SIAM J. OPTIM.
Vol. 24, No. 3, pp. 1178–1205

AN EFFICIENT MATRIX SPLITTING METHOD FOR THE SECOND-ORDER CONE COMPLEMENTARITY PROBLEM[*]

LEI-HONG ZHANG[†] AND WEI HONG YANG[‡]

Abstract. Given a symmetric and positive (semi)definite n-by-n matrix M and a vector, in this paper, we consider the matrix splitting method for solving the second-order cone linear complementarity problem (SOCLCP). The matrix splitting method is among the most widely used approaches for large scale and sparse classical linear complementarity problems, and its linear convergence is proved by [Luo and Tseng, *SIAM J. Control Optim.*, 30 (1992), pp. 408–425]. Our first contribution is to prove that, when the general matrix splitting algorithm is applied to SOCLCP with M symmetric and positive definite, it also converges at least linearly. Numerically, our second contribution is to propose a special and efficient matrix splitting algorithm, the block successive overrelaxation method, for solving the SOCLCP. The algorithm makes good use of the underlying geometry of the SOCLCP and each iteration only involves solving triangular linear systems and requires $O(n^2)$ flops; moreover, the algorithm does not destroy the sparse structure of M and is able to exploit the sparsity effectively. Our code BSOR_BN_L based on this method is tested against four other state-of-the-art methods on problems with dense, ill-conditioned symmetric and positive definite matrices, as well as on problems with large scale sparse, symmetric and positive (semi)definite matrices. The numerical results are quite encouraging and BSOR_BN_L exhibits a very good performance in terms of its speed, accuracy, and efficiency in exploiting the sparsity of M.

Key words. second-order cone, linear complementarity problem, matrix splitting method, regular splitting, successive overrelaxation, linear convergence, bisection iteration, Newton's iteration

AMS subject classifications. 90C33, 65K05, 65F99

DOI. 10.1137/13090938X

1. Introduction.

Let $M \in \mathbb{R}^{n\times n}$ be a symmetric matrix and $\mathbf{q} \in \mathbb{R}^n$. In this paper, we are concerned with the solution of the following second-order cone linear complementarity problem (SOCLCP):

$$\text{Find } \mathbf{x} \in \mathcal{K} \text{ such that } M\mathbf{x} + \mathbf{q} \in \mathcal{K} \quad \text{and} \quad \mathbf{x}^\top (M\mathbf{x} + \mathbf{q}) = 0, \tag{1.1}$$

where

$$\mathcal{K} := \mathcal{K}^{n_1} \times \mathcal{K}^{n_2} \times \cdots \times \mathcal{K}^{n_m} \tag{1.2}$$

is the Cartesian product of m *second-order cones* satisfying $\sum_{i=1}^m n_i = n$. A second-order cone (also known as *Lorentz cone*) in $\mathbb{R}^l$ is defined by

$$\mathcal{K}^l := \left\{(x_1, \mathbf{x}_2) \in \mathbb{R} \times \mathbb{R}^{l-1} : \|\mathbf{x}_2\| \le x_1\right\},$$

where $\|\cdot\|$ stands for the Euclidean norm. For simplicity of presentation, here and in what follows, we will denote the problem (1.1) by LCP$(\mathcal{K}, M, \mathbf{q})$ and the set of its solutions by SOL$(\mathcal{K}, M, \mathbf{q})$. The SOCLCP (1.1) can be viewed as a generalization

[*]Received by the editors February 13, 2013; accepted for publication (in revised form) April 10, 2014; published electronically August 7, 2014.
http://www.siam.org/journals/siopt/24-3/90938.html

[†]Corresponding author. Department of Applied Mathematics, Shanghai University of Finance and Economics, 777 Guoding Road, Shanghai 200433, People's Republic of China (longzlh@gmail.com). The work of this author was supported by the National Natural Science Foundation of China NSFC-11101257, NSFC-11371102, and the Basic Academic Discipline Program, the 11th five year plan of 211 Project for Shanghai University of Finance and Economics.

[‡]School of Mathematical Sciences, Fudan University, Shanghai, 200433, People's Republic of China (whyang@fudan.edu.cn). The work of this author was supported by the National Natural Science Foundation of China NSFC-11371102.

of the classical linear complementarity problem (LCP) [2, 4, 7, 9, 22, 23, 29]. Both the SOCLCP and the classical LCP belong to the symmetric cone complementarity problem [12, 13, 14, 19, 25, 34, 35, 36, 43]. There are real-world applications of the SOCLCP (1.1), for example, in robust Nash equilibria [15] and the three-dimensional incremental quasi-static problems with unilateral frictional contact [21]. Moreover, SOCLCPs also arise as systems resulting from the Karush–Kuhn–Tucker conditions of the second-order programming, which contains a wide range of applications in engineering design, control, finance, robust optimization, and combinatorial optimization (see, e.g., [1, 28]). To date, fruitful theoretical results for the SOCLCP have been developed and the reader can refer to, e.g., [6, 10, 15, 16, 17, 32, 34, 44, 42, 45] and the references therein for comprehensive discussions.

Numerically, there have been various methods for solving LCP$(\mathcal{K}, M, \mathbf{q})$. They include, for example, the smoothing Newton method [5, 10, 19], the smoothing-regularization method [16], the semismooth Newton method [32], the interior-point methods [24, 39, 30], the merit-function-based approaches [3, 6, 33], and the matrix splitting method [17]. The former four approaches are of local superlinear convergence but require solving an associated Newton equation in which $O(n^3)$ flops are involved. This might be inefficient when the dimension n gets large; moreover, these methods could not in general exploit the special structure embedded in M and its sparsity as well, and thus, they may perform inefficiently for large scale and sparse problems.

The matrix splitting method (see, e.g., [7]), on the other hand, is of some merit for the large scale and sparse problems. In fact, for the classical LCP, it was demonstrated recently by numerical experiments [31] that the matrix-splitting-type algorithm (the projected Gauss–Seidel with subspace minimization) can be implemented very efficiently for medium and large scale LCPs arising in computer game simulations and American options pricing. Typically, in a matrix splitting algorithm, by choosing a special structure of B and by splitting the matrix $M = B + C$, each iteration solves a related LCP involving the matrix B instead. One attractive feature of the classical LCP is that when B is chosen properly (for example triangular or block triangular [8]), the subproblem in each iteration can be solved rather efficiently or even trivially (see [7]); moreover, as the iteration mainly relies on the matrix-vector manipulation, the sparsity could be exploited. Thus, one only needs to take care of the speed of the outer iteration. It has been shown by Luo and Tseng [29] that when the splitting is regular (see Definition 2.2 in section 2), the convergence is at least linear and, therefore, the matrix splitting method is among the most widely used algorithms for large scale and sparse LCPs.

However, when turning to the SOCLCP (1.1), difficulty arises in the subproblem where a related SOCLCP associated with the matrix B should be solved. Different from the classical LCP, even if B is of a certain special structure, such subproblems are not easy to solve. For example, one widely used choice of B is the block lower triangular (see section 4 for more details), in which the subproblem reduces to solving sequentially m related SOCLCPs, each defined on a single second-order cone. This means that how to solve efficiently an SOCLCP LCP$(\mathcal{K}^l, A, \mathbf{u})$ defined on $\mathcal{K}^l$ is a bottleneck of the matrix splitting method for the SOCLCP (1.1). Efforts have been made in [17] to get around that trouble, where a special block successive over-relaxation (BSOR) method is proposed. In that implementation, the matrix B is chosen as a block lower triangular and each diagonal block matrix $B_{ii} \in \mathbb{R}^{n_i \times n_i}$ is of the form

$$B_{ii} = \begin{bmatrix} b_1 & \mathbf{0}^\top \\ \mathbf{b}_2 & \bar{B}_{ii} \end{bmatrix}, \quad b_1 \in \mathbb{R},\ \mathbf{b}_2 \in \mathbb{R}^{n_i - 1},\ \bar{B}_{ii} \in \mathbb{R}^{(n_i-1)\times(n_i-1)}. \tag{1.3}$$

It is shown that, with such a structure, every iteration in the matrix splitting algorithm can be decoupled into m subproblems, each of which can be transformed into equivalent single variable equations and could be solved by a specially designed Newton iteration [17]. However, we observe that there are two potential limitations: (i) The special structure requirement in the diagonal block B_{ii} may restrict the efficiency of the matrix splitting algorithm, and (ii) solving the ith one of the m decoupled subproblems in general still invokes $O(n_i^3)$ flops. Motivated by these two observations, in this paper, we attempt to improve the efficiency of the matrix splitting algorithm for the SOCLCP.

The improvement on the matrix splitting algorithm for the SOCLCP (1.1) is possible based on the authors' previous work [45], in which an efficient bisection-Newton (BN) iteration is proposed for solving $\mathrm{LCP}(\mathcal{K}^l, A, \mathbf{u})$ defined on a single second-order cone $\mathcal{K}^l$. The algorithm makes good use of the underlying geometry properties of $\mathrm{LCP}(\mathcal{K}^l, A, \mathbf{u})$ when the matrix A has the *globally uniquely solvable* (GUS) property [44]. Roughly speaking, the geometry property of $\mathrm{LCP}(\mathcal{K}^l, A, \mathbf{u})$ permits us to equivalently transform $\mathrm{LCP}(\mathcal{K}^l, A, \mathbf{u})$ into nonlinear equations with respect to a single variable s and guides the adjustment of s. It is worth mentioning that the treatment for our nonlinear equations of the variable s is different from that for the corresponding SOCLCP with the special matrix B_{ii} (1.3) in [17] in that (i) these two treatments are derived from different aspects, (ii) our method in general does not impose any requirement on the structure of the matrix, and (iii) when B_{ii} is chosen properly (for example, lower triangular), solving our nonlinear equations only involves solving triangular linear systems and requires $O(n_i^2)$ flops.

By imbedding our BN iteration into a block successive overrelaxation (BSOR) method, we propose an efficient matrix splitting algorithm for the SOCLCP (1.1). In our implementation, each iteration only involves solving triangular linear systems and only $O(n_i^2)$ flops are required; furthermore, the sparse structure of M can be preserved and exploited. These are appealing features, especially for large scale and sparse SOCLCPs. Our code BSOR_BN_L, based on the combination of BSOR with BN, is tested against other four state-of-the-art methods. The numerical results are quite encouraging and demonstrate that BSOR_BN_L has a very good performance in terms of both its speed and accuracy of the solution.

Another contribution of this paper is to prove that, if M is symmetric and positive definite, the general matrix splitting algorithm with regular splitting converges at least linearly, even when the subproblems are solved inexactly. This result generalizes the well-known linear convergence of the matrix splitting method for the classical LCP of Luo and Tseng [29]. As our method BSOR_BN_L has both efficient implementation and global and linear convergence, it appears to be rather attractive and highly competitive to the SOCLCP (1.1).

We organize this paper in the following way. In the next section, we describe the general matrix splitting algorithm. In section 3, we will first show the convergence when M is either symmetric and positive definite, or simply positive semidefinite; furthermore, we will prove the linear convergence rate of the general matrix splitting algorithm when applied to the symmetric and positive definite case. In section 4, we propose the special matrix splitting algorithm, the BSOR method, and discuss in detail how to solve subproblems in an efficient way, where the BN iteration [45] is introduced and imbedded efficiently in BSOR. Our encouraging numerical experiments, including the comparison with four other methods, are presented in section 5, and finally we draw some remarks and conclude the paper in section 6.

Notation. Throughout the paper, all vectors are column vectors and are typeset in bold, and for convenience, we will choose $\langle \mathbf{x}, \mathbf{y} \rangle$ or $\mathbf{x}^\top \mathbf{y}$ to denote the inner product of vectors $\mathbf{x}, \mathbf{y} \in \mathbb{R}^n$. When $\mathbf{x} \in \mathbb{R}^n$, we distinguish the ith subvector $\mathbf{x}_{[i]} \in \mathbb{R}^{n_i}$ from the ith element $x_i \in \mathbb{R}$, and thereby

$$\mathbf{x} = (\mathbf{x}_{[1]}^\top, \dots, \mathbf{x}_{[m]}^\top)^\top.$$

For a matrix $A \in \mathbb{R}^{n \times m}$, $A^\top$ denotes its transpose, and $\mathcal{R}(A) := \{\mathbf{x} \in \mathbb{R}^n | \mathbf{x} = A\mathbf{y}$ for some $\mathbf{y} \in \mathbb{R}^m\}$ and $\mathrm{Ker}(A) := \{\mathbf{y} \in \mathbb{R}^m | A\mathbf{y} = \mathbf{0}\}$ stand for the range and the kernel of A, respectively. Thus $\mathcal{R}(A)^\perp = \mathrm{Ker}(A^\top)$, where $\mathcal{R}(A)^\perp$ denotes the orthogonal complement of $\mathcal{R}(A)$. As usual, the norm of A is defined by $\|A\| := \max_{\|\mathbf{x}\|=1} \|A\mathbf{x}\|$, and I_n represents the identity matrix in $\mathbb{R}^{n \times n}$; in addition, we define a special matrix J_n by

$$J_n := \mathrm{diag}\{1, -I_{n-1}\} \in \mathbb{R}^{n \times n}. \tag{1.4}$$

For a set $C \subset \mathbb{R}^n$, we denote the boundary and the interior of C by $\mathrm{bd}(C)$ and $\mathrm{int}(C)$, respectively, and thus

$$\mathrm{bd}(\mathcal{K}^l) = \{(x_1, \mathbf{x}_2) \in \mathbb{R} \times \mathbb{R}^{l-1} : \|\mathbf{x}_2\| = x_1\}. \tag{1.5}$$

If $\mathbf{x} \in C$, then the *normal cone* (see, e.g., [18, Definition 5.2.3]) of C at $\mathbf{x}$ is defined by

$$N_{\mathcal{C}}(\mathbf{x}) := \left\{\mathbf{z} | \mathbf{z}^\top (\mathbf{y} - \mathbf{x}) \le 0 \ \forall \mathbf{y} \in C\right\},$$

and therefore, if $\mathbf{y} \in \mathrm{int}(C)$, the relation

$$\mathbf{z}^\top (\mathbf{y} - \mathbf{x}) < 0 \quad \text{for all nonzero } \mathbf{z} \in N_C(\mathbf{x})$$

is evident.

2. The general matrix splitting algorithm.

In this section, we will describe the basic framework of the matrix splitting algorithm by assuming, for the moment, that the splitting (B, C) of

$$M = B + C \tag{2.1}$$

is already given. The question of how to choose the specific (B, C) of a given matrix M for efficient implementation of the matrix splitting algorithm will be discussed in detail in section 4.

When the splitting (B, C) of M is given, the problem $\mathrm{LCP}(\mathcal{K}, M, \mathbf{q})$ (1.1) can be equivalently stated as finding $\mathbf{x} \in \mathcal{K}$ such that

$$B\mathbf{x} + (C\mathbf{x} + \mathbf{q}) \in \mathcal{K} \text{ and } \langle \mathbf{x}, B\mathbf{x} + (C\mathbf{x} + \mathbf{q}) \rangle = 0. \tag{2.2}$$

Let $\mathbf{x}^r$ be the current iteration. The iteration philosophy of the matrix splitting method is to use the current approximation $\mathbf{x}^r$ as the value of $\mathbf{x}$ in the term $C\mathbf{x}$, and then solve the next iteration $\mathbf{x}^{r+1}$ from (2.2). That is,

$$\mathbf{x}^{r+1} \in \mathrm{SOL}(\mathcal{K}, B, C\mathbf{x}^r + \mathbf{q}). \tag{2.3}$$

By introducing the *projection* $\Pi_{\mathcal{K}}(\mathbf{x})$ of a point $\mathbf{x} \in \mathbb{R}^n$ to the nearest point in $\mathcal{K}$, i.e.,

$$\Pi_{\mathcal{K}}(\mathbf{x}) := \arg \min_{\mathbf{y} \in \mathcal{K}} \|\mathbf{x} - \mathbf{y}\|,$$

it is known [9, Proposition 1.5.8] (see also [10]) that the solution $\mathbf{x}^{r+1}$ of (2.3) can be equivalently stated as

$$\mathbf{x}^{r+1} = \Pi_{\mathcal{K}}(\mathbf{x}^{r+1} - B\mathbf{x}^{r+1} - C\mathbf{x}^r - \mathbf{q}), \tag{2.4}$$

and similarly

$$\mathbf{x} \in \mathrm{SOL}(\mathcal{K}, M, \mathbf{q}) \Longleftrightarrow \mathbf{x} = \Pi_{\mathcal{K}}(\mathbf{x} - M\mathbf{x} - \mathbf{q}).$$

In general, the solution to (2.4) can only be achieved by means of iteration where rounding errors are unavoidable; therefore, analogous to the matrix splitting method for the classical LCP [29], we allow inexact computation for (2.4), which is realized by introducing a term $\mathbf{h}^r$ to (2.4) that is controlled by the gap $\|\mathbf{x}^{r+1} - \mathbf{x}^r\|$. To be more precise, we require that the approximation solution $\mathbf{x}^{r+1}$ satisfy the condition

$$\mathbf{x}^{r+1} = \Pi_{\mathcal{K}}(\mathbf{x}^{r+1} - B\mathbf{x}^{r+1} - C\mathbf{x}^r - \mathbf{q} + \mathbf{h}^r), \tag{2.5}$$

where

$$\|\mathbf{h}^r\| \le \zeta \|\mathbf{x}^{r+1} - \mathbf{x}^r\| \tag{2.6}$$

for some nonnegative parameter $\zeta \ge 0$ depending on the splitting (B, C). As has been pointed out in [29], such an approximation $\mathbf{x}^{r+1}$ is achievable for any iterative method that converges to the solution to (2.4). In particular, suppose the current iteration $\mathbf{x}^r \notin \mathrm{SOL}(\mathcal{K}, M, \mathbf{q})$ and $\{\mathbf{y}^k\}$ is the sequence converging to the solution $\mathbf{x}^{r+1}$ to (2.4), then we stop at any point $\mathbf{y}^k$ satisfying

$$\|(I_n - B)(\mathbf{y}^k - F(\mathbf{y}^k))\| \le \zeta \|\mathbf{x}^r - F(\mathbf{y}^k)\|, \tag{2.7}$$

where

$$F(\mathbf{y}) := \Pi_{\mathcal{K}}(\mathbf{y} - B\mathbf{y} - C\mathbf{x}^r - \mathbf{q}).$$

Condition (2.7) can be fulfilled because as k approaches infinity, $\mathbf{y}^k \to \mathbf{x}^{r+1} = F(\mathbf{x}^{r+1}) \ne \mathbf{x}^r$. Furthermore, it can be verified that by setting

$$\mathbf{x}^{r+1} := F(\mathbf{y}^k) \quad \text{and} \quad \mathbf{h}^r := (I_n - B)(\mathbf{y}^k - F(\mathbf{y}^k)),$$

$\mathbf{x}^{r+1}$ and $\mathbf{h}^r$ satisfy conditions (2.5) and (2.6).

Overall, the basic framework of the general matrix splitting algorithm can be summarized as Algorithm 1.

To specify the stopping criterion in `Step 3`, by considering the SOCLCP (1.1), a reasonable choice is to terminate the iteration whenever

$$\begin{aligned} \chi := & \sum_{i=1}^{m} \max\{\|\mathbf{x}_{[i]}(2:n_i)\| - \mathbf{x}_{[i]}(1), 0\} \\ & + \sum_{i=1}^{m} \max\{\|\mathbf{g}_{[i]}(2:n_i)\| - \mathbf{g}_{[i]}(1), 0\} + |\mathbf{x}^\top \mathbf{g}| \le \varepsilon. \end{aligned} \tag{2.8}$$

Here, we use the stylized version of the MATLAB language to denote by $\mathbf{x}_{[i]}(2:n_i)$ the subvector of $\mathbf{x}_{[i]}$, with elements from the second to the n_ith, and also define

$$\mathbf{g} := M\mathbf{x} + \mathbf{q}.$$

It is clear that the first and second terms in the left-hand side of (2.8) measure

Algorithm 1. The general matrix splitting algorithm.

```
Given a symmetric matrix $M \in \mathbb{R}^{n\times n}$ and $\mathbf{q} \in \mathbb{R}^n$, this algorithm solves the
SOCLCP (1.1) with the splitting $(B, C)$ of $M$.
Step 1. Select any $\mathbf{x}^0 \in \mathbb{R}^n$, and set $r = 0$.
Step 2. Compute a new iteration $\mathbf{x}^{r+1} \in \mathcal{K}$ satisfying
            $\mathbf{x}^{r+1} = \Pi_{\mathcal{K}}(\mathbf{x}^{r+1} - B\mathbf{x}^{r+1} - C\mathbf{x}^r - \mathbf{q} + \mathbf{h}^r)$,
        where $\mathbf{h}^r$ satisfies
            $\|\mathbf{h}^r\| \le \zeta\|\mathbf{x}^{r+1} - \mathbf{x}^r\|$,
        and $\zeta \ge 0$ is a parameter depending on $(B, C)$.
Step 3. Stop if the stopping rule is met; otherwise, $r := r + 1$ and goto Step 2.
```

the feasibility of $\mathbf{x}$ and $\mathbf{g}$, respectively, whereas the last term measures the complementarity condition $\mathbf{x}^\top(M\mathbf{x} + \mathbf{q}) = 0$. In our numerical testing, such a stopping rule is adopted with a recommended range $\varepsilon \in [10^{-7}, 10^{-4}]$ for ε. The relative accuracy χ_r is then defined by

$$\chi_r := \frac{\chi}{1 + \|\mathbf{q}\|_1 + \|M\|_1}. \tag{2.9}$$

For the choice of a proper splitting (B, C), we should ensure both the well-definedness of the subproblem (2.5) and the convergence of Algorithm 1 as well. For the former, a sufficient condition is that B is a $\mathcal{K}$-Q-matrix [17] whose definition is given as follows.

Definition 2.1. *If* LCP$(\mathcal{K}, B, \mathbf{q})$ *has a solution for any* $\mathbf{q} \in \mathbb{R}^n$, *then* B *is called a* $\mathcal{K}$-Q*-matrix. Moreover, if* B *is a* $\mathcal{K}$-Q*-matrix, then* (B, C) *is called a* $\mathcal{K}$-Q*-splitting.*

It has been well known that if M is positive definite (not necessarily symmetric), then it must be a $\mathcal{K}$-Q-matrix; in fact, when M is positive definite, then for any $\mathbf{q} \in \mathbb{R}^n$, LCP$(\mathcal{K}, M, \mathbf{q})$ admits a unique solution. For the convergence of Algorithm 1, on the other hand, one of the most general and widely used conditions of the splitting (B, C) is the so-called regular splitting defined as follows.

Definition 2.2. *Suppose that* (B, C) *is a splitting of* M. *If* $B - C$ *is positive (semi)definite, then the splitting* (B, C) *is said to be (weakly) regular.*

If (B, C) is a regular splitting of M, then the parameter ζ involved in Step 2 of Algorithm 1 is recommended as [29]

$$\zeta = \gamma/2 - \epsilon, \tag{2.10}$$

where $\epsilon \in (0, \gamma/2]$ and γ denotes the smallest eigenvalue of the symmetric part of $B - C$. In next section, we will establish the convergence of Algorithm 1 both for the symmetric and positive definite matrix, and for the positive semidefinite matrix as well; furthermore, we will also prove the linear convergence when M is symmetric and positive definite and the splitting (B, C) is regular.

3. Convergence analysis.

3.1. Preliminary properties.

To develop the convergence of the matrix splitting algorithm, in this subsection, we shall first provide some preparatory results. To begin with, we point out that the projection $\Pi_{\mathcal{K}}(\cdot)$ has several simple but useful properties: For example, according to the definition of $\mathcal{K}$, one can readily verify that

$$\Pi_{\mathcal{K}}(\mathbf{x}) = \begin{bmatrix} \Pi_{\mathcal{K}^{n_1}}(\mathbf{x}_{[1]}) \\ \vdots \\ \Pi_{\mathcal{K}^{n_m}}(\mathbf{x}_{[m]}) \end{bmatrix} \quad \text{and} \tag{3.1}$$

$$\langle \mathbf{z} - \Pi_{\mathcal{K}}(\mathbf{x}), \mathbf{x} - \Pi_{\mathcal{K}}(\mathbf{x}) \rangle \leq 0 \quad \forall \mathbf{z} \in \mathcal{K}. \tag{3.2}$$

By [5, Proposition 4.3] (see also [10]), $\Pi_{\mathcal{K}^n}(\mathbf{x})$ is also a *strongly semismooth* (for the definition, see [34, section 3]) function of $\mathbf{x}$. Moreover, we can expand the projection $\Pi_{\mathcal{K}^n}(\mathbf{x}+\mathbf{d})$ for any $\mathbf{d} \in \mathbb{R}^n$ up to the second-order term. To be more precise, let $\bar{\mathbf{x}} := \Pi_{\mathcal{K}^n}(\mathbf{x})$. The *critical cone* [34, p. 48], associated with the projection $\Pi_{\mathcal{K}^n}(\mathbf{x})$, of $\mathcal{K}^n$ at $\bar{\mathbf{x}}$ is defined as $\mathcal{C}_{\mathcal{K}^n}(\bar{\mathbf{x}}) := \mathcal{T}_{\mathcal{K}^n}(\bar{\mathbf{x}}) \cap \mathcal{R}(\mathbf{x}-\bar{\mathbf{x}})^{\perp}$, where $\mathcal{T}_{\mathcal{K}^n}(\bar{\mathbf{x}})$ is the tangent cone [9, p. 15] of $\mathcal{K}^n$ at $\bar{\mathbf{x}}$ and $\mathcal{R}(\mathbf{x}-\bar{\mathbf{x}})^{\perp}$ is the subspace in $\mathbb{R}^n$ that is orthogonal to $\mathbf{x}-\bar{\mathbf{x}}$. In particular, if $\mathbf{0} \neq \mathbf{x} \in -\mathrm{bd}(\mathcal{K}^n)$, then $\Pi_{\mathcal{K}^n}(\mathbf{x}) = \mathbf{0}$ and the critical cone associated with $\Pi_{\mathcal{K}^n}(\mathbf{x})$ is simply [44]

$$\mathcal{C}_{\mathcal{K}^n}(\mathbf{0}) = \{-tJ_n\mathbf{x} \mid t \geq 0\}. \tag{3.3}$$

Using the critical cone, it is known [34, Proposition 13] that, for any $\mathbf{x}, \mathbf{d} \in \mathbb{R}^n$, we have

$$\Pi'_{\mathcal{K}^n}(\mathbf{x};\mathbf{d}) = \Pi_{\mathcal{C}_{\mathcal{K}^n}(\bar{\mathbf{x}})}(\mathbf{d}), \tag{3.4}$$

where $\Pi'_{\mathcal{K}^n}(\mathbf{x};\mathbf{d})$ is the directional derivative of $\Pi_{\mathcal{K}^n}(\mathbf{x})$ at $\mathbf{x}$ along the direction $\mathbf{d}$. This together with [34, equation (5)] implies that

$$\begin{aligned}\Pi_{\mathcal{K}^n}(\mathbf{x}+\mathbf{d}) &= \Pi_{\mathcal{K}^n}(\mathbf{x}) + \Pi'_{\mathcal{K}^n}(\mathbf{x};\mathbf{d}) + O(\|\mathbf{d}\|^2) \\ &= \bar{\mathbf{x}} + \Pi_{\mathcal{C}_{\mathcal{K}^n}(\bar{\mathbf{x}})}(\mathbf{d}) + O(\|\mathbf{d}\|^2), \end{aligned} \tag{3.5}$$

where the second equality follows from (3.4).

Another two useful lemmas for the convergence are provided as follows.

LEMMA 3.1. *Let* $\mathbf{x}, \mathbf{y} \in \mathrm{bd}(\mathcal{K}^n)$. *Then* $\langle J_n\mathbf{x}, \mathbf{y}-\mathbf{x}\rangle \leq \|\mathbf{x}-\mathbf{y}\|^2/2$.

Proof. By $\mathbf{x}, \mathbf{y} \in \mathrm{bd}(\mathcal{K}^n)$ and (1.5), we have

$$\langle J_n\mathbf{x}, \mathbf{y}-\mathbf{x}\rangle = x_1(y_1-x_1) - \mathbf{x}_2^{\top}(\mathbf{y}_2-\mathbf{x}_2) = x_1y_1 - \mathbf{x}_2^{\top}\mathbf{y}_2 \tag{3.6}$$

and

$$\begin{aligned}\|\mathbf{x}-\mathbf{y}\|^2 &= x_1^2 - 2x_1y_1 + y_1^2 + (\|\mathbf{x}_2\|^2 - 2\mathbf{x}_2^{\top}\mathbf{y}_2 + \|\mathbf{y}_2\|^2) \\ &= 2(x_1^2 - x_1y_1 + y_1^2 - \mathbf{x}_2^{\top}\mathbf{y}_2) \geq 2(x_1y_1 - \mathbf{x}_2^{\top}\mathbf{y}_2), \end{aligned} \tag{3.7}$$

where the last inequality is due to $x_1^2 - 2x_1y_1 + y_1^2 \geq 0$. Then the assertion follows from (3.6) and (3.7). ☐

LEMMA 3.2 (see [44]). *The following statements hold:*

(i) *For nonzero vectors* $\mathbf{x}, \mathbf{y} \in \mathcal{K}^n$, $\langle \mathbf{x}, \mathbf{y}\rangle = 0$ *if and only if* $\mathbf{x} \in \mathrm{bd}(\mathcal{K}^n)$, $\mathbf{y} \in \mathrm{bd}(\mathcal{K}^n)$, *and* $\mathbf{y} = \mu J_n\mathbf{x}$ *for some* $\mu > 0$;

(ii) *let* $\mathbf{x} \in \mathcal{K}^n$. *Then* $\langle \mathbf{x}, \mathbf{y}\rangle > 0$ *for all nonzero vector* $\mathbf{y} \in \mathcal{K}^n$ *if and only if* $\mathbf{x} \in \mathrm{int}(\mathcal{K}^n)$.

3.2. Convergence. In this subsection, we will establish the convergence of Algorithm 1. For this purpose, we first observe that when M is symmetric, then it is clear, by the KKT condition, that the SOCLCP (1.1) is equivalent to the following quadratic second-order cone programming

$$\min_{\mathbf{x}\in\mathcal{K}} f(\mathbf{x}) := \frac{1}{2}\langle \mathbf{x}, M\mathbf{x}\rangle + \langle \mathbf{q}, \mathbf{x}\rangle. \tag{3.8}$$

Moreover, if M is additionally positive definite, then for any $\mathbf{q} \in \mathbb{R}^n$, (1.1) has a unique solution $\mathbf{x}^*$. When $\zeta = 0$ in `Step 2` (i.e., exact computation of the subproblem (2.4)), the convergence of Algorithm 1 is already proved in [17]. In the following theorem, we further generalize that result by permitting $\zeta > 0$ and establish the convergence of Algorithm 1 for the symmetric and positive definite matrix M with any regular splitting (B, C).

THEOREM 3.3. *Let M be a symmetric positive definite matrix and $\mathbf{q} \in \mathbb{R}^n$ be an arbitrary vector. Let (B, C) be a regular splitting of M. Then for any initial point $\mathbf{x}^0 \in \mathcal{K}$, the points $\{\mathbf{x}^r\}$ generated by Algorithm* 1 *with ζ given by* (2.10) *converge to $\mathbf{x}^*$, where $\mathbf{x}^*$ is the unique solution of* (1.1).

Proof. We first prove there exists $\kappa_1 > 0$ such that

$$\|\mathbf{x}^{r-1} - \mathbf{x}^r\|^2 \le \kappa_1(f(\mathbf{x}^{r-1}) - f(\mathbf{x}^r)) \quad \forall r \ge 1, \tag{3.9}$$

where $f(\mathbf{x})$ is given by (3.8). Fix any $r \ge 1$. According to (2.5) and (3.2), we obtain

$$\langle \mathbf{x}^r - \mathbf{x}^{r-1}, B\mathbf{x}^r + C\mathbf{x}^{r-1} + \mathbf{q} - \mathbf{h}^{r-1}\rangle \le 0.$$

Also, from $M = B + C$ and the definition of $f(\mathbf{x})$, we have that

$$\begin{aligned} &f(\mathbf{x}^r) - f(\mathbf{x}^{r-1}) \\ &\quad = \langle \mathbf{x}^r - \mathbf{x}^{r-1}, B\mathbf{x}^r + C\mathbf{x}^{r-1} + \mathbf{q}\rangle + \langle \mathbf{x}^r - \mathbf{x}^{r-1}, (C - B)(\mathbf{x}^r - \mathbf{x}^{r-1})\rangle/2. \end{aligned}$$

Combining the above two relations and (2.10), we have

$$\begin{aligned} f(\mathbf{x}^r) - f(\mathbf{x}^{r-1}) &\le \langle \mathbf{x}^r - \mathbf{x}^{r-1}, (C - B)(\mathbf{x}^r - \mathbf{x}^{r-1}) + 2\mathbf{h}^{r-1}\rangle/2 \\ &\le -\epsilon\|\mathbf{x}^r - \mathbf{x}^{r-1}\|^2. \end{aligned}$$

Hence (3.9) holds with $\kappa_1 = 1/\epsilon$. Thus we know that the sequence $\{f(\mathbf{x}^r)\}$ is strictly decreasing. Since M is positive definite, the sequence $\{\mathbf{x}^r\}$ generated by the algorithm is bounded. Let $\bar{\mathbf{x}}$ be an arbitrary accumulation point of the sequence $\{\mathbf{x}^r\}$ and $\{\mathbf{x}^{r_i}\}$ be a subsequence of $\{\mathbf{x}^r\}$ converging to $\bar{\mathbf{x}}$. Since $\{f(\mathbf{x}^r)\}$ is convergent, by (3.9), $\{\mathbf{x}^{r-1} - \mathbf{x}^r\}$ converges to $\mathbf{0}$. Hence, the sequence $\{\mathbf{x}^{r_i-1}\}$ also converges to $\bar{\mathbf{x}}$. Because $\mathbf{x}^{r_i}$ satisfies

$$\mathbf{x}^{r_i} \in \mathcal{K}, \quad B\mathbf{x}^{r_i} + C\mathbf{x}^{r_i-1} + \mathbf{q} - \mathbf{h}^{r_i-1} \in \mathcal{K}, \langle \mathbf{x}^{r_i}, B\mathbf{x}^{r_i} + C\mathbf{x}^{r_i-1} + \mathbf{q} - \mathbf{h}^{r_i-1}\rangle = 0,$$

passing to the limit shows that $\bar{\mathbf{x}}$ is a solution of (1.1). On the other hand, since M is positive definite, (1.1) has a unique solution $\mathbf{x}^*$. Thus, $\bar{\mathbf{x}} = \mathbf{x}^*$, which implies that any accumulation point of an arbitrary subsequence of $\{\mathbf{x}^r\}$ is $\mathbf{x}^*$, and consequently $\mathbf{x}^r \to \mathbf{x}^*$. ∎

Before concluding this subsection, we briefly mention that, when M is only positive semidefinite (not necessarily symmetric) but $\mathrm{SOL}(\mathcal{K}, M, \mathbf{q}) \ne \emptyset$, Theorem 5.6.1

in [7] provides one type of splitting (B, C) for which the matrix splitting algorithm (Algorithm 1) with $\zeta = 0$ (i.e., exact computation of the subproblem (2.4)) in `Step 2` converges to a solution of $\mathrm{LCP}(\mathcal{K}, M, \mathbf{q})$. Though the result [7, Theorem 5.6.1] is a customized result for the classical LCP, it can be readily verified that the conclusion also holds true for our SOCLCP (1.1). For the sake of completeness, we state this result below.

THEOREM 3.4 (see [7, Theorem 5.6.1]). *Let M be a positive semidefinite matrix and* $\mathrm{SOL}(\mathcal{K}, M, \mathbf{q}) \neq \emptyset$. *Let (B, C) be a splitting of M such that $B - M$ is symmetric positive definite. Then, the uniquely defined sequence of vectors $\{\mathbf{x}^r\}$ produced by Algorithm 1 with $\zeta = 0$ converges to some solution of the* $\mathrm{LCP}(\mathcal{K}, M, \mathbf{q})$.

Another effective treatment for the positive semidefinite case is the so-called *regularization*. The idea is to add a term νI_n $(\nu > 0)$ to M so that $M + \nu I_n$ is positive definite and, thereby, $\mathrm{LCP}(\mathcal{K}, M + \nu I_n, \mathbf{q})$ admits a unique solution, say $\mathbf{x}(\nu) \in \mathrm{SOL}(\mathcal{K}, M + \nu I_n, \mathbf{q})$. Upon taking a sequence of positive scales $\{\nu\} \to 0$, it generates a sequence $\{\mathbf{x}(\nu)\}$. The following theorem is again originally established for the classical LCP, but is also valid for the SOCLCP (1.1), whose proof is almost the same as that for [7, Theorem 5.6.2], and therefore is omitted here.

THEOREM 3.5. *Let M be a positive semidefinite matrix and* $\mathrm{SOL}(\mathcal{K}, M, \mathbf{q}) \neq \emptyset$. *Let $\nu > 0$. Then, the solution $\mathbf{x}(\nu) \in \mathrm{SOL}(\mathcal{K}, M + \nu I_n, \mathbf{q})$ converges to the least l_2-norm solution of* $\mathrm{LCP}(\mathcal{K}, M, \mathbf{q})$, *as $\nu \to 0$.*

3.3. Linear convergence. Now, we are in a position to establish the linear convergence of Algorithm 1. Our proof borrows the idea from Luo and Tseng [29] for the classical LCP. Throughout this subsection, we assume that M is symmetric and positive definite, and the parameter ζ in `Step 2` of Algorithm 1 is given by (2.10). It has been pointed out that, in this case, $\mathrm{SOL}(\mathcal{K}, M, \mathbf{q})$ admits a unique solution $\mathbf{x}^*$. That is, (1.1) holds for $\mathbf{x} = \mathbf{x}^*$, which implies that

$$\langle \mathbf{x}^*_{[i]}, (M\mathbf{x}^* + \mathbf{q})_{[i]} \rangle = 0 \quad \forall\, i = 1, \ldots, m. \tag{3.10}$$

Based on this observation, we can define the following four index sets:

$$\begin{aligned}
\Xi_1 &= \{i \in \{1, \ldots, m\} \mid \mathbf{x}^*_{[i]} \neq 0 \text{ and } \mathbf{x}^*_{[i]} \in \mathrm{bd}(\mathcal{K}^{n_i})\},\\
\Xi_2 &= \{i \in \{1, \ldots, m\} \mid \mathbf{x}^*_{[i]} = 0 \text{ and } (M\mathbf{x}^* + \mathbf{q})_{[i]} \in \mathrm{int}(\mathcal{K}^{n_i})\},\\
\Xi_3 &= \{i \in \{1, \ldots, m\} \mid \mathbf{x}^*_{[i]} = 0 \text{ and } (M\mathbf{x}^* + \mathbf{q})_{[i]} \in \mathrm{bd}(\mathcal{K}^{n_i})\},\\
\Xi_4 &= \{i \in \{1, \ldots, m\} \mid \mathbf{x}^*_{[i]} \in \mathrm{int}(\mathcal{K}^{n_i})\},
\end{aligned}$$

where $\mathbf{x}^*_{[i]} \in \mathbb{R}^{n_i}$ is the ith subvector of $\mathbf{x}^*$ partitioned according to (1.2). Note that for any $i \in \Xi_1$, by Lemma 3.2(i) and (3.10), we know

$$(M\mathbf{x}^* + \mathbf{q})_{[i]} = s_i J_{n_i} \mathbf{x}^*_{[i]} \tag{3.11}$$

for some $s_i \geq 0$, whereas for any $i \in \Xi_4$, (3.10) yields $(M\mathbf{x}^* + \mathbf{q})_{[i]} = \mathbf{0}$. Moreover, it is easy to check that these four index sets are mutually exclusive and

$$\Xi_1 \cup \Xi_2 \cup \Xi_3 \cup \Xi_4 = \{1, \ldots, n\}.$$

Table 1 gives a clear picture of these four index sets.

The proof of the linear convergence for Algorithm 1 requires some results on error bounds. One important error bound is to estimate the distance between any $\mathbf{x} \in \mathbb{R}^n$

Table 1
Four mutually exclusive index sets for $\mathbf{x}^*$.

Index set	$i \in \Xi_1$	$i \in \Xi_2$	$i \in \Xi_3$	$i \in \Xi_4$
$\mathbf{x}^*_{[i]}$	$\mathrm{bd}(\mathcal{K}^{n_i}) \setminus \{\mathbf{0}\}$	$\mathbf{0}$	$\mathbf{0}$	$\mathrm{int}(\mathcal{K}^{n_i})$
$(M\mathbf{x}^* + \mathbf{q})_{[i]}$	$\mathrm{bd}(\mathcal{K}^{n_i})$	$\mathrm{int}(\mathcal{K}^{n_i})$	$\mathrm{bd}(\mathcal{K}^{n_i})$	$\mathbf{0}$

and the solution $\mathbf{x}^*$. For this purpose, we define

$$F^{\mathrm{nat}}_{\mathcal{K}}(\mathbf{x}) := \mathbf{x} - \Pi_{\mathcal{K}}(\mathbf{x} - M\mathbf{x} - \mathbf{q})$$

as the *natural map* [9, p. 83] associated with complementarity problem (1.1). By [9, Proposition 1.5.8], $\mathbf{x}^* \in \mathrm{SOL}(M, \mathcal{K}, \mathbf{q})$ if and only if $F^{\mathrm{nat}}_{\mathcal{K}}(\mathbf{x}^*) = \mathbf{0}$. Since M is positive definite, by [9, Theorem 2.3.3], there exists a $\tau > 0$ such that

$$\|\mathbf{x} - \mathbf{x}^*\| \le \tau \|F^{\mathrm{nat}}_{\mathcal{K}}(\mathbf{x})\| \quad \forall \mathbf{x} \in \mathbb{R}^n. \tag{3.12}$$

With this error bound (3.12), we can further estimate the distance between the iteration $\mathbf{x}^r$ of Algorithm 1 and $\mathbf{x}^*$ as the following lemma shows.

Lemma 3.6. *Let* $\{\mathbf{x}^r\}$ *be the sequence generated by the matrix splitting algorithm. There exists an* $\alpha > 0$ *such that*

$$\|\mathbf{x}^r - \mathbf{x}^*\| \le \alpha \|\mathbf{x}^r - \mathbf{x}^{r-1}\|. \tag{3.13}$$

Proof. By (3.12), (2.6), and the nonexpansive property of the projection operator $\Pi_{\mathcal{K}}$, we have

$$\begin{aligned}
&\|\mathbf{x}^r - \mathbf{x}^*\| \\
&\quad \le \tau \|\mathbf{x}^r - \Pi_{\mathcal{K}}(\mathbf{x}^r - M\mathbf{x}^r - \mathbf{q})\| \\
&\quad = \tau \|\Pi_{\mathcal{K}}(\mathbf{x}^r - B\mathbf{x}^r - C\mathbf{x}^{r-1} - \mathbf{q} + \mathbf{h}^{r-1}) - \Pi_{\mathcal{K}}(\mathbf{x}^r - M\mathbf{x}^r - \mathbf{q})\| \\
&\quad \le \tau \|C(\mathbf{x}^r - \mathbf{x}^{r-1}) + \mathbf{h}^{r-1}\| \\
&\quad \le \tau (\|C\| + \zeta) \|\mathbf{x}^r - \mathbf{x}^{r-1}\|,
\end{aligned}$$

and the proof is completed. ◻

The next lemma is the key for the proof of the linear convergence of the matrix splitting method.

Lemma 3.7. *Let* $\{\mathbf{x}^r\}$ *be the sequence generated by the matrix splitting algorithm (Algorithm* 1*). There exist a* $\rho > 0$ *and a positive number* r_1 *such that for all* $r > r_1$

$$\langle M\mathbf{x}^* + \mathbf{q}, \mathbf{x}^r - \mathbf{x}^* \rangle \le \rho \|\mathbf{x}^r - \mathbf{x}^{r-1}\|^2. \tag{3.14}$$

Proof. By (3.10) and Lemma 3.2(ii), for any $i \in \Xi_4$, $(M\mathbf{x}^* + \mathbf{q})_{[i]} = 0$. Thus,

$$\langle M\mathbf{x}^* + \mathbf{q}, \mathbf{x}^r - \mathbf{x}^* \rangle = \sum_{i \in \Xi_1 \cup \Xi_2 \cup \Xi_3} \langle (M\mathbf{x}^* + \mathbf{q})_{[i]}, (\mathbf{x}^r - \mathbf{x}^*)_{[i]} \rangle. \tag{3.15}$$

For notational convenience, let

$$\begin{aligned}
\mathbf{y}^r &= B\mathbf{x}^r + C\mathbf{x}^{r-1} + \mathbf{q} - \mathbf{h}^{r-1} \quad \text{and} \\
\mathbf{v}^r &= \mathbf{x}^r - \mathbf{y}^r - (\mathbf{x}^* - M\mathbf{x}^* - \mathbf{q}).
\end{aligned}$$

According to the iteration scheme (2.5) in Algorithm 1, it follows that

$$\mathbf{x}^r = \Pi_{\mathcal{K}}(\mathbf{x}^r - \mathbf{y}^r).$$

By Theorem 3.3, we have $\mathbf{x}^r \to \mathbf{x}^*$. This and (2.6) imply that $\mathbf{y}^r \to M\mathbf{x}^* + \mathbf{q}$ and

$$\begin{aligned} \mathbf{v}^r &= (I - M)(\mathbf{x}^r - \mathbf{x}^*) + O(\|\mathbf{x}^r - \mathbf{x}^{r-1}\|) \\ &= O(\|\mathbf{x}^r - \mathbf{x}^{r-1}\|), \end{aligned} \tag{3.16}$$

where the last equality follows from Lemma 3.6.

For $i \in \Xi_1$, from (3.11), we know

$$(M\mathbf{x}^* + \mathbf{q})_{[i]} = s_i J_{n_i} \mathbf{x}^*_{[i]}$$

for some $s_i \geq 0$, and so

$$\sum_{i\in\Xi_1} \langle (M\mathbf{x}^* + \mathbf{q})_{[i]}, (\mathbf{x}^r - \mathbf{x}^*)_{[i]} \rangle = \sum_{i\in\Xi_1, s_i>0} \langle s_i J_{n_i} \mathbf{x}^*_{[i]}, \mathbf{x}^r_{[i]} - \mathbf{x}^*_{[i]} \rangle. \tag{3.17}$$

Fix $j \in \Xi_1$ satisfying $s_j > 0$. By $\mathbf{0} \neq \mathbf{x}^*_{[j]} \in \mathrm{bd}(\mathcal{K}^{n_j})$, we have

$$(\mathbf{x}^* - M\mathbf{x}^* - \mathbf{q})_{[j]} = (I_{n_j} - s_j J_{n_j})\mathbf{x}^*_{[j]} \notin \mathcal{K}^{n_j}.$$

Since

$$(\mathbf{x}^r - \mathbf{y}^r)_{[j]} \to (\mathbf{x}^* - M\mathbf{x}^* - \mathbf{q})_{[j]},$$

there exists an r_2 such that $(\mathbf{x}^r - \mathbf{y}^r)_{[j]} \notin \mathcal{K}^{n_j}$ for all $r > r_2$. Thus,

$$\mathbf{x}^r_{[j]} = \Pi_{\mathcal{K}^{n_j}}[(\mathbf{x}^r - \mathbf{y}^r)_{[j]}] \in \mathrm{bd}(\mathcal{K}^{n_j}).$$

Then by Lemma 3.1,

$$\langle J_{n_j} \mathbf{x}^*_{[j]}, \mathbf{x}^r_{[j]} - \mathbf{x}^*_{[j]} \rangle \leq \|\mathbf{x}^r_{[j]} - \mathbf{x}^*_{[j]}\|^2/2 \leq \|\mathbf{x}^r - \mathbf{x}^*\|^2/2. \tag{3.18}$$

Because the number of index $j \in \Xi_1$ satisfying $s_j > 0$ is finite, by (3.17) and (3.18), there exist a $\rho_1 > 0$ and an r_3 such that for all $r > r_3$,

$$\sum_{i\in\Xi_1} \langle (M\mathbf{x}^* + \mathbf{q})_{[i]}, (\mathbf{x}^r - \mathbf{x}^*)_{[i]} \rangle \leq \rho_1 \|\mathbf{x}^r - \mathbf{x}^*\|^2 \leq \rho_1 \alpha \|\mathbf{x}^r - \mathbf{x}^{r-1}\|^2, \tag{3.19}$$

where the last inequality follows from (3.13).

Now we prove that for any $i \in \Xi_2$, $\mathbf{x}^r_{[i]} = \mathbf{0}$ for all r sufficiently large, and thereby

$$\sum_{i\in\Xi_2} \langle (M\mathbf{x}^* + \mathbf{q})_{[i]}, (\mathbf{x}^r - \mathbf{x}^*)_{[i]} \rangle = 0. \tag{3.20}$$

Fix any $j \in \Xi_2$. Since $\mathbf{x}^r \to \mathbf{x}^*$ and $\mathbf{x}^*_{[j]} = \mathbf{0}$, we have

$$(\mathbf{x}^r - \mathbf{y}^r)_{[j]} \to -(M\mathbf{x}^* + \mathbf{q})_{[j]} \in -\mathrm{int}(\mathcal{K}^{n_j}).$$

This together with (3.1) and the fact

$$\Pi_{\mathcal{K}^{n_j}}(\mathbf{z}) = \mathbf{0} \quad \forall \mathbf{z} \in -\mathcal{K}^n,$$

shows that for all r sufficiently large,

$$\mathbf{x}^r_{[j]} = [\Pi_{\mathcal{K}}(\mathbf{x}^r - \mathbf{y}^r)]_{[j]} = \Pi_{\mathcal{K}^{n_j}}[(\mathbf{x}^r - \mathbf{y}^r)_{[j]}] = \mathbf{0}.$$

Since the number of index $j \in \Xi_2$ is finite, there exists an r_4 such that (3.20) holds for all $r > r_4$.

For the last case, suppose $j \in \Xi_3$. Since

$$(\mathbf{x}^* - M\mathbf{x}^* - \mathbf{q})_{[j]} = -(M\mathbf{x}^* + \mathbf{q})_{[j]} \in -\mathrm{bd}(\mathcal{K}^{n_j}),$$

implying $\Pi_{\mathcal{K}^{n_j}}[(\mathbf{x}^* - M\mathbf{x}^* - \mathbf{q})_{[j]}] = \mathbf{0}$, the relations (3.4) and (3.5) lead to

$$\begin{aligned} \mathbf{x}^r_{[j]} &= [\Pi_{\mathcal{K}}(\mathbf{x}^r - \mathbf{y}^r)]_{[j]} = \Pi_{\mathcal{K}^{n_j}}[(\mathbf{x}^r - \mathbf{y}^r)_{[j]}] \\ &= \Pi_{\mathcal{K}^{n_j}}[(\mathbf{x}^* - M\mathbf{x}^* - \mathbf{q} + \mathbf{v}^r)_{[j]}] \\ &= \Pi_{\mathcal{K}^{n_j}}[(\mathbf{x}^* - M\mathbf{x}^* - \mathbf{q})_{[j]}] + \Pi_{\mathcal{C}_{\mathcal{K}^{n_j}}(\mathbf{0})}(\mathbf{v}^r_{[j]}) + O(\|\mathbf{v}^r_{[j]}\|^2). \end{aligned} \tag{3.21}$$

By (3.3) furthermore, we have

$$\Pi_{\mathcal{C}_{\mathcal{K}^{n_j}}(\mathbf{0})}(\mathbf{v}^r_{[j]}) = tJ_{n_j}(M\mathbf{x}^* + \mathbf{q})_{[j]}$$

for some $t \geq 0$, which together with (3.21) implies that

$$\begin{aligned} \langle (M\mathbf{x}^* + \mathbf{q})_{[j]}, \mathbf{x}^r_{[j]} \rangle &= \langle (M\mathbf{x}^* + \mathbf{q})_{[j]}, tJ_{n_j}(M\mathbf{x}^* + \mathbf{q})_{[j]} \rangle + O(\|\mathbf{v}^r\|^2) \\ &= O(\|\mathbf{x}^r - \mathbf{x}^{r-1}\|^2), \end{aligned}$$

where the last equality follows from $(M\mathbf{x}^* + \mathbf{q})_{[j]} \in \mathrm{bd}(\mathcal{K}^{n_j})$ and (3.16). Therefore, there exist a $\rho_2 > 0$ and an r_5 such that for all $r > r_5$, we have

$$\sum_{i \in \Xi_3} \langle (M\mathbf{x}^* + \mathbf{q})_{[i]}, (\mathbf{x}^r - \mathbf{x}^*)_{[i]} \rangle \leq \rho_2 \|\mathbf{x}^r - \mathbf{x}^{r-1}\|^2. \tag{3.22}$$

Since these four scenarios are mutually exclusive, consequently, our assertion follows from (3.15), (3.19), (3.20), and (3.22). ▯

LEMMA 3.8. *Let $\{\mathbf{x}^r\}$ be the sequence generated by the matrix splitting algorithm (Algorithm 1). Then there exists an $r_0 > 0$ such that*

$$f(\mathbf{x}^r) - f(\mathbf{x}^*) \leq \kappa_2 \|\mathbf{x}^r - \mathbf{x}^{r-1}\|^2 \quad \forall r \geq r_0. \tag{3.23}$$

Proof. By the mean value theorem, there exists a $\mathbf{w}^r$ lying on the line segment joining $\mathbf{x}^r$ with $\mathbf{x}^*$ such that

$$f(\mathbf{x}^r) - f(\mathbf{x}^*) = \langle M\mathbf{w}^r + \mathbf{q}, \mathbf{x}^r - \mathbf{x}^* \rangle.$$

This combined with (3.14) and (3.13) shows that there exists an $r_0 > 0$ such that for all $r > r_0$,

$$\begin{aligned} f(\mathbf{x}^r) - f(\mathbf{x}^*) &= \langle M\mathbf{w}^r + \mathbf{q}, \mathbf{x}^r - \mathbf{x}^* \rangle \\ &= \langle M\mathbf{w}^r - M\mathbf{x}^*, \mathbf{x}^r - \mathbf{x}^* \rangle + \langle M\mathbf{x}^* + \mathbf{q}, \mathbf{x}^r - \mathbf{x}^* \rangle \\ &\leq \|M\| \|\mathbf{x}^r - \mathbf{x}^*\|^2 + \rho \|\mathbf{x}^r - \mathbf{x}^{r-1}\|^2 \\ &= (\|M\|\alpha + \rho) \|\mathbf{x}^r - \mathbf{x}^{r-1}\|^2, \end{aligned}$$

and the assertion follows. ▯

The following lemma ([29, Lemma 3.1]) is used by Luo and Tseng, and could also lead us to the linear convergence of $\{\mathbf{x}^r\}$, in the sense of the *R-linear* convergence.

LEMMA 3.9. *Let $\{\mathbf{x}^r\}$ be a sequence of vectors in $\mathcal{K}$ satisfying* (3.9) *for some scalar $\kappa_1 > 0$. If $f(\mathbf{x}^r)$ converges linearly, then $\{\mathbf{x}^r\}$ also converges linearly.*

With these preparatory results in hand, we finally are able to prove the linear convergence of Algorithm 1.

THEOREM 3.10. *Let $\{\mathbf{x}^r\}$ be the sequence generated by the matrix splitting algorithm (Algorithm* 1*). Then $f(\mathbf{x}^r)$ converges linearly to $f(\mathbf{x}^*)$ and $\{\mathbf{x}^r\}$ converges linearly to $\mathbf{x}^*$.*

Proof. Since (3.9) holds for $\{\mathbf{x}^r\}$ generated by Algorithm 1, by Lemma 3.9 we only need to prove that $f(\mathbf{x}^r)$ converges linearly to $f(\mathbf{x}^*)$.

By (3.9) and (3.23), it follows that

$$f(\mathbf{x}^r) - f(\mathbf{x}^*) \leq \kappa_1\kappa_2(f(\mathbf{x}^{r-1}) - f(\mathbf{x}^r)),$$

which implies that

$$(\kappa_1\kappa_2 + 1)(f(\mathbf{x}^r) - f(\mathbf{x}^*)) \leq \kappa_1\kappa_2(f(\mathbf{x}^{r-1}) - f(\mathbf{x}^*))$$

or, equivalently,

$$\frac{f(\mathbf{x}^r) - f(\mathbf{x}^*)}{f(\mathbf{x}^{r-1}) - f(\mathbf{x}^*)} \leq \frac{\kappa_1\kappa_2}{\kappa_1\kappa_2 + 1} < 1,$$

and the proof is complete. ∎

4. The BSOR method.

4.1. The block lower triangular splitting.

Now, we focus on the computational issue of the main procedure, `Step 2`, in the matrix splitting algorithm (Algorithm 1). Let $\mathbf{x}^r$ be the current iteration. Recall from (2.3) that this step is to obtain $\mathbf{x}^{r+1} \in \mathrm{SOL}(\mathcal{K}, B, C\mathbf{x}^r + \mathbf{q})$. The flexibility of choosing the splitting (B, C) makes it possible to decouple the subproblem $\mathrm{LCP}(\mathcal{K}, B, C\mathbf{x}^r + \mathbf{q})$ into m elementary SOCLCPs, each of which is defined on a single second-order cone. This is realizable, for example, if we choose the matrix B as block lower triangular and, thereby, M can be represented in the form:

$$M = B + C = \begin{bmatrix} B_{11} & & & \\ M_{21} & B_{22} & & \\ \vdots & \vdots & \ddots & \\ M_{m1} & \dots & \dots & B_{mm} \end{bmatrix} + C, \tag{4.1}$$

where $M_{ij} \in \mathbb{R}^{n_i \times n_j}$ are the block submatrices of M partitioned according to (1.2). Here, we have assumed, for the moment, that each diagonal block $B_{ii} \in \mathbb{R}^{n_i \times n_i}$ has the GUS property. Such splitting is already proposed in [17] and we refer to the matrix splitting algorithm with such a structure of B as the BSOR method. Keeping this and the following equivalent relation

$$\mathbf{x} \in \mathcal{K},\ \mathbf{y} \in \mathcal{K} \text{ and } \mathbf{x}^\top\mathbf{y} = 0 \Longleftrightarrow \mathbf{x}_{[i]} \in \mathcal{K}^{n_i},\ \mathbf{y}_{[i]} \in \mathcal{K}^{n_i},\ \text{and } \mathbf{x}_{[i]}^\top\mathbf{y}_{[i]} = 0$$

for $i = 1, 2, \dots, m$ in mind, we know that the next iteration $\mathbf{x}^{r+1} \in \mathcal{K}$ in `Step 2` of Algorithm 1 can be equivalently obtained via solving sequentially the following subproblems for $i = 1, 2, \ldots, m$:

$$\mathbf{x}_{[i]} \in \mathcal{K}^{n_i}, \ B_{ii}\mathbf{x}_{[i]} + \mathbf{t}^r_{[i]} \in \mathcal{K}^{n_i}, \ \mathbf{x}^\top_{[i]}(B_{ii}\mathbf{x}_{[i]} + \mathbf{t}^r_{[i]}) = 0, \tag{4.2}$$

where

$$\mathbf{t}^r_{[i]} := \begin{cases} \mathbf{q}^r_{[1]} & \text{if } i = 1, \\ \sum_{j=1}^{i-1} M_{ij}\mathbf{x}^{r+1}_{[j]} + \mathbf{q}^r_{[i]} & \text{if } i > 1, \end{cases} \tag{4.3}$$

and $\mathbf{q}^r := \mathbf{q} + C\mathbf{x}^r = [(\mathbf{q}^r_{[1]})^\top, \ldots, (\mathbf{q}^r_{[m]})^\top]^\top$.

Note that for each $i = 1, 2, \ldots, m$, the problem (4.2) solves $\mathbf{x}_{[i]}$ from the elementary problem LCP$(\mathcal{K}^{n_i}, B_{ii}, \mathbf{t}^r_{[i]})$. From this point of view, the efficiency of the approach for such elementary problems could determine the overall performance of BSOR. Unfortunately, this appears not to be an easy problem in general, and [17] suggests specifying every diagonal block B_{ii} as the structure (1.3). However, as we have pointed out in section 1 that there are two potential limitations, in the next subsections, we shall introduce and integrate the BN iteration [45] (with a slight modification) to BSOR, which could be implemented rather efficiently and is able to exploit the sparsity as well.

4.2. The BN iteration for LCP$(\mathcal{K}^l, A, \mathbf{u})$. Before we discuss why and how to integrate the BN iteration [45] into BSOR in an efficient way, we shall first take some effort to describe the underlying principle and basic procedures in the BN iteration for solving an elementary problem LCP$(\mathcal{K}^l, A, \mathbf{u})$ where A has the GUS property.

Suppose $\mathbf{a} \in \mathbb{R}^l$ is the unique solution of LCP$(\mathcal{K}^l, A, \mathbf{u})$. The BN iteration begins with the following simple observation (by Lemma 3.2; see also [17, Proposition 4.1]):

(C1) $\mathbf{u} \in \mathcal{K}^l \Longrightarrow \mathbf{a} = \mathbf{0}$;
(C2) $-A^{-1}\mathbf{u} \in \mathcal{K}^l \Longrightarrow \mathbf{a} = -A^{-1}\mathbf{u}$;
(C3) $\mathbf{u} \notin -A\mathcal{K}^l \cup \mathcal{K}^l \Longrightarrow \exists s^* > 0$ such that $\mathbf{a} = -(A - s^*J_l)^{-1}\mathbf{u} \in \mathrm{bd}(\mathcal{K}^l)$, where J_l is defined by (1.4).

As the first two cases are trivial, we can design methods targeting at finding the scalar $s^* > 0$ for (C3), which, however, is still not an easy problem. Fortunately, the basic properties [44] for the GUS of A bring us a useful geometry picture of LCP$(\mathcal{K}^l, A, \mathbf{u})$, which serves as the fundamental of the BN iteration. To be precise, for any $s \geq 0$, define $A_s := A - sJ_l$ and $\mathcal{K}_s := A_s\mathcal{K}^l = \{A_s\mathbf{x} | \mathbf{x} \in \mathcal{K}^l\}$. Obviously, $\mathcal{K}_s$ is a cone and $\mathcal{K}_0 = A\mathcal{K}^l$. It is shown in [44] that when A has the GUS property, the following important properties hold.

THEOREM 4.1. *For* LCP$(A, \mathcal{K}^l, \mathbf{u})$, *if A has the GUS property, then*

(i) *AJ_l has nonnegative eigenvalues and has exactly one positive real eigenvalue $\tau > 0$. Moreover,* $\mathrm{rank}(AJ_l - \tau I_l) = l - 1$. *There exists $\mathbf{w} \in \mathrm{int}(\mathcal{K}^l)$ such that $\mathbf{w}$ is the eigenvector of AJ_l associated with τ;*
(ii) *there exists a vector $\mathbf{v} \in \mathrm{int}(\mathcal{K}^l)$ such that $\mathbf{v}$ is the eigenvector of $A^\top J_l$ associated with τ;*
(iii) *the range $\mathcal{R}(A_\tau)$ of A_τ is a hyperplane in $\mathbb{R}^l$. Moreover $\mathcal{R}(A_\tau) = (J_l\mathbf{v})^\perp$, where $\mathbf{v} \in \mathrm{int}(\mathcal{K}^l)$ is the eigenvector of $A^\top J_l$ associated with the unique positive eigenvalue τ;*
(iv) *for all $0 < t < s < \tau$, $\mathcal{K}_t \backslash \{\mathbf{0}\} \subset \mathrm{int}(\mathcal{K}_s)$ and $\mathcal{K}_s$ lies in one side of $\mathcal{R}(A_\tau)$; if $s > t > \tau$, then $-\mathcal{K}^l \backslash \{\mathbf{0}\} \subset \mathrm{int}(\mathcal{K}_s)$, $\mathcal{K}_s \backslash \{\mathbf{0}\} \subset \mathrm{int}(\mathcal{K}_t)$, and $\mathcal{K}_t$ lies in the other side of $\mathcal{R}(A_\tau)$.*

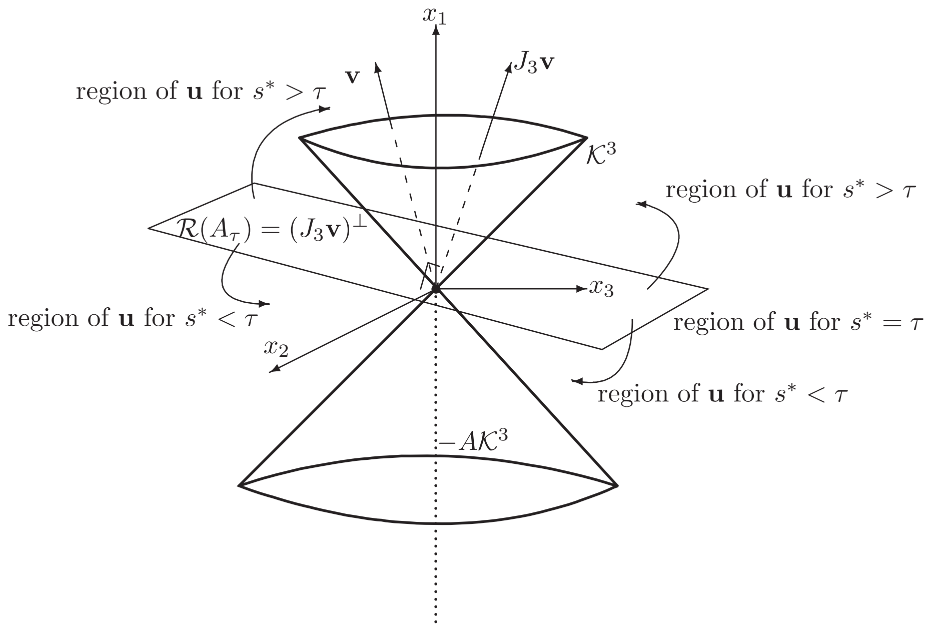
x_1
$\mathbf{v}$
$J_3\mathbf{v}$
region of $\mathbf{u}$ for $s^* > \tau$
$\mathcal{K}^3$
region of $\mathbf{u}$ for $s^* > \tau$
$\mathcal{R}(A_\tau) = (J_3\mathbf{v})^\perp$
x_3
region of $\mathbf{u}$ for $s^* < \tau$
region of $\mathbf{u}$ for $s^* = \tau$
x_2
region of $\mathbf{u}$ for $s^* < \tau$
$-A\mathcal{K}^3$

FIG. 1. *Geometry illustration of the three cases of s^* for $l = 3$.*

As an implication of Theorem 4.1 (and see more details in [45, 44]), we consider the desired value $s^* > 0$ separately in the following three mutually exclusive cases:

$$s^* = \begin{cases} = \tau & \text{if } (-\mathbf{u})^\top J_l \mathbf{v} = 0 \text{ or equivalently, } \mathbf{u} \in \mathcal{R}(A_\tau), \\ < \tau & \text{if } (-\mathbf{u})^\top J_l \mathbf{v} > 0, \\ > \tau & \text{if } (-\mathbf{u})^\top J_l \mathbf{v} < 0. \end{cases}$$

As an illustration for $l = 3$, Figure 1 provides a geometrical picture of these three cases.

For the first case $s^* = \tau$, a direct method, namely SOCLCP*tau* in [45], is designed for efficiently finding the solution $\mathbf{a} \in \mathrm{SOL}(\mathcal{K}^l, A, \mathbf{u})$, whereas for the others, Theorem 4.1 also suggests an iterative scheme to adjust the current approximation s_k to s^*. In particular, by defining

$$\mathbf{a}^k := -(A - s_k J_l)^{-1}\mathbf{u}, \tag{4.4}$$

the adjustment procedure for s_k should proceed as follows:

(i) in the case of $(-\mathbf{u})^\top J_l \mathbf{v} > 0$, the current approximation s_k should be decreased (resp., increased) if $\mathbf{a}^k \in \mathrm{int}(\mathcal{K}^l)$ (resp., $\mathbf{a}^k \notin \mathcal{K}^l$);

(ii) in the case of $(-\mathbf{u})^\top J_l \mathbf{v} < 0$, the current approximation s_k should be increased (resp., decreased) if $\mathbf{a}^k \in \mathrm{int}(\mathcal{K}^l)$ (resp., $\mathbf{a}^k \notin \mathcal{K}^l$).

This observation then facilitates the construction of a bisection procedure to approximate the value s^*. In fact, we know that for the case $(-\mathbf{u})^\top J_l \mathbf{v} > 0$, it is true that the lower α and the upper bound β of s^* is 0 and τ, respectively, while for $(-\mathbf{u})^\top J_l \mathbf{v} < 0$, the upper bound β can be obtained via detecting the smallest integer $\iota > 0$ satisfying

$$-(A - 2^\iota \tau J_l)^{-1}\mathbf{u} \notin \mathcal{K}^l.$$

Based on Theorem 4.1, one can easily see that $2^{\iota-1}\tau \leq s^* < 2^{\iota}\tau$ and, thus, in this case, $\alpha = 2^{\iota-1}\tau$ and $\beta = 2^{\iota}\tau$ serve as the lower and the upper bound for s^*, respectively. With the upper bound α and lower bound β of s^* in hand, we can halve the length of (α, β) in each iteration, according to the above adjustment procedure.

To make the above bisection procedure really efficient, we must take care of the efficiency in solving the linear system of (4.4). Fortunately, we have shown in [45] that every bisection iteration only requires $2l^2$ flops instead of $O(l^3)$. Such surprising saving in computation is possible due to the following simple but vital properties.

LEMMA 4.2. *Let $Q = \mathrm{diag}\{1, \bar{Q}\} \in \mathbb{R}^{l\times l}$ be any orthogonal matrix, i.e., $Q^\top Q = I_l$. Then it is true that* (i) $\mathbf{z} \in \mathcal{K}^l \iff Q\mathbf{z} \in \mathcal{K}^l$, (ii) $\mathbf{a} \in \mathrm{SOL}(\mathcal{K}^l, A, \mathbf{u}) \iff Q\mathbf{a} \in \mathrm{SOL}(\mathcal{K}^l, QAQ^\top, Q\mathbf{u})$, *and* (iii) $Q^\top J_l Q = J_l$.

Proof. The proof is straightforward based on the definitions of $\mathcal{K}^l$, $\mathrm{SOL}(\mathcal{K}^l, A, \mathbf{u})$, and the matrix J_l. $\square$

According to Lemma 4.2, the procedure described in [45] first transforms A, by an orthogonal matrix in the form $Q = \mathrm{diag}\{1, \bar{Q}\} \in \mathbb{R}^{l\times l}$, into either an upper or a lower Hessenberg matrix H and, analogously, transforms $\mathbf{a}$ and $\mathbf{u}$ to $\bar{\mathbf{a}} := Q\mathbf{a}$ and $\bar{\mathbf{u}} := Q\mathbf{u}$, respectively. As a result, the linear system (4.4) is equivalently transformed into

$$(H - s_k J_l)\bar{\mathbf{a}}^k = -\bar{\mathbf{u}} \tag{4.5}$$

for $\bar{\mathbf{a}}^k := Q\mathbf{a}^k$. Because the bisection procedure only needs to check if $\mathbf{a}^k \in \mathcal{K}^l$ or not, and by (i) of Lemma 4.2, it suffices to work directly on $\bar{\mathbf{a}}^k$ during the iteration. Moreover, since the coefficient matrix $H - s_k J_l$ of (4.5) remains (upper or lower) Hessenberg for different s_k, the computational costs of each bisection iteration (i.e., solving the linear system (4.5)) consequently can be reduced to $2l^2$ flops (note: A linear system of size l with an upper (or a lower) Hessenberg coefficient matrix can be solved in $2l^2$ flops [11, section 4.3.4]. See more details in [45]). This is the bisection procedure.

To accelerate the linear convergence of the bisection iteration, the Newton iteration can further be integrated efficiently. Suppose we have already obtained the orthogonal matrix $Q = \mathrm{diag}\{1, \bar{Q}\}$ and focus then on the equivalent problem $\mathrm{SOL}(\mathcal{K}^l, H, \bar{\mathbf{u}})$. Again, the Newton iteration is designed for the case (C3) and aims at finding the solution pair $(\bar{\mathbf{a}}, s^*)$ through the following nonlinear system:

$$\mathcal{F}(\mathbf{h}, s) = \begin{bmatrix} (H - sJ_l)\mathbf{h} + \bar{\mathbf{u}} \\ -\frac{1}{2}\mathbf{h}^\top J_l \mathbf{h} \end{bmatrix} = \mathbf{0}.$$

It is shown in [45] that when A (equivalently H) has the GUS property, then for any $\bar{\mathbf{u}} \notin -H\mathcal{K}^l \cup \mathcal{K}^l$, the Jacobian $\mathrm{D}_{(\mathbf{h},s)}\mathcal{F}(\mathbf{h}, s)$ at the solution pair $(\bar{\mathbf{a}}, s^*)$ of $\mathrm{LCP}(\mathcal{K}^l, H, \bar{\mathbf{u}})$ is nonsingular. This ensures the local quadratic convergence of the sequence $\{(\bar{\mathbf{a}}^k, s_k)\}$ generated by the Newton iteration:

$$\bar{\mathbf{a}}^{k+1} = \bar{\mathbf{a}}^k + \Delta\bar{\mathbf{a}} \quad \text{and} \quad s_{k+1} = s_k + \Delta s,$$

where $(\Delta\bar{\mathbf{a}}, \Delta s)$ is the solution to

$$\begin{bmatrix} H - s_k J_l & -J_l\bar{\mathbf{a}}^k \\ -(J_l\bar{\mathbf{a}}^k)^\top & 0 \end{bmatrix} \begin{bmatrix} \Delta\bar{\mathbf{a}} \\ \Delta s \end{bmatrix} = -\mathcal{F}(\bar{\mathbf{a}}^k, s_k). \tag{4.6}$$

Denoting $\mathcal{F}(\bar{\mathbf{a}}^k, s_k) := [\mathcal{F}_1^\top, \mathcal{F}_2]^\top$ with $\mathcal{F}_1 \in \mathbb{R}^l$ and $\mathcal{F}_2 \in \mathbb{R}$, and rewriting (4.6) as

$$(H - s_k J_l)\Delta\bar{\mathbf{a}} - J_l\bar{\mathbf{a}}^k\Delta s = -\mathcal{F}_1 \quad \text{and} \quad -(J_l\bar{\mathbf{a}}^k)^\top\Delta\bar{\mathbf{a}} = -\mathcal{F}_2,$$

it can be easily verified that $(\Delta\bar{\mathbf{a}}, \Delta s)$ of (4.6) are

$$\left\{\begin{array}{l} \Delta s = [(J_l\bar{\mathbf{a}}^k)^\top (H - s_kJ_l)^{-1}F_1 + F_2]/[(J_l\bar{\mathbf{a}}^k)^\top (H - s_kJ_l)^{-1}J_l\bar{\mathbf{a}}^k], \\ \Delta\bar{\mathbf{a}} = (H - s_kJ_l)^{-1}J_l\bar{\mathbf{a}}^k\Delta s - (H - s_kJ_l)^{-1}F_1. \end{array}\right. \tag{4.7}$$

Computationally, (4.7) can be completed via solving two vectors $\mathbf{b}_1 = (H-s_kJ_l)^{-1}F_1$ and $\mathbf{b}_2 = (H - s_kJ_l)^{-1}J_l\bar{\mathbf{a}}^k$ from $(H - s_kJ_l)\mathbf{b}_1 = F_1$ and $(H - s_kJ_l)\mathbf{b}_2 = J_l\bar{\mathbf{a}}^k$, respectively. Fortunately again, since the coefficient matrix $H - s_kJ_l$ of both systems is upper (or lower) Hessenberg for every s_k, they can be achieved with $3l^2$ flops.[1] Consequently, by additionally counting the updating for the right-hand side $F(\bar{\mathbf{a}}^k, s_k)$ to $F(\bar{\mathbf{a}}^{k+1}, s_{k+1})$, we know that a Newton step can be realized using $5l^2$ flops.

To combine the bisection and the Newton iteration, we make a slight difference from that in [45], where the Newton iteration is only triggered after a number of bisection iterations. Due to the local quadratic convergence and its low costs in computation, the Newton iteration is indeed preferable. Based on this observation, we suggest continuing to use the Newton iteration if the resulting iteration $(\bar{\mathbf{a}}^{k+1}, s_{k+1})$ is acceptable, in the sense that s_{k+1} is within the lower bound α and the upper bound β of s^*. The bisection iteration, on the other hand, would be employed only if s_{k+1} is outside $[\alpha, \beta]$, which, then is followed by the updating of $(\bar{\mathbf{a}}^{k+1}, s_{k+1})$ and halving of the length of the interval $[\alpha, \beta]$.[2] Because the bisection iteration converges globally, such a strategy is of both global convergence and local quadratic convergence. Our numerical experiments indicate that this type of combination performs slightly better than that of [45]. In summary, the main steps of the BN iteration based on these techniques can be presented in Algorithm 2 (BN for short).

We make four remarks for Algorithm 2 to conclude this subsection.

- For step (A) in Algorithm 2, the largest real eigenvalue $\tau > 0$ of $A^\top J_l$ and its corresponding eigenvector $\mathbf{v}$ can be obtained by several state-of-the-art eigensolvers, for example, the Arnoldi method [26, 27], the Jacobi-Davidson iteration [37], and the conjugate-gradient-type method [20], to name a few.
- Generally, the computational costs for (A) and (B) are $O(l^3)$. However, we will see in the next subsection that the computation in steps in (A) and (B) could be trivial if we choose a special splitting (B, C), without loss of the linear convergence of the matrix splitting method.
- For the stopping criterion in (C), we also adopt the one similar to (2.8). That is, we terminate the iteration either when $k > iter_{\mathsf{BN}}$ or when

$$\max\{\|\bar{\mathbf{a}}(2:l)\| - \bar{\mathbf{a}}(1), 0\} + \max\{\|\mathbf{c}(2:l)\| - \mathbf{c}(1), 0\} + |\bar{\mathbf{a}}^\top\mathbf{c}| \le \varepsilon_{\mathsf{BN}}, \tag{4.8}$$

  where $\mathbf{c} = H\bar{\mathbf{a}} + \bar{\mathbf{u}}$. In practice, we recommend $iter_{\mathsf{BN}} \in [30, 50]$ and $\varepsilon_{\mathsf{BN}} \in [10^{-8}, 10^{-6}]$.
- By integrating BN into the matrix splitting algorithm with the BSOR splitting (B, C) in (4.1), we have finally the following overall BSOR_BN algorithm (Algorithm 3).

[1] We can first use l^2 flops to transform $H - s_kJ_l$ into an upper (or lower) triangular matrix, and use $2l^2$ flops to obtain $\mathbf{b}_1 = (H - s_kJ_l)^{-1}F_1$ and $\mathbf{b}_2 = (H - s_kJ_l)^{-1}J_l\bar{\mathbf{a}}^k$ from the resulting upper (or lower) triangular systems.

[2] In practice, in order to ensure the global convergence, one can trigger the bisection iteration whenever, for example, more than 5 successive Newton iterations, without convergence, are observed.

ALGORITHM 2. THE BISECTION-NEWTON(BN) ITERATION FOR LCP($\mathcal{K}^l, A, \mathbf{u}$): FUNCTION $\mathbf{a} = \mathrm{BN}(A, \mathbf{u})$.

```
    INPUT: A matrix $A \in \mathbb{R}^{l\times l}$ with the GUS property and $\mathbf{u} \in \mathbb{R}^l$.
    OUTPUT: The solution $\mathbf{a} \in \mathrm{SOL}(\mathcal{K}^l, A, \mathbf{u})$.
    if $\mathbf{u} \in \mathcal{K}^l$ then
        $\mathbf{a} = \mathbf{0}$ and return
    if $-A^{-1}\mathbf{u} \in \mathcal{K}^l$ then
        $\mathbf{a} = -A^{-1}\mathbf{u}$ and return
(A) Find the eigenpair $(\tau, \mathbf{v})$ of $A^\top J_l$ where $\tau > 0$ and $\mathbf{v} \in \mathrm{int}(\mathcal{K}^l)$;
    Set $Index = -\mathbf{u}^\top J_l \mathbf{v}$;
(B) Find the orthogonal matrix $Q = \mathrm{diag}\{1, \bar{Q}\}$ such that $QAQ^\top = H$ is
      upper (or lower) Hessenberg and set $\bar{\mathbf{u}} := Q\mathbf{u}$;
    if $Index = 0$ then
        % call SOCLCPtau in [45]
        $s^* = \tau$; $\mathbf{a} = \mathsf{SOCLCPtau}(A, \mathbf{u}, \tau)$; return
    if $Index > 0$ then
        $\alpha = 0$; $\beta = \tau$;
    else
        Find the smallest integer $\iota > 0$ satisfying $-(H - 2^\iota \tau J_l)^{-1}\bar{\mathbf{u}} \notin \mathcal{K}^l$;
        Set $\alpha = 2^{\iota-1}\tau$; $\beta = 2^\iota \tau$;
    Set $k = 0$, $s_k = \frac{\alpha+\beta}{2}$, $\bar{\mathbf{a}}^k = -(H - s_k J_l)^{-1}\bar{\mathbf{u}}$;
(C) while the given stopping rule is not met do
        % Newton's step
        Solve $(\Delta\bar{\mathbf{a}}, \Delta s)$ by (4.7) and update $\bar{\mathbf{a}}^{k+1} = \bar{\mathbf{a}}^k + \Delta\bar{\mathbf{a}}$, $s_{k+1} = s_k + \Delta s$;
        if $s_{k+1} \notin [\alpha, \beta]$ then
            % switch to the bisection iteration
            $s_{k+1} = \frac{\alpha+\beta}{2}$; $\bar{\mathbf{a}}^{k+1} = -(H - s_{k+1}J_l)^{-1}\bar{\mathbf{u}}$;
            if $\bar{\mathbf{a}}^{k+1} \in \mathrm{int}(\mathcal{K}^l)$ then
                if $Index > 0$ then
                    $\beta = s_{k+1}$;
                else
                    $\alpha = s_{k+1}$;
            else
                if $Index > 0$ then
                    $\alpha = s_{k+1}$;
                else
                    $\beta = s_{k+1}$;
        Set $k := k + 1$;
    Set $\mathbf{a} = Q^\top \bar{\mathbf{a}}^k$;
```

4.3. The choice of the diagonal blocks of B. Now, returning to the general SOCLCP (1.1), we will discuss the last computational issue of the BSOR_BN algorithm (Algorithm 3) for choosing the diagonal blocks B_{ii} of B. We will see that BSOR_BN has great flexibility in choosing the structure of B_{ii}, which could play a crucial role for the performance of BSOR_BN, in the sense that the subproblems in step (D) of BSOR_BN could be solved quite efficiently, without loss of its linear convergence. In the following, we will propose three important structures of B_{ii}.

Lower triangular structure. We first provide a rather effective and useful structure of B_{ii}, for which the dominant computational costs in solving the subprob-

Algorithm 3. The block SOR (BSOR) method for LCP($\mathcal{K}, M, \mathbf{q}$):
FUNCTION $\mathbf{x} = \mathrm{BSOR\_BN}(\mathcal{K}, M, \mathbf{q})$.

```
    INPUT: $\mathcal{K}$, a symmetric matrix $M \in \mathbb{R}^{n\times n}$ and $\mathbf{q} \in \mathbb{R}^n$.
    OUTPUT: The solution $\mathbf{x} \in \mathrm{SOL}(\mathcal{K}, M, \mathbf{q})$.
    Select any $\mathbf{x}^0 \in \mathbb{R}^n$ and set $r = 0$;
    while $r < iter_{BSOR}$ or (2.8) is not met do
(D)     for $i = 1 : m$ do
            $\mathbf{x}^{r+1}_{[i]} = \mathsf{BN}(B_{ii}, \mathbf{t}^r_{[i]})$, where $\mathbf{t}^r_{[i]}$ is given by (4.3);
        Set $r = r + 1$;
```

lem (D) of Algorithm 3 could be $O(n_i^2)$. To see why this is possible, recall that the only computational trouble of BN (Algorithm 2) lies in steps (A) and (B), where the dominant eigenpair $(\tau, \mathbf{v})$ of $A^\top J_l$ and an orthogonal matrix Q are needed. However, if A is positive definite and is further lower triangular, then we have the following.

Lemma 4.3. *If $A \in \mathbb{R}^{l\times l}$ is positive definite and lower triangular, then the first entry $A(1,1)$ of A is the only positive eigenvalue of $A^\top J_l$ with the associated eigenvector $\mathbf{e}_1 := [1, 0, \ldots, 0]^\top \in \mathbb{R}^l$. That is, $(\tau, \mathbf{v}) = (A(1,1), \mathbf{e}_1)$.*

Proof. The proof is straightforward. $\square$

Lemma 4.3 shows that if each B_{ii} is positive definite and lower triangular, then the step (A) of Algorithm 2 is trivial; moreover, since B_{ii} is already lower triangular (is of course lower Hessenberg), the step (B) is trivial too (i.e., Q is the identity matrix). In addition, because $B_{ii} - s_k J_{n_i}$ is lower triangular for every s_k, completing a single bisection or a Newton step can further be reduced to n_i^2 and $4n_i^2$ flops, respectively. These are quite appealing features of the combination of BSOR with the BN algorithm for the SOCLCP (1.1). To specify B_{ii}, we can rely on the following well-known result (see, e.g., [7]).

Lemma 4.4. *If $M_{ii} = D_i + L_i + U_i \in \mathbb{R}^{n_i\times n_i}$ is symmetric and positive definite, where D_i, L_i, and U_i are the corresponding diagonal, strictly lower triangular, and strictly upper triangular parts of M_{ii}, then (B_{ii}, C_{ii}), where $B_{ii} = L_i + \omega_i^{-1} D_i$, is a regular splitting of M_{ii} if and only if $0 < \omega_i < 2$. In this case, B_{ii} is also positive definite.*

Based on Lemma 4.4, we can split each diagonal block matrix M_{ii} into B_{ii} and C_{ii}, where the parameter ω_i could vary for different $i = 1, 2, \ldots, m$, provided that they are within $(0, 2)$. Although the optimal value for ω_i is difficult to give, a practical range is $[1, 1.8]$. In our numerical testing, we choose $\omega_i = 1.4$ for all $i = 1, \ldots, m$. Noting that

$$B - C = L - L^\top + \mathrm{diag}\{B_{11} - C_{11}, \ldots, B_{mm} - C_{mm}\},$$

where $L \in \mathbb{R}^{n\times n}$ is the strictly block upper triangular part of M, we can ensure by Lemma 4.4 that (B, C) is a regular splitting of M. Combining all these techniques together, we have finally an efficient solver, namely, BSOR_BN_L (the letter "L" indicates that B_{ii} is lower triangular), for the SOCLCP (1.1) with M symmetric and positive definite. It deserves to be pointed out that, besides the low computational costs per iteration and the global and linear convergence, BSOR_BN_L does not destroy the sparsity of the original matrix M. This makes BSOR_BN_L a rather attractive and highly competitive method for the large and sparse SOCLCP (1.1).

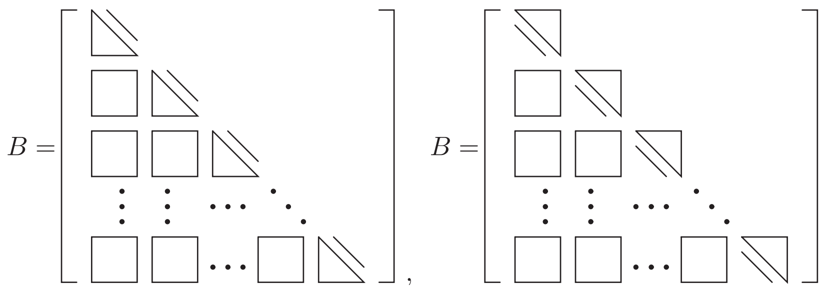

FIG. 2. *Structure of the matrix B resulting from the splittings* (4.9) *(left) and* (4.10) *(right), respectively.*

Hessenberg structure. Another type of B_{ii} that deserves to be mentioned is the Hessenberg structure. If we partition M_{ii} as

$$M_{ii} = L_i^H + (L_i^H)^\top + D_i^H,$$

where D_i^H is the *tribidiagonal* matrix (i.e., $D_i^H(i,j) = 0$ for all $|i-j| > 1$) and L_i^H satisfies $L_i^H(i,j) = 0$ for all $j > i+1$ (i.e., elements above the subdiagonal are all zeroes), then we can choose the splitting (B_{ii}, C_{ii}) of M_{ii} either as

$$B_{ii} = L_i^H + D_i^H + \Lambda_i, \qquad C_{ii} = (L_i^H)^\top - \Lambda_i \quad \text{or} \tag{4.9}$$

$$B_{ii} = (L_i^H)^\top + D_i^H + \Lambda_i, \qquad C_{ii} = L_i^H - \Lambda_i, \tag{4.10}$$

where $\Lambda_i = \operatorname{diag}\{\vartheta_1^i, \ldots, \vartheta_{n_i}^i\}$ is a certain positive (semi)definite matrix. The structures of B resulting from (4.9) and (4.10) look like Figure 2. The codes based on such a splitting will be called BSOR_BN_H (the letter "H" stands for Hessenberg) to distinguish it from BSOR_BN_L, and in our numerical testing, we will report our experience using the splitting (4.10). In this case, upon noticing that

$$\begin{aligned} B_{ii} + B_{ii}^\top &= L_i^H + (L_i^H)^\top + D_i^H + D_i^H + 2\Lambda_i = M_{ii} + (D_i^H + 2\Lambda_i) \quad \text{and} \\ B_{ii} - C_{ii}^\top &= (L_i^H)^\top - L_i^H + (D_i^H + 2\Lambda_i), \end{aligned}$$

we can always choose the diagonal matrix Λ_i so that $D_i^H + 2\Lambda_i$ is positive definite (for example by letting ϑ_j^i be sufficiently large) and, thereby, guarantee the linear convergence of BSOR_BN. According to our implementation of BN, for such types of splitting, step (B) in Algorithm 2 remains trivial, but we should pay additional computational costs for step (A). Nevertheless, we may gain a faster outer loop convergence of BSOR_BN_H in return for the computation of step (A). The question of how to choose proper diagonal matrices Λ_i to accelerate the speed of the matrix splitting algorithm BSOR_BN_H is certainly very interesting and may play an important role in BSOR_BN_H, but is beyond the scope of the present paper. In our numerical testing in the next section, we will simply choose Λ_i so that $D_i^H + 2\Lambda_i$ is dominant diagonally by a factor 0.1, i.e.,

$$\vartheta_j^i = \max\left\{ \frac{|D_i^H(j,j+1)| + |D_i^H(j-1,j)| - D_i^H(j,j) + 0.1}{2}, 0 \right\}, \tag{4.11}$$

to illustrate the working of the idea and also to provide a comparison with algorithm BSOR_BN_L and other methods.

Diagonal structure. Last we briefly discuss a special case of the foregoing lower triangular splitting, in which B becomes simply a diagonal matrix. Because the diagonal matrix is automatically lower triangular, the computational benefits for each subproblem in BSOR_BN_L are all inherited.[3] On the other hand, this choice of B has its own advantages in dealing with the SOCLCP (1.1) with M only symmetric and positive semidefinite. As we have seen in Theorem 3.4, when M is symmetric and positive semidefinite and $\mathrm{SOL}(\mathcal{K}, M, \mathbf{q}) \neq \emptyset$, one type of splitting (B, C) that guarantees the convergence is to ensure $B - M$ is symmetric and positive definite. To fulfill this condition and also to preserve the lower triangular structure of each B_{ii} (for releasing the computation burden in steps (A) and (B) of Algorithm 2), B should be diagonal, where one can make B, for example, *diagonally dominant* by choosing $B(i,i) > \sum_{j=1}^{n} |M(i,j)|$ to simplify the implementation.

5. Numerical experiments.

In this section, we will report our numerical experiments of two types of implementation of the BSOR method: BSOR_BN_L and BSOR_BN_H. Our objective is to show the efficiency of these methods in solving the SOCLCP (1.1) for the symmetric and positive definite case, as well as for the symmetric positive semidefinite case. In particular, we will evaluate the following three performances of our methods:

(a) the efficiency for medium scale, dense, ill-conditioned symmetric and positive definite matrices M;
(b) the efficiency for large scale, sparse, ill-conditioned symmetric and positive definite matrices M;
(c) the efficiency for large scale, sparse, symmetric positive and semidefinite matrices M.

For this purpose, we carry out the numerical testing upon MATLAB 7.13.0 (R2011b) on an iMac ME086CH/A with Intel Core i5@2.7 GHz and 8 GB memory, and compare the performance of our methods with four other state-of-the-art algorithms. In the next subsection, we will first briefly describe the test codes and list the parameters involved in BSOR_BN_L and BSOR_BN_H.

5.1. Test codes.

In order to evaluate the numerical performance and demonstrate clearly the efficiency of our method, we present the numerical results from the smoothing-regularization method [16] (SRM[4] for short), the matrix splitting method proposed in [17] (BSOR_HYYF for short), and two popular MATLAB software packages: SDPT3 [39, 40, 41] (version 4)[5] and SeDuMi [38] (version 1.3).[6] For the parameters involved in SRM [16], we use the same values, and adopt the stopping criteria that have been suggested and tested previously in [16]. For BSOR_HYYF, since we have pointed out that it shares the same outer loop framework with ours, and thus, in order to give a reasonable comparison, we set a similar stopping criterion for the outer loop iteration: We terminate the iteration whenever the iteration number $k > iter_{\max} = 1000$ or the iteration $\mathbf{x} = \mathbf{x}^k$ satisfies (2.8) with $\varepsilon = 10^{-7}$. Other parameters of BSOR_HYYF remain the same as those used in [17]. The software packages SDPT3 and SeDuMi do not directly solve SOCLCP in the form of (1.1), but when M

[3]In fact, if B_{ii} is diagonal, the computational costs in BN can be further reduced, as the linear system with the coefficient matrix $B_{ii} - s_k J_{n_i}$ can be solved using only $O(n_i)$ flops.

[4]The MATLAB code of SRM is available at: http://www-optima.amp.i.kyoto-u.ac.jp/∼hayashi/index_e.html.

[5]The MATLAB package of SDPT3 is available at: http://www.math.nus.edu.sg/∼mattohkc/sdpt3.html.

[6]The MATLAB package of SeDuMi is available at: http://sedumi.ie.lehigh.edu/downloads.

Table 2
Parameters in BSOR_BN_L *and* BSOR_BN_H.

Parameter	Value	Description
ε	$10^{-7} \sim 10^{-4}$	Stopping criterion of (2.8) for the outer loop of BSOR_BN.
$iter_{\mathsf{BSOR}}$	$500 \sim 1000$	Maximal number of iterations for the outer loop of BSOR_BN.
$\varepsilon_{\mathsf{BN}}$	$10^{-8} \sim 10^{-6}$	Stopping criterion of (4.8) for BN.
$iter_{\mathsf{BN}}$	$30 \sim 50$	Maximal number of iterations for BN.
ω_i	$1.0 \sim 1.8$	SOR parameters for BSOR_BN_L.
ϑ_j^i	Given by (4.11)	Diagonal elements of Λ_i (4.10) in BSOR_BN_H.

is symmetric and positive (semi)definite with decomposition

$$M = \tilde{M}^\top \tilde{M},$$

according to [40, section 4.6] and [38], the equivalent quadratic second-order cone programming (3.8) can be equivalently converted to the standard programming problems for SDPT3 and SeDuMi, respectively. The detailed procedures are omitted here because of the limit on the length of the paper but the reader can refer to [40, section 4.6] and [38]. The default values of SDPT3 are used and, for SeDuMi, we set pars.eps $= 10^{-10}$ as the desired accuracy. For our codes BSOR_BN_L and BSOR_BN_H, the meanings of the involved parameters and their recommended ranges are summarized in Table 2.

5.2. Testing for medium scale, dense, ill-conditioned symmetric and positive definite matrices. In this set of tests, for a prescribed positive integer pair (n, m) satisfying

$$n_1 = \cdots = n_m = \frac{n}{m}, \tag{5.1}$$

we generate 10 triplets $(M = \tilde{M}^\top \tilde{M}, \mathbf{q}, \mathbf{x}^0)$, where $\mathbf{q}$ and $\mathbf{x}^0$ are randomly generated with elements uniformly distributed in the interval $[-1, 1]$ and $\tilde{M}$ is generated as follows:

$$\tilde{M} = \mathtt{diag}([1 : \delta : 1 + (n-1) * \delta].\hat{\ }0.5) * \mathtt{orth}(\mathtt{randn(n,n)}), \quad \text{where } \delta = \frac{\mathtt{cond}}{n}.$$

We point out that the way we construct M fulfills the following two tasks: (1) it controls the condition number of M, which is $(1 - \frac{1}{n})\mathtt{cond} + 1 \approx \mathtt{cond}$, and (2) it is convenient to formulate the equivalent programming problems used in SDPT3 and SeDuMi, as both of them are formulated in terms of $\tilde{M}$, not M. In this set of testing, we choose $\mathtt{cond} = 10^6$ and set the stopping criteria for BSOR_BN_L and BSOR_BN_H as $\epsilon = 10^{-6}$, $\varepsilon_{\mathsf{BN}} = 10^{-8}$, $iter_{\mathsf{BSOR}} = 500$, and $iter_{\mathsf{BN}} = 30$.

Thus, for each pair (n, m), we have 10 SOCLCPs LCP$(\mathcal{K}, M, \mathbf{q})$. Feeding these problems into the tested six codes,[7] we record the number of iterations (# iter), the CPU time (measured by MATLAB function cputime), and the relative accuracy χ_r defined by (2.9) of each algorithm. To report these results, we finally average these statistics over the 10 cases and present them in Table 3.

These numerical results are quite encouraging. It shows very clearly the high efficiency of our methods for the SOCLCP (1.1). The following are our observations based on the numerical results on this set of tests.

[7]For SDPT3 and SeDuMi, we input the data $(\mathcal{K}, \tilde{M}, \mathbf{q})$.

TABLE 3
Number of iterations, CPU time, and the relative accuracy χ_r over 10 random tests with condition number 10^6.

	n	2000			4000			5000		
	m	10	100	200	10	100	200	10	100	1000
# iter	BSOR_BN_L	11.0	15.3	178.0	13.0	15.3	339.2	12.0	15.3	500
	BSOR_BN_H	19.7	24.1	23.3	21.0	24.0	25.3	21.3	23.7	500
	BSOR_HYYF	12.7	1000	1000	11.0	1000	1000	11.3	1000	1000
	SRM	6.0	7.0	6.3	6.5	7.0	7.0	6.7	7.0	7.0
	SDPT3	18.7	18.7	26.7	19.0	18.9	19.3	20.0	19.3	30.1
	SeDuMi	20.3	27.3	30.7	21.0	26.0	31.3	20.3	26.0	36.3
CPU(s)	BSOR_BN_L	1.5	4.5	90.3	7.0	6.4	167.9	10.9	7.4	1240.3
	BSOR_BN_H	3.8	9.8	12.7	17.8	11.0	13.6	30.1	11.9	1234.2
	BSOR_HYYF	6.0	164.1	247.7	29.7	297.2	387.6	49.2	385.5	1283.5
	SRM	304.3	387.7	418.6	2378.9	2885.2	2487.2	3941.5	5353.5	6291.3
	SDPT3	48.9	139.9	376.1	2765.6	596.8	1017.1	5667.6	1050.8	11969.1
	SeDuMi	102.0	137.1	156.3	1063.9	1314.3	1591.9	2063.7	2639.9	3701.2
χ_r	BSOR_BN_L	$3.0e-14$	$4.2e-14$	$2.1e-11$	$1.1e-14$	$4.3e-14$	$1.7e-11$	$5.3e-15$	$2.2e-14$	$5.9e-11$
	BSOR_BN_H	$6.6e-14$	$4.9e-14$	$5.7e-14$	$4.0e-14$	$4.3e-14$	$4.9e-14$	$4.0e-14$	$4.4e-14$	$4.6e-10$
	BSOR_HYYF	$1.3e-15$	$4.7e-12$	$9.8e-12$	$1.6e-16$	$6.9e-13$	$7.5e-13$	$2.0e-16$	$9.8e-13$	$3.7e-11$
	SRM	$5.7e-10$	$3.8e-10$	$1.9e-09$	$8.6e-10$	$1.2e-09$	$5.3e-10$	$8.2e-10$	$1.0e-09$	$2.8e-09$
	SDPT3	$3.4e-14$	$9.5e-14$	$2.8e-15$	$2.4e-14$	$8.3e-14$	$8.5e-14$	$3.1e-14$	$1.2e-13$	$1.5e-12$
	SeDuMi	$5.1e-12$	$1.9e-11$	$3.4e-11$	$1.4e-12$	$8.5e-12$	$1.8e-12$	$1.2e-16$	$2.4e-12$	$3.0e-11$

(1) Table 3 clearly shows the efficiency of both BSOR_BN_L and BSOR_BN_H in terms of the speeds and the relative accuracy as well. It is encouraging to note that for $(n, m) = (5000, 10)$ and $(n, m) = (5000, 100)$, BSOR_BN_L and BSOR_BN_H require only less than 30(s) CPU time to reach solutions with relative accuracy $\chi_r = 10^{-14}$, while SRM, SDPT3, and SeDuMi need more than 1000(s) to obtain the solutions with roughly the same order of accuracy.

(2) It is observed that for $n = 4000$ and $n = 5000$, both BSOR_BN_L and BSOR_BN_H use less CPU time for $m = 100$ than that for $m = 10$. This is explainable because when $(n, m) = (5000, 10)$, the eigensolver for step (A) in Algorithm 2 needs to find the dominant eigenpair of matrices of size $n_i = 500$, which costs more CPU time than that for $m = 100$ (i.e., $n_i = 50$).

(3) Because SDPT3 and SeDuMi employ the interior-point methods, the superlinear convergence is guaranteed and, therefore, the number of iterations and the accuracy change slightly with different (n, m). However, the CPU times increase very fast as n increases, which is due to the computational complexity $O(n^3)$ of the interior-point methods. By comparison, because the matrix splitting iteration converges linearly, the number of iterations increases as m increases. Note that BSOR_BN_H reaches the maximal iteration $iter_{\mathsf{BSOR}} = 500$ for $(n, m) = (5000, 1000)$. The CPU times of BSOR_BN_L and BSOR_BN_H increase very fast from $m = 100$ to $m = 1000$.

5.3. Testing for large scale sparse, symmetric and positive definite matrices. As we have pointed out that BSOR_BN_L is of advantage in exploiting the sparsity of M and does not need to employ any eigensolver for step (A) in Algorithm 2, we next conduct our numerical testing on large scale sparse problems with various condition numbers and compare it with SDPT3 and SeDuMi. Because the input data matrix for SDPT3 and SeDuMi is $\tilde{M}$, the number of nonzeros of $M = \tilde{M}^\top \tilde{M}$ is

TABLE 4
Numerical results over 10 *random tests with* $n = 10^4$ *and* `density` $= 0.0005$.

				BSOR_BN_L			SDPT3			SeDuMi		
`rc`	m	$d_M, d_{\tilde{M}}$	`cond`	# iter	CPU(s)	χ_r	# iter	CPU(s)	χ_r	# iter	CPU(s)	χ_r
10^{-1}	10	$2.6e-3, 5.5e-4$	$7.9e+1$	20.0	2.7	$5.4e-11$	11.7	47.4	$6.0e-07$	17.7	1065.2	$3.3e-08$
	100	$3.1e-3, 6.5e-4$	$8.3e+1$	22.7	12.4	$7.2e-11$	12.0	146.7	$1.3e-07$	21.7	1480.0	$1.1e-07$
	1000	$2.6e-3, 5.5e-4$	$8.7e+1$	27.7	234.1	$8.4e-11$	16.7	258.5	$2.8e-07$	25.7	8263.2	$4.6e-08$
10^{-2}	10	$2.6e-3, 5.4e-4$	$9.2e+3$	306.7	64.6	$8.1e-09$	20.2	84.7	$4.4e-05$	21.7	1388.6	$2.9e-06$
	100	$2.6e-3, 5.5e-4$	$9.0e+3$	484.3	263.5	$1.2e-08$	21.4	299.9	$4.6e-05$	27.0	2081.0	$1.0e-06$
	1000	$7.3e-4, 3.4e-3$	$1.2e+4$	594.0	5001.4	$1.1e-07$	20.7	243.1	$1.3e-04$	30.0	10322.6	$1.1e-06$
$10^{-2.5}$	10	$2.6e-3, 5.5e-4$	$9.7e+4$	800	191.3	$2.3e-03$	21.3	149.2	$1.1e-02$	27.7	1598.5	$1.1e-04$
	100	$3.4e-3, 7.3e-4$	$1.5e+5$	800	477.6	$1.6e-03$	20.7	263.3	$3.7e-03$	32.3	2260.3	$1.7e-05$
	1000	$3.7e-3, 7.9e-4$	$1.8e+5$	800	6924.5	$1.3e-03$	22.7	262.2	$1.2e-03$	29.0	9567.6	$5.7e-04$

larger than that of $\tilde{M}$. We choose to first generate the sparse $\tilde{M}$ with a prescribed sparsity and condition number, and then form $M = \tilde{M}^\top \tilde{M}$. In our testing, it appears that the number of nonzeros in M is about 5 times that in $\tilde{M}$. In particular, for a prescribed positive integer pair (n, m) satisfying (5.1), we generate 10 triplets $(M = \tilde{M}^\top \tilde{M}, \mathbf{q}, \mathbf{x}^0)$, where $\mathbf{q}$ and $\mathbf{x}^0$ are randomly generated with elements uniformly distributed in the interval $[-1, 1]$ and $\tilde{M}$ is generated as follows:

$$\tilde{M} = \texttt{sprandsym(n,density,rc,2)}. \tag{5.2}$$

The MATLAB function `sprandsym(n,density,rc,2)` returns an n-by-n symmetric and positive definite random matrix (having less structure than the matrix generated by `sprandsym(n,density,rc,1)`) with the prescribed `density` and an approximate condition number $\frac{1}{\texttt{rc}}$ (which implies that the condition number `cond` of M is approximately $\frac{1}{\texttt{rc}^2}$). Based on our current hardware environment, in this set of tests, we choose $n = 10^4$, `density` $= 0.0005$, and different `rc` so that `cond` varies from 10^2 to 10^5. Furthermore, we relax the stopping criterion of BSOR_BN_L to be $\epsilon = 10^{-4}$, $\varepsilon_{\mathsf{BN}} = 10^{-7}$, $iter_{\mathsf{BSOR}} = 800$, and $iter_{\mathsf{BN}} = 30$.

We recorded the numerical results from the three codes and listed in Table 4 the averages of numbers of iterations (# iter), the CPU times, and the relative accuracy χ_r over the 10 tests. Moreover, for each pair (n, m), we also provided the average condition number `cond` of M as well as the densities d_M and $d_{\tilde{M}}$ of M and $\tilde{M}$, respectively.

Table 4 clearly shows the numerical performance of each method as m and the condition numbers `cond` of M vary. In particular, we conclude that

(1) the accuracy of the computed solution decreases as m and `cond` increase for all methods, but, overall, BSOR_BN_L achieves more accurate solutions in most cases;
(2) all methods need more CPU time to converge as m and `cond` increase, but overall, BSOR_BN_L uses much less CPU time in most cases;
(3) the number of iterations of SDPT3 and SeDuMi changes slightly as m and `cond` increase, while the number of iterations of BSOR_BN_L increases when m and `cond` increase;

TABLE 5
Numerical results over 10 *random tests with* $n = 10^4$ *and* `density` $= 0.0005$.

			BSOR_BN_L			SDPT3			SeDuMi		
`rc`	m	$d_M, d_{\tilde{M}}$	# iter	CPU(s)	χ_r	# iter	CPU(s)	χ_r	# iter	CPU(s)	χ_r
10^{-1}	10	$2.6e-3, 5.5e-4$	16.7	50.6	$5.2e-09$	13.0	54.8	$1.1e-07$	17.3	1146.3	$3.2e-08$
	100	$3.4e-3, 7.3e-4$	15.7	23.9	$7.2e-09$	13.3	181.9	$1.8e-07$	23.3	1840.0	$5.4e-08$
	1000	$2.6e-3, 5.5e-4$	19.0	59.0	$7.2e-09$	18.0	202.0	$1.5e-09$	26.2	8166.7	$4.4e-08$
10^{-2}	10	$2.6e-3, 5.5e-4$	652.3	2746.0	$8.2e-09$	18.6	80.2	$2.3e-05$	22.7	1292.5	$1.9e-06$
	100	$3.4e-3, 7.3e-4$	447.0	641.6	$1.8e-07$	21.3	267.7	$2.6e-04$	28.1	1927.9	$2.3e-06$
	1000	$3.7e-3, 7.9e-4$	585.7	1896.7	$2.1e-08$	23.3	267.4	$2.4e-06$	31.0	9715.3	$7.3e-07$
$10^{-2.5}$	10	$2.6e-3, 5.5e-4$	800	3829.4	$4.1e-03$	21.3	82.9	$8.3e-03$	24.7	1432.6	$7.5e-05$
	100	$3.4e-3, 7.3e-4$	800	1226.4	$6.8e-03$	21.3	272.9	$1.3e-02$	31.7	2174.3	$1.5e-06$
	1000	$3.7e-3, 7.9e-4$	800	2626.0	$7.4e-03$	22.3	257.6	$3.1e-01$	32.0	10215.2	$7.5e-03$

(4) overall, from the achieved accuracy, the CPU time, and the fact that M contains about 5 times more nonzeros than $\tilde{M}$, BSOR_BN_L in general is much more efficient than the other two in this set of tests.

5.4. Testing for large scale sparse, symmetric and positive semidefinite matrices. In this set of numerical tests, we evaluate our method for LCP$(\mathcal{K}, M, \mathbf{q})$ with the large scale sparse, symmetric and positive semidefinite matrix M. In this situation, we can first apply Theorem 3.5 to regularize M into a symmetric positive definite matrix, and then employ BSOR_BN_L for the resulting regularization problem. In particular, in order to produce a specific such test problem for a prescribed pair (n, m), we first generate a random matrix $\tilde{M}$ by (5.2) with prescribed `density` and `rc` as well, and then form

$$M = \tilde{M}^\top T \tilde{M}, \quad \text{where} \quad T = \text{diag}\{\underbrace{0, \ldots, 0}_{5}, \underbrace{1, \ldots, 1}_{n-10}, \underbrace{0, \ldots, 0}_{5}\}.$$

The vectors $\mathbf{q}$ and $\mathbf{x}^0$ are randomly chosen.

In our testing, we set the same stopping criteria for BSOR_BN_L as in section 5.3, and choose $n = 10^4$, `density` $= 0.0005$, and `rc` varying from 10^{-1} to $10^{-2.5}$. Analogously, we observed that the number of nonzeros in M is about 5 times more than that in $\tilde{M}$. For a given pair (n, m), we generate 10 such test problems and feed $(\mathcal{K}, T\tilde{M}, \mathbf{q})$ to the codes SDPT3 and SeDuMi, while for our codes BSOR_BN_L, on the other hand, we set the regularization parameter $\nu = 10^{-10}$, and solve the problem LCP$(\mathcal{K}, M + \nu I_n, \mathbf{q})$. The numerical statistics are summarized in Table 5. It is observed that the performances of SDPT3 and SeDuMi are almost the same as that in section 5.3, but BSOR_BN_L has a different performance due to the regularization procedure. Overall, from the achieved accuracy, the CPU times and the fact that M contains about 5 times more nonzeros than $\tilde{M}$, BSOR_BN_L in general is very efficient and highly competitive with the other two in this set of tests.

6. Conclusion. In this paper, we have presented two noticeable contributions for the SOCLCP LCP$(\mathcal{K}, M, \mathbf{q})$. As a well-known and one of the most widely used methods in the classical LCP, the general matrix splitting method, when applied to LCP$(\mathcal{K}, M, \mathbf{q})$, is also proved to converge at least linearly, for a symmetric and positive definite matrix M, even under the present rounding errors. This result successfully generalizes the linear convergence of Luo and Tseng [29] into the SOCLCP (1.1). To implement the general matrix splitting method, we next proposed a highly efficient BSOR algorithm, in which the subproblems can be solved by a BN iteration. We have shown that, in each iteration, the resulting algorithm BSOR_BN_L only involves solving triangular linear systems and invokes $O(n_i^2)$ flops. Moreover, the algorithm does not destroy the sparse structure of M and is capable of exploiting the sparsity effectively. The efficiency of our methods is demonstrated in our numerical report, by comparing the numerical performance with other state-of-the-art algorithms.

To conclude the paper, we would like to make one last point clear. Although the current version of BSOR is not designed for the general SOCLCP where M is not positive semidefinite, it is able to deal with the case when M is positive semidefinite (not necessarily symmetric), due to the theoretical convergence result, Theorem 3.4, and the flexibility to choose the splitting (B, C), provided that $B - M$ is symmetric and positive definite. Since there is much flexibility in such a choice of (B, C), we did not expand this issue in detail in this paper, but will investigate it in the future.

Acknowledgments. We would like to thank the two anonymous referees for their helpful suggestions and comments which have improved the presentation of this paper significantly. The comments from one referee also led us to a better understanding of the performance of our method and its efficiency for large scale sparse problems. We also thank Professor Kim-Chuan Toh for the technical helps in implementing SDPT3 and Dr. Shunsuke Hayashi for generously providing us with their codes of the matrix splitting method (BSOR_HYYF) [17], which made our numerical experiments and comparisons possible.

REFERENCES

[1] F. Alizadeh and D. Goldfarb, *Second-order cone programming,* Math. Program., 95 (2003), pp. 3–51.

[2] J. Burke and S. Xu, *The global linear convergence of a noninterior path-following algorithm for linear complementarity problems,* Math. Oper. Res., 23 (1998), pp. 719–734.

[3] X. D. Chen and S. H. Pan, *A descent method for a reformulation of the second-order cone complementarity problem,* J. Comput. Appl. Math., 213 (2008), pp. 547–558.

[4] X. Chen, L. Qi, and D. Sun, *Global and superlinear convergence of the smoothing Newton method and its application to general box constrained variational inequalities,* Math. Comp., 67 (1998), pp. 519–540.

[5] X. D. Chen, D. F. Sun, and J. Sun, *Complementarity functions and numerical experiments on some smoothing Newton methods for second-order-cone complementarity problems,* Comput. Optim. Appl., 25 (2003), pp. 39–56.

[6] J.-S. Chen and P. Tseng, *An unconstrained smooth minimization reformulation of the second-order cone complementarity problem,* Math. Program. Ser. B, 104 (2005), pp. 293–327.

[7] R. W. Cottle, J.-S. Pang, and R. E. Stone, *The Linear Complementarity Problem,* Comp. Sci. Sci. Comput., Academic Press, Boston, MA, 1992.

[8] R. W. Cottle, G. H. Golub, and R. S. Sacher, *On the solution of large, structured linear complementarity problems: The block partitioned case,* Appl. Math. Optim., 4 (1978), pp. 347–363.

[9] F. Facchinei and J.-S. Pang, *Finite Dimensional Variational Inequality and Complementarity Problems,* Vol. I, Springer-Verlag, New York, 2003.

[10] M. Fukushima, Z.-Q. Luo, and P. Tseng, *Smoothing functions for second-order-cone complementarity problems,* SIAM J. Optim., 12 (2002), pp. 436–460.

[11] G. H. Golub and C. F. Van Loan, *Matrix Computations*, 3rd ed., Johns Hopkins University Press, Baltimore, MD, 1996.

[12] M. S. Gowda and R. Sznajder, *Automorphism invariance of P- and GUS-properties of linear transformations on Euclidean Jordan algebras,* Math. Oper. Res., 31 (2006), pp. 109–123.

[13] M. S. Gowda and R. Sznajder, *Some global uniqueness and solvability results for linear complementarity problems over symmetric cones,* SIAM J. Optim., 18 (2007), pp. 461–481.

[14] M. S. Gowda, R. Sznajder, and J. Tao, *Some P-properties for linear transformations on Euclidean Jordan algebras,* Linear Algebra Appl., 393 (2004), pp. 203–232.

[15] S. Hayashi, N. Yamashita, and M. Fukushima, *Robust Nash equilibria and second-order cone complementarity problems,* J. Nonlinear Convex Anal., 6 (2005), pp. 283–296.

[16] S. Hayashi, N. Yamashita, and M. Fukushima, *A combined smoothing and regularization method for monotone second-order cone complementarity problems,* SIAM J. Optim., 15 (2005), pp. 593–615.

[17] S. Hayashi, T. Yamaguchi, N. Yamashita, and M. Fukushima, *A matrix-splitting method for symmetric affine second-order cone complementarity problems,* J. Comput. Appl. Math., 175 (2005), pp. 335–353.

[18] J.-B. Hiriart-Urruty and C. Lemaréchal, *Convex Analysis and Minimization Algorithms* I, Springer-Verlag, New York, 1993.

[19] Z. H. Huang and T. Ni, *Smoothing algorithms for complementarity problems over symmetric cones,* Comput. Optim. Appl., 45 (2010), pp. 557–579.

[20] A. V. Knyazev, *Toward the optimal preconditioned eigensolver: Locally optimal block preconditioned conjugate gradient method,* SIAM J. Sci. Comput., 23 (2001), pp. 517–541.

[21] Y. Kanno, J. A. C. Martins, and A. Pinto da Costa, *Three-dimensional quasi-static frictional contact by using second-order cone linear complementarity problem,* Internat. J. Numer. Methods Engrg., 65 (2006), pp. 62–83.

[22] C. Kanzow, *Some noninterior continuation methods for linear complementarity problems,* SIAM J. Matrix Anal. Appl., 17 (1996), pp. 851–868.

[23] M. Kojima, N. Megiddo, T. Noma, and A. Yoshise, *A Unified Approach to Interior Point Algorithms for Linear Complementarity Problems,* Lecture Notes in Comput. Sci. 538, Springer-Verlag, Berlin, 1991.

[24] M. Kojima, S. Shindoh, and S. Hara, *Interior-point methods for the monotone semidefinite linear complementarity problem in symmetric matrices,* SIAM J. Optim., 7 (1997), pp. 86–125.

[25] L. Kong, J. Sun, and N. Xiu, *A regularized smoothing Newton method for symmetric cone complementarity problems,* SIAM J. Optim., 19 (2008), pp. 1028–1047.

[26] R. B. Lehoucq and D. C. Sorensen, *Deflation techniques for an implicitly restarted Arnoldi iteration,* SIAM J. Matrix Anal. Appl., 17 (1996), pp. 789–821.

[27] R. B. Lehoucq, D. C. Sorensen, and C. Yang, *ARPACK Users' Guide: Solution of Large-Scale Eigenvalue Problems with Implicitly Restarted Arnoldi Methods,* SIAM, Philadelphia, 1998.

[28] M. S. Lobo, L. Vandenberghe, S. Boyd, and H. Lebret, *Application of second-order cone programming,* Linear Algebra Appl., 284 (1998), pp. 193–228.

[29] Z.-Q. Luo and P. Tseng, *On the linear convergence of descent methods for convex essentially smooth minimization,* SIAM J. Control Optim., 30 (1992), pp. 408–425.

[30] R. D. C. Monteiro and T. Tsuchiya, *Polynomial convergence of primal-dual algorithms for the second-order cone program based on the MZ-family of directions,* Math. Program., 88 (2000), pp. 61–83.

[31] J. L. Morales, J. Nocedal, and M. Smelyanskiy, *An algorithm for the fast solution of symmetric linear complementarity problems,* Numer. Math., 111 (2008), pp. 251–266.

[32] S. Pan and J.-S. Chen, *A damped Gauss-Newton method for the second-order cone complementarity problem,* Appl. Math. Optim., 59 (2009) pp. 293–318.

[33] S. Pan and J.-S. Chen, *A linearly convergent derivative-free descent method for the second-order cone complementarity problem,* Optimization, 59 (2010), pp. 1173–1197.

[34] J.-S. Pang, D, Sun, and J. Sun, *Semismooth homeomorphisms and strong stability of semidefinite and Lorentz complementarity problems,* Math. Oper. Res., 28 (2003), pp. 39–63.

[35] S. H. Schmieta and F. Alizadeh, *Associative and Jordan algebras, and polynomial time interior-point algorithms for symmetric cones,* Math. Oper. Res., 26 (2001), pp. 543–564.

[36] S. H. Schmieta and F. Alizadeh, *Extension of primal-dual interior point algorithms to symmetric cones,* Math. Program. Ser. A, 96 (2003), pp. 409–438.

[37] G. L. G. Sleijpen and H. A. van der Vorst, *A Jacobi-Davidson iteration method for linear eigenvalue problems,* SIAM Rev., 42 (2000), pp. 267–293.

[38] J. F. Sturm, *Using SeDuMi* 1.02*, a MATLAB toolbox for optimization over symmetric cones,* Optim. Methods Softw., 11 (1999), pp. 625–653.

[39] K. C. Toh, M. J. Todd, and R. H. Tutuncu, *SDPT3–a MATLAB software package for semidefinite programming,* Optim. Methods Softw., 11 (1999), pp. 545–581.

[40] K. C. Toh, M. J. Todd, and R. H. Tutuncu, *On the Implementation and Usage of SDPT3-a MATLAB Software Package for Semidefinite-Quadratic-Linear Programming, Version* 4.0, http://www.math.nus.edu.sg/∼mattohkc/sdpt3/guide4-0-draft.pdf (2006).

[41] R. H. Tutuncu, K. C. Toh, and M. J. Todd, *Solving semidefinite-quadratic-linear programs using SDPT3,* Math. Program. Ser. B, 95 (2003), pp. 189–217.

[42] Y. Xia and J. M. Peng, *A continuation method for the linear second-order cone complementarity Problem,* in Computational Science and Its Applications-ICCSA 2005, Vol. 4, Lecture Notes in Comput. Sci. 3483, Springer, Berlin, 2005, pp. 290–300.

[43] A. Yoshise, *Interior point trajectories and a homogeneous model for nonlinear complementarity problems over symmetric cones,* SIAM J. Optim., 17 (2006), pp. 1129–1153.

[44] W. H. Yang and Y. Yuan, *The GUS-property of second-order cone linear complementarity problems,* Math. Program. Ser. A, 141 (2013), pp. 295–317.

[45] L.-H. Zhang and W. H. Yang, *An efficient algorithm for second-order cone linear complementarity problems*, Math. Comp., 83 (2014), pp. 1701–1726.