Proximal gradient algorithms: Applications in signal processing

Niccolò Antonello[a,b,*], Lorenzo Stella[c], Panagiotis Patrinos[a],
Toon van Waterschoot[a,b]

[a] KU Leuven, ESAT–STADIUS, Stadius Center for Dynamical Systems, Signal Processing and Data Analytics, Kasteelpark Arenberg 10, 3001 Leuven, Belgium
[b] KU Leuven, ESAT–ETC, e-Media Research Lab, Andreas Vesaliusstraat 13, 1000 Leuven, Belgium
[c] Amazon Research, Krausenstraße 38, 10117 Berlin, Germany

Abstract

Advances in numerical optimization have supported breakthroughs in several areas of signal processing. This paper focuses on the recent enhanced variants of the proximal gradient numerical optimization algorithm, which combine quasi-Newton methods with forward-adjoint oracles to tackle large-scale problems and reduce the computational burden of many applications. These proximal gradient algorithms are here described in an easy-to-understand way, illustrating how they are able to address a wide variety of problems arising in signal processing. A new high-level modeling language is presented which is used to demonstrate the versatility of the presented algorithms in a series of signal processing application examples such as sparse deconvolution, total variation denoising, audio de-clipping and others.

Keywords: Numerical optimization; Proximal gradient algorithm; Large-scale optimization

[☆] This research work was carried out at the ESAT Laboratory of KU Leuven, the frame of the FP7-PEOPLE Marie Curie Initial Training Network “Dereverberation and Reverberation of Audio, Music, and Speech (DREAMS)”, funded by the European Commission under Grant Agreement no. 316969, European Research Council under the European Union’s Horizon 2020 research and innovation program / ERC Consolidator Grant: SONORA no. 773268, KU Leuven Research Council CoE PFV/10/002 (OPTEC), KU Leuven Impulsfonds IMP/14/037, KU Leuven Internal Funds VES/16/032 and StG/15/043, KU Leuven C2-16-00449 “Distributed Digital Signal Processing for Ad-hoc Wireless Local Area Audio Networking”, FWO projects G086318N and G086518N, and Fonds de la Recherche Scientifique – FNRS and the Fonds Wetenschappelijk Onderzoek – Vlaanderen under EOS Project no 30468160 (SeLMA). The scientific responsibility is assumed by its authors.

[☆☆] The work of Lorenzo Stella was conducted prior to joining Amazon Research.

[*] Corresponding author

Email address: `niccolo.antonello@esat.kuleuven.be` (Niccolò Antonello)

1. Introduction

Signal processing and numerical optimization are independent scientific fields that have always been mutually influencing each other. Perhaps the most convincing example where the two fields have met is compressed sensing (CS) [1]. CS originally treated the classic signal processing problem of reconstructing a continuous signal from its digital counterparts using a sub-Nyquist sampling rate. The reconstruction is achieved by solving an optimization problem known as the least absolute shrinkage and selection operator (LASSO) problem [2]. Stemming from the visibility given by CS, LASSO gained popularity within the signal processing community. Indeed, LASSO is a specific case of a *structured nonsmooth optimization problem*, and so representative of a more generic class of problems encompassing constrained and nonconvex optimization.

Developing efficient algorithms capable of solving structured nonsmooth optimization problems has been the focus of recent research efforts in the field of numerical optimization, because classical methods (*e.g.*, Newton-type) do not directly apply. In the context of convex optimization, such problems can be conveniently transformed into conic form and solved in a robust and efficient way using *interior point methods*. These methods became very popular as they are applicable to a vast range of optimization problems [3]. Unfortunately, they do not scale well with the problem size as they heavily rely on matrix factorizations and are therefore efficient for medium-size problems only [4].

More recently, there has been a renewed interest towards *splitting algorithms* [5, 6, 7]. These are first-order algorithms that minimize nonsmooth cost functions with minimal memory requirements allowing to tackle large-scale problems. The main disadvantage of splitting algorithms is their low speed of convergence, and hence a significant research effort has been devoted to their tuning and acceleration. Notable splitting algorithms are the proximal gradient (PG) algorithm [8, 9, 10], also known as forward-backward splitting (FBS) [11] or iterative shrinkage-thresholding algorithm (ISTA) [12], the alternating direction method of multipliers (ADMM) [13], the Douglas-Rachford splitting (DRS) [14] and the Pock-Chambolle algorithm (PC) [15]. The first acceleration of PG can be traced back to [16] and is known as the fast proximal gradient (FPG) algorithm or as fast iterative shrinkage-thresholding algorithm (FISTA) [12]. More recent acceleration approaches of PG include the variable metric forward-backward (VMFB) algorithm [17, 18, 19, 20] and the application of quasi-Newton methods [21, 22, 23, 24].

Several surveys dedicated to these algorithms and their applications in signal processing have appeared [6, 7, 4, 25], mainly focusing on convex problems only. In fact, only recently some extensions and analysis for nonconvex problems have started to emerge [26, 27]. In convex problems there is no distinction between local and global minima. For this reason, these problems are in general easier to solve than their nonconvex counterpart which are characterized by cost functions with multiple local minima. Despite this, it was recently shown that nonconvex formulations might either give solutions that exhibit better performance for the specific signal processing application [28], or lead to computationally tractable problems [29], for which the presence of spurious local minima is less pronounced or absent, and thus local optimization coupled with a proper initialization often leads to global minima [27].

This paper will focus on the PG algorithm and its accelerated variants, with the aim of introducing the latest trends of this numerical optimization framework to the signal processing community. The recent advances in the acceleration of the PG algorithm combined with matrix-free operations provide a novel flexible framework: in many signal processing applications such improvements allow tackling previously intractable problems and real-time processing. This framework will be presented in an effective and timely manner, summarizing the concepts that have led to these recent advances and providing easily accessible and user-friendly software tools. In particular, the paper will focus on the following topics:

- *Accelerated variants of PG:* FISTA has received significant attention in the signal processing community. However, more recently, the PG algorithm has been accelerated using different techniques: it has been shown that Quasi-Newton methods [24, 23] can significantly improve the algorithm performance and make it more robust to ill-conditioning.

- *Non-convex optimization:* proximal gradient algorithms can treat both convex and nonconvex optimization problems. While many convex relaxations increase dimensionality [30] and may result in computationally intractable problems, proximal gradient algorithms are directly applicable to the original nonconvex problem thus avoiding scalability issues. It is indeed of equal importance to have modeling tools and robust, rapidly converging algorithms for both nonconvex and convex nonsmooth problems. This allows to quickly test different problem formulations independently of their smoothness and convexity.

- *Forward-adjoint oracles and matrix-free optimization:* one important feature of proximal gradient algorithms is that they usually only require direct and adjoint applications of the linear mappings involved in the problem: in particular, no matrix factorization is required and these algorithms can be implemented using *forward-adjoint oracles (FAOs)*, yielding *matrix-free optimization algorithms* [31, 32]. Many signal processing applications can readily make use of FAOs yielding a substantial decrease of the memory requirements.

- *A versatile, high-level modeling language:* many optimization frameworks owe part of their success to easily accessible software packages, *e.g.*, [33, 34]. These software packages usually provide intuitive interfaces where optimization problems can be described using mathematical notation. In this paper a new, open-source, high-level modeling language implemented in Julia [35] called `StructuredOptimization` will be presented. This combines efficient implementations of proximal gradient algorithms with a collection of FAOs and functions often used in signal processing, allowing the user to easily formulate and solve optimization problems.

A series of signal processing application examples will be presented throughout the paper in separate frames to support the explanations of various concepts. Additionally, these examples will include code snippets illustrating how easily problems are formulated in the proposed high-level modeling language.

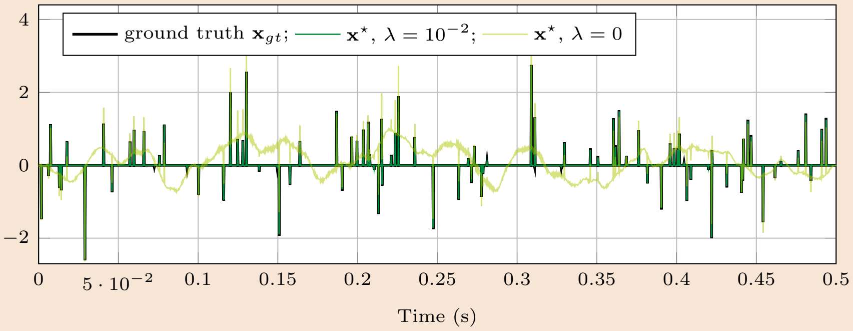
The paper is organized as follows: in Section 2 models and their usage in optimization are displayed through the description of inverse problems and the main differences between convex and nonconvex optimization. In Section 3 the classical proximal gradient algorithms and their accelerated variants are described. In Section 4 the concepts of FAOs and matrix-free optimization are introduced. Section 5 describes the types of problems that proximal gradient algorithms can tackle. Finally, in Section 6 the proposed high-modeling language is described and conclusions are given in Section 7.

2. Modeling

Models allow not only to describe physical phenomena, but also to transform signals and access hidden information that these often carry. Models may be *physical models*, obeying the laws of physics and describing *e.g.*, mechanical systems, electrical circuits or chemical reactions or *parametric models*, which are not necessarily linked to physical phenomena, and are purely defined by mathematical relations and numerical parameters. In general, both categories of models consist of defining an associated mapping that links an input signal $x(t)$ to an output signal $y(t)$. Here t may stand for time, but signals could be also N-dimensional *e.g.*, $x(t_1, \dots, t_N)$, and be functions of many quantities: position, temperature or the index of a pixel of a digital image. For computational convenience if the models are continuous, they must be discretized: the continuous signals involved are sampled and their samples stored either in vectors $\mathbf{x} = [x(t_1), \dots, x(t_n)]^\intercal \in \mathbb{R}^n$, in matrices $\mathbf{X} \in \mathbb{R}^{n_1 \times n_2}$ or in tensors $\boldsymbol{\mathcal{X}} \in \mathbb{R}^{n_1 \times \cdots \times n_N}$ depending on their dimensionality. In the following these vectors, matrices and tensors will be referred to as signals as well. The mapping $\mathcal{A} : \mathtt{D} \to \mathtt{C}$ associated with a model therefore links two (perhaps complex) finite-dimensional spaces $\mathtt{D}$ and $\mathtt{C}$ like, for example, $\mathcal{A} : \mathbb{C}^n \to \mathbb{C}^m$. Notice that in this work a (complex) finite-dimensional space $\mathtt{D}$ induces a Hilbert space with *real* inner product *i.e.*, $\langle \mathbf{x}, \mathbf{y} \rangle = \mathfrak{Re}(\mathbf{x}^{\mathsf{H}} \mathbf{y})$, where $^{\mathsf{H}}$ is the conjugate-transpose operation, and with norm $\|\mathbf{x}\| = \sqrt{\langle \mathbf{x}, \mathbf{x} \rangle}$. Mappings can also be defined between the Cartesian product of m and n finite-dimensional spaces: $\mathcal{A} : \mathtt{D}_1 \times \cdots \times \mathtt{D}_m \to \mathtt{C}_1 \times \cdots \times \mathtt{C}_n$ for example when dealing with multiple-input multiple-output models.

Depending on the nature of the models, mappings can be either *linear* or *nonlinear*. Such distinction often carries differences on the algorithms where these mappings are employed: as it will be described later, in optimization this often draws the line between convex and nonconvex optimization. For this reason, here the nonlinear mappings are indicated with the notation $\mathcal{A} : \mathtt{D} \to \mathtt{C}$ while the linear mappings with the notation $\mathsf{A} \in \mathscr{L}(\mathtt{D}, \mathtt{C})$, where $\mathscr{L}$ is the set of all linear mappings between $\mathtt{D}$ and $\mathtt{C}$.

Example 1 Sparse Deconvolution



Deconvolution seeks to recover the input signal $\mathbf{x}^\star$ from the available output signal $\mathbf{y}$ of a LTI system with FIR $\mathbf{h}$. In a single-channel case with low SNR, deconvolution can easily become an ill-posed problem ($\lambda = 0$) and regularization must be added in order to achieve meaningful results. If $\mathbf{x}^\star$ is assumed to be sparse, a sparsity-inducing regularization function can be included in the optimization problem (LASSO):

$$\mathbf{x}^\star = \operatorname*{argmin}_{\mathbf{x}} \quad \underbrace{\tfrac{1}{2}\|\mathbf{h} * \mathbf{x} - \mathbf{y}\|^2}_{f(\mathbf{x})} + \underbrace{\lambda\|\mathbf{x}\|_1}_{g(\mathbf{x})}, \tag{1}$$

where $*$ indicates convolution and λ is a scalar that balances the weight between the regularization function and the data fidelity function.

`StructuredOptimization` code snippet:

```
Fs = 4000 # sampling frequency
x = Variable(div(Fs,2)) # ‘ls‘ short-hand
                        # for ‘0.5*norm(...)^2‘
@minimize ls(conv(x,h)-y)+lambda*norm(x,1)
```

Models are often used to make predictions. For example it is possible to predict how a model behaves under the excitation of an input signal $\mathbf{x}$. The output signal $\mathbf{y} = \mathcal{A}\mathbf{x}$ can be computed by evaluating the mapping $\mathcal{A}$ associated with the model using a specific algorithm. Notice that here the notation $\mathcal{A}\mathbf{x}$ does not necessarily stand for the matrix-vector product. In many signal processing applications, however, one is interested in the opposite problem: an output signal $\mathbf{y}$ is at disposal, and the input signal $\mathbf{x}$ is to be found. Such problems are known as *inverse problems* and involve many signal processing applications some of which will be treated in this paper like denoising, source separation, channel equalization and system identification. Inverse problems are in general *ill-posed problems* and this makes it difficult to solve them. Three main challenges create this difficulty. Firstly the inverse of the mapping is needed. This is rarely available, it must be computed numerically and it may be unstable due to ill-conditioning. Secondly, $\mathbf{y}$ may be corrupted with noise or the model may be inaccurate, thus it is not possible to fully describe $\mathbf{y}$. Its noise and un-modeled features are then interpreted as effects caused by the input signal: this issue is called *over-fitting*. Finally inverse problems have in general non-unique solutions. These challenges are generally solved by *regularizing* the inverse problem: regularization attempts to exploit *prior information* over the structure of the sought signal which can simultaneously numerically stabilize the inversion of the mapping, find a unique solution and avoid over-fitting.

All of these concepts can be viewed in Example 1. Here a discrete linear time-invariant (LTI) model is used. This is a general parametric model capable of modeling many physical phenomena. Its linear mapping can consist of the discrete convolution between a finite impulse response (FIR) and the input signal: $\mathbf{y} = \mathbf{h} * \mathbf{x}$, where $\mathbf{h}$ is the signal containing the FIR taps. Example 1 treats the case of an inverse problem named *deconvolution*: this finds applications in a large number of signal processing fields and is known with different names such as channel equalization [36] or dereverberation [37]. What deconvolution seeks is to remove the effect of the channel from a signal $\mathbf{y}$ recorded by *e.g.*, a sensor: such signal could be a transmission received by an antenna or some speech recorded by a microphone, but many more examples could be made. The effect of the electromagnetic or acoustic channel corrupts either the sequence of bits of the transmission or the intelligibility of speech and should be therefore removed. This can be achieved by fitting the recorded signal $\mathbf{y}$ with the LTI model. Equation (1) shows the type of optimization problem that can bring such result. This consists of the minimization of a *cost function*, here consisting of the sum of two functions f and g. These are functions of $\mathbf{x}$ which in this context indicates the *optimization variables*. When the optimization variables minimize the cost function the *optimal solution* $\mathbf{x}^\star$ is obtained. Typically, solving (1) analytically is not possible and the optimal solution is reached numerically through the usage of specific iterative algorithms starting from an initial guess of the optimization variables $\mathbf{x}^0$. Here the functions f and g, associated with a lower case letter, are always assumed to be proper and closed. These always map an element of a finite-dimensional space to a single number belonging to the extended real line $\overline{\mathbb{R}} = \mathbb{R} \cup \{-\infty, +\infty\}$.

In Example 1 function f is a *data fidelity term*, representing the error between the model and the signal $\mathbf{y}$ in the least squares sense. The smaller this term is, the closer the output of the linear mapping, driven by the solution $\mathbf{x}^\star$, is to $\mathbf{y}$. On the other hand, g is a l_1-norm *regularization term*: this is a convex relaxation of the l_0-norm, *i.e.*, the norm counting the number of non-zero elements of $\mathbf{x}$, and promotes sparsity in the solution $\mathbf{x}^\star$. The coefficient λ balances the relative weight of the data fidelity and sparsity inducing terms in the cost function. Here, the signal $\mathbf{y}$ is corrupted using white noise as if it was recorded in a noisy environment or by a faulty sensor.

The figure of Example 1 shows what would happen if $\lambda = 0$, *i.e.*, the case of an un-regularized inverse problem. This solution has low-frequency oscillations, a sign of the numerical instability of the inverted linear mapping. Additionally, it is populated by white noise indicating over-fitting. On the other hand, if $\lambda \to \infty$ the prior knowledge would dominate the cost function leading to the sparsest solution possible, that is a null solution. Generally, λ needs careful tuning, a procedure that can be automatized by means of different strategies which may involve a sequence of similar optimization problems. Here, with a properly tuned λ the ground truth signal is recovered almost perfectly.

2.2. Convex and nonconvex problems

As it is important to choose the proper model to describe the available data, so it is to select the most convenient problem formulation. Different problem formulations can in fact yield optimization problems with different properties and one should be able to carefully choose the one that is best suited for the application of interest.

Perhaps the most fundamental distinction is between *convex* and *nonconvex* optimization problems. A problem of the form

$$\begin{aligned} &\underset{\mathbf{x}}{\text{minimize}} && \varphi(\mathbf{x}) \\ &\text{subject to} && \mathbf{x} \in \mathscr{C} \end{aligned} \tag{2}$$

is convex when φ is a convex function and $\mathscr{C}$ is a convex set.[1] The main advantage of convex problems lies in the fact that every local minimum is a global one. On the contrary, nonconvex problems can have *sub-optimal* local minima: these are identified as solutions, but it is usually not possible to determine whether there exist other solutions that further minimize the cost function. As a consequence of this, the *initialization* of iterative algorithms used in nonconvex optimization becomes crucial, since the quality of the solution found usually depends on it.

In order to avoid this issue, many nonconvex problems can be re-formulated or approximated by convex ones: it is often possible to relax the nonconvex functions by substituting them with convex ones that have similar properties. The LASSO is a good example of such a strategy: the original problem involves an l_0-norm, which is a nonconvex function. Despite very different mathematically, the l_1-norm behaves similarly by inducing sparsity over the optimization variables it is being applied to. Such relaxation however can have consequences on the solution and Example 2 displays such situation. Here the problem of *line spectral estimation* is treated: this has many applications like source localization [38], de-noising [39], and many others. A signal $\mathbf{y} \in \mathbb{R}^n$ is given and is modeled as a mixture of sinusoidal components. These lie on a fine grid of frequencies belonging to a discrete Fourier transform (DFT) and hence corresponding to the elements of a complex-valued signal $\mathbf{x}^\star \in \mathbb{C}^{sn}$ which must be estimated.

[1] Set $\mathscr{C}$ is convex if $\alpha\mathbf{x} + (1-\alpha)\mathbf{y} \in \mathscr{C}$, for any $\mathbf{x}, \mathbf{y} \in \mathscr{C}$ and $\alpha \in [0,1]$. Function φ is convex if its domain is convex and $\alpha\varphi(\mathbf{x}) + (1-\alpha)\varphi(\mathbf{y}) \geq \varphi(\alpha\mathbf{x} + (1-\alpha)\mathbf{y})$, for any $\mathbf{x}, \mathbf{y}$ in the domain of φ and $\alpha \in [0,1]$.

The optimization problem seeks for a sparse solution as these components are assumed to be only few.

Looking at the figure of Example 2 it is clear that the solution of the nonconvex problem outperforms the one obtained through the LASSO. The solution of LASSO has in fact many small spurious frequency components. These are not present in the solution of the nonconvex problem which also exhibits amplitudes that are much closer to the ones of the ground truth. This shows that convex relaxations may lead to poorer results than the ones obtained by solving the original nonconvex problem. However, as stated earlier, the presence of local minima requires the problem to be initialized carefully. Indeed, the improved performance of the solution obtained by solving the nonconvex problem in Example 2 would have been very hard to accomplish with a random initialization: most likely a “bad” local minimum would have been reached corresponding to a solution with completely wrong amplitudes and frequencies. Instead, by warm-starting the nonconvex problem with the solution of the LASSO, a “good” local minimum is found.

There are very few nonconvex problems that are not particularly affected by the initialization issue. A lucky case, under appropriate assumptions, is the one of robust principal component analysis (PCA) [40] (Example 4). In general, however, what is typically done is to come up with a good strategy for the initialization. Obviously, the path adopted in Example 2, *i.e.*, initializing the nonconvex problem with the solution of a convex relaxation, is not always accessible. In fact a general rule for initialization does not exist and this is usually problem-dependent: different strategies may involve random initialization using distributions that are obtained by analyzing the available data [41] (Example 3) or by solving multiple times the optimization problems while modifying parameters that govern the nonconvexity (Example 6).

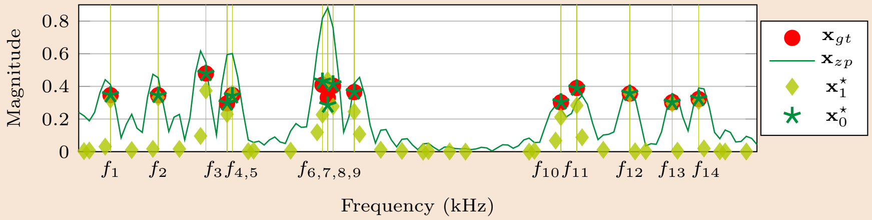
Despite these disadvantages, nonconvex optimization is becoming more and more popular for multiple reasons. Firstly, as Example 2 has just shown, sometimes the quality of the solution of a convex relaxation is not satisfactory. Secondly, convex relaxations may come at the cost of a larger optimization problem with respect to the original nonconvex one [30, 28, 42] and may be prohibitive in terms of both memory and computational power. Finally, sometimes convex relaxations are simply not possible, for example when nonlinear mappings are involved in the optimization problems. These nonlinear mappings are typically derived from complex models which have shown to produce outstanding results and are becoming very common for example in machine learning and model predictive control. For these reasons, having algorithms that are convergent both for convex and nonconvex problems is quite important.

3. Proximal gradient algorithms

All of the problems this paper treats, including the ones of Examples 1 and 2, can be formulated as

$$\underset{\mathbf{x}}{\text{minimize}} \qquad \varphi(\mathbf{x}) = f(\mathbf{x}) + g(\mathbf{x}) \tag{4}$$

Example 2 Line spectral estimation



Line spectral estimation seeks to accurately recover the frequencies and amplitudes of a signal $\mathbf{y} \in \mathbb{R}^n$ which consists of a mixture of N sinusoids. A simple solution is take the zero-padded DFT of $\mathbf{y}$: $\mathbf{x}_{zp} = \mathsf{F}[\mathbf{y}, 0, \dots 0] \in \mathbb{C}^{sn}$ where s is the super-resolution factor and $\mathsf{F} \in \mathscr{L}(\mathbb{R}^{sn}, \mathbb{C}^{sn})$ the DFT mapping. However, looking at $\mathbf{x}_{zp}$, spectral leakage causes components at close frequencies to merge. This issue is not present if the following optimization problems are used for line spectral estimation:

$$\mathbf{x}_1^\star = \underset{\mathbf{x}}{\text{argmin}} \, \frac{1}{2}\|\mathsf{S}\mathsf{F}_i\mathbf{x} - \mathbf{y}\|^2 + \lambda\|\mathbf{x}\|_1, \tag{3a}$$

$$\mathbf{x}_0^\star = \underset{\mathbf{x}}{\text{argmin}} \, \frac{1}{2}\|\mathsf{S}\mathsf{F}_i\mathbf{x} - \mathbf{y}\|^2 \text{ subject to } \|\mathbf{x}\|_0 \leq N, \tag{3b}$$

here $\mathbf{x} \in \mathbb{C}^{sn}$ consists of the candidate sparse sinusoidal components, $\mathsf{F}_i \in \mathscr{L}(\mathbb{C}^{sn}, \mathbb{R}^{sn})$ is the inverse DFT and $\mathsf{S} \in \mathscr{L}(\mathbb{R}^{sn}, \mathbb{R}^n)$ is a mapping that simply selects the first n elements. Problem (b) is nonconvex which means that it has several local minima and its convex relaxation (a) is typically solved instead (LASSO). Nevertheless, PG methods can solve (a) as well as (b): if a good initialization is given, *e.g.*, the solution of (a), improved results can be achieved.

`StructuredOptimization` **code snippet:**

```
x = Variable(s*n)                              # n = 2^8   s = 6
@minimize ls(ifft(x)[1:n]-y)+lambda*norm(x,1)      # (a)
@minimize ls(ifft(x)[1:n]-y) st norm(x,0) <= N     # (b)
```

where f is a smooth function (*i.e.*, it is differentiable, and its gradient ∇f is Lipschitz-continuous), while g is possibly nonsmooth. Despite its simplicity, problem (4) encompasses a large variety of applications. For example, constrained optimization can be formulated as (4): for a (nonempty) set $\mathscr{S}$, by setting g to be the *indicator function* of $\mathscr{S}$, that is

$$g(\mathbf{x}) = \delta_{\mathscr{S}}(\mathbf{x}) = \begin{cases} 0 & \text{if } \mathbf{x} \in \mathscr{S}, \\ +\infty & \text{otherwise,} \end{cases} \tag{5}$$

then (4) is equivalent to minimizing f subject to the constraint $\mathbf{x} \in \mathscr{S}$.

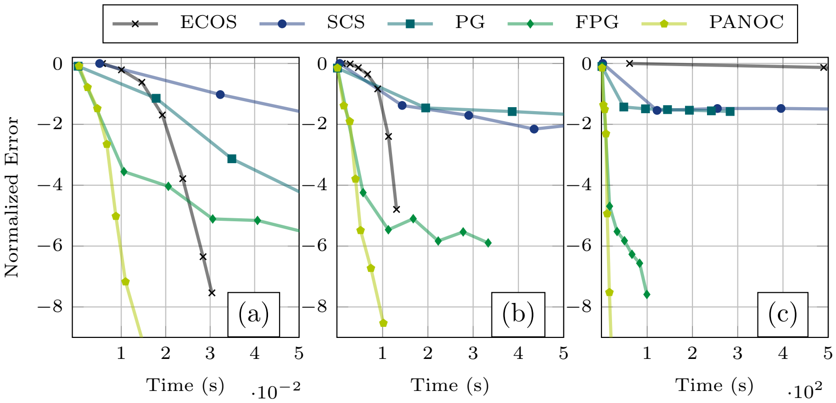


Figure 1: Comparison of the performances of different algorithms when solving the LASSO: $\text{argmin}_{\mathbf{x}} \|\mathbf{A}\mathbf{x} - \mathbf{y}\|_2^2 + \lambda \|\mathbf{x}\|_1$, where $\mathbf{A} \in \mathbb{R}^{n/5 \times n}$ is a random sparse matrix with $n/4$ non-zero elements and $\lambda = 10^{-3} \|\mathbf{A}^\intercal \mathbf{y}\|_\infty$. Different sizes of the problem are solved using the same random matrices for (a) $n = 10^3$, (b) $n = 10^4$ and (c) $n = 10^5$. Here the normalized error is defined as: $\log(\|\mathbf{x}^k - \mathbf{x}^\star\|_2 / \|\mathbf{x}^\star\|_2)$, where $\mathbf{x}^k$ is the k-th iterate $\mathbf{x}^\star$ the optimal solution.

The presence of a nonsmooth function, like for example the l_1-norm that appeared in the problems encountered so far, prevents from applying classical optimization algorithms such as gradient descent, nonlinear conjugate gradient or (quasi-)Newton methods [43]. These algorithms are in fact based on derivatives and do not apply to the minimization of non-differentiable functions. Although the definition of derivative can be generalized to nonsmooth functions as well through the usage of the *subdifferential*

$$\partial g(\mathbf{x}) = \{ \mathbf{v} \mid g(\mathbf{y}) \geq g(\mathbf{x}) + \langle \mathbf{v}, \mathbf{y} - \mathbf{x} \rangle \ \forall \mathbf{y} \}, \tag{6}$$

where here g is assumed to be convex, (see [44, Def. 8.3] for the subdifferential definition in the nonconvex case), its usage in these algorithms is often not possible or leads to restrictive convergence properties.

One of the most successful families of algorithms that can deal with nonsmooth cost functions is the one of *interior point methods*. These can in fact be applied to almost every convex problem by transforming it into a standard problem formulation called *conic form* [3]. Unfortunately, most of these algorithms usually require matrix factorizations and are therefore competitive only for small and medium-sized problems. Figure 1 displays this behavior by comparing the time it takes to achieve a specific accuracy of the solution for different sizes of a LASSO problem: here the embedded conic solver (ECOS) algorithm [45], which utilizes a standard path-following interior point method, performs very well for small-scale problems but cannot handle large-scale ones employing quite some time even to reach a solution of low accuracy. In order to overcome this issue while still embracing the large variety of problems that the conic form offers, splitting algorithms have been used also in this context using a variation of ADMM called splitting conic solver (SCS) [46]: as Figure 1 shows, this al-

g	$\mathrm{prox}_{\gamma g}$	**Properties**
$\|\mathbf{x}\|_0$	x_i if $\|x_i\| > \sqrt{2\gamma}$, 0 elsewhere	nonconvex, separable
$\|\mathbf{x}\|_1$	$\mathcal{P}_+(\mathbf{x}-\gamma) - \mathcal{P}_+(-\mathbf{x}-\gamma)$	convex, separable
$\|\mathbf{x}\|$	$\max\{0, 1-\gamma/\|\mathbf{x}\|\}\mathbf{x}$	convex
$\|\mathbf{X}\|_*$	$\mathbf{U}\,\mathrm{diag}\,(\mathcal{P}_+(\boldsymbol{\sigma}-\gamma))\,\mathbf{V}^{\mathsf{H}}$	convex
$\frac{1}{2}\|\mathbf{A}\mathbf{x}-\mathbf{b}\|^2$	$(\mathbf{A}^{\mathsf{H}}\mathbf{A}+\gamma^{-1}\mathsf{Id})^{-1}(\mathbf{A}^{\mathsf{H}}\mathbf{b}+\gamma^{-1}\mathbf{x})$	convex
$\mathscr{S}$	$\Pi_{\mathscr{S}}$	
$\{\mathbf{x} \mid \|\mathbf{x}\|_0 \leq m\}$	$\mathcal{P}_m\mathbf{x}$	nonconvex
$\{\mathbf{x} \mid \|\mathbf{x}\| \leq r\}$	$r/\|x\|\mathbf{x}$ if $\|\mathbf{x}\| > r$, $\mathbf{x}$ otherwise	convex
$\{\mathbf{X} \mid \mathrm{rank}(\mathbf{X}) \leq m\}$	$\mathbf{U}\,\mathrm{diag}(\mathcal{P}_m\boldsymbol{\sigma})\mathbf{V}^{\mathsf{H}}$	nonconvex
$\{\mathbf{x} \mid l \leq \mathbf{x} \leq u\}$	$\min\{u, \max\{l, x_i\}\}\ \forall\ i = 1,\dots,n$	convex, separable
$\{\mathbf{x} \| \mathbf{A}\mathbf{x} = \mathbf{b}\}$	$\mathbf{x} + \mathbf{A}^{\mathsf{H}}(\mathbf{A}\mathbf{A}^{\mathsf{H}})^{-1}(\mathbf{b}-\mathbf{A}\mathbf{x})$	convex

Table 1: Table showing the proximal operators of a selection of functions g and indicator function $\delta_{\mathscr{S}}$ with sets $\mathscr{S}$. Here given a n long vector $\mathbf{x}$, $\mathcal{P}_+\mathbf{x}$ returns $[\max\{0, x_1\}, \dots, \max\{0, x_n\}]^{\mathsf{T}}$ while $\mathcal{P}_m\mathbf{x}$ returns a copy of $\mathbf{x}$ with all elements set to 0 except for the m largest in modulus. The matrices $\mathbf{U}$ and $\mathbf{V}$ are the result of a SVD: $\mathbf{X} = \mathbf{U}\,\mathrm{diag}(\boldsymbol{\sigma})\mathbf{V}^{\mathsf{H}}$ where $\boldsymbol{\sigma}$ is the vector containing the singular values of $\mathbf{X}$. See [47, Sec. 6.9] for a more exhaustive list of proximal operators.

gorithm outperforms standard interior point methods for large-scale problems, reaching a solution of relatively low accuracy but at a much faster rate. However, SCS is in some cases outperformed by PG and FPG, which are introduced later in this section, despite the fact that ADMM has been observed to converge faster in many contexts [7]. This trend reversal of SCS is most likely caused by the transformation of the original problem into its conic form: this changes dramatically the problem structure and introduces additional slack variables which inevitably increase the already large size of the problem. Another advantage of proximal gradient algorithms is their compactness: splitting algorithms like ADMM or DRS ultimately require solving large linear systems which often becomes a computational bottleneck. This requires matrix factorizations or subroutines like conjugate gradient methods which are usually not necessary in proximal gradient algorithms. In Section 3.4 the results shown in Figure 1 will be further discussed.

3.1. Proximal mappings

One way to deal with nonsmooth functions in the objective function to be minimized, is through their *proximal mapping* (or *operator*) [48]. For a (possibly nonsmooth) function g, this is defined as

$$\mathbf{z}^\star = \mathrm{prox}_{\gamma g}(\mathbf{x}) = \underset{\mathbf{z}}{\mathrm{argmin}} \left\{ g(\mathbf{z}) + \frac{1}{2\gamma}\|\mathbf{z}-\mathbf{x}\|^2 \right\} \tag{7}$$

where γ a positive scalar. Here the minimization of g is penalized by the presence of an additional quadratic function that enforces the solution $\mathbf{z}^\star$ to be in the *proximity* of $\mathbf{x}$. Parameter γ controls this proximity and acts as a stepsize: small

	$g(\mathbf{x})$	$\mathrm{prox}_{\gamma g}(\mathbf{x})$	**Requirements**
Separable sum	$h_1(\mathbf{x}_1) + h_2(\mathbf{x}_2)$	$[\mathrm{prox}_{\gamma h_1}(\mathbf{x}_1)^\intercal, \mathrm{prox}_{\gamma h_2}(\mathbf{x}_2)^\intercal]^\intercal$	$\mathbf{x} = [\mathbf{x}_1^\intercal, \mathbf{x}_2^\intercal]^\intercal$
Translation	$h(\mathbf{x} + \mathbf{b})$	$\mathrm{prox}_{\gamma h}(\mathbf{x} + \mathbf{b}) - \mathbf{b}$	
Affine addition	$h(\mathbf{x}) + \langle \mathbf{a}, \mathbf{x} \rangle$	$\mathrm{prox}_{\gamma h}(\mathbf{x} - \gamma \mathbf{a})$	
Postcomposition	$ah(\mathbf{x}) + b$	$\mathrm{prox}_{a\gamma h}(\mathbf{x})$	$a > 0$
Precomposition	$h(\mathsf{A}\mathbf{x})$	$\mathbf{x} +$ $\mu^{-1}\mathsf{A}^* \left(\mathrm{prox}_{\mu\gamma h}(\mathsf{A}\mathbf{x}) - \mathsf{A}\mathbf{x}\right)$	$\mathsf{A}\mathsf{A}^* = \mu \mathsf{Id}$, $\mu \geq 0$
Regularization	$h(\mathbf{x}) + \frac{\rho}{2}\|\mathbf{x} - \mathbf{b}\|^2$	$\mathrm{prox}_{\tilde{\gamma} h}(\tilde{\gamma}(1/\gamma \mathbf{x} + \rho \mathbf{b}))$	$\tilde{\gamma} = \gamma/(1 + \gamma\rho)$, $\rho \geq 0$
Convex Conjugate	$\sup_{\mathbf{x}}\{\langle \mathbf{x}, \mathbf{u} \rangle - h(\mathbf{x})\}$	$\mathbf{u} - \gamma\, \mathrm{prox}_{(1/\gamma)h}(\mathbf{u}/\gamma)$	h convex

Table 2: Table showing different properties of proximal mappings.

values of γ will result in $\mathbf{z}^\star$ being very close to $\mathbf{x}$, while large ones will yield a solution close to the minimum of g.

For many functions the correspondent proximal mappings have closed-form solutions and can be computed very efficiently. Table 1 shows some examples for functions which are commonly used in applications. For example, the proximal mapping of $g(\cdot) = \lambda\|\cdot\|_1$, consists of a "soft-thresholding" operation of $\mathbf{x}$, while for l_0-norm it is the so-called "hard-thresholding" operation. When g is the indicator function of a set $\mathscr{S}$, cf. (5), then $\mathrm{prox}_{\gamma g} = \Pi_{\mathscr{S}}$, the projection onto $\mathscr{S}$. As Table 1 shows, many of these projections are also cheaply computable like in the case of boxes, norm balls, affine subspaces.

However, an analytical solution to (7) is not always available. For example, given two functions h_1 and h_2, the fact that $\mathrm{prox}_{\gamma h_1}$ and $\mathrm{prox}_{\gamma h_2}$ can be efficiently computed does not necessarily imply that $\mathrm{prox}_{\gamma(h_1+h_2)}$ is efficient as well. Additionally, the proximal mapping of the composition $g \circ \mathsf{A}$ of a function g with a linear operator A, is also not efficient in general. An exception to this is linear least squares: if $g(\cdot) = \frac{1}{2}\|\cdot - \mathbf{b}\|^2$ is composed with matrix $\mathbf{A}$, the proximal mapping of function $g(\mathbf{A}\mathbf{x}) = \frac{1}{2}\|\mathbf{A}\mathbf{x} - \mathbf{b}\|^2$ has in fact a closed-form solution, which however requires solving a linear system as Table 1 shows. When this linear system is large (*i.e.*, $\mathbf{A}$ has a large number of columns) such inversion may be infeasible to tackle with direct methods (such as QR decomposition or Cholesky factorization), and one may need to resort to iterative algorithms, *e.g.*, using conjugate gradient. In general, composition by a linear mapping results in an efficient closed-form proximal mapping only when A satisfies $\mathsf{A}\mathsf{A}^* = \mu\mathsf{Id}$ where $\mu \geq 0$, Id is the identity mapping and A^* is the adjoint mapping of A (see Section 4 for the definition of adjoint mapping). Linear mappings with such properties are called *tight frames*, and include orthogonal mappings like the discrete Fourier transform (DFT) and discrete cosine transform (DCT).

Many properties can be exploited to derive closed-form expressions for proximal operators: Table 2 summarizes some of the most important ones [5]. Among

Algorithm 1 Proximal Gradient Algorithm (PG)

```
Set $\mathbf{x}^0 \in \mathbb{R}^n$, and $\gamma \in (0, 2L_f^{-1}]$
for $k = 0, 1, \dots$ do
    $\mathbf{x}^{k+1} = \mathrm{prox}_{\gamma g}(\mathbf{x}^k - \gamma \nabla f(\mathbf{x}^k))$
```

these, the separable sum is particularly useful: if h_1 and h_2 have efficiently computable proximal mappings, then so does function $g(\mathbf{x}_1, \mathbf{x}_2) = h_1(\mathbf{x}_1) + h_2(\mathbf{x}_2)$. For example, using the properties in Table 2, it is very easy to compute the proximal mapping of $g(x) = \|\mathrm{diag}(\mathbf{d})\mathbf{x}\|_1 = \sum |d_i x_i|$.

If function g is convex then $\mathrm{prox}_{\gamma g}$ is everywhere well defined: (7) consists of the minimization of a strongly convex objective, and as such has a unique solution for any $\mathbf{x}$. When g is nonconvex, this may not hold. Existence of solutions to (7) in this case is guaranteed for example if g is lower bounded. For some $\mathbf{x}$ however, problem (7) may have multiple solutions, *i.e.*, for nonconvex g the operator $\mathrm{prox}_{\gamma g}$ is *set-valued* in general. As an example of this, consider set $\mathscr{B}_{0,m} = \{\mathbf{x} \mid \|\mathbf{x}\|_0 \leq m\}$, *i.e.*, the ℓ_0 pseudo-norm ball. Projecting a point $\mathbf{x} \in \mathbb{R}^n$ onto $\mathscr{B}_{0,m}$ amounts to setting to zero its $n - m$ smallest coefficients in magnitude. Consider $n = 5$, $m = 3$, and point $\mathbf{x} = [5.7, -2.4, 1.2, 1.2, 1.2]^\intercal$: in this case there are three points in $\mathscr{B}_{0,3}$ which are closest to $\mathbf{x}$. In fact

$$\Pi_{\mathscr{B}_{0,3}}(\mathbf{x}) = \{[5.7, -2.4, 1.2, 0, 0]^\intercal, [5.7, -2.4, 0, 1.2, 0]^\intercal, [5.7, -2.4, 0, 0, 1.2]^\intercal\}\,. \tag{8}$$

In practice, proximal mappings of nonconvex functions are evaluated by choosing a single element out of its set.

3.2. Proximal gradient method

A very popular algorithm to solve (4), when φ is the sum of a smooth function f and a (possibly) nonsmooth function g with efficient proximal mapping, is the *proximal gradient (PG)* algorithm: this combines the gradient descent, a well known first-order method, with the proximal mapping described in Section 3.1.

The PG algorithm is illustrated in Algorithm 1: here $\mathbf{x}^0$ is the initial guess, γ represents a stepsize and ∇f is the gradient of the smooth function f. The algorithm consists of alternating gradient (or *forward*) steps on f and proximal (or *backward*) steps on g, and is a particular case of the forward-backward splitting (FBS) algorithm for finding a zero of the sum of two monotone operators [49, 8]. The reason behind this terminology is apparent from the optimality condition of the problem defining the proximal operator (7): if $\mathbf{z} = \mathrm{prox}_{\gamma g}(\mathbf{x})$, then necessarily $\mathbf{z} = \mathbf{x} - \gamma \mathbf{v}$, with $\mathbf{v} \in \partial g(\mathbf{x})$, *i.e.*, $\mathbf{z}$ is obtained by an *implicit* (backward) subgradient step over g, as opposed to the explicit (forward) step over f.

The steps of the algorithm are visualized in Figure 2: the gradient step moves the iterate $\mathbf{x}^k$ towards the minimum of f, while the proximal step makes progress towards the minimum of g. This alternation will ultimately lead to the minimum of the sum of these two functions. In fact, in the convex case (*i.e.*,

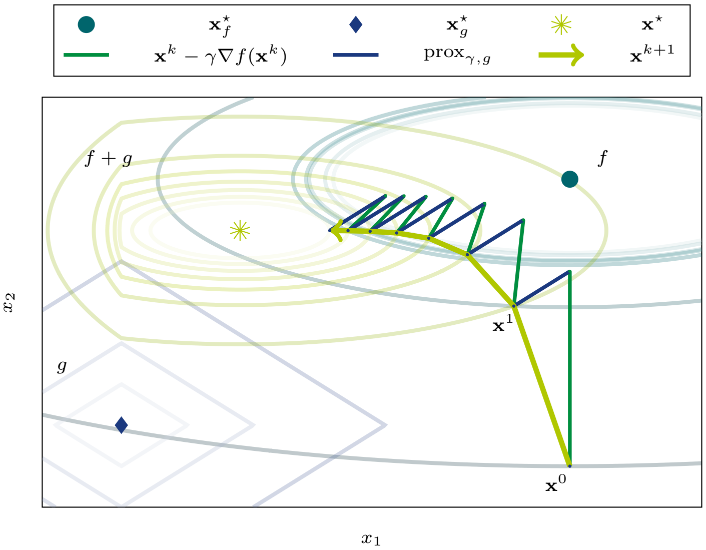


Figure 2: Figure showing an example of the path that the PG algorithm creates to reach the optimal value $\mathbf{x}^\star$. Here the minima of the functions f and g are shown in $\mathbf{x}_f^\star$ and $\mathbf{x}_g^\star$ respectively.

when both f and g are convex), the iterates $\mathbf{x}^k$ in Algorithm 1 are known to converge under minimal assumptions to a global minimum, for $\gamma \in (0, 2/L_f)$ where L_f is a Lipschitz constant of ∇f, see [5, Cor. 27.9]. Furthermore, in this case the algorithm converges with global *sublinear rate* $O(1/k)$ for the objective value, as stated in the following result.

Theorem 1 ([12, Thm. 3.1]). *If f and g are convex, then the sequence of iterates $\mathbf{x}^k$ generated by Algorithm 1 satisfies*

$$\varphi(\mathbf{x}^k) - \varphi_\star \leq \frac{\gamma L_f \|\mathbf{x}^k - \mathbf{x}^\star\|}{2k},$$

where $\mathbf{x}^\star$ is any solution to (4)*.*

Notice that when the Lipschitz constant L_f is not known, a suitable γ can be adaptively determined by means of a backtracking procedure [12]. Convergence of Algorithm 1 has also been studied in the nonconvex case (*i.e.*, where both f and g are allowed to be nonconvex): in this case convergence to a *critical point* of φ, *i.e.*, a point $\bar{\mathbf{x}}$ satisfying $-\nabla f(\bar{\mathbf{x}}) \in \partial g(\bar{\mathbf{x}})$, can be proved under the assumption that φ satisfies the *Kurdyka-Lojasiewicz* property [26, Def. 2.4]. This is a rather mild requirement, and is satisfied for example by all semi-algebraic functions, including the objectives in the examples in the present article.

Theorem 2 ([26, Thm. 5.1]). *In Algorithm 1, all accumulation points of the sequence $\mathbf{x}^k$ are critical points of φ. Suppose now that φ in* (4) *is lower bounded*

Algorithm 2 Fast proximal gradient algorithm (FPG) [12]

```
Set $\mathbf{v}^0 = \mathbf{x}^{-1} \in \mathbb{R}^n$, $\gamma \in (0, L_f^{-1}]$, and $\theta_0 = 1$
for $k = 0, 1, \dots$ do
    $\mathbf{x}^k = \operatorname{prox}_{\gamma g}(\mathbf{v}^k - \gamma \nabla f(\mathbf{v}^k))$
    $\theta_{k+1} = \frac{1}{2}\left(1 + \sqrt{1 + 4\theta_k^2}\right)$
    $\mathbf{v}^{k+1} = \mathbf{x}^k + (\theta_k - 1)\theta_{k+1}^{-1}(\mathbf{x}^k - \mathbf{x}^{k-1})$
```

and has the Kurdyka-Łojasiewicz *property [26, Def. 2.4], and that* $\gamma \in (0, L_f^{-1})$. *If the sequence* $\mathbf{x}^k$ *is bounded, then it converges to a critical point of* φ.

Fast variants of the algorithm exist, such as the fast proximal gradient (FPG) algorithm (also known as fast iterative shrinkage-thresholding algorithm (FISTA) [12]), shown in Algorithm 2: this is an extension of the optimal first-order methods for convex smooth problems, pioneered by Nesterov [16], to the case where the additional nonsmooth function g is present.

In addition to the original iterates $\mathbf{x}^k$, FPG computes an extrapolated sequence $\mathbf{v}^k$ by performing a linear combination of previous two iterates. Intuitively, this provides *intertia* to the computed sequence, which improves the convergence speed over PG, from $O(1/k)$ to $O(1/k^2)$.

Theorem 3 ([12, Thm. 4.4]). *If f and g are convex, then the sequence of iterates $\mathbf{x}^k$ generated by Algorithm 2 satisfies*

$$\varphi(\mathbf{x}^k) - \varphi_\star \le \frac{2\gamma L_f \|\mathbf{x}^k - \mathbf{x}^\star\|}{(k+1)^2},$$

where $\mathbf{x}^\star$ *is any solution to* (4).

This method is particularly appealing since the extrapolated sequence $\mathbf{v}^k$ only requires $O(n)$ floating point operations to be computed. However, the convergence of FPG has only been proven when both f and g are convex: an extension of this algorithm has been proposed in [50], that preserves the fast global convergence rate under the assumptions of Theorem 3, while converging to critical points under assumptions similar to those of Theorem 2.

3.3. Forward-backward envelope

Recently, new algorithms based on the PG algorithm have emerged: these rely on the concept of the *forward-backward envelope (FBE)* which was first introduced in [51]. In order to explain what the FBE is, one should look at the PG algorithm from a different perspective. Using the definition of $\operatorname{prox}_{\gamma g}$, with elementary manipulations the iterations of Algorithm 1 can be equivalently rewritten as,

$$\mathbf{x}^{k+1} = \operatorname*{argmin}_{\mathbf{z}} \Big\{ \overbrace{f(\mathbf{x}^k) + \langle \mathbf{z} - \mathbf{x}^k, \nabla f(\mathbf{x}^k) \rangle + \tfrac{1}{2\gamma}\|\mathbf{z} - \mathbf{x}^k\|^2}^{q_\gamma(\mathbf{z}, \mathbf{x}^k)} + g(\mathbf{z}) \Big\}, \tag{9}$$

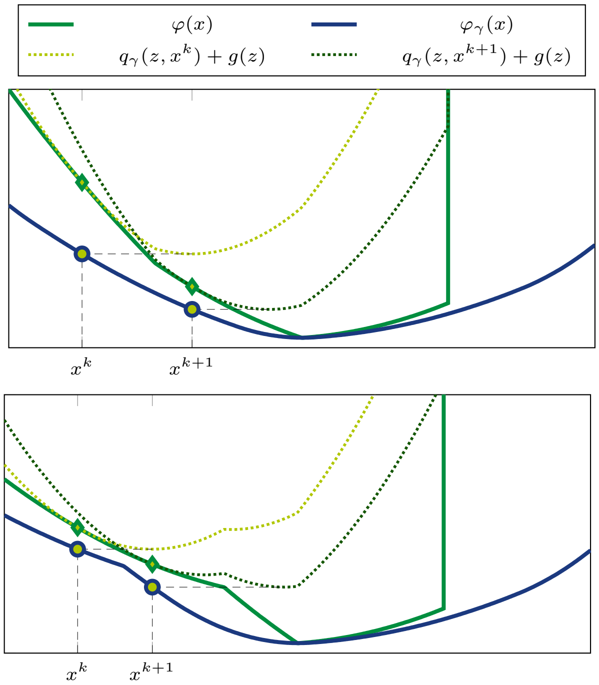


Figure 3: One step of PG amounts to a majorization-minimization over φ, when stepsizes γ is sufficiently small. The minimum value of such majorization is the forward-backward envelope φ_γ. Left: in the convex case, φ_γ is a smooth lower bound to the original objective φ. Right: in the nonconvex case φ_γ is not everywhere smooth.

that is, the minimization of g plus a *quadratic model* $q_\gamma(\mathbf{z}, \mathbf{x}^k)$ of f around the current iterate $\mathbf{x}^k$. When ∇f is Lipschitz continuous and $\gamma \leq L_f^{-1}$, then for all $\mathbf{x}$

$$\varphi(\mathbf{z}) = f(\mathbf{z}) + g(\mathbf{z}) \leq q_\gamma(\mathbf{z}, \mathbf{x}) + g(\mathbf{z}). \tag{10}$$

In this case the steps (9) of the PG algorithm are a *majorization-minimization* procedure. This is visualized, in the one-dimensional case, in Figure 3. The minimum *value* of (9) is the *forward-backward envelope* associated with problem (4), indicated by φ_γ:

$$\varphi_\gamma(\mathbf{x}) = \min_{\mathbf{z}} \left\{ f(\mathbf{x}) + \langle \mathbf{z} - \mathbf{x}, \nabla f(\mathbf{x}) \rangle + \tfrac{1}{2\gamma} \|\mathbf{z} - \mathbf{x}\|^2 + g(\mathbf{z}) \right\}. \tag{11}$$

The FBE has many noticeable properties: these are described in detail in [24] for the case where g is convex, and extended in [23] to the case where g is allowed to be nonconvex. First, φ_γ is a lower bound to φ, and the two functions share the same local minimizers. In particular, $\inf \varphi = \inf \varphi_\gamma$ and $\operatorname{argmin} \varphi = \operatorname{argmin} \varphi_\gamma$, hence minimizing φ_γ is equivalent to minimizing φ. Additionally, φ_γ is real-valued as opposed to φ which is *extended* real-valued: as Figure 3 shows, even

Algorithm 3 Proximal Averaged Newton-type algorithm for Optimality Conditions (PANOC)

```
Set $\mathbf{x}^0 \in \mathbb{R}^n$, $\gamma \in (0, L_f^{-1})$, and $\sigma \in (0, \frac{1}{2\gamma}(1-\gamma L_f))$
for $k = 0, 1, \ldots$ do
    $\mathbf{v}^k = \operatorname{prox}_{\gamma g}(\mathbf{x}^k - \gamma \nabla f(\mathbf{x}^k))$
    $\mathbf{d}^k = -\mathsf{H}_k(\mathbf{x}^k - \mathbf{v}^k)$ for some nonsingular $\mathsf{H}_k$
    $\mathbf{x}^{k+1} = (1-\tau_k)\mathbf{v}^k + \tau_k(\mathbf{x}^k + \mathbf{d}^k)$, for the largest value
        $\tau_k \in \{(1/2)^i \mid i \in \mathbb{N}\}$ such that $\varphi_\gamma(\mathbf{x}^{k+1}) \leq \varphi_\gamma(\mathbf{x}^k) - \sigma\|\mathbf{v}^k - \mathbf{x}^k\|^2$
```

at points where φ is $+\infty$, φ_γ has a finite value instead. Furthermore, when f is twice differentiable and g is convex, then φ_γ is continuously differentiable.

An important observation is that evaluating the FBE (11) essentially requires computing one proximal gradient step, *i.e.*, one step of Algorithm 1. This is an important feature from the algorithmic perspective: any algorithm that solves (4) by minimizing φ_γ (and thus needs its evaluation) requires exactly the same operations as Algorithm 1. In the next section one such algorithm is illustrated.

When applied to the *dual* of convex problems, the FBE has an important interpretation in terms of the augmented Lagrangian function. This relationship is thoroughly analyzed in [52]. An envelope function analogous to the FBE was also introduced in the context of the Douglas-Rachford splitting (DRS), and of its dual counterpart the alternating direction method of multipliers (ADMM), to obtain accelerated variants of the algorithms: these apply to nonconvex problems as well, the interested reader can refer to [53, 54].

3.4. Newton-type proximal gradient methods

In Section 3.3 it was shown that minimizing the FBE is equivalent to solving problem (4). Algorithm 3 is a generalization of the standard PG algorithm that minimizes the FBE using a backtracking line search.

The *Proximal Averaged Newton-type algorithm for Optimality Conditions (PANOC)* was proposed in [55], and the idea behind it is very simple: the PG algorithm is a fixed-point iteration for solving the system of nonsmooth, nonlinear equations $\mathcal{R}_\gamma(\mathbf{x}) = \mathbf{0}$, where

$$\mathcal{R}_\gamma(\mathbf{x}) = \mathbf{x} - \operatorname{prox}_{\gamma g}(\mathbf{x} - \gamma \nabla f(\mathbf{x})), \tag{12}$$

is the *fixed-point residual* mapping. In fact, it is immediate to verify that the iterates in Algorithm 1 satisfy

$$\mathbf{x}^{k+1} = \mathbf{x}^k - \mathcal{R}_\gamma(\mathbf{x}^k). \tag{13}$$

It is very natural to think of applying a Newton-type method, analogously to what is done for smooth, nonlinear equations [43, Chap. 11]:

$$\mathbf{x}^{k+1} = \mathbf{x}^k - \mathsf{H}_k \mathcal{R}_\gamma(\mathbf{x}^k), \tag{14}$$

where (H_k) is an appropriate sequence of nonsingular linear transformations. The update rule of Algorithm 3 is a convex combination of (13) and (14), dictated by the stepsize τ_k which is determined by backtracking line-search over the FBE. When $\tau_k = 1$ then (14) is performed; as $\tau_k \to 0$ then the update gets closer and closer to (13).

Note that Algorithm 3 reduces to Algorithm 1 for the choice $\mathsf{H}_k = \mathsf{Id}$: in this case the stepsize τ_k is always equal to 1. In fact, it is possible to prove very similar global convergence properties for general (possibly nonconvex) problems:

Theorem 4 ([23, Thm. 5.8]). *In Algorithm 3, all accumulation points of the sequence $\mathbf{x}^k$ are critical points of φ. Assume now that φ in* (4) *has the* Kurdyka-Łojasiewicz *property [26, Def. 2.4]. Suppose moreover that*

$$\|\mathbf{d}^k\| \leq D\|\mathbf{x}^k - \mathbf{v}^k\| \quad \textit{for all } k,$$

for some $D > 0$, that the sequence of iterates $\mathbf{x}^k$ is bounded, and that $f \in C^2$ around the cluster points of $\mathbf{x}^k$. Then $\mathbf{x}^k$ converges to a critical point of φ.

However, by carefully choosing H_k one can greatly improve the asymptotic convergence rate. In [55] the case of *quasi-Newton* methods is considered: start with $\mathsf{H}_0 = \mathsf{Id}$, and update it so as to satisfy the *(inverse) secant condition*

$$\mathbf{x}^{k+1} - \mathbf{x}^k = \mathsf{H}_{k+1}\left[\mathcal{R}_\gamma(\mathbf{x}^{k+1}) - \mathcal{R}_\gamma(\mathbf{x}^k)\right]. \tag{15}$$

This can be achieved via the (modified) Broyden method in which case the resulting sequence of H_k satisfies the so-called *Dennis-Moré condition* [56, Thm. 2.2] which ensures superlinear convergence of the iterates $\mathbf{x}^k$.

Theorem 5 ([55, Thm. III.5]). *Suppose that in Algorithm 3 the iterates $\mathbf{x}^k$ converge to a* strong local minimum [2] *$\mathbf{x}^\star$ of φ, around which $\nabla^2 f$ exists and is strictly continuous, and at which $\mathcal{R}_\gamma$ is strictly differentiable. If the sequence of H_k satisfies the Dennis-Moré condition [56, Thm. 2.2] then $\tau_k = 1$ is eventually always accepted in step 5, and the convergence is superlinear.*

See [23, Sec. 4.4] for assumptions under which $\mathcal{R}_\gamma$ is strictly differentiable.

Using full quasi-Newton updates requires computing and storing n^2 coefficients at every iteration of Algorithm 3, where n is the problem dimension. This is of course impractical for n larger than a few hundreds. Therefore, limited-memory methods such as L-BFGS can be used: this computes directions $\mathbf{d}^k$ using $O(n)$ operations [43], and is thus well suited for large-scale problems. Algorithm 4 illustrates how the L-BFGS method can be used in the context of

[2] We say that $\mathbf{x}^\star$ is a strong local minimum of φ if for some $\alpha > 0$, $\alpha\|\mathbf{x}-\mathbf{x}^\star\|^2 \leq \varphi(\mathbf{x})-\varphi(\mathbf{x}^\star)$ for all $\mathbf{x}$ sufficiently close to $\mathbf{x}^\star$.

Algorithm 4 L-BFGS two-loop recursion with memory M

```
Set for $i = k - M, \dots, k - 1$ $\begin{cases} \mathbf{s}^i = \mathbf{x}^{i+1} - \mathbf{x}^i \\ \mathbf{w}^i = \mathcal{R}_\gamma(\mathbf{x}^{i+1}) - \mathcal{R}_\gamma(\mathbf{x}^i) \\ \rho_i = \langle \mathbf{s}^i, \mathbf{w}^i \rangle \end{cases}$
Set $H = \rho_{k-1} / \langle \mathbf{w}^{k-1}, \mathbf{w}^{k-1} \rangle$, $\mathbf{d}^k = -\mathcal{R}_\gamma(\mathbf{x}^k)$
for $i = k - 1, \dots, k - M$ do
    $\alpha_i \leftarrow \langle \mathbf{s}^i, \mathbf{d}^k \rangle / \rho_i$
    $\mathbf{d}^k \leftarrow \mathbf{d}^k - \alpha_i \mathbf{w}^i$
$\mathbf{d}^k \leftarrow H \mathbf{d}^k$
for $i = k - M, k - M + 1, \dots, k - 1$ do
    $\beta_i \leftarrow \langle \mathbf{w}^i, \mathbf{d}^k \rangle / \rho_i$
    $\mathbf{d}^k \leftarrow \mathbf{d}^k + (\alpha_i - \beta_i) \mathbf{s}^i$
```

Algorithm 3 to compute directions: at each iteration, the M most recent pairs of vectors $\mathbf{s}^i = \mathbf{x}^{i+1} - \mathbf{x}^i$ and $\mathbf{w}^i = \mathcal{R}_\gamma(\mathbf{x}^{i+1}) - \mathcal{R}_\gamma(\mathbf{x}^i)$ are collected, and are used to compute the product $\mathsf{H}_k \mathcal{R}_\gamma(\mathbf{x}^k)$ implicitly (*i.e.*, without ever storing the full operator H_k in memory) for an operator H_k that approximately satisfies (15).

In Tables 3 and 4 comparisons between the different proximal gradient algorithms are shown. In most of the cases, the PANOC algorithm outperforms the other proximal gradient algorithms. It is worth noticing that its performance is sometimes comparable to the one of the FPG algorithm: Example 5 is a case when this is particularly evident. Although PANOC requires less iterations than FPG, as Table 3 shows, these are more expensive as they perform the additional backtracking line-search procedure. Example 5, which treats the an application of image processing, actually requires a low tolerance to achieve a satisfactory solution. It is in these particular cases, that (fast) PG becomes very competitive with PANOC: this is quite evident also in Figure 1 as it can be seen that for low accuracies of the solution the performance of FPG and PANOC is very similar. Of course, these observations are problem-dependent, and one should always verify empirically which algorithm performs better in the specific application.

When applied to the dual of convex problems, Algorithm 3 results in an extension of the so-called alternating minimization method (AMA) [57, 52].

Finally, it is worth mentioning that *semismooth Newton directions* can be employed in Algorithm 3, see [58]: these are appealing since, for many choices of function g, computing the line-search direction amounts to the solution of very sparse linear systems, see also [59] for examples.

4. Matrix-free optimization

In all of the algorithms described in Section 3 the gradient of f must be computed at every iteration. This operation is therefore fundamental as it can dramatically affect the overall performance of proximal gradient algorithms.

		PG	FPG	PANOC
DNN classifier (Ex. 3) $n = 73$, $\epsilon = 10^{-4}$ nonconvex	t	131.8	n/a	7.6
	k	50000	n/a	1370
Robust PCA (Ex. 4) $n = 3225600$, $\epsilon = 10^{-4}$ nonconvex	t	92.8	n/a	38.5
	k	697	n/a	81
Total variation (Ex. 5) $n = 524288$, $\epsilon = 10^{-3}$ convex	t	8.2	4.3	11.2
	k	582	278	259
Audio de-clipping (Ex. 6) $n = 2048$, $\epsilon = 10^{-5}$ nonconvex	t	368.5	n/a	66.2
	k	8908	n/a	732

Table 3: Table comparing the time t (in s) and the number of iterations k (mean per frame for Example 6) needed to solve the different examples using proximal gradient algorithms. The value n indicates the number of optimization variables of each problem and $\epsilon = \|\mathcal{R}_\gamma(\mathbf{x}^k)\|_\infty/\gamma$ the stopping criteria tolerance.

Consider the cost functions of Examples 1 and 2: in both cases the data fidelity function f consists of the composition of the squared norm with a linear mapping A. These linear mappings need to be evaluated numerically by means of a specified algorithm. A simple and very versatile algorithm that works for both examples, consists of performing a *matrix-vector product.* The function f can then be written as

$$f(\mathbf{x}) = \frac{1}{2}\|\mathbf{A}\mathbf{x} - \mathbf{y}\|^2, \tag{16}$$

i.e., the cost function of a linear least squares problem. In Example 1, $\mathbf{A} \in \mathbb{R}^{m\times n}$ would be a Toeplitz matrix whose columns contain shifted versions of the FIR $\mathbf{h}$. Instead, in Example 2, $\mathbf{A}$ would correspond to a complex-valued matrix containing complex exponentials resulting from the inverse DFT. By applying the *chain rule* the gradient of (16) at the iterate $\mathbf{x}^k$ reads:

$$\nabla f(\mathbf{x}^k) = \mathbf{A}^{\mathsf{H}}\left(\mathbf{A}\mathbf{x}^k - \mathbf{y}\right), \tag{17}$$

where $^{\mathsf{H}}$ indicates the conjugate-transpose operation. If $\mathbf{A}$ is a dense matrix, evaluating (17) then takes two matrix-vector products each one having a complexity $O\left(mn\right)$. Moreover $\mathbf{A}$ must be stored and this occupies $O(mn)$ bytes despite the redundancy of the information it carries.

Using a matrix-vector product as an algorithm to perform discrete convolution or an inverse DFT is actually not the best choice: it is well known that there exist a variety of algorithms capable of outperforming the matrix-vector product algorithm. For example, discrete convolution can be performed with a complexity of $O\left(n \log n\right)$ by transforming the signals $\mathbf{h}$ and $\mathbf{x}^k$ into the frequency domain, multiplying them and transforming the result back into the time domain. The memory requirements are also lower since now only the

	Using Matrices			Matrix-Free		
	PG	FPG	PANOC	PG	FPG	PANOC
Sparse Deconvolution (Ex. 1)	1174	520	360	253	127	89
Line Spectral Estimation (Ex. 2)	2773	1089	237	1215	489	108

Table 4: Table comparing the time (in ms) that different PG algorithms employ to solve the optimization problems of Examples 1 and 2 using matrices or matrix-free optimization.

$O(n)$ bytes of the FIR need to be stored. When $\mathbf{A}$ represents convolution, its conjugate-transpose matrix-vector product appearing in (17), corresponds to a cross-correlation: this operation can also be evaluated with the very same complexity and memory requirements as the convolution. Cross-correlation is in fact the *adjoint mapping* of convolution [60].

In general, given a linear mapping $\mathsf{A} \in \mathscr{L}(\mathtt{D},\mathtt{C})$ the adjoint mapping is its uniquely associated linear mapping $\mathsf{A}^* \in \mathscr{L}(\mathtt{C},\mathtt{D})$. In this context A is often called *forward mapping*. Formally, the adjoint mapping is defined by the equivalence of these inner products

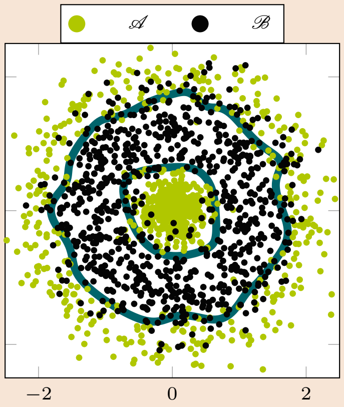
$$\langle \mathsf{A}\mathbf{x}, \mathbf{y}\rangle_{\mathtt{C}} = \langle \mathbf{x}, \mathsf{A}^*\mathbf{y}\rangle_{\mathtt{D}} \ \forall\, \mathbf{x} \in \mathtt{D},\ \forall\, \mathbf{y} \in \mathtt{C}. \tag{18}$$

The adjoint mapping A^* generalizes the conjugate-transpose operation and like A, it can be evaluated using different algorithms. It is now possible to define the *forward-adjoint oracles (FAOs)* of a linear mapping A: these oracles consist of two specific black-box algorithms that are used to compute the forward mapping A and its associated adjoint mapping A^* respectively. Avoiding the use of matrices in favor of FAOs for the evaluation of the linear mappings leads to *matrix-free optimization.* Table 4 shows the improvements in terms of computational time with respect to using matrices: clearly, for the aforementioned reasons, solving the optimization problems of Examples 1 and 2 using matrix-free optimization is substantially faster.

4.1. Directed acyclic graphs

Being generalizations of matrices, linear mappings share many features with them. For example it is possible to horizontally or vertically concatenate linear mappings that share their codomain or domain respectively. Additionally, it is possible to compose linear mappings, *e.g.*, $\mathsf{A} \in \mathscr{L}(\mathtt{K},\mathtt{C})$ can be composed with $\mathsf{B} \in \mathscr{L}(\mathtt{D},\mathtt{K})$ to construct $\mathsf{AB} \in \mathscr{L}(\mathtt{D},\mathtt{C})$. Although conceptually equivalent, these and other *calculus rules* are implemented in a substantially different fashion to what is typically done with matrices. With matrices the application of a calculus rule results in a new and independent matrix which is then used to evaluate the forward and adjoint mappings through matrix-vector products. On the contrary, combinations of linear mappings that are evaluated through FAOs constitute a directed acyclic graph (DAG) which preserves the structure of the calculus rules involved. Each node of these graphs is associated with a

Example 3 Deep Neural Network Classifier



`StructuredOptimization` **code snippet:**

```
#W1,W2,W3,b1,b2,b3 are variables
#S1,S2,S3  are sigmoid operators

L1= S1*(W1* D.+b1) #input layer
L2= S2*(W2*L1.+b2) #inner layer
y = S3*(W3*L2.+b3) #output layer
    # regularization
reg = lambda1*norm(W1,1)+
      lambda2*norm(W2,1)+
      lambda3*norm(W3,1)
@minimize crossentropy(y,yt)+reg
```

A deep neural network is a relatively simple model that, by composing multiple linear transformations and nonlinear *activation functions*, allows obtaining highly nonlinear mappings that can be used for classification or regression tasks [41]. This is achieved by *training* the network, *i.e.*, by finding the optimal parameters (the coefficients of the linear transformations) with respect to a *loss* function and some training data, and amounts to solving a highly nonconvex optimization problem. In this example, three layers are combined to perform classification of data points into two sets $\mathscr{A}$ and $\mathscr{B}$ depicted in the figure above. The following optimization problem is solved to train the network:

$$\begin{aligned}&\underset{\mathbf{W}_1,\mathbf{W}_2,\mathbf{W}_3,b_1,b_2,b_3}{\text{minimize}} \; -\sum_{n=1}^{N}\overbrace{(\tilde{y}_n\log(y_n)+(1-\tilde{y}_n)\log(1-y_n))}^{f}+\sum_{k=1}^{3}\overbrace{\lambda_k\|\text{vec}(\mathbf{W}_k)\|_1}^{g}\\ &\text{subject to} \quad \mathbf{y}=\mathcal{S}_3(\mathbf{W}_3\mathbf{L}_2+b_3),\; \mathbf{L}_2=\mathcal{S}_2(\mathbf{W}_2\mathbf{L}_1+b_2),\; \mathbf{L}_1=\mathcal{S}_1(\mathbf{W}_1\mathbf{D}+b_1).\end{aligned} \tag{19}$$

Here $\mathbf{D}\in\mathbb{R}^{N\times 2}$ are the training data points with binary labels (0 for $\mathscr{A}$ and 1 for $\mathscr{B}$) stored in $\tilde{\mathbf{y}}$. $\mathbf{W}_i$ and b_i are the weights and biases of the i-th layer which combination outputs $\mathbf{y}\in\mathbb{R}^N$. This output is fitted through the usage of a cross-entropy loss function f [41] to the labels $\tilde{\mathbf{y}}$. The nonlinear mappings $\mathcal{S}_i$ are sigmoid functions modeling the activations of the neurons. A regularization function g is added to simultaneously prevent over-fitting while enforcing the weights to be sparse matrices. Contour lines show the classifier obtained after the training.

particular mapping and the calculus rules are applied according to the way these nodes are connected. It is actually convenient to use these DAGs to evaluate the cost function together with its gradient, a strategy that is known as *automatic differentiation* in numerical analysis [61] and *back-propagation* in machine learning [41].

Figure 4a illustrates these concepts using a simple example of composition of two linear mappings. Here the gradient of the function $f(\mathbf{x}) = \tilde{f}(\mathsf{AB}\mathbf{x} - \mathbf{y})$ is needed. The gradient of this function at $\mathbf{x}^k$ reads:

$$\nabla f(\mathbf{x}^k) = \mathsf{B}^* \mathsf{A}^* \nabla \tilde{f}(\mathsf{AB}\mathbf{x}^k - \mathbf{y}), \tag{20}$$

and can be conveniently computed alongside the evaluation of $f(\mathbf{x}^k)$. The iterate $\mathbf{x}^k$ is initially "sent" through what is referred to here as the *forward DAG*: here the linear mappings B and A are applied in series to $\mathbf{x}^k$ using the corresponding forward oracles. After this, the *residual* $\mathbf{r}^k = \mathsf{AB}\mathbf{x}^k - \mathbf{y}$ can be computed. This can be readily used not only to compute $f(\mathbf{x}^k)$, but also to obtain the gradient: in fact after applying the gradient of $\tilde{f}$ to $\mathbf{r}^k$ the result is "sent back" through the *backward DAG*. This differs form the forward DAG since now the adjoint oracles of A and B are applied in a reversed order.

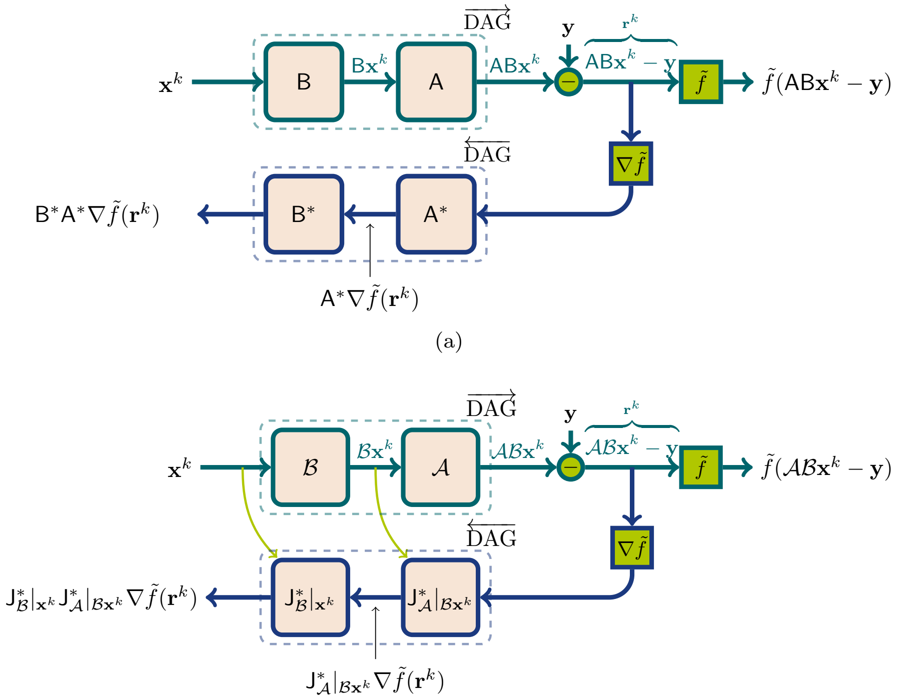
Similarly to Figure 4, in Example 2 two linear mappings were composed. The first linear mapping $\mathsf{F}_i \in \mathscr{L}(\mathbb{C}^{sn}, \mathbb{R}^{sn})$ consists of an inverse DFT. Its forward oracle is an inverse FFT while its adjoint oracle is a non-normalized FFT. The linear mapping $\mathsf{S} \in \mathscr{L}(\mathbb{R}^{sn}, \mathbb{R}^{n})$ converts the high resolution signal into a low resolution one. Its FAOs are extremely simple algorithms: the forward oracle selects the first n elements while the adjoint oracle performs the opposite, zero-padding its input.

So far only linear mappings were considered, but smooth nonlinear mappings can be combined as well using FAOs and DAGs. In fact when nonlinear mappings appear in f, this function and its gradient can be evaluated using an analogous strategy to the one described earlier. The main difference lies in the fact that the adjoint operator of a nonlinear mapping does not exist. However a nonlinear mapping $\mathcal{A} : \mathtt{D} \to \mathtt{C}$ can be linearized by differentiating it and obtaining a *linear* mapping called *Jacobian mapping* for which here the following notation is used: $\mathsf{J}_{\mathcal{A}}|_{\mathbf{x}^k} \in \mathscr{L}(\mathtt{D}, \mathtt{C})$ where $|_{\mathbf{x}^k}$ is used to indicate the point of the linearization. Using again the same example, this time with nonlinear mappings, the chain rule is again applied to $f(\mathbf{x}) = \tilde{f}(\mathcal{AB}\mathbf{x} - \mathbf{y})$:

$$\nabla f(\mathbf{x}^k) = \mathsf{J}^*_{\mathcal{B}}|_{\mathbf{x}^k} \mathsf{J}^*_{\mathcal{A}}|_{\mathcal{B}\mathbf{x}^k} \nabla \tilde{f}(\mathcal{AB}\mathbf{x}^k - \mathbf{y}). \tag{21}$$

Here, and visually in Figure 4b, it can be seen that the main difference with (20) and Figure 4a, is that the adjoint mappings are replaced by the adjoint Jacobian mappings of $\mathcal{A}$ and $\mathcal{B}$ linearized at $\mathcal{B}\mathbf{x}^k$ and $\mathbf{x}^k$ respectively. These quantities are already available since they are computed during the forward DAG evaluation: if these are stored during this phase they can be used later when sending back the residual to compute (21) as the small arrows in Figure 4b show.

These intuitive examples represent only one of the various calculus rules that can be use to combine mappings. Many other calculus rules can be applied to construct models with much more complex DAGs. Table 5 shows the most important calculus rules used to create models. As it can be seen, the *horizontal concatenation* rule is very similar to the horizontal concatenation of two matrices. However this creates a forward DAG that has two inputs $\mathbf{x}_1^k$ and $\mathbf{x}_2^k$ that are processed in parallel using two forward oracles whose result is then summed.


$\overrightarrow{\mathrm{DAG}}$
$\mathbf{y}$
$\mathbf{r}^k$
$\mathbf{x}^k$
B
Bx^k
A
ABx^k
$\mathsf{ABx}^k - \mathbf{y}$
$\tilde{f}$
$\tilde{f}(\mathsf{ABx}^k - \mathbf{y})$
$\overleftarrow{\mathrm{DAG}}$
$\nabla\tilde{f}$
$\mathsf{B}^*\mathsf{A}^*\nabla\tilde{f}(\mathbf{r}^k)$
B^*
A^*
$\mathsf{A}^*\nabla\tilde{f}(\mathbf{r}^k)$
(a)
$\overrightarrow{\mathrm{DAG}}$
$\mathbf{y}$
$\mathbf{r}^k$
$\mathbf{x}^k$
$\mathcal{B}$
$\mathcal{B}\mathbf{x}^k$
$\mathcal{A}$
$\mathcal{AB}\mathbf{x}^k$
$\mathcal{AB}\mathbf{x}^k - \mathbf{y}$
$\tilde{f}$
$\tilde{f}(\mathcal{AB}\mathbf{x}^k - \mathbf{y})$
$\overleftarrow{\mathrm{DAG}}$
$\nabla\tilde{f}$
$\mathsf{J}^*_{\mathcal{B}}|_{\mathbf{x}^k}\mathsf{J}^*_{\mathcal{A}}|_{\mathcal{B}\mathbf{x}^k}\nabla\tilde{f}(\mathbf{r}^k)$
$\mathsf{J}^*_{\mathcal{B}}|_{\mathbf{x}^k}$
$\mathsf{J}^*_{\mathcal{A}}|_{\mathcal{B}\mathbf{x}^k}$
$\mathsf{J}^*_{\mathcal{A}}|_{\mathcal{B}\mathbf{x}^k}\nabla\tilde{f}(\mathbf{r}^k)$

(b)

Figure 4: Forward and backward DAGs used to evaluate the gradient of a function composed with (a) linear mappings or (b) nonlinear mappings.

This rule can also be used to simply add up different optimization variables by setting the mappings to be identity, like in the least squares function of Example 4. By inspecting the chain rule it is possible to see how its backward DAG would look: this would be a reversed version of the forward DAG, having a single input $\nabla\tilde{f}(\mathbf{r}^k)$ and two outputs where the respective adjoint (Jacobian) mappings would be applied.

The final calculus rule of Table 5, called *output multiplication* has a similar forward DAG to the horizontal concatenation, with the summation being substituted with a multiplication. The resulting mapping, even when linear mappings are used, is always nonlinear. Due to this nonlinear behavior its backward DAG is more difficult to picture but still the very same concepts are applied.

Example 3 shows an example of a deep neural network (DNN), a type of nonlinear model which is extensively used in machine learning together with back-propagation. Recently, DNNs have been succesfully used in many areas of signal processing as well [62, 41]. In Example 3, many of the calculus rules of Table 5 are used to model a DNN that is trained to perform a nonlinear classification task. DNNs model the behavior of brain neurons. The neurons are divided in sequential groups called layers: these can be distinguished between

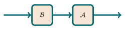
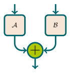
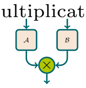
Rule	Input Mappings	Output Mapping	Chain Rule
Composition	$\mathsf{A} \in \mathscr{L}(\mathtt{K},\mathtt{C})$, $\mathsf{B} \in \mathscr{L}(\mathtt{D},\mathtt{K})$	$\mathsf{AB} \in \mathscr{L}(\mathtt{D},\mathtt{C})$	$\nabla(f(\mathsf{AB}\mathbf{x}^k)) = \mathsf{B}^*\mathsf{A}^*\nabla f(\mathsf{AB}\mathbf{x}^k)$
	$\mathcal{A} : \mathtt{K} \to \mathtt{C}$, $\mathcal{B} : \mathtt{D} \to \mathtt{K}$	$\mathcal{AB} : \mathtt{D} \to \mathtt{C}$	$\nabla(f(\mathcal{AB}\mathbf{x}^k)) = \mathsf{J}^*_{\mathcal{B}}\lvert_{\mathbf{x}^k}\mathsf{J}^*_{\mathcal{A}}\lvert_{\mathcal{B}\mathbf{x}^k}\nabla f(\mathcal{AB}\mathbf{x}^k)$
Horizontal Concatenation	$\mathsf{A} \in \mathscr{L}(\mathtt{D},\mathtt{C})$, $\mathsf{B} \in \mathscr{L}(\mathtt{K},\mathtt{C})$	$[\mathsf{A},\mathsf{B}] \in$ $\mathscr{L}(\mathtt{D}\times\mathtt{K},\mathtt{C})$	$\nabla(f(\mathsf{A}\mathbf{x}_1^k + \mathsf{B}\mathbf{x}_2^k)) = [\mathsf{A}^*\nabla f(\mathbf{r}^k), \mathsf{B}^*\nabla f(\mathbf{r}^k)]$
	$\mathcal{A} : \mathtt{D} \to \mathtt{C}$, $\mathcal{B} : \mathtt{K} \to \mathtt{C}$	$[\mathcal{A},\mathcal{B}]$: $\mathtt{D}\times\mathtt{K} \to \mathtt{C}$	$\nabla(f(\mathcal{A}\mathbf{x}_1^k + \mathcal{B}\mathbf{x}_2^k)) =$ $[\mathsf{J}^*_{\mathcal{A}}\lvert_{\mathbf{x}_1^k}\nabla f(\mathbf{r}^k), \mathsf{J}^*_{\mathcal{B}}\lvert_{\mathbf{x}_2^k}\nabla f(\mathbf{r}^k)]$
Output Multiplication	$\mathsf{A} \in \mathscr{L}(\mathtt{D},\mathtt{E})$, $\mathsf{B} \in \mathscr{L}(\mathtt{F},\mathtt{G})$	$\mathsf{A}(\cdot)\mathsf{B}(\cdot)$: $\mathtt{D}\times\mathtt{F} \to \mathtt{C}$	$\nabla(f(\mathsf{A}\mathbf{X}_1^k\mathsf{B}\mathbf{X}_2^k)) =$ $[\mathsf{A}^*\nabla f(\mathbf{R}^k)(\mathsf{B}\mathbf{X}_2^k)^{\mathsf{H}}, \mathsf{B}^*(\mathsf{A}\mathbf{X}_1^k)^{\mathsf{H}}\nabla f(\mathbf{R}^k)]$
	$\mathcal{A} : \mathtt{D} \to \mathtt{E}$, $\mathcal{B} : \mathtt{F} \to \mathtt{G}$	$\mathcal{A}(\cdot)\mathcal{B}(\cdot)$: $\mathtt{D}\times\mathtt{F} \to \mathtt{C}$	$\nabla(f(\mathcal{A}\mathbf{X}_1^k\mathcal{B}\mathbf{X}_2^k)) =$ $[\mathsf{J}^*_{\mathcal{A}}\lvert_{\mathbf{X}_1^k}\nabla f(\mathbf{R}^k)(\mathcal{B}\mathbf{X}_2^k)^{\mathsf{H}}, \mathsf{J}^*_{\mathcal{B}}\lvert_{\mathbf{X}_2^k}(\mathcal{A}\mathbf{X}_1^k)^{\mathsf{H}}\nabla f(\mathbf{R}^k)]$

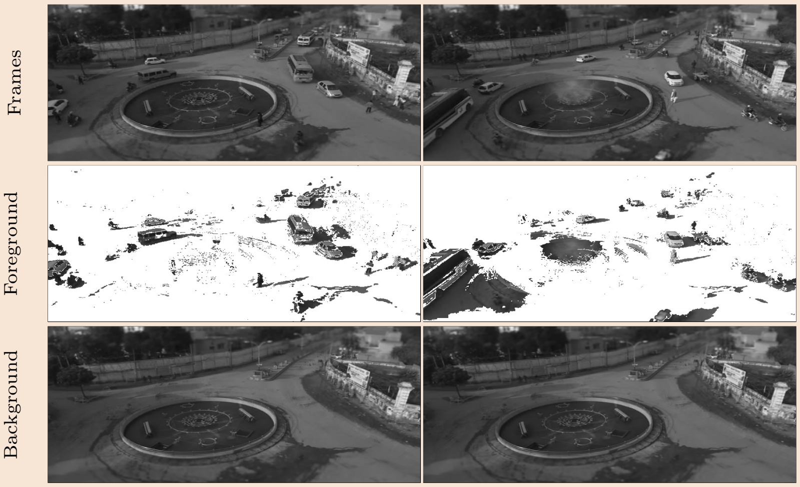
Table 5: Table showing different calculus rules to combine linear and nonlinear mappings. Here $\mathbf{r}^k$ and $\mathbf{R}^k$ indicate the residual inside the parentheses of f. For the output multiplication rule if $\mathtt{E} = \mathbb{R}^{n\times l}$ and $\mathtt{G} = \mathbb{R}^{l\times m}$ than $\mathtt{C} = \mathbb{R}^{n\times m}$.

output, input and inner layers. Specifically, in Example 3 only one inner layer is present. Each neuron belonging to a layer is connected with all of the other neurons of the neighbor layers. Neurons of the output layer are connected to $\mathbf{y}$, while those of the input layer are connected to the input, which in this problem is given and represented by $\mathbf{D}$. The connections between neurons are modeled by the matrices $\mathbf{W}_i$ which contain *weights* representing the importance of each connection and are estimated by solving (19). Additionally every layer has a bias term b_i that must also be estimated. Neurons can either be active or inactive and this behavior is modeled by the nonlinear mappings $\mathcal{S}_i$ which consists of sigmoid functions. The DAG of this DNN is not reported here for the sake of brevity, but the constraints of (19) well describe it. The addition of the bias term performs a horizontal concatenation while the operation $\mathbf{W}_i\mathbf{L}_i$ represents an output multiplication, which *connects* the different layers.

5. General problem formulation

It was already mentioned that the problem formulation (4) with its cost function $f(\mathbf{x}) + g(\mathbf{x})$ includes a wide variety of optimization problems. However, most of the times problems are formulated without having in mind this particular structure: typically there are M optimization variables representing different signals to estimate which can appear in multiple functions and constraints. This leads to a more general problem formulation which, after converting the constraints into indicator functions, can be summarized as follows

Example 4 Video background removal



The frames of a video can be viewed as a superposition of a moving foreground to a steady background. Splitting the background form the foreground can be a difficult task due to the continuous changes happening in different areas of the frames. The following optimization problem can be posed to deal with such task:

$$\begin{aligned} \underset{\mathbf{L},\mathbf{S}}{\text{minimize}} \quad & \frac{1}{2}\|\mathbf{L}+\mathbf{S}-\mathbf{Y}\|^2 + \lambda\|\text{vec}(\mathbf{S})\|_1 \\ \text{subject to} \quad & \text{rank}(\mathbf{L}) \leq 1. \end{aligned}$$

```
StructuredOptimization:
@minimize ls(L+S-Y)+
          lambda*norm(S,1)
st rank(L) <= 1
```

Here $\mathbf{Y} \in \mathbb{R}^{nm \times l}$ consists of a matrix in which the l-th column contains the pixel values of the vectorized l-th frame with dimensions $n \times m$. The optimization problem, also known as robust PCA, decomposes $\mathbf{Y}$ into a sparse matrix $\mathbf{S}$, representing the foreground changes and a rank-1 matrix $\mathbf{L}$ consisting of the constant background, whose columns are linearly dependent.

[63, 64]:

$$\underset{\mathbf{x}_1,\ldots,\mathbf{x}_M}{\text{minimize}} \sum_{i=1}^{N} h_i\left(\sum_{j=1}^{M} \mathsf{A}_{i,j}\mathbf{x}_j\right), \tag{22}$$

where the N functions $h_i : \mathtt{C}_1 \times \cdots \times \mathtt{C}_M \to \overline{\mathbb{R}}$ are composed with linear mappings $\mathsf{A}_{i,j} \;\; : \mathtt{D}_j \to \mathtt{C}_i$. Notice that here the eventual presence of nonlinear mappings is included in h_i.

In order to apply the framework described in the previous sections, (22) must be re-structured into (4) by splitting the different h_i into two groups: this means, one must appropriately partition the set of indexes $\{1,\dots,N\}$ into two subsets I_f, I_g, such that $\{1,\dots,N\} = I_f \cup I_g$ and $I_f \cap I_g = \emptyset$, and set

$$f(\mathbf{x}_1,\dots,\mathbf{x}_M) = \sum_{i \in I_f} h_i \left(\sum_{j=1}^{M} \mathsf{A}_{i,j} \mathbf{x}_j \right), \tag{23}$$

$$g(\mathbf{x}_1,\dots,\mathbf{x}_M) = \sum_{i \in I_g} h_i \left(\sum_{j=1}^{M} \mathsf{A}_{i,j} \mathbf{x}_j \right). \tag{24}$$

In order for f in (23) to be smooth, clearly one must have that h_i is smooth for all $i \in I_f$. When this is the case, denoting $\mathbf{r}_i = \sum_{j=1}^{M} \mathsf{A}_{i,j}\mathbf{x}_j$, one has that

$$\nabla f(\mathbf{x}_1,\dots,\mathbf{x}_M) = \left[\left(\sum_{i \in I_f} \mathsf{A}_{i,1}^* \nabla h_i(\mathbf{r}_i) \right)^{\mathsf{T}}, \dots, \left(\sum_{i \in I_f} \mathsf{A}_{i,M}^* \nabla h_i(\mathbf{r}_i) \right)^{\mathsf{T}} \right]^{\mathsf{T}}. \tag{25}$$

On the other hand, g in (24) should have an efficiently computable proximal mapping. This happens, for example, if all of the following conditions are met [64]:

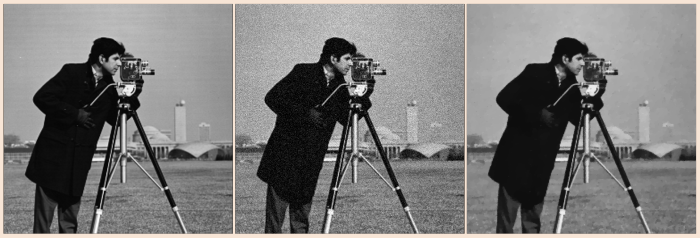
- for all $i \in I_g$, function h_i has an efficient proximal mapping;
- for all $i \in I_g$ and $j \in \{1,\dots,M\}$, mapping $\mathsf{A}_{i,j}$ satisfies $\mathsf{A}_{i,j}\mathsf{A}_{i,j}^* = \mu_{i,j}\mathsf{Id}$, where $\mu_{i,j} \geq 0$.
- for all $j \in \{1,\dots,M\}$, the cardinality $\operatorname{card}\{i \mid \mathsf{A}_{i,j} \neq 0\} = 1$.

These rules ensure that the separable sum and precomposition properties of proximal mappings, cf. Table 2, are applicable yielding an efficient proximal mapping for g.

Let the cost function of Example 4 be a test case to check these rules. This example treats the problem of robust PCA [65, 40] which has practical applications in surveillance, video restoration and image shadow removal. After converting the rank constraint on $\mathbf{L}$ into an indicator function $\delta_{\mathscr{S}}$, it is easy to see that $g(\mathbf{S},\mathbf{L}) = \lambda \|\mathrm{vec}(\mathbf{S})\|_1 + \delta_{\mathscr{S}}(\mathbf{L})$ can be written in terms of (24). Clearly g consists of a separable sum of functions that have efficient proximal mappings that can be viewed in Table 1. Additionally, g satisfies the conditions above and has therefore an efficiently computable proximal mapping: all linear mappings not equal to 0 are identity and satisfy $\mathsf{A}\mathsf{A}^* = \mu\mathsf{Id}$. Moreover, for every variable only one linear mapping is not equal to 0. On the contrary, if an additional constraint on $\mathbf{S}$ or $\mathbf{L}$ appeared or if another nonsmooth function was present in the cost function, *e.g.*, $\|\mathrm{vec}(\mathbf{L})\|_1$, the proximal mapping of g would have been difficult to compute.

5.1. Duality and smoothing

Example 5 Total variation de-noising



ground truth $\mathbf{X}_{gt}$ noisy $\mathbf{Y}$ denoised $\mathbf{X}^\star$

StructuredOptimization

```
V = Variation(size(Y)); U = Variable(size(V,1)...)
@minimize ls(-V'*U+Y) + conj(lambda*norm(U,2,1,2))
X = Y-V'*(~U)
```

Total variation de-noising seeks to remove noise from a noisy image whose pixels are stored in the matrix $\mathbf{Y} \in \mathbb{R}^{n \times m}$. This technique relies on the assumption that neighbor pixels of the sought uncorrupted image $\mathbf{X}^\star$ should be similar, namely that $\sqrt{|x^\star_{i+1,j} - x^\star_{i,j}|^2 + |x^\star_{i,j+1} - x^\star_{i,j}|^2}$ should be small, where $x^\star_{i,j}$ is the (i,j)-th component of $\mathbf{X}^\star$. This enforces the image to have sharp edges, namely a sparse gradient. The following optimization problem can be formulated:

$$\text{(a)}\ \underset{\mathbf{X}}{\text{minimize}}\ \frac{1}{2}\|\mathbf{X} - \mathbf{Y}\|^2 + \lambda \|\mathsf{V}\mathbf{X}\|_{2,1}\ \text{(b)}\ \underset{\mathbf{U}}{\text{minimize}}\ \frac{1}{2}\| - \mathsf{V}^*\mathbf{U} + \mathbf{Y}\|^2 + g^*(\mathbf{U})$$

Here the operator $\mathsf{V} \in \mathscr{L}(\mathbb{R}^{n \times m}, \mathbb{R}^{nm \times 2})$ maps $\mathbf{X}$ into a matrix having in its j-th column the vectorized forward finite difference gradient over the j-th direction. The operator V appears in the nonsmooth part of the cost function $g(\cdot) = \lambda\|\cdot\|_{2,1}$ and leads to a non-trivial proximal operator. Here the mixed norm $\|\cdot\|_{2,1}$ consists of the sum of the l_2-norm of the rows of $\mathsf{V}\mathbf{X}$. Using Fenchel's duality theorem it is possible to convert the problem into (b) which can instead be solved efficiency using proximal gradient algorithms.

Sometimes the rules that ensure that g has an efficiently computable proximal mapping are too stringent. However, even when these rules are not satisfied there are cases where it is still possible to apply proximal gradient algorithms. Consider the following problem:

$$\underset{\mathbf{x}}{\text{minimize}}\ f(\mathbf{x}) + g(\mathsf{A}\mathbf{x}) \tag{26}$$

where f and g are convex and A is a general linear mapping (for example, it is not a tight frame hence $g \circ \mathsf{A}$ does not have an efficient proximal mapping, cf. Section 3.1). If f is *strongly convex*, then the *dual problem* of (26) has a structure that well suits proximal gradient algorithms, as it will be now shown.

The dual problem can be derived through the usage of the *convex conjugate* functions which are defined as:

$$f^*(\mathbf{u}) = \sup_{\mathbf{x}} \{ \langle \mathbf{x}, \mathbf{u} \rangle - f(\mathbf{x}) \}. \tag{27}$$

Convex conjugation describes f in terms of dual variables $\mathbf{u}$: this conjugation has many properties that often can simplify optimization problems, see [66, 25] for an exhaustive review. Problem (26) can be expressed in terms of convex conjugate functions through its associated Fenchel dual problem [5]:

$$\underset{\mathbf{u}}{\text{minimize}} \;\; f^*(-\mathsf{A}^*\mathbf{u}) + g^*(\mathbf{u}). \tag{28}$$

Solving the Fenchel dual problem may be particularly desirable when $\mathsf{A} \in \mathscr{L}(\mathbb{R}^m, \mathbb{R}^n)$ and $n \gg m$, in which case the dual variables $\mathbf{u} \in \mathbb{R}^m$ are significantly less than the original ones $\mathbf{x} \in \mathbb{R}^n$. Furthermore, two properties of convex conjugate functions allow for (28) to be solved using proximal gradient algorithms. Firstly, proximal mappings and convex conjugate functions are linked by the *Moreau decomposition*:

$$\mathbf{x} = \operatorname{prox}_{\gamma g^*}(\mathbf{x}) + \gamma \operatorname{prox}_{(1/\gamma) g}(\mathbf{x}/\gamma). \tag{29}$$

This shows that whenever the proximal mapping of g is efficiently computable, so is that of g^*. Secondly, if f is strongly convex then its convex conjugate f^* has a Lipschitz gradient [67, Lemma 3.2]. This also implies that any solution of (28) can be converted back to the one of the original problem through [25]:

$$\mathbf{x}^\star = \nabla f^*(-\mathsf{A}^*\mathbf{u}^\star). \tag{30}$$

Therefore, under these assumptions it is possible to solve (28) using proximal gradient algorithms of Section 3.2. When this is done, the (fast) PG algorithm results in what is also known as (fast) *alternating minimization algorithm* (AMA) [57, 67].

Example 5 treats the classical image processing application of de-noising a digital image. Here in the original problem a linear operator appears in a nonsmooth function preventing its proximal mapping to be efficient. Function f here is the squared norm which is self-dual *i.e.*, $f(\cdot) = \frac{1}{2}\|\cdot\|^2 = f^*(\cdot)$. Hence it is possible to solve the dual problem instead, using proximal gradient algorithms: in the Fenchel dual problem the linear mapping is transferred into the smooth function f^* in terms of its adjoint allowing the usage of an efficient proximal mapping for the nonsmooth function g through (29). Once the dual solution $\mathbf{U}^\star$ is obtained, this can be easily converted back to the one of the original problem through the usage of (30): $\nabla f(\mathbf{X}^\star) = \mathbf{X}^\star - \mathbf{Y} = -\mathsf{V}^*\mathbf{U}^\star$.

Finally, it was assumed that the functions constructing f are differentiable. When this is not the case, proximal gradient algorithms can be still applied by "smoothing" the nonsmooth functions h_i that appear in f by means of the

Moreau envelope [68, 69]:

$$h_i^\beta(\mathbf{x}) = \min_{\mathbf{z}} \left\{ h_i(\mathbf{z}) + \tfrac{1}{2\beta}\|\mathbf{z}-\mathbf{x}\|^2 \right\}. \tag{31}$$

Moreau envelopes possess some very important properties related to optimization: similarly to the FBE, when h_i is convex, the function h_i^β is a real-valued, smooth lower bound to h_i, that shares with h_i its minimum points and values, see [5]. Furthermore, computing the value and gradient of h_i^β essentially requires one evaluation of $\mathrm{prox}_{\beta h_i}$:

$$\nabla h_i^\beta(\mathbf{x}) = \tfrac{1}{\beta}\left(\mathbf{x} - \mathrm{prox}_{\beta h_i}(\mathbf{x})\right). \tag{32}$$

However, using the Moreau envelope has the drawback that one has to fine-tune the parameter β which controls the level of smoothing. This is typically achieved through the usage of a continuation scheme that involves solving the optimization problem multiple times with a decreasing level of β to approach the solution of the original optimization problem with nonsmooth f.

The Moreau envelope can also be used in the dual problem of (26) when f is only convex but not strongly convex, meaning that its convex conjugate f^* is not guaranteed to be smooth. Strong convexity can be obtained in f by adding to it a regularization term:

$$f_\beta(\mathbf{x}) = f(\mathbf{x}) + \tfrac{\beta}{2}\|\mathbf{x}\|^2. \tag{33}$$

Then, the convex conjugate of f_β becomes the Moreau envelope of the convex conjugate of f, *i.e.*, $(f_\beta^*) = (f^*)^\beta$ [5, Prop. 14.1], which is now differentiable and allows to solve the dual problem (28) using proximal gradient algorithms.

6. A high-level modeling language: `StructuredOptimization`

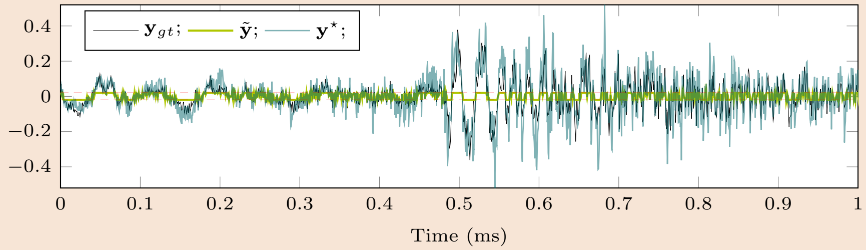
In all of the examples shown in this paper the formulations of the various optimization problems are placed side by side to some code snippets. This code corresponds to an open-source high-level modeling language implemented in Julia.

Julia is a relatively new open-source programming language which was specifically designed for scientific computing and offers high performance often comparable to low-level languages like C. Despite being a young programming language it has a rapidly growing community and already offers many packages in various fields [35].

The proposed high-level modeling language is provided in the software package `StructuredOptimization` [70]: this utilizes a syntax that is very close to the mathematical formulation of an optimization problem. This user-friendly interface acts as a parser to utilize three different packages that implement many of the concepts described in this paper:

- `ProximalOperators` is a library of proximal mappings of functions that are frequently used in signal processing and optimization. These can

Example 6 Audio de-clipping



When recording an audio signal generated from a loud source the microphone can saturate. This results in a *clipped audio signal* which can be severely corrupted with distortion artifacts. The figure above shows a frame of a clipped signal: the red dashed lines represent the saturation level C of the microphone. The samples of the true signal that are above or below these lines are *lost* during the audio recording. What *audio de-clipping* seeks is to recover these samples and remove the audio artifacts. This can be achieved by solving an optimization problem that combines the uncorrupted samples of the audio signal with the knowledge that the signal is sparse when transformed using the DCT.

$$\begin{aligned}
\underset{\mathbf{x},\mathbf{y}}{\text{minimize}} \quad & \frac{1}{2}\|\mathsf{F}_{i,c}\mathbf{x} - \mathbf{y}\|^2, \\
\text{subject to} \quad & \|\mathsf{M}\mathbf{y} - \mathsf{M}\tilde{\mathbf{y}}\| \leq \sqrt{\epsilon} \\
& \mathsf{M}_{+}\mathbf{y} \geq C \\
& \mathsf{M}_{-}\mathbf{y} \leq -C \\
& \|\mathbf{x}\|_0 \leq N
\end{aligned}$$

`StructuredOptimization`:

```
f = ls( idct(x) - y )
for N = 30:30:30*div(Nl,30)
 cstr = (
 norm(M*y-M*yt)<=sqrt(eps),
 Mp*y >= C, Mn*y <= -C,
 norm(x,0) <= N)
 @minimize f st cstr
 if norm(idct(~x)-~y)<=eps
 break end
end
```

Here $\mathbf{y}$ and $\mathbf{x}$ are both optimization variables representing the sought de-clipped signal and its DCT transform respectively. M, $\mathsf{M}_{\pm}$ select the uncorrupted and clipped samples and are used in the first three constraints to keep $\mathbf{y}$ either close to $\tilde{\mathbf{y}}$ at the uncorrupted samples or outside the saturation level respectively. The value N represents the number of active components in the DCT: as the code snippet shows, this value is tuned by solving the optimization problem multiple times by increasing N. As more active components are introduced, the cost function decreases: once its value reaches $\sqrt{\epsilon}$ the solution refinement is stopped.

be transformed and manipulated using the properties described in Section 3.1.

- `AbstractOperators` provides a library of FAOs that can be used to create and evaluate DAGs of linear and nonlinear mappings as described in Section 4. This package also offers a syntax analogue to the one that is typically used with matrices.

- `ProximalAlgorithms` is a library of optimization algorithms that includes the PG algorithm and its enhanced variants described in Sections 3.2 and 3.4.

When a problem is provided to `StructuredOptimization` this is automatically analyzed to check whether it falls within the sets of problems described in Section 5. Firstly, the various functions and constraints, which are conveniently converted into indicator functions, need to be split into the functions f and g. As it was described in Section 5 sometimes multiple splitting configurations are possible: `StructuredOptimization` adopts the simplest strategy possible, splitting the smooth functions form the nonsmooth ones. The nonsmooth functions are then analyzed to verify if the rules described in Section 5 are fulfilled to ensure an efficient proximal mapping of g exists. If this is the case, `StructuredOptimization` then provides the necessary inputs to the algorithms of `ProximalAlgorithms` to efficiently solve the problem.

Example 6 can be used as a showcase of the proposed high-level modeling language. This example treats the recovery of an audio signal corrupted by clipping [71, 72]. This recovery is performed using a weighted overlap-add method, *i.e.*, by splitting the audio signal into overlapping frames of length $n = 2^{10}$ and processing them serially, using an initialization strategy analogue to the one proposed in [73].

The high-level modeling language that `StructuredOptimization` provides is designed to be as much natural as possible. Firstly the optimization variables can be defined, *e.g.*, $\mathbf{x} \in \mathbb{R}^n$ is constructed by typing `x = Variable(n)`. By default the variables are initialized by vectors of zeros but it is possible to set different initializations *e.g.*, `Variable([0;1])` will be a variable of two elements initialized by the vector $[0, 1]^\intercal$. The user can also utilize different equivalent notations: for example in the first line of the code snippet of Example 6 the function f could be defined equivalently with `f = 0.5*norm(F*x-y)^2`, by firstly constructing the mapping $\mathsf{F}_{i,c}$ using the notation `F = IDCT(n)`. Similarly, the selection mappings applied to $\mathbf{y}$, *i.e.*, `Mp*y`, could be replaced equivalently by `y[idp]` where `idp` is an array of indexes corresponding to the ones of the selection mapping M_+.

Once the cost function `f` and the constraints `cstr` are defined, the problem can be solved by typing `@minimize f st cstr`. If an efficient proximal mapping is found, the problem is solved using a proximal gradient algorithm. As it can be seen, here this condition is fulfilled despite the fact that multiple constraints over the variable $\mathbf{y}$ are present: these still lead to an efficient proximal mapping since they are applied to non-overlapping slices of the variable $\mathbf{y}$ and are therefore separable.

The standard algorithm, PANOC, is then used to solve the problem, but if a specific one is to be used *e.g.*, the FPG algorithm, one can specify that: `@minimize cf st cstr with FPG()`. As the code snippet of Example 6 shows, the series of problems is set inside a loop: here every problem is automatically warm-started by the previous one, as the variables `x` and `y` are always linked to their respective data vectors which can be accessed by typing `~x`. More details about the software can be found in the documentation online. Finally, in line with the philosophy of reproducible research all the code that was used to create the examples and the various comparison of the algorithms is publicly available online [70].

Many other software packages based on proximal gradient algorithms have been recently developed. There are different MATLAB toolboxes: `FOM` provides several proximal gradient algorithms [74] and `ForBES` implements Newton-type accelerated proximal gradient algorithms [75]. `TFOCS` offers different splitting algorithms that can be used in combination with FAOs through the usage of the toolbox `Spot` [76]. `ProxImaL` [77], also implements different matrix-free splitting algorithms in the Python language with a particular focus to image processing applications.

7. Conclusions

The proximal gradient algorithms described in this paper can be applied to a wide variety of signal processing applications. Many examples were presented here to show this versatility with a particular focus on inverse problems of large-scale that naturally arises in many audio, communication, image and video processing applications. Recent enhancements of the PG algorithm have improved significantly its convergence speed. These offer the possibility of using quasi-Newton methods reaching solutions of high accuracy with a speed that was previously beyond the reach of most first-order methods. Additionally these algorithms can be easily combined with fast forward-adjoint oracles to compute the mappings involved leading to matrix-free optimization.

The applications illustrated in this paper are only a small portion of what these algorithms can tackle and it is envisaged that many others will benefit their properties. In fact, proximal gradient algorithms are relatively simple and they result in very compact implementations, which most of the time do not require additional subroutines, unlike other splitting algorithms *e.g.*, ADMM. This makes them particularly well suited for embedded systems and real-time applications. Additionally, many of the operations involved in this framework are parallel by nature: not only proximal mappings, which in many contexts are separable, but also matrix-free optimization, that utilizes graphs of forward-adjoint oracles, naturally lead to parallelism. This makes these algorithms also particularly fit for wireless sensor networks and many Internet-of-Things applications.

Finally, these algorithms can tackle nonconvex problems: machine learning showed how nonlinear models can reach outstanding results. It is envisaged that these algorithms with their flexibility can be used to create novel nonlinear filters by easily testing the effectiveness of new nonlinear models.

References

References

[1] E. J. Candès, M. B. Wakin, An introduction to compressive sampling, IEEE Signal Process. Mag. 25 (2) (2008) 21–30.

[2] R. Tibshirani, Regression shrinkage and selection via the lasso, Journal of the Royal Statistical Society. Series B (Methodological) (1996) 267–288.

[3] M. Grant, S. Boyd, Y. Ye, Disciplined convex programming, in: Global Optimization: From Theory to Implementation, Springer, 2006, pp. 155–210.

[4] V. Cevher, S. Becker, M. Schmidt, Convex optimization for big data: Scalable, randomized, and parallel algorithms for big data analytics, IEEE Signal Process. Mag. 31 (5) (2014) 32–43.

[5] H. H. Bauschke, P. L. Combettes, Convex analysis and monotone operator theory in Hilbert spaces, Springer, 2011.

[6] P. L. Combettes, J.-C. Pesquet, Proximal splitting methods in signal processing, in: Fixed-point algorithms for inverse problems in science and engineering, Springer, 2011, pp. 185–212.

[7] N. Parikh, S. P. Boyd, Proximal algorithms, Foundations and Trends in Optimization 1 (3) (2014) 127–239.

[8] P.-L. Lions, B. Mercier, Splitting algorithms for the sum of two nonlinear operators, SIAM J. Numer. Anal. 16 (6) (1979) 964–979.

[9] I. Daubechies, M. Defrise, C. De Mol, An iterative thresholding algorithm for linear inverse problems with a sparsity constraint, Comm. Pure Appl. Math. 57 (11) (2004) 1413–1457.

[10] P. L. Combettes, V. R. Wajs, Signal recovery by proximal forward-backward splitting, Multiscale Modeling & Simulation 4 (4) (2005) 1168–1200.

[11] F. Facchinei, J.-S. Pang, Finite-dimensional variational inequalities and complementarity problems, Springer, 2007.

[12] A. Beck, M. Teboulle, A fast iterative shrinkage-thresholding algorithm for linear inverse problems, SIAM Journal on Imaging Sciences 2 (1) (2009) 183–202.

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

[14] J. Douglas, H. H. Rachford, On the numerical solution of heat conduction problems in two and three space variables, Trans. Amer. Math. Soc. 82 (2) (1956) 421–439.

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

[16] Y. Nesterov, A method of solving a convex programming problem with convergence rate $O(1/k^2)$, Soviet Mathematics Doklady 27 (2) (1983) 372–376.

[17] J. F. Bonnans, J. C. Gilbert, C. Lemaréchal, C. A. Sagastizábal, A family of variable metric proximal methods, Math. Programming 68 (1-3) (1995) 15–47.

[18] E. Chouzenoux, J.-C. Pesquet, A. Repetti, Variable metric forward–backward algorithm for minimizing the sum of a differentiable function and a convex function, J. Optimiz. Theory App. 162 (1) (2014) 107–132.

[19] P. Frankel, G. Garrigos, J. Peypouquet, Splitting methods with variable metric for Kurdyka–Łojasiewicz functions and general convergence rates, J. Optimiz. Theory App. 165 (3) (2015) 874–900.

[20] E. Chouzenoux, J.-C. Pesquet, A. Repetti, A block coordinate variable metric forward–backward algorithm, J. Global Optimiz. 66 (3) (2016) 457–485.

[21] S. Becker, J. Fadili, A quasi-Newton proximal splitting method, in: Adv. Neural Inf. Process. Syst., 2012, pp. 2618–2626.

[22] J. D. Lee, Y. Sun, M. A. Saunders, Proximal Newton-type methods for minimizing composite functions, SIAM J. Optimiz. 24 (3) (2014) 1420–1443.

[23] A. Themelis, L. Stella, P. Patrinos, Forward-backward envelope for the sum of two nonconvex functions: Further properties and nonmonotone line-search algorithms, SIAM J. Optimiz. 28 (3) (2018) 2274–2303.

[24] L. Stella, A. Themelis, P. Patrinos, Forward-backward quasi-Newton methods for nonsmooth optimization problems, Computational Optimization and Applications 67 (3) (2017) 443–487. `doi:10.1007/s10589-017-9912-y`.

[25] N. Komodakis, J.-C. Pesquet, Playing with duality: An overview of recent primal-dual approaches for solving large-scale optimization problems, IEEE Signal Process. Mag. 32 (6) (2015) 31–54.

[26] H. Attouch, J. Bolte, B. F. Svaiter, Convergence of descent methods for semi-algebraic and tame problems: proximal algorithms, forward–backward splitting, and regularized Gauss–Seidel methods, Mathematical Programming 137 (1) (2013) 91–129.

[27] P. Jain, P. Kar, Non-convex optimization for machine learning, Foundations and Trends in Machine Learning 10 (3-4) (2017) 142–336.

[28] E. J. Candes, Y. C. Eldar, T. Strohmer, V. Voroninski, Phase retrieval via matrix completion, SIAM Review 57 (2) (2015) 225–251.

[29] N. Boumal, V. Voroninski, A. Bandeira, The non-convex Burer-Monteiro approach works on smooth semidefinite programs, Advances in Neural Information Processing Systems (2016) 2757–2765.

[30] Z.-Q. Luo, W.-K. Ma, A. M.-C. So, Y. Ye, S. Zhang, Semidefinite relaxation of quadratic optimization problems, IEEE Signal Process. Mag. 27 (3) (2010) 20–34.

[31] S. Diamond, S. Boyd, Matrix-free convex optimization modeling, in: Optimization and its Applications in Control and Data Sciences, Springer, 2016, pp. 221–264.

[32] J. Folberth, S. Becker, Efficient adjoint computation for wavelet and convolution operators [lecture notes], IEEE Signal Process. Mag. 33 (6) (2016) 135–147.

[33] M. Grant, S. Boyd, Y. Ye, CVX: Matlab software for disciplined convex programming (2008).
URL `http://cvxr.com/cvx/`

[34] S. Becker, E. Candes, M. Grant, TFOCS: Templates for first-order conic solvers (2012).
URL `http://cvxr.com/tfocs/`

[35] J. Bezanson, A. Edelman, S. Karpinski, V. B. Shah, Julia: A fresh approach to numerical computing, SIAM Review 59 (1) (2017) 65–98. `arXiv:http://dx.doi.org/10.1137/141000671`, `doi:10.1137/141000671`.

[36] C. R. Berger, Z. Wang, J. Huang, S. Zhou, Application of compressive sensing to sparse channel estimation, IEEE Commun. Mag. 48 (11).

[37] I. Kodrasi, S. Doclo, Joint dereverberation and noise reduction based on acoustic multi-channel equalization, IEEE/ACM Trans. Audio Speech Lang. Process. 24 (4) (2016) 680–693.

[38] S. I. Adalbjörnsson, T. Kronvall, S. Burgess, K. Åström, A. Jakobsson, Sparse localization of harmonic audio sources, IEEE/ACM Trans. Audio Speech Lang. Process. 24 (1) (2016) 117–129.

[39] B. N. Bhaskar, G. Tang, B. Recht, Atomic norm denoising with applications to line spectral estimation, IEEE Trans. Signal Process. 61 (23) (2013) 5987–5999.

[40] P. Netrapalli, U. Niranjan, S. Sanghavi, A. Anandkumar, P. Jain, Non-convex robust PCA, Advances in Neural Information Processing Systems (2014) 1107–1115.

[41] S. Theodoridis, Machine learning: a Bayesian and optimization perspective, Academic Press, 2015.

[42] S. Ling, T. Strohmer, Self-calibration and biconvex compressive sensing, Inverse Problems 31 (11).

[43] J. Nocedal, S. Wright, Numerical Optimization, 2nd Edition, Springer, New York, 2006.

[44] R. T. Rockafellar, R. J. Wets, Variational analysis, Springer, 2011.

[45] A. Domahidi, E. Chu, S. Boyd, ECOS: An SOCP solver for embedded systems, in: Proc. European Control Conf. (ECC), 2013, pp. 3071–3076.

[46] B. O'Donoghue, E. Chu, N. Parikh, S. Boyd, Conic optimization via operator splitting and homogeneous self-dual embedding, Journal of Optimization Theory and Applications 169 (3) (2016) 1042–1068.

[47] A. Beck, First-Order Methods in Optimization, SIAM, 2017.

[48] J.-J. Moreau, Proximité et dualité dans un espace Hilbertien, Bulletin de la Société mathématique de France 93 (1965) 273–299.

[49] R. E. Bruck, An iterative solution of a variational inequality for certain monotone operators in Hilbert space, Bulletin of the American Mathematical Society 81 (5) (1975) 890–893. doi:10.1090/s0002-9904-1975-13874-2.

[50] H. Li, Z. Lin, Accelerated proximal gradient methods for nonconvex programming, in: Advances in neural information processing systems, 2015, pp. 379–387.

[51] P. Patrinos, A. Bemporad, Proximal Newton methods for convex composite optimization, in: Proc. 52nd IEEE Conf. Decision Control (CDC), 2013, pp. 2358–2363.

[52] L. Stella, A. Themelis, P. Patrinos, Newton-type alternating minimization algorithm for convex optimization, IEEE Trans. Autom. Control 64 (2) (2019) 697–711. doi:10.1109/TAC.2018.2872203.

[53] P. Patrinos, L. Stella, A. Bemporad, Douglas-Rachford splitting: Complexity estimates and accelerated variants, in: Proc. 53rd IEEE Conf. Decision Control (CDC), 2014, pp. 4234–4239. doi:10.1109/CDC.2014.7040049.

[54] A. Themelis, P. Patrinos, Douglas-Rachford splitting and ADMM for nonconvex optimization: tight convergence results, arXiv preprint arXiv:1709.05747.

[55] L. Stella, A. Themelis, P. Sopasakis, P. Patrinos, A simple and efficient algorithm for nonlinear model predictive control, in: Proc. 56th IEEE Conf. Decision Control (CDC), 2017, pp. 1939–1944.

[56] J. E. Dennis, J. J. Moré, A characterization of superlinear convergence and its application to quasi-Newton methods, Mathematics of computation 28 (126) (1974) 549–560.

[57] P. Tseng, Applications of a splitting algorithm to decomposition in convex programming and variational inequalities, SIAM Journal on Control and Optimization 29 (1) (1991) 119–138. doi:10.1137/0329006.

[58] A. Themelis, M. Ahookhosh, P. Patrinos, On the acceleration of forward-backward splitting via an inexact Newton method, arXiv preprint arXiv:1811.02935.

[59] P. Patrinos, L. Stella, A. Bemporad, Forward-backward truncated newton methods for convex composite optimization, arXiv preprint arXiv:1402.6655.

[60] J. F. Claerbout, Earth soundings analysis: Processing versus inversion, Vol. 6, Blackwell Scientific Publications Cambridge, Massachusetts, USA, 1992.

[61] A. Griewank, A. Walther, Evaluating derivatives: principles and techniques of algorithmic differentiation, Siam, 2008.

[62] Y. LeCun, Y. Bengio, G. Hinton, Deep learning, Nature 521 (7553) (2015) 436–444.

[63] L. M. Briceño-Arias, P. L. Combettes, Convex variational formulation with smooth coupling for multicomponent signal decomposition and recovery, Numer. Math. Theory Methods Appl. 2 (2009) 485–508.

[64] L. M. Briceno-Arias, P. L. Combettes, J.-C. Pesquet, N. Pustelnik, Proximal algorithms for multicomponent image recovery problems, J. Math. Imaging and Vision 41 (1-2) (2011) 3–22.

[65] E. J. Candès, X. Li, Y. Ma, J. Wright, Robust principal component analysis?, Journal of the ACM (JACM) 58 (3) (2011) 11.

[66] P. L. Combettes, Đ. Dũng, B. C. Vũ, Dualization of signal recovery problems, Set-Valued and Variational Analysis 18 (3-4) (2010) 373–404.

[67] A. Beck, M. Teboulle, A fast dual proximal gradient algorithm for convex minimization and applications, Operations Research Letters 42 (1) (2014) 1–6. doi:10.1016/j.orl.2013.10.007.

[68] Y. Nesterov, Smoothing technique and its applications in semidefinite optimization, Mathematical Programming 110 (2) (2007) 245–259.

[69] A. Beck, M. Teboulle, Smoothing and first order methods: A unified framework, SIAM Journal on Optimization 22 (2) (2012) 557–580.

[70] L. Stella, N. Antonello, StructuredOptimization.jl (2017).
URL https://github.com/kul-forbes/StructuredOptimization.jl

[71] A. Adler, V. Emiya, M. G. Jafari, M. Elad, R. Gribonval, M. D. Plumbley, Audio inpainting, IEEE Trans. Audio Speech Lang. Process. 20 (3) (2012) 922–932.

[72] B. Defraene, N. Mansour, S. De Hertogh, T. van Waterschoot, M. Diehl, M. Moonen, Declipping of audio signals using perceptual compressed sensing, in: Proc. 2014 IEEE Global Conf. Signal Inf. Process. (GlobalSIP '14), Atlanta, GA, USA, 2014, pp. 114–125.

[73] S. Kitić, N. Bertin, R. Gribonval, Sparsity and cosparsity for audio declipping: a flexible non-convex approach, in: Proc. Int. Conf. Latent Variable Analysis and Signal Separation, Springer, 2015, pp. 243–250.

[74] A. Beck, N. Guttmann-Beck, FOM–a MATLAB toolbox of first order methods for solving convex optimization problems.
URL https://sites.google.com/site/fomsolver/

[75] L. Stella, P. Patrinos, ForBES (2016).
URL https://github.com/kul-forbes/ForBES

[76] E. van den Berg, M. P. Friedlander, SPOT - A Linear-Operator Toolbox (2014).
URL http://www.cs.ubc.ca/labs/scl/spot/

[77] F. Heide, S. Diamond, M. Nießner, J. Ragan-Kelley, W. Heidrich, G. Wetzstein, ProxImaL: Efficient image optimization using proximal algorithms, ACM Transactions on Graphics (TOG) 35 (4).