{4.28}$$

To shorten notation we define $\hat{d}^1 = \sum_{j=1}^{2} d_j^1 \mathbb{1}_{I_j^1}$ and $\hat{e}^1 = \sum_{j=1}^{2} e_j^1 \mathbb{1}_{I_j^1}$. Inserting $u^1(\alpha)$ into (4.2), we have

$$\begin{aligned} p_j^1(\alpha) &= \sum_{i=1}^{j} \left(A^T A u^1(\alpha) - A^T f\right)_i = \\ &= \sum_{i=1}^{j} \left(A^T A \hat{d}^1\right)_i + \alpha \sum_{i=1}^{j} \left(A^T A \hat{e}^1\right)_i - \sum_{i=1}^{j} \left(A^T f\right)_i, \end{aligned} \tag{4.29}$$

which we shorten as

$$p_j^1(\alpha) = \psi_j^1 + \alpha \xi_j^1 - \varphi_j. \tag{4.30}$$

Now we have found the solution u_α^1 and p_α^1 for $\alpha_1 \le \alpha \le \alpha_0$, where α_1 remains to be found. We know condition (4.5) is satisfied in that interval, i.e.,

$$|p_j^1(\alpha)| \le \alpha \quad \text{when} \quad \alpha \in [\alpha_1, \alpha_0]. \tag{4.31}$$

To find where the solution obtains a new jump, that is, to find α_1, we must solve

$$\|p^1(\alpha)\|_\infty = \alpha \quad \text{for } \alpha < \alpha_0. \tag{4.32}$$

That is, we need to find the largest $\alpha < \alpha_0$ such that $|p_j^1(\alpha)| = \alpha$ for some j. Specifically, we solve

$$p_j^1(\alpha) = \pm\alpha \tag{4.33}$$

for α for each $j = 1, \ldots., n-1$. Using the notation in (4.30), we denote the solutions as

$$\hat{\alpha}_j^{\pm} = \frac{\psi_j^1 - \phi_j^1}{\pm 1 - \xi_j^1}. \tag{4.34}$$

The values of $\hat{\alpha}_j^{\pm}$ are the α-values for when the corresponding constraint (4.5) holds with equality. Then, since $p(\alpha)$ is linear in α, the largest of the $\hat{\alpha}_j^{\pm}$-values correspond to the first element in the vector p for which the restriction (4.5) does not hold when we decrease α, and in turn the corresponding index at which we obtain a new jump. Hence, we set

$$\alpha_1 = \max_i \{|\hat{\alpha}_i^{\pm}| : 0 < \hat{\alpha}_i^{\pm} < \alpha_0\}. \tag{4.35}$$

In that case, we obtain a new jump in index j, where

$$j = \arg\max_i \{|\hat{\alpha}_i^{\pm}| : 0 < \hat{\alpha}_i^{\pm} < \alpha_0\}, \tag{4.36}$$

and the sign of the jump is

$$s_j = \operatorname{sgn}(p_j^1). \tag{4.37}$$

Now we follow the same procedure as is the previous iteration: we find the index and the sign of the new jump, and define the new expression for the solution $u^2(\alpha)$. From this we derive the S_2-matrix, and the d^2- and e^2-vectors. We then find $c^2(\alpha)$, $u^2(\alpha)$ and then $p^2(\alpha)$. By solving $\|p^2(\alpha)\|_\infty = \alpha$ for $\alpha < \alpha_1$, we find α_2, where the solution obtains a new jump. Then we know $u^2(\alpha)$ is valid for $\alpha \in [\alpha_2, \alpha_1]$

To generalize, we now assume that we are in a later iteration r where we have found that $u^r(\alpha)$ has k jumps in the indices $i_1 < \ldots < i_k$, with signs $s_{i_1}, \ldots, s_{i_k}$. (The reason we do not assume $k = r$, is because a jump may disappear. We will discuss this case shortly.) Then u is piecewise constant over $k+1$ intervals. We denote these intervals as I_j^r, for $j = 1, \ldots, k+1$. We define the interval $I_j^r := [i_{j-1}+1, \ldots, i_j]^T$ for $j = 2, \ldots, k-1$. The first and last intervals are, respectively, $I_1^r = [1, \ldots, i_1]^T$ and $I_{k+1}^r = [i_k+1, \ldots, n]^T$. Then we can write u as

$$u^r(\alpha) = \sum_{j=1}^{k+1} c_j^r(\alpha) \mathbb{1}_{I_j^r}. \tag{4.38}$$

Following the same argumentation as before, we get the system

$$\begin{pmatrix} c_1^r(\alpha) \\ \vdots \\ c_{k+1}^r(\alpha) \end{pmatrix} = S_r^{-1} \left[\begin{pmatrix} \mathbb{1}_{I_1^r} A^T f \\ \vdots \\ \mathbb{1}_{I_{k+1}^r} A^T f \end{pmatrix} + \alpha \begin{pmatrix} s_{i_1} \\ \vdots \\ s_{i_j} - s_{i_{j-1}} \\ \vdots \\ -s_{i_k} \end{pmatrix} \right], \tag{4.39}$$

where

$$S_r = \begin{bmatrix} \mathbb{1}_{I_1^r}^T A^T A \mathbb{1}_{I_1^r} & \cdots & \mathbb{1}_{I_1^r}^T A^T A \mathbb{1}_{I_{k+1}^r} \\ \vdots & \ddots & \vdots \\ \mathbb{1}_{I_{k+1}^r}^T A^T A \mathbb{1}_{I_1^r} & \cdots & \mathbb{1}_{I_{k+1}^r}^T A^T A \mathbb{1}_{I_{k+1}^r} \end{bmatrix}. \tag{4.40}$$

Again, we define the vectors

$$c^r(\alpha) = \begin{pmatrix} c_1^r(\alpha) \\ \vdots \\ c_{k+1}^r(\alpha) \end{pmatrix}, \ d^r = S_r^{-1} \begin{pmatrix} \mathbb{1}_{I_1^r} A^T f \\ \vdots \\ \mathbb{1}_{I_{k+1}^r} A^T f \end{pmatrix}, \ e^r = S_r^{-1} \begin{pmatrix} s_{i_1} \\ \vdots \\ s_{i_j} - s_{i_{j-1}} \\ \vdots \\ -s_{i_k} \end{pmatrix}, \tag{4.41}$$

so (4.39) becomes

$$c^r(\alpha) = d^r + \alpha e^r. \tag{4.42}$$

Define $\hat{d}^r = \sum\limits_{j=1}^{k+1} d_j^r \mathbb{1}_{I_j^r}$ and $\hat{e}^1 = \sum\limits_{j=1}^{k+1} e_j^r \mathbb{1}_{I_j^r}$. Then $u^r(\alpha) = \hat{d}^r + \alpha \hat{e}^r$, and

$$\begin{aligned} p_j^r(\alpha) &= \sum_{i=1}^{j} \left(A^T A \hat{d}^r \right)_i + \alpha \sum_{i=1}^{j} \left(A^T A \hat{e}^r \right)_i - \sum_{i=1}^{j} \left(A^T f \right)_i \\ &= \psi_j^r + \alpha \xi_j^r - \varphi_j. \end{aligned} \tag{4.43}$$

Say $u^r(\alpha)$ is the solution for $\alpha \in [\alpha_r, \alpha_{r-1}]$ where α_r is still unknown. Denote α_s as the α-value calculated by solving $\|p^r(\alpha)\|_\infty = \alpha$ for $\alpha < \alpha_{r-1}$. The solution yields

$$\alpha_s = \max_i \{ |\hat{\alpha}_i^\pm| : 0 < \hat{\alpha}_i^\pm < \alpha_{r-1} \}, \tag{4.44}$$

where

$$\hat{\alpha}_i^\pm = \frac{\psi_i^r - \phi_i^r}{\pm 1 - \xi_i^1}. \tag{4.45}$$

That is, α_s is the value of α where the next jump appears. Denote the index where the jump appears as

$$\hat{\jmath} = \arg\max_i \{ |\hat{\alpha}_i^\pm| : 0 < \hat{\alpha}_i^\pm < \alpha_{r-1} \}, \tag{4.46}$$

and the sign of the jumps as

$$s_{\hat{\jmath}} = \begin{cases} +1 & \text{if } \alpha_s \text{ was obtained from } \hat{\alpha}_i^+, \\ -1 & \text{if } \alpha_s \text{ was obtained from } \hat{\alpha}_i^-. \end{cases} \tag{4.47}$$

Lastly, we need to consider that a jump may disappear. We might for some α-value have that two adjacent constants in $c^r(\alpha)$ are equal. That is, the jump between them disappears. If we have that $c_j^r(\alpha_q) = c_{j+1}^r(\alpha_q)$ for $\alpha_q < \alpha_{r-1}$, we know that the corresponding jump at i_j is the first to disappear when we are decreasing α.

Therefore, in each iteration we need to calculate the $\tilde{\alpha}_j$s that solve

$$c_j^r(\alpha) = c_{j+1}^r(\alpha), \tag{4.48}$$

for $j = 1, \ldots, k$, given that $u^r(\alpha)$ has k jumps. The solution is

$$\tilde{\alpha}_j = \frac{d_j^r - d_{j+1}^r}{e_{j+1}^r - e_j^r} \quad \text{for} \quad j = 1, \ldots, k, \tag{4.49}$$

and hence

$$\alpha_q = \max_j\{\tilde{\alpha}_j : \tilde{\alpha}_j < \alpha_{r-1}\}, \tag{4.50}$$

and

$$i_j = \arg\max_j\{\tilde{\alpha}_j : \tilde{\alpha}_j < \alpha_{r-1}\}, \tag{4.51}$$

where the maximum is only over indices of current jumps. To sum up, say we are in iteration r. Let α_s, as defined in (4.44), denote the α-value where the next jump appear, and α_q, as defined in (4.50), denote the α-value where the next jump disappear. Define

$$\alpha_r = \max\{\alpha_s, \alpha_q\}. \tag{4.52}$$

Then the solution $u^r(\alpha)$ is valid for $\alpha \in [\alpha_r, \alpha_{r-1}]$ If max is attained for α_q we must remove the jump at index i_j. If it is attained for α_s, we add a new jump at index $\hat{j}$ with sign $s_{\hat{j}}$ as defined in (4.47). We continue the iterations until we reach a desired α-value, or until we obtain the desired number of jumps. Using the latter as a stopping criterion one must keep in mind that jumps might be deleted, so one can reach a given number of jumps in the solution several times.

Remark. We revisit the question of unique minimizers. Before we derived the lasso-path algorithm for total variation, at the end of Chapter 3.1 we assumed that the minimizer was always unique, even when A was not invertible. Now, with more insight to the solution method, we see that this is almost correct. We know that we have a unique solution as long as S is invertible. If we add the assumption that rank$(A) = m$, where $m \leq n$, we can expect that S is invertible as long as the solution u has no more than m jumps. So if we restrict the algorithm to stop when the solution has reached m jumps, we know that the solutions are unique for each α.

4.4 Simplifications

We bring attention to a computational simplification we can use when calculating the new S matrix. As we notice, each row and column in S corresponds to an interval I_{i_j}. By correspond, we mean that I_{i_j} is used to calculate each element in the row and column. Specifically, an interval I_{i_j} corresponds to the jth row and column of S.

When we obtain a new jump in the solution, say at node l where $i_{j-1} < l < i_j$, where we already have jumps at i_{j-1} and i_j, we have to split the interval I_j into two. When it comes to calculating S, this means that we only have to update the row and column corresponding to I_j, namely row j and column j. Since we gain a jump, we have to "split" row/column j by inserting an adjacent row and column. Then we insert the correct calculations in the two rows and columns corresponding to the old row j and column j.

In the event we lose a jump, we have to delete the corresponding row and column, and update the row below and the column to the right.

Another trait worth noticing is that u_α is linearly dependent of α. It follows that we only need to store the solutions at the nodes where the number of jump changes. For example, say we have the solutions u_{α_i} and $u_{\alpha_{i+1}}$. Then we can find the solution u_α for any α such that $\alpha_{i+1} < \alpha < \alpha_i$ by

$$u(\alpha) = \frac{\alpha - \alpha_{i+1}}{\alpha_i - \alpha_{i+1}} u_{\alpha_i} + \frac{\alpha_i - \alpha}{\alpha_i - \alpha i + 1} u_{\alpha_{i+1}}. \tag{4.53}$$

4.5 Computational Modifications

We present briefly the modifications needed to make the code work properly.

A problem we ran into was that when a jump was added, it was immediately deleted in the following iteration, or if a jump was deleted, it was added back again in the next iteration. This was probably due to rounding error. In the case where a jump is added to the solution, say where the solution is valid for $\alpha \in [\alpha_{k+1}, \alpha_k]$, the size of the jump is 0 for $\alpha = \alpha_k$. Therefore, when we calculate $\tilde{\alpha}$-values for when jump disappears, we *should* find that the $\tilde{\alpha}$-value corresponding to the newly added jump is equal to α_k, but because of rounding error it is slightly smaller. Hence, it was chosen as the next α-value, α_{k+1}, and the jump was deleted again. We solved this problem by demanding that a newly added or deleted jump could not, respectively, be deleted or added again in the next iteration. In the implementation we solved this by making a temporary index containing the newly added or deleted jump, and then force to ignore this index in the next iteration by setting its corresponding $\hat{\alpha}$- or $\tilde{\alpha}$ value to $-\infty$.

We also experienced that $\|p\|_\infty \leq \alpha$ was not always satisfied, and that we obtain a jump in the wrong direction in the solution. These problems can arise from the same circumstance: say two adjacent values of p are almost equal and gives the α-value for when a new jump is added, that is, $p_j(\alpha) \approx p_{j+1}(\alpha) \approx \alpha$. Then we have two cases that can go wrong. Either, the solution is supposed to obtain two new jumps, in j and $j+1$, where the first jump appears for α_k and the second for $\alpha_{k+1} = \alpha_k - \epsilon$ for some $\epsilon > 0$. What might happen, is that in the given iteration the second jump is added and the first jump is not found. Then, in the next iteration, the first jump might have a corresponding $\hat{\alpha}$-value that is larger than the α-value in the previous iteration, and therefore that jump is never found.

The other case is that the solution is only supposed to obtain one of the jumps, but due to rounding error the algorithm adds the wrong jump, which resulted in a solution with a jump in the wrong direction.

Both of these cases results in an invalid solution. To solve this problem, we check in each iteration that $\|p\|_\infty \leq \alpha$ is satisfied. If we find a value of p which is greater than α, we add the corresponding index to the solution. We also check that the sign of the reconstructed jumps are the same as the sign of the corresponding values of p. If that is not the case in an index, we remove the jump from the index set and update the solution.

Chapter 5

Taut String Algorithm

In this chapter we derive the taut string algorithm. We remind the reader that the minimization problem we are solving is

$$\min_u \quad \frac{1}{2}\|u-f\|_2^2 + \alpha\|Du\|_1. \tag{5.1}$$

We start by stating the optimality conditions in Chapter 5.1, before we continue with deriving a new, equivalent problem. In Chapter 5.2 we finish deriving the algorithm. We also discuss the complexity of the algorithm and compare it to the lasso-path algorithm.

5.1 Optimality Conditions

By inserting for $A = I$ into the optimality conditions (3.35)–(3.38), we see that the conditions (3.37) and (3.38) remain the same, but (3.35) and (3.36) simplify to

$$0 = \sum_{j=1}^{i} u_j - \sum_{j=1}^{i} f_j - p_i \qquad \text{for } i = 1, \dots, n-1, \tag{5.2}$$

$$0 = \sum_{j=1}^{i} u_j - \sum_{j=1}^{i} f_j \qquad \text{for } i = n. \tag{5.3}$$

We denote $F_u(i) := \sum_{j=1}^{i} u_j$ and $F_f(i) := \sum_{j=1}^{i} f_j$. Our aim is to derive a new problem depending on F_u and then solve for u. We define $F_u(0) = F_f(0) = 0$ such that we can find

$$u_i = F_u(i) - F_u(i-1) \quad \text{for } i = 1, \dots, n. \tag{5.4}$$

With the new notation (5.2) and (5.3) become

$$0 = F_u(i) - F_f(i) - p_i \qquad \text{for } i = 1, \dots, n-1, \tag{5.5}$$

$$0 = F_u(i) - F_f(i) \qquad \text{for } i = n. \tag{5.6}$$

We will use (5.5) and (5.6) to derive two other conditions depending on $(Du)_i$. First, we know that if $(Du)_i \neq 0$ then $p_i \in |(Du)_i|/(Du)_i$ is either $-\alpha$ or $+\alpha$. Hence we obtain for $i = 1, \dots, n-1$

$$F_u(i) = F_f(i) + p_i, \tag{5.7}$$

or, more specifically,

$$F_u(i) = \begin{cases} F_f(i) + \alpha & \text{if } (Du)_i > 0, \\ F_f(i) - \alpha & \text{if } (Du)_i < 0. \end{cases} \tag{5.8}$$

If $(Du)_i = 0$ we obtain

$$|F_f(i) - F_u(i)| \leq \alpha \quad \text{for } i = 1, \dots, n-1. \tag{5.9}$$

The conditions derived here characterize a new problem. Finding a minimizer u to the original problem (5.1) is equivalent to finding $F_u \in \mathbb{R}^{n+1}$ such that these conditions are satisfied. We state this as a theorem:

Theorem 5.1.1. The vector u is a minimizer of (5.1) if and only if we have that $u_i = F_u(i) - F_u(i-1)$, where $F_u \in \mathbb{R}^{n+1}$ satisfies

T1 $F_u(0) = F_f(0) = 0$,

T2 $F_u(n) = F_f(n)$,

T3 $F_u(i) = \begin{cases} F_f(i) + \alpha & \text{if } (Du)_i > 0 \\ F_f(i) - \alpha & \text{if } (Du)_i < 0, \end{cases} \quad \text{for } i = 1, \dots, n-1,$

T4 $|F_f(i) - F_u(i)| \leq \alpha \quad \text{for } i = 1, \dots, n-1.$

Reinterpreting the Solution Vectors

Until now we have regarded, appropriately, u and F_u as vectors with respectively n and $n+1$ elements. To gain more insight to our problem we want to change this perception of u and F_u. We do this to make the upcoming argumentation easier to follow. Additionally it allows for some explanatory figures which also help give us a better understanding.

Thus, without loss of generality, we interpret in the following u and F_u as (non discrete) functions on the domain $[0, n]$. More specifically, u is regarded as a piecewise constant spline function, and F_u as a piecewise linear spline function. We take u such that it is constant between the nodes $0, 1, \dots, n$. Hence for the ith element of the vector u, u_i, we have that the function u is constant equal to u_i on $(i-1, i]$, for $i = 1, \dots, n$. For F_u we have that the function takes the value $F_u(i)$ in node i, and is linear between the nodes $0, 1, \dots, n$. Hence the function F_u can be interpreted as the linear interpolation between the points $F_u(i)$, for $i = 0, \dots, n$.

The relation $F_u(i) = \sum_{j=1}^{i} u_j$ implies that u_i is the slope of F_u between node i and $i+1$. Hence u_i represent the derivative of F_u between node i and $i+1$. Analogously, f and F_f can be interpreted in the same way as u and F_u, respectively.

5.2 Derivation of the Method

With this new interpretation of u, F_u, f and F_f in mind, we go through and reinterpret the conditions **T1-T4**. Condition **T1-T2** states that the function F_u starts in the point $(0, 0)$ and ends in $(n, F_f(n))$. From **T3-T4** we read that $F_u(i)$ must lie in a tube of diameter 2α centered around $F_f(i)$ for $i = 1, \ldots, n-1$. Since F_u is linear between the nodes we can ”close” the tube with a linear spline between the end points and the upper and lower bounds, and state that the solution F_u must lie in this closed tube. See figure 5.1, left, for an illustration of such a tube.

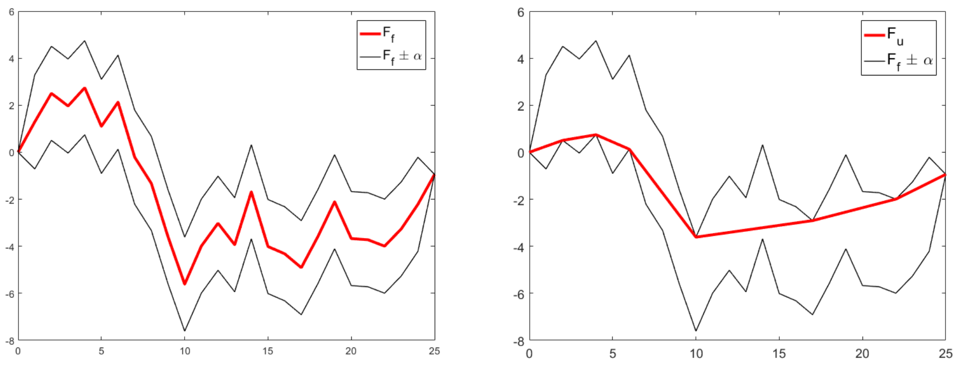


Figure 5.1: The figure to the right shows an example of a tube with radius $\alpha = 2$ centered around F_f, where F_f is constructed from 25 randomly generated data points f_i between -2.5 and $+2.5$. The black lines represent the walls of the tube, $F_f \pm \alpha$, and the red line is the center of the tube, F_f. In the figure to the right, the black lines represent the same walls as in the figure to the left. The red line represents the solution F_u, which resembles a taut string.

Here we have used 25 randomly generated numbers between -2.5 and $+2.5$ for the data points f_i. We set $\alpha = 2$, and included the walls of the tube, $F_f + \alpha$ and $F_f - \alpha$ as well as the center F_f.

It is shown in [3] that finding the minimizer u in Theorem 5.1.1 is equivalent to minimizing the length of the graph of F_u inside the tube. Therefore we can interpret F_u as a taut string fixed at the end points of the tube. This is the reason the algorithm we are deriving is called the *taut string algorithm*. Figure 5.1, right, gives an illustration of the solution F_u to the tube in figure 5.1, left.

Next, we consider a node i at which the slope of F_u changes. This is equivalent to u changing value, or $(Du)_i \neq 0$. If $(Du)_i > 0$ we know that u is monotonically increasing near i. Then it follows from Theorem 2.5 that F_u is convex at node i. For $(Du)_i < 0$ we use the same argument to show that u is monotonically decreasing which implies that F_u is concave at i. At the same time, if $(Du)_i \neq 0$, we know from **T3** that $F_u(i)$ is on the boundary of the tube, i.e., $F_u(i) = F_f(i) + \alpha$ if $(Du)_i > 0$ and $F_u(i) = F_f(i) - \alpha$ if $(Du)_i < 0$.

The preceding derivation only holds for points where the slope of F_u changes. Since F_u is a piecewise linear spline those points are the only points we need to describe F_u. We will from now on call them *knot points*. Between the knot points F_u is linear with slope given by u. Hence, since all knot points represent a change in slope of F_u, we know that all knot points must lie on the boundary of the tube, and that F_u is convex or concave in that knot point, as discussed in the previous paragraph.

The main idea of the taut string algorithm is to find F_u by working with two functions that serve as an upper and lower bound to F_u. Lets call the upper bound F_u^+ and the lower bound F_u^-. The algorithm works iteratively through the data points f_i and builds the functions F_u^+, F_u^- and F_u from left to right through a dynamic process.

Construction of F_u^+ and F_u^-

The upper and lower bound F_u^+ and F_u^- are only defined on the part of the interval where F_u is not yet constructed. They must for the most part satisfy the same conditions as F_u, with a few exceptions, which we will name shortly. Assume in the following that F_u has been found on $[0, j]$.

For F_u^+ and F_u^- condition **T1** becomes $F_u^+(j) = F_u^-(j) = F_u(j)$ when F_u is constructed on $[0, j]$. The upper bound F_u^+ shall only have knot points on the upper wall of the tube, while F_u^- will only have knot points on the lower wall. Then it follows that F_u^+ must be convex, and F_u^- must be concave.

For each node $2 \leq i \leq n-1$ the algorithm adds the boundary point $(i, F_f(i)+\alpha)$ as knot point to F_u^+ and the boundary point $(i, F_f(i)-\alpha)$ to F_u^-. After a new knot point is added we need to make sure that F_u^+ is still convex, and F_u^- is still concave.

We start with initializing $F_u(0) = F_u^+(0) = F_u^-(0) = 0$. In the first step we add the points $(1, F_f(1)+\alpha)$ and $(1, F_f(1)-\alpha)$ to F_u^+ and F_u^-, respectively. Then F_u^+ and F_u^- will always be linear, and checking for convexity and concavity is not necessary.

For the second iteration we will only discuss F_u^+. First we add the linear interpolation with the knot point $(2, F_f(2)+\alpha)$ to F_u^+. Then we must check if F_u^+ is convex. If that is not the case, we have a contradiction with F_u^+ being a convex function, and we remove the second knot point of F_u^+, and calculate a new linear interpolation between $F_u^+(0)$ and $F_u^+(2)$, which we set equal to F_u^+.

The same approach is repeated for all nodes: after $(i, F_f(i)+\alpha)$ is added to F_u^+ we go to the node to the left, $i-1$, and check for convexity. If F_u^+ is not convex around $i-1$ we delete the knot point. Then we compute a new interpolation between the penultimate knot point and $(i, F_f(i)+\alpha)$. Then we check again whether the new function F_u^+ is convex. We repeat the process until convexity of F_u^+ is achieved.

Analogously, we do the same computation for F_u^-, but instead of adding $(i, F_f(i)+\alpha)$ we add $(i, F_f(i)-\alpha)$, and we check for concavity instead of convexity.

Figure 5.2 and 5.3 shows an example of how F_u^+ is computed. Here we have only included a part of the upper tube wall (the blue line) as a reference point. Figure 5.2 is an example of an iteration where F_u^+ remains convex when a new knot point is added.

Figure 5.3 shows the opposite. Starting in figure 5.3a and adding the next knot point in figure 5.3b makes F_u^+ no longer convex. We see the intermediate steps taken in order to make F_u^+ convex again in figure 5.3c and 5.3d.

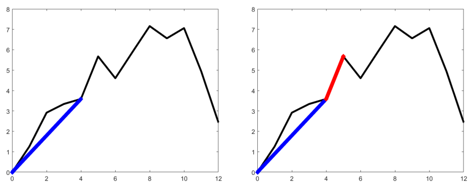
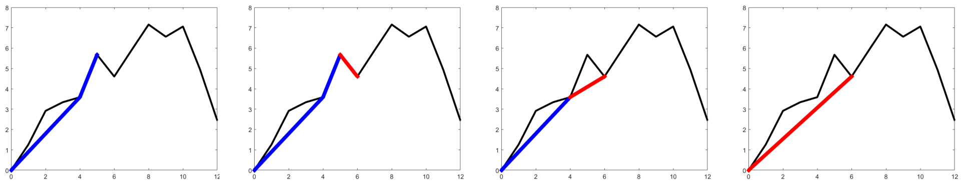


Figure 5.2: The figures show one iteration in the construction of F_u^+ where F_u^+ remains convex when a new knot point is added. To the left we have F_u^+ (the blue line) before the iteration. To the right the red and blue line together make up F_u^+. The red line represents the newly added line segment. In both pictures the black line shows a section of the upper wall of the tube.



(a) **(b)** **(c)** **(d)**

Figure 5.3: The figures show one iteration in the construction of F_u^+ where convexity of F_u^+ is not automatically remained when a new knot point is added. The figure to the left shows F_u^+ (the blue line) before the iteration. The two figures in the middle show the intermediate steps we need to make to ensure convexity. Here both the red and the blue line together make up F_u^+. Convexity is achieved in the figure to the right. Here F_u^+ is represented by the red line. In all the figures the black line shows a section of the upper wall of the tube.

Assume now that we are in iteration i, where $i > j$, and that F_u^+ and F_u^- have been computed over the interval $[j, i]$. Then F_u^+ is the largest convex minorant of the upper wall of the tube on the interval $[j, i]$, and F_u^- is the smallest concave majorant of the lower wall of the tube on the interval $[j, i]$.

Extension of F_u

Now assume that

$$F_u^+ < F_u^- \tag{5.10}$$

somewhere on $[j, i]$, and that this hasn't happened before for the same j. Then we can find the next line segment to our solution F_u.

To find this new line segment of F_u we need to make some observations. By the convexity of F_u^+, the concavity of F_u^- and the fact that $F_u^+(j) = F_u^-(j) = F_u(j)$, we know that (5.10) can only occur if $F_u^+ < F_u^-$ on $(j, j+1]$. This implies that either F_u^+ or F_u^- has been changed on the interval $(j, j+1]$ in the last iteration, which means we have deleted all knot points between j and i. The function this holds for will consequently be a straight line between j and i.

For the rest of the derivation we assume this is the case for F_u^+. We do this to make the argumentation easier to follow, and the theory is easily extended to the other case. Hence, F_u^+ is now a function that is linear between the nodes j and i.

Denote j' the first knot point of F_u^- after j. Then we have that

$$F_u^+(j') < F_u^-(j'), \tag{5.11}$$

and since $F_u^-(j')$ is on the lower wall of the tube, we know that F_u^+ has to lie partly outside of the tube.

We want to find the next knot in F_u, and decide on which wall it lies. We denote the next knot point as $\hat{\jmath}$. Then F_u is linear between j and $\hat{\jmath}$, and since all knot points lie on one of the walls, we have that

$$\begin{aligned} F_u(\hat{\jmath}) &= F_f(\hat{\jmath}) + \alpha \quad \text{or} \\ F_u(\hat{\jmath}) &= F_f(\hat{\jmath}) - \alpha. \end{aligned} \tag{5.12}$$

Now we need to decide at which node $\hat{\jmath}$ has to be. If $\hat{\jmath} \geq j'$, then we must have that $F_u(j') \geq F_u^-(j')$, else F_u would lie partly outside of the tube.

We consider the case $\hat{\jmath} \geq i$. Since F_u and F_u^+ are in this case linear on $[j, i]$ we must always have that $F_u(i) \leq F_u^+(i)$. If not, F_u would lie partly outside the upper wall of the tube, since all knot points of F_u^+ are on the upper wall. But because of (5.11), that would imply that $F_u(j') \leq F_u^+(j') < F_u^-(j')$, i.e, F_u would lie outside the lower wall of the tube. Hence we must have $\hat{\jmath} < i$.

The only possible knot point for F_u on the upper wall would be $(i, F_u^+(i))$, since all knot points to the left have been removed since they contradict **T1-T4**. We have thus proved that all knots $\hat{\jmath} > i$ are invalid, and therefore the only remaining knot points to be considered will lie on the lower wall.

Because of the concavity of F_u^-, the next node of F_u, $\hat{\jmath}$, cannot lie between j' and i, because then we would have that $F_u(\hat{\jmath}) < F_u^-(j')$, and hence F_u would lie partly outside of the tube. Also $\hat{\jmath}$ cannot come before j', since j' is already constructed as the first knot after j on the lower wall which doesn't contradict **T1-T4**. The only possible next knot point for F_u is therefore j'.

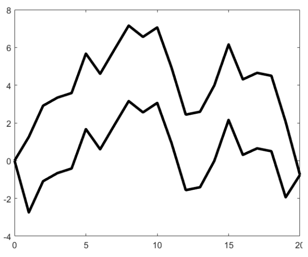
In conclusion, if F_u^+ is linear and $F_u^+ < F_u^-$, we can add the knot point $(j', F_u^-(j'))$ to F_u, where j' is the first knot point of F_u^- after j.

After the new line segment is appended to F_u, we need to update F_u^+ and F_u^-. First we remove the knot point $(j, F_u(j))$ from both F_u^+ and F_u^-. Then we set F_u^+ equal to the interpolation between $(j', F_u(j'))$ and $(i, F_u^+(i))$.

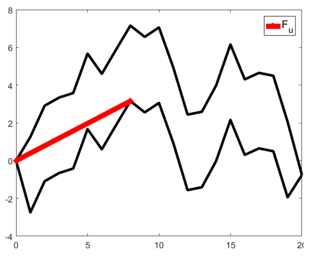
Hence, j' is now the new start point for F_u^+ and F_u^-. The function F_u^+ is still a straight line, and if $F_u^+ < F_u^-$ in $(j', j'+1]$, the procedure of extending F_u has to be repeated.

The case considered here was that F_u^+ is a function which is linear between j and i. The other possibility when (5.10) occurs, is that F_u^- is linear. The same arguments hold for this case. Hence, the second knot point of F_u^+ is added to F_u. The knot point $(j, F_u(j))$ is removed from F_u^+ and F_u^-, and F_u^- is equal to the interpolation between the last knot point of F_u and $(i, F_u^-(i))$.

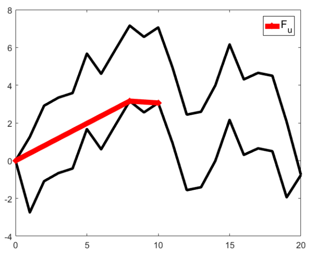
Figure 5.4 shows an example of the construction of F_u including the intermediate values of F_u^+ and F_u^-. The figures on the top show how F_u looks when F_u^+ and F_u^- are constructed as illustrated in the figures on the bottom.



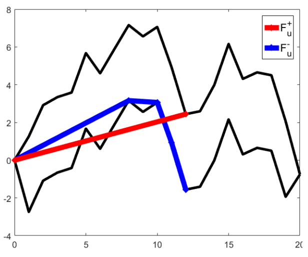
(a)



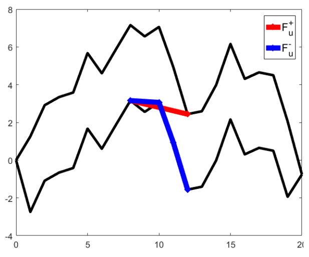
(b)



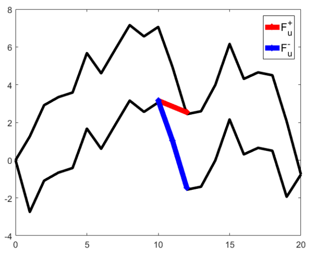
(c)



(d)



(e)



(f)

Figure 5.4: The figures on the top show the construction of F_u (the red line), parallel to the update of F_u^+ and F_u^- in the bottom figures. Figure 5.4d shows an example of the first time F_u^+ (the red line) is linear and lies below F_u^- (the blue line). Then figure 5.4b shows the first line segment of F_u, and then F_u^+ and F_u^- is updated in figure 5.4e. In figure 5.4c a new line segment is added to F_u, before F_u^+ and F_u^- is updated in figure 5.4f.

Computational Simplification

Now we have derived the concept of the solution method. Before going further with the pseudo-code, we make an observation which simplifies our process.

We know that checking for convexity or concavity for F_u^+ or F_u^-, respectively, is equivalent to looking at their slopes. Hence, since u is the slope of F_u we will now work with u^+ and u^- to represent the upper and lower bounds. These are now vectors containing the slopes of F_u^+ and F_u^- respectively.

When we add a line segment to F_u^+ and F_u^- its slope is

$$F_f(i) + \alpha - (F_f(i-1) + \alpha) = F_f(i) - \alpha - (F_f(i-1) - \alpha) = f_i \tag{5.13}$$

for $i = 2, ..., n-1$. For the first line segment, when $i = 1$, the slope is $f_1 + \alpha$ for F_u^+ and $f_1 - \alpha$ for F_u^-. For the last line segment, when $i = n$, the slope is $f_n - \alpha$ for F_u^+ and $f_n + \alpha$ for F_u^-.

Now when we go through the algorithm, we create two new variables s^+ and s^- which represent the slope of the new line segments. Thus in node i, where $2 \leq i \leq n-1$, we have $s^+ = s^- = f_i$. Then we check convexity and concavity by simply comparing s^+ and s^- to the last element of u^+ and u^-, respectively.

When we must remove a knot point from u^+, we compute the new slope as the weighted mean of s^+ and the last element of u^+ over the new interval and replace this as the last element of u^+. The same computation holds for removal of knots points of the lower bound u^- as well. To check if the first line segment of the upper bound is above that of the lower bound we can simply compare the first elements of u^+ and u^-.

Complexity

We argue that the complexity of the taut string algorithm is $\mathcal{O}(n)$:

Each index is added to the index sets only one time. If we have k jumps, that means $n-k$ indices have been removed. If an index is removed, it is never added back. Removing indices takes a constant number of calculations, as we only have to calculate the new slope. Also, adding indices to the solution only takes a constant number of calculations, as we only have to calculate a new slope for one of the bounds. In total, for each index we only have to make a constant number of calculations, and therefore the complexity is $\mathcal{O}(n)$.

Possibility of a Taut String Algorithm for $A \neq I$

We looked into whether we could derive a similar taut string algorithm for non-diagonal A. It turned out we could not, which we will explain briefly. If we look at (4.2), we see why the taut string approach is not possible for when A is not diagonal. If we have a general matrix A such that the matrix $A^T A$ is a full matrix, the right hand side of (4.2) is a linear combination of all u_is, instead of just the cumulative sum of the u_is for $i \leq j$. In particular, in the first iteration we would have $p_1 = (A^T Au)_1 - (A^T f)_1$, where p_1 only depends on $(Du)_1$, but $(A^T Au)_1$ is dependent of u_i for all $i = 1, \dots, n$. Therefore, if we choose an upper bound for u_1 and u_2 as we do in the taut string algorithm, there is no way we can see if condition (3.35) holds or not. Therefore, we can not solve the problem using the same approach.

Comparison to Lasso-Path Algorithm

There are some differences between the taut string algorithm and the lasso-path algorithm worth mentioning. One difference is that when a jump is added to the solution in the taut string algorithm, it is never removed. That is, if we solve (5.1) for decreasing values of α, the solution will only obtain new jumps as α decreases. In the lasso-path for $A \neq I$ on the other hand, jumps will often, even in the easier examples, disappear. We will see examples of this in Chapter 7, in particular in figure 7.3. Therefore, when using the taut string algorithm, if we have obtained a jump, we know that it will always be part of the solution for all smaller values of α, while for the lasso-path, the main features of the solution might change.

Another difference is that with the taut string algorithm, we are only solving for one value of α at the time. The lasso-path on the other hand, solves for all values of α as long as we still have a unique solution. This means that if we know for which parameter values we want to solve the problem, the taut string can be expected to be faster. If we are trying to find the best solution, whatever that may be, and need to run simulations for many parameter values, the lasso-path algorithm can be a better choice.

Chapter 6

Numerical Experiments with Taut String Algorithm

In this chapter we will present some numerical experiments with the taut string algorithm. In Chapter 6.1 we present the test functions we will be using. We test the influence the parameter α and the level of added noise have on the reconstructed function, in Chapter 6.2 and 6.3 respectively. In Chapter 6.4 we test the algorithm for a piecewise smooth signal.

6.1 Test Functions

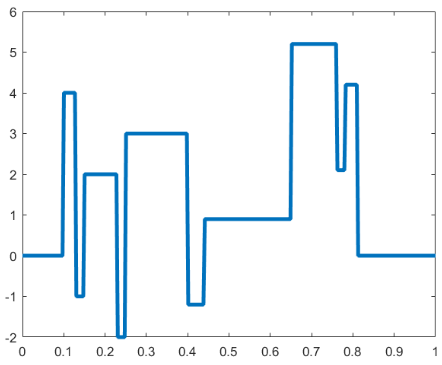
The test function we will be using for most of the experiments is a block signal, which shown in figure 6.1a. It is inspired by a similar block signal in [2], and is defined as follows:

$$\begin{aligned}
&u_0(t) = \sum h_j K(t - t_j) \quad \text{where} \\
&K(t) = (1 - \text{sgn}(t))/2, \\
&(t_j) = (0.1, 0.13, 0.15, 0.23, 0.25, 0.40, 0.44, 0.65, 0.76, 0.78, 0.81), \\
&(h_j) = (4, -5, 3, -4, 5, -4.2, 2.1, 4.3, -3.1, 2.1, -4.2).
\end{aligned} \tag{6.1}$$

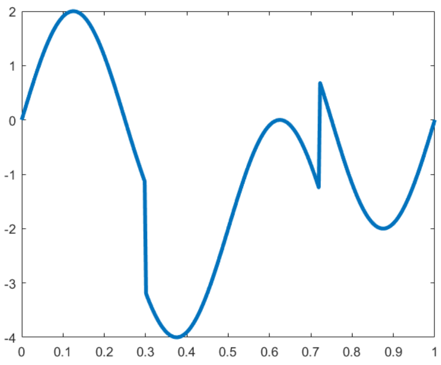
We will also test the algorithm on the heavisine signal shown in figure 6.1b also from [2], which is defined as

$$u_0(t) = 2\sin(4\pi t) - \text{sgn}(t - 0.3) - \text{sgn}(0.72 - t). \tag{6.2}$$

For all the experiments we set $n = 300$.



(a)



(b)

Figure 6.1: Test signals. To the left is the block signal. Heavisine signal to the right.

6.2 Influence of Parameter in Reconstruction

We start by comparing the results for different values of α using a noise free signal. We use the test data $f = u_0$ where u_0 is the block signal. The results for four different α-values are shown in figure 6.2.

The results are shown from larger to smaller α-values. We have included these examples as they illustrate the algorithm well.

We see that for a large enough value of α, say $\alpha = 100$ in this case, we get that u is a constant function. As we decrease α we see that the solution gains more and more jumps. For small values of α the algorithm reconstructs all of the jumps of u_0.

This result is what we expect, both from the construction of the functional, but also if we think about the lasso-path algorithm. There we know that we gain more and more jumps as we decrease α. This is exactly what happens here.

We observe that the smallest details appear last. The largest jumps, or the jumps belonging to the larger blocks, appear earlier than the smaller jumps.

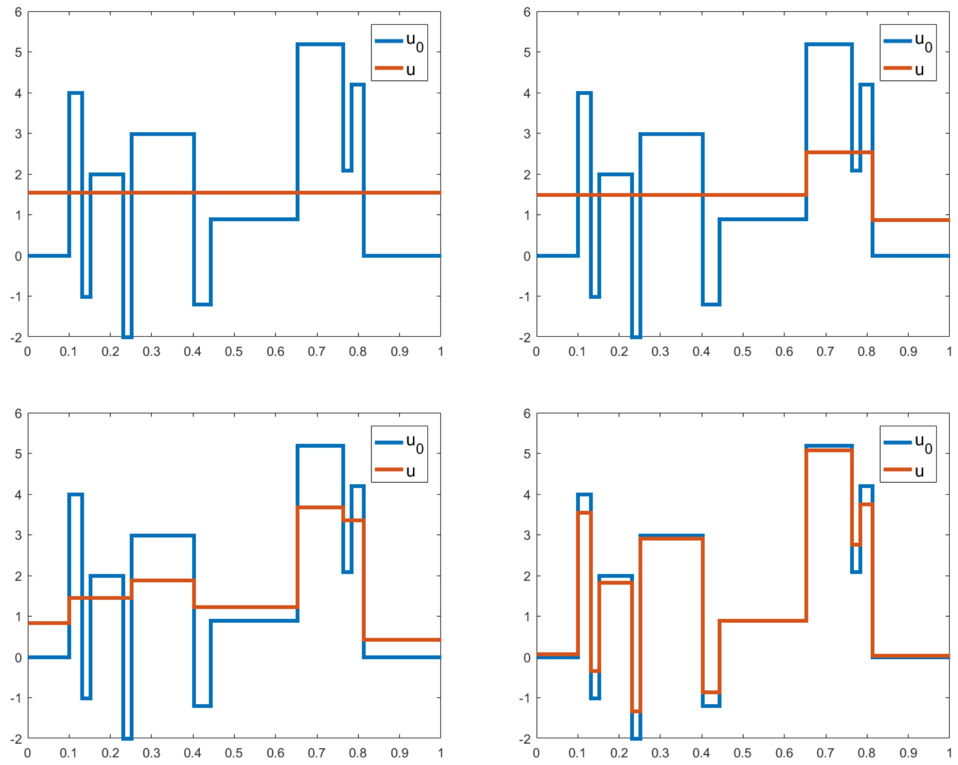


Figure 6.2: Some selected results of the taut string algorithm applied to the noise free block signal. The red lines show the reconstructed function u and the blue lines show the function u_0. From left to right we have, top level: $\alpha = 100$ and $\alpha = 50$, bottom level: $\alpha = 25$ and $\alpha = 2$. The number of jumps in u increases as α decreases, and all the jumps are reconstructed for small enough values of α.

6.3 Influence of Noise in Reconstruction

We want to examine how different levels of noise affect the results. We use the test data $f = u_0 + \varepsilon$, where u_0 is still the block function, and where the noise ε are normally distributed random numbers.

We choose the noise ε by using the built-in MATLAB function `randn` to draw the noise from the standard normal distribution. In order to obtain different levels of noise for the different experiments, we scale the noise drawn from the standard normal distribution with a scalar c. Hence, for each experiment we have that $\varepsilon \sim \mathcal{N}(1, c^2)$.

We start with small noise, and use $c = 0.1$. A selected result, for when $\alpha = 2$, is shown in figure 6.3. The results are fairly similar to the results of the algorithm applied to the noise free signal. We are able to reconstruct all the jumps of u_0, and the appear in the same order, that is, the more ”significant” jumps appear first. As α decreases the

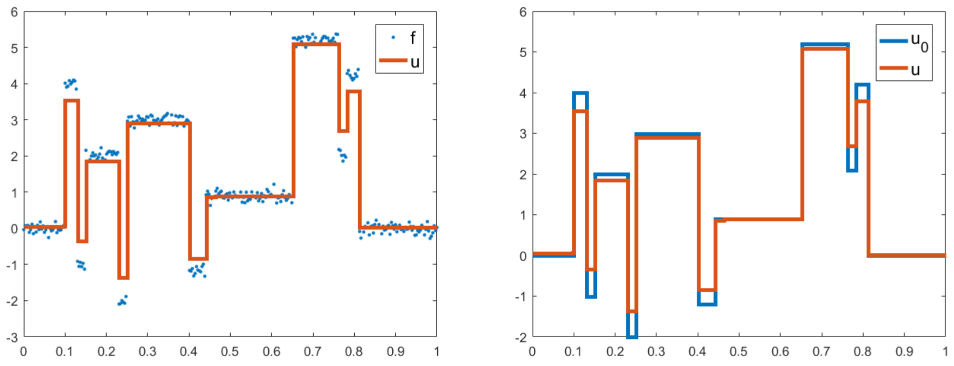


Figure 6.3: Results of the taut string algorithm applied to block signal with small noise ($c = 0.1$). The dots show the data f, the blue line show the original signal u_0, and the red line show the reconstructed function u. The results are shown for regularization parameter $\alpha = 2$. All the correct jumps are reconstructed, but we have also obtained some small, wrong jumps.

reconstruction gets closer to u_0. Although we also have some small, wrongly detected jumps, they are so small we can separate between them and the correct ones.

We continue with a larger noise level, achieved by setting $c = 0.5$. See figure 6.4 for the results. We show the result for the same regularization parameter as in the last experiment, so $\alpha = 2$. Though the variance in the noise is much larger than in the previous example, we still achieve good results. We are able to reconstruct all the jumps. However, now we have in addition obtained a lot of wrong jumps. Nevertheless, all of the wrongly detected jumps are small compared to the correct jumps, so we can still distinguish between them. Also, the wring jumps are only "smoothing out" the correct jumps. That is, we have not obtained any new local maxima/minima in the reconstructed function.

We set $c = 1$ to obtain a large level of noise. We see the results in figure 6.5. Here we have included the results for two different α-values, in particular $\alpha = 4$ and $\alpha = 2$. What is interesting here, is that for $\alpha = 4$, we still have the same type of wrongly detected jumps as we did in the result for $c = 0.5$, that is, the jumps are small and does not contribute to any new maxima/minima. But we are not able to construct all the jumps of the original signal, as we see, we are missing the rightmost positive jump. For $\alpha = 2$, we have obtained the last correct jump, but at the same time we also obtained a new type of wrongly detected jumps, which have given new local maxima/minima to the reconstructed function. They are still smaller than most of the correct jumps of u, except from the last correct jump we obtained. This means that we can still find most of the main jumps of the signal, but not all, and the number of maxima/minima has changed.

If the goal is to find the maximum or minimum value of the function, or how many local maxima/minima there are, then we can still find that when small to medium noise is present. The extra jumps we obtain in those cases doesn't affect those features.

For large noise, however, this is no longer the case. The extra jumps obtained in the reconstruction changes the number of local maxima/minima. Still, we have not yet reached a level of noise such that we can't distinguish between those false maxima/minima and

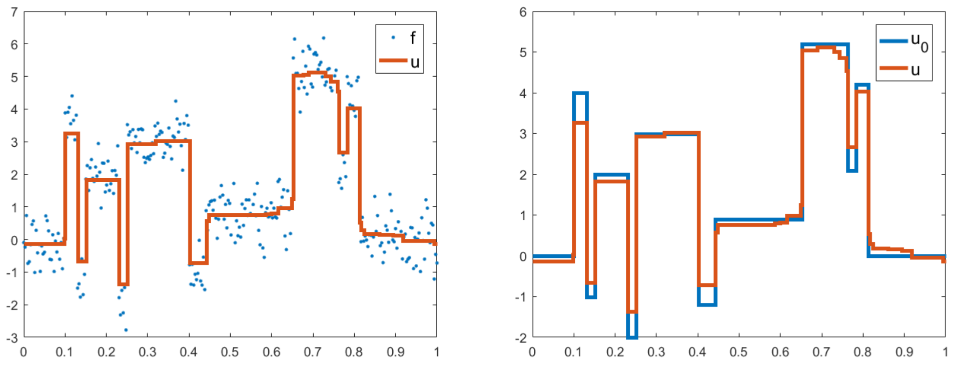


Figure 6.4: Results of the taut string algorithm applied to block signal with medium noise $(c = 0.5)$. The dots show the data f, the blue line shows the original signal u_0, and the red line shows the reconstructed function u. The results are shown for $\alpha = 2$. We reconstruct all the jumps in the original signal, while obtaining many small, wrong jumps.

the correct ones. Except perhaps for the rightmost positive, correct jump in u, all of the correct jumps in u are much larger than the wrong ones, and therefore we can still separate between them.

We run another experiment for $c = 2.2$, that is, a variance of almost 5. See the results in figure 6.6. We have included the result for $\alpha = 4$, which is when we obtained the last jump of the original signal, the rightmost positive jump. Through all these experiments we see that this is the hardest jump to find. The jumps still appear roughly in the same order, that is, the more "significant" jumps appear first. For the large level of noise in this last experiment, the wrongly detected jumps are so large that they are indistinguishable from the correct ones.

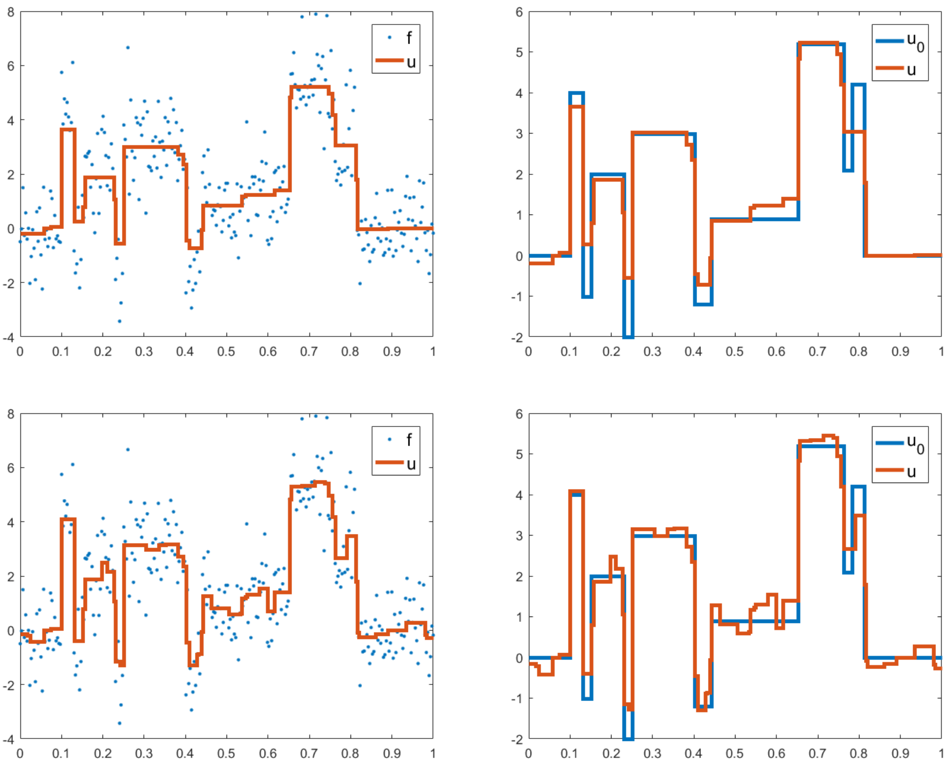


Figure 6.5: Results of the taut string algorithm applied to block signal with large noise $(c = 1)$. The dots show the data f, the blue line shows the original signal u_0, and the red line shows the reconstructed function u. The upper plots are the results for $\alpha = 4$, and the lower ones for $\alpha = 2$. We reconstruct all of the jumps in the original signal, but we also obtain many wrongly detected jumps.

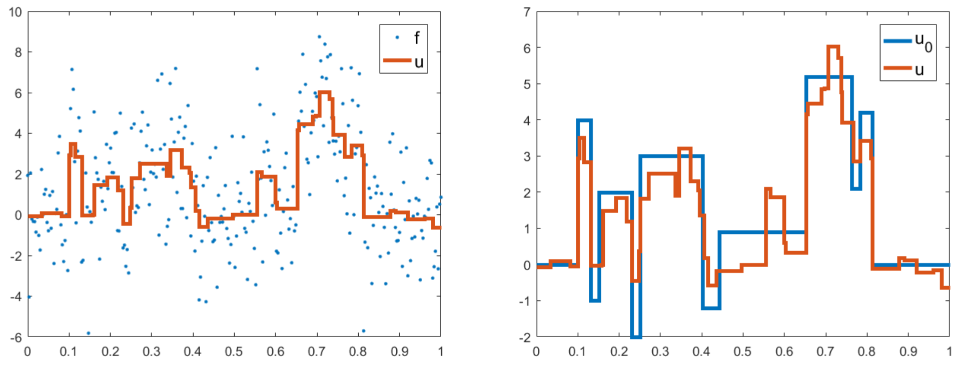


Figure 6.6: Results of the taut string algorithm applied to block signal with large noise $c = 2.2$). The dots show the data f, the blue line shows the original signal u_0, and the red line shows the reconstructed function u. The regularization parameter is $\alpha = 4$. The noise is so large that we cannot separate between the wrongly detected jumps and the correct ones.

6.4 Piecewise Smooth Test Data

We have seen that the taut string algorithm is good at reconstructing piecewise constant signals. Now we want to test the algorithm on a piecewise smooth signal. We use the heavisine signal we introduced in Chapter 6.1. We do the experiment without noise. The results are shown in figure 6.7.

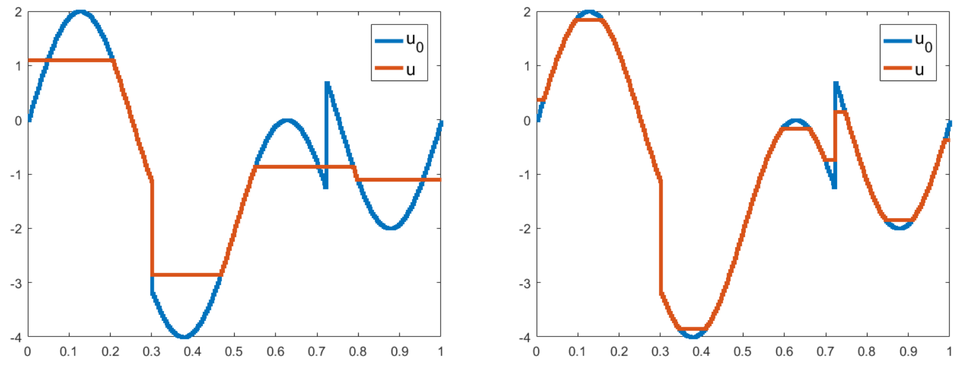


Figure 6.7: Results of the taut string algorithm applied to the heavisine signal. The blue line is the original signal, the red line is the reconstructed signal. To the left $\alpha = 20$, to the right $\alpha = 1$.

We show the result for both large and small values of α. For both of them, we see that the reconstruction u of the signal is either constant, or equal to the original function u_0. Also, u is constant near every extrema of u_0. We see that if we choose α small enough, we are able to find the correct number of extrema of u_0.

Chapter 7

Numerical Experiments with Lasso-Path Algorithm for TV Regularization

In this chapter we introduce some numerical experiments with the lasso-path algorithm for total variation regularization. In Chapter 7.1 and 7.2 we look at the reconstruction when the signal is transformed by moving averages. In Chapter 7.3 we test the algorithm on signals transformed by random matrices.

7.1 Reconstruction from Moving Averages

We start with $A \in \mathbb{R}^{300\times 300}$ as the moving average over $2k+1$ elements. In other words, define A by

$$(Au)_i = \frac{1}{2k+1}\sum_{j=i-k}^{i+k} u_j, \tag{7.1}$$

which results in a banded matrix with $2k+1$ diagonals:

$$A = \frac{1}{2k+1}\begin{bmatrix} \ddots & & \\ & 1 & \\ & & \ddots \end{bmatrix}. \tag{7.2}$$

In the following experiments we choose $k = 3$, that is, we take the average over 7 adjacent nodes. We use the noise free test data $f = Au_0$ where u_0 is the block signal defined in (6.1). The original signal u_0 and the transformed signal f are shown in figure 7.1, for comparison. We see that A averages out the jumps, while we still see all the features of the signal.

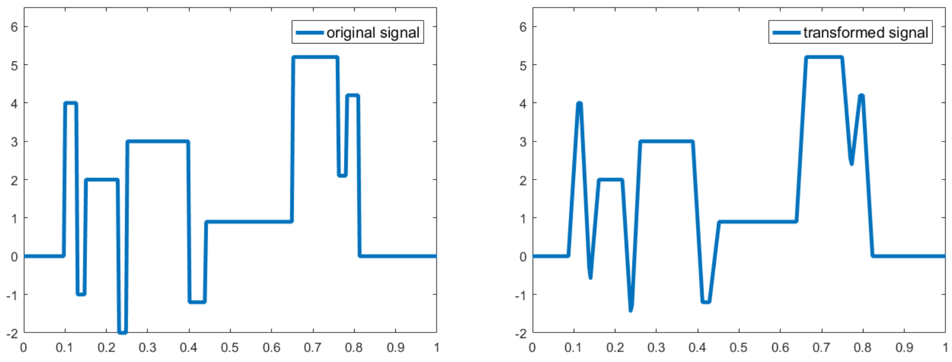


Figure 7.1: The block signal to the left. The result of taking the moving average of the block signal over 7 adjacent nodes, to the right.

Some selected results are shown in figure 7.2. Here we have chosen the solutions for $\alpha = 68.3296, \alpha = 37.3173, \alpha = 25.9434$ and $\alpha = 0.9819$. We see that when α decreases towards zero, the reconstruction u comes closer to the original signal u_0, as expected.

It is perhaps not easy to spot in the plots, but the jumps are not always at the correct indices. Sometimes, the jumps are even moving, as we have illustrated in figure 7.3. As the jump moves to the left, we obviously have to delete the old jump. Therefore, we see that even in this relatively ”easy” example, it is still important to take into consideration that we might have disappearing jumps.

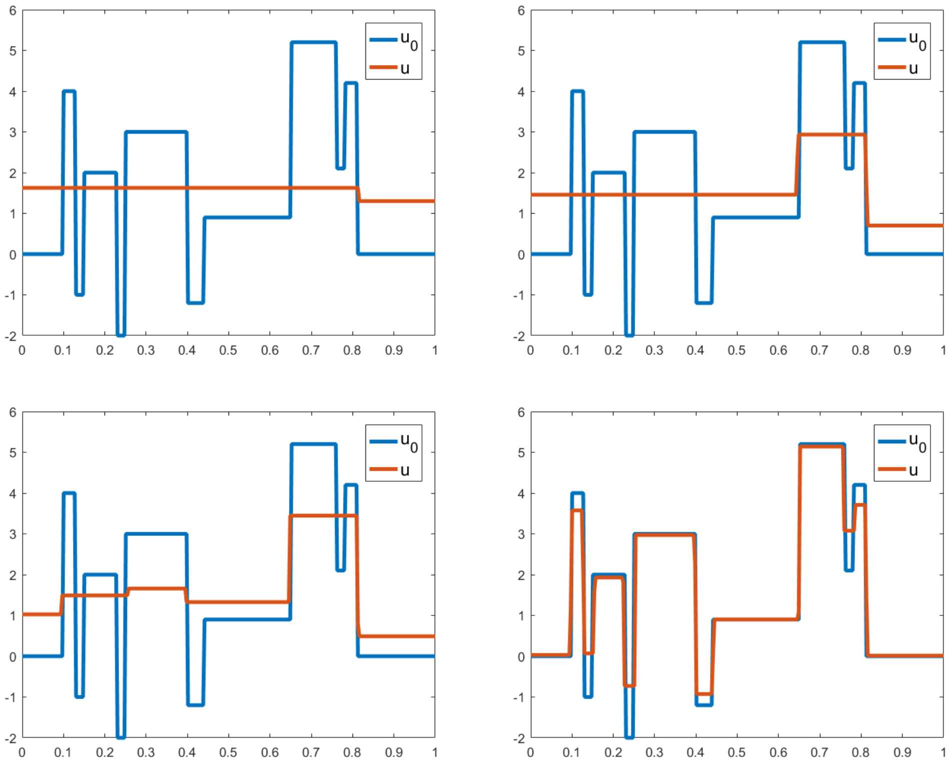


Figure 7.2: Results of the lasso-path algorithm applied to the transformed block signal. The red line represents the reconstructed signal u, and the blue line shows the original function u_0.

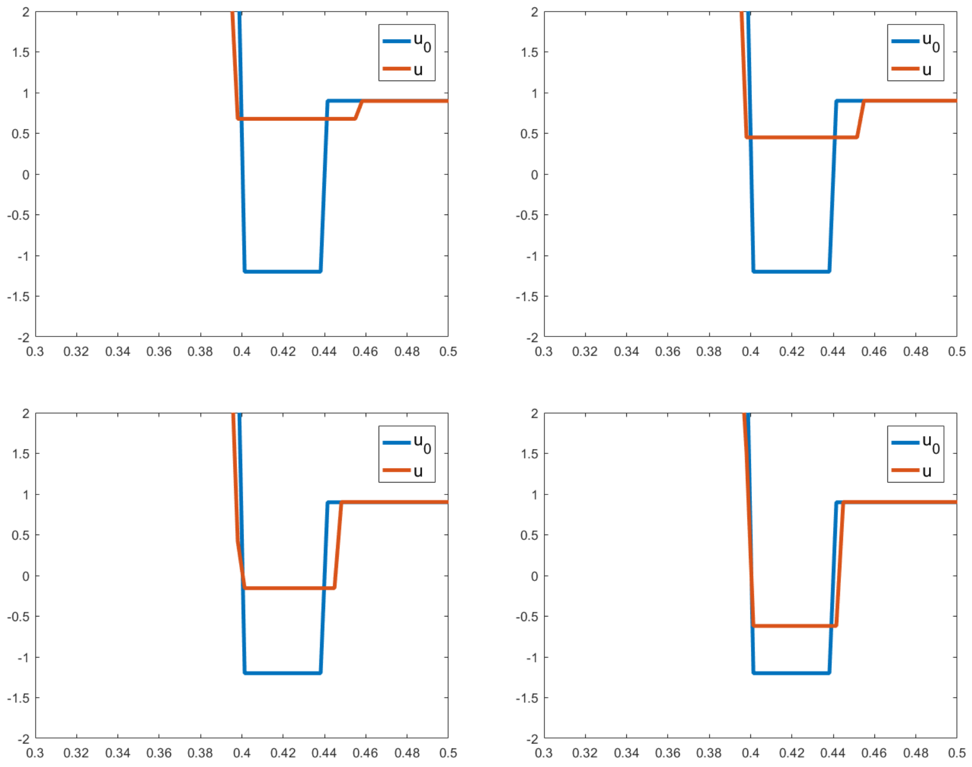


Figure 7.3: Illustration of how a jump in the reconstructed function u (red line) moves over several iterations when we apply the lasso-path algorithm to the noise free block signal transformed by the moving average over 7 adjacent nodes. The blue line represents the original function u_0. In each iteration, and hence for smaller values of α, the jump comes closer to the correct jump.

7.2 Reconstruction from Noisy Moving Averages

We continue with the same transformation as before, where $A \in \mathbb{R}^{300\times 300}$ is given as in (7.2), for $k = 3$. Now the test data is given as $f = Au_0 + \varepsilon$, where u_0 is the block signal, and $\varepsilon \sim \mathcal{N}(1, c^2)$, for different values of c.

We start with $c = 0.2$. A few results are displayed in figure 7.4, where we have plotted both the transformed solution and the solution. From the lower level of the figure we see that we are able to recreate all significant jumps. It took 55 iterations to obtain all the significant jumps, and then the solution contained 20 jumps in total.

We have also obtained a wrong jump. This does not change the number of maxima/minima from that of the true solution, so the method works well in that regard.

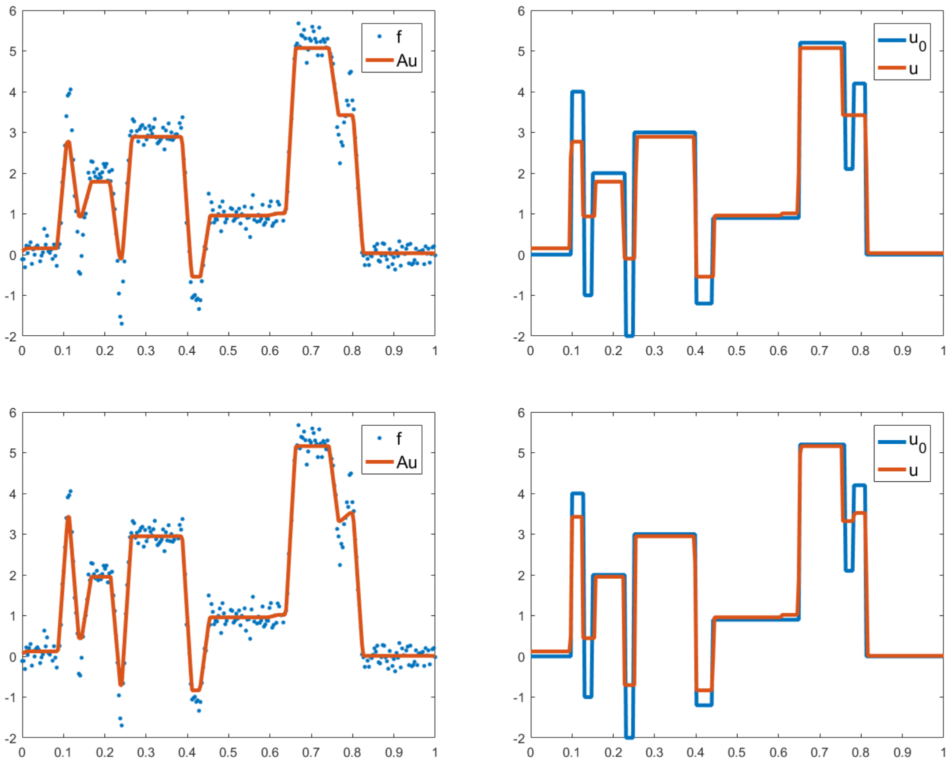


Figure 7.4: Results of lasso-path algorithm applied to the transformed block signal with small noise present ($c = 0.2$). To the left is the data f (dots), and the red line is the transformation of the reconstructed signal, Au. To the right is the original signal u_0 (blue), and the reconstructed signal u us the red line. The top level is the first time the solution contains 20 jumps. The bottom level is the third, and last, time the solution contains 20 jumps.

We decided to show the results for when we reached a given number of jumps in the solution. In this case we chose 20 jumps. Since the algorithm also deletes jumps, it is not given that we only reach 20 jumps one time, which we display here. In this case, the solution had 20 jumps three times, and we plotted the first and last occurrence. We see how different the results are: in the first case we had not yet obtained the last jump of the original function, but the third time we had.

However, we still see that we have a lot more jumps than the original function, which has 11 jumps. Most of them, all except one in this case, are adjacent jumps making up the correct jumps. In other words, we have only found one jump in the wrong position.

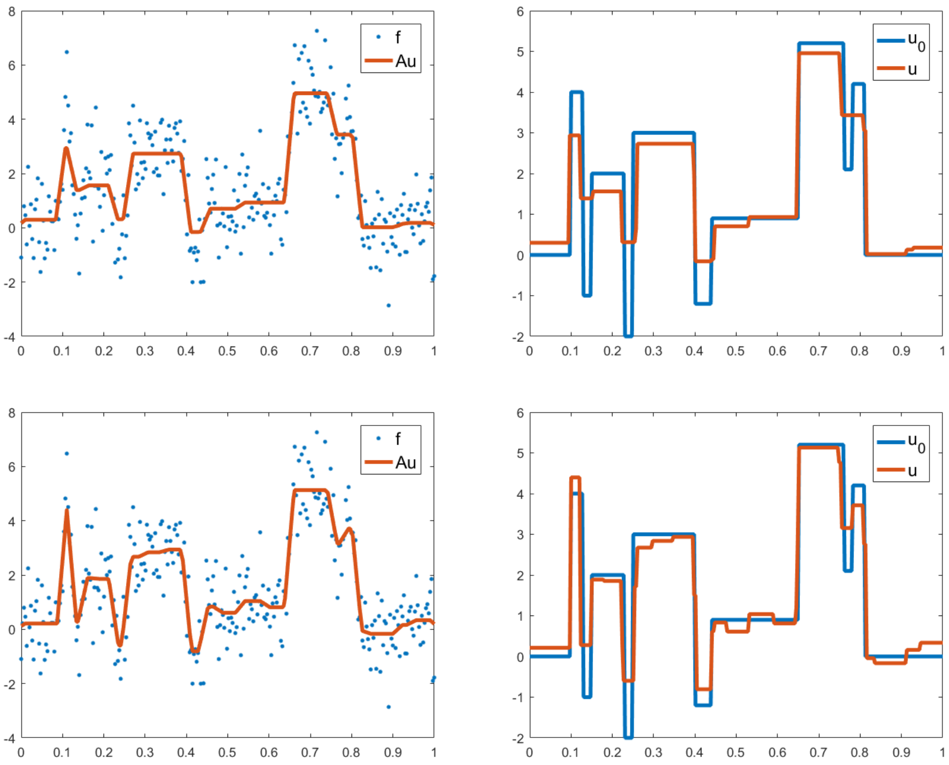


Figure 7.5: Results of lasso-path algorithm applied to the transformed block signal with large noise present ($c = 1$). To the left is the data f (dots), and the red line is the transformation of the reconstructed signal, Au. To the right is the original signal u_0 (blue), and the reconstructed signal u us the red line. The top level shows the solution when it has 20 jumps. The bottom level shows that we can recreate all the significant jumps.

For $c = 1$, we show some results in figure 7.5. The upper level shows the result when the number of jumps in the solution is 20, which happens several times, but the results are almost the same each time. In the lower level we have found all the significant jumps. When we find all the significant jumps, we have 24 jumps in the solution. Many are still adjacent "correct" jumps, but we see that we also have many more wrong jumps. Some of the wrong jumps alter the number of maxima/minima, so in that regard the method is no longer so good. The wrong jumps are so large that we can't differentiate between the rightmost positive, correct jump and the wrong ones. It took 82 iterations to find all the significant jumps this time.

7.3 Reconstruction from Randomly Transformed Data

For all the experiments in this chapter, we let A be a $m \times n$ matrix of normally distributed random numbers, where $m = 50$ and $n = 300$. Since this is a more difficult problem, we start by using a simpler block signal, defined by

$$\begin{aligned} &u_0(t) = \sum h_j K(t - t_j) \quad \text{where} \\ &K(t) = (1 - \text{sgn}(t))/2, \\ &(t_j) = (0.1, 0.25, 0.46, 0.61, 0.70, 0.85), \\ &(h_j) = (4, -7, 3, -4, 5, -5.2, -3.1, 2.1, -4.2). \end{aligned} \tag{7.3}$$

Now the test data is $f = Au_0$ where u_0 is defined in (7.3) and A is constructed using the built in `randn`-function in MATLAB to draw numbers from the standard normal distribution. See figure 7.6 for some selected results.

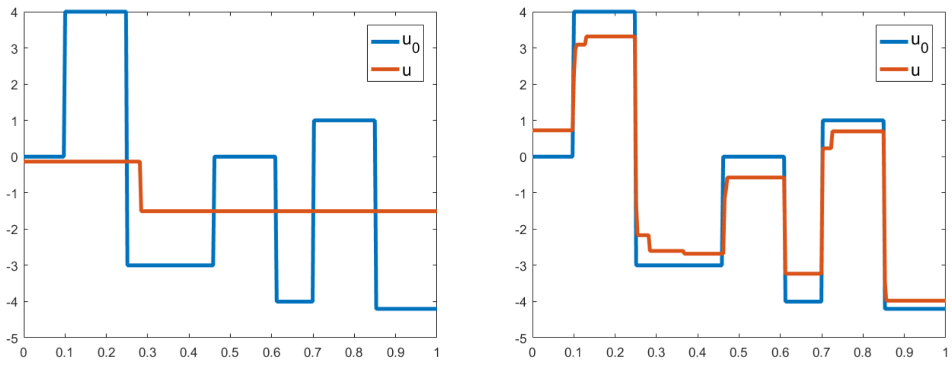


Figure 7.6: Result of lasso-path algorithm to a block signal transformed by a random matrix of $m = 50$ measurements, when no noise is present. The red line shows the reconstructed function u, and the blue line shows the original signal u_0. To the left we have the solution for $\alpha = 6.36 \cdot 10^3$, and to the right the solution when $\alpha = 368$. The method reconstruct the signal well.

We see that we are able to find all the jumps. We can reconstruct the original signal u_0 very well using this noise free test data. We see that some jumps are displaced, but as we decrease α, we come closer to the original signal. We obtain a few wrong jumps, but they are much smaller than the correct ones, so we are able to separate between the two types of jumps.

We run the algorithm again with the same block signal transformed by a random matrix A plus some small noise. In this case, relatively small noise can be using $c = 3$, for which we get the signal to noise ratio $\dfrac{\|Au_0\|_2}{\|\varepsilon\|_2} \approx 17$. We see the result of this in figure 7.7.

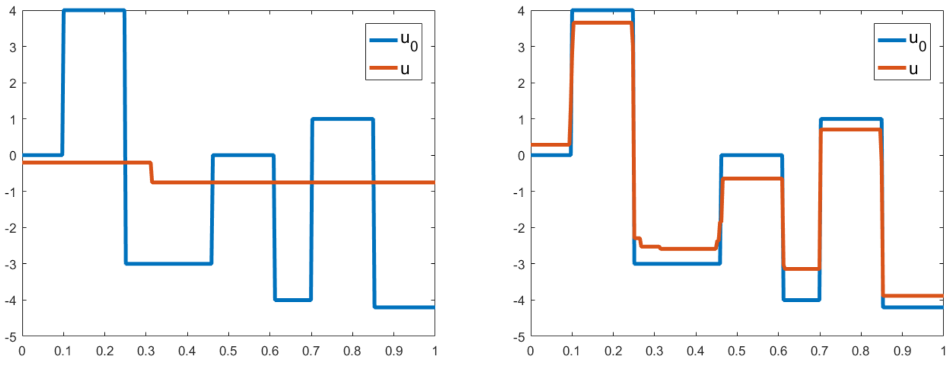


Figure 7.7: Result of lasso-path algorithm applied to a block signal transformed by a random matrix of $m = 50$ measurements, when small noise is present (signal to noise ratio is approximately 17). The red line shows the reconstructed function u, and the blue line shows the original signal u_0. To the left we have the solution for $\alpha = 7.699 \cdot 10^3$, and to the right the solution when $\alpha = 365.55$. The method reconstructs the signal well.

We see that the results are very similar to when no noise is present. We see that we are able to find all the jumps. The first jump is displaced, but as α decreases, we come closer to the original signal. The wrong jumps we obtain are small compared to the correct ones. Because of this good result, we try an even more difficult data set, by adding large noise to the transformed signal. We use $c = 20$, for which we get a signal to noise ratio of approximately 2.5. The results are shown in figure 7.8.

It seems like the method still works very well. We are able to reconstruct all of the jumps, while we get some small, wrongly detected jumps. A difference from the experiment with smaller noise, is that the middle block is detected but shifted to the left. However, it is still a good reconstruction in regard to finding the number of blocks in the function, and hence the number of maxima/minima. The results show that the significant jumps appear first.

We now try the algorithm again, now for the more challenging situation of the block signal in (6.1). We run it for both a noise free data set, and a data set with large noise, say $c = 10$, which yields a signal to noise ratio of approximately 4.4. See figure 7.9 for the results.

Now the method has more trouble reconstructing the signal. In the noise free case (left), we are still able to find all the significant jumps, but we have also obtained several wrong jumps. Some of these wrongly detected jumps has also changed the number of maxima/minima in the function, and they are so large that it is difficult to distinguish between them and the correct jumps.

In the case where we added a relatively large amount of noise to the signal, we see from figure 7.9 (right) that we are no longer able to find all jumps of the original signal. We have also obtained several wrong, large jumps, and also new maxima/minima. For this

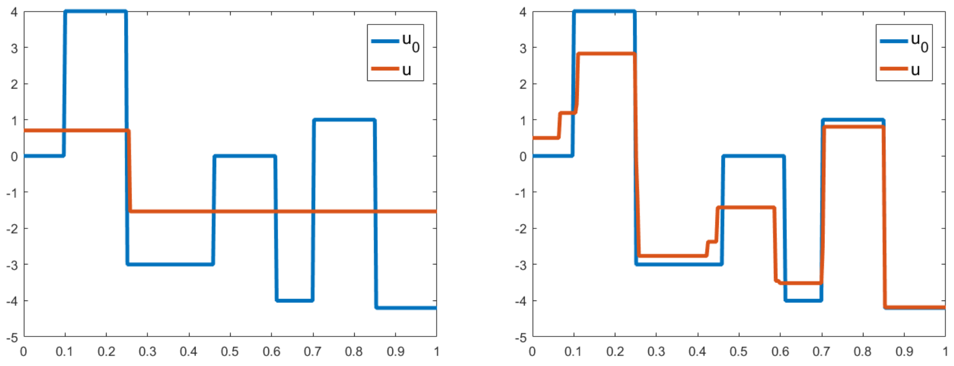


Figure 7.8: Result of lasso-path algorithm applied to a block signal transformed by a random matrix of $m = 50$ measurements, when large noise is present (signal to noise ratio is approximately 2.5). The red line shows the reconstructed function u, and the blue line shows the original signal u_0. To the left we have the solution for $\alpha = 6.8485 \cdot 10^3$, and to the right the solution when $\alpha = 1.1277 \cdot 10^3$. The method reconstructs the signal reasonably well.

much noise present the method does no longer work well.

Lastly, we run an experiment for a larger number of measurements in the random matrix A. We want to see if we can find how many measurements one needs to obtain a reasonable construction of the block signal in (6.1). We set $m = 80$ and run the experiments for noiseless data. The results are shown in figure 7.10 (right).

As we see, with $m = 80$, we obtain a much better reconstruction than for $m = 50$ in figure 7.9 (left). All the jumps are clearly reconstructed, and the wrong jumps are small in comparison. Since the results were so good, we try to decrease the number of measurements to $m = 60$. The results are shown in figure 7.10 (left).

We see that the results are still much better than for $m = 50$. The only problem we have is the recurring difficulty of reconstructing the rightmost positive jump. It remains small in this experiment, even smaller than some of the wrongly detected jumps. All the other correctly reconstructed jumps, however, are large compared to the wrong ones. For both cases $m = 60$ and $m = 80$, the wrong jumps have not contributed to extra maxima/minima, as they did for $m = 50$.

To sum up, we have found that with 50 measurements, we were able to reconstruct a signal of 6 jumps very well, even with large noise present. When we increased the number of jumps in the signal to 11, 50 measurements were not enough to reconstruct even the noise free signal. When we increased the number of measurements to 60, we were able to reconstruct the signal reasonably well. For 80 measurements, the reconstruction was even better.

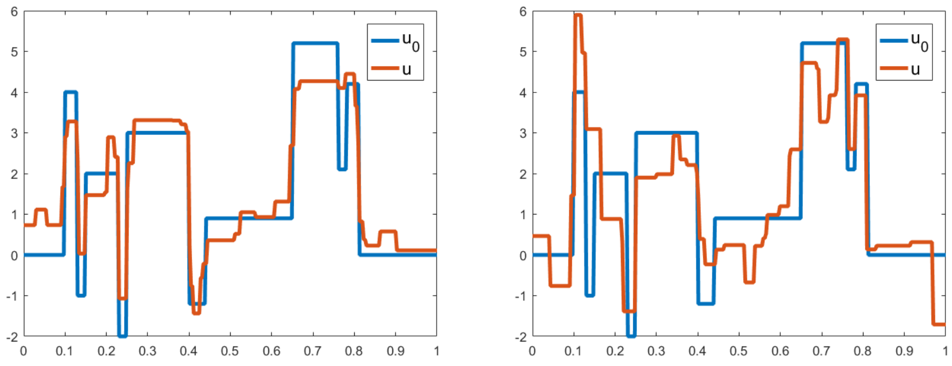


Figure 7.9: Result of lasso-path algorithm applied to a block signal transformed by a random matrix with $m = 50$ measurements. The red line shows the reconstructed function u, and the blue line shows the original signal u_0. To the left is the result for a noise free test signal, and the solution shown is for $\alpha = 22.1559$. To the right is the result of noisy test signal (signal to noise ratio is approximately 4.4), and the solution is for $\alpha = 47.3767$. For this many jumps in the original signal, we do not obtain a good reconstruction when $m = 50$.

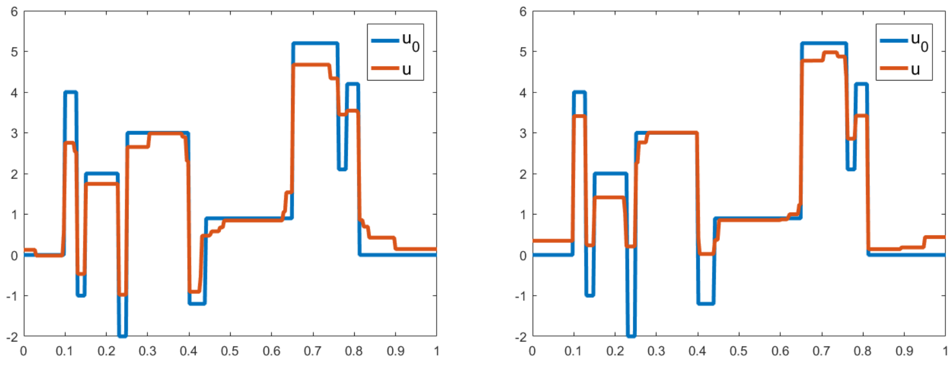


Figure 7.10: Result of lasso-path algorithm applied to a block signal transformed by a random matrix with $m = 60$ measurements to the left, and $m = 80$ measurements to the right. The blue line shows the original signal u_0, and the red line represents the reconstructed solution u. In both cases the method reconstructs the signal reasonably well.

Chapter 8 Conclusion

We have successfully derived and implemented two methods for denoising one dimensional signals. Both of the methods yield the unique, exact solution to the problem for each regularization parameter.

First we derived the lasso-path algorithm for total variation regularization. We implemented the algorithm and performed a small number of numerical experiments. In general the results were very good, as we were able to recreate all of the jumps in the original signal in most of the cases where not too much noise was present. In the case of the random matrix operator, it was more difficult to produce good results. We saw that how well the method reconstructed the signal was dependent of the number of measurements in the random matrix, and also on the number of jumps in the original signal.

We also derived the taut string algorithm, which is used for solving the same total variation regularization problem under the assumption that the original signal is not transformed. We implemented the algorithm and performed a few numerical experiments. For noiseless test data, we saw that the algorithm worked as expected, as the reconstructed function came closer to the original signal as the regularization parameter was decreased. When too much noise was added, we had trouble reconstructing the true solution as we obtained too many wrong jumps that was indistinguishable from the correct jumps.

The largest difference between the two algorithms is that the taut string algorithm only solves for one parameter-value at a time. In the lasso-path algorithm we obtained a solution for a range of parameters in one iteration, which could lead one to think it would over all be the best method. However, the taut string algorithm is very fast, as it is only of complexity $\mathcal{O}(n)$, and can be applicable in some cases where we know in advance a specific parameter value for which we want to solve the problem. If we do not know for which parameter values to solve, and want to run several simulations, the lasso-path algorithm might be faster. On the other hand, if the solution has many jumps, the lasso-path might be slow, because in each iteration we have to solve a linear system where the size of the matrix S depends of the number of jumps. Lastly, the taut string algorithm only works when A is the identity matrix, while the lasso-path algorithm works for any linear transformation A, so it has a wider range of use.

For the lasso-path algorithm we obtain over all more jumps in the solution than with the taut string algorithm. We have seen that many of these wrong jumps are adjacent to a correct jump, and some are wrong jumps at wrong positions. This means that we obtain a lot more jumps in the reconstruction than there are in the original signal. This makes it very difficult to estimate how many iterations it will take to find a good reconstruction, because we cannot use the number of jumps as a reference point.

Obtaining good results heavily relies on choosing the right parameter values. This is a difficult task, but a natural continuation in the study of the problem is this thesis. What a ”good result” is, varies depending on the problem we are studying and what we want to achieve. The best parameter for noise reduction is not necessarily the best parameter for finding the correct number of maxima/minima of the function. Also, if we keep the lasso-path algorithm in mind, it might be more interesting to consider the number of jumps in the solution than the parameter value. One could perhaps try to find a way to distinguish between the types of wrongly detected jumps. In addition, since we have seen that the reconstruction for random matrices depends on the number of jumps in the signal, the noise, and the number of measurements, this may also be a topic for further investigation.

References

[1] E. J. Candès. Compressive sampling. In *Proceedings of the International Congress of Mathematicians*, volume III, page 1433–1452, Madrid, Spain, 2006. European Mathematical Society.

[2] D. L. Donoho and I. M. Johnstone. Ideal spatial adaptation by wavelet shrinkage. *biometrika*, pages 425–455, 1994.

[3] M. Grasmair. The equivalence of the taut string algorithm and BV-regularization. *Journal of Mathematical Imaging and Vision*, 27(1):59–66, 2007.

[4] O. Güler. *Foundations of optimization*, volume 258 of *Graduate Texts in Mathematics*. Springer Science & Business Media, 2010.

[5] J.-B. Hiriart-Urruty and C. Lemaréchal. *Convex analysis and minimization algorithms I: fundamentals*. Springer science & business media, 1993.

[6] R. Kimmel, N. A. Sochen, and J. Weickert, editors. *Scale Space and PDE Methods in Computer Vision*, Lecture Notes in Computer Science. Springer.

[7] J. Mairal and B. Yu. Complexity analysis of the lasso regularization path. *arXiv preprint arXiv:1205.0079*, 2012.

[8] E. Mammen and S. van de Geer. Locally adaptive regression splines. *The Annals of Statistics*, 25(1):387–413, 1997.

[9] L.I. Rudin, S. Osher, and E. Fatemi. Nonlinear total variation based noise removal algorithms. *Physica D: Nonlinear Phenomena*, 60(1–4):259 – 268, 1992.

[10] G. Steidl, S. Didas, and J. Neumann. Relations between higher order TV regularization and support vector regression. In *[6]*, pages 515–527, 2005.