Journal of Machine Learning Research 19 (2018) 1-82 Submitted 11/13; Revised 10/18; Published 11/18

Modular Proximal Optimization for Multidimensional Total-Variation Regularization

Álvaro Barbero ALVARO.BARBERO@INV.UAM.ES
Instituto de Ingeniería del Conocimiento and Universidad Autónoma de Madrid
Francisco Tomás y Valiente 11, Madrid, Spain

Suvrit Sra* SUVRIT@MIT.EDU
Laboratory for Information and Decision Systems
Massachusetts Institute of Technology (MIT), Cambridge, MA

Editor: Vishwanathan S V N

Abstract

We study *TV regularization*, a widely used technique for eliciting structured sparsity. In particular, we propose efficient algorithms for computing prox-operators for ℓ_p-norm TV. The most important among these is ℓ_1-norm TV, for whose prox-operator we present a new geometric analysis which unveils a hitherto unknown connection to taut-string methods. This connection turns out to be remarkably useful as it shows how our geometry guided implementation results in efficient weighted and unweighted 1D-TV solvers, surpassing state-of-the-art methods. Our 1D-TV solvers provide the backbone for building more complex (two or higher-dimensional) TV solvers within a modular proximal optimization approach. We review the literature for an array of methods exploiting this strategy, and illustrate the benefits of our modular design through extensive suite of experiments on (i) image denoising, (ii) image deconvolution, (iii) four variants of fused-lasso, and (iv) video denoising. To underscore our claims and permit easy reproducibility, we provide all the reviewed and our new TV solvers in an easy to use multi-threaded C++, Matlab and Python library.

Keywords: proximal optimization, total variation, regularized learning, sparsity, non–smooth optimization

1. Introduction

Sparsity impacts the entire data analysis pipeline, touching algorithmic, modeling, as well as practical aspects. Most commonly, sparsity is elicited via ℓ_1-norm regularization (Tibshirani, 1996; Candès and Tao, 2004). However, numerous applications rely on more refined "structured" notions of sparsity, e.g., groupwise-sparsity (Meier et al., 2008; Liu and Zhang, 2009; Yuan and Lin, 2006; Bach et al., 2011), hierarchical sparsity (Bach, 2010; Mairal et al., 2010), gradient sparsity (Rudin et al., 1992; Vogel and Oman, 1996; Tibshirani et al., 2005), or sparsity over structured 'atoms' (Chandrasekaran et al., 2012).

Such regularizers typically arise in optimization problems of the form

$$\min_{\boldsymbol{x}\in\mathbb{R}^n} \quad \Phi(\boldsymbol{x}) := \ell(\boldsymbol{x}) + r(\boldsymbol{x}), \tag{1.1}$$

∗. An initial version of this work was performed during 2013-14, when the author was with the Max Planck Institute for Intelligent Systems, Tübingen, Germany, and with Carnegie Mellon University, Pittsburgh.

where $\ell : \mathbb{R}^n \to \mathbb{R}$ is a smooth loss function (often convex), while $r : \mathbb{R}^n \to \mathbb{R} \cup \{+\infty\}$ is a lower semicontinuous, convex, and nonsmooth regularizer that induces sparsity.

We focus on instances of (1.1) where r is a weighted *anisotropic Total-Variation* (TV) regularizer[1], which for a vector $\boldsymbol{x} \in \mathbb{R}^n$ and fixed weights $\boldsymbol{w} \geq 0$ is defined as

$$r(\boldsymbol{x}) \stackrel{\text{def}}{=} \mathrm{Tv}_p^1(\boldsymbol{w}; \boldsymbol{x}) \stackrel{\text{def}}{=} \Big(\sum\nolimits_{j=1}^{n-1} w_j |x_{j+1} - x_j|^p\Big)^{1/p} \quad p \geq 1. \tag{1.2}$$

More generally, if X is an order-m tensor in $\mathbb{R}^{\prod_{j=1}^m n_j}$ with entries $\mathsf{X}_{i_1,i_2,\ldots,i_m}$ ($1 \leq i_j \leq n_j$ for $1 \leq j \leq m$); we define the weighted *m-dimensional anisotropic TV* regularizer as

$$\mathrm{Tv}_{\boldsymbol{p}}^m(\mathsf{W}; \mathsf{X}) \stackrel{\text{def}}{=} \sum_{k=1}^{m} \sum_{I_k = \{i_1,\ldots,i_m\}\setminus i_k} \Bigg(\sum_{j=1}^{n_k - 1} w_{I_k,j} |\mathsf{X}_{j+1}^{[k]} - \mathsf{X}_j^{[k]}|^{p_k}\Bigg)^{1/p_k}, \tag{1.3}$$

where $\mathsf{X}_j^{[k]} \equiv \mathsf{X}_{i_1,\ldots,i_{k-1},j,i_{k+1},\ldots,i_m}$, $w_{I_k,j} \geq 0$ are weights, and $\boldsymbol{p} \equiv [p_k \geq 1]$ for $1 \leq k \leq m$. If X is a matrix, expression (1.3) reduces to (note, $p, q \geq 1$)

$$\mathrm{Tv}_{p,q}^2(\mathsf{W}; \mathsf{X}) = \sum_{i=1}^{n_1}\Big(\sum_{j=1}^{n_2-1} w_{1,j}|x_{i,j+1} - x_{i,j}|^p\Big)^{1/p} + \sum_{j=1}^{n_2}\Big(\sum_{i=1}^{n_1-1} w_{2,i}|x_{i+1,j} - x_{i,j}|^q\Big)^{1/q}, \tag{1.4}$$

These definitions look formidable; already 2D-TV (1.4) or even the simplest 1D-TV (1.2) are fairly complex, which further complicates the overall optimization problem (1.1). Fortunately, this complexity can be "localized" by invoking *prox-operators* (Moreau, 1962), which are now widely used across machine learning (Sra et al., 2011; Parikh et al., 2014).

The main idea of using prox-operators while solving (1.1) is as follows. Suppose Φ is a convex lsc function on a set $\mathcal{X} \subset \mathbb{R}^n$. The *prox-operator* of Φ is defined as the map

$$\mathrm{prox}_\Phi \stackrel{\text{def}}{=} \boldsymbol{y} \mapsto \operatorname*{argmin}_{\boldsymbol{x}\in\mathcal{X}} \quad \tfrac{1}{2}\|\boldsymbol{x} - \boldsymbol{y}\|_2^2 + \Phi(\boldsymbol{x}) \quad \text{for} \quad \boldsymbol{y} \in \mathbb{R}^n. \tag{1.5}$$

A popular method based on prox-operators is the *proximal gradient method* (also known as 'forward backward splitting'), which performs a gradient (forward) step followed by a proximal (backward) step to iterate

$$\boldsymbol{x}_{k+1} = \mathrm{prox}_{\eta_k r}(\boldsymbol{x}_k - \eta_k \nabla \ell(\boldsymbol{x}_k)), \quad k = 0, 1, \ldots. \tag{1.6}$$

Numerous other proximal methods exist—see e.g., (Beck and Teboulle, 2009; Nesterov, 2007; Combettes and Pesquet, 2009; Kim et al., 2010; Schmidt et al., 2011).

To implement the proximal-gradient iteration (1.6) efficiently, we require a subroutine that computes the prox-operator prox_r. An additional concern is whether the overall algorithm requires an *exact* computation of prox_r, or merely a moderately *inexact* computation. This concern is justified: rarely does r admit an exact algorithm for computing prox_r. Fortunately, proximal methods easily admit inexactness, e.g., (Schmidt et al., 2011; Salzo and Villa, 2012; Sra, 2012), which allows approximate prox-operators (as long as the approximation is sufficiently accurate).

We study both exact and inexact prox-operators in this paper, contingent upon the ℓ_p-norm used and on the data dimensionality m.

1. We use the term "anisotropic" to refer to the specific TV penalties considered in this paper.

1.1. Contributions

In particular, we review, analyze, implement, and experiment with a variety of fast algorithms. The ensuing contributions of this paper are summarized below.

- Geometric analysis that leads to a new, efficient version of the classic Taut String Method (Davies and Kovac, 2001), whose origins can be traced back to (Barlow, 1972) – this version turns out to perform better than most of the recently developed TV proximity methods.
- A previously unknown connection between (a variation of) this classic algorithm and Condat's *unweighted* TV method (Condat, 2012). This connection provides a geometric, more intuitive interpretation and helps us define a hybrid taut-string algorithm that combines the strengths of both methods, while also providing a new efficient algorithm for *weighted* ℓ_1-norm 1D-TV proximity.
- Efficient prox-operators for general ℓ_p-norm ($p \geq 1$) 1D-TV. In particular,
  - For $p = 2$, we present a specialized Newton method based on the root-finding strategy of Moré and Sorensen (1983),
  - For the general $p \geq 1$ case we describe both "projection-free" and projection based first-order methods.
- Scalable proximal-splitting algorithms for computing 2D (1.4) and higher-D TV (1.3) prox-operators. We review an array of methods in the literature that use prox-splitting, and through extensive experiments show that a splitting strategy based on alternating reflections is the most effective in practice. Furthermore, this modular construction of 2D and higher-D TV solvers allows reuse of our fast 1D-TV routines and exploitation of the massive parallelization inherent in matrix and tensor TV.
- The final most important contribution of our paper is a well-tuned, multi-threaded open-source C++, Matlab and Python implementation of all the reviewed and developed methods.[2]

To complement our algorithms, we illustrate several applications of TV prox-operators to: (i) image and video denoising; (ii) image deconvolution; and (iii) four variants of fused-lasso.

Note: We have invested great efforts to ensure reproducibility of our results. In particular, given the vast attention that TV problems have received in the literature, we believe it is valuable to both users of TV and other researchers to have access to our code, data sets, and scripts, to independently verify our claims, if desired.[3]

1.2. Related Work

The literature on TV is too large to permit a comprehensive review here. Instead, we mention the most directly related work to help place our contributions in perspective.

We focus on *anisotropic*-TV, in contrast to *isotropic*-TV (Rudin et al., 1992). Several proposals for designing an anisotropic variant of TV have been proposed in the literature:

2. See https://github.com/albarji/proxTV
3. This material shall be made available at: http://suvrit.de/work/soft/tv.html

in this paper we use the definition given in Bioucas-Dias and Figueiredo (2007), which follows the already presented Equation (1.2). Alternative definitions of anisotropic TV include instances such as a general TV defined in the continuous domain in terms of Wulff shapes (Esedoglu and Osher, 2004), or making use of estimates of the directional information (Steidl and Teuber , 2009), to name a few. Although the definition used here is simpler, it arises frequently in image denoising and signal processing, and quite a few TV-based denoising algorithms exist (Zhu and Chan, 2008, see e.g.).

The anisotropic TV regularizers $\mathrm{Tv}_1^{\mathrm{1D}}$ and $\mathrm{Tv}_{1,1}^{\mathrm{2D}}$ arise in image denoising and deconvolution (Dahl et al., 2010), in the fused-lasso (Tibshirani et al., 2005), in logistic fused-lasso (Kolar et al., 2010), in change-point detection (Harchaoui and Lévy-Leduc, 2010), in graph-cut based image segmentation (Chambolle and Darbon, 2009), in submodular optimization (Jegelka et al., 2013); see also the related work in (Vert and Bleakley, 2010). This broad applicability and importance of anisotropic TV is the key motivation towards developing carefully tuned proximity operators.

There is a rich literature of methods tailored to anisotropic TV, e.g., those developed in the context of fused-lasso (Friedman et al., 2007; Liu et al., 2010), graph-cuts (Chambolle and Darbon, 2009), ADMM-style approaches (Combettes and Pesquet, 2009; Wahlberg et al., 2012), fast methods based on dynamic programming (Johnson, 2013) or KKT conditions analysis (Condat, 2012). However, it seems that anisotropic TV norms other than ℓ_1 have not been studied much in the literature, although recognized as a form of Sobolev semi-norms (Pontow and Scherzer, 2009).

For 1D-TV and for the particular ℓ_1 norm, there exist several direct methods that are exceptionally fast. We treat this problem in detail in Section 2, and hence refer the reader to that section for discussion of closely related work on fast solvers. We note here, however, that in contrast to many of the previous fast solvers, our solvers allow weights, a capability that can be very important in applications (Jegelka et al., 2013).

Regarding 2D-TV, Goldstein T. (2009) presented a so-called "Split-Bregman" (SB). It turns out that this method is essentially a variant of the well-known ADMM method. In contrast to the 2D approach presented here, the SB strategy followed by Goldstein T. (2009) is to rely on ℓ_1-soft thresholding substeps instead of 1D-TV substeps. From an implementation viewpoint, the SB approach is somewhat simpler, but not necessarily more accurate. Incidentally, sometimes such direct ADMM approaches turn out to be less effective than ADMM methods that rely on more complex 1D-TV prox-operators (Ramdas and Tibshirani, 2014).

It is worth highlighting that it is not just proximal solvers such as FISTA (Beck and Teboulle, 2009), SpaRSA (Wright et al., 2009), SALSA (Afonso et al., 2010), TwIST (Bioucas-Dias and Figueiredo, 2007), Trip (Kim et al., 2010), that can benefit from our fast prox-operators. All other 2D and higher-D TV solvers, e.g., (Yang et al., 2013), as well as the recent ADMM based trend-filtering solvers of Tibshirani (2014) immediately benefit, not only in speed but also by gaining the ability to solve weighted problems.

1.3. Summary of the Paper

The remainder of the paper is organized as follows. In Section 2 we consider prox operators for 1D-TV problems when using the most common ℓ_1 norm. The highlight of this section

is our analysis on taut-string TV solvers, which leads to the development a new hybrid method and a weighted TV solver (Sections 2.3, 2.4). Thereafter, we discuss variants of 1D-TV (Section 3), including a specialized $\mathrm{Tv}_2^{\mathrm{1D}}$ solver, and a more general $\mathrm{Tv}_p^{\mathrm{1D}}$ method based on a gradient projection strategy. Subsequently, we describe multi-dimensional TV problems and study their prox-operators in Section 4, paying special attention to 2D-TV; for both 2D and multi-D, prox-splitting methods are used. After these theoretical sections, we describe experiments and applications in Section 5. In particular, extensive experiments for 1D-TV are presented in Section 5.1 and Section 5.2; 2D-TV experiments are in Section 5.3, while an application of multi-D TV is the subject of Section 5.4. The appendices to the paper include further technical details and additional information about the experimental setup.

2. TV-L1: Fast Prox-Operators for $\mathrm{Tv}_1^{\mathrm{1D}}$

We begin with the 1D-TV problem (1.2) for an ℓ_1 norm choice, for which we review several carefully tuned algorithms. Using such well–tuned algorithms pays off: we can find fast, robust, and low-memory (in fact, in place) algorithms, which are not only of independent value, but also ideal building blocks for scalably solving 2D- and higher-D TV problems.

Computation of the ℓ_1-norm TV prox-operator can be compactly written as the problem

$$\min_{\boldsymbol{x}\in\mathbb{R}^n} \quad \tfrac{1}{2}\|\boldsymbol{x}-\boldsymbol{y}\|_2^2 + \lambda\|\boldsymbol{D}\boldsymbol{x}\|_1, \tag{2.1}$$

where $\boldsymbol{D}$ is the *differencing matrix*, all zeros except $d_{ii} = -1$ and $d_{i,i+1} = 1$ $(1 \le i \le n-1)$.

To solve (2.1) we will analyze an approach based on the line of "taut-string" methods. We first introduce these methods for the unweighted *TV-L1* problem (2.1), before discussing the elementwise weighted TV problem (2.6). Most of the previous fastest methods handle only unweighted-TV. It is often nontrivial to extend them to handle weighted-TV, a problem that is crucial to several applications, e.g., segmentation (Chambolle and Darbon, 2009) and certain submodular optimization problems (Jegelka et al., 2013).

A remarkably efficient approach to TV-L1 was presented in (Condat, 2012). We will show Condat's fast algorithm can be interpreted as a "linearized" version of the taut-string approach, a view that paves the way to obtain an equally fast solver for weighted TV-L1.

Before proceeding we note that other than (Condat, 2012), other efficient methods to address unweighted $\mathrm{Tv}_1^{\mathrm{1D}}$ proximity have been proposed. Johnson (2013) shows how solving $\mathrm{Tv}_p^{\mathrm{1D}}$ proximity is equivalent to computing the data likelihood of an specific Hidden Markov Model (HMM), which suggests a dynamic programming approach based on the well-known Viterbi algorithm for HMMs. The resulting algorithm is very competitive, and guarantees an overall $O(n)$ performance while requiring approximately $8n$ storage. Another similarly performing algorithm was presented by Kolmogorov et al (2015) in the form of a message passing method. We will also consider these algorithms in our experimental comparison in §5.1.

Yet another family of methods is based on projected-Netwon (PN) techniques: we also present in Appendix E a PN approach for its instructive value, and also because it provides key subroutines for solving TV problems with $p > 1$. Our derivation may also be helpful to readers seeking to implement efficient prox-operators for problems that have structure

similar to TV, for instance ℓ_1-trend filtering (Kim et al., 2009; Tibshirani, 2014). Indeed, the PN approach proves to be foundational for the fast "group fused-lasso" algorithms of (Wytock et al., 2014).

2.1. The Taut-String Method for $\mathbf{Tv}_1^{\mathbf{1D}}$

While taut-string methods seem to be largely unknown in machine learning, they have been widely applied in statistics—see e.g., (Grasmair, 2007; Davies and Kovac, 2001; Barlow, 1972).

We start by transforming the problem as follows. For TV-L1, elementary manipulations, e.g., using Proposition A.4, yield the dual (re-written as a minimization problem)

$$\min_{\boldsymbol{u}} \ \tfrac{1}{2}\|\boldsymbol{D}^T\boldsymbol{u}\|_2^2 - \boldsymbol{u}^T\boldsymbol{D}\boldsymbol{y}, \ \text{s.t.} \ \|\boldsymbol{u}\|_\infty \leq \lambda. \tag{2.2}$$

Without changing the minimizer, the objective (2.2) can be replaced by $\|\boldsymbol{D}^T\boldsymbol{u} - \boldsymbol{y}\|_2^2$, which then unfolds into

$$(u_1 - y_1)^2 + \sum\nolimits_{i=2}^{n-1} (-u_{i-1} + u_i - y_i)^2 + (-u_{n-1} - y_n)^2 .$$

Introducing the fixed extreme points $u_0 = u_n = 0$, we can replace the problem (2.2) by

$$\min_{\boldsymbol{u}} \ \sum\nolimits_{i=1}^{n} (-u_{i-1} + u_i - y_i)^2 , \ \text{s.t.} \ \|\boldsymbol{u}\|_\infty \leq \lambda, \quad u_0 = u_n = 0. \tag{2.3}$$

Now we perform a change of variables by defining the new set of variables $\boldsymbol{s} = \boldsymbol{r} - \boldsymbol{u}$, where $r_i := \sum_{k=1}^{i} y_k$ is the cumulative sum of input signal values. Thus, (2.3) becomes

$$\min_{\boldsymbol{s}} \ \sum\nolimits_{i=1}^{n} (-r_{i-1} + s_{i-1} + r_i - s_i - y_i)^2 , \ \text{s.t.} \ \|\boldsymbol{s} - \boldsymbol{r}\|_\infty \leq \lambda, \quad r_0 - s_0 = r_n - s_n = 0,$$

which upon simplification becomes

$$\min_{\boldsymbol{s}} \ \sum\nolimits_{i=1}^{n} (s_{i-1} - s_i)^2 , \quad \text{s.t.} \quad \|\boldsymbol{s} - \boldsymbol{r}\|_\infty \leq \lambda, s_0 = 0, \ s_n = r_n. \tag{2.4}$$

Now the key trick: problem (2.4) can be shown to share the same optimum as

$$\min_{\boldsymbol{s}} \ \sum_{i=1}^{n} \sqrt{1 + (s_{i-1} - s_i)^2}, \ \text{s.t.} \ \|\boldsymbol{s} - \boldsymbol{r}\|_\infty \leq \lambda, \quad s_0 = 0, \ s_n = r_n. \tag{2.5}$$

A proof of this relationship may be found in (Steidl et al., 2005); for completeness, and also because it will help us generalize to the weighted $\mathrm{Tv}_1^{\mathrm{1D}}$ variant, we include an alternative proof in Appendix C.

The name "taut-string" is explained as follows. The objective in (2.5) can be interpreted as the Euclidean length of a polyline through the points $(i, \boldsymbol{s}_i)$. Thus, (2.5) seeks the minimum length polyline (the *taut-string*) crossing a tube of height λ with center the cumulative sum $\boldsymbol{r}$ and having the fixed endpoints (s_0, s_n). An example illustrating this description is shown in Figure 1.

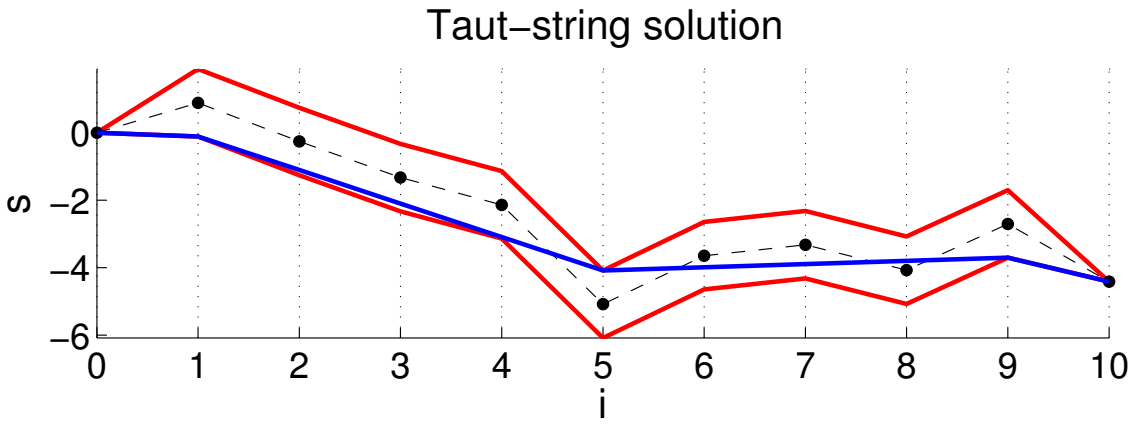


Figure 1: Example of the taut string method. The cumulative sum $\boldsymbol{r}$ of the input signal values $\boldsymbol{y}$ is shown as the dashed line; the black dots mark the points (i, r_i). The bottom and top of the λ-width tube are shown in red. The taut string solution $\boldsymbol{s}$ is shown as a blue line.

Once the taut string is found, the solution for the original TV problem (2.1) can be recovered by observing that

$$s_i - s_{i-1} \quad = \quad r_i - u_i - (r_{i-1} - u_{i-1}) \quad = \quad y_i - u_i + u_{i-1} \quad = \quad x_i,$$

where we used the primal-dual relation $\boldsymbol{x} = \boldsymbol{y} - \boldsymbol{D}^T\boldsymbol{u}$. Intuitively, the above argument shows that the solution to the TV-L1 proximity problem is obtained as the discrete gradient of the taut string, or as the slope of its segments.

It remains to describe how to find the taut string. The most widely used approach seems to be the one due to Davies and Kovac (2001). This approach starts from the fixed point $s_0 = 0$, and incrementally computes the *greatest convex minorant* of the upper bounds on the λ tube, as well as the *smallest concave majorant* of the lower bounds on the λ tube. When both curves intersect, the *left-most* point where either the majorant or the minorant touched the tube is used to fix a first segment of the taut string. The procedure is then resumed at the end of the identified segment, and iterated until all taut string segments have been obtained. Pseudocode of this method is presented as Algorithm 1, while an example of this procedure is shown in Figure 2.

It is important to note that since we have a discrete number of points in the tube, the greatest convex minorant can be expressed as a piecewise linear function with segments of monotonically increasing slope, while the smallest concave majorant is another piecewise linear function with segments of monotonically decreasing slope. Another relevant fact is that each segment in the tube upper/lower bound enters the minorant/majorant exactly once in the algorithm, and is also removed exactly once. This limits the extent of the inner loops in the algorithm, and in fact an analysis of the computational complexity of this behavior leads to an overall $O(n)$ performance (Davies and Kovac, 2001).

In spite of this, Condat (2012) notes that maintaining the minorant and majorant functions in memory is inefficient, and views a taut-string approach as potentially inferior to

Algorithm 1 Taut string algorithm for TV-L1-proximity

Inputs: input signal $\boldsymbol{y}$ of length n, regularizer λ.
Initialize $i = 0$, $concmajorant = \varnothing$, $convminorant = \varnothing$, $\boldsymbol{r}_i = \sum_{k=1}^{i} \boldsymbol{y}_k$.
while $i < n$ **do**
    Add new segment: $concmajorant = concmajorant \cup ((i-1, \boldsymbol{r}_{i-1} - \lambda) \to (i, \boldsymbol{r}_i - \lambda))$.
    while *concmajorant* is not concave **do**
        Merge the last two segments of *concmajorant*
    end while
    Add new segment: $convminorant = convminorant \cup ((i-1, \boldsymbol{r}_{i-1} + \lambda) \to (i, \boldsymbol{r}_i + \lambda))$.
    while *convminorant* is not convex **do**
        Merge the last two segments of *convminorant*
    end while
    if slope(left-most segment in *concmajorant*) > slope(lest-most segment in *convminorant*) **then**
        break = left-most point where either the majorant or the minorant touched the tube
        if *break* $\in$ *convminorant* **then**
            Remove left-most segment of the minorant, add it to the taut-string solution $\boldsymbol{x}$.
            Majorant is recalculated as a straight line from *break* to its last point.
        end if
        if *break* $\in$ *concmajorant* **then**
            Remove left-most segment of the majorant, add it to the taut-string solution $\boldsymbol{x}$.
            Minorant is recalculated as a straight line from *break* to its last point.
        end if
    end if
    $i++$
end while
Add last segment from either the majorant or minorant to the solution $\boldsymbol{x}$.

his proposed method. To this observation we make two claims: Condat's method can be interpreted as a linearized version of the taut-string method (see Section 2.2); and that a careful implementation of the taut-string method can be highly competitive in practice.

2.1.1. Efficient Implementation of Taut-Strings

We propose now an efficient implementation of the taut-string method. The main idea is to carefully use double-ended queues (Knuth, 1997) to store the majorant and minorant information. Therewith, all majorant/minorant operations such as appending a segment or removing segments from either the beginning or the end of the majorant can be performend in constant time. Note however that usual double-ended queue implementations use doubly linked lists, dynamic arrays or circular buffers: these approaches require dynamically reallocating memory chunks at some of the insert or remove operations. But in the taut-string algorithm, the maximum number of segments of the majorant/minorant is just the size of the input signal (n), and also the number of segments to be inserted in the queue throughout the algorithm will be n. Making use of these facts we implement a specialized queue based on a contiguous array of fixed length n. New segments are added from the start of the array on, and a couple of pointers are maintained to keep track of the first and last valid segments in the array, much in the way of a circular buffer. This implementation, however, does not require of the usual circular logic. Overall, this double-ended queue

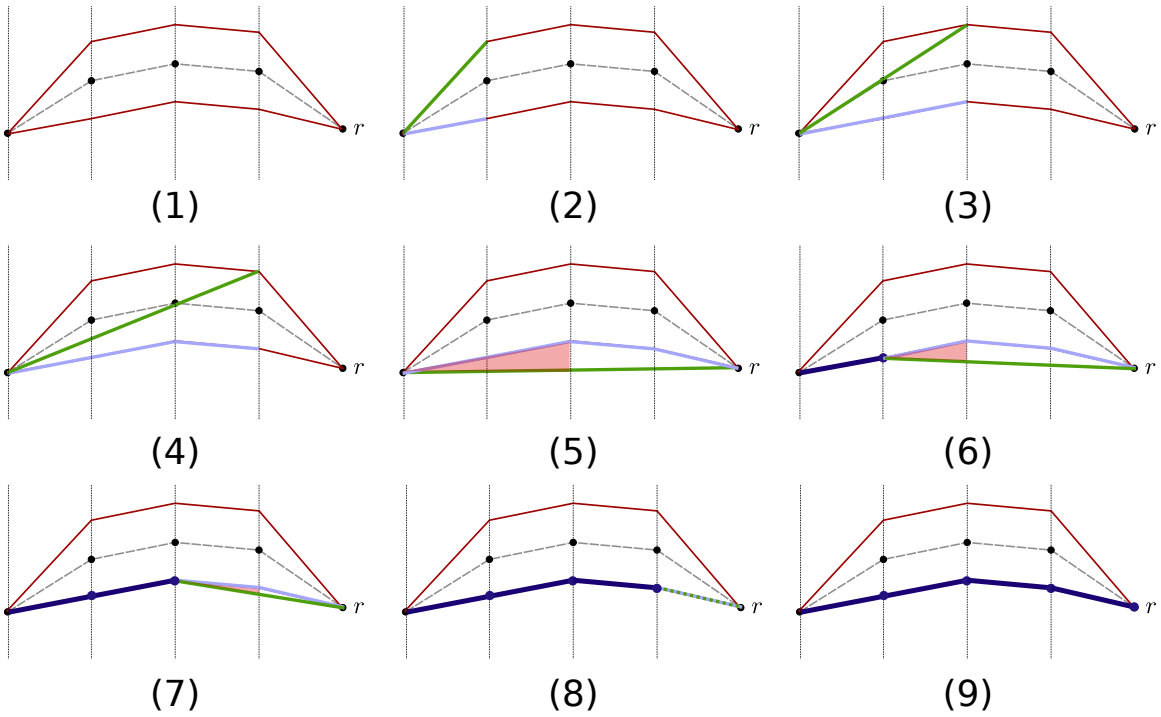


Figure 2: Example of the evolution of the taut string method. The smallest concave majorant (blue) and largest convex minorant (green) are updated are every step. At step (1) the algorithm is initialized. Steps (2) to (4) successfully manage to update majorant and minorant without producing crossings between them. Note how while the concave majorant keeps adding segments without issue, the convex minorant must remove and merge existing segments with new ones to mantain a convex function from the origin to the new points. At step (5) the end of the tube is reached, but the minorant and majorant slopes overlap, and so it is necessary to break the segment at the left-most point where the majorant/minorant touched the tube. Since the left-most touching point is in the concave majorant it's leftmost segment is removed and placed in the solution, while the convex minorant is updated as a straight line from the detected breakpoint to the last explored point, resulting in (6). The algorithm would then continue adding segments, but since the majorant/minorant slopes are still crossing, the procedure of fixing segments to the solution is repeated through steps (6), (7) and (8). Finally at step (9) the slopes are no longer crossing and the method would continue adding tube segments, but since the end of the tube has already been reached the algorithm stops.

requires a single memory allocation at the beginning of the algorithm, keeping the rest of queue operations free from memory management and all but the simplest pointer or index algebra.

We also store for each segment the following values: x length of the segment, y length and slope. Slopes might seem as redundant given the other two factors, but given the number of times the algorithm requires comparing slopes between segments (e.g., to preserve convexity/concavity) it pays off to precompute these values. This fact together with

other calculation and code optimization details produces our implementation; these can be reviewed in the code itself at `https://github.com/albarji/proxTV`.

2.2. Linearized Taut-String Method for $\mathbf{Tv}_1^{\mathbf{1D}}$

We now present a variant, linearized version of the taut-string method. Surprisingly, the resulting algorithm turns out to be equivalent to the fast algorithm of Condat (2012), though now with a clearer interpretation based on taut-strings.

The key idea is to build linear approximations to the greatest convex minorant and smallest concave majorant, producing exactly the same results but significantly reducing the bookkeeping of the method to a handful of simple variables. We therefore replace the greatest convex minorant and smallest convex majorant by a *greatest affine minorant* and *smallest affine majorant*.

An example of the method is presented in Figure 3. A proof showing that this linearization does not change the resultant taut-string is given in Appendix D. In what follows, we describe the linearized method in depth.

Details. Linearized taut-string requires only the following bookkeeping variables:

1. i_0: index of the current segment start
2. $\bar{\delta}$: slope of the majorant
3. $\underline{\delta}$: slope of the minorant
4. $\bar{h}$: height of majorant w.r.t. the λ-tube center
5. $\underline{h}$: height of minorant w.r.t. λ-tube center
6. $\bar{i}$: index of last point where $\bar{\delta}$ was updated—potential majorant break point
7. $\underline{i}$: index of last point where $\underline{\delta}$ was updated—potential minorant break point.

Figure 4 gives a geometric interpretation of these variables; we use these variables to detect minorant-majorant intersections, without the need to compute or store them explicitly.

Algorithm 2 presents full pseudocode of the linearized taut-string method. Broadly, the algorithm proceeds in the same fashion as the classic taut-string method, updating the affine approximations to the majorant and minorant at each step, and introducing a breakpoint whenever the slopes of these two functions cross.

More precisely, at each each iteration the method steps one point further through the tube, updating the minorant/majorant slopes ($\underline{\delta}$, $\bar{\delta}$) as well as their heights at the current point ($\underline{h}$, $\bar{h}$). To check for minorant/majorant crossings it suffices to compare the slopes ($\underline{\delta}$, $\bar{\delta}$), or equivalently, to check whether the height of the minorant $\underline{h}$ falls below the tube bottom (since the minorant follows the tube ceiling) or the height of the majorant $\bar{h}$ grows above the tube ceiling (since the majorant follows the tube bottom). We make use of this last variant, since updating heights turns out to be slightly cheaper than updating slopes, and so it is faster to ensure no crossing will take place before performing such updates.

When a crossing is detected, we perform similar steps as in the classic taut-string method but with one significant difference: the algorithm is completely restarted at the newly introduced breakpoint. This restart idea is in contrast with the classic method, where we simply re-use the previously computed information about the minorant and majorant to update their estimates and continue working with them. In the linearized version we do

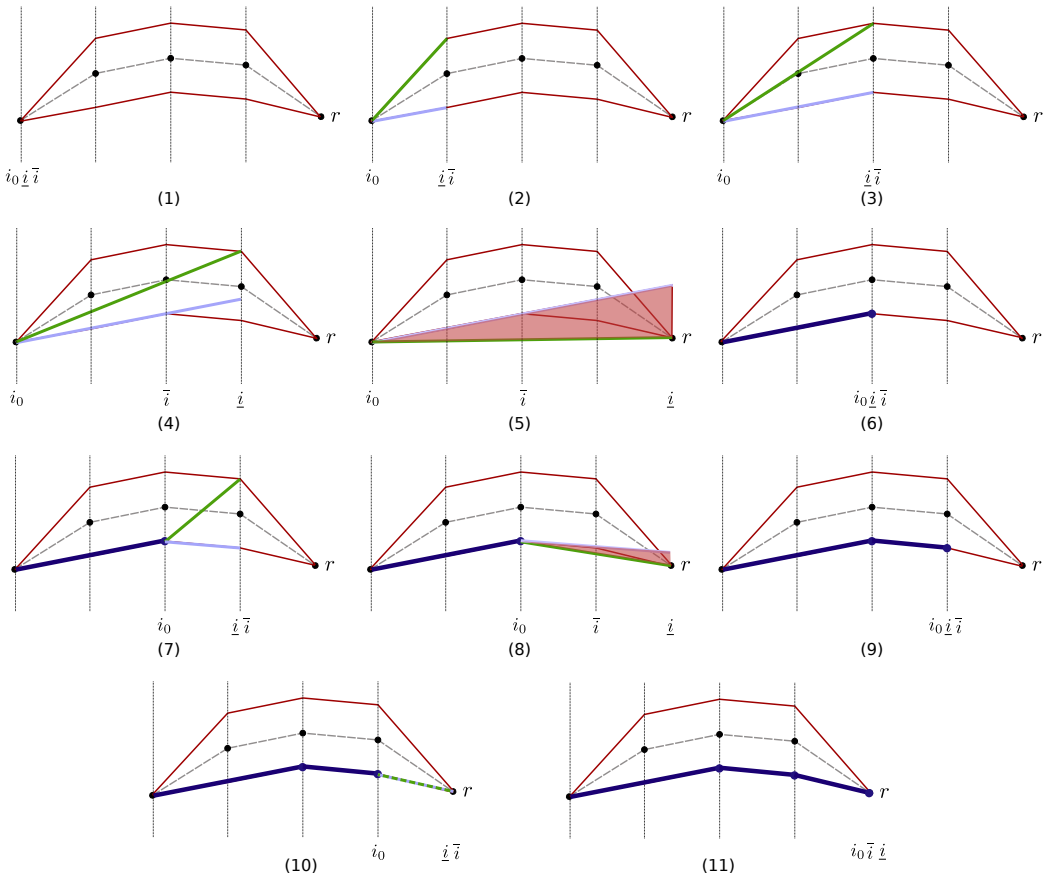


Figure 3: Example of the evolution of the linearized taut string method. The smallest affine majorant of the tube bottom (blue) and greatest affine minorant of the tube ceiling (green) are updated at every step. At step (1) the algorithm is initialized. Steps (2) to (4) successfully manage to update majorant/minorant without crossings. At step (5), however, the slopes cross, and so it is necessary to break the segment. Since the left-most tube touching point is the one in the majorant, the majorant is broken down at that point and its left-hand side is added to the solution, resulting in (6). The method is then restarted at the break point, with majorant/minorant being updated at step (7), though at step (8) once again a crossing is detected. Hence, at step (9) a breaking point is introduced again and the algorithm is restarted once more. Following this, step (10) manages to update majorant/minorant slopes up to the end of the tube, and so at step (11) the final segment is built using the (now equal) slopes.

not keep enough information to perform such an operation, so all data about minorant and majorant is discarded and the algorithm begins anew. Because of this choice the same tube segment might be reprocessed up to $O(n)$ times in the method, and therefore the overall worst case performance is $O(n^2)$. This fact was already observed in (Condat, 2012).

In what follows we describe the rationale behind the height update formulae.

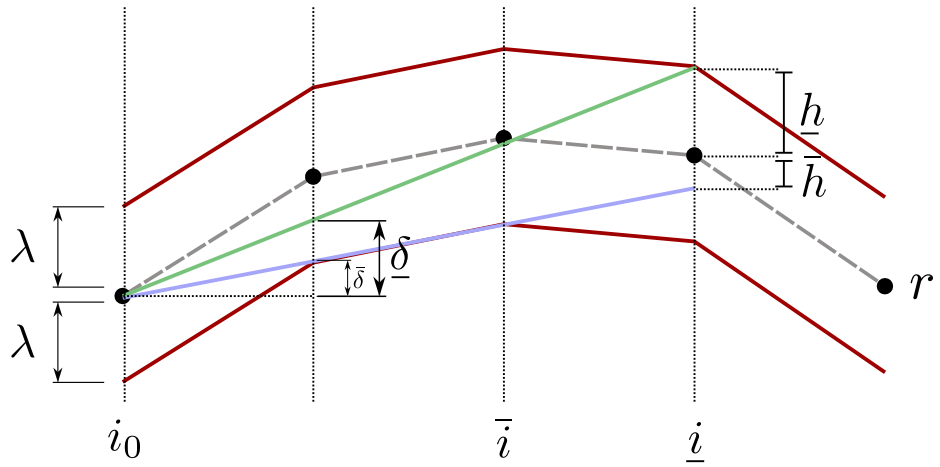


Figure 4: Illustration of the geometric concepts involved in the linearized taut string method. The greatest linear minorant (of the tube ceiling) is depicted in green, while the smallest linear majorant (of the tube bottom) is shown in blue. The δ slopes and h heights are presented updated up to the index shown as $\underline{\mathrm{i}}$.

Height variables. To implement the method described above, the height variables h are not strictly necessary as they can be obtained from the slopes δ. However, explicitly including them leads to efficient updating rules at each iteration, as we show below.

Suppose we are updating the heights and slopes from their estimates at step $i-1$ to step i. Updating the heights is immediate given the slopes, since

$$h_i = h_{i-1} + \delta - y_i.$$

In other words, since we are following a line with slope δ, the change in height from one step to the next is given by precisely such a slope. Note, however, that in this algorithm we do not compute absolute heights but instead relative heights with respect to the λ–tube center. Therefore we need to account for the change in the tube center between steps $i-1$ and i, which is given by $r_i - r_{i-1} = y_i$. This completes the update, which is shown in Algorithm 2 as lines 4 and 11.

However, it is possible that the new height h runs over or under the tube. This would mean that we cannot continue using the current slope in the majorant or minorant, and a recalculation is needed, which again can be done efficiently by using the height information. Assume without loss of generality that the starting index of the current segment is 0 and the absolute height of the starting point of the segment is given by α. Then, for adjusting the minorant slope $\bar{\delta}_i$ so that it touches the tube ceiling at the current point, we note that

$$\bar{\delta}_i = \frac{\lambda + r_i - \alpha}{i} = \frac{\lambda + (\bar{h}_i - \bar{h}_i) + r_i - \alpha}{i},$$

where we have also added and subtracted the current value of $\bar{h}_i$. Observe that this value was computed using the estimate δ_{i-1} of the slope so far, so we can rewrite it as the projection of the initial point in the segment following such a slope, that is, as $\bar{h}_i = i\bar{\delta}_i - r_i + \alpha$. Doing

Algorithm 2 Linearized taut string algorithm for TV-L1-proximity

1: Initialize $i = \bar{i} = \underline{i} = \bar{h} = \underline{h} = 0$, $\underline{\delta} = y_0 + \lambda$, $\bar{\delta} = y_0 - \lambda$
2: **while** $i < n$ **do**
3: Find tube height: $\tilde{\lambda} = \lambda$ if $i < n-1$, else $\tilde{\lambda} = 0$
4: Update majorant height following current slope: $\bar{h} = \bar{h} + \bar{\delta} - y_i$.
5: /* Check for ceiling violation: majorant is above tube ceiling */
6: **if** $\bar{h} > \tilde{\lambda}$ **then**
7: Build valid segment up to last majorant breaking point: $\boldsymbol{x}_{i_0+1:\bar{i}} = \bar{\delta}$.
8: Start new segment after break: $(i_0, \underline{i}) = \bar{i}$, $\underline{\delta} = y_i + 2\lambda$, $\bar{\delta} = y_i$, $\underline{h} = \lambda$, $\bar{h} = -\lambda$, $i = \bar{i} + 1$
9: **continue**
10: **end if**
11: Update minorant height following current slope: $\underline{h} = \underline{h} + \underline{\delta} - y_i$.
12: /* Check for bottom violation: minorant is below tube bottom */
13: **if** $\underline{h} < -\tilde{\lambda}$ **then**
14: Build valid segment up to last minorant breaking point: $\boldsymbol{x}_{i_0+1:\underline{i}} = \underline{\delta}$.
15: Start new segment after break: $(i_0, \bar{i}) = \underline{i}$, $\underline{\delta} = y_i$, $\bar{\delta} = -2\lambda + y_i$, $\underline{h} = \lambda$, $\bar{h} = -\lambda$, $i = \underline{i} + 1$
16: **continue**
17: **end if**
18: /* Check if majorant height is below the floor */
19: **if** $\bar{h} \leq -\tilde{\lambda}$ **then**
20: Correct slope: $\bar{\delta} = \bar{\delta} + \frac{\tilde{\lambda} - \bar{h}}{i - i_0}$
21: The majorant now touches the floor: $\bar{h} = -\tilde{\lambda}$
22: This is a possible majorant breaking point: $\bar{i} = i$
23: **end if**
24: /* Check if minorant height is above the ceiling */
25: **if** $\underline{h} \geq \tilde{\lambda}$ **then**
26: Correct slope: $\underline{\delta} = \underline{\delta} + \frac{-\tilde{\lambda} - \underline{h}}{i - i_0}$
27: The minorant now touches the ceiling: $\underline{h} = \tilde{\lambda}$
28: This is a possible minorant breaking point: $\underline{i} = i$
29: **end if**
30: Continue building current segment: $i = i + 1$
31: **end while**
32: Build last valid segment: $\boldsymbol{x}_{i_0+1:n} = \bar{\delta}$.

so for one of the added heights $\bar{h}_i$ produces

$$\bar{\delta}_i = \frac{\lambda + (i\bar{\delta}_{i-1} - r_i + \alpha) - \bar{h}_i + r_i - \alpha}{i} = \bar{\delta}_{i-1} + \frac{\lambda - \bar{h}_i}{i},$$

which generates a simple updating rule. A similar derivation holds for the minorant. The resulting updates are included in the algorithm in lines 20 and 26. After recomputing this slope we need to adjust the corresponding height back to the tube: since the heights are relative to the tube center we can just set $\bar{h} = \lambda$, $\underline{h} = -\lambda$; this is done in lines 21 and 27.

Notice also that the special case of the last point in the tube where the taut-string must meet $s_n = r_n$ is handled by line 3, where $\tilde{\lambda}$ is set to 0 at such a point to enforce this constraint. Overall, one iteration of the method is very efficient, as mostly just additions and subtractions are involved with the sole exception of the division required for the slope updates, which are not performed at every iteration. Moreover, no additional memory is

	Classic	**Linearized (Condat's)**
Worst-case performance	$O(n)$	$O(n^2)$
In–memory	No	Yes
Other considerations	Fast bookkeeping through double-ended queues	Very fast iteration, cache friendly

Table 1: Comparison of the main features of reviewed taut-string algorithms.

required beyond the constant number of bookkeeping variables, and in-place updates are also possible because y_i values for already fixed sections of the taut-string are not required again, so the output $\boldsymbol{x}$ and the input $\boldsymbol{y}$ can both refer to the same memory locations.

The resulting algorithm turns out to be equivalent, almost line by line, to the method of Condat (2012), even though its theoretical grounds are radically different: while the approach presented here has a strong geometric basis due to its taut-string relationship, (Condat, 2012) is based solely on analysis of KKT conditions. Therefore, we have shown that Condat's fast TV method is, in fact, a linearized taut-string algorithm.

2.3. Comparison of Taut-String Methods and a Hybrid Strategy

Table 1 summarizes the main features of the classic and linearized taut-string methods reviewed so far. Although the classic taut-string method has been largely neglected in the machine learning literature, its guarantee in linear performance makes it an attractive choice. Furthermore, although we could not find any references on implementation details of this method, we have empirically seen that a very efficient solver can be produced by making use of a double-ended queue to bookkeep the majorant/minorant information.

In contrast to this, the linearized taut-string method (equivalent to Condat (2012)) features a much better performance per step in the tube traversal, mainly due to not requiring additional memory and making use of only a small constant number of variables, making the method friendly for CPU cache or registers calculation. As a tradeoff of keeping such scarce information in memory, the method does not guarantee linear performance, falling to a quadratic theoretical runtime in the worst case. This fact was already observed in (Condat, 2012), though such worst case was deemed as pathological, claiming a $O(n)$ performance in all practical situations. We shall review these claims in the experimental sections in this manuscript.

The key points of Table 1 show that no taut-string variant is clearly superior. While the classic method provides a safe linear time solution to the problem, the linearized method is potentially faster but riskier in terms of worst case performance. Following these observations we propose here a simple hybrid method combining both approaches: run the linearized algorithm up to a prefixed number of steps n^S, $S \in (1, 2)$, and if the solution has not yet been found, we switch to the classic method. We therefore limit the worst-case scenario to $O(n^S) + O(n) \simeq O(n^S)$, because once the classic method kicks, it will ensure an $O(n)$ performance guarantee.

Implementation of this hybrid method is easy upon realizing the similarities between algorithms: a switch–check is added to the linearized method every time a segment of the taut-string has been identified (Algorithm 2, lines 7, 14). If it is confirmed that the method

has already run for n^S steps without reaching the solution, the remaining part of the signal for which the taut-string has not yet been found is passed on to the classic method, whose solution is concatenated to the part the linearized method managed to find so far. We also report the empirical performance of this method in the experimental section.

2.4. Taut-string Methods for Weighted $\mathbf{Tv}_1^{\mathbf{1D}}$

Several applications TV require penalizing the discrete gradients individually, which can be done by solving the *weighted TV-L1* problem

$$\min_{\boldsymbol{x}} \quad \tfrac{1}{2}\|\boldsymbol{x}-\boldsymbol{y}\|_2^2 + \sum_{i=1}^{n-1} w_i|x_{i+1}-x_i|, \tag{2.6}$$

where the weights $\{w_i\}_{i=1}^{n-1}$ are all positive. To solve (2.6) using a taut-string approach, we again begin with its dual (written as a minimization problem)

$$\min_{\boldsymbol{u}} \quad \tfrac{1}{2}\|\boldsymbol{D}^T\boldsymbol{u}\|_2^2 - \boldsymbol{u}^T\boldsymbol{D}\boldsymbol{y} \quad \text{s.t.} \quad |u_i| \le w_i, \quad 1 \le i < n. \tag{2.7}$$

Then, we repeat the derivation of the unweighted taut-string method but with a few key modifications. More precisely, we transform (2.7) by introducing $u_0 = u_n = 0$ to obtain

$$\min_{\boldsymbol{u}} \ \sum_{i=1}^{n} (y_i - u_i + u_{i-1})^2 \quad \text{s.t. } |u_i| \le w_i, \quad 1 \le i < n.$$

Then, we perform the change of variables $\boldsymbol{s} = \boldsymbol{r} - \boldsymbol{u}$, where $r_i := \sum_{k=1}^{i} y_k$, and consider

$$\min_{\boldsymbol{s}} \sum_{i=1}^{n} (s_i - s_{i-1})^2 \quad \text{s.t. } |s_i - r_i| \le w_i, \quad 1 \le i < n \ s_0 = 0, \ s_n = r_n.$$

Finally, applying Theorem C.1 we obtain the equivalent *weighted taut-string* problem

$$\min_{\boldsymbol{s}} \sum_{i=1}^{n} \sqrt{1 + (s_i - s_{i-1})^2} \quad \text{s.t. } |s_i - r_i| \le w_i, \ 1 \le i < n, \ s_0 = 0, \ s_n = r_n. \tag{2.8}$$

Problem (2.8) differs from its unweighted counterpart (2.5) in the constraints $|s_i - r_i| \le w_i$ $(1 \le i < n)$, which allow different weights for each component instead of using the same value λ. Our geometric intuition also carries over to the weighted problem, albeit with a slight modification: the tube we are trying to traverse now has varying widths at each step instead of the previous fixed λ width—Figure 5 illustrates this idea.

As a consequence of the above derivation and intuition, taut-string methods can be produced to solve the weighted $\mathrm{Tv}_1^{1\mathrm{D}}$ problem. The original formulation of the classic taut-string method in (Davies and Kovac, 2001) defines the limits of the tube through possibly varying bottom and ceiling values (l_i, u_i) $\forall i$, and so this method easily extends to solve the weighted TV problem by assigning $l_i = r_i - w_i$, $u_i = r_i + w_i$. In our pseudocode in Algorithm 1 we just need to replace λ by the appropriate w_i values.

Similar considerations apply for the linearized version (Algorithm 2), in particular, when checking ceiling/floor violations as well as when checking slope recomputations and restarts, we must account for varying tube heights. Algorithm 3 presents the precise modifications that we must make to Algorithm 2 to handle weights. Regarding the convergence of this method, the proof of equivalence with the classic taut-string method still holds in the weighted case (see Appendix D).

The very same analysis as portrayed in Table 1 applies here: both the benefits and problems of the two taut-string solvers carry on to the weighted variant of the problem.

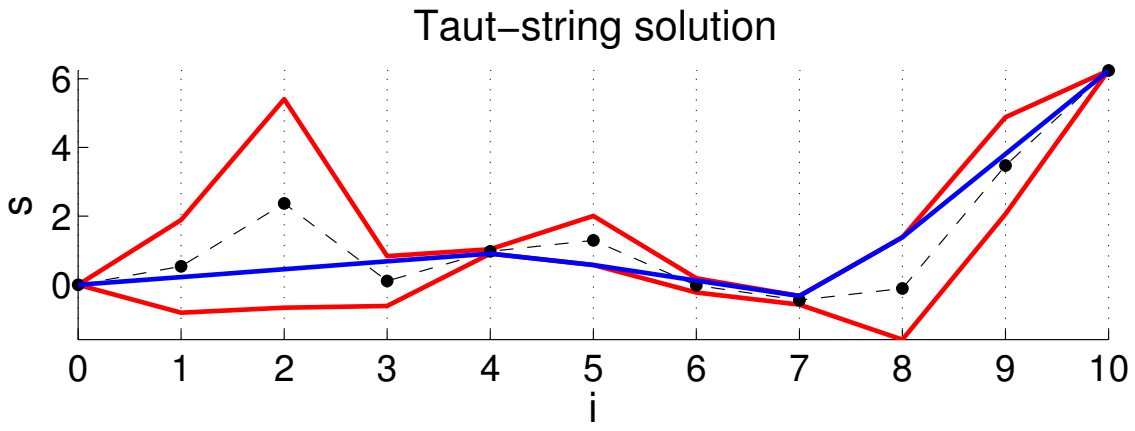


Figure 5: Example of the weighted taut string method with $\boldsymbol{w} = (1.35, 3.03, 0.73, 0.06, 0.71, 0.20, 0.12, 1.49, 1.41)$. The cumulative sum $\boldsymbol{r}$ of the input signal values $\boldsymbol{y}$ is shown as the dashed line, with the black dots marking the points $(i, \boldsymbol{r}_i)$. The bottom and ceiling of the tube are shown in red, which vary in width at each step following the weights $\boldsymbol{w}_i$. The weighted taut string solution $\boldsymbol{s}$ is shown as a blue line.

Algorithm 3 Modified lines for weighted version of Algorithm 2

3: Find tube height: $\tilde{\lambda} = w_{i+1}$ if $i < n-1$, else $\tilde{\lambda} = 0$

8: Start new segment after break: $(i_0, \underline{i}) = \bar{i}$, $\underline{\delta} = y_i + w_{i-1} + w_i$, $\bar{\delta} = y_i + w_{i-1} - w_i$, $\underline{h} = w_i$, $\bar{h} = -w_i$, $i = \bar{i} + 1$

15: Start new segment after break: $(i_0, \bar{i}) = \underline{i}$, $\underline{\delta} = y_i + w_{i-1} - w_i$, $\bar{\delta} = y_i + w_{i-1} + w_i$, $\underline{h} = w_i$, $\bar{h} = -w_i$, $i = \underline{i} + 1$

3. Other One-Dimensional TV Variants

While more infrequent, replacing the ℓ_1 norm of the standard TV regularizer by an ℓ_p-norm version can also be useful. In this section we focus first on a specialized solver for $p = 2$, before discussing a less efficient but more general solver for any ℓ_p with $p \geq 1$. We also briefly cover the $p = \infty$ case.

3.1. TV-L2: Proximity for $\mathbf{Tv}_2^{\mathbf{1D}}$

For TV-L2 proximity ($p = 2$) the dual to the prox-operator for (1.2) reduces to

$$\min_{\boldsymbol{u}} \ \phi(\boldsymbol{u}) := \tfrac{1}{2}\|\boldsymbol{D}^T\boldsymbol{u}\|_2^2 - \boldsymbol{u}^T\boldsymbol{D}\boldsymbol{y}, \quad \text{s.t. } \|\boldsymbol{u}\|_2 \leq \lambda. \tag{3.1}$$

Problem (3.1) is nothing but a version of the well-known trust-region subproblem (TRS), for which a variety of numerical approaches are known (Conn et al., 2000).

We derive a specialized algorithm based on the classic Moré-Sorensen Newton (MSN) method of (Moré and Sorensen, 1983). This method in general can be quite expensive, but for (3.1) the Hessian is tridiagonal which can be well-exploited (see Appendix E). Curiously, experiments show that for a limited range of λ values, even ordinary gradient-projection

(GP) can be competitive. But for overall best performance, a hybrid MSN-GP approach is preferable.

Towards solving (3.1), consider its KKT conditions:

$$\begin{aligned} &(\boldsymbol{D}\boldsymbol{D}^T + \alpha\boldsymbol{I})\boldsymbol{u} = \boldsymbol{D}\boldsymbol{y}, \\ &\alpha(\|\boldsymbol{u}\|_2 - \lambda) = 0, \quad \alpha \geq 0, \end{aligned} \tag{3.2}$$

where α is a Lagrange multiplier. There are two possible cases: either $\|\boldsymbol{u}\|_2 < \lambda$ or $\|\boldsymbol{u}\|_2 = \lambda$.

If $\|\boldsymbol{u}\|_2 < \lambda$, then the KKT condition $\alpha(\|\boldsymbol{u}\|_2 - \lambda) = 0$, implies that $\alpha = 0$ must hold and $\boldsymbol{u}$ can be obtained immediately by solving the linear system $\boldsymbol{D}\boldsymbol{D}^T\boldsymbol{u} = \boldsymbol{D}\boldsymbol{y}$. This can be done in $O(n)$ time owing to the bidiagonal structure of $\boldsymbol{D}$. Conversely, if the solution to $\boldsymbol{D}\boldsymbol{D}^T\boldsymbol{u} = \boldsymbol{D}\boldsymbol{y}$ lies in the interior of the ball $\|\boldsymbol{u}\|_2 \leq \lambda$, then it solves (3.2). Therefore, this case is trivial, and we need to consider only the harder case $\|\boldsymbol{u}\|_2 = \lambda$.

For any given α one can obtain the corresponding vector $\boldsymbol{u}$ as $\boldsymbol{u}_\alpha = (\boldsymbol{D}\boldsymbol{D}^T + \alpha\boldsymbol{I})^{-1}\boldsymbol{D}\boldsymbol{y}$. Therefore, optimizing for $\boldsymbol{u}$ reduces to the problem of finding the "true" value of α.

An obvious approach is to solve $\|\boldsymbol{u}_\alpha\|_2^2 = \lambda^2$. Less obvious is the *MSN equation*

$$h_\alpha := \lambda^{-1} - \|\boldsymbol{u}_\alpha\|_2^{-1} = 0, \tag{3.3}$$

which has the benefit of being almost linear in the search interval, which results in fast convergence (Moré and Sorensen, 1983). Thus, the task is to find the root of the function h_α, for which we use Newton's method, which in this case leads to the iteration

$$\alpha \leftarrow \alpha - h_\alpha / h'_\alpha. \tag{3.4}$$

Some calculation shows that the derivative h' can be computed as

$$\frac{1}{h'_\alpha} = \frac{\|\boldsymbol{u}_\alpha\|_2^3}{\boldsymbol{u}_\alpha^T(\boldsymbol{D}\boldsymbol{D}^T + \alpha\boldsymbol{I})^{-1}\boldsymbol{u}_\alpha}. \tag{3.5}$$

The key idea in MSN is to eliminate the matrix inverse in (3.5) by using the Cholesky decomposition $\boldsymbol{D}\boldsymbol{D}^T + \alpha\boldsymbol{I} = \boldsymbol{R}_\alpha^T\boldsymbol{R}_\alpha$ and defining a vector $\boldsymbol{q}_\alpha = (\boldsymbol{R}_\alpha^T)^{-1}\boldsymbol{u}$, so that $\|\boldsymbol{q}_\alpha\|_2^2 = \boldsymbol{u}_\alpha^T(\boldsymbol{D}\boldsymbol{D}^T + \alpha\boldsymbol{I})^{-1}\boldsymbol{u}_\alpha$. As a result, the Newton iteration (3.4) becomes

$$\begin{aligned} \alpha - \frac{h_\alpha}{h'_\alpha} &= \alpha - (\|\boldsymbol{u}_\alpha\|_2^{-1} - \lambda^{-1}) \cdot \frac{\|\boldsymbol{u}_\alpha\|_2^3}{\boldsymbol{u}_\alpha^T(\boldsymbol{D}\boldsymbol{D}^T + \alpha\boldsymbol{I})^{-1}\boldsymbol{u}_\alpha}, \\ &= \alpha - \frac{\|\boldsymbol{u}_\alpha\|_2^2 - \lambda^{-1}\|\boldsymbol{u}_\alpha\|_2^3}{\|\boldsymbol{q}_\alpha\|_2^2}, \\ &= \alpha - \frac{\|\boldsymbol{u}_\alpha\|_2^2}{\|\boldsymbol{q}_\alpha\|_2^2}\left(1 - \frac{\|\boldsymbol{u}_\alpha\|_2}{\lambda}\right), \end{aligned}$$

and therefore

$$\alpha \quad \leftarrow \quad \alpha - \frac{\|\boldsymbol{u}_\alpha\|_2^2}{\|\boldsymbol{q}_\alpha\|_2^2}\left(1 - \frac{\|\boldsymbol{u}_\alpha\|_2}{\lambda}\right). \tag{3.6}$$

As shown for TV-L_1 (Appendix E), the tridiagonal structure of $(\boldsymbol{D}\boldsymbol{D}^T + \alpha\boldsymbol{I})$ allows one to compute both $\boldsymbol{R}_\alpha$ and $\boldsymbol{q}_\alpha$ in linear time, so the overall iteration runs in $O(n)$ time.

Algorithm 4 MSN based TV-L2 proximity

Initialize: $\alpha = 0$, $\boldsymbol{u}_\alpha = 0$.
while $\left|\|\boldsymbol{u}_\alpha\|_2^2 - \lambda\right| > \epsilon_\lambda$ **or** $\mathrm{gap}(\boldsymbol{u}_\alpha) > \epsilon_{\mathrm{gap}}$ **do**
  Compute Cholesky decomp. $\boldsymbol{D}\boldsymbol{D}^T + \alpha\boldsymbol{I} = \boldsymbol{R}_\alpha^T\boldsymbol{R}_\alpha$.
  Obtain $\boldsymbol{u}_\alpha$ by solving $\boldsymbol{R}_\alpha^T\boldsymbol{R}_\alpha\boldsymbol{u}_\alpha = \boldsymbol{D}\boldsymbol{y}$.
  Obtain $\boldsymbol{q}_\alpha$ by solving $\boldsymbol{R}_\alpha^T\boldsymbol{q}_\alpha = \boldsymbol{u}_\alpha$.
  $\alpha = \alpha - \frac{\|\boldsymbol{u}_\alpha\|_2^2}{\|\boldsymbol{q}_\alpha\|_2^2}\left(1 - \frac{\|\boldsymbol{u}_\alpha\|_2}{\lambda}\right)$.
end while
return $\boldsymbol{u}_\alpha$

Algorithm 5 GP algorithm for TV-L_2 proximity

Initialize $\boldsymbol{u}^0 \in \mathbb{R}^N$, $t = 0$.
while ($\neg$ converged) **do**
  Gradient update: $\boldsymbol{v}^t = \boldsymbol{u}^t - \frac{1}{4}\nabla f(\boldsymbol{u}^t)$.
  Projection: $\boldsymbol{u}^{t+1} = \max(1 - \lambda/\|\boldsymbol{v}^t\|_2, 0) \cdot \boldsymbol{v}^t$.
  $t \leftarrow t + 1$.
end while
return $\boldsymbol{u}^t$.

The above ideas are presented as pseudocode in Algorithm 4. As a stopping criterion two conditions are checked: whether the duality gap is small enough, and whether $\boldsymbol{u}$ is close enough to the boundary. This latter check is useful because intermediate solutions could be dual-infeasible, thus making the duality gap an inadequate optimality measure on its own. In practice we use tolerance values $\epsilon_\lambda = 10^{-6}$ and $\epsilon_{\mathrm{gap}} = 10^{-5}$.

Even though Algorithm 4 requires only linear time per iteration, it is fairly sophisticated, and in fact a much simpler method can be devised. This is illustrated here by a gradient-projection method with a *fixed* stepsize α_0, whose iteration is

$$\boldsymbol{u}^{t+1} = P_{\|\cdot\|_2 \leq \lambda}(\boldsymbol{u}^t - \alpha_0 \nabla\phi(\boldsymbol{u}^t)). \tag{3.7}$$

The theoretically ideal choice for the stepsize α_0 is given by the inverse of the Lipschitz constant L of the gradient $\nabla\phi(\boldsymbol{u})$ (Nesterov, 2007; Beck and Teboulle, 2009). Since $\phi(\boldsymbol{u})$ is a convex quadratic, L is simply the largest eigenvalue of the Hessian $\boldsymbol{D}\boldsymbol{D}^T$. Owing to its special structure, the eigenvalues of the Hessian have closed-form expressions, namely $\lambda_i = 2 - 2\cos\left(\frac{i\pi}{n+1}\right)$ (for $1 \leq i \leq n$). The largest one is $\lambda_n = 2 - 2\cos\left(\frac{(n-1)\pi}{n}\right)$, which tends to 4 as $n \to \infty$; thus the choice $\alpha_0 = 1/4$ is a good and cheap approximation. Pseudocode showing the whole procedure is presented in Algorithm 5. Combining this with the fact that the projection $P_{\|\cdot\|_2 \leq \lambda}$ is also trivial to compute, the GP iteration (3.7) turns out to be very attractive. Indeed, sometimes it can even outperform the more sophisticated MSN method, though only for a very limited range of λ values. Therefore, in practice we recommend a hybrid of GP and MSN, as suggested by our experiments (see §5.2.1).

3.2. TV-Lp: Proximity for $\mathbf{Tv}_p^{1D}$

For TV-L_p proximity (for $1 < p < \infty$) the dual problem becomes

$$\min_{\boldsymbol{u}} \ \phi(\boldsymbol{u}) := \tfrac{1}{2}\|\boldsymbol{D}^T\boldsymbol{u}\|_2^2 - \boldsymbol{u}^T\boldsymbol{D}\boldsymbol{y}, \ \text{ s.t. } \|\boldsymbol{u}\|_q \leq \lambda, \tag{3.8}$$

where $q = 1/(1-1/p)$. Problem (3.8) is not particularly amenable to Newton-type approaches, as neither PN (Appendix E), nor MSN-type methods (§3.1) can be applied easily. It is partially amenable to gradient-projection (GP), for which the same update rule as in (3.7) applies, but unlike the $q = 2$ case, the projection step here is much more involved. Thus, to complement GP, we may favor the projection-free Frank-Wolfe (FW) method. As expected, the overall best performing approach is actually a hybrid of GP and FW. We summarize both choices below.

3.2.1. Efficient Projection onto the ℓ_q-ball

The problem of projecting onto the ℓ_q-norm ball is

$$\min_{\boldsymbol{w}} \quad d(\boldsymbol{w}) := \tfrac{1}{2}\|\boldsymbol{w}-\boldsymbol{u}\|_2^2, \quad \text{s.t. } \ \|\boldsymbol{w}\|_q \leq \lambda. \tag{3.9}$$

For this problem, it turns out to be more convenient to address its Fenchel dual

$$\min_{\boldsymbol{w}} \quad d^*(\boldsymbol{w}) := \tfrac{1}{2}\|\boldsymbol{w}-\boldsymbol{u}\|_2^2 + \lambda\|\boldsymbol{w}\|_p, \tag{3.10}$$

which is actually nothing but $\operatorname{prox}_{\lambda\|\cdot\|_p}(\boldsymbol{u})$. The optimal solution, say $\boldsymbol{w}^*$, to (3.9) can be obtained by solving (3.10), by using the Moreau-decomposition (A.6) which yields

$$\boldsymbol{w}^* = \boldsymbol{u} - \operatorname{prox}_{\lambda\|\cdot\|_p}(\boldsymbol{u}).$$

Projection (3.9) is computed many times within GP, so it is crucial to solve it rapidly and accurately. To this end, we first turn (3.10) into a differentiable problem and then derive a projected-Newton method following our approach presented in Appendix E.

Assume therefore, without loss of generality that $\boldsymbol{u} \geq 0$, so that $\boldsymbol{w} \geq 0$ also holds (the signs can be restored after solving this problem). Thus, instead of (3.10), we solve

$$\min_{\boldsymbol{w}} \quad d^*(\boldsymbol{w}) := \tfrac{1}{2}\|\boldsymbol{w}-\boldsymbol{u}\|_2^2 + \lambda\big(\textstyle\sum_i w_i^p\big)^{1/p} \quad \text{s.t. } \boldsymbol{w} \geq 0. \tag{3.11}$$

The gradient of d^* may be compactly written as

$$\nabla d^*(\boldsymbol{w}) = \boldsymbol{w} - \boldsymbol{u} + \lambda\|\boldsymbol{w}\|_p^{1-p}\boldsymbol{w}^{p-1}, \tag{3.12}$$

where $\boldsymbol{w}^{p-1}$ denotes elementwise exponentiation of $\boldsymbol{w}$. Elementary calculation yields

$$\begin{aligned}\tfrac{\partial^2}{\partial w_i \partial w_j} d^*(\boldsymbol{w}) &= \delta_{ij}\big(1 + \lambda(p-1)\big(\tfrac{w_i}{\|w\|_p}\big)^{p-2}\|\boldsymbol{w}\|_p^{-1}\big) + \lambda(1-p)\big(\tfrac{w_i}{\|w\|_p}\big)^{p-1}\big(\tfrac{w_j}{\|w\|_p}\big)^{p-1}\|\boldsymbol{w}\|_p^{-1} \\ &= \delta_{ij}\big(1 - c\hat{\boldsymbol{w}}_i^{p-2}\big) + c\bar{\boldsymbol{w}}_i\bar{\boldsymbol{w}}_j,\end{aligned}$$

where $c := \lambda(1-p)\|\boldsymbol{w}\|_p^{-1}$, $\hat{\boldsymbol{w}} := \boldsymbol{w}/\|\boldsymbol{w}\|_p$, $\bar{\boldsymbol{w}} := (\boldsymbol{w}/\|\boldsymbol{w}\|_p)^{p-1}$, and δ_{ij} is the Dirac delta. In matrix notation, this Hessian's diagonal plus rank-1 structure becomes apparent

$$\boldsymbol{H}(\boldsymbol{w}) = \operatorname{Diag}\big(1 - c\hat{\boldsymbol{w}}^{p-2}\big) + c\bar{\boldsymbol{w}} \cdot \bar{\boldsymbol{w}}^T \tag{3.13}$$

To develop an efficient Newton method it is imperative to exploit this structure. It is not hard to see that for a set of non-active variables $\bar{I}$ the reduced Hessian takes the form

$$\boldsymbol{H}_{\bar{I}}(\boldsymbol{w}) = \operatorname{Diag}\big(\mathbf{1} - c\hat{\boldsymbol{w}}_{\bar{I}}^{p-2}\big) + c\bar{\boldsymbol{w}}_{\bar{I}}\bar{\boldsymbol{w}}_{\bar{I}}^T. \tag{3.14}$$

With the shorthand $\Delta = \operatorname{Diag}\big(\mathbf{1} - c\hat{\boldsymbol{w}}_{\bar{I}}^{p-2}\big)$, the matrix-inversion lemma yields

$$\boldsymbol{H}_{\bar{I}}^{-1}(\boldsymbol{w}) = \big(\Delta + c\bar{\boldsymbol{w}}_{\bar{I}}\bar{\boldsymbol{w}}_{\bar{I}}^T\big)^{-1} = \Delta^{-1} - \frac{\Delta^{-1}c\bar{\boldsymbol{w}}_{\bar{I}}\bar{\boldsymbol{w}}_{\bar{I}}^T\Delta^{-1}}{1 + c\bar{\boldsymbol{w}}_{\bar{I}}^T\Delta^{-1}\bar{\boldsymbol{w}}_{\bar{I}}}. \tag{3.15}$$

Furthermore, since in PN the inverse of the reduced Hessian always operates on the reduced gradient, we can rearrange the terms in this operation for further efficiency; that is,

$$\boldsymbol{H}_{\bar{I}}(\boldsymbol{w})^{-1}\nabla_{\bar{I}}f(\boldsymbol{w}) = \boldsymbol{v}\odot\nabla_{\bar{I}}f(\boldsymbol{w}) - \frac{\big(\boldsymbol{v}\odot\bar{\boldsymbol{w}}_{\bar{I}}\big)\big(\boldsymbol{v}\odot\bar{\boldsymbol{w}}_{\bar{I}}\big)^T\nabla_{\bar{I}}f(\boldsymbol{w})}{1/c + \bar{\boldsymbol{w}}_{\bar{I}}\big(\boldsymbol{v}\odot\bar{\boldsymbol{w}}_{\bar{I}}\big)}, \tag{3.16}$$

where $\boldsymbol{v} := \big(\mathbf{1} - c\hat{\boldsymbol{w}}_{\bar{I}}^{p-2}\big)^{-1}$, and $\odot$ denotes componentwise product.

The relevant point of the above derivations is that the Newton direction, and thus the overall PN iteration can be computed in $O(n)$ time, which results in a highly effective solver.

3.2.2. Frank-Wolfe Algorithm for TV-L_p Proximity

The Frank-Wolfe (FW) algorithm (see e.g., Jaggi (2013) for a recent overview), also known as the conditional gradient method (Bertsekas, 1999) solves differentiable optimization problems over compact convex sets, and can be quite effective if we have access to a subroutine to solve linear problems over the constraint set.

The generic FW iteration is illustrated in Algorithm 6. FW offers an attractive strategy for TV-L_p because both the descent-direction as well as stepsizes can be computed easily. Specifically, to find the descent direction we need to solve

$$\min_{\boldsymbol{s}} \quad \boldsymbol{s}^T\left(\boldsymbol{D}\boldsymbol{D}^T\boldsymbol{u} - \boldsymbol{D}\boldsymbol{y}\right), \quad \text{s.t.} \quad \|\boldsymbol{s}\|_q \le \lambda. \tag{3.17}$$

This problem can be solved by observing that $\max_{\|\boldsymbol{s}\|_q\le 1}\boldsymbol{s}^T\boldsymbol{z}$ is attained by some vector $\boldsymbol{s}$ proportional to $\boldsymbol{z}$, of the form $|\boldsymbol{s}^*| \propto |\boldsymbol{z}|^{p-1}$. Therefore, $\boldsymbol{s}^*$ in (3.17) is found by taking $\boldsymbol{z} = \boldsymbol{D}\boldsymbol{D}^T\boldsymbol{u} - \boldsymbol{D}\boldsymbol{y}$, computing $\boldsymbol{s} = -\operatorname{sgn}(\boldsymbol{z})\odot|\boldsymbol{z}|^{p-1}$ and then rescaling $\boldsymbol{s}$ to meet $\|\boldsymbol{s}\|_q = \lambda$.

Algorithm 6 Frank-Wolfe (FW)

Inputs: f, compact convex set $\mathcal{D}$.
Initialize $\boldsymbol{x}_0 \in \mathcal{D}$, $t = 0$.
while stopping criteria not met **do**
  Find descent direction: $\min_{\boldsymbol{s}}\boldsymbol{s}\cdot\nabla f(\boldsymbol{x}_t)$ s.t. $\boldsymbol{s}\in\mathcal{D}$.
  Determine stepsize: $\min_\gamma f(\boldsymbol{x}_t + \gamma(\boldsymbol{s}-\boldsymbol{x}_t))$ s.t. $\gamma\in[0,1]$.
  Update: $\boldsymbol{x}_{t+1} = \boldsymbol{x}_t + \gamma(\boldsymbol{s}-\boldsymbol{x}_t)$
  $t \leftarrow t+1$.
end while
return $\boldsymbol{x}_t$.

The stepsize can also be computed in closed form owing to the objective function being quadratic. Note the update in FW takes the form $\boldsymbol{u} + \gamma(\boldsymbol{s} - \boldsymbol{u})$, which can be rewritten as $\boldsymbol{u} + \gamma\boldsymbol{d}$ with $\boldsymbol{d} = \boldsymbol{s} - \boldsymbol{u}$. Using this notation the optimal stepsize is obtained by solving

$$\min_{\gamma\in[0,1]} \tfrac{1}{2}\|\boldsymbol{D}^T(\boldsymbol{u} + \gamma\boldsymbol{d})\|_2^2 - (\boldsymbol{u} + \gamma\boldsymbol{d})^T\boldsymbol{D}\boldsymbol{y}.$$

A brief calculation on the above problem yields

$$\gamma^* = \min\left\{\max\left\{\hat{\gamma}, 1\right\}, 0\right\},$$

where $\hat{\gamma} = -(\boldsymbol{d}^T\boldsymbol{D}\boldsymbol{D}^T\boldsymbol{u} + \boldsymbol{d}^T\boldsymbol{D}\boldsymbol{y})/(\boldsymbol{d}^T\boldsymbol{D}\boldsymbol{D}^T\boldsymbol{d})$ is the unconstrained optimal stepsize. We note that following (Jaggi, 2013) we also check a "surrogate duality-gap"

$$g(\boldsymbol{x}) = \boldsymbol{x}^T\nabla f(\boldsymbol{x}) - \min_{\boldsymbol{s}\in\mathcal{D}} \boldsymbol{s}^T\nabla f(\boldsymbol{x}) = (\boldsymbol{x} - \boldsymbol{s}^*)^T\nabla f(\boldsymbol{x}),$$

at the end of each iteration. If this gap is smaller than the desired tolerance, the real duality gap is computed and checked; if it also meets the tolerance, the algorithm stops.

3.3. Prox Operator for TV-L$_\infty$

The final case is $\mathrm{Tv}_\infty^{1\mathrm{D}}$ proximity. We mention this case only for completeness. The dual to the prox-operator here is

$$\min_{\boldsymbol{u}} \quad \tfrac{1}{2}\|\boldsymbol{D}^T\boldsymbol{u}\|_2^2 - \boldsymbol{u}^T\boldsymbol{D}\boldsymbol{y}, \quad \text{s.t. } \|\boldsymbol{u}\|_1 \leq \lambda. \tag{3.18}$$

This problem can be again easily solved by invoking GP, where the only non-trivial step is projection onto the ℓ_1-ball. But the latter is an extremely well-studied operation (see e.g., Condat (2016); Liu and Ye (2009); Kiwiel (2008)), and so $O(n)$ time routines for this purpose are readily available. By integrating them in our GP framework an efficient prox solver is obtained.

4. Prox Operators for Multidimensional TV

We now move onto discussing how to use the efficient 1D-TV prox operators derived above within a prox-splitting framework to handle multidimensional TV (1.3) proximity.

4.1. Proximity Stacking

The basic composite objective (1.1) is a special case of the more general class of models where one may have several regularizers, so that we now solve

$$\min_{\boldsymbol{x}} \quad f(\boldsymbol{x}) + \sum\nolimits_{i=1}^{m} r_i(\boldsymbol{x}), \tag{4.1}$$

where each r_i (for $1 \leq i \leq m$) is lsc and convex.

Just like the basic problem (1.1), the more complex problem (4.1) can also be tackled via proximal methods. The key to doing so is to use *inexact proximal methods* along with a technique we should call **proximity stacking**. Inexact proximal methods allow one to use approximately computed prox operators without impeding overall convergence, while

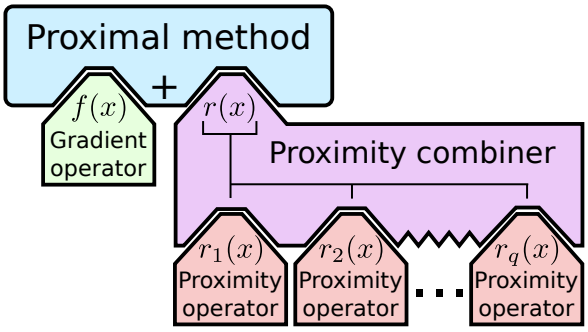


Figure 6: Design schema in proximal optimization for minimizing the function $f(\boldsymbol{x}) + \sum_{i=1}^{m} r_i(\boldsymbol{x})$. Proximal stacking makes the sum of regularizers appear as a single one to the proximal method, while retaining modularity in the design of each proximity step through the use of a combiner method. For non-smooth f the same schema applies by just replacing the f gradient operator by its corresponding proximity operator.

proximity stacking allows one to compute the prox operator for the entire sum $r(\boldsymbol{x}) = \sum_{i=1}^{m} r_i(\boldsymbol{x})$ by "stacking" the individual r_i prox operators. This stacking leads to a highly modular design; see Figure 6 for a visualization. In other words, proximity stacking involves computing the prox operator

$$\operatorname{prox}_r(\boldsymbol{y}) := \operatorname*{argmin}_{\boldsymbol{x}} \quad \tfrac{1}{2}\|\boldsymbol{x} - \boldsymbol{y}\|_2^2 + \sum\nolimits_{i=1}^{m} r_i(\boldsymbol{x}), \tag{4.2}$$

by iteratively invoking the individual prox operators $\operatorname{prox}_{r_i}$ and then combining their outputs. This mixing is done by means of a combiner method, which guarantees convergence to the solution of the overall $\operatorname{prox}_r(\boldsymbol{y})$.

Different proximal combiners can used for computing prox_r (4.2). In what follows we briefly describe some of the possibilities. The crux of all of them is that their key steps will be proximity steps over the individual r_i terms. Thus, using proximal stacking and combination, any convex machine learning problem with multiple regularizers can be solved in a highly modular proximal framework. After this section we exemplify these ideas by applying them to two- and higher-dimensional TV proximity, which we then use within proximal solvers for addressing a wide array of applications.

4.1.1. Proximal Dykstra (PD)

The Proximal Dykstra method (Combettes and Pesquet, 2009) solves problems of the form

$$\min_{\boldsymbol{x}} \quad \tfrac{1}{2}\|\boldsymbol{x} - \boldsymbol{y}\|_2^2 + r_1(\boldsymbol{x}) + r_2(\boldsymbol{x}),$$

which is a particular case of (4.2) for $m = 2$. The method follows the procedure detailed in Algorithm 7, which is guaranteed to converge to the desired solution. Using PD for proximal stacking for 2D Total-Variation was previously proposed in (Barbero and Sra, 2011).

It has also been shown that the application of this method is equivalent to performing alternating projections onto certain dual polytopes (Jegelka et al., 2013), a procedure whose

Algorithm 7 Proximal Dykstra

Inputs: r_1, r_2, input signal $\boldsymbol{y} \in \mathbb{R}^n$.
Initialize $\boldsymbol{x}_0 = \boldsymbol{y}$, $\boldsymbol{p}_0 = \boldsymbol{q}_0 = 0$, $t = 0$.
while stopping criteria not met **do**
  r_2 proximity operator: $\boldsymbol{z}_t = \text{prox}_{r_2}(\boldsymbol{x}_t + \boldsymbol{p}_t)$.
  r_2 step: $\boldsymbol{p}_{t+1} = \boldsymbol{x}_t + \boldsymbol{p}_t - \boldsymbol{z}_t$.
  r_1 proximity operator: $\boldsymbol{x}_{t+1} = \text{prox}_{r_1}(\boldsymbol{z}_t + \boldsymbol{q}_t)$.
  r_1 step: $\boldsymbol{q}_{t+1} = \boldsymbol{z}_t + \boldsymbol{q}_t - \boldsymbol{x}_{t+1}$.
  $t \leftarrow t + 1$.
end while
Return $\boldsymbol{x}_t$.

Algorithm 8 Parallel-Proximal Dykstra

Inputs: $r_1, \ldots, r_m$, input signal $\boldsymbol{y} \in \mathbb{R}^n$.
Initialize $\boldsymbol{x}_0 = \boldsymbol{y}$, $\boldsymbol{z}_0^i = 0$, for $i = 1, \ldots, m$; $t = 0$
while stopping criterion not met **do**
  for $i = 1$ to m in *parallel* **do**
    $\boldsymbol{p}_t^i = \text{prox}_{r_i}(\boldsymbol{z}_t^i)$
  end for
  $\boldsymbol{x}_{t+1} = \frac{1}{m} \sum_i \boldsymbol{p}_t^i$
  for $i = 1$ to m in *parallel* **do**
    $\boldsymbol{z}_{t+1}^i = \boldsymbol{x}_{t+1} + \boldsymbol{z}_t^i - \boldsymbol{p}_t^i$
  end for
  $t \leftarrow t + 1$
end while
Return $\boldsymbol{x}_t$

effectiveness varies depending on the relative orientation of such polytopes. A more efficient method based on reflections instead of projections is possible, as we will see below.

More generally, if more than two regularizers are present (i.e., $m > 2$), then it is more fitting to use *Parallel-Proximal Dykstra* (PPD) (Combettes, 2009) (see Alg. 8), a generalization obtained via the "product-space trick" of Pierra (1984). This parallel proximal method is attractive because it not only combines an arbitrary number of regularizers, but also allows parallelizing the calls to the individual prox operators. This feature allows us to develop a highly parallel implementation for multidimensional TV proximity (§4.3).

4.1.2. Alternating Reflections – Douglas-Rachford (DR)

The Douglas-Rachford (DR) method was originally devised for minimizing the sum of two (nonsmooth) convex functions (Combettes and Pesquet, 2009), in the form:

$$\min_{\boldsymbol{x}} \quad f_1(\boldsymbol{x}) + f_2(\boldsymbol{x}), \tag{4.3}$$

such that $(\text{ri dom } f_1) \cap (\text{ri dom } f_2) \neq \emptyset$. The method operates by iterating a series of reflections, and in its simplest form can be written as

$$\boldsymbol{z}_{k+1} = \tfrac{1}{2}\left[R_{f_1} R_{f_2} + I\right] \boldsymbol{z}_k, \tag{4.4}$$

where the *reflection operator* $R_\phi := 2\,\text{prox}_\phi - I$. This method is not cleanly applicable to problem (4.2) because of the squared norm term. Nevertheless in (Jegelka et al., 2013) a suitable transformation was proposed by making use of arguments from submodular optimization; a minimal background on this topic is given in Appendix A. We summarize the key ideas from (Jegelka et al., 2013) below.

Assume $m = 2$ and r_1, r_2 being Lovász extensions to some submodular functions (Total-Variation is the Lovász extension of a submodular graph-cut problem, see Bach (2013)). Defining $\hat{r}_1(\boldsymbol{x}) = r_1(\boldsymbol{x}) - \boldsymbol{x}^T\boldsymbol{y}$, $\hat{r}_1$ is also a Lovász extension of some submodular function (see Appendix A). Therefore, we may consider the problem

$$\text{prox}_r(\boldsymbol{y}) := \operatorname*{argmin}_{\boldsymbol{x}} \quad \tfrac{1}{2}\|\boldsymbol{x}\|_2^2 + \hat{r}_1(\boldsymbol{x}) + r_2(\boldsymbol{x}),$$

which can be rewritten (using Proposition A.11) as

$$\min_{a,b} \|a - b\|_2, \quad \text{s.t.} \quad a \in -B_{\hat{r}_1},\, b \in B_{r_2}, \tag{4.5}$$

where B_r denotes the base polytope of submodular function corresponding to r (see Appendix A). The original solution can be recovered through $\boldsymbol{x} = \boldsymbol{a} - \boldsymbol{b}$. Problem (4.5) is still not in a form amenable to DR (4.3)—nevertheless, if we apply DR to the indicator functions of the sets $-B_{\hat{r}_1}, B_{r_2}$, that is, to the problem

$$\min_{\boldsymbol{x}} \quad \delta_{-B_{\hat{r}_1}}(\boldsymbol{x}) + \delta_{B_{r_2}}(\boldsymbol{x}),$$

it can be shown (Bauschke, 2004) that the sequence (4.4) generated by DR is divergent, but that after a correction through projection converges to the desired solution of (4.5). Such solution is given by the pair

$$\boldsymbol{b} = \Pi_{B_{r_2}}(\boldsymbol{z}_k), \quad \boldsymbol{a} = \Pi_{-B_{\hat{r}_1}}(\boldsymbol{b}). \tag{4.6}$$

Although in this derivation many concepts have been introduced, suprisingly all the operations in the algorithm can be reduced to performing proximity steps. Note first that the projections onto a base polytope required to get a solution (4.6) can be written in terms of proximity operators (Proposition A.12), which in this case implies

$$\begin{aligned}\Pi_{B_{r_2}}(\boldsymbol{z}) &= \boldsymbol{z} - \text{prox}_{r_2}(\boldsymbol{z}),\\ \Pi_{-B_{\hat{r}_1}}(\boldsymbol{z}) &= \boldsymbol{z} + \text{prox}_{\hat{r}_2}(-\boldsymbol{z}) = \boldsymbol{z} + \text{prox}_{r_2}(-\boldsymbol{z} + \boldsymbol{y}),\end{aligned}$$

where we use the fact that for $f(\boldsymbol{x}) = \phi(\boldsymbol{x}) + \boldsymbol{u}^T\boldsymbol{x}$, $\text{prox}_f(\boldsymbol{x}) = \text{prox}_\phi(\boldsymbol{x} - \boldsymbol{u})$. The reflection operations in which the DR iteration is based (4.4) can also be written in terms of proximity steps, as we are applying DR to the indicator functions $\delta_{-B_{\hat{r}_1}}, \delta_{B_{r_2}}$, and proximity for an indicator function equals projection.

This alternating reflections variant of DR is presented in Algorithm 9. Note that in contrast with the original DR method, this variant does not require tuning any hyperparameters, thus enhancing its practicality.

Algorithm 9 Alternating reflections – Douglas Rachford (DR)

Inputs: r_1, r_2 Lovász extensions of some submodular function, input signal $\boldsymbol{y} \in \mathbb{R}^n$.
Initialize $\boldsymbol{z}_0 \in \mathbb{R}^n$, $t = 0$.
Define the following operations:
$\Pi_{-B_{\hat{r}_1}}(\boldsymbol{z}) \stackrel{\text{def}}{=} \boldsymbol{z} + \text{prox}_{r_1}(-\boldsymbol{z} + \boldsymbol{y})$.
$\Pi_{B_{r_2}}(\boldsymbol{z}) \stackrel{\text{def}}{=} \boldsymbol{z} - \text{prox}_{r_2}(\boldsymbol{z})$.
$R_{-B_{\hat{r}_1}}(\boldsymbol{z}) \stackrel{\text{def}}{=} 2\Pi_{-B_{\hat{r}_1}}(\boldsymbol{z}) - \boldsymbol{z}$.
$R_{B_{r_2}}(\boldsymbol{z}) \stackrel{\text{def}}{=} 2\Pi_{B_{r_2}}(\boldsymbol{z}) - \boldsymbol{z}$.
while stopping criteria not met **do**
$\boldsymbol{z}_{t+1} = \frac{1}{2}\left[R_{-B_{\hat{r}_1}} R_{B_{r_2}} + I\right] \boldsymbol{z}_k$
$t \leftarrow t + 1$.
end while
$\boldsymbol{b} = \Pi_{B_{r_2}}(z_t), \quad \boldsymbol{a} = \Pi_{-B_{\hat{r}_1}}(\boldsymbol{b})$.
Return $\boldsymbol{x}^* = \boldsymbol{a} - \boldsymbol{b}$.

4.1.3. Alternating-Direction Method of Multipliers (ADMM)

Although many times presented as a particular algorithm for solving problems involving the minimization of a certain objetive $f(x) + g(Lx)$ with L a linear operator (Combettes and Pesquet, 2009), the Alternating-Direction Method of Multipliers can be thought as a general splitting strategy for solving the unconstrained minimization of a sum of functions. This strategy boils down to transforming a problem in the form $\min_{\boldsymbol{x}} \sum_{i=1}^m f_i(\boldsymbol{x})$ into a saddle-point problem by introducing consensus constraints and incorporating them into the objective through augmented Lagrange multipliers,

$$\begin{aligned}\min_{\boldsymbol{x}} \quad \sum_{i=1}^m f_i(\boldsymbol{x}) &= \min_{\boldsymbol{x},\boldsymbol{z}_1,\ldots,\boldsymbol{z}_m} \quad \sum_{i=1}^m f_i(\boldsymbol{z}_i) \quad \text{s.t. } \boldsymbol{z}_1 = \boldsymbol{x}, \ldots, \boldsymbol{z}_m = \boldsymbol{x}, \\ &\equiv \min_{\boldsymbol{x},\boldsymbol{z}_1,\ldots,\boldsymbol{z}_m} \max_{\boldsymbol{u}_1,\ldots,\boldsymbol{u}_m} \quad \sum_{i=1}^m \left(f_i(\boldsymbol{z}_i) + \boldsymbol{u}_i^T(\boldsymbol{z}_i - \boldsymbol{x}) + \frac{\rho}{2}\|\boldsymbol{z}_i - \boldsymbol{x}\|_2 \right).\end{aligned}$$

The method then proceeds to solve this problem by alternating steps of minimization on $\boldsymbol{x}$, minimization on every $\boldsymbol{z}_i$, and a gradient step on every $\boldsymbol{u}_i$.

In (Yang et al., 2013) a proposal using this method was presented to solve m–dimensional anisotropic TV (1.3). This approach applies equally to the more general proximal stacking framework under discussion here (4.2), by the transformation

$$\begin{aligned}\text{prox}_r(\boldsymbol{y}) := \underset{\boldsymbol{x}}{\text{argmin}} \quad & \tfrac{1}{2}\|\boldsymbol{x} - \boldsymbol{y}\|_2^2 + \textstyle\sum_{i=1}^m r_i(\boldsymbol{x}), \\ \equiv \min_{\boldsymbol{x},\boldsymbol{z}_1,\ldots,\boldsymbol{z}_m} \max_{\boldsymbol{u}_1,\ldots,\boldsymbol{u}_m} \quad & \tfrac{1}{2}\|\boldsymbol{x} - \boldsymbol{y}\|_2^2 + \sum_{i=1}^m \left(f_i(\boldsymbol{z}_i) + \boldsymbol{u}_i^T(\boldsymbol{z}_i - \boldsymbol{x}) + \frac{\rho}{2}\|\boldsymbol{z}_i - \boldsymbol{x}\|_2 \right).\end{aligned}$$

The steps for obtaining a solution then follow as Algorithm 10. Similar to Parallel Proximal Dykstra, this approach allows computing the prox-operator of each function r_i in parallel.

Algorithm 10 Alternating Direction Method of Multipliers (ADMM)

Inputs: $r_1, \ldots, r_m$, input signal $\boldsymbol{y} \in \mathbb{R}^n$.
Initialize $\boldsymbol{x}_0 = \boldsymbol{z}_0^i = \boldsymbol{y}$ for $i = 1, \ldots, m$; $t = 0$
while stopping criterion not met **do**
  $\boldsymbol{x}_{t+1} = \frac{\boldsymbol{y} + \sum_{i=1}^m (\boldsymbol{u}_t^i + \rho \boldsymbol{z}_t^i)}{1 + m\rho}$.
  for $i = 1$ to m in *parallel* **do**
    $\boldsymbol{z}_t^i = \operatorname{prox}_{\frac{\lambda}{\rho} r_i}(-\frac{1}{\rho}\boldsymbol{u}_t^i + \boldsymbol{x}_{t+1})$
    $\boldsymbol{u}_{t+1}^i = \boldsymbol{u}_{t+1} + \rho(\boldsymbol{z}_{t+1}^i - \boldsymbol{x}_{t+1})$
  end for
  $t \leftarrow t + 1$
end while
Return $\boldsymbol{x}_t$

4.1.4. Dual Proximity Methods

Another family of approaches to solve (4.2) is to compute the global proximity operator using the Fenchel duals $\operatorname{prox}_{r_i^*}$. This can be advantageous in settings where the dual prox-operator is easier to compute than the primal operator; isotropic Total-Variation problems are an instance of such a setting, and thus investigating this approach for their anisotropic variants is worthwhile.

Indeed, in the context of image processing a popular splitting approach is given by Chambolle and Pock (2011), which consider a problem in the form

$$\min_{\boldsymbol{x}} \quad F(\boldsymbol{K}\boldsymbol{x}) + G(\boldsymbol{x}),$$

for $\boldsymbol{K}$ some linear operator, F, G convex lower-semicontinuous functions. Through a strategy similar to ADMM an equivalent saddle point problem can be obtained,

$$\min_{\boldsymbol{x}} \max_{\boldsymbol{y}} \quad (\boldsymbol{K}\boldsymbol{x})^T \boldsymbol{y} + G(\boldsymbol{x}) - F^*(\boldsymbol{y}),$$

with F^* convex conjugate of F. This problem is then solved by alternating maximization on $\boldsymbol{y}$ and minimization on $\boldsymbol{x}$ through proximity steps, as

$$\begin{aligned}
\boldsymbol{y}_{t+1} &= \operatorname{prox}_{\sigma F^*}(\boldsymbol{y}_t + \sigma \boldsymbol{K} \bar{\boldsymbol{x}}_t) \\
\boldsymbol{x}_{t+1} &= \operatorname{prox}_{\tau G}(\boldsymbol{x}_t - \tau \boldsymbol{K}^* \boldsymbol{y}_{t+1}) \\
\bar{\boldsymbol{x}}_{t+1} &= \boldsymbol{x}_{t+1} + \theta(\boldsymbol{x}_{t+1} - \boldsymbol{x}_t),
\end{aligned}$$

where $\boldsymbol{K}^*$ is the conjugate transpose of $\boldsymbol{K}$. σ, τ and θ are algorithm parameters that should be either selected under some bounds (Chambolle and Pock, 2011, Algorithm 1) or readjusted every iteration making use of Lipschitz convexity of G (Chambolle and Pock, 2011, Algorithm 2), resulting in an accelerating scheme much in the style of FISTA (Beck and Teboulle, 2009). The overall procedure can also be shown to be an instance of preconditioned ADMM, where the preconditioning is given by the application of a proximity step for the maximization of $\boldsymbol{y}$ (instead of the usal dual gradient step of ADMM) and the auxiliary point $\bar{\boldsymbol{x}}$. Note also how proximity is computed over the dual F^* instead of the primal prox_F.

Now, this decomposition strategy can be applied for some instances of proximal stacking (4.2) when the r_i terms allow the particular composition

$$\sum_{i=1}^{m} r_i(\boldsymbol{x}) = F\left(\begin{bmatrix} \boldsymbol{K}_1 \\ \vdots \\ \boldsymbol{K}_m \end{bmatrix} \boldsymbol{x}\right) = F(\boldsymbol{K}\boldsymbol{x}),$$

which does not hold in general but holds for 2D TV (1.4) when taking the identities

$$F(\boldsymbol{x}) = \|\boldsymbol{x}\|_1, G(\boldsymbol{x}) = \tfrac{1}{2}\|\boldsymbol{x} - \boldsymbol{y}\|_2^2,$$
$$\boldsymbol{K} = \begin{bmatrix} \boldsymbol{I} \otimes \boldsymbol{D} \\ \boldsymbol{D} \otimes \boldsymbol{I} \end{bmatrix},$$

with $\boldsymbol{D}$ the differencing matrix as before, $\otimes$ denotes Kronecker product, and $\boldsymbol{x}$ a vectorization of the 2D input. The iterates above can then be applied easily: proximity over G is trivial and proximity over F^* is also easy upon realizing that $\text{prox}_{\|\cdot\|_1^*} = \text{prox}_{\delta_{\|\cdot\|_\infty \leq 1}} = \Pi_{\|\cdot\|_\infty \leq 1}$, which is solved through thresholding.

A generalization of this approach is presented by Condat (2014), who considers

$$\min_{\boldsymbol{x}} \quad f(\boldsymbol{x}) + g(\boldsymbol{x}) + \sum_{i=1}^{m} r_i(\boldsymbol{L}_i\boldsymbol{x}),$$

a problem that cleanly fits into (4.2) with $f(\boldsymbol{x}) = \frac{1}{2}\|\boldsymbol{x} - \boldsymbol{y}\|_2^2$, $g(\boldsymbol{x}) = 0$, $\boldsymbol{L} = \boldsymbol{I}$. The procedure to find a solution is proposed as

$$\begin{aligned}
\bar{\boldsymbol{x}}^{t+1} &= \text{prox}_{\tau g^*}\left(\boldsymbol{x}^t - \tau\nabla f(\boldsymbol{x}^t) - \tau\sum_{i=1}^{m} \boldsymbol{L}_i^*\boldsymbol{u}_i^t\right) \\
\boldsymbol{x}_{n+1} &= \rho\bar{\boldsymbol{x}}^{t+1} + (1-\rho)\boldsymbol{x}^t \\
\bar{\boldsymbol{u}}_i^{t+1} &= \text{prox}_{\sigma h_i^*}(\boldsymbol{u}_i^t + \sigma\boldsymbol{L}_i(2\bar{\boldsymbol{x}}_{t+1} - \boldsymbol{x}_t)) \quad \forall i = 1, \ldots, m\,, \\
\boldsymbol{u}_i^{t+1} &= \rho\bar{\boldsymbol{u}}_i^{t+1} + (1-\rho)\boldsymbol{u}_i^t \quad \forall i = 1, \ldots, m\,,
\end{aligned}$$

for τ, ρ parameters of the algorithm. When applying this procedure to 2D TV ($m = 2$, $r_1(\boldsymbol{x})$ = proximity over rows, $r_2(\boldsymbol{x})$ = proximity over columns) an algorithm almost equivalent to Chambolle and Pock (2011) is obtained, the only difference being that here the gradient of f is used, instead of the prox_G operation.

Yet another related method is the splitting approach of Kolmogorov et al (2015), which for $m = 2$ performs the following splitting:

$$\begin{aligned}
&\min_{\boldsymbol{x}} \quad \tfrac{1}{2}\|\boldsymbol{x} - \boldsymbol{y}\|_2^2 + r_1(\boldsymbol{x}) + r_2(\boldsymbol{x}), \\
\equiv &\min_{\boldsymbol{x},\boldsymbol{x}'} \quad \|\boldsymbol{x} - \boldsymbol{y}\|_2^2 + r_1(\boldsymbol{x}) + r_2(\boldsymbol{x}') \quad \text{s.t. } \boldsymbol{x} = \boldsymbol{x}', \\
\equiv &\min_{\boldsymbol{x},\boldsymbol{x}'}\max_{\boldsymbol{z}} \quad \|\boldsymbol{x} - \boldsymbol{y}\|_2^2 + r_1(\boldsymbol{x}) + r_2(\boldsymbol{x}') + \boldsymbol{z}^T(\boldsymbol{x} - \boldsymbol{x}'), \\
\equiv &\min_{\boldsymbol{x}}\max_{\boldsymbol{z}} \quad \|\boldsymbol{x} - \boldsymbol{y}\|_2^2 + r_1(\boldsymbol{x}) - r_2^*(\boldsymbol{z}) + \boldsymbol{x}^T\boldsymbol{z}.
\end{aligned}$$

where we have made use of the Fenchel dual $r_2^*(\boldsymbol{z}) = \max_{\boldsymbol{x}'} \boldsymbol{z}^T\boldsymbol{x}' - r_2(\boldsymbol{x}')$. This problem can be solved through a primal-dual minimization:

$$\begin{aligned}\boldsymbol{z}^{t+1} &= \operatorname{prox}_{\sigma^t r_2^*}\left(\boldsymbol{z}^t + \sigma^t(\boldsymbol{x}^t + \theta^t(\boldsymbol{x}^t - \boldsymbol{x}^{t-1}))\right),\\ \boldsymbol{x}^{t+1} &= \operatorname{prox}_{\tau^t(\|\cdot-\boldsymbol{y}\|_2^2+r_1)}\left(\boldsymbol{x}^t - \tau^t\boldsymbol{z}^{t+1}\right).\end{aligned}$$

The primal proximity operator over the squared norm term plus r_1 can be rewritten in terms of $\operatorname{prox}_{r_1}$ as

$$\begin{aligned}\operatorname{prox}_{\tau(r_1+\frac{1}{2}\|\cdot-\boldsymbol{y}\|_2^2)}(\boldsymbol{w}) &= \operatorname*{argmin}_{\boldsymbol{x}} r_1(\boldsymbol{x}) + \frac{1+\tau^{-1}}{2}\|\boldsymbol{x} - (1+\tau^{-1})^{-1}(\boldsymbol{y}+\tau^{-1}\boldsymbol{w})\|_2^2,\\ &= \operatorname{prox}_{(1+\tau^{-1})^{-1}r_1}\left((1+\tau^{-1})^{-1}(\boldsymbol{y}+\tau^{-1}\boldsymbol{w})\right).\end{aligned}$$

Regarding the dual step, in the previously presented methods the decompositions allowed to disentangle the effect of a linear operator L_i from each r_i. The present decomposition, however, does not take into account this possibility, thus increasing the complexity of computing r_2^*. To address this difficulty the Moreau decomposition (A.3) is helpful, as

$$\begin{aligned}\operatorname{prox}_{\sigma r_2^*}(\boldsymbol{w}) &= \boldsymbol{w} - \sigma\left(\operatorname*{argmin}_{\boldsymbol{x}} r_2(\boldsymbol{x}) + \frac{\sigma}{2}\|\boldsymbol{x} - \sigma^{-1}\boldsymbol{w}\|_2^2\right),\\ &= \boldsymbol{w} - \sigma\operatorname{prox}_{\sigma^{-1}r_2}(\sigma^{-1}\boldsymbol{w}),\end{aligned}$$

thus solving the dual proximity operator in terms of the primal $\operatorname{prox}_{r_2}$. Regarding the algorithm parameters θ, τ and σ, they can be adjusted at every iteration for greater performance making use of Lipschitz convexity (Chambolle and Pock, 2014).

Lastly, and again for $m = 2$, both r_1 and r_2 can be exploited in their dual forms as shown in Chambolle and Pock (2015) through the splitting

$$\begin{aligned}&\min_{\boldsymbol{x}} \quad \tfrac{1}{2}\|\boldsymbol{x}-\boldsymbol{y}\|_2^2 + r_1(\boldsymbol{x}) + r_2(\boldsymbol{x}),\\ \equiv &\min_{\boldsymbol{x},\boldsymbol{x}_1,\boldsymbol{x}_2} \quad \tfrac{1}{2}\|\boldsymbol{x}-\boldsymbol{y}\|_2^2 + r_1(\boldsymbol{x}_1) + r_2(\boldsymbol{x}_2) \quad \text{s.t. } \boldsymbol{x}=\boldsymbol{x}_1, \boldsymbol{x}=\boldsymbol{x}_2\\ \equiv &\min_{\boldsymbol{x},\boldsymbol{x}_1,\boldsymbol{x}_2}\max_{\boldsymbol{z}_1,\boldsymbol{z}_2} \quad \tfrac{1}{2}\|\boldsymbol{x}-\boldsymbol{y}\|_2^2 + r_1(\boldsymbol{x}_1) + \boldsymbol{z}_1^T(\boldsymbol{x}-\boldsymbol{x}_1) + r_2(\boldsymbol{x}_2) + \boldsymbol{z}_2^T(\boldsymbol{x}-\boldsymbol{x}_2).\end{aligned}$$

Minimizing this Lagrangian over $\boldsymbol{x}, \boldsymbol{x}_1, \boldsymbol{x}_2$ and making use of Fenchel duals we arrive at

$$\max_{\boldsymbol{z}_1,\boldsymbol{z}_2} \quad -\tfrac{1}{2}\|\boldsymbol{z}_1+\boldsymbol{z}_2\|_2^2 - r_1^*(\boldsymbol{u}_1) - r_2(\boldsymbol{u}_2^*) + (\boldsymbol{u}_1+\boldsymbol{u}_2)^T\boldsymbol{y},$$

which can be solved through an accelerated alternating minimization as

$$\begin{aligned}t_{k+1} &= \frac{1+\sqrt{1+4t_k^2}}{2},\\ \bar{\boldsymbol{x}}^{k+1} &= \boldsymbol{x}_2^k + \frac{t_k-1}{t_{k+1}}(\boldsymbol{x}_2^k - \boldsymbol{x}_2^{k-1}),\\ \boldsymbol{x}_1^{k+1} &= \operatorname{prox}_{r_1^*}(y - \bar{\boldsymbol{x}}_2^k),\\ \boldsymbol{x}_2^{k+1} &= \operatorname{prox}_{r_2^*}(y - \boldsymbol{x}_1^{k+1}),\end{aligned}$$

where once again we can resort to the Moreau decomposition to compute the dual proximity operators.

4.2. Two-Dimensional TV

Recall that for a matrix $\boldsymbol{X} \in \mathbb{R}^{n_1 \times n_2}$, the anisotropic 2D-TV regularizer takes the form

$$\mathrm{Tv}^2_{p,q}(\boldsymbol{X}) := \sum\nolimits_{i=1}^{n_1} \Big(\sum\nolimits_{j=1}^{n_2-1} |x_{i,j+1} - x_{i,j}|^p \Big)^{1/p} + \sum\nolimits_{j=1}^{n_2} \Big(\sum\nolimits_{i=1}^{n_1-1} |x_{i+1,j} - x_{i,j}|^q \Big)^{1/q}. \quad (4.7)$$

This regularizer applies a $\mathrm{Tv}^{1\mathrm{D}}_p$ regularization over each row of $\boldsymbol{X}$, and a $\mathrm{Tv}^{1\mathrm{D}}_q$ regularization over each column. Introducing differencing matrices $\boldsymbol{D}_n$ and $\boldsymbol{D}_m$ for the row and column dimensions, the regularizer (4.7) can be rewritten as

$$\mathrm{Tv}^{2\mathrm{D}}_{p,q}(\boldsymbol{X}) = \sum\nolimits_{i=1}^{n} \|\boldsymbol{D}_n \boldsymbol{x}_{i,:}\|_p + \sum\nolimits_{j=1}^{m} \|\boldsymbol{D}_m \boldsymbol{x}_{:,j}\|_q, \quad (4.8)$$

where $\boldsymbol{x}_{i,:}$ denotes the i-th row of $\boldsymbol{X}$, and $\boldsymbol{x}_{:,j}$ its j-th column. The corresponding $\mathrm{Tv}^{2\mathrm{D}}_{p,q}$-proximity problem is

$$\min\nolimits_{\boldsymbol{X}} \quad \tfrac{1}{2}\|\boldsymbol{X} - \boldsymbol{Y}\|^2_{\mathrm{F}} + \lambda\, \mathrm{Tv}^{2\mathrm{D}}_{p,q}(\boldsymbol{X}), \quad (4.9)$$

where we use the Frobenius norm $\|\boldsymbol{X}\|_{\mathrm{F}} = \sqrt{\sum_{ij} x^2_{i,j}} = \|\mathrm{vec}(\boldsymbol{X})\|_2$, where $\mathrm{vec}(\boldsymbol{X})$ is the vectorization of $\boldsymbol{X}$. Using (4.8), problem (4.9) becomes

$$\min\nolimits_{\boldsymbol{X}} \quad \tfrac{1}{2}\|\boldsymbol{X} - \boldsymbol{Y}\|^2_{\mathrm{F}} + \lambda \Big(\sum\nolimits_i \|\boldsymbol{D}_n x_{i,:}\|_p \Big) + \lambda \Big(\sum\nolimits_j \|\boldsymbol{D}_m x_{:,j}\|_q \Big), \quad (4.10)$$

where the parentheses make explicit that $\mathrm{Tv}^{2\mathrm{D}}_{p,q}$ is a combination of two regularizers: one acting over the rows and the other over the columns. Formulation (4.10) fits the model solvable by the strategies presented above, though with an important difference: each of the two regularizers that make up $\mathrm{Tv}^{2\mathrm{D}}_{p,q}$ is itself composed of a sum of several (n or m) 1D-TV regularizers. Moreover, each of the 1D row (column) regularizers operates on a different row (columns), and can thus be solved independently.

4.3. Higher-Dimensional TV

Going even beyond $\mathrm{Tv}^{2\mathrm{D}}_{p,q}$ is the general multidimensional TV (1.3), which we recall below.

Let X be an order-m tensor in $\mathbb{R}^{\prod_{j=1}^m n_j}$, whose components are indexed as $\mathsf{X}_{i_1,i_2,\ldots,i_m}$ ($1 \le i_j \le n_j$ for $1 \le j \le m$); we define TV for X as

$$\mathrm{Tv}^m_{\boldsymbol{p}}(\mathsf{X}) \overset{\text{def}}{=} \sum_{k=1}^{m} \sum_{\{i_1,\ldots,i_m\}\setminus i_k} \Big(\sum_{j=1}^{n_k-1} |\mathsf{X}_{i_1,\ldots,i_{k-1},j+1,i_{k+1},\ldots,i_m} - \mathsf{X}_{i_1,\ldots,i_{k-1},j,i_{k+1},\ldots,i_m}|^{p_k} \Big)^{1/p_k}, \quad (4.11)$$

where $\boldsymbol{p} = [p_1, \ldots, p_m]$ is a vector of scalars $p_k \ge 1$. This corresponds to applying a 1D-TV to each of the 1D fibers of X along each of the dimensions.

Introducing the *multi-index* $\boldsymbol{i}(k) = (i_1, \ldots, i_{k-1}, i_{k+1}, \ldots, i_m)$, which iterates over every 1-dimensional fiber of X along the k-th dimension, the regularizer (4.11) can be written more compactly as

$$\mathrm{Tv}^m_{\boldsymbol{p}}(\mathsf{X}) = \sum\nolimits_{k=1}^{m} \sum\nolimits_{\boldsymbol{i}(k)} \|\boldsymbol{D}_{n_k} \boldsymbol{x}_{\boldsymbol{i}(k)}\|_{p_k}, \quad (4.12)$$

where $\boldsymbol{x}_{\boldsymbol{i}(k)}$ denotes a row of X along the k-th dimension, and $\boldsymbol{D}_{n_k}$ is a differencing matrix of appropriate size for the 1D-fibers along dimension k (of size n_k). The corresponding m-dimensional-TV proximity problem is

$$\min_{\mathsf{X}} \ \tfrac{1}{2}\|\mathsf{X}-\mathsf{Y}\|_{\mathrm{F}}^2 + \lambda \operatorname{Tv}_{\boldsymbol{p}}^m(\mathsf{X}), \tag{4.13}$$

where $\lambda > 0$ is a penalty parameter, and the Frobenius norm for a tensor just denotes the ordinary sum-of-squares norm over the vectorization of such tensor.

Problem (4.13) looks very challenging, but it enjoys decomposability as suggested by (4.12) and made more explicit by writing it as a sum of $\operatorname{Tv}^{1\mathrm{D}}$ terms

$$\min_{\mathsf{X}} \ \tfrac{1}{2}\|\mathsf{X}-\mathsf{Y}\|_{\mathrm{F}}^2 + \sum\nolimits_{k=1}^{m} \sum\nolimits_{\boldsymbol{i}(k)} \operatorname{Tv}_{p_k}^{1\mathrm{D}}\big(\boldsymbol{x}_{\boldsymbol{i}(k)}\big). \tag{4.14}$$

The proximity task (4.14) can be regarded as the sum of m proximity terms, each of which further decomposes into a number of inner $\operatorname{Tv}^{1\mathrm{D}}$ terms. These inner terms are trivial to address since, as in the 2D-TV case, each of the $\operatorname{Tv}^{1\mathrm{D}}$ terms operates on different entries of X. Regarding the m major terms, we can handle them by applying any of the combiner strategies presented above for $m > 2$, which ultimately yield the prox operator for $\operatorname{Tv}_{\boldsymbol{p}}^m$ by just repeatedly calling $\operatorname{Tv}^{1\mathrm{D}}$ prox operators. Most importantly, both proximal stacking and the natural decomposition of the problem provide a vast potential for parallel multithreaded computing, which is valuable when dealing with such complex and high-dimensional data.

5. Experiments and Applications

We will now demostrate the effectiveness of the various solvers covered in a wide array of experiments, as well as showing many of their practical applications. We will start by focusing on the $\operatorname{Tv}_1^{1\mathrm{D}}$ methods, moving then to other 1D-TV variants, and then to multidimensional TV.

All the solvers implemented for this paper were coded in C++ for efficiency. Our publicy available library **proxTV** includes all these implementations, plus bindings for easy usage in Matlab or Python: https://github.com/albarji/proxTV. Matrix operations have been implented by exploiting the LAPACK (Fortran) library (Anderson et al., 1999).

5.1. $\operatorname{Tv}_1^{1\mathrm{D}}$ Experiments and Applications

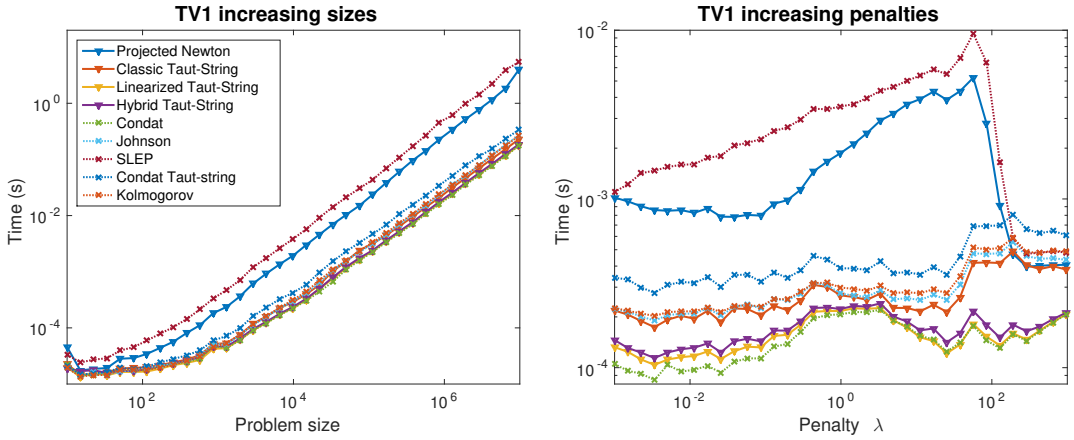
Since the most important components of the presented modular framework are the efficient $\operatorname{Tv}_1^{1\mathrm{D}}$ prox operators, let us begin by highlighting their empirical performance. We will do so both on synthetic and natural images data.

5.1.1. Running Time Results for Synthetic Data

We test the solvers under two scenarios of synthetic signals:

I) Increasing input size ranging from $n = 10^1$ to $n = 10^7$. A penalty $\lambda \in [0, 50]$ is chosen at random for each run, and the data vector $\boldsymbol{y}$ with uniformly random entries $y_i \in [-2\lambda, 2\lambda]$ (proportionally scaled to λ).

II) Varying penalty parameter λ ranging from 10^{-3} (negligible regularization) to 10^3 (the TV term dominates); here n is set to 1000 and y_i is randomly generated in the range $[-2, 2]$ (uniformly).



(a) (b)

Figure 7: Running times (in secs) for proposed and state of the art solvers for $\mathrm{Tv}_1^{1\mathrm{D}}$-proximity with increasing a) input sizes, b) penalties. Both axes are on a log-scale.

We benchmark the performance of the following methods, including both our proposals and state of the art methods found in the literature:

- Our proposed Projected Newton method (Appendix E).
- Our efficient implementation of the classic taut string method.
- Another implementation of the classic taut string method by Condat (2012).
- An implementation of the linearized taut string method.
- Our proposed hybrid taut string approach.
- The **FLSA** function (C implementation) of the SLEP library of Liu et al. (2009) for $\mathrm{Tv}_1^{1\mathrm{D}}$-proximity (Liu et al., 2010).
- The state-of-the-art method of Condat (2012), which we have seen to be equivalent to a linearized taut-string method.
- The dynamic programming method of Johnson (2013), which guarantees linear running time.
- The message passing method of Kolmogorov et al (2015), which allows generalization for computing a Total Variation regularizer on a tree.

Another implementation of the classic taut string method, found in the literature, has been added to the benchmark to test whether the implementation we have proposed is on par with the state of the art. We would like to note the surprising lack of widely available implementations of this method: the only working and efficient code we could find was part of the same paper where Condat's method was proposed.

For Projected Newton and SLEP a duality gap of 10^{-5} is used as the stopping criterion. For the hybrid taut-string method the switch parameter is set as $S = 1.05$. The rest of algorithms do not have parameters.

Timing results are presented in Figure 7 for both experimental scenarios. The following interesting facts are drawn from these results

- Direct methods (Taut string methods, Condat, Johnson, Kolmogorov) prove to be much faster than iterative methods (Projected Newton, SLEP).

- Although Condat's (and hence linearized taut string) method, has a theoretical worst-case performance of $O(n^2)$, the practical performance seems to follow an $O(n)$ behavior, at least for these synthetic signals.

- Even if Johnson and Kolmogorov methods have a guaranteed running time of $O(n)$, they turn out to be slower than the linearized taut string and Condat's methods. This is in line with our previous observations of the cache-friendly properties of in-memory methods; in contrast Johnson's method requires an extra $\sim 8n$ memory storage. Kolmogorov's method has less memory requirements but nevetheless shows similar behavior.

- The same performance observation applies to the classic taut string method. It is also noticeable that our implementation of this method turns out to be faster than previously available implementations (Condat's Taut-string), even becoming slightly faster than the state of the art Johnson and Kolmogorov methods. This result is surprising, and shows that the full potential of the classic taut-string method has been largely unexploited by the research community, or at least that proper efficient implementations of this method have not been made readily available so far.

5.1.2. Worst Case Scenario

The point about comparing $O(n)$ and $O(n^2)$ algorithms deserves more attention. As an illustrative experiment we have generated a signal following the worst case description in Condat (2012), and tested again the methods above on it, for increasing signal lengths. Figure 8 plots the results. Condat's method and consequently the linearized taut string method shows much worse performance than the rest of the direct methods. It is also remarkable how the hybrid method manages to avoid quadratic runtimes in this case.

5.1.3. Running Times on Natural Images

In the light of the previous results the following question arises: in practical settings, are the problems to be solved closer to the worst or the average runtime scenario? This fact will determine whether the guaranteed linear time or the more risky quadratic methods are more apt for practical use. To test this we devise the following experiment: we take a large benchmark of natural images and run each solver over all the rows and columns of all the images in the set, counting total running times, for different regularization values. The benchmark is made from images obtained from the data sets detailed in Table 2. We run this benchmark for the methods showing better performance in the experiments above: our implementation of the classic taut-string method, Condat's method ($\equiv$ linearized taut-string method), our proposed Hybrid taut-string method, Johnson's method and Kolmogorov et al's method.

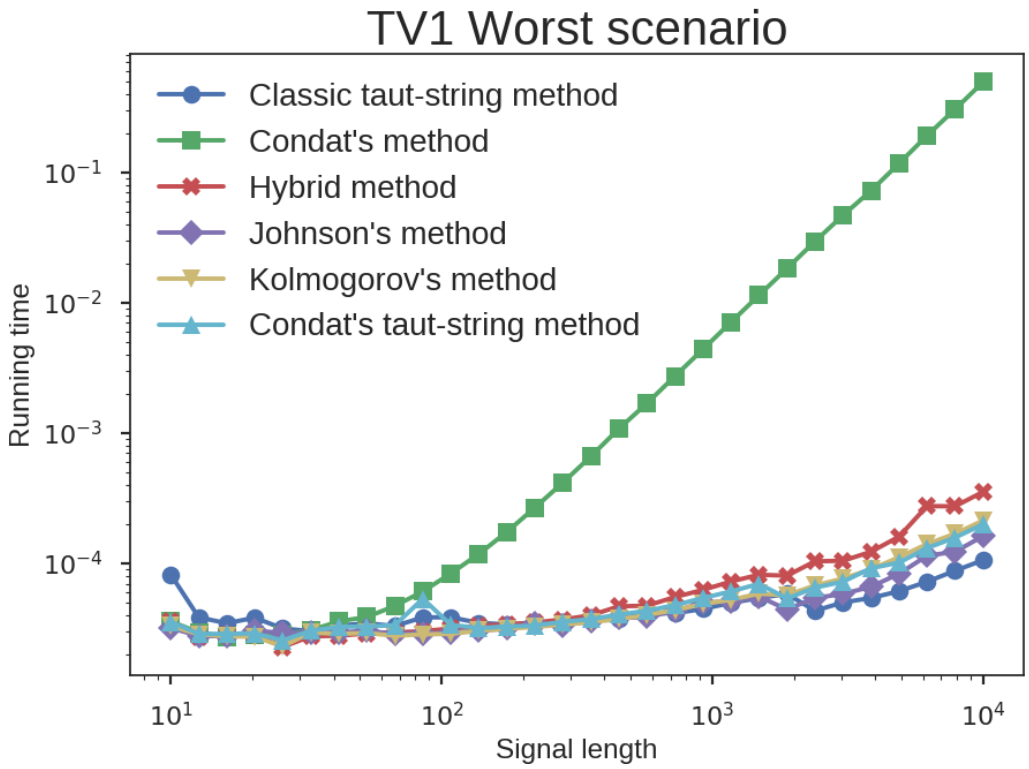


Figure 8: Running times (in secs) for proposed and state of the art solvers for $\mathrm{Tv}_1^{1\mathrm{D}}$-proximity in the worst-case scenario for Condat's method, for increasing input sizes. Both axes are on a log-scale.

Data set	Images	Average image size
INRIA holidays (Jegou et al, 2008)	812	$1817 \times 2233 \times 3$ px
LSVRC 2010 val set (Russakovsky et al, 2015)	50000	$391 \times 450 \times 3$ px

Table 2: Detail of image data sets used for large-scale $\mathrm{Tv}_1^{1\mathrm{D}}$ experiments.

Figure 9 shows runtime results for different penalty values over the whole INRIA holidays data set (Jegou et al, 2008), while Figure 10 shows similar results for the whole Large Scale Visual Recognition Challenge 2010 validation data set (Russakovsky et al, 2015). The following facts of interest can be observed:

- Condat's method (linearized taut-string) shows top performance for low penalty values, but bad scaling when moving to higher penalties. This can be explained using the geometric intuition developed above: for large penalty values the width of the tube is very large, and thus the taut-string will be composed of very long segments.

This is troublesome for a linearized taut-string method, as each backtrack will require recomputing a large number of steps. On the contrary for smaller penalties the tube will be narrow, and the taut-string composed of many small segments, thus resulting in very cheap backtracking costs.

- The performance of Classic taut-string, Johnson and Kolmogorov becomes slightly worse for large penalties, but suffers significantly less than the linearized taut-string. Surprisingly, the best performing approach tends to be the classic taut-string method.

- The proposed hybrid strategy closely follows the performance of Condat's method for the low penalty regime, while adapting to a behaviour akin to Kolmogorov for large penalties, thus resulting in very good performances over the whole regularization spectrum.

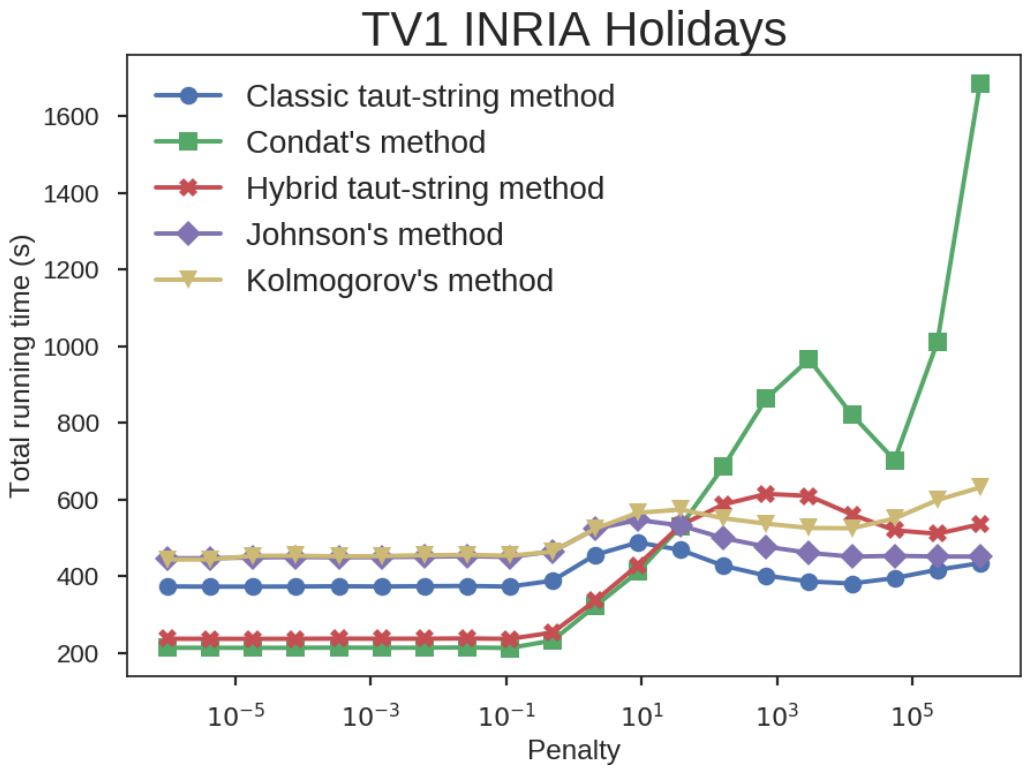


Figure 9: Running times (in secs) for the top performing proposed and state of the art solvers for $\mathrm{Tv}_1^{1\mathrm{D}}$-proximity over the whole INRIA Holidays data set, for increasing penalties.

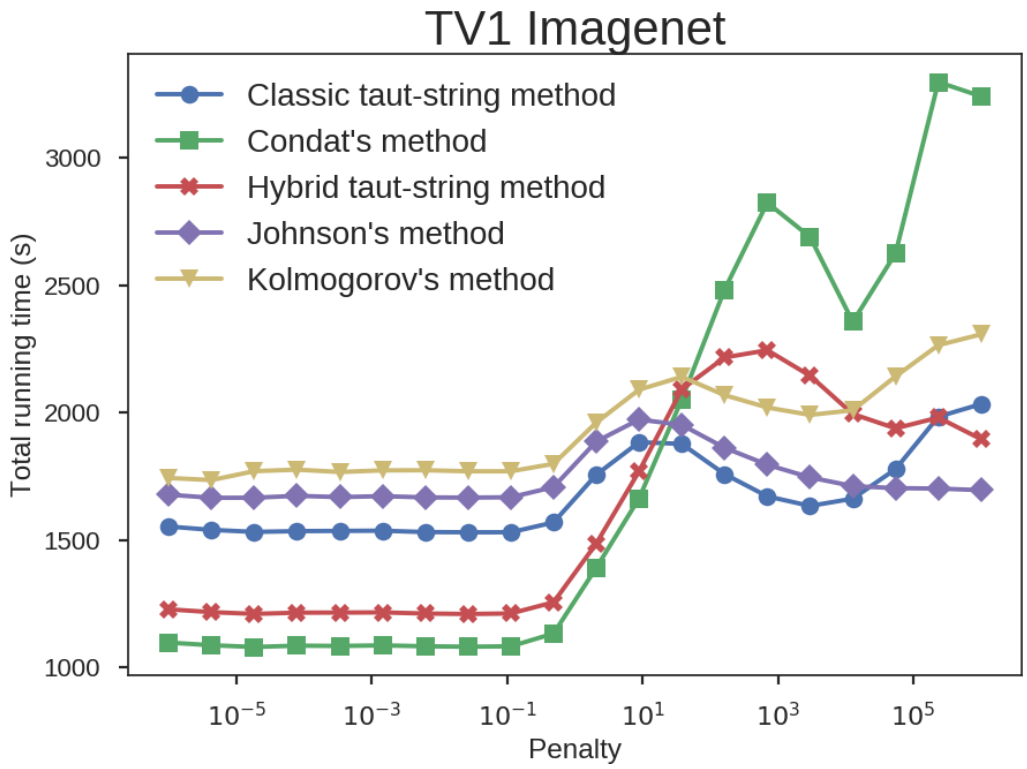


Figure 10: Running times (in secs) for the top performing proposed and state of the art solvers for $\mathrm{Tv}_1^{1\mathrm{D}}$-proximity over the whole Large Scale Visual Recognition Challenge 2010 validation data set, for increasing penalties.

5.1.4. Running Time Results for Weighted TV-L1

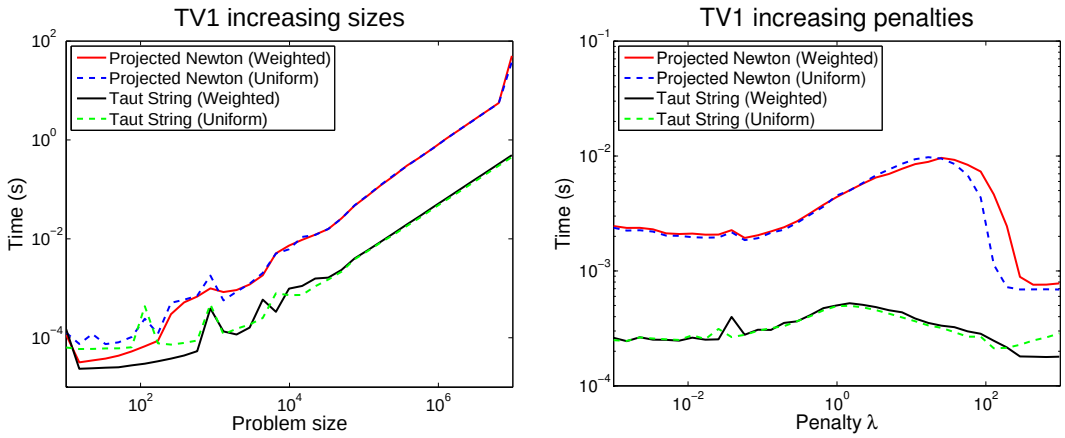
An advantage of the solvers proposed in this paper is their flexibility to easily deal with the more difficult, weighted version of the TV-L1 proximity problem. To illustrate this, Figure 11 shows the running times of the Projected Newton and (linearized) Taut String methods when solving both the standard and weighted TV-L1 prox operators.

Since for this set of experiments a whole vector of weights $\boldsymbol{w}$ is needed, we have adjusted the experimental scenarios as follows:

I) n is generated as in the general setting, penalties $\boldsymbol{w} \in [0, 100]$ are chosen at random for each run, and the data vector $\boldsymbol{y}$ with uniformly random entries $y_i \in [-2\lambda, 2\lambda]$, with λ the mean of $\boldsymbol{w}$, using also this λ choice for the uniform (unweighted) case.

II) λ and n are generated as in the general setting, and the weights vector $\boldsymbol{w}$ is drawn randomly from the uniform distribution $\boldsymbol{w}_i \in [0.5\lambda, 1.5\lambda]$.

As can be readily observed, performance for both versions of the problem is almost identical, even if the weighted problem is conceptually harder. Conversely, adapting the



(a) (b)

Figure 11: Running times (in secs) for Projected Newton and Taut String solvers for weighted and uniform $\mathrm{Tv}_1^{1\mathrm{D}}$-proximity with increasing a) input sizes, b) penalties. Both axes are on a log-scale.

other reviewed algorithms to address this problem while keeping up with performance is not a straightforward task.

We would also like to point out that in the paper Kumar et al (2015) a practical application of this method for energy minimization in computer vision is presented, where exactly the code behind this paper has been put to use.

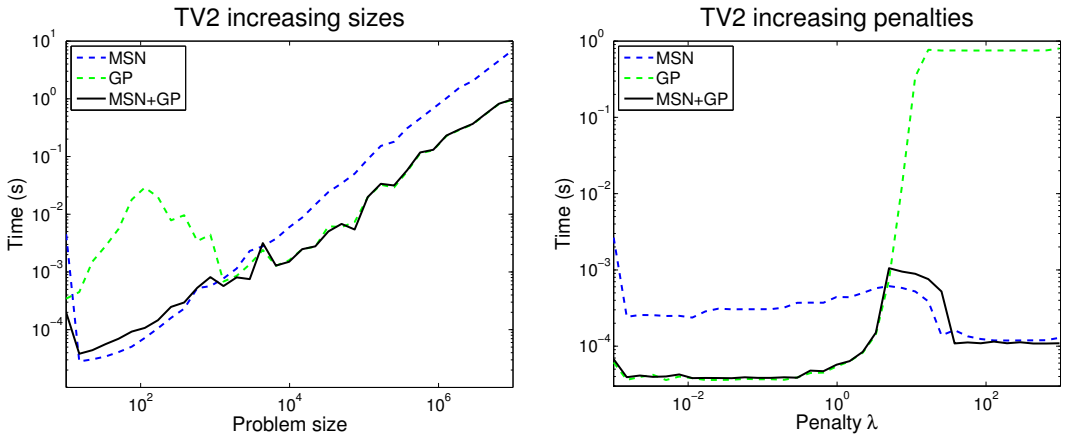
5.2. Experiments for other 1D-TV Variants

In this section we present experiments for other choices of the ℓ_p norm in $\mathrm{Tv}_p^{1\mathrm{D}}$, namely $p = 2$, $p = \infty$ and any general $p \geq 1$.

5.2.1. Running Time Results for TV-L2

Next we show results for $\mathrm{Tv}_2^{1\mathrm{D}}$ proximity. To our knowledge, this version of TV has not been explicitly treated before, so there do not exist highly-tuned solvers for it. Thus, we show running time results only for the MSN and GP methods. We use a duality gap of 10^{-5} as the stopping criterion; we also add an extra boundary check for MSN with tolerance 10^{-6} to avoid early stopping due to potentially infeasible intermediate iterates. Figure 12 shows results for the two experimental scenarios under test.

The results indicate that the performance of MSN and GP differs noticeably in the two experimental scenarios. While the results for the first scenario (Figure 12(a)) might suggest that GP converges faster than MSN for large inputs, it actually does so depending on the size of λ relative to $\|\boldsymbol{y}\|_2$. Indeed, the second scenario (Figure 12(b)) shows that although for small values of λ, GP runs faster than MSN, as λ increases, GP's performance worsens dramatically, so much that for moderately large λ, it is unable to find an acceptable solution even after 10,000 iterations (an upper limit imposed in our implementation). Conversely,



(a) (b)

Figure 12: Running times (in secs) for MSN, GP and a hybrid MSN+GP approach for $\mathrm{Tv}_2^{1\mathrm{D}}$-proximity with increasing a) input sizes, b) penalties. Both axes are on a log-scale.

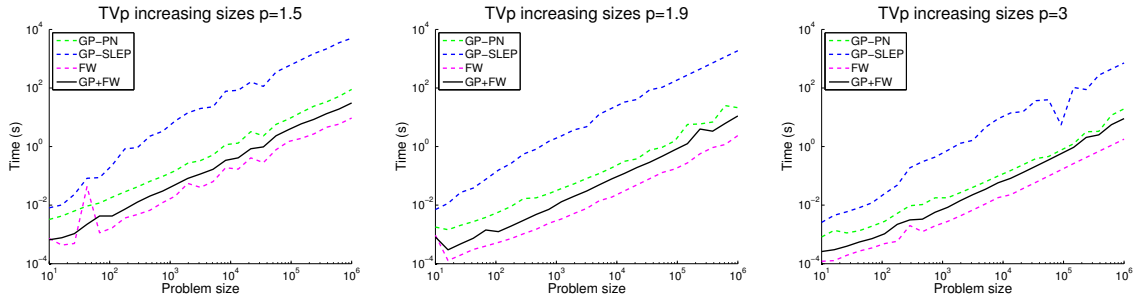
MSN finds a solution satisfying the stopping criterion under every situation, thus showing a more robust behavior.

These results suggest that it is preferable to employ a hybrid approach that combines the strengths of MSN and GP. Such a hybrid approach is guided using the following (empirically determined) rule of thumb: if $\lambda < \|\boldsymbol{y}\|_2$ use GP, otherwise use MSN. Further, as a safeguard, if GP is invoked but fails to find a solution within 50 iterations, the hybrid should switch to MSN. This combination guarantees rapid convergence in practice. Results for this hybrid approach are also included in the plots in Figure 12, and show how it successfully mimics the behavior of the better algorithm amongst MSN and GP.

5.2.2. Running Time Results for TV-Lp

Now we show results for $\mathrm{Tv}_p^{1\mathrm{D}}$ proximity. Again, to our knowledge efficient solvers for this version of TV are not available; still proposals for solving the ℓ_q-ball projection problem do exist. For these experiments we decided to use a method based on a zero finding approach readily available as the *epp* function in SLEP library (Liu et al., 2009). Consequently, we present here a comparison between this reference projection subroutine and our PN–based projection when embedded in our proposed Gradient Projection solver of §3.2. The alternative proposal given by the Frank–Wolfe algorithm of §3.2.2 is also present in the comparison. We use a duality gap of 10^{-5} as stopping criterion both for GP and FW. Figure 13 shows results for the two experimental scenarios under test, for p values of 1.5, 1.9 and 3.

A number of interesting conclusions can be drawn from the results. First, our Projected Newton ℓ_q-ball subroutine is far more efficient than *epp* when in the context of the GP solver. Two factors seem to be the cause of this: in the first place our Projected Newton



(a) (b) (c)

Figure 13: Running times (in secs) for GP with PN projection, GP with SLEP's *epp* projection, FW and a hybrid GP+FW algorithm, for $\mathrm{Tv}_p^{\mathrm{1D}}$-proximity with increasing input sizes and three different choices of p. Both axes are on a log-scale.

approach proves to be faster than the zero finding method used by *epp*. Secondly, in order for the GP solver to find a solution within the desired duality gap, the projection subroutine must provide very accurate results (about 10^{-12} in terms of duality gap). Given its Newton nature, our ℓ_q-ball subroutine scales better in term of running times as a factor of the desired accuracy, which explains he observed differences in performance.

It is also of relevance noting that Frank–Wolfe is significantly faster than Projected Newton. This should discourage the use of Projected Newton, but we find it to be extremely useful in the range of λ penalties where λ is large, but not enough to render the problem trivial ($\boldsymbol{w} = 0$ solution). In this range the two variants of PN and also FW are unable to find a solution within the desired duality gap (10^{-5}), getting stuck at suboptimal solutions. We solve this issue by means of a hybrid GP+FW algorithm, in which updates from both methods are interleaved at a ratio of 10 FW updates per 1 GP update, as FW updates are faster. As both algorithms guarantee improvement in each iteration but follow different procedures for doing so, they complement each other nicely, resulting in a superior method attaining the objective duality gap and performing faster than GP.

5.2.3. Running Time Results for TV-L∞

For completeness we also include results for our $\mathrm{Tv}_\infty^{\mathrm{1D}}$ solver based on GP + a standard ℓ_1-projection subroutine. Figure 15 presents running times for the two experimental scenarios under test. Since ℓ_1-projection is an easier problem than the general ℓ_q-projection the resultant algorithm converges faster to the solution than the general GP $\mathrm{Tv}_p^{\mathrm{1D}}$ prox solver, as expected.

5.2.4. Application: Proximal Optimization for Fused-Lasso

We now present a key application that benefits from our TV prox operators: **Fused-Lasso** (FL) (Tibshirani et al., 2005), a model that takes the form

$$\min_{\boldsymbol{x}} \quad \tfrac{1}{2}\|\boldsymbol{A}\boldsymbol{x} - \boldsymbol{y}\|_2^2 + \lambda_1\|\boldsymbol{x}\|_1 + \lambda_2 \mathrm{Tv}_1^{\mathrm{1D}}(\boldsymbol{x}). \tag{5.1}$$

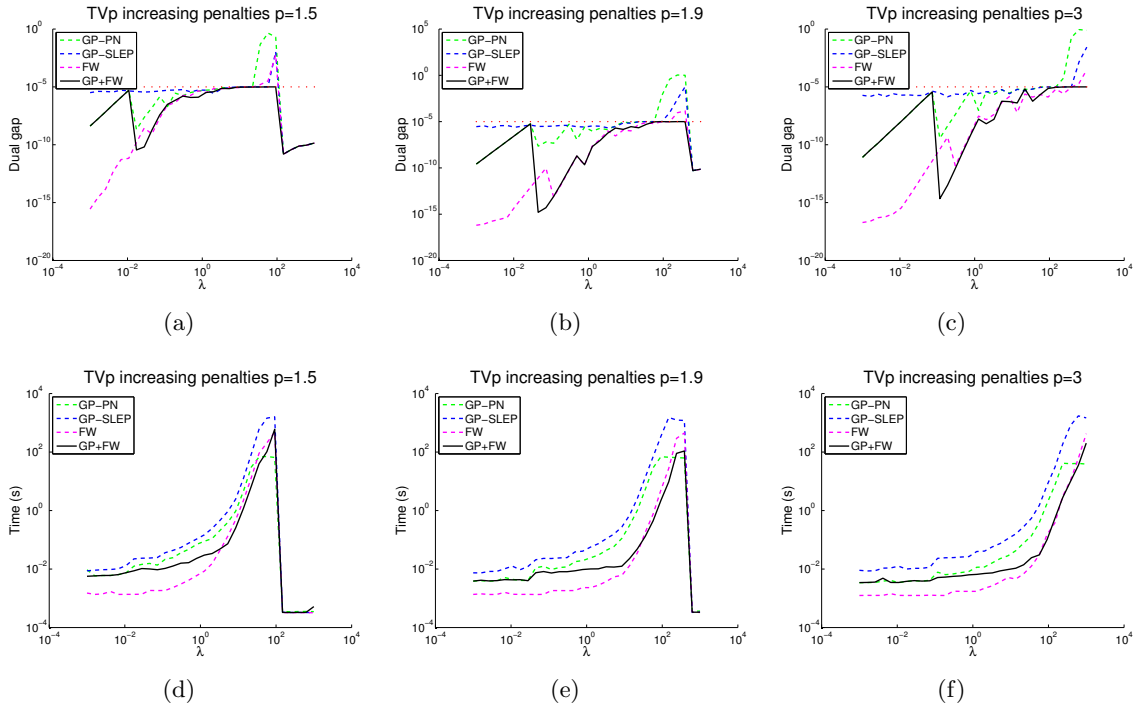
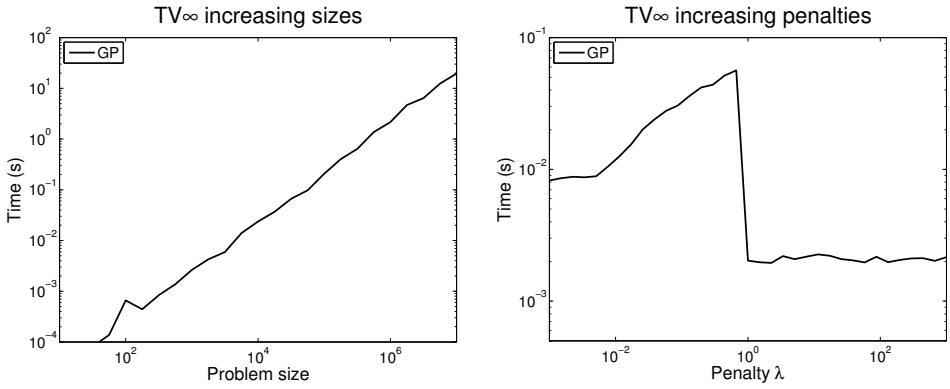


Figure 14: Attained duality gaps (a-c) and running times (d-f, in secs) for GP with PN projection, GP with SLEP's *epp* projection, FW and a hybrid GP+FW algorithm, for $\mathrm{Tv}_p^{1\mathrm{D}}$-proximity with increasing penalties and three different choices of p. Both axes are on a log-scale.

The ℓ_1-norm in (5.1) forces many x_i to be zero, while $\mathrm{Tv}_1^{1\mathrm{D}}$ favors nonzero components to appear in blocks of equal values $x_{i-1} = x_i = x_{i+1} = \ldots$. The FL model has been successfully applied in several bioinformatics applications (Tibshirani and Wang, 2008; Rapaport and Vert, 2008; Friedman et al., 2007), as it encodes prior knowledge about consecutive elements in microarrays becoming active at once.

Following the ideas presented in Sec. 4, since the FL model uses two regularizers, we can use Proximal Dykstra as the combiner to handle the prox operator. To illustrate the benefits of this framework in terms of reusability, we apply it to several variants of FL.

- **Fused-Lasso (FL):** Least-squares loss $+\ell_1 + \mathrm{Tv}_1^{1\mathrm{D}}$ as in (5.1)
- **ℓ_p-Variable Fusion (VF):** Least-squares loss $+\ell_1 + \mathrm{Tv}_p^{1\mathrm{D}}$. Though Variable Fusion was already studied by Land and Friedman (1997), their approach proposed an ℓ_p^p-like regularizer in the sense that $r(\boldsymbol{x}) = \sum_{i=1}^{n-1} |x_{i+1} - x_i|^p$ is used instead of the TV regularizer $\mathrm{Tv}_p^{1\mathrm{D}}(x) = \left(\sum_{i=1}^{n-1} |x_{i+1} - x_i|^p\right)^{1/p}$. Using Tv_p leads to a more conservative penalty that does not oversmooth the estimates. This FL variant seems to be new.



(a) (b)

Figure 15: Running times (in secs) for GP for $\mathrm{Tv}_{\infty}^{1\mathrm{D}}$-proximity with increasing a) input sizes, b) penalties. Both axes are on a log-scale.

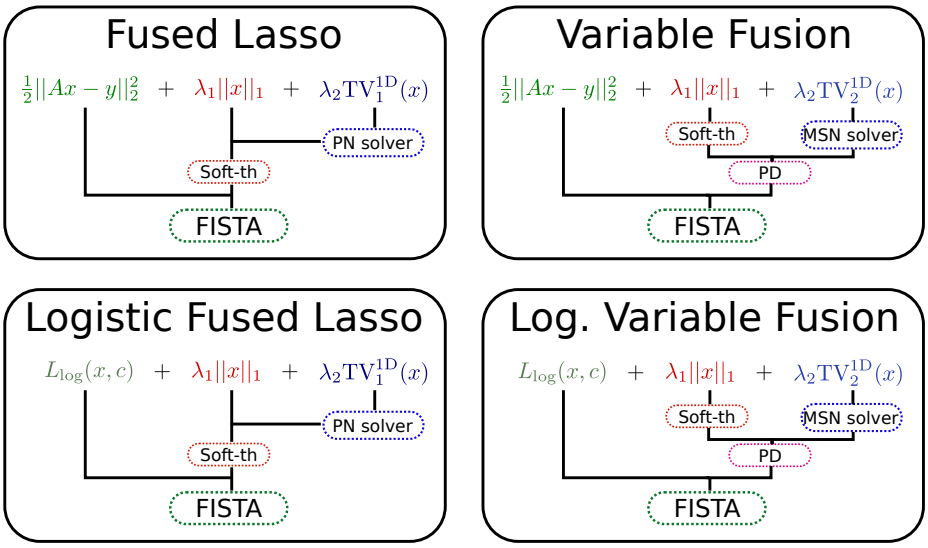


Figure 16: Fused-Lasso models addressed by proximal splitting.

- **Logistic-fused lasso (LFL):** Logistic-loss $+\ell_1 + \mathrm{Tv}_1^{1\mathrm{D}}$, where the loss takes the form $\ell(\boldsymbol{x}, c) = \sum_i \log\left(1 + e^{-y_i(\boldsymbol{a}_i^T\boldsymbol{x}+c)}\right)$, and can be used in a FL formulation to obtain models more appropriate for classification on a data set $\{(\boldsymbol{a}_i, y_i)\}$ (Kolar et al., 2010).
- **Logistic + ℓ_p-fusion (LVF):** Logistic loss $+\ell_1 + \mathrm{Tv}_p^{1\mathrm{D}}$.

To solve these variants of FL, all that remains is to compute the gradients of the loss functions, but this task is trivial. Each of these four models can be then solved easily by invoking any proximal splitting method by appropriately plugging in gradient and prox operators. Incidentally, the **SLEP** library (Liu et al., 2010) includes an implementation

of FISTA (Beck and Teboulle, 2009) carefully tuned for Fused Lasso, which we base our experiments on. Figure 16 shows a schematic of the algorithmic modules for solving each FL model.

Remark: A further algorithmic improvement can be obtained by realizing that for $r(\boldsymbol{x}) = \lambda_1\|\boldsymbol{x}\|_1 + \lambda_2 \mathrm{Tv}_1^{1\mathrm{D}}(\boldsymbol{x})$ the prox operator $\mathrm{prox}_r \equiv \mathrm{prox}_{\lambda_1\|\cdot\|_1} \circ \mathrm{prox}_{\lambda_2 \mathrm{Tv}_1^{1\mathrm{D}}(\cdot)}$. Such a decomposition does not usually hold, but it can be shown to hold for this particular case (Yu, 2013; Rinaldo, 2009; Tibshirani et al., 2005). Therefore, for FL and LFL we can compute the proximal operator for the combined regularizer r directly, thus removing the need for a combiner algorithm. This is also shown in Figure 16.

5.2.5. Fused-Lasso Experiments: Simulation

The standard FL model has been well-studied in the literature, so a number of practical algorithms addressing it have already been proposed. The aforementioned Fused-Lasso algorithm in the **SLEP** library can be regarded as the state of the art, making extensive use of an efficient proximity subroutine (FLSA). Our experiments on $\mathrm{Tv}_1^{1\mathrm{D}}$-proximity (§5.1) have already shown superiority of our prox solvers over FLSA; what remains to be checked is whether this benefit has a significant impact on the overall FL solver. To do so, we compare running times with synthetic data.

We generate random matrices $\boldsymbol{A} \in \mathbb{R}^{n\times m}$ with i.i.d. entries drawn from a zero mean, unit variance gaussian. We set the penalties to $\lambda_1 = \lambda_2 = 10$. We select the vector of responses $\boldsymbol{y}$ using the formula $\boldsymbol{y} = \mathrm{sgn}(\boldsymbol{A}\boldsymbol{x}_t + \boldsymbol{v})$, where $\boldsymbol{x}_t$, and $\boldsymbol{v}$ are random vectors whose entries have variances 1 and 0.01, respectively. The numerical results are summarized in Figure 17, which compares out of the box SLEP (version 4.0) (Liu et al., 2009) against the very same algorithm employing our fast taut–string $\mathrm{Tv}_1^{1\mathrm{D}}$ solver instead of the default FLSA subroutine of SLEP. Comparison is done by showing the relative distance to the problem's optimum versus time. The optimal values in each setting were estimated by running both algorithms for a very large number of iterations.

The plots show a clear trend: when the input matrices feature a very large column dimension the use of our taut-string $\mathrm{Tv}_1^{1\mathrm{D}}$ solver turns into speedups in optimization times, which however become negligible for matrices with a more balanced rows/columns ratio. This result is reasonable, as the vector x under optimization has size equal to the number of columns of the data matrix A. If A has a large number of columns the cost of solving $\mathrm{Tv}_1^{1\mathrm{D}}$ is significant, and thus any improvement in this step has a noticeable impact on the overall algorithm. Conversely, when the number of rows in A is large the cost of computing the gradient of the loss function ($\nabla \frac{1}{2}\|\boldsymbol{A}\boldsymbol{x} - \boldsymbol{y}\|_2^2 = \boldsymbol{A}^T(\boldsymbol{A}\boldsymbol{x} - \boldsymbol{y})$) dominates, getting limited benefits from such improvements in prox computations. Therefore, it is for data with a very large number of features where our proposed method can provide a useful speedup.

5.2.6. Fused-Lasso Experiments: Microarray Classification

Now we report results of applying the four FL models on a series of problems from bioinformatics. We test the FL models on binary classification tasks for the following real microarray data sets: ArrayCGH (Stransky et al., 2006), Leukemias (Golub et al., 1999), Colon (U. Alon et al., 1999), Ovarian (Rogers et al., 2005) and Rat (Hua et al., 2009). Each data set was split into three equal parts (ensuring similar proportion of classes in every split) for

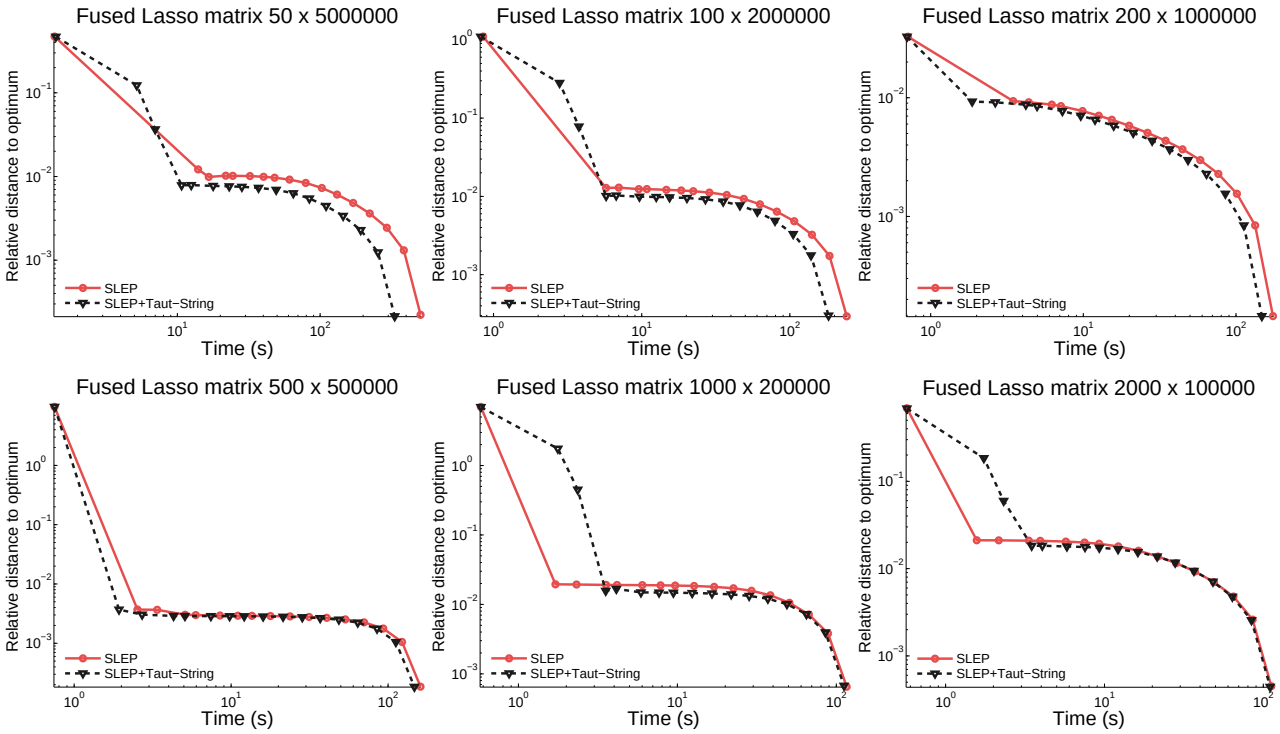


Figure 17: Relative distance to optimum vs time of the Fused Lasso optimizers under comparison, for the different layouts of synthetic matrices.

Data set	FL	VF-ℓ_2	LFL	LVF-ℓ_2
ArrayCGH	73.6%	73.6%	**84.2%**	73.6%
Leukemias	**92.0%**	88.0%	**92.0%**	88.0%
Colon	**77.2%**	**77.2%**	**77.2%**	**77.2%**
Ovarian	**88.8%**	83.3%	83.3%	83.3%
Rat	68.8%	65.5%	**72.1%**	**72.1%**

Table 3: Classification accuracies for the presented Fused–Lasso models on microarray data. For the Variable Fusion models an ℓ_2 version of TV was employed.

training, validation and test. The penalty parameters were found by exhaustive grid search in the range $\lambda_1, \lambda_2 \in [10^{-4}, 10^2]$ to maximize classification accuracy on the validation splits.

Table 3 shows test accuracies. In general, as expected the logistic-loss based FL models yield better classification accuracies than those based on least-squares, as such loss function tends to be more appropriate for classification problems. However the Ovarian data set proves to be an exception, showing better performance under a squared loss. Regarding

the TV-regularizer, the classic $\text{Tv}_1^{1\text{D}}$-penalty seems to perform better in general, with the $\text{Tv}_2^{1\text{D}}$-penalty showing competitive results in some settings.

5.3. 2D-TV: Experiments and Applications

We address now several practical applications that benefit from two-dimensional TV regularization; our results show again how the presented $\text{Tv}_{p,q}^{2\text{D}}$ prox operators fits in seamlessly into our modular framework to produce efficient proximal splitting solvers.

5.3.1. Image Denoising through Anisotropic Filtering

Our first example is related to the classic problem of image denoising, but with the twist that we deal with noise of an anisotropic character. More specifically, suppose that the true image $\mu \in \mathbb{R}^{n\times m}$ is contaminated by additive noise $\boldsymbol{N}$, so that only $\mu_0 = \mu + \boldsymbol{N}$ is observed. The denoising problem estimates μ given just the noisy version μ_0. This problem is highly ill-posed and as such not approachable unless additional assumptions on the noise (or on the underlying image) are made.

Isotropic and anisotropic models: an extremely common choice is to simply assume the noise to be gaussian, or some other zero-mean distribution. Under these conditions, a classic method to perform such denoising task is the **Rudin-Osher-Fatemi** (ROF) model (Rudin et al., 1992), which finds an approximation $\boldsymbol{X}$ to the original image by solving

$$\min_{\boldsymbol{X}} \quad \|\boldsymbol{X} - \mu_0\|_{\text{F}}^2 + \lambda \sum_{i=2}^{n}\sum_{j=2}^{m} \|\partial x_{i,j}\|_2, \tag{5.2}$$

where $\partial x_{i,j}$ is the *discrete gradient*

$$\partial x_{i,j} = \begin{bmatrix} x_{i,j} - x_{i-1,j} \\ x_{i,j} - x_{i,j-1} \end{bmatrix}.$$

That is, it is the vector of differences of $\boldsymbol{X}_{i,j}$ and its neighbors along both axes.

The objective of the first term in the ROF model is to penalize any deviation of $\boldsymbol{X}$ from the observed image μ_0, while the second term can be readily recognized as a mixed $(2,1)$-norm over the discrete gradient of $\boldsymbol{X}$. This regularizer models caters to some prior knowledge: in natural images sharp discontinuities in intensity between neighboring points only appear in borders of objects, while the rest of the pixels usually show smooth variations in intensity. It makes sense, therefore, to penalize large values of the gradient, as sharp changes have a higher probability of having being produced by noise. Conversely, as the mean of the noise is zero, it is also sensible to maintain the denoised image $\boldsymbol{X}$ close to the observed μ_0. Merging these two goals produces the ROF model (5.2).

A closer look at the ROF regularizer reveals that it follows the spirit of the reviewed 2D-TV regularizer which also penalizes sharp variations between neighboring pixels. Indeed, all such regularizers are broadly categorized as TV regularizers within the image processing community. It is clear, though, that the ROF regularizer (5.2) does not coincide with the $\text{Tv}_{p,q}^{2\text{D}}$ regularizer used in this paper. Some authors (Bioucas-Dias and Figueiredo, 2007) differentiate between these regularizers by naming the ROF approach as **isotropic TV** and the $\text{Tv}_{p,q}^{2\text{D}}$-style approach as **anisotropic TV**. This naming comes from the fact that

isotropic TV penalizes each component of the discrete gradient $\partial x_{i,j}$ following an ℓ_2 norm, whereas the anisotropic $\mathrm{Tv}^{2\mathrm{D}}_{p,q}$-norm and in particular $\mathrm{Tv}^{2\mathrm{D}}_{1,1}$-norm, penalize rows and columns independently.

While image filtering using isotropic TV is generally preferred for natural images denoising (Bioucas-Dias et al., 2006), in some settings anisotropic filtering can produce better results, and in fact has been favored by some authors in the past (Choksi et al., 2010; Li and Santosa, 1996). This is specially true on those images that present a "blocky" structure, and thus are better suited to the structure modeled by the $\mathrm{Tv}^{2\mathrm{D}}_{p,q}$-norm. Therefore, efficient methods to perform anisotropic filtering are also important.

Anisotropic denoising experiments: denoising using the anisotropic $\mathrm{Tv}^{2\mathrm{D}}_{p,q}$-norm reduces to solving

$$\min_{\boldsymbol{X}} \quad \|\boldsymbol{X} - \mu_0\|_{\mathrm{F}}^2 + \lambda \mathrm{Tv}^{2\mathrm{D}}_{p,q}(\boldsymbol{X}). \tag{5.3}$$

But (5.3) is nothing but the $\mathrm{Tv}^{2\mathrm{D}}_{p,q}$-proximity problem, and hence can be directly solved by applying the 2D-TV prox operators described above. We solve (5.3) below for the choice $p = q = 1$ (which is common in practice), for the following selection of algorithms:

- Proximal Dykstra (§ 4.1.1)
- The Douglas-Rachford variant based on alternating projections (§ 4.1.2)
- The Split Bregman method of Goldstein T. (2009), which follows an ADMM–like approach to split the ℓ_1 norm apart from the discrete gradient operator, thus not requiring the use of a 1D-TV prox operator.
- Chambolle-Pock's method applied to 2D TV (§ 4.1.4).
- Condat's general splitting method (§ 4.1.4).
- Kolmogorov et al primal-dual method (§ 4.1.4).
- Yang's method (ADMM) (§ 4.1.3)
- The maximum flow approach by Goldfarb and Yin (2009), which shows the relationship between the 2D-TV proximity minimization and the maximum flow problem over a grid, and thus applies an efficient maximum flow method to solve a discrete-valued version of 2D-TV.

In Proximal Dykstra, Douglas-Rachford and ADMM we use the linearized taut–string strategy presented before as solver for the base proximity operators. All algorithm parameters were set as recommended in their corresponding papers or public implementations, except for Proximal Dykstra and Douglas-Rachford, which are parameter free. For Chambolle-Pock we tried both the scheme with fixed algorithm parameters (Chambolle and Pock, 2011, Algorithm 1) and the scheme with acceleration (Chambolle and Pock, 2011, Algorithm 2); however the accelerated version did not converge to the desired solution within enough accuracy (relative difference of 10^{-5}), therefore only the results for the fixed version are reported. For Kolmogorov we follow the recommendations in Chambolle and Pock (2014), taking into account the Lipschitz constants of the optimized functions and selecting the parameter updating strategy that produced faster performance in the experiments: $\theta^{t+1} = \frac{1}{\sqrt{1+\tau^t}}, \tau^{t+1} = \theta^{t+1}\tau^t, \sigma^{t+1} = \frac{\sigma^t}{\theta^{t+1}}, \theta^0 = 1, \tau^0 = \frac{1}{2}, \sigma^0 = 1$.

Image	Gaussian	Speckle	Poisson	Salt & Pepper
randomQR	0.2	0.3	∅	∅
shape	0.05	∅	∅	∅
trollface	∅	1	∅	∅
diagram	∅	∅	✓	∅
text	∅	∅	∅	0.1
comic	0.05	∅	✓	∅
contour	∅	∅	✓	0.4
phantom	∅	2	✓	∅

Table 4: Types of noise and parameters for each test image. A ∅ indicates that such noise was not applied for the image. *Gaussian* and *Speckle* correspond to gaussian additive and multiplicative (respectively) noises with zero mean and the indicated variance. *Salt & Pepper* noise turns into black or white the indicated fraction of image pixels. *Poisson* regenerates each pixel by drawing a random value from a Poisson distribution with mean equal to the original pixel value, thus producing a more realistic noise.

The images used in the experiments are displayed in Appendix F as Figure 25. To test the filters under a variety of scenarios, different kinds of noise were introduced for each image. Table 4 gives details on this, while the noisy images are shown in Figure 26. All QR barcode images used the same kind and parameters of noise. Noise was introduced using Matlab's *imnoise* function.

Values for the regularization parameter λ were found by maximizing the quality of the reconstruction, measured using **Improved Signal-to-Noise Ratio** (ISNR) (Afonso et al., 2010). ISNR is defined as

$$\mathrm{ISNR}(\boldsymbol{X}, \mu, \mu_0) = 10 \log_{10} \frac{\|\mu_0 - \boldsymbol{X}\|_{\mathrm{F}}^2}{\|\boldsymbol{X} - \mu\|_{\mathrm{F}}^2},$$

where μ is the original image, μ_0 its noisy variant, and $\boldsymbol{X}$ the reconstruction.

To compare the algorithms we run all of them for each image and measured its ISNR and relative distance to the optimal objective value of the current solution at each iteration through their execution. The only exception to this procedure is the method of Goldfarb and Yin, which is non–iterative and thus always returns an exact solution, and so we just measure the time required to finish. The optimal objective value was estimated by running all methods for a very large number of iterations and taking the minimum value of them all. This produced the plots shown in Figures 18–19. From them the following observations are of relevance:

- Condat's method and Chambolle-Pock's method are reduced to essentially the same algorithm when applied to the particular case of anisotropic 2D TV denoising. Furthermore, they seem to perform slowly when compared to other methods.
- ADMM (Yang's method) exhibits slow performance at the beginning, but when run for sufficient time is able to achieve a good approximation to the optimum.

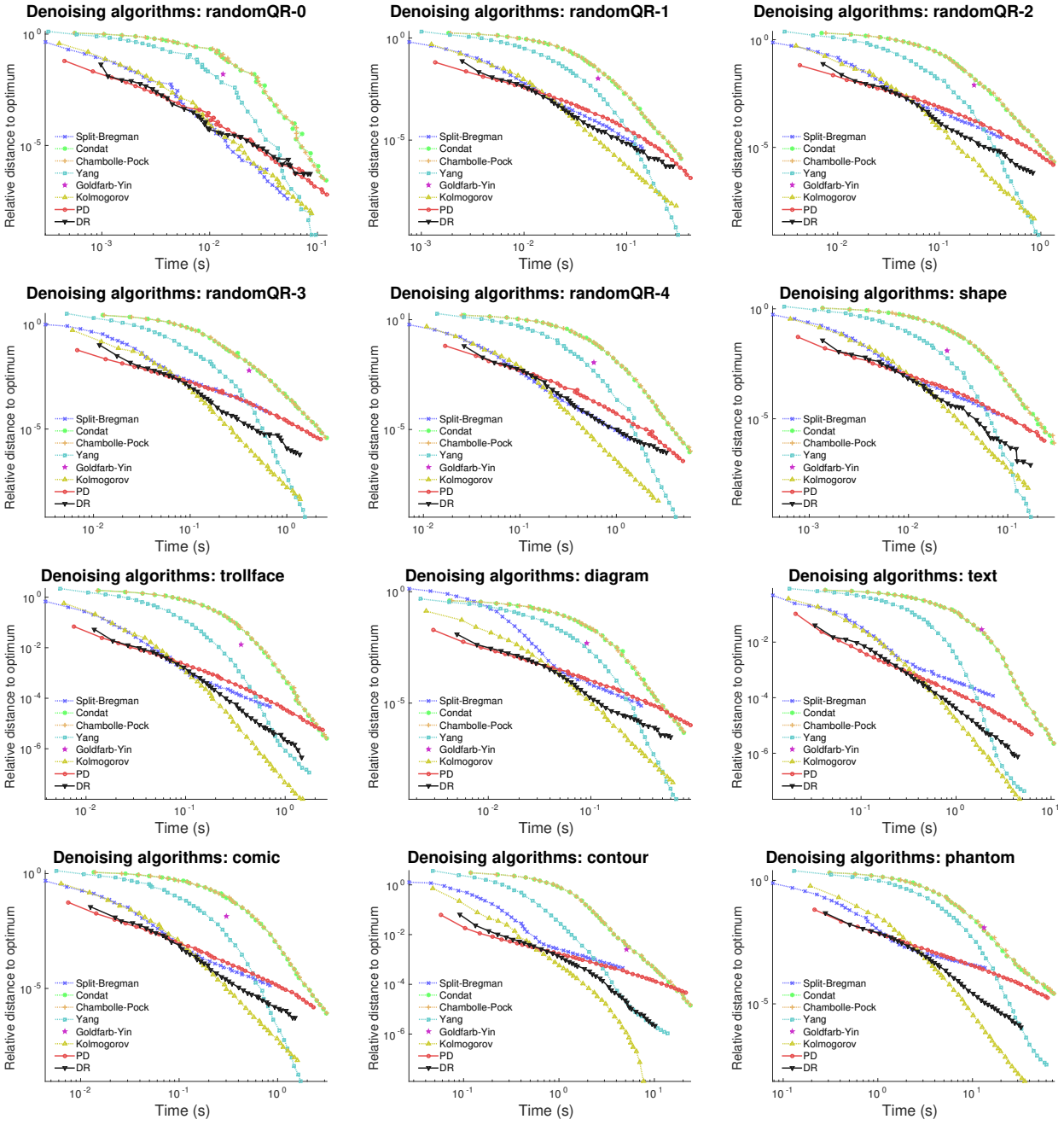


Figure 18: Relative distance to optimum vs time of the denoising 2D-TV algorithms under comparison, for the different images considered in the experiments.

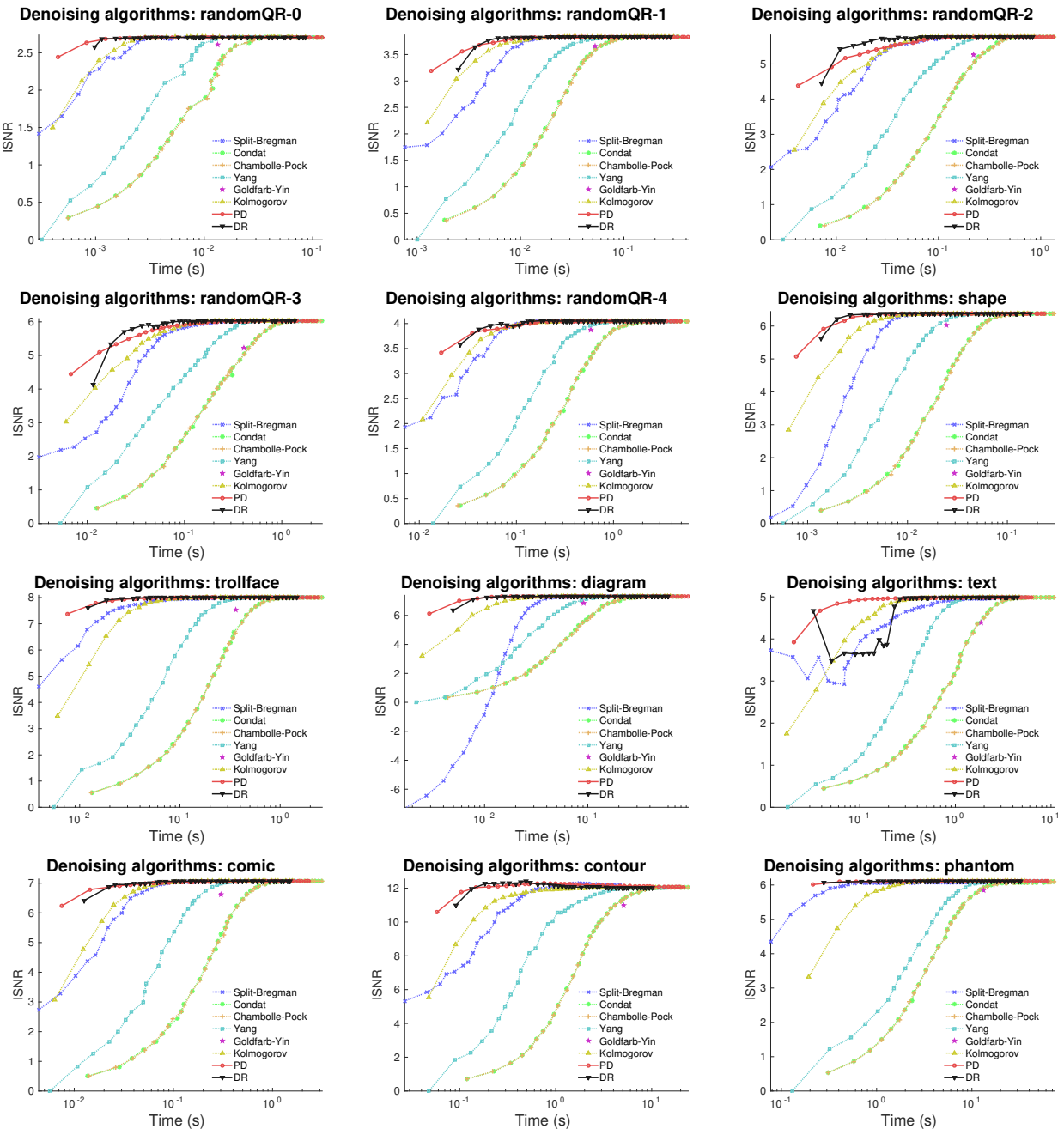


Figure 19: Increased Signal to Noise Ratio (ISNR) vs time of the denoising 2D-TV algorithms under comparison, for the different images considered in the experiments.

- The Split Bregman method, in spite of being an ADMM–like method much like Condat's or Chambolle-Pock, performs significantly better than those. We attribute this to the very efficient implementation provided by its authors, and to the fact that a fast approximate method is employed to compute the required matrix inversions throughout the method.

- The method by Goldfarb and Yin is slower than other approaches and seems to provide suboptimal solutions. We attribute this to the fact that this method solves a discrete (integer–rounded) approximation to the problem. We acknowledge that other methods exploiting the Total Variation - Minimum-cut relationship have been proposed with varying speed results, e.g. (Duan and Tai, 2012), however the suboptimality issues still apply.

- The method by Kolmogorov et al, when properly accelerated by a suitable choice of adaptive stepsizes, seems to be the best choice for finding very accurate solutions, though it is very closely followed by ADMM.

- The parameter free methods PD and DR are the fastest to achieve a mid-quality solution, with Douglas-Rachford performing better than Proximal Dykstra.

Considering these facts, the method of choice among the ones considered depends on the desired accuracy. We argue, however, that for the purpose of image processing a mid-quality solution is sufficient. The ISNR plots of Figure 19 certainly seem to support this, as the perceived quality of the reconstruction, roughly approximated by the ISNR, saturates rapidly and no significant improvements are obtained through further optimization. Given this, the proposed methods seem to be the best suited for the considered task.

For quick reference, Table 5 presents a summary of key points of the compared methods, along with some recommendations about when to put them to use.

5.3.2. Parallelization Experiments

In addition to the previous experiments and to illustrate the parallelization potential of the presented anisotropic filtering method, Figure 20 plots running times for the PD algorithm as the number of processor core ranges from 1 through 16. We see that for the smaller images, the gains due to more processors essentially flatten out by 8 cores, where synchronization and memory contention offsets potential computational gains (first row). For the larger images, there is steadier speedup as the number of cores increase (in each plot there seems to be a "bump" at 14 processors; we attribute this to a quirk of the multicore machine that we used). From all the plots, however, the message is clear: our TV prox operators exploit parallelization well, and show substantial speedups as more processor cores become available.

We should also note in passing that the Split Bregman method, which in the previous experiments showed a reasonable performance, turns out to be much harder to parallelize. This fact was already observed by Jie Wang et al. (2014) in the context of isotropic TV. Therefore when several processor cores are available the proposed modular strategy seems to be even more suitable to the task.

Method		Key points
Douglas Rachford	+	Fast convergence to medium-quality
	+	Embarrassingly parallel
	−	Slow for higher accuracies
	⇒	Ideal for standard denoising tasks
Proximal Dykstra	+	Attainable accuracies similar to DR
	−	But slower than DR
	⇒	Use DR instead
Split Bregman	+	Eventually performs similarly to DR
	−	Slow convergence at first iterations
	⇒	Use DR instead
Chambolle–Pock	−	Slow
	⇒	Use other method instead
Condat	+	Solves objectives involving a sum of smooth/non–smooth functions with linear operators
	−	Reduces to Chambolle–Pock when solving basic image denoising
	⇒	Use only when dealing with more complex functionals
ADMM (Yang)	+	More accurate
	−	Slightly slower than Kolmogorov
	−	Bad behavior for mid-quality solutions
	⇒	Use Kolmogorov instead
Kolmogorov	+	More accurate
	−	Slower than DR for low accuracies
	⇒	Useful when extremely accurate solutions are required
Goldfarb-Yin	+	Solves the discrete version of the problem
	−	Slow
	−	Poor accuracy for the continuous version
	⇒	Apply only when solving the discrete problem

Table 5: Summary of key points of the compared $\mathrm{Tv}^{2\mathrm{D}}_{1,1}$ proximity (denoising) methods.

5.3.3. Anisotropic Image Deconvolution

Taking a step forward we now confront the problem of **image deconvolution** (or image deblurring). This setting is more complex since the task of image recovery is made harder by the presence of a **convolution kernel** K that distorts the image as

$$\mu_0 = \boldsymbol{K} * \mu + \boldsymbol{N},$$

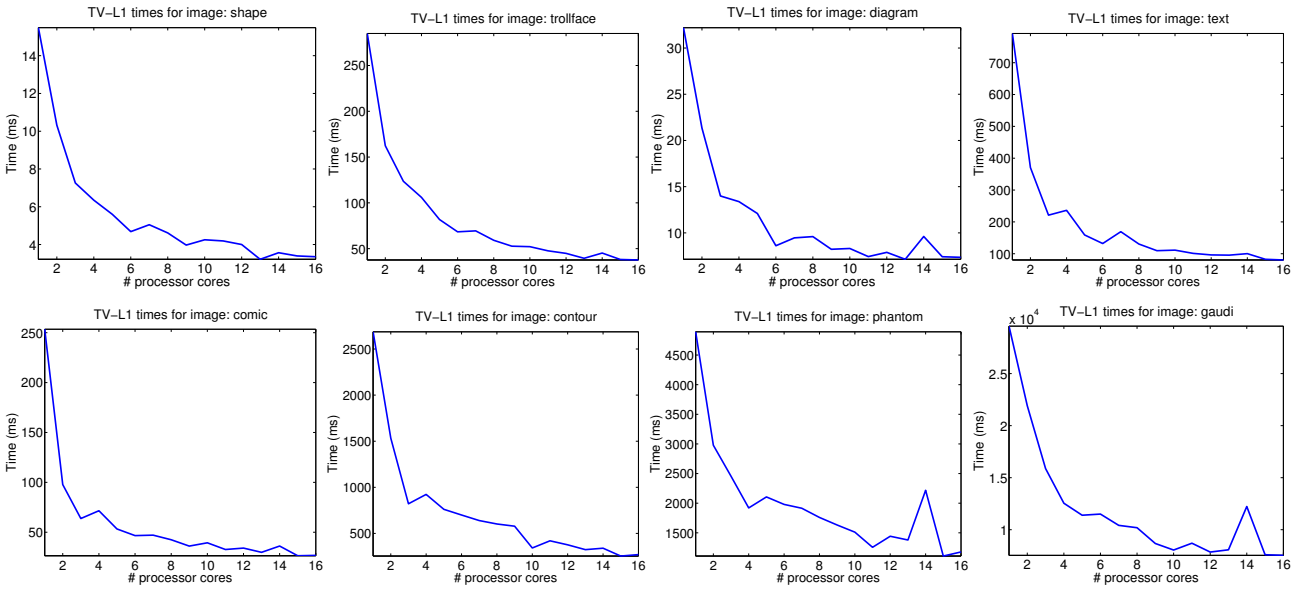


Figure 20: Multicore speedups on different images

where $\boldsymbol{N}$ is noise as before and $*$ denotes convolution. To recover the original image μ from the observed μ_0, it is common to solve the following deconvolution problem

$$\min_{\boldsymbol{X}} \quad \tfrac{1}{2}\|\boldsymbol{K} * \boldsymbol{X} - \mu\|_{\mathrm{F}}^2 + \lambda r(\boldsymbol{X}). \tag{5.4}$$

As before, the regularizer $r(\boldsymbol{X})$ can be isotropic or anisotropic TV, among others. Here we focus again on the anisotropic TV case to show how the presented solvers can also be used for this image task.

Problem (5.4) also fits the proximal splitting framework, and so we employ the popular FISTA (Beck and Teboulle, 2009) method for image processing. The gradient of the loss can be dealt efficiently by exploiting $\boldsymbol{K}$ being a convolution operator, which through the well–known convolution theorem is equivalent to a dot product in the frequencies space, and so the computation is done by means of fast Fourier transforms and products. Several other solvers that explicitly deal with convolution operators are also available (Afonso et al., 2010; Bioucas-Dias and Figueiredo, 2007). A notable solver specific for the isotropic case is given by the work of Krishnan and Fergus (2009), that handles even nonconvex isotropic TV-norms ($0 < p < 1$). But this approach does not extend to the anisotropic case, so we focus on general proximal splitting.

We use the same test images as for our denoising experiments (Figure 25), with identical noise patterns (Table 4) for the QR images, and gaussian noise with variance 0.05 for the rest. In addition, we convolve each image with a different type of kernel to assess the behavior for a variety of convolutions; Table 6 shows the kernels applied. We constructed these kernels using Matlab's *fspecial* function; the convolved images are shown in Figure 28.

The values for the regularizer λ were determined by maximizing the reconstruction quality measured in ISNR. Since deconvolution is much more expensive than denoising,

Image	Convolution	Parameters
randomQR	Motion	Length 5, Angle 35^o
shape	Average	Size 3×3
trollface	Disk	Radius 5
diagram	Motion	Length 5, Angle 0^o
text	Average	Size 1×10
comic	Gaussian	Size 15, Deviation 2
contour	Disk	Radius 5
phantom	Motion	Length 100, Angle 240^o

Table 6: Convolution kernels used for each test image. *Average* substitutes each pixel with the average of its surrounding $n \times m$ neighbors. *Disk* performs the same operation within a disk-shaped neighborhood of the shown radius. *Gaussian* uses a $n \times n$ neighborhood and assigns different weights to each neighbor following the value of a gaussian distribution of the indicated deviation centered at the current pixel. *Motion* emulates the distortions produced when taking a picture in motion, defining a neighborhood following a vector of the indicated length and angle.

instead of performing an exhaustive search for the best λ, we used a Focused Grid Search strategy (Barbero et al., 2008, 2009) to find the best performing values.

Any denoising subroutine can be plugged into the aforementioned deconvolution methods, however for comparison purposes we run our experiments with the best proposed method, Douglas Rachford (Alternating Reflections), and the best competing method among those reviewed from the literature, Kolmogorov et al. A key parameter in deconvolution performance is for how long should these methods be run at each FISTA iteration. To select this, we first run FISTA with 100 iterations of Douglas Rachford per step, for a large number of FISTA steps, and take the final objective value as an estimate of the optimum. Then we find the minimum number of Douglas Rachford and Kolmogorov iterations for which FISTA can achieve a relative distance to such optimum below 10^{-3}. The reason for doing this is that for larger distances the attained ISNR values are still far from convergence. This turned to be 5 iterations for Douglas Rachford an 10 for Kolmogorov. We then run FISTA for such configurations of the inner solvers, and others with a larger number of inner iterations, for comparison purposes.

Figures 21-22 show the evolution of objective values and ISNR for all the tested configurations. In general, Douglas Rachford seems to be slightly better at finding more accurate solutions, and also faster at converging to the final ISNR value. We explain this by the fact that the major advantage of Douglas Rachford is its aforementioned ability to find medium–quality solutions in a very small number of iterations: this is why with a small number of inner DR iterates we can converge to good ISRN levels.

For reference we also provide the resultant deconvoluted images as Figure 29.

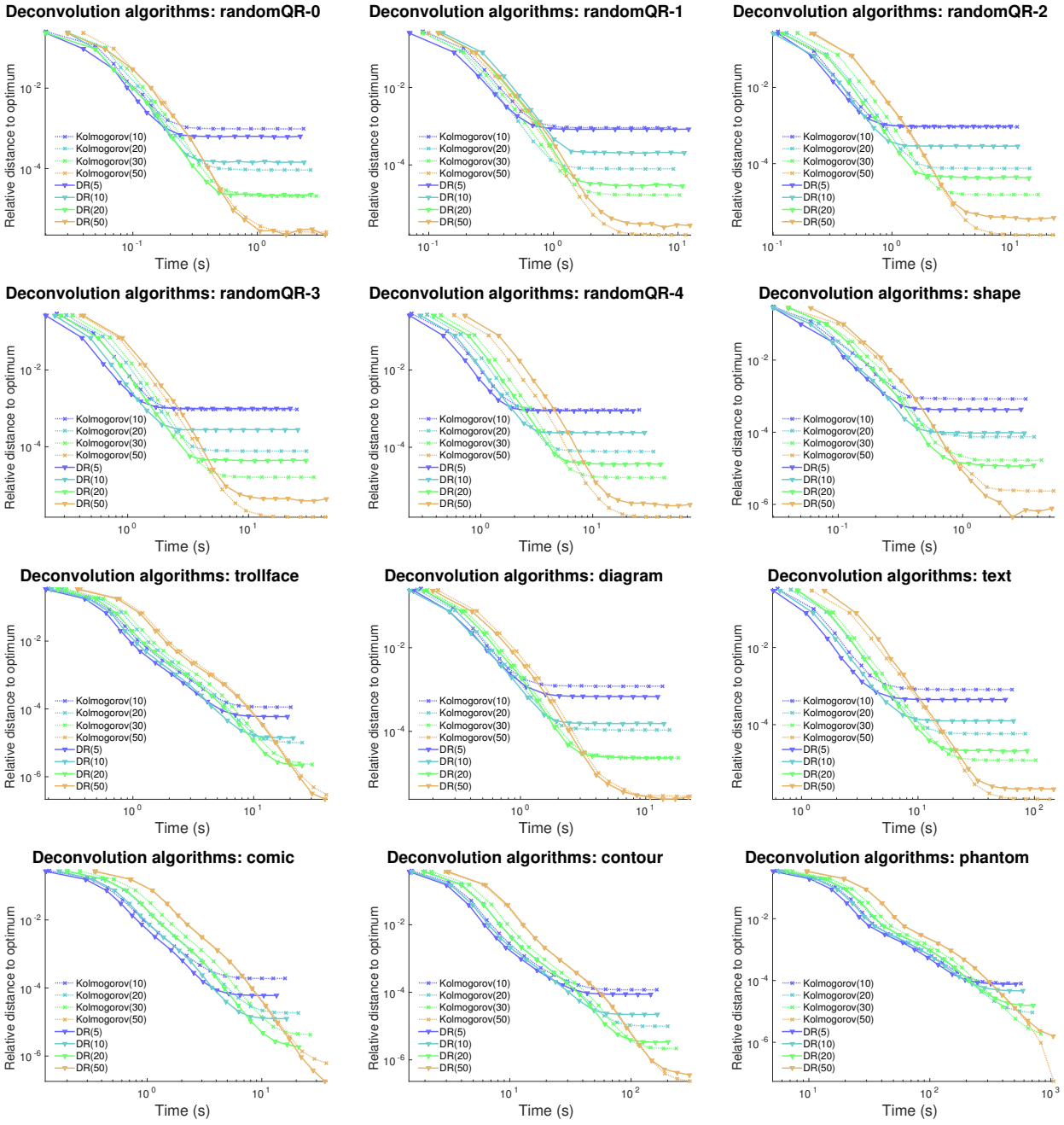


Figure 21: Relative distance to optimum vs time of the deconvolution 2D-TV algorithms under comparison, for the different images considered in the experiments.

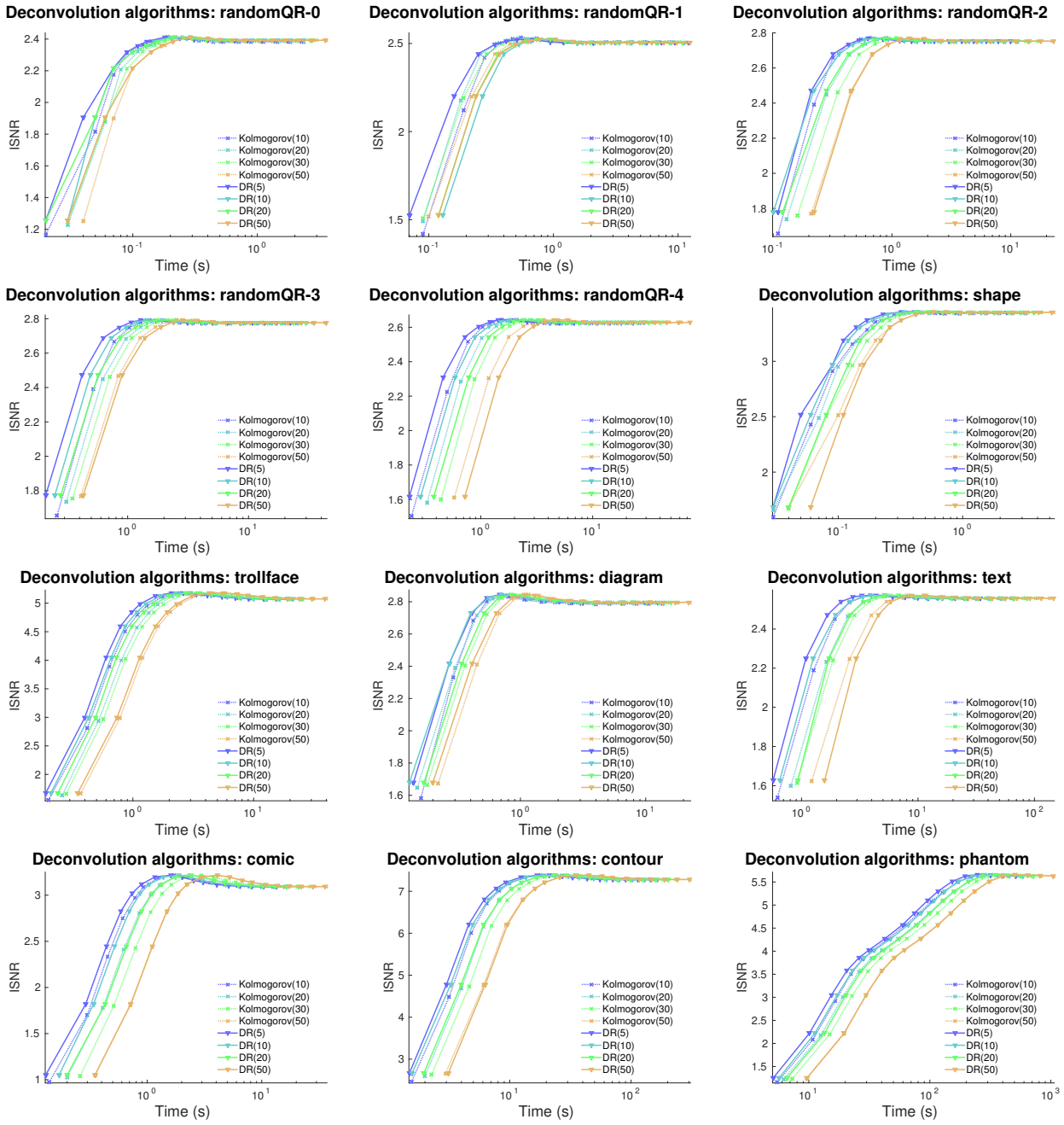


Figure 22: Increased Signal to Noise Ratio (ISNR) vs time of the deconvolution 2D-TV algorithms under comparison, for the different images considered in the experiments.

5.3.4. 2D Fused-Lasso Signal Approximator

The **Fused–Lasso Signal Approximator** (FLSA) (Friedman et al., 2007) can be regarded as a particular case of Fused-Lasso where the input matrix $\boldsymbol{A}$ is the identity matrix $\boldsymbol{I}$, i.e.,

$$\min_{\boldsymbol{x}} \quad \tfrac{1}{2}\|\boldsymbol{x}-\boldsymbol{y}\|_2^2 + \lambda_1\|\boldsymbol{x}\|_1 + \lambda_2 \mathrm{Tv}_1^{1\mathrm{D}}(\boldsymbol{x}).$$

This problem can be solved immediately using the methods presented in §5.2.4. A slightly less trivial problem is the one posed by the 2D variant of FLSA:

$$\min_{\boldsymbol{X}} \quad \tfrac{1}{2}\|\boldsymbol{X}-\boldsymbol{Y}\|_{\mathrm{F}}^2 + \lambda_1\|\mathrm{vec}(\boldsymbol{X})\|_1 + \lambda_2 \mathrm{Tv}_{1,1}^{2\mathrm{D}}(\boldsymbol{X}). \tag{5.5}$$

Friedman et al. (2007) used this model for denoising images where a large number of pixels are known to be completely black (intensity 0), which aligns well with the structure imposed by the ℓ_1 regularizer.

Akin to the 1D-case, 2D-FLSA (5.5) can also be solved by decomposing its computation into two prox operators (Friedman et al., 2007); formally,

$$\mathrm{prox}_{\lambda_1\|\cdot\|_1+\lambda_2\mathrm{Tv}_{1,1}^{2\mathrm{D}}(\cdot)}(\boldsymbol{Y}) = \mathrm{prox}_{\lambda_1\|\cdot\|_1}\big(\mathrm{prox}_{\lambda_2\mathrm{Tv}_{1,1}^{2\mathrm{D}}(\cdot)}(\boldsymbol{Y})\big).$$

Thus, to solve (5.5) we merely invoke one of the presented $\mathrm{Tv}_{1,1}^{2\mathrm{D}}$ prox operators and then apply soft-thresholding to the results. Since soft-thresholding is done in closed form, the performance of a 2D-FLSA solver depends only on its ability to compute $\mathrm{Tv}_{1,1}^{2\mathrm{D}}$-proximity efficiently. We can then safely claim that the results summarized in table 5 apply equivalently to 2D-FLSA, and so the proposed Douglas Rachford method performs best when reconstruction ISNR is the primary concern.

5.4. Application of Higher-Dimensional TV

We now apply the presented multidimensional TV regularizer to anisotropic filtering for **video denoising**. The extension to videos from images is natural. Say a video contains f frames of size $n \times m$ pixels; this video can be viewed as a 3D-tensor $\mathsf{X} \in \mathbb{R}^{n\times m\times f}$, on which a 3D-TV based filter can be effected by

$$\min_{\mathsf{X}} \quad \tfrac{1}{2}\|\mathsf{X}-\mathsf{U}_0\|_{\mathrm{F}}^2 + \lambda \mathrm{Tv}_{p_1,p_2,p_3}^{3\mathrm{D}}(\mathsf{X}), \tag{5.6}$$

where U_0 is the observed noisy video, and $\mathrm{Tv}_{p_1,p_2,p_3}^{3\mathrm{D}} = \mathrm{Tv}_{\boldsymbol{p}}^3$ with $\boldsymbol{p} = [p_1, p_2, p_3]$. Application of the filter (5.6) is nothing but computation of the prox operator, which can be done using the Parallel-Proximal Dykstra (PPD) algorithm presented in Sec. 4.

We apply this idea to the video sequences detailed in Table 7. All of the sequences are made of grayscale pixels. Figure 30 in the Appendix shows some of the frames of the *salesman* sequence. We noise every frame of these sequences by applying gaussian noise with zero mean and variance 0.01, using Matlab's *imnoise* function. Then we solve problem 5.6 for each sequence, adjusting the regularization value so as to maximize ISNR of the reconstructed signal. We test the following algorithms, which have been previously applied in the literature for solving 3D-TV, with the only exception Parallel Proximal Dykstra:

- Parallel Proximal Dykstra (§ 4.1.1).

Sequence	Frame resolution	Number of frames	Total number of pixels
salesman	288 × 352	50	5 million
coastguard	176 × 144	300	7.6 million
bicycle	720 × 576	30	12.4 million

Table 7: Size details of video sequences used in the video denoising experiments.

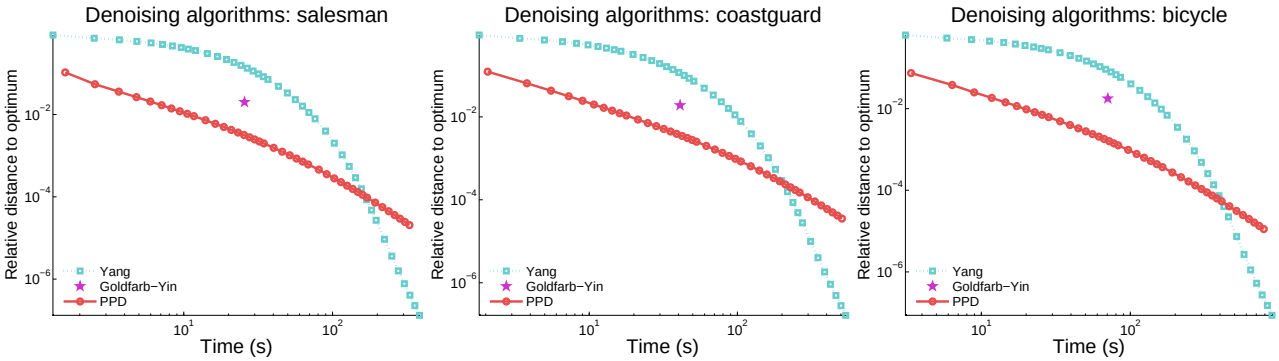


Figure 23: Relative distance to optimum vs time of the denoising 3D-TV algorithms under comparison, for the different video sequences considered in the experiments.

- Yang's method, which is based on ADMM (§ 4.1.1)
- The maximum flow approach by Goldfarb and Yin (2009), which features an implementation for 3D grids, thus solving a discrete-valued version of 3D-TV.

For both PPD and ADMM we again make use of linearized taut-string 1D TV solver. We must also point out that other image denoising methods seem amenable for extension into the multidimensional setting, such as Condat's and Chambolle-Pock methods. However in the light of our image denoising results we do not deem them as good choices for this problem. A more reasonable choice might be to extend Split-Bregman to multiple dimensions, but such an extension has not been implemented or proposed as far as we know. We would also like to note that we have considered extending the Douglas Rachford method to a multidimensional setting, however such task is complex and thus we decided to focus on Parallel Proximal Dykstra.

Similarly to our previous image denoising experiments, we ran the algorithms under comparison for each video sequence and measured its ISNR and relative distance to the optimal objective value of the current solution at each iteration through their execution. Again the exception is the Goldfarb-Yin method, which is non–iterative and so we only report the time required for its termination. The optimal objective value was estimated by running all methods for a very large number of iterations and taking the minimum value of them all. This produced the plots shown in Figures 23–24. From them the following observations are of relevance:

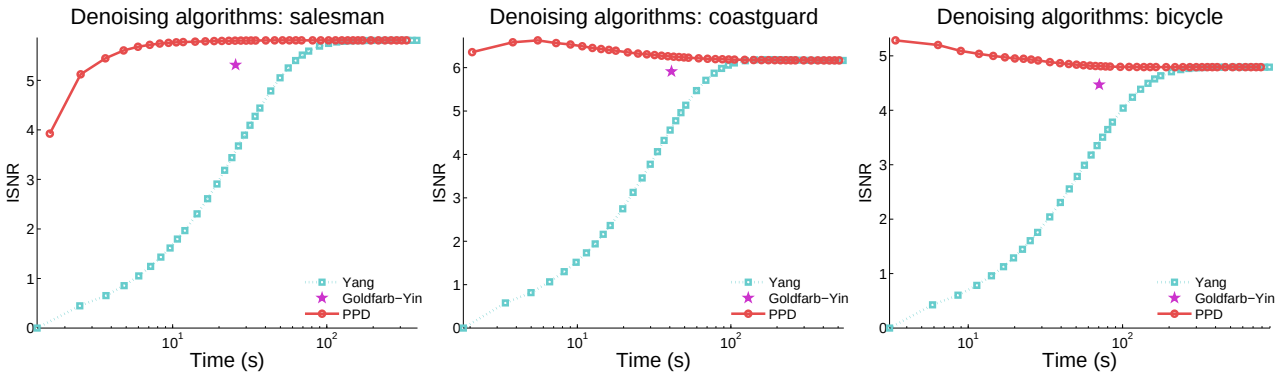


Figure 24: Increased Signal to Noise Ratio (ISNR) vs time of the denoising 3D-TV algorithms under comparison, for the different video sequences considered in the experiments.

- Following the pattern observed in the image denoising experiments, ADMM (Yang's method) is best suited for finding very accurate solutions.
- The method by Goldfarb and Yin again provides suboptimal solutions, due to the discrete approximation it uses.
- Parallel Proximal Dykstra is the fastest to achieve a mid-quality solution.
- Intermediate solutions prior to convergence of the PPD run result in better ISNR values for the *coastguard* and *bicycle* data sets. This hints that the denoising model used in this experiment may not be optimal for these kind of signals; indeed, more advanced denoising models abound in the signal processing literature. Hence we do not claim novel results in terms of ISNR quality, but just in solving this classic denoising model more efficiently.

The ISNR plots in Figure 24 also show how both Parallel Proximal Dykstra and ADMM (Yang's method) converge to equivalent solutions in practice. Therefore, for the purpose of video denoising PPD seems to be the best choice, unless for some reason a high degree of accuracy is required, for which ADMM should be preferred.

Acknowledgments

ÁB acknowledges partial financial support from Spain's grants TIN2010-21575-C02-01, TIN2013-42351-P, S2013/ICE-2845 CASI-CAM-CM, TIN2016-76406-P, TIN2015-70308-REDT (MINECO/FEDER EU) and project "FACIL—Ayudas Fundación BBVA a Equipos de Investigación Científica 2016" during the long research period leading to the writing of this manuscript. We thank R. Tibshirani for bringing (Johnson, 2013) to our attention, and S. Jegelka for alerting us to the importance of weighted total-variation problems.

Appendix A. Mathematical Background

We begin by recalling a few basic ideas from convex analysis; we recommend the recent book (Bauschke and Combettes, 2011) for more details.

Let $\mathcal{X} \subset \mathbb{R}^n$ be any set. A function $r : \mathcal{X} \to \mathbb{R} \cup \{-\infty, +\infty\}$ is called *lower semicontinuous* if for every $\boldsymbol{x} \in \mathcal{X}$ and a sequence $(\boldsymbol{x}_k)$ that converges to $\boldsymbol{x}$, it holds that

$$\boldsymbol{x}_k \to \boldsymbol{x} \implies r(\boldsymbol{x}) \leq \liminf_k r(\boldsymbol{x}_k). \tag{A.1}$$

The set of proper lsc convex functions on $\mathcal{X}$ is denoted by $\Gamma_0(\mathcal{X})$ (such functions are also called *closed convex functions*). The *indicator function* of a set C is defined as

$$\delta_C : \mathcal{X} \to [0, \infty] : \boldsymbol{x} \mapsto \begin{cases} 0, & \text{if } \boldsymbol{x} \in C; \\ \infty, & \text{if } \boldsymbol{x} \notin C, \end{cases} \tag{A.2}$$

which is lsc if and only if C is closed.

The *convex conjugate* of r is given by $r^*(\boldsymbol{z}) := \sup_{\boldsymbol{x} \in \operatorname{dom} r} \ \langle \boldsymbol{x}, \boldsymbol{z} \rangle - r(\boldsymbol{x})$, and a particularly important example is the Fenchel conjugate of a norm $\|\cdot\|$

$$\text{if } r = \|\cdot\|, \quad \text{then } \ r^* = \delta_{\|\cdot\|_* \leq 1}, \tag{A.3}$$

where the norm $\|\cdot\|_*$ is dual to $\|\cdot\|$. Let r and h be proper convex functions. The *infimal convolution* of r with h is the convex function given by $(r \square h)(\boldsymbol{x}) := \inf_{\boldsymbol{y} \in \mathcal{X}} \big(r(\boldsymbol{y}) + h(\boldsymbol{x} - \boldsymbol{y})\big)$. For our purposes, the most important special case is infimal convolution of a convex function with the squared euclidean norm, which yields the *Moreau envelope* (Moreau, 1962).

Proposition A.1 *Let $r \in \Gamma_0(\mathcal{X})$ and let $\gamma > 0$. The* Moreau envelope *of r indexed by γ is*

$$E_r^\gamma(\cdot) := r \,\square\, (\tfrac{1}{2\gamma}\|\cdot\|_2^2). \tag{A.4}$$

The Moreau envelope (A.4) *is convex, real-valued, and continuous.*

Proof See e.g. (Bauschke and Combettes, 2011, Prop. 12.15). ■

Using the Moreau envelope (A.4), we now formally introduce prox operators.

Definition A.2 (Prox operator) *Let $r \in \Gamma_0(\mathcal{X})$, and let $\boldsymbol{y} \in \mathcal{X}$. Then $\operatorname{prox}_r \boldsymbol{y}$ is the unique point in $\mathcal{X}$ that satisfies $E_r^1(\boldsymbol{y}) = \min_{\boldsymbol{x} \in \mathcal{X}} (r(\boldsymbol{x}) + \frac{1}{2}\|\boldsymbol{x} - \boldsymbol{y}\|_2^2)$, i.e.,*

$$\operatorname{prox}_r(\boldsymbol{y}) := \operatorname*{argmin}_{\boldsymbol{x} \in \mathcal{X}} r(\boldsymbol{x}) + \tfrac{1}{2}\|\boldsymbol{x} - \boldsymbol{y}\|_2^2, \tag{A.5}$$

and the nonlinear map $\operatorname{prox}_r : \mathcal{X} \to \mathcal{X}$ is called the prox operator *of r.*

Sometimes the Fenchel conjugate r^* is easier to use than r; similarly, sometimes the operator $\operatorname{prox}_{r^*}$ is easier to compute than prox_r. The result below shows the connection.

Proposition A.3 (Moreau decomposition) *Let $r \in \Gamma_0(\mathcal{X})$, $\gamma > 0$, and $\boldsymbol{y} \in \mathcal{X}$. Then,*

$$\boldsymbol{y} = \operatorname{prox}_{\gamma r} \boldsymbol{y} + \gamma \operatorname{prox}_{r^*/\gamma}(\gamma^{-1}\boldsymbol{y}). \tag{A.6}$$

Proof A brief exercise; see e.g., (Bauschke and Combettes, 2011, Thm. 14.3). ■

This decomposition provides the necessary tools to exploit useful primal–dual relations. For the sake of clarity we also present an additional result regarding a particular primal-dual relation that plays a key role in our algorithms.

Proposition A.4 *Let $f \in \Gamma_0(\mathcal{X})$ and $r \in \Gamma_0(\mathcal{Z})$. The problems below form a primal-dual pair.*

$$\inf_{\boldsymbol{x}\in\mathcal{X}} \quad f(\boldsymbol{x}) + r(\boldsymbol{B}\boldsymbol{x}) \quad \text{s.t.} \quad \boldsymbol{B}\boldsymbol{x} \in \mathcal{Z} \tag{A.7}$$

$$\inf_{\boldsymbol{u}\in\mathcal{Z}} \quad f^*(-\boldsymbol{B}^T\boldsymbol{u}) + r^*(\boldsymbol{u}). \tag{A.8}$$

Proof Introduce an extra variable $\boldsymbol{z} = \boldsymbol{B}\boldsymbol{x}$, dual function is

$$g(\boldsymbol{u}) = \inf_{\boldsymbol{x}\in\mathcal{X}} \; f(\boldsymbol{x}) + \boldsymbol{u}^T\boldsymbol{B}\boldsymbol{x} + \inf_{\boldsymbol{z}\in\mathcal{Z}} \; r(\boldsymbol{z}) - \boldsymbol{u}^T\boldsymbol{z},$$

which upon rewriting using Fenchel conjugates yields (A.8). ■

Notions on submodular optimization are also required to introduce some of the decomposition techniques for 2D-TV in this paper. For a more thorough read on this topic we recommend the monograph Bach (2013).

Definition A.5 (Submodular function) *A set-function $F : 2^V \to \mathbb{R}$, for 2^V the power set of some set V, is submodular if and only if it fulfills the diminishing returns property, that is, for $A \subseteq B \subseteq V$ and $k \in V$, $k \notin B$ we have*

$$F(A \cup \{k\}) - F(A) \geq F(B \cup \{k\}) - F(B).$$

Intuitively, a set-function is submodular if adding a new element to the set results in less value as the set grows in size.

Definition A.6 (Modular function) *A set-function $F : 2^V \to \mathbb{R}$, for 2^V the power set of some set V, $F(\emptyset) = 0$ is modular (and also submodular) if and only if there exists $\boldsymbol{s} \in \mathbb{R}^p$ such that $F(A) = \sum_{k\in A} \boldsymbol{s}_k$.*

That is, a function is modular if it always assigns the same value for each element added to the set, regardless of the other elements in the set. A common shorthand for modular functions is $s(A) = \sum_{k\in A} \boldsymbol{s}_k$.

Submodular functions can be thought as convex functions in the realm of discrete optimization, in the sense that they feature useful properties that allow for efficient optimization. Similarly, modular functions are connected to linear functions. To make such connections explicit we require of the following geometric concepts.

Definition A.7 (Base polytope) *The base polytope B_F of a submodular function F is the polyhedron given by*

$$B_F = \{y \in \mathbb{R}^n : y(A) \leq F(A) \; \forall A \subseteq V, \quad y(V) = F(V)\}.$$

That is, the base polytope is a polyhedron defined through linear inequality constraints on the values of F for every one of the n elements of the powerset 2^V, and an equality constraint for the complete set. This results in a combinatorial number of contraints, but fortunately this polytope will not be used directly.

Definition A.8 (Support function) *The support function h_A for some non-empty closed convex set $A \in \mathbb{R}^n$ is given by*

$$h_A(\boldsymbol{x}) = \sup\left\{\boldsymbol{x}^T\boldsymbol{a} : \boldsymbol{x} \in A\right\}.$$

The support function is useful when connected with the following definition.

Definition A.9 (Lovász extension) *Suppose a set-function F such that $F(\emptyset) = 0$. Its Lovász extension $f : \mathbb{R}^p \to \mathbb{R}$ is defined through the following mechanism. Take $\boldsymbol{w} \in \mathbb{R}^p$ input to f, and order its components in decreasing order $\boldsymbol{w}_{j_1} \geq \ldots \geq \boldsymbol{w}_{j_p}$, then*

$$f(\boldsymbol{w}) = \sum_{k=1}^{p}[F(\{j_1, \ldots, j_k\}) - F(\{j_1, \ldots, j_{k-1}\})].$$

Other equivalent definitions are possible: see Bach (2013) for details. The following result links all the definitions so far.

Proposition A.10 *For F submodular function such that $F(\emptyset) = 0$ we have*

- *Its Lovász extension f is a convex function.*
- *The support function of its base polytope is equal to its Lovász extension, that is, $h_{B_F}(\boldsymbol{x}) = f(\boldsymbol{x})$.*
- *The problem $\min_{S\subseteq V} F(S)$ is dual to $\min_{\boldsymbol{x}} f(\boldsymbol{x}) + \frac{1}{2}\|\boldsymbol{x}\|_2^2$, with $S^* = \{k|\boldsymbol{x}_k^* \geq 0\}$.*

For proofs on these points we refer to Bach (2013). The takeaway from them is that any minimization on a submodular function can be cast into a convex optimization problem. Furthermore, for those convex minimization problems whose objective turns out to be the Lovász extension of some other function, we can trace the steps the other way round, obtaining the minimization of a submodular function.

Consider now a composite problem $\min_{S\subseteq V} \sum_j F_j(S)$. The following results hold

Proposition A.11 *The problem $\min_{S\subseteq V} \sum_j F_j(S)$ is equivalent to $\min_x \sum_j f_j(x) + \frac{1}{2}\|x\|_2^2$, with $S^* = \{k|x_k^* \geq 0\}$. Furthermore it is also equivalent to $\min_{y_j \in B_{F_j}\, \forall j} \frac{1}{2}\|\sum_j y_j\|_2^2$, with $x^* = -\sum_j y_j^*$.*

Proof The first equivalence is a direct result of the properties of Lovász extensions (Bach, 2013), in particular that for F, G set-functions with Lovász extensions f, g, the Lovász

extension of $F + G$ is $f + g$. For the second equivalence we have:

$$\begin{aligned}
\min_{\boldsymbol{x}} \sum_j f_j(\boldsymbol{x}) + \tfrac{1}{2}\|\boldsymbol{x}\|_2^2 &= \min_{\boldsymbol{x}} \sum_j h_{B_{F_j}} + \tfrac{1}{2}\|\boldsymbol{x}\|_2^2, \\
&= \min_{\boldsymbol{x}} \sum_j \max_{y_j \in B_{F_j}} \boldsymbol{y}_j^T \boldsymbol{x} + \tfrac{1}{2}\|\boldsymbol{x}\|_2^2, \\
&= \max_{\boldsymbol{y}_j \in B_{F_j}\, \forall j} \min_{\boldsymbol{x}} \left(\sum_j \boldsymbol{y}_j^T \boldsymbol{x} + \tfrac{1}{2}\|\boldsymbol{x}\|_2^2 \right), \\
&= \min_{\boldsymbol{y}_j \in B_{F_j}\, \forall j} \tfrac{1}{2}\|\sum_j \boldsymbol{y}_j\|_2^2,
\end{aligned}$$

and the dual relationship $\boldsymbol{x}^* = -\sum_j \boldsymbol{y}_j^*$ comes from solving the inner $\min_{\boldsymbol{x}}$ problem for $\boldsymbol{x}$. ■

Therefore any decomposable submodular minimization, or sum of Lovász extensions plus ℓ_2 term, can be casted into a geometric problem in terms of the base polytopes. For two functions the resultant problem is of special interest if rewritten as

$$\min_{\substack{\boldsymbol{y}_1 \in B_{F_1} \\ \boldsymbol{y}_2 \in B_{F_2}}} \tfrac{1}{2}\|\boldsymbol{y}_1 + \boldsymbol{y}_2\|_2^2 \quad = \quad \min_{\substack{\boldsymbol{y}_1 \in B_{F_1} \\ -\boldsymbol{y}_2 \in -B_{F_2}}} \tfrac{1}{2}\|\boldsymbol{y}_1 - (-\boldsymbol{y}_2)\|_2^2 \quad = \quad \min_{\substack{\boldsymbol{a} \in B_{F_1} \\ \boldsymbol{b} \in -B_{F_2}}} \tfrac{1}{2}\|\boldsymbol{a} - \boldsymbol{b}\|_2^2$$

with $\boldsymbol{a} = \boldsymbol{y}_1$, $\boldsymbol{b} = -\boldsymbol{y}_2$, as this results in the classic geometric problem of finding the closest points between two convex sets. Many algorithms have been proposed to tackle problems in this form, most of them making use of alternating projection operations onto the two sets. Thus, a legitimate concern is how easy it is to compute such projections for B_{F_1} and $-B_{F_2}$.

Proposition A.12 *Given a submodular function F and its base polytope B_F, the projections $\Pi_{B_F}(\boldsymbol{z})$ and $\Pi_{-B_F}(\boldsymbol{z})$ of a point $\boldsymbol{z}$ onto B_F or its negated counterpart can be computed as*

$$\begin{aligned}
\Pi_{B_F}(\boldsymbol{z}) &= \boldsymbol{z} - \operatorname{prox}_f(\boldsymbol{z}), \\
\Pi_{-B_F}(\boldsymbol{z}) &= \boldsymbol{z} + \operatorname{prox}_f(-\boldsymbol{z}),
\end{aligned}$$

with prox *proximity operator of a function, f the Lovász extension of F.*

Proof We start with the proximity of f and work our way to a relationship with the projection operator,

$$\begin{aligned}
\operatorname{prox}_f(\boldsymbol{z}) &\equiv \min_{\boldsymbol{x}} f(\boldsymbol{x}) + \tfrac{1}{2}\|\boldsymbol{x} - \boldsymbol{z}\|_2^2, \\
&= \max_{\boldsymbol{y} \in B_F} \min_{\boldsymbol{x}} \boldsymbol{y}^T \boldsymbol{x} + \tfrac{1}{2}\|\boldsymbol{x} - \boldsymbol{z}\|_2^2, \\
&= \max_{\boldsymbol{y} \in B_F} \boldsymbol{y}^T(\boldsymbol{z} - \boldsymbol{y}) + \tfrac{1}{2}\|(\boldsymbol{z} - \boldsymbol{y}) - \boldsymbol{z}\|_2^2, \\
&= \min_{\boldsymbol{y} \in B_F} \tfrac{1}{2}\|\boldsymbol{y}\|_2^2 - \boldsymbol{y}^T \boldsymbol{z}, \\
&\equiv \min_{\boldsymbol{y} \in B_F} \tfrac{1}{2}\|\boldsymbol{y} - \boldsymbol{z}\|_2^2 = \Pi_{B_F}(\boldsymbol{z}),
\end{aligned}$$

where solving the inner minimization problem for $\boldsymbol{x}$ gives the primal–dual relationship $\boldsymbol{x}^* = \boldsymbol{z} - \boldsymbol{y}^*$. Using this we can obtain the solution for the projection problem from the proximity problem, as $\Pi_{B_F}(\boldsymbol{z}) = \boldsymbol{z} - \text{prox}_f(\boldsymbol{z})$. Projection onto the negated base polytope follows from the basic geometric argument $\Pi_{-B_F}(\boldsymbol{z}) = -\Pi_{B_F}(-\boldsymbol{z})$. ■

Appendix B. proxTV Toolbox

All the Total–Variation proximity solvers in this paper have been implemented as the **proxTV** toolbox for C++, Matlab and Python, available at `https://github.com/albarji/proxTV`. The toolbox has been designed to be used out of the box in a user friendly way; for instance, the top–level Matlab function `TV` solves Total–Variation proximity for a given signal under a variety of settings. For instance

```
>> TV(X, lambda)
```

solves Tv_1 proximity for a signal `X` of any dimension and a regularization value `lambda`. The weighted version of this problem is also seamlessly tackled by just providing a vector of weights of the appropriate length as the `lambda` parameter.

If a third parameter `p` is provided as

```
>> TV(X, lambda, p)
```

the general Tv_p proximity problem is addressed, whereupon an adequate solver is chosen by the library.

More advanced uses of the library are possible, allowing to specify which norm `p` and regularizer `lambda` values to use for each dimension of the signal, and even applying combinations of several different Tv_p regularizers along the same dimension. Please refer to the documentation within the toolbox for further information.

Appendix C. Proof on the Equality of Taut-String Problems

Theorem C.1 (Equality of taut-string problems) *Given the problems*

$$\min_{\boldsymbol{s}} \sum_{i=1}^{n} (\boldsymbol{s}_i - \boldsymbol{s}_{i-1})^2 \,, \;\; s.t. \;\; |\boldsymbol{s}_i - \boldsymbol{r}_i| \leq \boldsymbol{w}_i \, \forall i = 1, \ldots, n-1 \,, \boldsymbol{s}_0 = 0, \boldsymbol{s}_n = \boldsymbol{r}_n, \tag{C.1}$$

and

$$\min_{\hat{\boldsymbol{s}}} \sum_{i=1}^{n} \sqrt{1 + (\hat{\boldsymbol{s}}_i - \hat{\boldsymbol{s}}_{i-1})^2}, \;\; s.t. \;\; |\hat{\boldsymbol{s}}_i - \boldsymbol{r}_i| \leq \boldsymbol{w}_i \, \forall i = 1, \ldots, n-1 \,, \hat{\boldsymbol{s}}_0 = 0, \hat{\boldsymbol{s}}_n = \boldsymbol{r}_n, \tag{C.2}$$

for a non-zero vector $\boldsymbol{w}$, both problems share the same minimum $\boldsymbol{s}^* = \hat{\boldsymbol{s}}^*$.

Proof

The Lagrangian of problem C.1 takes the form

$$L(\boldsymbol{s}, \boldsymbol{\alpha}, \boldsymbol{\beta}) = \sum_{i=1}^{n} (\boldsymbol{s}_i - \boldsymbol{s}_{i-1})^2 + \sum_{i=1}^{n-1} \boldsymbol{\alpha}_i (\boldsymbol{s}_i - \boldsymbol{r}_i - \boldsymbol{w}_i) + \sum_{i=1}^{n-1} \boldsymbol{\beta}_i (-\boldsymbol{w}_i - \boldsymbol{s}_i + \boldsymbol{r}_i),$$

and its Karush-Kuhn-Tucker optimality conditions are given by

$$(\boldsymbol{s}_{i+1} - \boldsymbol{s}_i) - (\boldsymbol{s}_i - \boldsymbol{s}_{i-1}) = \boldsymbol{\alpha}_i - \boldsymbol{\beta}_i, \tag{C.3}$$
$$|\boldsymbol{s}_i - \boldsymbol{r}_i| \leq \boldsymbol{w}_i, \tag{C.4}$$
$$\boldsymbol{\alpha}_i, \boldsymbol{\beta}_i \geq 0, \tag{C.5}$$
$$\boldsymbol{\alpha}_i(\boldsymbol{s}_i - \boldsymbol{r}_i - \boldsymbol{w}_i) = 0, \tag{C.6}$$
$$\boldsymbol{\beta}_i(-\boldsymbol{w}_i - \boldsymbol{s}_i + \boldsymbol{r}_i) = 0, \tag{C.7}$$

$\forall i = 1, \ldots, n-1$, and where the first equation comes from the fact that $\frac{\partial L(\boldsymbol{s}, \boldsymbol{\alpha}, \boldsymbol{\beta})}{\partial \boldsymbol{s}} = 0$ at the minimum.

As the only difference between problems C.1 and C.2 is in the form of the objective, the KKT conditions for problem C.2 take the same form, but for the first one,

$$\frac{(\hat{\boldsymbol{s}}_{i+1} - \hat{\boldsymbol{s}}_i)}{\sqrt{1 + (\hat{\boldsymbol{s}}_{i+1} - \hat{\boldsymbol{s}}_i)^2}} - \frac{(\hat{\boldsymbol{s}}_i - \hat{\boldsymbol{s}}_{i-1})}{\sqrt{1 + (\hat{\boldsymbol{s}}_i - \hat{\boldsymbol{s}}_{i-1})^2}} = \hat{\boldsymbol{\alpha}}_i - \hat{\boldsymbol{\beta}}_i, \tag{C.8}$$
$$|\hat{\boldsymbol{s}}_i - \boldsymbol{r}_i| \leq \boldsymbol{w}_i, \tag{C.9}$$
$$\hat{\boldsymbol{\alpha}}_i, \hat{\boldsymbol{\beta}}_i \geq 0, \tag{C.10}$$
$$\hat{\boldsymbol{\alpha}}_i(\hat{\boldsymbol{s}}_i - \boldsymbol{r}_i - \boldsymbol{w}_i) = 0, \tag{C.11}$$
$$\hat{\boldsymbol{\beta}}_i(-\boldsymbol{w}_i - \hat{\boldsymbol{s}}_i + \boldsymbol{r}_i) = 0, \tag{C.12}$$

$\forall i = 1, \ldots, n-1$, and where we use hat notation for the dual coefficients to tell them apart from those of problem C.1.

Suppose $\boldsymbol{s}^*$ minimizer to problem C.1, hence fulfilling the conditions C.3-C.7. In particular this means that it is feasible to assign values to the dual coefficients $\boldsymbol{\alpha}, \boldsymbol{\beta}$ in such a way that the conditions above are met. If we set $\hat{\boldsymbol{s}} = \boldsymbol{s}^*$ in the conditions C.8-C.12 the following observations are of relevance

- Condition C.9 becomes the same as condition C.4, and so it is immediately met.
- The operator $f(x) = \frac{x}{\sqrt{1+x^2}}$ is contractive and monotonous.
- The couple $(\boldsymbol{\alpha}_i, \boldsymbol{\beta}_i)$ cannot be both non–zero at the same time, since $\boldsymbol{\alpha}_i > 0$ enforces $\boldsymbol{s}_i = \boldsymbol{r}_i + \boldsymbol{w}_i$ and $\boldsymbol{\beta}_i > 0$ enforces $\boldsymbol{s}_i = \boldsymbol{r}_i - \boldsymbol{w}_i$, and $\boldsymbol{w}_i$ is non–zero.
- Hence and because $\boldsymbol{\alpha}_i, \boldsymbol{\beta}_i \geq 0$ and condition C.3 holds, when $(\boldsymbol{s}_{i+1} - \boldsymbol{s}_i) - (\boldsymbol{s}_i - \boldsymbol{s}_{i-1}) > 0$ then $\boldsymbol{\alpha}_i > 0$, $\boldsymbol{\beta}_i = 0$, and when $(\boldsymbol{s}_{i+1} - \boldsymbol{s}_i) - (\boldsymbol{s}_i - \boldsymbol{s}_{i-1}) < 0$ then $\boldsymbol{\alpha}_i = 0$, $\boldsymbol{\beta}_i > 0$.
- $f(\boldsymbol{s}_{i+1} - \boldsymbol{s}_i) - f(\boldsymbol{s}_i - \boldsymbol{s}_{i-1})$ has the same sign as $(\boldsymbol{s}_{i+1} - \boldsymbol{s}_i) - (\boldsymbol{s}_i - \boldsymbol{s}_{i-1})$, since f is monotonous and as such preserves ordering.
- Since f is contractive, condition C.8 can be met by setting $(\hat{\boldsymbol{\alpha}}_i, \hat{\boldsymbol{\beta}}_i) = (k\boldsymbol{\alpha}_i, k\boldsymbol{\beta}_i)$ for some $0 \leq k < 1$. Note that this works because $(\boldsymbol{\alpha}_i, \boldsymbol{\beta}_i)$ cannot be both zero at the same time.
- Condition C.10 is met for those choices of $\hat{\boldsymbol{\alpha}}_i, \hat{\boldsymbol{\beta}}_i$, as C.5 was met for $\boldsymbol{\alpha}_i, \boldsymbol{\beta}_i$ and $0 \leq k < 1$.

- Conditions C.11 and C.12 are also met for those choices of $\hat{\boldsymbol{\alpha}}_i, \hat{\boldsymbol{\beta}}_i$, as $\hat{\boldsymbol{\alpha}}_i(\boldsymbol{s}_i - \boldsymbol{r}_i - \boldsymbol{w}_i) = k\boldsymbol{\alpha}_i(\boldsymbol{s}_i - \boldsymbol{r}_i - \boldsymbol{w}_i) = 0$ and $\hat{\boldsymbol{\beta}}_i(-\boldsymbol{w}_i - \boldsymbol{s}_i + \boldsymbol{r}_i) = k\boldsymbol{\beta}_i(-\boldsymbol{w}_i - \boldsymbol{s}_i + \boldsymbol{r}_i) = 0$.

Therefore, all of the optimality conditions C.8-C.12 for problem C.2 are met for $\boldsymbol{s}^*$ solution of problem C.1, and so a minimum of problem C.1 is also a minimum for problem C.2.

The proof can be repeated the other way round by setting $\boldsymbol{s} = \hat{\boldsymbol{s}}^*$ optimal for problem C.2, defining the operator $f^{-1}(x) = \frac{x}{\sqrt{1-x^2}}$, and observing that this operator is monotonous and expansive, so we can establish $(\boldsymbol{\alpha}_i, \boldsymbol{\beta}_i) = (k\hat{\boldsymbol{\alpha}}_i, k\hat{\boldsymbol{\beta}}_i)$ for some $k \geq 1$ and the optimality conditions C.3-C.7 for problem C.1 are met following a similar reasoning to the one presented above. Thus, a minimum for problem C.2 is also a minimum for problem C.1, which joined with the previous result completes the proof.

■

Appendix D. Proof on the Equivalence of Linearized Taut-String Method

Proposition D.1 *Using affine approximations to the greatest convex minorant and the smallest concave majorant does not change the solution of the taut-string method.*

Proof Let us note $\cap(f)$ as the smallest concave majorant of some function f taking integer values, $\cup(f)$ as the greatest concave minorant, $\bar{a}(f)$ as the smallest affine majorant and $\underline{a}(f)$ as the greatest affine minorant. By definition we have

$$\underline{a}(f(i)) \leq \cup(f(i)) \leq f(i) \leq \cap(f(i)) \leq \bar{a}(f(i)) \quad \forall i \in \mathbb{Z}$$

Consider now the nature of the taut-string problem, where a vertically symmetric tube of radius λ_i at each section is modelled by following the majorant of the tube bottom $(f - \lambda)$ and the minorant of the tube ceiling $(f + \lambda)$. We work the inequalities above as:

$$\begin{aligned} f(i) - \lambda_i \leq \cap(f(i) - \lambda_i) \leq \bar{a}(f(i) - \lambda_i) \\ \underline{a}(f(i) + \lambda_i) \leq \cup(f(i) + \lambda_i) \leq f(i) + \lambda_i \end{aligned}$$

We will show that an overlap of smallest concave majorant / greatest convex minorant takes place iff the same overlap happens when using the affine approximations. We formally define overlap as the setting where for a point i we have $\cup(f_i + \lambda_i) \leq \cap(f_i - \lambda_i)$.

One side of the implication is easy: if $\cup(f(i) + \lambda_i) \leq \cap(f(i) - \lambda)$ for some i, then using the relations above we have $\underline{a}(f(i) + \lambda_i) \leq \cup(f(i) + \lambda_i) \leq \cap(f(i) - \lambda_i) \leq \bar{a}(f(i) - \lambda_i)$, and so the affine approximation detects any overlap taking place in the concave/convex counterpart.

The opposite requires the key observation that in the taut-string method both majorant and minorant functions are clamped to the same point of origin: $f(0) = 0$ at the start of the method, or the point where the last segment was fixed after each restart. Let us assume $f(0) = 0$ without loss of generality. Suppose now that an overlap is detected by the affine approximation. Because of this affine nature the majorant/minorant slopes are constant,

i.e.

$$\bar{\delta}_1 = \bar{\delta}_2 = \ldots = \bar{\delta}_n = \bar{\delta}, \quad \underline{\delta}_1 = \underline{\delta}_2 = \ldots = \underline{\delta}_n = \underline{\delta}.$$

However, if we consider the convex/concave approximations these slopes can increase/decrease as the segment progresses, that is:

$$\delta_1^{\cup} \leq \delta_2^{\cup} \leq \ldots \leq \delta_n^{\cup}, \quad \delta_1^{\cap} \geq \delta_2^{\cap} \geq \ldots \geq \delta_n^{\cap}.$$

Consider now the majorant/minorant values, expressed through the slopes and taking into account the observation above about the starting point.

$$\cap(f(i) - \lambda_i) = \sum_{j=1}^{i} \delta_j^{\cap}, \quad \cup(f(i) + \lambda_i) \quad = \sum_{j=1}^{i} \delta_j^{\cup}, \quad \bar{a}(f(i) - \lambda_i) = i\bar{\delta}, \quad \underline{a}(f(i) + \lambda_i) \quad = i\underline{\delta}.$$

Since an overlap has been detected in the affine approximation, we have that for some point i

$$i\underline{\delta} = \underline{a}(f(i) + \lambda_i) \leq \bar{a}(f(i) - \lambda_i) = i\bar{\delta},$$

so $\underline{\delta} \leq \bar{\delta}$. Consider now the values of the affine minorant/majorant at the point immediately after the origin,

$$\underline{a}(f_1 - \lambda_1) = \underline{\delta}, \quad \bar{a}(f_1 + \lambda_1) = \bar{\delta}.$$

We will show now that the convex/convex counterpart must take exactly the same values at these points. To do so we take into account the following fact: there must exist points x and y, $x, y \leq i$, where

$$\underline{a}(f_x + \lambda_x) = f_x + \lambda_x = \cup(f_x + \lambda_x), \quad \bar{a}(f_y - \lambda_y) = f_y - \lambda_y = \cap(f_y - \lambda_y),$$

that is to say, the affine minorant/majorant must touch the tube ceiling/bottom at some point, otherwise we could obtain a greater minorant / smaller majorant by reducing this distance. The equalities to the convex minorant / concave majorant are then obtained by exploiting the inequalities at the beginning of the proof.

By the already presented inequalities $\cup(f_1 + \lambda_1) \geq \underline{a}(f_1 + \lambda_1)$, but let us suppose for a moment $\cup(f_1 + \lambda_1) > \underline{a}(f_1 + \lambda_1)$. This would imply $\delta_1^{\cup} > \underline{\delta}$. We then would have that at the touching point x

$$f_x + \lambda_x = \underline{a}(f_x + \lambda_x) = x\underline{\delta} < x\delta_1^{\cup} \leq \cup(f_1 + \lambda_1),$$

as the slopes in a convex minorant must be monotonically increasing. However, such function would not be a valid convex minorant, as it would grow over $f + \lambda$. Therefore $\cup(f_1 + \lambda_1) = \underline{a}(f_1 + \lambda_1)$ must hold. Using a symmetric argument, $\cap(f_1 - \lambda_1) = \bar{a}(f_1 - \lambda_1)$ can also be shown to hold. Joining this with the previous facts we have that

$$\cup(f_1 + \lambda_1) = \underline{a}(f_1 + \lambda_1) = \underline{\delta} \leq \bar{\delta} = \bar{a}(f_1 - \lambda_1) = \cap(f_1 - \lambda_1),$$

and therefore the overlap detected by the affine approximation is detected through its convex/concave version as well through $\cup(f_1 + \lambda_1) \leq \cap(f_1 - \lambda_1)$.

■

Appendix E. Projected-Newton for Weighted $\mathbf{Tv}_1^{\mathbf{1D}}$

In this appendix we present details of a projected-Newton (PN) approach to solving the weighted-TV problem (2.6). Although taut-string approaches are empirically superior to this PN approach, the details of this derivation prove to be useful when developing subroutines for handling ℓ_p-norm TV prox-operators, but perhaps their greatest use lies in presenting a general method that could be applied to other problems that have structures similar to TV, e.g., group total-variation (Alaız et al., 2013; Wytock et al., 2014) and ℓ_1-trend filtering (Kim et al., 2009; Tibshirani, 2014).

The weighted-TV dual problem (2.7) is a bound-constrained QP, so it could be solved using a variety of methods such as TRON (Lin and Moré, 1999), L-BFGS-B (Byrd et al., 1994), or projected-Newton (PN) (Bertsekas, 1982). Obviously, these methods will be inefficient if invoked off-the-shelf; exploitation of problem structure is a must for solving (2.7) efficiently. PN lends itself well to such structure exploitation; we describe the details below.

PN runs iteratively in three key steps: first it identifies a special subset of *active variables* and uses these to compute a *reduced* Hessian. Then, it uses this Hessian to scale the gradient and move in the direction opposite to it, damping with a stepsize, if needed. Finally, the next iterate is obtained by projecting onto the constraints, and the cycle repeats. PN can be regarded as an extension of the gradient-projection method (GP, Bertsekas (1999)), where the components of the gradient that make the updating direction infeasible are removed; in PN both the gradient and the Hessian are *reduced* to guarantee this feasibility.

At each iteration PN selects the active variables

$$I := \{i \mid (u_i = -w_i \text{ and } [\nabla\phi(\boldsymbol{u})]_i > \epsilon) \quad \text{or} \quad (u_i = w_i \text{ and } [\nabla\phi(\boldsymbol{u})]_i < -\epsilon)\}, \tag{E.1}$$

where $\epsilon \geq 0$ is small scalar. This corresponds to the set of variables at a bound, and for which the gradient points inside the feasible region; that is, for these variables to further improve the objective function we would have to step out of bounds. It is thus clear that these variables are of no use for this iteration, so we define the complementary set $\bar{I} := \{1 \dots n\} \setminus I$ of indices not in I, which are the variables we are interested in updating. From the Hessian $\boldsymbol{H} = \nabla^2\phi(u)$ we extract the *reduced Hessian* $\boldsymbol{H}_{\bar{I}}$ by selecting rows and columns indexed by $\bar{I}$, and in a similar way the *reduce gradient* $[\nabla\phi(\boldsymbol{u})]_{\bar{I}}$. Using these we perform a Newton–like "reduced" update in the form

$$\boldsymbol{u}_{\bar{I}} \leftarrow P(\boldsymbol{u}_{\bar{I}} - \alpha \boldsymbol{H}_{\bar{I}}^{-1}[\nabla\phi(\boldsymbol{u})]_{\bar{I}}), \tag{E.2}$$

where α is a stepsize, and P denotes projection onto the constraints, which for box–constraints reduces to simple element–wise projection. Note that only the variables in the set $\bar{I}$ are updated in this iterate, leaving the rest unchanged. While such update requires computing the inverse of the reduced Hessian $\boldsymbol{H}_{\bar{I}}$, which in the general case can amount to computational costs in the $O(n^3)$ order, we will see now how exploiting the structure of the problem allows us to perform all the steps above efficiently.

First, observe that for (2.7) the Hessian is

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

Next, observe that whatever the active set I, the corresponding reduced Hessian $\boldsymbol{H}_{\bar{I}}$ remains symmetric tridiagonal. This observation is crucial because then we can quickly compute the updating direction $\boldsymbol{d}_{\bar{I}} = \boldsymbol{H}_{\bar{I}}^{-1}[\nabla\phi(\boldsymbol{u})]_{\bar{I}}$, which can be done by solving the linear system $\boldsymbol{H}_{\bar{I}}\boldsymbol{d}_{\bar{I}} = [\nabla\phi(\boldsymbol{u}^t)]_{\bar{I}}$ as follows:

1. Compute the Cholesky decomposition $\boldsymbol{H}_{\bar{I}} = \boldsymbol{R}^T\boldsymbol{R}$.
2. Solve the linear system $\boldsymbol{R}^T\boldsymbol{v} = [\nabla\phi(\boldsymbol{u})]_{\bar{I}}$ to obtain $\boldsymbol{v}$.
3. Solve the linear system $\boldsymbol{R}\boldsymbol{d}_{\bar{I}} = \boldsymbol{v}$ to obtain $\boldsymbol{d}_{\bar{I}}$.

Because the reduced Hessian is also tridiagonal, its Cholesky decomposition can be computed in *linear time* to yield a bidiagonal matrix $\boldsymbol{R}$, which in turn allows to solve the subsequent linear systems also in linear time. Extremely efficient routines to perform all these tasks are available in the LAPACK libraries (Anderson et al., 1999).

The next crucial ingredient is efficient selection of the stepsize α. The original PN algorithm Bertsekas (1982) recommends Armijo-search along projection arc. However, for our problem this search is inordinately expensive. So we resort to a backtracking strategy using quadratic interpolation (Nocedal and Wright, 2000), which works admirably well. This strategy is as follows: start with an initial stepsize $\alpha_0 = 1$. If the current stepsize α_k does not provide sufficient decrease in ϕ, build a quadratic model using $\phi(\boldsymbol{u})$, $\phi(\boldsymbol{u} - \alpha_k\boldsymbol{d})$, and $\partial_{\alpha_k}\phi(\boldsymbol{u})$. Then, the stepsize α_{k+1} is set to the value that minimizes this quadratic model. In the event that at some point of the procedure the new α_{k+1} is larger than or too similar to α_k, its value is halved. In this fashion, quadratic approximations of ϕ are iterated until a good enough α is found. The goodness of a stepsize is measured using the following Armijo-like sufficient descent rule

$$\phi(\boldsymbol{u}) - \phi(P\left[\boldsymbol{u} - \alpha_k\boldsymbol{d}\right]) \geq \sigma \cdot \alpha_k \cdot (\nabla\phi(\boldsymbol{u}) \cdot \boldsymbol{d}),$$

where a tolerance $\sigma = 0.05$ works well practice.

Note that the gradient $\nabla\phi(\boldsymbol{u})$ might be misleading in the condition above if $\boldsymbol{u}$ has components at the boundary and $\boldsymbol{d}$ points outside this boundary (because then, due to the subsequent projection no real improvement would be obtained by stepping outside the feasible region). To address this concern, we modify the computation of the gradient $\nabla\phi(\boldsymbol{u})$, zeroing our the entries that relate to direction components pointing outside the feasible set.

The whole stepsize selection procedure is shown in Algorithm 11. The costliest operation in this procedure is the evaluation of ϕ, which, nevertheless can be done in linear time. Furthermore, in practice a few iterations more than suffice to obtain a good stepsize.

Overall, a full PN iteration as described above runs at $O(n)$ cost. Thus, by exploiting the structure of the problem, we manage to reduce the $O(n^3)$ cost per iteration of a general

Algorithm 11 Stepsize selection for Projected Newton

Initialize: $\alpha_0 = 1$, $k = 0$, $\boldsymbol{d}$, tolerance parameter σ
while $\phi(\boldsymbol{u}) - \phi(P[\boldsymbol{u} - \alpha_k \boldsymbol{d}]) < \sigma \cdot \alpha_k \cdot (\nabla\phi(\boldsymbol{u}) \cdot \boldsymbol{d})$ **do**
  Minimize quadratic model: $\alpha_{k+1} = \frac{\alpha_k^2 \partial_{\alpha_k}\phi(\boldsymbol{u})}{2(\phi(\boldsymbol{u}) - \phi(\boldsymbol{u} - \alpha_k) + \alpha_k \partial_{\alpha_k}\phi(\boldsymbol{u}))}$.
  if $\alpha_{k+1} > \alpha_k$ **or** $\alpha_{k+1} \simeq \alpha_k$, **then** $\alpha_{k+1} = \frac{1}{2}\alpha_k$.
  $k \leftarrow k + 1$
end while
return α_k

Algorithm 12 PN algorithm for TV-L1-proximity

Let $\boldsymbol{W} = \mathrm{Diag}(w_i)$; solve $\boldsymbol{D}\boldsymbol{D}^T\boldsymbol{W}\boldsymbol{u}^* = \boldsymbol{D}\boldsymbol{y}$.
if $\|\boldsymbol{W}^{-1}\boldsymbol{u}^*\|_\infty \leq 1$, **return** $\boldsymbol{u}^*$.
$\boldsymbol{u}^0 = P[\boldsymbol{u}^*]$, $t = 0$.
while $\mathrm{gap}(\boldsymbol{u}) > \epsilon$ **do**
  Identify set of active constraints I; let $\bar{I} = \{1 \dots n\} \setminus I$.
  Construct reduced Hessian $\boldsymbol{H}_{\bar{I}}$.
  Solve $\boldsymbol{H}_{\bar{I}}\boldsymbol{d}_{\bar{I}} = [\nabla\phi(\boldsymbol{u}^t)]_{\bar{I}}$.
  Compute stepsize α using backtracking + interpolation (Alg. 11).
  Update $\boldsymbol{u}_{\bar{I}}^{t+1} = P[\boldsymbol{u}_{\bar{I}}^t - \alpha\boldsymbol{d}_{\bar{I}}]$.
  $t \leftarrow t + 1$.
end while
return $\boldsymbol{u}^t$.

PN algorithm to a linear-cost method. The pseudocode of the resulting method is shown as Algorithm 12. Note that in the special case when the weights $\boldsymbol{W} := \mathrm{Diag}(w_i)$ are so large that the unconstrained optimum coincides with the constrained one, we can obtain $\boldsymbol{u}^*$ directly via solving $\boldsymbol{D}\boldsymbol{D}^T\boldsymbol{W}\boldsymbol{u}^* = \boldsymbol{D}\boldsymbol{y}$ (which can also be done at $O(n)$ cost). The duality gap of the current solution is used as a stopping criterion, where we use a tolerance of $\epsilon = 10^{-5}$ in practice.

Appendix F. Testing Images and Videos, and Experimental Results

The images used in the experiments are displayed in what follows, along with their noisy/denoised and convoluted/deconvoluted versions for each algorithm tested. QR barcode images were generated by encoding random text using Google chart API[4]. Images *shape* and *phantom* [5] are publicly available and frequently used in image processing. *trollface* and *comic* [6] are also publicly available. *gaudi*, used in the multicore experiments, is a high resolution 3197×3361 photograph of Gaudi's Casa Batlló[7]. The rest of the images were originally created by the authors.

4. `http://code.google.com/intl/en-EN/apis/chart/`
5. Extracted from `http://en.wikipedia.org/wiki/File:Shepp_logan.png`
6. Author: Francisco Molina. `http://www.afrikislife.net/english/`
7. Extracted from `http://www.flickr.com/photos/jeffschwartz/202423023/`

For the video experiments, the *salesman*, *coastguard* and *bicycle* sequences were used, which are publicly available at BM3D (2013). As an example, frames from the first video are displayed in what follows, along with their noisy/denoised versions.

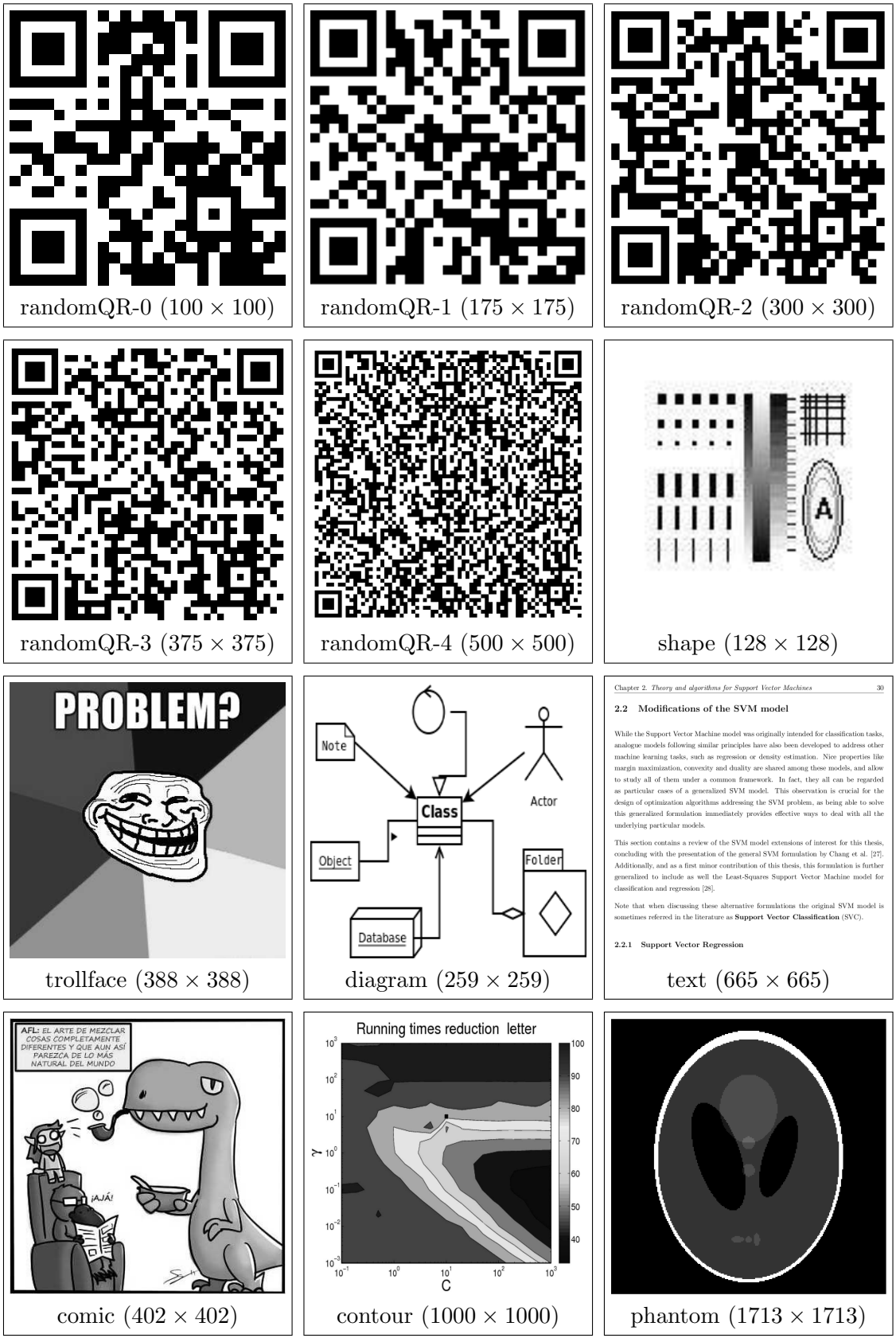


Figure 25: Test images used in the experiments together with their sizes in pixels. Images displayed have been scaled down to fit in page.

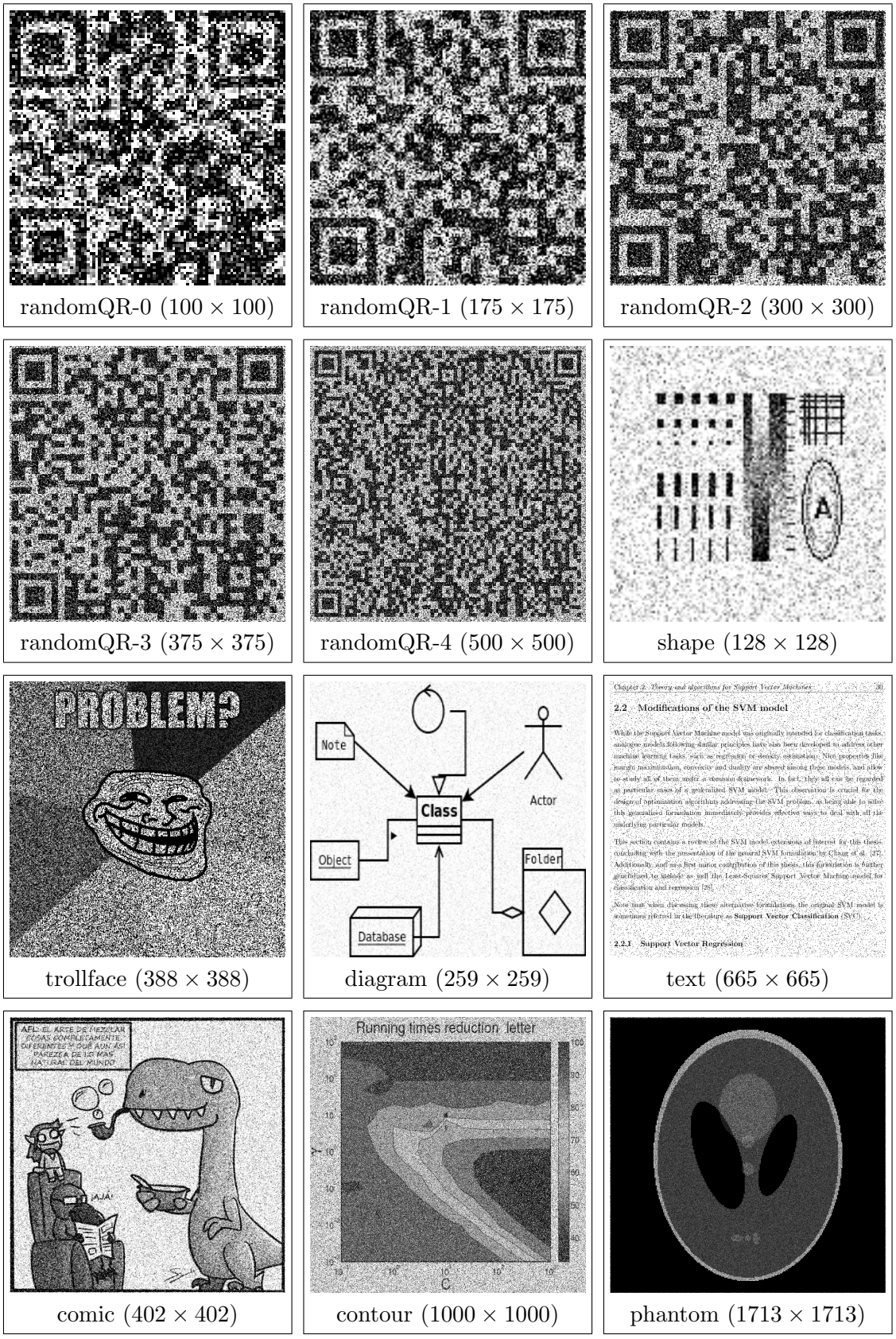
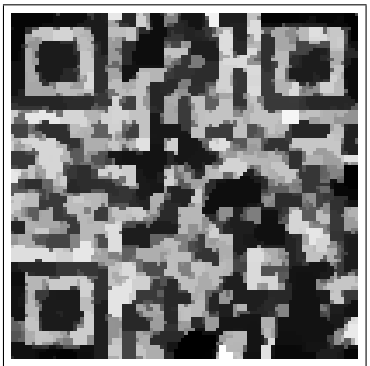
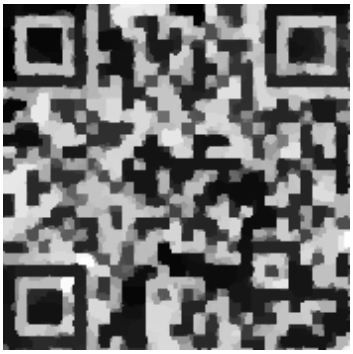


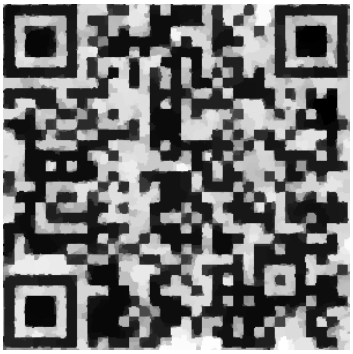
Figure 26: Noisy versions of images used in the experiments.



randomQR-0 (100 × 100)



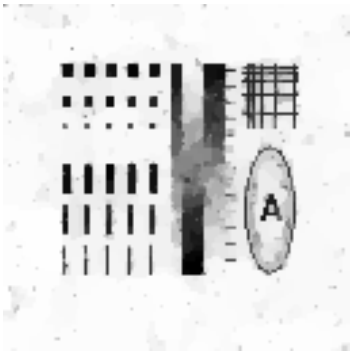
randomQR-1 (175 × 175)



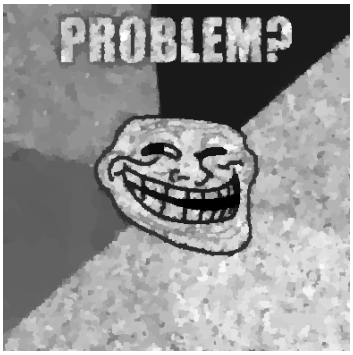
randomQR-2 (300 × 300)

randomQR-3 (375 × 375)

randomQR-4 (500 × 500)



shape (128 × 128)



trollface (388 × 388)

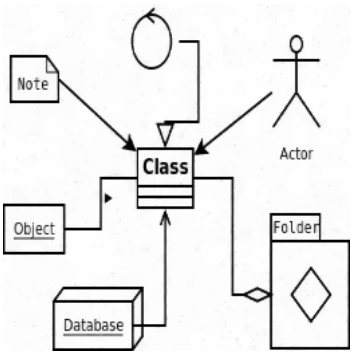
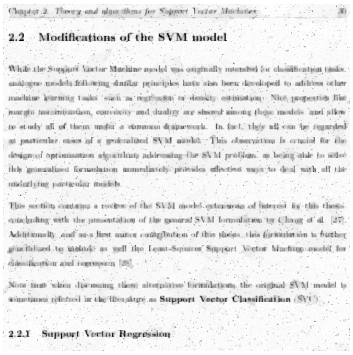
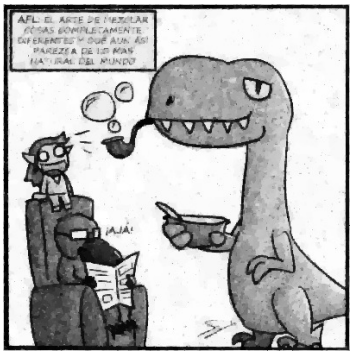


diagram (259 × 259)

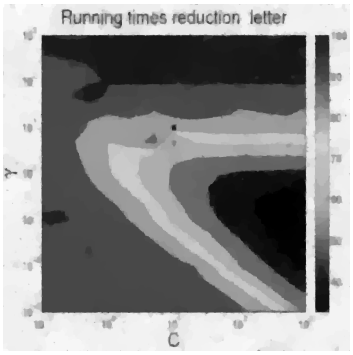


2.2 Modifications of the SVM model

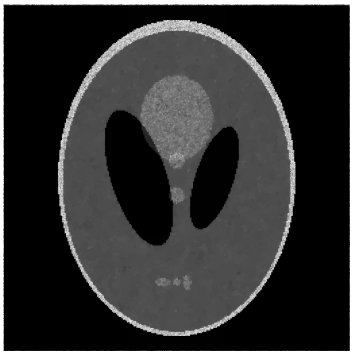
text (665 × 665)



comic (402 × 402)

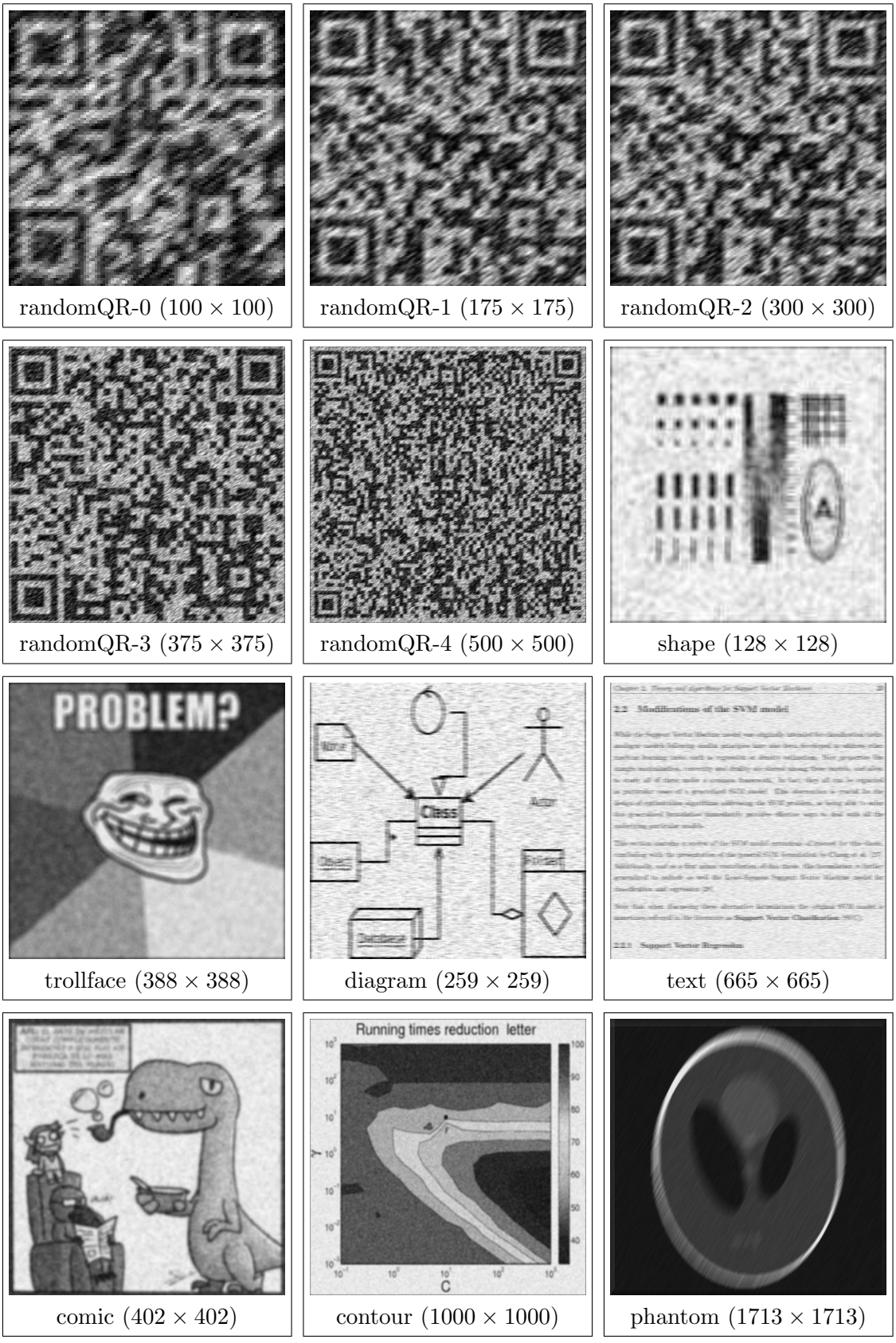


contour (1000 × 1000)



phantom (1713 × 1713)

Figure 27: Denoising results for the test images.



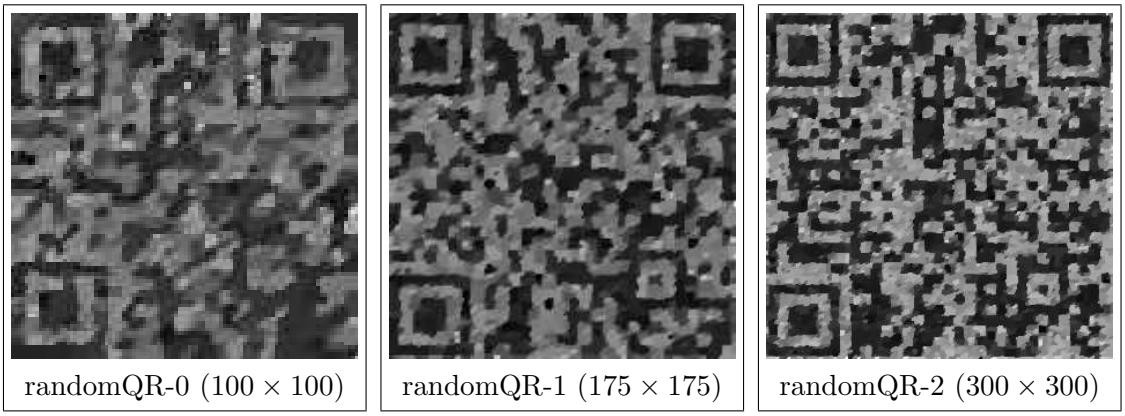
randomQR-0 (100×100) | randomQR-1 (175×175) | randomQR-2 (300×300)

randomQR-3 (375×375) | randomQR-4 (500×500) | shape (128×128)

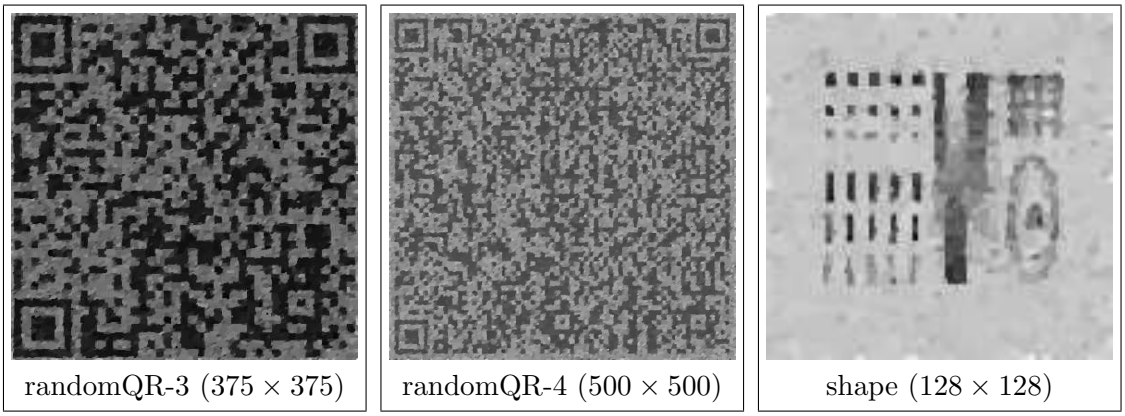
trollface (388×388) | diagram (259×259) | text (665×665)

comic (402×402) | contour (1000×1000) | phantom (1713×1713)

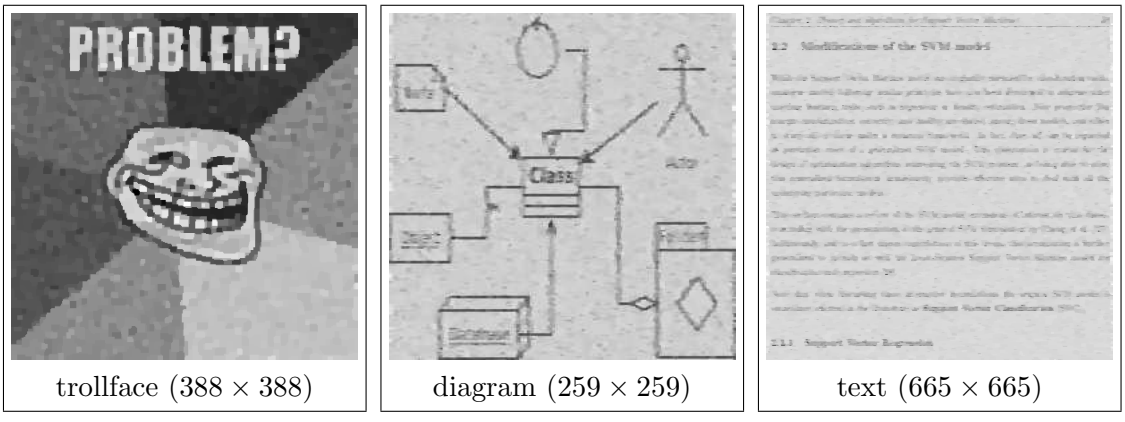
Figure 28: Noisy and convoluted versions of images used in the experiments.



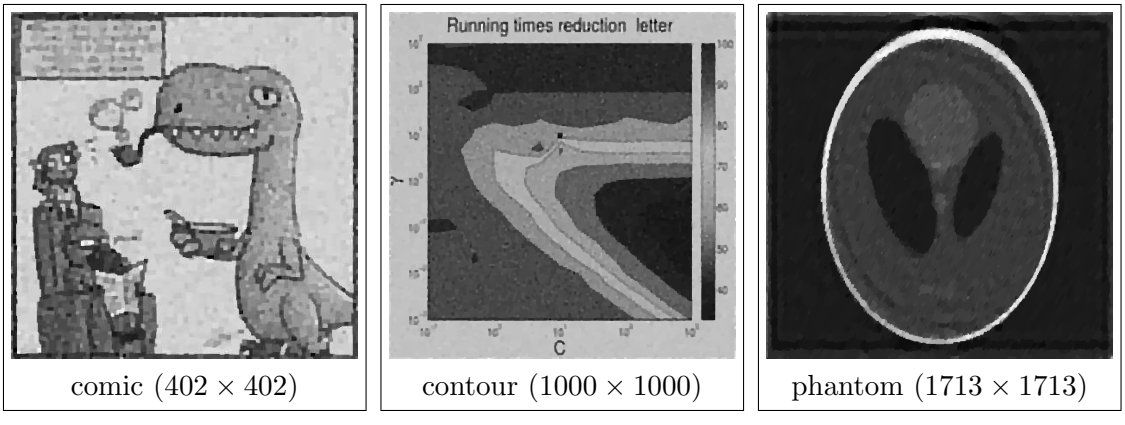
randomQR-0 (100 × 100) randomQR-1 (175 × 175) randomQR-2 (300 × 300)



randomQR-3 (375 × 375) randomQR-4 (500 × 500) shape (128 × 128)



trollface (388 × 388) diagram (259 × 259) text (665 × 665)



comic (402 × 402) contour (1000 × 1000) phantom (1713 × 1713)

Figure 29: Deconvolution results for the test images.

Figure 30: A selection of frames from the *salesman* video sequence.

Figure 31: Noisy frames from the *salesman* video sequence.

Figure 32: Denoised frames from the *salesman* video sequence.

References

M. V. Afonso, J. M. Bioucas-Dias, and M. A. T. Figueiredo. Fast image recovery using variable splitting and constrained optimization. *IEEE Transactions on Image Processing*, 19(9), September 2010.

C. M. Alaız, Á. Barbero, and J. R. Dorronsoro. Group fused lasso. *Artificial Neural Networks and Machine Learning–ICANN 2013*, page 66, 2013.

E. Anderson et al. *LAPACK Users' Guide*. Society for Industrial and Applied Mathematics, Philadelphia, PA, third edition, 1999. ISBN 0-89871-447-8 (paperback).

F. Bach. Structured sparsity-inducing norms through submodular functions. In *NIPS*, 2010.

Bach, Francis Learning with Submodular Functions: A Convex Optimization Perspective *arXiv preprint arXiv:1111.6453*

F. Bach, R. Jenatton, J. Mairal, and G. Obozinski. Convex optimization with sparsity-inducing norms. In S. Sra, S. Nowozin, and S. J. Wright, editors, *Optimization for Machine Learning*. MIT Press, 2011.

Á. Barbero, J. López, and J. R. Dorronsoro. Finding Optimal Model Parameters by Discrete Grid Search. In *Advances in Soft Computing: Innovations in Hybrid Intelligent Systems 44*, pages 120–127. Springer, 2008.

Barbero, A., Sra, S. Fast Newton-type methods for total variation regularization. *In Proceedings of the 28th International Conference on Machine Learning (ICML-11) (pp. 313-320).*

Á. Barbero, J. López, and J. R. Dorronsoro. Finding Optimal Model Parameters by Deterministic and Annealed Focused Grid Search. *Neurocomputing*, 72(13-15):2824–2832, 2009. ISSN 0925-2312. doi: DOI:10.1016/j.neucom.2008.09.024.

Barlow, R. E., Bartholomew, D. J., Bremner, J. M., Brunk, H. D. Statistical inference under order restrictions: The theory and application of isotonic regression *New York: Wiley, 1972*

H. H. Bauschke and P. L. Combettes. *Convex analysis and monotone operator theory in Hilbert spaces*. CMS Books in Mathematics. Springer, 2011.

Heinz H. Bauschke, Patrick L. Combettes, D. Russell Luke Finding best approximation pairs relative to two closed convex sets in Hilbert spaces *Journal of Approximation Theory 127 (2004) 178–192*

A. Beck and M. Teboulle. A Fast Iterative Shrinkage-Thresholding Algorithm for Linear Inverse Problems. *SIAM Journal of Imgaging Sciences*, 2(1):183–202, 2009.

D. P. Bertsekas. Projected newton methods for optimization problems with simple constraints. *SIAM Journal on Control and Optimization*, 20(2), March 1982.

D. P. Bertsekas. *Nonlinear Programming*. Athena Scientific, 2nd edition, September 1999.

J. M. Bioucas-Dias and M. A. T. Figueiredo. A new twist: Two-step iterative shrinkage/thresholding algorithms for image restoration. *IEEE Transactions on Image Processing*, 16(12):2992–3004, December 2007.

J. M. Bioucas-Dias, M. A. T. Figueiredo, and J. P. Oliveira. Total variation-based image deconvolution: A majorization-minimization approach. In *ICASSP Proceedings*, 2006.

BM3D. Bm3d software and test sequences, 2013. URL `http://www.cs.tut.fi/~foi/GCF-BM3D/`.

R. H. Byrd, P. Lu, J. Nocedal, and C. Zhu. A limited memory algorithm for bound constrained optimization. Technical report, Northwestern University, 1994.

E. J. Candès and T. Tao. Near-optimal signal recovery from random projections: universal encoding strategies. *IEEE Trans. Info. Theory*, 52:5406–5425, 2004.

A. Chambolle and J. Darbon. On total variation minimization and surface evolution using parametric maximum flows. *International Journal of Computer Vision*, 84(3), 2009.

A. Chambolle and T. Pock. A first-order primal-dual algorithm for convex problems with applications to imaging. *Journal of Mathematical Imaging and Vision*, 40(1):120–145, 2011.

Chambolle, A., Pock, T. On the ergodic convergence rates of a first-order primal-dual algorithm *Mathematical Programming. September 2016, Volume 159, Issue 1, pp 253–287*

Chambolle, A., Pock, T. A remark on accelerated block coordinate descent for computing the proximity operators of a sum of convex functions. *SMAI Journal of Computational Mathematics*, 129-54

V. Chandrasekaran, B. Recht, P. A. Parrilo, and A. S. Willsky. The Convex Geometry of Linear Inverse Problems. *Foundations of Computational Mathematics*, 12(6), 2012.

R. Choksi, Y. van Gennip, and A. Oberman. Anisotropic Total Variation Regularized L1-Approximation and Denoising/Deblurring of 2D Bar Codes. Technical report, McGill University, July 2010.

P. L. Combettes. Iterative construction of the resolvent of a sum of maximal monotone operators. *Journal of Convex Analysis*, 16:727–748, 2009.

P. L. Combettes and J.-C. Pesquet. Proximal splitting methods in signal processing. *arXiv:0912.3522*, 2009.

L. Condat. A direct algorithm for 1d total variation denoising. Technical report, GREYC laboratory, CNRS-ENSICAEN-Univ. of Caen, 2012.

L. Condat. A generic proximal algorithm for convex optimization - application to total variation minimization. *IEEE SIGNAL PROC. LETTERS*, 21(8):985–989, 2014.

L. Condat. Fast projection onto the simplex and the ℓ_1 ball. *Mathematical Programming*, 158(1-2):575-585, 2016.

A. R. Conn, N. I. M. Gould, and P. L. Toint. *Trust-Region Methods*. SIAM, 2000.

J. Dahl, P. C. Hansen, S. H. Jensen, and T. L. Jensen. Algorithms and software for total variation image reconstruction via first-order methods. *Numer Algor*, 53:67–92, 2010.

P. L. Davies and A. Kovac. Local extremes, runs, strings and multiresolution. *The Annals of Statistics*, 29(1):1–65, 2001.

Y. Duan and X.-C. Tai. Domain decomposition methods with graph cuts algorithms for total variation minimization. *Adv Comput Math*, 36:175–199, 2012. doi: 10.1007/s10444-011-9213-4.

Esedoglu, Selim and Osher, Stanley J. Decomposition of images by the anisotropic Rudin-Osher-Fatemi model *Communications on Pure and Applied Mathematics: A Journal Issued by the Courant Institute of Mathematical Sciences*, 77:(12): 1609–1626, 2014.

J. Friedman, T. Hastie, H. Höfling, and R. Tibshirani. Pathwise coordinate optimization. *Annals of Applied Statistics*, 1(2):302–332, Aug. 2007.

D. Goldfarb and W. Yin. Parametric maximum flow algorithms for fast total variation minimization. *SIAM Journal on Scientific Computing*, 31(5):3712–3743, 2009.

O. S. Goldstein T. The Split Bregman Method for L1 Regularized Problems. *SIAM Journal on Imaging Sciences*, 2(2):323–343, 2009.

T. R. Golub et al. Molecular classification of cancer. *Science*, 286(5439):531–537, October 1999.

M. Grasmair. The equivalence of the taut string algorithm and bv-regularization. *Journal of Mathematical Imaging and Vision*, 27(1):59–66, 2007. ISSN 0924-9907. doi: 10.1007/s10851-006-9796-4. URL `http://dx.doi.org/10.1007/s10851-006-9796-4`.

Z. Harchaoui and C. Lévy-Leduc. Multiple Change-Point Estimation With a Total Variation Penalty. *Journal of the American Statistical Association*, 105(492):1480–1493, 2010.

J. Hua, W. D. Tembe, and E. R. Dougherty. Performance of feature-selection methods in the classification of high-dimension data. *Pattern Recognition*, 42:409–424, 2009.

K. Ito and K. Kunisch. An active set strategy based on the augmented lagrangian formulation for image restoration. *ESAIM: Mathematical Modelling and Numerical Analysis*, 33 (1):1–21, 1999. URL `http://eudml.org/doc/193911`.

M. Jaggi. Revisiting frank-wolfe: Projection-free sparse convex optimization. In *Proceedings of the 30th International Conference on Machine Learning,*, 2013.

S. Jegelka, F. Bach, and S. Sra. Reflection methods for user-friendly submodular optimization. *Advances in Neural Information Processing Systems* 2013: 1313–1321.

Jegou, H., Douze, M., Schmid, C. Hamming Embedding and Weak geometry consistency for large scale image search *Proceedings of the 10th European conference on Computer vision, October, 2008* `http://lear.inrialpes.fr/~jegou/data.php#holidays`

N. A. Johnson. A dynamic programming algorithm for the fused Lasso and l_0-segmentation. *J. Computational and Graphical Statistics*, 2013.

D. Kim, S. Sra, and I. Dhillon. A scalable trust-region algorithm with application to mixed-norm regression. In *International Conference on Machine Learning*, 2010.

S. Kim, K. Koh, S. Boyd, and D. Gorinevsky. ℓ_1 trend filtering. *SIAM Review*, 51(2): 339–360, 2009. doi: 10.1137/070690274.

K. C. Kiwiel. Variable fixing algorithms for the continuous quadratic knapsack problem. *J. Optim. Theory Appl.*, 136:445–458, 2008.

Knuth, Donald E. The art of computer programming, volume 1: fundamental algorithms. *CA, USA: Addison Wesley Longman Publishing Co., Inc*

M. Kolar, L. Song, A. Ahmed, and E. Xing. Estimaging time-varying networks. *The Annals of Applied Statistics*, 4(1):94–123, 2010.

Kolmogorov, V., Pock, T., Rolinek, M. Total variation on a tree *SIAM J. Imaging Sci., 9(2), 605–636.*

D. Krishnan and R. Fergus. Fast image deconvolution using hyper-laplacian priors. In *Advances in Neural Information Processing Systems*, 2009.

Kumar, K.S., Barbero, A., Jegelka, S., Sra, S., and Bach, F. Convex optimization for parallel energy minimization. *arXiv preprint arXiv:1503.01563.*

S. R. Land and J. H. Friedman. Variable fusion: A new adaptive signal regression method. Technical Report 656, Department of Statistics, Carnegie Mellon University Pittsburgh,

1997.

Y. Li and F. Santosa. A computational algorithm for minimizing total variation in image restoration. *IEEE Transactions on Image Processing*, 5(6):987–995, 1996. URL `http://dblp.uni-trier.de/db/journals/tip/tip5.html#LiS96`.

C.-J. Lin and J. J. Moré. Newton's method for large bound-constrained optimization problems. *SIAM Journal on Optimization*, 9(4):1100–1127, 1999.

H. Liu and J. Zhang. Estimation Consistency of the Group Lasso and its Applications. In *Int. Conf. Mach. Learning (ICML)*, 2009.

J. Liu and J. Ye. Efficient Euclidean projections in linear time. In *ICML*, Jun. 2009.

J. Liu, S. Ji, and J. Ye. *SLEP: Sparse Learning with Efficient Projections*. Arizona State University, 2009. *http://www.public.asu.edu/ jye02/Software/SLEP*.

J. Liu, L. Yuan, and J. Ye. An efficient algorithm for a class of fused lasso problems. In *ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, 2010.

J. Mairal, R. Jenatton, G. Obozinski, and F. Bach. Network Flow Algorithms for Structured Sparsity. In *NIPS*, 2010. To appear.

B. Martinet. Régularisation d'inéquations variationnelles par approximations successives. *Modélisation Mathématique et Analyse Numérique*, 4(R3):154–158, 1970.

L. Meier, S. van de Geer, and P. Bühlmann. The group lasso for logistic regression. *J. R. Statist. Soc.*, 70:53–71, 2008.

J. J. Moré and D. C. Sorensen. Computing a trust region step. *SIAM Journal of Scientific Computing*, 4(3), September 1983.

J. J. Moreau. Fonctions convexes duales et points proximaux dans un espace hilbertien. *C. R. Acad. Sci. Paris Sér. A Math.*, 255:2897–2899, 1962.

Y. Nesterov. Gradient methods for minimizing composite objective function. Technical Report 76, Catholic University of Louvain, CORE, 2007.

J. Nocedal and S. J. Wright. *Numerical Optimization*. Springer Verlag, 2000.

N. Parikh, S. Boyd, et al. Proximal algorithms. *Foundations and Trends® in Optimization*, 1(3):127–239, 2014.

G. Pierra. Decomposition through formalization in a product space. *Mathematical Programming*, 28(1):96–115, 1984.

C. Pontow and O. Scherzer. A derivative free approach for total variation regularization. *arXiv:0911.1293*, 2009. URL `http://arxiv.org/abs/0911.1293`.

A. Ramdas and R. J. Tibshirani. Fast and flexible admm algorithms for trend filtering. *arXiv:1406.2082*, 2014.

F. Rapaport and E. B. J.-P. Vert. Classification of arrayCGH data using fused SVM. *Bioinformatics*, 24(13):i375–i382, 2008.

A. Rinaldo. Properties and refinements of the fused lasso. *Annals of Statistics*, 37(5B): 2922–2952, 2009.

R. T. Rockafellar. Monotone operators and hte proximal point algorithm. *SIAM J. Control and Opt.*, 14(5):877–898, 1976.

S. Rogers, M. Girolami, C. Campbell, and R. Breitling. The latent process decomposition of cdna microarray data sets. *IEEE/ACM Trans. Comp. Bio. and Bioinformatics*, 2(2), April-June 2005.

L. I. Rudin, S. Osher, and E. Fatemi. Nonlinear total variation based noise removal algorithms. *Physica D*, 60:259–268, 1992.

Russakovsky, O., Deng, J., Su, H., Krause, J., Satheesh, S., Ma, S., Huang, Z., Karpathy A., Khosla, A., Bernstein, M., Berg, A.C., Fei-Fei, L. ImageNet Large Scale Visual Recognition Challenge *International Journal of Computer Vision (IJCV), Year 2015, Volume 115, Number 3, pages 211-252* `http://image-net.org/challenges/LSVRC/2010/download-public`

S. Salzo and S. Villa. Inexact and accelerated proximal point algorithms. *J. Convex Analysis*, 19(4), 2012.

M. Schmidt, N. L. Roux, and F. Bach. Convergence Rates of Inexact Proximal-Gradient Methods for Convex Optimization. In *Advances in Neural Information Processing Systems (NIPS)*, 2011.

S. Sra. Scalable nonconvex inexact proximal splitting. In *Advances in Neural Information Processing Systems*, 2012.

S. Sra, S. Nowozin, and S. Wright, editors. *Optimization for machine learning*. MIT Press, 2011.

G. Steidl, S. Didas, and J. Neumann. Relations between higher order tv regularization and support vector regression. In *Scale-Space*, pages 515–527, 2005.

G. Steidl and T. Teuber Anisotropic smoothing using double orientations. In *International Conference on Scale Space and Variational Methods in Computer Vision*, pages 477–489, 2009. Springer, Berlin, Heidelberg.

N. Stransky et al. Regional copy number-independent deregulation of transcription in cancer. *Nature Genetics*, 38(12):1386–1396, December 2006.

R. Tibshirani. Regression shrinkage and selection via the lasso. *J. R. Statist. Soc.*, 58(1): 267–288, 1996.

R. Tibshirani and P. Wang. Spatial smoothing and hot spot detection for CGH data using the fused lasso. *Biostatistics*, 9(1):18–29, 2008.

R. Tibshirani, M. Saunders, S. Rosset, J. Zhu, and K. Knight. Sparsity and smoothness via the fused lasso. *J. Royal Stat. Soc.: Series B*, 67(1):91–108, 2005.

R. J. Tibshirani. Adaptive piecewise polynomial estimation via trend filtering. *The Annals of Statistics*, 42(1):285–323, 02 2014. doi: 10.1214/13-AOS1189.

U. Alon et al. Broad patterns of gene expression revealed by clustering analysis of tumor and normal colon tissues probed by oligonucleotide arrays. *Proc. Natl. Acad. Sci. USA*, 96:6745–6750, June 1999.

J.-P. Vert and K. Bleakley. Fast detection of multiple change-points shared by many signals using group LARS. In *Advances in Neural Information Processing Systems*, 2010.

C. R. Vogel and M. E. Oman. Iterative methods for total variation denoising. *SIAM Journal on Scientific Computing*, 17(1):227–238, 1996.

B. Wahlberg, S. Boyd, M. Annergren, and Y. Wang. An ADMM Algorithm for a Class of Total Variation Regularized Estimation Problems. In *Proceedings 16th IFAC Symposium on System Identification*, volume 16, 2012.

J. Wang and Q. Li and S. Yang and W. Fan and P. Wonka and J. Ye. A Highly Scalable Parallel Algorithm for Isotropic Total Variation Models In *Proceedings of the 31st International Conference on Machine Learning (ICML-14)*, pages 235-243, 2014.

S. J. Wright, R. D. Nowak, and M. A. T. Figueiredo. Sparse reconstruction by separable approximation. *IEEE Trans. Sig. Proc.*, 57(7):2479–2493, 2009.

M. Wytock, S. Sra, and J. Z. Kolter. Fast Newton Methods for the Group Fused Lasso. In *Conference on Uncertainty in Artificial Intelligence*, 2014.

S. Yang, J. Wang, W. Fan, X. Zhang, P. Wonka, and J. Ye. An Efficient ADMM Algorithm for Multidimensional Anisotropic Total Variation Regularization Problems. In *ACM Knowledge Discovery and Data Mining (KDD)*, Chicago, Illinois, USA, August 2013.

Y. Yu. On decomposing the proximal map. In *Advances in Neural Information Processing Systems*, 2013.

M. Yuan and Y. Lin. Model Selection and Estimation in Regression with Grouped Variables. *J. R. Statist. Soc. B*, 68(1):49–67, 2006.

M. Zhu and T. Chan. An efficient primal-dual hybrid gradient algorithm for total variation image restoration. Technical report, UCLA CAM, 2008.