Performance of first- and second-order methods for ℓ1-regularized least squares problems

Noname manuscript No.
(will be inserted by the editor)

Performance of First- and Second-Order Methods for ℓ_1-Regularized Least Squares Problems

Kimon Fountoulakis · Jacek Gondzio

Received: date / Accepted: date

Abstract We study the performance of first- and second-order optimization methods for ℓ_1-regularized sparse least-squares problems as the conditioning of the problem changes and the dimensions of the problem increase up to one trillion. A rigorously defined generator is presented which allows control of the dimensions, the conditioning and the sparsity of the problem. The generator has very low memory requirements and scales well with the dimensions of the problem.

Keywords ℓ_1-regularised least-squares · First-order methods · Second-order methods · Sparse least squares instance generator · Ill-conditioned problems

1 Introduction

We consider the problem

$$\text{minimize } f_\tau(x) := \tau\|x\|_1 + \frac{1}{2}\|Ax - b\|_2^2, \tag{1}$$

where $x \in \mathbb{R}^n$, $\|\cdot\|_1$ denotes the ℓ_1-norm, $\|\cdot\|_2$ denotes the Euclidean norm, $\tau > 0$, $A \in \mathbb{R}^{m\times n}$ and $b \in \mathbb{R}^m$. An application that is formulated as in (1) is

Kimon Fountoulakis
School of Mathematics and Maxwell Institute, The University of Edinburgh, James Clerk Maxwell Building, The King's Buildings, Peter Guthrie Tait Road, Edinburgh EH9 3JZ, Scotland UK
Tel.: +44 131 650 5083
E-mail: K.Fountoulakis@sms.ed.ac.uk

Jacek Gondzio
School of Mathematics and Maxwell Institute, The University of Edinburgh, James Clerk Maxwell Building, The King's Buildings, Peter Guthrie Tait Road, Edinburgh EH9 3JZ, Scotland UK
Tel.: +44 131 650 8574
Fax: +44 131 650 6553
E-mail: J.Gondzio@ed.ac.uk

sparse data fitting, where the aim is to approximate n-dimensional sampled points (rows of matrix A) using a linear function, which depends on less than n variables, i.e., its slope is a sparse vector. Let us assume that we sample m data points (a_i, b_i), where $a_i \in \mathbb{R}^n$ and $b_i \in \mathbb{R}$ $\forall i = 1, 2, \cdots, m$. We assume linear dependence of b_i on a_i:

$$b_i = a_i^\intercal x + e_i \quad \forall i = 1, 2, \cdots, m,$$

where e_i is an error term due to the sampling process being innacurate. Depending on the application some statistical information is assumed about vector e. In matrix form the previous relationship is:

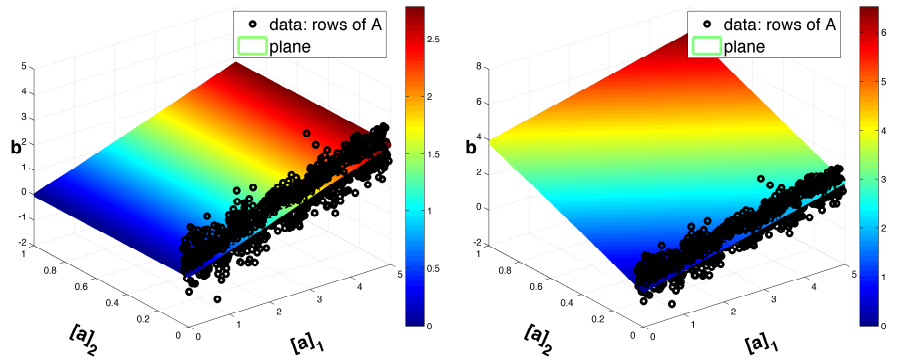
$$b = Ax + e, \tag{2}$$

where $A \in \mathbb{R}^{m \times n}$ is a matrix with a_i's as its rows and $b \in \mathbb{R}^m$ is a vector with b_i's as its components. The goal is to find a sparse vector x (with many zero components) such that the error $\|Ax - b\|_2$ is minimized. To find x one can solve problem (1). The purpose of the ℓ_1 norm in (1) is to promote sparsity in the optimal solution [1]. An example that demonstrates the purpose of the ℓ_1 norm is presented in Figure 1. Figure 1 shows a two dimensional instance where $n = 2$, $m = 1000$ and matrix A is full-rank. Notice that the data points a_i $\forall i = 1, 2, \cdots, m$ have large variations with respect to feature $[a_i]_1$ $\forall i$, where $[\cdot]_j$ is the jth component of the input vector, while there is only a small variation with respect to feature $[a_i]_2$ $\forall i$. This property is captured when problem (1) is solved with $\tau = 30$. The fitted plane in Figure 1a depends only on the first feature $[a]_1$, while the second feature $[a]_2$ is ignored because $[x^*]_2 = 0$, where x^* is the optimal solution of (1). This can be observed through the level sets of the plane shown with the colored map; for each value of $[a]_1$ the level sets remain constant for all values of $[a]_2$. On the contrary, this is not the case when one solves a simple least squares problem ($\tau = 0$ in (1)). Observe in Figure 1a that the fitted plane depends on both features $[a]_1$ and $[a]_2$.

A variety of sparse data fitting applications originate from the fields of signal processing and statistics. Five representative examples are briefly described below.

- Magnetic Resonance Imaging (MRI): A medical imaging tool used to scan the anatomy and the physiology of a body [27].
- Image inpainting: A technique for reconstructing degraded parts of an image [7].
- Image deblurring: Image processing tool for removing the blurriness of a photo caused by natural phenomena, such as motion [21].
- Genome-Wide Association study (GWA): DNA comparison between two groups of people (with/without a disease) in order to investigate factors that a disease depends on [42].
- Estimation of global temperature based on historic data [22].

Data fitting problems frequently require the *analysis of large scale data sets*, i.e., gigabytes or terabytes of data. In order to address large scale problems



(a) ℓ_1 regularized (b) ℓ_2 regularized

Fig. 1: Demonstration of the purpose of the ℓ_1 norm for data fitting problems.

there has been a resurgence in methods with computationally inexpensive iterations. For example many first-order methods were recovered and refined, such as coordinate descent [17,24,34,39–41,44,45], alternating direction method of multipliers [9,15,18,23,43], proximal first-order methods [2,12,33] and first-order smoothing methods [5,6,30]. The previous are just few representative examples, the list is too long for a complete demonstration, many other examples can be found in [1,11]. Often the goal of modern first-order methods is to reduce the computational complexity per iteration, while preserving the theoretical worst case iteration complexity of classic first-order methods [29]. Many modern first order methods meet the previous goal. For instance, coordinate descent methods can have up to n times less computational complexity per iteration [35,34].

First-order methods have been very successful in various scientific fields, such as support vector machine [46], compressed sensing [14], image processing [12] and data fitting [22]. Several new first-order type approaches have recently been proposed for various imaging problems in the special issue edited by M. Bertero, V. Ruggiero and L. Zanni [8]. However, even for the simple unconstrained problems that arise in the previous fields there exist more challenging instances. Since first-order methods do not capture sufficient second-order information, their performance might degrade unless the problems are well conditioned [16]. On the other hand, the second-order methods capture the curvature of the objective function sufficiently well, but by consensus they are usually applied only on medium scale problems or when high precision accuracy is required. In particular, it is frequently claimed [2,5,6,20,38] that the second-order methods do not scale favourably as the dimensions of the problem increase because of their high computational complexity per iteration. Such claims are based on an assumption that a *full* second-order information has to be used. However, there is evidence [16,19] that for non-trivial problems, *inexact* second-order methods can be very efficient.

In this paper we will exhaustively study the performance of first- and second-order methods. We will perform numerical experiments for large-scale problems with sizes up to one trillion of variables. We will examine conditions under which certain methods are favoured or not. We hope that by the end of this paper the reader will have a clear view about the performance of first- and second-order methods.

Another contribution of the paper is the development of a rigorously defined instance generator for problems of the form of (1). The most important feature of the generator is that it scales well with the size of the problem and can inexpensively create instances where the user controls the sparsity and the conditioning of the problem. For example see Subsection 8.9, where an instance of one trillion variables is created using the proposed generator. We believe that the flexibility of the proposed generator will cover the need for generation of various good test problems.

This paper is organised as follows. In Section 2 we briefly discuss the structure of first- and second-order methods. In Section 3 we give the details of the instance generator. In Section 4 we provide examples for constructing matrix A. In Section 5, we present some measures of the conditioning of problem (1). These measures will be used to examine the performance of the methods in the numerical experiments. In Section 6 we discuss how the optimal solution of the problem is selected. In Section 7 we briefly describe known problem generators and explain how our propositions add value to the existing approaches. In Section 8 we present the practical performance of first- and second-order methods as the conditioning and the size of the problems vary. Finally, in Section 9 we give our conclusions.

2 Brief discussion on first- and second-order methods

We are concerned with the performance of unconstrained optimization methods which have the following intuitive setting. At every iteration a convex function $Q_\tau(y;x) : \mathbb{R}^n \to \mathbb{R}$ is created that locally approximates f_τ at a given point x. Then, function Q_τ is minimized to obtain the next point. An example that covers the previous setting is the Generic Algorithmic Framework (GFrame) which is given below. Details of GFrame for each method used in this paper are presented in Section 8.

Loosely speaking, close to the optimal solution of problem (1), the better the approximation Q_τ of f_τ at any point x the fewer iterations are required to solve (1). On the other hand, the practical performance of such methods is a trade-off between careful incorporation of the curvature of f_τ, i.e. second-order derivative information in Q_τ and the cost of solving subproblem (3) in GFrame.

Discussion on two examples of Q_τ which consider different trade-off follows. First, let us fix the structure of Q_τ for problem (1) to be

$$Q_\tau(y;x) := \tau\|y\|_1 + \frac{1}{2}\|Ax-b\|^2 + (A^\intercal(Ax-b))^\intercal(y-x) + \frac{1}{2}(y-x)^\intercal H(y-x), \quad (4)$$

1: Initialize $x_0 \in \mathbb{R}^n$ and $y_0 \in \mathbb{R}^n$
For $k = 0, 1, 2, \ldots$ until some termination criteria are satisfied
2: Create a convex function $Q_\tau(y; y_k)$ that approximates f_τ in a neighbourhood of y_k
3: Approximately (or exactly) solve the subproblem

$$x_{k+1} \approx \arg\min_y \; Q(y; y_k) \tag{3}$$

4: Find a step-size $\alpha > 0$ based on some criteria and set

$$y_{k+1} = x_k + \alpha(x_{k+1} - x_k)$$

end-for
5: Return approximate solution x_{k+1}

Algorithm : Generic Framework (GFrame)

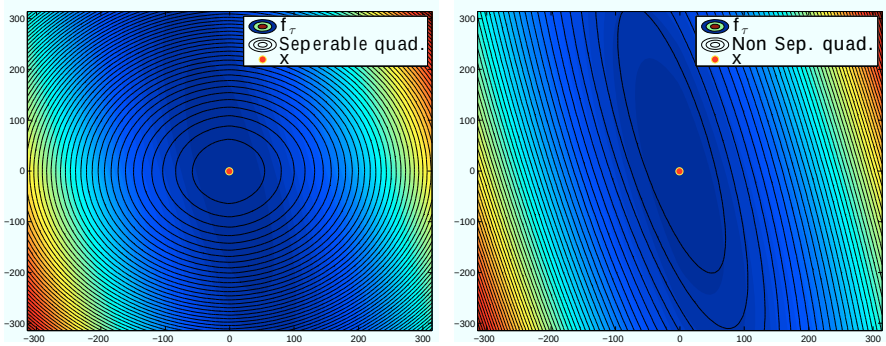
where $H \in \mathbb{R}^{n \times n}$ is a positive definite matrix. Notice that the decision of creating Q_τ has been reduced to a decision of selecting H. Ideally, matrix H should be chosen such that it represents curvature information of $1/2\|Ax-b\|^2$ at point x, i.e. matrix H should have similar spectral decomposition to $A^\intercal A$. Let $B(x) := \{v \in \mathbb{R}^n \mid \|v - x\|_2^2 \leq 1\}$ be a unit ball centered at x. Then, H should be selected in an optimal way:

$$\min_{H \succeq 0} \int_B |f_\tau(y) - Q_\tau(y; x)| dB. \tag{5}$$

The previous problem simply states that H should minimize the sum of the absolute values of the residual $f_\tau - Q_\tau$ over B. Using twice the fundamental theorem of calculus on f_τ from x to y we have that (5) is equivalent to

$$\min_{H \succeq 0} \frac{1}{2} \int_B \Big| (y-x)^\intercal (A^\intercal A - H)(y-x) \Big| dB. \tag{6}$$

It is trivial to see that the best possible H is simply $H = A^\intercal A$. However, this makes every subproblem (3) as difficult to be minimized as the original problem (1). One has to reevaluate the trade-off between a matrix H that sufficiently well represents curvature information of $1/2\|Ax-b\|^2$ at a point x compared to a simple matrix H that is not as good approximation but offers an inexpensive solution of subproblem (3). An example can be obtained by setting H to be a positively scaled identity, which gives a solution to problem (5) $H = \lambda_{max}(A^\intercal A) I_n$, where $\lambda_{max}(\cdot)$ denotes the largest eigenvalue of the input matrix and I_n is the identity matrix of size $n \times n$. The contours of such a function Q_τ compared to those of function f_τ are presented in Subfigure 2a. Notice that the curvature information of function f_τ is lost, this is because nearly all spectral properties of $A^\intercal A$ are lost. However, for such a function Q_τ the subproblem (3) has an inexpensive closed form solution known as iterative shrinkage-thresholding [12,27]. The computational complexity per iteration is so low that one hopes that this will compensate for the losses of curvature information. Such methods, are called first-order methods and have been shown to be efficient for some large scale problems of the form of (1) [2].



(a) Separable quadratic (b) Non separable quadratic

Fig. 2: Demonstration of the contours of two different types of function Q_τ, which locally approximate function f_τ at point x. In the left subfigure, function Q_τ is a simple separable quadratic for which, $H = \lambda_{max}(A^\intercal A)I_n$ in (4), that is frequently used in first-order methods. In the right subfigure, function Q_τ is a non separable quadratic (9) which is used in some of the second-order methods.

Another approach of constructing Q_τ involves the approximation of the ℓ_1-norm with the pseudo-Huber function

$$\psi_\mu(x) = \sum_{i=1}^{n} \Big((\mu^2 + x_i^2)^{\frac{1}{2}} - \mu \Big), \tag{7}$$

where $\mu > 0$ is an approximation parameter. This approach is frequently used by methods that aim in using at every iteration full information from the Hessian matrix $A^\intercal A$, see for example [16,19]. Using (7), problem (1) is replaced with

$$\text{minimize } f_\tau^\mu(x) := \tau\psi_\mu(x) + \frac{1}{2}\|Ax - b\|^2. \tag{8}$$

The smaller μ is the better the approximation of problem (8) to (1). The advantage is that f_τ^μ in (8) is a smooth function which has derivatives of all degrees. Hence, smoothing will allow access to second-order information and essential curvature information will be exploited. However, for very small μ certain problems arise for optimization methods of the form of GFrame, see [16]. For example, the optimal solution of (1) is expected to have many zero components, on the other hand, the optimal solution of (8) is expected to have many nearly zero components. However, for small μ one can expect to obtain a good approximation of the optimal solution of (1) by solving (8). For the smooth problem (8), the convex approximation Q_τ at x is:

$$Q_\tau(y; x) := f_\tau^\mu(x) + \nabla f_\tau^\mu(x)^\intercal (y - x) + \frac{1}{2}(y - x)^\intercal \nabla^2 f_\tau^\mu(x)(y - x). \tag{9}$$

The contours of such a function Q_τ compared to function f_τ are presented in Subfigure 2b. Notice that Q_τ captures the curvature information of function f_τ. However, minimizing the subproblem (3) might be a more expensive operation. Therefore, we rely on an approximate solution of (3) using some iterative method which requires only simple matrix-vector product operations with matrices A and $A^\intercal$. In other words we use only an approximate second-order information. It is frequently claimed [2,5,6,20,38] that second-order methods do not scale favourably with the dimensions of the problem because of the more costly task of solving approximately the subproblems in (3), instead of having an inexpensive closed form solution. Such claims are based on an assumption that *full* second-order information has to be used when solving subproblem (3). Clearly, this is not necessary: an *approximate* second-order information suffices. Studies in [16,19] provided theoretical background as well as the preliminary computational results to illustrate the issue. In this paper, we provide rich computational evidence which demonstrates that second-order methods can be very efficient.

3 Instance Generator

In this section we discuss an instance generator for (1) for the cases $m \geq n$ and $m < n$. The generator is inspired by the one presented in Section 6 of [32]. The advantage of our modified version is that it allows to control the properties of matrix A and the optimal solution x^* of (1). For example, the sparsity of matrix A, its spectral decomposition, the sparsity and the norm of x^*, since A and x^* are defined by the user.

Throughout the paper we will denote the i^{th} component of a vector, by the name of the vector with subscript i. Whilst, the i^{th} column of a matrix is denoted by the name of the matrix with subscript i.

3.1 Instance Generator for $m \geq n$

Given $\tau > 0$, $A \in \mathbb{R}^{m\times n}$ and $x^* \in \mathbb{R}^n$ the generator returns a vector $b \in \mathbb{R}^m$ such that $x^* := \arg\min_x f_\tau(x)$. For simplicity we assume that the given matrix A has rank n. The generator is described in Procedure IGen below.

In procedure IGen, given τ, A and x^* we are aiming in finding a vector b such that x^* satisfies the optimality conditions of problem (1)

$$A^\intercal(Ax^* - b) \in -\tau\partial\|x^*\|_1,$$

where $\partial\|x\|_1 = [-1,1]^n$ is the subdifferential of the ℓ_1-norm at point x. By fixing a subradient $g \in \partial\|x^*\|_1$ as defined in (10) and setting $e = b - Ax^*$, the previous optimality conditions can be written as

$$A^\intercal e = \tau g. \tag{11}$$

1: Initialize $\tau > 0$, $A \in \mathbb{R}^{m\times n}$ with $m \geq n$ and rank n, $x^* \in \mathbb{R}^n$
2: Construct $g \in \mathbb{R}^n$ such that $g \in \partial\|x^*\|_1$:

$$g_i \in \begin{cases} \{1\}, & \text{if } x_i^* > 0 \\ \{-1\}, & \text{if } x_i^* < 0 \\ [-1,1], & \text{if } x_i^* = 0 \end{cases} \quad \forall i = 1, 2, \cdots, n \tag{10}$$

3: Set $e = \tau A(A^\intercal A)^{-1} g$
4: Return $b = Ax^* + e$

Procedure : Instance Generator (IGen)

The solution to the underdetermined system (11) is set to $e = \tau A(A^\intercal A)^{-1}g$ and then we simply obtain $b = Ax^* + e$; Steps 3 and 4 in IGen, respectively. Notice that for a general matrix A, Step 3 of IGen can be very expensive. Fortunately, using elementary linear transformations, such as Givens rotations, we can iteratively construct a sparse matrix A with a known singular value decomposition and guarantee that the inversion of matrix $A^\intercal A$ in Step 3 of IGen is trivial. We provide a more detailed argument in Section 4.

3.2 Instance Generator for $m < n$

In this subsection we extend the instance generator that was proposed in Subsection 3.1 to the case of matrix $A \in \mathbb{R}^{m\times n}$ with more columns than rows, i.e. $m < n$. Given $\tau > 0$, $B \in \mathbb{R}^{m\times m}$, $N \in \mathbb{R}^{m\times n-m}$ and $x^* \in \mathbb{R}^n$ the generator returns a vector $b \in \mathbb{R}^m$ and a matrix $A \in \mathbb{R}^{m\times n}$ such that $x^* := \arg\min_x f_\tau(x)$.

For this generator we need to discuss first some restrictions on matrix A and the optimal solution x^*. Let

$$S := \{i \in \{1, 2, \cdots, n\} \mid x_i^* \neq 0\} \tag{12}$$

with $|S| = s$ and $A_S \in \mathbb{R}^{m\times s}$ be a collection of columns from matrix A which correspond to indices in S. Matrix A_S must have rank s otherwise problem (1) is not well-defined. To see this, let $\text{sign}(x_S^*) \in \mathbb{R}^s$ be the sign function applied component-wise to x_S^*, where x_S^* is a vector with components of x^* that correspond to indices in S. Then problem (1) reduces to the following:

$$\text{minimize } \tau\text{sign}(x_S^*)^\intercal x_s + \frac{1}{2}\|A_S x_s - b\|_2^2, \tag{13}$$

where $x_s \in \mathbb{R}^s$. The first-order stationary points of problem (13) satisfy

$$A_S^\intercal A_S x_s = -\tau\text{sign}(x_S^*) + A_S^\intercal b.$$

If $\text{rank}(A_S) < s$, the previous linear system does not have a unique solution and problem (1) does not have a unique minimizer. Having this restriction in mind, let us now present the instance generator for $m < n$ in Procedure IGen2 below.

1: Initialize $\tau > 0$, $B \in \mathbb{R}^{m\times m}$ with rank m, $N \in \mathbb{R}^{m\times n-m}$, $x^* \in \mathbb{R}^n$ with $S := \{1, 2, \cdots, s\}$ and $s \leq m$
2: Construct $g \in \mathbb{R}^m$ such that $g \in \partial\|x^*(1, 2, \cdots, m)\|_1$:

$$g_i \in \begin{cases} \{1\}, & \text{if } x_i^* > 0 \\ \{-1\}, & \text{if } x_i^* < 0 \\ [-1, 1], & \text{if } x_i^* = 0 \end{cases} \quad \forall i = 1, 2, \cdots, m \tag{14}$$

3: Set $e = \tau B^{-\intercal} g$
4: Construct matrix $\tilde{N} \in \mathbb{R}^{m\times n-m}$ with the following loop:
For $k = 1, 2, \ldots, n - m$

$$\tilde{N}_i = \frac{\xi\tau}{|N_i^\intercal e|} N_i, \text{ where } \xi \text{ is a random variable in } [-1, 1]$$

end-for
5: Return $A = [B, \tilde{N}]$ and $b = Ax^* + e$

Procedure : Instance Generator 2 (IGen2)

In IGen2, given τ, B, N and x^* we are aiming in finding a vector b and a matrix $\tilde{N}$ such that for $A = [B, \tilde{N}]$, x^* satisfies the optimality conditions of problem (1)

$$A^\intercal(Ax^* - b) \in -\tau\partial\|x^*\|_1.$$

Without loss of generality it is assumed that all nonzero components of x^* correspond to indices in $S = \{1, 2, \cdots, s\}$. By fixing a partial subradient $g \in \partial\|x^*(1, 2, \cdots, m)\|_1$ as in (14), where $x^*(1, 2, \cdots, m) \in \mathbb{R}^m$ is a vector which consists of the first m components of x^*, and defining a vector $e = b - Ax^*$, the previous optimality conditions can be written as:

$$e = \tau B^{-\intercal} g \quad \text{and} \quad \tilde{N}^\intercal e \in \tau[-1, 1]^{n-m}. \tag{15}$$

It is easy to check that by defining $\tilde{N}$ as in Step 4 of IGen2 conditions (15) are satisfied. Finally, we obtain $b = Ax^* + e$.

Similarly to IGen in Subsection (3.1), for Step 3 in IGen2 we have to perform a matrix inversion, which generally can be an expensive operation. However, in the next section we discuss techniques how this matrix inversion can be executed using a sequence of elementary orthogonal transformations.

4 Construction of matrix A

In this subsection we provide a paradigm on how matrix A can be inexpensively constructed such that its singular value decomposition is known and its sparsity is controlled. We examine the case of instance generator IGen where $m \geq n$. The paradigm can be easily extended to the case of IGen2, where $m < n$.

Let $\Sigma \in \mathbb{R}^{m\times n}$ be a rectangular matrix with the singular values $\sigma_1, \sigma_2, \cdots, \sigma_n$ on its diagonal and zeros elsewhere:

$$\Sigma = \begin{bmatrix} \mathrm{diag}(\sigma_1, \sigma_2, \ldots, \sigma_n) \\ \hline O_{m-n\times n} \end{bmatrix},$$

where $O_{m-n\times n} \in \mathbb{R}^{m-n\times n}$ is a matrix of zeros, and let $G(i,j,\theta) \in \mathbb{R}^{n\times n}$ be a Givens rotation matrix, which rotates plane i-j by an angle θ:

$$G(i,j,\theta) = \begin{bmatrix} 1 & \cdots & 0 & \cdots & 0 & \cdots & 0 \\ \vdots & \ddots & \vdots & & \vdots & & \vdots \\ 0 & \cdots & c & \cdots & -s & \cdots & 0 \\ \vdots & & \vdots & \ddots & \vdots & & \vdots \\ 0 & \cdots & s & \cdots & c & \cdots & 0 \\ \vdots & & \vdots & & \vdots & \ddots & \vdots \\ 0 & \cdots & 0 & \cdots & 0 & \cdots & 1 \end{bmatrix},$$

where $i, j \in \{1, 2, \cdots, n\}$, $c = \cos\theta$ and $s = \sin\theta$. Given a sequence of Givens rotations $\{G(i_k, j_k, \theta_k)\}_{k=1}^{K}$ we define the following composition of them:

$$G = G(i_1, j_1, \theta_1)G(i_2, j_2, \theta_2)\cdots G(i_K, j_K, \theta_K).$$

Similarly, let $\tilde{G}(l, p, \vartheta) \in \mathbb{R}^{m\times m}$ be a Givens rotation matrix where $l, p \in \{1, 2, \cdots, m\}$ and

$$\tilde{G} = \tilde{G}(l_1, p_1, \vartheta_1)\tilde{G}(l_2, p_2, \vartheta_2)\cdots\tilde{G}(l_{\tilde{K}}, p_{\tilde{K}}, \vartheta_{\tilde{K}})$$

be a composition of $\tilde{K}$ Givens rotations. Using G and $\tilde{G}$ we define matrix A as

$$A = (P_1\tilde{G}P_2)\Sigma G^\intercal, \tag{16}$$

where $P_1, P_2 \in \mathbb{R}^{m\times m}$ are permutation matrices. Since the matrices $P_1\tilde{G}P_2$ and G are orthonormal it is clear that the left singular vectors of matrix A are the columns of $P_1\tilde{G}P_2$, Σ is the matrix of singular values and the right singular vectors are the columns of G. Hence, in Step 3 of IGen we simply set $(A^\intercal A)^{-1} = G(\Sigma^\intercal\Sigma)^{-1}G^\intercal$, which means that Step 3 in IGen costs two matrix-vector products with G and a diagonal scaling with $(\Sigma^\intercal\Sigma)^{-1}$. Moreover, the sparsity of matrix A is controlled by the numbers K and $\tilde{K}$ of Givens rotations, the type, i.e. (i, j, θ) and (l, p, ϑ), and the order of Givens rotations. Also, notice that the sparsity of matrix $A^\intercal A$ is controlled only by matrix G. Examples are given in Subsection 4.1.

It is important to mention that other settings of matrix A in (16) could be used, for example different combinations of permutation matrices and Givens rotations. The setting chosen in (16) is flexible, it allows for an inexpensive construction of matrix A and makes the control of the singular value decomposition and the sparsity of matrices A and $A^\intercal A$ easy.

Notice that matrix A does not have to be calculated and stored. In particular, in case that the method which is applied to solve problem (1) requires only matrix-vector product operations using matrices A and $A^\intercal$, one can simply consider matrix A as an operator. It is only required to predefine the triplets (i_k, j_k, θ_k) $\forall k = 1, 2, \cdots, K$ for matrix G, the triplets (l_k, p_k, θ_k) $\forall k = 1, 2, \cdots, \tilde{K}$ for matrix $\tilde{G}$ and the permutation matrices P_1 and P_2. The previous implies that the generator is inexpensive in terms of memory requirements. Examples of matrix-vector product operations with matrices A and $A^\intercal$ in case of (16) are given below in Algorithms MvPA and MvPAt, respectively.

Given a matrix A defined as in (16) and an input vector $x \in \mathbb{R}^n$, do
Set $y_0 = x$
For $k = 1, 2, \ldots, K$
    $y_k = G_k^\intercal y_{k-1}$
end-for
Set $\tilde{y}_0 = P_2 \Sigma y_K$
For $k = 1, 2, \ldots, \tilde{K}$
    $\tilde{y}_k = \tilde{G}_{\tilde{K}-k+1} \tilde{y}_{k-1}$
end-for
Return $P_1 \tilde{y}_{\tilde{K}}$

Algorithm : Matrix-vector product with A (MvPA)

Given a matrix A defined as in (16) and input vector $y \in \mathbb{R}^m$, do
Set $\tilde{x}_0 = P_1^\intercal y$
For $k = 1, 2, \ldots, \tilde{K}$
    $\tilde{x}_k = \tilde{G}_k^\intercal \tilde{x}_{k-1}$
end-for
Set $x_0 = \Sigma^\intercal P_2^\intercal \tilde{x}_{\tilde{K}}$
For $k = 1, 2, \ldots, K$
    $x_k = G_{K-k+1} x_{k-1}$
end-for
Return x_K

Algorithm : Matrix-vector product with $A^\intercal$ (MvPAt)

4.1 An example using Givens rotation

Let us assume that m, n are divisible by two and $m \geq n$. Given the singular values matrix Σ and rotation angles θ and ϑ, we construct matrix A as

$$A = (P\tilde{G}P)\Sigma G^\intercal,$$

where P is a random permutation of the identity matrix, G is a composition of $n/2$ Givens rotations:

$$G = G(i_1, j_1, \theta)G(i_2, j_2, \theta) \cdots, G(i_k, j_k, \theta), \cdots, G(i_{n/2}, j_{n/2}, \theta)$$

with

$$i_k = 2k-1, \quad j_k = 2k \quad \text{for} \quad k = 1, 2, 3, \cdots, \frac{n}{2}$$

and $\tilde{G}$ is a composition of $m/2$ Givens rotations:

$$\tilde{G} = \tilde{G}(l_1, p_1, \vartheta)\tilde{G}(l_2, p_2, \vartheta)\cdots, \tilde{G}(l_k, p_k, \vartheta), \cdots, \tilde{G}(l_{m/2}, p_{m/2}, \vartheta)$$

with

$$l_k = 2k-1, \quad p_k = 2k \quad \text{for} \quad k = 1, 2, 3, \cdots, \frac{m}{2}.$$

Notice that the angle θ is the same for all Givens rotations G_k, this means that the total memory requirement for matrix G is low. In particular, it consists only of the storage of a 2×2 rotation matrix. Similarly, the memory requirement for matrix $\tilde{G}$ is also low.

4.2 Control of sparsity of matrix A and $A^\intercal A$

We now present examples in which we demonstrate how sparsity of matrix A can be controlled through Givens rotations.

In the example of Subsection 4.1, two compositions of $n/2$ and $m/2$ Givens rotations, denoted by G and $\tilde{G}$, are applied on an initial diagonal rectangular matrix Σ. If $n = 2^3$ and $m = 2n$ the sparsity pattern of the resulting matrix $A = (P\tilde{G}P)\Sigma G^\intercal$ is given in Subfigure 3a and has 28 nonzero elements, while the sparsity pattern of matrix $A^\intercal A$ is given in Subfigure 4a and has 16 nonzero elements. Notice in this subfigure that the coordinates can be clustered in pairs of coordinates $(1, 2)$, $(3, 4)$, $(5, 6)$ and $(7, 8)$. One could apply another stage of Givens rotations. For example, one could construct matrix $A = (P\tilde{G}\tilde{G}_2P)\Sigma(G_2G)^\intercal$, where

$$G_2 = G(i_1, j_1, \theta)G(i_2, j_2, \theta)\cdots, G(i_k, j_k, \theta), \cdots, G(i_{n/2-1}, j_{n/2-1}, \theta)$$

with

$$i_k = 2k, \quad j_k = 2k+1 \quad \text{for} \quad k = 1, 2, 3, \cdots, \frac{n}{2} - 1.$$

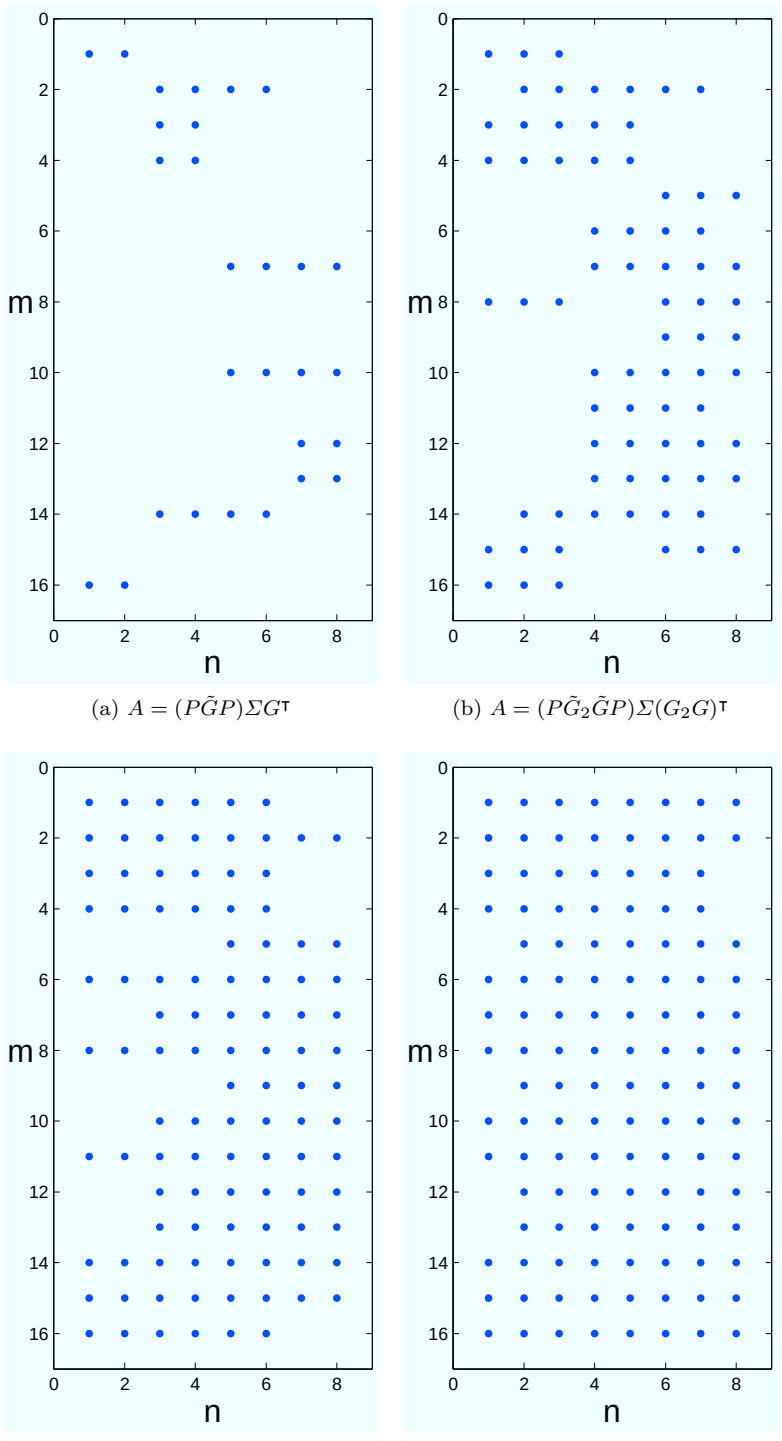
and

$$\tilde{G}_2 = \tilde{G}(l_1, p_1, \theta)\tilde{G}(l_2, p_2, \theta)\cdots, \tilde{G}(l_k, p_k, \theta), \cdots, \tilde{G}(l_{m/2-1}, p_{m/2-1}, \theta)$$

with

$$l_k = 2k, \quad p_k = 2k+1 \quad \text{for} \quad k = 1, 2, 3, \cdots, \frac{m}{2} - 1.$$

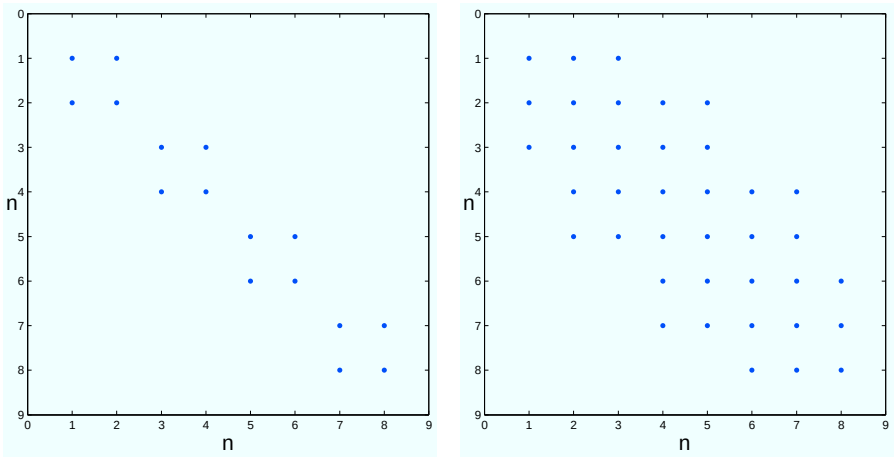
Matrix $A = (P\tilde{G}\tilde{G}_2P)\Sigma(G_2G)^\intercal$ has 74 nonzeros and it is shown in Subfigure 3b, while matrix $A^\intercal A$ has 38 nonzeros and it is shown in Subfigure 4b. By rotating again we obtain the matrix $A = (P\tilde{G}\tilde{G}_2\tilde{G}P)\Sigma(GG_2G)^\intercal$ in Subfigure 3c with 104 nonzero elements and matrix $A^\intercal A$ in Subfigure 4c with 56 nonzero elements. Finally, the fourth Subfigures 3d and 4d show matrix $A = (P\tilde{G}_2\tilde{G}\tilde{G}_2\tilde{G}P)\Sigma(G_2GG_2G)^\intercal$ and $A^\intercal A$ with 122 and 62 nonzero elements, respectively.



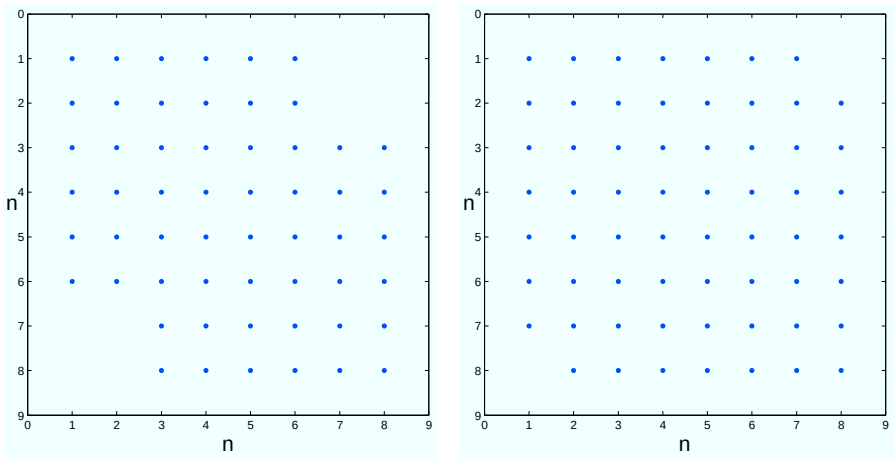
(a) $A = (P\tilde{G}P)\Sigma G^{\intercal}$ (b) $A = (P\tilde{G}_2\tilde{G}P)\Sigma(G_2G)^{\intercal}$

(c) $A = (P\tilde{G}\tilde{G}_2\tilde{G}P)\Sigma(GG_2G)^{\intercal}$ (d) $A = (P\tilde{G}_2\tilde{G}\tilde{G}_2\tilde{G}P)\Sigma(G_2GG_2G)^{\intercal}$

Fig. 3: Sparsity pattern of four examples of matrix A, the Givens rotations G and G_2 are explained in Subsections 4.1 and 4.2.



(a) $A^\intercal A$, $A = (P\tilde{G}P)\Sigma G^\intercal$ (b) $A^\intercal A$, $A = (P\tilde{G}_2\tilde{G}P)\Sigma(G_2G)^\intercal$



(c) $A^\intercal A$, $A = (P\tilde{G}\tilde{G}_2\tilde{G}P)\Sigma(GG_2G)^\intercal$ (d) $A^\intercal A$, $A = (P\tilde{G}_2\tilde{G}\tilde{G}_2\tilde{G}P)\Sigma(G_2GG_2G)^\intercal$

Fig. 4: Sparsity pattern of four examples of matrix $A^\intercal A$, where the Givens rotations G and G_2 are explained in Subsections 4.1 and 4.2

.

5 Conditioning of the problem

Let us now precisely define how we measure the conditioning of problem (1). For simplicity, throughout this section we assume that matrix A has more rows than columns, $m \geq n$, and it is full-rank. Extension to the case of matrix A with more columns than rows is easy and we briefly discuss this at the end of this section.

We denote with span$(\cdot)$ the span of the columns of the input matrix. Moreover, S is defined in (12), S^c is its complement.

Two factors are considered that affect the conditioning of the problem. First, the usual condition number of the second-order derivative of $1/2\|Ax - b\|_2^2$ in (1), which is simply $\kappa(A^\intercal A) = \lambda_1(A^\intercal A)/\lambda_n(A^\intercal A)$, where $0 < \lambda_n \leq \lambda_{n-1} \leq \cdots \leq \lambda_1$ are the eigenvalues of matrix $A^\intercal A$. It is well-known that the larger $\kappa(A^\intercal A)$ is, the more difficult problem (1) becomes.

Second, the conditioning of the optimal solution x^* of problem (1). Let us explain what we mean by the conditioning of x^*. We define a constant $\rho > 0$ and the index set $\mathcal{I}_\rho := \{i \in \{1, 2, \cdots, n\} \mid \lambda_i(A^\intercal A) \geq \rho\}$. Furthermore, we define the projection $P_\rho = G_\rho G_\rho^\intercal$, where $G_\rho \in \mathbb{R}^{n \times r}$, $r = |\mathcal{I}_\rho|$ and matrix G_ρ has as columns the eigenvectors of matrix $A^\intercal A$ which correspond to eigenvalues with indices in $\mathcal{I}_\rho$. Then, the conditioning of x^* is defined as

$$\kappa_\rho(x^*) = \begin{cases} \frac{\|x^*\|_2}{\|P_\rho x^*\|_2}, & \text{if } P_\rho x^* \neq 0 \\ +\infty, & \text{otherwise} \end{cases} \tag{17}$$

For the case $P_\rho x^* \neq 0$, the denominator of (17) is the mass of x^* which exists in the space spanned by eigenvectors of $A^\intercal A$ which correspond to eigenvalues that are larger than or equal to ρ.

Let us assume that there exists some ρ which satisfies $\lambda_n(A^\intercal A) \leq \rho \ll \lambda_1(A^\intercal A)$. If $\kappa_\rho(x^*)$ is large, i.e., $\|P_\rho x^*\|_2$ is close to zero, then the majority of the mass of x^* is "hidden" in the space spanned by eigenvectors which correspond to eigenvalues that are smaller than ρ, i.e., the orthogonal space of span(G_ρ). In Section 1 we referred to methods that do not incorporate information which correspond to small eigenvalues of $A^\intercal A$. Therefore, if the previous scenario holds, then we expect the performance of such methods to degrade. In Section 8 we empirically verify the previous arguments.

If matrix A has more columns than rows then the previous definitions of conditioning of problem (1) are incorrect and need to be adjusted. Indeed, if $m < n$ and $\text{rank}(A) = \min(m, n) = m$, then $A^\intercal A$ is a rank deficient matrix which has m nonzero eigenvalues and $n - m$ zero eigenvalues. However, we can restrict the conditioning of the problem to a neighbourhood of the optimal solution of x^*. In particular, let us define a neighbourhood of x^* so that all points in this neighbourhood have nonzeros at the same indices as x^* and zeros elsewhere, i.e. $\mathcal{N} := \{x \in \mathbb{R}^n \mid x_i \neq 0 \ \forall i \in S, \ x_i = 0 \ \forall i \in S^c\}$. In this case an important feature to determine the conditioning of the problem is the ratio of the largest and the smallest nonzero eigenvalues of $A_S^\intercal A_S$, where A_S is a submatrix of A built of columns of A which belong to set S.

6 Construction of the optimal solution

Two different techniques are employed to generate the optimal solution x^* for the experiments presented in Section 8. The first procedure suggests a simple random generation of x^*, see Procedure OsGen below.

1: Given the required number $s \leq \min(m, n)$ of nonzeros in x^* and a positive constant $\gamma > 0$ do:
2: Choose a subset $S \subseteq \{1, 2, \cdots, n\}$ with $|S| = s$.
3: $\forall i \in S$ choose x_i^* uniformly at random in $[-\gamma, \gamma]$ and $\forall j \notin S$ set $x_j^* = 0$.

Procedure : Optimal solution Generator (OsGen)

The second and more complicated approach is given in Procedure OsGen2. This procedure is applicable only in the case that $m \geq n$, however, it can be easily extended to the case of $m < n$. We focus in the former scenario since all experiments in Section 8 are generated by setting $m \geq n$.

1: Given the required number $s \leq \min(m, n)$ of nonzeros in x^*, a positive constant $\gamma > 0$, the right singular vectors G and singular values Σ of matrix A do:
2: Solve approximately

$$x^* := \underset{\substack{x \in \mathbb{R}^n \\ \text{subject to: } \|x\|_0 \leq s,}}{\arg\min} \quad \|G^\intercal x - \gamma(\Sigma^\intercal \Sigma)^{-1} 1_n\|^2 \tag{18}$$

where $1_n \in \mathbb{R}^n$ is a vector of ones and $\|\cdot\|_0$ is the zero norm which returns the number of nonzero components of the input vector. Problem (18) can be solved approximately using an Orthogonal Matching Pursuit (OMP) [28] solver implemented in [3].

Procedure : Optimal solution Generator 2 (OsGen2)

The aim of Procedure OsGen2 is to find a sparse x^* with $\kappa_\rho(x^*)$ arbitrarily large for some ρ in the interval $\lambda_n(A^\intercal A) \leq \rho \ll \lambda_1(A^\intercal A)$. In particular, OsGen2 will return a sparse x^* which can be expressed as $x^* = Gv$. The coefficients v are close to the inverse of the eigenvalues of matrix $A^\intercal A$. Intuitively, this technique will create an x^* which has strong dependence on subspaces which correspond to small eigenvalues of $A^\intercal A$. The constant γ is used in order to control the norm of x^*.

The sparsity constraint in problem (18), i.e., $\|x\|_0 \leq s$, makes the approximate solution of this problem difficult when we use OMP, especially in the case that s and n are large. To avoid this expensive task we can ignore the sparsity constraint in (18). Then we can solve exactly and inexpensively the unconstrained problem and finally we can project the obtained solution in the feasible set defined by the sparsity constraint. Obviously, there is no guarantee that the projected solution is a good approximation to the one obtained in Step 2 of Procedure OsGen2. However, for all experiments in Section 8 that we applied this modification we obtained sufficiently large $\kappa_\rho(x^*)$. This means that our objective to produce ill-conditioned optimal solutions was met, while we kept the computational costs low. The modified version of Procedure OsGen2 is given in Procedure OsGen3.

1: Given the required number $s \leq \min(m, n)$ of nonzeros in x^*, two non negative integers s_1 and s_2 such that $s_1 + s_2 = s$, a positive constant $\gamma > 0$, the right singular vectors G and singular values Σ of matrix A do:
2: Solve exactly

$$x^* := \arg\min_{x \in \mathbb{R}^n} \|G^\intercal x - \gamma(\Sigma^\intercal \Sigma)^{-1} 1_n\|^2 \tag{19}$$

where $1_n \in \mathbb{R}^n$ is a vector of ones. Problem (19) can be solved exactly and inexpensively because $G^\intercal$ is an orthonormal matrix.
3: Maintain the positions and the values of the s_1 smallest and s_2 largest (in absolute values) components of x^*.
4: Set the remaining components of x^* to zero.

Procedure : Optimal solution Generator 3 (OsGen3)

7 Existing Problem Generators

So far in Section 3.1 we have described in details our proposed problem generator. Moreover, in Section 4 we have described how to construct matrices A such that the proposed generator is scalable with respect to the number of unknown variables. We now briefly describe existing problem generators and explain how our propositions add value to the existing approaches.

Given a regularization parameter τ existing problem generators are looking for A, b and x^* such that the optimality conditions of problem (1):

$$A^\intercal(Ax^* - b) \in -\tau\partial\|x^*\|_1, \tag{20}$$

are satisfied. For example, in [32] the author fixes a vector of noise e and an optimal solution x^* and then finds A and b such that (20) is satisfied. In particular, in [32] matrix $A = BD$ is used, where B is a fixed matrix and D is a scaling matrix such that the following holds.

$$(BD)^\intercal e \in \tau\partial\|x^*\|_1,$$

Matrix D is trivial to calculate, see Section 6 in [32] for details. Then by setting $b = Ax^* + e$ (20) is satisfied. The advantage of this generator is that it allows control of the noise vector e, in comparison to our approach where the vector noise e has to be determined by solving a linear system. On the other hand, one does not have direct control over the singular value decomposition of matrix A, since this depends on matrix D, which is determined based on the fixed vectors e and x^*.

Another representative example is proposed in [26]. This generator, which we discovered during the revision of our paper, proposes the same setting as in our paper. In particular, given A, x^* and τ one can construct a vector b (or a noise vector e) such that (20) is satisfied. However, in [26] the author suggests that b can be found using a simple iterative procedure. Depending on matrix A and how ill-conditioned it is, this procedure might be slow. In this paper, we suggest that one can rely on numerical linear algebra tools, such as Givens rotation, in order to inexpensively construct b (or a noise vector e) using straightforwardly scalable operations. Additionally, we show in Section

(8) that a simple construction of matrix A is sufficient to extensively test the performance of methods.

8 Numerical Experiments

In this section we study the performance of state-of-the-art first- and second-order methods as the conditioning and the dimensions of the problem increase. The scripts that reproduce the experiments in this section as well as the problem generators that are described in Section 3 can be downloaded from: `http://www.maths.ed.ac.uk/ERGO/trillion/`.

8.1 State-of-the-art Methods

A number of efficient first- [2,13,24,34,35,38–41,44,45] and second-order [4, 10,16,19,25,36,37] methods have been developed for the solution of problem (1). In this section we examine the performance of the following state-of-the-art methods. Notice that the first three methods FISTA, PSSgb and PCDM do not perform smoothing of the ℓ_1-norm, while pdNCG does.

- FISTA (Fast Iterative Shrinkage-Thresholding Algorithm) [2] is an optimal first-order method for problem (1), which adheres to the structure of GFrame. At a point x, FISTA builds a convex function:

$$Q_\tau(y;x) := \tau\|y\|_1 + \frac{1}{2}\|Ax-b\|^2 + (A^\intercal(Ax-b))^\intercal(y-x) + \frac{L}{2}\|y-x\|_2^2,$$

  where L is an upper bound of $\lambda_{max}(A^\intercal A)$, and solves subproblem (3) exactly using shringkage-thresholding [12,27]. An efficient implementation of this algorithm can be found as part of TFOCS (Templates for First-Order Conic Solvers) package [6] under the name N83. In this implementation the parameter L is calculated dynamically.
- PCDM (Parallel Coordinate Descent Method) [35] is a randomized parallel coordinate descent method. The parallel updates are performed asynchronously and the coordinates to be updated are chosen uniformly at random. Let ϖ be the number of processors that are employed by PCDM. Then, at a point x, PCDM builds ϖ convex approximations:

$$Q_\tau^i(y_i;x) := \tau|y_i| + \frac{1}{2}\|Ax-b\|^2 + (A_i^\intercal(Ax-b))(y_i-x_i) + \frac{\beta L_i}{2}(y_i-x_i)^2,$$

  $\forall i = 1,2,\cdots,\varpi$, where A_i is the ith column of matrix A and $L_i = (A^\intercal A)_{ii}$ is the ith diagonal element of matrix $A^\intercal A$ and β is a positive constant which is defined in Subsection 8.3. The Q_τ^i functions are minimized exactly using shrinkage-thresholding.

– PSSgb (Projected Scaled Subgradient, Gafni-Bertsekas variant) [37] is a second-order method. At each iteration of PSSgb the coordinates are separated into two sets, the working set $\mathcal{W}$ and the active set $\mathcal{A}$. The working set consists of all coordinates for which, the current point x is nonzero. The active set is the complement of the working set $\mathcal{W}$. The following local quadratic model is build at each iteration

$$Q_\tau(y;x) := f_\tau(x) + (\tilde{\nabla} f_\tau(x))^\intercal (y-x) + \frac{1}{2}(y-x)^\intercal H(y-x),$$

where $\tilde{\nabla} f_\tau(x)$ is a sub-gradient of f_τ at point x with the minimum Euclidean norm, see Subsection 2.2.1 in [37] for details. Moreover, matrix H is defined as:

$$H = \begin{bmatrix} H_{\mathcal{W}} & 0 \\ 0 & H_{\mathcal{A}} \end{bmatrix},$$

where $H_{\mathcal{W}}$ is an L-BFGS (Limited-memory Broyden-Fletcher-Goldfarb-Shanno) Hessian approximation with respect to the coordinates $\mathcal{W}$ and $H_{\mathcal{A}}$ is a positive diagonal matrix. The diagonal matrix $H_{\mathcal{A}}$ is a scaled identity matrix, where the Shanno-Phua/Barzilai-Borwein scaling is used, see Subsection 2.3.1 in [37] for details. The local model is minimized exactly since the inverse of matrix H is known due to properties of the L-BFGS Hessian approximation $H_{\mathcal{W}}$.

– pdNCG (primal-dual Newton Conjugate Gradients) [16] is also a second-order method. At every point x pdNCG constructs a convex function Q_τ exactly as described for (9). The subproblem (3) is solved *inexactly* by reducing it to the linear system:

$$\nabla^2 f_\tau^\mu(x)(y-x) = -\nabla f_\tau^\mu(x),$$

which is solved approximately using preconditioned Conjugate Gradients (PCG). A simple diagonal preconditioner is used for all experiments. The preconditioner is the inverse of the diagonal of matrix $\nabla^2 f_\tau^\mu(x)$.

8.2 Implementation details

Solvers pdNCG, FISTA and PSSgb are implemented in MATLAB, while solver PCDM is a C++ implementation. We expect that the programming language will not be an obstacle for pdNCG, FISTA and PSSgb. This is because these methods rely only on basic linear algebra operations, such as the dot product, which are implemented in *C++* in MATLAB by default. The experiments in Subsections 8.4, 8.5, 8.6 were performed on a Dell PowerEdge R920 running Redhat Enterprise Linux with four Intel Xeon E7-4830 v2 2.2GHz processors, 20MB Cache, 7.2 GT/s QPI, Turbo (4x10Cores).

The huge scale experiments in Subsection 8.9 were performed on a Cray XC30 MPP supercomputer. This work made use of the resources provided by ARCHER (`http://www.archer.ac.uk/`), made available through the Edinburgh Compute and Data Facility (ECDF) (`http://www.ecdf.ed.ac.uk/`).

According to the most recent list of commercial supercomputers, which is published in TOP500 list (`http://www.top500.org`), ARCHER is currently the $25th$ fastest supercomputer worldwide out of 500 supercomputers. ARCHER has a total of $118,080$ cores with performance $1,642.54$ TFlops/s on LINPACK benchmark and $2,550.53$ TFlops/s theoretical peak perfomance. The most computationally demanding experiments which are presented in Subsection 8.9 required more than half of the cores of ARCHER, i.e., $65,536$ cores out of $118,080$.

8.3 Parameter tuning

We describe the most important parameters for each solver, any other parameters are set to their default values. For pdNCG we set the smoothing parameter to $\mu = 10^{-5}$, this setting allows accurate solution of the original problem with an error of order $\mathcal{O}(\mu)$ [31]. For pdNCG, PCG is terminated when the relative residual is less that 10^{-1} and the backtracking line-search is terminated if it exceeds 50 iterations. Regarding FISTA the most important parameter is the calculation of the Lipschitz constant L, which is handled dynamically by TFOCS. For PCDM the coordinate Lipschitz constants L_i $\forall i = 1, 2, \cdots, n$ are calculated exactly and parameter $\beta = 1 + (\omega - 1)(\varpi - 1)/(n - 1)$, where ω changes for every problem since it is the degree of partial separability of the fidelity function in (1), which is easily calculated (see [35]), and $\varpi = 40$ is the number of cores that are used. For PSSgb we set the number of L-BFGS corrections to 10.

We set the regularization parameter $\tau = 1$, unless stated otherwise. We run pdNCG for sufficient time such that the problems are adequately solved. Then, the rest of the methods are terminated when the objective function f_τ in (1) is below the one obtained by pdNCG or when a predefined maximum number of iterations limit is reached. All comparisons are presented in figures which show the progress of the objective function against the wall clock time. This way the reader can compare the performance of the solvers for various levels of accuracy. We use logarithmic scales for the wall clock time and terminate runs which do not converge in about 10^5 seconds, i.e., approximately 27 hours.

8.4 Increasing condition number of $A^\intercal A$

In this experiment we present the performance of FISTA, PCDM, PSSgb and pdNCG for increasing condition number of matrix $A^T A$ when Procedure OsGen is used to construct the optimal solution x^*. We generate six matrices A and two instances of x^* for every matrix A; twelve instances in total.

The singular value decomposition of matrix A is $A = \Sigma G^\intercal$, where Σ is the matrix of singular values, the columns of matrices I_m and G are the left and right singular vectors, respectively, see Subsection 4.1 for details about the construction of matrix G. The singular values of matrices A are chosen

uniformly at random in the intervals $[0, 10^q]$, where $q = 0, 1, \cdots, 5$, for each of the six matrices A. Then, all singular values are shifted by 10^{-1}. The previous resulted in a condition number of matrix $A^\intercal A$ which varies from 10^2 to 10^{12} with a step of times 10^2. The rotation angle θ of matrix G is set to $2\pi/3$ radians. Matrices A have $n = 2^{22}$ columns, $m = 2n$ rows and rank n. The optimal solutions x^* have $s = n/2^7$ nonzero components for all twelve instances.

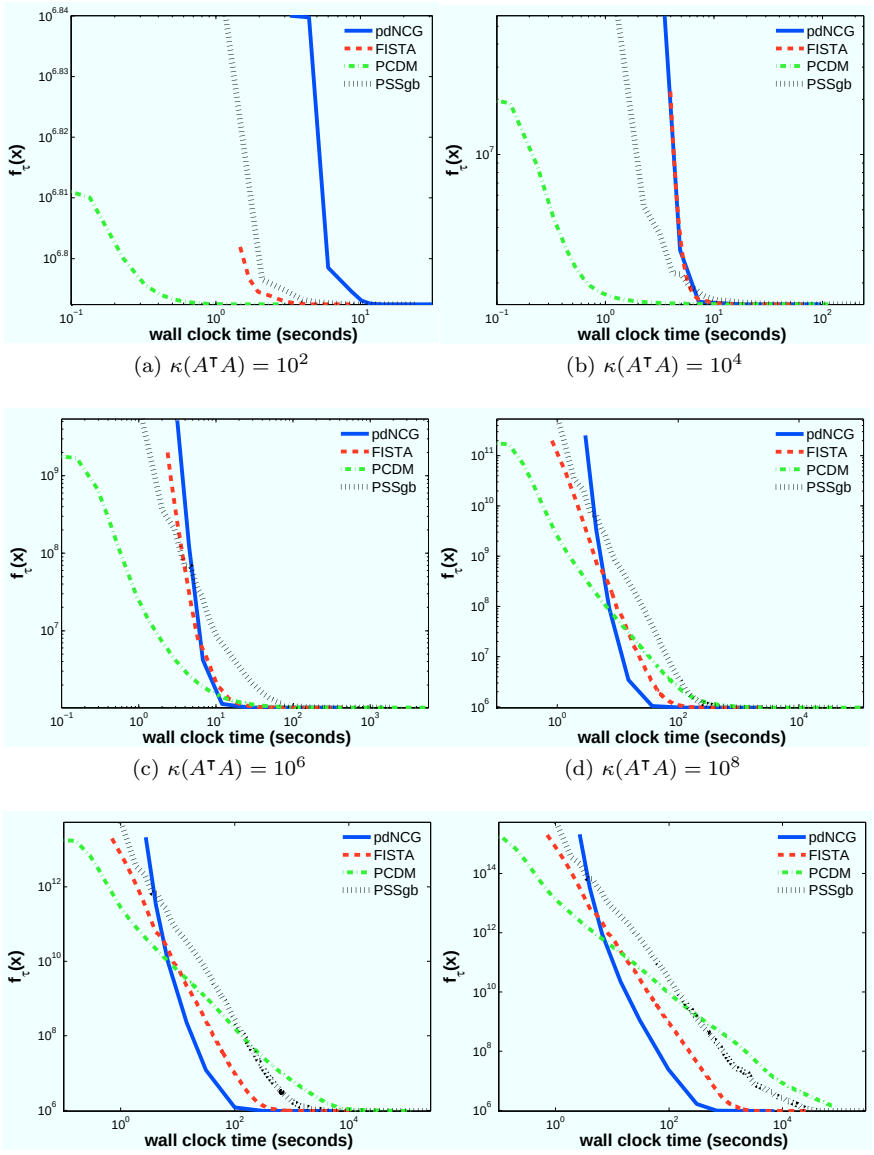
For the first set of six instances we set $\gamma = 10$ in OsGen, which resulted in $\kappa_{0.1}(x^*) \approx 1$ for all experiments. The results are presented in Figure 5. For these instances PCDM is clearly the fastest for $\kappa(A^\intercal A) \leq 10^4$, while for $\kappa(A^\intercal A) \geq 10^6$ pdNCG is the most efficient.

For the second set of six instances we set $\gamma = 10^3$ in Procedure OsGen, which resulted in the same $\kappa_{0.1}(x^*)$ as before for every matrix A. The results are presented in Figure 6. For these instances PCDM is the fastest for very well conditioned problems with $\kappa(A^\intercal A) \leq 10^2$, while pdNCG is the fastest for $\kappa(A^\intercal A) \geq 10^4$.

We observed that pdNCG required at most 30 iterations to converge for all experiments. For FISTA, PCDM and PSSgb the number of iterations was varying between thousands and tens of thousands iterations depending on the condition number of matrix $A^\intercal A$; the larger the condition number the more the iterations. However, the number of iterations is not a fair metric to compare solvers because every solver has different computational cost per iteration. In particular, FISTA, PCDM and PSSgb perform few inner products per iteration, which makes every iteration inexpensive, but the number of iterations is sensitive to the condition number of matrix $A^\intercal A$. On the other hand, for pdNCG the empirical iteration complexity is fairly stable, however, the number of inner products per iteration (mainly matrix-vector products with matrix A) may increase as the condition number of matrix $A^\intercal A$ increases. Inner products are the major computational burden at every iteration for all solvers, therefore, the faster an algorithm converged in terms of wall-clock time the less inner products that are calculated. In Figures 5 and 6 we display the objective evaluation against wall-clock time (log-scale) to facilitate the comparison of different algorithms.

8.5 Increasing condition number of $A^\intercal A$: non-trivial construction of x^*

In this experiment we examine the performance of the methods as the condition number of matrix $A^\intercal A$ increases, while the optimal solution x^* is generated using Procedure OsGen3 (instead of OsGen) with $\gamma = 100$ and $s_1 = s_2 = s/2$. Two classes of instances are generated, each class consists of four instances (A, x^*) with $n = 2^{22}$, $m = 2n$ and $s = n/2^7$. Matrix A is constructed as in Subsection 8.4. The singular values of matrices A are chosen uniformly at random in the intervals $[0, 10^q]$, where $q = 0, 1, \cdots, 3$, for all generated matrices A. Then, all singular values are shifted by 10^{-1}. The previous resulted in a condition number of matrix $A^\intercal A$ which varies from 10^2 to

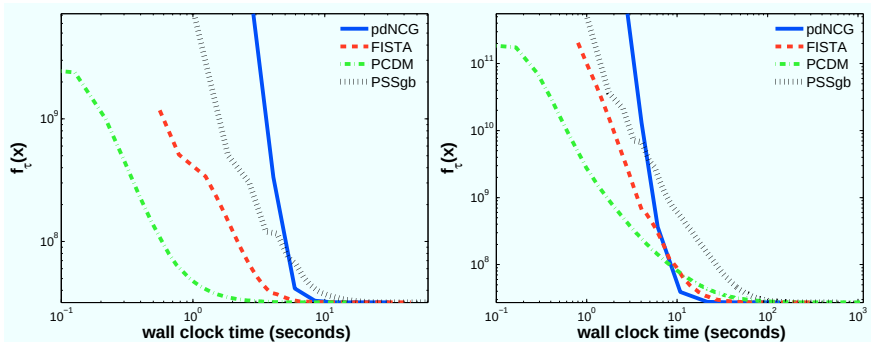


(a) $\kappa(A^\intercal A) = 10^2$ (b) $\kappa(A^\intercal A) = 10^4$

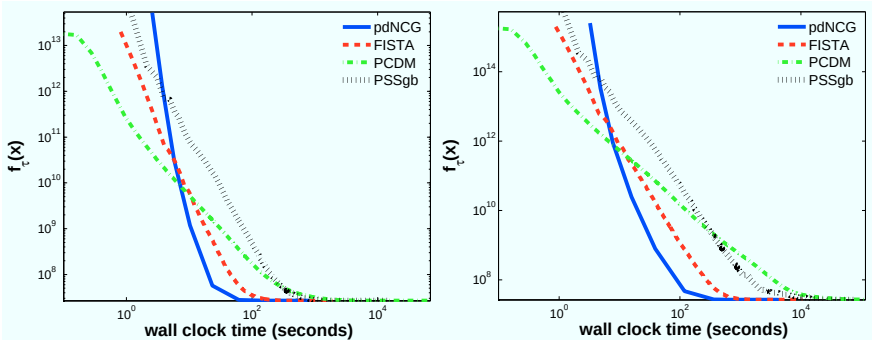
(c) $\kappa(A^\intercal A) = 10^6$ (d) $\kappa(A^\intercal A) = 10^8$

(e) $\kappa(A^\intercal A) = 10^{10}$ (f) $\kappa(A^\intercal A) = 10^{12}$

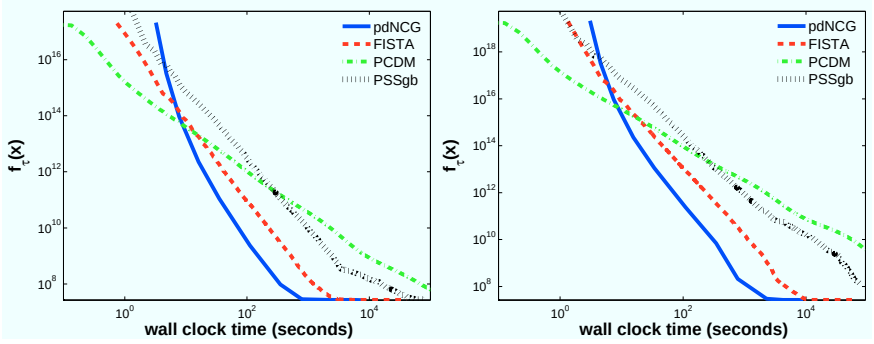
Fig. 5: Performance of pdNCG, FISTA, PCDM and PSSgb on synthetic S-LS problems for increasing condition number of matrix $A^\intercal A$ and $\gamma = 10$ in Procedure OsGen. The axes are in log-scale. In this figure f_τ denotes the objective value that was obtained by each solver.



(a) $\kappa(A^\intercal A) = 10^2$ (b) $\kappa(A^\intercal A) = 10^4$



(c) $\kappa(A^\intercal A) = 10^6$ (d) $\kappa(A^\intercal A) = 10^8$



(e) $\kappa(A^\intercal A) = 10^{10}$ (f) $\kappa(A^\intercal A) = 10^{12}$

Fig. 6: Performance of pdNCG, FISTA, PCDM and PSSgb on a synthetic S-LS problem for increasing condition number of matrix $A^\intercal A$ and $\gamma = 10^3$ in Procedure OsGen. The axes are in log-scale.

10^8 with a step of times 10^2. The condition number of the generated optimal solutions was on average $\kappa_{0.1}(x^*) \approx 40$.

The two classes of experiments are distinguished based on the rotation angle θ that is used for the composition of Givens rotations G. In particular, for the first class of experiments the angle is $\theta = 2\pi/10$ radians, while for the second class of experiments the rotation angle is $\theta = 2\pi/10^3$ radians. The difference between the two classes is that the second class consists of matrices $A^\intercal A$ for which, a major part of their mass is concentrated in the diagonal. This setting is beneficial for PCDM since it uses information only from the diagonal of matrices $A^\intercal A$. This setting is also beneficial for pdNCG since it uses a diagonal preconditioner for the inexact solution of linear systems at every iteration.

The results for the first class of experiments are presented in Figure 7. For instances with $\kappa(A^\intercal A) \geq 10^6$ PCDM was terminated after $1,000,000$ iterations, which corresponded to more than 27 hours of wall-clock time.

The results for the second class of experiments are presented in Figure 8. Notice in this figure that the objective function is only slightly reduced. This does not mean that the initial solution, which was the zero vector, was nearly optimal. This is because noise with large norm, i.e., $\|Ax^* - b\|$ is large, was used in these experiments, therefore, changes in the optimal solution did not have large affect on the objective function.

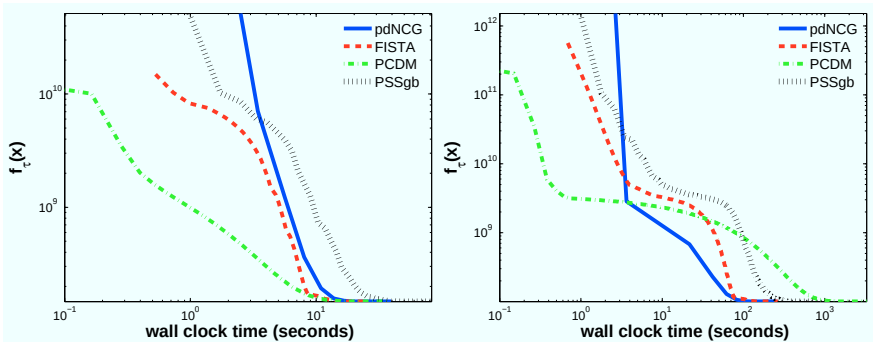
8.6 Increasing dimensions

In this experiment we present the performance of pdNCG, FISTA, PCDM and PSSgb as the number of variables n increases. We generate four instances where the number of variables n takes values 2^{20}, 2^{22}, 2^{24} and 2^{26}, respectively. The singular value decomposition of matrix A is $A = \Sigma G^\intercal$. The singular values in matrix Σ are chosen uniformly at random in the interval $[0, 10]$ and then are shifted by 10^{-1}, which resulted in $\kappa(A^\intercal A) \approx 10^4$. The rotation angle θ of matrix G is set to $2\pi/10$ radians. Moreover, matrices A have $m = 2n$ rows and rank n. The optimal solutions x^* have $s = n/2^7$ nonzero components for each generated instance. For the construction of the optimal solutions x^* we use Procedure OsGen3 with $\gamma = 100$ and $s_1 = s_2 = s/2$, which resulted in $\kappa_{0.1}(x^*) \approx 3$ on average.

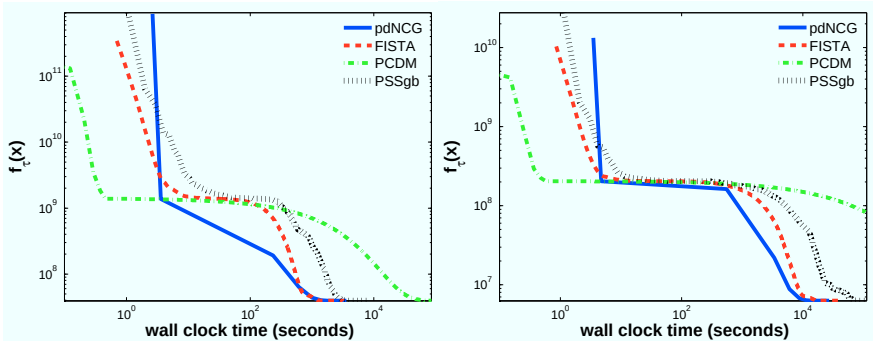
The results of this experiment are presented in Figure 9. Notice that all methods have a linear-like scaling with respect to the size of the problem.

8.7 Increasing density of matrix $A^\intercal A$

In this experiment we demonstrate the performance of pdNCG, FISTA, PCDM and PSSgb as the density of matrix $A^\intercal A$ increases. We generate four instances (A, x^*). For the first experiment we generate matrix $A = \Sigma G^\intercal$, where Σ is the matrix of singular values, the columns of matrices I_m and G are the



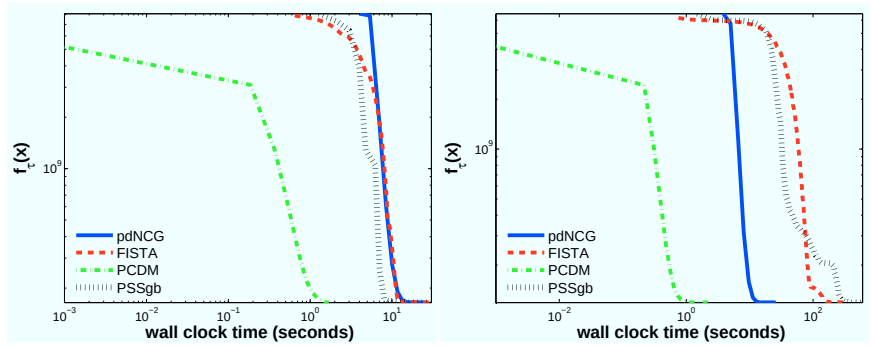
(a) $\kappa(A^\intercal A) = 10^2$ (b) $\kappa(A^\intercal A) = 10^4$



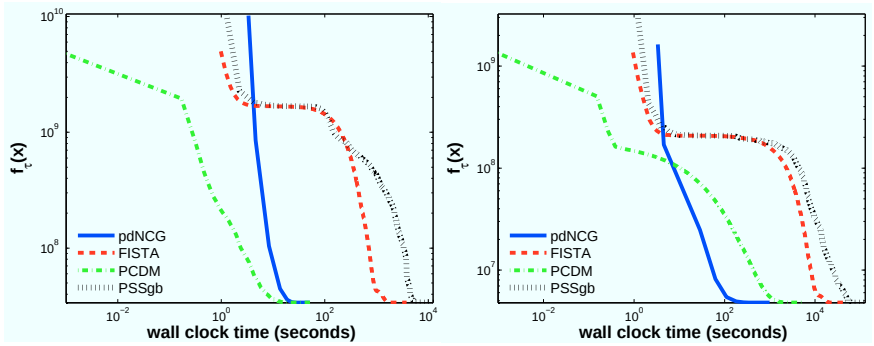
(c) $\kappa(A^\intercal A) = 10^6$ (d) $\kappa(A^\intercal A) = 10^8$

Fig. 7: Performance of pdNCG, FISTA, PCDM and PSSgb on synthetic S-LS problems for increasing condition number of matrix $A^\intercal A$. The optimal solutions have been generated using Procedure OsGen3 with $\gamma = 100$ and $s_1 = s_2 = s/2$. The axes are in log-scale. The rotation angle θ in G was $2\pi/10$. For condition number $\kappa(A^\intercal A) \geq 10^6$ PCDM was terminated after $1,000,000$ iterations, which corresponded to more than 27 hours of wall-clock time.

left and right singular vectors, respectively. For the second experiment we generate matrix $A = \Sigma(G_2 G)^\intercal$, where the columns of matrices I_m and $G_2 G$ are the left and right singular vectors of matrix A, respectively; G_2 has been defined in Subsection 4.2. Finally, for the third and fourth experiments we have $A = \Sigma(G G_2 G)^\intercal$ and $A = \Sigma(G_2 G G_2 G)^\intercal$, respectively. For each experiment the singular values of matrix A are chosen uniformly at random in the interval $[0, 10]$ and then are shifted by 10^{-1}, which resulted in $\kappa(A^\intercal A) \approx 10^4$. The rotation angle θ of matrices G and G_2 is set to $2\pi/10$ radians. Matrices A have $m = 2n$ rows, rank n and $n = 2^{22}$. The optimal solutions x^* have $s = n/2^7$ nonzero components for each experiment. Moreover, Procedure OsGen3 is used with $\gamma = 100$ and $s_1 = s_2 = s/2$ for the construction of x^* for each experiment, which resulted in $\kappa_{0.1}(x^*) \approx 2$ on average.



(a) $\kappa(A^\intercal A) = 10^2$ (b) $\kappa(A^\intercal A) = 10^4$



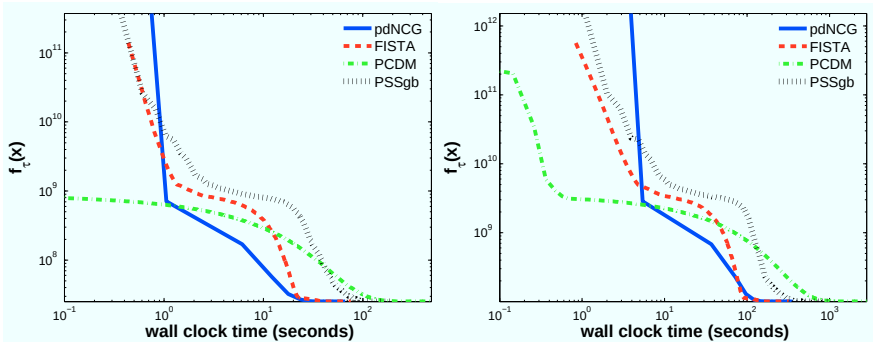
(c) $\kappa(A^\intercal A) = 10^6$ (d) $\kappa(A^\intercal A) = 10^8$

Fig. 8: Performance of pdNCG, FISTA, PCDM and PSSgb on synthetic S-LS problems for increasing condition number of matrix $A^\intercal A$. The optimal solutions have been generated by using Procedure OsGen3 with $\gamma = 100$ and $s_1 = s_2 = s/2$. The rotation angle θ in G was $2\pi/10^3$. The axes are in log-scale.

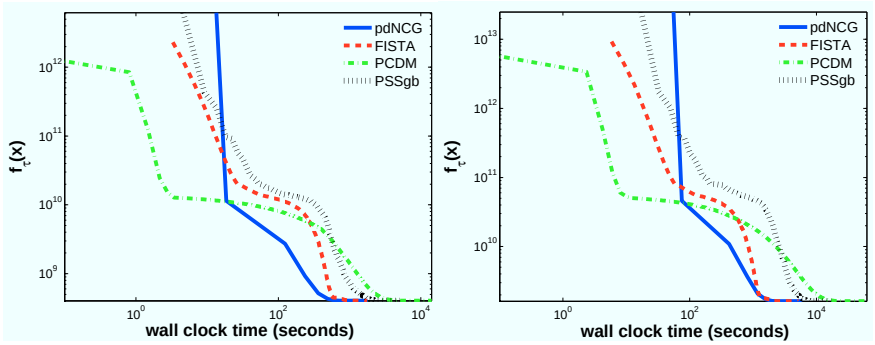
The results of this experiment are presented in Figure 10. Observe, that all methods had a robust performance with respect to the density of matrix A.

8.8 Varying parameter τ

In this experiment we present the performance of pdNCG, FISTA, PCDM and PSSgb as parameter τ varies from 10^{-4} to 10^4 with a step of times 10^2. We generate four instances (A, x^*), where matrix $A = \Sigma G^\intercal$ has $m = 2n$ rows, rank n and $n = 2^{22}$. The singular values of matrices A are chosen uniformly at random in the interval $[0, 10]$ and then are shifted by 10^{-1}, which resulted in $\kappa(A^\intercal A) \approx 10^4$ for each experiment. The rotation angles θ for matrix G in A is set to $2\pi/10$ radians. The optimal solution x^* has $s = n/2^7$ nonzero components for all instances. Moreover, the optimal solutions are generated



(a) $n = 2^{20}$ (b) $n = 2^{22}$



(c) $n = 2^{24}$ (d) $n = 2^{26}$

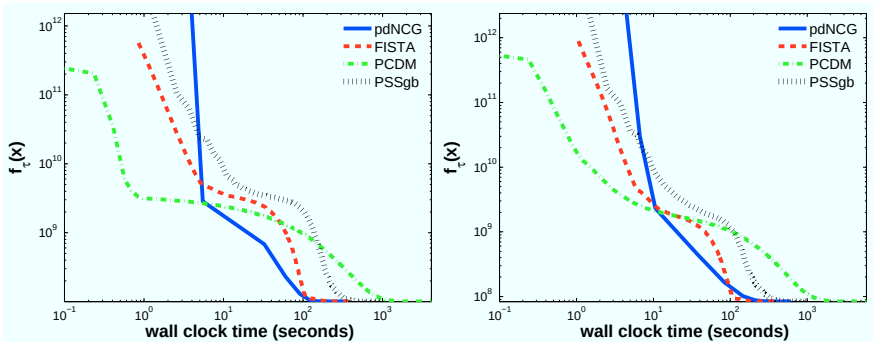
Fig. 9: Performance of pdNCG, FISTA, PCDM and PSSgb on a synthetic S-LS problem for increasing number of variables n. The axes are in log-scale.

using Procedure OsGen3 with $\gamma = 100$, which resulted in $\kappa_{0.1}(x^*) \approx 3$ for all four instances.

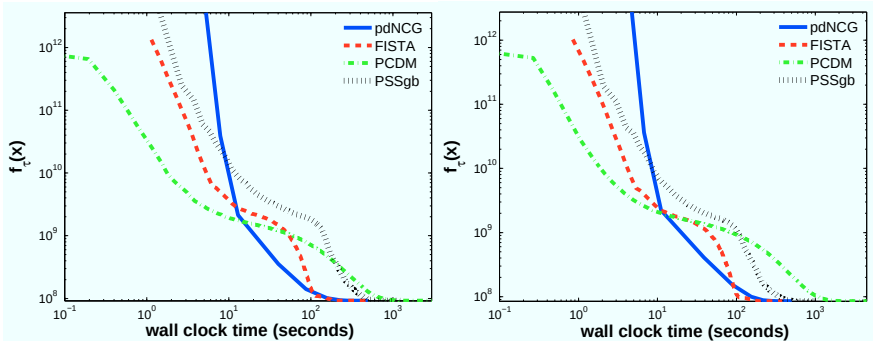
The performance of the methods is presented in Figure 11. Notice in Subfigure 11d that for pdNCG the objective function f_τ is not always decreasing monotonically. A possible explanation is that the backtracking line-search of pdNCG, which guarantees monotonic decrease of the objective function [16], terminates in case that 50 backtracking iterations are exceeded, regardless if certain termination criteria are satisfied.

8.9 Performance of a second-order method on huge scale problems

We now present the performance of pdNCG on synthetic huge scale (up to one trillion variables) S-LS problems as the number of variables and the number of processors increase.



(a) $nnz(A^\intercal A) = 2^{23}$ (b) $nnz(A^\intercal A) \approx 2^{24}$

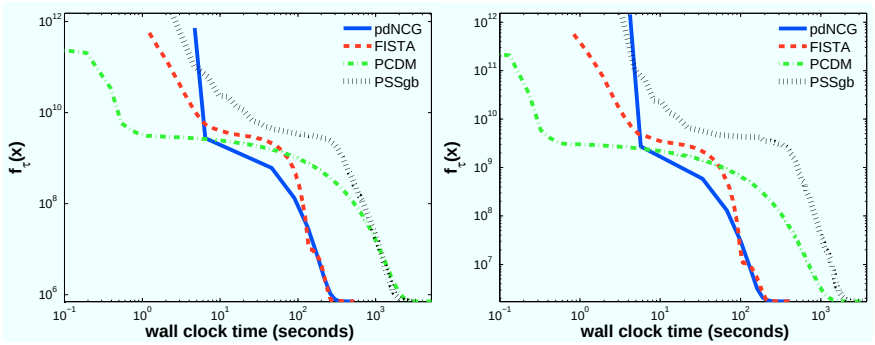


(c) $nnz(A^\intercal A) \approx 2^{24} + 2^{23}$ (d) $nnz(A^\intercal A) \approx 2^{25}$

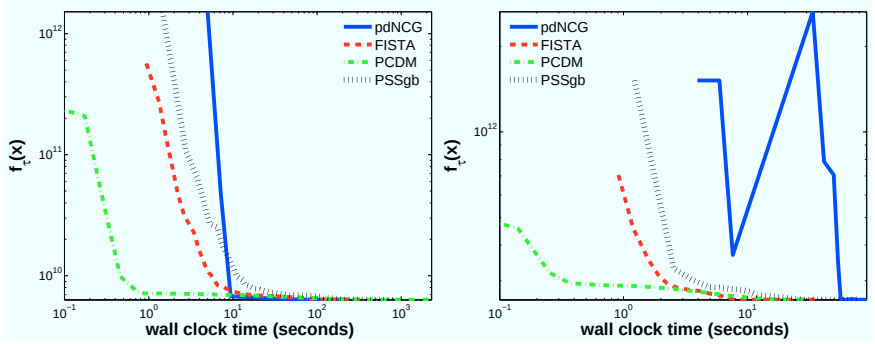
Fig. 10: Performance of pdNCG, FISTA, PCDM and PSSgb on synthetic S-LS problems for increasing number of nonzeros of matrix A. The axes are in log-scale.

We generate six instances (A, x^*), where the number of variables n takes values 2^{30}, 2^{32}, 2^{34}, 2^{36}, 2^{38} and 2^{40}. Matrices $A = \Sigma G^\intercal$ have $m = 2n$ rows and rank n. The singular values σ_i for $i = 1, 2, \cdots, n$ of matrices A are set to 10^{-1} for odd i's and 10^2 for even i's. The rotation angle θ of matrix G is set to $2\pi/3$ radians. The optimal solutions x^* have $s = n/2^{10}$ nonzero components for each experiment. In order to simplify the practical generation of this problem the optimal solutions x^* are set to have $s/2$ components equal to -10^4 and the rest of nonzero components are set equal to 10^{-1}.

Details of the performance of pdNCG are given in Table 1. Observe the nearly linear scaling of pdNCG with respect to the number of variables n and the number of processors. For all experiments in Table 1 pdNCG required 8 Newton steps to converge, 100 PCG iterations per Newton step on average, where every PCG iteration requires two matrix-vector products with matrix A.



(a) $\tau = 10^{-4}$ (b) $\tau = 10^{-2}$



(c) $\tau = 10^{2}$ (d) $\tau = 10^{4}$

Fig. 11: Performance of pdNCG, FISTA, PCDM and PSSgb on synthetic S-LS problems for various values of parameter τ. The axes are in log-scale. Observe in Subfigure 11d that for pdNCG the objective function f_τ is not always decreasing monotonically. This is due to the backtracking line-search of pdNCG, which terminates in case that the maximum number of backtracking iterations is exceeded regardless if certain termination criteria are satisfied.

n	Processors	Memory (terabytes)	Time (seconds)
2^{30}	64	0.192	1,923
2^{32}	256	0.768	1,968
2^{34}	1024	3.072	1,986
2^{36}	4096	12.288	1,970
2^{38}	16384	49.152	1,990
2^{40}	65536	196.608	2,006

Table 1: Performance of pdNCG for synthetic huge scale S-LS problems. All problems have been solved to a relative error of order 10^{-4} of the obtained solution

9 Conclusion

In this paper we developed an instance generator for ℓ_1-regularized sparse least-squares problems. The generator is aimed for the construction of very large-scale instances. Therefore it scales well as the number of variables increases, both in terms of memory requirements and time. Additionally, the generator allows control of the conditioning and the sparsity of the problem. Examples are provided on how to exploit the previous advantages of the proposed generator. We believe that the optimization community needs such a generator to be able to perform fair assessment of new algorithms.

Using the proposed generator we constructed very large-scale sparse instances (up to one trillion variables), which vary from very well-conditioned to moderately ill-conditioned. We examined the performance of several representative first- and second-order optimization methods. The experiments revealed that *regardless of the size of the problem*, the performance of the methods crucially depends on the conditioning of the problem. In particular, the first-order methods PCDM and FISTA are faster for problems with small or moderate condition number, whilst, the second-order method pdNCG is much more efficient for ill-conditioned problems.

Acknowledgements This work has made use of the resources provided by ARCHER (`http://www.archer.ac.uk/`), made available through the Edinburgh Compute and Data Facility (ECDF) (http://www.ecdf.ed.ac.uk/).

The authors are grateful to Dr Kenton D' Mellow for providing guidance and helpful suggestions regarding the use of ARCHER and the solution of large scale problems.

References

1. F. Bach, R. Jenatton, J. Mairal, and G. Obozinski. Optimization with sparsity-inducing penalties. *Journal Foundations and Trends in Machine Learning*, 4(1):1–106, 2012.
2. A. Beck and M. Teboulle. A fast iterative shrinkage-thresholding algorithm for linear inverse problems. *SIAM J. Imaging Sci.*, 2(1):183–202, 2009.
3. S. Becker. CoSaMP and OMP for sparse recovery. `http://www.mathworks.co.uk/matlabcentral/fileexchange/32402-cosamp-and-omp-for-sparse-recovery`, 2012.
4. S. Becker and J. Fadili. A quasi-Newton proximal splitting method. In F. Pereira, C.J.C. Burges, L. Bottou, and K.Q. Weinberger, editors, *Advances in Neural Information Processing Systems 25*, pages 2618–2626. Curran Associates, Inc., 2012.
5. S. R. Becker, J. Bobin, and E. J. Candès. Nesta: A fast and accurate first-order method for sparse recovery. *SIAM J. Imaging Sciences*, 4(1):1–39, 2011.
6. S. R. Becker, E. J. Candés, and M. C. Grant. Templates for convex cone problems with applications to sparse signal recovery. *Mathematical Programming Computation*, 3(3):165–218, 2011. Software available at `http://tfocs.stanford.edu`.
7. M. Bertalmio, G. Sapiro, C. Ballester, and V. Caselles. Image inpainting. *Proceedings of the 27th annual conference on Computer graphics and interactive techniques (SIGGRAPH)*, pages 417–424, 2000.
8. M. Bertero, V. Ruggiero, and L. Zanni. Special issue: Imaging 2013. *Computational Optimization and Applications*, 54:211–213, 2013.
9. S. Boyd, N. Parikh, E. Chu, B. Peleato, and J. Eckstein. Distributed optimization and statistical learning via the alternating direction method of multipliers. *Journal Foundations and Trends in Machine Learning*, 3(1):1–122, 2011.

10. R. H. Byrd, J. Nocedal, and F. Oztoprak. An inexact successive quadratic approximation method for convex l-1 regularized optimization. *Math. Program., Ser. B*, 2015. DOI: 10.1007/s10107-015-0941-y.
11. A. Chambolle, V. Caselles, D. Cremers, M. Novaga, and T. Pock. An introduction to total variation for image analysis. *Radon Series Comp. Appl. Math*, 9:263–340, 2010.
12. A. Chambolle, R. A. DeVore, N. Y. Lee, and B. J. Lucier. Nonlinear wavelet image processing: Variational problems, compression, and noise removal through wavelet shrinkage. *IEEE Trans. Image Process.*, 7(3):319–335, 1998.
13. K.-W. Chang, C.-J. Hsieh, and C.-J. Lin. Coordinate descent method for large-scale ℓ_2-loss linear support vector machines. *Journal of Machine Learning Research*, 9:1369–1398, 2008.
14. D. L. Donoho. Compressed sensing. *IEEE Trans. Inf. Theory*, 52(4):1289–1306, 2006.
15. J. Eckstein. Augmented lagrangian and alternating direction methods for convex optimization: A tutorial and some illustrative computational results. *RUTCOR Research Reports*, 2012.
16. K. Fountoulakis and J. Gondzio. A second-order method for strongly convex ℓ_1-regularization problems. *Mathematical Programming (accepted)*, 2015. DOI: 10.1007/s10107-015-0875-4, Software available at `http://www.maths.ed.ac.uk/ERGO/pdNCG/`.
17. J. Friedman, T. Hastie, and R. Tibshirani. Regularization paths for generalized linear models via coordinate descent. *Journal of Machine Learning Research*, 9:627–650, 2008.
18. T. Goldstein, B. O'Donoghue, and S. Setzer. Fast alternating direction optimization methods. *Technical report, CAM Report 12-35, UCLA*, 2012.
19. J. Gondzio. Matrix-free interior point method. *Computational Optimization and Applications*, 51(2):457–480, 2012.
20. E. T. Hale, W. Yin, and Y. Zhang. Fixed-point continuation method for ℓ_1-minimization: Methodology and convergence. *SIAM J. Optim.*, 19(3):1107–1130, 2008.
21. P. C. Hansen, J. G. Nagy, and D. P. O'Leary. *Deblurring Images: Matrices, Spectra and Filtering*. SIAM, Philadelphia, PA., 2006.
22. P. C. Hansen, V. Pereyra, and G. Scherer. *Least Squares Data Fitting with Applications*. JHU Press, 2012.
23. B. He and X. Yuan. On the $\mathcal{O}(1/t)$ convergence rate of alternating direction method. *SIAM Journal on Numerical Analysis*, 50(2):700–709, 2012.
24. C.-J. Hsieh, K.-W. Chang, C.-J. Lin, S. S. Keerthi, and S. Sundararajan. A dual coordinate descent method for large-scale linear SVM. *Proceedings of the 25th international conference on Machine Learning, ICML 2008*, pages 408–415, 2008.
25. S.-J. Kim, K. Koh, M. Lustig, S. Boyd, and D. Gorinevsky. An interior-point method for large-scale ℓ_1-regularized least squares. *IEEE Journal Of Selected Topics In Signal Processing*, 1(4):606–617, 2007.
26. D. A. Lorenz. Constructing test instances for basis pursuit denoising. *IEEE Trans. Signal Process.*, 61(5):1210–1214, 2013.
27. M. Lustig, D. Donoho, and J. M. Pauly. Sparse MRI: The application of compressed sensing for rapid MR imaging. *Magnetic Resonance in Medicine*, 58(6):1182–1195, 2007.
28. D. Needell and J. A. Tropp. Cosamp: Iterative signal recovery from incomplete and inaccurate samples. *Applied and Computational Harmonic Analysis*, 26(3):301–321, 2009.
29. Y. Nesterov. *Introductory Lecture Notes On Convex Optimization. A Basic Course*. Kluver, Boston, 2004.
30. Y. Nesterov. Smooth minimization of non-smooth functions. *Mathematical Programming*, 103(1):127–152, 2005.
31. Y. Nesterov. Smooth minimization of non-smooth functions. *Math. Program.*, 103(1):127–152, 2005.
32. Yu. Nesterov. Gradient methods for minimizing composite functions. *Mathematical Programming*, 140(1):125–161, 2013.
33. N. Parikh and S. Boyd. Proximal algorithms. *Journal Foundations and Trends in Optimization*, 1(3):123–231, 2013.
34. P. Richtárik and M. Takáč. Iteration complexity of randomized block-coordinate descent methods for minimizing a composite function. *Math. Program. Ser. A*, 144(1):1–38, 2014.

35. P. Richtárik and M. Takáč. Parallel coordinate descent methods for big data optimization. *Math. Program. Ser. A*, pages 1–52, 2015. DOI: 10.1007/s10107-015-0901-6.
36. K. Scheinberg and X. Tang. Practical inexact proximal quasi-Newton method with global complexity analysis. Technical report, March 2014. arXiv:1311.6547 [cs.LG].
37. M. Schmidt. Graphical model structure learning with l1-regularization. *PhD thesis, University British Columbia*, 2010.
38. S. Shalev-Shwartz and A. Tewari. Stochastic methods for ℓ_1-regularized loss minimization. *Journal of Machine Learning Research*, 12(4):1865–1892, 2011.
39. P. Tseng. Convergence of a block coordinate descent method for nondifferentiable minimization. *Journal of Optimization Theory and Applications*, 109(3):475–494, 2001.
40. P. Tseng. Efficiency of coordinate descent methods on huge-scale optimization problems. *SIAM J. Optim.*, 22:341–362, 2012.
41. P. Tseng and S. Yun. A coordinate gradient descent method for nonsmooth separable minimization. *Math. Program., Ser. B*, 117:387–423, 2009.
42. S. Vattikuti, J. J. Lee, C. C. Chang, S. D. Hsu, and C. C. Chow. Applying compressed sensing to genome-wide association studies. *GigaScience*, 3(10):1–17, 2014.
43. Y. Wang, J. Yang, W. Yin, and Y. Zhang. A new alternating minimization algorithm for total variation image reconstruction. *SIAM Journal on Imaging Sciences*, 1(3):248–272, 2008.
44. S. J. Wright. Accelerated block-coordinate relaxation for regularized optimization. *SIAM Journal on Optimization*, 22(1):159–186, 2012.
45. T. T. Wu and K. Lange. Coordinate descent algorithms for lasso penalized regression. *The Annals of Applied Statistics*, 2(1):224–244, 2008.
46. G.-X. Yuan, K.-W. Chang, C.-J. Hsieh, and C.-J. Lin. A comparison of optimization methods and software for large-scale ℓ_1-regularized linear classification. *Journal of Machine Learning Research*, 11:3183–3234, 2010.