CompAMa
www.compama.co.usb.ve

Bull. Comput. Appl. Math., Vol.5, No.1, 2017
ISSN 2244-8659

A convex optimization approach for solving large scale linear systems

Debora Cores [*], *Johanna Figueroa* [†]

CompAMa Vol.5, No.1, pp.51-74, 2017 - Accepted November 11, 2016

Abstract

The well-known Conjugate Gradient (CG) method minimizes a strictly convex quadratic function for solving large-scale linear system of equations when the coefficient matrix is symmetric and positive definite. In this work we present and analyze a non-quadratic convex function for solving any large-scale linear system of equations regardless of the characteristics of the coefficient matrix. For finding the global minimizers, of this new convex function, any low-cost iterative optimization technique could be applied. In particular, we propose to use the low-cost globally convergent Spectral Projected Gradient (SPG) method, which allow us to extend this optimization approach for solving consistent square and rectangular linear system, as well as linear feasibility problem, with and without convex constraints and with and without preconditioning strategies. Our numerical results indicate that the new scheme outperforms state-of-the-art iterative techniques for solving linear systems when the symmetric part of the coefficient matrix is indefinite, and also for solving linear feasibility problems.

Keywords: Nonlinear convex optimization, spectral gradient method, large-scale linear systems.

[*]Departamento de Cómputo Científico y Estadística, Universidad Simón Bolívar (USB), Caracas 1080-A, Venezuela (cores@usb.ve).

[†]Departamento de Matemática de la Facultad de Ciencias y Tecnología, Universidad de Carabobo (UC), Valencia 2005, Venezuela (jfigueroa1@uc.edu.ve).

1 Introduction

The strictly convex quadratic function

$$q(x) = \frac{1}{2}x^T Ax - b^T x, \tag{1}$$

where A is a symmetric and positive definite (SPD) matrix, is the classical and suitable choice to apply unconstrained minimization techniques for solving the large-scale linear system $Ax = b$. In that case, the most effective iterative method for the unconstrained minimization is the Conjugate Gradient (CG) method; see, e.g., [1–4]. In the SPD case, (1) has also been the suitable choice to develop and analyze gradient or residual iterative schemes for solving $Ax = b$; see, e.g., [5].

In many real applications as signal processing, structural analysis, data fitting, and linear programming, among others, the coefficient matrix A could be rectangular and could have no desirable characteristics as symmetry, positive definiteness, positive definiteness of the symmetric part, or diagonal dominance. Hence, in the presence of no desirable characteristics, the CG method and gradient related methods cannot be applied directly to the minimization of (1). Nevertheless, they can always be applied to the minimization of the convex quadratic $\frac{1}{2}x^T A^T Ax - (A^T b)^T x$, for solving the normal equations: $A^T Ax = A^T b$. Now, this so-called least-squares approach has some advantages but also some well-known disadvantages, including the possible drastic increase of the condition number of the coefficient matrix $A^T A$; see, e.g., [4, 6, 7].

In order to extend the unconstrained minimization approach, in this work we propose a new convex function for solving any large-scale linear system of equations regardless of the characteristics of the coefficient matrix. For this new non-quadratic convex function, any low-cost optimization technique could be applied. In particular, we propose to use the Spectral Projected Gradient (SPG) method [8–10], since it is an effective low-cost optimization scheme that has been successfully applied in many large-scale real applications; see, e.g., [11].

The rest of this document is organized as follows. In section 2 we present our proposal, we discuss the linear problems to be considered, and we briefly describe the SPG method. In section 3 we present some numerical experiments in which the performance of the proposed scheme is studied and compared with the well-established state-of-the-art methods CG, GMRES,

BICGSTAB, ORM, RA, and Kaczmarz, for different scenarios. Finally, in section 4, we present some concluding remarks.

2 Optimization approach

For solving the linear system $Ax = b$, where $A \in \mathbb{R}^{m\times n}$ and $b \in \mathbb{R}^m$, consider the function:

$$f(x) = \sum_{i=1}^{m} f_i(x) = \sum_{i=1}^{m} (e^{-r_i(x)} + e^{r_i(x)}), \tag{2}$$

where $f_i(x) = e^{-r_i(x)} + e^{r_i(x)}$ and $r_i(x)$ is the ith component of the residual vector $r(x) = b - Ax$.

The most relevant properties of the function $f(x)$ are established in Propositions 1, 2 and 3

Proposition 1. *The function f defined in (2) is a convex function.*

Proof. Let $g : \mathbb{R} \to \mathbb{R}$ be given by $g(z) = e^z + e^{-z}$. Since $g''(z) = e^z + e^{-z} > 0$ for all $z \in \mathbb{R}$, then g is convex. Hence, $f_i(x)$ is convex for all $1 \leq i \leq m$. Consequently, $f(x) = \sum_{i=1}^{m} f_i(x)$ is the sum of convex functions and so it is also a convex function. □

Notice that since $g'(z) = e^z - e^{-z}$, then the unique global minimizer of $g(z) = e^z + e^{-z}$ is reached when $e^z = e^{-z}$, i.e., when $z = 0$. Notice also that $g(0) = 2$.

Proposition 2. *If $Ax^* = b$ for some vector $x^* \in \mathbb{R}^n$, then $f(x^*) = 2m$.*

Proof. Since $Ax^* = b$ then the residual vector satisfies $r_i(x^*) = 0$, for all i, $1 \leq i \leq n$, and $f_j(x^*) = e^{-r_j(x^*)} + e^{r_j(x^*)} = 2$, for all j, $1 \leq j \leq m$. Adding the m values we obtain $f(x^*) = 2m$. □

Notice that Proposition 2 reveals an effective criterion for determining if a given vector x^* solves the system $Ax = b$.

Proposition 3. *If the system $Ax = b$ is consistent then f attains its global minimum value $2m$. Moreover, any global minimizer of f solves the linear system $Ax = b$.*

Proof. Let $x \in \mathbb{R}^n$ and let $x^* \in \mathbb{R}^n$ such that $Ax^* = b$. Since $f_i(x) = e^{-r_i(x)} + e^{r_i(x)}$ attains a minimum value when $r_i(x) = 0$ then $f_i(x) \geq 2$. Hence by Proposition 2

$$f(x) = \sum_{i=1}^{m} f_i(x) \geq 2m = f(x^*). \tag{3}$$

Let us now consider $\widetilde{x}$ for which $A\widetilde{x} - b \neq 0$, that is, there exists k, $1 \leq k \leq n$, such that $A_k\widetilde{x} - b_k \neq 0$, where A_k represents the k-th row of A. Therefore, $f_k(\widetilde{x}) = e^{(A_k\widetilde{x}-b_k)} + e^{-(A_k\widetilde{x}-b_k)} > 2$. Furthermore, for all $i = 1, ..., m$, with $i \neq k$, $f_i(\widetilde{x}) \geq 2$. Consequently, $f(\widetilde{x}) > 2m$. Hence, $\widetilde{x}$ is not a minimizer of f. □

Observe that the objective function (2) can be written in a more general way as

$$f(x) = \sum_{i=1}^{m} \varphi(r_i(x)), \tag{4}$$

where $\varphi : \mathbb{R} \to \mathbb{R}^+$, $\varphi \in \mathcal{C}^2(\mathbb{R})$, φ even, strictly convex and coercive and whose global minimum is reached at 0. Moreover, propositions 1, 2 and 3 are valid for any function φ satisfying the previous conditions. In particular, considering $\varphi(t) = t^2$ corresponds to the linear least square problem.

2.1 Problems to be solved

According to Propositions 1, 2 and 3, solving a consistent linear system $Ax = b$ is equivalent to solving the following unconstrained convex minimization problem:

$$\text{Find } x^*, \text{ such that } x^* = arg\left(\min_{x \in \mathbb{R}^n} f(x)\right). \tag{5}$$

An important aspect of this optimization approach is that it can be used for any consistent square or rectangular linear systems, regardless of the characteristics of the coefficient matrix.

Since the function f involves exponential terms, then a suitable scaling parameter $\delta > 0$ can always be found to solve

$$\frac{1}{\delta}Ax = \frac{1}{\delta}b, \tag{6}$$

instead of $Ax = b$, to avoid the appearance of big numbers that could cause loss of accuracy or even overflow. Clearly, x solves $Ax = b$ if and only if x solves (6). Another option to avoid loss of accuracy when solving (5) is to use preconditioning strategies for the linear system $Ax = b$. The way of choosing $\delta > 0$ and the use of preconditioning strategies will be fully described in Section 3 of numerical experiments.

In some real applications, lower and upper bounds must be imposed to the solution vector. In this case, we are interested in solving the following problem:

$$\begin{cases} \text{Find } x \text{ such that } Ax = b \\ \text{subject to:} \quad x \in \Omega, \end{cases} \tag{7}$$

where Ω is a convex set. In particular, we are interested in the problem:

$$\begin{cases} \text{Find } x \text{ such that } Ax = b \\ \text{subject to:} \\ \qquad l_i \leq x_i \leq u_i \text{ if } i \in \mathfrak{D} \subseteq \{1, 2, \cdots, n\}. \end{cases} \tag{8}$$

In this case the optimization approach (5) allows us to add convex constraints, that is, to find the solution of $Ax = b$ within a given convex set, obtaining the following convex optimization problem:

$$\begin{cases} \text{Find } x^* = arg\,(\min_{x \in \mathbb{R}^n} f(x)) \\ \text{Subject to:} \\ \qquad l_i \leq x_i \leq u_i \text{ if } i \in \mathfrak{D}. \end{cases} \tag{9}$$

For some other applications, as image recovery or inverse problems, the solution of a linear feasibility problem is required:

$$\text{Find } x \text{ such that } Ax \leq b, \tag{10}$$

where $A \in \mathbb{R}^{m \times n}$, $b \in \mathbb{R}^m$ and $x \in \mathbb{R}^n$.

Our proposal can be used for solving feasibility problems transforming problem (10) into an $m \times (m + n)$ constrained linear systems, as follows:

$$\begin{cases} \text{Find } x \text{ such that } \widetilde{A}\tilde{x} = b \\ \text{Subject to:} \\ \qquad \tilde{x}_i \geq 0 \quad \text{if} \quad n + 1 \leq i \leq n + m \end{cases} \tag{11}$$

where $\widetilde{A} = \begin{pmatrix} A & I_m \end{pmatrix}_{m \times (m+n)}$, I_m is the $m \times m$ identity matrix, $\tilde{x}_i = x_i$, for $1 \leq i \leq n$, and $\tilde{x}_i$, for $n + 1 \leq i \leq n + m$, are auxiliary variables or slack variables. Moreover, if some components must be bounded, these restrictions can be added to problem (11) and it can be reduced to problem (9).

2.2 The SPG machinery

The different problems described in Section 2.1, for which our optimization approach can be applied, can be solved by any iterative low-cost optimization method that can handle convex constraints. In particular, we consider the Spectral Projected Gradient (SPG) method [8–10], which is nowadays a well-established nonmonotone numerical scheme for solving large-scale convex constrained optimization problems when the projection onto the feasible set can be performed efficiently [11]. The SPG method has been extended to some other constrained optimization settings; see, e.g., [12, 13]. The attractiveness of the SPG method is mainly based on its simplicity. Moreover, it is globally convergent, i.e., the sequence that it generates converges to stationary points from any initial guess. For more details on the convergence properties of the SPG method see [8],[9] and [10].

We now discuss the most important features of the SPG method for solving nonlinear optimization problems of the form:

$$\min_{x\in\Omega} f(x), \tag{12}$$

where Ω is a closed convex set in $\mathbb{R}^n$ and $f : \mathbb{R}^n \longrightarrow \mathbb{R}$ is a function with continuous partial derivatives in an open set that contains Ω. Starting from a given initial $x_0 \in \mathbb{R}^n$ the iterations are given by

$$x_{k+1} = x_k + \alpha_k d_k, \tag{13}$$

where $d_k = P_\Omega(x_k - \lambda_k g_k) - x_k$, $g_k = \bigtriangledown f(x_k)$, P_Ω denotes the projection onto Ω and λ_k is the spectral choice of step length, given by:

$$\lambda_k = \frac{s_{k-1}^T s_{k-1}}{s_{k-1}^T y_{k-1}}, \tag{14}$$

where $s_{k-1} = x_k - x_{k-1}$ and $y_{k-1} = g_k - g_{k-1}$.

The parameter $\alpha_k > 0$ in (13) is found by a nonmonotone line search to accomplish the following condition:

$$f(x_{k+1}) \leq \max_{0\leq j\leq \min\{k,M-1\}} f(x_{k-j}) + \gamma\alpha_k g_k^T d_k, \tag{15}$$

where M is a nonnegative integer and γ is a small positive number. In this work, we set $M = 10$ and $\gamma = 10^{-4}$. For more details concerning practical issues see [11]. Notice that the SPG method can also be applied to the unconstrained minimization problem (5) by setting $\Omega = \mathbb{R}^n$.

3 Numerical experiments

From now on, the application of the SPG method on the convex minimization problems described in section 2.1, will be denoted by OPAPLS (Optimization Approach for Linear Systems). In order to illustrate the advantages of the proposed strategy, we compare OPAPLS with the following methods: GMRES, BICGSTAB, RA, ORM, CG, and Kaczmarz method. The methods GMRES, BICGSTAB, CG and Kaczmarz are well-known and they can be found for instances in [6], [14] and [4], respectively. The strategies ORM and RA are iterative schemes for which the search direction is the residual vector; see [15] and [16].

The CG and RA methods correspond to the use of $\varphi(t) = t^2$ in the general formulation (4). So, comparisons between different functions φ are presented in our numerical experiments.

The Optimal Richardson Method (ORM), introduced in [15], is a variation of the classical Richardson's method for solving $Ax = b$, which uses the following iterative scheme:

$$x_{k+1} = x_k + \lambda_k r_k, \tag{16}$$

where the step length λ_k is chosen as follows:

$$\lambda_k = \frac{r_k^T A r_k}{(Ar_k)^T A r_k}. \tag{17}$$

For more details on the ORM scheme see [5].

The residual RA method solves $Ax = b$ by solving the minimization problem:

$$\text{Find } x^* = arg \min_{x \in \mathbb{R}^n} \|r(x)\|^2. \tag{18}$$

This scheme generates the iterates using plus or minus the residual vector at x_k as search direction, as follows:

$$x_{k+1} = x_k + sgn(\beta_k)\frac{1}{\beta_{k-1}} r_k, \tag{19}$$

where, $sgn(z)$ represents the sign of the real variable z, and

$$\beta_k = \frac{r_k^T A r_k}{r_k^T r_k}.$$

In the RA scheme, a globalization strategy is incorporated to guarantee convergence from any initial guess and any positive initial step length. For more details on the RA scheme see [16].

In all our experiments, the calculations were done on a i7 4710MQ at 2.50GHz with Matlab R2013b. For BICGSTAB and GMRES (with the standard restart parameters 20 and 40) we used the available commands in Matlab. For RA and ORM we used the algorithms presented in [16] and [15], respectively. For the CG method we used a Matlab implementation based on the algorithm described in [6, p. 289]. The Kaczmarz method was implemented according to Algorithm 1 (below), as described in [14, pp. 41], with the relaxation parameter $\omega = 1$, and following the natural ordering for the projections. In Algorithm 1, A_i represents the i-th row of A, and the stopping criterion is evaluated after every sweep of m projections.

Algorithm 1: Kaczmarz Scheme

Require $A \in \mathbb{R}^{m\times n}$, $b \in \mathbb{R}^m$, $x_0 \in \mathbb{R}^n$;
Ensure x_0, *cycles*;
for $k = 1, 2, \cdots,$ until convergence **do**
  for $i = 1, 2, \cdots, m$ **do**
    $x_0 = x_0 + \dfrac{b_i - A_i^T x_0}{A_i^T A_i} A_i$;
  end for
end for
$cycles = k$;

In all the forthcoming tables, we report the results using the following notation: the number of iterations (*iter*), the required cpu time until convergence (*tcpu*) in seconds, the number of flops ($flops$) which represents the required computational work, and the relative norm of the residual $\left(\frac{\|r\|}{\|b\|}\right)$. For OPAPLS, $fcnt$ represents the number of evaluations of the objective function and *gcnt* the number of evaluations of the gradient of the objective function. For GMRES, we report the number of iterations and not the number of cycles. Let us recall that a cycle is made of the iterations in between two restarts.

The number of flops for each method, once k iterations or k cycles for Kaczmarz method have been performed, is obtained as follow: For OPAPLS $(gcnt + fcnt) \cdot n \cdot m + 2 \cdot n \cdot k$ flops are used to obtain the solution. The

Kaczmarz method uses m inner products of length n to obtain the square of the norm of each row vector. Furthermore, it uses one inner product of length n for each row A_i. Therefore, in a cycle it uses $n \cdot m$ flops. So, $n \cdot m \cdot k + m \cdot n$ flops are required by Kaczmarz method. GMRES(l) requires per iteration one matrix-vector multiplication and $2(k \mod l)$ inner products of length n, that is, $n^2 \cdot k + 2(k \mod l) \cdot n$ flops. BICGSTAB requires two matrix-vector multiplication and four inner products of length n, that is, $(2n^2 + 4 \cdot n) \cdot k$ flops. The ORM and RA methods require per iteration one matrix-vector multiplication and two inner products of length n, which implies the execution of $(n^2 + 2 \cdot n) \cdot k$ flops. For the case of the conjugate gradient method, one matrix-vector, two inner products, and two vector summations per iteration are required. Then, the CG method requires $(n^2 + 4 \cdot n) \cdot k$ flops.

In all methods and examples presented in this work, the initial guess is the null vector. The stopping criterion for solving unconstrained problems is

$$\frac{\|r\|}{\|b\|} < 10^{-10}.$$

For OPAPLS other simple stopping criteria are available, among them, $|f(x_k) - 2m| < \varepsilon$ and $\|\nabla f(x_k)\| < \varepsilon$, for unconstrained problems and $\|P_\Omega (x_k - \lambda_k \nabla f(x_k)) - x_k\| < \varepsilon$ for solving problem (9). We set 20.000 as the maximum number of cycles for Kaczmarz method, or iterations for all the other methods. The symbol $**$ in our tables indicates that the corresponding method does not converge, with the desired tolerance before reaching the maximum number of iterations.

In our first experiment, we consider orthogonal matrices of dimension 10000×10000, from the Matlab gallery, which are shown in Table 1. The right hand side vector is given as $b = (1, 1, \cdots, 1)^T$. The number of iterations required by each method are shown in Table 2.

Tab. 1: Description of orthogonal test matrices, from the Matlab gallery.

Matrix	Description	Matlab Commands
Q1	$a_{i,j} = \sqrt{2/(m+1)} \sin(ij\pi/(m+1))$,	A=gallery('orthog',m,1)
Q2	$a_{i,j} = 2/\sqrt{2m+1} \sin(2ij\pi/(2m+1))$	A=gallery('orthog',m,2)
Q3	A permutation of a lower Hessenberg matrix, whose $a_{1,j} = \frac{1}{\sqrt{m}}$	A=gallery('orthog',m,4)
Q4	Householder matrix, $\sum_{i=1}^{m} a_{i,j} = 0$, $2 \le j \le m$ and $\sum_{i=1}^{m} a_{i,1} = \sqrt{m}$	A=gallery('orthog',m,7)

We can observe, in Table 2, that for all methods few iterations are required for convergence. Indeed, these matrices are well conditioned. In particular, notice that our proposal, OPAPLS, is competitive with well-established

Tab. 2: Required iterations for GMRES(20), GMRES(40), BICGSTAB, RA and OPAPLS for solving linear systems with the orthogonal matrices described in Table 1, without preconditioning.

	GMRES(20)	GMRES(40)	BICGSTAB	RA	OPAPLS
Matrix	*Iter*	*Iter*	*Iter*	*Iter*	*Iter*
Q1	2	2	2	14	6
Q2	2	2	2	14	6
Q3	19	19	18	**	6
Q4	2	2	3	5	6

methods for solving orthogonal linear systems. For the third matrix, $Q3$, we observe an increase in the number of iterations when using GMRES(20) and GMRES(40). For this example the symmetric part of the coefficient matrix is indefinite, and as a consequence RA does not converge; see [16].

For all the linear problems, presented in the remainder of this work, a scaling factor is applied. The matrix A and the vector b are multiplied by $\frac{1}{\delta}$, where $\delta = \max\left(\max_{i,j}|a_{i,j}|, \max_i|b_i|\right)$.

For example, for linear systems, we solve (6) instead of $Ax = b$. Since the initial guess $x_0 = 0$, then the initial residual is the vector b, and as a consequence the scaling factor guarantees that $|r_i(x_0)| \leq 1$ for all i. Moreover, using the global convergence of the SPG method and Proposition 3, it follows that the elements of the residual vector, during the convergence process, will be bounded above in absolute value by a small positive number. This fact could be verified numerically with the parameter $r_{max} = max_{1\leq k\leq iter}\|r(x_k)\|_\infty$. Consequently, the described scaling factor $\delta > 0$ avoids the appearance of big numbers that could cause loss of accuracy when the function $f(x)$ in (2) is evaluated during the iterations. Finally, in order to make a fair comparison between all strategies, we apply the same scaling factor to all the considered methods.

For our second experiment we compare the performance of OPAPLS with all the other methods to solve linear systems in which the coefficient matrix has a definite symmetric part. We consider ten matrices, taken from the Matlab gallery, described in Table 3. The right hand side vector is given by $b = (1, 1, \cdots, 1)^T$. The obtained results are shown in Table 4.

Table 4 shows that the proposed strategy OPAPLS is competitive with the other strategies for several matrices. However, OPAPLS does not converge for the matrices identified as, $PSD3$, $PSD4$, $PSD8$ and $PSD10$. Notice that the coefficient matrices in all these cases are ill-conditioned. Note

Tab. 3: Description of the matrices with definite symmetric part, from the Matlab gallery.

Matrix	Description	Matlab Commands
PSD1	Sparse adjacency matrix from NASA airfoil	Matlab demo: airfoil
PSD2	Singular toeplitz lower Hessenberg	$PSD2 = gallery('chow', m, 1, 1)$
PSD3	Circulant matrix	$PSD3 = gallery('circul', v)$ $v_i = \begin{cases} 10^{-6}, & if \quad i = 1 \\ 1, & if \quad i = \frac{m}{2} \\ -1, & if \quad i = m \\ 0, & \text{otherwise} \end{cases}$
PSD4	Diagonally dominant, ill conditioned, tridiagonal matrix	$PSD4 = gallery('dorr', m, 1)$
PSD5	Perturbed Jordan block	$PSD5 = gallery('forsythe', m, -1, 2)$
PSD6	Matrix whose eigenvalues lie on the vertical line	$PSD6 = gallery('hanowa', m, m)$
PSD7	Jordan block	$PSD7 = gallery('jordbloc', m, 2)$
PSD8	Tridiagonal matrix with real sensitive eigenvalues	$PSD8 = -gallery('lesp', m)$
PSD9	Pentadiagonal Toeplitz matrix	$PSD9 = gallery('toeppen', m, 1, 10, m, -10, -1)$
PSD10	Upper triangular matrix discussed by Wilkinson and others	$PSD10 = gallery('triw', m, -0.5, 2)$

Tab. 4: Required iterations for GMRES(20), GMRES(40), BICGSTAB, RA, ORM and OPAPLS matrices with definite symmetric part, described in Table 3, without preconditioning.

		GMRES(20)	GMRES(40)	BICGSTAB	RA	ORM	OPAPLS	
Matrix	m	*Iter*	*Iter*	*Iter*	*Iter*	*Iter*	*Iter*	r_{max}
PSD1	4253	61	60	**	64	75	240	5.9110394
PSD2	1000	389	229	216	532	1044	9289	3.1843281
PSD3	5000	**	**	1	2	**	**	9.2318003
PSD4	500	**	2047	283	3540	**	**	1.0000000
PSD5	5000	28	28	**	29	28	38	1.0000000
PSD6	5000	17	17	11	38	27	13	0.8331333
PSD7	5000	27	27	**	28	27	33	1.0000000
PSD8	5000	4090	2560	**	11112	**	**	6.7975901
PSD9	5000	4	4	2	5	4	5	9.9760922
PSD10	5000	3020	3067	**	3408	3151	**	9.6055586

also that, for these same examples, some other methods do not converge as well. Therefore, in order to improve the condition number of the coefficient matrices a preconditioning strategy is applied.

In tables 5 y 6 we report the results for the matrices of Table 3, when the following two preconditioning strategies are used:

(a) Incomplete LU factorization with drop tolerance (ILU): the preconditioning matrix is obtained, in Matlab, with the command [L,U]=ilu(A,0.3).

(b) The SSOR preconditioning strategy: the preconditioning matrix is given by $(D - \omega E)D^{-1}(D - \omega F)$, where $-E$ is the strict lower triangular part of A, $-F$ is the strict upper triangular part of A, D is the diagonal part of A and we consider $\omega = 1$.

We observe from tables 5 and 6 that the ILU preconditioning is more effective than the SSOR preconditioning strategy for the proposed methodology. Notice that, in general, all the methods reached convergence when ILU

Tab. 5: Required iterations for GMRES(20), GMRES(40), BICGSTAB, RA, ORM and OPAPLS matrices with definite symmetric part, described in Table 3, with preconditioning (a).

	GMRES(20)	GMRES(40)	BICGSTAB	RA	ORM	OPAPLS	
Matrix	*Iter*	*Iter*	*Iter*	*Iter*	*Iter*	*Iter*	r_{max}
PSD1	53	53	29	57	56	5	1.00000000
PSD2	396	229	523	216	1038	5	0.61803399
PSD3	3	3	2	5	6	14	0.20000000
PSD4	1	1	1	2	1	5	0.86963715
PSD5	2	2	2	4	3	10	0.75000000
PSD6	13	13	8	23	18	6	0.99975858
PSD7	1	1	1	2	1	5	0.50000000
PSD8	7	7	4	13	11	5	0.70186762
PSD9	4	4	2	5	4	5	0.99979956
PSD10	1	1	1	2	2	6	1.00000000

Tab. 6: Required iterations for GMRES(20), GMRES(40), BICGSTAB, RA, ORM and OPAPLS matrices with definite symmetric part, described in Table 3, with preconditioning (b).

	GMRES(20)	GMRES(40)	BICGSTAB	RA	ORM	OPAPLS	
Matrix	*Iter*	*Iter*	*Iter*	*Iter*	*Iter*	*Iter*	r_{max}
PSD1	1	1	1	2	1	5	1.00000000
PSD2	13	13	7	16	14	18	0.63952238
PSD3	**	**	**	**	**	**	1.00000000
PSD4	4773	2452	180	1939	1850	**	6.75233780
PSD5	2	2	2	4	3	10	0.75000000
PSD6	8	8	5	12	10	18	0.86230000
PSD7	1	1	1	2	1	5	0.50000000
PSD8	5	5	3	7	6	8	0.69433608
PSD9	2	2	1	3	2	5	0.99979553
PSD10	1	1	1	2	1	6	1.00000000

preconditioning strategy is used. However, others preconditioning strategies could be used and adapted to this new scheme to improve the performance of the method.

In our third experiment we work with systems for which the coefficient matrix has an indefinite symmetric part. These systems are presented in two groups for which the coefficient matrices are generated in two different ways. In all cases, the solution for each system is the vector $(1, 1, 1, ..., 1)^T$.

The first group, described in Table 7, was generated using the Matlab

gallery. In the second group all the matrices are tridiagonal $m \times m$, with the structure shown in (20).

$$\begin{pmatrix} a_{11} & -1 & & & \\ 1 & a_{22} & -1 & & \\ & \ddots & \ddots & \ddots & \\ & & 1 & a_{m-1m-1} & -1 \\ & & & 1 & a_{mm} \end{pmatrix} \tag{20}$$

where $a_{ii} = a + \left(\frac{i-1}{m-1}\right)(a_{max} - a)$, $\forall\ i = 1, 2, ..., m$, and a and a_{max} are constants such that the symmetric part of the matrix is indefinite. These values and the values considered for m are shown in Table 8.

Tab. 7: Description of test matrices with an indefinite symmetric part, from the Matlab gallery.

Matrix	Description	Matlab Commands
PSI1	Tridiagonal and nonsymmetric	A=gallery('clement',m, 0)
PSI2	$a_{i,j} = 0$ or $a_{i,j} = 1$, $1 \leq i, j \leq m$	A=gallery('dramadah', m)
PSI3	$a_{i,i} = 1$, $a_{i,i-1} = -1$, $a_{i,i+s} = 1$, $s = 1, ..., k$, $i = 1, ..., m$	A=gallery('grcar', m)
PSI4	$a_{1j} = 1$, $a_{i,i-1} = 1$, $1 \leq i \leq m$	A=gallery('leslie', m)
PSI5	Random, orthogonal, and upper Hessenberg	A=gallery('randhess', m)
PSI6	$a_{i,i-2} = 1$, $a_{i,i-1} = 10$, $a_{i,i} = 4$, $a_{i,i+1} = 3$, $a_{i,i+2} = 1$, $1 \leq i, j \leq n$	$A = gallery('toeppen', m, 1, 10, 4, 3, 1)$
PSI7	Analogue to PSI6	$A = gallery('toeppen', 5000, -2, 10, 0, 4, 3)$
PSI8	$a_{i,i-1} = -5$, $a_{i,i} = 1.8$, $a_{i,i+1} = 2$, $1 \leq i, j \leq n$	A=gallery('tridiag', m, -5,1.8,2)
PSI9	Analogue to PSI6	$A = gallery('toeppen', m, 10, 3, -10, -1, 1)$
PSI10	Analogue to PSI1	A=gallery('clement', m, 0)

For the set of systems described in Table 7 we only report the results obtained using Kaczmarz method and OPAPLS. None of the other methods converge for this matrices, except the strategies GMRES(20) and GMRES(40) for the matrix $PSI3$. The convergence of the OPAPLS strategy is attained for all matrices and the results are shown in Table 9. Kaczmarz method fails for the matrix $PSI5$ and, in general, it uses more cpu time than OPAPLS.

The numerical results obtained using OPAPLS and Kaczmarz method for the matrices described in Table 8 are shown in Table 10. For these examples none of the methods GMRES(20), GMRES(40), BICGSTAB, RA and ORM converge with the desired tolerance in the number of iterations allowed. In all these examples the symmetric part of the matrix associated with each system is indefinite. So, we only report the results of Kaczmarz method and OPAPLS.

Tab. 8: Dimension and parameters a and $amax$ for tridiagonal matrices with an indefinite symmetric part.

Matrix	m	a	a_{max}
PSID1	5000	-3	10
PSID2	5000	-20	200
PSID3	5000	-20	980
PSID4	10000	-3	4
PSID5	10000	-3	100
PSID6	10000	-3	997
PSID7	5001	-3	3
PSID8	5001	-10	10
PSID9	5001	-100	100
PSID10	5001	-500	500
PSID11	10001	-3	3
PSID12	10001	-10	10
PSID13	10001	-100	100
PSID14	10001	-500	500

Tab. 9: Number of iterations, cpu time, and flops required by Kaczmarz and OPAPLS to solve linear systems from the Matlab gallery with an indefinite symmetric part, described in Table 7, without preconditioning.

		Kaczmarz			OPAPLS			
Matrix	m	$cycles$	$tcpu$	$flops(10^9)$	$Iter$	$tcpu$	$flops(10^9)$	r_{max}
PSI1	5000	20	1.184	0.525000000	4	0.0124	0.25004000	1.0000000
PSI2	1000	10839	127.679	10.84000000	2740	90.505	5.48748000	1.0000000
PSI3	5000	58	3.441	1.475000000	40	0.165	2.05040000	1.0000000
PSI4	5000	1	0.064	0.0468	5043	13.339	364.200430	9.3932957
PSI5	5000	**	**	**	4	3.778	0.25004000	1.0000000
PSI6	5000	34	2.335	0.87500000	27	0.101	1.40027000	1.0000000
PSI7	5000	51	3.569	1.30000000	37	0.129	1.90037000	1.0000000
PSI8	5000	258	17.648	6.47500000	98	0.290	5.07598000	8.6559621
PSI9	6000	1375	128.278	49.5360000	285	1.358	20.9914200	7.5095462
PSI10	5001	20	1.171	0.52521002	4	0.006	0.25014002	1.0000000

According to the results in Table 10, the number of iterations, as well as the cpu time used for OPAPLS, increases as the length of the interval $[a, a_{max}]$ grows; and the condition number of the matrix depends on this length. For Kaczmarz method the opposite occurs: the computational work and the cpu time decrease when the values of $|a|$, $|a_{max}|$, and the length of the interval $[a, a_{max}]$ increase. We observe that the angle between two consecutive hyper-planes depends on these values. Let π_i and π_j be consecutive hyper-planes whose normal vectors are A_i and A_j, respectively, where A_i and A_j are the i-th and j-th rows of A. Then, if $L = a_{max} - a$, and $\theta_{i,j}$ denotes the angle between π_i and π_j, by definition of the angle between hyper-planes, we have

Tab. 10: Number of iterations, cpu time, and flops required by Kaczmarz and OPAPLS to solve tridiagonal linear systems with an indefinite symmetric part, described in Table 8, without preconditioning.

		Kaczmarz			OPAPLS			
Matrix	m	$cycles$	$tcpu$	$flops(10^{10})$	$Iter$	$tcpu$	$flops(10^{10})$	r_{max}
PSID1	5000	1654	105.614	4.13750000	1975	4.971	9.8919750	8.3491821
PSID2	5000	89	5.603	0.22500000	4305	10.453	21.574305	7.5422094
PSID3	5000	22	1.386	0.05750000	11634	27.521	58.216634	8.3739901
PSID4	10000	6209	1427.399	62.1000000	1053	5.456	21.102106	9.1480524
PSID5	10000	363	84.360	3.64000000	6063	28.604	121.47213	7.6934738
PSID6	10000	38	8.739	0.39000000	15737	72.468	314.92147	8.2741977
PSID7	5001	3763	241.754	9.413764400	973	2.586	4.8754224	7.3971303
PSID8	5001	1076	68.915	2.693577100	1202	3.087	6.0261115	5.8441152
PSID9	5001	102	6.477	0.257603010	4843	11.829	24.237034	6.4805663
PSID10	5001	23	1.491	0.060024002	5193	12.604	26.003090	8.1546386
PSID11	10001	7304	1698.359	73.0646110	1391	6.945	27.858353	6.3738294
PSID12	10001	2082	474.943	20.8341660	2821	13.965	56.486937	7.3010586
PSID13	10001	191	43.539	1.92038400	4381	20.896	87.736306	5.3416070
PSID14	10001	39	8.952	0.40008000	9060	42.599	181.35438	8.6500760

that:

$$|cos(\theta_{i,j})| = \frac{|A_i^T A_j|}{\|A_i\|_2 \|A_j\|_2} = \frac{|a_{ii} - a_{jj}|}{\sqrt{2 + a_{ii}^2}\sqrt{2 + a_{jj}^2}} = \frac{L}{(m-1)\sqrt{2 + a_{ii}^2}\sqrt{2 + a_{jj}^2}}.$$

So, $|cos(\theta_{ij})|$ decreases directly proportionally to L and inversely proportionally to m, $|a_{ii}|$ and $|a_{jj}|$, for $1 \leq i, j \leq m$, $a \leq a_{ii}, a_{jj} \leq a_{max}$. We note that for a fixed m, when the values of $|a|$, $|a_{max}|$ and L decrease, the cosine of most angles between hyper-planes increases, that is, the angle θ_{ij} decreases implying that the Kaczmarz method requires many more cycles for convergence, which increases the number of flops and the cpu time.

For some matrices in Table 10, we observe that, the Kaczmarz method uses fewer number of flops than OPAPLS. However, it requires more cpu time. The Kaczmarz method requires the computation of an inner product between a row of A and a vector to obtain a new vector for the next row, and so on, until it reaches the last row. This sequential calculations increases the required cpu time to obtain the solution.

In our fourth experiment, we consider the solution of consistent dense linear systems for which the coefficient matrix A was obtained in the following way: Take C as a 100×100 dense matrix from the Matlab gallery and consider the matrix A such that, for $1 \leq i, j \leq 100$, $a_{ij} = c_{ij}$ if $i < j$; for $i > j$, $a_{ij} = -c_{ji}$; and $a_{ii} = a + \left(\frac{i-1}{m-1}\right)(a_{max} - a)$, where a and a_{max} are given constants such that the symmetric part of A is indefinite. The Matlab commands used to generate the matrix C, and the values assigned to a and a_{max} are shown in Table 11.

Tab. 11: Description of the Matlab commands, and the parameters a and a_{max}, for dense matrices with an indefinite symmetric part.

Matrix	Matlab Commands for matrix C	a	$amax$
DENSEPSI1	C=gallery('lehmer',100)	-1	6
DENSEPSI2	C=gallery('toeppd',100)	-1	6
DENSEPSI3	C=gallery('pei',100)	-1	6
DENSEPSI4	C=gallery('randhess',100)	-1	6
DENSEPSI5	C=gallery('parter',100)	-1	6
DENSEPSI6	C=gallery('lehmer',100)	-2	6
DENSEPSI7	C=gallery('toeppd',100)	-2	6
DENSEPSI8	C=gallery('pei',100)	-2	6
DENSEPSI9	C=gallery('randhess',100)	-2	6
DENSEPSI10	C=gallery('parter',100)	-2	6
DENSEPSI11	C=gallery('lehmer',100)	-2	10
DENSEPSI12	C=gallery('toeppd',100)	-2	10
DENSEPSI13	B=gallery('pei',100)	-2	10
DENSEPSI14	C=gallery('randhess',100)	-2	10
DENSEPSI15	C=gallery('parter',100)	-2	10

For these examples, we report, in tables 12 and 13, the number of iterations required by OPAPLS and all the other methods to reach convergence. Table 12 shows the results obtained when the solution vector x is $(1, 1, \cdots, 1)^T$; and Table 13 shows the results obtained when $b = (1, 1, \cdots, 1)^T$.

Tab. 12: Number of iterations required by GMRES(20), GMRES(40), BICGSTAB, Kaczmarz and OPAPLS for solving dense linear systems for which the coefficient matrix has an indefinite symmetric part, described in Table 11, with the solution vector is $(1, 1, \cdots, 1)^T$, without preconditioning.

	$GMRES(20)$	$GMRES(40)$	BICGSTAB	Kaczmarz	OPAPLS	
Matrix	$iter$	$iter$	$iter$	$cycles$	$iter$	r_{max}
DENSEPSI1	**	**	216	131	495	1.0000000
DENSEPSI2	717	581	**	2484	649	1.9270749
DENSEPSI3	159	23	61	738	749	1.0000000
DENSEPSI4	**	1267	**	191	1680	3.2593333
DENSEPSI5	**	**	**	**	7566	1.0000000
DENSEPSI6	**	32	124	133	181	1.0000000
DENSEPSI7	97	811	**	6169	1218	1.1647514
DENSEPSI8	**	22	64	734	266	1.0000000
DENSEPSI9	**	**	**	823	2489	3.3312492
DENSEPSI10	**	**	**	120	112	1.0000000
DENSEPSI11	**	34	133	87	205	1.0000000
DENSEPSI12	440	17	36	1294	447	5.1925796
DENSEPSI13	16	16	36	495	303	1.0000000
DENSEPSI14	**	**	335	1379	4154	3.0944160
DENSEPSI15	**	**	**	**	**	1.0000000

The results from tables 12 and 13 indicate clearly that the Kaczmarz

Tab. 13: Number of iterations required by GMRES(20), GMRES(40), BICGSTAB, Kaczmarz and OPAPLS for solving dense linear systems for which the coefficient matrix has an indefinite symmetric part, described in Table 11, with $b = (1, 1, \cdots, 1)^T$, without preconditioning.

	$GMRES(20)$	$GMRES(40)$	BICGSTAB	Kaczmarz	OPAPLS	
Matrix	$iter$	$iter$	$iter$	$cycles$	$iter$	r_{max}
DENSEPSI1	**	41	**	143	611	3.3281386
DENSEPSI2	773	316	**	2700	830	4.2149836
DENSEPSI3	179	24	146	866	829	3.9461871
DENSEPSI4	**	16188	**	147	2333	8.5489338
DENSEPSI5	**	**	**	**	**	8.9455954
DENSEPSI6	**	**	**	141	195	3.3564561
DENSEPSI7	1072	873	**	6949	806	4.0339567
DENSEPSI8	**	23	164	846	273	3.9644313
DENSEPSI9	**	**	**	1043	3956	8.3114409
DENSEPSI10	**	**	**	**	**	9.0260881
DENSEPSI11	**	**	**	93	246	1.9881283
DENSEPSI12	464	23	**	1339	453	8.4347586
DENSEPSI13	17	17	89	570	360	4.7793751
DENSEPSI14	**	**	301	1846	4991	8.5786460
DENSEPSI15	**	**	**	**	**	8.9939191

method and the proposed OPAPLS scheme are more effective for solving these small systems for which the dense coefficient matrix has an indefinite symmetric part. Moreover, the results obtained in Tables 9, 10, 12 and 13 seem to indicate that the proposed method, OPAPLS, is the best option for solving linear systems when the coefficient matrix has an indefinite symmetric part.

In our fifth experiment we consider the solution of consistent rectangular linear systems, whose matrices ($A \in \mathbb{R}^{m \times n}$) are described in Table 14. The solution for each system is the vector $(1, 1, 1, ..., 1)^T$. The first seven matrices were taken from the collection available in the portal matrix market (www.matrixmarket.com). The matrices labeled as $R8$ and $R9$ were obtained following a model presented in [6], which combines some matrices in the following way: $R_8 = \begin{pmatrix} R1 \\ \widehat{R4} \end{pmatrix}_{1252 \times 320}$ and $R9 = \begin{pmatrix} R1 \\ \widehat{R6} \end{pmatrix}_{1641 \times 320}$, with $\widehat{R4} = \begin{pmatrix} 0_{219 \times 235} & R4 \end{pmatrix}$ and $\widehat{R6} = \begin{pmatrix} 0_{608 \times 132} & R6 \end{pmatrix}$.

The last two matrices in Table 14 were generated in Matlab. Matrix $R10$ was obtained with the Matlab command gallery('lauchli', m, μ), which is an $(m + 1) \times m$ matrix such that the first row has all the component equal to one and the following m rows coincide with $\mu I_{m \times m}$, where μ is a given scalar. Matrix $R11$ was generated with the Matlab command gallery('sprand',m,n,d) and it has, approximately, $m \times n \times d$ random nonzero entries, where d is the nonzero density of the matrix.

Tab. 14: Description of rectangular matrices in $\mathbb{R}^{m\times n}$, $m > n$.

Matrix	Description	m	n
R1	WELL1033	1033	320
R2	WELL1850	1850	712
R3	ABB313	313	176
R4	ASH219	219	85
R5	ASH331	331	104
R6	ASH608	608	188
R7	ASH958	958	292
R8	ARTF1252	1252	320
R9	ARTF1641	1641	320
R10	$gallery('lauchli', 5000, 10^{-4})$	5001	5000
R11	sprand(m,n.0.002)	4000	3000

The Kaczmarz method and the OPAPLS strategy are the only methods that can be directly applied to rectangular systems. The methods CG and RA are applied to the corresponding normal equations $A^TAx = A^Tb$. In particular, instead of the standard CG method, we use the specialized version CGNE, also known as Craig's method, fully described in [4, Ch. 8]. For CGNE and RA an additional matrix-vector product with A^T must be computed at each iteration, and so mn flops must be added to their required number of flops.

We report, in Table 15, the number of iterations and cpu time required to reach convergence in each case.

From Table 15, we can observe that, in general, CGNE requires less cpu time that the others methods. Among the others methods our methodology reach convergence for all the problems with a competitive cpu time. However, all strategies attained convergence, except Kaczmarz for matrix R1. Notice that Kaczmarz and OPAPLS can be applied directly to the rectangular system. Contrary, RA and CGNE involve the normal equations system in their formulations, even thought the matrix A^TA is not generated.

For our last experiment we use the scheme proposed in (9) to solve linear systems of equations, square or rectangular, subject to box constraints on the variables. The solution of each system is the vector $(1, 1, \cdots, 1)^T$. The competitors GMRES, BICGSTAB, Kaczmarz, RA, ORM and CG cannot be applied to constraint problems. Hence, for this experiment, we only report the results using OPAPLS. We consider the set of squared matrices described in Table 7 with different constraints: $0 \leq x_i \leq 2$, for $1 \leq i \leq n$, $-5 \leq x_i \leq 5$ for $1 \leq i \leq n$ and $-100 \leq x_i \leq 100$ for $1 \leq i \leq n$. The obtained results

Tab. 15: Number of iterations and cpu time required by RA, CGNE, Kaczmarz and OPAPLS for solving rectangular systems, whose coefficient matrices were described in Table 14, without preconditioning.

	RA		*CGNE*		*Kaczmarz*		*OPAPLS*		
Matrix	*iter*	*tcpu*	*iter*	*tcpu*	*cycles*	*tcpu*	*iter*	*tcpu*	r_{max}
R1	1380	12.887	185	0.095	**	**	3380	2.934	2.0202712
R2	1113	94.350	491	0.843	16931	111.562	2059	3.180	2.1468575
R3	146	0.252	81	0.000	122	0.093	129	0.053	1.0000000
R4	54	0.034	42	0.000	29	0.015	58	0.012	1.0000000
R5	42	0.024	34	0.000	19	0.014	43	0.011	1.0000000
R6	56	0.176	49	0.000	19	0.028	60	0.021	1.0000000
R7	61	0.513	511	0.015	22	0.060	66	0.032	1.0000000
R8	431	5.402	122	0.046	484	1.595	667	0.639	4.2227713
R9	64	0.992	47	0.043	27	0.185	54	0.062	1.0000000
R10	2	17.788	1	0.028	1	0.054	1	0.004	1.0000000
R11	910	5.021	511	0.043	1	0.045	1094	4.332	1.0000000

are shown in Table 16. Table 17 shows the values for r_{max} obtained in the experiment of Table 17.

Tab. 16: Number of iterations and cpu time required by OPAPLS to solve linear systems whose matrix has an indefinite symmetric part, described in Table 7, subject to box constraints, without preconditioning.

	Unconstrained		$0 \leq x_i \leq 2$		$-5 \leq x_i \leq 5$		$-100 \leq x_i \leq 100$	
Matrix	*Iter*	*tcpu*	*Iter*	*tcpu*	*Iter*	*tcpu*	*Iter*	*tcpu*
PSI1	4	0.02	6	0.015	5	0.015	5	0.015
PSI2	2740	112.81	2413	81.572	2413	82.134	2413	81.588
PSI3	37	0.20	50	0.202	50	0.218	50	0.202
PSI4	5043	17.43	4	0.000	4	0.015	4	0.015
PSI5	4	4.59	6	4.945	6	5.085	6	5.038
PSI6	27	0.14	37	0.156	33	0.124	33	0.124
PSI7	37	0.16	47	0.171	47	0.171	47	0.171
PSI8	97	0.37	83	0.296	112	0.327	107	0.312
PSI9	308	1.81	294	1.591	306	1.482	291	1.606
PSI10	4	0.02	6	0.015	5	0.031	5	0.015

We solve some additional squared systems with box constraints using OPAPLS. The additional systems are the ones previously described in Table 8, and for these problems we use the same box constraints, described in Table 16. The right hand side vector for each system is set in such a way the

Tab. 17: Maximum value taken by the elements of the residual vector using OPAPLS in the experiment shows in Table 16.

	$0 \leq x_i \leq 2, 1 \leq i \leq n$	$-5 \leq x_i \leq 5, 1 \leq i \leq n$	$-100 \leq x_i \leq 100, 1 \leq i \leq n$
Matrix	r_{max}	r_{max}	r_{max}
PSI1	1.00000000	1.00000000	1.00000000
PSI2	1.00000000	1.00000000	1.00000000
PSI3	1.00000000	1.00000000	1.00000000
PSI4	1.00000000	1.0000000	1.00000000
PSI5	1.00000000	1.00000000	1.00000000
PSI6	1.00000000	1.00000000	1.0000000
PSI7	1.00000000	1.00000000	1.00000000
PSI8	1.76000000	8.75703240	8.65596210
PSI9	2.50000000	6.10399220	7.50954620
PSI10	1.00000000	1.00000000	1.00000000

solution vector is $(1, 1, 1, ..., 1)^T$. The required number of iterations and cpu time are shown in Table 18. Table 19 shows the values for r_{max} obtained in the experiment of Table 19.

Tab. 18: Number of iterations and cpu time required by OPAPLS to solve linear systems whose matrix has an indefinite symmetric part, described in Table 8, subject to box constraints, without preconditioning.

	Unconstrained		$0 \leq x_i \leq 2$		$-5 \leq x_i \leq 5$		$-100 \leq x_i \leq 100$	
Matrix	*Iter*	*tcpu*	*Iter*	*tcpu*	*Iter*	*tcpu*	*Iter*	*tcpu*
PSID1	1975	5.08	1879	5.085	2079	5.725	1819	4.992
PSID2	4305	11.06	5103	13.026	4620	12.183	4196	10.935
PSID3	11634	28.54	8390	21.325	9565	24.102	10795	27.222
PSID4	1053	5.50	1287	7.129	1558	8.408	1033	5.678
PSID5	6063	29.25	8229	40.887	7367	36.660	6939	34.679
PSID6	15737	73.78	16326	78.952	10359	50.544	16033	78.234
PSID7	973	2.66	902	2.745	654	1.903	981	3.026
PSID8	1202	3.27	1205	3.354	977	2.839	1201	3.572
PSID9	4843	11.93	4295	11.138	2410	6.614	4554	11.824
PSID10	5193	12.83	9321	23.727	6328	16.052	5195	13.416
PSID11	1391	7.05	1415	7.441	1012	5.569	1384	7.987
PSID12	2821	14.05	1875	9.921	1956	10046	2709	14.211
PSID13	4381	20.88	3948	19.905	4366	21.668	4341	21.652
PSID14	9060	42.35	8513	41.901	8161	40.045	7976	39.374

The results of tables 16 and 18 indicate that our methodology permits to

Tab. 19: Maximum value taken by the elements of the residual vector using OPAPLS in the experiment shows in Table 18.

	$0 \leq x_i \leq 2{,}1 \leq i \leq n$	$-5 \leq x_i \leq 5{,}1 \leq i \leq n$	$-100 \leq x_i \leq 100{,}1 \leq i \leq n$
Matrix	r_{max}	r_{max}	r_{max}
PSID1	1.0906727	6.3565440	8.3491821
PSID2	1.0000000	6.0120183	7.5422094
PSID3	1.0008155	5.9938245	8.3739901
PSID4	1.1997200	6.7983198	9.1480524
PSID5	1.0093910	5.9728412	7.6934738
PSID6	1.0000652	5.9943319	8.2741977
PSID7	1.2494000	5.7753408	7.3971303
PSID8	1.0653689	6.3549091	5.8441152
PSID9	1.0000000	5.9530400	6.4805663
PSID10	1.0000000	5.9830990	8.1546386
PSID11	1.2498500	6.2392268	6.3738294
PSID12	1.0903636	6.3560000	7.3010586
PSID13	1.0000000	5.9421503	5.3416070
PSID14	1.0002467	6.0055888	8.6500760

obtain particular solutions within a small cpu time perturbation.

The machinery OPAPLS also works for solving rectangular systems subject to box constraints. In order to observe its performance, we now solve consistent underdetermined systems subject to constraints. The set of underdetermined systems to be considered are described in Table 20. Table 21 shows, in the first column the label for each example, in the second column the name of the matrix, and in the third column the imposed constraints that force a particular solution. The obtained results using OPAPLS are shown in the last two columns.

Tab. 20: Description of some rectangular matrices in $\mathbb{R}^{m\times n}$, $m < n$.

Matrix	Matlab command	m	n
R12	sprand(m,n.0.1)	500	4000
R13	sprand(m,n.0.1)	2000	5000

The underdetermined systems considered for this experiment have an infinite number of solutions. Particulary, the proposed strategy found one of them. There is not much difference in cpu time when solving constrained problems since the projection over the box constraint set is simple and requires low computational cost.

Tab. 21: Number of iterations and cpu time required by OPAPLS to solve underdetermined linear systems, whose matrices were described in Table 20, subject to constraints, without preconditioning.

Problem	Matrix	Constraint	*Iter*	*tcpu*	r_{max}
CS1	*R*12	No constraint	47	1.263	1.0
CS2	*R*12	$x_1 = x_2 = x_3 = 1$	51	1.294	1.0
CS3	*R*12	$x_1 = x_2 = x_3 = 1$, $0 \leq x_i \leq 2$ for $i > 3$	51	1.326	1.0
CS4	*R*12	$x_1 = x_2 = x_3 = 1$, $-5 \leq x_i \leq 5$ for $i > 3$	51	1.326	1.0
CS5	*R*12	$x_1 = x_2 = x_3 = 1$, $-100 \leq x_i \leq 100$ for $i > 3$	51	1.404	1.0
CS6	*R*12	$x_1 = x_{\frac{n}{2}} = x_n = 1$	51	1.357	1.0
CS7	*R*12	$x_1 = x_{\frac{n}{2}} = x_n = 1$ and $0 \leq x_i \leq 2$, $i \notin \{1, \frac{n}{2}, n\}$	51	1.357	1.0
CS8	*R*12	$x_1 = x_{\frac{n}{2}} = x_n = 1$ and $-5 \leq x_i \leq 5$, $i \notin \{1, \frac{n}{2}, n\}$	51	1.341	1.0
CS9	*R*12	$x_1 = x_{\frac{n}{2}} = x_n = 1$, and $-100 \leq x_i \leq 100$ $i \notin \{1, \frac{n}{2}, n\}$	51	1.419	1.0
CS10	*R*13	No constraint	97	12.932	1.0
CS11	*R*13	$x_1 = x_2 = x_3 = 1$	95	12.526	1.0
CS12	*R*13	$x_1 = x_2 = x_3 = 1$, $0 \leq x_i \leq 2$ for $i > 3$	95	12.339	1.0
CS13	*R*13	$x_1 = x_2 = x_3 = 1$, $-5 \leq x_i \leq 5$ for $i > 3$	95	12.448	1.0
CS14	*R*13	$x_1 = x_2 = x_3 = 1$ and $-100 \leq x_i \leq 100$ for $i > 3$	95	12.792	1.0
CS15	*R*13	$x_1 = 1$, $x_{\frac{n}{2}} = 1$, $x_n = 1$	95	12.698	1.0
CS16	*R*13	$x_1 = x_{\frac{n}{2}} = x_n = 1$ and $0 \leq x_i \leq 2$ for $i \notin \{1, \frac{n}{2}, n\}$	103	13.291	1.0
CS17	*R*13	$x_1 = x_{\frac{n}{2}} = x_n = 1$ and $-5 \leq x_i \leq 5$ for $i \notin \{1, \frac{n}{2}, n\}$	103	14.071	1.0
CS18	*R*13	$x_1 = x_{\frac{n}{2}} = x_n = 1$ and $-100 \leq x_i \leq 100$ $i \notin \{1, \frac{n}{2}, n\}$	103	13.322	1.0

4 Conclusions

We have presented an optimization strategy for solving different kinds of consistent linear systems. The proposed method finds the solution by searching a local minimizer of a novel non-quadratic convex function. In the case of solving linear systems, a relevant feature is that the new scheme does not require the coefficient matrix to be square. In this work, we use the Spectral Projected Gradient (SPG) method to solve the optimization problems. The SPG is a globally convergent method that has a low computational and low storage cost, and it only requires first order information. However, any other globally convergent low-cost optimization method can be used.

Our numerical results indicate that the new machinery is suitable for solving large-scale and sparse, as well as small and dense, problems for which the coefficient matrix has no special characteristics. Moreover, it allows one to add easily convex constraints to the optimization approach. Adding convex constraints is useful for several different reasons. One of them is that it imposes regularity to the optimization problem. Another advantage is that if the linear problem has an infinite number of solutions, a specific type of solution can be found by conveniently setting the convex constraints. Furthermore, the proposed strategy also allows to solve linear feasibility problems, since these problems can be treated as a linear system of equation subject to box constraints, for which some slack variables are introduced. On the other

hand, the choice of the scaling parameter guarantees that the value of the new non quadratic function is bounded above for all iterations. So, overflow or loss of accuracy can be avoided.

Finally, the new optimization machinery can also benefit from the use of preconditioning strategies, which plays a key role in the presence of very ill-conditioned problems. However, in the case of the matrices with indefinite symmetric part where generic preconditioning techniques are still under development, the OPAPLS method seems to be a competitive approach for this kind of problems.

References

[1] D. Bertsekas. *Nonlinear Programming*. Athena Scientific, Belmont, Massachusetts, 2 edition, 1999. ISBN 978-1886529007.

[2] G. Golub and C. Van Loan. *Matrix computations*. Johns Hopkins University Press, 3 edition, 1996. ISBN 978-0801854149.

[3] M.R. Hestenes. *Conjugate Direction Methods in Optimization*. Springer-Verlag New York, 1 edition, 1980. ISBN 978-1-4612-6048-6.

[4] Y. Saad. *Iterative methods for sparse linear systems*. SIAM, Philadelphia, 1 edition, 2003. ISBN 978-0-89871-534-7, doi: 10.1137/1.9780898718003.

[5] C. Brezinski. *Projection methods for systems of equations*. North Holland, Amsterdam, 1 edition, 1997. ISBN 978-0444827777.

[6] A. Björck. *Numerical methods for least square problems*. SIAM, Philadelphia, 1996. ISBN 978-0-89871-360-2.

[7] P.C Hansen, V. Pereya, and G. Scherer. *Least Squares Data Fitting with Applications*. Johns Hopkins University Press, Maryland, USA, 1 edition, 2013. ISBN 9781421407869.

[8] E.G. Birgin, J.M. Martínez, and M. Raydan. Algorithm 813: SPG - software for convex-constrained optimization. *ACM Transactions on Mathematical Software*, 27(3):340–349, 2001. doi: 10.1145/502800.502803.

[9] E.G. Birgin, J.M. Martínez, and M. Raydan. Nonmonotone spectral projected gradient methods on convex sets. *SIAM J. Opt.*, 10(4):1196–1211, 2000. doi: 10.1137/S1052623497330963.

[10] Raydan M. The barzilai and borwein gradient method for the large-scale unconstrained minimization problem. *SIAM J. Opt.*, 7(1):26–33, 1997. doi: 10.1137/S1052623494266365.

[11] E.G. Birgin, J.M. Martínez, and M. Raydan. Spectral projected gradient methods: Review and perspectives. *Journal of Statistical Sofware*, 60(3):1–21, 2014. doi: 10.18637/jss.v060.i03.

[12] M.A. Diniz-Ehrhardt, M.A. Gomes-Ruggiero, J.M. Martínez, and S.A. Santos. Augmented lagrangian algorithms based on the spectral projected gradient method for solving nonlinear programming problems. *Journal of Optimization Theory and Applications*, 123(3):497–517, 2004. doi: 10.1007/s10957-004-5720-5.

[13] M.A. Gomes-Ruggiero, J.M. Martínez, and S.A Santos. Spectral projected gradient method with inexact restoration for minimization with nonconvex constraints. *SIAM Journal on Scientific Computing*, 31(3):1628–1652, 2009. doi: 10.1137/070707828.

[14] R. Escalante and M. Raydan. *Alternating projection methods.* SIAM, Philadelphia, 1 edition, 2011. ISBN 978-1-611971-93-4.

[15] C. Brezinski. Variations on richardson's method and acceleration. *Bull. Soc. Math. Belg.*, 3(Supplement):33–44, 1996. doi: 10.1.1.11.7024.

[16] W. La Cruz and M. Raydan. Residual iterative schemes for large-scale nonsymmetric positive definite linear systems. *Computacional and applied mathematics*, 27(2):151–173, 2008. doi: 10.1590/S0101-82052008000200003.