A Simple and Exact Algorithm to Solve ℓ^1 Linear Problems
Application to the Compressive Sensing Method

Igor Ciril[1], Jérôme Darbon[2] and Yohann Tendero[3]
[1]*DR2I, Institut Polytechnique des Sciences Avancées, 94200, Ivry-sur-Seine, France*
[2]*Division of Applied Mathematics, Brown University, U.S.A.*
[3]*LTCI, Télécom ParisTech, Université Paris-Saclay, 75013, Paris, France*

Keywords: Basis Pursuit, Compressive Sensing, Inverse Scale Space, Sparsity, ℓ^1 Regularized Linear Problems, Non-smooth Optimization, Maximal Monotone Operator, Phase Transition.

Abstract: This paper considers ℓ^1-regularized linear inverse problems that frequently arise in applications. One striking example is the so called compressive sensing method that proposes to reconstruct a high dimensional signal $\mathbf{u} \in \mathbb{R}^n$ from low dimensional measurements $\mathbb{R}^m \ni \mathbf{b} = A\mathbf{u}$, $m \ll n$. The basis pursuit is another example. For most of these problems the number of unknowns is very large. The recovered signal is obtained as the solution to an optimization problem and the quality of the recovered signal directly depends on the quality of the solver. Theoretical works predict a sharp transition phase for the exact recovery of sparse signals. However, to the best of our knowledge, other state-of-the-art algorithms are not effective enough to *accurately* observe this transition phase. This paper proposes a simple algorithm that computes an exact ℓ^1 minimizer under the constraints $A\mathbf{u} = \mathbf{b}$. This algorithm can be employed in many problems: as soon as A has full row rank. In addition, a numerical comparison with standard algorithms available in the literature is exhibited. These comparisons illustrate that our algorithm compares advantageously: the aforementioned transition phase is empirically observed with a much better quality.

1 INTRODUCTION

Compressive sensing and sparsity promoting methods have gained a significant interest. These methods are used for e.g., data analysis, signal and image processing or inverse problems and amounts to find a suitable $\mathbf{u}$ which satisfies

$$A\mathbf{u} = \mathbf{b} \tag{1}$$

where $\mathbf{b}$ can be seen as some observed data, $A \in \mathcal{M}_{m\times n}(\mathbb{R})$ and $m \ll n$. The columns of A can for instance model an acquisition device or may represent some frame or dictionary designed to adequately encode $\mathbf{u}$. The problem (1) is ill-posed. Therefore, some *a priori* information is needed. The most standard assumption is that $\mathbf{u}$ is "simple", i.e., $\mathbf{u}$ has minimal ℓ^0 pseudo norm. However, finding an ℓ^0 minimizer of (1) is difficult as the optimization problem is non-smooth and non-convex. Hence, methods (Foucart, 2013; Jain et al., 2011; Mallat and Zhang, 1993; Pati et al., 1993; Tropp and Gilbert, 2007; Blumensath and Davies, 2009; Dai and Milenkovic, 2009; Maleki, 2009) that aim at tackling ℓ^0 pseudo norm minimization usually guarantee only an optimal solution with high probability and, to the best of our knowledge, for a specific class of matrices A. The second class of methods proposes to use an ℓ^1 relaxation and the problem becomes

$$\begin{cases} \inf\limits_{\mathbf{u}\in\mathbb{R}^n} & \|\mathbf{u}\|_{\ell^1} \\ \text{s.t.} & A\mathbf{u} = \mathbf{b}. \end{cases} \tag{P_{ℓ^1}}$$

Problem (P_{ℓ^1}) is known as the "basis pursuit problem", see, e.g., (Chen et al., 2001). Under various assumptions, the minimizers remain the same if one replaces the ℓ^0 pseudo-norm by the ℓ^1 norm (see, e.g., (Candes and Tao, 2005; Candes and Tao, 2006; Donoho, 2006; Donoho and Elad, 2003) and the references therein). Problem (P_{ℓ^1}) enjoys strong uniform guarantees for sparse recovery, see, e.g. (Needell and Vershynin, 2009). Loosely speaking, one can easily check if a vector is optimal for (P_{ℓ^1}) and the theoretical works bridge the gap between ℓ^1 and ℓ^0. Yet, problem (P_{ℓ^1}) is a convex one but it is a non-smooth optimization problem in high dimensions (n can be, e.g., the number of pixels of an image). For these

Ciril, I., Darbon, J. and Tendero, Y.
A Simple and Exact Algorithm to Solve ℓ^1 Linear Problems - Application to the Compressive Sensing Method.
DOI: 10.5220/0006624600540062
In *Proceedings of the 13th International Joint Conference on Computer Vision, Imaging and Computer Graphics Theory and Applications (VISIGRAPP 2018) - Volume 4: VISAPP*, pages 54-62
ISBN: 978-989-758-290-5

reasons developing efficient algorithmic solutions is still a challenge in many cases. For instance, the standard CVX solver is not suitable (Grant and Boyd, , Sec. 1.3) for very large problems that arise from signal/image processing applications. Therefore, many methods have been proposed to solve (P_{ℓ^1}) for large signal/images, see, e.g. (Osher et al., 2005; Hale et al., 2007; Beck and Teboulle, 2009; Yin et al., 2008; Zhang et al., 2011; Moeller and Zhang, 2016).
This paper proposes a new algorithm that computes an exact solution to (P_{ℓ^1}) that can be employed in many problems. Indeed, we only assume that A is full row rank. This paper also exhibits a numerical comparison with standard algorithms available in the literature. These comparisons illustrate that our algorithm compares advantageously: the theoretically predicted transition phase, see e.g., (Maleki and Donoho, 2010; Candès et al., 2006) is empirically observed with a much better quality.

Outline of this Paper. The paper is organized as follows. Section 2 gives a very compact, yet self-contained, presentation of the formalism that underlies the algorithm proposed in this paper. The construction yields to algorithm 1 page 3. Section 3 proposes a numerical evaluation and comparison of our algorithm with some other state-of-the-art solution solving (P_{ℓ^1}). We demonstrate in this section that our method outperforms other state-of-the-art methods in terms of quality. Discussions and conclusions are summarized in section 4. The appendix 4 on page 8 contains a Matlab code of the algorithm proposed in this paper and a glossary containing notations and definitions. In the sequel, Latin numerals refer to the glossary of notations page 9.

2 ALGORITHM

This section presents the algorithm proposed in this paper and its underlying mathematical formalism. The method we propose in this paper begins by computing a solution to the dual problem associated to (P_{ℓ^1}) then computes a solution to the primal. The first step involves the exact computation of a finite sequence. The last iterate of this sequence is a solution to the dual problem. The second step consists in computing a constrained least squares problem. We now give the formalism that underlies this algorithm then state the algorithm (see algorithm 1 page 3). Hereinafter, we assume that $A \in \mathcal{M}_{m\times n}(\mathbb{R})$ is full row rank. Consider the Lagrangian $\mathcal{L} : \mathbb{R}^n \times \mathbb{R}^m \to \mathbb{R}$ of (P_{ℓ^1})

$$\mathcal{L}(\mathbf{u},\boldsymbol{\lambda}) := J(\mathbf{u}) + \langle \boldsymbol{\lambda}, A\mathbf{u}\rangle + \langle \boldsymbol{\lambda}, -\mathbf{b}\rangle,$$

where $J(\cdot) = \|\cdot\|_{\ell^1}$, and the Lagrangian dual function $g : \mathbb{R}^m \to \mathbb{R}\cup\{+\infty\}$ defined by

$$\begin{aligned} g(\boldsymbol{\lambda}) &:= -\inf_{\mathbf{u}\in\mathbb{R}^n} \mathcal{L}(\mathbf{u},\boldsymbol{\lambda}) \\ &= -\inf_{\mathbf{u}\in\mathbb{R}^n} \left\{J(\mathbf{u}) - \left\langle -A^\dagger\boldsymbol{\lambda}, \mathbf{u}\right\rangle\right\} - \langle \boldsymbol{\lambda}, -\mathbf{b}\rangle \\ &= J^*\left(-A^\dagger\boldsymbol{\lambda}\right) + \langle \boldsymbol{\lambda}, \mathbf{b}\rangle = \chi_{\{\mathbf{B}_\infty\}}\left(-A^\dagger\boldsymbol{\lambda}\right) + \langle \boldsymbol{\lambda}, \mathbf{b}\rangle . \quad (2) \end{aligned}$$

In 2, J^* denotes the Legendre-Fenchel transform **(ix)** of the convex function $J \in \Gamma_0(\mathbb{R}^n)$ (see **(v)** and **(vi)** and $\chi_{\{\mathbf{B}_\infty\}}$ the convex characteristic function of the ℓ^∞ **(iii)** unit ball B_∞ **(vii)**. (We recall that hereinafter Latin numerals refer to the glossary of notations page 9). Consider also the Lagrangian dual problem

$$\inf_{\boldsymbol{\lambda}\in\mathbb{R}^m} \quad g(\boldsymbol{\lambda}). \qquad (D_{\ell^1})$$

(From (Aubin and Cellina, 2012, Prop. 1, p.163 and Thm. 2, p. 167) problems $(P_{\ell^1}$ and (D_{ℓ^1}) have solutions). We wish to solve (D_{ℓ^1}). To do so, we consider the evolution equation, explicitly given, for every $t \geqslant 0$, by

$$\begin{cases} \dfrac{\mathrm{d}^+\boldsymbol{\lambda}}{\mathrm{dt}}(t) = -\Pi_{\partial g(\boldsymbol{\lambda}(t))}(\mathbf{0}) \\ \boldsymbol{\lambda}(0) = \boldsymbol{\lambda}_0. \end{cases} \qquad (3)$$

In (3), $\frac{\mathrm{d}^+}{\mathrm{dt}}$ denotes the right-derivative **(viii)**, $\partial g(\boldsymbol{\lambda}) = \left\{\mathbf{b} + \sum_{i\in \mathrm{S}(\boldsymbol{\lambda})} \eta_i \tilde{A}\mathbf{e}_i : \eta_i \geqslant 0, i \in \mathrm{S}(\boldsymbol{\lambda})\right\}$, the matrix $\tilde{A} \in \mathcal{M}_{m\times 2n}(\mathbb{R})$ is defined by the column-wise concatenation $\tilde{A} := [A \mid -A]$ and the set $\mathrm{S}(\boldsymbol{\lambda})$ is defined by

$$\mathrm{S}(\boldsymbol{\lambda}) := \left\{i \in \{1,\ldots,2n\} : \left\langle \boldsymbol{\lambda}, \tilde{A}\mathbf{e}_i\right\rangle = 1\right\}, \qquad (4)$$

where $\mathbf{e}_i$ denotes the i-th canonical vector of $\mathbb{R}^{2n}$. We further denote by $\Pi_{\partial g(\boldsymbol{\lambda}(t))}$ the Euclidean projection **(x)** operator on the non-empty closed convex set $\partial g(\boldsymbol{\lambda}(t))$ and $\boldsymbol{\lambda}_0 \in \mathrm{dom}\ \partial g$ **(iv)** is some initial state. (We set $\boldsymbol{\lambda}_0 = \mathbf{0}$ in our experiments).

Using similar arguments to the ones developed in (Hiriart-Urruty and Lemarechal, 1996a, Chap. 8, section 3.4), one can easily prove that the trajectory converges to a solution of (D_{ℓ^1}) and that it is constant after some finite time $t_K \in [0, +\infty)$. It remains to show how one can compute (3) and then compute a solution to (P_{ℓ^1}).

By the same arguments as the ones developed in, e.g., (Hiriart-Urruty and Lemarechal, 1996a, Lem. 3.4.5, p. 381) the trajectory (3) is piecewise affine. This means that (3) can be obtained from a sequence. In other words there exist $(\mathbf{d}_k, t_k)_k$ such that $\boldsymbol{\lambda}_{k+1} = \boldsymbol{\lambda}_k + (t_{k+1} - t_k)\,\mathbf{d}_k$ (we set $t_0 = 0$) with $\mathbf{d}_k = -\Pi_{\partial g(\boldsymbol{\lambda}(t_k))}(\mathbf{0})$ (where Π denotes the projection operator **(x)**). It remains to give a formula for the sca-

lars $(t_{k+1}-t_k)$. The scalars $(t_{k+1}-t_k)$ are given by

$$\bar{\Delta t}_k := (t_{k+1}-t_k) = \min\left\{\frac{1-\langle \tilde{A}\mathbf{e}_i, \boldsymbol{\lambda}_k\rangle}{\langle \tilde{A}\mathbf{e}_i, \mathbf{d}_k\rangle},\ i\in \mathrm{S}^+(\mathbf{d}_k)\right\}, \quad (5)$$

$$\text{where } \mathrm{S}^+(\mathbf{d}_k) := \left\{i\in\{1,\dots,2n\} : \langle \mathbf{d}_k, \tilde{A}\mathbf{e}_i\rangle > 0\right\}. \quad (6)$$

Thus, to compute a solution to (D_{ℓ^1}) one can simply compute (3) using (4)-(6). The cornerstone of this algorithm is the computation of a projection on a convex cone given by the maximal monotone operator ∂g. (One can use for instance a constrained least squares solver, available in, e.g., Matlab, to compute the solution). The computation of $\bar{\Delta t}_k$ is straightforward: it relies on the evaluation of the signs of inner products. It remains to compute a solution to (P_{ℓ^1}) from a solution $\bar{\boldsymbol{\lambda}}$ of (D_{ℓ^1}).

Given a solution of (D_{ℓ^1}) denoted by $\bar{\boldsymbol{\lambda}}$, we can compute a solution $\bar{\mathbf{u}}$ to (P_{ℓ^1}) by solving the following constrained least squares problem

$$\begin{cases} \min\limits_{\mathbf{u}\in\mathbb{R}^n} & \|A\mathbf{u}-\mathbf{b}\|_{\ell^2} \\ \text{s.t.} & u_i \geqslant 0 \text{ if } \langle \bar{\boldsymbol{\lambda}}, A\mathbf{e}_i\rangle = -1, \\ & u_i \leqslant 0 \text{ if } \langle \bar{\boldsymbol{\lambda}}, A\mathbf{e}_i\rangle = 1, \\ & u_i = 0 \text{ otherwise,} \end{cases} \quad (7)$$

where $\mathbf{e}_i$ denotes the i-th canonical vector of $\mathbb{R}^n$. The construction is now complete. Before we state the algorithm, let us make the following remarks.

Remark 1. The reconstruction formula given by (7) is different from the reconstruction methods that can sometimes be found in the literature (see, e.g., (Moeller and Zhang, 2016, algorithm 6, p. 11)). However, for matrices satisfying compressive sensing assumptions (see, e.g., (Candes and Tao, 2006; Donoho, 2006)), the signal can be obtained from an unconstrained least-squares solution to $A\mathbf{u}=\mathbf{b}$. Indeed, the support constraint issued form $\bar{\boldsymbol{\lambda}}$ boils down to solving, in the least squares sense, $B\mathbf{u}=\mathbf{b}$, where B is a sub-matrix formed from A by removing appropriate columns. Notice that in this case, there is no sign constraint on u_i contrarily to (7). In addition, in many cases, the unconstrained least squares solution can be computed using a Moore-Penrose pseudo inverse formula. However, the least squares solution and (7) will, in general, differ: they have same ℓ^0 pseudo norms but different ℓ^1 norms.

Remark 2. To compute $\mathbf{d}_k := -\Pi_{\partial g(\boldsymbol{\lambda}_k)}(\mathbf{0})$ consider the set $G := \left\{\sum_{i\in \mathrm{S}(\boldsymbol{\lambda})} \eta_i \tilde{A}\mathbf{e}_i : \eta_i \geqslant 0, i\in \mathrm{S}(\boldsymbol{\lambda}_k)\right\}$. We have $\mathbf{d}_k = -\Pi_G(-\mathbf{b}) - \mathbf{b}$. This vector can be computed from a constrained least squares problem similar to (7). For instance, one can use the *lsqnonneg* Matlab function. In our implementation, we used (Meyer 2013) that is supposedly faster than the Matlab routine.

Input: Matrix A, $\mathbf{b}$
Output: $\bar{\mathbf{u}}$ solution to (P_{ℓ^1})
Set $k=0$ and $\boldsymbol{\lambda}_k = \mathbf{0}\in\mathbb{R}^m$;
repeat
1. Compute $\mathrm{S}(\boldsymbol{\lambda}_k)$ (see (4));
2. Compute $\mathbf{d}_k := -\Pi_{\partial g(\boldsymbol{\lambda}_k)}(\mathbf{0})$ (see remark (2));
3. Compute $\mathrm{S}^+(\mathbf{d}_k)$ then $\bar{\Delta t}_k$ (see (6) and (5));
4. Update current point $\boldsymbol{\lambda}_{k+1} := \boldsymbol{\lambda}_k + \bar{\Delta t}_k \mathbf{d}_k$;
5. Set $k=k+1$ and set $\boldsymbol{\lambda} := \frac{\boldsymbol{\lambda}}{\|A^\dagger \boldsymbol{\lambda}\|_\infty}$ if $\|A^\dagger\boldsymbol{\lambda}\|_\infty > 1$;

until $\mathbf{d}_k = \mathbf{0}$ *(see remark (2))*;
Compute $\bar{\mathbf{u}}$ using (7).

Algorithm 1: Algorithm computing $\bar{\mathbf{u}}$ solution to (P_{ℓ^1}).

The stopping condition, namely $\mathbf{d}_k = \mathbf{0}$, was replaced by $\|\mathbf{d}_k\|_{\ell^2(\mathbb{R}^m)} \vee \|\bar{\Delta t}_k \mathbf{d}_k\|_{\ell^2(\mathbb{R}^m)} < 10^{-10}$ in all of our experiments. In step (3), to prevent numerical issues, we test if the computed $\bar{\Delta t}_k$ satisfies $\bar{\Delta t}_k \geqslant 0$. If $\bar{\Delta t}_k < 0$ we set $\bar{\Delta t}_k := 0$ and which ends the while loop at the current position.

A Matlab code for algorithm 1 is given in appendix (4). The computations of (2) can be done at relatively low precision. Indeed, algorithm 1 converges for any starting point in $\Omega := \{\mathbf{x} \ : \ A^\dagger \mathbf{x} \in B_\infty\}$ and step (5) ensures that every iterate belongs to Ω. This also means that the current error (e.g., due to finite quantization of numbers) does not depend on previous errors: algorithm 1 is error forgetting.

Claim: Algorithm 1 converges after finitely many iterations to an ℓ^1 minimizer under the constraints $A\mathbf{u}=\mathbf{b}$.

For the sake of completeness we now give some theoretical remarks on algorithm 1. The sequence $(\mathbf{d}_k)_k$ is the steepest Euclidean descent direction for g defined in (2). The time steps calculated in algorithm 1 step 3 guarantee that $g(\boldsymbol{\lambda}(t_{k+1})) < g(\boldsymbol{\lambda}(t_k))$. In other words, this algorithm can be interpreted as a descent method on the dual problem (D_{ℓ^1}). This descent method is ruled by an evolution equation governed by the maximal monotone operator (Brézis, 1973, Chap. 3) ∂g. It is well known that this evolution converges to a solution of (D_{ℓ^1}) (Aubin and Cellina, 2012, Chap. 3). In addition, since g is polyhedral (Rockafellar, 1997, Section 19), the evolution equation (3) is piecewise affine. As we have seen, the trajectory (3) satisfies $\boldsymbol{\lambda}(t) = \boldsymbol{\lambda}(t_K)$ for some $t_K \in [0,+\infty)$ and any $t \geqslant t_K$. Thus, algorithm 1 computes exactly the trajectory (3) and converges after finitely many iterations to a solution to (D_{ℓ^1}). The last step allows us to retrieve a solution to (P_{ℓ^1}) using (7).

3 NUMERICAL EXPERIMENTS

This section proposes an empirical evaluation of the following methods to solve (P_{ℓ^1}): OMP (Pati et al., 1993), CoSamp (Needell and Tropp, 2009), AISS (Burger et al., 2013) GISS (Moeller and Zhang, 2016), and algorithm 1. These methods are compared in terms of quality and number of iterations needed. The first paragraph gives the experimental setup considered. The second paragraph gives the criteria used for the comparisons in terms of quality. The last paragraph reports the results in terms of number of iterations needed.

In these experiments the sensing matrix A always has 1000 columns. The entries of A are drawn from i.i.d. realizations of a centered Gaussian distribution. Without loss of generality we normalize the columns of A to unit Euclidean norm. The number of rows of A, i.e., the dimension of the ambient space m, vary in $M := \{50,\dots,325\}$ with increments of 25. For each number of rows, we vary the *sparsity level* s between 5% and 40% with increments of 5% and therefore consider the discrete set $S := \{0.05,\dots,0.4\}$. The sparsity level is related to the ℓ^0 norm of $\mathbf{u}$ by "$\|\mathbf{u}\|_{\ell^0} = \text{round}\,(s \times m)$" following (Candès et al., 2006). The positions of the non-zero entries of $\mathbf{u}$ are chosen randomly, with uniform probability. The non-zero entries of $\mathbf{u}$ are drawn from a uniform distribution on $[-1,1]$. To do so, for each parameter (i.e., sparsity level s, dimension of ambient space m, type of values for $\mathbf{u}$, i.e., Bernoulli or uniform) we repeated the experiments $1,000$ times. The implementation of CoSamp and OMP we used are due to S. Becker (code updated on Dec 12th, 2012) and can be found on Mathworks file exchange[1]. The implementation of AISS and GISS we used are the ones given by the authors of (Burger et al., 2013; Moeller and Zhang, 2016). The implementation of CoSamp requires an estimate of the sparsity. In all the experiments we used an oracle for CoSamp, i.e., provided the exact sparsity of the source element. We now give the criteria used for the comparisons in terms of quality.

We need to define the "*success*" of algorithm 1. In our opinion, two options can be considered. A natural choice would be to define the *success* as "the output of algorithm 1 is a solution to (P_{ℓ^1})". However, this criterion would be verified for every output. Indeed, algorithm 1 ends with some $\bar{\mathbf{u}}$ that numerically verifies an optimality condition to (P_{ℓ^1}). Thus, this choice seems uninformative. Another natural choice is to define the *success* as "the output of algorithm 1 is equal to the source element $\mathbf{u}$". This choice can be justified by several theoretical works, see, e.g., (Candes and Tao, 2005; Candes and Tao, 2006; Donoho, 2006; Donoho and Elad, 2003). Thus, this second criterion, namely *the output is equal to the source element*, is chosen for the numerical experiments proposed thereafter. Note that this criterion seems slightly in favor of methods specifically designed for the compressive sensing method compared to methods that propose to solve (P_{ℓ^1}). Here, this means that the comparisons are slightly biased in favor of (Pati et al., 1993; Needell and Tropp, 2009). We need to deal with the finite numerical precision. Thus, we define that a reconstruction is *a success* if the relative error satisfies $\frac{\|\mathbf{u}-\mathbf{u}_{est}\|_{\ell^2}}{\|\mathbf{u}\|_{\ell^2}} < 10^{-10}$. Hence, for any $(m,s) \in M \times S$, the empirical probability of success is given by

$$P_{(m,s)} := \frac{1}{\#\text{ of tests}} \sum_i \mathbb{1}_{\left\{\frac{\|\mathbf{u}^i-\mathbf{u}^i_{est}\|_{\ell^2}}{\|\mathbf{u}^i\|_{\ell^2}} < 10^{-10}\right\}}(i), \quad (8)$$

where $\mathbf{u}^i_{est}$ (resp. $\mathbf{u}^i$) is the estimated signal (resp. source signal) and # denotes the cardinality of a set. Each method is tested on the same data by using the same random seed. Remark this type of experimental setup has been used before, for instance in (Jain et al., 2011). Figure 2 depicts the empirical probability of success 8 for OMP, CoSamp, AISS, GISS and algorithm 1. We also consider the difference of probability of success between algorithm 1 and the other methods:

$$D_{(m,s)} := P^{algo\ (1)}_{(m,s)} - P_{(m,s)}, \quad (9)$$

where $m \in M,\ s \in S$, $P^{algo\ 1}_{(m,s)}$ (resp. $P_{(m,s)}$) denotes the quantity (8) obtained with algorithm 1 (resp. OMP, CoSamp, AISS or GISS). It is easy to see that if (9) is positive then algorithm 1 performed better than the other method in terms of *success* as defined above. These differences are depicted in figure 3. It is easily seen that OMP and CoSamp succeed at retrieving the source signal with a probability of approximately 80% for a larger set of parameters than any other method. However, they produce correct results with a probability of nearly 100% for a much smaller set of parameters than AISS, GISS, and algorithm 1. An overview of the similarities and differences between OMP (Pati et al., 1993), CoSamp (Needell and Tropp, 2009), AISS (Burger et al., 2013) GISS (Moeller and Zhang, 2016), and algorithm 1 is given in table 1. In this table, we recall the assumption(s) on A, the variables to maintain during the iterations, the update rule and the type of convergence the method enjoys. In table 1, we also give the empirical probability that a least x% of signals are successfully reconstructed. This statistical indicator is defined by

$$P_{\geqslant x} = \frac{\#\left\{(m,s) \in M \times S \,:\, P_{(m,s)} \geqslant x\right\}}{\#M \cdot \#S}, \quad (10)$$

[1] https://fr.mathworks.com/matlabcentral/fileexchange/32402-cosamp-and-omp-for-sparse-recovery

Table 1: Overview of main similarities and differences between OMP, CoSaMP, AISS, GISS and algorithm 1. Below, ERC stands for exact recovery condition see, e.g., (Pati et al., 1993) and RIC stands for restricted isometry constant see, e.g., (Needell and Tropp, 2009).

Method	**OMP** (Pati et al., 1993)	**CoSamp** (Needell and Tropp, 2009)	**AISS** (Burger et al., 2013)	**GISS** (Moeller and Zhang, 2016)	**Algorithm** 1
Assumption(s)	A satisfies ERC	A satisfies RIC	$\exists \mathbf{u} \in \mathbb{R}^n : A\mathbf{u} = \mathbf{b}$	A satisfies ERC	A full row rank
Variable(s)	$\mathbf{u}_k \in \mathbb{R}^n$	$\mathbf{u}_k \in \mathbb{R}^n$	$\mathbf{u}_k \in \mathbb{R}^n$ and $\mathbf{p}_k \in \mathbb{R}^m$	$\mathbf{u}_k \in \mathbb{R}^n$ and $\mathbf{p}_k \in \mathbb{R}^m$	$\boldsymbol{\lambda}_k \in \mathbb{R}^m$
Update(s) rule(s)	least squares in $\mathbb{R}^n$	least squares and hard threshold in $\mathbb{R}^n$	constrained least squares in $\mathbb{R}^n$	least squares in $\mathbb{R}^n$	projection onto cone of $\mathbb{R}^m$
Type of convergence	Probabilistic	Probabilistic	Exact	Exact	Exact
Probability of at least 90 % of success (10)	0.5104	**0.5521**	0.4688	0.4063	0.4688
Probability of at least 95 % of success (10)	0.3854	**0.4583**	0.4375	0.3333	0.4479
Probability of at least 99 % of success (10)	0.1042	0.1250	0.3438	0.1042	**0.4271**
Probability of at least 99.9 % of success (10)	0	0	0.1250	0	**0.3958**
Probability of at least 100 % of success (10)	0	0	0.0521	0	**0.3958**

where $P_{(m,s)}$ is defined by (8). We now report results concerning the number of iterations needed to achieve convergence.

Figure 1 gives the average number of iterations needed by algorithm 1. The maximal average number of iterations is bounded from above by 600. Comparing figures 1 and 2 it can be noticed that when the algorithm succeeds in recovering the source element the average number of iterations remains below (about) 250. This suggests modifying algorithm 1 to use an early stopping criterion. The stopping criterion would be either the optimality condition is reached or a maximal number of iteration is attained. The computational efficiency of the other algorithms considered here can be found in, e.g., (Moeller and Zhang, 2016).

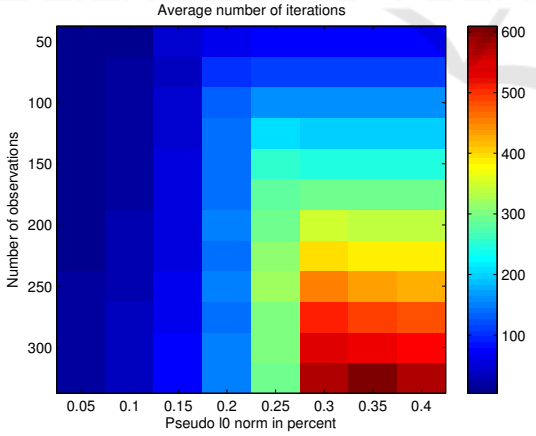

Figure 1: Average number of iteration needed by algorithm 1. This figure suggest an early stopping criterion. See text.

4 CONCLUSIONS

In this paper, a new fast algorithm to solve ℓ^1-regularized linear problems was proposed. The algorithm is exact: it produces an exact ℓ^1 minimizer under the constraints $A\mathbf{u} = \mathbf{b}$ in finite time. The method relies on the calculation of a finite sequence of feasible points of the dual. This sequence converges to a solution of the dual problem. A solution of the primal problem is then obtained by a simple least squares with signs constraints. Numerical comparisons with existing methods of the literature has been proposed for the noiseless compressive sensing (also called basis pursuit) problem. These comparisons allowed us to illustrate the accuracy of the algorithm developed in this paper compared to existing methods for the compressive sensing paradigm. Indeed, the number of parameters[2] where the ℓ^1 reconstruction was nearly always equal to the ℓ^0 (sparsest) one is bigger with our algorithm. The algorithm developed in this paper only requires that A is full row rank. Hence, it can be used in many problems. For instance, it can be applied with matrices that do not comply with RIP-like properties. The initialization of our algorithm was chosen arbitrarily as the null vector. Future work could investigate the benefits of a smarter initial guess to further speed up the method. Another ongoing work studies plausible stopping criteria.

ACKNOWLEDGEMENTS

The authors thanks Said Ladjal and Karim Trabelsi for their fruitful comments. The authors also thank the university of Valencia for its hospitality. The Authors thank M. Mller for providing the implementation of the GISS algorithm.

[2] ℓ^0 pseudo norm, ambient dimension and dimension of the observed data.

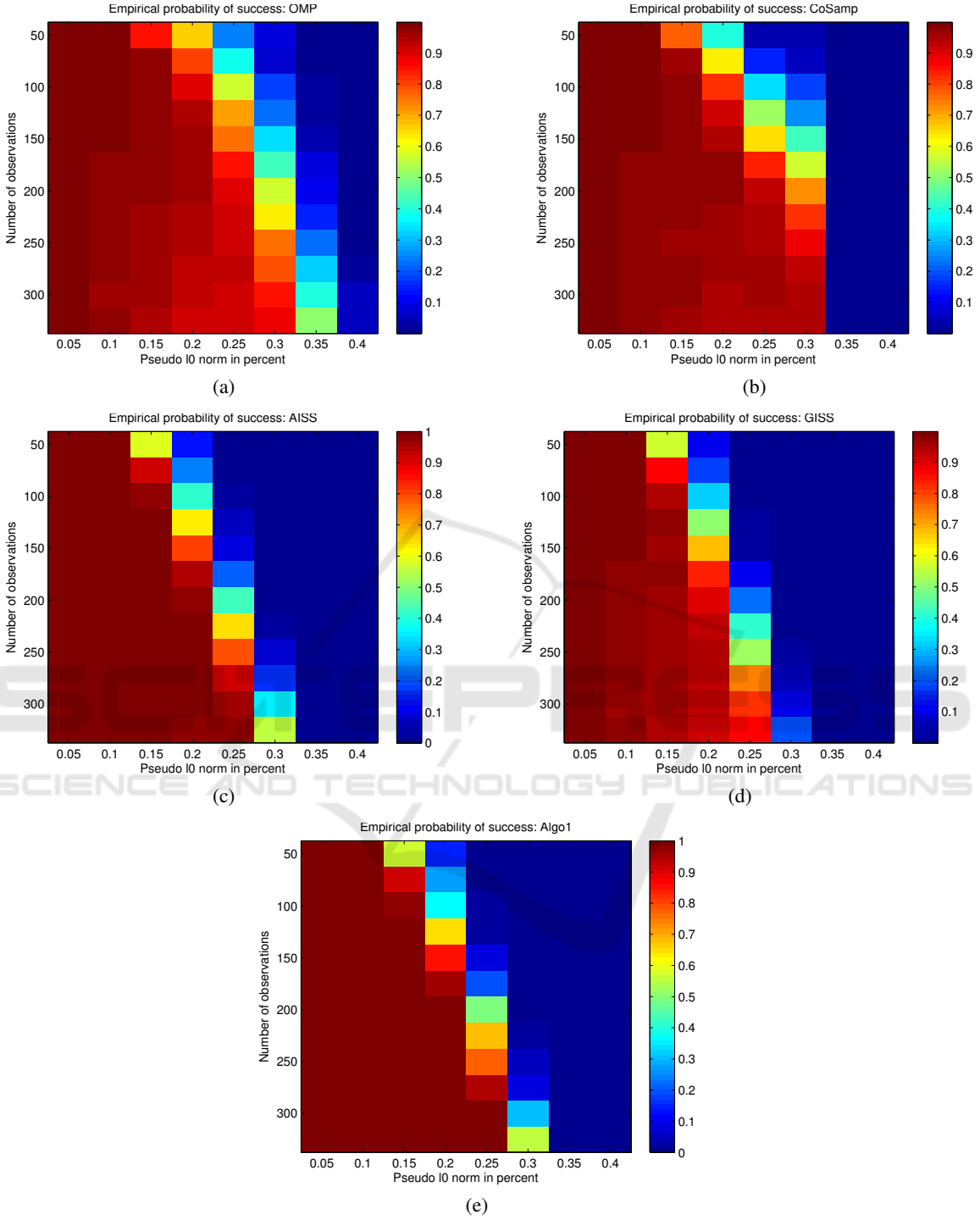

Figure 2: Empirical probability of success (8). Panel (a): OMP (Pati et al., 1993), panel (b): CoSamp (Needell and Tropp, 2009), panel (c): AISS (Burger et al., 2013), panel (d) GISS (Moeller and Zhang, 2016) and panel (e) algorithm 1. The non-zero entries of the source element $\mathbf{u}$ are drawn from a uniform distribution on $[-1,1]$. The entries in A are drawn from i.i.d. realization of a Gaussian distribution.

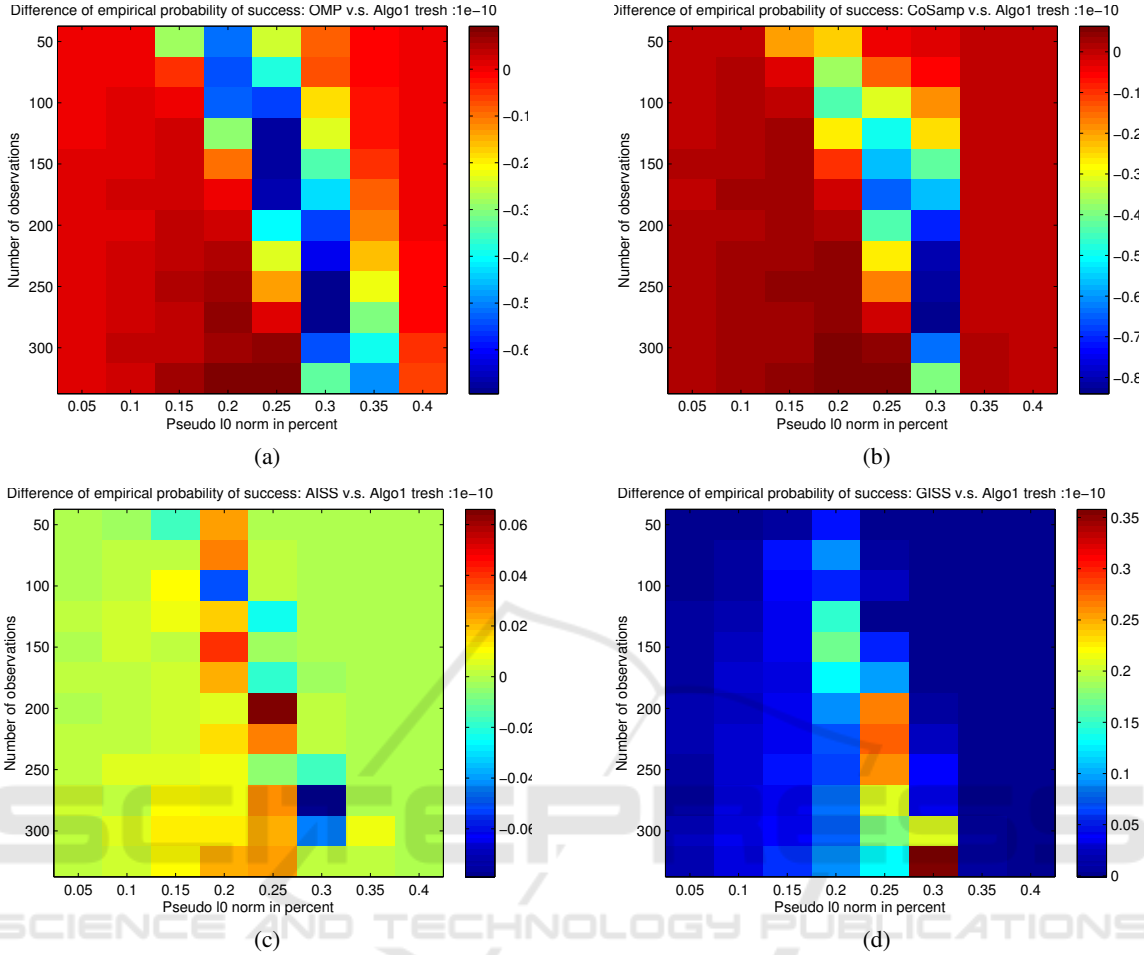

Figure 3: Difference of probability of success (9). Panel (a): Algo 1-OMP (Pati et al., 1993), panel (b): Algo 1-CoSamp (Needell and Tropp, 2009), panel (c): Algo 1-AISS (Burger et al., 2013) and panel (d) Algo 1-GISS (Moeller and Zhang, 2016). A positive value indicates that algorithm 1 performs better, a negative value the contrary.

REFERENCES

Aubin, J.-P. and Cellina, A. (2012). *Differential Inclusions: Set-Valued Maps and Viability Theory*. Grundlehren der mathematischen Wissenschaften. Springer Berlin Heidelberg.

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

Blumensath, T. and Davies, M. E. (2009). Iterative hard thresholding for compressed sensing. *Applied and Computational Harmonic Analysis*, 27(3):265–274.

Brézis, H. (1973). *Opérateurs Maximaux Monotones et Semi-Groupes de Contractions Dans Les Espaces de Hilbert*. North-Holland Publishing Co., American Elsevier Publishing Co.

Burger, M., Möller, M., Benning, M., and Osher, S. (2013). An adaptive inverse scale space method for compressed sensing. *Mathematics of Computation*, 82(281):269–299.

Candès, E. J., Romberg, J., and Tao, T. (2006). Robust uncertainty principles: Exact signal reconstruction from highly incomplete frequency information. *IEEE Transactions on information theory*, 52(2):489–509.

Candes, E. J. and Tao, T. (2005). Decoding by linear programming. *Information Theory, IEEE Transactions on*, 51(12):4203–4215.

Candes, E. J. and Tao, T. (2006). Near-optimal signal recovery from random projections: Universal encoding strategies? *Information Theory, IEEE Transactions on*, 52(12):5406–5425.

Chen, S. S., Donoho, D. L., and Saunders, M. A. (2001). Atomic decomposition by basis pursuit. *SIAM review*, 43(1):129–159.

Dai, W. and Milenkovic, O. (2009). Subspace pursuit for compressive sensing signal reconstruction. *Information Theory, IEEE Transactions on*, 55(5):2230–2249.

Donoho, D. L. (2006). Compressed sensing. *Information Theory, IEEE Transactions on*, 52(4):1289–1306.

Donoho, D. L. and Elad, M. (2003). Optimally sparse representation in general (nonorthogonal) dictionaries via $\ell 1$ minimization. *Proceedings of the National Academy of Sciences*, 100(5):2197–2202.

Foucart, S. (2013). Stability and robustness of weak orthogonal matching pursuits. In *Recent advances in harmonic analysis and applications*, pages 395–405. Springer.

Grant, M. C. and Boyd, S. P. *The CVX Users Guide, Release 2.1*.

Hale, E. T., Yin, W., and Zhang, Y. (2007). A Fixed-Point Continuation Method for l1-Regularized Minimization with Applications to Compressed Sensing. *CAAM Technical Report TR07-07*.

Hiriart-Urruty, J.-B. and Lemarechal, C. (1996a). *Convex Analysis and Minimization Algorithms I: Fundamentals*. Grundlehren der mathematischen Wissenschaften. Springer Berlin Heidelberg.

Hiriart-Urruty, J.-B. and Lemarechal, C. (1996b). *Convex Analysis and Minimization Algorithms II: Advanced Theory and Bundle Methods*. Grundlehren der mathematischen Wissenschaften. Springer Berlin Heidelberg.

Jain, P., Tewari, A., and Dhillon, I. S. (2011). Orthogonal matching pursuit with replacement. In *Advances in Neural Information Processing Systems*, pages 1215–1223.

Maleki, A. (2009). Coherence analysis of iterative thresholding algorithms. In *Communication, Control, and Computing, 2009. Allerton 2009. 47th Annual Allerton Conference on*, pages 236–243. IEEE.

Maleki, A. and Donoho, D. L. (2010). Optimally tuned iterative reconstruction algorithms for compressed sensing. *IEEE Journal of Selected Topics in Signal Processing*, 4(2):330–341.

Mallat, S. and Zhang, Z. (1993). Matching pursuits with time-frequency dictionaries. *Signal Processing, IEEE Transactions on*, 41(12):3397–3415.

Meyer, M. C. (2013). A simple new algorithm for quadratic programming with applications in statistics. *Communications in Statistics-Simulation and Computation*, 42(5):1126–1139.

Moeller, M. and Zhang, X. (2016). Fast Sparse Reconstruction: Greedy Inverse Scale Space Flows. *Mathematics of Computation*, 85(297):179–208.

Needell, D. and Tropp, J. A. (2009). CoSaMP: Iterative signal recovery from incomplete and inaccurate samples. *Applied and Computational Harmonic Analysis*, 26(3):301–321.

Needell, D. and Vershynin, R. (2009). Uniform uncertainty principle and signal recovery via regularized orthogonal matching pursuit. *Foundations of computational mathematics*, 9(3):317–334.

Osher, S., Burger, M., Goldfarb, D., Xu, J., and Yin, W. (2005). An iterative regularization method for total variation-based image restoration. *Multiscale Modeling & Simulation*, 4(2):460–489.

Pati, Y. C., Rezaiifar, R., and Krishnaprasad, P. (1993). Orthogonal matching pursuit: Recursive function approximation with applications to wavelet decomposition. In *Conference Record of The Twenty-Seventh Asilomar Conference on Signals, Systems and Computers*, pages 40–44. IEEE.

Rockafellar, R. T. (1997). *Convex Analysis*. Princeton University Press, Princeton, NJ.

Tropp, J. A. and Gilbert, A. C. (2007). Signal recovery from random measurements via orthogonal matching pursuit. *Information Theory, IEEE Transactions on*, 53(12):4655–4666.

Yin, W., Osher, S., Goldfarb, D., and Darbon, J. (2008). Bregman iterative algorithms for \ell_1-minimization with applications to compressed sensing. *SIAM Journal on Imaging Sciences*, 1(1):143–168.

Zhang, X., Burger, M., and Osher, S. (2011). A unified primal-dual algorithm framework based on bregman iteration. *Journal of Scientific Computing*, 46(1):20–46.

APPENDIX

Program Code

Matlab code for algorithm 1.

```
function [solution_primal,solution_dual,nb_iteration]=compute_solution_l1(A,
                                                    ...,observed_vect, param)

% Step. 0 Initialization
lambda_init=param.initdual;
A=[A -A];

%% Step1. Compute solution dual
[solution_dual,active_set,nb_iteration]=compute_solution_dual(A,
                            observed_vect,...,lambda_init,param);

%% Step2. Compute solution primal
solution_primal=compute_solution_primal(A,observed_vect,active_set,param);



%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%NESTED FUNCTIONS: Compute Dual + Primal %%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%



%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%  Compute_solution_dual %%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function [lambda,active_set,nb_iteration]=compute_solution_dual(A,
                                    observed_vect,...,lambda,param)
%% 0. Initialize DO WHILE loop and pre-compute some stuff
shall_continue=true;
Atransp=A';
nb_iteration=0;

while shall_continue
 % 1. Compute active set
    active_set=compute_active_set(A,lambda,param);

 % 2. Compute direction
    direction=compute_direction(A,observed_vect,active_set,param);

 % 3. Compute kick time
    timestep=compute_kick_time(A,direction,lambda,param);

 % 4. Check termination condition
    if (norm(direction*timestep)<param.tolcomputedual...
          ||norm(direction)<param.tolcomputedual||timestep==realmax)
     % 5.1 Stop
     shall_continue=false;
    else  % 5.2 Move current point in dual
```

```
     lambda=update_lambda(Atransp,lambda,direction,timestep,param);
     end; % endif
    nb_iteration=nb_iteration+1
end; % end whil
end % end compute_solution_dual function
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%% update_lambda %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function lambda=update_lambda(Atransp,lambda,direction,...
                              timestep,param)

% 0. Safery check : if not valid set t to 0 then quit (termination valid)
    if  isempty(timestep) timestep=0; end; %end if
% 1. Move in dual feasible set
    lambda=lambda+timestep*direction;

% 1.1 : reproject new positon onto feasible set
    if (param.leastquarefit_flag)
    % compute projection onto  dual feasible set
    norm_Atransplambda=max(abs(Atransp*lambda))+param.tolcomputedual;
    if (norm_Atransplambda>1+param.tolcomputedual)
        lambda=(1+param.tolcomputedual)*lambda/norm_Atransplambda;
    end;
    end; % end if
end % end: update_lambda functon
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%% compute_active_set %%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function active_set=compute_active_set(A,lambda,param)
active_set=(((lambda')*A)>=1-param.tolcomputedual);
end % end : compute_active_set function
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%% compute_direction %%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function direction=compute_direction(A,observed_vect,active_set,param)
% the next is equal to the proj of 0 onto the set b+ active cols of A
direction=-observed_vect-projection_simplicial_cone(A,-observed_vect,...
    active_set,param);
end % end : compute_direction function
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%% compute_kick_time %%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function t=compute_kick_time(A,direction,lambda,param)

% 0. Compute S_C^+(lambda) set
 active_set_plus=(((direction')*A)>param.tolcomputedual);

% 1. Compute kick time
% 1.1: verify possible emptyness
if (~any(active_set_plus)) % empty active set
        t=realmax; %'+infty (from its definition)
else
% 1.2 non empty case
     Aactiveplus=A(:,active_set_plus);
     t=min((1-((lambda')*Aactiveplus))./((direction')*Aactiveplus));
     % safety check t must be non negative
     if (t<0) t=0; % we set t:=0; then quit (termination condition valid)
    end;
end; % end if
end % end: compute_kick_time function


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%% compute_solution_primal %%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function sol_primal=compute_solution_primal(A,observed_vect,...
                                            active_set,param)
[~,sol_primal]=projection_simplical_cone(A,-observed_vect,...
                                         active_set,param);
sol_primal=-sol_primal(1:end/2)+sol_primal(end/2+1:end);
end % end : compute_solution_primal function
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% END NESTED FUNCTIONS%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
end% end: compute_solution_l1 function
```

Notations

(i) (Vectors) Vectors of, e.g., $\mathbb{R}^n$ are denoted in bold typeface, e.g., $\mathbf{x}$. Other objects like scalars or functions are denoted in non-bold typeface

(ii) (Inner product) $\langle \mathbf{x}, \mathbf{y} \rangle = \sum_i x_i y_i$

(iii) (the $\ell^\infty(\mathbb{R}^n)$ unit ball) $B_\infty = \{\mathbf{u} : \langle \mathbf{u}, \mathbf{e}_i \rangle \leqslant 1, i = 1, \ldots, 2n\}$ where $\mathbf{e}_i$ denotes the canonical vector

(iv) (Effective domain) dom f: The domain of a convex function f is the (convex, possibly empty) set dom $f = \{\mathbf{x} \in \mathbb{R}^n \ : \ f(\mathbf{x}) \in \mathbb{R}\}$

(v) (Convex function) A function $f : \mathbb{R}^n \to \mathbb{R} \cup \{+\infty\}$ is said to be convex if $\forall(\mathbf{x}, \mathbf{y}) \in \mathbb{R}^n \times \mathbb{R}^n$ and $\forall \alpha \in (0, 1)$ $f(\alpha \mathbf{x} + (1 - \alpha)\mathbf{y}) \leqslant \alpha f(\mathbf{x}) + (1 - \alpha) f(\mathbf{y})$ holds true (in $\mathbb{R} \cup \{+\infty\}$)

(vi) (Set $\Gamma_0(\mathbb{R}^n)$) The set of lower semi-continuous, convex functions with dom $f \neq \varnothing$ is denoted by $\Gamma_0(\mathbb{R}^n)$

(vii) (Characteristic function of a set) $\chi_E(\mathbf{x}) = 0$ if $\mathbf{x} \in E$ and $\chi_E(\mathbf{x}) = +\infty$ otherwise

(viii) (Right derivative) $\frac{\mathrm{d}^+ \boldsymbol{\lambda}(t)}{\mathrm{d}t} := \lim_{h \to 0^+} \frac{\boldsymbol{\lambda}(t+h) - \boldsymbol{\lambda}(t)}{h}$

(ix) (Convex conjugate) For any f convex that satisfies dom $f \neq \varnothing$, the function f^* defined by $\mathbb{R}^n \ni \mathbf{s} \mapsto f^*(\mathbf{s}) := \sup_{\mathbf{x} \in \mathrm{dom}\, f} \{\langle \mathbf{s}, \mathbf{x} \rangle - f(\mathbf{x})\}$. (See , e.g., (Hiriart-Urruty and Lemarechal, 1996b, Def. 1.1.1, p. 37))

(x) (Euclidean projection) $\Pi_C(\mathbf{x}) = \arg\min_{\mathbf{y} \in C} \|\mathbf{y} - \mathbf{x}\|_{\ell^2}$ for $C \neq \varnothing$ closed and convex