Inexact subgradient algorithm with a non-asymptotic convergence guarantee for copositive programming problems

Mitsuhiro Nishijima[1] Pierre-Louis Poirion[2] Akiko Takeda[3]

January 16, 2026

Abstract

In this paper, we propose a subgradient algorithm with a non-asymptotic convergence guarantee to solve copositive programming problems. The subproblem to be solved at each iteration is a standard quadratic programming problem, which is NP-hard in general. However, the proposed algorithm allows this subproblem to be solved inexactly. For a prescribed accuracy $\epsilon > 0$ for both the objective function and the constraint arising from the copositivity condition, the proposed algorithm yields an approximate solution after $O(\epsilon^{-2})$ iterations, even when the subproblems are solved inexactly. We also discuss exact and inexact approaches for solving standard quadratic programming problems and compare their performance through numerical experiments. In addition, we apply the proposed algorithm to the problem of testing complete positivity of a matrix and derive a sufficient condition for certifying that a matrix is not completely positive. Experimental results demonstrate that we can detect the lack of complete positivity in various doubly nonnegative matrices that are not completely positive.

Key words. Copositive programming, Semi-infinite programming, Subgradient-based method, Standard quadratic programming, Completely positive matrices

[1]Center for Advanced Intelligence Project, RIKEN, 1-4-1, Nihonbashi, Chuo-ku, 1030027, Tokyo, Japan. (mitsuhiro.nishijima@riken.jp).

[2]Center for Advanced Intelligence Project, RIKEN, 1-4-1, Nihonbashi, Chuo-ku, 1030027, Tokyo, Japan. (pierre-louis.poirion@riken.jp).

[3]Graduate School of Information Science and Technology, The University of Tokyo, 7-3-1 Hongo, Bunkyo-ku, 1138656, Tokyo, Japan; Center for Advanced Intelligence Project, RIKEN, 1-4-1, Nihonbashi, Chuo-ku, 1030027, Tokyo, Japan (takeda@mist.i.u-tokyo.ac.jp).

1 Introduction

A copositive programming problem is a conic linear programming problem involving a copositive cone. Many studies have shown that various computationally challenging optimization problems can be reformulated as copositive programming problems in a unified manner [10, 11, 14]. Motivated by this, researchers have proposed various approaches to solving copositive programming problems.

Approximations for copositive cones are widely used to solve copositive programming problems. Typically, the cone consisting of sums of a real symmetric positive semidefinite matrix and a symmetric entrywise nonnegative matrix provides an inner approximation to a copositive cone. However, except in a few specific situations [22, 31], it is difficult to assess the accuracy of this approximation theoretically. To approximate copositive cones to arbitrary accuracy, various approximation hierarchies have been proposed [1, 16, 36, 37, 51–53, 58]. Bundfuss and Dür [13] provided an adaptive approximation scheme for copositive cones based on a simplicial partition and showed its asymptotic convergence. Žilinskas [55] also utilized a simplicial partition to solve a copositive programming problem with a single scalar variable. Moreover, cutting-plane methods have been used to solve copositive programming problems [4, 24], although they lack theoretical results on asymptotic or non-asymptotic convergence.

Copositive programming is a subclass of convex semi-infinite programming, so we can apply methods for solving convex semi-infinite programming problems to copositive programming problems [20]. Ahmed, Dür, and Still [2] interpreted the approximation schemes for copositive programming proposed in [9, 13] as discretization methods for semi-infinite programming problems and derived their asymptotic convergence rates. More recently, researchers have established theoretical results for copositive programming through the lens of semi-infinite programming, including strong duality, optimality conditions, and representations of the faces of copositive cones [21, 32–35].

In this paper, we also reformulate copositive programming problems as convex semi-infinite programming problems, specifically as convex programming problems with a single nonsmooth functional constraint, and propose a subgradient algorithm (Algorithm 1) to solve them. The bottleneck of the proposed algorithm is solving an NP-hard subproblem at each iteration, but it allows the subproblem to be solved inexactly. The proposed algorithm is based on the subgradient algorithm in [44, Equation (3.2.24)]. If we apply the subgradient algorithm in [44, Equation (3.2.24)] to the convex semi-infinite programming problem directly, we have to solve the subproblem exactly to find a constraint that attains the maximal violation. The subproblem is nonconvex in general and solving it exactly is sometimes demanding, especially for large-scale problems.

In particular, when we apply the proposed algorithm to a copositive programming problem, the subproblem at each iteration reduces to a standard quadratic programming problem. It is well known that this problem is NP-hard [41]. As discussed in Section 4, there are several approaches to solving this subproblem either exactly or inexactly. To solve the subproblem exactly, the subproblem can be reformulated as a mixed-integer linear programming problem whose size depends only on the size of the copositive programming problem [23]. To solve the subproblem inexactly, one may employ the polynomial-time approximation scheme [9], which requires evaluating only a finite number of objective values and does not require external solvers, as well as its randomized variant. In addition, we compare their performance through numerical experiments. The results show that exact computation is acceptable when the size of the problem is small, whereas inexact computation becomes a viable alternative as the size increases. In contrast to the proposed algorithm, existing approaches for copositive programming problems typically require solving potentially large-scale problems with external solvers. Many of these methods solve linear programming or semidefinite programming problems as approximations. The higher the desired accuracy, the larger the variable dimension and the number of constraints in the approximate problems, regardless of the size of the original copositive programming problems.

Moreover, we establish a theoretical non-asymptotic convergence result for the proposed algorithm, which also covers the case where the subproblems are solved inexactly. In Theorem 3.4, we provide an explicit iteration-complexity bound that guarantees that an approximate solution achieves a given accuracy. Specifically, for a prescribed accuracy $\epsilon > 0$ with respect to both the objective function and the constraint induced by copositivity, the proposed algorithm yields an approximate solution after $O(\epsilon^{-2})$ iterations. In contrast, most existing methods either lack theoretical guarantees on the accuracy of approximate solutions or only provide asymptotic convergence.

Our algorithm is closely related to the subgradient algorithms proposed by Nesterov [43], Beck et al. [6], and Wei, Haskell, and Zhao [56] for convex semi-infinite programming problems. In particular, the algorithm presented by Wei, Haskell, and Zhao [56] also permits inexact computation of subproblems. The four algorithms differ in the rules for selecting the objective or constraint function to improve at each iteration, choosing the step size, and computing the solution to output. These differences yield a smaller iteration-complexity bound for our algorithm to achieve a solution within a given accuracy than the others; see Table 1 for details. Additionally, our algorithm does not rely on a compactness assumption on the constraint set, whereas the other algorithms use a parameter that depends on such an assumption.

Furthermore, we apply the proposed algorithm to testing complete positivity of a matrix. Determining whether a matrix is completely positive is NP-hard in general settings [18], and developing practical methods for this task has been recognized as an important problem [7]. Various approaches have been proposed to detect complete positivity, some of which rely on specific matrix structures [17, 57], while others can handle unstructured matrices [4, 8, 29, 45]. Following [4], we formulate the problem of testing complete positivity as a copositive programming problem with a ball constraint. By applying the proposed algorithm to this formulation, we obtain a sufficient condition ensuring that an input matrix is not completely positive (Theorem 4.8). Our method does not require any structural assumptions on the input matrix. In addition, numerical experiments demonstrate that we can detect that certain doubly nonnegative matrices are not completely positive.

The organization of this paper is as follows. In Section 2, we introduce and recall the notation and concepts used in this paper. In Section 3, we propose an inexact subgradient algorithm for solving general convex semi-infinite programming problems and show its non-asymptotic convergence. In Section 4, we apply the proposed algorithm to general and specific copositive programming problems and discuss how to implement the algorithm. In particular, we present several approaches to solving the subproblem, a standard quadratic programming problem, that needs to be solved at each iteration. In Section 5, we verify the effectiveness of the methods introduced in the previous sections through numerical experiments. Finally, Section 6 concludes the paper.

2 Preliminaries

For a positive integer n, we define $[n] \coloneqq \{1, \dots, n\}$. For a finite set C, we use $|C|$ to denote the number of elements in C. For a real number a, we use $|a|$ to denote the absolute value of a.

Let V be a finite-dimensional real vector space equipped with a norm $\|\cdot\|$ induced by an inner product denoted by $\langle \cdot, \cdot \rangle$. We define

$$B(x, r) \coloneqq \{y \in V \mid \|x - y\| \le r\}$$

as the closed ball with center $x \in V$ and radius $r > 0$. Let S be a non-empty closed convex subset of V. For $x \in V$, the distance between the point x and the set S is defined as

$$\operatorname{dist}(x, S) \coloneqq \min_{y \in S} \|x - y\|.$$

The point $y^* \in S$ that attains $\operatorname{dist}(x, S) = \|x - y^*\|$ is uniquely determined. We call the point y^* the *projection* of x onto S and denote it by $P_S(x)$. Let $f \colon V \to (-\infty, +\infty]$ be a convex function. Let $\operatorname{dom} f \coloneqq \{x \in V \mid f(x) < +\infty\}$ denote the effective domain of f. For $x \in \operatorname{dom} f$, the *subdifferential* of f at x is defined by

$$\partial f(x) \coloneqq \{d \in V \mid f(y) \ge f(x) + \langle d, y - x \rangle \text{ for all } y \in V\}.$$

To specify the variable, we may write $\partial_x f(x)$ for $\partial f(x)$. The function f is *subdifferentiable* on a set C in V if $\partial f(x)$ is non-empty for all $x \in C$.

We use boldface lowercase letters such as $\boldsymbol{a}$ to denote vectors. For a vector $\boldsymbol{a}$, we denote its transpose by $\boldsymbol{a}^\top$ and its ith element by a_i. We use $\mathbb{R}^n$ and $\mathbb{R}^n_+$ to denote the space of n-dimensional real vectors and the set of entrywise nonnegative vectors in $\mathbb{R}^n$, respectively. The space $\mathbb{R}^n$ is equipped with the standard inner product and the 2-norm, given by $\|\boldsymbol{a}\|_2 \coloneqq \sqrt{\boldsymbol{a}^\top \boldsymbol{a}}$ for $\boldsymbol{a} \in \mathbb{R}^n$. The zero vector is written as $\mathbf{0}$. We use $\boldsymbol{e}_i$ to denote the vector whose ith element is 1 and the other elements are 0. The $(n-1)$-dimensional standard simplex in $\mathbb{R}^n$ is defined as

$$\Delta^{n-1} := \left\{ \boldsymbol{\delta} \in \mathbb{R}^n_+ \;\middle|\; \sum_{i=1}^{n} \delta_i = 1 \right\}.$$

We use boldface uppercase letters such as $\boldsymbol{A}$ to denote matrices. We write A_{ij} for the (i,j)th element of a matrix $\boldsymbol{A}$. We define $\mathcal{S}^n$ to be the space of real $n \times n$ symmetric matrices. The space $\mathcal{S}^n$ is equipped with the inner product defined as $\langle \boldsymbol{A}, \boldsymbol{B} \rangle := \sum_{i,j=1}^{n} A_{ij} B_{ij}$ for $\boldsymbol{A}, \boldsymbol{B} \in \mathcal{S}^n$ and the induced Frobenius norm defined as $\|\boldsymbol{A}\|_{\mathrm{F}} := \sqrt{\langle \boldsymbol{A}, \boldsymbol{A} \rangle}$. The zero matrix is written as $\boldsymbol{O}$. A matrix $\boldsymbol{A} \in \mathcal{S}^n$ is called *copositive* if $\boldsymbol{x}^\top \boldsymbol{A} \boldsymbol{x} \geq 0$ holds for all $\boldsymbol{x} \in \mathbb{R}^n_+$. The cone of copositive matrices, the *copositive cone* for short, in $\mathcal{S}^n$ is denoted by $\mathcal{COP}^n$.

3 An inexact subgradient algorithm for convex semi-infinite programming problems

A copositive programming problem can be regarded as a convex semi-infinite programming problem. In this section, we propose an inexact subgradient algorithm for general convex semi-infinite programming problems and establish a non-asymptotic convergence guarantee.

The semi-infinite programming problem considered here is of the form

$$\begin{aligned} \underset{x \in V}{\text{minimize}} \quad & f(x) \\ \text{subject to} \quad & g(x; \boldsymbol{\delta}) \leq 0 \text{ for all } \boldsymbol{\delta} \in \Delta, \\ & x \in S. \end{aligned} \tag{3.1}$$

Throughout this section, we make the following assumptions on the components of Problem (3.1).

Assumption 3.1.

(a) V is a finite-dimensional real vector space. The space V is equipped with an inner product $\langle \cdot, \cdot \rangle$, and we denote by $\|\cdot\|$ the norm induced by the inner product.

(b) S is a non-empty closed convex subset of V.

(c) The set of optimal solutions of (3.1), denoted by S^*, is non-empty.

(d) Δ is a non-empty compact subset of $\mathbb{R}^n$.

(e) $f\colon V \to (-\infty, +\infty]$ is a closed convex function that is subdifferentiable on S. Moreover, there exists a positive constant L_f such that $\|d\| \le L_f$ for all $x \in S$ and $d \in \partial f(x)$.

(f) For every $\boldsymbol{\delta} \in \Delta$, $g(\cdot; \boldsymbol{\delta})\colon V \to (-\infty, +\infty]$ is a closed convex function that is subdifferentiable on S. Moreover, there exists a positive constant L_g such that $\|d\| \le L_g$ for all $x \in S$, $\boldsymbol{\delta} \in \Delta$, and $d \in \partial_x g(x; \boldsymbol{\delta})$.

(g) For every $x \in S$, the inclusion $\Delta \subseteq \operatorname{dom} g(x; \cdot)$ holds and the function $g(x; \cdot)$ is continuous on Δ.

For the constants L_f and L_g that appear in (e) and (f), respectively, we define $L \coloneqq \max\{L_f, L_g\}$. Furthermore, the optimal value of (3.1) is denoted by f^*.

We define $G\colon V \to (-\infty, +\infty]$ as

$$G(x) \coloneqq \max_{\boldsymbol{\delta} \in \Delta} g(x; \boldsymbol{\delta}). \tag{3.2}$$

Then we can recast the semi-infinite constraint in (3.1), requiring $g(x; \boldsymbol{\delta}) \le 0$ for all $\boldsymbol{\delta} \in \Delta$, as a single nonsmooth functional constraint $G(x) \le 0$.

Remark 3.2. The latter claim in (e) of Assumption 3.1, i.e., the boundedness of the subdifferential of f at every $x \in S$, holds if f is subdifferentiable and L_f-Lipschitz continuous on an open subset of V that contains S. Similarly, the boundedness of the subdifferential of $g(\cdot; \boldsymbol{\delta})$ at every $x \in S$ stated in (f) holds if $g(\cdot; \boldsymbol{\delta})$ is subdifferentiable and L_g-Lipschitz continuous on an open subset of V that contains S. From the assumptions in (f) and (g), we see that G is a closed convex function with $\operatorname{dom} G \subseteq S$. Since S is a closed convex set and since f and G are closed convex functions, S^* is a closed convex set.

We present the proposed algorithm in Algorithm 1. In principle, the per-iteration subproblem is to compute $G(x_k)$, the functional-constraint violation. If the violation exceeds ϵ, we update x_k to reduce it; otherwise, we update x_k to improve the objective. However, computing $G(x_k)$ is generally nonconvex, so we seek $\boldsymbol{\delta}_k \in \Delta$ that makes $g(x_k; \boldsymbol{\delta}_k)$ approximate $G(x_k)$ well. For example, we can use the sampling approach proposed by Wei, Haskell, and Zhao [56, Section 4] to approximate $G(x_k)$. In Section 4, we discuss how to compute $G(x_k)$ exactly and inexactly when applying Algorithm 1 to copositive programming problems.

We stop Algorithm 1 after a finite number of iterations in practice. If we stop after the Nth iteration, we define

$$I_N \coloneqq \{k \in [N] \mid g(x_k; \boldsymbol{\delta}_k) \le \epsilon\}, \tag{3.3}$$

and for a minimizer k^* of $f(x_k)$ over $k \in I_N$, we output x_{k^*} as the approximate solution of (3.1). Lemma 3.3 provides an iteration-complexity bound ensuring that the

Algorithm 1 An inexact subgradient algorithm for convex semi-infinite programming problems

```
Input: Initial point $x_1 \in S$ and error tolerance $\epsilon > 0$
For $k = 1, 2, 3, \ldots$: Find $\boldsymbol{\delta}_k \in \Delta$ that approximately attains $G(x_k)$, i.e., with
    $g(x_k; \boldsymbol{\delta}_k) \simeq G(x_k)$.
    (i) If $g(x_k; \boldsymbol{\delta}_k) \le \epsilon$, take $d_k \in \partial f(x_k)$ and let
            $x_{k+1} := P_S\left(x_k - \frac{\epsilon}{\|d_k\|^2} d_k\right)$.
    (ii) Otherwise, namely if $g(x_k; \boldsymbol{\delta}_k) > \epsilon$, take $d_k \in \partial_x g(x_k; \boldsymbol{\delta}_k)$ and let
            $x_{k+1} := P_S\left(x_k - \frac{g(x_k; \boldsymbol{\delta}_k)}{\|d_k\|^2} d_k\right)$.
```

approximate solution is well defined (i.e., the set I_N is non-empty) and that it attains a prescribed accuracy. For ease of description, we define

$$\xi_k := G(x_k) - g(x_k; \boldsymbol{\delta}_k),$$

which is nonnegative by the definition of G, for each k.

Lemma 3.3. *Suppose that Assumption 3.1 holds. Let $\epsilon > 0$ and N be a positive integer such that*

$$N \ge \frac{L^2 \operatorname{dist}(x_1, S^*)^2}{\epsilon^2}. \tag{3.4}$$

Then the set I_N defined in (3.3) is non-empty, and for a minimizer k^* of $f(x_k)$ over $k \in I_N$, the following two inequalities hold:

$$f(x_{k^*}) \le f^* + \epsilon,$$
$$G(x_{k^*}) \le \epsilon + \xi_{k^*}.$$

Proof. The set S^* of optimal solutions is closed and convex (see Remark 3.2). For convenience, let $x^* := P_{S^*}(x_1)$ and $r_k := \|x_k - x^*\|$ for $k = 1, \ldots, N+1$.

To derive a contradiction, we assume that $f(x_k) > f^* + \epsilon$ for all $k \in I_N$. For every

$k \in I_N$, we have

$$\begin{aligned} r_{k+1}^2 &\overset{\text{(a)}}{=} \left\| P_S\left(x_k - \frac{\epsilon}{\|d_k\|^2} d_k\right) - x^* \right\|^2 \\ &\overset{\text{(b)}}{\leq} \left\| x_k - \frac{\epsilon}{\|d_k\|^2} d_k - x^* \right\|^2 \\ &= r_k^2 + \frac{2\epsilon}{\|d_k\|^2} \langle d_k, x^* - x_k \rangle + \frac{\epsilon^2}{\|d_k\|^2} \\ &\overset{\text{(c)}}{\leq} r_k^2 + \frac{2\epsilon}{\|d_k\|^2} (f^* - f(x_k)) + \frac{\epsilon^2}{\|d_k\|^2} \\ &\overset{\text{(d)}}{<} r_k^2 - \frac{\epsilon^2}{\|d_k\|^2} \\ &\overset{\text{(e)}}{\leq} r_k^2 - \frac{\epsilon^2}{L^2}, \end{aligned}$$

where (a) follows from the update rule in (i) of Algorithm 1, (b) holds by $x^* \in S$ and the nonexpansiveness of the projection P_S, (c) follows from $f^* = f(x^*)$ and $d_k \in \partial f(x_k)$, (d) is a consequence of the assumption for contradiction, and (e) follows from $\|d_k\| \leq L$. In addition, for every $k \in [N] \setminus I_N$, we have

$$\begin{aligned} r_{k+1}^2 &\overset{\text{(a)}}{=} \left\| P_S\left(x_k - \frac{g(x_k; \boldsymbol{\delta}_k)}{\|d_k\|^2} d_k\right) - x^* \right\|^2 \\ &\leq \left\| x_k - \frac{g(x_k; \boldsymbol{\delta}_k)}{\|d_k\|^2} d_k - x^* \right\|^2 \\ &= r_k^2 + \frac{2g(x_k; \boldsymbol{\delta}_k)}{\|d_k\|^2} \langle d_k, x^* - x_k \rangle + \frac{g(x_k; \boldsymbol{\delta}_k)^2}{\|d_k\|^2} \\ &\overset{\text{(b)}}{\leq} r_k^2 + \frac{2g(x_k; \boldsymbol{\delta}_k)}{\|d_k\|^2} (g(x^*; \boldsymbol{\delta}_k) - g(x_k; \boldsymbol{\delta}_k)) + \frac{g(x_k; \boldsymbol{\delta}_k)^2}{\|d_k\|^2} \\ &\overset{\text{(c)}}{<} r_k^2 - \frac{\epsilon^2}{\|d_k\|^2} \\ &\overset{\text{(d)}}{\leq} r_k^2 - \frac{\epsilon^2}{L^2}, \end{aligned}$$

where (a) follows from the update rule in (ii) of Algorithm 1, (b) follows from $d_k \in \partial_x g(x_k; \boldsymbol{\delta}_k)$, (c) holds because $g(x_k; \boldsymbol{\delta}_k) > \epsilon$ and $g(x^*; \boldsymbol{\delta}_k) \leq 0$, and (d) follows from $\|d_k\| \leq L$. Therefore,

$$r_{k+1}^2 < r_k^2 - \frac{\epsilon^2}{L^2} \tag{3.5}$$

holds for every $k \in [N]$. Summing Inequality (3.5) over k from 1 to N, we have

$$r_{N+1}^2 < r_1^2 - \frac{\epsilon^2 N}{L^2}. \tag{3.6}$$

In particular, the right-hand side of (3.6) is positive since r_{N+1}^2 is nonnegative. On the other hand, since N satisfies (3.4), we have

$$r_1^2 - \frac{\epsilon^2 N}{L^2} \leq r_1^2 - \mathrm{dist}(x_1, S^*)^2 = 0,$$

which is a contradiction. Hence, there exists $k \in I_N$ such that $f(x_k) \leq f^* + \epsilon$. This implies that the set I_N is non-empty and $f(x_{k^*}) \leq f^* + \epsilon$.

In addition, we have

$$G(x_{k^*}) = g(x_{k^*}; \boldsymbol{\delta}_{k^*}) + \xi_{k^*} \leq \epsilon + \xi_{k^*},$$

where the equality follows from the definition of ξ_{k^*} and the inequality follows from $k^* \in I_N$. This completes the proof. □

Typically, we introduce a nonnegative parameter α and find $\boldsymbol{\delta}_k \in \Delta$ such that the inequality

$$\xi_k \coloneqq G(x_k) - g(x_k; \boldsymbol{\delta}_k) \leq \alpha\epsilon \tag{3.7}$$

holds at each iteration. If we know a way to evaluate the exact value of $G(x)$, we can set $\alpha = 0$. Setting α to a positive value allows us to compute $G(x)$ approximately. Since ξ_{k^*} is also bounded by $\alpha\epsilon$, Lemma 3.3 directly implies the following theorem.

Theorem 3.4. *Suppose that Assumption 3.1 holds. Let $\epsilon > 0$, $\alpha \geq 0$, and N be a positive integer satisfying (3.4). For each iteration indexed by k, we find $\boldsymbol{\delta}_k \in \Delta$ satisfying (3.7). Then the set I_N defined in (3.3) is non-empty, and for a minimizer k^* of $f(x_k)$ over $k \in I_N$, we have the following two inequalities:*

$$f(x_{k^*}) \leq f^* + \epsilon,$$
$$G(x_{k^*}) \leq (1+\alpha)\epsilon.$$

In the case where $\alpha = 0$, i.e., where we solve subproblems exactly, we can view Algorithm 1 as the application of the subgradient algorithm proposed by Nesterov [44, Equation (3.2.24)] to Problem (3.1). The algorithm in [44] is comparable to the algorithms proposed in [43, Equation (3.2.13)] and [6], both applicable to convex programming problems with a single nonsmooth functional constraint. Based on the algorithm [6], Wei, Haskell, and Zhao [56] proposed an algorithm for solving convex semi-infinite programming problems. Similar to Algorithm 1, this algorithm allows one to solve the subproblem inexactly at each iteration. When applied to Problem (3.1), the four algorithms differ in which function (f or G) they improve at each iteration, in the choice of the step size, and in how the output is computed.

Except for Algorithm 1 and Nesterov's algorithm [44], the other algorithms require the compactness of S and use the value $D \coloneqq \max_{x,y \in S} \|x - y\| < +\infty$, the diameter of S, in the step size. On the other hand, as shown in Theorem 3.4, Algorithm 1 and the

Table 1: This table summarizes the number of iterations required to obtain $\bar{x} \in S$ such that $f(\bar{x}) \leq f^* + \epsilon$ and $G(\bar{x}) \leq \epsilon$ when we solve Problem (3.1) using each algorithm. Here, S is assumed to be a compact set of diameter D. Bregman distances are utilized in the algorithms of [6] and [56], but we show the number of iterations when the Bregman distance is Euclidean. For the algorithm in [56], the inequality $G(\bar{x}) \leq \epsilon$ holds in expectation. In addition, the algorithm in [56] yields $\bar{x} \in S$ such that $f(\bar{x}) \leq f^* + \frac{3}{7}\epsilon$, while Algorithm 1 yields $\bar{x} \in S$ such that $f(\bar{x}) \leq f^* + \frac{1}{1+\alpha}\epsilon$. The number of iterations in the algorithm of [6] is obtained by taking the limit as $\beta \to 1$, where $\beta < 1$ is a parameter appearing in [6, Corollary 2.1].

Exactness	Method	#Iter.
Exact	[43]	$\frac{3}{2} + 3\frac{D^2L^2}{\epsilon^2}$
	[6]	$\frac{(1+\log 2)^2}{2(2-\sqrt{2})^2}\frac{D^2L^2}{\epsilon^2} \simeq 4.2\frac{D^2L^2}{\epsilon^2}$
	[44]	$\frac{D^2L^2}{\epsilon^2}$
Inexact	[56]	$98\frac{D^2L^2}{\epsilon^2}$
	Algorithm 1	$(1+\alpha)^2\frac{D^2L^2}{\epsilon^2}$

algorithm proposed by Nesterov [44] do not assume the compactness of S but instead use $\mathrm{dist}(x_1, S^*)$, the distance between the initial point and the set of optimal solutions, to derive the number of iterations required to obtain a solution within a prescribed accuracy. Note that $\mathrm{dist}(x_1, S^*) \leq D$ holds when S is a compact set of diameter D.

Under the compactness of S, Table 1 summarizes the number of iterations required to obtain $\bar{x} \in S$ such that $f(\bar{x}) \leq f^* + \epsilon$ and $G(\bar{x}) \leq \epsilon$ when solving (3.1) with each algorithm. If we set the violation measure α to a reasonable value, e.g., $\alpha = 1$ as in the numerical experiments conducted in Section 5, Algorithm 1 requires fewer iterations than the inexact algorithm of Wei, Haskell, and Zhao [56] and even the exact algorithm of Beck et al. [6].

Note that the algorithm proposed by Nesterov [43] requires neither the error tolerance ϵ nor the number N of iterations in advance and admits an error bound with respect to the current iteration index k. However, the other algorithms use the error tolerance or the number of iterations when running.

4 Application to copositive programming problems

Copositive programming is a special case of convex semi-infinite programming; thus we can apply Algorithm 1 to copositive programming problems. In Section 4.1, we discuss how to implement the proposed algorithm to solve general copositive programming problems. In Section 4.2, we apply the proposed algorithm to the problem of testing whether a matrix is completely positive, which can be formulated as a copositive programming problem with a ball constraint. Using Theorem 3.4, we derive a sufficient condition for a matrix not to be completely positive. In Section 4.3, we discuss a potential approach to solving copositive programming problems involving copositive cones over symmetric cones.

4.1 General copositive programming problems

The copositive programming problem considered here is of the form

$$\begin{aligned} \underset{\boldsymbol{x}}{\text{minimize}} \quad & \boldsymbol{c}^\top \boldsymbol{x} \\ \text{subject to} \quad & \boldsymbol{A}_0 + \sum_{i=1}^m x_i \boldsymbol{A}_i \in \mathcal{COP}^n, \\ & \boldsymbol{x} \in S, \end{aligned} \tag{4.1}$$

where $\boldsymbol{A}_0, \dots, \boldsymbol{A}_m \in \mathcal{S}^n$, $\boldsymbol{c} \in \mathbb{R}^m$, and S is a non-empty closed convex subset of $\mathbb{R}^m$. Since $\boldsymbol{A} \in \mathcal{S}^n$ belongs to $\mathcal{COP}^n$ if and only if $\boldsymbol{\delta}^\top \boldsymbol{A}\boldsymbol{\delta} \geq 0$ for all $\boldsymbol{\delta} \in \Delta^{n-1}$, we can reformulate (4.1) in the form of (3.1). Specifically, the optimal value of (4.1) is equal to that of the following convex semi-infinite programming problem:

$$\begin{aligned} \underset{\boldsymbol{x}}{\text{minimize}} \quad & f(\boldsymbol{x}) \coloneqq \boldsymbol{c}^\top \boldsymbol{x} \\ \text{subject to} \quad & g(\boldsymbol{x}; \boldsymbol{\delta}) \coloneqq \boldsymbol{\delta}^\top \left(-\boldsymbol{A}_0 - \sum_{i=1}^m x_i \boldsymbol{A}_i \right) \boldsymbol{\delta} \leq 0 \text{ for all } \boldsymbol{\delta} \in \Delta^{n-1}, \\ & \boldsymbol{x} \in S. \end{aligned} \tag{4.2}$$

To implement Algorithm 1 in practice, we need to deal with the following issues. The first issue is to compute the value $L = \max\{L_f, L_g\}$. The second issue is to find $\boldsymbol{\delta} \in \Delta^{n-1}$ such that $G(\boldsymbol{x}) - g(\boldsymbol{x}; \boldsymbol{\delta}) \leq \alpha\epsilon$ for a given $\boldsymbol{x} \in S$, where $G(\boldsymbol{x})$ is defined analogously to (3.2).

First, we discuss how to estimate the value L. The functions f and g are differentiable with respect to $\boldsymbol{x}$, so we have

$$\partial f(\boldsymbol{x}) = \{\boldsymbol{c}\}, \tag{4.3}$$

$$\partial_{\boldsymbol{x}} g(\boldsymbol{x}; \boldsymbol{\delta}) = \{(-\boldsymbol{\delta}^\top \boldsymbol{A}_1 \boldsymbol{\delta}, \dots, -\boldsymbol{\delta}^\top \boldsymbol{A}_m \boldsymbol{\delta})^\top\} \tag{4.4}$$

for each $\boldsymbol{x} \in \mathbb{R}^m$ and $\boldsymbol{\delta} \in \Delta^{n-1}$. From (4.3), we can set L_f to $\|\boldsymbol{c}\|_2$. In addition, from

(4.4), we have

$$\|(-\boldsymbol{\delta}^\top \boldsymbol{A}_1\boldsymbol{\delta}, \ldots, -\boldsymbol{\delta}^\top \boldsymbol{A}_m\boldsymbol{\delta})^\top\|_2 = \sqrt{\sum_{i=1}^m \left(\sum_{k,l=1}^n (\boldsymbol{A}_i)_{kl}\delta_k\delta_l\right)^2}$$
$$\leq \sqrt{\sum_{i=1}^m \left(\max_{1\leq k\leq l\leq n} |(\boldsymbol{A}_i)_{kl}|\right)^2}, \tag{4.5}$$

where the last inequality follows from $\boldsymbol{\delta} \in \Delta^{n-1}$. The right-hand side of (4.5) does not depend on $\boldsymbol{\delta} \in \Delta^{n-1}$, so it can be set to L_g. Thus, we can set

$$L = \max\left\{\|\boldsymbol{c}\|_2, \sqrt{\sum_{i=1}^m \left(\max_{1\leq k\leq l\leq n} |(\boldsymbol{A}_i)_{kl}|\right)^2}\right\}. \tag{4.6}$$

Next, we discuss how to find $\boldsymbol{\delta} \in \Delta^{n-1}$ such that $G(\boldsymbol{x}) - g(\boldsymbol{x}; \boldsymbol{\delta}) \leq \alpha\epsilon$. We note that $G(\boldsymbol{x})$ is the negative of the optimal value of the standard quadratic programming problem

$$\gamma(\boldsymbol{Q}) \coloneqq \min_{\boldsymbol{\delta}\in\Delta^{n-1}} \boldsymbol{\delta}^\top \boldsymbol{Q}\boldsymbol{\delta} \tag{4.7}$$

with the coefficient matrix $\boldsymbol{Q} \in \mathcal{S}^n$ given by $\boldsymbol{A}_0 + \sum_{i=1}^m x_i\boldsymbol{A}_i$. In the following sub-subsections, we give an overview of several approaches to solving a standard quadratic programming problem of the form (4.7), both exactly (the case where the violation parameter α equals 0) and inexactly (the case where $\alpha > 0$). An exact method is presented in Section 4.1.1. Inexact methods are presented in Sections 4.1.2, 4.1.3, and 4.1.4. Whereas the exact method requires external solvers to solve a mixed-integer linear programming problem, only a finite number of function evaluations are performed in the inexact methods. In Sections 4.1.2, 4.1.3, and 4.1.4, we only consider the case where $\alpha = 1$. This is without loss of generality because for general $\alpha > 0$, we regard $\alpha\epsilon$ as a new ϵ.

We briefly summarize the three inexact methods introduced in the subsequent sub-subsections. The method presented in Section 4.1.2 is deterministic and was originally proposed as a polynomial-time approximation scheme by Bomze and de Klerk [9]. The method in Section 4.1.3 is a brute-force randomized approach. As shown in Section 5.1, this method has limited practical value; however, we include it to motivate the method in Section 4.1.4. The method in Section 4.1.4 is a randomized approach that combines the ideas of Sections 4.1.2 and 4.1.3. When n is large, this method is expected to require fewer function evaluations to approximate $\gamma(\boldsymbol{Q})$ than the method in Section 4.1.2.

In the following methods, we frequently use a lower bound for $\gamma(\boldsymbol{Q})$. The choice of the lower bound is arbitrary, and there are many alternatives that are summarized in the literature [12]. In our study, because we need to calculate the lower bound per iteration, it might be preferable to adopt a computationally inexpensive one. As such a bound, under the convention that $1/0 = \infty$, $a + \infty = \infty$ for any $a \in \mathbb{R} \cup \{\infty\}$, and $1/\infty = 0$, Bomze, Locatelli, and Tardella [12] provided the following lower bound for $\gamma(\boldsymbol{Q})$:

$$\underline{\gamma}(\boldsymbol{Q}) \coloneqq \min_{1 \le i \le j \le n} Q_{ij} + \frac{1}{\displaystyle\sum_{k=1}^{n} \frac{1}{Q_{kk} - \displaystyle\min_{1 \le i \le j \le n} Q_{ij}}}.$$

They showed that this bound outperforms other simple closed-form lower bounds reported in [12, Section 2].

Moreover, we also use a Lipschitz constant of the function $\boldsymbol{\delta}^\top \boldsymbol{Q} \boldsymbol{\delta}$ on Δ^{n-1}. An estimate of the constant is as follows:

$$K(\boldsymbol{Q}) \coloneqq \max_{\boldsymbol{\delta} \in \Delta^{n-1}} \|\nabla(\boldsymbol{\delta}^\top \boldsymbol{Q} \boldsymbol{\delta})\|_2 = 2 \max_{\boldsymbol{\delta} \in \Delta^{n-1}} \|\boldsymbol{Q}\boldsymbol{\delta}\|_2. \tag{4.8}$$

Since $\|\boldsymbol{Q}\boldsymbol{\delta}\|_2$ is convex and Δ^{n-1} is compact, the maximization problem in (4.8) attains its maximum at an extreme point of Δ^{n-1}, i.e., at one of $\boldsymbol{e}_1, \dots, \boldsymbol{e}_n$. Letting $\boldsymbol{q}_i$ be the ith column of the matrix $\boldsymbol{Q}$ for each $i \in [n]$, it follows from (4.8) that

$$K(\boldsymbol{Q}) = 2 \max_{1 \le i \le n} \|\boldsymbol{q}_i\|_2.$$

If $\boldsymbol{Q} = \boldsymbol{O}$, then the optimal value $\gamma(\boldsymbol{O})$ of (4.7) is 0, and the set of optimal solutions is Δ^{n-1}. Therefore, we may assume that $\boldsymbol{Q} \neq \boldsymbol{O}$. Under this assumption, it follows that $K(\boldsymbol{Q}) > 0$.

4.1.1 Exact method through mixed-integer linear programming

To solve the standard quadratic programming problem in (4.7) exactly, we can utilize the mixed-integer linear programming reformulations proposed by Gondzio and Yıldırım [23]. In their paper, various mixed-integer linear programming reformulations of standard quadratic programming problems are presented. Among them, we adopt the following reformulation based on [23, Proposition 2]:[*1]

$$
\begin{array}{lll}
\underset{\boldsymbol{\delta},\boldsymbol{y},\boldsymbol{z},v}{\text{minimize}} & v & \\
\text{subject to} & \boldsymbol{e}_j^\top \boldsymbol{Q}\boldsymbol{\delta} \le v + z_j \text{ for all } j \in [n], & \\
& \displaystyle\sum_{i=1}^{n} \delta_i = 1, & \\
& \delta_j \le y_j \text{ for all } j \in [n], & (4.9)\\
& z_j \le \left(\displaystyle\max_{1\le i\le n} Q_{ij} - \underline{\gamma}(\boldsymbol{Q})\right)(1-y_j) \text{ for all } j \in [n], & \\
& \boldsymbol{\delta} \in \mathbb{R}_+^n, & \\
& \boldsymbol{z} \in \mathbb{R}_+^n, & \\
& y_j \in \{0,1\} \text{ for all } j \in [n]. &
\end{array}
$$

For an optimal solution $(\boldsymbol{\delta}^*, \boldsymbol{y}^*, \boldsymbol{z}^*, v^*)$ of (4.9), the vector $\boldsymbol{\delta}^*$ represents an optimal solution of (4.7) and v^* represents its optimal value $\gamma(\boldsymbol{Q})$.

Note that finding an optimal solution of the standard quadratic programming problem in (4.7) is more demanding than merely testing copositivity of the matrix $\boldsymbol{Q}$. Anstreicher [3] proposed a mixed-integer linear programming formulation for testing copositivity of a symmetric matrix. However, this formulation is not directly connected to standard quadratic programming problems. Therefore, the mixed-integer programming formulation in (4.9) is more suitable for computing an optimal solution of (4.7).

4.1.2 Inexact deterministic method through a regular grid of the standard simplex

The first inexact method for solving the standard quadratic programming problem is to discretize the standard simplex Δ^{n-1} by the regular grid defined as

$$
\Delta_r^{n-1} := \{\boldsymbol{\delta} \in \Delta^{n-1} \mid \text{Every element of } r\boldsymbol{\delta} \text{ is a nonnegative integer}\} \tag{4.10}
$$

for a positive integer r. From [9, Theorem 3.2] (see also [42, Theorem 2]), we have

$$
\min_{\boldsymbol{\delta}\in\Delta_r^{n-1}} \boldsymbol{\delta}^\top \boldsymbol{Q}\boldsymbol{\delta} - \gamma(\boldsymbol{Q}) \le \frac{1}{r}\left(\max_{1\le i\le n} Q_{ii} - \underline{\gamma}(\boldsymbol{Q})\right). \tag{4.11}
$$

In other words, for a given $\epsilon > 0$, if r satisfies

$$
r \ge \frac{1}{\epsilon}\left(\max_{1\le i\le n} Q_{ii} - \underline{\gamma}(\boldsymbol{Q})\right), \tag{4.12}
$$

[*1]Using their terminology, we adopt formulation (MILP2) with the lower bound $\ell_1(\boldsymbol{Q})$. They claim that this reformulation is suitable for large-scale instances because of its robust practical performance [23, page 319].

it follows from (4.11) that

$$\min_{\boldsymbol{\delta}\in\Delta_r^{n-1}} \boldsymbol{\delta}^\top \boldsymbol{Q}\boldsymbol{\delta} - \gamma(\boldsymbol{Q}) \le \epsilon. \tag{4.13}$$

Since Δ_r^{n-1} is a finite set regardless of r, only a finite number of function evaluations is required to solve $\min_{\boldsymbol{\delta}\in\Delta_r^{n-1}} \boldsymbol{\delta}^\top \boldsymbol{Q}\boldsymbol{\delta}$. The number of evaluations is bounded by

$$|\Delta_r^{n-1}| = \binom{n+r-1}{r} \le n^r,$$

which is polynomial in n if we regard r as a constant. If the lower bound $\underline{\gamma}(\boldsymbol{Q})$ is weak, the number $|\Delta_r^{n-1}|$ of evaluations can become prohibitively large. In such cases, rather than using the closed-form bound, it may be preferable to use the bounds obtained by solving semidefinite programming problems as presented in [12].

4.1.3 Inexact randomized method through uniform sampling from the standard simplex

The second inexact method for solving the standard quadratic programming problem is to discretize the standard simplex Δ^{n-1} by uniformly sampling from it. Let $\boldsymbol{\delta}_1, \dots, \boldsymbol{\delta}_M$ be a collection of independent and identically distributed random vectors uniformly distributed on Δ^{n-1}. In Proposition 4.2 shown below, we provide a lower bound for the number M of samples sufficient to ensure that

$$\min_{1\le i\le M} \boldsymbol{\delta}_i^\top \boldsymbol{Q}\boldsymbol{\delta}_i - \gamma(\boldsymbol{Q}) \le \epsilon \tag{4.14}$$

holds with probability at least $1-\phi$ for a given $\phi \in (0,1)$. Throughout this subsubsection, P denotes the uniform distribution on Δ^{n-1}.

Lemma 4.1. *Let $r \in (0, \sqrt{2}]$. For any $\boldsymbol{x} \in \Delta^{n-1}$, we have*

$$P(B(\boldsymbol{x}, r) \cap \Delta^{n-1}) \ge \left(\frac{r}{\sqrt{2}}\right)^{n-1}.$$

Proof. Let $\mathrm{vol}(C)$ denote the $(n-1)$-dimensional Lebesgue measure of an $(n-1)$-dimensional set C in $\mathbb{R}^n$. Then we have

$$p(\boldsymbol{x}) := P(B(\boldsymbol{x}, r) \cap \Delta^{n-1}) = \frac{\mathrm{vol}(B(\boldsymbol{x}, r) \cap \Delta^{n-1})}{\mathrm{vol}(\Delta^{n-1})}.$$

First, we show that

$$\min_{\boldsymbol{x}\in\Delta^{n-1}} p(\boldsymbol{x}) = p(\boldsymbol{e}_1). \tag{4.15}$$

Since $p(\boldsymbol{x})$ is nonnegative, we have

$$\arg\min_{\boldsymbol{x}\in\Delta^{n-1}} p(\boldsymbol{x}) = \arg\min_{\boldsymbol{x}\in\Delta^{n-1}} p(\boldsymbol{x})^{n-1} = \arg\min_{\boldsymbol{x}\in\Delta^{n-1}} \operatorname{vol}(B(\boldsymbol{x}, r)\cap\Delta^{n-1})^{n-1}.$$

By [30, Lemma 6.21], the function $\operatorname{vol}(B(\boldsymbol{x}, r)\cap\Delta^{n-1})^{n-1}$ with respect to $\boldsymbol{x}\in\Delta^{n-1}$ is concave, so it achieves the minimum at an extreme point of Δ^{n-1}. By the symmetry of Δ^{n-1}, it does so at $\boldsymbol{x} = \boldsymbol{e}_1$.

For each $i\in\{2,\dots,n\}$, we define

$$\boldsymbol{e}_i(r) \coloneqq \left(1-\frac{r}{\sqrt{2}}\right)\boldsymbol{e}_1 + \frac{r}{\sqrt{2}}\boldsymbol{e}_i.$$

The convex hull of the set $\{\boldsymbol{e}_1, \boldsymbol{e}_2(r), \dots, \boldsymbol{e}_n(r)\}$, denoted by $\Delta^{n-1}(r)$, is an $(n-1)$-dimensional simplex with edge length r. Since $\boldsymbol{e}_1, \boldsymbol{e}_2(r), \dots, \boldsymbol{e}_n(r) \in B(\boldsymbol{e}_1, r)\cap\Delta^{n-1}$ and $B(\boldsymbol{e}_1, r)\cap\Delta^{n-1}$ is convex, the inclusion

$$\Delta^{n-1}(r)\subseteq B(\boldsymbol{e}_1, r)\cap\Delta^{n-1} \tag{4.16}$$

holds. Therefore,

$$p(\boldsymbol{x}) \geq p(\boldsymbol{e}_1) = \frac{\operatorname{vol}(B(\boldsymbol{e}_1, r)\cap\Delta^{n-1})}{\operatorname{vol}(\Delta^{n-1})} \geq \frac{\operatorname{vol}(\Delta^{n-1}(r))}{\operatorname{vol}(\Delta^{n-1})} = \left(\frac{r}{\sqrt{2}}\right)^{n-1},$$

holds for any $\boldsymbol{x}\in\Delta^{n-1}$, where the first inequality follows from (4.15), the second inequality follows from (4.16), and the last equality follows from the fact that the volume of an $(n-1)$-dimensional simplex with edge length l is given by

$$\frac{\sqrt{n}}{(n-1)!}\left(\frac{l}{\sqrt{2}}\right)^{n-1},$$

so we obtain the desired result. □

Below, we present the main result of this subsubsection. For notational convenience, we define

$$m(\rho,\phi) \coloneqq \frac{\log\phi}{\log(1-\rho)} \tag{4.17}$$

for $\rho,\phi\in(0,1)$.

Proposition 4.2. *Let $\boldsymbol{\delta}_1,\dots,\boldsymbol{\delta}_M$ be a collection of independent and identically distributed random vectors uniformly distributed on Δ^{n-1}. In addition, let m denote the function defined in (4.17). For $\epsilon\in(0,\sqrt{2}K(\boldsymbol{Q})]$ and $\phi\in(0,1)$, if the number M of samples satisfies*

$$M \geq m\left(\left(\frac{\epsilon}{\sqrt{2}K(\boldsymbol{Q})}\right)^{n-1}, \phi\right), \tag{4.18}$$

then, with probability at least $1-\phi$, we have

$$\min_{1\leq i\leq M} \boldsymbol{\delta}_i^\top\boldsymbol{Q}\boldsymbol{\delta}_i - \gamma(\boldsymbol{Q}) \leq \epsilon.$$

Proof. Let $\boldsymbol{\delta}^*$ be an optimal solution of $\min_{\boldsymbol{\delta}\in\Delta^{n-1}} \boldsymbol{\delta}^\top \boldsymbol{Q}\boldsymbol{\delta}$. Then the probability that $\boldsymbol{\delta}_i \notin B(\boldsymbol{\delta}^*, \epsilon/K(\boldsymbol{Q}))$ holds for all $i \in [M]$ is $\prod_{i=1}^M P(\boldsymbol{\delta}_i \notin B(\boldsymbol{\delta}^*, \epsilon/K(\boldsymbol{Q})))$ and it is bounded by

$$\begin{aligned}\prod_{i=1}^M P\left(\boldsymbol{\delta}_i \notin B\left(\boldsymbol{\delta}^*, \frac{\epsilon}{K(\boldsymbol{Q})}\right)\right) &= \prod_{i=1}^M \left\{1 - P\left(\boldsymbol{\delta}_i \in B\left(\boldsymbol{\delta}^*, \frac{\epsilon}{K(\boldsymbol{Q})}\right)\right)\right\}\\ &\leq \left\{1 - \left(\frac{\epsilon}{\sqrt{2}K(\boldsymbol{Q})}\right)^{n-1}\right\}^M\\ &\leq \phi,\end{aligned}$$

where we use Lemma 4.1 to derive the first inequality and the second inequality follows from (4.18). This implies that with probability at least $1-\phi$, there exists $j \in [M]$ such that $\boldsymbol{\delta}_j \in B(\boldsymbol{\delta}^*, \epsilon/K(\boldsymbol{Q}))$. Then we have

$$\min_{1\leq i\leq M} \boldsymbol{\delta}_i^\top \boldsymbol{Q}\boldsymbol{\delta}_i - (\boldsymbol{\delta}^*)^\top \boldsymbol{Q}\boldsymbol{\delta}^* \leq \boldsymbol{\delta}_j^\top \boldsymbol{Q}\boldsymbol{\delta}_j - (\boldsymbol{\delta}^*)^\top \boldsymbol{Q}\boldsymbol{\delta}^* \leq K(\boldsymbol{Q})\|\boldsymbol{\delta}_j - \boldsymbol{\delta}^*\|_2 \leq \epsilon,$$

so we obtain the desired result. □

Remark 4.3. By using [56, Proposition 4.3], we can obtain a result similar to Proposition 4.2. Let $\bar{\gamma}(\boldsymbol{Q})$ be such that $\max_{\boldsymbol{\delta}\in\Delta^{n-1}} \boldsymbol{\delta}^\top\boldsymbol{Q}\boldsymbol{\delta} \leq \bar{\gamma}(\boldsymbol{Q})$ and $\underline{\gamma}(\boldsymbol{Q}) < \bar{\gamma}(\boldsymbol{Q})$ hold, and let $\boldsymbol{\delta}_1, \ldots, \boldsymbol{\delta}_M$ be a collection of independent and identically distributed random vectors uniformly distributed on Δ^{n-1}. By applying [56, Proposition 4.3] to the standard quadratic programming problem $\gamma(\boldsymbol{Q})$, we see that for a given $\epsilon > 0$, if the number M of samples satisfies

$$M \geq m\left(\left(\frac{\epsilon}{2\sqrt{2}K(\boldsymbol{Q})}\right)^{n-1}, \frac{\epsilon}{2(\bar{\gamma}(\boldsymbol{Q}) - \underline{\gamma}(\boldsymbol{Q}))}\right), \tag{4.19}$$

the expected value of $\min_{1\leq i\leq M} \boldsymbol{\delta}_i^\top\boldsymbol{Q}\boldsymbol{\delta}_i$ is less than or equal to $\gamma(\boldsymbol{Q}) + \epsilon$.

We note that the number of samples shown in (4.18) and that in (4.19) are exponential in n. If ρ is sufficiently small, then we have

$$m(\rho, \phi) \simeq \frac{1}{\rho}\log\left(\frac{1}{\phi}\right). \tag{4.20}$$

The error tolerance ϵ is assumed to be small in practice, thus we obtain

$$\begin{aligned}(4.18) &\simeq \left(\frac{\sqrt{2}K(\boldsymbol{Q})}{\epsilon}\right)^{n-1}\log\left(\frac{1}{\phi}\right), \qquad (4.21)\\ (4.19) &\simeq \left(\frac{2\sqrt{2}K(\boldsymbol{Q})}{\epsilon}\right)^{n-1}\log\left(\frac{2(\bar{\gamma}(\boldsymbol{Q}) - \underline{\gamma}(\boldsymbol{Q}))}{\epsilon}\right).\end{aligned}$$

4.1.4 Inexact randomized method through uniform sampling from a regular grid of the standard simplex

Generally, randomized approaches are used when the scale of problems is large and deterministic approaches cannot handle them. However, we see from Proposition 4.2 and (4.21) that the randomized method in Section 4.1.3 requires an exponential number of samples with respect to n to obtain an ϵ-approximate solution. In this subsubsection, we discretize the standard simplex by uniformly sampling from its regular grid rather than from the standard simplex. In Proposition 4.5 shown below, we provide a probabilistic error bound on the optimal value, in which the number of function evaluations is expected to be less than the number of points in the regular grid if n is large. Recall that a regular grid Δ_r^{n-1} of the standard simplex Δ^{n-1} is defined as (4.10). For simplicity, for $\boldsymbol{\delta} \in \Delta_r^{n-1}$, we define

$$G_r(\boldsymbol{\delta}) \coloneqq B\left(\boldsymbol{\delta}, \frac{\sqrt{2}}{r}\right) \cap \Delta_r^{n-1}.$$

Lemma 4.4. *For any $\boldsymbol{\delta} \in \Delta_r^{n-1}$, we have $|G_r(\boldsymbol{\delta})| \geq n$. In particular, the equality holds if $\boldsymbol{\delta}$ is any of $\boldsymbol{e}_1, \ldots, \boldsymbol{e}_n$.*

Proof. The vector $\boldsymbol{\delta}$ has at least one positive element. In addition, every element of $\boldsymbol{\delta}$ is one of $0, 1/r, 2/r, \ldots, 1-1/r, 1$. Therefore, there exists $i \in [n]$ such that $\delta_i \geq 1/r$. We prove $|G_r(\boldsymbol{\delta})| \geq n$ by showing that

$$\left\{\boldsymbol{\delta} - \frac{1}{r}\boldsymbol{e}_i + \frac{1}{r}\boldsymbol{e}_j \;\middle|\; j \in [n]\right\} \subseteq G_r(\boldsymbol{\delta}). \tag{4.22}$$

For every $j \in [n]$, we define $\boldsymbol{\delta}(j) \coloneqq \boldsymbol{\delta} - \frac{1}{r}\boldsymbol{e}_i + \frac{1}{r}\boldsymbol{e}_j \in \Delta_r^{n-1}$. When $j = i$, we have $\boldsymbol{\delta}(i) = \boldsymbol{\delta}$, so $\boldsymbol{\delta}(i) \in G_r(\boldsymbol{\delta})$ holds. When $j \neq i$, we have $\|\boldsymbol{\delta}(j) - \boldsymbol{\delta}\|_2 = \sqrt{2}/r$, so $\boldsymbol{\delta}(j) \in G_r(\boldsymbol{\delta})$ holds. Thus, we obtain $|G_r(\boldsymbol{\delta})| \geq n$.

In particular, the converse inclusion in (4.22) holds when $\boldsymbol{\delta} = \boldsymbol{e}_1$. Note that the index i that satisfies $\delta_i \geq 1/r$ has to be 1, and $\boldsymbol{\delta}(j) = (1 - \frac{1}{r})\boldsymbol{e}_1 + \frac{1}{r}\boldsymbol{e}_j$ holds. Let $\hat{\boldsymbol{\delta}} \in G_r(\boldsymbol{e}_1)$. If $\hat{\delta}_1 \leq 1 - 2/r$, we have

$$\|\hat{\boldsymbol{\delta}} - \boldsymbol{e}_1\|_2 \geq |\hat{\delta}_1 - 1| \geq \frac{2}{r} > \frac{\sqrt{2}}{r},$$

which does not happen since $\hat{\boldsymbol{\delta}} \in B(\boldsymbol{e}_1, \sqrt{2}/r)$. If $\hat{\delta}_1 = 1 - 1/r$, there exists $j \in \{2, \ldots, n\}$ such that $\hat{\delta}_j = 1/r$ and $\hat{\delta}_k = 0$ for all $k \in \{2, \ldots, n\} \setminus \{j\}$. Then we have $\hat{\boldsymbol{\delta}} = \boldsymbol{\delta}(j)$. If $\hat{\delta}_1 = 1$, it follows that $\hat{\boldsymbol{\delta}} = \boldsymbol{e}_1$, so $\hat{\boldsymbol{\delta}} = \boldsymbol{\delta}(1)$ holds. Therefore, the set $G_r(\boldsymbol{e}_1)$ does not contain the elements other than $\boldsymbol{\delta}(1), \ldots, \boldsymbol{\delta}(n)$ and we obtain $|G_r(\boldsymbol{e}_1)| = n$. By symmetry, $|G_r(\boldsymbol{\delta})| = n$ also holds when $\boldsymbol{\delta}$ is any of $\boldsymbol{e}_1, \ldots, \boldsymbol{e}_n$. □

Proposition 4.5. *For a positive integer r, let $\boldsymbol{\delta}_1, \ldots, \boldsymbol{\delta}_M$ be a collection of independent and identically distributed random vectors uniformly distributed on Δ_r^{n-1}. For $\phi \in (0,1)$, if the number M of samples satisfies*

$$M \geq m\left(\frac{n}{|\Delta_r^{n-1}|}, \phi\right), \tag{4.23}$$

then, with probability at least $1-\phi$, we have

$$\min_{1\leq i\leq M} \boldsymbol{\delta}_i^\top \boldsymbol{Q}\boldsymbol{\delta}_i - \gamma(\boldsymbol{Q}) \leq \frac{1}{r}\left(\sqrt{2}K(\boldsymbol{Q}) + \max_{1\leq i\leq n} Q_{ii} - \underline{\gamma}(\boldsymbol{Q})\right). \tag{4.24}$$

In particular, for a given $\epsilon > 0$, if r satisfies

$$r \geq \frac{1}{\epsilon}\left(\max_{1\leq i\leq n} Q_{ii} - \underline{\gamma}(\boldsymbol{Q})\right), \tag{4.25}$$

then, with probability at least $1-\phi$, we have

$$\min_{1\leq i\leq M} \boldsymbol{\delta}_i^\top \boldsymbol{Q}\boldsymbol{\delta}_i - \gamma(\boldsymbol{Q}) \leq \left(\frac{\sqrt{2}K(\boldsymbol{Q})}{\max_{1\leq i\leq n} Q_{ii} - \underline{\gamma}(\boldsymbol{Q})} + 1\right)\alpha\epsilon.$$

Proof. Let $\boldsymbol{\delta}^*$ be an optimal solution of $\min_{\boldsymbol{\delta}\in\Delta_r^{n-1}} \boldsymbol{\delta}^\top \boldsymbol{Q}\boldsymbol{\delta}$. Then the probability that $\boldsymbol{\delta}_i \notin B(\boldsymbol{\delta}^*, \sqrt{2}/r)$ holds for all $i \in [M]$ is $(1 - |G_r(\boldsymbol{\delta}^*)|/|\Delta_r^{n-1}|)^M$ and it is bounded by

$$\left(1 - \frac{|G_r(\boldsymbol{\delta}^*)|}{|\Delta_r^{n-1}|}\right)^M \leq \left(1 - \frac{n}{|\Delta_r^{n-1}|}\right)^M \leq \phi,$$

where we use Lemma 4.4 to derive the first inequality and the second inequality follows from (4.23). This implies that with probability at least $1-\phi$, there exists $j \in [M]$ such that $\boldsymbol{\delta}_j \in G_r(\boldsymbol{\delta}^*)$. Then we have

$$\begin{aligned}\min_{1\leq i\leq M} \boldsymbol{\delta}_i^\top \boldsymbol{Q}\boldsymbol{\delta}_i - \gamma(\boldsymbol{Q}) &\leq (\boldsymbol{\delta}_j^\top \boldsymbol{Q}\boldsymbol{\delta}_j - (\boldsymbol{\delta}^*)^\top \boldsymbol{Q}\boldsymbol{\delta}^*) + ((\boldsymbol{\delta}^*)^\top \boldsymbol{Q}\boldsymbol{\delta}^* - \gamma(\boldsymbol{Q})) \\ &\leq K(\boldsymbol{Q})\|\boldsymbol{\delta}_j - \boldsymbol{\delta}^*\|_2 + \frac{1}{r}\left(\max_{1\leq i\leq n} Q_{ii} - \underline{\gamma}(\boldsymbol{Q})\right) \\ &\leq \frac{1}{r}\left(\sqrt{2}K(\boldsymbol{Q}) + \max_{1\leq i\leq n} Q_{ii} - \underline{\gamma}(\boldsymbol{Q})\right),\end{aligned}$$

where we use (4.11) to derive the second inequality and we use $\boldsymbol{\delta}_j \in G_r(\boldsymbol{\delta}^*)$ to derive the third inequality. □

Proposition 4.5 suggests the possibility that we can compute an approximate value of $\gamma(\boldsymbol{Q})$ with a smaller number of function evaluations than that required by the method in Section 4.1.2. Using the approximation shown in (4.20), we see that

$$(4.23) \simeq \frac{1}{n} \log\left(\frac{1}{\phi}\right) |\Delta_r^{n-1}|.$$

The value $\frac{1}{n}\log(\frac{1}{\phi})$ is expected to be less than 1 if n is large. In such a case, with a smaller number of function evaluations than that in Section 4.1.2, we can derive an approximation of $\gamma(\boldsymbol{Q})$ that enjoys the theoretical guarantee shown in (4.24), although this bound is weaker than (4.11).

4.2 Testing complete positivity of a matrix

A matrix $\boldsymbol{A} \in \mathcal{S}^n$ is said to be *completely positive* if there exist a positive integer k and $\boldsymbol{a}_1, \ldots, \boldsymbol{a}_k \in \mathbb{R}^n_+$ such that $\boldsymbol{A} = \sum_{i=1}^k \boldsymbol{a}_i \boldsymbol{a}_i^\top$. For a matrix $\boldsymbol{C} \in \mathcal{S}^n$, the problem of testing the complete positivity of $\boldsymbol{C}$ can be formulated as the following copositive programming problem with a ball constraint:

$$\begin{aligned} \underset{\boldsymbol{X}}{\text{minimize}} \quad & \langle \boldsymbol{C}, \boldsymbol{X} \rangle \\ \text{subject to} \quad & \boldsymbol{X} \in \mathcal{COP}^n, \\ & \boldsymbol{X} \in B(\boldsymbol{O}, 1). \end{aligned} \tag{4.26}$$

Without loss of generality, we may assume that $\|\boldsymbol{C}\|_{\mathrm{F}} = 1$. Since $\boldsymbol{X} = \boldsymbol{O}$ is a feasible solution of this problem, its optimal value is at most 0. In addition, by the duality between completely positive cones and copositive cones [27, Theorem 2.1], the matrix $\boldsymbol{C}$ is completely positive if and only if the optimal value is 0.

We can reformulate (4.26) in the form of (3.1), namely,

$$\begin{aligned} \underset{\boldsymbol{X}}{\text{minimize}} \quad & f(\boldsymbol{X}) := \langle \boldsymbol{C}, \boldsymbol{X} \rangle \\ \text{subject to} \quad & g(\boldsymbol{X}; \boldsymbol{\delta}) := \langle \boldsymbol{X}, -\boldsymbol{\delta}\boldsymbol{\delta}^\top \rangle \leq 0 \text{ for all } \boldsymbol{\delta} \in \Delta^{n-1}, \\ & \boldsymbol{X} \in B(\boldsymbol{O}, 1). \end{aligned} \tag{4.27}$$

Therefore, we can directly apply Algorithm 1 developed in Section 3 to Problem (4.27). As in Section 4.1, the subproblem $G(\boldsymbol{X}) := \max_{\boldsymbol{\delta} \in \Delta^{n-1}} g(\boldsymbol{X}; \boldsymbol{\delta})$ that needs to be solved at each iteration is essentially the standard quadratic programming problem with coefficient matrix $\boldsymbol{X}$. The calculation of the constant L_f is the same as that in Section 4.1; we can set $L_f = 1$ under the assumption that $\|\boldsymbol{C}\|_{\mathrm{F}} = 1$. In addition, we can set $L_g = 1$ since $\partial_{\boldsymbol{X}} g(\boldsymbol{X}; \boldsymbol{\delta}) = \{-\boldsymbol{\delta}\boldsymbol{\delta}^\top\}$ and

$$\|-\boldsymbol{\delta}\boldsymbol{\delta}^\top\|_{\mathrm{F}} = \|\boldsymbol{\delta}\|_2^2 \leq \left(\sum_{i=1}^n \delta_i\right)^2 = 1$$

hold for any $\boldsymbol{\delta} \in \Delta^{n-1}$. Then $L \coloneqq \max\{L_f, L_g\}$ equals 1 and Theorem 3.4 can be translated into the following proposition.

Proposition 4.6. *We apply Algorithm 1 to (4.27), in which we set the initial point $\boldsymbol{X}_1$ to $\boldsymbol{O}$. Let N be a positive integer such that*

$$N \geq \frac{1}{\epsilon^2}. \tag{4.28}$$

For each iteration indexed by k, we find $\boldsymbol{\delta}_k \in \Delta^{n-1}$ such that $G(\boldsymbol{X}_k) - g(\boldsymbol{X}_k; \boldsymbol{\delta}_k) \leq \alpha\epsilon$. Then the set

$$I_N \coloneqq \{k \in [N] \mid g(\boldsymbol{X}_k; \boldsymbol{\delta}_k) \leq \epsilon\} \tag{4.29}$$

is non-empty and for a minimizer k^* of $f(\boldsymbol{X}_k)$ over $k \in I_N$, the following two inequalities hold:

$$\begin{aligned} f(\boldsymbol{X}_{k^*}) &\leq f^* + \epsilon, \\ G(\boldsymbol{X}_{k^*}) &\leq (1+\alpha)\epsilon. \end{aligned}$$

Proof. Since $\boldsymbol{X}_1 = \boldsymbol{O}$ and the set S^* of optimal solutions of Problem (4.27) is included in $B(\boldsymbol{O}, 1)$, we have

$$\operatorname{dist}(\boldsymbol{X}_1, S^*) \leq 1. \tag{4.30}$$

Substituting $L = 1$ and (4.30) into (3.4), Theorem 3.4 leads to the desired result. □

Remark 4.7. By vectorizing the matrices in (4.26), we can reformulate Problem (4.26) as a problem of the form (4.2), to which the discussion in Section 4.1 applies. Specifically, letting $\boldsymbol{E}_i \coloneqq \boldsymbol{e}_i\boldsymbol{e}_i^\top$ for each $i = 1, \ldots, n$ and $\boldsymbol{E}_{ij} \coloneqq (\boldsymbol{e}_i\boldsymbol{e}_j^\top + \boldsymbol{e}_j\boldsymbol{e}_i^\top)/\sqrt{2}$ for each $1 \leq i < j \leq n$ and introducing variables

$$\boldsymbol{x} = (x_{11}, x_{12}, x_{22}, \ldots, x_{1n}, \ldots, x_{nn})^\top \in \mathbb{R}^{\frac{n(n+1)}{2}},$$

we can reformulate Problem (4.26) as

$$\begin{aligned} \underset{\boldsymbol{x}}{\text{minimize}} \quad & \tilde{f}(\boldsymbol{x}) \coloneqq \sum_{i=1}^{n} C_{ii}x_{ii} + \sum_{1\leq i<j\leq n} \sqrt{2}C_{ij}x_{ij} \\ \text{subject to} \quad & \tilde{g}(\boldsymbol{x}; \boldsymbol{\delta}) \coloneqq \boldsymbol{\delta}^\top \left(-\sum_{i=1}^{n} x_{ii}\boldsymbol{E}_i - \sum_{1\leq i<j\leq n} x_{ij}\boldsymbol{E}_{ij} \right) \boldsymbol{\delta} \leq 0 \text{ for all } \boldsymbol{\delta} \in \Delta^{n-1}, \\ & \boldsymbol{x} \in B(\boldsymbol{0}, 1). \end{aligned} \tag{4.31}$$

However, the constant $L \coloneqq \max\{L_{\tilde{f}}, L_{\tilde{g}}\}$ calculated as (4.6) depends on n unlike the formulation in (4.27). The coefficient vector, denoted by $\boldsymbol{c} \in \mathbb{R}^{n(n+1)/2}$, corresponding to the objective function in (4.31) satisfies $\|\boldsymbol{c}\|_2 = \|\boldsymbol{C}\|_{\mathrm{F}} = 1$, so we have $L_{\tilde{f}} = 1$. In addition, by the discussion in Section 4.1, the constant $L_{\tilde{g}}$ can be set to

$$L_{\tilde{g}} = \sqrt{\sum_{i=1}^{n} \left(\max_{1 \le k \le l \le n} |(\boldsymbol{E}_i)_{kl}| \right)^2 + \sum_{1 \le i < j \le n} \left(\max_{1 \le k \le l \le n} |(\boldsymbol{E}_{ij})_{kl}| \right)^2} = \frac{\sqrt{n^2 + 3n}}{2}.$$

Therefore, we have $L = \sqrt{n^2 + 3n}/2$ and the iteration-complexity bound shown in (3.4) increases as n rises.

Since the approximate solution $\boldsymbol{X}_{k^*}$ may be an infeasible solution of Problem (4.26), the negativity of the approximate optimal value $\langle \boldsymbol{C}, \boldsymbol{X}_{k^*} \rangle$ does not necessarily mean that $\boldsymbol{C}$ is not completely positive. However, as shown in the following theorem, if the approximate optimal value is less than a threshold, $\boldsymbol{C}$ is guaranteed not to be completely positive.

Theorem 4.8. *We apply Algorithm 1 to (4.27), in which we set the initial point $\boldsymbol{X}_1$ to $\boldsymbol{O}$. Let N be a positive integer satisfying (4.28). For each iteration indexed by k, we find $\boldsymbol{\delta}_k \in \Delta^{n-1}$ such that $G(\boldsymbol{X}_k) - g(\boldsymbol{X}_k; \boldsymbol{\delta}_k) \le \alpha\epsilon$. After N iterations of the algorithm, we let $k^* \in \arg\min\{\langle \boldsymbol{C}, \boldsymbol{X}_k \rangle \mid k \in I_N\}$, where I_N is defined as (4.29). If the inequality*

$$\langle \boldsymbol{C}, \boldsymbol{X}_{k^*} \rangle < -n(1+\alpha)\epsilon \tag{4.32}$$

holds, $\boldsymbol{C}$ is not completely positive.

Proof. By Proposition 4.6, the inequality

$$G(\boldsymbol{X}_{k^*}) \le (1+\alpha)\epsilon \tag{4.33}$$

holds. Let $\bar{\boldsymbol{X}} := P_{\mathcal{COP}^n}(\boldsymbol{X}_{k^*})$. Then it follows that

$$\|\boldsymbol{X}_{k^*} - \bar{\boldsymbol{X}}\|_{\mathrm{F}} \le n \max\{G(\boldsymbol{X}_{k^*}), 0\} \le n(1+\alpha)\epsilon,$$

where we use [28, Equation (6.16)] to derive the first inequality and we use (4.33) to derive the second inequality. Using this inequality, the Cauchy–Schwarz inequality, and $\|\boldsymbol{C}\|_{\mathrm{F}} = 1$, we have

$$|\langle \boldsymbol{C}, \boldsymbol{X}_{k^*} \rangle - \langle \boldsymbol{C}, \bar{\boldsymbol{X}} \rangle| \le n(1+\alpha)\epsilon.$$

Combining this with the assumption in (4.32), we see that

$$\langle \boldsymbol{C}, \bar{\boldsymbol{X}} \rangle \le \langle \boldsymbol{C}, \boldsymbol{X}_{k^*} \rangle + n(1+\alpha)\epsilon < 0.$$

Since $\bar{\boldsymbol{X}} \in \mathcal{COP}^n$, this inequality implies that $\boldsymbol{C}$ is not completely positive. $\square$

Remark 4.9. We note that the inequality $\langle \boldsymbol{C}, \boldsymbol{X}_{k^*} \rangle \geq -1$ holds by the Cauchy–Schwarz inequality, $\|\boldsymbol{C}\|_{\mathrm{F}} = 1$, and the constraint $\boldsymbol{X}_{k^*} \in B(\boldsymbol{O}, 1)$. This implies that ϵ must satisfy $\epsilon < \frac{1}{n(1+\alpha)}$ to use the condition in (4.32). For example, we can set

$$\epsilon := \frac{1}{tn(1+\alpha)}$$

for a constant $t > 1$, in which case only

$$t^2(1+\alpha)^2 n^2 = O(n^2) \tag{4.34}$$

iterations are necessary to test non-complete positivity by using Theorem 4.8. Note, however, that t may need to be chosen large in order to detect non-complete positivity, in which case the number of iterations in (4.34) also becomes large. We revisit the choice of t in Section 5.3.

Remark 4.10. The number of iterations shown in (4.28) is necessary to guarantee that the set I_N is non-empty. However, in running the algorithm, we may find a matrix $\boldsymbol{X}_k$ satisfying $g(\boldsymbol{X}_k; \boldsymbol{\delta}_k) \leq \epsilon$ even if $k < 1/\epsilon^2$. If we obtain a matrix $\boldsymbol{X}_k$ such that $g(\boldsymbol{X}_k; \boldsymbol{\delta}_k) \leq \epsilon$ and $\langle \boldsymbol{C}, \boldsymbol{X}_k \rangle < -n(1+\alpha)\epsilon$, the matrix $\boldsymbol{X}_k$ provides a certificate that $\boldsymbol{C}$ is not completely positive.

4.3 The case of symmetric cones beyond nonnegative orthants

In the previous subsections, we discussed how to solve copositive programming problems constrained by the copositive cone $\mathcal{COP}^n$. Recently, studies have been conducted on copositive cones determined by *symmetric cones* (self-dual and homogeneous cones) beyond nonnegative orthants, and on their associated copositive programming problems [46–50]. Let $\mathbb{K}$ be a symmetric cone in a finite-dimensional real vector space $\mathbb{V}$ equipped with an inner product denoted by $\bullet$. *The copositive cone over the symmetric cone* $\mathbb{K}$, denoted by $\mathcal{COP}(\mathbb{K})$, is the cone of self-adjoint linear transformations A on $\mathbb{V}$ such that $x \bullet A(x) \geq 0$ for all $x \in \mathbb{K}$. If $\mathbb{K}$ is the nonnegative orthant $\mathbb{R}^n_+$, the cone $\mathcal{COP}(\mathbb{R}^n_+)$ can be identified with $\mathcal{COP}^n$.

In principle, we can use Algorithm 1 to solve copositive programming problems involving a copositive cone over a symmetric cone $\mathbb{K}$. We let e be an interior point in $\mathbb{K}$ and define

$$\Delta_{\mathbb{K},e} := \{\delta \in \mathbb{K} \mid e \bullet \delta = 1\}.$$

The set $\Delta_{\mathbb{K},e}$ can be regarded as a generalization of the standard simplex; indeed, by setting $\mathbb{K}$ to $\mathbb{R}^n_+$ and e to the vector with all elements 1, we have $\Delta_{\mathbb{K},e} = \Delta^{n-1}$. Generally, $\Delta_{\mathbb{K},e}$ is a compact slice of $\mathbb{K}$, and $A \in \mathcal{COP}(\mathbb{K})$ if and only if $\delta \bullet A(\delta) \geq 0$ for all $\delta \in \Delta_{\mathbb{K},e}$. The subproblem that we need to solve at each iteration of Algorithm 1 is essentially

$$\min_{\delta \in \Delta_{\mathbb{K},e}} \delta \bullet Q(\delta), \tag{4.35}$$

where Q is a self-adjoint linear transformation on $\mathbb{V}$. Problem (4.35) corresponds to the standard quadratic programming problem in (4.7). Whereas we can utilize the mixed-integer linear programming reformulation to solve standard quadratic programming problems globally, it is unknown how to obtain a global solution to (4.35).

In what follows, we explore a potential approach to deriving a global solution to Problem (4.35). It follows from [19, Section 3] that the optimal value of (4.35) is equal to that of the following copositive programming problem with a scalar variable:

$$\max_{\lambda \in \mathbb{R}} \{\lambda \mid Q - \lambda e \otimes e \in \mathcal{COP}(\mathbb{K})\}.$$

An optimal solution of this problem, denoted by λ^*, can be found by combining membership testing for $\mathcal{COP}(\mathbb{K})$ with the bisection method. Despite the absence of established numerical methods for the membership problem of $\mathcal{COP}(\mathbb{K})$, Orlitzky [50] explores a potential extension of a recursive method to solve the membership problem for $\mathcal{COP}(\mathbb{R}^n_+)$ to that for $\mathcal{COP}(\mathbb{K})$. Since $Q - \lambda^* e \otimes e$ lies in the boundary of $\mathcal{COP}(\mathbb{K})$, there exists $\delta^* \in \Delta_{\mathbb{K},e}$ such that

$$0 = \delta^* \bullet (Q - \lambda^* e \otimes e)(\delta^*) = \delta^* \bullet Q(\delta^*) - \lambda^*,$$

i.e., $\delta^* \bullet Q(\delta^*) = \lambda^*$. Therefore, δ^* is an optimal solution of Problem (4.35).

5 Numerical experiments

In this section, we verify the effectiveness of the methods introduced so far through numerical experiments. As outlined in Section 4.1, four distinct methods for solving standard quadratic programming problems were presented. We compare these approaches in Section 5.1. In Section 5.2, we compare the performance of these methods when they are incorporated into Algorithm 1 for solving copositive programming problems. In Section 5.3, we demonstrate that the approach introduced in Section 4.2 is effective in detecting non-complete positivity of a matrix.

All experiments were conducted in MATLAB (R2025a) on a computer with an Apple M1 and 16 GB memory. The Gurobi solver [25] (version 12.0.2) was used to solve mixed-integer linear programming problems.

5.1 Comparison among the methods for solving standard quadratic programming problems

We introduced four methods to solve standard quadratic programming problems in Sections 4.1.1, 4.1.2, 4.1.3, and 4.1.4. In this subsection, we compare them numerically in terms of solution accuracy and computational time. The standard quadratic programming problem solved here is of the form (4.7). For each $n \in \{5, 10, 50, 100, 500, 1000\}$, we generated ten coefficient matrices $\boldsymbol{Q} \in \mathcal{S}^n$ whose elements independently followed the uniform distribution on the interval $[-1, 1]$.

First, we compare the methods presented in Sections 4.1.1, 4.1.2, and 4.1.3 (denoted MILP, Grid, and SUnif, respectively). We set the parameters in Grid and SUnif as follows. In both methods, we set ϵ that appears in (4.12) and (4.18) to 1 or 0.1. When we used Grid, we set r to the minimum positive integer satisfying (4.12). When we used SUnif, we set the number M of samples to the minimum integer satisfying (4.18) and set ϕ to 0.05, so that (4.14) holds with probability at least 0.95. For each (n, ϵ), we solved ten instances using each method and then took the average of the results. The maximum execution time was set to 3600 s.

Table 2 shows the average objective values obtained by solving standard quadratic programming problems with each method and the average computational time required to solve them. We note that although we also solved the problems with $n = 5000$, we could not solve some of the ten instances within the maximum execution time regardless of the methods used.

We were able to solve the problems with MILP within the maximum execution time in the case of $n \leq 500$, but the computational time exceeded the maximum execution time in the case of $n = 1000$. Whether we were able to solve the problems with Grid within the maximum execution time depended on the setting of ϵ. When $\epsilon = 0.1$, we were not able to solve the problems with Grid within the maximum execution time. On the other hand, when $\epsilon = 1$, we were able to solve the problems with $n = 1000$ in an average time of 8.4 s. In addition, the objective values obtained by using Grid were much more accurate than the theoretical bounds shown in (4.13). The values in the "Dev. from MILP" column were much smaller than ϵ appearing in the respective row. There is no reason to use SUnif to solve standard quadratic programming problems because the computational time of SUnif was always longer than that of MILP.

Second, to see the effectiveness of the randomized method introduced in Section 4.1.4 (denoted GUnif), we compare it with the other randomized method, SUnif, in terms of solution accuracy and computational time. Recall that in SUnif we sample from the standard simplex Δ^{n-1}, whereas in GUnif we sample points from the regular grid Δ_r^{n-1} for some positive integer r. For each n, we solved only one of the ten instances. We set ϵ to 1. For each (n, ϵ) we set r as in the previous experiment and choose the number M of samples to be the smallest integer greater than or equal to $c|\Delta_r^{n-1}|$, where $c \in \{5 \times 10^{-1}, 1 \times 10^{-1}, 1 \times 10^{-2}\}$. We sampled M points $\boldsymbol{\delta}_1, \dots, \boldsymbol{\delta}_M$ from the standard simplex Δ^{n-1} when we used SUnif, whereas we sampled from the grid Δ_r^{n-1} when we used GUnif. Then we calculated the value $\min_{1 \leq i \leq M} \boldsymbol{\delta}_i^\top \boldsymbol{Q} \boldsymbol{\delta}_i$. Strictly speaking, our choice of M does not follow the results in Sections 4.1.3 and 4.1.4. Here, we compare sampling from the standard simplex with sampling from the regular grid by matching the number of samples.

Table 2: The results of solving standard quadratic programming problems by using MILP, Grid, and SUnif. The average objective values obtained by solving standard quadratic programming problems with each method are listed in the "Obj. val." column and the average computational time required to solve them is listed in the "Time [s]" column. The "Dev. from MILP" column denotes the difference between each average objective value and that obtained by using MILP for the same n. The "M" column denotes the average number M of points $\boldsymbol{\delta}_1, \dots, \boldsymbol{\delta}_M \in \Delta^{n-1}$ that are used to calculate an approximate optimal value $\min_{1 \leq i \leq M} \boldsymbol{\delta}_i^\top \boldsymbol{Q} \boldsymbol{\delta}_i$ for a standard quadratic programming problem of the form (4.7). The symbol >`realmax` in the "M" column indicates that the average number of points exceeds `realmax` in MATLAB, which is the largest finite floating-point number in IEEE double precision (approximately 1.7977×10^{308}).

n	Method	ϵ	M	Obj. val.	Dev. from MILP	Time [s]
5	MILP	—	—	-7.96×10^{-1}	—	3.3×10^{-3}
	Grid	1	1.5×10^{1}	-7.90×10^{-1}	5.54×10^{-3}	2.9×10^{-3}
		0.1	6.3×10^{3}	-7.96×10^{-1}	1.01×10^{-4}	1.6×10^{-2}
	SUnif	1	1.2×10^{3}	-6.27×10^{-1}	1.69×10^{-1}	2.8×10^{-2}
		0.1	1.2×10^{7}	-7.75×10^{-1}	2.04×10^{-2}	5.3×10^{-1}
10	MILP	—	—	-8.69×10^{-1}	—	7.7×10^{-3}
	Grid	1	5.5×10^{1}	-8.64×10^{-1}	5.00×10^{-3}	2.7×10^{-3}
		0.1	3.3×10^{6}	-8.69×10^{-1}	6.00×10^{-6}	2.8
	SUnif	1	3.6×10^{8}	-6.84×10^{-1}	1.85×10^{-1}	3.3
		0.1	3.6×10^{16}	—	—	>3600
50	MILP	—	—	-9.68×10^{-1}	—	2.6×10^{-1}
	Grid	1	1.3×10^{3}	-9.67×10^{-1}	9.66×10^{-4}	7.3×10^{-3}
		0.1	9.9×10^{16}	—	—	>3600
	SUnif	1	2.8×10^{55}	—	—	>3600
		0.1	2.8×10^{104}	—	—	>3600
100	MILP	—	—	-9.78×10^{-1}	—	1.0
	Grid	1	5.1×10^{3}	-9.78×10^{-1}	1.39×10^{-4}	2.3×10^{-2}
		0.1	2.5×10^{22}	—	—	>3600
	SUnif	1	1.0×10^{126}	—	—	>3600
		0.1	1.0×10^{225}	—	—	>3600
500	MILP	—	—	-9.95×10^{-1}	—	1.4×10^{2}
	Grid	1	1.3×10^{5}	-9.95×10^{-1}	6.89×10^{-5}	8.2×10^{-1}
		0.1	5.7×10^{35}	—	—	>3600
	SUnif	1	>`realmax`	—	—	>3600
		0.1	>`realmax`	—	—	>3600
1000	MILP	—	—	>3600	—	>3600
	Grid	1	5.0×10^{5}	-9.98×10^{-1}	—	8.4
		0.1	5.0×10^{41}	—	—	>3600
	SUnif	1	>`realmax`	—	—	>3600
		0.1	>`realmax`	—	—	>3600

Table 3: The results of solving standard quadratic programming problems by using GUnif and SUnif. The objective values obtained by solving standard quadratic programming problems with each method are listed in the "Obj. val." column and the computational time required to solve them is listed in the "Time [s]" column. The objective value and time for GUnif and SUnif are compared for each row, respectively, and the better results are shown in bold. The "GUnif−MILP" column denotes the difference between the objective value obtained by using GUnif and that obtained by using MILP for the same instance. The "M" column denotes the number M of points $\boldsymbol{\delta}_1, \ldots, \boldsymbol{\delta}_M \in \Delta^{n-1}$ that are used to calculate an approximate optimal value $\min_{1 \le i \le M} \boldsymbol{\delta}_i^\top \boldsymbol{Q} \boldsymbol{\delta}_i$ for a standard quadratic programming problem of the form (4.7).

			Obj. val.			Time [s]	
n	c	M	GUnif	SUnif	GUnif−MILP	GUnif	SUnif
50	5×10^{-1}	638	$\mathbf{-9.35 \times 10^{-1}}$	-9.30×10^{-2}	3.86×10^{-2}	1.3×10^{-2}	$\mathbf{4.7 \times 10^{-3}}$
	1×10^{-1}	128	$\mathbf{-6.62 \times 10^{-1}}$	-8.64×10^{-2}	3.12×10^{-1}	$\mathbf{4.0 \times 10^{-3}}$	5.2×10^{-3}
	1×10^{-2}	13	$\mathbf{-6.57 \times 10^{-1}}$	-6.72×10^{-2}	3.17×10^{-1}	$\mathbf{2.3 \times 10^{-3}}$	2.4×10^{-3}
100	5×10^{-1}	2525	$\mathbf{-9.22 \times 10^{-1}}$	-4.79×10^{-2}	1.99×10^{-2}	3.1×10^{-2}	$\mathbf{9.3 \times 10^{-3}}$
	1×10^{-1}	505	$\mathbf{-8.99 \times 10^{-1}}$	-3.34×10^{-2}	4.31×10^{-2}	8.8×10^{-3}	$\mathbf{6.2 \times 10^{-3}}$
	1×10^{-2}	51	$\mathbf{-5.70 \times 10^{-1}}$	-1.99×10^{-2}	3.71×10^{-1}	3.5×10^{-3}	$\mathbf{2.5 \times 10^{-3}}$
500	5×10^{-1}	62 625	$\mathbf{-9.95 \times 10^{-1}}$	-1.37×10^{-2}	1.75×10^{-3}	1.4	$\mathbf{5.0 \times 10^{-1}}$
	1×10^{-1}	12 525	$\mathbf{-9.47 \times 10^{-1}}$	-1.31×10^{-2}	4.97×10^{-2}	3.1×10^{-1}	$\mathbf{9.9 \times 10^{-2}}$
	1×10^{-2}	1253	$\mathbf{-9.48 \times 10^{-1}}$	-1.06×10^{-2}	4.87×10^{-2}	5.2×10^{-2}	$\mathbf{1.6 \times 10^{-2}}$
1000	5×10^{-1}	250 250	$\mathbf{-9.88 \times 10^{-1}}$	-7.60×10^{-3}	4.83×10^{-3}	1.3×10^{1}	**4.3**
	1×10^{-1}	50 050	$\mathbf{-9.82 \times 10^{-1}}$	-5.96×10^{-3}	1.10×10^{-2}	2.7	$\mathbf{8.6 \times 10^{-1}}$
	1×10^{-2}	5005	$\mathbf{-9.42 \times 10^{-1}}$	-5.25×10^{-3}	5.10×10^{-2}	2.7×10^{-1}	$\mathbf{9.1 \times 10^{-2}}$

Tables 3 shows the objective values obtained by solving standard quadratic programming problems with each method and the computational time required to solve them. Here, we present only the results for the case $\epsilon = 1$; similar results were also obtained for $\epsilon = 0.1$. We did not display the results of $n = 5, 10$ because in these cases the numbers of points were always less than 50.

We observed that the objective values obtained by using GUnif were always better than those obtained with SUnif. However, in many cases, the computational time for GUnif was longer than that for SUnif. The MATLAB Profiler showed that our implementation of GUnif spent most time in the built-in function `sort`, which we used to implement uniform sampling from the regular grid. A more efficient implementation of GUnif would shorten its computational time.

5.2 Solving general copositive programming problems

In this subsection, we compare the performance of methods for solving standard quadratic programming problems when they are incorporated into Algorithm 1 for copositive programming. The copositive programming problem considered here is of the form (4.1). We let $n \in \{5, 10, 50, 100, 500, 1000\}$ and set $m = 5$. For each (n, m), we created ten instances of copositive programming, each generated as follows. The set S was defined as $\mathbb{R}^5$. Each element of the vector $\boldsymbol{c} \in \mathbb{R}^5$ was independently sampled from the chi distribution with 1 degree of freedom. The $(1,1), \ldots, (5,5)$th elements of $\boldsymbol{A}_0 \in \mathcal{S}^n$ were set to 0. The other elements of $\boldsymbol{A}_0$ were independently sampled from the chi distribution with 1 degree of freedom, and then 0.01 was added to each. For each $i \in [5]$, the (i,i)th element of $\boldsymbol{A}_i \in \mathcal{S}^n$ was set to 1 and the $(1,1), \ldots, (i-1, i-1), (i+1, i+1), \ldots, (5,5)$th elements were set to 0. The other elements of $\boldsymbol{A}_i$ were independently sampled from the standard normal distribution. For each $j \in [5]$, the (j,j)th element of the slack matrix $\boldsymbol{A}_0 + \sum_{i=1}^{5} x_i \boldsymbol{A}_i$ is x_j, which must be nonnegative due to copositivity. This implies that every feasible solution $\boldsymbol{x} \in \mathbb{R}^5$ of the copositive programming problem is entrywise nonnegative. Since $\boldsymbol{c}$ is also entrywise nonnegative, we have $\boldsymbol{c}^\top \boldsymbol{x} \geq 0$. In addition, the nonnegativity of $\boldsymbol{A}_0$ implies that $\boldsymbol{x} = \boldsymbol{0}$ is a feasible solution, at which the objective value is 0. Therefore, the optimal value of the problem is 0 and $\boldsymbol{x} = \boldsymbol{0}$ is an optimal solution.

Regardless of the subproblem solver, the initial point $\boldsymbol{x}_1 \in \mathbb{R}^5$ of Algorithm 1 was set to the vector with all elements equal to 1. At each iteration indexed by k, we found $\boldsymbol{\delta}_k \in \Delta^{n-1}$ satisfying (3.7). We see that the distance between $\boldsymbol{x}_1$ and the set of optimal solutions is less than or equal to $\sqrt{5}$ since $\boldsymbol{x} = \boldsymbol{0}$ is optimal. Based on this and (3.4), we stopped the algorithm when the number of iterations was greater than or equal to $5L^2/\epsilon^2$ or the computational time exceeded 3600 s.

Following Section 5.1, we used MILP and Grid to solve a standard quadratic programming problem that appears as a subproblem at each iteration of Algorithm 1. When using MILP to solve subproblems, we set $\alpha = 0$ and $\epsilon = 2$, and when using Grid,

Table 4: The results of applying Algorithm 1 to copositive programming problems and solving the subproblems using MILP, Grid, and GUnif. The average numbers of iterations are listed in the "#Iter." columns, the average objective values obtained using each method are listed in the "Obj. val." column, and the average computational time required is listed in the "Time [s]" column.

n	Method	#Iter.	Obj. val.	Time [s]
5	MILP	6.4	-3.02	6.3×10^{-2}
	Grid	2.5×10^1	-2.22	2.0×10^{-2}
10	MILP	1.1×10^1	-3.48	6.6×10^{-2}
	Grid	4.1×10^1	-2.62	1.8×10^{-1}
50	MILP	1.6×10^1	-1.51	1.5
	Grid	1	—	>3600
100	MILP	1.9×10^1	-1.86	9.3
	Grid	1	—	>3600
500	MILP	2.8×10^1	-1.08	9.2×10^2
	Grid	1	—	>3600
1000	GUnif	1.2×10^2	-1.06×10^2	6.7×10^2

we set $\alpha = 1$ and $\epsilon = 1$, so that $\epsilon(1 + \alpha) = 2$ in both cases. When using Grid to solve a standard quadratic programming problem of the form (4.7) at each iteration, we set r to the minimum positive integer satisfying (4.12).

In preliminary experiments, we observed that for both MILP and Grid the computational time exceeded 3600 s for $n \geq 1000$. To solve these instances, we utilized GUnif. We also set $\alpha = 1$ and $\epsilon = 1$. To solve a standard quadratic programming problem of the form (4.7) at each iteration, we set r to the minimum positive integer satisfying (4.12) and sampled 100 000 points from Δ_r^{n-1}.

Existing methods based on a simplicial partition [13, 55] and cutting-plane techniques [4, 24] introduced in Section 1 can be used to solve copositive programming problems. These methods seem effective for finding approximate solutions in practice, as evidenced by numerical results reported in previous studies. However, our method differs from them in that it has a non-asymptotic convergence guarantee, which the others lack. For this reason, we decided not to compare our method with existing methods numerically.

Table 4 shows the average numbers of iterations, the average objective values obtained using each method, and the average computational time required to solve copositive programming problems. Here, we present only the results for the case $\epsilon(1+\alpha) = 2$; similar results were also obtained for $\epsilon(1 + \alpha) = 0.2$. Except for $n = 5$, the computational time for MILP was shorter than that for Grid. We observed that when using Grid, solving standard quadratic programming problems was time-consuming; indeed, even the first iteration was not completed for instances that exceeded 3600 s. This result contrasts with that in Table 2, where the computational time for Grid was shorter than that for MILP. This indicates that whether Grid or MILP is better depends on the instance. Further investigation is needed to determine under what conditions the computational time for Grid is shorter than that for MILP.

The number of samples in GUnif is not consistent with Proposition 4.5 and is much smaller than the number of points in the grid.[*2] The average objective values obtained using GUnif were rather coarse. However, when MILP and Grid take too long to solve problems, employing GUnif to obtain approximate solutions may be a viable alternative.

5.3 Testing for non-complete positivity of exceptional doubly nonnegative matrices

A real symmetric matrix is said to be *doubly nonnegative* if it is positive semidefinite and entrywise nonnegative. It is well known that every completely positive matrix is doubly nonnegative, and the converse holds if and only if the order is less than or equal to 4; see [26, page 101] and [39]. We can easily check whether a given matrix is doubly nonnegative, so we are interested in deciding whether a given doubly nonnegative matrix is completely positive. In Theorem 4.8, we provided a sufficient condition for a given matrix not to be completely positive. In this subsection, we demonstrate that this theorem is effective for doubly nonnegative matrices that are not completely positive. For convenience, following the terminology of [54], we call a doubly nonnegative matrix that is not completely positive *exceptional*.[*3]

From the numerical results in Sections 5.1 and 5.2, we observe that, even when using MILP, the standard quadratic programming problem arising at each iteration of Algorithm 1 can be solved within a reasonable time, provided that the problem size is not large. For this reason, in the subsequent experiments we used only MILP to solve standard quadratic programming problems, and thus set the violation measure α to 0. We set the error tolerance ϵ and the number of iterations as described in Remark 4.9. For the constant t, specified separately for the two experiments below, Algorithm 1 was terminated when the number of iterations reached $1/\epsilon^2 = t^2n^2$ or when the computational time exceeded 3600 s. As mentioned in Remark 4.10, for an input matrix $\boldsymbol{C} \in \mathcal{S}^n$, we also stopped Algorithm 1 as soon as we had a matrix $\boldsymbol{X}_k$ satisfying $g(\boldsymbol{X}_k; \boldsymbol{\delta}_k) \le \epsilon$ and $\langle \boldsymbol{C}, \boldsymbol{X}_k \rangle < -n\epsilon$.

First, we tried to detect non-complete positivity for the ten 6×6 exceptional doubly nonnegative matrices presented in [5, Appendix B]. We normalized each matrix and used the normalized matrix as the input matrix $\boldsymbol{C}$ in (4.26). In this experiment, we set the constant t that appears in (4.34) to 55. Table 5 shows the results of detecting

[*2]If we set r to the minimum positive integer satisfying (4.25), the number of points in Δ_r^{n-1} sometimes exceeded `realmax` in MATLAB.

[*3]Burer, Anstreicher, and Dür [15] refer to an exceptional doubly nonnegative matrix as a *bad* matrix.

Table 5: The results of detecting non-complete positivity for the exceptional doubly nonnegative matrices presented in [5, Appendix B]. In the "Non-CP" column, "Y" denotes successful detection, whereas "N" indicates failure to detect non-complete positivity.

Name	Non-CP	Time [s]
`extremal_rand_1`	Y	2.3×10^2
`extremal_rand_2`	Y	4.7×10^1
`extremal_rand_3`	Y	1.6×10^1
`extremal_rand_4`	Y	1.7×10^2
`extremal_rand_5`	N	4.4×10^2
`extremal_rand_6`	Y	8.4
`extremal_rand_7`	Y	2.2×10^1
`extremal_rand_8`	Y	4.3
`extremal_rand_9`	Y	1.2×10^1
`extremal_rand_10`	Y	1.6×10^1

Table 6: The results of detecting non-complete positivity for the exceptional doubly nonnegative matrices presented in [54]. In the "Non-CP" column, "Y" denotes successful detection, whereas "N" indicates failure to detect non-complete positivity.

n	τ	Non-CP	Time [s]
10	0.24	Y	7.8×10^1
20	0.37	Y	3.1×10^2
30	0.39	Y	2.7×10^3
40	0.40	N	>3600

non-complete positivity for the ten matrices. Except for the matrix `extremal_rand_5`, we successfully detected non-complete positivity for all the other matrices.

Next, we tried to detect non-complete positivity for exceptional doubly nonnegative matrices provided by Štrekelj and Zalar [54]. For a positive integer m and $\boldsymbol{a} \in \mathbb{R}^m$, let $\boldsymbol{C}(\boldsymbol{a}) \in \mathcal{S}^n$ be the matrix whose (i, j)th element is

$$c_i c_j \int_0^1 \left(1 + 2\sum_{k=1}^{m} a_k \cos(2k\pi x)\right) \cos(2(i-1)\pi x)\cos(2(j-1)\pi x)dx, \tag{5.1}$$

where $c_i = 1$ if $i = 1$ and $c_i = \sqrt{2}$ if $i \geq 2$. Note that we can calculate (5.1) analytically; see [54, Equation (2.3)]. Then, inspired by the method for constructing exceptional doubly nonnegative matrices presented in [54, Equation (1.12)], we solved the following semidefinite feasibility problem:

$$\begin{aligned}
&\left\langle \boldsymbol{C}_{1:5,1:5}(\boldsymbol{a}), \begin{pmatrix} 1 & -1 & 1 & 1 & -1 \\ -1 & 1 & -1 & 1 & 1 \\ 1 & -1 & 1 & -1 & 1 \\ 1 & 1 & -1 & 1 & -1 \\ -1 & 1 & 1 & -1 & 1 \end{pmatrix} \right\rangle = -\tau, \\
&\boldsymbol{C}(\boldsymbol{a}) \text{ is positive semidefinite,} \\
&\boldsymbol{a} \in \mathbb{R}^m_+,
\end{aligned} \tag{5.2}$$

where τ is a given positive value and $\boldsymbol{C}_{1:5,1:5}(\boldsymbol{a})$ denotes the principal submatrix obtained by extracting the first through fifth rows and columns of $\boldsymbol{C}(\boldsymbol{a})$. If $\boldsymbol{a}$ is a feasible solution of this problem, then $\boldsymbol{C}(\boldsymbol{a})$ is an exceptional doubly nonnegative matrix. For each $n \in \{10, 20, 30, 40\}$, we set $m = n + 1$ and took τ as shown in Table 6 such that τ was as large as possible and Problem (5.2) was feasible. After obtaining $\boldsymbol{a}$, we normalized the matrix $\boldsymbol{C}(\boldsymbol{a})$ and used the normalized matrix as the input matrix $\boldsymbol{C}$ in (4.26). We solved Problem (5.2) using the modeling language YALMIP [38] (version 20250626) and the MOSEK solver [40] (version 11.0.24). In this experiment, we set the constant t that appears in (4.34) to 15. Table 6 shows the results of detecting non-complete positivity for the matrices. Despite the inability to ascertain the non-complete positivity of the matrix with $n = 40$ due to the time limit, we were able to detect it for the matrices with $n = 10, 20, 30$.

However, the choice of the constant t in (4.34) is crucial, and in the two experiments we selected t by trial and error. The smaller t is, the fewer the iterations, but at the same time the harder it is to detect non-complete positivity for a matrix. In the first experiment, setting $t = 50$ failed to detect non-complete positivity for the matrix `extremal_rand_1`. In the second experiment, setting $t = 5$ failed to detect non-complete positivity for any of the exceptional doubly nonnegative matrices. We need to consider how to set t to balance the number of iterations and the performance of detecting non-complete positivity.

6 Conclusion

In this paper, based on the subgradient algorithm in [44, Equation (3.2.24)], which is applicable to convex programming problems with a single nonsmooth functional constraint, we proposed an algorithm with a non-asymptotic convergence guarantee to solve copositive programming problems. In contrast to the algorithm in [44, Equation (3.2.24)], the proposed algorithm allows us to solve the subproblem, which is a standard quadratic programming problem, inexactly at each iteration. We discussed how to solve the standard quadratic programming problem exactly and inexactly.

Through the numerical experiments, we identified trends indicating which method is preferable under which circumstances for solving standard quadratic programming problems. For small-scale problems, the exact method for standard quadratic programming via mixed-integer linear programming is acceptable. For medium-scale problems, the exact method may be effective in some cases, whereas the inexact deterministic method obtained by discretizing the standard simplex using a regular grid may be effective in others. Note, however, that the inexact method has the advantage that it does not require external solvers. For large-scale problems where the above two methods take too long to solve, uniform sampling from the regular grid might be another option to solve standard quadratic programming problems inexactly, although its solution accuracy is coarse.

Moreover, we applied the proposed algorithm to the problem of testing complete positivity of a matrix. By using its convergence result, we provided a sufficient condition for certifying that the matrix is not completely positive. In the numerical experiments, we were able to detect non-complete positivity in various doubly nonnegative matrices that are not completely positive.

Acknowledgments The first author is supported by JSPS Grant-in-Aid for Research Activity Start-up JP25K23344. The second and third authors are supported by JSPS Grant-in-Aid for Scientific Research(B) JP23H03351. The third author is also supported by JST CREST Grant Number JPMJCR24Q2.

References

[1] A.A. Ahmadi and A. Majumdar. DSOS and SDSOS optimization: more tractable alternatives to sum of squares and semidefinite optimization. *SIAM J. Appl. Algebra Geom.*, 3(2):193–230, 2019. doi:10.1137/18M118935X.

[2] F. Ahmed, M. Dür, and G. Still. Copositive programming via semi-infinite optimization. *J. Optim. Theory Appl.*, 159(2):322–340, 2013. doi:10.1007/s10957-013-0344-2.

[3] K.M. Anstreicher. Testing copositivity via mixed–integer linear programming. *Linear Algebra Appl.*, 609:218–230, 2021. doi:10.1016/j.laa.2020.09.002.

[4] R. Badenbroek and E. de Klerk. An analytic center cutting plane method to determine complete positivity of a matrix. *INFORMS J. Comput.*, 34(2):1115–1125, 2022. doi:10.1287/ijoc.2021.1108.

[5] R. Badenbroek and E. de Klerk. Simulated annealing for convex optimization: rigorous complexity analysis and practical perspectives. *J. Optim. Theory Appl.*, 194(2):465–491, 2022. doi:10.1007/s10957-022-02034-x.

[6] A. Beck, A. Ben-Tal, N. Guttmann-Beck, and L. Tetruashvili. The CoMirror algorithm for solving nonsmooth constrained convex problems. *Oper. Res. Lett.*, 38(6):493–498, 2010. doi:10.1016/j.orl.2010.08.005.

[7] A. Berman, M. Dür, and N. Shaked-Monderer. Open problems in the theory of completely positive and copositive matrices. *Electron. J. Linear Algebra*, 29:46–58, 2015. doi:10.13001/1081-3810.2943.

[8] A. Berman and U.G. Rothblum. A note on the computation of the CP-rank. *Linear Algebra Appl.*, 419(1):1–7, 2006. doi:10.1016/j.laa.2006.04.001.

[9] I.M. Bomze and E. de Klerk. Solving standard quadratic optimization problems via linear, semidefinite and copositive programming. *J. Glob. Optim.*, 24(2):163–185, 2002. doi:10.1023/A:1020209017701.

[10] I.M. Bomze, M. Dür, E. de Klerk, C. Roos, A.J. Quist, and T. Terlaky. On copositive programming and standard quadratic optimization problems. *J. Glob. Optim.*, 18(4):301–320, 2000. doi:10.1023/A:1026583532263.

[11] I.M. Bomze and M. Gabl. Optimization under uncertainty and risk: quadratic and copositive approaches. *Eur. J. Oper. Res.*, 310(2):449–476, 2023. doi:10.1016/j.ejor.2022.11.020.

[12] I.M. Bomze, M. Locatelli, and F. Tardella. New and old bounds for standard quadratic optimization: dominance, equivalence and incomparability. *Math. Program.*, 115(1):31–64, 2008. doi:10.1007/s10107-007-0138-0.

[13] S. Bundfuss and M. Dür. An adaptive linear approximation algorithm for copositive programs. *SIAM J. Optim.*, 20(1):30–53, 2009. doi:10.1137/070711815.

[14] S. Burer. On the copositive representation of binary and continuous nonconvex quadratic programs. *Math. Program.*, 120(2):479–495, 2009. doi:10.1007/s10107-008-0223-z.

[15] S. Burer, K.M. Anstreicher, and M. Dür. The difference between 5×5 doubly nonnegative and completely positive matrices. *Linear Algebra Appl.*, 431(9):1539–1552, 2009. doi:10.1016/j.laa.2009.05.021.

[16] E. de Klerk and D.V. Pasechnik. Approximation of the stability number of a graph via copositive programming. *SIAM J. Optim.*, 12(4):875–892, 2002. doi:10.1137/S1052623401383248.

[17] P.J.C. Dickinson and M. Dür. Linear-time complete positivity detection and decomposition of sparse matrices. *SIAM J. Matrix Anal. Appl.*, 33(3):701–720, 2012. doi:10.1137/110848177.

[18] P.J.C. Dickinson and L. Gijben. On the computational complexity of membership problems for the completely positive cone and its dual. *Comput. Optim. Appl.*, 57(2):403–415, 2014. doi:10.1007/s10589-013-9594-z.

[19] F. Flores-Bazán, G. Cárcamo, and S. Caro. Extensions of the standard quadratic optimization problem: strong duality, optimality, hidden convexity and S-lemma. *Appl. Math. Optim.*, 81(2):383–408, 2020. doi:10.1007/s00245-018-9502-0.

[20] M.A. Goberna and M.A. López. Recent contributions to linear semi-infinite optimization: an update. *Ann. Oper. Res.*, 271(1):237–278, 2018. doi:10.1007/s10479-018-2987-8.

[21] M.A. Goberna, A.B. Ridolfi, and V.N. Vera de Serio. New applications of linear semi-infinite optimization theory in copositive optimization. *Optimization*, to appear. doi:10.1080/02331934.2024.2411165.

[22] Y.G. Gökmen and E.A. Yıldırım. On standard quadratic programs with exact and inexact doubly nonnegative relaxations. *Math. Program.*, 193(1):365–403, 2022. doi:10.1007/s10107-020-01611-0.

[23] J. Gondzio and E.A. Yıldırım. Global solutions of nonconvex standard quadratic programs via mixed integer linear programming reformulations. *J. Glob. Optim.*, 81(2):293–321, 2021. doi:10.1007/s10898-021-01017-y.

[24] C. Guo, M. Bodur, and J.A. Taylor. Copositive duality for discrete energy markets. *Manag. Sci.*, to appear. doi:10.1287/mnsc.2023.00906.

[25] Gurobi Optimization. *Gurobi Optimizer Reference Manual*, 12.0 edition, 2025. URL: https://docs.gurobi.com/projects/optimizer/en/current/.

[26] M. Hall, Jr. A survey of combinatorial analysis. In I. Kaplansky, E. Hewitt, M. Hall, Jr., and R. Fortet, editors, *Some Aspects of Analysis and Probability*, pages 35–104. John Wiley & Sons, New York, NY, 1958.

[27] M. Hall, Jr. and M. Newman. Copositive and completely positive quadratic forms. *Math. Proc. Camb. Philos. Soc.*, 59(2):329–339, 1963. doi:10.1017/S0305004100036951.

[28] J.-B. Hiriart-Urruty and A. Seeger. A variational approach to copositive matrices. *SIAM Rev.*, 52(4):593–629, 2010. doi:10.1137/090750391.

[29] F. Jarre and K. Schmallowsky. On the computation of C^* certificates. *J. Glob. Optim.*, 45(2):281–296, 2009. doi:10.1007/s10898-008-9374-y.

[30] M. Jerrum. *Counting, Sampling and Integrating: Algorithms and Complexity.* Birkhäuser Verlag, Basel, Switzerland, 2003. doi:10.1007/978-3-0348-8005-3.

[31] S. Kim, M. Kojima, and K.-C. Toh. Doubly nonnegative relaxations are equivalent to completely positive reformulations of quadratic optimization problems with block-clique graph structures. *J. Glob. Optim.*, 77(3):513–541, 2020. doi:10.1007/s10898-020-00879-y.

[32] O.I. Kostyukova and T.V. Tchemisova. Optimality conditions for linear copositive programming problems with isolated immobile indices. *Optimization*, 69(1):145–164, 2020. doi:10.1080/02331934.2018.1539482.

[33] O.I. Kostyukova and T.V. Tchemisova. On equivalent representations and properties of faces of the cone of copositive matrices. *Optimization*, 71(11):3211–3239, 2022. doi:10.1080/02331934.2022.2027939.

[34] O.I. Kostyukova and T.V. Tchemisova. On strong duality in linear copositive programming. *J. Glob. Optim.*, 83(3):457–480, 2022. doi:10.1007/s10898-021-00995-3.

[35] O.I. Kostyukova, T.V. Tchemisova, and O.S. Dudina. Immobile indices and CQ-free optimality criteria for linear copositive programming problems. *Set-Valued Var. Anal.*, 28(1):89–107, 2020. doi:10.1007/s11228-019-00527-y.

[36] J.B. Lasserre. New approximations for the cone of copositive matrices and its dual. *Math. Program.*, 144(1–2):265–276, 2014. doi:10.1007/s10107-013-0632-5.

[37] M. Laurent and L.F. Vargas. On the exactness of sum-of-squares approximations for the cone of 5×5 copositive matrices. *Linear Algebra Appl.*, 651:26–50, 2022. doi:10.1016/j.laa.2022.06.015.

[38] J. Löfberg. YALMIP: a toolbox for modeling and optimization in MATLAB. In *Proceedings of the 2004 IEEE International Symposium on Computer Aided Control Systems Design*, pages 284–289, 2004. doi:10.1109/CACSD.2004.1393890.

[39] J.E. Maxfield and H. Minc. On the matrix equation $X'X = A$. *Proc. Edinb. Math. Soc.*, 13(2):125–129, 1962. doi:10.1017/S0013091500014681.

[40] MOSEK ApS. *MOSEK Optimization Toolbox for MATLAB*, 11.0.29 edition, 2025. URL: https://docs.mosek.com/11.0/toolbox/index.html.

[41] K.G. Murty and S.N. Kabadi. Some NP-complete problems in quadratic and nonlinear programming. *Math. Program.*, 39(2):117–129, 1987. doi:10.1007/BF02592948.

[42] Y. Nesterov. Random walk in a simplex and quadratic optimization over convex polytopes. Technical report, CORE, Catholic University of Louvain, Louvain-la-Neuve, Belgium, 2003.

[43] Y. Nesterov. *Introductory Lectures on Convex Optimization: A Basic Course.* Springer, New York, NY, first edition, 2004. doi:10.1007/978-1-4419-8853-9.

[44] Y. Nesterov. *Lectures on Convex Optimization.* Springer, Cham, Switzerland, second edition, 2018. doi:10.1007/978-3-319-91578-4.

[45] J. Nie. The $\mathcal{A}$-truncated K-moment problem. *Found. Comput. Math.*, 14(6):1243–1276, 2014. doi:10.1007/s10208-014-9225-9.

[46] M. Nishijima and B.F. Lourenço. Facial structure of copositive and completely positive cones over a second-order cone. *arXiv e-prints*, 2025. doi:10.48550/arXiv.2502.04006.

[47] M. Nishijima and B.F. Lourenço. Non-facial exposedness of copositive cones over symmetric cones. *J. Math. Anal. Appl.*, 545(2):129166, 2025. doi:10.1016/j.jmaa.2024.129166.

[48] M. Nishijima and K. Nakata. Approximation hierarchies for copositive cone over symmetric cone and their comparison. *J. Glob. Optim.*, 88(4):831–870, 2024. doi:10.1007/s10898-023-01319-3.

[49] M. Nishijima and K. Nakata. Generalizations of doubly nonnegative cones and their comparison. *J. Oper. Res. Soc. Jpn.*, 67(3):84–109, 2024. doi:10.15807/jorsj.67.84.

[50] M. Orlitzky. Gaddum's test for symmetric cones. *J. Glob. Optim.*, 79(4):927–940, 2021. doi:10.1007/s10898-020-00960-6.

[51] P.A. Parrilo. Semidefinite programming based tests for matrix copositivity. In *Proceedings of the 39th IEEE Conference on Decision and Control*, pages 4624–4629, 2000. doi:10.1109/CDC.2001.914655.

[52] J. Peña, J. Vera, and L.F. Zuluaga. Computing the stability number of a graph via linear and semidefinite programming. *SIAM J. Optim.*, 18(1):87–105, 2007. doi:10.1137/05064401X.

[53] J. Sponsel, S. Bundfuss, and M. Dür. An improved algorithm to test copositivity. *J. Glob. Optim.*, 52(3):537–551, 2012. doi:10.1007/s10898-011-9766-2.

[54] T. Štrekelj and A. Zalar. Construction of exceptional copositive matrices. *Linear Algebra Appl.*, 727:368–384, 2025. doi:10.1016/j.laa.2025.08.010.

[55] J. Žilinskas. Copositive programming by simplicial partition. *Informatica*, 22(4):601–614, 2011. doi:10.15388/Informatica.2011.345.

[56] B. Wei, W.B. Haskell, and S. Zhao. The CoMirror algorithm with random constraint sampling for convex semi-infinite programming. *Ann. Oper. Res.*, 295(2):809–841, 2020. doi:10.1007/s10479-020-03766-7.

[57] C. Xu. Completely positive matrices of order five. *Acta Math. Appl. Sin.*, 17(4):550–562, 2001. doi:10.1007/BF02669709.

[58] E.A. Yıldırım. On the accuracy of uniform polyhedral approximations of the copositive cone. *Optim. Methods Softw.*, 27(1):155–173, 2012. doi:10.1080/10556788.2010.540014.