A Divide-and-Conquer Algorithm for 1D Total Variation Denoising

İlker Bayram

Dept. of Electronics and Communication Engineering, Istanbul Technical University, Istanbul, Turkey (e-mail : ibayram@itu.edu.tr).

Abstract

We propose a finite terminating algorithm for total variation denoising. The algorithm solves the problem by merging solutions of subproblems, obtained by partitioning the data. The idea rests on the 'tube' interpretation of the optimality conditions for total variation denoising. We demonstrate that the algorithm is comparable in speed with existing non-iterative algorithms.

Keywords: Total variation, denoising, finite terminating, tube methods, divide and conquer.

1. Introduction

In this paper, we consider the 1D total variation (TV) denoising problem and propose a new finite-terminating algorithm for its solution. Given a noisy observation signal, instead of attempting to solve the TV denoising directly, we break up the observation signal into smaller pieces to obtain subproblems. Then we derive a finite-terminating procedure that can merge the solutions of these subproblems to obtain the solution of the original problem. We also discuss how to recursively apply this procedure to obtain the solution of the subproblems.

Specifically, for a data vector $y \in \mathbb{R}^N$ and a weight vector $\lambda \in \mathbb{R}^{N-1}$, consider the total variation denoising problem,

$$\min_t \left\{ F_y(t,\lambda) = \frac{1}{2}\left\|y-t\right\|_2^2 + \sum_{i=1}^{N-1} \lambda_i \left|t_i - t_{i+1}\right| \right\}. \tag{1}$$

Let us denote the solution of this problem as $x(\lambda)$. That is,

$$x(\lambda) = \arg\min_t F_y(t,\lambda). \tag{2}$$

Now let K be an integer with $1 \leq K \leq N$ and suppose λ^* satisfies,

$$\lambda_i^* = \lambda_i, \quad \text{for } i \neq K, \tag{3a}$$

$$\lambda_K^* > \lambda_K. \tag{3b}$$

In this paper, we describe a procedure that takes $x(\lambda)$ to $x(\lambda^*)$ in a finite number of steps. If it $x(\lambda)$ is not already available, the procedure can also be used to obtain $x(\lambda)$, as will be clear in the sequel.

A problem as described above is of interest because it allows us to reduce the original length-N problem into problems of smaller sizes. To see this, suppose $x(\lambda^*)$ is the solution we seek. Also, let λ be such that $\lambda_K = 0$. Then, $F_y(\lambda, t)$ can be written as,

$$F_y(\lambda,t) = \left[\frac{1}{2}\sum_{i=1}^{K}\|y_i - t_i\|_2^2 + \sum_{i=1}^{K-1}\lambda_i\,|t_i - t_{i+1}|\right] + \left[\frac{1}{2}\sum_{i=K+1}^{N}\|y_i - t_i\|_2^2 + \sum_{i=K+1}^{N-1}\lambda_i\,|t_i - t_{i+1}|\right] \quad (4)$$

The two terms in square brackets are decoupled. Therefore, these two problems can be solved separately and then merged using the procedure described in this paper. In fact, we can iterate this procedure to further reduce the problem sizes. Finally, notice that for $N = 1$, the problem can be easily solved by setting $x_1 = y_1$. This allows us to solve the whole length-N problem using a 'divide-and-conquer' approach.

Previous Work

The algorithm developed in this paper makes use of the 'tube' interpretation of the solution [1, 2, 3, 4]. These methods rely on a transformation of the optimality conditions for the TV denoising problem. According to this interpretation, it turns out that solving the TV denoising problem is equivalent to finding the shortest line segment residing in a tube, determined by the input signal and the regularization parameter. Using this interpretation, Mammen and van de Geer [3] describe three algorithms, for 1D (first and higher order) TV denoising problems (see also [5] for generalizations). The first two algorithms are derived based on continuation ideas (see e.g. [6, 7, 8, 9, 10] for discussion on other continuation schemes of interest), where the size of the tubes are modified at each pass. The third algorithm in [3], owing to the special properties of TV solutions, constructs a piecewise constant solution by merging neighboring segments. Davies and Kovac [2] describe a finite terminating procedure with $O(N)$ complexity which starts from one end of the tube and grows two candidate solutions, which follow the upper and lower boundary of the tube. The actual solution is grown by monitoring the agreement between these two candidate solutions. More recently, Condat [1] proposed a finite-terminating algorithm with a fast implementation for the same problem. The algorithm operates locally, starting from one end of the 'tube' and moves to the other end of the tube, making sure that at each junction, the shortest path was chosen.

Although finite-terminating algorithms based on tube methods have appeared in the statistics literature, as Condat points out in [1], it appears that they are not well known in the signal processing literature. We think this is

partly because the focus in the signal processing literature is more on 2D and higher dimensional problems (see e.g. [11, 12, 13, 14, 15, 16]) and a straightforward extension of the tube methods to higher dimensions is not possible due to a lack of a natural order for multi-dimensional integer lattices. Nevertheless, iterative algorithms for related 1D problems have appeared in the literature [17, 18, 19, 20, 21, 22]. These works derive iterative algorithms based on the alternating direction method of multipliers (ADMM) [23, 24] or through majorization-minimization (MM) schemes [25, 26], occasionally using the dual description of TV used in [15]. Although a direct extension of 1D tube methods to 2D is not possible, we note that the insight gained from 1D problems can be put to use to derive tube methods for higher dimensional problems as well [4]. In this paper, we will restrict our attention to the 1D denoising problem. Although the proposed algorithm in discussion will not be applicable to the 2D total variation formulations, we think it could also be of interest in anisotropic schemes that are 'direction sensitive' [27, 28, 29].

Contribution

The algorithm proposed in this paper is not a special case of the algorithms mentioned above. In fact, its 'divide-and-conquer' strategy is inherently different from previous work which attempt to solve the whole problem at once. We also note that the algorithm can be used along with existing algorithms. Specifically, it can be used to merge the solutions of smaller subproblems, possibly solved by an algorithm of the user's choice and obtain the solution to a problem with additional data. Such a feature can be useful in situations where a limit on resources restricts the problem size that can be solved by available algorithms. Also, if a certain algorithm is known to perform well for small-size problems and poorly for large-size problems, the proposed algorithm can be used to obtain a 'hybrid' algorithm so as to make use of the mentioned algorithm for large size problems while still enjoying the high performance valid for small size problems.

Outline

In Section 2, we recall the optimality conditions and their interpretation that leads to the 'tube' methods. Based on these conditions, we present the algorithm for taking $x(\lambda)$ to $x(\lambda^*)$, in Section 3. In Section 4, we discuss how to use this algorithm to obtain the solution to a full TV-denoising problem. Section 5 contains a discussion of complexity as well as comparison of speed with other algorithms. Section 6 is the conclusion.

2. Optimality Conditions

We start by recalling the optimality conditions and review their interpretation presented in [3], from a discrete-time perspective.

Zeros of the Subdifferential

First, observe that we can express the 'penalty term' as

$$\sum_{i=1}^{N-1} \lambda_i \, |t_i - t_{i+1}| = \|\Lambda \, D \, t\|_1 \tag{5}$$

where D is an $N-1 \times N$ matrix defined as,

$$D = \begin{bmatrix} 1 & -1 & 0 & 0 & \dots & 0 \\ 0 & 1 & -1 & 0 & \dots & 0 \\ & & \dots & & & \\ 0 & 0 & \dots & 1 & -1 & 0 \\ 0 & 0 & \dots & 0 & 1 & -1 \end{bmatrix}, \tag{6}$$

and Λ is a diagonal matrix with λ_i's on the diagonals. Now let x denote the optimal solution to (1). The subdifferential [30, 31] of $\|\Lambda \, D \, \cdot \|_1$ evaluated at x is,

$$\partial \|\Lambda \, D \, \cdot \|_1 \Big|_x = D^T \, \Lambda \, \operatorname{sgn}(D \, x) \tag{7}$$

where 'sgn' denotes the componentwise (set-valued) sign function, which is defined for a single component $z \in \mathbb{R}$ as,

$$\operatorname{sgn}(z) = \begin{cases} \{-1\}, & \text{if } z < 0, \\ [-1, 1], & \text{if } z = 0, \\ \{1\}, & \text{if } z > 0. \end{cases} \tag{8}$$

Therefore, x is the optimal solution of (1) if and only if [30, 31] there exists, a vector u of length $(N-1)$, where

$$u_i \in \begin{cases} [-\lambda_i, \lambda_i] & \text{if } x_i = x_{i+1}, \\ \{\lambda_i\} & \text{if } x_i > x_{i+1}, \\ \{-\lambda_i\} & \text{if } x_i < x_{i+1}. \end{cases} \tag{9}$$

for $i = 1, 2, \dots, N-1$, such that

$$0 = x - y + D^T \, u. \tag{10}$$

In the sequel, we will transform these conditions to obtain a more geometrically revealing interpretation.

A Closer Look at the Matrices

Note that D^T is an $N \times N-1$ matrix. Observe also that for a special $N \times N$ matrix A, we have,

$$A\,D^T = \begin{bmatrix} 1 & 0 & 0 & \dots & 0 \\ 1 & 1 & 0 & \dots & 0 \\ & & \ddots & & \\ 1 & 1 & 1 & \dots & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & \dots & 0 \\ -1 & 1 & 0 & \dots & 0 \\ 0 & -1 & 0 & \dots & 0 \\ & & \vdots & & \\ 0 & \dots & 0 & 1 & 0 \\ 0 & \dots & 0 & -1 & 1 \\ 0 & \dots & 0 & 0 & -1 \end{bmatrix} \tag{11}$$

$$= \begin{bmatrix} I_{N-1} \\ 0 \end{bmatrix} \tag{12}$$

where I_{N-1} denotes the $N-1 \times N-1$ identity vector. Observe that A is an invertible matrix.

Transforming the Optimality Conditions

Consider now the optimality condiditions in (10). If we apply A to this system, we obtain,

$$\underbrace{\begin{bmatrix} y_1 \\ y_1 + y_2 \\ \vdots \\ \sum_{i=1}^{N-1} y_i \\ \sum_{i=1}^{N} y_i \end{bmatrix}}_{r} - \underbrace{\begin{bmatrix} x_1 \\ x_1 + x_2 \\ \vdots \\ \sum_{i=1}^{N-1} x_i \\ \sum_{i=1}^{N} x_i \end{bmatrix}}_{s} = \begin{bmatrix} u_1 \\ u_2 \\ \vdots \\ u_{N-1} \\ 0 \end{bmatrix}. \tag{13}$$

Notice that, since A is invertible, (13) is equivalent to (10). Now if we define $s_0 = 0$, then we have $x_i = s_i - s_{i-1}$ for $i = 1, 2, \dots, N$. Given this observation, the optimality conditions in terms of r and s are

$$s_0 = 0, \tag{14a}$$

$$s_N = r_N, \tag{14b}$$

and

$$s_i \in \begin{cases} [r_i - \lambda_i, r_i + \lambda_i], & \text{if } s_i = \big(s_{i-1} + s_{i+1}\big)/2, \\ \{r_i - \lambda_i\}, & \text{if } s_i > \big(s_{i-1} + s_{i+1}\big)/2, \\ \{r_i + \lambda_i\}, & \text{if } s_i < \big(s_{i-1} + s_{i+1}\big)/2, \end{cases} \tag{14c}$$

for $i = 1, 2, \dots, N-1$.

Tubes

To better understand these conditions, let us consider an example. Suppose the data y is as given in Fig. 1a. The running sum of this data, that is r, is shown in Fig. 1b with black dots. For each black dot, except the rightmost one, two hollow circles are placed with a vertical distance of λ_i; one above, one below the black dot. Also, a hollow circle is placed at $(0, 0)$ (for the condition (14a)) and at (N, r_N) (for the condition (14b)). We connect the hollow circles to obtain the tube drawn with solid lines. Now let s be the running sum of the solution and imagine we linearly interpolate s to obtain a continuous curve. The conditions (14) ask that this curve be the shortest curve in the tube that connects the leftmost point to the rightmost point. This curve is depicted in Fig. 1c with solid lines. Observe that, at the breakpoints (i.e., at the indices $i = 3$ and $i = 7$ for Fig. 1c) the third and the second conditions in (14c) respectively are in effect. For the rest of the indices (except for $i = 0$ and $i = N = 10$), the first condition in (14c) holds.

3. A Continuation Scheme

Given a weight vector $\lambda \in \mathbb{R}^{N-1}$ as defined prior to (1), suppose the minimizer of (1) is given as $x(\lambda)$. Also, let $s(\lambda)$ be defined as,

$$s_0(\lambda) = 0, \tag{15}$$

$$s_i(\lambda) = \sum_{k=1}^{i} x_i(\lambda), \qquad \text{for } i = 1, 2, \ldots, N. \tag{16}$$

In the following, we denote $s(\lambda)$ by s for simplicity of notation. We will refer to s_i as

- an 'upper corner' if '$s_i < (s_{i-1} + s_{i+1})/2$',
- a 'lower corner' if '$s_i > (s_{i-1} + s_{i+1})/2$'.

Referring to Fig. 1c, s_3 is an upper corner and s_7 is a lower corner. Notice that an upper corner s_i lies on the upper boundary (i.e., $s_i = r_i + \lambda_i$) and a lower corner lies on the lower boundary (i.e., $s_i = r_i - \lambda_i$). Suppose that we collect the indices of the corners in a set C. By convention, we also include 0 and N in C, although these are not indices of lower or upper corners according to our definition. Notice that in this case, C becomes a non-empty subset of integers in the range $[0, N]$.

For an index i, let us define $\mathcal{L}(i, C)$ as the greatest element of C less than i and $\mathcal{R}(i, C)$ as the smallest element of C greater than i. Note that $\mathcal{L}(i, C)$ and $\mathcal{R}(i, C)$ give the indices of the closest corners to s_i on the left and on the right

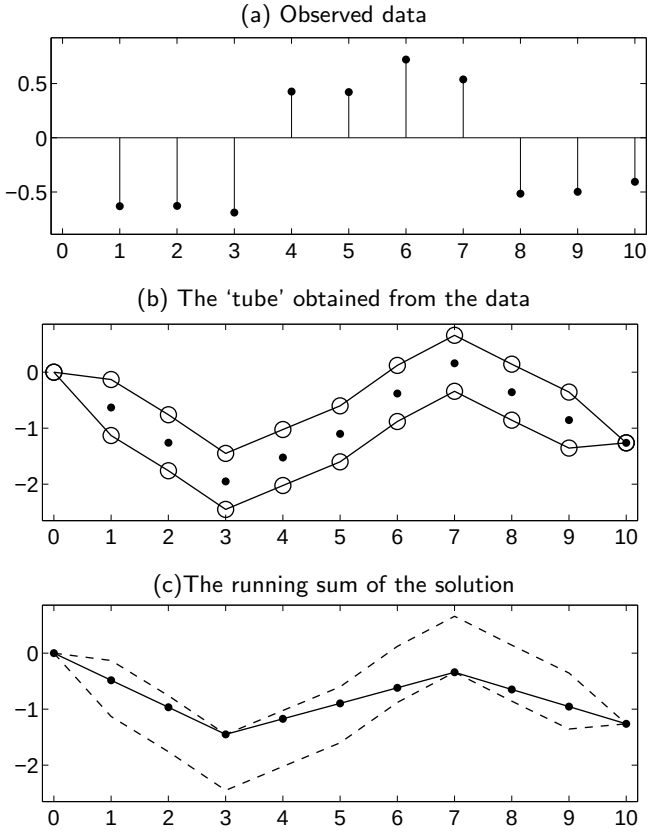

Figure 1: (a) y is the observed noisy data. (b) The black dots indicate the running sum of y. The hollow circles define the 'tube' which contains the running sum of the solution to (1). (c) If we linearly interpolate the running sum of the solution (shown with black dots), we obtain the shortest curve inside the tube that connects the leftmost hollow circle and the rightmost hollow circle.

respectively[1]. By the optimality conditions, if s_i is not a corner, we have,

$$s_i = \frac{s_{\mathcal{R}(i,C)} - s_{\mathcal{L}(i,C)}}{\mathcal{R}(i,C) - \mathcal{L}(i,C)} \left(i - \mathcal{L}(i,C)\right)) + s_{\mathcal{L}(i,C)} \tag{17}$$

$$= \frac{s_{\mathcal{L}(i,C)} - s_{\mathcal{R}(i,C)}}{\mathcal{L}(i,C) - \mathcal{R}(i,C)} \left(i - \mathcal{R}(i,C)\right) + s_{\mathcal{R}(i,C)} \tag{18}$$

In words, s_i can be obtained by a linear interpolation of the nearest corners to the left and right of the i^{th} position. As a consequence, it suffices to keep the values of the corner points only. In the algorithm that we describe, we will only modify the values of the corners and not the non-corner points, in order to keep the algorithm simple. To convert the pair (s, C), to the solution, we will make use of the following function

```
1: function CONVERT(C, s, N)
2:     L ← 0
3:     while L < N do
4:         R ← R(L, C)
5:         for i = L + 1 : R − 1 do
6:             s_i ← (s_R − s_L)/(R − L) (i − L) + s_L
7:         L ← R
8:     for i = 1 : N do
9:         x_i ← s_i − s_{i−1}
10:    return (x)
```

A Remark on Continuation Methods

Continuation methods that allow to vary a parameter of the problem continuously, typically start with the observation that the 'structure of the problem' (e.g. the set of 'active variables') is left unchanged for a neighborhood of the given value of the parameter (see e.g. [9, 7, 8, 10]). Based on this observation, in order to obtain the solution for different values of the parameter, a procedure that consists of two steps is employed. In the first step, the range of values of the parameter (where there is no change in the set of active variables) are computed. Following this, the second step consists of a modification to the set of active variables, in order to cross the boundary computed in the first step.

For our problem, the parameter of interest is λ_K, the K^{th} entry of the weight vector and the 'active variables' are the lower and upper corner points. Below, we discuss how the two substeps mentioned above are realized. We start with the computation of the (upper) bounds for λ_K that will not require any change in the set of corner points, C.

[1]The values $\mathcal{L}(0, C)$ and $\mathcal{R}(N, C)$, which are undefined according to our description, will not be needed in the sequel.

Incrementing the Regularization Parameter

Let e_K be the unit vector whose entries are all zero except for the K^{th} entry. Let us monitor how the solution changes as we go from λ to $\tilde{\lambda} = \lambda + \epsilon\, e_K$, for a small ϵ.

Now let $\tilde{s} = s(\lambda + \epsilon e_K)$, $L := \mathcal{L}(K, C)$, $R := \mathcal{R}(K, C)$. Notice that, for a sufficiently small ϵ, as we go from s to $\tilde{s}$, the indices of the corners will not change. Moreover, save for s_K, we will have

$$\tilde{s}_i = s_i, \quad \text{for } i \in C, i \neq K. \tag{19}$$

Consequently, we will also have

$$\tilde{s}_i = s_i, \quad \text{if } i < L, \text{ or } i > R. \tag{20}$$

Therefore, all we have to monitor are the indices i in the set $[L, R]$.

In the following, we will assume that s_K is a lower corner. This does not lead to a loss of generality because any upper corner can be transformed to a lower corner by multiplying the data by '-1'. Therefore, if s_K is actually an upper corner, we can multiply the data by '-1', apply the algorithm and multiply the resulting solution by '-1' again.

First, since s_K is a lower corner, if there is no change in the set of corner points, setting

$$\tilde{s}_K = s_K - \epsilon \tag{21}$$

will preserve the optimality conditions.

We now investigate the indices between $[L, R]$ in separate cases below and derive upper bounds on ϵ. If all of these upper bounds are satisfied, then the set of corner points remain the same.

- **Case I :** $L < i < K$

  For $L < i < K$, we should have, by (17),

$$\tilde{s}_i = \frac{s_K - \epsilon - s_L}{K - L}\,(i - L) + s_L. \tag{22}$$

Recall that the optimality condidions require $\tilde{s}_i \geq r_i - \lambda_i$. This constrains ϵ as,

$$\epsilon \leq (s_K - s_L) - \frac{K - L}{i - L}(r_i - \lambda_i - s_L), \quad \text{for } L < i < K. \tag{23}$$

These give $K - L$ conditions on ϵ.

Note that we are interested in the greatest ϵ that satisfies these constraints. Below, we summarize how to find the greatest ϵ that satisfies the constraints set by the indices between L and K.

```
1: function CheckLeft(C, K, r, s, λ)
2:     L ← L(K)
3:     if K − L < 2 then
4:         return (0, ∞)
```

5: **for** $L < i < K$ **do**
6: $\epsilon_i \leftarrow (s_K - s_L) - \frac{K-L}{i-L}(r_i - \lambda_i - s_L)$
7: $j \leftarrow \text{argmin}_{L<i<K}\ \epsilon_i$
8: **return** (j, ϵ_j)

- **Case II :** $K < i < R$

We can similarly derive the conditions for $K < i < R$ as,

$$\epsilon \leq (s_K - s_R) - \frac{K-R}{i-R}(r_i - \lambda_i - s_R), \quad \text{for } K < i < R. \tag{24}$$

We will make use of the function below in the sequel.

1: **function** CHECKRIGHT(C, K, r, s, λ)
2: $R \leftarrow \mathcal{R}(K)$
3: **if** $R - K < 2$ **then**
4: **return** $(0, \infty)$
5: **for** $K < i < R$ **do**
6: $\epsilon \leftarrow (s_K - s_R) - \frac{K-R}{i-R}(r_i - \lambda_i - s_R)$
7: $j \leftarrow \text{argmin}_{K<i<R}\ \epsilon_i$
8: **return** (j, ϵ_j)

- **Case III :** $i = L$

In order for $\tilde{s}_L$ to remain a corner, we should have that,

$$\frac{\tilde{s}_K - s_L}{K-L} < s_L - s_{L-1}, \text{ if } s_L \text{ is a lower corner,} \tag{25a}$$

$$\frac{\tilde{s}_K - s_L}{K-L} > s_L - s_{L-1}, \text{ if } s_L \text{ is an upper corner.} \tag{25b}$$

Because $\tilde{s}_K < s_K$, (25a) will always be satisfied. Therefore we only need to check what happens if s_L is an upper corner. In this case, (25b) is equivalent to,

$$\epsilon < (s_K - s_L) - (K-L)(s_L - s_{L-1}), \text{ if } s_L \text{ is an upper corner.} \tag{26}$$

1: **function** CHECKLEFTCORNER(C, K, s)
2: $L \leftarrow \mathcal{L}(K, C)$
3: **if** s_L is not an upper corner **then**
4: $\epsilon \leftarrow \infty$
5: **else**
6: $\epsilon \leftarrow (s_K - s_L) - (K-L)(s_L - s_{L-1})$
7: **return** ϵ

- **Case IV :** $i = R$

Similarly, for $\tilde{s}_R$ to remain a corner, we need

$$\epsilon < (s_K - s_R) - (K-R)(s_{R+1} - s_R), \text{ if } s_R \text{ is an upper corner.} \tag{27}$$

If s_R is a lower corner, then it will not pose any constraint on ϵ.

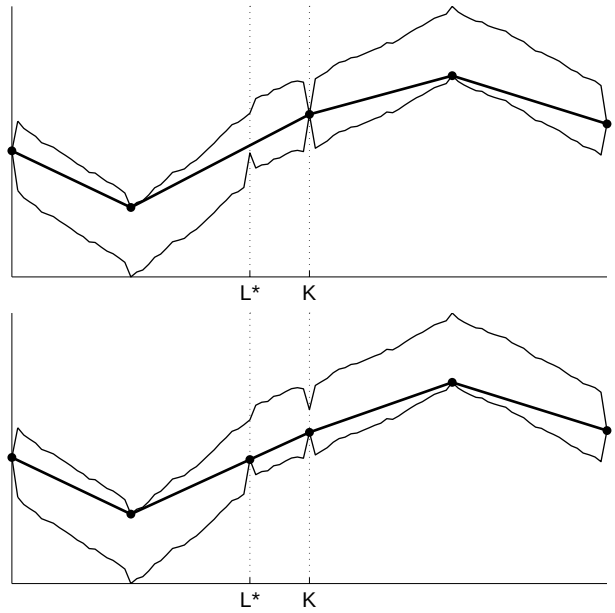

Figure 2: Top figure : Initially, we have two tubes to be connected at K. As we increase λ_K, without any modification to the corner list, we hit the bump at L^* shown in the bottom figure. In order to further increase λ_K, we add L^* to the corner list.

1: **function** CHECKRIGHTCORNER(C, K, s)
2: $R \leftarrow \mathcal{R}(K, C)$
3: **if** s_R is not an upper corner **then**
4: $\epsilon \leftarrow \infty$
5: **else**
6: $\epsilon \leftarrow (s_K - s_R) - (K - R)(s_{R+1} - s_R)$
7: **return** ϵ

- **Case V :** $i = K$

  Finally, if $\tilde{s}_K$ is to remain a lower corner, we need,

$$\frac{s_R - \tilde{s}_K}{R - K} < \frac{\tilde{s}_K - s_L}{K - L} \tag{28}$$

This is equivalent to,

$$\epsilon < \frac{(L - K)(s_R - s_K) + (R - K)(s_K - s_L)}{R - L}. \tag{29}$$

1: **function** CHECKNODE(C, K, s)
2: $R \leftarrow \mathcal{R}(K, C)$
3: $L \leftarrow \mathcal{L}(K, C)$
4: $\epsilon \leftarrow \frac{(L-K)(s_R-s_K)+(R-K)(s_K-s_L)}{R-L}$

5: **return** ϵ

Each case above sets a constraint on ϵ. As long as ϵ satisfies all of these constraints, there will be no change in the set of corner points, because the optimality conditions are still satisfied if we decrease s_K by ϵ. We now discuss how to increase ϵ beyond one of the constraints. This will typically require a modification of the corner set. As above, we study what happens in a number of different cases. The cases are distinguished based on where the active constraining index lies.

- **Case (a) : Constraint set by** $L < i < K$

  Using the notation above, suppose we compute ϵ_j for $j \in [L, R]$ and find that, for some L^* with $L < L^* < K$,

$$\epsilon_{L^*} \leq \epsilon_j \quad \forall j \in [L, R]. \tag{30}$$

This is demonstrated in Fig. 2. The starting configuration is shown in the top figure in Fig. 2. As the gap at K is allowed to increase, the curve will first touch the lower boundary at L^* for some ϵ^*. For this ϵ^*, the optimum line is depicted in Fig. 2. Note that if we further allow the gap at K to increase beyond ϵ^*, we cannot decrease the value of s_{L^*} while still residing in the tube. However, if we declare it as a corner, it will satisfy the condition for being a lower corner. Therefore, in order to cross the boundary ϵ^*, we add L^* to the set of corners C and note the value of s_{L^*} as $r_{L^*} - \lambda_{L^*}$.

- **Case (b) : Constraint set by** $K < i < R$

  Suppose now that for for some R^* with $K < R^* < R$,

$$\epsilon_{R^*} \leq \epsilon_j \quad \forall j \in [L, R]. \tag{31}$$

By symmetry, in this case, we add R^* to the corner list C and update s_{R^*} as $r_{R^*} - \lambda_{R^*}$.

- **Case (c) : Constraint set by** $i = L$

  Suppose that,

$$\epsilon_L \leq \epsilon_j \quad \forall j \in [L, R]. \tag{32}$$

This case is depicted in Fig. 3. When ϵ reaches ϵ_L, the slopes of the segments on the two sides of L are equal, i.e.

$$s_L = (s_{L-1} + s_{L+1})/2. \tag{33}$$

If we keep L as a corner and increase ϵ beyond ϵ_L, the optimality conditions will be violated at L. Note however, that if we remove L from the corner list, for a sufficiently small increase beyond ϵ_L, the optimality conditions will be satisfied.

- **Case (d) : Constraint set by** $i = R$

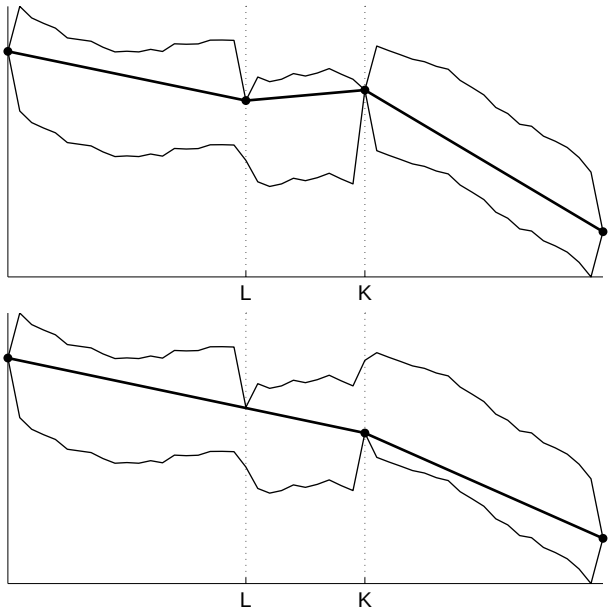

Figure 3: Top figure : Initially, we have an upper corner at L. Bottom figure : As λ_K is increased, the optimal solution is such that $s_L = (s_{L-1} + s_{L+1})/2$. If λ_K is further increased, L should be removed from the corner list.

If

$$\epsilon_R \leq \epsilon_j \quad \forall j \in [L, R], \tag{34}$$

in order to increase ϵ beyond ϵ_R, similarly as in Case III, we remove R from the corner list.

- **Case (e) : Constraint set by $i = K$**
  If

$$\epsilon_K \leq \epsilon_j \quad \forall j \in [L, R], \tag{35}$$

similarly as in Cases III and IV, we remove K from the corner list. Notice that in this case, further increasing ϵ will not affect the other corner points.

By the foregoing discussion, we conclude that Algorithm 1 can be used to take $s(\lambda)$ to $s\big(\lambda + (\lambda_K^* - \lambda_K)\, e_K\big)$. Note that if one wants to obtain the time-domain solutions, the function 'Convert' needs to be called.

Remark 1. In Cases III and IV above, we saw that if the corners to the left and to the right of K, say at positions L_l and R_l are lower corners, then they will remain so, no matter how much λ_K is incremented. This means that the segments to the left of L_l and to the right of R_l are not affected by the change at K. Such behavior is evidence to the locality of the TV filter as observed by other authors (see [32] and [2] which gives the credit for a similar observation

Algorithm 1 Algorithm for Incrementing λ

procedure GROW($\lambda, \lambda^*_K, K, C, s, r$)
  $L \leftarrow \mathcal{L}(K, C)$
  $R \leftarrow \mathcal{R}(K, C)$
  if s_K is an upper corner **then**
    $r \leftarrow -r$; $s \leftarrow -s$; $\gamma \leftarrow -1$
  else
    $\gamma \leftarrow 1$
  while $\lambda_K < \lambda^*_K$ **do**
    $(\epsilon_l, j_l) \leftarrow$ CheckLeft(C, K, r, s, λ)
    $(\epsilon_r, j_r) \leftarrow$ CheckRight(C, K, r, s, λ)
    $(\epsilon_{lC}) \leftarrow$ CheckLeftCorner(C, K, s)
    $(\epsilon_{lR}) \leftarrow$ CheckRightCorner(C, K, s)
    $(\epsilon_N) \leftarrow$ CheckNode(C, K, s)
    $\epsilon \leftarrow \min(\epsilon_l, \epsilon_r, \epsilon_{lC}, \epsilon_{rC}, \epsilon_N)$
    if $\epsilon > \lambda^*_K - \lambda_K$ **then**
      $s_K \leftarrow s_K - (\lambda^*_K - \lambda_K)$
      break ▷ We are done!
    else
      $s_K \leftarrow s_K - \epsilon$
      $\lambda_K \leftarrow \lambda_K + \epsilon$
    switch ϵ **do**
      case $\epsilon = \epsilon_l$ ▷ Case (a)
        Add j_l to the corner list C
        Update s_{j_l}
      case $\epsilon = \epsilon_l$ ▷ Case (b)
        Add j_r to the corner list C
        Update s_{j_r}
      case $\epsilon = \epsilon_{lC}$ ▷ Case (c)
        Remove L from the corner list C
      case $\epsilon = \epsilon_{lC}$ ▷ Case (d)
        Remove R from the corner list C
      case $\epsilon = \epsilon_N$ ▷ Case (e)
        Remove K from the corner list C
        $\lambda_K \leftarrow \lambda^*_K$
  $s \leftarrow \gamma\, s$
  return (s, C)

to Lutz Dümbgen). This local behavior will allow us to bound the expected computational complexity in Section 5.

4. Complete Algorithm

As noted in the Introduction, the procedure described in Algorithm 1 allows to merge the solution of two problems of length-$N/2$ to obtain the solution of a problem of length-N. If we iterate the procedure on the two subproblems, we can further subdivide the problems. Therefore, the procedure can be used to construct a divide-and-conquer type algorithm [33].

In order to obtain a divide-and-conquer algorithm, we also need to know the solution to a problem of a small size. We note at this point that if $\lambda_i = 0$ for all i, then $s_i = r_i$ is the optimal solution.

Divide-and-conquer type algorithms are usually described recursively. However, for implementation, a bottom-up description can be more useful. Algorithm 2 describes such a procedure.

Algorithm 2 Divide-and-Conquer Algorithm

procedure DAC(r, λ)
  $N \leftarrow \text{length}(r)$
  for $i = 1 : N$ **do**
    $s_i \leftarrow r_i$ ▷ Initialize the solution
    $\gamma_i \leftarrow 0$ ▷ γ will equal λ at exit
  $s_0 \leftarrow 0$
  $C \leftarrow [0, N]$ ▷ Initially every point is a corner
  $D \leftarrow [1, N-1]$ ▷ Ordered set of nodes to be visited
  while D is non-empty **do**
    $E \leftarrow \emptyset$ ▷ Holds the visited nodes on this pass
    for $i = 1 : 2 : |D|$ **do**
      $K \leftarrow D_i$
      $(s, C) \leftarrow \text{Grow}(\gamma, \lambda_K, K, C, s, r)$
      $\gamma_K \leftarrow \lambda_K$
      $E \leftarrow E \cup \{K\}$ ▷ K is visited
    $D \leftarrow D - E$ ▷ Remove the visited nodes from D
  $x \leftarrow \text{Convert}(s, C, N)$
  return x

Algorithm 2 solves the TV denoising problem by using the 'Grow' function described in Algorithm 1, starting from subproblems of size 1. Note however that, if we have a fast algorithm for moderate size problems, which possibly performs poorly as the problem size grows, then such an algorithm can also be used to obtain the solution to a problem of large size, using the 'Grow' function. We found such a hybrid algorithm to be a viable alternative to Algorithm 2.

Pseudocode for such an algorithm is provided in Algorithm 3[2]. Algorithm 3 employs an unspecified 'TV' function which returns the solution of a TV denoising problem, namely the corner index set C and the running sum of the the solution to the TV denoising problem.

Algorithm 3 A Hybrid Algorithm for TV Denoising

procedure HYBRIDTV(r, λ, M)
  $N \leftarrow \text{length}(r)$
  $D \leftarrow \emptyset$ ▷ Will hold the nodes to be visited
  $\gamma \leftarrow \lambda$ ▷ Initialize γ as a null vector
  for $i = 1 : M : N - M$ **do**
    $j \leftarrow \min(i + M - 1, N)$
    $t \leftarrow \begin{bmatrix} r_i & r_{i+1} & \dots & r_j \end{bmatrix}$ ▷ Subproblem's data
    $\alpha \leftarrow \begin{bmatrix} \lambda_i & \lambda_{i+1} & \dots & \lambda_j - 1 \end{bmatrix}$
    $(u, B) \leftarrow \text{TV}(t, \alpha)$
    $\begin{bmatrix} s_i & s_{i+1} & \dots & s_j \end{bmatrix} \leftarrow u$
    $C \leftarrow C \cup (B + i)$
    $\gamma \leftarrow \begin{bmatrix} \gamma & \alpha \end{bmatrix}$
    if $j < N$ **then**
      $D \leftarrow D \cup \{j\}$
      $\gamma_j \leftarrow 0$
  while D is non-empty **do**
    $E \leftarrow \emptyset$ ▷ Holds the visited nodes on this pass
    for $i = 1 : 2 : |D|$ **do**
      $K \leftarrow D_i$
      $(s, C) \leftarrow \text{Grow}(\gamma, \lambda_K, K, C, s, r)$
      $\gamma_K \leftarrow \lambda_K$
      $E \leftarrow E \cup \{K\}$ ▷ K is visited
    $D \leftarrow D - E$ ▷ Remove the visited nodes from D
  $x \leftarrow \text{Convert}(s, C, N)$
  return x

5. Complexity Analysis

Worst Case Analysis

Algorithm 1

In order to derive a bound on Algorithm 2, we first analyze Algorithm 1. Suppose the input length is N and $K = N/2$. Also, let the closest corner to the left of K be N_1 samples away from K. In order to find bounds on ϵ, the function 'CheckLeft' and 'CheckLeftCorner' requires us to compute N_1 linear

[2]Matlab Code for this algorithm is available at '`http://web.itu.edu.tr/ibayram/TVDenoise/`'.

terms. Suppose now that at some point, the corner list to the left of K, needs to be modified. There are two possibilities.

(i) If a new corner, say $N_2 < N_1$ samples away from K is to be added (i.e. Case(a) occurs), then an additional N_2 computations are required. In this case, further modifications to the corner list can only be in the form of additions of corners that monotonely approach K. Therefore, the total cost in this case is bounded by,

$$N_1 + N_2 + \ldots + N_k, \tag{36}$$

where $N/2 > N_i > N_{i+1}$ and $k < N/2$. Therefore, in this case, the complexity is $O(N^2)$.

(ii) If the left corner is to be removed (i.e. Case(c) occurs), then a corner N_2 samples away from K will be considered in order to compute the next value of ϵ, where $N_2 > N_1$. This increases the computation count by N_2. Now, for this new corner, the two conditions in this current list have to be considered, again. Suppose, we keep on removing left corners like this for k more times. In this case, by the above discussion, the computation count is bounded by

$$N_1 + N_2 + \ldots + N_k + O(N^2). \tag{37}$$

Note that $N_i < N_{i+1}$, $N_i < N/2$ and $k < N/2$. Therefore, the total computational complexity is bounded as $O(N^2)$.

By symmetry, we can argue that the right side of K will also require $O(N^2)$ computations. Therefore Algorithm 1 needs $O(N^2)$ computations.

Algorithm 2

From the ongoing analysis, we can conclude that, for a constant c, the complexity of Algorithm 2 is bounded by,

$$c\,N^2 + c\,2\left(\frac{N}{2}\right)^2 + c\,4\left(\frac{N}{4}\right)^2 + \ldots < \sum_{i=0}^{\infty} c\,2^i\left(\frac{N}{2^i}\right)^2 \tag{38}$$

$$= c\,2\,N^2. \tag{39}$$

In words, the recursive algorithm has $O(N^2)$ complexity as well.

Probabilistic Analysis

As noted in Case III in Section 3 (see also Remark 1), if there is a lower corner, N_1 samples to the left of K, then Case (ii) cannot occur and therefore the number of computations are bounded by

$$N_1 + (N_1 - 1) + \ldots + 1 = N_1(N_1 + 1)/2. \tag{40}$$

Now assume that for a certain type of input, the probability of coming across a lower corner, i samples away from the end point is given by $P(i)$. In that case, the expected complexity (taking into account the segments both on the left and on the right of K) of merging two segments of length N is bounded by,

$$\sum_{i=1}^{N} P(i)\, i\,(i+1)/2 \leq \sum_{i=1}^{\infty} P(i)\, i\,(i+1)/2 \tag{41}$$

If the probability mass function $P(i)$ has finite variance, then both terms in (41) will converge. In this case, since the right hand side of (41) will be a constant independent of N, we obtain that the 'Grow' function has $O(1)$ expected complexity. It follows that the complexity of Algorithm 2 is bounded by

$$c+2\,c + 2^2\,c + \ldots + 2^{\log_2(N)}\,c \tag{42}$$

$$= c\,(N + N/2 + N/4 + \ldots + 1) \leq 2c\,N, \tag{43}$$

where c is a constant. Thus, the expected complexity of Algorithm 2 is $O(N)$.

Running Times

We now present a speed comparison of the 'Taut String Algorithm'[3] [2] with the hybrid algorithm described in Algorithm 3. In Algorithm 3, we used one of the homotopy algorithms (Algorithm 2) described in [3], which we refer to as the 'homotopy algorithm', to solve problems of size 200. We considered input signals of length $N = 2^i$ for $i = 8, \ldots, 16$. For each N, we generated random piecewise constant signals that have approximately $N/100$ constant pieces, scattered uniformly. Also, the lengths of the constant pieces follow a uniform distribution. Then, we added white Gaussian noise to obtain the observation signals (SNR $\approx$ 16 dB). We took the regularization parameter for the TV term as $\lambda = 3\sigma$, where σ denotes the noise standard deviation. Fig. 4a shows the (log) average running times for each N value (averaged over 5 realizations each). We observe that the running time increases linearly with the input length, although the worst case complexity is $O(N^2)$ for both algorithms. Observe that the original homotopy algorithm performs poorly compared to the Taut String algorithm. However, the hybrid algorithm, which merges the solutions of small size problems obtained using the homotopy algorithm, runs faster than the Taut String algorithm. We also would like to note that the improvement in speed due to the hybrid algorithm is not as much as the improvement reported by Condat in [1]. We also included in Fig. 4 the projected running times of Condat's algorithm. These projected values are obtained by dividing the running time of the Taut string algorithm by a constant factor of 55/25, as reported in [1].

Fig. 4b shows the (log) average running times for the same experiment but when a larger regularization parameter is used ($\lambda = 6\sigma$). For this value of λ, the homotopy algorithm runs faster than the taut string algorithm for $N \leq 2^{10}$.

[3] We used the Matlab implementation by Lutz Dümbgen [34].

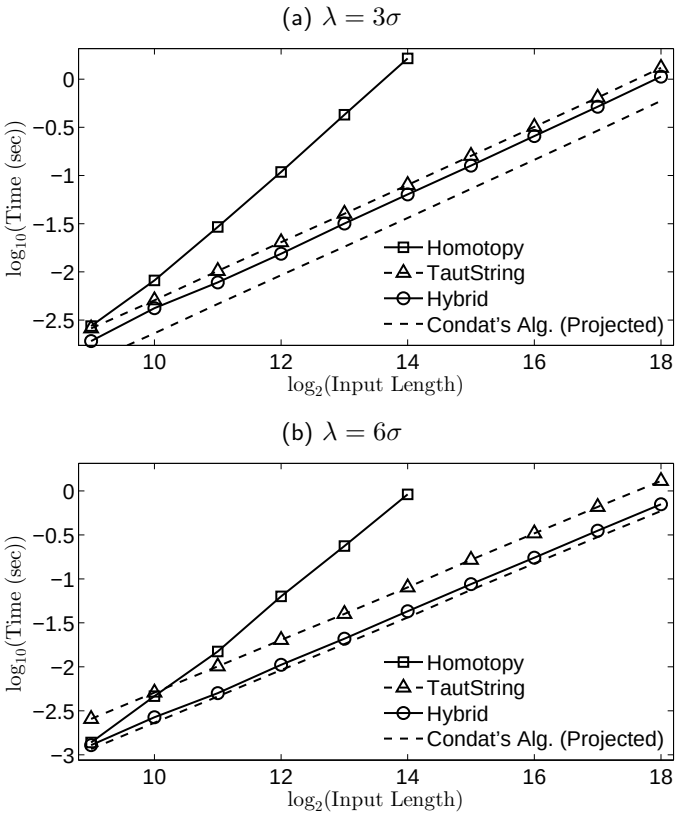

Figure 4: Running times in Matlab for the 'Taut String' algorithm [2] (Matlab code written by Lutz Dümbgen) and the proposed hybrid algorithm as a function of input length. The two curves are very close to each other with the hybrid algorithm slightly better than the Taut String implementation.

But for higher N values, its performance degrades. The hybrid algorithm allows to extend the good performance of the homotopy algorithm to high values of N. This observation is in line with the claim in the Introduction that the hybrid algorithm can indeed enhance the performance if it is supplied with a fast algorithm that can solve problems of small size.

One comment about the dependence of the algorithm on the value of the regularization parameter λ is in order. We see from Fig. 4 that the running time of the taut string algorithm does not depend on λ, as is also expected [1]. On the other hand, the homotopy algorithm employed in the experiments starts from a large value of λ and gradually decreases it until it reaches the desired value (see [3]). Therefore, the homotopy algorithm requires fewer computations for higher λ values and the observed behavior is actually expected. But Algorithm 1, which is used to merge the solutions obtained by the homotopy algorithm, actually starts from low values of λ (at the junctions) and slowly increases λ. Therefore, it is expected to perform faster for lower values of λ. In this respect, we think that the time-dependence of the total hybrid algorithm is mainly due to the homotopy algorithm and not Algorithm 1.

6. Conclusion

We proposed a method for the 1D TV denoising problem that allows to merge the solutions of small size TV problems so as to obtain the solution to a larger size problem. When used with a high-performance algorithm for small size problems, the method obtains the desired solution with speed comparable to those of existing fast algorithms.

Although the algorithm solves the 1D denoising problem, we think that it could be of use in high dimensional problems as well. In fact, we think that the availability of fast algorithms for 1D denoising is likely to lead to the developments of practical schemes for higher dimensional problems. For instance, if we have at hand distorted data of an image which consists of a small number of directional structures with known orientation, the algorithm can be used in a recovery scheme that is sensitive to the directional features, as discussed in [29, 27]. We hope to use algorithms as described in this paper to pursue such problems in the near future.

Another extension of the algorithm that would be of interest is total variation problems involving complex valued data. Such problems might arise for instance in spectrogram denoising, where the data consists of the STFT coefficients of the signal (see e.g. [35]). The extension of the presented algorithms for complex valued data is not trivial because the 'tubes' for complex valued data actually reside in three dimensional space. Nevertheless, we think that the divide and conquer approach presented in this paper would be useful in deriving an algorithm for complex valued data as well.

References

[1] L. Condat, A direct algorithm for 1D total variation denoising, IEEE Signal Processing Letters 20 (11) (2013) 1054–1057.

[2] P. L. Davies, A. Kovac, Local extremes, runs, strings and multiresolution, The Annals of Statistics 29 (1) (2001) 1–48.

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

[4] W. Hinterberger, M. Hintermüller, K. Kunisch, M. Von Oehsen, O. Scherzer, Tube methods for BV regularization, Journal of Mathematical Imaging and Vision 19 (2003) 219–235.

[5] L. Dümbgen, A. Kovac, Extensions of smoothing via taut strings, Electronic Journal of Statistics 3 (2009) 41–75.

[6] S. Rosset, J. Zhu, Piecewise linear regularized solution paths, The Annals of Statistics 35 (3) (2007) 1012–1030.

[7] J. J. Fuchs, Fast implementation of a $\ell_1 - \ell_1$ regularized sparse representations algorithm, in: Proc. IEEE Int. Conf. on Acoustics, Speech and Signal Proc. (ICASSP), 2009.

[8] D. M. Malioutov, M. Çetin, A. S. Willsky, Homotopy continuation for sparse signal representation, in: Proc. IEEE Int. Conf. on Acoustics, Speech and Signal Proc. (ICASSP), 2005.

[9] B. Efron, T. Hastie, I. Johnstone, R. Tibshirani, Least angle regression, Annals of Statistics 32 (2) (2004) 407–499.

[10] M. R. Osborne, B. Presnell, B. A. Turlach, A new approach to variable selection in least squares problems, IMA Journal of Numerical Analysis 20 (3) (2000) 389–403.

[11] L. Rudin, S. Osher, E. Fatemi, Nonlinear total variation based noise removal algorithms, Physica D 60 (1-4) (1992) 259–268.

[12] A. Beck, M. Teboulle, Fast gradient-based algorithms for constrained total variation image denoising and deblurring problems, IEEE Trans. Image Proc. 18 (11) (2009) 2419–2434.

[13] M. Zhu, T. F. Chan, An efficient primal-dual hybrid gradient algorithm for total variation image restoration, UCLA CAM Report [08-34], May, 2008 (May 2008).

[14] C. R. Vogel, M. E. Oman, Iterative methods for total variation denoising, SIAM J. Sci. Comput. 17 (1996) 227–238.

[15] A. Chambolle, An algorithm for total variation minimization and applications, Journal of Mathematical Imaging and Vision 20 (1-2) (2004) 89–97.

[16] J. M. Bioucas-Dias, M. A. T. Figueiredo, J. P. Oliveira, Total variation-based image deconvolution: A majorization-minimization approach, in: Proc. IEEE Int. Conf. on Acoustics, Speech and Signal Proc. (ICASSP), 2006.

[17] F. I. Karahanoğlu, I. Bayram, D. Van De Ville, A signal processing approach to generalized 1-D total variation, IEEE Trans. Signal Processing 59 (11) (2011) 5265–5274.

[18] I. W. Selesnick, I. Bayram, Total variation filtering, `http://cnx.org/content/m31292/1.1/`, connexions web site (2009).

[19] I. W. Selesnick, Total variation denoising (An MM algorithm), `http://cnx.org/content/m44934/latest/`, connexions web site (2012).

[20] I. W. Selesnick, S. Arnold, V. R. Dantham, Polynomial smoothing of time series with additive step discontinuities, IEEE Trans. Signal Processing 60 (12) (2012) 6305–6318.

[21] S. Yang, J. Wang, W. Fan, X. Zhang, P. Wonka, J. Ye, An efficient ADMM algorithm for multidimensional anisotropic total variation regularization problems, in: Proc. ACM SIGKDD Int. Conf. on Knowledge Discovery and Data Mining, 2013.

[22] B. Wahlberg, S. Boyd, M. Annergren, Y. Wang, An ADMM algorithm for a class of total variation regularized estimation problems, in: Proc. IFAC Symp. on System Identification, 2012.

[23] J. Eckstein, D. P. Bertsekas, On the Douglas-Rachford splitting method and the proximal point algorithm for maximal monotone operators, Mathematical Programming 55 (3) (1992) 293–318.

[24] S. Boyd, N. Parikh, E. Chu, B. Peleato, J. Eckstein, Distributed optimization and statistical learning via the alternating direction method of multipliers, Foundations and Trends in Machine Learning 3 (1) (2011) 1–122.

[25] M. A. T. Figueiredo, J. M. Bioucas-Dias, R. D. Nowak, Majorization-minimization algorithms for wavelet-based image restoration, IEEE Trans. Image Proc. 16 (12) (2007) 2980–2991.

[26] D. R. Hunter, K. Lange, A tutorial on MM algorithms, Amer. Statist. 58 (1) (2004) 30–37.

[27] D. R. Pipa, S. H. Chan, T. Q. Nguyen, Directional decomposition based total variation image restoration, in: Proc. Eur. Sig. Proc. Conf (EUSIPCO), 2012.

[28] S. Esedoglu, Blind deconvolution of bar code signals, Inverse Problems 20 (1) (2004) 121–135.

[29] I. Bayram, M. E. Kamaşak, Directional total variation, IEEE Signal Processing Letters 19 (12) (2012) 781–784.

[30] J.-B. Hiriart-Urruty, C. Lemaréchal, Fundamentals of Convex Analysis, Springer, 2004.

[31] R. T. Rockafellar, Convex Analysis, Princeton University Press, 1996.

[32] C. Louchet, L. Moisan, Total variation as a local filter, SIAM Journal on Imaging Sciences 4 (2) (2011) 651–694.

[33] T. H. Cormen, C. E. Leiserson, R. L. Rivest, C. Stein, Introduction to Algorithms, 3rd Edition, MIT Press, 2009.

[34] L. Dümbgen, Matlab code for the taut string algorithm, `http://www.imsv.unibe.ch/content/staff/personalhomepages/duembgen/software/multiscale_densities/index_eng.html`.

[35] I. Bayram, M. E. Kamaşak, A simple prior for audio signals, IEEE Trans. Audio, Speech and Language Proc. 21 (6) (2013) 1190–1200.