Trends in Computational and Applied Mathematics, **23**, N. 2 (2022), 349-361
Sociedade Brasileira de Matemática Aplicada e Computacional
Online version ISSN 2676-0029
www.scielo.br/tcam
doi: 10.5540/tcam.2022.023.02.00349

A New Approach to the Splitting Factor Preconditioner Applied to Linear Programming Problems

P. A. KIKUCHI[1*] and A. R. L. OLIVEIRA[2]

Received on September 14, 2021 / Accepted on December 27, 2021

ABSTRACT. In this paper, we present the results of a new approach to the splitting factor preconditioner, which is a preconditioner based on the Incomplete Cholesky factorization and the splitting preconditioner. In previous work for small linear programming problems, the preconditioner was applied in all iterations of the interior point method and compared with the splitting preconditioner also applied in all iterations. In this paper, we will do a hybrid approach, in which in the first iterations the preconditioner is the Incomplete Cholesky Factorization, and in the last iterations, the preconditioner used is the splitting factor preconditioner or the splitting preconditioner. The results obtained show that even in the hybrid approach, the splitting factor preconditioner achieved better performance.

Keywords: preconditioner, linear programming, signal processing, compressive sensing.

1 INTRODUCTION

A linear optimization problem consists of minimizing or maximizing a linear objective function, subject to a finite set of linear constraints. Restrictions can be either equality or inequality. The standard form of a linear optimization problem is called a primal problem and is given by:

$$\begin{array}{ll} \text{minimize} & c^T x \\ \text{subject to} & Ax = b \\ & x \geq 0, \end{array} \tag{1.1}$$

where $A \in \mathbb{R}^{m\times n}$ is a constraint matrix, $x \in \mathbb{R}^n$ is a column vector, whose components are called primal variables, and $b \in \mathbb{R}^m$ and $c \in \mathbb{R}^n$ are column vectors, with c being the costs associated with the elements of x.

*Corresponding author: Paula Aparecida Kikuchi – E-mail: paula.kikuchi@uems.br
[1]Universidade Estadual de Mato Grosso do Sul, R. Emílio Mascoli, 275, 79950-000 Naviraí, MS, Brazil – E-mail: paula.kikuchi@uems.br https://orcid.org/0000-0002-9202-9112
[2]Departamento de Matemática Aplicada, IMECC, Universidade Estadual de Campinas, R. Sérgio Buarque de Holanda, 651, 13083-859 Campinas, SP, Brazil – E-mail: aurelio@ime.unicamp.br https://orcid.org/0000-0002-6471-4710

We define a vector $\bar{x}$ such that $A\bar{x} = b$, $\bar{x} \geq 0$, as a feasible solution. A x^* solution is optimal when, in addition to being a feasible solution, it admits the lowest possible value for the objective function.

In case the restrictions $Ax = b$ are inconsistent, we will have no feasible solution. This problem is called infeasible. If there is a feasible solution, the objective function can be unlimited or limited in the domain, being thus called an unlimited and limited problem, respectively. There is no optimal solution if, and only if, the problem is infeasible or unlimited.

For a given primal problem, we can always build an associated problem, which we call dual problem [1], which consists of the same given components, arranged differently. The dual problem of (1.1) is given by:

$$\begin{array}{ll} \text{maximize} & b^T y \\ \text{subject to} & A^T y \leq c \\ & y \in \mathbb{R}^m, \end{array} \tag{1.2}$$

which is equivalent to

$$\begin{array}{ll} \text{maximize} & b^T y \\ \text{subject to} & A^T y + z = c \\ & z \geq 0 \\ & y \in \mathbb{R}^m, \end{array} \tag{1.3}$$

where y being a column vector belonging to $\mathbb{R}^m$, called a dual variable vector, and z a column vector belonging to $\mathbb{R}^n$, a slack variable vector.

Below we define the Optimality Conditions (algebraic conditions are satisfied when solving linear programming problems).

Optimality Conditions

Given a (x, y, z) point, it will be optimal for primal and dual problems if, and only if, the following conditions are satisfied:

$$\begin{cases} Ax = b \\ A^T y + z = c \\ XZe = 0 \\ (x, z) \geq 0, \end{cases} \tag{1.4}$$

X and Z are diagonal matrices formed by the elements of the vectors x and z, respectively, and e the vector of numbers one.

1.1 Newton's Method Applied to Optimality Conditions for Linear Programming Problems

Note that we can rewrite the optimality conditions (1.4) for a linear programming problem in a slightly different way, defining a function F from $\mathbb{R}^{2n+m}$ to $\mathbb{R}^{2n+m}$:

$$F(x,y,z) = \begin{pmatrix} F_p \\ F_d \\ F_a \end{pmatrix} = \begin{pmatrix} A^T y - c + z \\ Ax - b \\ XZe \end{pmatrix} = 0,$$

supposing that $(x,z) \geq 0$.

To solve this nonlinear system, we applied the Newton method [12] to the Optimality Conditions, approximating F by the truncated Taylor series and proceeding in a similar way to the process described in [12] referring to the Newton method for several variables. Thus, we obtain:

$$\begin{aligned} F\left(x^{k+1}\right) &\approx F\left(x^k\right) + \nabla F\left(x^k\right)^T \left(x^{k+1} - x^k\right) = 0 \\ \Rightarrow -F\left(x^k\right) &= \nabla F\left(x^k\right)^T \left(x^{k+1} - x^k\right) \\ \Rightarrow x^{k+1} &= x^k - \left(\nabla F\left(x^k\right)\right)^{-T} F\left(x^k\right) \\ \Rightarrow x^{k+1} &= x^k + d \,, d = -\left(\nabla F\left(x^k\right)\right)^{-T} F\left(x^k\right). \end{aligned}$$

The conditions such that the method converges can be found at [12].

2 PRIMAL-DUAL METHOD

The 1984 publication of [7] was probably the most significant event in linear programming since the simplex method [16]. One of the reasons why the paper aroused great interest was because the author stated that the method had an excellent performance for large linear problems. This paper caused a revolution in the research of linear programming problems, leading to computational and theoretical advances in this area. From this paper and other works, the Interior point methods emerged, which until today have been developing. The theory, together with computational experiments, shows that primal-dual algorithms perform better than other Interior Point Methods, as well as perform better than the Simplex method for large problems [16]. We describe below the Predictor-Corrector Method.

2.1 Predictor-Corrector Method

The Mehrotra Predictor-Corrector Method [9], which is used in this paper, is based on three components:

- Affine-scale direction, which corresponds to the predictor step, which consists of finding a direction of the affine-scale optimization problem.

- Centering direction, defined by the σ parameter of the primal-dual path-following method [16], remembering that the centering direction prevents solutions along the iterations from approaching the coordinated axes, thus maintaining the strictly positive $x_i z_i$ product. In the primal-dual path-following method we add a perturbation μ to the complementarity condition ($XZe = \mu e$), the centering direction allows us to take larger steps of Newton's direction, since the positivity condition is only violated with a step size larger than when we consider μ equal to zero.
- Correction direction, which corresponds to the correcting step, in which we calculate the non-linear correction, trying to compensate for the linear approximation of the Newton Method. For example, in a linear programming problem in the standard form, the non-linear term is given by XZe.

The predictor-corrector method consists of applying the Newton Method twice, using the same Hessian matrix. Unlike other methods, which disregard the residue of non-linear terms and their linear approximation using the Newton method, in this method we will introduce the corrections for the approximations of these non-linear terms XZe.

Thus, the search direction is obtained by solving two different linear systems, but with the same matrix of coefficients. Initially, the direction *affine-scale* $(\Delta_a x, \Delta_a y, \Delta_a z)$, also called *predictive direction*, is calculated by solving the system:

$$\begin{pmatrix} A & 0 & 0 \\ 0 & A^T & I \\ Z & 0 & X \end{pmatrix} \begin{pmatrix} \Delta_a x \\ \Delta_a y \\ \Delta_a z \end{pmatrix} = \begin{pmatrix} r_p \\ r_d \\ r_a \end{pmatrix}, \tag{2.1}$$

where $r_p = b - Ax$, $r_d = c - A^T y - z$ and $r_a = -XZe$. Thus, the right side is modified by making $r_p = r_d = 0$, and replacing r_a with $r_c = \mu e - \Delta_a X \Delta_a Z e$, where the number μ is the centering parameter, $\Delta_a X = diag(\Delta_a x)$ and $\Delta_a Z = diag(\Delta_a z)$; these would be the residuals of the next iteration, if the primal step size (α_p) and the dual step size (α_d) are equal to one. The obtained system is in the form:

$$\begin{pmatrix} A & 0 & 0 \\ 0 & A^T & I \\ Z & 0 & X \end{pmatrix} \begin{pmatrix} \Delta_c x \\ \Delta_c y \\ \Delta_c z \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \\ r_c \end{pmatrix}, \tag{2.2}$$

and with that, we get the so-called *centering correction direction* $(\Delta_c x, \Delta_c y, \Delta_c z)$.

The search direction $(\Delta x, \Delta y, \Delta z)$, finally, will be given by the sum of the two previous directions:

$$(\Delta x, \Delta y, \Delta z) = (\Delta_a x, \Delta_a y, \Delta_a z) + (\Delta_c x, \Delta_c y, \Delta_c z).$$

We can determine the search direction avoiding this sum, for that, instead of solving the system (2.2), we replace r_a in (2.1) with r_m, where

$$r_m = r_a + r_c = -XZe + \mu e - \Delta_a X \Delta_a Z e.$$

Thus, the system is given by:

$$\begin{pmatrix} A & 0 & 0 \\ 0 & A^T & I \\ Z & 0 & X \end{pmatrix} \begin{pmatrix} \Delta x \\ \Delta y \\ \Delta z \end{pmatrix} = \begin{pmatrix} r_p \\ r_d \\ r_m \end{pmatrix}. \tag{2.3}$$

Succinctly, in the predictor-corrector method, we first determine the predictive direction through the system (2.1), and then we solve the system (2.3), to determine the direction of search. For more details and a better understanding of the theory that comprises Interior Point Methods, see [15].

The resolution of linear systems, such as (2.1), is the computationally most expensive step of the interior point method. Fortunately, in this case, the two linear systems to be solved share the same coefficient matrix, which, in general, is large and sparse. Note that we can reformulate (2.1) to obtain linear systems with matrices that are symmetrical, more compact, and easier to handle than the original. This reformulation is possible because in all iterations, the components (x,z) are strictly positive and thus the matrices $X = diag(x)$ and $Z = diag(z)$ are non-singular. Thus, we can eliminate the variable Δz em (2.1), obtaining the equivalent system:

$$\begin{pmatrix} -D & A^T \\ A & 0 \end{pmatrix} \begin{pmatrix} \Delta x \\ \Delta y \end{pmatrix} = \begin{pmatrix} r_d - (X)^{-1} r_a \\ r_p \end{pmatrix}, \tag{2.4}$$

where $D = (X)^{-1}Z$. The system (2.4) is known as *augmented system.* After solving the previous system, we can calculate Δz using the equation:

$$\Delta z = (X)^{-1}(r_a - Z\Delta x).$$

Since the D matrix is non-singular, we can reduce the augmented system by eliminating Δx from the first equation and replacing it in the second, obtaining the following linear system:

$$A(D)^{-1}A^T\Delta y = r_p + A((D)^{-1}r_d - (Z)^{-1}r_a), \tag{2.5}$$

known as *normal equations*. Since $A(D)^{-1}A^T$ is the Schur complement of D in

$$\begin{pmatrix} -D & A^T \\ A & 0 \end{pmatrix}, \tag{2.6}$$

it is also usual to say that we use the Schur complement to solve the linear system.

Briefly, in the Interior Point Methods, a linear system corresponding to the Newton Method applied to the optimal conditions of the problem is obtained. The resolution of such a system can be done through direct methods, such as LU factorization and Cholesky factorization, or iterative methods, such as the Conjugate Gradient Method [14]. Cholesky factorization [5] is the most used method for solving the linear system, but as the calculation of factors can be very expensive, since, in problems with sparse matrix, its structure can be affected by filling, iterative methods prove to be a good alternative. As the system matrix is positive definite, we use the Conjugate Gradient Method for the resolution. This method performs well when the matrices are well-conditioned, if this does not occur, preconditioning becomes necessary.

3 PRECONDITIONERS FOR LINEAR SYSTEMS

The idea of preconditioning has been known for a long time [13], an example is a work proposed in [4].

Given the $Ax = b$ system, consider the following equivalent system:

$$M^{-1}AN^{-1}\tilde{x} = \tilde{b}, \text{ where } \quad \tilde{x} = Nx \quad \text{e} \quad \tilde{b} = M^{-1}b. \tag{3.1}$$

The system (3.1) is said to be preconditioned, and $M^{-1}AN^{-1}$ is called a preconditioned matrix.

When $N^T = M$, where N^T corresponds to the transpose of the matrix N, if A is symmetric, we obtain a symmetric preconditioned system $M^{-1}AM^{-T}$. In [5] preconditioner is defined as the matrix $P = M \cdot N$, so, if $N^T = M$, the preconditioner P is symmetrical.

In order to achieve rapid convergence, what we want to achieve is a $M^{-1}AN^{-1}$ matrix closer to the identity matrix than matrix A. For that, several preconditioning techniques was proposed [5, 13]. In this paper, we will focus on two preconditioners, incomplete Cholesky factorization and the splitting preconditioner [11].

3.1 Incomplete Factorizations

Given a sparse positive definite symmetric matrix, when we obtain its Cholesky Factorization (LL^T), fills in entries that are null in A may occur. Thus, L can be much denser than A, causing the need for more storage space and higher computational cost for the resolution of the linear system.

When an incomplete factorization of A is obtained, filling in certain entries is rejected. Thus, we can obtain a factorization of $A \approx \widetilde{L}\widetilde{L}^T$, imposing that $\widetilde{L}$ presents some sparse pattern similar to that of A. A proof of the existence of incomplete Cholesky factorization can be found in [10].

To increase the efficiency of preconditioners, several strategies have been proposed for the construction of preconditioners based on incomplete factorization. To decide whether an element is discarded during factorizing a matrix, two rules are established; one takes into account the position of the non-zero elements of the original matrix and the other takes into account the numerical value of the padding [2].

Below we highlight a technique that corresponds to the last-mentioned rule:

- Drop tolerance - Given a limit tolerance τ, non-zero elements are accepted in the incomplete factor if they are greater than τ.

Besides, we can also fix the amount of fill allowed in incomplete factors. This technique is known as *fixed fill-in* and predetermines the pattern of non-null elements of the incomplete factor, which is not necessarily that of matrix A. Thus, we can determine that there is no filling, that is, the positions in which all non-null elements of the original matrix are coincident with that of the incomplete factor, which is equivalent, for example, to the incomplete Cholesky factorization

without filling; or determine that a fixed number of padding be accepted in each column of the incomplete factor, which is the case with the improved incomplete Cholesky factorization technique [6].

3.2 Incomplete Factor of the Splitting Preconditioner

In the interior point methods that use iterative approaches, specifically, the conjugate gradient method, we can use the splitting preconditioner MM^T [11]. Let $A = [\mathscr{B} \quad \mathscr{N}]P$, where P is a permutation matrix such that $\mathscr{B}$ is non singular, the matrix $AD^{-1}A^T$, with $A = [\mathscr{B} \quad \mathscr{N}]P$, can be rewrite as follows:

$$AD^{-1}A^T = ([\mathscr{B} \quad \mathscr{N}]P)D^{-1}([\mathscr{B} \quad \mathscr{N}]P)^T = ([\mathscr{B} \quad \mathscr{N}]P)D^{-1}\left(P^T\begin{bmatrix}\mathscr{B}^T\\ \mathscr{N}^T\end{bmatrix}\right) = \tag{3.2}$$

$$[\mathscr{B} \quad \mathscr{N}]\begin{bmatrix}D_{\mathscr{B}}^{-1} & 0\\ 0 & D_{\mathscr{N}}^{-1}\end{bmatrix}\begin{bmatrix}\mathscr{B}^T\\ \mathscr{N}^T\end{bmatrix} = \mathscr{B}D_{\mathscr{B}}^{-1}\mathscr{B}^T + \mathscr{N}D_{\mathscr{N}}^{-1}\mathscr{N}^T.$$

Given the matrix D, where k corresponds to $k-$th iteration of the method, the elements of the diagonal matrix D^k are given by:

$$d_{ii}^k = \frac{z_i^k}{x_i^k}, \qquad 1 \le i \le n, \tag{3.3}$$

where all elements d_{ii} are positive. The preconditioner MM^T is built taking into account the behavior of the matrix D in the final iterations of the interior point method.

As the method approaches to the solution, we can split the primal and dual variables, x_i^k e z_i^k, in two subsets $\mathscr{B}$ and $\mathscr{N}$, such that, $\mathscr{B}$ corresponds to the subset that approaches to $x^* > 0$ and $z^* = 0$, and $\mathscr{N}$ to the subset that approaches to $x^* = 0$ and $z^* > 0$, with x^* and z^* solutions of the linear programming problem. We are considering that the problem is non-degenerate.

Considering $M = \mathscr{B}D_{\mathscr{B}}^{-\frac{1}{2}}$, we obtain the preconditioned matrix

$$\begin{aligned} D_{\mathscr{B}}^{\frac{1}{2}}\mathscr{B}^{-1}(AD^{-1}A^T)\mathscr{B}^{-T}D_{\mathscr{B}}^{\frac{1}{2}} &= I + D_{\mathscr{B}}^{\frac{1}{2}}\mathscr{B}^{-1}\mathscr{N}D_{\mathscr{N}}^{-1}\mathscr{N}^T\mathscr{B}^{-T}D_{\mathscr{B}}^{\frac{1}{2}} \\ &= I + WW^T, \text{with} \quad W = D_{\mathscr{B}}^{\frac{1}{2}}\mathscr{B}^{-1}\mathscr{N}D_{\mathscr{N}}^{-\frac{1}{2}}. \end{aligned} \tag{3.4}$$

The preconditioner is more efficient close to the solution, because (3.4) will be closer to the identity matrix. Applying incomplete Cholesky factorization to $\mathscr{B}D_{\mathscr{B}}^{-1}\mathscr{B}^T$ (splitting preconditioner) we determine the separating factor $\widehat{L}$, which is the preconditioner presented in [8]. Thus, we obtain the preconditioned matrix:

$$\widehat{L}^{-1}AD^{-1}A^T\widehat{L}^{-T} = \widehat{L}^{-1}\mathscr{B}D_{\mathscr{B}}^{-1}\mathscr{B}^T\widehat{L}^{-T} + \widehat{L}^{-1}\mathscr{N}D_{\mathscr{N}}^{-1}\mathscr{N}^T\widehat{L}^{-T}. \tag{3.5}$$

Note that since $\widehat{L}\widehat{L}^T$ is an approximation of $\mathscr{B}D_{\mathscr{B}}^{-1}\mathscr{B}^T$, we have to $\widehat{L}^{-1}\mathscr{B}D_{\mathscr{B}}^{-1}\mathscr{B}^T\widehat{L}^{-T}$ is close to the identity matrix. So the preconditioner works well in the final iterations, similarly to splitting preconditioner.

In the paper [8], the results obtained in the resolution of linear programming problems are presented, in which the splitting factor preconditioner and the splitting preconditioner are compared. The splitting factor preconditioner is used in all iterations in the paper [8], that is, it is not a hybrid approach, to obtain it we apply the incomplete Cholesky factorization in the matrix $\mathscr{B}D_{\mathscr{B}}^{-1}\mathscr{B}^T$ referring to the splitting preconditioner, on what $A = [\mathscr{B} \quad \mathscr{N}]P$ and P is a permutation matrix such that $\mathscr{B}$ is non-singular [11]. The incomplete factor $\widehat{L}$ obtained is called the splitting factor. The preconditioned matrix is $\widehat{L}^{-1}AD^{-1}A^T\widehat{L}^{-T}$. In this paper, we will work with a hybrid approach to preconditioning. In the first iterations, we will use incomplete Cholesky factorization, in the last iterations, we will use the splitting factor. We will compare this approach with that used in the final iterations of the splitting preconditioner. For the phase change we will follow the following criterion used in [3]:

- The initial gap $(x_0^T z_0)$ of the linear programming problem is reduced a factor of 10^6 or
- The number of internal iterations of the Conjugate Gradient Method reaches $\frac{m}{2}$, where m is the dimension of $AD^{-1}A^T$.

Thus, unlike [8], in this paper we work with preconditioner changes.

4 COMPUTATIONAL TESTS

We present the tests experiments for linear programming problems. The implementation was performed in Matlab.

4.1 Computational experiments for linear programming problems

We present the results obtained using in the first iterations the incomplete Cholesky factorization preconditioner and in the final iterations the splitting factor preconditioner, as well as the results obtained using the incomplete Cholesky factorization preconditioner and in the last ones the splitting preconditioner. For preconditioner change criteria we will follow the criterion used in [3]. We emphasize that, despite working with the splitting factor preconditioner as done in [8], in this paper we work in a hybrid approach, using such preconditioners only in the final iterations.

A predictor-corrector interior point method for bounded variables is implemented. In the final iterations, the matrix of the splitting preconditioner ($\mathscr{B}D_{\mathscr{B}}^{-1}\mathscr{B}^T$) is determined and then the incomplete Cholesky factorization of this matrix, which corresponds to the splitting factor, is used as preconditioner. To determine the incomplete Cholesky factor, the Matlab internal function ichol is used. In it, we use drop tolerance with a value equal to 10^{-3}. We also use a diagonal shift with an appropriate α value.

The experiments are performed with an own implementation in Matlab R2017a on Linux, in a computer with 4 GB of memory and an Intel® Core(TM) i3-2367M 1.40GHz processor.

The 39 problems considered in this test can be found at:

- Computational Optimization and Applications (COAP) `http://users.clas.ufl.edu/hager/coap/Pages/matlabpage.html` in the mat format.

Table 1 presents the information on the problems used. In all problems the primal variables are nonnegative and nnz(A) corresponds to the number of non-null elements of the matrix A.

In Table 2 the results obtained are presented, where the time obtained for the solution is given in seconds, It corresponds to the number of iterations of the interior point method, Ch corresponds to which iteration the change of phases occurred and FO indicates whether the solution reached is optimal. When the problem obtained a Not a Number solution, we indicate it with NaN.

The values in red in the column referring to time (Table 2), correspond to the problems in which the method with the splitting factor preconditioner was faster. In the It column, which corresponds to the number of iterations required for convergence, the values in red refer to the problems with fewer iterations in relation to the splitting preconditioner. We denote in bold when the same number of iterations was observed for the two preconditioners.

The method with the new preconditioner and hybrid approach was able to solve all the 39 problems, while the splitting preconditioner with hybrid approach was able to solve 22 of the 39 since for degen2, ken_07, ken_11, ken_13, nug05, nug06, nug07, nug08, osa_07, osa_14, osa_30, osa_60, pds_02, pds_06, pds_10, qap8 and scorpion problems it did not achieve convergence. We highlight that the nug05, nug06, and nug07 problems converged to the wrong values, and we indicate it by * in the table, the nug08, pds_10 and qap8 were finalized after running more than 12 hours and had not yet converged, these problems are indicated by ** in the table. In blue we indicate the time and number of iterations obtained by the method with the splitting factor preconditioner for these seventeen problems.

The new hybrid approach obtained a better time in 11 problems, and for the problems grow7, grow15 and scsd8 the differences are significant.

We reduce the number of iterations in just one problem: czprob. However we can notice from the bold values, that the same number of iterations was obtained for most problems, 18 in total.

Comparing with the paper [8], we note that in the hybrid approach the implementation referring to the splitting factor preconditioner obtained an almost 13 times faster time for the czprob, for the fit2d problem it was 50.8 times, for the ken_07 problem it was 36.8 times faster, for the pds_02 problem it was 161.4 times faster and for the qap8 problem 208 times faster. We also note that in the paper [8], when using the splitting preconditioner in all iterations, only two problems did not converge. Eight more problems than in paper [8] were also considered, they are ken_11, ken_13, osa_07, osa_14, osa_30, osa_60, pds_06 and pds_10. This better performance achieved by the hybrid approach, which we present in this work, was already expected, since the hybrid approach in which we use the splitting preconditioner only in the final iterations has a better performance already known in the literature [3]. As in [8] there was no change of preconditioners, we already

Table 1: General problem data.

Problem	Rows(m)	Columns(n)	nnz(A)
adlittle	56	138	424
agg2	516	758	4740
agg3	516	758	4756
blend	74	114	522
czprob	929	3562	10708
degen2	444	757	4201
fit1d	24	1049	13427
fit2d	25	10524	129042
grow7	140	301	2612
grow15	300	645	5620
israel	174	316	2443
kb2	43	68	313
ken_07	2426	3602	8404
ken_11	14694	21349	49058
ken_13	28632	42659	97246
nug05	210	225	1050
nug06	372	486	2232
nug07	602	931	4214
nug08	912	1632	7296
osa_07	1118	25067	144812
osa_14	2337	54797	317097
osa_30	4350	104374	604488
osa_60	10280	243246	1408073
pds_02	2953	7716	16571
pds_06	9881	29351	63220
pds_10	16558	49932	107605
qap8	912	1632	7296
recipe	91	204	687
sc50a	50	78	160
sc50b	50	78	148
sc105	105	163	340
sc205	205	317	665
scagr7	129	185	465
scorpion	388	466	1534
scsd1	77	760	2388
scsd6	147	1350	4316
scsd8	397	2750	8584
sctap1	300	660	1872
stocfor1	117	165	501

Table 2: Splitting factor and splitting preconditioners.

	Splitting factor				Splitting			
Problem	**Time**	**It**	**Ch**	**FO**	**Time**	**It**	**Ch**	**FO**
adlittle	1.05	**22**	0	O	0.68	22	0	O
agg2	**3.42**	27	16	O	3.44	26	16	O
agg3	3.11	26	16	O	2.59	25	16	O
blend	1.16	**20**	0	O	0.64	20	0	O
czprob	9.90	**58**	23	O	5.91	61	23	O
degen2	**4.52**	**21**	6	O	-	-	-	NaN
fit1d	**0.69**	**30**	0	O	0.88	30	0	O
fit2d	**5.04**	**33**	0	O	5.33	33	0	O
grow7	**0.45**	**16**	10	O	1.92	16	10	O
grow15	**1.32**	**18**	11	O	8.71	18	11	O
israel	**7.92**	38	0	O	8.27	35	0	O
kb2	1.22	**19**	0	O	0.46	19	0	O
ken_07	**2.90**	**16**	11	O	-	-	-	NaN
ken_11	**44.07**	**24**	18	O	-	-	-	NaN
ken_13	**180.55**	**31**	25	O	-	-	-	NaN
nug05	**0.45**	**11**	4	O	-	-	-	*
nug06	**1.39**	**12**	4	O	-	-	-	*
nug07	**5.99**	**16**	4	O	-	-	-	*
nug08	**12.46**	**14**	4	O	-	-	-	**
osa_07	**12.16**	**36**	18	O	-	-	-	NaN
osa_14	**31.79**	**42**	19	O	-	-	-	NaN
osa_30	**67.51**	**49**	24	O	-	-	-	NaN
osa_60	**230.23**	**74**	31	O	-	-	-	NaN
pds_02	**6.25**	**36**	18	O	-	-	-	NaN
pds_06	**91.32**	**57**	29	O	-	-	-	NaN
pds_10	**209.94**	**74**	50	O	-	-	-	**
qap8	**11.24**	**15**	4	O	-	-	-	**
recipe	3.26	**18**	0	O	0.72	18	0	O
sc50a	**0.29**	**15**	1	O	0.42	15	1	O
sc50b	**0.23**	**14**	1	O	0.37	14	1	O
sc105	0.71	**16**	2	O	0.70	16	2	O
sc205	2.15	**20**	5	O	1.51	20	5	O
scagr7	1.27	**23**	2	O	0.60	23	2	O
scorpion	**2.30**	**26**	14	O	-	-	-	NaN
scsd1	**0.45**	**17**	7	O	0.87	17	7	O
scsd6	**1.01**	**19**	7	O	1.97	19	7	O
scsd8	**2.85**	**18**	7	O	5.67	18	7	O
sctap1	3.33	**30**	0	O	2.72	30	0	O
stocfor1	1.65	**23**	0	O	0.96	23	0	O

know that initially the splitting preconditioner and the splitting factor preconditioner would not obtain a desirable performance, since (3.4) would not be close to the identity matrix.

CONCLUSIONS

Given the splitting factor preconditioner previously presented in the literature, in this work we seek to make its approach more robust, for that we used a hybrid approach in which in the first iterations we used the incomplete Cholesky factorization preconditioner, switching in the last iterations to the splitting factor preconditioner. As a comparison to this approach, we also used a hybrid approach, initially considering the incomplete Cholesky factorization preconditioner and, in the final iterations, the splitting preconditioner. For the experiments we used 39 linear programming problems with the experiments being performed in Matlab. The new hybrid approach using the splitting factor preconditioner solves all of them, and the method using the splitting preconditioner in the final iterations solves 22 problems. Comparing the times obtained, the new approach reaches a better time in 11 problems, and in general the differences are not large. It reduces the number of iterations of the interior point method in one problem, and the same number of iterations was reached for the majority of the test problems. Comparing with the results obtained in the paper without a hybrid approach, the splitting factor preconditioner achieved better times, especially in 5 problems, whereas the splitting preconditioner converged only in 22 problems. Thus, comparing the hybrid approach with the non-hybrid one, we conclude that the hybrid approach on in this paper is better, since both the splitting preconditioner and the splitting factor preconditioner have better performance close to the solution. Regarding the comparison of using the splitting and splitting factor preconditioners in the final iterations, we concluded that the splitting factor preconditioner proved to be more efficient in the problems addressed in this work, which are small.

Acknowledgments

This work has been supported by CNPq.

REFERENCES

[1] M.S. Bazaraa, J.J. Jarvis & H.D. Sherali. “Linear programming and network flows”. John Wiley & Sons, Hoboken, NJ (2011).

[2] M. Benzi. Preconditioning techniques for large linear systems: a survey. *Journal of computational Physics*, **182**(2) (2002), 418–477.

[3] S. Bocanegra, F. Campos & A.R. Oliveira. Using a hybrid preconditioner for solving large-scale linear systems arising from interior point methods. *Computational Optimization and Applications*, **36**(2) (2007), 149–164.

[4] L. Cesari. Sulla risoluzione dei sistemi di equazioni lineari per approssimazioni successive. *Atti Accad. Naz. Lincei. Rend. Cl. Sci. Fis. Mat. Nat.*, **25**(6a) (1937), 422–428.

[5] G.H. Golub & C.F. Van Loan. “Matrix computations”. JHU Press, Baltimore, MD, 3 ed. (2012).

[6] M.T. Jones & P.E. Plassmann. An improved incomplete Cholesky factorization. *ACM Transactions on Mathematical Software (TOMS)*, **21**(1) (1995), 5–17.

[7] N. Karmarkar. A new polynomial-time algorithm for linear programming. In “Proceedings of the sixteenth annual ACM symposium on Theory of computing”. ACM, Washington D. C. (1984), p. 302–311.

[8] P.A. Kikuchi & A.R.L. Oliveira. New Preconditioners Applied to Linear Programming and the Compressive Sensing Problems. *SN Oper. Res. Forum*, **1**(36) (2020).

[9] S. Mehrotra. On the implementation of a primal-dual interior point method. *SIAM Journal on optimization*, **2**(4) (1992), 575–601.

[10] J. Meijerink & H.A. Van Der Vorst. An iterative solution method for linear systems of which the coefficient matrix is a symmetric M-matrix. *Mathematics of computation*, **31**(137) (1977), 148–162.

[11] A.R. Oliveira & D.C. Sorensen. A new class of preconditioners for large-scale linear systems from interior point methods for linear programming. *Linear Algebra and its applications*, **394** (2005), 1–24.

[12] M.A.G. Ruggiero & V.L.d.R. Lopes. “Cálculo numérico”. Makron Books do Brasil, São Paulo, SP (1997).

[13] Y. Saad & H.A. Van Der Vorst. Iterative solution of linear systems in the 20th century. *Journal of Computational and Applied Mathematics*, **123**(1) (2000), 1–33.

[14] L.N. Trefethen & D. BAU III. “Numerical linear algebra”, volume 50. Siam, Philadelphia, PA (1997).

[15] S.J. Wright. “Primal-dual interior-point methods”, volume 54. Society for Industrial and Applied Mathematics, Philadelphia, PA (1987).

[16] S.J. Wright. “Primal-dual interior-point methods”. SIAM, Philadelphia, PA (1997).