UNIVERSITY OF TWENTE

MASTER THESIS

An application of a proximal method with a fractional Laplacian regularizer for inverse problems

Author:
Martijn HEUTINCK

Supervisor:
Dr. rer. nat. José A. IGLESIAS MARTÍNEZ

A thesis submitted in fulfillment of the requirements for the degree of Master of Mathematics

in the group

Mathematics of Imaging and AI
Department of Systems, Analysis and Computational Sciences

December 20, 2022

Contents

1 Introduction

In most scientific fields calculating the inverse of some matrix or operator is done quite often. This is usually done to find a certain value from an observation or measurement that was made. This situation can be expressed concisely with the following equation:

$$Au = f \tag{1}$$

In this simple equation A can be a matrix or an operator describing some physical phenomena and f is usually a vector, a matrix or a function that has been measured or observed. The u is the desired quantity or function that needs to be retrieved. In a ideal world the inverse could be taken and the answer to this question would be:

$$u = A^{-1}f \tag{2}$$

Unfortunately, calculating an inverse is not always trivial if at all possible. A multitude of reasons can make it difficult to get to an inverse. An example of these difficulties is that the inverse does not always exist. An operator or a matrix is not always invertible. Another example is that the size of the calculations that need to be done to get to the inverse can be too big. It would simply take too much space for a computer. Most importantly, the operator or matrix can be ill-conditioned. Meaning that a small error would lead to a big difference in the eventual answer. These obstacles encountered in these calculations gave rise to the field of inverse problems and there are more obstacles that are not mentioned.

These inverses are of significant interest and a lot of research is being done in this field. Analysis has been done and numerical methods have been developed to support practical issues where these inverse problems arise. One of these places where these problems arise is in image reconstructions [14, 16, 23]. Image reconstruction is done in a lot of places, where the most well known is in the medical field. X-ray computed tomography [14] is a very well known situation where image reconstruction takes place. There are many more examples, both in the medical field and in other fields like geophysics [31], which explains why there is so much interest in this topic. This research will contribute to the development of image reconstructions with a focus on edge preserving regularization. A new method based on a fractional Laplacian operator will be investigated and compared to already existing edge preserving methods. Edge preserving methods are of great interest because of reconstructions consisting of separate objects and the need to exactly know where the edges are. A comparison will be made between the already existing method based on total variation [30] in regards to their reconstruction accuracy and speed in a linear inverse problem related to tomography. The accuracy will be measured using the SSIM index, which is a quality measure from image processing that correlates with subjective assessment better than mean squared error. Furthermore, the fractional Laplacian operator will also be tested on a non-linear operator. This time it will be compared to an algorithm called FISTA with a Haar wavelet basis to

simulate the edge preserving nature. It will only be compared on reconstruction accuracy in this case. This research will show that a fractional Laplacian as regularizer creates very comparable reconstructions to already existing edge preserving reconstruction algorithms, while at the same time being faster and easily extendable to non-linear operators.

2 Theory

In this theory section the basic concepts of inverse problems will be introduced. This will start of with inverse problems for measurements perturbed by noise, to continue with regularization methods and finish with how optimization algorithms work that solve these kind of inverse problems.

2.1 Inverse problems and ill-posedness

An inverse is not always trivial. To understand why, an introduction into inverse problems is going to be given now. The ideal situation, which is a measurement f that has been produced from the data we want to recover, can be described with the equation:

$$Au = f \tag{3}$$

This was already seen in the introduction, but unfortunately, this equation rarely describes the real world situations accurately. Almost all of the time, observations and measurements are perturbed by noise. This noise can have many causes and needs to be accounted for. Therefore, a better description than equation 3 is the following:

$$Au = f + n = f_n \tag{4}$$

Where f_n is the observations or measurement perturbed by the noise. This noise causes a number of issues. To understand the issues, the definition of well-posed and ill-posed problems is necessary. Using [25] where the following definition can be found:

Definition 1.

A problem is called well-posed in the sense of Hadamard if:

1. There exists at least one solution. (Existence)
2. There is at most one solution. (Uniqueness)
3. The solution depends continuously on data. (Stability)

A problem is called ill-posed if one of these conditions is not satisfied. In [25] this definition is only used in the case that A is a matrix and u, n, f_n are vectors. The same definition also holds, when A is an operator and u, n, f_n are functions, which can be seen in [12].

In most situations the assumption is made that the forward problem is well-posed, while the backward problem is ill-posed. These problems are described in both Korolev and Latz [25] and Clason [12]. One of the problems is the stability of the solution. In the example that is used in [25], they show that with A being a matrix with a high conditioning number, the inverse is dominated by the noise. The reconstruction of u using the inverse results in $\widetilde{u} = A^{-1}f_n = u + A^{-1}n$.

In this situation there is: $\|A^{-1}n\|_2 \approx \|n\|_2/\lambda_k$, with λ_k being the smallest eigenvalue. This shows that the reconstruction can become arbitrarily large based on the noise.

To extend this analysis to inverse problems for functions a couple of new definitions will be introduced, namely self-adjoint operators and compact operators. Starting off with the adjoint. The adjoint is defined as follows . Let H be an Hilbert space and $A : H \to H$. Then the adjoint operator A^* has to satisfy the next equality for all $f, g \in H$:

$$\langle Af, g\rangle = \langle f, A^*g\rangle$$

In the case that an operator is self-adjoint it means that $A = A^*$ and that the shown equality is the same as: $\langle Af, g\rangle = \langle f, Ag\rangle$.

Now the definition of a compact operator will be introduced. Let V, W be a Banach space with operator $T : V \to W$. T is a compact operator if it maps the closed unit ball of V into a compact subset of W, meaning in particular that the subset needs to be closed and bounded [11]. The converse only holds if the dimensions of the space are finite. In infinite dimensions closed and bounded do not mean compact.

With these definitions in place it is possible to state the spectral theorem and what a spectral decomposition is. Let A be a self-adjoint and compact operator working on Hilbert space H with $A : H \to H$. Then there exist orthogonal projections: $P_i : H \to H$ with $E_i = \mathcal{R}(P_i)$. Here E_i denotes the eigenspace which is denoted as $E_\lambda = \{f \in H \mid A(f) = \lambda f\} = \mathcal{N}(A - \lambda Id)$. Moreover, E_i is one dimensional and can have both finite and infinite number of eigenspaces with $i \in \mathbb{N}$ and corresponding real numbers λ_i, with $|\lambda_1| \geq |\lambda_2| \geq |\lambda_3| \geq ....$, all non-zero but not necessarily distinct. This results in the following equalities:

1. $A = \sum_i \lambda_i P_i$
2. $P_i P_j = \delta_{ij} P_i$

Moreover, if there are infinitely many eigenvalues λ_i, then $\lim_{i\to\infty} \lambda_i = 0$.

If a clever choice is made for each E_i, namely by choosing in each one of them a unit vector g_i, then the following also needs to hold:

$$A(f) = \sum_i \lambda_i \langle f, g_i\rangle g_i$$

Here g_i is in fact an orthonormal system and this holds for all $f \in H$. From these definitions and theorems it is easy to understand how noise can have a big impact on the reconstruction error. If A^{-1} is bounded it would be possible to do the following [18]:

$$A^{-1}(f) = \sum_i \frac{1}{\lambda_i} \langle f, g_i\rangle g_i$$

This shows the obvious problem of the decaying eigenvalues and why the backward problem can be ill-posed as was shown in the matrix example. This

entire problem can be resolved by using Tikhonov regularization. Regularization will be discussed in more detail in the next section, but it will be shortly shown here. Using Tikhonov regularization the problem gets altered to solving the following:

$$u_\alpha \in arg\min_u \|Au - (f+n)\|_H^2 + \alpha \|u\|_H^2$$

With this added term the inversion also changes. The inversion now looks like this:

$$A^{-1}(f+n) = \sum_i \frac{\lambda_i}{\lambda_i^2 + \alpha} \langle f+n, g_i \rangle g_i$$

Which means that even if the eigenvalues decay to zero that the result will be useful.

2.2 Regularization

The problem of the decaying eigenvalues shows the necessary insights into inverse problems. To solve these problems, there need to be methods that stabilizes and guarantees a solution. Subtracting the noise is not possible because most of the time the value is not known and not a constant. Therefore one of the ideas to solve this problem is through regularization techniques.

One of the classical methods of regularizing is the Tikhonov regularization. This method solves the following equation:

$$\min_u(\|Au - f_n\|^2 + \alpha\|Lu\|^2) \tag{5}$$

By adding the term $\alpha\|Lu\|^2$ an attempt is made to increase the stability of the inverse problem, while at the same time not deviate too much from the ideal solution u. The balance between stability and deviation from the optimal solution can be adjusted by tuning the parameter α. Additionally an operator L that works on u dictates how the regularization works in combination with the norm that is imposed.

One of the algorithms that solve these kind of minimization problems is called FISTA [5]. FISTA is an iterative solver that solves the following minimization problem:

$$\min_u(\|Au - f_n\|_{L^2}^2 + \alpha\|u\|_{L^1}) \tag{6}$$

This is a slightly altered version of the Tikhonov regularization and has been proven to work very well [5]. The idea is to promote sparse solutions with few nonzero coefficients. The minimization (6) is iteratively solved using a least square approximation in combination with a soft-thresholding function. This soft-thresholding function is the explicit solution of:

$$\min_u(\|u - f_n\|_{L^2}^2 + \alpha\|u\|_{L^1}) \tag{7}$$

Iterating over it obtains very good results and solves (6). This is called forward-backward splitting and will be explained in the next paragraph. Another important result has come from another variation on Tikhonov regularization, namely:

$$\min_u(\|Au - f_n\|_{L^2}^2 + \alpha\|\boldsymbol{\nabla} u\|_{L^1}) \tag{8}$$

This is a very well known minimization problem with total variation (TV) regularization. This minimization (8) will also be discussed in more detail further on. The main idea of this method is to have regularization which preserves edges very well. This minimization problem can however not be solved using forward-backward splitting and can also be somewhat slow due to the minimization over the L^1 norm of the gradient of u. If the norm of the regularization was an L^2 norm then the minimization would look like this:

$$\min_u(\|Au - f_n\|_{L^2}^2 + \alpha\|\boldsymbol{\nabla} u\|_{L^2}^2) \tag{9}$$

This model can now be solved using forward-backward splitting once again and is easier to solve than (8). The forward-backward splitting would consist of solving (10) and a least squares approximation.

$$\min_u(\|u - f_n\|_{L^2}^2 + \alpha\|\boldsymbol{\nabla} u\|_{L^2}^2) \tag{10}$$

The solution to this minimization problem (10) is relatively simple in Fourier space. In this solution the $\mathcal{F}$ sign will mean that a function has been Fourier transformed. With some simple calculations, the solution to (10) looks like this:

$$\mathcal{F}(u) = \frac{\mathcal{F}(f_n)}{1 + \alpha \mid k \mid^2} \tag{11}$$

However in this case the reconstruction of the sharp edges is once again lost. This is due to the effect that the norm has on the reconstruction process. This can be shown by the following example. Take $f : [0, 1] \to \mathbb{R}$ with boundary conditions $f(0) = 0$ and $f(1) = 1$. Now the objective is to find the function which is minimal in L^2space and the function that is minimal in L^1 space. For the L^2 norm it means to solve:

$$\min_f \int_0^1 |f'(x)|^2 \, dx$$

This yields the unique minimizer $f(x) = x$. Doing the same for the L^1 norm gets the following minimization:

$$\min_f \int_0^1 |f'(x)| \, dx$$

The answer in this space allows a step function where it does not matter where the jump is as long as it is within the interval $[0, 1]$. So a possible answer would be:

$$f(x) = \begin{cases} 0 & \text{if } x < \frac{1}{2} \\ 1 & \text{if } x \geq \frac{1}{2} \end{cases}$$

This is not the only answer, the answer $f(x) = x$ also works in the L^1 space.

2.2.1 Research question

This previous example shows how important the norm of the regularizer is and what kind of influence it can have on the reconstruction. This also shows a little bit of the underlying problem of getting regularization that is computationally fast while being edge preserving. In this report a fractional Sobolev norm will be used on the regularization with the goal to be computationally faster than the well established TV-regularization (8) and to test how well this norm preserves edges in comparison to TV-regularization. This research therefore has two questions:

1. Is using a fractional Sobolev norm faster than using TV-regularization?
2. How well does a fractional Sobolev norm preserve edges?

In chapter 3 it will be explained in more detail what a fractional Sobolev norm is and how a fractional Laplacian can induce this norm. This will at the same time show that the fractional Laplacian operator can be solved as a slight alteration of (10). How well it preserves edges it shown in the results of chapter 5, where it will also be showcased on non-linear operators.

2.3 Optimization

To actually calculate these minimization problems, computational means will be necessary. To understand how they work, the basics of these methods are useful to know, therefore a brief introduction will be given now.

In general it is not necessary for a function to have a minimizer or that it has only one minimizer. About an arbitrary function nothing can be said. Therefore a couple of assumptions need to be made. For the purpose of talking about optimization it is assumed that $f(x)$ will have at least one minimizer [7]. Starting off with how differentiability is used. Assume that there is a function $f : \mathbb{R}^n \to \mathbb{R}$, which is differentiable, and that $f(x)$ has a minimum that is located at the value $x^\star$. Obtaining this minimum of this function can now be written as follows:

$$\min_{x \in \mathbb{R}^n} f(x)$$

Since $x^\star$ is a minimum that means that it would be an answer to this problem. A necessary condition for $x^\star$ to be a minimizer is that the gradient needs to vanish, meaning that:

$$\nabla f(x^\star) = 0$$

This means that minimizing a differentiable function can also be thought of as finding an $x^\star$ such that $\nabla f(x^\star) = 0$. Now convexity will also be assumed on $f(x)$, and this is introduced with the following inequality:

$$f(y) \geq f(x) + \langle y - x, \nabla f(x) \rangle \tag{12}$$

This holds for all x, y in the domain on which $f(x)$ exists. Convexity makes it such that if a point has vanishing gradient, then it has to be a global minimum. With convexity it is still possible to have multiple points that have a vanishing gradient, but all these points then have to be a global optimum.

To compute a minimum for a large problem, iterative methods are going to be necessary. Many of these iterative methods have the same basic form.

Starting off with the initialization: Start with $k = 0$ and x_0 as starting point. The starting point can be an arbitrary guess or somewhere in the neighborhood of the minimizer because prior information is available.

For as long as the algorithm has not converged to a minimizer, which is numerically the same as saying it has not reached a stopping criterion yet, the following steps will be repeated:

- First the direction to move $d_k \in \mathbb{R}^n$ will be calculated.
- Second the step size $a_k \geq 0$ is calculated.
- Thirdly the actual step will be made, which corresponds to $x_{k+1} = x_k + a_k d_k$
- Lastly the index will be updated with $k = k + 1$

These steps are repeated until a minimum has been found [7]. A stopping criteria that is often used is $\|x_k - x_{k-1}\| < \epsilon$, where ϵ is a number that says if the progress of the iterations is smaller than it, the algorithm is close enough that a minimum has been obtained. The ϵ can be any number but it should be several orders of magnitude smaller than the expected optimum $\|x^\star\|$. A choice that is often made is to have ϵ a magnitude of 10^{-3} smaller than the optimum, but it can happen that it need to be even more. The question remaining is how to calculate d_k and a_k. There are a lot of ways to calculate these and not all will be discussed. The main variation that is of interest for this research is the (accelerated) gradient descent method. The gradient descent method uses as the name implies the gradient to calculate the direction in which to make a step. In its simplest form gradient descent calculates the step using the following rule:

$$d_k = -\nabla f(x_k)$$

This is the direction of the steepest descent at x_k. This can often be calculated relatively easily, but the convergence can be slow. This is however completely dependent on f, there are situations where taking the gradient is not even possible. Because of the possible slow convergence, a momentum term can be added to accelerated the convergence. This looks as follows:

$$p_k = x_k - x_{k-1}$$

combined with

$$d_k = -\nabla f(x_k) + \frac{b_k}{a_k} p_k$$

Where $b_k, a_k \in \mathbb{R}$. The conjugate gradient descent method and the "heavy ball" method use this as their direction. Another update rule that is often seen is Nesterov's method which describes the step direction, using the gradient as an expected future position, as this:

$$d_k = -\nabla f(x_k + b_k p_k) + \frac{b_k}{a_k} p_k$$

These momentum terms can in some cases greatly reduce the number of iterations that are required to converge. The intuition behind both methods can be seen in figure (1):

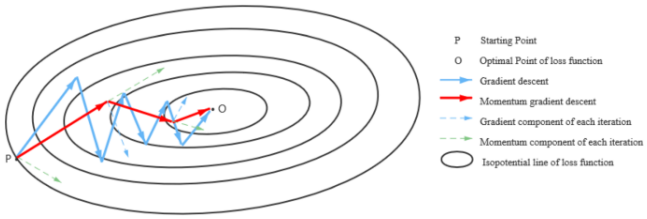


Figure 1: Comparison between gradient descent without momentum and with momentum [33]

Now the step size, there are also multiple ways of calculating this. The easiest way to do the step size is to fix it at a certain value. This is not always advised since the step size can be crucial to the convergence of the algorithm. A fixed step size is not even guaranteed to converge. However with gradient descent it is possible to pick the step size in such a manner that the convergence is guaranteed. To do this, it is assumed that $f(x)$ has a Lipschitz gradient, which means that the following holds:

$$\|\nabla f(x) - \nabla f(y)\|_2 \leq L\|x - y\|_2 \tag{13}$$

with $L > 0$. This assumption ensures that the gradient of the function changes in a controlled manner. So there will not be any erratic changes in the descent direction from point to point. Now using Cauchy-Schwartz inequality this equations can be transformed into:

$$(\nabla f(x) - \nabla f(y))^T(x - y) \leq L\|x - y\|_2^2$$

From this point it is not difficult to show that (13) is equivalent to the following statement:

$$f(y) \leq f(x) + \nabla f(x)^T(y - x) + \frac{L}{2}\|x - y\|_2^2 \tag{14}$$

To prove it, take a function $g(t) = f(x+t(y-x))$ with $t \in [0, 1]$. The function is chosen specifically because for convex functions this inequality always holds $f(x+t(y-x)) \leq tf(y)+(1-t)f(x)$. Choosing $g(t)$ this way allows the following formulation:

$$g'(t) - g'(0) = (\nabla f(x + t(y - x)) - \nabla f(x))^T(y - x) \leq tL\|x - y\|_2^2$$

Now for the last step, take the following:

$$\begin{aligned} f(y) = g(1) &= g(0) + \int_0^1 g'(t)dt \\ &\leq g(0) + \int_0^1 g'(0) + tL\|x - y\|_2^2 \\ &= g(0) + g'(0) + \frac{L}{2}\|x - y\|_2^2 \\ &= f(x) + \nabla f(x)^T(y - x) + \frac{L}{2}\|x - y\|_2^2 \end{aligned}$$

This is an important result as it creates a lower bound under which it is not necessary to reduce the time step and still guarantee convergence. Take the fixed step size to be $a_k = \frac{1}{L}$, then the gradient descent method looks like this:

$$x_{k+1} = x_k - \frac{1}{L}\nabla f(x_k)$$

Now using the previous result, a substitution will be done with $y = x_{k+1}$, yielding the next equation:

$$\begin{aligned} f(x_{k+1}) &\leq f(x_k) + \langle x_{k+1} - x_k, \nabla f(x_k) \rangle + \frac{L}{2}\|x_k - x_{k+1}\|_2^2 \\ &= f(x_k) + \left\langle -\frac{1}{L}\nabla f(x_k), \nabla f(x_k) \right\rangle + \frac{L}{2}\|\frac{1}{L}\nabla f(x_k)\|_2^2 \\ &= f(x_k) - \frac{1}{L}\|\nabla f(x_k)\|_2^2 + \frac{1}{2L}\|\nabla f(x_k)\|_2^2 \\ &= f(x_k) - \frac{1}{2L}\|\nabla f(x_k)\|_2^2 \end{aligned}$$

This is an important result as it shows that $f(x_{k+1}) < f(x_k)$ for as long as the algorithm is not converged. It also shows that if the gradient is large then the progress made in each iteration will also be large. This shows that for gradient descent if the Lipschitz constant can be found there is a guarantee that the algorithm will converge. As mentioned earlier, there are other methods to calculate the step size, but those methods will not be discussed seen as they will not be used. Another point that deserves a mention is that when the assumption of strong convexity is made for $f(x)$, then an upper bound can be found as well. A function is strong convex if the following inequality holds:

$$(\nabla f(x) - \nabla f(y))^T (x - y) \geq m\|x - y\|_2^2$$

for all x, y in the domain of the function, with $m \in \mathbb{R}$ and $m > 0$.

With this assumption it is possible to significantly improve the convergence rate to an optimum, however the assumption made on $f(x)$ is also more restrictive and not always possible in optimization problems. It is therefore noteworthy in the cases that the objective function is indeed strong convex, but it is unfortunately not always reasonable to assume it.

The next step that does need to be addressed is some of the assumptions that have been made on the function $f(x)$. At the beginning it was assumed that $f(x)$ was convex and differentiable. Sadly, it is not always the case that a function is differentiable or sometimes called smooth. A very simple example would be the l_1 norm, if a function is minimized over this norm then it not differentiable. This problem can fortunately be resolved through the means of subgradients. What are subgradients? To answer this it is necessary to first look back at (12). When $f(x)$ is not differentiable at some point x, a subgradient might be found. A subgradient of the function $f(x)$ at point x is a vector g such that:

$$f(y) \geq f(x) + \langle y - x, g \rangle$$

for all y in the domain of $f(x)$. This subgradient does not have to be unique, it is possible that there are multiple subgradients for the same value of x. The collection of all subgradients at point x, is called the subdifferential:

$$\partial f(x) = \{g : f(y) \geq f(x) + \langle y - x, g \rangle, \forall y \in \mathbb{R}^n\}$$

With this established two general properties can be established:

1. When $f(x)$ is convex and differentiable at point x, then the subdifferential contains only one element, namely the gradient.

2. If $f(x)$ is convex and a function on $\mathbb{R}^n$, then within the domain where $f(x)$ is convex the subdifferential is non-empty and bounded at all x in the interior of the domain of $f(x)$.

When a function is not convex these points do not have to hold in general. How this subdifferential works is going to be shown on the function $f(x) = \|x\|_1$. This is a good example and it will give useful insight into an algorithm that will be shown later on. So the question would be to find the subdifferential of $\partial\|x\|_1$ at the point x. According to (12) this would mean that:

$$\|y\|_1 \geq \|x\|_1 + \langle y - x, g\rangle$$

for all $y \in \mathbb{R}^n$. Now a new set will be introduced that looks like this:

$$\mathit{\Gamma}(x) = \{n : x_n \neq 0\}$$

This set contains all indices of x that are not zero. This in combination with the right side of the inequality allows us to rewrite the equation into something more useful:

$$\begin{aligned}\|x\|_1 + \langle y - x, g\rangle &= \sum_{n=1}^{N} |x_n| + \sum_{n=1}^{N} g_n(y_n - x_n) \\ &= \sum_{n\in\Gamma} |x_n| - g_n x_n + \sum_{n=1}^{N} g_n y_n\end{aligned}$$

Now set g_n to be the following:

$$g_n = sign(x_n) = \begin{cases} 1 & \text{if } \ x_n \geq 0 \\ -1 & \text{if } \ x_n < 0 \end{cases}$$

With this choice of g_n it follows that $g_n x_n = |x_n|$ for all $n \in \Gamma$ thus the sum reduces to:

$$\sum_{n\in\Gamma} |x_n| - g_n x_n = \sum_{n\in\Gamma} |x_n| - |x_n| = 0$$

With this smart choice of $g_n = sign(x_n)$, the inequality reduces to:

$$\|y\|_1 \geq \langle y, g\rangle$$

This equation will hold as long as $|g_n| \leq 1$ for all n. The vector g would look as follows:

$$\begin{aligned} g_n &= sign(x_n) && \text{if } \ n \in \Gamma \\ |g_n| &\leq 1 && \text{if } \ n \notin \Gamma \end{aligned}$$

This means that $g_n \in \partial \|x\|_1$.

With this in place some optimality conditions can be stated for convex functions. Let $f(x)$ be a convex function, then $x^\star$ is a solution to the problem:

$$\min_{x \in \mathbb{R}^N} f(x)$$

if and only if

$$0 \in \partial f(x)$$

The proof of this is trivial. Use the fact that if $0 \in \partial f(x)$, then

$$\begin{aligned} f(y) &\geq f(x^\star) + \langle y - x^\star, 0 \rangle \\ &= f(x^\star) \end{aligned}$$

for all y within the domain of $f(x)$. Which proves that $x^\star$ is a solution. Now the other way, if

$$f(y) \geq f(x^\star)$$

for all y within the domain of $f(x)$. Then it must also be true that

$$f(y) \geq f(x^\star) + \langle y - x^\star, 0 \rangle$$

which shows that $0 \in \partial f(x)$ has to hold.

With this all ready, it is possible to adapt the gradient descent method to a method which also works on non-smooth functions. The basics will again be the same:

$$x_{k+1} = x_k + a_k d_k$$

Now d_k can be any subgradient at x_k. This could mean that at every step there are many choices for d_k, which also impacts the speed at which you converge if you even converge at all. With the right step size it can be analytically proven that this subgradient method will converge even if it is slow to reach an optimum [28]. However this analysis will not be discussed, because it is long and the subgradient method will not be used. The main reason that the subgradient method converges slowly and that for a fixed step size, it does not have to converge, is because it is possible to have large subgradients near and at the solution. A simple example of this can be the function $f(x) = \|x\|_{l^1}$, which would look like this if $f : \mathbb{R} \to \mathbb{R}$.

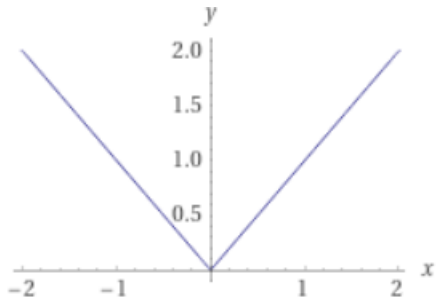


Figure 2: Example function for subgradient method

To counteract this problem it is possible to add a differentiable regularization term. This regularization terms is chosen in such a way that if $x^\star$ is a minimizer for $f(x)$, then it is also a minimizer for the function with regularization. To that end the following regularization is chosen:

$$\min_{x \in \mathbb{R}^n} f(x) + \delta \left\| x - x^\star \right\|_2^2$$

with $\delta > 0$ and $\delta \in \mathbb{R}$. With this as the new minimization problem, the problem has become strongly convex. Furthermore, the only subgradient at the solution is the zero vector. This gets completely rid of the problem that the subgradient method had. There is however a problem with new idea and that is that it requires that $x^\star$ is known in advance. This is in all practicality not very useful, because the goal was to find $x^\star$ by solving a minimization problem. However, this new minimization can be transformed into an iterative algorithm, which is called the proximal algorithm or proximal point method. This algorithm uses the following iterations:

$$x_{k+1} = arg \min_{x \in \mathbb{R}^n} \left(f(x) + \frac{1}{2a_k} \left\| x - x_k \right\|_2^2 \right) \tag{15}$$

With $f(x)$ being convex and $f(x) + \delta \parallel x - z \parallel_2^2$ being strongly convex for all $\delta > 0$ and $z \in \mathbb{R}^N$. The mapping of x_k to x_{k+1} is well defined. To make notation easier later on, a definition of the proximal operator is going to be made. This looks as follows:

$$prox_{a_k} f = arg \min_{x \in \mathbb{R}^n} \left(f(x) + \frac{1}{2a_k} \left\| x - x_k \right\|_2^2 \right) \tag{16}$$

The original optimization problem has now been altered such that there is a sequence of many optimization problems. This however is sometimes easier to solve. The regularizer will make every iteration of the algorithm easier to solve while naturally disappearing as the algorithm gets closer to the solution. This even converges for a fixed step size.

This proximal point algorithm can be interpreted as a variation on gradient descent. To show this, a differential equation for gradient flow will be used.

This has the equations:

$$\begin{aligned} x'(t) =& -\nabla f(x) \\ x(0) =& x_0 \end{aligned}$$

The points for which the system has an equilibrium, are the points such that $\nabla f(x) = 0$. This corresponds perfectly with the minimizers for $f(x)$ because the function is convex. Now this differential equation is going to discretized using the forward Euler method. This yield the approximation:

$$\frac{x(t+h) - x(t)}{h} \approx -\nabla f(x)$$

Where h needs to be small. Now this equation turns into gradient descent iterations when it is rewritten as this:

$$x_{k+1} = x_k - h\nabla f(x_k)$$

The same differential equations is now going to be discretized using the backward Euler method or also known as backward difference. This yields this equation:

$$\frac{x(t) - x(t-h)}{h} \approx -\nabla f(x)$$

Where h needs to be small. This is going to be rewritten, where the iterates are looking as follows:

$$x_{k+1} = x_k - h\nabla f(x_{k+1})$$

Now if h is chosen equal to a_k and the equations is rewritten a bit more into:

$$0 = \nabla f(x_{k+1}) + \frac{1}{a_k}(x_{k+1} - x_k)$$

In this formula it is not very easy to calculate an iteration. This is because the next point x_{k+1} has to be found such that it obeys the equation. This however is the same as what the proximal operator does. What is even more so, if $f(x)$ is differentiable then the two equations are equivalent.

$$\begin{aligned} x_{k+1} =& arg\min_{x \in \mathbb{R}^n} \left(f(x) + \frac{1}{2a_k} \|x - x_k\|_2^2 \right) \\ & \Updownarrow \\ 0 =& \nabla f(x_{k+1}) + \frac{1}{a_k}(x_{k+1} - x_k) \end{aligned}$$

This is a nice result, because the proximal point method can be interpreted as a backward Euler discretization for gradient flow. The backward Euler method is known to be stable [28], so this is a nice result for the proximal point method. What is also note worthy is the fact that differentiability of $f(x)$ was assumed in order for the equivalence relation to work. However, $f(x)$ need not to be differentiable for the proximal operator to work. It can sometimes also be computed

if $f(x)$ does not have a gradient. Functions whose proximal is easy to compute are therefore of interest. Below it will be calculated analytically for $\|\cdot\|_{l^1}$ on $\mathbb{R}^n$, while the proposed algorithm based on a fractional Sobolev seminorm also has an easy to compute proximal.

Solving every iteration of the proximal point method could be quite tedious. Therefore it might sometimes be nice that when $f(x)$ is differentiable to replace it with its linear approximation $f(x_k) + \langle x - x_k, \nabla f(x_k)\rangle$. This will be put into (15), which turns it into:

$$\begin{aligned}
x_{k+1} = prox_{a_k} f(x_k) &= arg\min_{x\in\mathbb{R}^n}\left(f(x) + \frac{1}{2a_k}\|x - x_k\|_2^2\right)\\
&\approx arg\min_{x\in\mathbb{R}^n}\left(f(x_k) + \langle x - x_k, \nabla f(x_k)\rangle + \frac{1}{2a_k}\|x - x_k\|_2^2\right)\\
&= arg\min_{x\in\mathbb{R}^n}\left(\frac{1}{2a_k}\|\nabla f(x_k)\|_2^2 + \langle x - x_k, \nabla f(x_k)\rangle + \frac{1}{2a_k}\|x - x_k\|_2^2\right)\\
&= arg\min_{x\in\mathbb{R}^n}\left(\frac{1}{2a_k}\|x - x_k + a_k\nabla f(x_k)\|_2^2\right)\\
&= x_k - a_k\nabla f(x_k)
\end{aligned}$$

This shows, that taking a linear approximation for $f(x)$, reduces the proximal method into standard gradient descent. Now the problem is going to be altered slightly. The objective function is going to broken into two parts which would look like this:

$$f(x) = g(x) + h(x)$$

Now the assumptions are made that $g(x)$ is convex and differentiable, while $h(x)$ is only convex but there exists a proximal operator for $h(x)$. A simple example that has already been seen is (6). The same derivation as was seen earlier is now again going to be done, but now on $f(x) = g(x) + h(x)$. The linear approximation of $g(x)$ will be inserted resulting in:

$$\begin{aligned}
x_{k+1} &= arg\min_{x\in\mathbb{R}^n}\left(g(x_k) + \langle x - x_k, \nabla g(x_k)\rangle + h(x) + \frac{1}{2a_k}\|x - x_k\|_2^2\right)\\
&= arg\min_{x\in\mathbb{R}^n}\left(h(x) + \frac{1}{2a_k}\|x - x_k + a_k\nabla g(x_k)\|_2^2\right)\\
&= prox_{a_k} h(x_k - a_k\nabla g(x_k))
\end{aligned}$$

Now there is a new update rule that is called forward-backward splitting. Here the forward refers to the gradient step, while the backward refers to the proximal step because that was equivalent to the backward Euler method. To show how this would work, it will be applied to (6). The minimization problem was:

$$\min_u(\|Au - f_n\|_{L^2}^2 + \alpha\|u\|_{L^1})$$

Now take $g(u) = \|Au - f_n\|_2^2$, which is clearly a convex and differentiable function. Furthermore, $h(u) = \alpha\|u\|_{L^1}$ and $g'(u) = 2A^T(Au - f_n)$. Now this

gets inserted into the proximal operator for the l_1 norm that was shown in an earlier example.

$$\begin{aligned} prox_{a_k} h(u) &= arg \min_{z \in \mathbb{R}^n} \left(h(z) + \frac{1}{2a_k} \|u - z\|_2^2 \right) \\ &= arg \min_{z \in \mathbb{R}^n} \left(\alpha \|z\|_1 + \frac{1}{2a_k} \|u - z\|_2^2 \right) \\ &= \Lambda_{\alpha, a_k}(u) \end{aligned}$$

Where Λ_{α,a_k} is the soft-thresholding operator:

$$\Lambda_{\alpha,a_k}(u)_i = \begin{cases} (|u_i| - a_k\alpha) sgn(u_i) & \text{if } |u_i| \geq a_k\alpha \\ 0 & \text{if } |u_i| < a_k\alpha \end{cases}$$

Now the gradient step for $g(x)$ and the proximal step for $h(x)$ are going to be combined. This will come together in this equation:

$$u_{k+1} = \Lambda_{\alpha,a_k}(u_k + 2a_k A^T(Au - f_n))$$

This method is called iterative soft-thresholding algorithm or in short ISTA. ISTA outperforms the subgradient method very significantly and is the predecessor of FISTA which outperforms ISTA. FISTA will be discussed in a later chapter.

3 Theoretical basis

In this research a new type of regularization will be considered regarding Tikhonov minimization problems. The main idea behind this new minimization problem is the regularization term which tries to preserve sharp edges which is at the same time computational efficient. The idea of having sharp edges is not new. In [30] they already suggested a denoising algorithm based on total variation. This translates to the minimization problem that is as follows:

$$\min_u(\|Au - f_n\|_{L^2}^2 + \alpha\| \bigtriangledown u\|_{L^1}) \tag{17}$$

The algorithm to solve this type of minimization can be quite slow in the case of a linear operator. In the case of a non-linear operator it is even worse and there is still discussion on how to use TV-regularization. However in this research instead of using the gradient on u, a fractional Laplacian operator will be used as regularizer. The minimization problem would look as follows:

$$\min_u(\|Au - f_n\|_{L^2}^2 + \alpha\| \bigtriangleup^{\frac{s}{2}} u\|_{L^2}^2) \tag{18}$$

A more detailed explanation will be given in the next section to the properties of this fractional Laplacian operator and why this would be a good idea to use this instead of the TV-regularization. The main purpose of this research is to investigate how well a fractional Laplacian operator performs compared to the already existing minimization (17) on a linear operator and its performance to FISTA on a non-linear operator.

3.1 Fractional Laplacian

In the recent years Fractional derivatives have gained a lot of interest [2]. This is because these operators appear in a variety of fields and models. To name a few, fracture mechanics, turbulence and models with periodic boundary conditions [10, 13, 15, 20, 29]. These are not the only field. In image denoising it has already been shown that it can work as well. In [2] they show that a fractional derivative can have its advantages. They apply it on the well known model of Rudin-Osher-Fatemi (ROF) [2, 30]. This model is the well known total variation based image denoising model also known as TV-regularization. This method of regularization is a celebrated method and is known to work very well. Starting with the model, which looks as follows:

$$E(u) = \alpha|Du|(\Omega) + \|u - f\|^2 \tag{19}$$

In this model $u \in BV(\Omega) \cap L^2(\Omega)$ is sought to be minimized. Furthermore Ω is the domain on which the image exists [2, 30], f is the noisy image, $BV(\Omega)$ is the set of functions with bounded variation [4] and α is the regularization parameter . The term $|Du|(\Omega)$ is the total variation, which is the regularizer that allows discontinuities. This is good, because in images the discontinuities

represent the sharp edges [2]. To understand why these discontinuities are allowed a deeper dive into the set $BV(\Omega)$ is now going to be made. Starting of with its definition:

$$BV(\Omega) = \left\{u \in L^1(\Omega) : V(u, \Omega) < \infty\right\}$$

This definition however is not very enlightening since $V(u, \Omega)$ is not defined. So, this is what is going to be done next:

$$V(u, \Omega) = \sup \left\{ \int_\Omega u(x) div(\phi(x)) dx : \phi \in C_c^1(\Omega, \mathbb{R}^n), \|\phi\|_{L^\infty(\Omega)} \leq 1 \right\}$$

This can also notated as $V(u, \Omega) = \int_\Omega |Du|$. Which is exactly the same as the total variation term.

The (19) can now be seen to be a generalization of (17) with some assumptions. If u is smooth, take:

$$\phi = \begin{cases} \frac{\nabla u}{|\nabla u|} & \text{if } |\nabla u \neq 0 \\ 0 & \text{if } |\nabla u = 0 \end{cases}$$

Which results in:

$$\min_u (\|Au - f\|_{L^2}^2 + \alpha \| \bigtriangledown u\|_{L^1}) \tag{20}$$

On the domain Ω, which is the same as (17). A small side note has to be made that the operator A has been added to the equation. As long as it is linear and non-zero, this is not a problem. An example function is now going to be given, if:

$$u(x) = \begin{cases} 1 & \text{if } |x > \frac{1}{2} \\ 0 & \text{if } |x \leq \frac{1}{2} \end{cases} \tag{21}$$

With $\Omega = [0, 1]$ has this as a graph:

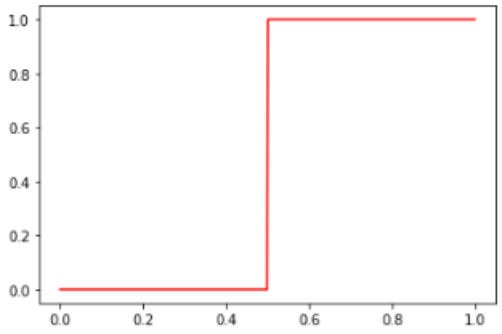


Figure 3: Example function

For this example function it holds that $\int_\Omega |Du| = 1$. Meaning that $u \in BV(\Omega)$. This will now be shown. For the one dimensional example the $V(u, \Omega)$ can be written a bit different [4] namely:

$$V(u, \Omega) = \sup_{P \in \Theta} \sum_{i=0}^{n-1} |u(x_{i+1}) - u(x_i)|$$

Where Θ is the ordered set of elements consisting of all partitions of the interval. This can be written more clearly as follows:

$$\Theta = \{P = \{a = x_0, x_1, ..., x_n = b\} \mid x_i < x_{i+1}, \quad 0 \leq i \leq n\}$$

From this it is very easy to see why $\int_\Omega |Du| = 1$ would hold for this function. This is because the only part where the summation is not 0 is at the jump. Meaning that if that jumps happens between x_j and x_{j+1} that $\sum_{i=0}^{n-1} |u(x_{i+1}) - u(x_i)| = |u(x_{j+1}) - u(x_j)| = 1 - 0 = 1$. This holds for any partition that is made so $V(u, \Omega) = 1$, meaning that this example function is in the set of bounded variation.

A disadvantages that come up in [2] is that total variation is difficult computation wise. Therefore one might get the idea to alter the space in which a solution is sought. A natural idea would be to seek the solution in a Sobolev space. A Sobolev space is defined as follows:

$$W^{k,p}(\Omega) = \left\{u \in L^P(\Omega) : D^\alpha u \in L^P(\Omega) \ \forall |\alpha| \leq k\right\} \tag{22}$$

Where D^α denotes the α-th weak derivative. A special case for $p = 2$ is denoted as as $W^{k,2} = H^k$. What is easy to see is that if $k = 0$ the Sobolev space reduces to an L^p space. With this definition of a Sobolev space it is easy to understand why this could be a good alternative to $BV(\Omega)$. The least regular Sobolev space is the space $W^{1,1}$. Unfortunately $u \notin W^{1,1}$ in the case of the example (21), this is due to the fact that the weak derivative does not exist at $\frac{1}{2}$. The next idea would then be to go to a space with even less regularity. Such a space would be a fractional Sobolev space. The definition for this is next. Take $s \in (0, 1)$ fixed and for any $p \in [1, \infty)$, the fractional Sobolev space is defined as:

$$W^{s,p}(\Omega) = \left\{u \in L^p(\Omega) : \frac{|u(x) - u(y)|}{|x - y|^{\frac{n}{p} + s}} \in L^p(\Omega \times \Omega)\right\} \tag{23}$$

This space also has a norm which is:

$$\|u\|_{W^{s,p}(\Omega)} = \left(\int_\Omega |u|^p dx + \int_\Omega \int_\Omega \frac{|u(x) - u(y)|^p}{|x - y|^{n+sp}} dxdy\right)^{\frac{1}{p}} \tag{24}$$

In comparison to the Sobolev spaces the fractional Sobolev spaces have some strange properties. An explanation will now be given as to why this definition

of a fractional Sobolev space makes sense. In order to do this a comparison is going to be made. Starting with $x, y \in \Omega$ for $\frac{|u(x)-u(y)|}{|x-y|^{\frac{n}{p}+s}} \in L^p(\Omega \times \Omega)$ to hold it needs to be true that:

$$\iint_{\Omega\times\Omega} \frac{|u(x)-u(y)|^p}{|x-y|^{n+sp}} dxdy < \infty \tag{25}$$

The key point is that for $q = n + sp \geq n$,

$$\int_{\Omega} \frac{1}{|x-y|^q} dy = \infty.$$

This imposes on (25) that $|u(x) - u(y)| \to 0$ faster then $|x - y| \to 0$ for all $x, y \in \Omega$. When looking back at the example (21). It can now be seen that for this example $u \in H^s$ as long as $s < \frac{1}{2}$ holds.

Focusing on the case $p = 2$, the fractional Sobolev space $W^{s,2}(\mathbb{R}^n) = H^s(\mathbb{R}^n)$ is a Hilbert space and is strictly related to the fractional Laplacian operator [17]. To showcase this take a $u \in \mathcal{L}$, where $\mathcal{L}$ is the Schwartz space of rapidly decaying functions and $s \in (0, 1)$. The fractional Laplacian operator is defined by:

$$(-\triangle)^s u(x) = C(n, s) P.V. \int_{\mathbb{R}^n} \frac{u(x) - u(y)}{|x-y|^{n+2s}} dy \tag{26}$$

Here the $C(n, s)$ is a dimensional constant that is exactly given by:

$$C(n, s) = \left(\int_{\mathbb{R}^n} \frac{1 - \cos(\zeta_1)}{|\zeta|^{n+2s}} \right)^{-1}$$

Furthermore the *P.V.* stands for principle value. This is because of the singularity that exists within the integral at $x = y$. This however can be resolved, for $s \in (0, \frac{1}{2})$ it can be shown that this integral in not really singular near x [17]. What is even more so is that (26) can be rewritten such that it is a weighted second order differential quotient. Again let $s \in (0, 1)$ and $u \in \mathcal{L}$ then:

$$(-\triangle)^s u(x) = -\frac{1}{2} C(n, s) \int_{\mathbb{R}^n} \frac{u(x+y) + u(x-y) - 2u(x)}{|y|^{n+2s}} dy$$

for all $x \in \mathbb{R}^n$ [17]. Now it will be shown why changing the space matters for the computation time. Fourier transforms are known to be a very fast computation wise. To that end the fractional Sobolev space will now also be defined via its Fourier transform. This definitions is as follows:

$$\widehat{H}^s(\mathbb{R}^n) = \left\{ u \in L^2(\mathbb{R}^n) : \int_{\mathbb{R}^n} (1 + |\xi|^2)^s |\mathcal{F}u(\xi)|^2 d\xi < \infty \right\}$$

In this case the $\mathcal{F}$ denotes the Fourier transformation. To enforce a norm on this space a small step back needs to be taken. First the Gagliardo seminorm has to be introduced, this is as follows:

$$[u]_{W^{s,p}(\Omega)} = \left(\int_{\Omega} \int_{\Omega} \frac{|u(x) - u(y)|^p}{|x - y|^{n+sp}} dxdy \right)^{\frac{1}{p}}$$

When compared with the norm for fractional functions it is easy to see that this seminorm is simply only the second part of it. With this seminorm the norm for the functions in the fourier space can be written like this:

$$[u]^2_{H^s(\mathbb{R}^n)} = 2C(n,s)^{-1} \int_{\mathbb{R}^n} |\xi|^{2s} |\mathcal{F}u(\xi)|^2 d\xi$$

With this available it is now possible to show that for $s \in (0,1)$ and $u \in \mathcal{L}$ and the fractional operator defined as 26, that the following statement holds:

$$(-\triangle)^s u(x) = \mathcal{F}^{-1}(|\xi|^{2s}(\mathcal{F}u)) \qquad \forall \xi \in \mathbb{R}^n$$

This result is extremely useful to since this means that the following relations is also true:

$$\mathcal{F}((-\triangle)^s u(x)) = |\xi|^{2s}(\mathcal{F}u) \qquad \forall \xi \in \mathbb{R}^n$$

This shows that the computation of the fractional operator can be done very quickly in Fourier space as it is nothing more than a multiplication with its frequencies and Fourier transforms are known to be fast by using the fast Fourier transformation, which is a algorithm. Moreover, it shows that the fractional Laplacian can be viewed as a pseudo-differential operator. With this Fourier transformation now properly defined it is also easy to see that periodicity is at hand. This is because a Fourier series, which is what is used to approximate any function that gets transformed into Fourier domain, is a sum of sinusiods. This means that the Fourier transformation induces periodicity. As a result of this the domain on which the regularization works can be expanded to $\mathbb{T}^n$ which is an n-dimensional torus. This is interesting because now the following can be shown (this follows from the citations of [2]), as long as $s < \frac{1}{2}$:

$$BV(\mathbb{T}^n) \cap L^\infty(\mathbb{T}^n) \subset H^s(\mathbb{T}^n)$$

This means that TV-regularization is completely embedded within the fractional Sobolev space. So, a fractional Laplacian operator has the same capabilities as TV-regularization while at the same time being far easier to compute.

3.2 FISTA and its fractional Laplacian version

FISTA stands for Fast Iterative Shrinkage Thresholding Algorithm and is an improvement of the previous algorithm called ISTA [5]. FISTA is one of the many algorithms and approaches that solve convex but non smooth minimization problems. FISTA is solves the following Tikhonov minimization:

$$\min_u(\|Au - f_n\|_{L^2}^2 + \alpha\|u\|_{L^1}) \tag{27}$$

This is the same as has been seen in (6). The L^1 regularization term is a very important part, it induces sparsity into the solution. What needs to be mentioned is that the model that is shown has regularization in L^1, but computers cannot work with continuous function and discretize them in order to be able to work with them. This means that in practice FISTA would solve a sequence in l_1. Meaning that the output of u would be some kind of vector of which it makes sense that sparsity might be a good attribute. Another benefit of this type of of regularization is that the L^1 norm is less susceptible to outliers compared to other norms like L^2.

FISTA almost works the same as ISTA. In FISTA a standard gradient decent step is taken, which is combined with a soft thresholding step. So the general computational outlay looks as follows [5]:

$$\begin{aligned} y_k =& \Lambda(u_k - 2LA^*(Au - f_n)) \\ t_{k+1} =& \frac{1 + \sqrt{1 + 4t_k^2}}{2} \\ u_{k+1} =& y_k + \left(\frac{t_k - 1}{t_{k+1}}\right)(y_k - y_{k-1}) \end{aligned} \tag{28}$$

Where L is a constant corresponding to the Lipschitz constant of the gradient of$\|Au - f_n\|_{L^2}^2$. Furthermore the algorithm is initialized with $t_1 = 1$ and $u_1 = y_0 \in \mathbb{R}^n$. The Λ signifies the soft thresholding step which looks as follows:

$$\Lambda(u)_i = \begin{cases} (|u_i| - \alpha)sgn(u_i) & \text{if } |u_i| \geq \alpha \\ 0 & \text{if } |u_i| < \alpha \end{cases} \tag{29}$$

This soft thresholding steps causes small components of u to be reduced to zero. Everything between $(-\alpha, \alpha)$ will be set to zero. If it is not in that interval it will be the line $y = x \pm \alpha$, where it is either a plus or a minus depending on which side of the aforementioned interval you are. This is easier to see by using the graph.

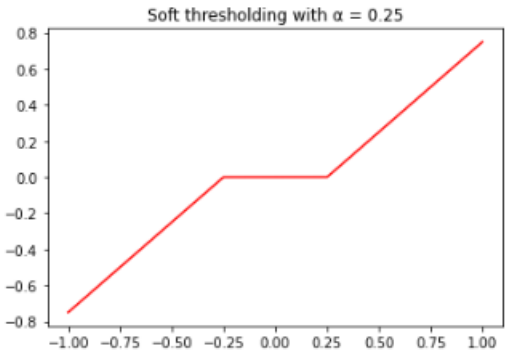


Figure 4

The choice of α is thus very important for this algorithm. If it is chosen to low, a lot of noise will stay in the solution, while if it is to high the original signal might be compressed. The only difference between FISTA and ISTA is that FISTA in its gradient descent step uses Nesterov momentum while ISTA does a regular gradient descent.

Another comparison that needs to be made is that the computational steps from (28) and(29) are sequential. In [5] it is shown that this equates to solving the minimization (6). It is important that this is possible, because the proposed minimization with the fractional Laplacian as regularizer, is going to have a similar sequential computation step. The main difference is that the proximal step will now be a fractional Laplacian. This means that the proximal step that was shown in (29) will become this for the fractional Laplacian:

$$\Lambda(u) = \mathcal{F}^{-1}\left(\frac{\mathcal{F}(u)}{1+\alpha \mid k \mid^{2s}}\right)$$

This is almost the same as equations (10) only now with the fractional constant s in it.

FISTA is an algorithm designed for linear inverse problems and guarantees that when well implemented it will come close to u_{ideal}. In a non-linear setting however, FISTA is not guaranteed to converge to u_{ideal}. Local minima can cause the algorithm to get stuck in a solution that is not close to u_{ideal}. However, this is a problem for all algorithms, but with the right initialization it will often converge to u_{ideal} even in a non-linear situation. Another slight adaptation that needs to be made in the non-linear setting is, instead of using the adjoint of A it is necessary to use $(A'(f))^*$. This is the adjoint of the Fréchet derivative and it will be explained a bit more in the section on the non-linear operator that is used.

FISTA is one of the algorithms that the new optimization method is going to be compared against. FISTA has a good reputation as an inverse problem solving algorithm. It is quite time efficient with good results [5] and therefore is a suitable algorithm to compare the performance with.

3.3 Bases

A mathematical basis is used to store coefficients instead of functions. Therefore the following definition will be given for a basis in a separable Hilbert space:

Definition 2

A set of basis elements $\beta = \{e_1, e_2, ...\}$ is an orthonormal basis of the separable Hilbert space H if:

- $\langle e_j, e_k \rangle = \delta_{jk}$ *for* $j, k \in \mathbb{N}$
- *For all* $f \in H$ *there are unique coordinates* $\lambda_k = \langle e_k, f \rangle$ *such that*

$$f = \sum_{k \in \mathbb{N}} \lambda_k e_k$$

Here δ_{jk} is the Kronecker delta.

Furthermore, it is possible that this space is infinite in dimensions causing the basis to also be infinite in elements. This means that computationally it would only be possible to approximate functions as a computer cannot hold infinite basis elements in its memory nor is the computation time of the basis coefficients feasible. Therefore, it is important to choose an appropriate basis that can approximate any function or vector reasonably well with a reasonable amount of nonzero coefficients. Therefore the following basis has been chosen.

3.3.1 Haar wavelets

Haar wavelets is a particular set of wavelets thought of by the mathematician Haar. It is a rectangular wave as can be seen from the figure below:

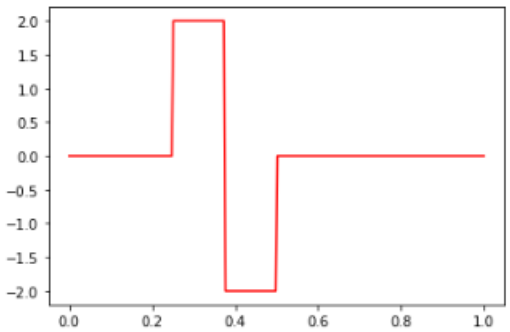


Figure 5: Example of a Haar wavelet

This wavelet can be captured with the following mathematical equation:

$$\varphi(x) = \begin{cases} 1 & \text{if } |0 \leq x < 0.5 \\ -1 & \text{if } |0.5 \leq x < 1 \\ 0 & \text{else} \end{cases} \tag{30}$$

In combination with:

$$\psi(x) = \begin{cases} 1 & \text{if } |0 \leq x \leq 1 \\ 0 & \text{else} \end{cases} \tag{31}$$

this $\varphi(x)$ function can be shifted and transformed such that it forms a complete orthonormal basis[32]. This is done with the following shifts:

$$\varphi_{j,k}(x) = \psi(2^j x - k) \tag{32}$$

for j a nonnegative integer and k an integer within the bounds: $0 \leq k \leq 2^j - 1$. These j and k can be captured in the following set: $\Lambda = \{1\} \cup \{(j,k) : j \in \mathbb{N}_0, 0 \leq k \leq 2^j - 1\}$. The j is responsible for the scale of the Haar wavelet, while the k defines the location of the wavelet.

All of these wavelets are orthogonal to each other if they do not have the same j, k. Using these wavelets any function can be recreated within the domain of $[0, 1]$ [32], this is because the set of wavelets is complete. This means that the set of Haar wavelets is a basis with the added benefit that any function can be represented as it's Haar coefficients, which are found using the inner product from definition 2. Another benefit of these Haar wavelets is that they work very well on discontinuous function due to their own discontinuous nature. This is the main reason that Haar wavelets are chosen as a basis. The Haar wavelets can be seen as a discretization of Du, meaning that on images it can simulate the regularization term $\|\nabla u\| \approx \|Hu\|$ [8]. In this case H is the Haar matrix containing all Haar wavelets. These Haar wavelets will be used as a basis within the programs to approximate functions. With the Haar wavelets as basis it is possible to simulate the edge preserving nature of TV-regularization within FISTA. As mentioned earlier TV-regularization is not easy to implement on non-linear operators and therefore FISTA will be used on the non-linear operator. It is however unfair to compare the fractional Laplacian operator to FISTA on edge preserving nature when the fractional Laplacian is intended for this job and FISTA not necessarily. Therefore a basis of Haar wavelets is necessary, because this enables FISTA to reconstruct sharp jumps since it minimizes over the Haar wavelet coefficients [8]. This will make the comparison on the non-linear operator more legitimate to make.

3.4 SSIM

SSIM stands for structural similarity index measurement. This was a new type of measurement to compare the similarity of images. It was developed in 2002 by Zhou Wang, & A. Bovik [34]. the main method to compare similarity of images at that point was to use mean squared error, which was not satisfying as was shown by them [34]. Instead of looking at a summation of the errors they proposed to look at a combination of three factors, namely: loss of correlation, luminescence distortion and contrast distortion. This is all captured into one formula which is as follows:

Let $x = \{x_i | i = 1, 2, ..., N\}$ and $y = \{y_i | i = 1, 2, ..., N\}$ corresponding to the pixels of the image on which the SSIM score Q is going to be calculated. Then the quality index is defined as:

$$Q = \frac{4\sigma_{xy}\bar{x}\bar{y}}{(\sigma_x^2 + \sigma_y^2)(\bar{x}^2 + \bar{y}^2)}$$

where

$$\bar{x} = \frac{1}{N}\sum_{i=1}^{N} x_i, \qquad \bar{y} = \frac{1}{N}\sum_{i=1}^{N} y_i$$

$$\sigma_x^2 = \frac{1}{N-1}\sum_{i=1}^{N}(x_i - \bar{x})^2, \qquad \sigma_y^2 = \frac{1}{N-1}\sum_{i=1}^{N}(y_i - \bar{y})^2$$

$$\sigma_{xy} = \frac{1}{N-1}\sum_{i=1}^{N}(x_i - \bar{x})(y_i - \bar{y})$$

The range of Q is $[-1, 1]$, where the best value is 1 and consequently the worst is -1. To see how the three factors are integrated into Q the formula is going to be split up. This results into:

$$Q = \frac{\sigma_{xy}}{\sigma_x\sigma_y} \cdot \frac{2\bar{x}\bar{y}}{\bar{x}^2 + \bar{y}^2} \cdot \frac{2\sigma_x\sigma_y}{\sigma_x^2 + \sigma_y^2}$$

The first component is the correlation coefficient between x and y. The second component measures how close the mean luminescence is between x and y and the third component measures how similar the contrasts of the images are.

It was shown that this method outperformed mean squared error by a large margin in its relation to subjective image assessment [34]. Up to this day SSIM is one of the most used measuring tools on images and is also very useful in image reconstructions. Therefore this will be one of the methods which will be used to compare the results on.

3.5 Radon transformation

The Radon transform is an integral transformation, which takes a function that is defined on a plane, to a function defined on the space of lines. This can be extended to higher dimensions, but that is not necessary in this case. The value at a particular line is equal to the line integral over the function over that line. To understand this easier, there is the next illustration:

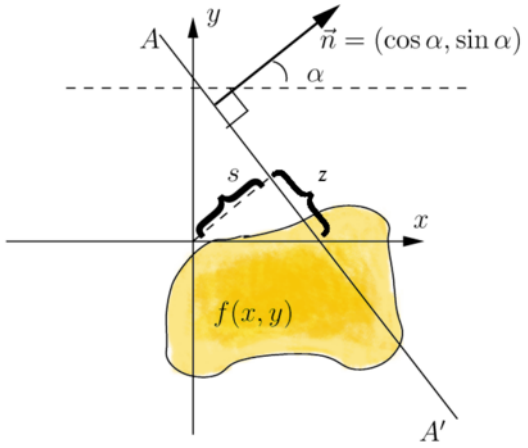


Figure 6: Visualization of the Radon transformation [6]

Let $f : \mathbb{R}^2 \to \mathbb{R}$, its Radon transform defined as:

$$Rf(\alpha, s) = \int_{-\infty}^{\infty} f(x(z), y(z))dz \tag{33}$$

Where the coordinates can be expressed as: $(x(z), y(z)) = ((z\sin(\alpha) + s\cos(\alpha)), (-z\cos(\alpha) + s\sin(\alpha)))$. Here is s the distance from the origin to the line and z the parameterization of the line. This means that for a fixed angle there are infinite amount of lines going through $f(x, y)$, just like an X-ray from one side would be. This happens for all angles which is how a CT-scan works. The Radon transformation is a linear operator because it is an integral. This is the operator that will be used to compare the performances of TV-regularization with the fractional Laplacian. The domain on which the radon transform will work is $\Omega \subset \mathbb{R}^2$ and is a compact set. This means that the Radon transform is also a compact operator with:

$$R : L^2(\Omega) \to L^2\left([0, \pi] \times \left(\mathbb{R}^+ \cup \{0\}\right)\right)$$

It can even be stated that $R(L^2(\Omega)) \subset H^{\frac{1}{2}}$[27]. This means in particular that R is compact and that its inversion is ill-posed. The singular values, which

are the eigenvalues of A^*A, that is self-adjoint and compact, will decay to zero meaning that noise can have a big impact as was seen in chapter 2.2.

Now that the Radon transform has been defined it would also be useful to get back. Therefore the backprojection of the Radon transform will be defined next. Let $h : \mathbb{R}^2 \to \mathbb{R}$, its backprojection is defined as:

$$Bh(x,y) = \frac{1}{\pi}\int_0^\pi h(\alpha, s)d\alpha = \frac{1}{\pi}\int_0^\pi h(\alpha, x\cos(\alpha) + y\sin(\alpha))d\alpha \tag{34}$$

The backprojection is necessary for the implementation of the adjoint of the Radon transform.

3.6 Autoconvolution

Autoconvolution is a non-linear operator with practical applications in stochastics and spectroscopy, which follows from the references of [22]. Using this operator the reconstruction performance of the fractional Laplacian will be shown. To do that first the theoretical backing of this autoconvolution needs to be established. Let $X = Y = L^2[0,1]$, further define

$$D(A) := \left\{ f \in L^2\left[0, \frac{1}{2}\right] \text{ such that } f(t) \geq 0 \ \ \forall t \in [0,1] \right\}.$$

Now $D(A) \subset X$ holds. Now define $A : D(A) \subset X \to Y$ through the operation [3]:

$$A(f(t)) = (f * f)(t) = \int_0^t f(t-\tau)f(\tau)d\tau \tag{35}$$

with $t \in [0,1]$. This is the autoconvolution, which is the convolution of a function with itself. Through the choice of D, the operator is weakly continuous and weakly closed in $L^2[0,1]$ [3]. in Now the Fréchet derivative $A'(f)$ and its adjoint $(A'(f))^*$ needs to be determined. The Fréchet derivative is defined as follows:

Let E, F be normed spaces with $U \subset E$, which is an open subset of E. The functional $\phi : U \to F$ is Fréchet differentiable at $f \in U$ if and only if there exists an ϕ' such that for all $\epsilon > 0$, there exists a $\delta > 0$:

$$\begin{gathered} \|h\| \leq \delta \\ a + h \in U \\ \|\phi(a+h) - \phi(a) - \phi'(h)\| \leq \epsilon \|h\| \end{gathered}$$

The definition can be found in [1] along with other properties of it. With this definition it follows that the Fréchet derivative $A'(f) : X \to X$ is:

$$[A'(f)h](t) = 2\int_0^t f(t-\tau)h(\tau)d\tau \tag{36}$$

Given that $\tau \in [0,1]$. This derivative is a bounded, linear operator. From this the adjoint can be constructed using the inner product definition of the adjoint. So take $f, h, v \in X$ and take the $L^2[0,1]$ inner product:

$$\begin{aligned}\langle A'(f)v, h\rangle &= \int_0^1 2 \int_0^t f(t-\tau)v(\tau)d\tau h(t)dt \\ &= \int_0^1 v(\tau) \int_\tau^1 2f(t-\tau)h(t)dtd\tau \\ &= \langle v, A'(f)^* h\rangle\end{aligned}$$

using this inner product and the substitution $\widetilde{h}(t) = h(1-t)$ it is possible to rewrite the found inner product a little bit more acquiring the following result:

$$\begin{aligned}(A'(f)^* h)(\tau) &= 2 \int_\tau^1 f(t-\tau)h(t)d\tau \\ &= 2 \int_0^{1-\tau} f(t)\widetilde{h}((1-\tau)-t)dt \\ &= 2(f * \widetilde{h})(1-\tau) = 2\widetilde{(f * \widetilde{h})}(t)\end{aligned}$$

It is important to rewrite it in this way, because this also needs to be implemented in Python. Therefore it needs to be written in a way that can be implemented.

4 Computational approach

The computational side of this research has been done in Python. Here, the most important functions will be discussed in the order that the scripts are built up. First the linear operator, the Radon transform, will be discussed and in the second part the non-linear operator which is the auto convolution.

4.1 The Radon transform

The Radon transformation is not a difficult operator to implement. In the code below there is the discrete version of the Radon transform.

Algorithm 1 Discrete Radon Transformation

```
def discrete_radon_transform(image, steps):
  R = np.zeros((steps, len(image)), dtype='float64')
  for s in range(steps):
    rotation = rotate(image, -s*180/steps).astype('float64')
    R[:,s] = sum(rotation)
  R = R/len(image)
  return R
```

What is most important to observe is that instead of rotating the lines over which to integrate, the image gets rotated. This amounts to the same result. Furthermore every line is a row of a matrix, so the line integral is equivalent to the sum of the row. This works because an image is in a computer nothing more than a matrix. The input steps allows anyone to determine how fast the rotation happens. In the results it can be seen that the images have been kept square.

The adjoint of the Radon transformation is a backprojection algorithm. This code is given below:

Algorithm 2 Discrete Radon adjoint transformation

```
def discrete_radon_adjoint_transform(image,steps):
  R = np.zeros((steps, len(image)), dtype='float64')
  Z = image.T
  for i in range(steps):
    rotation = np.zeros((steps, len(image)), dtype='float64')
    for j in range(len(Z)):
      p = Z[i][j]/len(image)
      rotation[:,j] += p
    rotation = rotate(rotation,i*180/steps).astype('float64')
    R = R+rotation
    return R
```

Every pixel is divided equally over a row according to the indexing that is shown in the code. This happens while rotating the matrix. This ensures that the value from the line integral, gets put back under the same angle as the angle under which the line integral was taken. Additionally, when comparing both algorithms it is visible that one turns the matrix clockwise and the other turns it counter clockwise. This is also important to get the angles correct.

These operators can now be implemented into the optimization algorithms. This is not shown here, because the implementation of the optimization will be shown in the non-linear section and they are the same. The operators can be swapped out for one another. This is in order to avoid discussing the same implementation twice. Another part that will not be discussed is the implementation of the TV-regularization. This is because the implementation for TV-regularization used the primal-dual code of [9, 21], in which the operators have been swapped with ours to maintain consistency with the fractional Laplacian method. There are some minor adjustments which are that the operators are swapped out with the ones that are coded here.

4.2 The auto convolution

In the Autoconvolution it is necessary to have a basis. Therefore, the basis of Haar wavelets will be created first.

4.2.1 Computation of the Haar wavelets

In the next two code fragments the starting point of the Haar wavelets can be found. The Haar wavelets are wavelets that are meant to approximate any function in the interval $[0, 1]$. The function Psi is the constant function and is over the entire interval the value 1.

Algorithm 3 Psi

```
def Psi(t):
  if 0<=t<=1:
    phi = 1
  else:
    phi = 0
  return phi
```

The function Phi is the implementation of (30).

Algorithm 4 Phi

```
def Phi(t):
  if 0<=t<0.5:
    psi = 1
  elif 0.5<=t<=1:
    psi = -1
  else:
    psi = 0
  return psi
```

Both these functions take an input t, which simply corresponds to an x-value on the x-axis. In the next function, called Haar, all the possible translations are coded. This function takes three inputs J, K and Length. Here the length corresponds to the interval on which the specific wavelet is being plotted. Furthermore, the J controls the width and the height of the wavelet, while the K shifts it over the x-axis.

Algorithm 5 Haar wavelets

```
def Haar(J,K,length):
  output =[]
  for i in length:
    if J == 0:
      output.append(Phi(i))
    else:
      output.append(Phi(-K+i*2**J) * 2**(J/2))
  return output
```

From the three aforementioned functions all the Haar wavelets can be created and this will be used in the next function. In the function uppertrianglematrix a Haar wavelet matrix will be build. A Haar wavelet matrix is a $N \times N$ matrix with in every row a single Haar wavelet plotted on N points. The input is J which corresponds to a power of 2. Meaning that if an input of 4 is done into this function a matrix of 16×16 will come out. This function always produces 2^J wavelets, contained in a single matrix.

Algorithm 6 Haar matrix

```
def uppertrianglematrix(J):
  Haarmatrix = []
  psi = []
  t = np.linspace(0,1,2**J)
  P = np.linspace(0,J-1,J)
  P = P.tolist()
  P = [int(P) for P in P]
  for z in t:
    psi.append(Psi(z))
  Haar matrix.append(psi)
  for i in P:
    K = (2**i)-1
    K = np.linspace(0,K,K+1)
    K = K.tolist()
    K = [int(K) for K in K]
      for j in K:
        H = Haar(i,j,t)
        Haar matrix.append(H)
    return Haar matrix
```

As mentioned in the theoretical backing of the Haar wavelets the K is dependent on the J, which can also be seen in the code fragment. P is the list of all the values that J can be which are all integer values from 0 to J-1. Then for every value that is in P a separate linspace of integers of K is made. Meaning that for a higher value of J there are more integers that K can be. This is logical as the wavelet gets compressed for a higher J meaning that more horizontal shifts are possible until the domain of $[0, 1]$ is filled. Using this Haar matrix it is possible to calculate the Haar coefficients for any function on the interval $[0, 1]$ or in other words go from space domain to Haar domain.

4.2.2 Setup of the autoconvolution

To go from the space domain to the Haar domain a couple of preliminary settings need to in place. These settings can be found in the next code fragment.

```
n = 8
N = 2**n
a = 0
b = 1
x = np.linspace(a,b,N)
function = eval("parabool(x)")
H = uppertrianglematrix(n)
```

These settings are adjustable, but necessary to get everything to run. The n is corresponding to the big J that was mentioned in 4.1. It indicates which

power of 2 is going to be used as can be seen with N. The a and b are the borders of the domain on which the functions will work. These are usually set to $a = 0$ and $b = 1$ which is the same as the interval that was seen before $[0, 1]$. This can be adjusted but then the basis needs to be taken into account. The x is the discretization of the interval that is chosen with a and b into N points. Function is the discretization of any function that gets put into the eval function. This can be a self coded Python function as well as an already existing function in packages like Numpy. Lastly H is the Haar matrix. This needs to be in the memory, because it will be used multiple times and recalculating it every time would be computationally more expensive. With this setup done it is possible to get the Haar coefficients. This is done in the next code fragment.

Algorithm 7 Haar wavelet transform

```
def xlambda(function):
  xlambda = []
  function = np.array(function)
  for i in range(len(function)):
    xlambda.append(np.dot(function,H[i])/N)
    return xlambda
```

This function calculates the Haar coefficients corresponding to the function that gets put in. The autoconvolution and some other calculations will be done with the Haar coefficients. This transformation can be mathematically defined as follows:

$$\varphi_\lambda = \begin{cases} \psi & \lambda = 1 \\ \varphi_{j,k} & \lambda = (j,k) \end{cases} \tag{37}$$

for $\forall \lambda \in \Lambda$ as defined in (30). Now $x_\lambda = \langle f, \varphi_\lambda \rangle$, where x_λ is the Haar coefficients corresponding to the Haar wavelet φ_λ [3].

The end result should again be in the space domain. Therefor the next function is the transform back from Haar coefficients to the function or the coefficients of the space domain.

Algorithm 8 inverse Haar transformation

```
def reconstruction(yeta):
  recon = np.zeros(N)
  for i in range(len(yeta)):
        Y= [x*yeta[i] for x in H[i]]
    recon = recon +Y
    return recon
```

This can also be mathematically written in the form:

$$f = \sum_{\lambda \in \Lambda} x_\lambda \varphi_\lambda \tag{38}$$

In preparation of the autoconvolution and other necessary functions the next two matrices are made. Starting with the function "kappa". This function creates a $N \times N \times N$ matrix consisting of all the convolutions of all of the Haar wavelets.

Algorithm 9 Convolution of Haar wavelets

```
def kappa():
  K = []
  for i in range(len(H[0])):
    P = []
    for j in range(len(H[0])):
      T = np.convolve(np.array(H[i]),np.array(H[j]),mode='full')/N
      T = T[:N]
      P.append(T)
    K.append(P)
  return K
K = kappa()
```

This function "kappa" will allow for a quick calculations of the autoconvolution of any function. The elements however of the matrix that this function creates are still in the space domain. Therefore, the next function transforms the matrix from space domain into Haar wavelet domain. This function is "kappaeta".

Algorithm 10 Kappaeta

```
def kappaeta():
  Keta = []
  for i in range(len(H[0])):
    P = []
    for j in range(len(H[0])):
      T = np.matmul(K[i],H[j])
      P.append(T)
    Keta.append(P)
  return Keta
keta = kappaeta()
```

With these functions in place the autoconvolution can take place, hence the following function. The autoconvolution takes three inputs. The a and b

input correspond to the borders of the domain on which the autoconvolution takes place. The x input is the discretization of the function that needs to be autoconvolved. As can be seen from the script the Haar coefficients are calculated for the input function and together with the Haar convolution matrix K, the autoconvolution is performed.

Algorithm 11 Discrete autoconvolution

```
def discreteautoconvolve(a,b,x):
    auto =np.zeros(N)
    xlam =xlambda(x)
    for i in range(len(xlam)):
        for j in range(len(xlam)):
            T = K[i][j]
            p = xlam[i]*xlam[j]*T*(b-a)
            auto = auto+p

    return auto
```

To understand why the autoconvolution can be done this way the mathematical explanation will be given now. This will also show why the functions "kappa" and "kappaeta" are nice to have in place. Starting of with the operator, there was:

$$A(f(t)) = \int_0^t f(t-\tau)f(\tau)d\tau$$

This operator is going to be reformulated to make its computer implementation practical. In other words it is going to be rewritten such that it would work on sequences in l_2. This is done in the following steps:

$$\begin{aligned} A(f(t)) &= A\left(\sum_{\lambda\in\Lambda} x_\lambda \varphi_\lambda(t)\right) \\ &= \int_0^t \sum_{\lambda\in\Lambda} x_\lambda \varphi_\lambda(t-\tau) \sum_{\mu\in\Lambda} x_\mu \varphi_\mu(\tau) d\tau \\ &= \sum_{\lambda\in\Lambda}\sum_{\mu\in\Lambda} x_\lambda x_\mu \left[\varphi_\lambda * \varphi_\mu\right](t) \end{aligned}$$

It is easy to see that the function "kappa" creates the matrix that comes forth from the convolution of $\varphi_\lambda * \varphi_\mu$. What should be noted additionally is that the output of the operator is in the space domain and not in the Haar domain. To get it into the Haardomain the following calculation is done:

$$y_\eta = \langle A(f), \varphi_\eta \rangle = \sum_{\lambda\in\Lambda}\sum_{\mu\in\Lambda} x_\lambda x_\mu \langle \varphi_\lambda * \varphi_\mu, \varphi_\eta \rangle$$

The function "kappaeta" creates the matrix that corresponds to the inner-product $\langle \varphi_\lambda * \varphi_\mu, \varphi_\eta \rangle$. This should hold for $\forall \eta \in \Lambda$. In the function "kappaeta" this calculation is done $\forall \eta$ and therefore produces a matrix of size $N \times N \times N$. This has as benefit that when calculating $\mathbf{x}^T K \mathbf{x}$, where K is the aformentioned matrix, the result will be all the Haarcoefficients or Haarvector of the autoconvolution.

4.2.3 The adjoint of the Autoconvolution

To use any of the optimization methods with a basis in least squares the adjoint of the operator is a necessary part. In this case the derivative of the adjoint is necessary, because of the non-linear operator. This can however be computed quite simply, but some shifting of the vectors is in order. Therefore there is the following code fragment.

Algorithm 12 zconstruct

```
def zconstruct(h):
    perturbationvector = []
    psi = []
    t = np.linspace(0,1,N)
    P = np.linspace(0,n-1,n)
    P = P.tolist()
    P = [int(P) for P in P]
    for z in t:
        psi.append(Psi(z))
    z1 = np.dot(h,psi)/N
    perturbationvector.append(z1)

    for i in P:
        K = (2**i)-1
        K = np.linspace(0,K,K+1)
        K = K[::-1]
        K = K.tolist()
        K = [int(K) for K in K]
        for j in K:
            H = Haar(i,j,t)
            H = np.array(H)
            ztilt = np.dot(h,-1*H)/N
            perturbationvector.append(ztilt)
    return perturbationvector
```

This "zconstruct" is very similar to the creation of the Haar matrix that was seen earlier. The main difference is that the Haar wavelets are being ordered differently. Furthermore instead of creating the entire matrix the inner product

Algorithm 13 Adjoint of Fréchet derivative

```
def FPadjoint(z, x):
    product = []
    xlam = xlambda(x)
    for i in range(len(xlam)):
        T = np.matmul(keta[i],z)/N
        product.append(T)
    output = 2*np.matmul(xlam,product)
    adjoint = reconstruction(output)
    adjoint = adjoint[::-1]
    return adjoint
```

is immediately taken with the input vector h. This h corresponds to the perturbationvector that can be seen in the Fréchet derivative. The output of this function is again a vector, which is used in the next code fragment to create the adjoint.

In "FPadjoint", the adjoint is created based on the input function. The input function gets put in through the variable x, while the perturbationvector corresponds to the input z. With some simple multiplications, the adjoint is created. One important detail is at the end of the function, namely once it is put back into spacedomain, the adjoint needs to be flipped. It could also have been done in the Haar domain but then the vector needed to be inversed by making all the shifts back that were made in "zconstruct". This is more difficult then flipping the entire vector backwards in spacedomain, therefore it is being done in spacedomain.

These function are again made such that the Fréchet derivative can be represented in l_2. Therefore, the mathematical explanation also follows from the derivation of the Fréchet derivative and its adjoint. Starting of with what is already known, the adjoint of the Fréchet derivative is as follows:

$$(A'(f)^*h)(\tau) = 2\widetilde{(f * \widetilde{h})}(t)$$

where $\widetilde{h}(t) = h(1-t)$. This would be the same as writting it as follows:

$$(A'(f)^*h)(\tau) = 2\sum_{\lambda\in\Lambda}\sum_{\mu\in\Lambda}\widetilde{x_\lambda \widetilde{g_\mu}\varphi_\lambda * \varphi_\mu}$$

where $\widetilde{g_\mu} = \left\langle \widetilde{h}, \varphi_\mu \right\rangle$. This can then again be adapted such that the output is in the Haardomain creating:

$$\langle A'(f)^*h, \varphi_\eta\rangle = 2\sum_{\lambda\in\Lambda}\sum_{\mu\in\Lambda} x_\lambda \widetilde{g_\mu} < \varphi_\lambda * \varphi_\mu, \varphi_\eta >$$

This can be shortened into:

$$\langle A'(f)^* h, \varphi_\eta \rangle = 2(\widetilde{\mathbf{x}^T K_\eta \mathbf{g}})$$

The last question remaining is how the shifting impacts the Haardomain. Looking at $\widetilde{g_\mu} = \left\langle \widetilde{h}, \varphi_\mu \right\rangle$, this can be divided into two cases:

$$\widetilde{g_1} = \left\langle \widetilde{h}, \psi \right\rangle$$
$$\widetilde{g_{j,k}} = \left\langle \widetilde{h}, \varphi_{j,k} \right\rangle$$

Starting with the first case and remembering that $\widetilde{h}(t) = h(1-t)$ on the domain $[0,1]$. This means that

$$\widetilde{g_1} = \left\langle \widetilde{h}, \psi \right\rangle = \left\langle h, \widetilde{\psi} \right\rangle = \langle h, \psi \rangle = g_1$$

This can be done because the inner product between two sequence from which one is backwards is the same as when the other vector would have been backwards. This however does not matter for ψ, because it is the constants function with a value of 1. Flipping that backwards is the same as not flipping it. Going to the second case the first step of which function is put backwards stays the same, in an inner product it does not matter.

$$\widetilde{g_{j,k}} = \left\langle \widetilde{h}, \varphi_{j,k} \right\rangle = \langle h, \widetilde{\varphi_{j,k}} \rangle$$

However for the other Haar wavelets it does matter that they are flipped, so the same steps will not hold. However when looking at $\widetilde{\varphi_{j,k}}$ it can be seen that it is the same as another wavelet that is multiplied with -1, but with another index. The result of this observations leads to the following index: $(j,k)' = (j, 2^j - 1 - k)$. With this index, it creates the equality: $\widetilde{\varphi_{j,k}} = -\varphi_{(j,k)'}$. With this result the next steps hold:

$$\widetilde{g_{j,k}} = \left\langle \widetilde{h}, \varphi_{j,k} \right\rangle = \langle h, \widetilde{\varphi_{j,k}} \rangle = \left\langle h, -\varphi_{(j,k)'} \right\rangle = -g_{(j,k)'}$$

The function "zconstruct" creates this indexing and makes the vector that corresponds to $\widetilde{\mathbf{g}}$. With this as input the function FPadjoint calculates: $y = 2(\widetilde{\mathbf{x}^T K \widetilde{\mathbf{g}}})$ and puts it back into the spacedomain.

4.2.4 Optimization algorithms

Everything has now been setup so that FISTA and the fractional Laplacian can work. First the standard FISTA will be shown:

Algorithm 14 FISTA

```
def AutoFista(B, alpha, iterations, tol):
  x = function*1
  L = 2
  y=x
  t=1
  i=0
  teller = 0
  for i in range(iterations):
    x0=x
    t0=t
    h = zconstruct((B - discreteautoconvolve(a,b,x)))
    y = y + FPadjoint(h,x)/L
    y = xlambda(y)
    x= shrinkage_operator(y, alpha)
    x = reconstruction(x)
    y = reconstruction(y)
    if np.max(np.abs(x-x0))<tol:
      print(teller)
      return x

    t=(1.+np.sqrt(1.+4.*t**2))/2.
    y=x+((t0-1.)/t)*(x-x0)
    teller +=1
  return  x
```

In this implementation of FISTA their are a couple of details that need to be discussed. The solution to a least squares problem would look like something along the lines of $A^*(Ax-b)$. This is exactly what happens in this function, it is only a little bit hidden. What happens is that $Ax-b$ is the perturbationvector used in "zconstruct" to calculate the adjoint. So the adjoint is based on the input of $Ax-b$. Furthermore there is the "shrinkage_operator". This is the function that does the soft thresholding as described in how FISTA works. The implementation is seen in the next code fragment. The soft threshold is done on the Haar coefficients as can be seen from the functions used in the FISTA implementation.

Algorithm 15 Soft thresholding

```
def shrinkage_operator(x,alpha):
  Shr = np.sign(x)*np.maximum(np.abs(x)-alpha, 0.)
  return Shr
```

The Shrinkage operator flattens the values that are within the domain of

$(-\alpha, \alpha)$. The implementation of the fractional Laplacian is almost the same. The main difference is that instead of the soft threshold there is a function that enforces the fractional Laplacian.

Algorithm 16 Fractional Laplacian regularized inverse autoconvolution

```
def AutoLaplace(B, s, alpha, iterations, tol):
  x = function*1
  L = 2
  y=x
  t=1
  i=0
  teller = 0
  for i in range(iterations):
    x0=x
    t0=t
    h = zconstruct((B - discreteautoconvolve(a,b,x)))
    y = y + FPadjoint(h,x)/L
    x= Laplacerecover1d(y, s, alpha)

        if np.max(np.abs(x-x0))<tol:
      print(teller)
      return x

    t=(1.+np.sqrt(1.+4.*t**2))/2.
    y=x+((t0-1.)/t)*(x-x0)
    teller +=1
  return  x
```

To perform the fractional Laplacian fast the input gets transformed to the fourier domain as can be seen below. Furthermore the input is shifted such that the zero frequency is in the middle. The input s is the fraction that can be put in causing the derivative to be a fractional derivative in Fourier space. This is possible because in Fourier space the derivative of a function is equal to the following: $\hat{f}'(t) = iw\hat{f}(t)$. This makes calculating the fractional Laplacian very easy since it equates to: $(-\triangle)^{\frac{s}{2}}\hat{f}(t) = |w|^s\hat{f}(t)$. The output then gets shifted back and inverse Fourier transformed. A small residual for very small frequencies is removed from the output that gets there due to computational inaccuracy. This has to be removed because it messes up the rest of the algorithm.

Algorithm 17 Proximal of fractional Laplacian regularizer

```
def Laplacerecover1d(A, s, alpha):
  V = fft(A)
  V = np.fft.fftshift(V)
  L = np.zeros((len(V)),dtype = 'complex_')
  for j in range(len(V)):
    R = ((j-len(V)/2)**2)**s
    uhat = V[j]/(1+alpha*R)
    L[j] += uhat
  L = np.fft.ifftshift(L)
  output = ifft(L)
  output = output.real
  return output
```

The only thing that now remains is to run the optimization functions.

5 Results

Just as in the computational approach this section will be divided into the results for the linear operator and the non-linear operator. Therefore it starts with a comparison between TV-regularization and the fractional Laplacian regularization on the Radon transform, and in the second part of the results the non-linear operator will be discussed.

5.1 Results for the linear operator

The comparison between the fractional Laplacian and TV-regularization will be done on the Shepp-Logan phantom. This phantom looks like this:

Figure 7: Shepp-Logan phantom

This image is often used for X-rays and computerized tomography. This is the image that both algorithms will try to reconstruct. The Phantom and its Radon transform that are used in the algorithm contain 160×160 pixels. The Radon transform applied to this phantom creates the following sinogram (8):

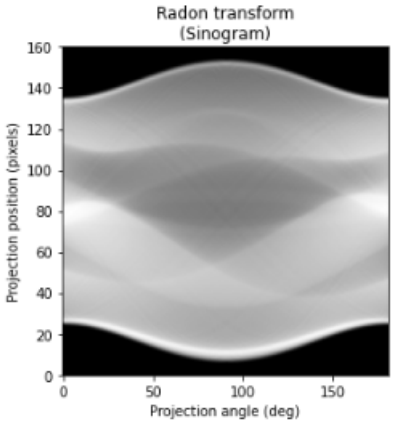
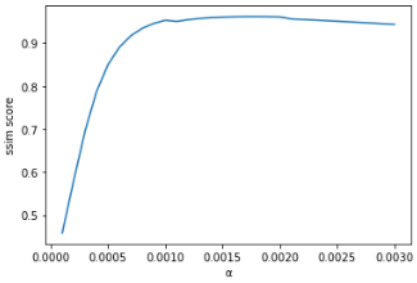
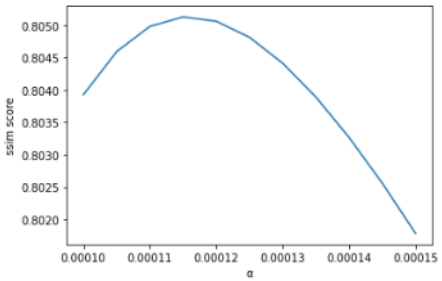


Figure 8: Sinogram of the Shepp-Logan phantom

This sinogram with added noise is the input for both algorithms. The noise that is added, will be based on a Gaussian with as standard deviation 5% of the maximum value of the pixels of the sinogram and a mean of zero. This level of noise will be used on all the examples that are shown. In the fractional Laplacian the fractional index will be set to $s = 0.49$. This will be used throughout the results. How much the regularization term contributes depends on the α. It is therefore necessary to determine a suitable α in advance for the reconstructions. This is visible in the following two diagrams (9).



(a) Reconstruction scores based on SSIM and the value of α for TV-regularization



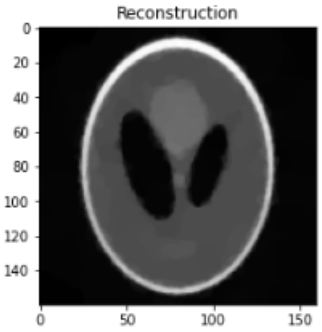
(b) Reconstruction scores based on SSIM and the value of α for the fractional Laplacian regularization

Figure 9: The SSIM scores are plotted against the value of α for both types of regularization

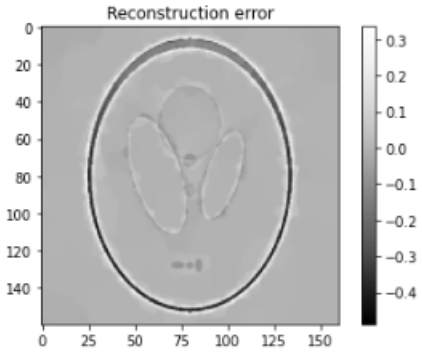
The α that is used is always an approximation however this should be close enough to the optimal value.

To start the comparison between the regularization methods let us first show

the TV-regularization computed using primal-dual [9]. The reconstruction that it made, is accompanied by its reconstruction error:



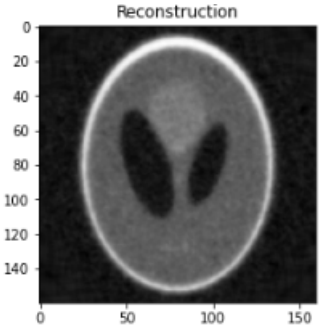
(a) Reconstruction of the Shepp-Logan phantom using TV-regularization



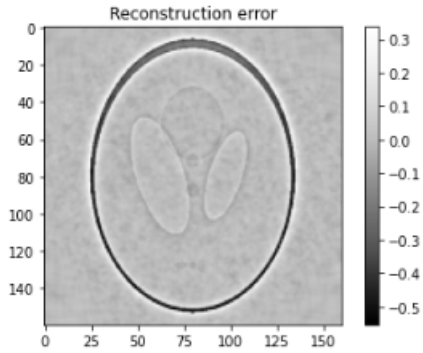
(b) Error plot of the reconstruction in comparison to the original

Figure 10: Results of the reconstruction using TV-regularization. The reconstruction has an SSIM score of 0.913 and a MSE of 0.005, using the best α based on the SSIM scores from figure (9)

The running time of the algorithm to make this reconstruction was 315 seconds rounded to whole seconds. This is equal to 5 minutes and 15 seconds. Now the reconstruction using a fractional Laplacian as regularizer will be shown.



(a) Reconstruction of the Shepp-Logan phantom using a fractional Laplacian as regularizer



(b) Error plot of the reconstruction in comparison to the original

Figure 11: Results of the reconstruction using a fractional Laplacian as regularizer. The reconstruction has an SSIM score of 0.804 and a MSE of 0.008, using the best α based on the SSIM scores from figure (9) and $s = 0.49$

The running time for this algorithm was 91 seconds or 1 minute and 31 seconds. Looking at the scores of both reconstructions it is easily seen that they score quite comparable. Both in the SSIM score and the mean-squared error (MSE) there is a bit of a difference The TV-regularization scores a little bit better on the MSE, but the biggest difference is the SSIM score. It is clear that the TV-regularization has a reconstruction that is a bit better. However, the biggest difference is in the computation time. The optimization based on the fractional Laplacian is more then 3 times as fast as the TV-regularization with not a bad reconstruction either. When looking at the reconstruction it is also possible to see slight differences. The TV-regularization has some staircasing issues while the fractional Laplacian has been smoothed out. From the reconstruction errors it is however clearly visible that both methods are quite good at reconstructing the edges. Another point that needs to be made is that in both cases the reconstruction times seem somewhat long. This is because of the operator and our simple implementation of it. If it was strictly a denoising problem without an operator A, both algorithms would be much faster. However this is not the case so for every iteration both algorithms need to run the operator and its adjoint, causing the longer reconstruction times. In both cases the reconstruction algorithm ran for a 1000 iterations and each iteration only had one instance of the operator and its adjoint. It is not the case that one of the algorithms has to use the operator more than the other algorithm.

To get a better grasp on the performance of the optimization method two more reconstruction have been done. The first one is on a paraboloid. This is a continuous function resulting in the following reconstructions:

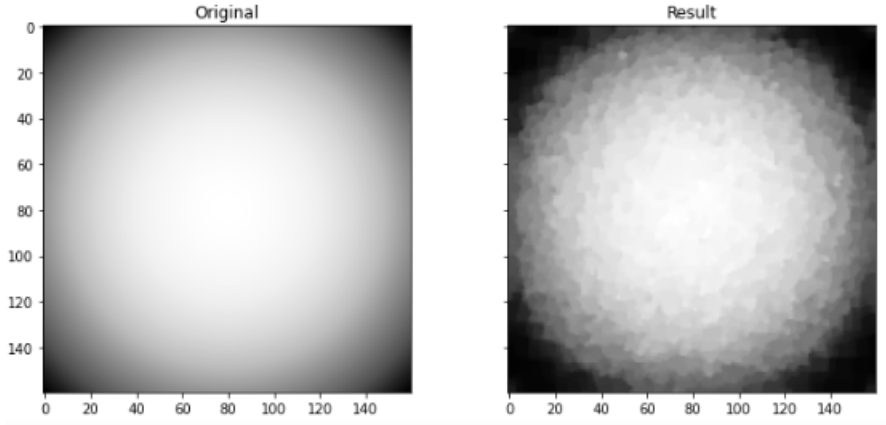


Figure 12: Reconstruction of a paraboloid using TV-regularization

On the left there is the original image of the paraboloid. On the right there is the reconstruction using TV-regularization. Now this reconstruction is compared to the reconstruction made with the fractional Laplacian as regularizer.

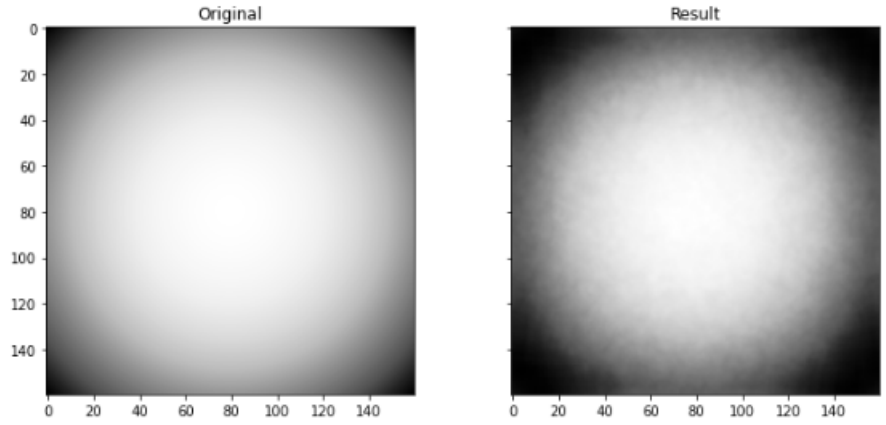


Figure 13: Reconstruction of a paraboloid using a fractional Laplacian as regularizer

What is clearly visible from the comparison is that the fractional Laplacian is much more smooth than the reconstruction from the TV-regularizer. The staircasing effect is quite visible in the reconstruction from the TV-regularizer. The fractional Laplacian outperforms the TV-regularization in this kind of instance by a great margin. Now an instance with a big discontinuity is used. Both algorithm will try to reconstruct a square with value 1, while everything outside of the square has a value of zero. This is what the reconstructions look like:

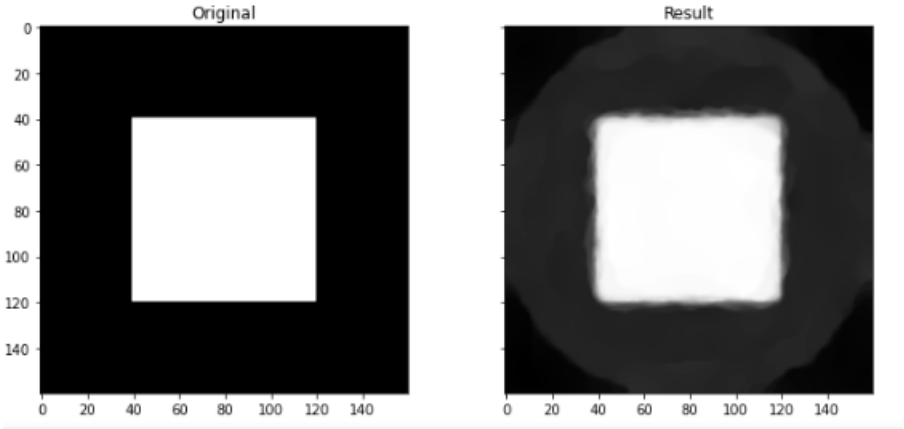


Figure 14: Reconstruction of a square with TV regularization

Now the same square is reconstructed using the fractional Laplacian.

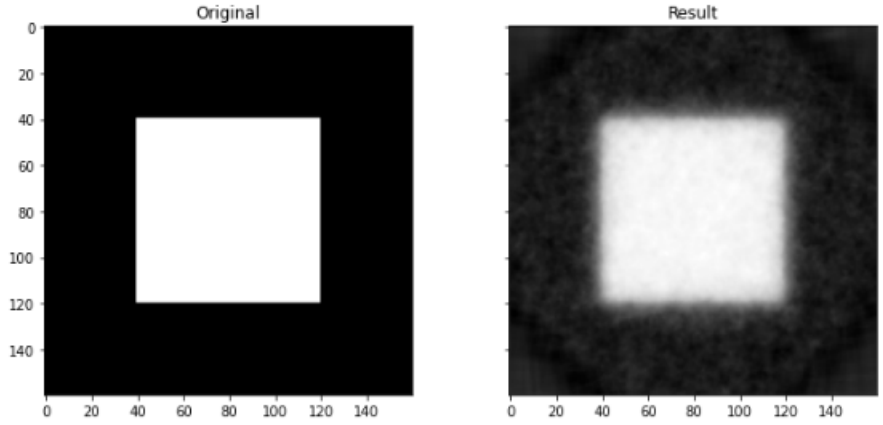
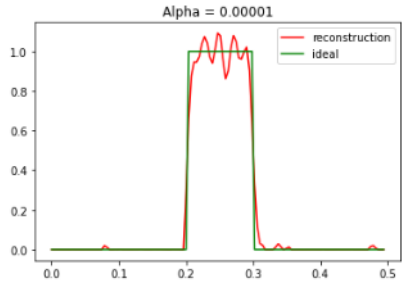


Figure 15: Reconstruction of a square with a fractional Laplacian as regularizer

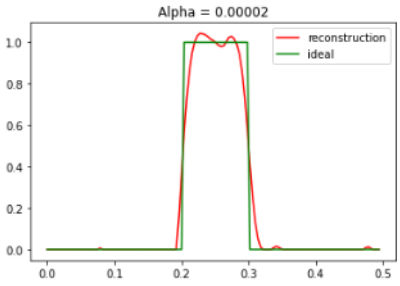
A clear circle can be seen around the square in both cases which is caused by the operator that is currently being used. However in this case the TV-regularization performs better, there is no noise in the separate areas of the picture. The edges seem to be quite distinct in both cases.

5.2 Results for the non-linear operator

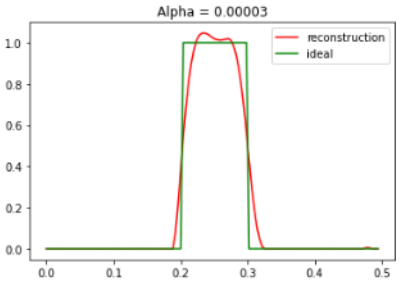
First a suitable α will be determined for the rest of these results. The choice of α is important for it significantly influences the reconstruction. This can be seen in the following figures:



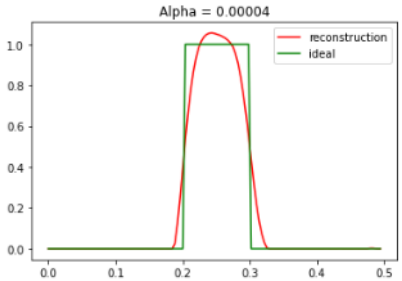
(a) MSE: 0.0045 & SSIM: 0.854



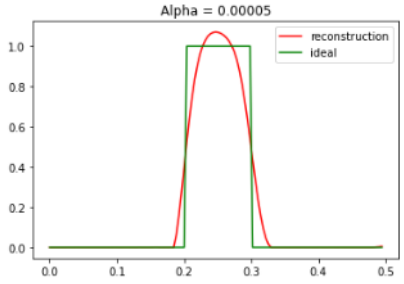
(b) MSE: 0.0077 & SSIM: 0.881



(c) MSE: 0.0097 & SSIM: 0.864



(d) MSE: 0.0113 & SSIM: 0.842

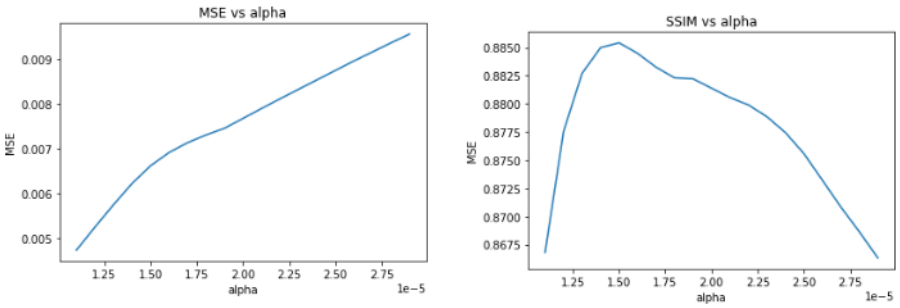


(e) MSE: 0.0128 & SSIM: 0.821

Figure 16

The choice of α has a lot of influence on how the reconstructions look like. With very small increases the reconstructions become more and more smooth. These approximations have also been made for the upcoming comparisons for both FISTA and the fractional Laplacian. The α changes per function that

is being reconstructed. Furthermore, to give more insight into this the Mean-Squared error (MSE) and the SSIM scores have also been calculated. The MSE keeps increasing with increasing α. The SSIM score however seems to have a optimum somewhere around $\alpha = 2 \cdot 10^{-5}$. Therefore both the SSIM and the MSE have been plotted for increasing α in the following plots:



(a) Progression of the Mean-Squared Error

(b) Progression of the SSIM score

Figure 17

From these calculations, the conclusion follows that $\alpha = 1,5 \cdot 10^{-5}$ should be approximately the optimal α. The same approximation is also done for the other functions that are shown, however figures like (17) will not be shown every time. The reconstruction with this α looks as follows:

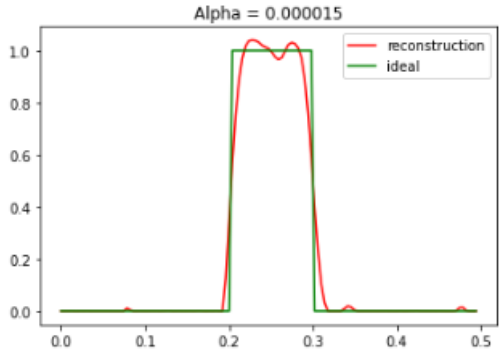
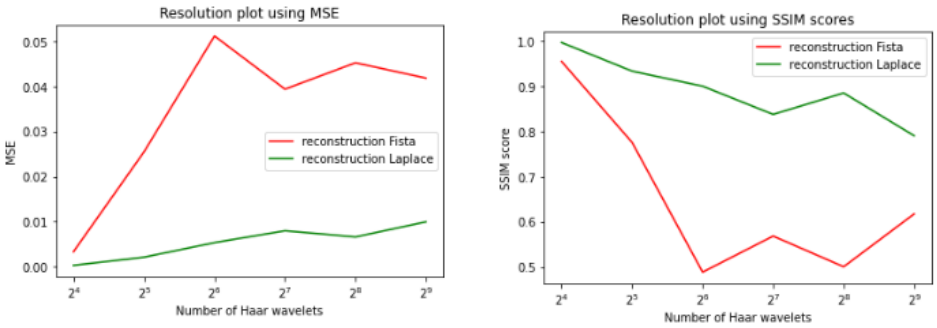


Figure 18: Optimal reconstruction

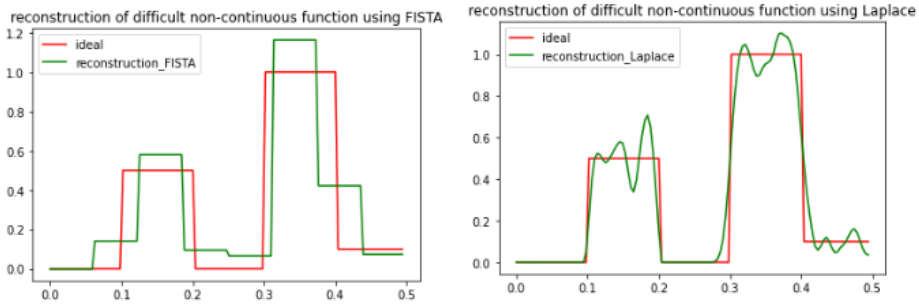
Continuing with the next step, the reconstruction algorithm will be compared to an already existing algorithm called FISTA. They are going to be compared on the reconstruction depending on the amount of Haar wavelets used. The amount of Haar wavelets is important because that corresponds to the amount of plotting points that are present. Starting at sixteen Haar wavelets and with steps of factor two the results look like this:



(a) (b)

Figure 19: Resolution of MSE and SSIM scores with increasing amount of Haar wavelets

As can be seen from the graphs, for every power of two, FISTA is getting outperformed. The MSE is lower while the SSIM is higher for the new optimization method. This is however still on a relatively simple function. Therefore the following comparisons will be on a continuous function and a more difficult discontinuous function. This difficulty will lie in the fact that the function will have more jumps. The continuous function that will be used is $f(x) = 0.5\sin(4\pi x)$. This function makes a complete cycle in the domain $[0; 0, 5]$. Below the comparison can be seen between the more difficult non-continuous function and their respective reconstructions.

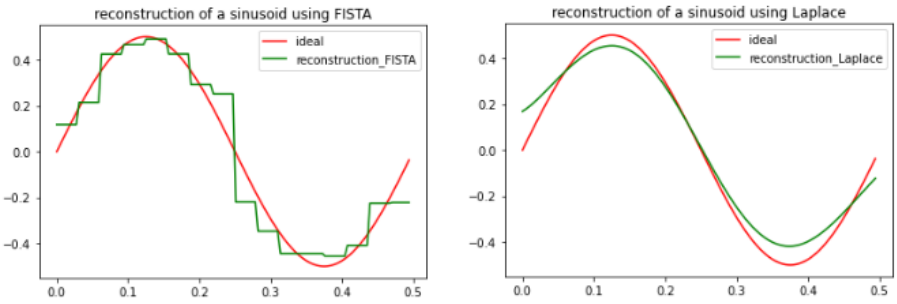


(a) MSE: 0.065 & SSIM: 0.376 (b) MSE: 0.011 & SSIM: 0.701

Figure 20: Comparison of the reconstructions on a more difficult non-continuous function

As can be seen from the comparison both algorithms have a varying degree of success with the reconstruction of this more difficult discontinuous function. The reconstruction of FISTA has really nice square waves as the original signal does, but fails to locate the exact jumps up and down. Even more so, it does not

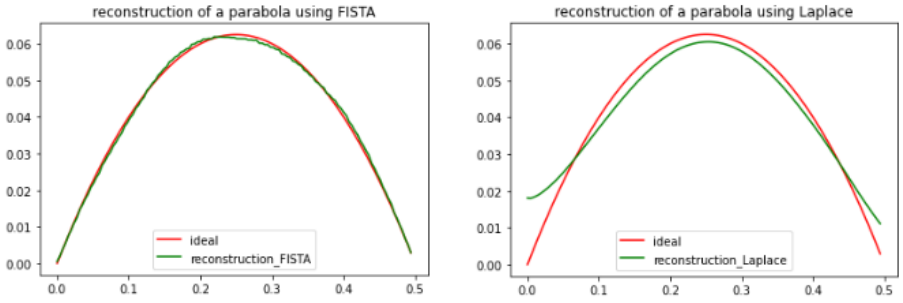
locate the correct height of the horizontal parts of the non-continuous function. In comparison, the minimization based on a fractional Laplacian is far more accurate with the timing of the jumps, but they are not completely vertical as FISTA does have. Nonetheless it does locate the heights of the jumps very well, but is incapable of making a horizontal line. During the horizontal parts there are some oscillation, which were to be expected. When looking at the MSE and the SSIM scores, it is visible that the fractional Laplacian performs better overall. Now the same comparison is made on the sinusoid.



(a) MSE: 0.022 & SSIM: 0.686 (b) MSE: 0.003 & SSIM: 0.963

Figure 21: Comparison of the reconstructions on a more difficult continuous function

Again, when looking at the MSE and the SSIM scores the fractional Laplacian outperforms FISTA. The reconstruction made by the Laplacian is very smooth while the reconstruction of FISTA is very square like. This is because of how the regularization works in combination with the Haar wavelet basis. It looks as if the new optimization method is outperforming FISTA in every reconstruction. Unfortunately there are some cases in which the new optimization method struggles. In the next example, a parabola is being reconstructed.



(a) MSE: 6.45e-07 & SSIM: 0.9998 (b) MSE: 1.85e-05 & SSIM: 0.991

Figure 22: Comparison of the reconstruction of a parabola

As can be seen from the scores, the reconstruction of FISTA is slightly better. When looking closely at the reconstruction of FISTA it is visible that the reconstruction is not completely smooth, but it is obvious when compared to the reconstruction of based on the fractional Laplacian, that it is closer. The difference however is small.

6 Conclusion

Although the code could be more thoroughly optimized, it is easy to see that there is a big discrepancy in the reconstruction times. This discrepancy would not change if the code is better optimized since both algorithms are running on the same language, libraries and implementation of operators to be inverted. This indicates that the fractional Laplacian operator as a regularizer is faster then the TV-regularization as was expected. The reconstructions however are not the same, based on the SSIM score it can be concluded that the TV-regularization is better. This however does not mean that the fractional Laplacian operator is bad, it still reconstructs at an SSIM score of around 0.8 with relatively sharp edges and some fuzziness throughout its reconstruction and it behaves better for very smooth regions. A general convergence analysis of the fractional Laplacian operator is not given. This is because of the simple nature of the algorithm. Its analysis is almost completely the same as other optimization problems that get solved through forward-backward splitting. The only step that is a bit different is the proximal step, but a very easy expression is already given for that, making the analysis very similar to already existing convergence analysis. Because of its easy proximal the fractional Laplacian can also be used on a non-linear operator quite easily. The results from this were compared to FISTA in Haar basis and it shows that the fractional Laplacian has some good reconstructions even with a non-linear operator. In the case of discontinuous functions and periodic functions the fractional Laplacian reconstructed them significantly better than FISTA did. The one instance that FISTA did better was a non-periodic continuous function and even there the fractional Laplacian did not score much worse than FISTA did. One comment that could be made, is that the initialization with a scaled version of the sought after solution is not realistic for applications. This is a fair comment, but only holds for the non-linear case where it was used to avoid local minima. In the linear case it was initialized from all zeros. However even for the non-linear case it is not too relevant as this was a showcase of the potential of the fractional Laplacian. FISTA had the same advantages, but their respective reconstructions look very different and shows the potential of the fractional Laplacian.

To summarize this all, the fractional Laplacian is a computationally cheap way to get regularization that can find edges. It does not give the best reconstruction quality compared to TV-regularization, however it is certainly not bad. For large problems the fractional Laplacian could offer a solution if peak resolution is not demanded and computation time is an issue. Furthermore, the fractional Laplacian can also be easily used on non-linear inverse problems with a decent reconstruction.

References

[1] Al-Mohy, A. H., Higham, N. J., & Relton, S. D. (2013). Computing the Fréchet Derivative of the Matrix Logarithm and Estimating the Condition Number. SIAM Journal on Scientific Computing, 35(4), C394–C410. https://doi.org/10.1137/120885991

[2] Antil, H., & Bartels, S. (2017). Spectral Approximation of Fractional PDEs in Image Processing and Phase Field Modeling. Computational Methods in Applied Mathematics, 17(4), 661–678. https://doi.org/10.1515/cmam-2017-0039

[3] Anzengruber, S. W., & Ramlau, R. (2009). Morozov's discrepancy principle for Tikhonov-type functionals with nonlinear operators. Inverse Problems, 26(2), 025001. https://doi.org/10.1088/0266-5611/26/2/025001

[4] Ambrosio, L., Fusco, N., & Pallara, D. (2000). Functions of Bounded Variation and Free Discontinuity Problems (Oxford Mathematical Monographs) (1st ed.). Oxford University Press.

[5] Beck, A., & Teboulle, M. (2009). A Fast Iterative Shrinkage-Thresholding Algorithm for Linear Inverse Problems. SIAM Journal on Imaging Sciences, 2(1), 183–202. https://doi.org/10.1137/080716542

[6] Begemotv2718. (2005, October 30). Radon transform. Wikipedia. https://commons.wikimedia.org/wiki/File:Radon_transform.png

[7] Boyd, S. P., Boyd, S. P., Vandenberghe, L., & Cambridge University Press. (2004). Convex Optimization. Cambridge University Press.

[8] Cai, J. F., Dong, B., Osher, S., & Shen, Z. (2012). Image restoration: Total variation, wavelet frames, and beyond. Journal of the American Mathematical Society, 25(4), 1033–1089. https://doi.org/10.1090/s0894-0347-2012-00740-1

[9] Chambolle, A., & Pock, T. (2011). A first-order primal-dual algorithm for convex problems with applications to imaging. Journal of Mathematical Imaging and Vision, 40(1), 120–145. https://doi.org/10.1007/s10851-010-0251-1

[10] Chen, W. (2006, June). A speculative study of 2/3-order fractional Laplacian modeling of turbulence: Some thoughts and conjectures. Chaos: An Interdisciplinary Journal of Nonlinear Science, 16(2), 023126. https://doi.org/10.1063/1.2208452

[11] Clason, C. (2020). Introduction to Functional Analysis (Compact Textbooks in Mathematics) (1st ed. 2020). Birkhäuser.

[12] Clason, C. (2021). Regularization of inverse problems. Lecture notes. https://doi.org/10.48550/arXiv.2001.00617

[13] Dabkowski, M. (2011). Eventual Regularity of the Solutions to the Supercritical Dissipative Quasi-Geostrophic Equation. Geometric and Functional Analysis, 21(1), 1–13. https://doi.org/10.1007/s00039-011-0108-9

[14] Defrise, M., & Gullberg, G. T. (2006). Image reconstruction. Physics in Medicine and Biology, 51(13), R139–R154. https://doi.org/10.1088/0031-9155/51/13/r09

[15] del-Castillo-Negrete, D., Carreras, B. A., & Lynch, V. E. (2004, August). Fractional diffusion in plasma turbulence. Physics of Plasmas, 11(8), 3854–3864. https://doi.org/10.1063/1.1767097

[16] Demoment, G. (1989). Image reconstruction and restoration: overview of common estimation structures and problems. IEEE Transactions on Acoustics, Speech, and Signal Processing, 37(12), 2024–2036. https://doi.org/10.1109/29.45551

[17] Di Nezza, E., Palatucci, G., & Valdinoci, E. (2012). Hitchhiker's guide to the fractional Sobolev spaces. Bulletin Des Sciences Mathématiques, 136(5), 521–573. https://doi.org/10.1016/j.bulsci.2011.12.004

[18] Engl, H. W., Hanke, M., & Neubauer, A. (1996). Regularization of Inverse Problems (Mathematics and Its Applications, 375) (1996th ed.). Springer.

[19] Friedberg, S., Insel, A., & Spence, L. (2018, September 7). Linear Algebra (5th ed.). Pearson.

[20] Ge, J., Everett, M. E., & Weiss, C. J. (2015, May 1). Fractional diffusion analysis of the electromagnetic field in fractured media — Part 2: 3D approach. GEOPHYSICS, 80(3), E175–E185. https://doi.org/10.1190/geo2014-0333.1

[21] GitHub - pierrepaleo/ChambollePock: A Python implementation of the Chambolle-Pock algorithm for image processing applications examples. (n.d.). GitHub. Retrieved September 10, 2022, from https://github.com/pierrepaleo/ChambollePock

[22] Gorenflo, R., & Hofmann, B. (1994). On autoconvolution and regularization. Inverse Problems, 10(2), 353–373. https://doi.org/10.1088/0266-5611/10/2/011

[23] Gull, S., & Daniell, G. (1978). Image reconstruction from incomplete and noisy data. Nature, 272(5655), 686–690. https://doi.org/10.1038/272686a0

[24] Kirsch, A. (2022). An Introduction to the Mathematical Theory of Inverse Problems (Applied Mathematical Sciences) (3rd ed. 2021). Springer.

[25] Korolev, Y., & Latz, J. (2021). inverse Problems. Lecture notes, Michaelmas term 2020, University of Cambridge. http://www.damtp.cam.ac.uk/research/cia/inverse-problems-michaelmas-2020

[26] Parikh, N., & Boyd, S. (2013). Proximal Algorithms (Foundations and Trends(r) in Optimization). Now Publishers Inc.

[27] Natterer, F. (2001). The Mathematics of Computerized Tomography (Classics in Applied Mathematics, Series Number 32). SIAM: Society for Industrial and Applied Mathematics.

[28] Romberg., J., Egerstedt, M. B., & Davenport, M. A. (2021). Convex Optimization. ECE 6270: Convex Optimization Spring 2021. https://mdav.ece.gatech.edu/ece-6270-spring2021/notes/10-subgradients.pdf

[29] Roncal, L., & Stinga, P. R. (2015). Fractional Laplacian on the torus. Commun. Contemp. Math, 18(3), 26.

[30] Rudin, L. I., Osher, S., & Fatemi, E. (1992, November). Nonlinear total variation based noise removal algorithms. Physica D: Nonlinear Phenomena, 60(1–4), 259–268. https://doi.org/10.1016/0167-2789(92)90242-f

[31] Snieder, R., & Trampert, J. (1999). Inverse Problems in Geophysics. CISM International Centre for Mechanical Sciences, 119–190. https://doi.org/10.1007/978-3-7091-2486-4_3

[32] Stankovic, R. S., & Falkowski, B. J. (2003). The Haar wavelet transform: its status and achievements. Computers and Electrical Engineering, 29(1), 25–44. https://www.elsevier.com/locate/compeleceng

[33] Zhang, L., Zhang, Z., & Guan, C. (2021). Accelerating privacy-preserving momentum federated learning for industrial cyber-physical systems. Complex &Amp; Intelligent Systems, 7(6), 3289–3301. https://doi.org/10.1007/s40747-021-00519-2

[34] Zhou Wang, & Bovik, A. (2002, March). A universal image quality index. IEEE Signal Processing Letters, 9(3), 81–84. https://doi.org/10.1109/97.995823