A Study in Total Variation Denoising and its Validity in a Recently Proposed Algorithm

by Vittorio Bisin

A Thesis in Applied Mathematics submitted in partial fulfillment of the requirements for the degree of Master of Science in Mathematics.

Courant Institute of Mathematical Sciences New York University

September 2017

Acknowledgments

I would like to thank Professor Fernandez-Granda and my father.

Abstract

The scope of this Master's thesis is to reproduce a variation of a denoising model proposed by Chen, Yu, and Pock (2015) [**5**], in a simplified context, and to verify whether the model provides a justification for adopting "total variation denoising," a method widely used in the area of signal processing.

I begin by reviewing the denoising problem and common approaches to its solution. I then introduce the recent Chen, Yu, and Pock (2015) algorithm, my version of their model, and my results. Indeed, I confirm that total variation denoising and its finite difference operator, are justified in this context - they represent the outcome of the denoising optimization algorithm used in this thesis.

Contents

1 Introduction

Consider the problem of uncovering an original signal from a noisy, perturbed one:

$$\text{observed signal} = \text{original signal} + \text{noise}$$

More specifically, if the vector $y \in \mathbb{R}^n$ denotes the observed signal and $z \in \mathbb{R}^n$ is the noise, the original signal $x \in \mathbb{R}^n$ needs to be recovered from $y = x + z$. To recover the original signal we need prior information regarding its structure and the distribution of the noise; Figure 1 shows a piece-wise constant function with added Gaussian noise, as an example. This problem, called 'denoising', has become very active in the area of signal processing in the past three decades [**11**].

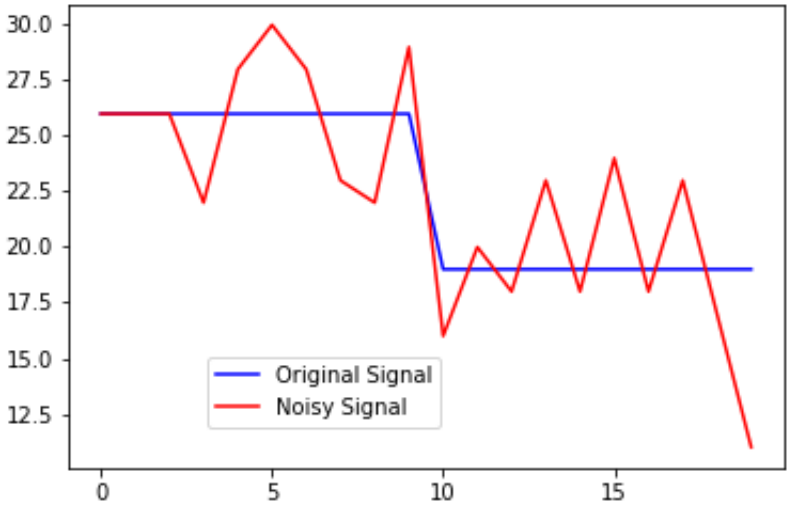


Figure 1: Original Denoising Problem (with noise $\sim$ N(0,25))

The objective of this thesis is twofold: first of all I reproduce, in a simplified context, a denoising model introduced by Chen, Yu, and Pock (2015) [**5**]; secondly, I verify whether

a commonly used denoising procedure, called "total variation denoising," is justified in this context, in that it represents the outcome of (my simplified adaptation of) the optimization algorithm for denoising used by Chen, Yu, and Pock (2015) [**5**].

1.1 The Representation of the Signal x

To better specify and understand the denoising exercise I perform in this thesis, some background is useful.[1] First of all, assumptions are needed on the original signal x. In particular, I follow the common standard in this literature by adopting a representation of x as a linear combination of basic elements, called "atoms." Let $\phi_i \in \mathbb{R}^n$ denote the i-th atom and let $\Phi = \{\phi_1, \phi_2, ..., \phi_m\}$ be called a "dictionary." Then:

$$x = \sum_{i=1}^{m} c_i \phi_i = \Phi c$$

where $c = \{c_1, c_2, ..., c_m\}$ is the representation (or transform) of x in the dictionary.

Furthermore, in this thesis, I restrict the analysis to an environment where the (linearly independent) atoms are $m >> n$. In this case the linear model is "overcomplete;" that is, there is a (possibly large) dimension of vectors $c \in \mathbb{R}^m$ which satisfy the linear representation $x = \sum_{i=1}^{m} c_i \phi_i$.

Finally, again following the common standard in this literature, I assume that x has a "sparse" linear representation, in that it can be represented by a (small) subset of the atoms:

$$x = \sum_{i=1}^{l} c_i \phi_i \quad \text{with} \quad l << m.$$

A sparse linear model can be quite efficient over its higher dimensional representation for

[1] I follow [**7**] here.

processing signals of interest that tend to be highly structured. Sparse linear models are better able to exploit this structure, process, and decompose the signal into an accurate dictionary - they are used in areas of compression, denoising, and inverse problems [**7**]. However, in the context of an overcomplete sparse model, selecting a sparse representation out of all the possible representations of x becomes a fundamental computational problem associated to denoising,.[2] In principle, a sparse representation can be obtained as a solution of the problem

$$\min_{c \in \mathbb{R}^m} \; || \, c \, ||_0$$

$$\text{subject to } x = \Phi c,$$

but this problem is generally computationally intractable.

1.2 Some Denoising Algorithms

Several methods have been exploited in the literature to denoise a signal; see [**7**]. I here discuss those which are most relevant to illustrate the exercise I perform in this thesis. I concentrate on environments where the signal can be represented by an overcomplete sparse linear model, though the sparse representation is not necessarily known, as discussed in the previous section.

[2] In the simplest case in which, instead, $\Phi = \{\phi_1, \phi_2, ..., \phi_m\}$ constitutes an orthonormal basis of dictionary elements, decomposing a signal in $\mathbb{R}^n$ is a straight inverse problem: letting $\Phi^T = \Phi^{-1} = \Psi = [\psi_1 \;\; \psi_2 \;\; ... \;\; \psi_m]$, so that

$$x = \Phi\Psi x = \Phi\Phi^{-1}x = \sum_{i=1}^{m} (\psi_i \cdot x)\phi_i,$$

implies

$$c_i = \psi_i \cdot x.$$

Useful examples of orthonormal basis include the discrete Fourier transform (DFT) and the short-time Fourier transform, discussed in detail in [**7**].

1.2.1 Thresholding

The conceptually simplest way to denoise a sparse signal with an overcomplete dictionary is via thresholding. The idea behind thresholding is to exploit the difference in the underlying structure between signal and noise: the signal is a superposition of dictionary atoms whereas the noise is not. A consequence of this underlying incoherence between dictionary atoms and noise is that the signal will be sparse under an appropriately defined transformation [**7**].

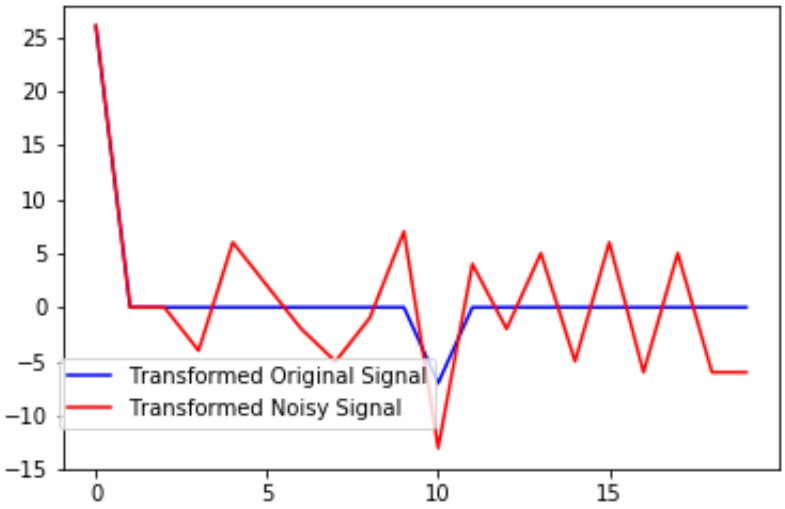


Figure 2: Signals Mapped using the Finite Difference Basis

Assume first the case in which a sparse representation of the signal x,

$$x = Bc,$$

for a basis B of $\mathbb{R}^n$ is known. Under appropriate assumptions about the noise z, this representation is equivalent to mapping the signal x to a domain where the signal is

sparse and the noise is not; that is, such that $c = B^{-1}x$ is sparse, but $B^{-1}z$ is not. A simple and effective denoising algorithm, in this case, called "hard thresholding" involves discarding all coefficients $d = B^{-1}y$ below a certain threshold, η:

$$H_\eta(d_i) = \begin{cases} 0 & \text{if } \mid d_i \mid \leq \eta \\ d_i & \text{otherwise} \end{cases}$$

The denoised signal $\hat{x}$ is obtained as:

$$\hat{x} = B\hat{c},$$

where

$$\hat{c} = H_\eta(d) = H_\eta(B^{-1}y) = H_\eta(B^{-1}x + B^{-1}z) = H_\eta(c + B^{-1}z).$$

As an example, in Figure 2 I apply the "finite difference" transformation[3], via the basis

$$B = \begin{bmatrix} 1 & -1 & 0 & \dots & 0 \\ 0 & 1 & -1 & \dots & 0 \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ 0 & 0 & 0 & \dots & -1 \\ 0 & 0 & 0 & \dots & 1 \end{bmatrix},$$

to the signal in Figure 1[4]. I then obtain a representation of the noise which is significantly more dense than the representation of the original signal. The procedure, as it can be seen form Figure 3 (with $\eta = 15$), mainly eliminates the dense noise: in fact hard

[3]A version of this transformation, via convolution, is adopted often and repeatedly in what follows; see Appendix 4.1.

[4]this is under the structural assumption that the original signal is piecewise constant and hence has a sparse gradient - or equivalently a sparse finite difference

thresholding does a good job of removing small noise when the original piece-wise signal is constant, but is unable to identify the noise when the signal is non-constant (i.e. the 10th entry).

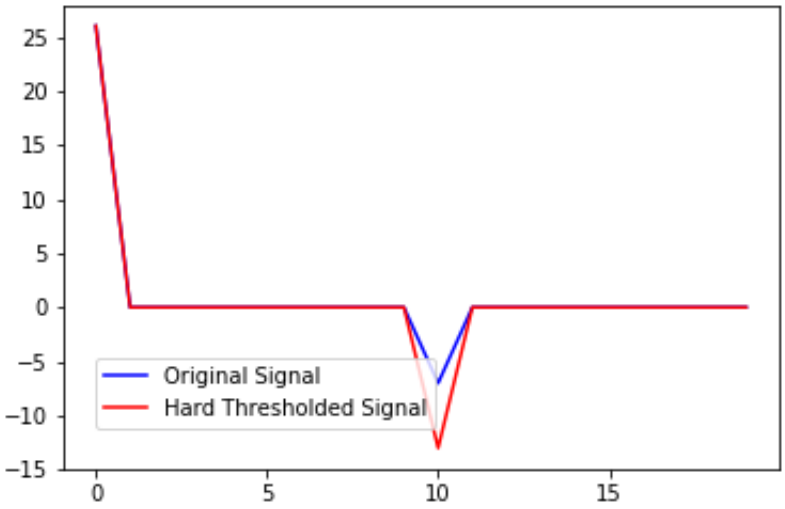


Figure 3: Hard Thresholding (with $\eta = 15$)

1.2.2 Basis Pursuit Denoising

The hard thresholding denoising procedure I just illustrated assumes a sparse representation for the signal. But, as I noted already in Section 1.1, it is generally the case that a sparse representation of the signal is not known and hence it has to be uncovered while denoising. In this case, a different useful procedure operates by means of a regularized linear regression, exploiting the (not necessarily sparse) representation $x = \Phi c$. A specific regularized linear regression, denoted "basis pursuit" [**7**] denoising, is obtained as follows:

$$\hat{x} = \Phi\hat{c}, \text{ where } \hat{c} = \underset{c\in\mathbb{R}^m}{\operatorname{argmin}} \mid\mid y - \Phi c \mid\mid_2^2 + \lambda \mid\mid c \mid\mid_1 .$$

In this regression, the l_1 norm in the regularization term and is used to promote sparsity (the higher the λ the more the regularizer will promote sparse solutions).[5] As we can see in Figure 4 this is by the geometry of the l_1 norm (in 2D a diamond) vs. the l_2 norm (in 2D a circle). The solution set, consisting of the overlap between error and constraint, will more likely be a corner solution in the l_1 norm case. In higher dimensions corner solutions result in a higher number of zero entries (i.e. a sparse solution). On the other hand, in the l_2 case, Figure 4 shows that the red ellipse and blue area are less likely to overlap on the y-axis (where $x = 0$), lowering the probability of a high dimensional sparse solution [**10**].

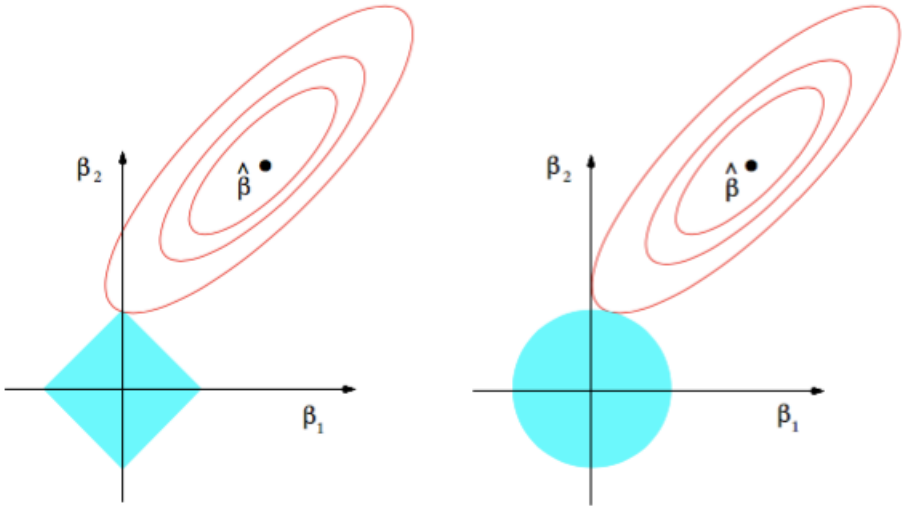


Figure 4: The overlap between the red ellipses (error function) and blue areas (constraint regions) are the solution set. Left is the l_1 2D norm ball and right is the l_2 2D norm ball. Figure from [**10**]

[5]In the limit for $\lambda \to \infty$, $\hat{c}$ converges to 0; conversely, in the limit for $\lambda \to 0$, $\hat{c}$ converges to the least squares regression coefficient.

1.2.3 Total Variation Denoising

Consider instead the case in which $\Phi^T x$ is sparse, meaning that x has a correlation with a small subset of the atoms in the dictionary. Once again, knowing precisely this correlation's structure would allow an adaptation of the thresholding procedure introduced in Section 1.2.1. But, in the more general and interesting case where this correlation needs to be uncovered while denoising, a procedure similar to basis pursuit can be easily adopted. Indeed, we can still force a sparse solution by adding a regularizer in the l_1 norm [**7**]:

$$\hat{x} = \underset{x \in \mathbb{R}^m}{\operatorname{argmin}} \mid\mid y - x \mid\mid_2^2 + \lambda \mid\mid \Phi^T x \mid\mid_1$$

A common variation of this procedure, however, adopts a different regularising operator, though very similar to the above l_1 norm. This operator, the "finite difference" operator, induces what is called "total variation" denoising [**15**, **4**].[6] The idea behind total variation denoising is to smooth the noisy sum of gradients so that it becomes small and well-approximates the original signal's; as illustrated by the example in Figure 1, in fact, the sum of the absolute values of the noisy gradient is very large, whereas the sum in the original gradient is small. Total variation is defined as follows:

$$TV(x) = \mid\mid \nabla_x x \mid\mid_1$$

Formally, then, "total variation denoising" for a signal x is defined to be:

[6] The concept of total variation denoising was first pioneered by Rudin, Osher, and Fatemi in their well-known 1992 paper[**15**]. They used a total variation regularizer as above to denoise images, solved using a second order partial differential equation (the Euler-Lagrange Equation), where the constraints are determined by the noise [**11**]. As before, I refer to Appendix 4.1 for a more detailed discussion of the finite difference operator.

$$\hat{x} = \underset{x \in \mathbb{R}^m}{\operatorname{argmin}} \mid\mid y - x \mid\mid_2^2 + \lambda TV(x)$$

Using the total variation norm as a regularizer is a powerful tool to smooth the gradient of the predicted signal, while of course keeping the squared l_2 norm between the original and predicted signal for data fidelity.[7]

1.2.4 Chen, Yu, and Pock (2015)

In their 2015 paper Chen, Yu, and Pock [**5**] propose a denoising algorithm which has both high computational efficiency and high restoration quality. This algorithm builds on the literature discussed in the previous sections, while also containing several innovations. The core components of the algorithm can be be illustrated in steps.[8]

The noisy signal y is mapped to a domain where it is possibly sparse while the noise is not, as in the thresholding example introduced in Section 1.2.1. More specifically, Chen, Yu, and Pock (2015) posit a general transformation, a convolution $w * y$, whose kernel w is to be learned optimally (see below).[9]

Denoising is obtained by applying a real valued function f to $w * y$. The function f is called the 'influence function' and is parametrized by a Gaussian 'radial basis function'

[7]More generally, the total variation of a (2-dimensional) image, x, is the l_1 norm of its x_1 and x_2 axis components:

$$TV(x) = \mid\mid \nabla_{x_1} x \mid\mid_1 + \mid\mid \nabla_{x_2} x \mid\mid_1$$

The finite difference operator is commonly used for image denoising because images can be well approximated by piecewise constant functions (i.e. having sparse gradients). For image denoising total variation is especially useful in dealing with edges, an especially difficult problem in this area. However, the model in Rudin, Osher, Fatemi (1992) [**15**] has two main disadvantages: i) when edges are not prominent in the image, the model yields a 'staircase effect' (the creation of artificial boundaries or edges throughout the image) and ii) the term in the total variation norm is not differentiable, which may cause some added difficulty in solving for the denoised image. [**11**]

[8]I present their model in the context of denoising a 1-dimensional signal, as my thesis restricts to signals, rather than for a 2-dimensional image as in the original paper [**5**].

[9]I discuss convolutions more in detail in Appendix 4.1.

g,[10] which acts point-wise on the convoluted signal $w * y$ and depends monotonically only on the data point's distance from a pre-determined center:

$$f(w * y) = g(w * y) \cdot \alpha = \sum_{j=1}^{M} \alpha_j \exp(-\frac{|| \, w * y - \mu_j \, ||^2}{2\sigma^2}),$$

where α_j is the weight for the j-th component of the basis function, $|| \cdot ||$ denotes the Euclidean norm, $\mu_j \in \mathbb{R}^n$ is the center for the j-th basis function, $\sigma \in \mathbb{R}$ is the standard deviation of each basis Gaussian function.[11] The influence function is a model-free estimator, i.e. it is only dependent on the input data to determine its local properties and distribution.

The parameters of the kernel w and of the influence function α are chosen optimally via a training procedure to minimize, over multiple samples $s = 1, 2, ..., S$, the sum of squared residuals (SSR) of the denoised signal with respect to the original (noiseless) signal. The parameters obtained through this procedure can then be used to denoise a new out-of-sample signal y.[12]

[10] See Appendix 4.2 for details regarding Radial Basis Functions (RBF's)

[11] The authors use the Gaussian radial basis function as well as the Triangular basis function: $\phi_{triangular} = \phi(w * y) =$

$$\begin{cases} 1\text{-} \frac{|w*y-\mu|}{\sigma} & \text{if } | \, w * y - \mu \, | \leq \sigma \\ 0 & \text{if } | \, w * y - \mu \, | > \sigma \end{cases}$$

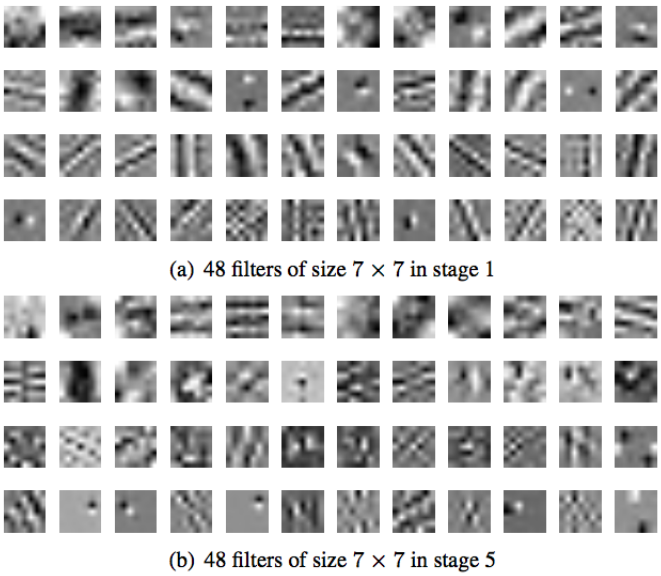
[12] In fact, the actual algorithm Chen, Yu, and Pock (2015) implement is a more complicated one where the training procedure iterates on a nonlinear reaction diffusion process. More in detail, the iterative procedure produces at each step $t = 1, 2, ..., T$ a noisy signal $u_t \in \mathbb{R}^N$ for given u_{t-1}. It is initialized by setting $u_0 = y$, the noisy input signal. Also, denoising is obtained via multiple kernels and influence functions $(w_{i,t}, f_{i,t})$, for $i = 1, 2, ...N_w$. The signal u_t then satisfies at each step t the following nonlinear diffusion equation formulated as a discrete partial differential equation (PDE):

$$\frac{u_t - u_{t-1}}{\Delta t} = -\sum_{i=1}^{N_w} w_{i,t}^T * f_{i,t}(w_{i,t} * u_{t-1}) - \lambda_t(u_{t-1} - u_0),$$

where $w_{i,t}$ represents the i-th kernel at stage t, $f_{i,t}$ the i-th influence function at state t, $\alpha_i^t \in \mathbb{R}^M$ the M-dimensional vector of weights associated to the components of the influence function $f_{i,t}$, and $\lambda \in \mathbb{R}^N$ a reaction term parameter.

Finally, the parameters λ^t, α_i^t, w_i^t are chosen at each stage t to minimize the cost function over

Figure 5: On the left is the observed image (Gaussian noise with σ=25) and on the right is the denoised result with 27dB. Image from [**5**]



(a) 48 filters of size 7 × 7 in stage 1

(b) 48 filters of size 7 × 7 in stage 5

Figure 6: A sample of the learnt filters. Image from [**5**]

The results of their paper are quite good, they have a PSNR (dB) for the 68 images to be about 31 and 28 for Gaussian noise with σ=15 and σ=25, respectively. Figure 5 shows

multiple samples $s = 1, 2, ..., S$:

$$L(\lambda_t, \alpha_{i,t}, w_{i,t}) = \sum_{s=1}^{S} l(u_t^s, x^s) = \frac{1}{2} \sum_{s=1}^{S} || \, u_t^s - x^s \, ||;$$

where u_t^s and x^s denote, respectively, the noisy signal from the reaction-diffusion process at state t and the noiseless signal, in sample s.

an example of their original image and denoising result with 24 learned kernels (each of size 5x5). Their influence functions also seem to make intuitive sense, adaptively shifting between imaging smoothing (forward diffusion) and sharpening (backward diffusion). Furthermore, their learnt kernels (see Figure 6) also seem to represent particular aspects of the original images - with visible first, second, and higher order derivatives - useful for structure (e.g. edge) detection. They conclude that the primary force in denoising the original signal is the training of the influence functions with the original data.

2 The Denoising Exercise

As already noted, in this thesis I aim to reproduce a variation of the denoising model introduced by Chen, Yu, and Pock paper [**5**]. Furthermore, I attempt to verify whether 'total variation denoising' is justified in this context.

More precisely, the algorithm in Chen, Yu, and Pock learns several linear kernels $w_{i,t}$, for $i = 1, 2, ..., N_w$ and iterates over T periods. I restrict the exercise in this thesis to just one such filter and simplify the algorithm by running it over one single stage (i.e., $N_w = T = 1$ in Chen, Yu, and Pock's notation). If in my exercise the single kernel w resulting from the optimization algorithm were to be the finite difference of the signal, it would justify decomposing a piecewise constant signal by taking its finite difference (see Figure 2) and indeed it would justify "total variation denoising."[13]

2.1 Algorithm

Following Chen, Yu, and Pock (2015) the algorithm I implement in this thesis can be decomposed into the following operations: i) convolve the observed piecewise constant signal with the kernel, w; ii) input the convolved signal into the influence function, a Gaussian RBF[14]; iii) return the output of the influence function into the original basis by multiplying the denoised vector by a "cumulative matrix," C;[15] iv) train the model

[13]Furthermore, because images tend to have sparse gradients and therefore can be thought of as 2-dimensional piecewise constant signals, this result would also strengthen the case for total variation image denoising.

[14]This is motivated by the fact that in their paper the Gaussian RBF provides better results than the Triangular RBF

[15]Specifically, the matrix I use is

$$C = \begin{bmatrix} 1 & 0 & \dots & \dots & 0 \\ 1 & 1 & 0 & \dots & 0 \\ \dots & \dots & \dots & \dots & \dots \\ 1 & 1 & 1 & 1 & 1 \end{bmatrix}$$

by minimizing the sum of the squared errors of the denoised signal with the original (noiseless) signal x over many samples $s = 1, 2, ..., S$; and finally v) test the trained model on out-of-sample noisy signals.

Formally, the algorithm is represented by the following:

$$\min_{\alpha,w} \frac{1}{2} \sum_{s=1}^{S} || \ x^s - C \cdot (g(w * y^s) \cdot \alpha) \ ||_2^2$$

where $x^s \in \mathbb{R}^N$ is the noiseless signal and $y^s \in \mathbb{R}^N$ the observed, noisy signal, in sample s; $C \in \mathbb{R}^{NxN}$ is the cumulative matrix; $g(w * y^s) \in \mathbb{R}^{NxM}$ the Gaussian radial basis functions for the s-th sample; w the learned linear kernel; and $\alpha \in \mathbb{R}^M$ the vector of RBF weights.

2.2 Procedure

The code for the minimization algorithm is written using only the NumPy module in Python, and hence is easily transparent and alterable.

I first create randomly generated piecewise constant signals, which make up the noiseless, original signals in my training set.[16] I then add a small amount of Gaussian noise (with mean 0 and standard deviation 5) to each point in the original signal, to make up the noisy, observed signals in the training set. For signals of length 30 the average SSR per sample of the training set is consistently around 300.

Both the parameters to be determined, α and w, have three initialization options: randomly generated[17], well-initialized[18], and 'nearly well-initialized'[19]. The model re-

In the case where the kernel is the finite difference operator, the convolved signal is the derivative of the original signal and the cumulative matrix maps it back into the original domain

[16] In doing so I constrain how large each 'jump' between values in the original signal must be

[17] Each entry takes value uniformly chosen in $[-5, 5]$.

[18] Where w is the finite difference operator and α is the hard-thresholding operator with $\eta = 15$.

[19] Where w is $[-1.5, 1.5]$ and each value of α is initialized to 1

quires three important parameters to be initialized: 'centers difference,' σ, and jump size. Centers difference regulates the centers of the basis functions as well as their number, σ determines the standard deviation of each one of the Gaussian RBF's, and jump size determines the minimum jump of each original piecewise constant signal.

The centers difference parameter does not greatly affect the results, as long as it is less than 1. If the centers difference parameter is greater than 1 the radial basis function does not associate to each input a unique basis function center, which may prevent the algorithm from efficiently denoising the input signal. The σ parameter on the other hand controls the standard deviation of each Gaussian radial basis function. The main constraint in choosing this parameter is to prevent 'holes' inside the RBF (see top graph of Figure 21 in Appendix 4.2), areas where the RBF does not have support. These holes are symptoms of the RBF not being defined for those values, causing a large generalization error. To prevent this I increase σ to be at least over $\frac{1}{2}$, in such a way the Gaussian functions will overlap. I also tried running the stochastic gradient descent algorithm minimizing over σ, but this led to several holes inside the RBF, which in turn led to a low training set error but a high test error. The last and most important parameter to initialize is the 'jump size' parameter. The jump size parameter sets a lower bound on the change of values (or 'jumps') in the original piecewise constant signal. Since I add a small level of Gaussian noise, this means that the larger the jump size, the more easily distinguishable the signal will be from the noise. If I do not fix a large enough minimal jump size, the function will only marginally be able to denoise the observed signal.

I then initialize the centers of the Gaussian RBF's for the training set, to do so I firstly compute $y^s * w$, $\forall s \in S$ and find the minimum and maximum of this set. I then

construct an interval containing these extrema[20] successively increasing values inside the interval by the predefined parameter, centers difference.[21] In such a way I not only define the values of each one of the centers, μ_j but also fix the number of basis functions M.

The above parameters are then inputed into the mini-batch stochastic gradient descent algorithm (see Appendix 4.3 for more details). For each batch of the stochastic gradient descent iteration I calculate the average gradient (see Appendix 4.4 for the gradient computations), average step size (determined by the Armijo backtracking rule - see Appendix 4.3), and then update α and/or w. The algorithm varies significantly between small and large batches: having a large number of oscillations and noise for the small ones, and converging very slowly for the large ones; to fix this I designed the algorithm to increase the number of batches as the error begins to oscillate. The stochastic gradient descent algorithm outputs the optimal values of α and/or w. I then create another sample of randomly generated piecewise constant functions compromising the test set, and compare the accuracy results between training and testing.

[20]To not influence the Gaussians centered on the boundary of the interval I include a padding by adding values to the interval's ends.

[21]That is, the interval is an arithmetic progression with common difference equal to the 'centers difference' parameter.

3 Results

I report here on the results of the denoising experiments. I first report results on the restricted case where the algorithm is set to optimize only the influence function; that is, where the algorithm chooses α optimally for fixed kernel w. I then report results on the general experiment where the optimization occurs jointly over the influence function and the kernel, α and w. In both cases I distinguish between experiments with a small and a large jump size.

3.1 Optimizing the Influence Function

3.1.1 A Small Jump Size

I set the algorithm to optimize the influence function, parametrized by a Gaussian RBF, by minimizing the α parameter. The kernel w is set to be the finite difference operator. For this experiment I choose a relatively large training set size of $5,000$ examples. The σ value is chosen to avoid holes, $\sigma = 1$; and the centers difference parameter is chosen to be 0.5. I initialize each entry of α to be 1. Furthermore, I fix the jumps of the piecewise function to be at least 25, attempting to clearly differentiate the original signal with the Gaussian Noise, distributed as $N(0, 25)$.

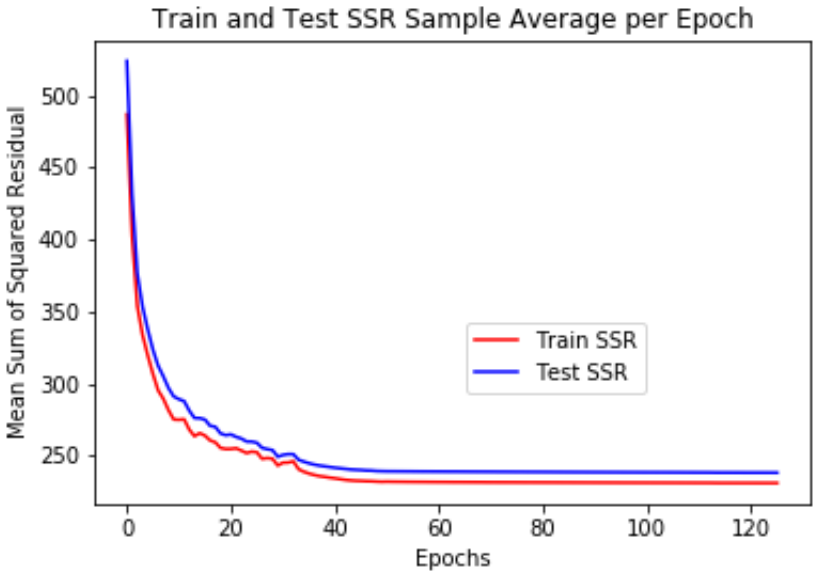


Figure 7: Train and Test SSR Average per Sample for a small jump size

The algorithm converged slowly but relatively smoothly (see Figure 7); it converged to an average SSR per sample of 220, removing about $\frac{1}{3}$ of the original SSR of 310. As hypothesized, because of the large σ value, the algorithm's discrepancy between train and test errors is quite low. Figure 8 shows the algorithm denoising a training sample with a similar accuracy as it denoises a test sample (Figure 9). The predicted signal does not seem to be a piece-wise constant function, and is still affected by the input signal's small noise. Another way to see this is by examining the α values (see Figure 10). For centers near 0 we can see that α does not give exactly 0 weight, but rather smooths them to some value inside $[0, |5|]$. We can also see this by plotting the influence function applied to a linear function[22] with slope 1 (see Figure 11): values near 0 are slightly shrunken (in absolute value), but are not cropped to 0.

[22]The range of this linear signal is the range of the input to the RBF ($w * y^s \ \forall s \in [0, 5000]$).

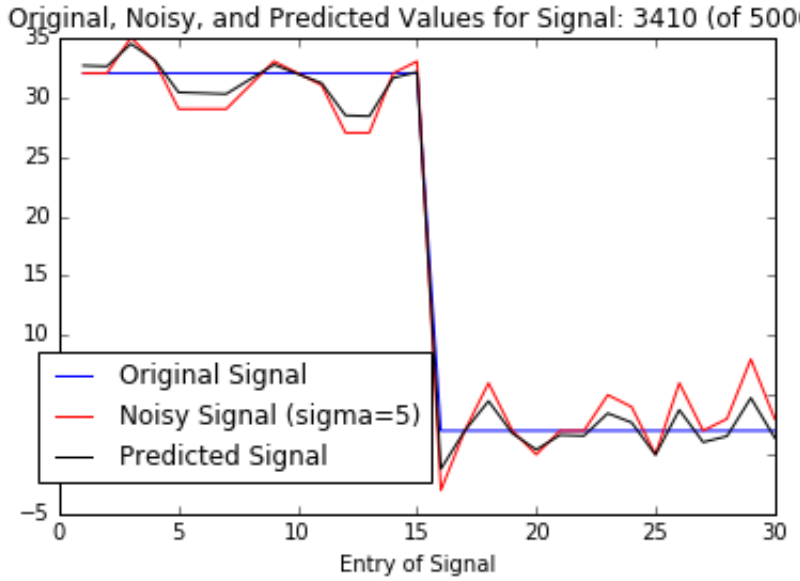


Figure 8: Observed and predicted training signal for a small jump size

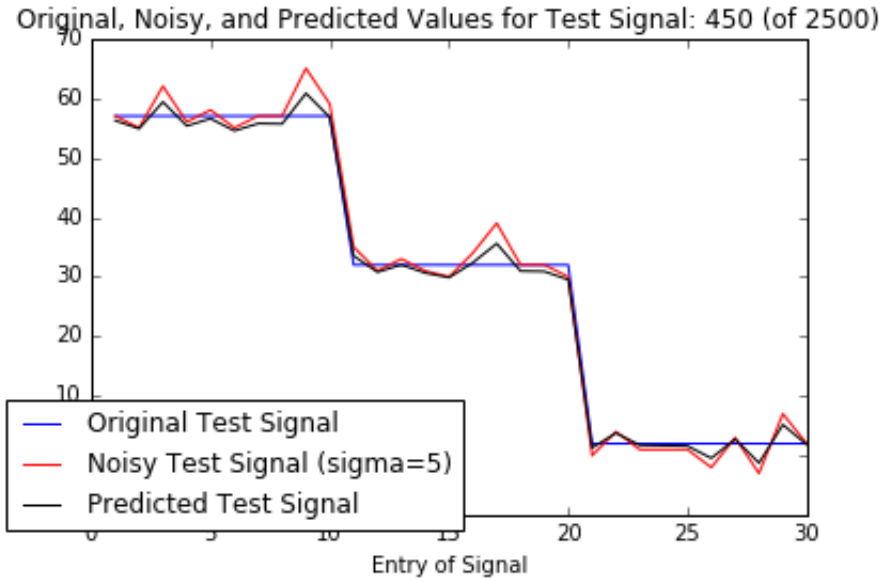


Figure 9: Observed and predicted test signal for a small jump size

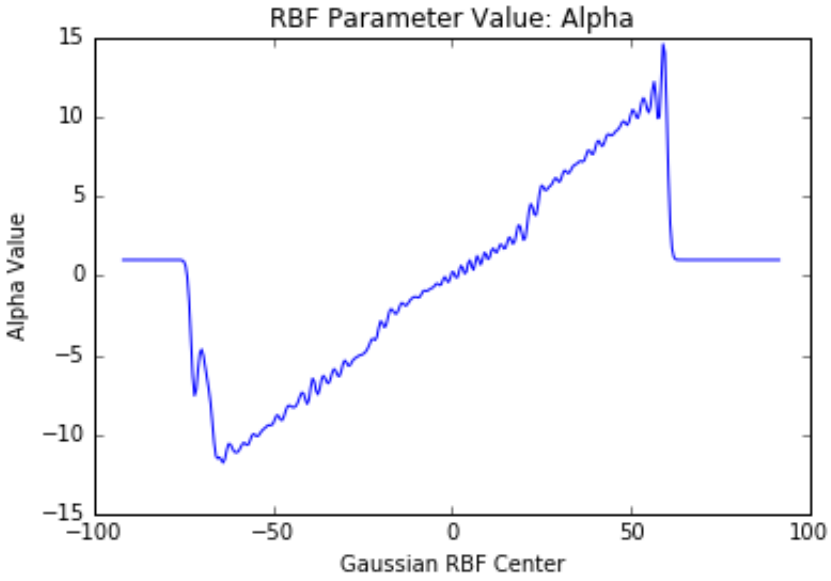


Figure 10: α values for a small jump size

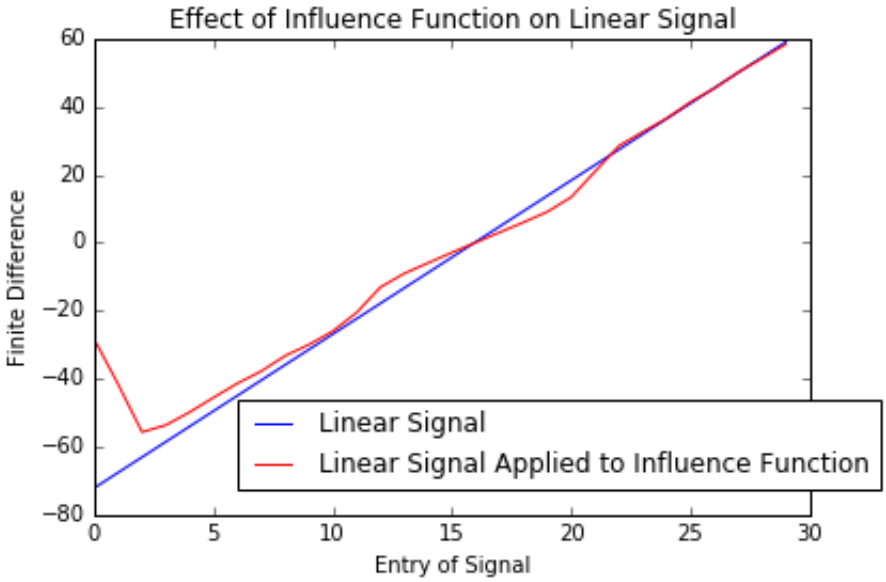


Figure 11: Influence function for a small jump size

3.1.2 A Large Jump Size

I then run a similar experiment but choose a larger jump size, 40. The result is that the algorithm significantly better differentiates between signal and noise, removing more than $\frac{2}{3}$ of the original SSR (see Figure 12). Figure 13 shows the algorithm denoising the signal, with visibly better results than in the case above, and because of the large σ value the algorithm also does well denoising the test set (Figure 14). The algorithm thresholds the small noise significantly better in the 40 jump size than 25 jump size case.

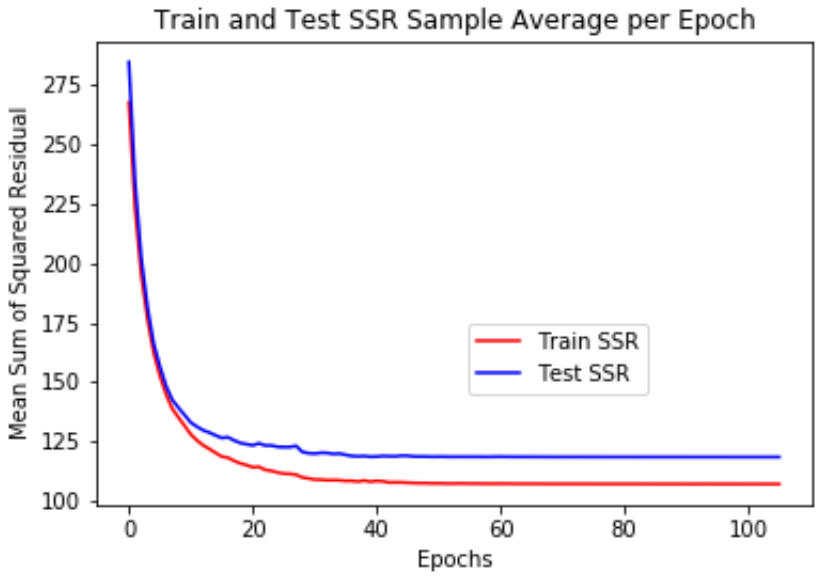


Figure 12: Train and Test SSR Average per Sample for a large jump size

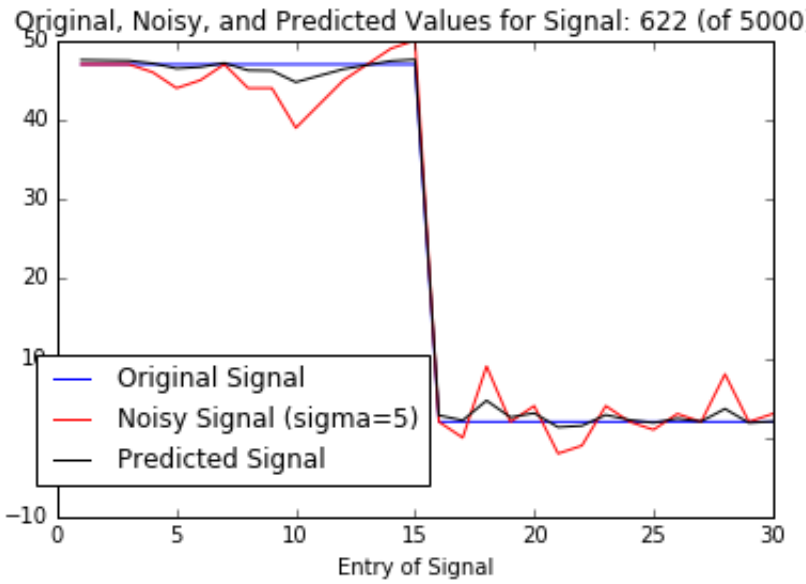


Figure 13: Observed and predicted training signal for a large jump size

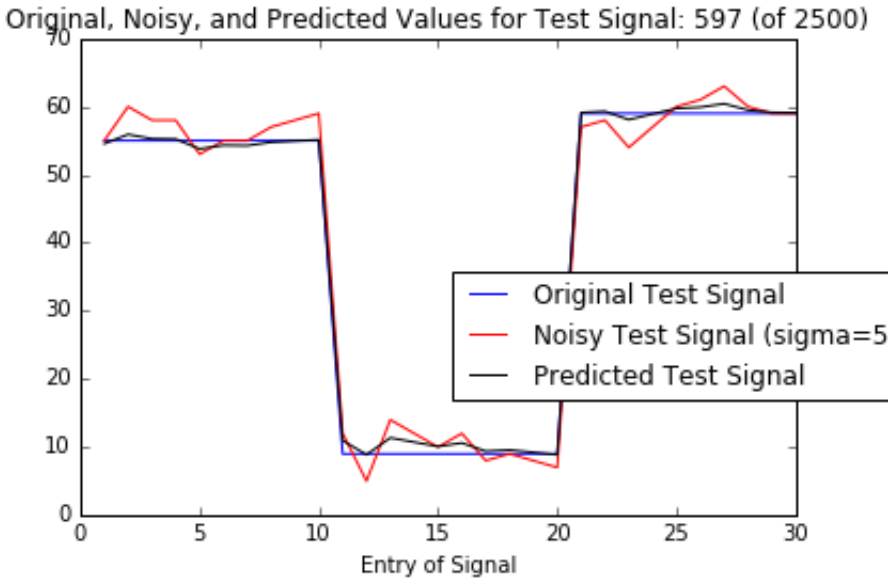


Figure 14: Observed and predicted test signal for a large jump size

The final estimate of α (see Figure 15) is what we expect, with values corresponding to

centers with low-valued signals being thresholded to 0 - or very near 0. On the other hand, centers representing higher valued signals receive much greater weight, expressing the pattern that the finite differences of the function tend to be much larger than those of the noise (concentrated in $[-30, 40]$).[23] This pattern can also clearly be seen in the influence function depicted in Figure 16. The result is that the influence function thresholds the values near 0, while giving greater weight to the large input values.

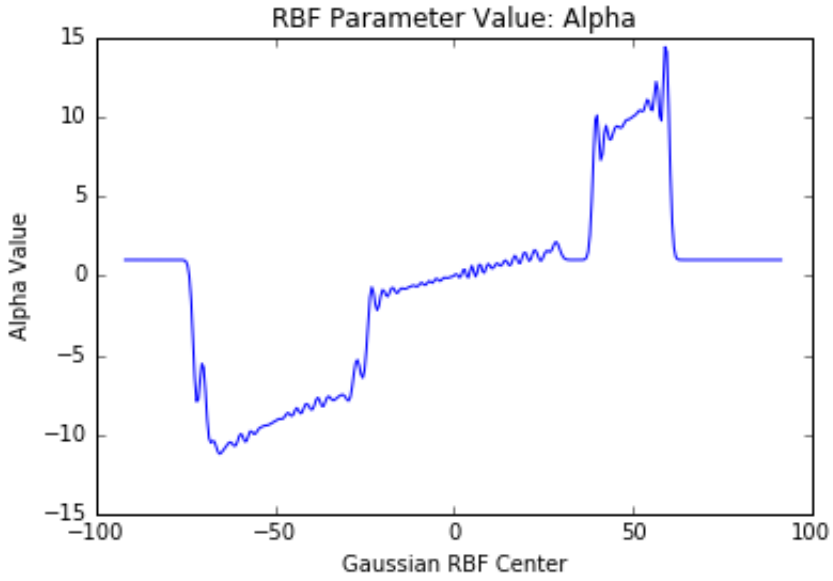


Figure 15: α values for a large jump size

[23]Here the centers of the RBF represent the finite differences of the signals because w is set to be the finite difference operator.

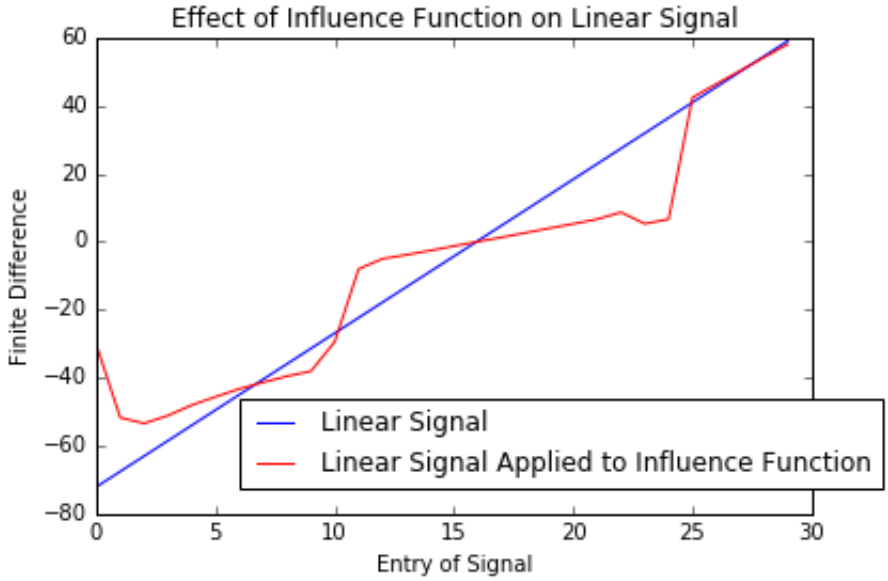


Figure 16: Influence function for a large jump size

3.2 Optimizing the Influence Function and Kernel

In this exercise, I set the algorithm to optimize jointly the influence function and the kernel, that is, both α and w. I choose a training set sample size of $5,000$ and $\sigma = 1$. In jointly minimizing α and w I encounter a computational problem: the noise in the stochastic gradient descent algorithm would greatly increase the value of w, increasing $w * y^s$ - far more than any center I had initialized.[24]. This leads the RBF centers to become 0, causing the gradients to go to 0, in turn causing the function to converge very far from the global minimum. A possible solution to this problem would be to simply increase the dimension of the basis functions - having enough centers so that $w * y^s$ would no longer be too large. However, this would require a very large number of basis functions, causing the algorithm to converge extremely slowly. Instead I opt to adopt

[24]Precisely, in the RBF center, $exp(\frac{-((w*y^s)_i-\mu_j)^2}{2\sigma^2})$, every $w * y_i^s$ would be far greater than every μ_j

the following procedure: I optimize for α and w by first minimizing over α, fixing w to the finite difference operator; once this had converged, I initialize w to $[-1.2, 1.2]$ and optimize over both α and w. In such a way the algorithm is close to the optimal values for both α and w and there is less noise in arriving to the minimum.

3.2.1 A Small Jump Size

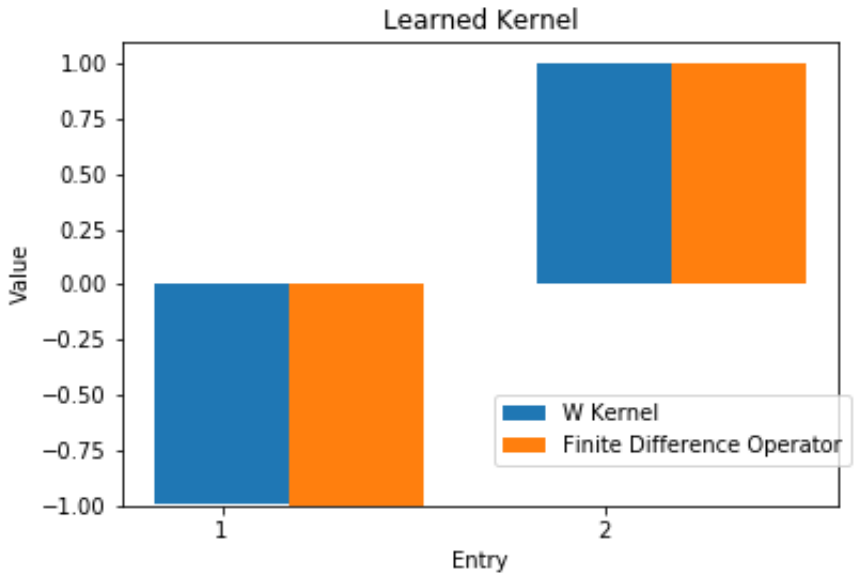


Figure 17: Finite Difference Operator and Learnt Kernel for a small jump size

This solution scheme works quite well when initializing the size of the jumps to 25. As initially predicted w does indeed converge to the finite difference operator. The estimate for w is in $[0.996, -0.998]$ (see Figure 17). Since I initialize α to be the optimal value found in the previous section (minimizing α with w fixed to the finite difference operator), and the optimal value of w is, in fact, the finite difference operator, the SSR converges to the same value as in the previous section

3.2.2 A Large Jump Size

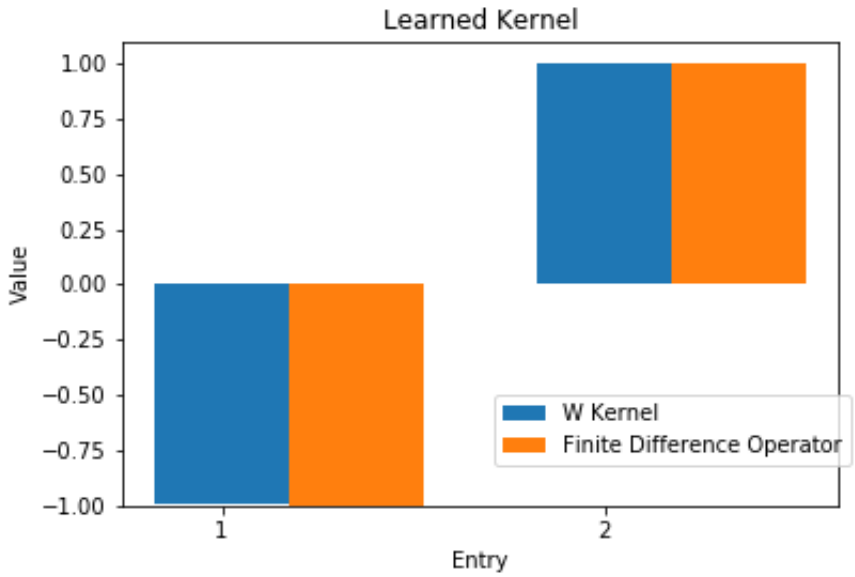


Figure 18: Finite Difference Operator and Learnt Kernel for a large jump size

I initialize the jump size to be 40 and as in the case above, w converges to the finite difference operator (see Figure 18). Furthermore, as in the case above, the SSR converges to the same value as when I optimize only the influence function.

3.3 Discussion and Conclusions

The graph which best explains the operation of the algorithm is the influence function graph for a large jump size (Figure 16). The algorithm first applies a form of hard thresholding to the noisy signal, setting all finite differences below a certain η to 0, thereby removing a large fraction of the noise[25]. The second operation the algorithm undergoes is smoothing certain signal differences which may be more ambiguous - that in certain cases correspond to the real signal in the training set, other times to noise. In such a way, by smoothing (i.e. either trimming or incrementing) the entry value, the algorithm minimizes the average possible error of these ambiguous values, most probably not removing the error entirely; for this reason the algorithm performs far better when thresholding a signal value than when smoothing one.

When the jump size parameter is low the jumps in the original signal are closer in size to the small Gaussian noise, so the algorithm has to more often smooth the entry signal rather than threshold it. As explained above, when the algorithm is forced to smooth an entry signal it leads to greater errors than when it can simply threshold it. In such a way, it is clear why increasing the jump size so drastically ameliorates the results, because the algorithm can more often threshold the signal rather than smooth it. Nevertheless, it is not clear how restrictive of an assumption the jump size parameter is when actually working in the environment denoising is generally applied to, a randomly selected set of images. In this context, the jumps in a 1D signal represent edges in a 2D image. The greater the difference between edges in an image, the better the algorithm will translate and successfully denoise the image.

In optimizing the influence function and the kernel I show that the optimal kernel is

[25] As described in Section 1.2.1 hard thresholding the noisy finite differences will reveal a sparse result, since by assumption the original signal is a piecewise smooth function - and hence has sparse gradients (i.e. finite differences).

the finite difference operator. In other words, the optimal change of basis to represent and then denoise the input signal is the finite difference operator. Because the finite difference operator is simply the 1D analogue of the total variation of the signal, this result can be interpreted to imply that the total variation of a piecewise constant signal is the optimal change of basis to denoise it. However, a possible problem with this assertion is the inclusion of the cumulative matrix C into the objective function. The cumulative matrix recomputes the original signal from its finite differences. But if the optimal kernel were not the finite difference of the signal, it is unclear what the cumulative matrix would do to the linearly transformed signal. The cumulative matrix is necessary to revert the linearly transformed signal into its original basis, but including it in the objective function may also implicitly assume the kernel to be the finite difference operator.

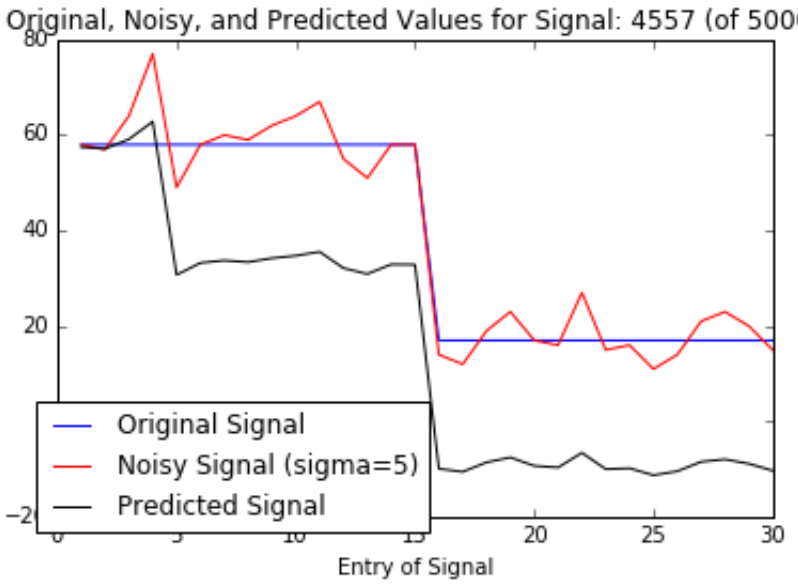


Figure 19: A large predicted training signal level jump error caused by recalculating the original signal via the cumulative matrix

Moreover, using the cumulative matrix also brings about a further complication in

estimating the original signal. If the predicted finite differences of the signal were incorrect at any point, then the entire remaining portion of the signal would be recomputed incorrectly. An example of this complication can be seen in Figure 19. There, the algorithm incorrectly judges the 2nd noisy jump as a jump in the original signal, decreasing the predicted signal to about 30. The algorithm then seems to accurately smooth and predict the remainder of the original signal, but with a very large level error caused only by that early finite difference miscalculation.

4 Appendix

In this appendix I discuss in detail some concepts and techniques referenced throughout the thesis.

4.1 Convolution Operator

The convolution operator is a very commonly used tool in signal and image processing. For signal processing the convolution operator's inputs are a kernel (or a vector) and a signal, and outputs another signal. The reason the convolution operator is so useful in signal processing is that for each entry in the original signal the kernel is applied to it and its surrounding neighbors, so local entries affect each other. In such a way, convolution can be efficiently used for effects that point operations cannot, such as: blurring, sharpening, and detection edges. It is defined for discrete function f and g so that:

$$(f * g)[i] = \sum_{n=-N}^{N} f[i-n]g[n]$$

Let {-N...N} is the support of g [**2**]

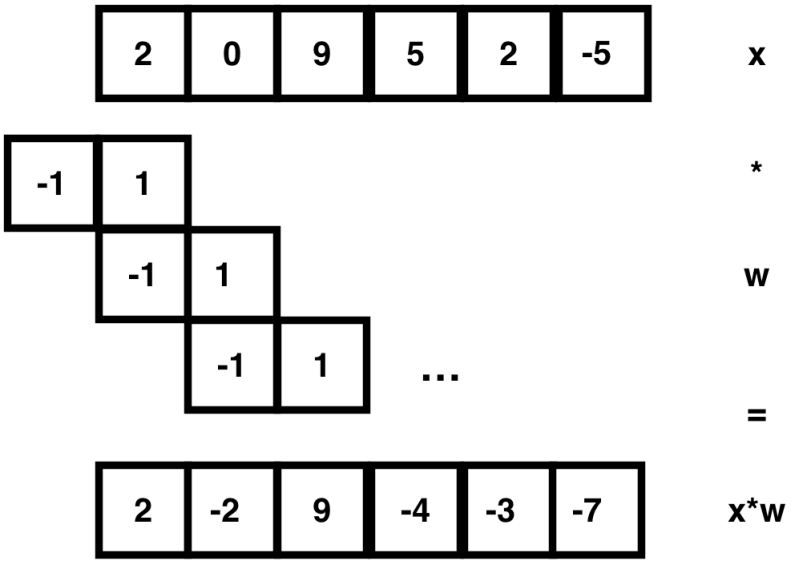


Figure 20: Convolution between the finite difference operator (w) and a signal (x)

Figure 20 shows the convolution between the finite difference operator (w=[-1,1]) and the vector x. The result that any convolution can alternatively be described by a dot product can be easily shown using the Convolution Theorem involving the Fourier Transform, however, we can intuitively see this since the convolution operator is simply a repeated local linear transformations on the original signal.[**2**] In such a way, if we have a signal u $\in \mathbb{R}^N$ and a kernel k $\in \mathbb{R}^L$, we can alternatively find a matrix K $\in \mathbb{R}^{NxN}$, so that:

$$u * k = u \cdot K$$

4.2 Radial Basis Function

A radial basis function is a real-valued function whose value monotonically depends only on the data point's distance from a predetermined center. Radial basis functions

have become very useful in different areas of machine learning with their primary use in function interpolation and data approximation.[**12**] There are several different types of radial basis functions, in this paper I use a 1-dimensional Gaussian radial basis function:

$$f(x) = \sum_{j=1}^{m} \alpha_j \exp(-\frac{|| \, x - \mu_j \, ||^2}{2\sigma^2})$$

where

x $\in \mathbb{R}$ is the input

f(x) $\in \mathbb{R}$ is the output

α_j the weight for the jth basis function

$|| \cdot ||$ denotes a norm ($\mathbb{R}^n \to \mathbb{R}$), usually the Euclidean norm

$\mu_j \in \mathbb{R}$ is the center for the jth basis function

$\sigma \in \mathbb{R}$ is the standard deviation of each basis Gaussian function

The Gaussian RBF has two parameters to be initialized: μ and σ. A primary problem when applying a radial basis function is determining how many basis function to use (i.e. m). If we choose a number of basis functions equal to the dimension of the input signal to estimate (i.e. $n = m$) , then we will have a perfectly linearly determined system, which leads to over fitting and a high generalization error. Instead if we use $m << n$ then the problem is a more interesting one, where we can solve for the optimal $\vec{\alpha}$=$[\alpha_1, \alpha_2, ..., \alpha_m]$ using a linear regression. The problem is set up so that:

$$\begin{bmatrix} \exp(-\frac{||x_1-\mu_1||^2}{2\sigma^2}) & \exp(-\frac{||x_1-\mu_2||^2}{2\sigma^2}) & \dots & \exp(-\frac{||x_1-\mu_M||^2}{2\sigma^2}) \\ \dots\dots\dots\dots & \dots\dots\dots\dots & \dots & \dots\dots\dots\dots \\ \exp(-\frac{||x_N-\mu_1||^2}{2\sigma^2}) & \exp(-\frac{||x_N-\mu_2||^2}{2\sigma^2}) & \dots & \exp(-\frac{||x_N-\mu_M||^2}{2\sigma^2}) \end{bmatrix} \begin{bmatrix} \alpha_1 \\ \dots \\ \alpha_M \end{bmatrix} = \begin{bmatrix} f(x_1) \\ \dots \\ f(x_N) \end{bmatrix}$$

[**9**, **16**]

The σ parameter is a bit easier to determine, the main concern is to prevent 'holes' in the radial basis function regions where none of the kernels have adequate support (see top of Figure 21). This can be solved by either increasing the σ (i.e. the standard deviation of each Gaussian function) or by renormalizing the radial basis functions (see bottom of figure 21). The renormalized radial basis function, $n(x)_j$, is defined so that:

$$n(x)_j = \frac{\frac{\exp(-\frac{||x-\mu_j||^2}{2\sigma^2})}{\lambda}}{\frac{\sum_{k=1}^{m}\exp(-\frac{||x-\mu_k||^2}{2\sigma^2})}{\lambda}}$$

[**9**]

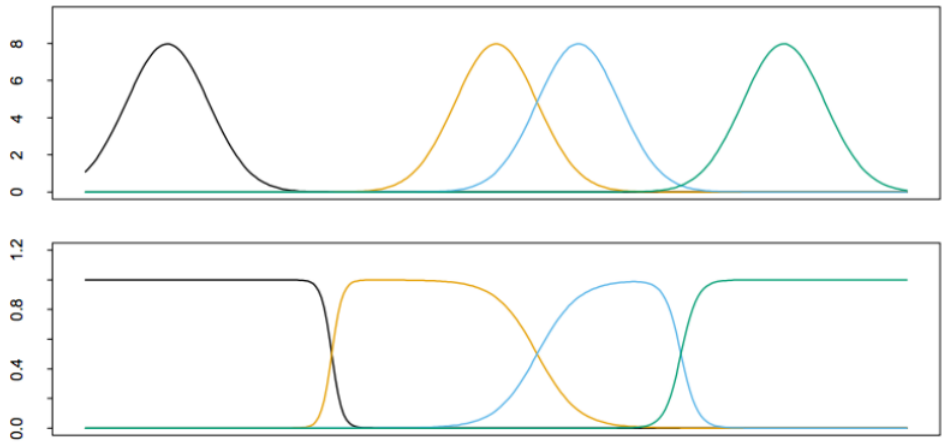


Figure 21: Gaussian RBFs with holes (top) and renormalized Gaussian RBFs (bottom). Graph from [**9**]

4.2.1 Radial Basis Function and Neural Network

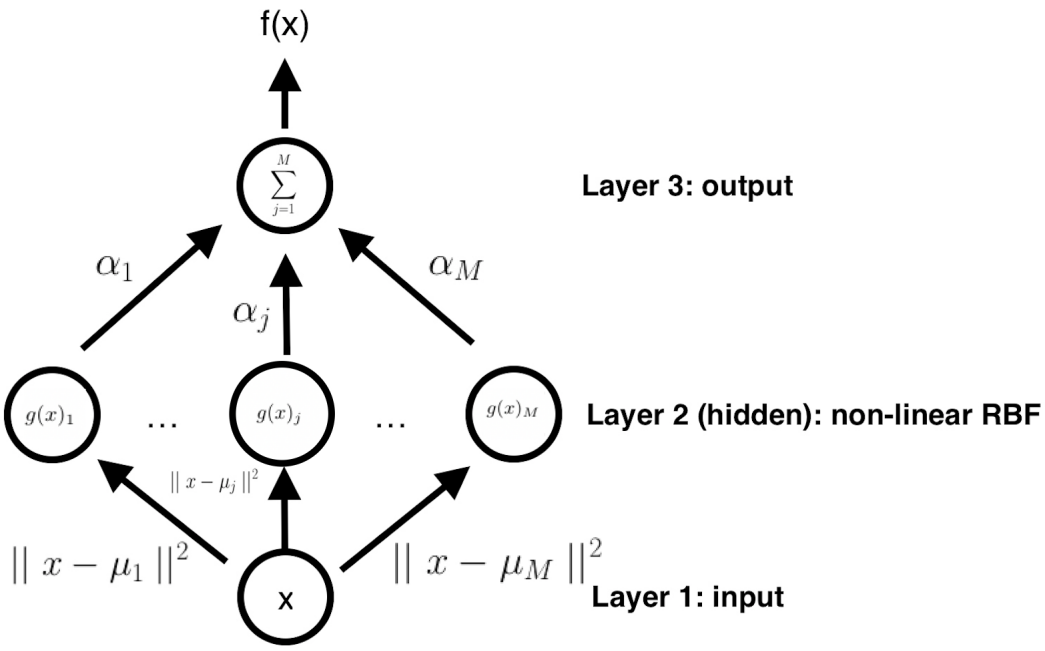


Figure 22: A radial basis function depicted as a 3-layer neural network with M neurons (where g_j is the jth radial basis function).

Radial basis functions used for function approximations can also be thought of as neural networks (and are denoted 'radial basis function networks'), in fact they were originally conceived as so.[**13**] RBF's can be interpreted as feed-forward neural networks with three layers: the input, a non-linear hidden layer (where the activation function is the RBF), and the output (see Figure 22). In our case, the second layer is the Gaussian non-linearity where each neuron has a specific center (μ_j) then multiplied by its respective weight (α_j).

4.3 Stochastic Gradient Descent

I minimize the differentiable objective function with respect to the l_2 convex norm, so the function is guaranteed to converge to its global (and not just local) minimum. I use a gradient based algorithm to minimize the influence function and kernel - stochastic gradient descent[**6**]. Stochastic gradient descent is widely used in machine learning, it can be very efficient for minimizing values with a large training set. Unlike the original gradient descent method which computes the gradient and updates the variables once for each iteration of the training set, stochastic gradient descent calculates the gradient and updates the parameter for every example inside the training set. Although usually faster, this sometimes results in a large amount of inter-sample noise, making the algorithm slowly oscillate down the error gradient.

For the stochastic gradient descent algorithm, an important parameter to fix is the step size per iteration. This parameter is important because setting the step size too small may cause the algorithm to converge extremely slowly, while setting it too large may cause it to diverge. Ideally we would search for the minimum of the function along the direction of descent (the gradient), called a 'line search'. However, this can be computationally expensive, so instead I chose to implement a faster backtracking line search heuristic which approximates this minimum from below - the Armijo backtracking algorithm[**6**].

4.4 Gradients

Recall the optimization problem:

$$\min_{\alpha,w} \frac{1}{2} \sum_{s=1}^{S} || \ x^s - C \cdot (g(w * y^s) \cdot \alpha) \ ||_2^2$$

For a fixed sample, $s \in S$, the problem becomes:

$$h = \frac{1}{2} \mid\mid x - C \cdot (g(w * y) \cdot \alpha) \mid\mid_2^2$$

$$\min_{\alpha,w} \frac{1}{2} \mid\mid x - C \cdot (g(w * y) \cdot \alpha) \mid\mid_2^2 = \min_{\alpha,w} h$$

And so there are two gradients to compute: $\frac{\partial h}{\partial \alpha}$ and $\frac{\partial h}{\partial w}$

Let us define γ to be the terms inside the norm: $\gamma = x - C \cdot (g(w * y) \cdot \alpha)$

$\implies h = \frac{1}{2} \mid\mid \gamma \mid\mid = \frac{1}{2} \sum_{i=1}^{N} (\gamma_i)^2$

So the gradients become:

$$\frac{dh}{d\alpha} = \frac{dh}{d\gamma} \cdot \frac{d\gamma}{d\alpha}$$
$$\frac{dh}{dw} = \frac{dh}{d\gamma} \cdot \frac{d\gamma}{dw}$$

4.4.1 $\frac{dh}{d\alpha}$

A) $\frac{dh}{d\gamma}$

h is a scalar and γ is a N dimensional vector so $\frac{dh}{d\gamma}$ will be an N dimensional vector

$h = \frac{1}{2} \mid\mid \gamma \mid\mid = \frac{1}{2}(\gamma_1^2 + \gamma_2^2 + ... + \gamma_N^2)$

$\implies \frac{\partial h}{\partial \gamma_i} = \gamma_i$

B) $\frac{d\gamma}{d\alpha}$

γ is an N dimensional vector and α is an M dimensional vector so $\frac{d\gamma}{d\alpha}$ will be an NxM matrix

Let $(w * y)_i$ be the ith entry of the convolution between w and y.

$\frac{d\gamma_i}{d\alpha} = \frac{d}{d\alpha} x_i - C_i \cdot (g[(w * y)_i] \cdot \alpha) = \frac{d}{d\alpha} x_i - C_i \cdot \sum_{j=1}^{M} (g[(w * y)_i]_j \alpha_j) =$

$= \frac{d}{d\alpha} x_i - C_i \cdot [g(w * y)_{i,1} \alpha_1 + g(w * y)_{i,2} \alpha_2 \ ... \ + \ ... \ g(w * y)_{i,M} \alpha_M]$

$\implies \quad \frac{\partial \gamma_i}{\partial \alpha_j} = -C_i \cdot g(w * y)_{i,j}$

4.4.2 $\frac{dh}{dw}$

A) $\frac{dh}{d\gamma}$

Same as above

B) $\frac{d\gamma}{dw}$

γ is a N dimensional vector and w is a K dimensional vector so $\frac{d\gamma}{dw}$ will be a NxK matrix

$$\gamma_i = x_i - [g[(w * y)_i] \cdot \alpha] = x_i - C_i \cdot \sum_{j=1}^{M} \exp(\frac{-((w*y)_i - \mu_j)^2}{2\sigma^2})\alpha_j$$

$$\implies \frac{\partial \gamma_i}{\partial w_k} = -C_i \cdot \sum_{j=1}^{M} \alpha_j \frac{-2y_i((w*y)_i - \mu_j)}{2\sigma^2} \exp(\frac{-((w*y)_i - \mu_j)^2}{2\sigma^2})$$

References

[1] José M Bioucas-Dias and Mário AT Figueiredo. A new twist: Two-step iterative shrinkage/thresholding algorithms for image restoration. *IEEE Transactions on Image processing*, 16(12):2992–3004, 2007.

[2] Ronald Bracewell. *The Fourier transform and its applications*. McGraw-Hill New York, 1986.

[3] Antonin Chambolle. An algorithm for total variation minimization and applications. *Journal of Mathematical imaging and vision*, 20(1):89–97, 2004.

[4] Antonin Chambolle, Vicent Caselles, Daniel Cremers, Matteo Novaga, and Thomas Pock. An introduction to total variation for image analysis. *Theoretical foundations and numerical methods for sparse recovery*, 9(263-340):227, 2010.

[5] Yunjin Chen, Wei Yu, and Thomas Pock. On learning optimized reaction diffusion processes for effective image restoration. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 5261–5269, 2015.

[6] Carlos Fernandez-Granda. NYU Mathematics MATH-GA 2840: Lecture notes on optimization methods, February 2016.

[7] Carlos Fernandez-Granda. NYU Mathematics MATH-GA 2840: Lecture notes on sparse linear models and denoising, February 2016.

[8] Mário AT Figueiredo. On gaussian radial basis function approximations: Interpretation, extensions, and learning strategies. In *Pattern Recognition, 2000. Proceedings. 15th International Conference on*, volume 2, pages 618–621. IEEE, 2000.

[9] Jerome Friedman, Trevor Hastie, and Robert Tibshirani. *The elements of statistical learning*, volume 1. Springer series in statistics New York, 2001.

[10] Trevor Hastie, Robert Tibshirani, and Martin Wainwright. *Statistical learning with sparsity: the lasso and generalizations*. CRC press, 2015.

[11] Mushtaq Ahmad Khan, Wen Chen, Asmat Ullah, and Zhuojia Fu. A mesh-free algorithm for rof model. *EURASIP Journal on Advances in Signal Processing*, 2017 (1):53, 2017.

[12] S Khowaja and SZS Shah. Noise reduction technique for images using radial basis function neural networks. *Mehran University Research Journal of Engineering and Technology*, 33(3):278–285, 2014.

[13] D Lowe. Multi-variable functional interpolation and adaptive networks. *Complex Systems*, 2:321–355.

[14] Pietro Perona and Jitendra Malik. Scale-space and edge detection using anisotropic diffusion. *IEEE Transactions on pattern analysis and machine intelligence*, 12(7): 629–639, 1990.

[15] Leonid I Rudin, Stanley Osher, and Emad Fatemi. Nonlinear total variation based noise removal algorithms. *Physica D: Nonlinear Phenomena*, 60(1-4):259–268, 1992.

[16] R Schaback. A practical guide to radial basis functions. *Electronic Resource*, 11, 2007.

[17] Yilun Wang, Junfeng Yang, Wotao Yin, and Yin Zhang. A new alternating minimization algorithm for total variation image reconstruction. *SIAM Journal on Imaging Sciences*, 1(3):248–272, 2008.