Max–Planck–Institut für biologische Kybernetik
Max Planck Institute for Biological Cybernetics

Technical Report No. 194

Fast algorithms for total-variation based optimization

Álvaro Barbero[12] and Suvrit Sra[1]

26 August, 2010

This Technical Report has been approved by:

Director at MPIK

Postdoc at MPIK

Max–Planck–Institut für biologische Kybernetik
Max Planck Institute for Biological Cybernetics

Technical Report No. 194

Fast algorithms for total-variation based optimization

Álvaro Barbero[12] and Suvrit Sra[1]

26 August, 2010

[1] MPI für biologische Kybernetik
[2] Universidad Autónoma de Madrid and Instituto de Ingeniería del Conocimiento

Fast algorithms for total-variation based optimization

Álvaro Barbero and Suvrit Sra

Abstract. We derive a number of methods to solve efficiently simple optimization problems subject to a total-variation (TV) regularization, under different norms of the TV operator and both for the case of 1-dimensional and 2-dimensional data. In spite of the non-smooth, non-separable nature of the TV terms considered, we show that a dual formulation with strong structure can be derived. Taking advantage of this structure we develop adaptions of existing algorithms from the optimization literature, resulting in efficient methods for the problem at hand. Experimental results show that for 1-dimensional data the proposed methods achieve convergence within good accuracy levels in practically linear time, both for L1 and L2 norms. For the more challenging 2-dimensional case a performance of order $O(N^2 \log_2 N)$ for $N \times N$ inputs is achieved when using the L2 norm. A final section suggests possible extensions and lines of further work.

---

1 Introduction

The total-variation (TV) operator usually appears in the context of image and signal procesing problems, and can be defined as follows. Given a signal defined as a function $f : \Omega \rightarrow \mathcal{R}$, where Ω is a bounded open subset of $\mathcal{R}^D$, the total variation of the signal is defined as

$$TV_d(f) = \int_\Omega ||\nabla f(x)||_p dx,$$

with $||\nabla f(x)||_p$ the Lp norm ($p \geq 1$) of the gradient of f at x [13]. If Ω is a discrete space which can be represented as an $N \times N$ matrix $\boldsymbol{X}$ (an square image), this formula can be simplified down to

$$\begin{aligned} TV_2(\boldsymbol{X}) &= \sum_{i,j} ||\partial^+ \boldsymbol{X}_{i,j}||_p, \\ \partial^+ \boldsymbol{X}_{i,j} &= (\boldsymbol{X}_{i+1,j} - \boldsymbol{X}_{i,j}\ ,\ \boldsymbol{X}_{i,j+1} - \boldsymbol{X}_{i,j}), \end{aligned} \tag{1.1}$$

i.e. it is the norm of the discrete gradients of the image. Similarly, for a 1-dimensional signal the TV is defined as

$$TV_1(\boldsymbol{x}) = \sum_i ||\boldsymbol{x}_{i+1} - \boldsymbol{x}_i||_p = \sum_i |\boldsymbol{x}_{i+1} - \boldsymbol{x}_i| \tag{1.2}$$

where the norm gets reduced to the absolute value (or L1 norm), as each discrete gradient consists of a single-entry vector.

One common application of the TV operator is the field of image denoising, where it was introduced by Rudin, Osher and Fatemi in what became to be known as the ROF model [23]. In this setting only a noisy version y of a true image x is known, and we strive to recover y from x under the assumptions that x should be similar to y and that the true image x should be smooth, i.e. the gradient of x should be small. In the ROF model this is translated to the following optimization problem

$$\min_x \frac{1}{2}||x - y||_2^2 + \lambda\, TV_d(x), \tag{1.3}$$

where λ is a penalty parameter for non-smoothness.

In this work we will show how problems in the form (1.3) can be solved efficiently via a transformation to a simpler dual formulation, which allows to take advantage of the structure induced by the total variation norm. We will then analyze several specific cases of this formulation, and show how standard optimization algorithms can be

adapted to make use of this structure, resulting in efficient and accurate solvers. Finally we will discuss possible extensions and means of improvement of this work.

2 Dual and structure of the TV problem

2.1 Matrix formulation of the TV problem

The first step towards deriving the dual of the TV problem (1.3) is rewriting it in matrix form. This is easy for the 1-dimensional case, where we define the matrix

$$\boldsymbol{D} = \begin{pmatrix} -1 & 1 & & & & & \\ & -1 & 1 & & & & \\ & & -1 & 1 & & & \\ & & & & \ddots & & \\ & & & & & -1 & 1 \\ & & & & & & -1 \quad 1 \end{pmatrix} \in \mathbb{R}^{(N-1)\times N}, \tag{2.1}$$

so that the 1-dimensional TV operator is

$$TV_1(\boldsymbol{x}) \approx ||\boldsymbol{D}\boldsymbol{x}||_p. \tag{2.2}$$

Note that this formulation is equivalent to (1.2) only when $p = 1$. For other p-norms we can consider this approach a generalizacion of the 1-dimensional TV operator. This results in the optimization problem

$$\min_{\boldsymbol{x}} \frac{1}{2}||\boldsymbol{x} - \boldsymbol{y}||_2^2 + \lambda\|\boldsymbol{D}\boldsymbol{x}\|_p. \tag{2.3}$$

Regarding the 2-dimensional case, the counterpart of $\boldsymbol{D}$ is defined as

$$\boldsymbol{B} = \begin{pmatrix} -1 & 1 & & & & & & & & & & & & & & \\ & \ddots & \ddots & & & & & & & & & & & & & \\ & & -1 & 1 & & & & & & & & & & & & \\ & & & & -1 & 1 & & & & & & & & & & \\ & & & & & \ddots & \ddots & & & & & & & & & \\ & & & & & & -1 & 1 & & & & & & & & \\ & & & & & & \ddots & \ddots & \ddots & \ddots & & & & & & \\ & & & & & & & & -1 & 1 & & & & & \\ & & & & & & & & & \ddots & \ddots & & & & \\ & & & & & & & & & & -1 & 1 & & & \\ -1 & & & & 1 & & & & & & & & & & \\ & -1 & & & & & 1 & & & & & & & & \\ & & \ddots & & & & & \ddots & & & & & & & \\ & & & \ddots & & & & & \ddots & & & & & & \\ & & & & \ddots & & & & & \ddots & & & & & \\ & & & & & \ddots & & & & & \ddots & & & & \\ & & & & & & \ddots & & & & & \ddots & & & \\ & & & & & & & -1 & & & & & 1 & & \end{pmatrix} \in \mathbb{R}^{2(N-1)N\times N^2},$$

so that 2-dimensional TV operator is

$$TV_2(\boldsymbol{x}) \approx ||\boldsymbol{B}\ vec(\boldsymbol{X})||_p, \tag{2.4}$$

where $vec(\boldsymbol{X})$ returns a vectorized version of $\boldsymbol{X}$, constructed by stacking the columns of $\boldsymbol{X}$. This results in the optimization problem

$$\min_{\boldsymbol{x}} \frac{1}{2}||\boldsymbol{x}-\boldsymbol{y}||_2^2 + \lambda\|\boldsymbol{B}\ vec(\boldsymbol{X})\|_p. \tag{2.5}$$

Again, this formulation is equivalent to (1.1) only when $p = 1$. Conversely, these expressions give rise to different models for any $p > 1$. These differences can be stated more clearly when interpreting them as mixed norms. Recall that a mixed norm is defined as

$$||x||_{p,q} = \left(\sum_a \left(\sum_b |x_{ab}|^p\right)^{q/p}\right)^{1/q},$$

which essentially boils down to considering a block structure in x, applying a norm p in each block, and then applying a norm q along blocks. Under this point of view, the original TV operator (1.1) can be regarded as a $(p, 1)$ mixed norm, in which each block is a two-entry vector component. On the other hand, our proposed approach (2.4) is nothing but a standard p norm. Though this results in a different optimization problem (and thus, model), we will see that our approach has computational advantages over the more difficult to handle mixed norm.

Whatever the case, we see that the TV operator can be rewritten (or approximated) as an uncostrained convex problem in the form

$$\min_{\boldsymbol{x}} \frac{1}{2}||\boldsymbol{x}-\boldsymbol{y}||_2^2 + \lambda\|\boldsymbol{M}\boldsymbol{x}\|_p. \tag{2.6}$$

While the first term is differentiable, the second term is non-smooth and, more importantly, non-separable, i.e. cannot be broken down into independent components involving single entries $\boldsymbol{x}_i$. This kind of problem has been addressed thoroughly in the literature in the form of the so-called "proximity operators" [8], which have the form

$$Prox_R(y) = \arg\min_{x\in X} \frac{1}{2}||x-y||_2^2 + R(x), \tag{2.7}$$

where R is another operator. Thus, problem (2.6) is nothing but the proximity operator of the total variation. We will later see in more detail the importance of this connection. For now, the interesting point is the fact that if we had a matrix $\boldsymbol{M} = \boldsymbol{I}$ then (2.6) would turn out to be separable. What is more, proximity operators for this cases exist, which allow to find a solution in closed form for the $p = 1$ and $p = 2$ norm cases, and in roughly linear time for the $p = \infty$ norm [16]. This observation will also be of use in our proposed methods.

2.2 Duality of the TV problem

We show now how the proximity problem (2.6) can be casted into a simple dual problem, and how we can recover the solution of (2.6) from the solution of this new problem. This can be done by applying Theorem 31.2 in [21], though we present here an alternative, more straightforward proof. To do so, we first perform the derivations for a more general form as follows.

Let $f : \mathbb{R}^n \to \mathbb{R}$ be a twice-differentiable, strictly convex function; let $r : \mathbb{R}^m \to \mathbb{R}$ be a closed, proper convex function, and let $\boldsymbol{B} \in \mathbb{R}^{m\times n}$ be a matrix (can be a linear operator in general, but we ignore that for now). Consider the optimization problem

$$\min_{\boldsymbol{x}} \quad f(\boldsymbol{x}) + \lambda r(\boldsymbol{B}\boldsymbol{x}), \tag{2.8}$$

where $\lambda > 0$ is a scalar. To derive the dual of (2.8) introduce an auxiliary variable $\boldsymbol{z} = \boldsymbol{B}\boldsymbol{x}$, whereby (2.8) becomes

$$\min_{\boldsymbol{x},\boldsymbol{z}} \quad f(\boldsymbol{x}) + \lambda r(\boldsymbol{z}), \qquad \text{s.t.} \quad \boldsymbol{z} = \boldsymbol{B}\boldsymbol{x}. \tag{2.9}$$

Let $\boldsymbol{u}$ be the dual-variables corresponding to the equality constraint. Then, the Lagrangian to (2.9) is

$$L(\boldsymbol{x}, \boldsymbol{z}, \boldsymbol{w}) = f(\boldsymbol{x}) + \lambda r(\boldsymbol{z}) + \boldsymbol{w}^T(\boldsymbol{B}\boldsymbol{x} - \boldsymbol{z}). \tag{2.10}$$

The dual function $g(\boldsymbol{w}) = \inf_{\boldsymbol{x},\boldsymbol{z}} L(\boldsymbol{x}, \boldsymbol{z}, \boldsymbol{w})$, so that

$$g(\boldsymbol{w}) = \left(\inf_{\boldsymbol{x}} f(\boldsymbol{x}) + \boldsymbol{w}^T\boldsymbol{B}\boldsymbol{x}\right) + \left(\inf_{\boldsymbol{z}} -\boldsymbol{w}^T\boldsymbol{z} + \lambda r(\boldsymbol{z})\right). \tag{2.11}$$

Both infima in (2.11) may be written as suprema, so that

$$g(\boldsymbol{w}) = \big(-\sup_{\boldsymbol{x}} -\boldsymbol{w}^T\boldsymbol{B}\boldsymbol{x} - f(\boldsymbol{x})\big) - \big(\sup_{\boldsymbol{z}} \boldsymbol{w}^T\boldsymbol{z} - \lambda r(\boldsymbol{z})\big), \tag{2.12}$$

which are nothing but the Fenchel-duals[1] of f and (λr). Thus we obtain

$$g(\boldsymbol{w}) = -f^*(-\boldsymbol{B}^T\boldsymbol{w}) - \lambda r^*(\lambda^{-1}\boldsymbol{w}). \tag{2.13}$$

Now, in our case we have $f(\boldsymbol{x}) = \frac{1}{2}\|\boldsymbol{x}-\boldsymbol{y}\|_2^2$, and $r(\boldsymbol{x}) = \|\boldsymbol{x}\|_p$. Then $r^*(\boldsymbol{w}) = \delta_{\|\boldsymbol{w}\|_q \leq 1}$, the indicator function for dual norm q, which is defined as $\frac{1}{p}+\frac{1}{q}=1$. Thus, (2.13) simplifies in this case to

$$g(\boldsymbol{w}) = \begin{cases} -\frac{1}{2}\boldsymbol{w}^T\boldsymbol{B}\boldsymbol{B}^T\boldsymbol{w} + \boldsymbol{w}^T\boldsymbol{B}\boldsymbol{y} & \text{where } \|\boldsymbol{w}\|_q \leq \lambda, \\ -\infty & \text{otherwise.} \end{cases} \tag{2.14}$$

Note: completing the square allows us to write the dual-optimization problem as

$$\min_{\boldsymbol{w}} \quad \tfrac{1}{2}\|\boldsymbol{B}^T\boldsymbol{w} - \boldsymbol{y}\|_2^2 \quad \text{s.t. } \|\boldsymbol{w}\|_q \leq \lambda. \tag{2.15}$$

Which is, in fact, an instance of the so-called Trust Region Subproblem, in which we optimize a local quadratic model of some function subject to not leaving a trust region defined by an Lq-norm. This kind of problems have been studied in depth in the literature, and as we shall see, standard algorithms exist to find a solution efficiently.

Once the solution w^* to (2.15) has been found we can recover its related primal solution x^* by noting that, by the Karush–Kuhn–Tucker optimality conditions, the gradient of the Lagrangian w.r.t. to the primal variables must be 0 at the optimum. In particular

$$\begin{aligned} \frac{\partial L(\boldsymbol{x},\boldsymbol{z},\boldsymbol{w})}{\partial \boldsymbol{x}} &= 0, \\ \frac{\partial}{\partial \boldsymbol{x}}\tfrac{1}{2}\|\boldsymbol{x}-\boldsymbol{y}\|_2^2 + \boldsymbol{B}^T\boldsymbol{w} &= 0, \\ \boldsymbol{x} &= \boldsymbol{y} - \boldsymbol{B}^T\boldsymbol{w}, \end{aligned}$$

therefore being able to recover the primal solution easily. What is more, we can also compute the dual gap with this by noting that for $\boldsymbol{x}$ and $\boldsymbol{w}$ feasible solutions of the primal and dual problems we have

$$\begin{aligned} \text{gap}(\boldsymbol{x},\boldsymbol{u}) &= \tfrac{1}{2}\|\boldsymbol{x}-\boldsymbol{y}\|_2^2 + \lambda\|\boldsymbol{B}\boldsymbol{x}\|_p - (-\tfrac{1}{2}\|\boldsymbol{B}^T\boldsymbol{w}\|_2^2 + \boldsymbol{w}^T\boldsymbol{B}y), \\ &= \lambda\|\boldsymbol{B}\boldsymbol{x}\|_p - \boldsymbol{w}^T\boldsymbol{B}y + \|\boldsymbol{B}^T\boldsymbol{w}\|_2^2, \\ &= \lambda\|\boldsymbol{B}\boldsymbol{x}\|_p - \boldsymbol{w}^T\boldsymbol{B}y + \boldsymbol{w}^T\boldsymbol{B}\boldsymbol{B}^T\boldsymbol{w}, \\ &= \lambda\|\boldsymbol{B}\boldsymbol{x}\|_p - \boldsymbol{w}^T\boldsymbol{B}(\boldsymbol{y} - \boldsymbol{B}^T\boldsymbol{w}), \\ &= \lambda\|-\boldsymbol{B}\boldsymbol{x}\|_p + \boldsymbol{w}^T(-\boldsymbol{B}\boldsymbol{x}), \end{aligned}$$

which is fast to compute once $-\boldsymbol{B}\boldsymbol{x}$ is known. The dual gap is a good measure of the quality of an approximate solution $\tilde{\boldsymbol{w}}$ of the dual, thus being useful as a stopping criterion for an optimization algorithm.

2.3 Structure induced by the TV operator

The special structure of matrices $\boldsymbol{D}$ and $\boldsymbol{B}$ allows to perform very fast products and factorizations, not only because of their sparsity, but also because of the way their non-zero entries are distributed. Taking advantage of these facts will prove to be vital for the implementation of an efficient algorithm.

In the 1-dimensional case the matrix $\boldsymbol{D}$ presents itself as a bidiagonal matrix, which allows to compute matrix-vector products in linear time as follows

$$\boldsymbol{D}\boldsymbol{v} = \boldsymbol{v}_{2:N} - \boldsymbol{v}_{1:N-1}, \quad \boldsymbol{D}^T\boldsymbol{v} = \begin{bmatrix} 0 \\ \boldsymbol{v} \end{bmatrix} - \begin{bmatrix} \boldsymbol{v} \\ 0 \end{bmatrix},$$

[1] Fenchel-dual is defined as: $f^*(\boldsymbol{w}) = \sup_{\boldsymbol{x}} \boldsymbol{w}^T\boldsymbol{x} - f(\boldsymbol{x})$

where $\boldsymbol{v}_{i:j}$ denotes the vector formed by the entries $\boldsymbol{v}_i$ to $\boldsymbol{v}_j$ (like in MATLAB notation). Furthermore, the matrices $\boldsymbol{D}\boldsymbol{D}^T$ and $\boldsymbol{D}^T\boldsymbol{D}$ present a tridiagonal structure, which allow for an efficent Cholesky-like factorization in the form $\boldsymbol{M} = \boldsymbol{L}\boldsymbol{\Delta}\boldsymbol{L}^T$. Also methods for fast eigendecomposition for $\boldsymbol{D}\boldsymbol{D}^T$ are available. We will make use of these fast factorizations in our proposed algorithms. Technical details about how to perform them are presented in Appendix A.

Regarding the matrix $\boldsymbol{B}$ used in the 2-dimensional case, it can be nicely rewritten in terms of $\boldsymbol{D}$ and the Kronecker product operator as

$$\boldsymbol{B} = \begin{pmatrix} \boldsymbol{I}_N & \otimes & \boldsymbol{D} \\ \boldsymbol{D} & \otimes & \boldsymbol{I}_N \end{pmatrix}, \tag{2.16}$$

with $\boldsymbol{I}_N$ the $N \times N$ identity matrix. The Kronecker product hence allows to express $\boldsymbol{B}$ compactly, while featuring also a series of nice properties [24]. In our application the most useful of these properties is

$$\boldsymbol{Y} = \boldsymbol{C}\boldsymbol{X}\boldsymbol{B}^T \equiv \boldsymbol{y} = (\boldsymbol{B} \otimes \boldsymbol{C})\boldsymbol{x}, \tag{2.17}$$

with $\boldsymbol{y} = vec(\boldsymbol{Y})$ and $\boldsymbol{x} = vec(\boldsymbol{X})$. In other words, we are able to transform an equation system expressed through Kronecker products into a matrix equation system. This is of immediate use for computing matrix-vector products involving $\boldsymbol{B}$, as

$$\boldsymbol{B}\boldsymbol{v} = \begin{bmatrix} \boldsymbol{D}\boldsymbol{V} \\ \boldsymbol{V}\boldsymbol{D}^T \end{bmatrix}, \quad \boldsymbol{B}^T\boldsymbol{w} = \boldsymbol{D}^T\boldsymbol{W}_1 + \boldsymbol{W}_2\boldsymbol{D}, \tag{2.18}$$

with $\boldsymbol{v} \in \mathbb{R}^{N^2}$, $\boldsymbol{w} \in \mathbb{R}^{2(N-1)N}$, $\boldsymbol{v} = vec(\boldsymbol{V}) \in \mathbb{R}^{N \times N}$, $\boldsymbol{w} = \begin{bmatrix} vec(\boldsymbol{W}_1) \\ vec(\boldsymbol{W}_2) \end{bmatrix}$, $\boldsymbol{W}_1 \in \mathbb{R}^{(N-1)\times N}$, $\boldsymbol{W}_2 \in \mathbb{R}^{N \times (N-1)}$. This saves the need for maintaining an explicit representation for $\boldsymbol{B}$ in memory.

We shall use these structures properties to guarantee fast computations in the methods we will introduce next.

3 1-dimensional Total Variation solvers

3.1 Alternating-Direction Method of Multipliers solver

The primal problem can be solved directly by using the so-called Alternating-Direction Method of Multipliers (ADMM), [8]. This method works on objective functions in the form

$$\min_{\boldsymbol{x},\boldsymbol{b}} f(\boldsymbol{x}) + g(\boldsymbol{b}) \quad \text{s.t.} \quad \boldsymbol{L}\boldsymbol{x} = \boldsymbol{b},$$

which is our case with $f(\boldsymbol{x}) = \frac{1}{2}\|\boldsymbol{x} - \boldsymbol{y}\|_2^2$, $g(\boldsymbol{b}) = \lambda\|\boldsymbol{b}\|_p$ and $\boldsymbol{L} = \boldsymbol{D}$. The method proceeds by alternatively computing the proximity operator of f and g, eventually converging to an optimum of the problem. The general procedure is shown in Algorithm 1, where $prox_f^{\boldsymbol{L}}(\boldsymbol{y}) = \arg\min_{\boldsymbol{x}} f(\boldsymbol{x}) + \frac{1}{2}\|\boldsymbol{L}\boldsymbol{x} - \boldsymbol{y}\|_2^2$.

Algorithm 1 Alternating-Direction Method of Multipliers

Inputs: functions $f(\boldsymbol{x})$, $g(\boldsymbol{b})$, matrix $\boldsymbol{L}$, augmented Lagrangian index $\gamma > 0$, stopping tolerance τ.
Initialization: set initial values for $\boldsymbol{p}$, $\boldsymbol{z}$.
while stopping criterion $> \tau$ **do**
  $\boldsymbol{x} = prox_{\gamma f}^{\boldsymbol{L}}(\boldsymbol{p} - \boldsymbol{z})$
  $\boldsymbol{s} = \boldsymbol{L}\boldsymbol{x}$
  $\boldsymbol{p} = prox_{\gamma \boldsymbol{g}}(\boldsymbol{s} + \boldsymbol{z})$
  $\boldsymbol{z} = \boldsymbol{z} + \boldsymbol{s} - \boldsymbol{p}$
end while

The speed of the algorithm depends heavily on how efficiently can we solve the proximity operators for f and g. Luckily, in the problem at hand they result to be solvable in closed form or at a linear cost, as on the one hand

$$
\begin{aligned}
prox^{\boldsymbol{L}}_{\gamma f}(\boldsymbol{p}-\boldsymbol{z}) &= \arg\min_{\boldsymbol{x}} \gamma f(\boldsymbol{x}) + \tfrac{1}{2}\|\boldsymbol{L}\boldsymbol{x}-\boldsymbol{p}+\boldsymbol{z}\|_2^2, \\
&= \arg\min_{\boldsymbol{x}} \gamma \tfrac{1}{2}\|\boldsymbol{x}-\boldsymbol{y}\|_2^2 + \tfrac{1}{2}\|\boldsymbol{L}\boldsymbol{x}-\boldsymbol{p}+\boldsymbol{z}\|_2^2. \\
0 &= \frac{\partial}{\partial \boldsymbol{x}}, \\
&= \gamma(\boldsymbol{x}-\boldsymbol{y}) + \boldsymbol{D}^T(\boldsymbol{D}\boldsymbol{x}-\boldsymbol{p}+\boldsymbol{z}), \\
(\gamma\boldsymbol{I}+\boldsymbol{D}^T\boldsymbol{D})\boldsymbol{x} &= \gamma\boldsymbol{y} + \boldsymbol{D}^T(\boldsymbol{p}-\boldsymbol{z}),
\end{aligned}
$$

and so the proximity operator can be computed by solving the tridiagonal equation system defined by $\gamma\boldsymbol{I}+\boldsymbol{D}^T\boldsymbol{D}$, which is singular for a large enough $\gamma > 0$. On the other hand we have

$$
\begin{aligned}
prox_{\gamma \boldsymbol{g}}(\boldsymbol{s}+\boldsymbol{z}) &= \arg\min_{\boldsymbol{x}} \tfrac{1}{2}\|\boldsymbol{x}-\boldsymbol{s}-\boldsymbol{z}\|_2^2 + \gamma g(\boldsymbol{x}), \\
&= \arg\min_{\boldsymbol{x}} \tfrac{1}{2}\|\boldsymbol{x}-\boldsymbol{s}-\boldsymbol{z}\|_2^2 + \gamma\lambda\|\boldsymbol{x}\|_p, \\
&= prox_{\gamma\lambda\|\cdot\|_p}(s+z),
\end{aligned}
$$

which essentially is the proximity operator for a standard Lp norm. It has been shown that for the L1 and L2 cases this operator can be computed in closed form, while for the L∞ case a solution can be found in linear time [8]. For other choices of p no efficient algorithm exists up to our knowledge. Therefore the ADMM method will only be useful when using the aforementioned norms.

Taking all these considerations into account, a modified algorithm for TV-Lp norm problem is shown as Algorithm 2. As a stopping criterion we measure the absolute amount of change in $\boldsymbol{x}$, as $\|\boldsymbol{x}^t - \boldsymbol{x}^{t-1}\|_1$.

Algorithm 2 Alternating-Direction Method of Multipliers for TV-Lp norm

Inputs: reference data $\boldsymbol{y}$, augmented Lagrangian index $\gamma > 0$, stopping tolerance τ, norm order p, penalty term λ.
Initialization: set initial values for $\boldsymbol{p}$, $\boldsymbol{z}$.
Precompute tridiagonal factorization of $(\gamma\boldsymbol{I}+\boldsymbol{D}^T\boldsymbol{D})$. (LAPACK: DPTTRF)
while stopping criterion $> \tau$ **do**
  Solve tridiagonal system $(\gamma\boldsymbol{I}+\boldsymbol{D}^T\boldsymbol{D})\boldsymbol{x} = \gamma\boldsymbol{y} + \boldsymbol{D}^T(\boldsymbol{p}-\boldsymbol{z})$. (LAPACK: DPTTRS)
  $\boldsymbol{s} = \boldsymbol{D}\boldsymbol{x}$
  $\boldsymbol{p} = prox_{\gamma\lambda\|\cdot\|_p}(s+z)$
  $\boldsymbol{z} = \boldsymbol{z} + \boldsymbol{s} - \boldsymbol{p}$
end while

3.2 TV-L1 case

We now introduce two algorithms to solve (2.15) efficiently for the case where $p = 1$ (TV-L1 norm). First note that the dual norm for this case is $q = \infty$, which is equivalent to impose a lower and upper bound on every entry of w,

$$
\begin{aligned}
\min_{\boldsymbol{w}} \quad & \frac{1}{2}||\boldsymbol{D}^T\boldsymbol{w}-\boldsymbol{y}||_2^2 \\
\text{s.t.} \quad & -\lambda \leq \boldsymbol{w} \leq \lambda,
\end{aligned}
\tag{3.1}
$$

which is a very simple box-constrained quadratic problem. Several methods are available in the literature to solve this class of problems efficiently, like TRON [18], L-BFGS-B [6], Projected Newton [4] or SBB [17]. Here we will adapt the last two ones.

Algorithm 3 Projected Newton for box constraints

Inputs: objective function $f(x)$, box constraints $l \leq x \leq u$, initial guess x^0, comparison tolerance ϵ, stopping tolerance τ, Armijo parameters β, σ.
Definitions: $([x]_P)_i = \min\{\max\{x_i, l_i\}, u_i\}$, orthogonal projection of x onto feasible set.
Initialization $x = x^0$.
while stopping criterion $> \tau$ **do**
  Compute gradient at x: $\nabla f(x)$.
  Identify set of active constraints
  $I = \{i | (u_i - \epsilon \leq x, \nabla f(x)_i \leq -\epsilon) \text{or} (l_i + \epsilon \geq x, \nabla f(x)_i \geq \epsilon)\}$
  Construct modified Hessian
  $H_{ij} = \begin{cases} 0 & \text{if } i \in I \text{ or } j \in I, i \neq j \\ \frac{\partial f(x)}{\partial x_i \partial x_j} & \text{else} \end{cases}$
  Compute updating direction: $d = H^{-1} \nabla f(x)$.
  Armijo rule: find smallest integer $m >= 0$ such that
  $f(x) - f([x - \beta^m d]_P) \geq \sigma(\beta^m \sum_{i \notin I} \nabla f(x)_i d_i + \sum_{i \in I} \nabla f(x)_i (x_i - [x_i - \beta^m d_i]_P)$.
  Update position: $x' = [x - \beta^m d]_P$.
end while

3.2.1 Projected Newton

Algorithm 3 shows the general procedure of a Projected Newton method [4] in its box constrained variant. We will adapt this method to the special case in which the objective function is quadratic in the form $f(x) = x^T Ax + b \cdot x + c$ and A is a tridiagonal symmetric matrix with strictly positive diagonal entries α and sub-diagonal entries β. Problem 3.1 is a special instance of this with $A = DD^T$, which produces $\alpha_i = 2 \; \forall i$ and $\beta_i = -1 \; \forall i$. We will see that by adapting Projected Newton to this particular structure we are able to run every iteration of the algorithm in linear time. To do so, note first that we can rearrange the rows of the modified Hessian H in a way that we have the block matrix

$$H = \begin{pmatrix} \frac{\partial f(x)}{\partial x_1 \partial x_1} & & & & & \\ & \frac{\partial f(x)}{\partial x_2 \partial x_2} & & & & \\ & & \ddots & & & \\ & & & \frac{\partial f(x)}{\partial x_i \partial x_i} & \cdots & \frac{\partial f(x)}{\partial x_i \partial x_N} \\ & & & \vdots & \ddots & \vdots \\ & & & \frac{\partial f(x)}{\partial x_N \partial x_i} & \cdots & \frac{\partial f(x)}{\partial x_N \partial x_N} \end{pmatrix} = \begin{pmatrix} diag(A_{I,I}) & \\ & A_{C,C} \end{pmatrix}$$

where $A_{I,I}$ denotes the matrix formed by the rows/columns regarding variables in the active set I, and $A_{C,C}$ the matrix formed by the rows/columns regarding variables not in I (not active).

Now, it should be realized that as the updating direction is in the form $d = H^{-1} \nabla f(x)$, $d_i = \frac{1}{\alpha_i} \nabla f(x)_i$ for those entries $i \in I$. As $\alpha_i > 0$, this implies that the variables at active contraints are to be updated in a direction pointing outside the feasible set. As the updated point is in the form $[x - \beta^m d]_P$, those $x_i - \beta^m d_i$ will be projected back to their original x_i value. Hence, no update is performed for variables in active constraints, and so we can consider the equivalent update

$$\begin{aligned} x'_N &= [x_N - \beta^m d_N]_P, \\ x'_I &= x_I, \end{aligned}$$

with $d_N = A^{-1}_{N,N} \nabla f(x)_N$. In this way we only need to deal with a reduced update, involving only non-active constraints. To compute $A_{C,C}$ we can make use of the following proposition.

Proposition 1. *Given $A \in \mathcal{R}^{N \times N}$ tridiagonal matrix and an index set C from elements in $[1, N]$, the matrix A_C which results from removing from A the rows and columns which indexes are not in C (equivalently, keeping only the rows and columns with indexes in C) is also tridiagonal. Furthermore, if A is symmetric then the diagonal and sub/superdiagonal entries of A_C are in the form*

$$\begin{aligned}\alpha' &= \alpha_C, \\ \beta' &= (\beta \odot e_{I-1})_I,\end{aligned}$$

where x_C is the vector formed by only the entries of x indexed by elements of C, I is the complementary set of I, $I-1$ stands for the set of indexes of C being substracted 1, e_C is an indicator vector with zeros everywhere except in the entries indexed by C, which are 1, and $\odot$ is the point product.

Proof. The proof for α' is immediate, as α_i only appears in $A_{C,C}$ if $i \in C$. For β', consider first that only one row/column i is to be removed. If $i = 1$ only β_1 does not appear in $A_{C,C}$ from the sub/superdiagonals. Same applies for $i = N$, where only β_{N-1} vanishes. If any other i applies, then both β_{i-1} and β_i are vanish, but note that in the resulting matrix the entries immediately to the right and under $\alpha_i - 1$ will be valued 0. Therefore, what is obtained is essentially the result of setting β_{i-1} to 0 and removing β_i from the resulting β'. Applying this procedure to every $i \in C$ produces the presented formula. □

The tridiagonality property of the reduced Hessian $A_{C,C}$, allows to solve the system $A_{N,N}d_N = \nabla f(x)_N$ for d_C efficiently through the use of the tridiagonal factorization of $A_{N,N}$ (see Appendix A).

Regarding the Armijo rule, we can make use again of the fact that $x'_I = x_I$, and so the rule simplifies down to $f(x) - f([x - \beta^m d]_P) \geq \sigma\beta^m \nabla f(x)_C \cdot d_C$. We can further simplify this by defining $\delta = [x - \beta^m d]_P - x$ and noting that

$$\begin{aligned}f(x) - f(x+\delta) &= f(x) - (f(x) + \frac{1}{2}\delta^T A \delta^T + x^T A \delta^T + b^T \delta) \\ &= -(\frac{1}{2}\delta^T A + x^T A + b^T)\delta \\ &= -(\frac{1}{2}\delta_C^T A_{\bar{I},\bar{I}} + x^T \begin{pmatrix} A_{C,C} \\ A_{I,C} \end{pmatrix} + b_C^T)\delta_C\end{aligned}$$

The non-zero entries of $A_{C,C}$ have already been computed, and multiplying this matrix with any vector can be done in linear time, owing again to this matrix being tridiagonal. However, $A_{I,C}$ has neither been computed, nor it is tridiagonal. Nevertheless it should be realized that its rows contain at most two nonzero entries that can be easily located. This can be seen by noting that for any $i \in I$ the row $A_{i,C}$ cannot contain the α_i term as $i \notin C$, and will only keep β_{i-1} if $i - 1 \in C$, β_i if $i + 1 \in C$. Thus, for each row of $A_{I,C}$ we can decide fastly which entries are non-zero, and are able to multiply it with a vector spending at most two product operations. It should also be realized that the term $x^T \begin{pmatrix} A_{C,C} \\ A_{I,C} \end{pmatrix}$ is indendent of δ_C, and as such we only need to compute it once per iteration, regardless of the number of Armijo rule evaluations needed.

Finally, another point of improvement of the standard Projected Newton is the gradient recomputation. As in this case we have a quadratic function, the gradient can be updated as

$$\begin{aligned}\nabla f(x+\delta) &= \nabla f(x) + A\delta \\ &= \nabla f(x) + \begin{pmatrix} A_{C,C} \\ A_{I,C} \end{pmatrix} \delta_C\end{aligned}$$

where we can use similar strategies to the ones presented for the evaluation of the Armijo rule.

Summing up, the adapted Projection Newton algorithm for quadratic functions with tridiagonal Hessian is shown in Algorithm 4. As a stopping criterion we use the quantity of total violation of the Karush–Kuhn–Tucker optimality conditions, which can be shown to be

$$viol(x) = ||\nabla f(x)_C||_1.$$

Algorithm 4 Projected Newton for tridiagonal Hessian problems

Inputs: objective function in the form $f(x) = x^T Ax + b^T x + c$, A tridiagonal defined by (α, β), box constraints $l \leq x \leq u$, initial guess x^0, comparison tolerance ϵ, stopping tolerance τ, Armijo parameters β, σ.
Definitions: $([x]_P)_i = \min\{\max\{x_i, l_i\}, u_i\}$, orthogonal projection of x onto feasible set.
Initialization $x = x^0$, $\nabla f(x) = Ax + b$.
while stopping criterion $> \tau$ **do**
  Identify set of active constraints
  $I = \{i | (u_i - \epsilon \leq x, \nabla f(x)_i \leq -\epsilon) \text{or} (l_i + \epsilon \geq x, \nabla f(x)_i \geq \epsilon)\}$
  Construct nonzero entries of reduced Hessian: $A_{C,C}$
  Compute tridiagonal factorization: $A_{C,C} = L\Delta L^T$. (LAPACK: DPTTRF)
  Compute updating direction d solving system $A_{C,C} d_C = \nabla f(x)_C$. (LAPACK: DPTTRS)
  Armijo rule: find smallest integer $m >= 0$ such that for $\delta = [x - \beta^m d]_P - x$,
  $-(\frac{1}{2}\delta_C^T A_{C,C} + x^T \begin{pmatrix} A_{C,C} \\ A_{I,C} \end{pmatrix} + b_C^T)\delta_C \geq \sigma\beta^m \nabla f(x)_C \cdot d_C$.
  Update position: $x' = x + \delta$.
  Update gradient: $\nabla f(x') = \nabla f(x) + \begin{pmatrix} A_{C,C} \\ A_{I,C} \end{pmatrix} \delta_C$.
end while

3.2.2 Subspace BB

The Subspace BB method [17] can be regarded as an extension of the Barzilai and Borwein method for the nonnegative constrained case [3], which combines Barzilai-Borwein stepsizes with a projected gradient approach. Being a first-order method (no Hessian information is required), the matrix $\boldsymbol{D}$ is only used through matrix-vector products, and so each iteration can be run in linear time owing to the bidiagonality of $\boldsymbol{D}$.

An outline of the SBB algorithm adapted to the problem at hand is shown as Algorithm 5. At each iteration the set of active constraints is identified, a gradient $\nabla \tilde{f}$ confined to the subspace defined by the non-active contraints is constructed, and finally a BB-like stepsize is performed in the direction of the (full) gradient, projecting the result back into the feasible set.

Even though the original SBB method was designed to deal only with nonnegativity constraints, it can be easily extended to a box constrained setting. Furthermore in the original SBB algorithm every M iterations a sufficient descent check is performed, decreasing a stepsize multiplier if this check is not met; we have removed this check in our implementation, as in all the experiments performed the algorithm managed to achieve convergence without resorting to it.

Algorithm 5 Subspace BB for TV-L1

Inputs: reference data $\boldsymbol{y}$, penalty parameter λ, initial guess $\boldsymbol{x}^0$, stopping tolerance τ.
Definitions: $([\boldsymbol{x}]_P)_i = \min\{\max\{\boldsymbol{x}_i, -\lambda\}, \lambda\}$, orthogonal projection of $\boldsymbol{x}$ onto feasible set.
Initialization $\boldsymbol{x} = \boldsymbol{x}^0$.
while stopping criterion $> \tau$ **do**
  Compute gradient $\nabla f = \boldsymbol{D}(\boldsymbol{D}^T \boldsymbol{x} - \boldsymbol{y})$
  Identify set of active constraints
  $I = \{i | (-\lambda == \boldsymbol{x}_i, \nabla f(x)_i \leq 0) \text{or} (\lambda == \boldsymbol{x}_i, \nabla f(x)_i \geq 0)\}$
  Construct subspace gradient: $[\nabla \tilde{f}]_i = [\nabla f]_i$ if $i \notin I$, 0 otherwise.
  if iteration number is odd **then**
    Compute stepsize: $\alpha = \frac{\|\nabla \tilde{f}\|_2^2}{\langle \nabla \tilde{f}, \boldsymbol{D}\boldsymbol{D}^T \nabla \tilde{f} \rangle}$.
  else
    Compute stepsize: $\alpha = \frac{\langle \nabla \tilde{f}, \boldsymbol{D}\boldsymbol{D}^T \nabla \tilde{f} \rangle}{\|\boldsymbol{D}\boldsymbol{D}^T \nabla \tilde{f}\|_2^2}$.
  end if
  $\boldsymbol{x} = [\boldsymbol{x} - \alpha \nabla f]_+$.
end while

3.3 TV-L2 case

When $p = 2$ in (2.3) the dual problem also fancies a $q = 2$ norm, and so the feasible space is an euclidean ball of radius λ.

$$\min_{\boldsymbol{w}} \quad \tfrac{1}{2}\|\boldsymbol{D}^T\boldsymbol{w} - \boldsymbol{y}\|_2^2 \quad \text{s.t. } \|\boldsymbol{w}\|_2 \leq \lambda. \tag{3.2}$$

This is the most common situation in Trust Region Subproblems, and as such a large number of solvers are available in the literature, like LSTRS [22], IP-SSM [11] and the method proposed by Moré and Sorensen [19]. A priori LSTRS is the best option, as it is a well-stablished method and good public implementations are available. However, the same LSTRS authors recommend to use Moré and Sorensen's method for the particular case in which the Cholesky decomposition of $\boldsymbol{A} + \delta\boldsymbol{I}$ can be computed efficiently, $\boldsymbol{A}$ the Hessian of the objective function and $\delta \geq 0$. As we will see, this is the situation in our particular problem, and so we will adapt this method for our case.

The motivation behind Moré and Sorensen's method can be easily explained by analyzing the KKT conditions of the problem at hand. We will do this first for a general quadratic objective function in the form $\boldsymbol{g}^T\boldsymbol{w} + \frac{1}{2}\boldsymbol{w}^T\boldsymbol{A}\boldsymbol{w}$, as in [19]. The Lagrangian of the problem can be written as

$$L(\boldsymbol{w}, \alpha) = \boldsymbol{g}^T\boldsymbol{w} + \tfrac{1}{2}\boldsymbol{w}^T\boldsymbol{A}\boldsymbol{w} + \alpha(\|\boldsymbol{w}\|_2^2 - \lambda^2),$$

which at the optimum must meet

$$\begin{aligned} \frac{\partial L(\boldsymbol{w}, \alpha)}{\partial \boldsymbol{w}} &= 0, \\ \boldsymbol{g} + \boldsymbol{A}\boldsymbol{w} + \alpha\boldsymbol{w} &= 0, \\ (\boldsymbol{A} + \alpha\boldsymbol{I})\boldsymbol{w} &= -\boldsymbol{g}. \end{aligned}$$

Also, as a Lagrange multiplier, $\alpha \geq 0$ and due to complementarity constraints $\alpha(\|\boldsymbol{w}\|_2^2 - \lambda^2) = 0$. Joining all these constraints, we can rewrite the problem as a Constraint Satisfaction Problem in the form

$$\text{find}_{\boldsymbol{w}} \quad \text{s.t.} \begin{cases} (\boldsymbol{A} + \alpha\boldsymbol{I})\boldsymbol{w} = -\boldsymbol{g} \\ \alpha \geq 0 \\ \alpha(\|\boldsymbol{w}\|_2 - \lambda) = 0 \end{cases}. \tag{3.3}$$

Two situations might happen in the solution of (3.3): either $\boldsymbol{w}$ is the interior of the feasible set ($\|\boldsymbol{w}\|_2 \leq \lambda$) or it is in the boundary ($\|\boldsymbol{w}\|_2 = \lambda$). If $\boldsymbol{w}$ is in the interior we necessarily have $\alpha = 0$, and then the solution is given by finding a solution to the system $\boldsymbol{A}\boldsymbol{w} = -\boldsymbol{g}$. The argument is also valid backwise: if the solution w of the system $\boldsymbol{A}\boldsymbol{w} = -\boldsymbol{g}$ is in the interior of the feasible set, then w solves (3.3). Thus, we can easily check whether the solution is in the interior, and if it is not, we can safely assume $\|\boldsymbol{w}\|_2 = \lambda$ and work only in the boundary case, which we address in the following.

The boundary condition is the key to solving the problem. To see this, suppose for now that we have α s.t. $(\boldsymbol{A} + \alpha\boldsymbol{I})$ is definite postive. Then we have

$$\begin{aligned} \lambda^2 &= \|\boldsymbol{w}\|_2^2 \\ &= \|-(\boldsymbol{A} + \alpha\boldsymbol{I})^{-1}\boldsymbol{g}\|_2^2 \\ &= \boldsymbol{g}^T(\boldsymbol{A} + \alpha\boldsymbol{I})^{-1^T}(\boldsymbol{A} + \alpha\boldsymbol{I})^{-1}\boldsymbol{g} \end{aligned}$$

We can now use the eigendecomposition $\boldsymbol{A} = \boldsymbol{Q}\Lambda\boldsymbol{Q}^T$, with $\Lambda = \text{diag}(\lambda_1, \ldots, \lambda_N)$ diagonal matrix of eigenvalues sorted in increasing order and $\boldsymbol{Q}$ orthogonal matrix of eigenvectors,

$$
\begin{aligned}
\lambda^2 &= \boldsymbol{g}^T(\boldsymbol{Q}\Lambda\boldsymbol{Q}^T + \alpha\boldsymbol{I})^{-1^T}(\boldsymbol{Q}\Lambda\boldsymbol{Q}^T + \alpha\boldsymbol{I})^{-1}\boldsymbol{g}, \\
&= \boldsymbol{g}^T(\boldsymbol{Q}(\Lambda + \alpha\boldsymbol{I})\boldsymbol{Q}^T)^{-1^T}(\boldsymbol{Q}(\Lambda + \alpha\boldsymbol{I})\boldsymbol{Q}^T)^{-1}\boldsymbol{g}, \\
&= \boldsymbol{g}^T\boldsymbol{Q}(\Lambda + \alpha\boldsymbol{I})^{-1}\boldsymbol{Q}^T\boldsymbol{Q}(\Lambda + \alpha\boldsymbol{I})^{-1}\boldsymbol{Q}^T\boldsymbol{g}, \\
&= \boldsymbol{g}^T\boldsymbol{Q}(\Lambda + \alpha\boldsymbol{I})^{-1}(\Lambda + \alpha\boldsymbol{I})^{-1}\boldsymbol{Q}^T\boldsymbol{g}, \\
&= \sum_i \frac{\gamma_i^2}{(\lambda_i + \alpha)^2}, \\
&= \psi(\alpha)
\end{aligned}
$$

with $\gamma_i = \boldsymbol{g}^T\boldsymbol{Q}_i$, $\boldsymbol{Q}_i$ i-th column of $\boldsymbol{Q}$. Thus, the norm of the solution vector is essentially a nonlinear function ψ of α. Therefore, to solve the problem we need to find an α s.t. $\psi(\alpha) = \lambda^2$. However, the nonlinearity of ψ together with the existence of second-order poles at the points where $\alpha = -\lambda_i$ might make this task difficult. In [19] the following strategy is recommended: locate the rightmost pole, which is the one associated with the smallest eigenvalue λ_1, and try to find an α that satisfies the constraint in the "poles-free" interval $\alpha \in (-\lambda_1, \infty)$. This approach is reasonable, as $\lim_{\alpha\to-\lambda_1} = \infty$, $\lim_{\alpha\to\infty} = 0$ and by continuity $\exists\ \alpha^*$ s.t. $\psi(\alpha^*) = \lambda^2$. Unfortunately, this argument might fail, in particular if $\gamma_1 = 0$ then α^* might be located in the interval $(-\infty, -\lambda_1]$. This is the so-called "hard-case", and to deal with it additional safeguarding measures need to be taken in Moré and Sorensen's algorithm, also resulting in slower convergence. Fortunately, in the problem at hand the hard case cannot take place, namely,

Proposition 2. *In a problem in the form (3.2), Moré and Sorensen's "hard case" cannot happen.*

Proof. The rows of the Hessian of (3.2), $\boldsymbol{A} = \boldsymbol{D}\boldsymbol{D}^T$, are linearly independent, thus rendering $\boldsymbol{A}$ positive definite. As such, every one of its eigenvalues $\lambda_i > 0$. Consequently, the rightmost pole is located at $-\lambda_1 < 0$, and as by the KKT conditions (3.3) $\alpha \geq 0$, then the solution $\alpha^* \in [0, \infty)$. □

Corollary 3. *It follows that the maximum value of the $\psi(\alpha)$ function is attained at $\alpha = 0$. Therefore, for any $\lambda > \lambda_{MAX} = \psi(0)$ the solution $\boldsymbol{w}$ is always the same point in the interior of the feasible set. Thus, the range of λ values generating different models is bounded by $\lambda \in [0, \lambda_{MAX}]$.*

Therefore, we do not need to worry about the hard case, and can just look for α^* in the pole-free, smooth interval $[0, \infty)$. To do so, Moré and Sorensen recommend to use Newton's method for locating the roots of the function

$$
\phi(\alpha) = \frac{1}{\lambda} - \frac{1}{\|w\|_2} = 0,
$$

which is almost linear in the search interval, guaranteeing very fast convergence. Recall that Newton's method involves updating our estimate of the function's root as $\alpha' = \alpha - \frac{\phi(\alpha)}{\phi'(\alpha)}$ with $\phi'(\alpha)$ derivative of $\phi(\alpha)$. By using $\|w\|_2 = \sum_i \frac{\gamma_i^2}{(\lambda_i+\alpha)^2}$ we can see that this derivative is

$$
\phi'(\alpha) = -\frac{1}{\|w\|_2^{\frac{3}{2}}} \sum_i \gamma_i^2 \frac{1}{(\lambda_i + \alpha)^2},
$$

and so the update would be

$$
\alpha' = \alpha + \frac{\|w\|_2^2}{\sum_i \frac{\gamma_i^2}{(\lambda_i+\alpha)^3}} \cdot \frac{\|w\|_2 - \lambda}{\lambda}.
$$

This can be further simplified by computing the Cholesky decomposition $\boldsymbol{A} + \alpha\boldsymbol{I} = \boldsymbol{R}^T\boldsymbol{R}$, noting that $\boldsymbol{R}^T\boldsymbol{R}\boldsymbol{w} = -\boldsymbol{g}$, and defining $\boldsymbol{R}^T\boldsymbol{q} = \boldsymbol{w}$, as

$$
\begin{aligned}
\|\boldsymbol{w}\|_2^2 &= \boldsymbol{q}^T\boldsymbol{R}\boldsymbol{R}^T\boldsymbol{q}, \\
&= \boldsymbol{q}^T\boldsymbol{Q}(\Lambda + \alpha\boldsymbol{I})\boldsymbol{Q}^T\boldsymbol{q}, \\
&= \boldsymbol{\gamma}^T(\Lambda + \alpha\boldsymbol{I})^{-2}\boldsymbol{\gamma}^T,
\end{aligned}
$$

thus

$$\begin{aligned} \boldsymbol{q} &= \boldsymbol{Q}(\Lambda + \alpha \boldsymbol{I})^{-\frac{3}{2}} \boldsymbol{\gamma}, \\ \|\boldsymbol{q}\|_2^2 &= \boldsymbol{\gamma}^T (\Lambda + \alpha \boldsymbol{I})^{-\frac{3}{2}^T} \boldsymbol{Q}^T \boldsymbol{Q} (\Lambda + \alpha \boldsymbol{I})^{-\frac{3}{2}} \boldsymbol{\gamma}, \\ &= \sum_i \frac{\gamma_i^2}{(\lambda_i + \alpha)^3}, \end{aligned}$$

and so the update is more compactly expressed as

$$\alpha' = \alpha + \frac{\|\boldsymbol{w}\|_2^2}{\|\boldsymbol{q}\|_2^2} \cdot \frac{\|\boldsymbol{w}\|_2 - \lambda}{\lambda}.$$

The procedure as a whole is presented in Algorithm 6. As α is initialized as 0, the first iteration will check whether the optimal w^* is in the interior of the feasible set, stopping if that is the case or moving on to an $\alpha > 0$ otherwise. As a stopping criterion we can use the distance to the boundary $|\|w\|_2^2 - \lambda|$, except for that first iteration in which we will also stop if w is in the interior, i.e. $\|w\|_2^2 \leq \lambda$. Note that this algorithm can be very costly for a general $\boldsymbol{A} + \alpha \boldsymbol{I}$, as Cholesky decomposition is an $O(N^3)$ procedure for the general case. Fortunately in our case the tridiagonality of $\boldsymbol{A}$ allows to do this through a tridiagonal factorization (see Appendix A). In practice this method is able to converge in a very small number of iterations, thus scaling linearly with the size of the problem.

Algorithm 6 Moré-Sorensen algorithm for definite positive Hessian (optimized for tridiagonal)

Inputs: objective function in the form $f(w) = \frac{1}{2}\boldsymbol{w}^T \boldsymbol{A}\boldsymbol{w} + \boldsymbol{g}^T \boldsymbol{w}$, $\boldsymbol{A}$ definite positive, trust region parameter λ, stopping tolerance τ.
Initialization $\alpha = 0$
while stopping criterion $> \tau$ **do**
  Compute Cholesky decomposition $\boldsymbol{A} + \alpha \boldsymbol{I} = \boldsymbol{R}^T \boldsymbol{R}$. (LAPACK DPTTRF)
  Obtain $\boldsymbol{w}$ solving system $\boldsymbol{R}^T \boldsymbol{R}\boldsymbol{w} = -\boldsymbol{g}$. (LAPACK DPTTRS)
  Obtain $\boldsymbol{q}$ solving system $\boldsymbol{R}^T \boldsymbol{q} = \boldsymbol{w}$. (LAPACK DGTSV)
  Update α as: $\alpha' = \alpha + \frac{\|\boldsymbol{w}\|_2^2}{\|\boldsymbol{q}\|_2^2} \cdot \frac{\|\boldsymbol{w}\|_2 - \lambda}{\lambda}$
end while

4 2-dimensional Total Variation solvers

4.1 2-dimensional ADMM

Given that the primal problem for the 2-dimensional case has a similar form to the 1-dimensional one, we can apply once again the ADMM method to solve the problem. Note however that the structure of $\boldsymbol{B}$ is noticeably more complex than that of $\boldsymbol{D}$, and so some changes in the algorithm are required.

In principle, the pseudocode presented in Algorithm 2 for the 1-d case could be applied unchanged here. However, when attempting to solve the equation system induced by the matrix $\gamma \boldsymbol{I}_{N^2} + \boldsymbol{B}^T \boldsymbol{B}$ we can no longer rely on tridiagonality, nor precompute a tridiagonal factorization. To make up for this, note that using the Kronecker product properties we can write

$$\begin{aligned} \gamma \boldsymbol{I}_{N^2} + \boldsymbol{B}^T \boldsymbol{B} &= \gamma \boldsymbol{I}_{N^2} + \begin{bmatrix} (\boldsymbol{I}_N \otimes \boldsymbol{D}^T) & (\boldsymbol{D}^T \otimes \boldsymbol{I}_N) \end{bmatrix} \begin{bmatrix} \boldsymbol{I}_N \otimes \boldsymbol{D} \\ \boldsymbol{D} \otimes \boldsymbol{I}_N \end{bmatrix} \\ &= \gamma \boldsymbol{I}_{N^2} + \boldsymbol{I}_N \otimes \boldsymbol{D}^T \boldsymbol{D} + \boldsymbol{D}^T \boldsymbol{D} \otimes \boldsymbol{I}_N \\ &= \gamma (\boldsymbol{I}_N \otimes \boldsymbol{I}_N) + \boldsymbol{I}_N \otimes \boldsymbol{D}^T \boldsymbol{D} + \boldsymbol{D}^T \boldsymbol{D} \otimes \boldsymbol{I}_N \\ &= \boldsymbol{I}_N \otimes (\gamma \boldsymbol{I}_N + \boldsymbol{D}^T \boldsymbol{D}) + \boldsymbol{D}^T \boldsymbol{D} \otimes \boldsymbol{I}_N \end{aligned}$$

Considering now the system we need to solve, we have $\boldsymbol{A}x = \boldsymbol{c}$, where $\boldsymbol{A} = \gamma \boldsymbol{I}_{N^2} + \boldsymbol{B}^T \boldsymbol{B} = \boldsymbol{I}_N \otimes (\gamma \boldsymbol{I}_N + \boldsymbol{D}^T \boldsymbol{D}) + \boldsymbol{D}^T \boldsymbol{D} \otimes \boldsymbol{I}_N$ and $\boldsymbol{c} = \gamma \boldsymbol{y} + \boldsymbol{B}(\boldsymbol{p} - \boldsymbol{z})$. We can now make use of the Kronecker product property (2.17), obtaining

$$(\gamma \boldsymbol{I}_N + \boldsymbol{D}^T \boldsymbol{D})\boldsymbol{X} + \boldsymbol{X}\boldsymbol{D}^T \boldsymbol{D} = \boldsymbol{C}, \tag{4.1}$$

where $\boldsymbol{C}$ is $\boldsymbol{c}$ in matrix form, which can be expressed conveniently using

$$\begin{aligned}
\boldsymbol{c} &= \gamma\boldsymbol{y} + \boldsymbol{B}(\boldsymbol{p}-\boldsymbol{z}) \\
&= \gamma\boldsymbol{y} + \begin{bmatrix}(\boldsymbol{I}_N \otimes \boldsymbol{D}^T) & (\boldsymbol{D}^T \otimes \boldsymbol{I}_N)\end{bmatrix}(\boldsymbol{p}-\boldsymbol{z}) \\
&= \gamma\boldsymbol{y} + \begin{bmatrix}(\boldsymbol{I}_N \otimes \boldsymbol{D}^T) & (\boldsymbol{D}^T \otimes \boldsymbol{I}_N)\end{bmatrix}\begin{bmatrix}\boldsymbol{p}_1 - \boldsymbol{z}_1 \\ \boldsymbol{p}_2 - \boldsymbol{z}_2\end{bmatrix} \\
\boldsymbol{C} &= \gamma\boldsymbol{Y} + \boldsymbol{D}^T(\mathbf{P}_1 - \boldsymbol{Z}_1) + (\mathbf{P_2} - \boldsymbol{Z}_2)\boldsymbol{D},
\end{aligned}$$

where $\boldsymbol{p}_1$ is the vector containing the first half of $\boldsymbol{p}$, $\boldsymbol{p}_2$ the second half, equivalently for $\boldsymbol{z}_1$, $\boldsymbol{z}_2$, and $\mathbf{P}_1$, $\boldsymbol{Z}_1 \in \mathcal{R}^{(N-1)\times N}$, $\mathbf{P}_2$, $\boldsymbol{Z}_2 \in \mathcal{R}^{N\times(N-1)}$ are their respective matrix forms.

The equation system posed in (4.1) is recognised as the well-known Sylvester equation, for which a solution is known since long [2]. Given a Sylvester equation in the form $\boldsymbol{AX} + \boldsymbol{XB} = \boldsymbol{C}$ we can find the solution by first computing the Schur decomposition of matrices $\boldsymbol{A}$ and $\boldsymbol{B}$ as

$$\begin{aligned}
\boldsymbol{A} &= \mathbf{T}\mathbf{U}\mathbf{T}^T \\
\boldsymbol{B} &= \mathbf{Q}\boldsymbol{V}\mathbf{Q}^T,
\end{aligned}$$

where $\mathbf{T}$ and $\mathbf{Q}$ are orthogonal matrices, $\mathbf{U}$ is lower triangular and $\boldsymbol{V}$ upper triangular. A decomposition of this form is always guaranteed to exist. Now the Sylvester equation can be transformed as

$$\begin{aligned}
\boldsymbol{AX} + \boldsymbol{XB} &= \boldsymbol{C} \\
\mathbf{T}\mathbf{U}\mathbf{T}^T\boldsymbol{X} + \boldsymbol{X}\mathbf{Q}\boldsymbol{V}\mathbf{Q}^T &= \boldsymbol{C} \\
\mathbf{U}\mathbf{T}^T\boldsymbol{X} + \mathbf{T}^T\boldsymbol{X}\mathbf{Q}\boldsymbol{V}\mathbf{Q}^T &= \mathbf{T}^T\boldsymbol{C} \\
\mathbf{U}\mathbf{T}^T\boldsymbol{X}\mathbf{Q} + \mathbf{T}^T\boldsymbol{X}\mathbf{Q}\boldsymbol{V} &= \mathbf{T}^T\boldsymbol{C}\mathbf{Q}.
\end{aligned} \tag{4.2}$$

Defining now $\tilde{\boldsymbol{X}} = \mathbf{T}^T\boldsymbol{X}\mathbf{Q}$, $\tilde{\boldsymbol{C}} = \mathbf{T}^T\boldsymbol{C}\mathbf{Q}$ we have the system

$$\mathbf{U}\tilde{\boldsymbol{X}} + \tilde{\boldsymbol{X}}\boldsymbol{V} = \tilde{\boldsymbol{C}}.$$

Making use now of the triangular structure of $\mathbf{U}$, $\boldsymbol{V}$ it is easy to see that the entries of the solution $\tilde{\boldsymbol{X}}$ can be computed as

$$[\tilde{\boldsymbol{X}}]_{ij} = \frac{[\tilde{C}]_{ij} - \sum_{k=1}^{i-1}\mathbf{U}_{ik}[\tilde{\boldsymbol{X}}]_{kj} - \sum_{k=1}^{j-1}[\tilde{\boldsymbol{X}}]_{ik}\boldsymbol{V}_{kj}}{\mathbf{U}_{ii} + \boldsymbol{V}_{jj}}.$$

As for the computation of $[\tilde{\boldsymbol{X}}]_{ij}$ only those entries of $\tilde{\boldsymbol{X}}$ in previous rows/columns are needed, we can safely compute the entries of $\tilde{\boldsymbol{X}}$ one at a time following either a row or column order. Once $\tilde{\boldsymbol{X}}$ has been obtained we can recover our original problem solution $\boldsymbol{X}$ by noting that by the orthogonality of $\mathbf{T}$ and $\mathbf{Q}$ we have $\boldsymbol{X} = \mathbf{T}\tilde{\boldsymbol{X}}\mathbf{Q}^T$.

Turning now our attention back to the equation system (4.1) we need to solve in the ADMM algorithm, the first step would be to compute the Schur decompositions of $\boldsymbol{D}^T\boldsymbol{D}$ and $\gamma\boldsymbol{I}_N + \boldsymbol{D}^T\boldsymbol{D}$. Both matrices are tridiagonal symmetric, and it can be shown [20] that in this case the resulting Schur decomposition is equivalent to the eigenvalue decomposition, where the eigenvectors have been made orthogonal. This fact is advantageous, as we obtain a decomposition $\boldsymbol{D}^T\boldsymbol{D} = \mathbf{T}\mathbf{U}\mathbf{T}^T$ in which $\mathbf{U}$ is a diagonal matrix containing the eigenvalues of $\boldsymbol{D}^T\boldsymbol{D}$. What is more, the eigendecomposition of $\gamma\boldsymbol{I}_N + \boldsymbol{D}^T\boldsymbol{D}$ can be shown to be $\mathbf{T}(\mathbf{U} + \gamma\boldsymbol{I}_N)\mathbf{T}^T$. Thus, we only need to compute the eigendecomposition of the tridiagonal matrix $\boldsymbol{D}^T\boldsymbol{D}$, for which a LAPACK routine exists (DSTEV). Once that is achieved we can obtain $\tilde{X}$ using (4.2), which in our particular case simplifies down to the easier formula.

$$\tilde{\boldsymbol{X}}_{ij} = \frac{\tilde{\boldsymbol{C}}_{ij}}{\mathbf{U}_{ii} + \mathbf{U}_{jj} + \gamma}, \tag{4.3}$$

hence allowing to compute $\tilde{\boldsymbol{X}}_{ij}$ in $\mathrm{O}(N^2)$ time. Unfortunately, computing the Schur decomposition and afterwards performing the operation $\boldsymbol{X} = \mathbf{T}\tilde{\boldsymbol{X}}\mathbf{T}^T$ both involve a cost of order $\mathrm{O}(N^3)$, turning this into the overall iteration

Algorithm 7 Alternating-Direction Method of Multipliers for 2-dimensional TV-Lp norm

Inputs: reference data $\boldsymbol{Y}$, augmented Lagrangian index $\gamma > 0$, stopping tolerance τ, norm order p, penalty term λ.
Initialization: set initial values for $\mathbf{P}_1$, $\mathbf{P}_2$, $\boldsymbol{Z}_1$, $\boldsymbol{Z}_2$.
Precompute Schur decomposition $(\boldsymbol{D}^T\boldsymbol{D}) = \mathbf{T}\mathbf{U}\mathbf{T}^T$. (LAPACK: DSTEV)
while stopping criterion $> \tau$ **do**
  Compute $\boldsymbol{C} = \boldsymbol{D}^T(\mathbf{P}_1 - \boldsymbol{Z}_1) + (\mathbf{P}_2 - \boldsymbol{Z}_2)\boldsymbol{D} + \gamma\boldsymbol{Y}$.
  Compute $\tilde{\boldsymbol{C}} = \mathbf{T}^T\boldsymbol{C}\mathbf{T}$.
  Solve Sylvester equation $\mathbf{U}\tilde{\boldsymbol{X}} + \tilde{\boldsymbol{X}}\mathbf{U} = \tilde{\boldsymbol{C}}$. (using (4.3))
  Recover $\boldsymbol{X} = \mathbf{T}\tilde{\boldsymbol{X}}\mathbf{T}^T$.
  $\boldsymbol{S}_1 = \boldsymbol{D}\boldsymbol{X}$, $\boldsymbol{S}_2 = \boldsymbol{X}\boldsymbol{D}$
  $\begin{bmatrix} \mathbf{P}_1 \\ \mathbf{P}_2 \end{bmatrix} = prox_{\gamma\lambda\|\cdot\|_p}\left(\begin{bmatrix} vec(\boldsymbol{S}_1 + \boldsymbol{Z}_1) \\ vec(\boldsymbol{S}_2 + \boldsymbol{Z}_2) \end{bmatrix}\right)$
  $\boldsymbol{Z}_1 = \boldsymbol{Z}_1 + \boldsymbol{S}_1 - \mathbf{P}_1$, $\boldsymbol{Z}_2 = \boldsymbol{Z}_2 + \boldsymbol{S}_2 - \mathbf{P}_2$
end while

cost of the algorithm. While more efficient methods for operating with the eigenvectors of $\boldsymbol{D}\boldsymbol{D}^T$ can be devised, it is unclear whether this can be ported to the eigenvectors of $\boldsymbol{D}^T\boldsymbol{D}$.

Considering all this we could already write the pseucode of the ADMM method, replacing in Algorithm 2 the lines corresponding to the solution of the linear equation system. However, it will prove profitable to rewrite the full algorithm in terms of matrices instead of vectorized versions of them. This version is presented in Algorithm 7. Note that the original working vectors of the algorithm $\boldsymbol{p}$, $\boldsymbol{z}$ and $\boldsymbol{s}$ have been transformed into couples of matrices containing the first or second half of them. As it can be observed in the Algorithm, this allows to cast vector operations like $\boldsymbol{B}\boldsymbol{x}$ into two separate, fast matrix products $\boldsymbol{S}_1 = \boldsymbol{D}\boldsymbol{X}$, $\boldsymbol{S}_2 = \boldsymbol{X}\boldsymbol{D}^T$, thus saving the need to store $\boldsymbol{B}$ explicitly in memory. Only in the proximity step it is required to transform those matrices into vector form and then reorganize the output back to matrix form.

As already stated, the O(N^3) computational cost of the algorithm is dominated by the multiplications with matrix $\mathbf{T}$ and the Schur decomposition (though this is only computed once). Regarding the number of iterations for convergence, this quantity seems to be highly sensitive to the selected value for the parameter γ, for which the optimal value depends heavily on the size of the input N and the penalty paramter λ. No easy way to choose γ is available up to our knowledge.

4.2 2-dimensional TV-L1 case

Even though Projected Newton stands among the algorithms presented for the 1-dimensional TV-L1 case, unfortunately its applicability is limited when dealing with bidimensional entries. The problem to be solved is

$$\min_{\boldsymbol{w}} \quad \tfrac{1}{2}\|\boldsymbol{B}^T\boldsymbol{w} - \boldsymbol{y}\|_2^2 \quad \text{s.t. } -\lambda \leq \boldsymbol{w} \leq \lambda, \tag{4.4}$$

featuring a Hessian $\boldsymbol{B}\boldsymbol{B}^T$. In spite of this Hessian presenting a nice structure based on Kronecker products, this structure breaks when rows/columns are removed to obtain the reduced Hessian, a necessary step in the Projected Newton algorithm. Without this structure the ability to perform a Newton step in linear time is lost, thus rendering the method inefficient.

This is not the case for the SBB algorithm, which still manages to maintain a good running time per iteration. Algorithm 8 shows an outline of this method adapted for the 2-dimensional case. As expected, the only difference strives in the use of $\boldsymbol{B}$ instead of the 1-dimensional $\boldsymbol{D}$, which can be computed efficiently in matrix form using the properties in (2.18).

4.3 2-dimensional TV-L2 case

As in the 1-dimensional case, when $p = 2$ the problem to solve is nothing but a Trust Region Subproblem, hence making Moré-Sorensen the algorithm of choice. However, in this case the problem to solve,

$$\min_{\boldsymbol{w}} \quad \tfrac{1}{2}\|\boldsymbol{B}^T\boldsymbol{w} - \boldsymbol{y}\|_2^2 \quad \text{s.t. } \|\boldsymbol{w}\|_2 \leq \lambda, \tag{4.5}$$

presents a semidefinite-positive hessian $\boldsymbol{B}\boldsymbol{B}^T$, and thus it contains zero-valued eigenvalues. Recall that the Moré-Sorensen method consisted in finding the root of the function $\psi(\alpha)$, which presented second-order poles at the $-\lambda_i$

Algorithm 8 Subspace BB for 2-dimensional TV-L1

Inputs: reference data $\boldsymbol{y}$, penalty parameter λ, initial guess $\boldsymbol{x}^0$, stopping tolerance τ.
Definitions: $([\boldsymbol{x}]_P)_i = \min\{\max\{\boldsymbol{x}_i, -\lambda\}, \lambda\}$, orthogonal projection of $\boldsymbol{x}$ onto feasible set.
Initialization $\boldsymbol{x} = \boldsymbol{x}^0$.
while stopping criterion $> \tau$ **do**
  Compute gradient $\nabla f = \boldsymbol{B}(\boldsymbol{B}^T\boldsymbol{x} - \boldsymbol{y})$
  Identify set of active constraints
    $I = \{i | (-\lambda == \boldsymbol{x}_i, \nabla f(x)_i \leq 0) \text{or} (\lambda == \boldsymbol{x}_i, \nabla f(x)_i \geq 0)\}$
  Construct subspace gradient: $[\nabla \tilde{f}]_i = [\nabla f]_i$ if $i \notin I$, 0 otherwise.
  if iteration number is odd **then**
    Compute stepsize: $\alpha = \frac{\|\nabla \tilde{f}\|_2^2}{\langle \nabla \tilde{f}, \boldsymbol{B}\boldsymbol{B}^T \nabla \tilde{f} \rangle}$.
  else
    Compute stepsize: $\alpha = \frac{\langle \nabla \tilde{f}, \boldsymbol{B}\boldsymbol{B}^T \nabla \tilde{f} \rangle}{\|\boldsymbol{B}\boldsymbol{B}^T \nabla \tilde{f}\|_2^2}$.
  end if
  $\boldsymbol{x} = [\boldsymbol{x} - \alpha \nabla f]_+$.
end while

values. Therefore one might expect $\lim_{\alpha \to 0} \psi(\alpha) = \infty$. However, we will see that in fact this does not happen in the problem at hand.

Proposition 4. *In a problem in the form (4.5), Moré and Sorensen's "hard case" cannot happen. Furthermore* $\lim_{\alpha \to 0} \psi(\alpha) = c \in \mathbb{R}$.

Proof. The Hessian of (3.2), $\boldsymbol{A} = \mathbf{B}\mathbf{B}^T$, is a Gram matrix, and as such, positive-semidefinite. To see that no poles appear at zero despite the existence of $\lambda_i = 0$ eigenvalues we need to prove that $\gamma_i = 0 \; \forall i$ s.t. $\lambda_i = 0$. This can be achieved by using the SVD decomposition $\mathbf{B} = \mathbf{Q}\boldsymbol{\Sigma}\boldsymbol{V}^T$, where $\mathbf{Q}$ is the eigenvector matrix of $\mathbf{B}\mathbf{B}^T = \boldsymbol{A}$, $\boldsymbol{V}^T$ is the eigenvector matrix of $\mathbf{B}^T\mathbf{B}$, and $\boldsymbol{\Sigma}$ is a diagonal matrix containing the root of the eigenvalues of $\mathbf{B}\mathbf{B}^T = \boldsymbol{A}$ (or equivalently, $\mathbf{B}^T\mathbf{B}$). By definition, $\gamma_i = -\boldsymbol{y}^T\mathbf{B}^T\boldsymbol{Q}_i$, and applying the SVD decomposition we find

$$\begin{aligned} \gamma_i &= -\boldsymbol{y}^T\mathbf{B}^T\boldsymbol{Q}_i \\ &= -\boldsymbol{y}^T\boldsymbol{V}^T\boldsymbol{\Sigma}\mathbf{Q}^T\boldsymbol{Q}_i \\ &= -\boldsymbol{y}^T\boldsymbol{V}^T\boldsymbol{\Sigma}\boldsymbol{e}_i \\ &= -\lambda_i(\boldsymbol{y}^T\boldsymbol{V}^T\boldsymbol{e}_i) \end{aligned}$$

where $\boldsymbol{e}_i$ is an all-zeros vector except for the i-th entry, which is valued 1. From this expression it is clear that whenever $\lambda_i = 0$, $\gamma_i = 0$. Therefore the are no poles at $\alpha = 0$, and so $\lim_{\alpha \to 0} \psi(\alpha) = c \in \mathbb{R}$. Consequently, the rightmost pole is located at some point $\tilde{\alpha} < 0$, and as by the KKT conditions (3.3) $\alpha \geq 0$, then the solution $\alpha^* \in [0, \infty)$. □

Corollary 5. *Equivalently to the 1-dimensional case, it follows that the maximum value of the* $\psi(\alpha)$ *function is attained at* $\alpha = 0$. *Therefore, for any* $\lambda > \lambda_{MAX} = \psi(0)$ *the solution* $\boldsymbol{w}$ *is always the same point in the interior of the feasible set. Thus, the range of* λ *values generating different models is bounded by* $\lambda \in [0, \lambda_{MAX}]$.

Being again in the "easy-case" allows us to follow a similar approach to the one applied in the 1-dimensional case. In spite of this, the singularity of the Hessian $\boldsymbol{A}$ produces some difficulties. To begin with, it is not possible to compute the Cholesky decomposition for a non definite-positive matrix. Furthermore, on each iteration of the algorithm we are required to solve the system $(\boldsymbol{A} + \alpha \boldsymbol{I}_{2N(N-1)})\boldsymbol{w} = -\boldsymbol{g}$, $\boldsymbol{g} = \mathbf{B}\boldsymbol{y}$, which is singular for $\alpha = 0$. We will see how to tackle these problems.

Consider the system $(\boldsymbol{A} + \alpha \boldsymbol{I}_{2N(N-1)})\boldsymbol{w} = -\boldsymbol{g}$ to be solved. Making use again of the properties of the Kronecker product, we find that

$$
\begin{aligned}
(\boldsymbol{A} + \alpha \boldsymbol{I}_{2N(N-1)})\boldsymbol{w} &= -\boldsymbol{g} \\
(\mathbf{B}\mathbf{B}^T + \alpha \boldsymbol{I}_{2N(N-1)})\boldsymbol{w} &= -\boldsymbol{g} \\
\left(\begin{bmatrix} \boldsymbol{I}_N \otimes \boldsymbol{D} \\ \boldsymbol{D} \otimes \boldsymbol{I}_N \end{bmatrix} \begin{bmatrix} (\boldsymbol{I}_N \otimes \boldsymbol{D}^T) & (\boldsymbol{D}^T \otimes \boldsymbol{I}_N) \end{bmatrix} + \alpha \boldsymbol{I}_{2N(N-1)} \right) \boldsymbol{w} &= -\boldsymbol{g} \\
\left(\begin{bmatrix} \boldsymbol{I}_N \otimes \boldsymbol{D}\boldsymbol{D}^T & \boldsymbol{D}^T \otimes \boldsymbol{D} \\ \boldsymbol{D} \otimes \boldsymbol{D}^T & \boldsymbol{D}\boldsymbol{D}^T \otimes \boldsymbol{I}_N \end{bmatrix} + \alpha \begin{bmatrix} \boldsymbol{I}_N \otimes \boldsymbol{I}_{N-1} & \\ & \boldsymbol{I}_{N-1} \otimes \boldsymbol{I}_N \end{bmatrix} \right) \boldsymbol{w} &= -\boldsymbol{g} \\
\begin{bmatrix} \boldsymbol{I}_N \otimes (\boldsymbol{D}\boldsymbol{D}^T + \alpha \boldsymbol{I}_{N-1}) & \boldsymbol{D}^T \otimes \boldsymbol{D} \\ \boldsymbol{D} \otimes \boldsymbol{D}^T & (\boldsymbol{D}\boldsymbol{D}^T + \alpha \boldsymbol{I}_{N-1}) \otimes \boldsymbol{I}_N \end{bmatrix} \boldsymbol{w} &= -\boldsymbol{g} \\
\begin{bmatrix} \boldsymbol{I}_N \otimes (\boldsymbol{D}\boldsymbol{D}^T + \alpha \boldsymbol{I}_{N-1}) & \boldsymbol{D}^T \otimes \boldsymbol{D} \\ \boldsymbol{D} \otimes \boldsymbol{D}^T & (\boldsymbol{D}\boldsymbol{D}^T + \alpha \boldsymbol{I}_{N-1}) \otimes \boldsymbol{I}_N \end{bmatrix} \begin{bmatrix} \boldsymbol{w}_1 \\ \boldsymbol{w}_2 \end{bmatrix} &= -\begin{bmatrix} \boldsymbol{g}_1 \\ \boldsymbol{g}_2 \end{bmatrix},
\end{aligned}
$$

where we have split $\boldsymbol{g}$ and $\boldsymbol{w}$ into two halves, $\boldsymbol{g}_1$, $\boldsymbol{g}_2$ and $\boldsymbol{w}_1$, $\boldsymbol{w}_2$. We can now break down the system into two smaller though related equation systems, which we can cast into matrix equation systems as follows

$$
\begin{cases}
(\boldsymbol{I}_N \otimes (\boldsymbol{D}\boldsymbol{D}^T + \alpha \boldsymbol{I}_{N-1}))\boldsymbol{w}_1 + (\boldsymbol{D}^T \otimes \boldsymbol{D})\boldsymbol{w}_2 = -\boldsymbol{g}_1 \\
(\boldsymbol{D} \otimes \boldsymbol{D}^T)\boldsymbol{w}_1 + ((\boldsymbol{D}\boldsymbol{D}^T + \alpha \boldsymbol{I}_{N-1}) \otimes \boldsymbol{I}_N)\boldsymbol{w}_2 = -\boldsymbol{g}_2
\end{cases}
$$

$$
\begin{cases}
(\boldsymbol{D}\boldsymbol{D}^T + \alpha \boldsymbol{I}_{N-1})\boldsymbol{W}_1 + \boldsymbol{D}\boldsymbol{W}_2\boldsymbol{D} = -\boldsymbol{G}_1 \\
\boldsymbol{D}^T\boldsymbol{W}_1\boldsymbol{D}^T + \boldsymbol{W}_2(\boldsymbol{D}\boldsymbol{D}^T + \alpha \boldsymbol{I}_{N-1}) = -\boldsymbol{G}_2
\end{cases},
$$

where $\boldsymbol{W}_1$, $\boldsymbol{G}_1 \in \mathcal{R}^{(N-1)\times N}$, $\boldsymbol{W}_2$, $\boldsymbol{G}_2 \in \mathcal{R}^{N\times(N-1)}$ are the matrix versions of $\boldsymbol{w}_1$, $\boldsymbol{g}_1$, $\boldsymbol{w}_2$, $\boldsymbol{g}_2$. Systems of these kind are known as generalized Sylvester equations, and in the general case they lack of an easy solution [24]. Fortunately, once again the special structure of our problem makes the solving of this equation system feasible. To clarify the following derivation, we will use the definitions $\boldsymbol{M} = \boldsymbol{D}\boldsymbol{D}^T$ and $\bar{\boldsymbol{M}} = \boldsymbol{D}\boldsymbol{D}^T + \alpha \boldsymbol{I}_{N-1}$, so that we can rewrite the equation system as

$$
\begin{cases}
\bar{\boldsymbol{M}}\boldsymbol{W}_1 + \boldsymbol{D}\boldsymbol{W}_2\boldsymbol{D} = -\boldsymbol{G}_1 \\
\boldsymbol{D}^T\boldsymbol{W}_1\boldsymbol{D}^T + \boldsymbol{W}_2\bar{\boldsymbol{M}} = -\boldsymbol{G}_2
\end{cases}
$$

$$
\begin{cases}
\boldsymbol{W}_1 = \bar{\boldsymbol{M}}^{-1}(-\boldsymbol{G}_1 - \boldsymbol{D}\boldsymbol{W}_2\boldsymbol{D}) \\
\boldsymbol{W}_2 = (-\boldsymbol{G}_2 - \boldsymbol{D}^T\boldsymbol{W}_1\boldsymbol{D}^T)\bar{\boldsymbol{M}}^{-1}
\end{cases},
$$

introducing the second equation into the first one, we get

$$
\begin{aligned}
\boldsymbol{W}_1 &= \bar{\boldsymbol{M}}^{-1}(-\boldsymbol{G}_1 - \boldsymbol{D}(-\boldsymbol{G}_2 - \boldsymbol{D}^T\boldsymbol{W}_1\boldsymbol{D}^T)\bar{\boldsymbol{M}}^{-1}\boldsymbol{D}) \\
\bar{\boldsymbol{M}}\boldsymbol{W}_1 &= -\boldsymbol{G}_1 - \boldsymbol{D}(-\boldsymbol{G}_2 - \boldsymbol{D}^T\boldsymbol{W}_1\boldsymbol{D}^T)\bar{\boldsymbol{M}}^{-1}\boldsymbol{D} \\
\bar{\boldsymbol{M}}\boldsymbol{W}_1 &= -\boldsymbol{G}_1 + \boldsymbol{D}\boldsymbol{G}_2\bar{\boldsymbol{M}}^{-1}\boldsymbol{D} + \boldsymbol{M}\boldsymbol{W}_1\boldsymbol{D}^T\bar{\boldsymbol{M}}^{-1}\boldsymbol{D} \\
\boldsymbol{G}_1 - \boldsymbol{D}\boldsymbol{G}_2\bar{\boldsymbol{M}}^{-1}\boldsymbol{D} &= -\bar{\boldsymbol{M}}\boldsymbol{W}_1 + \boldsymbol{M}\boldsymbol{W}_1\boldsymbol{D}^T\bar{\boldsymbol{M}}^{-1}\boldsymbol{D} \\
\boldsymbol{M}^{-1}(\boldsymbol{G}_1 - \boldsymbol{D}\boldsymbol{G}_2\bar{\boldsymbol{M}}^{-1}\boldsymbol{D}) &= -\boldsymbol{M}^{-1}\bar{\boldsymbol{M}}\boldsymbol{W}_1 + \boldsymbol{M}^{-1}\boldsymbol{M}\boldsymbol{W}_1\boldsymbol{D}^T\bar{\boldsymbol{M}}^{-1}\boldsymbol{D} \\
\boldsymbol{C} &= -(\boldsymbol{I}_{N-1} + \alpha\boldsymbol{M}^{-1})\boldsymbol{W}_1 + \boldsymbol{W}_1\boldsymbol{D}^T\bar{\boldsymbol{M}}^{-1}\boldsymbol{D},
\end{aligned}
$$

with $\boldsymbol{C} = \boldsymbol{M}^{-1}(\boldsymbol{G}_1 - \boldsymbol{D}\boldsymbol{G}_2\bar{\boldsymbol{M}}^{-1}\boldsymbol{D})$, which is nothing else but an standard Sylvester equation. Suppose now that we can compute the eigendecomposition $\boldsymbol{M} = \mathbf{T}\mathbf{U}\mathbf{T}^T$, and we define $\bar{\mathbf{U}} = \mathbf{U} + \alpha\boldsymbol{I}_{N-1}$, so that we also have the eigendecomposition $\bar{\boldsymbol{M}} = \mathbf{T}\bar{\mathbf{U}}\mathbf{T}^T$. Then the equation to solve turns out to be

$$
\begin{aligned}
\boldsymbol{C} &= -\mathbf{T}\mathbf{U}^{-1}\bar{\mathbf{U}}\mathbf{T}^T\boldsymbol{W}_1 + \boldsymbol{W}_1\boldsymbol{D}^T\mathbf{T}\bar{\mathbf{U}}^{-1}\mathbf{T}^T\boldsymbol{D} \\
\boldsymbol{C}\boldsymbol{D}^T\mathbf{T} &= -\mathbf{T}\mathbf{U}^{-1}\bar{\mathbf{U}}\mathbf{T}^T\boldsymbol{W}_1\boldsymbol{D}^T\mathbf{T} + \boldsymbol{W}_1\boldsymbol{D}^T\mathbf{T}\bar{\mathbf{U}}^{-1}\mathbf{T}^T\boldsymbol{D}\boldsymbol{D}^T\mathbf{T} \\
\boldsymbol{C}\boldsymbol{D}^T\mathbf{T} &= -\mathbf{T}\mathbf{U}^{-1}\bar{\mathbf{U}}\mathbf{T}^T\boldsymbol{W}_1\boldsymbol{D}^T\mathbf{T} + \boldsymbol{W}_1\boldsymbol{D}^T\mathbf{T}\bar{\mathbf{U}}^{-1}\mathbf{U} \\
\mathbf{T}^T\boldsymbol{C}\boldsymbol{D}^T\mathbf{T} &= -\mathbf{U}^{-1}\bar{\mathbf{U}}\mathbf{T}^T\boldsymbol{W}_1\boldsymbol{D}^T\mathbf{T} + \mathbf{T}^T\boldsymbol{W}_1\boldsymbol{D}^T\mathbf{T}\bar{\mathbf{U}}^{-1}\mathbf{U}.
\end{aligned}
$$

Defining now

$$\begin{aligned}\tilde{\boldsymbol{C}} &= \mathbf{T}^T\boldsymbol{C}\boldsymbol{D}^T\mathbf{T} \\ &= \mathbf{U}^{-1}(\mathbf{T}^T\boldsymbol{G}_1 - \mathbf{T}^T\boldsymbol{D}\boldsymbol{G}_2\mathbf{T}\bar{\mathbf{U}}^{-1}\mathbf{T}^T\boldsymbol{D})\boldsymbol{D}^T\mathbf{T} \\ &= \mathbf{U}^{-1}(\mathbf{T}^T\boldsymbol{G}_1\boldsymbol{D}^T\mathbf{T} - \mathbf{T}^T\boldsymbol{D}\boldsymbol{G}_2\mathbf{T}\mathbf{U}\bar{\mathbf{U}}^{-1}) \qquad (4.6) \\ \tilde{\boldsymbol{W}}_1 &= \mathbf{T}^T\boldsymbol{W}_1\boldsymbol{D}^T\mathbf{T},\end{aligned}$$

we can rewrite the equation as

$$\tilde{\boldsymbol{C}} = -\mathbf{U}^{-1}\bar{\mathbf{U}}\tilde{\boldsymbol{W}}_1 + \tilde{\boldsymbol{W}}_1\bar{\mathbf{U}}^{-1}\mathbf{U},$$

where the coefficient matrices multiplying $\tilde{\boldsymbol{W}}_1$ are diagonal. Hence we can solve for $\tilde{\boldsymbol{W}}_1$ in a similar fashion to our solution of the 2-dimensional ADMM method, resulting in

$$[\tilde{\boldsymbol{W}}_1]_{ij} = \frac{\tilde{\boldsymbol{C}}_{ij}}{-\mathbf{U}_{ii}^{-1}\bar{\mathbf{U}}_{ii} + \bar{\mathbf{U}}_{jj}^{-1}\mathbf{U}_{jj}} = \frac{\tilde{\boldsymbol{C}}_{ij}}{-1 - \frac{\alpha}{\mathbf{U}_{ii}} + \frac{\mathbf{U}_{jj}}{\mathbf{U}_{jj}+\alpha}}. \qquad (4.7)$$

With this we can recover the original $\boldsymbol{W}_1$, $\boldsymbol{W}_2$ solutions for our system by noting that $\boldsymbol{W}_1\boldsymbol{D}^T = \mathbf{T}\tilde{\boldsymbol{W}}_1\mathbf{T}^T$, and so

$$\begin{aligned}\boldsymbol{W}_2 &= (-\boldsymbol{G}_2 - \boldsymbol{D}\mathbf{T}\tilde{\boldsymbol{W}}_1\mathbf{T}^T)\mathbf{T}\bar{\mathbf{U}}^{-1}\mathbf{T}^T \qquad (4.8) \\ &= (-\boldsymbol{G}_2\mathbf{T} - \boldsymbol{D}\mathbf{T}\tilde{\boldsymbol{W}}_1)\bar{\mathbf{U}}^{-1}\mathbf{T}^T \qquad (4.9) \\ \boldsymbol{W}_1 &= \mathbf{T}\bar{\mathbf{U}}^{-1}\mathbf{T}^T(-\boldsymbol{G}_1 - \boldsymbol{D}\boldsymbol{W}_2\boldsymbol{D}). \qquad (4.10)\end{aligned}$$

Following this procedure we are able to find a solution for the Sylvester system. The most expensive part of the algorithm is computing the eigendecomposition and performing multiplications with the $\mathbf{T}$ matrices. Fortunately these tasks can be performed on $\mathrm{O}(N^2 \log_2 N)$ time as we show in Appendix A.

With all this in mind we can solve efficiently the equation system $(\boldsymbol{A}+\alpha\boldsymbol{I}_{2N(N-1)})\boldsymbol{w} = -\boldsymbol{g}$. Note however that for the Moré-Sorensen algorithm we need to compute a vector $\boldsymbol{q}$ obtained by solving the system $\boldsymbol{R}^T\boldsymbol{q} = \boldsymbol{w}$, with $\boldsymbol{R}$ Cholesky decomposition matrix of $(\boldsymbol{A} + \alpha\boldsymbol{I}_{2N(N-1)})$. Unfortunately, as stated before, computing the Cholesky decomposition is not easy in this case, and does not even exist for $\alpha = 0$. To deal with this, note that the only purpose of $\boldsymbol{q}$ in the algorithm is to compute the norm $\|\boldsymbol{q}\|_2^2$, and so we can write

$$\begin{aligned}\|\boldsymbol{q}\|_2^2 &= (\boldsymbol{w}^T\boldsymbol{R}^{-1})(\boldsymbol{R}^{-1^T}\boldsymbol{w}) \\ &= \boldsymbol{w}^T(\boldsymbol{R}^T\boldsymbol{R})^{-1}\boldsymbol{w} \\ &= \boldsymbol{w}^T(\boldsymbol{A} + \alpha\boldsymbol{I}_{2N(N-1)})^{-1}\boldsymbol{w} \\ &= \boldsymbol{w}^T\boldsymbol{v},\end{aligned}$$

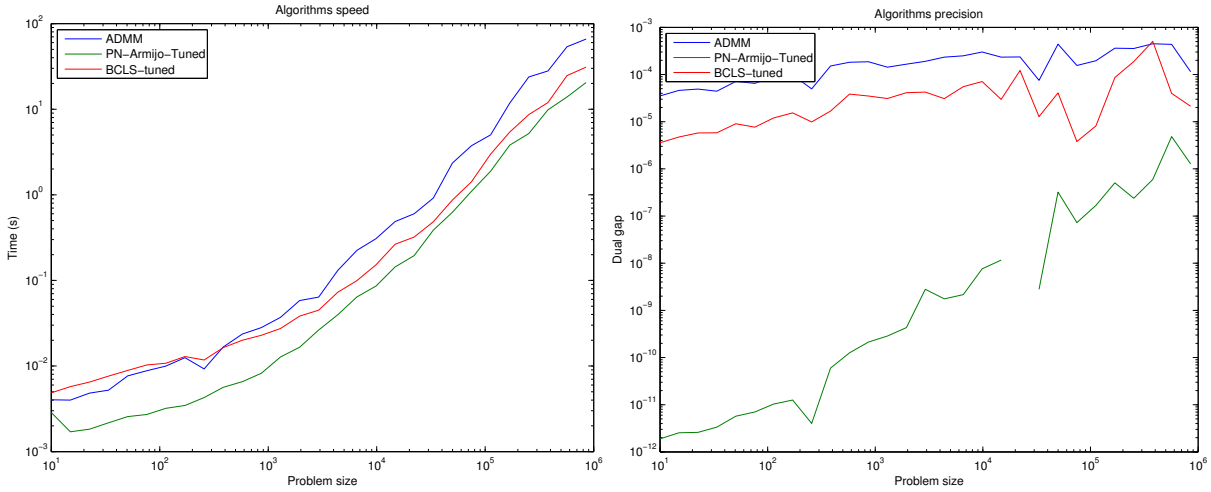
with $\boldsymbol{v} = (\boldsymbol{A} + \alpha\boldsymbol{I}_{2N(N-1)})^{-1}\boldsymbol{w}$, that is, $\boldsymbol{v}$ can be found by solving the system $(\boldsymbol{A} + \alpha\boldsymbol{I}_{2N(N-1)})\boldsymbol{v} = \boldsymbol{w}$, which has the same form as $(\boldsymbol{A} + \alpha\boldsymbol{I}_{2N(N-1)})\boldsymbol{w} = -\boldsymbol{g}$, and thus can be solved following the same procedure.

Algorithm 9 Moré-Sorensen algorithm for 2-dimensional TV-L2

Inputs: reference image $\boldsymbol{Y}$, trust region parameter λ, stopping tolerance τ, lower bound α_{min}.
Initialization $\alpha = \alpha_{min}$
Compute $\boldsymbol{g} = \boldsymbol{B}\boldsymbol{y}$.
while stopping criterion $> \tau$ **do**
  Obtain $\boldsymbol{w}$ solving system $(\boldsymbol{A} + \alpha\boldsymbol{I}_{2N(N-1)})\boldsymbol{w} = -\boldsymbol{g}$.
  Obtain $\boldsymbol{v}$ solving system $(\boldsymbol{A} + \alpha\boldsymbol{I}_{2N(N-1)})\boldsymbol{v} = \boldsymbol{w}$.
  Update α as: $\alpha' = \alpha + \frac{\|\boldsymbol{w}\|_2^2}{\boldsymbol{w}^T\boldsymbol{v}} \cdot \frac{\|\boldsymbol{w}\|_2 - \lambda}{\lambda}$
end while

Algorithm 10 Subsystem solver for Algorithm 9

Inputs: lagrange parameter α, independent vector $\boldsymbol{v}$.
Split independent vector $\boldsymbol{v}$ into matrices $\boldsymbol{G}_1$, $\boldsymbol{G}_2$.
Compute $\tilde{\boldsymbol{C}}$ using (4.6).
Compute $\tilde{\boldsymbol{W}}_1$ using (4.7).
Compute $\boldsymbol{W}_2$, $\boldsymbol{W}_1$ using (4.9,4.8).
Assemble back the solution $\boldsymbol{w}$ from $\boldsymbol{W}_1$, $\boldsymbol{W}_2$.



(a) Running times (b) Dual gap

Figure 1: Running times and dual gap (accuracy) for increasing input data sizes, for the Alternating-Direction Method of Multipliers (ADMM), Projected Newton (PN-Armijo-Tuned) and SBB (BCLS-tuned), for the TV L1 1-dimensional problem

The resulting algorithm is presented as Algorithm 9, the procedure to solve the equation systems arising in each step being shown as Algorithm 10. Note that in formulas (4.6,4.7,4.8,4.9) some of the involved terms do not depend on α, and thus can be precomputed for further savings. Finally, even though the equation systems can be solved for $\alpha = 0$, this situation is prone to suffer from numerical instabilities. To overcome this we define a lower bound $\alpha_{min} \leq \alpha$ such that α is always kept in the interval $\alpha \in [\alpha_{min}, \infty)$. Even if the solution of the problem is an $\boldsymbol{w}$ generated by $\alpha = 0$, Proposition 4 guarantees that a solution for $\alpha = \alpha_{min}$ for a sufficiently small α_{min} (e.g. 10^{-7}) is a reasonable approximation, and in practice produces small a enough dual gap.

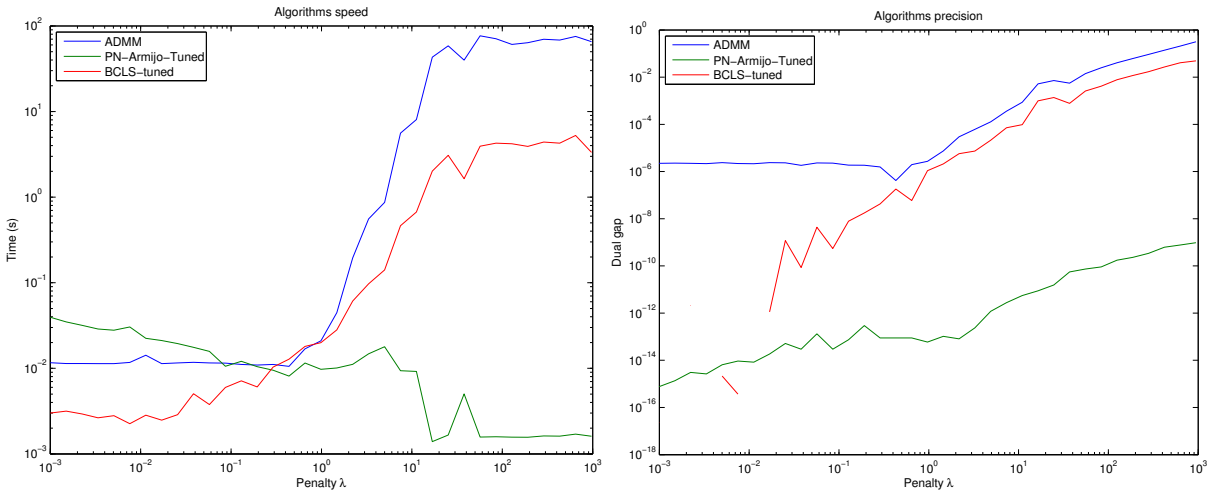
5 Experiments

In this section we present some numerical results showing the performance of the developed algorithms. We will test the running times and accuracy of the solution in two different scenarios: for a fixed-magnitude penalty parameter λ and increasing sizes of the input data, and for a fixed size of the input data and increasing penalty λ. For the first scenario we will run several trials over the algorithms in which we will select a random λ in the range $[1 - 100]$, and increasing values of the input entries ranging from 10 up to 10^6. In the second scenario we will maintain a fixed input of size 1000 and vary λ from 10^{-3} to 10^3.

5.1 1-dimensional case

Figure 1 shows running times in seconds and dual gap obtained by the algorithms presented to solve the TV L1 problem. For ADMM a coefficient $\gamma = 1$ was selected. For Projected Newton the Armijo paramters were $\beta = 0.9$ and $\sigma = 0.1$.The three algorithms present a similar tendency in their running times as the input grows larger, though Projected Newton stands as a clear winner, both in running times and accuracy of the solution, specially in the case of small input data.

On the other hand, when the size of the input is fixed and we increase the penalty λ, large differences arise between the methods, as can be observed in Figure 2. For small penalties ADMM and specially SBB are very



(a) Running times

(b) Dual gap

Figure 2: Running times and dual gap (accuracy) for increasing λ penalties, for the Alternating-Direction Method of Multipliers (ADMM), Projected Newton (PN-Armijo-Tuned) and SBB (BCLS-tuned), for the TV L1 1-dimensional problem.

fast, but the performance largely degrades for large penalties. Conversely Projected Newton runs fastest when the penalty is large. What is more, Projected Newton provides solutions with good accuracy for all cases. Therefore we can conclude that for this case Projected Newton is the algorithm of choice.

For L2 case we compare ADMM against Moré-Sorensen in Figure 3. For ADMM we use the rule-of-thumb of selecting $\gamma = \sqrt{N}$, which seems to work well in practice. Both methods perform adequately, though Moré-Sorensen is clearly faster and generally obtains better accuracy in spite of showing larger variance in the quality of the solutions.

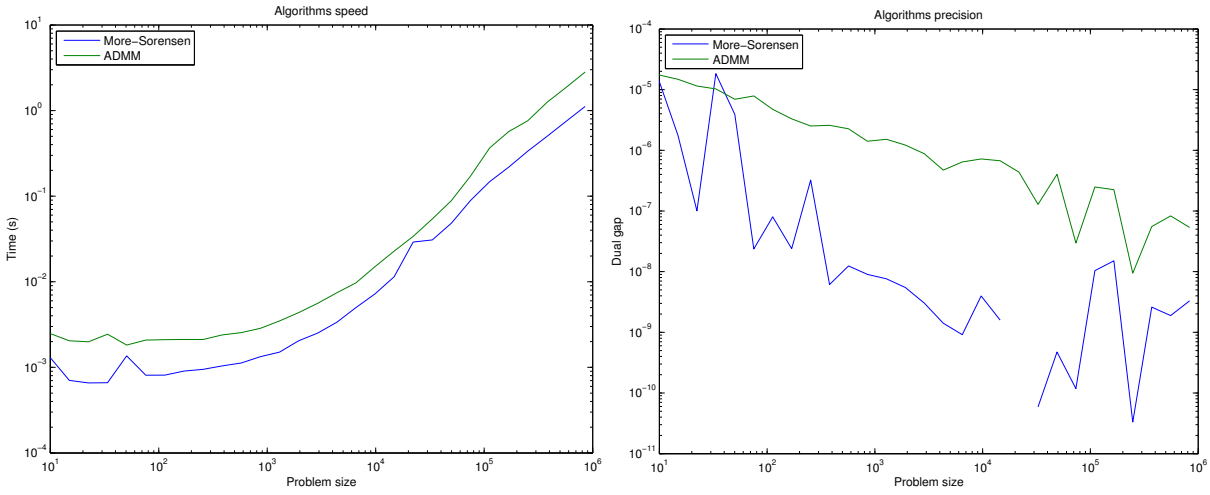
Regarding the effect of λ, in Figure 4 we can clearly see that the performance of ADMM largely degrades for large values of the penalty, as well as the accuracy of the solution. Running the algorithm with different γ values somehow alleviates this problem, but there does not seem to be a clear rule for choosing γ approriately. Conversely Moré-Sorensen seems to work well under any situation and it is parameter-free. Therefore we state that Moré-Sorensen is the best method for this case of the TV problem.

5.2 2-dimensional case

When using the L1 norm in the 2-dimensional case, the obtained results are poor. Figure 5 shows the performance of the ADMM method with parameter $\gamma = 1$. By comparing this to previous results for the 1-dimensional case, we can see that the convergence speed is slow. For the proposed extension of SBB to 2 dimensions (results not shown), even though the cost per iteration is slow, a very large number of iterations are required for convergence, turning the method impractical. In section 6 we suggest some alternative ideas to tackle this problem.

Figure 6 presents the results for Alternating-Direction Method of Multipliers (ADMM) and Moré-Sorensen algorithms in the L2 case. For ADMM a $\gamma = \sqrt{N}$ was selected. This time ADMM seems to perform better for small sizes of input data, although eventually both methods achieve similar performance. Regarding the accuracy, though, Moré-Sorensen provides better quality solutions.
It must be pointed out that most of the time required by Moré-Sorensen is spent in performing the DST operation. For this particular implementation the DST was computed by casting it into a FFT operation and using MATLAB's internal fft operation. This is not the most efficient way to implement a DFT, and thus a further speed improvement could be gained by refining this point.

When analyzing the performance of these algorithms for increasing penalty values, we observe a behaviour quite similar to the one shown for the 1-dimensional case (see Figure 7). While for small penalties both methods behave similarly, for large penalties the performance of ADMM degrades significantly. On the other hand both methods provide similar quality of solutions on average, though ADMM is more stable. Unfortunately the two of them fail to provide acceptable solutions for large penalties. Note however that such large penalties might be too strong for standard applications of the model, and so may not be as relevant in practice.



(a) Running times

(b) Dual gap

Figure 3: Running times and dual gap (accuracy) for increasing input data sizes, for the Alternating-Direction Method of Multipliers (ADMM) and Moré-Sorensen, for the TV L2 1-dimensional problem.

6 Extensions and further work

More complex models making use of a Total Variation regularization could be solved by using the proposed methods as intermediate solvers. For instance, the function

$$\min_{\boldsymbol{x}} f(\boldsymbol{x}) + \lambda \|\boldsymbol{M}\boldsymbol{x}\|_p,$$

either for $\boldsymbol{M} = \boldsymbol{D}$ or $\boldsymbol{M} = \boldsymbol{B}$ could be solved by a Trust Region algorithm like the one presented in [16], where at each step a quadratic model approximating f is constructed and minimized by solving a proximity operator in the form

$$\arg\min_{\boldsymbol{x}} \tfrac{1}{2}\|\boldsymbol{x} - \boldsymbol{y}\|_2^2 + \lambda \|\boldsymbol{M}\boldsymbol{x}\|_p,$$

for a certain $\boldsymbol{y}$, which is exactly the problem we have solved here. In this kind of setting the ability to solve fastly each of these subproblems is crucial for the performance of the overall algorithm. Having proven that our proposed algorithms are able to solve this class of problems efficiently and to a good degree of accuracy, we believe that a Trust Region approach based on them could provide good results also for this general problems.

Other proposals for TV-like problems are found in the literature, e.g. in the context of image deblurring. In [7] an L1 norm is proposed for the fidelity term instead of the standard L2 norm, resulting in an optimization problem in the form

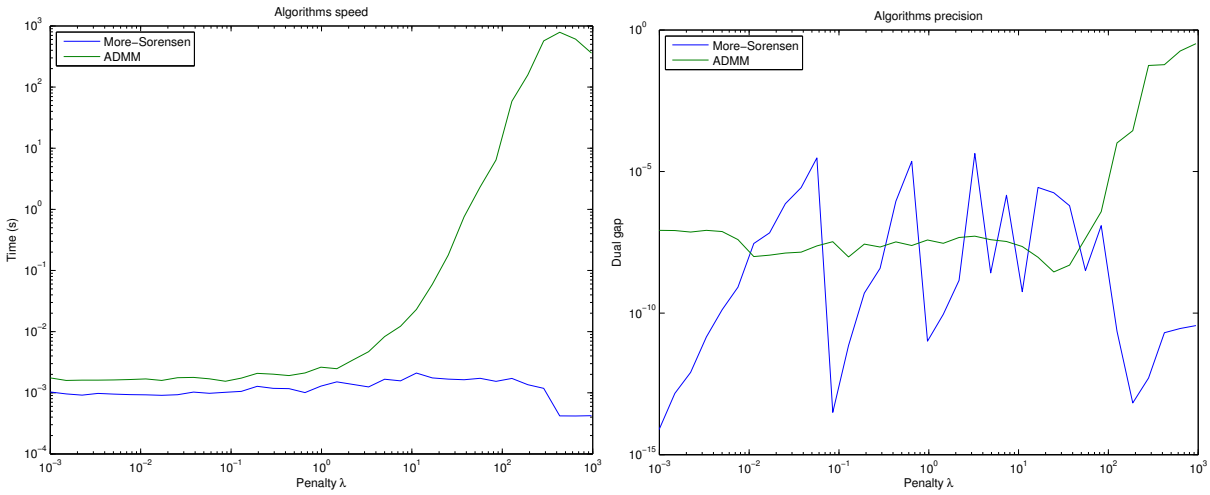
$$\min_{\boldsymbol{x}} ||\boldsymbol{x} - \boldsymbol{y}||_1 + \lambda ||\boldsymbol{D}\boldsymbol{x}||_1.$$

Another proposal in [15] uses the p power of the TV Lp norm, resulting in

$$\min_{\boldsymbol{x}} \tfrac{1}{2}\|\boldsymbol{x} - \boldsymbol{y}\|_2^2 + \lambda \|\boldsymbol{D}\boldsymbol{x}\|_p^p.$$

In [5] yet more complex extensions of the TV operator are presented. Whether the algorithms proposed in this work can be extended to these setting is an open question at the moment.

Additionally it would be of interest to test whether an Interior Point (IP) method [25] could overperform the presented algorithms for the case of the L1 norm. These algorithms are known for their good convergence properties and their flexibility to tackle a broad range of optimization problems. However, their convergence speed largely depends on the ability to solve efficiently a relaxed KKT satisfiability equation system. Software libraries such as CVXOPT [9] allow to plug-in customized solvers for this system, so that the IP method can take advantage of it. A



(a) Running times (b) Dual gap

Figure 4: Running times and dual gap (accuracy) for increasing λ penalties, for the Alternating-Direction Method of Multipliers (ADMM) and Moré-Sorensen, for the TV L2 1-dimensional problem.

preliminary study has revealed that a TV L1 solver based on this approach could be feasible, though further work is needed to construct a functional implementation. Furthermore whether this approach could be extended to the L1 2-dimensional case, where we lack of an efficient algorithm, is another point of interest.

Finally, as already mentioned, the most frequent application of the TV regularization is within the context of image processing, in which it is used for operations such as denoising, inpainting and deblurring [10]. For these applications a number of software tools are already publicly available. Therefore, to correctly assess the performance of the algorithms presented here a comparison againts those implementations should be made.

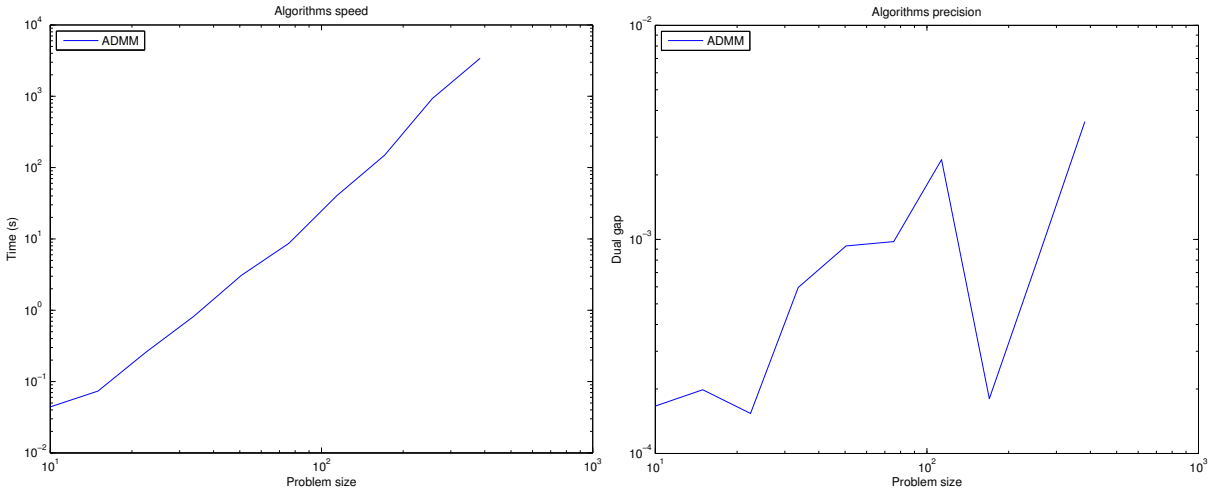
A Fast matrix factorizations

Matrices $\boldsymbol{D}\boldsymbol{D}^T$ and $\boldsymbol{D}^T\boldsymbol{D}$ present the following tridiagonal structure

$$\boldsymbol{D}\boldsymbol{D}^T = \begin{pmatrix} 2 & -1 & & & & \\ -1 & 2 & -1 & & & \\ & -1 & 2 & -1 & & \\ & & & \ddots & & \\ & & & -1 & 2 & -1 \\ & & & & -1 & 2 \end{pmatrix} \in \mathbb{R}^{(N-1)\times(N-1)},$$

$$\boldsymbol{D}^T\boldsymbol{D} = \begin{pmatrix} 1 & -1 & & & & \\ -1 & 2 & -1 & & & \\ & -1 & 2 & -1 & & \\ & & & \ddots & & \\ & & & -1 & 2 & -1 \\ & & & & -1 & 1 \end{pmatrix} \in \mathbb{R}^{N\times N}.$$

Any tridiagonal matrix can be factorized in the form $\boldsymbol{M} = \boldsymbol{L}\boldsymbol{\Delta}\boldsymbol{L}^T$ [14], the factors having the following structure



(a) Running times (b) Dual gap

Figure 5: Running times and dual gap (accuracy) for increasing input data sizes ($N \times N$), for the Alternating-Direction Method of Multipliers (ADMM), for the TV L1 2-dimensional problem.

$$\boldsymbol{L} = \begin{pmatrix} 1 & & & & & \\ l_1 & 1 & & & & \\ & l_2 & 1 & & & \\ & & \ddots & \ddots & & \\ & & & l_{N-2} & 1 & \\ & & & & l_{N-1} & 1 \end{pmatrix}, \quad \boldsymbol{\Delta} = \begin{pmatrix} \delta_1 & & & \\ & \delta_2 & & \\ & & \ddots & \\ & & & \delta_N \end{pmatrix}.$$

The entries l and δ can be easily computed as $\delta_1 = \alpha_1$, $l_i = \alpha_i - \frac{\beta_{i-1}^2}{\delta_{i-1}}$, $\delta_i = \frac{\beta_i}{\delta_i}$. The LAPACK library [1] includes a very fast routine (DPTTRF) that computes this decomposition, as well as a routine to solve a tridiagonal system of equations based on this decomposition (DPTTRS).

Additionally, the eigendecomposition of $\boldsymbol{M} = \boldsymbol{D}\boldsymbol{D}^T = \mathbf{T}\mathbf{U}\mathbf{T}^T$ can also be computed efficiently, with $\mathbf{T}$ eigenvectors matrix and $\mathbf{U}$ diagonal matrix with the eigenvalues. A general eigendecomposition incurs in costs of order O(N^3). We can easen this by invoking the arguments in [20], which state that for this particular matrix the eigenvalues have the closed form expression

$$\lambda_i = 2 - 2\cos\left(\frac{i\pi}{n+1}\right),$$

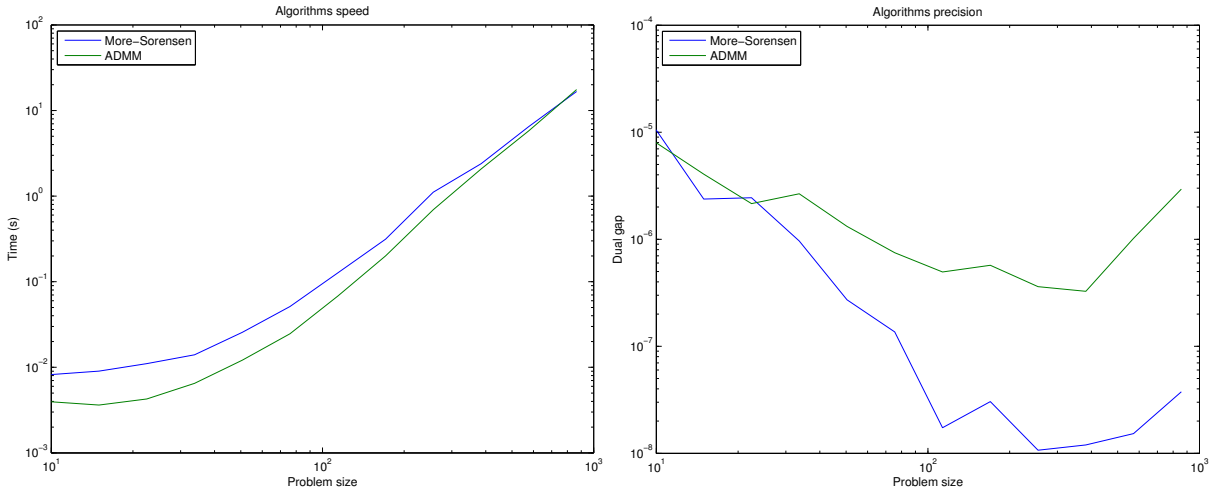
n size of the $\boldsymbol{M}$ matrix, which in our case corresponds to $n = N - 1$. On the other hand, the eigenvectors follow the expression

$$\mathbf{T}_{kj} = \beta_j \sin\left(\frac{kj\pi}{n+1}\right),$$

for appropriate normalization factors β_j such that $\|v_j\|_2 = 1\ \forall j$, which can be computed easily. Thus the eigendecomposition can be performed in closed form with a cost of order O(N^2). In addition to this, multiplications by $\mathbf{T}$ can be performed fast by noting that, e.g. in the $\mathbf{T}\boldsymbol{v}$ case we have

$$[\mathbf{T}\boldsymbol{v}]_i = \sum_{j=1}^{N-1} \boldsymbol{v}_j \beta_j \sin\left(\frac{ij\pi}{n+1}\right).$$

This calculation resembles the transformation known as Discrete Sine Transformation of type 1 (DST-I, DST for short), which presents the following expression



(a) Running times

(b) Dual gap

Figure 6: Running times and dual gap (accuracy) for increasing input data sizes ($N \times N$), for the Alternating-Direction Method of Multipliers (ADMM) and Moré-Sorensen, for the TV L2 2-dimensional problem.

$$[DST(\boldsymbol{x})]_i = \sum_{j=1}^{n} \boldsymbol{x}_j \sin\left(\frac{ij\pi}{n+1}\right).$$

By using the transformation $\tilde{\boldsymbol{v}}_i = \boldsymbol{v}_i\beta_i$, which can be written as $\tilde{\boldsymbol{v}} = diag(\beta)\boldsymbol{v}$ we find out that

$$[\mathbf{T}v]_i = [DST(\tilde{\boldsymbol{v}})]_i = [DST(diag(\beta)\boldsymbol{v})]_i.$$

Therefore we can compute each resulting row of the product by matrix $\mathbf{T}$ by performing a DST instead. The DST operation presents simmetries that allow the use of a divide-and-conquer strategy to speed-up its computation, much in the way the Fast Fourier Transform is performed, resulting in computational costs of order O($N \log_2 N$). Publicly available libraries like FFTW [12] offer efficient methods to perform this computation. All other possible products involving $\mathbf{T}$ can be computed similarly, namely

$$\begin{aligned}
\left[v^T\mathbf{T}\right]_i &= [diag(\beta)DST(\boldsymbol{v})]_i^T, \\
\left[\mathbf{T}^Tv\right]_i &= [diag(\beta)DST(\boldsymbol{v})]_i, \\
\left[v^T\mathbf{T}^T\right]_i &= [DST(diag(\beta)\boldsymbol{v})]_i^T.
\end{aligned}$$

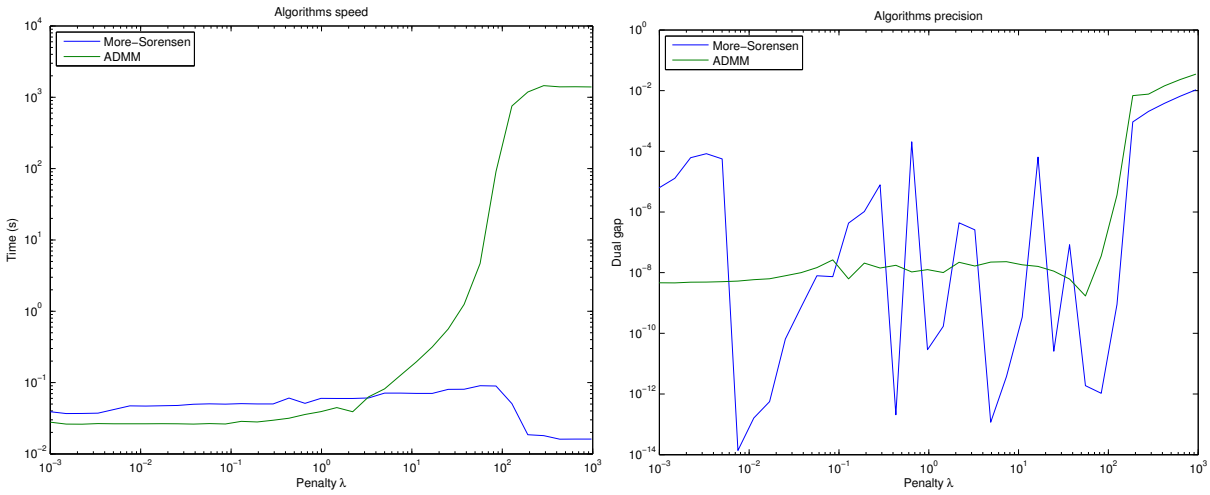
Therefore a computation in the form $\tilde{m}\boldsymbol{v}$ results in a cost of order O($N^2 \log_2(N)$). Note also that the Schur decomposition of a symmetric matrix (like $\boldsymbol{D}\boldsymbol{D}^T$) coincides with its eigendecomposition, and thus can be computed in the same way.

For the case of the matrix $\boldsymbol{D}^T\boldsymbol{D}$, it can be shown that the eigenvalues are the same as those for $\boldsymbol{D}\boldsymbol{D}^T$ plus a zero eigenvalue. Unfortunately there is no simple rule to compute the eigenvectors (up to our knowledge), and thus if the eigenvectors are required a general eigedecomposition for tridiagonal matrices (available in LAPACK) must be performed.

References

[1] E. Anderson, Z. Bai, C. Bischof, S. Blackford, J. Demmel, J. Dongarra, J. Du Croz, A. Greenbaum, S. Hammarling, A. McKenney, and D. Sorensen. *LAPACK Users' Guide*. Society for Industrial and Applied Mathematics, Philadelphia, PA, third edition, 1999. ISBN 0-89871-447-8 (paperback). 22

[2] R. H. Bartels and G. W. Stewart. Solution of the matrix equation AX + XB = C. *Communications of the ACM*, 15(9):820–826, September 1972. 13



(a) Running times

(b) Dual gap

Figure 7: Running times and dual gap (accuracy) for increasing λ penalties, for the Alternating-Direction Method of Multipliers (ADMM) and Moré-Sorensen, for the TV L2 2-dimensional problem.

[3] Jonathan Barzilai and Jonathan M. Borwein. Two-point step size gradient methods. *IMA Journal of Numerical Analysis*, 8:141–148, 1988. 9

[4] Dimitri P. Bertsekas. Projected newton methods for optimization problems with simple constraints. *SIAM J. Control and Optimization*, 20(2), March 1982. 6, 7

[5] Antoni Buades, Triet M. Le, Jean-Michel Morel, and Luminita A. Vese. Fast cartoon + texture image filters. *IEEE Transactions on Image Processing*, 19(8), August 2010. 20

[6] Richard H. Byrd, Peihuang Lu, Jorge Nocedal, and Ciyou Zhu. A limited memory algorithm for bound constrained optimization. Technical report, Northwestern University, 1994. 6

[7] Tony F. Chan and Selim Esedoglu. Aspects of total variation regularized l1 function approximation. Technical report, UCLA Mathematics Department, February 2004. 20

[8] Patrick L. Combettes and Jean-Christophe Pesquet. Proximal splitting methods in signal processing. *arXiv*, 2009. 3, 5, 6

[9] Joachim Dahl and Lieven Vandenberghe. Cvxopt: Python software for convex optimization. URL `http://abel.ee.ucla.edu/cvxopt/index.html`. 20

[10] Joachim Dahl, Per Christian Hansen, Søren Holdt Jensen, and Tobias Lindstrøm Jensen. Algorithms and software for total variation image reconstruction via first-order methods. *Numer Algor*, (53):67–92, 2010. 21

[11] Jennifer B. Erway and Philip E. Gill. A subspace minimization method for the trust-region step. *SIAM J. Optim.*, 20(3):1439–1461, 2009. 10

[12] Matteo Frigo and Steven G. Johnson. The design and implementation of FFTW3. *Proceedings of the IEEE*, 93(2):216–231, 2005. Special issue on "Program Generation, Optimization, and Platform Adaptation". 23

[13] Donald Goldfarb and Wotao Yin. Second-order cone programming methods for total variation-based image restoration. *SIAM J. Sci. Comput.*, 27(2):622–645, 2005. 1

[14] Gene H. Golub and Gerard Meurant. *Matrices, Moments and Quadrature with Applications*. Princeton University Press, 2010. 21

[15] Christian Hansen, James G. Nagy, and Dianne P. O'Leary. *Deblurring images: matrices, spectra and filtering*. SIAM, 2006. 20

[16] Dongmin Kim, Suvrit Sra, and Inderjit Dhillon. A scalable trust-region algorithm with application to mixed-norm regression. In *Proceedings of the 27th International Conference on Machine Learning*, 2010. 3, 20

[17] Dongmin Kim, Suvrit Sra, and Inderjit S. Dhillon. A non-monotonic method for large-scale nonnegative least squares. Technical report, University of Texas at Austin and Max-Planck-Institute for Biological Cybernetics, May 2010. 6, 9

[18] Chih-Jen Lin and Jorge J. Moré. Newton's method for large bound-constrained optimization problems. *SIAM J. Optim.*, 9(4):1100–1127, 1999. 6

[19] Jorge J. Moré and D. C. Sorensen. Computing a trust region step. *SIAM J. Sci. Stat. Comput.*, 4(3), September 1983. 10, 11

[20] Dianne P. O'Leary. *Scientific computing with case studies*. SIAM, 2009. 13, 22

[21] Tyrrell Rockafellar. *Convex Analysis*. Princeton University Press, 1997. 3

[22] Marielba Rojas, Sandra A. Santos, and Danny C. Sorensen. Algorithm 873: LSTRS: MATLAB software for large-scale trust-region subproblems and regularization. *ACM Trans. Math. Software*, 34(2):11, 2008. 10

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

[24] Charles F. Van Loan. The ubiquitous kronecker product. *Journal of Computational and Applied Mathematics*, 123:85–100, 2000. 5, 16

[25] Stephen J. Wright. *Primal-Dual Interior Point Methods*. SIAM, 1997. 20