IMPLEMENTATION OF INTERIOR POINT METHODS

IMPLEMENTATION OF INTERIOR POINT METHODS FOR SECOND ORDER CONIC OPTIMIZATION

By

Bixiang Wang, Ph.D.

A Thesis

Submitted to the School of Graduate Studies

in Partial Fulfilment of the Requirements

for the Degree

Master of Science

McMaster University

MASTER OF SCIENCE (2003) McMaster University
(Computing and Software) Hamilton, Ontario

TITLE: Implementation of Interior Point Methods for Second Order Conic Optimization.

AUTHOR: Bixiang Wang (McMaster University)

SUPERVISOR: Dr. Tamás Terlaky

NUMBER OF PAGES: ix, 113

Abstract

This thesis deals with the implementation of interior point methods for second order conic optimization (SOCO) problems. In an SOCO, we minimize a linear function subject to the intersection of an affine set, and product of second order cones. SOCO problems have a variety of applications in engineering, including antenna array design and truss design.

We first introduce a primal-dual interior point algorithm that is based on self-regular search directions. This algorithm employs the homogeneous self-dual embedding model, with Nesterov-Todd scaling, and the Mehrotra predictor-corrector technique. Then we discuss efficient implementation of search directions and step length with MATLAB and Watson Sparse Matrix Package. We also introduce an adaptive strategy that is used to choose the barrier parameter to define the self-regular search directions.

We test the algorithm on a variety of problems from the DIMACS set, and compare these results with state of the art software like SeDuMi. Numerical testing shows that our implementation is robust and takes fewer iterations than SeDuMi on some problems.

Acknowledgments

This thesis was completed under the guidance of my supervisor, Dr. Tamás Terlaky, an active expert in the area of linear and nonlinear optimization. I would like to thank Dr. Terlaky for his invaluable advice, encouragement and help during the preparation of this thesis. I am also grateful to Dr. Kartik Krishnan for his careful review and insightful comments on this work. My special thanks go to the members of the examination committee: Dr. Kartik Krishnan, Dr. Ned Nedialkov (Chair), Dr. Jiming Peng and Dr. Tamás Terlaky.

I appreciate the kind help from all members of the Advanced Optimization Laboratory in the Department of Computing and Software. I also acknowledge the assistance from all my friends in Hamilton.

This work was partially supported by the NSERC (Natural Sciences and Engineering Research Council of Canada), and McMaster University. I thank the NSERC for the Postgraduate Scholarship, and McMaster University for the Ashbaugh Scholarship.

Finally, I am indebted to my wife Haiyan Zhao and my daughter Yuxin Wang. Without their continuing love and support, this thesis would have been impossible.

Contents

List of Figures

List of Tables

Preface

Interior point methods are developed for solving optimization problems by following a smooth path inside the feasible region. These are the most efficient methods for searching optimal solutions for large problems with sparse structures. In this thesis, we deal with the implementation of novel interior point methods for second order conic optimization problems that minimizes a linear function subject to the intersection of an affine set and product of quadratic cones. Our implementation is based on the homogeneous self-dual embedding model and self-regular search directions. We also employ the predictor-corrector techniques in the implementation to improve the performance of the program. This thesis is organized as follows.

In Chapter 1, we review the basic results for second order conic optimization problems that include the properties of quadratic cones and duality theory. The Goldman-Tucker homogeneous model is also presented in this chapter.

Chapter 2 is devoted to the interior point methods for the conic optimization problems with the scaled Newton search directions. We first show that the standard Newton search directions do not work for the conic problems, and then introduce the scaled Newton systems and the scaled Newton search directions. We also discuss how to explore the sparse structures of the Newton systems, and how to determine the Newton steps at each iteration. Finally, we deal with the predictor-corrector techniques and present an algorithm based on scaled Newton search directions.

In Chapter 3, we discuss the interior point methods for the conic optimization problems with self-regular search directions. We first review some typical examples of self-regular functions and then define the corresponding self-regular search directions. An algorithm based on the self-regular search directions are also presented in this chapter.

In Chapter 4, we investigate the interior point methods for the conic optimization problems with upper bounds constraints. We first introduce the Newton systems for this type of problems and then solve the corresponding self-regular search directions explicitly. Finally, we present an algorithm for the conic problems with upper bounds constraints.

Chapter 5 is devoted to the implementation of the algorithms discussed in the previous chapters. We first describe the computational environment of our software, and then elaborate on the general structure of the program. After that, we discuss the procedure for solving the search directions in detail and depict the flow charts for the data communication between MATLAB and the Watson Sparse Matrix Package. The starting point and stopping criteria are also presented in this chapter. Finally,

some issues related to line search are discussed.

In Chapter 6, we benchmark our software by a set of testing problems. We first present our numerical results, and then examine the effect of self-regular search directions with different barrier degree parameters. We also compare the numerical results produced by our software and SeDuMi.

Finally, in Chapter 7, we conclude the thesis with some suggestions for future work.

The main contributions of this thesis are summarized as follows. In theoretic aspects, we propose a strategy to build the sparse structures of coefficient matrices of linear systems for a second order conic optimization problem. We also propose a method for computing the Newton step length for this problem. To our knowledge, the formulas given in Section 2.3.3 for building the sparse matrices and dense matrices, and the formulas given in Section 2.4.1 for computing the Newton step length are not documented in the literature.

In practical aspects, we have developed an efficient software package to solve second order conic optimization problems. We have implemented the proposed strategy for building the sparse structures of Newton systems. We have also implemented and benchmarked the back tracking and forward tracking techniques to determine the Newton step length. Our software is tested by solving the DIMACS set of SOCO test problems. We also evaluated the performance of our software with different values of the barrier degree parameter that influence the quality of self-regular search directions.

Chapter 1

Second Order Conic Optimization Problems

In this chapter, we introduce second order conic optimization (SOCO) problems and discuss their fundamental properties. A SOCO problem involves the minimization of a linear objective function subject to the intersection of an affine set and product of second order cones. In order to discuss SOCO problems, we first review the properties of second order cones. Then we present the duality theory for SOCO problems. Finally, we point out that solving SOCO problems is equivalent to solving a homogeneous self-dual SOCO problem that is based on the Goldman-Tucker homogeneous models [3, 13, 28, 32].

1.1 Quadratic Cones

Second order cones are used to define the SOCO problems. In this section, we introduce second order cones and discuss their properties. All results in this section are standard and the reader may consult, e.g., paper [3].

Definition 1.1.1 *Assume that K is a subset of $\mathbb{R}^n$. We say K is a convex cone if K is a convex set and $\lambda x \in K$ for all $\lambda \geq 0$ and $x \in K$.*

Definition 1.1.2 *Let K be a convex cone and $x \in K$. We say x is an interior point of K if there exists $\epsilon > 0$ such that for all y, if $\|y - x\| < \epsilon$, then $y \in K$. The set consisting of all interior points of K is denoted by* $\mathrm{int}(K)$.

Throughout this thesis, $\|x\|$ is used to denote by the 2-norm of x, i.e.,

$$\|x\| = \sqrt{x_1^2 + x_2^2 + \cdots + x_n^2}, \qquad x \in \mathbb{R}^n.$$

There are three types of convex cones that are particularly interesting. These convex cones are the linear cones, the quadratic cones, and the rotated quadratic cones, which are defined as follows.

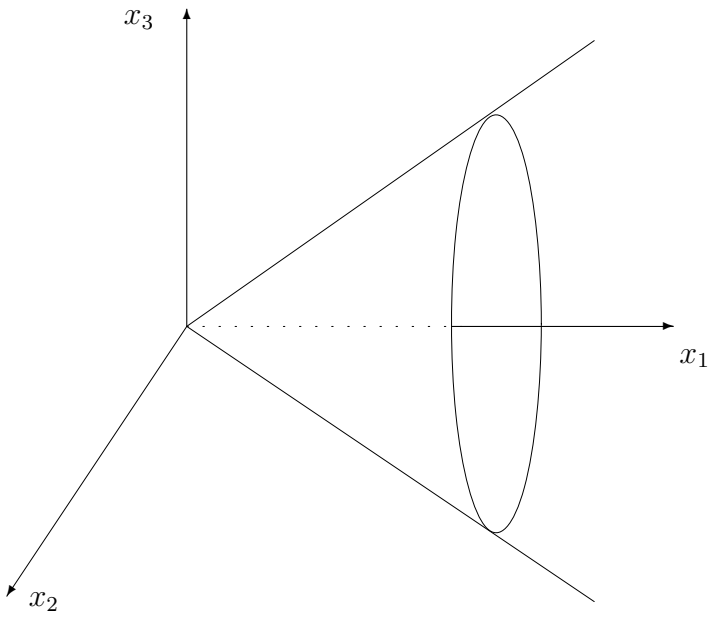
Definition 1.1.3 *The linear cone $\mathbb{R}_+$ is defined as*

$$\mathbb{R}_+ := \{x \in \mathbb{R} : \ x \geq 0\}.$$

Definition 1.1.4 *The quadratic cone K_q is defined as*

$$K_q := \left\{x \in \mathbb{R}^n : \ x_1^2 \geq \|x_{2:n}\|^2, \ x_1 \geq 0\right\}.$$

In what follows, $\|\cdot\|$ is used to denote the 2-norm of a vector, and $x_{2:n} = (x_2, x_3, \cdots, x_n)$. We notice that the quadratic cone K_q is also referred to as the second order cone, Lorentz cone, and ice-cream cone. The three-dimensional second order cone is shown in the following:



There is another type of convex cones that can be transformed into quadratic cones. These cones are called the rotated quadratic cones.

Definition 1.1.5 *The rotated quadratic cone K_r is defined as*

$$K_r := \left\{x \in \mathbb{R}^n : \ 2x_1x_2 \geq \|x_{3:n}\|^2, \ x_1, x_2 \geq 0\right\}.$$

For each of these convex cones, we associate it with two matrices T and Q that are defined as follows.

Definition 1.1.6 *(i) if K is a linear cone $\mathbb{R}_+$, then*

$$T := 1, \quad Q := 1;$$

(ii) if K is a quadratic cone K_q, then

$$T := I_{n\times n}, \quad Q := \mathrm{diag}(1, -1, \cdots, -1);$$

(iii) if K is a rotated quadratic cone K_r, then

$$T := \begin{pmatrix} \frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} & 0 & \cdots & 0 \\ \frac{1}{\sqrt{2}} & -\frac{1}{\sqrt{2}} & 0 & \cdots & 0 \\ 0 & 0 & 1 & \cdots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & \cdots & 1 \end{pmatrix}$$

$$Q := \begin{pmatrix} 0 & 1 & 0 & \cdots & 0 \\ 1 & 0 & 0 & \cdots & 0 \\ 0 & 0 & -1 & \cdots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & \cdots & -1 \end{pmatrix}.$$

Observe that matrices T and Q are both symmetric matrices. Also, one can verify that the matrices T and Q are both orthogonal, that is,

$$TT = I \qquad \text{and} \qquad QQ = I.$$

The cone K can be characterized by the matrix Q associated with it, that is, if K is a quadratic cone K_q, then

$$K_q = \left\{ x \in \mathbb{R}^n : \ x^T Q x \geq 0, \ x_1 \geq 0 \right\}, \tag{1.1.1}$$

and if K is a rotated quadratic cone K_r, then

$$K_r = \left\{ x \in \mathbb{R}^n : \ x^T Q x \geq 0, \ x_1, \ x_2 \geq 0 \right\}. \tag{1.1.2}$$

The matrix T is used to deal with the rotated quadratic cones because it transforms rotated quadratic cones into the corresponding quadratic cones, i.e., we have the following results.

Theorem 1.1.1 *Assume that K_r is a rotated quadratic cone in $\mathbb{R}^n$ and T is the matrix associated with K_r. If K_q is the quadratic cone in $\mathbb{R}^n$, then $TK_r = K_q$, where TK_r is given by:*

$$TK_r := \{Tx : \ x \in K_r\}.$$

Proof. Assume that Q is the matrix associated with K_r, and Q_q is the matrix associated with K_q. Then by simple computations, we find that

$$Q = TQ_qT \qquad \text{and} \qquad Q_q = TQT. \tag{1.1.3}$$

In the following, we first prove $TK_r \subseteq K_q$, and then $K_q \subseteq TK_r$.

(i) Proof of $TK_r \subseteq K_q$. To this end, we take $x \in K_r$. Then it follows from (1.1.2) that

$$x^TQx \geq 0, \ x_1 \geq 0, \ x_2 \geq 0. \tag{1.1.4}$$

Let $y = Tx$. By (1.1.3) and (1.1.4), we have

$$y^TQ_qy = x^TTQ_qTx = x^TQx \geq 0,$$

and

$$y_1 = \frac{1}{\sqrt{2}}x_1 + \frac{1}{\sqrt{2}}x_2 \geq 0,$$

which implies that $y = Tx \in K_q$. Since x is an arbitrary element of K_r, we conclude $TK_r \subseteq K_q$.

(ii) Proof of $K_q \subseteq TK_r$. Let $y \in K_q$ be an arbitrary element. Then it follows from (1.1.1) and Definition 1.1 that

$$y^TQ_qy \geq 0, \ y_1 \geq |y_2|. \tag{1.1.5}$$

Let $x = Ty$. By (1.1.3) and (1.1.5) we find

$$x^TQx = y^TTQTy = y^TQ_qy \geq 0,$$

and

$$x_1 = \frac{1}{\sqrt{2}}y_1 + \frac{1}{\sqrt{2}}y_2 \geq 0, \qquad x_2 = \frac{1}{\sqrt{2}}y_1 - \frac{1}{\sqrt{2}}y_2 \geq 0,$$

which implies that $x = Ty \in K_r$. Since $TT = I$, we find that $y = Tx$ with $x \in K_r$, that is, $y \in TK_r$. This implies $K_q \subseteq TK_r$. ∎

When we present the dual problems for SOCO problems, we have to use the dual cones of given convex cones. The following is the definition of dual cones.

Definition 1.1.7 *Assume that K is a convex cone in $\mathbb{R}^n$. Then the dual cone K^* of K is given by*

$$K^* := \left\{ z \in \mathbb{R}^n : \ z^T x \geq 0, \ \forall x \in K \right\}.$$

If $K^* = K$, then we say K is self-dual.

An important property of the linear, quadratic and rotated quadratic cones is self-duality, i.e., we have the following results.

Theorem 1.1.2 *Assume that K is a linear, quadratic or rotated quadratic cone in $\mathbb{R}^n$. Then $K^* = K$.*

Proof. The proof depends on the types of convex cones and is divided into three cases.

Case 1: K is the linear cone R_+. In this case, by definition, we have

$$K^* = \{z \in \mathbb{R} : \ zx \geq 0, \ \forall x \in \mathbb{R}_+\} = \{z \in \mathbb{R} : \ zx \geq 0, \ \forall x \geq 0\}$$

$$= \{z \in \mathbb{R} : \ z \geq 0\} = \mathbb{R}_+ = K.$$

Case 2: K is the quadratic cone K_q. In this case, the proof consists of two steps. We first show $K \subseteq K^*$, and then $K^* \subseteq K$.

(i) Proof of $K \subseteq K^*$. Let x be an arbitrary element of K. Then we want to show $x \in K^*$, that is, for any $y \in K$, we want to prove $x^T y \geq 0$. Let $x = (x_1, x_{2:n})^T$ and $y = (y_1, y_{2:n})^T$. Since $x, y \in K = K_q$, by definition, we have

$$x_1 \geq \|x_{2:n}\|, \quad y_1 \geq \|y_{2:n}\|. \tag{1.1.6}$$

Then it follows from (1.1.6) and the Cauchy inequality that

$$x^T y = x_1 y_1 + (x_{2:n})^T y_{2:n} \geq x_1 y_1 - \|x_{2:n}\| \|y_{2:n}\| \geq 0,$$

which implies $x^T y \geq 0$ for any $y \in K$, and therefore $x \in K^*$. Since x is an arbitrary element of K, we conclude $K \subseteq K^*$.

(ii) Proof of $K^* \subseteq K$. Let $z = (z_1, z_{2:n})^T \in K^*$. Then we want to show $z \in K$, that is, we want to prove $z_1 \geq \|z_{2:n}\|$. Since $z \in K^*$ and $x = (1, 0, \cdots, 0)^T \in K$, by the definition of K^*, we have $z^T x \geq 0$, i.e., $z_1 \geq 0$. Now we consider the following two cases: $z_{2:n} = 0$ and $z_{2:n} \neq 0$.

(1) $z_{2:n} = 0$. In this case, since $z_1 \geq 0$, we have $z_1 \geq \|z_{2:n}\|$, which implies $z \in K$.

(2) $z_{2:n} \neq 0$. In this case, we first show

$$z_1 > 0. \tag{1.1.7}$$

We argue by contradiction. Assume $z_1 \leq 0$. Since we already know $z_1 \geq 0$, we must have $z_1 = 0$. Now, let $x = (\|z_{2:n}\|, -z_{2:n})^T$. Then we see $x \in K$. Since $z \in K^*$ and $x \in K$, it follows from the definition of K^* that

$$z^T x \geq 0. \tag{1.1.8}$$

On the other hand, since $z_1 = 0$ we have

$$z^T x = -\|z_{2:n}\|^2. \tag{1.1.9}$$

Then it follows from (1.1.8) and (1.1.9) that $z_{2:n} = 0$, which contradicts the assumption $z_{2:n} \neq 0$. This contradiction shows that the assumption $z_1 \leq 0$ is wrong, and therefore (1.1.7) follows.

Next, we prove $z \in K$ when $z_{2:n} \neq 0$. Here, we argue by contradiction again. Assume that z does not belong to K, that is,

$$z_1 < \|z_{2:n}\|. \tag{1.1.10}$$

Let $x = (x_1, x_{2:n})^T$ with $x_1 = z_1$ and $x_{2:n} = -\frac{z_1}{\|z_{2:n}\|} z_{2:n}$. By simple computations, we see $x_1 = \|x_{2:n}\|$, and hence $x \in K$. Since $z \in K^*$ and $x \in K$ we get

$$z^T x \geq 0. \tag{1.1.11}$$

On the other hand, by (1.1.7) and (1.1.10) we see

$$z^T x = z_1 x_1 + z_{2:n}^T x_{2:n} = z_1(z_1 - \|z_{2:n}\|) < 0, \tag{1.1.12}$$

which contradicts (1.1.11). This contradiction shows that assumption (1.1.10) is wrong, and therefore we have $z \in K$. Since z is an arbitrary element of K^*, it follows $K^* \subseteq K$.

Case 3: K is a rotated quadratic cone K_r in $\mathbb{R}^n$. In this case, we denote by K_q the corresponding quadratic cone in $\mathbb{R}^n$, and T the matrix associated with K_r in Definition 1.1.6. Then it follows from Theorem 1.1.1 that $K_r = TK_q$. In what follows, we first prove $K_r^* \subseteq K_r$, and then $K_r \subseteq K_r^*$.

(i) Proof of $K_r^* \subseteq K_r$. $\forall z \in K_r^*$, we want to prove $z \in K_r$, which is equivalent to prove $Tz \in K_q$. $\forall x \in K_q$, we have $Tx \in K_r$. Since $z \in K_r^*$, we find

$$z^T(Tx) \geq 0, \quad \forall x \in K_q,$$

that is,

$$(Tz)^T x \geq 0, \quad \forall x \in K_q. \tag{1.1.13}$$

We notice that (1.1.13) implies $Tz \in K_q^*$. Since $K_q^* = K_q$ as shown above, we get $Tz \in K_q$, which implies $T(Tz) \in K_r$, i.e., $z \in K_r$. Since z is an arbitrary element of K_r^*, we conclude $K_r^* \subseteq K_r$.

(ii) Proof of $K_r \subseteq K_r^*$. For all $z \in K_r$, we want to show $z \in K_r^*$. This is equivalent to prove

$$z^T(Tx) \geq 0, \quad \forall x \in K_q. \tag{1.1.14}$$

Since $z \in K_r$, we know $Tz \in K_q = K_q^*$. Therefore $\forall x \in K_q$, we get $(Tz)^T x \geq 0$, which implies (1.1.14). Since (1.1.14) holds for all $x \in K_q$, it follows $z \in K_r^*$. Since z is an arbitrary element of K_r, we conclude $K_r \subseteq K_r^*$. The proof is complete. ∎

An important property of the quadratic and rotated quadratic cones is their invariance under a special linear transformation. This special transformation is called the scaling matrix and is defined as follows.

Definition 1.1.8 *Assume that K is the linear, quadratic, or rotated quadratic cone in $\mathbb{R}^n$, and G is an $n \times n$ symmetric matrix with the first entry being positive. We say G is a scaling matrix of K if G satisfies the condition:*

$$GQG = Q, \tag{1.1.15}$$

where Q is the matrix associated with K given in Definition 1.1.6.

We notice that if G is a scaling matrix of K, then under the transform of G, the cone K remains unchanged. More precisely, the following is true.

Theorem 1.1.3 *Assume that G is a scaling matrix of cone K in $\mathbb{R}^n$, and θ is a positive number. Let $\bar{x} = \theta G x$ for $x \in K$. Then*

(i) $\bar{x}^T Q \bar{x} = \theta^2 x^T Q x$;

(ii) $\bar{x} \in K \Longleftrightarrow x \in K$;

(iii) $\bar{x} \in \text{int}(K) \Longleftrightarrow x \in \text{int}(K)$.

Proof. (i) By (1.1.15), we have

$$\bar{x}^T Q \bar{x} = (\theta G x)^T Q (\theta G x) = \theta^2 x^T Q x.$$

(ii) We first prove: $x \in K \Longrightarrow \bar{x} \in K$, that is, if $x \in K$, then we want to prove $\bar{x} \in K$. This can be done in three steps according to the types of cone K.

Case 1: K is the linear cone $\mathbb{R}_+$. If $K = \mathbb{R}_+$, then, by the definition we have $G = 1$. Therefore, $\bar{x} = \theta G x = \theta x$, which implies that $\bar{x} \geq 0$ since $x \in \mathbb{R}_+$, i.e., $\bar{x} \in K$ in this case.

Case 2: K is the quadratic cone K_q. In this case, we denote the first row of the matrix G by $(g_1,\ g_{2:n})$. Then it follows from (1.1.15) that

$$g_1^2 - \|g_{2:n}\|^2 = 1. \tag{1.1.16}$$

By assumption, we know $g_1 > 0$, which along with (1.1.16) implies

$$g_1 \geq \|g_{2:n}\|. \tag{1.1.17}$$

If $x = (x_1,\ x_{2:n})^T \in K_q$, it follows from Definition (1.1) that

$$x_1 - \|x_{2:n}\| \geq 0, \qquad x_1 \geq 0.$$

Then, by the Cauchy inequality and (1.1.17) we have

$$\begin{aligned} \bar{x}_1 &= \theta(g_1 x_1 + g_{2:n} \cdot (x_{2:n})^T) \\ &\geq \theta(g_1 x_1 - \|g_{2:n}\|\|x_{2:n}\|) \\ &\geq \theta(\|g_{2:n}\| x_1 - \|g_{2:n}\|\|x_{2:n}\|) \\ &\geq \theta\|g_{2:n}\|(x_1 - \|x_{2:n}\|) \\ &\geq 0. \end{aligned} \tag{1.1.18}$$

On the other hand, we see that $\bar{x}^T Q \bar{x} = \theta^2 x^T Q x \geq 0$, which along with (1.1.18) implies $\bar{x} \in K_q$.

Case 3: K is the rotated quadratic cone K_r. If $K = K_r$, then denote by T and Q the matrix given in Definition (1.1.6) for K_r. Let K_q be the quadratic cone in $\mathbb{R}^n$ and Q_q be the matrix given in Definition (1.1.6) for K_q. We notice that if G is a scaling matrix of K_r, i.e., $GQG = Q$, then $\bar{G} := TGT$ is a scaling matrix of K_q. This is because by (1.1.3):

$$\bar{G}Q_q\bar{G} = (TGT)Q_q(TGT) = TG(TQ_qT)GT = TGQGT = T(GQG)T = TQT = Q_q.$$

Now, Let $x \in K_r$ and $\bar{x} = \theta Gx$. Then we want to prove $\bar{x} \in K_r$. Since $TT = I$, we find that

$$\bar{x} = \theta Gx = \theta(TT)G(TT)x = \theta T(TGT)Tx = \theta T\bar{G}Tx.$$

From $x \in K_r$ and Theorem 1.1.1, it follows that $Tx \in K_q$. Since $\bar{G}$ is a scaling matrix of K_q, from Case 2 in the above, we have $\theta\bar{G}(Tx) \in K_q$. Then it follows from Theorem 1.1.1 again that $T(\theta\bar{G}(Tx)) \in K_r$, i.e., $\bar{x} = \theta(T(\bar{G}(Tx))) \in K_r$.

So far, we have proved that $x \in K \Longrightarrow \bar{x} \in K$. In order to finish the proof of (ii), we have to show that $\bar{x} \in K \Longrightarrow x \in K$, which can be done in a similar way to the above procedure, and therefore the details are omitted here.

(iii) The proof of (iii) is essentially the same as the proof of (ii), with minor modifications. The details are omitted. ∎

As we will see later, when we solve a SOCO problem, we have to choose the suitable scaling matrices for the given cones. Now the question is: how to choose the scaling matrices such that the SOCO problems can be solved efficiently. This question has been studied by many authors such as Monteiro and Tsuchiya [14]; Nesterov and Todd [16], and Tsuchiya [28]. Since the NT scaling matrices proposed in [16] can be computed cheaply for SOCO problems, we use the NT scaling matrices in our implementation, which are defined as follows.

Definition 1.1.9 *Assume that K is the linear, quadratic or rotated quadratic cone in $\mathbb{R}^n$, $x, s \in \mathrm{int}(K)$ and $e_1 = (1, 0, \cdots, 0)^T$. Then a positive number θ and a matrix G based on x and s are defined as follows.*

$$\theta^2 = \sqrt{\frac{s^T Q s}{x^T Q x}}, \tag{1.1.19}$$

$$g = \frac{\theta^{-1} s + \theta Q x}{\sqrt{2}\sqrt{x^T s + \sqrt{x^T Q x s^T Q s}}}, \tag{1.1.20}$$

(i) if K is the linear cone, then $G = 1$;
(ii) if K is the quadratic cone, then

$$G = -Q + \frac{(e_1 + g)(e_1 + g)^T}{1 + e_1^T g}; \tag{1.1.21}$$

(iii) if K is the rotated quadratic cone, then

$$G = -Q + \frac{(Te_1 + g)(Te_1 + g)^T}{1 + (Te_1)^T g}; \tag{1.1.22}$$

We can verify that the matrix G given in the above definition is indeed a scaling matrix for cone K, that is, we have the following results.

Theorem 1.1.4 *Under the assumptions of Definition 1.1.9, the matrix G is a scaling matrix of K, i.e., G satisfies*

$$GQG = Q., \qquad G_{11} > 0,$$

where G_{11} is the first entry of G. Furthermore, G is a symmetric and positive definite matrix.

Proof. The proof is divided into three cases according to the types of cone K.

Case 1: K is the linear cone. In this case, $Q = 1$ and $G = 1$. Of course, $GQG = Q$.

Case 2: K is the quadratic cone. Assume $K = K_q$ and denote by $g = (g_1, g_{2:n})^T$. Then it follows form Definition 1.1.9 that $g_1 > 0$ and

$$\begin{aligned} g_1^2 - \|g_{2:n}\|^2 &= g^T Q g = \left(\frac{\theta^{-1}s+\theta Qx}{\sqrt{2}\sqrt{x^T s+\sqrt{x^T Qxs^T Qs}}}\right)^T Qg \\ &= \left(\frac{\theta^{-1}s^T Q+\theta x^T}{\sqrt{2}\sqrt{x^T s+\sqrt{x^T Qxs^T Qs}}}\right) g \\ &= \left(\frac{\theta^{-1}s^T Q+\theta x^T}{\sqrt{2}\sqrt{x^T s+\sqrt{x^T Qxs^T Qs}}}\right)\left(\frac{\theta^{-1}s+\theta Qx}{\sqrt{2}\sqrt{x^T s+\sqrt{x^T Qxs^T Qs}}}\right) \\ &= \frac{2\left(x^T s+\sqrt{x^T Qxs^T Qs}\right)}{2\left(x^T s+\sqrt{x^T Qxs^T Qs}\right)} \\ &= 1. \end{aligned}$$

This shows that

$$g_1^2 - \|g_{2:n}\|^2 = 1. \tag{1.1.23}$$

Again, by Definition 1.1.9 we see that

$$\begin{aligned} GQG &= \left(-Q + \frac{(e_1+g)(e_1+g)^T}{1+e_1^T g}\right) QG \\ &= \left(-I + \frac{(e_1+g)(Qe_1+Qg)^T}{1+e_1^T g}\right) G \\ &= \left(-I + \frac{(e_1+g)(Qe_1+Qg)^T}{1+e_1^T g}\right)\left(-Q + \frac{(e_1+g)(e_1+g)^T}{1+e_1^T g}\right) \\ &= Q - \frac{(e_1+g)(e_1+g)^T}{1+e_1^T g} - \frac{(e_1+g)(e_1+g)^T}{1+e_1^T g} + \frac{(e_1+g)(e_1+g)^T Q(e_1+g)(e_1+g)^T}{(1+e_1^T g)^2} \\ &= Q - \frac{2(e_1+g)(e_1+g)^T}{1+e_1^T g} + \frac{(e_1+g)((1+g_1)^2-\|g_{2:n}\|^2)(e_1+g)^T}{(1+e_1^T g)^2}. \end{aligned} \tag{1.1.24}$$

It follows from (1.1.23) and (1.1.24) that

$$GQG = Q - \frac{2(e_1+g)(e_1+g)^T}{1+e_1^T g} + \frac{2(e_1+g)(e_1+g)^T}{1+e_1^T g},$$

which implies $GQG = Q$, as required. Also, by Definition 1.1.9, we see matrix G is symmetric and positive definite. This completes the proof when $K = K_q$.

Case 3: K is the rotated quadratic cone. Assume that K is a rotated quadratic cone K_r in $\mathbb{R}^n$, and K_q is the corresponding quadratic cone in $\mathbb{R}^n$. Let G and G_q be the matrices for K_r and K_q given in Definition 1.1.9, Q and Q_q for K_r and K_q given in Definition 1.1.6, respectively. Then it follows from Definition 1.1.9 that

$$G = TG_qT, \tag{1.1.25}$$

where T is the matrix for K_r given in Definition 1.1.6. Therefore, by what is proved in Case 2, we find that

$$\begin{aligned} GQG &= (TG_qT)Q(TG_qT) = (TG_q)(TQT)(G_qT) \\ &= (TG_q)Q_q(G_qT) = T(G_qQ_qG_q)T = TQ_qT \\ &= Q, \end{aligned}$$

as required. Since G_q is symmetric and positive definite, it follows from 1.1.25 that G is also a symmetric and positive definite matrix. The proof is complete. ∎

Next, we calculate the inverse of the NT scaling matrices that are needed when we solve SOCO problems.

Theorem 1.1.5 *Assume that K is the quadratic or rotated quadratic cone, and G is the scaling matrix given in Definition 1.1.9. Then we have*

$$G^{-1} = QGQ, \tag{1.1.26}$$
$$G^2 = -Q + 2gg^T, \tag{1.1.27}$$
$$G^{-2} = -Q + 2(Qg)(Qg)^T, \tag{1.1.28}$$
$$s = \theta^2 G^2 x. \tag{1.1.29}$$

Proof. (i) Proof of (1.1.26). Since G is a scaling matrix of cone K, we get

$$GQG = Q.$$

Therefore, multiplying both sides by Q yields

$$GQGQ = QQ = I.$$

It follows that

$$G^{-1} = QGQ,$$

which is the same as (1.1.26).

(ii) Proof of (1.1.27). We first show that (1.1.27) is true when K is the quadratic cone K_q. In this case, by Definition 1.1.9, we have

$$G = -Q + \frac{(e_1+g)(e_1+g)^T}{1+e_1^T g} = \begin{pmatrix} g_1 & g_{2:n}^T \\ g_{2:n} & I + \frac{(g_{2:n})(g_{2:n})^T}{1+g_1} \end{pmatrix}.$$

Therefore, we get

$$G^2 = \begin{pmatrix} g_1 & g_{2:n}^T \\ g_{2:n} & I + \frac{(g_{2:n})(g_{2:n})^T}{1+g_1} \end{pmatrix} \begin{pmatrix} g_1 & g_{2:n}^T \\ g_{2:n} & I + \frac{(g_{2:n})(g_{2:n})^T}{1+g_1} \end{pmatrix}$$

$$= \begin{pmatrix} g_1^2 + \|g_{2:n}\|^2 & g_1 g_{2:n}^T + g_{2:n}^T + \frac{\|g_{2:n}\|^2 g_{2:n}^T}{1+g_1} \\ g_1 g_{2:n} + g_{2:n} + \frac{\|g_{2:n}\|^2 g_{2:n}}{1+g_1} & g_{2:n} g_{2:n}^T + I + \frac{2g_{2:n}g_{2:n}^T}{1+g_1} + \frac{\|g_{2:n}\|^2 g_{2:n} g_{2:n}^T}{(1+g_1)^2} \end{pmatrix}.$$

Using (1.1.23), we find from the above that

$$G^2 = \begin{pmatrix} 2g_1^2 - 1 & 2g_1 g_{2:n}^T \\ 2g_1 g_{2:n} & I + 2g_{2:n} g_{2:n}^T \end{pmatrix} = -Q + 2gg^T,$$

which means that (1.1.27) is true for the quadratic cone. Next, we prove (1.1.27) holds for the rotated quadratic cone K_r. In this case, we adopt the notations used in (1.1.25). We also denote by g and g_q the vectors for K_r and K_q defined by (1.1.20), respectively, and T the the matrix associated with K_r. Then it follows from (1.1.20) that

$$g = T g_q. \tag{1.1.30}$$

By (1.1.25), (1.1.30) and (1.1.27) for K_q, we find that

$$\begin{aligned} G^2 &= (TG_qT)(TG_qT) = TG_q(TT)G_qT = TG_q^2T \\ &= T(-Q_q + 2g_q g_q^T)T = -TQ_qT + 2(Tg_q)(Tg_q)^T \\ &= -Q + 2gg^T, \end{aligned}$$

as required.

(iii) Proof of (1.1.28). It follows from (1.1.26) and (1.1.27) that

$$\begin{aligned} G^{-2} &= (QGQ)(QGQ) = QG(QQ)GQ = QG^2Q \\ &= Q(-Q + 2gg^T)Q = -Q + 2(Qg)(Qg)^T, \end{aligned}$$

that proves (1.1.28).

(iv) Proof of (1.1.29). It follows from (1.1.27) and (1.1.20) that

$$\begin{aligned}\theta^2 G^2 x &= \theta\left(-Q+2gg^T\right)x = \theta\left(-Qx+2gg^Tx\right)\\ &= \theta^2\left(-Qx+2g\left(\frac{\theta^{-1}s^Tx+\theta x^TQx}{\sqrt{2}\sqrt{x^Ts+\sqrt{x^TQxs^TQs}}}\right)\right)\\ &= \theta^2\left(-Qx+\sqrt{2}g\theta^{-1}\left(s^Tx+\sqrt{x^TQxs^TQs}\right)\right)\\ &= \theta^2\left(-Qx+\theta^{-1}(\theta^{-1}s+\theta Qx)\right) = \theta^2(-Qx+\theta^{-2}s+Qx) = s,\end{aligned}$$

which completes the proof. ∎

Next, we introduce the concept of complementarity condition, which is needed when we transform the SOCO problems into a system of equations.

Definition 1.1.10 *Assume that K is a convex cone and $x, s \in K$. Then the condition $x^Ts = 0$ is called the complementarity condition.*

In order to characterize the complementarity condition, we need the concept of arrow head matrix.

Definition 1.1.11 *Assume that $x = (x_1, \cdots, x_n)^T$ is a vector in $\mathbb{R}^n$. Then the arrow head matrix mat(x) associated with x is given by*

$$\mathrm{mat}(x) = \begin{pmatrix} x_1 & (x_{2:n})^T \\ x_{2:n} & x_1 I \end{pmatrix},$$

where $x_{2:n} = (x_2, \cdots, x_n)^T$.

By simple computations, we can verify that the inverse of an arrow matrix is given by the following formula.

Lemma 1.1.1 *Let* mat*(x) be the arrow head matrix associated with vector $x \in \mathrm{int}(K_q)$. Then the inverse of* mat*(x) is:*

$$(\mathrm{mat}(x))^{-1} = \frac{1}{x_1^2 - \|x_{2:n}\|^2}\begin{pmatrix} x_1 & -(x_{2:n})^T \\ -x_{2:n} & (x_1 - \frac{\|x_{2:n}\|}{x_1})I + \frac{x_{2:n}(x_{2:n})^T}{x_1} \end{pmatrix}.$$

The following statements demonstrates that the complementarity condition can be expressed by arrow head matrices.

Theorem 1.1.6 *Let K be the quadratic or rotated quadratic cone and $x, s \in K$. Then $x^T s = 0$ is equivalent to $XSe = 0$, where $X = \mathrm{mat}(Tx)$, $S = \mathrm{mat}(Ts)$, T is the matrix associated with K, and $e = (1, 0, \cdots, 0)$.*

Proof. By definition, we see

$$XSe = X(Ts) = \begin{pmatrix} x^T s \\ (Tx)_1 (Ts)_{2:n} + (Ts)_1 (Tx)_{2:n} \end{pmatrix}. \tag{1.1.31}$$

In what follows, we first prove $XSe = 0$ implies $x^T s = 0$, and then prove $x^T s = 0$ implies $XSe = 0$.

(i) Proving $XSe = 0$ implies $x^T s = 0$. Assume $XSe = 0$. Then it follows from (1.1.31) that $x^T s = 0$, as desired.

(ii) Proving $x^T s = 0$ implies $XSe = 0$. Assume $x^T s = 0$. Then we want to prove $XSe = 0$. We note that $Tx, Ts \in K_q$. Therefore, by Cauchy inequality, we have

$$\begin{aligned} x^T s &= (Tx)^T (Ts) = (Tx)_1 (Ts)_1 + ((Tx)_{2:n})^T (Ts)_{2:n} \\ &\geq (Tx)_1 (Ts)_1 - \|(Tx)_{2:n}\| \|(Ts)_{2:n}\| \\ &\geq \sqrt{(Tx)_1^2 - \|(Tx)_{2:n}\|^2} \sqrt{(Ts)_1^2 - \|(Ts)_{2:n}\|^2} \\ &\geq 0. \end{aligned} \tag{1.1.32}$$

If $x^T s = 0$, it follows from (1.1.32) that

$$(Tx)_1 (Ts)_1 + ((Tx)_{2:n})^T (Ts)_{2:n} = 0, \tag{1.1.33}$$

and

$$\sqrt{(Tx)_1^2 - \|(Tx)_{2:n}\|^2} = 0, \qquad \sqrt{(Ts)_1^2 - \|(Ts)_{2:n}\|^2} = 0. \tag{1.1.34}$$

Hence, by (1.1.34)we get

$$(Tx)_1 = \|(Tx)_{2:n}\|, \qquad (Ts)_1 = \|(Ts)_{2:n}\|. \tag{1.1.35}$$

Substituting (1.1.35) into (1.1.33), we find

$$\|(Tx)_{2:n}\| \|(Ts)_{2:n}\| + ((Tx)_{2:n})^T (Ts)_{2:n} = 0. \tag{1.1.36}$$

We notice that (1.1.36) implies that the angle between the vectors $(Tx)_{2:n}$ and $(Ts)_{2:n}$ is $-\pi$. Hence there exists λ such that

$$(Tx)_{2:n} = \lambda (Ts)_{2:n}. \tag{1.1.37}$$

Substituting (1.1.37) into (1.1.33), we get

$$(Tx)_1(Ts)_1 + \lambda\|(Ts)_{2:n}\|^2 = 0. \tag{1.1.38}$$

Substituting (1.1.35) into (1.1.38), we find

$$(Ts)_1\left((Tx)_1 + \lambda(Ts)_1\right) = 0. \tag{1.1.39}$$

If $(Ts)_1 \neq= 0$, it follows from (1.1.39) that

$$\lambda = -\frac{(Tx)_1}{(Ts)_1}. \tag{1.1.40}$$

Substituting (1.1.40) into (1.1.37), we have

$$(Tx)_{2:n} = -\frac{(Tx)_1}{(Ts)_1}(Ts)_{2:n},$$

that is,

$$(Tx)_1(Ts)_{2:n} + (Ts)_1(Tx)_{2:n} = 0. \tag{1.1.41}$$

So far, we have proved that (1.1.41) holds if $(Ts)_1 \neq 0$. If $(Ts)_1 = 0$, then it follows from (1.1.35) that $(Ts)_{2:n} = 0$. Therefore, in this case, (1.1.41) is also valid. In a word, we find that if $x^T s = 0$, then

$$\begin{cases} x^T s = 0, \\ \\ (Tx)_1(Ts)_{2:n} + (Ts)_1(Tx)_{2:n} = 0. \end{cases} \tag{1.1.42}$$

This along with (1.1.31) implies $XSe = 0$. The proof is complete. ∎

After discussion of quadratic cones, we are now ready to present the primal and dual SOCO problems.

1.2 Duality Theory

In this section, we introduce the primal SOCO problems and dual SOCO problems. We also review the duality theory for conic optimization. All of these results are well-known, and the reader may consult, e.g., book [23].

The primal SOCO problem in standard form is stated as follows:

$$\begin{array}{ll} \text{minimize} & c^T x \\ \text{subject to} & Ax = b, \\ & x \in K, \end{array} \tag{P}$$

where $x, c \in I\!R^n$, $b \in I\!R^m$, A is an $m \times n$ matrix with $\text{rank}(A) = m$ and $m \leq n$, $K = K_1 \times K_2 \times \cdots \times K_k$ with K_i $(i = 1, \cdots, k)$ being a linear, quadratic or rotated quadratic cone.

For a SOCO problem, the vector x is called the *decision variable*, the linear function $c^T x$ is called the *objective function*, $Ax = b$ and $x \in K$ are said to be the *constraints*. Let $\mathcal{F}_P = \{x : \ Ax = b, \ x \in K\}$. Then the following concepts are standard. If $\mathcal{F}_P$ is nonempty, we say problem (P) is *feasible*. Otherwise, we say problem (P) is *infeasible*. If $x \in \mathcal{F}_P$, we say x is a *feasible solution*; if $x \in \mathcal{F}_P$ and $x \in \text{int}(K)$, then we say x is a *strictly feasible solution*. If $x^* \in \mathcal{F}_P$ and $c^T x^* \leq c^T x$ for all $x \in \mathcal{F}_P$, then x^* is called a *primal optimal solution*. Problem (P) is said to be *strictly feasible* if it has at least one strictly feasible solution.

In the sequel, a point $x \in K$ is partitioned according to its components in the individual cones K_i, $(i = 1, \cdots, k)$, i.e.,

$$x = \begin{pmatrix} x_{(1)} \\ x_{(2)} \\ \vdots \\ x_{(k)} \end{pmatrix}, \tag{1.2.1}$$

where $x_{(i)} \in K_i$, $(i = 1, \cdots, k)$. If $x \in K$, then we use $\text{mat}(x)$ to represent the block-diagonal matrix with each block being the arrow head matrix $\text{mat}(x_{(i)})$, $(i = 1, \cdots, k)$, which is given in Definition 1.1.11, i.e.,

$$\text{mat}(x) := \begin{pmatrix} \text{mat}(x_{(1)}) & 0 & \cdots & 0 \\ 0 & \text{mat}(x_{(2)}) & \vdots & \vdots \\ \vdots & \cdots & \ddots & 0 \\ 0 & \cdots & 0 & \text{mat}(x_{(k)}) \end{pmatrix}. \tag{1.2.2}$$

We also use the following notations:

$$T := \begin{pmatrix} T_1 & 0 & \cdots & 0 \\ 0 & T_2 & \vdots & \vdots \\ \vdots & \cdots & \ddots & 0 \\ 0 & \cdots & 0 & T_k \end{pmatrix}, \tag{1.2.3}$$

and

$$Q := \begin{pmatrix} Q_1 & 0 & \cdots & 0 \\ 0 & Q_2 & \vdots & \vdots \\ \vdots & \cdots & \ddots & 0 \\ 0 & \cdots & 0 & Q_k \end{pmatrix}, \tag{1.2.4}$$

where T_i and Q_i are matrices associated with K_i, $(i = 1, \cdots, k)$, given in Definition 1.1.6. Also, for all $x, s \in \text{int}(K)$, let θ_i and G_i be the positive number and the NT scaling matrix for K_i, $(i = 1, \cdots, k)$, defined in Definition 1.1.9, respectively. Then we introduce the notations:

$$G := \begin{pmatrix} G_1 & 0 & \cdots & 0 \\ 0 & G_2 & \vdots & \vdots \\ \vdots & \cdots & \ddots & 0 \\ 0 & \cdots & 0 & G_k \end{pmatrix}, \tag{1.2.5}$$

and

$$\Theta := \begin{pmatrix} \theta_1 I_{n_1} & 0 & \cdots & 0 \\ 0 & \theta_2 I_{n_2} & \vdots & \vdots \\ \vdots & \cdots & \ddots & 0 \\ 0 & \cdots & 0 & \theta_k I_{n_k} \end{pmatrix}, \tag{1.2.6}$$

where n_i, $(i = 1, \cdots, k)$, is the dimension of K_i, i.e., $K_i \subset I\!R^{n_i}$. Further, e is used to denote the following constant vector:

$$e = \begin{pmatrix} e^{(1)} \\ e^{(2)} \\ \vdots \\ e^{(k)} \end{pmatrix}, \tag{1.2.7}$$

where $e^{(i)} \in K_i$, $(i = 1, \cdots, k)$, with the first component being 1 and all other components being 0.

In order to explore the properties of the primal SOCO problem (P), we need to introduce another SOCO problem, the so-called dual SOCO problem. The standard form of the dual problem associated with problem (P) is given by

$$\begin{aligned} \text{maximize} \quad & b^T y \\ \text{subject to} \quad & A^T y + s = c, \\ & s \in K^*, \end{aligned}$$

where $y \in I\!R^m$, $s \in I\!R^n$, and K^* is the dual cone of K. Since K is the product of linear, quadratic and rotated quadratic cones, it follows from Theorem 1.1.2 that $K^* = K$. Hence the dual problem can be actually rewritten as

$$\begin{array}{ll} \text{maximize} & b^T y \\ \text{subject to} & A^T y + s = c, \\ & s \in K. \end{array} \tag{D}$$

Analogous to the primal SOCO problem, we have the following concepts for the dual problem (D). Let $\mathcal{F}_D = \big\{(y, s) : \ A^T y + s = c, \ s \in K\big\}$. Then we say problem (D) is *feasible* if $\mathcal{F}_D$ is nonempty. Otherwise, we say problem (D) is *infeasible*. If $(y, s) \in \mathcal{F}_D$, we say (y, s) is a *dual feasible solution*; if $(y, s) \in \mathcal{F}_D$ and $s \in \text{int}(K)$, then we say (y, s) is a *strictly dual feasible solution*. If $(y^*, s^*) \in \mathcal{F}_D$ and $b^T y^* \geq b^T y$ for all $(y, s) \in \mathcal{F}_D$, then (y^*, s^*) is called a *dual optimal solution*. If x^* and (y^*, s^*) are primal optimal solution and dual optimal solution, respectively, then we say (x^*, y^*, s^*) is a *primal-dual optimal solution*. The dual problem (D) is said to be *strictly feasible* if it has at least one *strictly dual feasible solution*.

We notice that our goal is to solve the primal SOCO problems. One may ask why we need to introduce the dual SOCO problems. This is because that the dual problems are closely related to the primal problems, and we can get useful information about the solutions of primal problems by analyzing the corresponding dual problems. The relations between the solutions of primal problems and dual problems are revealed by the duality theory. In order to deal with the duality theory, we need the following terminology.

For any $x \in I\!R^n$, $y \in I\!R^m$ and $s \in I\!R^n$, the difference between the objective values of the primal problem (P) and dual problem (D), $c^T x - b^T y$, is called the *duality gap*, and the value $x^T s$ is called the *complementarity gap*. If further $x, s \in K$ and $x^T s = 0$, then we say x and s are *complementary*.

We are now in a position to describe the duality relations of problems (P) and (D).

Theorem 1.2.1 [Weak Duality Theorem] *Let x be a primal feasible solution and (y, s) be a dual feasible solution. Then, the primal objective value is bounded from below by the dual objective value, i.e.,*

$$c^T x \geq b^T y.$$

Furthermore, $c^T x - b^T y = x^T s \geq 0$, and $x^T s = 0$ implies that x is an optimal solution of (P) and (y, s) is an optimal solution of (D).

Proof. It follows from (P) and (D) that

$$c^T x - b^T y = x^T c - (Ax)^T y = x^T (c - A^T y) = x^T s \geq 0, \tag{1.2.8}$$

since $x,\ s \in K$. By (1.2.8), we see that the dual objective value $b^T y$ is always a lower bound of the primal objective value $c^T x$ as long as x and (y, s) are feasible solutions for (P) and (D), respectively. Therefore, if x and (y, s) are feasible solutions with zero duality gap, i.e., $c^T x = b^T y$, then x must be an optimal solution for (P) and (y, s) an optimal solution for (D). ∎

We notice that in order to solve problems (P) and (D), we need to find a pair of primal and dual feasible solutions with zero duality. Now, the question is: under what condition, there exist such primal and dual feasible solutions. The answer of this question is provided by the strong duality theorem.

Theorem 1.2.2 [Strong Duality Theorem] *If both primal problem (P) and dual problem (D) are strictly feasible, then problems (P) and (D) have optimal solutions with equal optimal values.*

Proof. We refer the readers to [5] for proof of this theorem. Since the proof is lengthy, the details are omitted here. ∎

We notice that, by Theorem 1.2.2, if both primal problem (P) and dual problem (D) are strictly feasible, then the optimal values of both problems are attainable. In this case, we say problems (P) and (D) are both *solvable*. However, if the dual problem (D) is not strictly feasible, then the optimal value of the primal problem (P) may be unattainable, and vice versa. We refer the reader to [5] for more results in this case.

The Strong Duality Theorem demonstrates that the strict feasible solutions of the primal and dual problems are so important to find the optimal solutions. This leads to the following definition for the so called *interior point condition*.

Definition 1.2.1 [Interior Point Condition] *(IPC) We say problems (P) and (D) satisfy the interior point condition if there exist $x \in \mathcal{F}_P$ and $(y, s) \in \mathcal{F}_D$ such that $x,\ s \in \text{int}(K)$.*

Assume the IPC holds. Then, Theorem 1.2.1 and Theorem 1.2.2 imply that solving primal problem (P) or dual problem (D) is equivalent to solving the system:

$$\begin{aligned} Ax - b &= 0, \qquad x \in K, \\ A^T y + s - c &= 0, \qquad s \in K, \\ x^T s &= 0, \end{aligned} \tag{1.2.9}$$

where the first line is the primal feasibility, the second line is the dual feasibility, and the last line is the complementarity condition. System (1.2.9) is referred to as optimality conditions. By Theorem 1.2.1, it follows that system (1.2.9) is actually equivalent to

$$\begin{aligned} Ax - b &= 0, \qquad x \in K, \\ A^T y + s - c &= 0, \qquad s \in K, \\ b^T y - c^T x &\geq 0. \end{aligned} \tag{1.2.10}$$

This means that, if the IPC holds, then the optimal solutions of problems (P) and (D) can be obtained by the solutions of system (1.2.10). In the next section, we discuss how to solve this system.

1.3 The Goldman-Tucker Homogeneous Model

We have seen, in Section 2, that solving a SOCO problem is equivalent to solving the corresponding system (1.2.10) for the optimality conditions. In this section, we present a method to find a solution for system (1.2.10), if a solution exists, or to detect the infeasibility of system (1.2.10). All results in this section are well-known, and the reader may consult, e.g., book [29].

In order to solve system (1.2.10), we first introduce a slack variable κ for the inequality:

$$\begin{aligned} Ax - b &= 0, \\ A^T y + s - c &= 0, \\ b^T y - c^T x - \kappa &= 0, \end{aligned} \tag{1.3.1}$$

where $x, s \in K,\ \kappa \geq 0$. Now, the question is: how to solve system (1.3.1). Here the idea is to homogenize system (1.3.1) first, and then solve the resulting homogeneous model somehow. By homogenizing (1.3.1), we arrive at the following system of equations, which is called the Goldman-Tucker homogeneous model:

$$\begin{aligned} Ax - b\tau &= 0, \\ A^T y + s - c\tau &= 0, \\ b^T y - c^T x - \kappa &= 0, \end{aligned} \tag{1.3.2}$$

where $x,\ s \in K,\ \tau \geq 0,\ \kappa \geq 0$.

We mention that the Goldman-Tucker homogeneous models have been studied by many authors and we refer the readers to [10, 17, 25] for more details. From

(1.3.2) we see that the Goldman-Tucker homogeneous model is closely related to the original system (1.3.1). We can obtain important information about the solutions of system (1.3.1) by examining the solutions of the Goldman-Tucker model. The relations between the Goldman-Tucker homogeneous model and the original system (1.3.1) can be described as follows.

Theorem 1.3.1 *Assume that (x, τ, y, s, κ) is a solution of (1.3.2). Then the following holds:*

(i) either $\tau = 0$ or $\kappa = 0$, or both;

(ii) $x^T s + \tau\kappa = 0$;

(iii) If $\tau > 0$, then $(x, y, s, \kappa)/\tau$ is a solution of (1.3.1), and $(x, y, s)/\tau$ is a primal-dual optimal solution;

(iv) If $\kappa > 0$, then (1.3.1) is unsolvable and at least one of the strict inequalities

$$b^T y > 0 \tag{1.3.3}$$

and

$$c^T x < 0 \tag{1.3.4}$$

holds. If (1.3.3) holds, then the primal problem (P) is infeasible. If (1.3.4) holds, then the dual problem (D) is infeasible.

Proof. (a) Proof of (i). To this end, we first show $\tau\kappa \leq 0$, which can be derived by starting with the third equation in (1.3.2), i.e.,

$$\begin{aligned} \tau\kappa &= \tau(b^T y - c^T x) = (b\tau)^T y - (c\tau)^T x \\ &= (Ax)^T y - (A^T y + s)^T x = y^T Ax - y^T Ax - s^T x = -s^T x \\ &\leq 0. \end{aligned}$$

The last inequality follows from the fact $x, s \in K$ and $K^* = K$. On the other hand, since $\tau \geq 0$ and $\kappa \geq 0$ we have $\tau\kappa \geq 0$. Therefore it follows that $\tau\kappa = 0$, which implies (i).

(b) Proof of (ii). It follows from (1.3.2) that

$$\begin{aligned} x^T s + \tau\kappa &= x^T(-A^T y + c\tau) + \tau\kappa \\ &= -x^T A^T y + \tau x^T c + \tau\kappa \\ &= -(Ax)^T y + \tau c^T x + \tau\kappa \\ &= -\tau b^T y + \tau c^T x + \tau\kappa \\ &= -\tau(b^T y - c^T x - \kappa) \\ &= 0, \end{aligned}$$

which is the same as (ii).

(c) Proof of (iii). Let $\tau > 0$ and $(\bar{x}, \bar{y}, \bar{s}, \bar{\kappa}) = (\frac{x}{\tau}, \frac{y}{\tau}, \frac{s}{\tau}, \frac{\kappa}{\tau})$. Then, by (i), we have $\bar{\kappa} = 0$. Therefore, it follows from (1.3.2) that

$$\begin{aligned} A\bar{x} - b &= 0, \qquad \bar{x} \in K, \\ A^T\bar{y} + \bar{s} - c &= 0, \qquad \bar{s} \in K, \\ b^T\bar{y} - c^T\bar{x} &= 0, \end{aligned}$$

which implies that $(\bar{x}, \bar{y}, \bar{s}, \bar{\kappa})$ is a solution of (1.3.1), and $(\bar{x}, \bar{y}, \bar{s})$ is a pair of primal-dual optimal solutions.

(d) Proof of (iv). Step 1: if $\kappa > 0$, then $b^T y - c^T x = \kappa > 0$, which implies at least one of $b^T y > 0$ and $c^T x < 0$ is true.

Step 2: if $\kappa > 0$ and $b^T y > 0$, then we want to prove that the primal problem (P) is infeasible. We argue by contradiction. Assume that the primal problem (P) is feasible, that is, there exist $\tilde{x} \in K$ such that

$$A\tilde{x} - b = 0. \tag{1.3.5}$$

Since $\kappa > 0$, it follows from (i) that $\tau = 0$. Then by system (1.3.2) we have

$$A^T y + s = 0, \qquad s \in K. \tag{1.3.6}$$

Now, let us calculate $b^T y$ in this case. By (1.3.5) and (1.3.6), we find that

$$b^T y = (A\tilde{x})^T y = \tilde{x}^T A^T y = -\tilde{x}^T s \leq 0,$$

which contradicts $b^T y > 0$. Therefore, the primal problem (P) must be infeasible in this case.

Step 3: if $\kappa > 0$ and $c^T x < 0$, then we want to prove that the dual problem (D) is infeasible. We argue again by contradiction. Assume that the dual problem (D) is feasible, that is, there exist $\tilde{y} \in I\!R^m$ and $\tilde{s} \in K$ such that

$$A^T\tilde{y} + \tilde{s} - c = 0. \tag{1.3.7}$$

Now, let us calculate $c^T x$ in this case. Since $\kappa > 0$, by (i) we have $\tau = 0$. This along with system (1.3.2) implies $Ax = 0$. Then, it follows from (1.3.7) that

$$c^T x = (A^T\tilde{y} + \tilde{s})^T x = \tilde{y}^T Ax + \tilde{s}^T x = \tilde{s}^T x \geq 0,$$

which contradicts $c^T x < 0$. Therefore, the dual problem (D) must be infeasible in this case.

Step 4: if $\kappa > 0$, then we want to show that system (1.3.1) is unsolvable. Since $\kappa > 0$, it follows from Steps 2 and 3 that either the primal problem (P) or the dual problem (D) is infeasible. This implies that system (1.3.1) is unsolvable. The proof is complete. ∎

We mention that, in practice, for numerical reason we consider τ (or κ) to be zero if τ (or κ) is less than a specified small positive number; otherwise, we consider τ (or κ) to be positive. From Theorem 1.3.1, we see that any solution of the Goldman-Tucker homogeneous model with positive τ or κ can produce an optimal solution or certificate of infeasibility for the primal problem (P) or dual problem (D). Therefore, problem (P) and problem (D) can be solved by first finding a solution of the corresponding homogeneous model. This leads to the following question: how to solve the Goldman-Tucker homogeneous models. Here, we use interior point methods (IPMs) to find a solution for the homogeneous models. As we will see later, in order to use IPMs, we first need a strictly feasible starting point, and the existence of such a point is also needed to ensure the solvability of the optimality conditions. But from Theorem 1.3.1 we see that the Goldman-Tucker homogeneous model does not have a strictly feasible point because for any solution, either $\tau = 0$ or $\kappa = 0$. In order to overcome this difficulty, we introduce a new optimization problem that always possesses a strictly feasible solution and provides desired information about the solutions of the Goldman-Tucker homogeneous model and the original optimization problems (P) and (D).

Given vectors $x^{(0)}, s^{(0)} \in \text{int}(K)$, $y^{(0)} \in I\!R^m$, and positive numbers $\tau^{(0)}, \kappa^{(0)}, \nu^{(0)}$, we introduce the following optimization problem for the Goldman-Tucker homogeneous model:

$$
\begin{array}{lrcl}
\text{minimize} & \beta\nu & & \\
\text{subject to} & Ax - b\tau - r_p\nu & = & 0, \\
& -A^T y + c\tau - s - r_d\nu & = & 0, \\
& b^T y - c^T x - \kappa - r_g\nu & = & 0, \\
& r_p^T y + r_d^T x + r_g\tau & = & -\beta, \\
& y \text{ is free, } x,\ s \in K, & \tau, & \kappa \geq 0,\ \nu \text{ is free,}
\end{array}
\tag{1.3.8}
$$

where

$$\begin{aligned} r_p &= (Ax^{(0)} - b\tau^{(0)})/\nu^{(0)}, \\ r_d &= (-A^T y^{(0)} + c\tau^{(0)} - s^{(0)})/\nu^{(0)}, \\ r_g &= (b^T y^{(0)} - c^T x^{(0)} - \kappa^{(0)})/\nu^{(0)}, \\ \beta &= -(r_p^T y^{(0)} + r_d^T x^{(0)} + r_g \tau^{(0)}). \end{aligned} \tag{1.3.9}$$

Here we notice that r_p, r_d and r_g represent the infeasibility of the initial point for the Goldman-Tucker homogeneous model. Actually, if the initial point $(x^{(0)}, \tau^{(0)}, s^{(0)}, \kappa^{(0)}, y^{(0)})$ satisfies the Goldman-Tucker homogeneous model, then r_p, r_d and r_g are all equal to zero. Furthermore, if $\nu = 0$, then the first three constraints of (1.3.8) reduce to the Goldman-Tucker homogeneous model. The new variable ν is introduced to make sure that every given initial point is feasible for problem (1.3.8). The last constraint is added in order to simplify the dual problem of (1.3.8). In fact, the dual problem of (1.3.8) is equivalent to itself and this explains why optimization problem (1.3.8) is called a self-dual model. The important features of problem (1.3.8) are stated as follows.

Theorem 1.3.2 *The optimization problem (1.3.8) has the following properties.*

(i) Problem (1.3.8) is self-dual, that is, the dual problem of (1.3.8) takes the same form as itself.

(ii) If $x^{(0)} \in \text{int}(K),\ \tau^{(0)} > 0,\ y^{(0)} \in I\!R^m,\ s^{(0)} \in \text{int}(K),\ \kappa^{(0)} > 0,\ \nu^{(0)} > 0,$ *then* $\left(x^{(0)},\ \tau^{(0)},\ y^{(0)},\ s^{(0)},\ \kappa^{(0)},\ \nu^{(0)}\right)$ *is a strictly feasible point for problem (1.3.8).*

(iii) The optimal value of problem (1.3.8) is zero. Furthermore, if $(x, \tau, y, s, \kappa, \nu)$ *is a feasible solution, then* $\beta\nu = x^T s + \tau\kappa$.

Proof. (a) Proof of (i). In order to find the dual problem for (1.3.8), we first have to transform (1.3.8) into the standard form of primal problem (P). To this end, we split the free variables y and ν as usual: $y = y^+ - y^-$ and $\nu = \nu^+ - \nu^-$ with

$$y^+ = \max\{y, 0\},\ \ y^- = -\min\{y, 0\},\ \ \nu^+ = \max\{\nu, 0\},\ \ \nu^- = -\min\{\nu, 0\}.$$

Then problem (1.3.8) can be rewritten as:

$$\begin{aligned} \text{minimize} \quad & \bar{c}^T \bar{x} \\ \text{subject to} \quad & \bar{A}\bar{x} = \bar{b}, \\ & \bar{x} \in \bar{K}, \end{aligned} \tag{1.3.10}$$

where

$$\bar{c} = \begin{pmatrix} 0, & 0, & 0, & 0, & 0, & 0, & \beta, & -\beta \end{pmatrix}^T,$$

$$\bar{x} = \begin{pmatrix} x, & \tau, & y^+, & y^-, & s, & \kappa, & \nu^+, & \nu^- \end{pmatrix}^T,$$

$$\bar{b} = \begin{pmatrix} 0, & 0, & 0, & -\beta \end{pmatrix}^T,$$

$$\bar{A} = \begin{pmatrix} A & -b & 0 & 0 & 0 & 0 & -r_p & r_p \\ 0 & c & -A^T & A^T & -I & 0 & -r_d & r_d \\ -c^T & 0 & b^T & -b^T & 0 & -1 & -r_g & r_g \\ r_d^T & r_g & r_p^T & -r_p^T & 0 & 0 & -r_p & r_p \end{pmatrix},$$

$$\bar{K} = K \times I\!R_+ \times (I\!R_+)^m \times (I\!R_+)^m \times K \times I\!R_+ \times I\!R_+ \times I\!R_+.$$

Then by the definition of dual problem (D), after some computations, we find that the dual problem for problem (1.3.10) is given by the following:

$$\begin{array}{lll} - & \text{minimize} & \bar{c}^T\tilde{x} \\ & \text{subject to} & \bar{A}\tilde{x} = \bar{b}, \\ & & \tilde{x} \in \bar{K}, \end{array} \tag{1.3.11}$$

from which, we see that the dual problem (1.3.11) has the same form as the primal problem (1.3.8).

(b) Proof of (ii). Since the point $\left(x^{(0)},\ \tau^{(0)},\ y^{(0)},\ s^{(0)},\ \kappa^{(0)},\ \nu^{(0)}\right)$ is an interior point and it satisfies all constraints in (1.3.8), by definition, it is a strictly feasible point for (1.3.8).

(c) Proof of (iii). We first prove that the optimal value is zero. By (ii), we know that problem (1.3.8) is strictly feasible. This along with (i) implies that the dual problem of (1.3.8) is also strictly feasible. Therefore, it follows from the Strong Duality Theorem that the optimal values of problem (1.3.8) and its dual problem are equal. Assume that χ is the optimal value of problem (1.3.8). Then it follows from (1.3.11) that the optimal value of the dual problem is $-\chi$. Therefore, at optimality, we have $\chi = -\chi$, which implies that $\chi = 0$, i.e., the optimal value of problem (1.3.8) is zero.

Next, we prove $\beta\nu = x^T s + \tau\kappa$. Multiplying the equations in (1.3.8) by y^T, x^T, τ, and ν, respectively, then adding up the resulting equalities, we get:

$$\begin{aligned} & y^T(Ax - b\tau - r_p\nu) + x^T(-A^T y + c\tau - s - r_d\nu) \\ & +\tau(b^T y - c^T x - \kappa - r_g\nu) + \nu(r_p^T y + r_d^T x + r_g\tau) \quad = \quad -\beta\nu. \end{aligned}$$

Simplifying the right-hand side yields:

$$-x^T s - \tau\kappa = -\beta\nu,$$

which implies the desired result. The proof is complete. ∎

Here, we mention that the self-dual homogeneous models have been studied by many authors for a variety of optimization problems, including linear optimization and conic optimization problems. For more details, we refer the reader to [3, 10, 28, 29, 30, 32] and the references therein.

Next, let us look at the relations between the solutions of problem (1.3.8) and solutions of problems (P) and (D). Assume that $\left(x^{(0)},\ \tau^{(0)},\ y^{(0)},\ s^{(0)},\ \kappa^{(0)},\ \nu^{(0)}\right)$ is a strictly feasible point, that is, $x^{(0)},\ s^{(0)} \in \text{int}(K)$, $\tau^{(0)},\ \ \kappa^{(0)},\ \nu^{(0)} > 0$. Then by Theorem 1.3.2 we see that

$$\beta = \frac{(x^{(0)})^T s^{(0)} + \tau^{(0)}\kappa^{(0)}}{\nu^{(0)}} > 0. \tag{1.3.12}$$

If $(x, \tau, y, s, \kappa, \nu)$ is an optimal solution of (1.3.8), then by Theorem 1.3.2 we find that the optimal objective value $\beta\nu$ must be zero. This along with (1.3.12) implies that $\nu = 0$. In this case, the first three constraints of problem (1.3.8) reduce to

$$\begin{aligned} Ax - b\tau &= 0, \\ -A^T y + c\tau - s &= 0, \\ b^T y - c^T x - \kappa &= 0, \end{aligned} \tag{1.3.13}$$

which shows that (x, τ, y, s, κ) is a solution of the Goldman-Tucker homogeneous model. From the solution of Goldman-Tucker homogeneous model, we can apply Theorem 1.3.1 and then acquire information about the solutions of the original optimization problem (P). The relations between the solutions of problem (1.3.8) and the primal problem (P) are summarized as follows.

Corollary 1.3.1 *Assume that $(x, \tau, y, s, \kappa, \nu)$ is an optimal solution of (1.3.8). Then the following is true.*

(i) If $\tau > 0$, then $(x, y, s)/\tau$ is a pair of primal-dual optimal solutions for problems (P) and (D).

(ii) If $\kappa > 0$, then at least one of the strict inequalities $b^T y > 0$ and $c^T x < 0$ holds. If the first inequality holds, then the primal problem (P) is infeasible. If the second inequality holds, then the dual problem (D) is infeasible.

By Corollary 1.3.1, we see that solving problem (P) amounts to solving the self-dual model (1.3.8). Once we get an optimal solution for problem (1.3.8), then either an optimal solution or certificate of infeasibility for the original problem (P) follows immediately. Now, the question is: how to find an optimal solution for the artificial problem (1.3.8). In order to answer this question, we notice that a point is an optimal solution for problem (1.3.8) if and only if it is a feasible point with zero objective value, i.e., $\beta\nu = 0$. By Theorem 1.3.2, we know that $\beta\nu = x^T s + \tau\kappa$. Therefore, finding an optimal solution for problem (1.3.8) is equivalent to solving the following system of equations:

$$\begin{aligned} Ax - b\tau - r_p\nu &= 0, \\ -A^T y + c\tau - s - r_d\nu &= 0, \\ b^T y - c^T x - \kappa - r_g\nu &= 0, \\ r_p^T y + r_d^T x + r_g\tau &= -\beta, \\ x^T s + \tau\kappa &= 0, \\ y \text{ is free, } x,\ s \in K, \quad \tau, \quad \kappa \geq 0,\ \nu &\text{ is free.} \end{aligned} \tag{1.3.14}$$

We note that the first four equations are actually the feasibility conditions for problem (1.3.8), whereas, the fifth equation implies zero objective value. Since $x^T s \geq 0$ and $\tau\kappa \geq 0$, it follows from (1.3.14) that both $x^T s = 0$ and $\tau\kappa = 0$. But, by Theorem 1.1.6, we know that $x^T s = 0$ is equivalent to $XSe = 0$, where $X = \text{mat}(Tx)$, $S = \text{mat}(Ts)$, T and e are given by (1.2.3) and (1.2.7), respectively. Therefore, system (1.3.14) can

be rewritten as:

$$\begin{aligned} Ax - b\tau - r_p\nu &= 0, \\ -A^T y + c\tau - s - r_d\nu &= 0, \\ b^T y - c^T x - \kappa - r_g\nu &= 0, \\ r_p^T y + r_d^T x + r_g\tau &= -\beta, \\ XSe &= 0, \\ \tau\kappa &= 0, \\ y \text{ is free, } x,\ s \in K, &\quad \tau, \quad \kappa \geq 0,\ \nu \text{ is free.} \end{aligned} \tag{1.3.15}$$

When we solve system (1.3.15), we actually solve a sequence of perturbed systems. More precisely, we replace the nonlinear equations $XSe = 0$ and $\tau\kappa = 0$ by $XSe = \mu e$ and $\tau\kappa = \mu$, respectively, and then solve the following system with decreasing values of the parameter $\mu > 0$:

$$\begin{aligned} Ax - b\tau - r_p\nu &= 0, \\ -A^T y + c\tau - s - r_d\nu &= 0, \\ b^T y - c^T x - \kappa - r_g\nu &= 0, \\ r_p^T y + r_d^T x + r_g\tau &= -\beta, \\ XSe &= \mu e, \\ \tau\kappa &= \mu, \\ y \text{ is free, } x,\ s \in K, &\quad \tau, \quad \kappa \geq 0,\ \nu \text{ is free.} \end{aligned} \tag{1.3.16}$$

In what follows, we assume that the constraint matrix A has full row rank, i.e., the rank of A is equal to m, the number of the equality constraints in problem (P). Note that this is not a restrictive assumption because if the rank of A is less than m, then either the redundant constraints can be removed or the system $Ax = b$ is not consistent. In the latter case, the primal problem (P) is infeasible. Since the self-dual problem (1.3.8) is strictly feasible, therefore, if the matrix A has full row rank, then it follows from [1, 11, 14, 15] that, for each fixed $\mu > 0$, the perturbed system (1.3.16) has a unique solution, denoted by $(x_\mu, \tau_\mu, y_\mu, s_\mu, \kappa_\mu, \nu_\mu)$. Further-

more, when $\mu \to 0$, the sequence of perturbed solutions $(x_\mu, \tau_\mu, y_\mu, s_\mu, \kappa_\mu, \nu_\mu)$ converges to a point that is a solution of system (1.3.15). Although, for each fixed $\mu > 0$, the solution $(x_\mu, \tau_\mu, y_\mu, s_\mu, \kappa_\mu, \nu_\mu)$ is unique, it is a solution of a nonlinear system, thus we cannot solve the system exactly, and approximate solutions can be attained by some numerical procedures, such as Newton's method. Usually, the set $\{(x_\mu, \tau_\mu, y_\mu, s_\mu, \kappa_\mu, \nu_\mu) : \mu > 0\}$ is called the central path, and thus a solution of self-dual homogeneous model (1.3.8) can be obtained by following the central path as $\mu \to 0$. Now the question is: how to solve the central path as $\mu \to 0$. This is the question we plan to answer in the remaining part of the thesis. Particularly, the algorithms and implementation issues are discussed there.

Chapter 2

Interior Point Methods for SOCO Problems with Newton Search Directions

In this chapter, we present interior point methods to solve SOCO problems with Newton search directions. We first derive the Newton system for the nonlinear system (1.3.16) without using scaling techniques. As illustrated below, the Newton system without scaling is not well-defined. Then, in Section 2, we introduce a scaled Newton system and derive the corresponding scaled Newton search direction. In order to improve the efficiency to solve the Newton system, in Section 3, we discuss how to explore the sparse structure of the linear system and use the sparse matrix package WSMP to find the solutions. After the Newton search direction is found, then we determine the Newton step in Section 4 that guarantees the next iteration point stays in the interior of the cone. In order to reduce the number of iterations and therefore save the computational time, we discuss the Mehrotra's predictor-corrector technique in Section 5. Finally, in Section 6, we present an algorithm to solve SOCO problems by the interior point methods with scaled Newton search directions and the predictor-corrector technique. Let's first look at Newton search directions without scaling.

2.1 Newton Search Directions without Scaling

In this section, we first simplify system (1.3.16) that determines the central path for the self-dual homogeneous model (1.3.8). Then, we derive the Newton search directions for the resulting system without scaling. Finally, an example is presented

that demonstrates the Newton search directions without scaling are not well-defined for general SOCO problems. Therefore, scaled Newton search directions are necessary for solving the central path. We refer the reader to book [29] for more details about Newton search directions.

Let's first simplify system (1.3.16). Multiplying the equations in (1.3.16) in order by y^T, x^T, τ and ν, respectively, and then summing up the resulting equalities, we get

$$\begin{aligned} &y^T(Ax - b\tau - r_p\nu) + x^T(-A^Ty + c\tau - s - r_d\nu) \\ &+\tau(b^Ty - c^Tx - \kappa - r_g\nu) + \nu(r_p^Ty + r_d^Tx + r_g\tau) = -\beta\nu. \end{aligned}$$

Simplifying the left-hand side, we find

$$x^Ts + \tau\kappa = \beta\nu. \tag{2.1.1}$$

On the other hand, by $XSe = \mu e$ and $\tau\kappa = \mu$, we have

$$x^Ts + \tau\kappa = (k+1)\mu. \tag{2.1.2}$$

It follows from (2.1.1) and (2.1.2) that

$$\mu = \frac{\beta}{k+1}\nu. \tag{2.1.3}$$

Substituting (2.1.3) into system (1.3.16) yields

$$\begin{aligned} Ax - b\tau - r_p\nu &= 0, \\ -A^Ty + c\tau - s - r_d\nu &= 0, \\ b^Ty - c^Tx - \kappa - r_g\nu &= 0, \\ r_p^Ty + r_d^Tx + r_g\tau &= -\beta, \\ XSe &= \tfrac{\beta}{k+1}\nu e, \\ \tau\kappa &= \tfrac{\beta}{k+1}\nu. \end{aligned} \tag{2.1.4}$$

Let $\nu^{(0)} = 1$. Then, by (1.3.9), we find that the constant β is given by

$$\begin{aligned} \beta &= -(r_p^Ty^{(0)} + r_d^Tx^{(0)} + r_g\tau^{(0)}) = -(y^{(0)})^T(Ax^{(0)} - b\tau^{(0)}) \\ &\quad -(x^{(0)})^T(-A^Ty^{(0)} + c\tau^{(0)} - s^{(0)}) - \tau^{(0)}(b^Ty^{(0)} - c^Tx^{(0)} - \kappa^{(0)}) \\ &= (x^{(0)})^Ts^{(0)} + \tau^{(0)}\kappa^{(0)} \\ &> 0. \end{aligned} \tag{2.1.5}$$

The last inequality is obtained by the assumption that the initial points $x^{(0)}, s^{(0)} \in \text{int}(K)$ and $\tau^{(0)}, \kappa^{(0)} > 0$. We note that the fourth equation in (2.1.4) is redundant. Actually, multiplying the first three equations in (2.1.4) in order by y^T, x^T and τ, respectively, we obtain

$$\left(r_p^T y + r_d^T x + r_g \tau\right) \nu = -(x^T s + \tau\kappa). \tag{2.1.6}$$

On the other hand, it follows from the last two equations in (2.1.4) that

$$x^T s + \tau\kappa = \beta\nu. \tag{2.1.7}$$

Therefore, by (2.1.6) and (2.1.7), we find

$$\left(r_p^T y + r_d^T x + r_g \tau\right) \nu = -\beta\nu. \tag{2.1.8}$$

Note that (2.1.3) and (2.1.5) imply $\nu > 0$ since μ is a positive parameter. Therefore, (2.1.8) implies the fourth equation in system (2.1.4). By first removing the fourth equation from system (2.1.4), and then substituting (2.1.5) into (2.1.4), we arrive at

$$\begin{aligned} Ax - b\tau - r_p\nu &= 0, \\ -A^T y + c\tau - s - r_d\nu &= 0, \\ b^T y - c^T x - \kappa - r_g\nu &= 0, \\ XSe &= \mu^{(0)}\nu e, \\ \tau\kappa &= \mu^{(0)}\nu, \end{aligned} \tag{2.1.9}$$

where $\mu^{(0)}$ is given by

$$\mu^{(0)} = \frac{(x^{(0)})^T s^{(0)} + \tau^{(0)}\kappa^{(0)}}{k+1}. \tag{2.1.10}$$

By system (1.3.16), we see the central path is parameterized by $\mu > 0$, and we follow the central path as $\mu \to 0$. On the other hand, it follows from (2.1.3) that $\mu \to 0$ is equivalent to $\nu \to 0$. Hence, the central path is actually determined by system (2.1.9) with parameter $\nu > 0$.

Next, we look at how to solve system (2.1.9). Since this is a system of nonlinear equations, generally speaking, it is difficult to find the exact solutions. In this case, we usually try to find the approximate solutions of system (2.1.9) instead of its exact solutions. Newton methods are fundamental tools to solve nonlinear equations approximately. By Newton methods, we actually solve nonlinear equations by

solving a sequence of linearized equations. Assume that the solution for the current iteration is (x, τ, y, s, κ). Then the solution for the next iteration can be written as $(x+\Delta x, \tau+\Delta\tau, y+\Delta y, s+\Delta s, \kappa+\Delta\kappa)$, where $(\Delta x, \Delta\tau, \Delta y, \Delta s, \Delta\kappa)$ is the unknown quantity that we want to solve. Substitute $(x+\Delta x, \tau+\Delta\tau, y+\Delta y, s+\Delta s, \kappa+\Delta\kappa)$ into system (2.1.9). Then we find the following system of nonlinear equations that must be satisfied by $(\Delta x, \Delta\tau, \Delta y, \Delta s, \Delta\kappa)$:

$$\begin{aligned} A\Delta x - b\Delta\tau &= r_p\nu - (Ax - b\tau), \\ -A^T\Delta y + c\Delta\tau - \Delta s &= r_d\nu - (-A^Ty + c\tau - s), \\ b^T\Delta y - c^T\Delta x - \Delta\kappa &= r_g\nu - (b^Ty - c^Tx - \kappa), \\ X(\Delta S)e + S(\Delta X)e &= \nu\mu^{(0)}e - XSe - (\Delta X)(\Delta S)e, \\ \kappa\Delta\tau + \tau\Delta\kappa &= \nu\mu^{(0)} - \kappa\tau - \Delta\kappa\Delta\tau, \end{aligned} \tag{2.1.11}$$

where $X = \mathrm{mat}(Tx)$, $S = \mathrm{mat}(Ts)$, $\Delta X = \mathrm{mat}(T\Delta x)$ and $\Delta S = \mathrm{mat}(T\Delta s)$, i.e.,

$$X = \begin{pmatrix} \mathrm{mat}(T_1x^{(1)}) & 0 & \cdots & 0 \\ 0 & \mathrm{mat}(T_2x^{(2)}) & \vdots & \vdots \\ \vdots & \cdots & \ddots & 0 \\ 0 & \cdots & 0 & \mathrm{mat}(T_kx^{(k)}) \end{pmatrix}, \tag{2.1.12}$$

$$S = \begin{pmatrix} \mathrm{mat}(T_1s^{(1)}) & 0 & \cdots & 0 \\ 0 & \mathrm{mat}(T_2s^{(2)}) & \vdots & \vdots \\ \vdots & \cdots & \ddots & 0 \\ 0 & \cdots & 0 & \mathrm{mat}(T_ks^{(k)}) \end{pmatrix}, \tag{2.1.13}$$

and

$$\Delta X = \begin{pmatrix} \mathrm{mat}(T_1\Delta x^{(1)}) & 0 & \cdots & 0 \\ 0 & \mathrm{mat}(T_2\Delta x^{(2)}) & \vdots & \vdots \\ \vdots & \cdots & \ddots & 0 \\ 0 & \cdots & 0 & \mathrm{mat}(T_k\Delta x^{(k)}) \end{pmatrix}, \tag{2.1.14}$$

$$\Delta S = \begin{pmatrix} \mathrm{mat}(T_1\Delta s^{(1)}) & 0 & \cdots & 0 \\ 0 & \mathrm{mat}(T_2\Delta s^{(2)}) & \vdots & \vdots \\ \vdots & \cdots & \ddots & 0 \\ 0 & \cdots & 0 & \mathrm{mat}(T_k\Delta s^{(k)}) \end{pmatrix}, \tag{2.1.15}$$

where T_i is the matrix associated with cone K_i given in Definition 1.1.6 ($i = 1, \cdots, k$). Ignore the second order term $\Delta X \Delta S e$ and $\Delta\kappa\Delta\tau$ in (2.1.11). Then we get the following system of linear equations:

$$\begin{aligned} A\Delta x - b\Delta\tau &= r_p\nu - (Ax - b\tau), \\ -A^T\Delta y + c\Delta\tau - \Delta s &= r_d\nu - (-A^T y + c\tau - s), \\ b^T\Delta y - c^T\Delta x - \Delta\kappa &= r_g\nu - (b^T y - c^T x - \kappa), \\ X(\Delta S)e + S(\Delta X)e &= \nu\mu^{(0)}e - XSe, \\ \kappa\Delta\tau + \tau\Delta\kappa &= \nu\mu^{(0)} - \kappa\tau. \end{aligned} \tag{2.1.16}$$

Usually, we refer the solution $(\Delta x, \Delta\tau, \Delta y, \Delta s, \Delta\kappa)$ of system (2.1.16) as Newton search direction.

Next, let's solve system (2.1.16) directly and see what problems arise. Multiplying the fourth equality in (2.1.16) by S^{-1}, and then by $XSe = SXe$ we get

$$S^{-1}XT\Delta s + T\Delta x = \nu\mu^{(0)}S^{-1}e - Xe. \tag{2.1.17}$$

Multiplying the second equality in ((2.1.16)) by $S^{-1}XT$ yields

$$-S^{-1}XTA^T\Delta y + S^{-1}XTc\Delta\tau - S^{-1}XT\Delta s = S^{-1}XT(r_d\nu + A^T y - c\tau + s). \tag{2.1.18}$$

Adding (2.1.17) to (2.1.18), we get

$$T\Delta x - S^{-1}XTA^T\Delta y + S^{-1}XTc\Delta\tau = \tilde{r}_1, \tag{2.1.19}$$

where

$$\tilde{r}_1 = \nu\mu^{(0)}S^{-1}e - Xe + S^{-1}XT(r_d\nu + A^T y - c\tau + s).$$

Multiplying (2.1.19) by AT, we find

$$A\Delta x - ATS^{-1}XTA^T\Delta y + ATS^{-1}XTc\Delta\tau = AT\tilde{r}_1. \tag{2.1.20}$$

Subtracting (2.1.20) from the first equality in (2.1.16), we have

$$ATS^{-1}XTA^T\Delta y - b\Delta\tau - ATS^{-1}XTc\Delta\tau = r_p\nu - Ax + b\tau - AT\tilde{r}_1. \tag{2.1.21}$$

From (2.1.21) we see that the unknown variable Δy is given by the solution of the linear system of the form:

$$ATS^{-1}XTA^T\Delta y = \tilde{r}_2, \tag{2.1.22}$$

where

$$\tilde{r}_2 = r_p\nu - Ax + b\tau - AT\tilde{r}_1 + b\Delta\tau + ATS^{-1}XTc\Delta\tau.$$

If for each $i = 1, \cdots, k$, the cone K_i is a linear cone $\mathbb{R}_+$, then the SOCO problem (P) reduces to a linear optimization problem. In this case, T is the identity matrix, X and S are both diagonal matrices with positive entries. Therefore, the matrix $ATS^{-1}XTA^T$ is symmetric and positive definite as long as the matrix A has full row rank. This enables us to solve system (2.1.22) effectively by using Cholesky factorization methods (see, e.g., [13, 32]). But, for a general SOCO problem, the matrix $ATS^{-1}XTA^T$ is neither symmetric nor positive definite, and therefore Cholesky factorization methods do not apply. Next, let's first look at an example that demonstrates the matrix $ATS^{-1}XTA^T$ is not symmetric.

Example 2.1.1 Let $x = (\sqrt{2}, 1, 0)^T$, $s = (\sqrt{2}, 0, 1)^T$ and

$$A = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \end{pmatrix}. \tag{2.1.23}$$

Then $x, s \in \text{int}(K_q) \subseteq \mathbb{R}^3$, and rank$(A)$ =2. In this case, T is the identity matrix. The arrow head matrices X and S, corresponding to x and s, respectively, are given by

$$X = \begin{pmatrix} \sqrt{2} & 1 & 0 \\ 1 & \sqrt{2} & 0 \\ 0 & 0 & \sqrt{2} \end{pmatrix},$$

and

$$S = \begin{pmatrix} \sqrt{2} & 0 & 1 \\ 0 & \sqrt{2} & 0 \\ 1 & 0 & \sqrt{2} \end{pmatrix}.$$

The inverse matrix of S is:

$$S^{-1} = \begin{pmatrix} \sqrt{2} & 0 & -1 \\ 0 & \frac{1}{\sqrt{2}} & 0 \\ -1 & 0 & \sqrt{2} \end{pmatrix}.$$

By simple computations, we find

$$S^{-1}X = \begin{pmatrix} 2 & \sqrt{2} & -\sqrt{2} \\ \frac{1}{\sqrt{2}} & 1 & 0 \\ -\sqrt{2} & -1 & 2 \end{pmatrix}. \tag{2.1.24}$$

It follows from (2.1.23) and (2.1.24) that

$$ATS^{-1}XTA^T = \begin{pmatrix} 2 & \sqrt{2} \\ \frac{1}{\sqrt{2}} & 1 \end{pmatrix},$$

which shows that the matrix $ATS^{-1}XTA^T$ is not symmetric.

Next, we present an example that demonstrates the matrix $ATS^{-1}XTA^T$ may be singular. This example can be found in [18].

Example 2.1.2 Let $x = (1, 0.8, 0.5)^T$, $s = (1, 0.7, 0.7)^T$ and

$$A = \begin{pmatrix} 0 & \sqrt{3.69} + 0.7 & 1 \end{pmatrix}. \tag{2.1.25}$$

Then $x, s \in \text{int}(K_q) \subseteq I\!R^3$, rank$(A)$ =1, $T = I$. The arrow head matrices X and S, corresponding to x and s, are given by

$$X = \begin{pmatrix} 1 & 0.8 & 0.5 \\ 0.8 & 1 & 0 \\ 0.5 & 0 & 1 \end{pmatrix},$$

and

$$S = \begin{pmatrix} 1 & 0.7 & 0.7 \\ 0.7 & 1 & 0 \\ 0.7 & 0 & 1 \end{pmatrix}.$$

The inverse matrix of S is:

$$S^{-1} = \frac{1}{0.02} \begin{pmatrix} 1 & -0.7 & -0.7 \\ -0.7 & 0.51 & 0.49 \\ -0.7 & 0.49 & 0.51 \end{pmatrix}.$$

Hence, we have

$$S^{-1}X = \frac{1}{0.02}\begin{pmatrix} 0.09 & 0.1 & -0.2 \\ 0.047 & -0.05 & 0.14 \\ -0.053 & -0.07 & 0.16 \end{pmatrix}. \tag{2.1.26}$$

By (2.1.25) and (2.1.26) we find

$$ATS^{-1}XTA^T = 0,$$

which means that the matrix $ATS^{-1}XTA^T$ is singular.

We notice that Example 2.1.2 shows Newton search directions for a general SOCO problem may not be well-defined. Even the Newton directions are well-defined, Example 2.1.1 suggests that Newton systems may not be solved efficiently by Cholesky factorization methods, since Cholesky factorization methods only work for symmetric and positive definite matrices. Nevertheless, these problems can be solved by scaling the original SOCO problems somehow. In fact, by choosing proper scaling, the scaled SOCO problems are well-defined and the resulting systems are symmetric and positive definite. The scaled SOCO problems are the subject we discuss in the next section.

2.2 Newton Search Directions with Scaling

As noticed in the last section, the Newton search directions for the general SOCO problems may not be well-defined or may not be solved efficiently. In order to deal with these problems, the scaling techniques are used. More precisely, we first transform the original unknown variables to the corresponding new variables by using the scaling matrices that are given in Definition 1.1.9. Then, we derive the Newton search directions for the scaled SOCO problems. Finally, we scale the new variables back to the original space. The scaling techniques can be found, e.g., [14, 16, 28].

Assume that G and Θ are matrices given in (1.2.5) and (1.2.6), respectively. Then the new variables $(\tilde{x}, \tilde{\tau}, \tilde{y}, \tilde{s}, \tilde{\kappa}, \tilde{\nu})$ are defined by

$$\tilde{x} = \Theta G x, \quad \tilde{s} = (\Theta G)^{-1}s, \quad \tilde{\nu} = \nu, \quad \tilde{\tau} = \tau, \quad \tilde{\kappa} = \kappa, \quad \tilde{y} = y. \tag{2.2.1}$$

By (2.2.1) we find

$$x = (\Theta G)^{-1}\tilde{x}, \quad s = (\Theta G)\tilde{s}, \quad \nu = \tilde{\nu}, \quad \tau = \tilde{\tau}, \quad \kappa = \tilde{\kappa}, \quad y = \tilde{y}. \tag{2.2.2}$$

Substituting (2.2.2) into (1.3.8) yields

$$\begin{aligned}
&\text{minimize} && \beta\tilde{\nu} \\
&\text{subject to} && A(\Theta G)^{-1}\tilde{x} - b\tilde{\tau} - r_p\tilde{\nu} &&= 0, \\
& && -A^T\tilde{y} + c\tilde{\tau} - (\Theta G)\tilde{s} - r_d\tilde{\nu} &&= 0, \\
& && b^T\tilde{y} - c^T(\Theta G)^{-1}\tilde{x} - \tilde{\kappa} - r_g\tilde{\nu} &&= 0, \\
& && r_p^T\tilde{y} + r_d^T(\Theta G)^{-1}\tilde{x} + r_g\tilde{\tau} &&= -\beta,
\end{aligned} \tag{2.2.3}$$

where, $\tilde{y}$ is free, $\tilde{x},\ \tilde{s} \in K,\ \tilde{\tau},\ \tilde{\kappa} \geq 0$ and $\tilde{\nu}$ is free. Let

$$\tilde{A} = A(\Theta G)^{-1}, \qquad \tilde{c} = (\Theta G)^{-1}c, \qquad \tilde{r}_d = (\Theta G)^{-1}r_d. \tag{2.2.4}$$

Then problem (2.2.3) can be rewritten as

$$\begin{aligned}
&\text{minimize} && \beta\tilde{\nu} \\
&\text{subject to} && \tilde{A}\tilde{x} - b\tilde{\tau} - r_p\tilde{\nu} &&= 0, \\
& && -\tilde{A}^T\tilde{y} + \tilde{c}\tilde{\tau} - \tilde{s} - \tilde{r}_d\tilde{\nu} &&= 0, \\
& && b^T\tilde{y} - \tilde{c}^T\tilde{x} - \tilde{\kappa} - r_g\tilde{\nu} &&= 0, \\
& && r_p^T\tilde{y} + \tilde{r}_d^T\tilde{x} + r_g\tilde{\tau} &&= -\beta,
\end{aligned} \tag{2.2.5}$$

where, $\tilde{y}$ is free, $\tilde{x},\ \tilde{s} \in K,\ \tilde{\tau}, \tilde{\kappa} \geq 0$ and $\tilde{\nu}$ is free.

We notice that problem (2.2.5) has the same form as problem (1.3.8). Then, repeating the procedure for deriving the central path for problem (1.3.8) in Section 3 of Chapter 1, we find that the central path for problem (2.2.5) is determined by the solutions of the following system with parameter $\tilde{\mu} > 0$:

$$\begin{aligned}
\tilde{A}\tilde{x} - b\tilde{\tau} - r_p\tilde{\nu} &= 0, \\
-\tilde{A}^T\tilde{y} + \tilde{c}\tilde{\tau} - \tilde{s} - \tilde{r}_d\tilde{\nu} &= 0, \\
b^T\tilde{y} - \tilde{c}^T\tilde{x} - \tilde{\kappa} - r_g\tilde{\nu} &= 0, \\
r_p^T\tilde{y} + \tilde{r}_d^T\tilde{x} + r_g\tilde{\tau} &= -\beta, \\
\tilde{X}\tilde{S}e &= \tilde{\mu}e, \\
\tilde{\kappa}\tilde{\tau} &= \tilde{\mu}.
\end{aligned} \tag{2.2.6}$$

Note that system (2.2.6) is the analogue of system (1.3.16) for problem (1.3.8). In the last section, we proved that system (1.3.16) is equivalent to system (2.1.9). By a similar argument, we can also prove that system (2.2.6) is equivalent to the following:

$$\begin{aligned} \tilde{A}\tilde{x} - b\tilde{\tau} - r_p\tilde{\nu} &= 0, \\ -\tilde{A}^T\tilde{y} + \tilde{c}\tilde{\tau} - \tilde{s} - \tilde{r}_d\tilde{\nu} &= 0, \\ b^T\tilde{y} - \tilde{c}^T\tilde{x} - \tilde{\kappa} - r_g\tilde{\nu} &= 0, \\ \tilde{X}\tilde{S}e &= \mu^{(0)}\tilde{\nu}e, \\ \tilde{\kappa}\tilde{\tau} &= \mu^{(0)}\tilde{\nu}, \end{aligned} \tag{2.2.7}$$

where $\mu^{(0)}$ is given by (2.1.10). Then the central path for the scaled optimization problem (2.2.5) is determined by the solutions of system (2.2.7) parameterized by the parameter $\tilde{\nu} > 0$. Next, we derive the Newton search directions for system (2.2.7).

Assume that the solution for the current iteration is $(\tilde{x}, \tilde{\tau}, \tilde{y}, \tilde{s}, \tilde{\kappa})$. Then the solution for the next iteration can be written as $(\tilde{x} + \Delta\tilde{x}, \tilde{\tau} + \Delta\tilde{\tau}, \tilde{y} + \Delta\tilde{y}, \tilde{s} + \Delta\tilde{s}, \tilde{\kappa} + \Delta\tilde{\kappa})$, where $(\Delta\tilde{x}, \Delta\tilde{\tau}, \Delta\tilde{y}, \Delta\tilde{s}, \Delta\tilde{\kappa})$ is the unknown quantity that we want to solve. Substitute $(\tilde{x} + \Delta\tilde{x}, \tilde{\tau} + \Delta\tilde{\tau}, \tilde{y} + \Delta\tilde{y}, \tilde{s} + \Delta\tilde{s}, \tilde{\kappa} + \Delta\tilde{\kappa})$ into system (2.2.7). Then we find the following system of nonlinear equations for $(\Delta\tilde{x}, \Delta\tilde{\tau}, \Delta\tilde{y}, \Delta\tilde{s}, \Delta\tilde{\kappa})$:

$$\begin{aligned} \tilde{A}\Delta\tilde{x} - b\Delta\tilde{\tau} &= r_p\tilde{\nu} - (\tilde{A}\tilde{x} - b\tilde{\tau}), \\ -\tilde{A}^T\Delta\tilde{y} + \tilde{c}\Delta\tilde{\tau} - \Delta\tilde{s} &= \tilde{r}_d\tilde{\nu} - (-\tilde{A}^T\tilde{y} + \tilde{c}\tilde{\tau} - \tilde{s}), \\ b^T\Delta\tilde{y} - \tilde{c}^T\Delta\tilde{x} - \Delta\tilde{\kappa} &= r_g\tilde{\nu} - (b^T\tilde{y} - \tilde{c}^T\tilde{x} - \tilde{\kappa}), \\ \tilde{X}(\Delta\tilde{S})e + \tilde{S}(\Delta\tilde{X})e &= \tilde{\nu}\mu^{(0)}e - \tilde{X}\tilde{S}e - (\Delta\tilde{X})(\Delta\tilde{S})e, \\ \tilde{\kappa}\Delta\tilde{\tau} + \tilde{\tau}\Delta\tilde{\kappa} &= \tilde{\nu}\mu^{(0)} - \tilde{\kappa}\tilde{\tau} - \Delta\tilde{\kappa}\Delta\tilde{\tau}, \end{aligned} \tag{2.2.8}$$

where $\tilde{X} = \mathrm{mat}(T\tilde{x})$, $\tilde{S} = \mathrm{mat}(T\tilde{s})$, $\Delta\tilde{X} = \mathrm{mat}(T\Delta\tilde{x})$ and $\Delta\tilde{S} = \mathrm{mat}(T\Delta\tilde{s})$. Substi-

tuting (2.2.1) into (2.2.8) yields

$$\begin{aligned} A\Delta x - b\Delta\tau &= r_p\nu - (Ax - b\tau), \\ -A^T\Delta y + c\Delta\tau - \Delta s &= r_d\nu - (-A^T y + c\tau - s), \\ b^T\Delta y - c^T\Delta x - \Delta\kappa &= r_g\nu - (b^T y - c^T x - \kappa), \\ \tilde{X}T(\Theta G)^{-1}\Delta s + \tilde{S}T(\Theta G)\Delta x &= \nu\mu^{(0)}e - \tilde{X}\tilde{S}e - (\Delta\tilde{X})(\Delta\tilde{S})e, \\ \kappa\Delta\tau + \tau\Delta\kappa &= \nu\mu^{(0)} - \kappa\tau - \Delta\kappa\Delta\tau, \end{aligned} \tag{2.2.9}$$

where $\tilde{X} = \mathrm{mat}(T\tilde{x}) = \mathrm{mat}(T\Theta Gx)$, $\tilde{S} = \mathrm{mat}(T\tilde{s}) = \mathrm{mat}(T(\Theta G)^{-1}s)$, $\Delta\tilde{X} = \mathrm{mat}(T\Delta\tilde{x}) = \mathrm{mat}(T\Theta G\Delta x)$, $\Delta\tilde{S} = \mathrm{mat}(T\Delta\tilde{s}) = \mathrm{mat}(T(\Theta G)^{-1}\Delta s)$.

When we solve system (2.2.9), we actually ignore the second order terms $(\Delta\tilde{X})(\Delta\tilde{S})e$ and $\Delta\kappa\Delta\tau$, or approximate these two terms by known quantities. By doing so, we obtain the following system of linear equations that determines the Newton search directions:

$$\begin{aligned} A\Delta x - b\Delta\tau &= r_p\nu - (Ax - b\tau), \\ -A^T\Delta y + c\Delta\tau - \Delta s &= r_d\nu - (-A^T y + c\tau - s), \\ b^T\Delta y - c^T\Delta x - \Delta\kappa &= r_g\nu - (b^T y - c^T x - \kappa), \\ \tilde{X}T(\Theta G)^{-1}\Delta s + \tilde{S}T(\Theta G)\Delta x &= \nu\mu^{(0)}e - \tilde{X}\tilde{S}e - E_{xs}, \\ \kappa\Delta\tau + \tau\Delta\kappa &= \nu\mu^{(0)} - \kappa\tau - E_{\kappa\tau}, \end{aligned} \tag{2.2.10}$$

where E_{xs} and $E_{\kappa\tau}$ are approximations to the second order terms $(\Delta\tilde{X})(\Delta\tilde{S})e$ and $\Delta\kappa\Delta\tau$, respectively. Later, we will discuss how to calculate E_{xs} and $E_{\kappa\tau}$ based on the current iteration point (x, τ, y, s, κ). Next, we derive the explicit formulas for solutions of (2.2.10) when the matrix A has full row rank.

For convenience, we denote by r_1, r_2, r_3, r_4, r_5 the corresponding right-hand sides

of equations in (2.2.10), that is,

$$\begin{aligned} r_1 &= r_p\nu - (Ax - b\tau), \\ r_2 &= r_d\nu + (A^Ty - c\tau + s), \\ r_3 &= r_g\nu - (b^Ty - c^Tx - \kappa), \\ r_4 &= \nu\mu^{(0)}e - \tilde{X}\tilde{S}e - E_{xs}, \\ r_5 &= \nu\mu^{(0)} - \kappa\tau - E_{\kappa\tau}. \end{aligned} \tag{2.2.11}$$

Substituting (2.2.11) into (2.2.10) yields

$$\begin{aligned} A\Delta x - b\Delta\tau &= r_1, \\ -A^T\Delta y + c\Delta\tau - \Delta s &= r_2, \\ b^T\Delta y - c^T\Delta x - \Delta\kappa &= r_3, \\ \tilde{X}T(\Theta G)^{-1}\Delta s + \tilde{S}T(\Theta G)\Delta x &= r_4, \\ \kappa\Delta\tau + \tau\Delta\kappa &= r_5. \end{aligned} \tag{2.2.12}$$

By the last equation in (2.2.12), we find

$$\Delta\kappa = \frac{r_5 - \kappa\Delta\tau}{\tau}. \tag{2.2.13}$$

By the fourth equation in (2.2.12), we have

$$\Delta s = \Theta GT(\tilde{X})^{-1}r_4 - (\Theta G)^2\Delta x. \tag{2.2.14}$$

Substituting (2.2.13) and (2.2.14) into (2.2.12), we get

$$\begin{aligned} A\Delta x - b\Delta\tau &= r_1, \\ -A^T\Delta y + c\Delta\tau + (\Theta G)^2\Delta x &= r_2 + \Theta GT(\tilde{X})^{-1}r_4, \\ b^T\Delta y - c^T\Delta x + \tfrac{\kappa}{\tau}\Delta\tau &= r_3 + \tfrac{r_5}{\tau}. \end{aligned} \tag{2.2.15}$$

Let $D = (\Theta G)^{-1}$. Then system (2.2.15) can be rewritten as

$$\begin{pmatrix} A & 0 & -b \\ D^{-2} & -A^T & c \\ -c^T & b^T & \frac{\kappa}{\tau} \end{pmatrix} \begin{pmatrix} \Delta x \\ \Delta y \\ \Delta\tau \end{pmatrix} = \begin{pmatrix} r_1 \\ r_2' \\ r_3' \end{pmatrix}, \tag{2.2.16}$$

where

$$\begin{aligned} r_2' &= r_2 + \Theta G T(\tilde{X})^{-1} r_4, \\ r_3' &= r_3 + \tfrac{r_5}{\tau}. \end{aligned} \tag{2.2.17}$$

It follows from the second equation in (2.2.16) that

$$\Delta x = D^2 r_2' + D^2 A^T \Delta y - D^2 c \Delta\tau. \tag{2.2.18}$$

Substituting (2.2.18) into (2.2.16), we find

$$\begin{pmatrix} AD^2A^T & -AD^2c - b \\ b^T - c^T D^2 A^T & \frac{\kappa}{\tau} + c^T D^2 c \end{pmatrix} \begin{pmatrix} \Delta y \\ \Delta\tau \end{pmatrix} = \begin{pmatrix} r_1' \\ r_3'' \end{pmatrix}, \tag{2.2.19}$$

where r_1' and r_3'' are given by

$$\begin{aligned} r_1' &= r_1 - AD^2 r_2', \\ r_3'' &= r_3' + c^T D^2 r_2'. \end{aligned} \tag{2.2.20}$$

Denote by

$$a_1 = -AD^2c - b, \quad a_2 = -b + AD^2c, \quad a_3 = \frac{\kappa}{\tau} + c^T D^2 c. \tag{2.2.21}$$

Then system (2.2.19) can be rewritten as

$$\begin{pmatrix} AD^2A^T & a_1 \\ -a_2^T & a_3 \end{pmatrix} \begin{pmatrix} \Delta y \\ \Delta\tau \end{pmatrix} = \begin{pmatrix} r_1' \\ r_3'' \end{pmatrix}. \tag{2.2.22}$$

By the second equation in (2.2.22), we find

$$\Delta\tau = \frac{r_3'' + a_2^T \Delta y}{a_3}. \tag{2.2.23}$$

Substituting (2.2.23) into (2.2.22), we obtain

$$\left(AD^2A^T + \bar{a}\hat{a}^T\right) \Delta y = \xi, \tag{2.2.24}$$

where $\bar{a}$, $\hat{a}$ and ξ are given by

$$\bar{a} = \frac{a_1}{a_3}, \qquad \hat{a} = a_2, \qquad \xi = r_1' - \frac{r_3''}{a_3} a_1. \tag{2.2.25}$$

From the above procedure, we see that, in order to find the search directions, we first need to solve Δy from (2.2.24), then $\Delta\tau$ from (2.2.23), Δx from (2.2.18), Δs from (2.2.14), finally, $\Delta\kappa$ from (2.2.13). In other words, solving Newton system (2.2.10) consists of the following five steps.

Step 1. Solve (2.2.24) for Δy;
Step 2. Calculate (2.2.23) for $\Delta\tau$;
Step 3. Calculate (2.2.18) for Δx;
Step 4. Calculate (2.2.14) for Δs;
Step 5. Calculate (2.2.13) for $\Delta\kappa$.

We notice that, among the above five steps, Step 1 for solving Δy is the most expensive part for solving the Newton search directions. Once Δy is found, then $\Delta\tau$, Δx, Δs and $\Delta\kappa$ can be computed cheaply by just matrix-vector multiplications. In the next section, we are devoted to solving the linear system (2.2.24) for Δy.

2.3 Solving Newton Systems

In this section, we solve linear system (2.2.24) that is the most time-consuming part to solve the Newton search directions. In order to solve (2.2.24) efficiently, the sparse structure of the linear system is explored and a sparse matrix package is used. Next, we first introduce Sherman-Morrison formula (see, e.g., [9, 13, 32]) that is useful to solve system (2.2.24). Then, we discuss how to construct a sparse linear system if (2.2.24) is dense.

2.3.1 Sherman-Morrison Formula

The following is the so-called Sherman-morrison formula (see, e.g., [9, 13, 32]) that is used to calculate the inverse of a matrix with a special structure:

$$(P + RS^T)^{-1} = P^{-1} - P^{-1}R(I + S^TP^{-1}R)^{-1}S^TP^{-1}, \tag{2.3.1}$$

where P is an $n \times n$ nonsingular matrix, R and S are both $n \times k$ matrices with the same low rank, i.e., $\text{rank}(R) = \text{rank}(S) = k << n$. By (1.2.5) and (1.2.6), we see that $D = (\Theta G)^{-1}$ is a symmetric and positive definite matrix. This implies that AD^2A^T is also a symmetric and positive definite matrix since we assume A has full row rank. Set $P = AD^2A^T$, $R = \bar{a}$ and $S = \hat{a}$, where $\bar{a}$ and $\hat{a}$ are given by (2.2.25). Then, by Sherman-Morrison formula (2.3.1), the solution for system (2.2.24) can be expressed as

$$\Delta y = (AD^2A^T)^{-1}\xi - \frac{(\hat{a}^T(AD^2A^T)^{-1}\xi)(AD^2A^T)^{-1}\bar{a}}{1 + \hat{a}^T(AD^2A^T)^{-1}\bar{a}}. \tag{2.3.2}$$

Since computations of inverse matrices are costly, we here solve $(AD^2A^T)^{-1}\xi$ and $(AD^2A^T)^{-1}\bar{a}$ by solving the related linear systems instead of calculating them directly. To this end, we denote by

$$v_0 = (AD^2A^T)^{-1}\xi, \qquad v_1 = (AD^2A^T)^{-1}\bar{a}. \tag{2.3.3}$$

Then we find v_0 and v_1 satisfy the following linear systems, respectively:

$$(AD^2A^T)v_0 = \xi, \tag{2.3.4}$$

and

$$(AD^2A^T)v_1 = \bar{a}. \tag{2.3.5}$$

We notice that AD^2A^T is a symmetric and positive definite matrix, and thus Cholesky factorization can be used to solve systems (2.3.3) and (2.3.4) efficiently. Since the two systems (2.3.3) and (2.3.4) have the same coefficient matrix, therefore, only one Cholesky factorization is needed. This can reduce the computational time significantly because Cholesky factorization is the most expensive part for solving a linear system. Once we get v_0 and v_1 from (2.3.4) and (2.3.5), then it follows from (2.3.2) that

$$\Delta y = v_0 - \frac{\hat{a}^T v_0}{1 + \hat{a}^T v_1} v_1. \tag{2.3.6}$$

In practice, the matrix AD^2A^T often has sparse structure, that is, most elements of AD^2A^T are zeros. In this case, the sparse structure of the matrix can be exploited and a sparse matrix package can be used to reduce the computational cost for Cholesky factorization. In this thesis, we adopt the Watson Sparse Matrix Package [8] (WSMP for short) that is a robust and efficient software package for solving large sparse systems of linear equations on IBM RS6000 workstations. In some circumstances, even the matrix AD^2A^T is dense, we can still apply the WSMP to reduce the cost for computing the Cholesky factorization. Let's explain more about this case. Assume that K_i is a linear cone $\mathbb{R}^+$ for each $i = 1, \cdots, k$. Thus, by definition, the matrix D is a diagonal matrix. Let $D = \text{diag}(d_1, \cdots, d_k)$, and denote the ith column of A by a_i. Then it follows

$$AD^2A^T = \sum_{i=1}^{k} d_i^2 a_i a_i^T. \tag{2.3.7}$$

From (2.3.7), we see that the matrix AD^2A^T might be dense even if the matrix A has only one dense column. In order to recover the sparsity of the matrix AD^2A^T in this case, we can separate the few dense columns from the matrix A. In the next section, we discuss how to perform this separation.

2.3.2 Separating Dense Columns of A

As noticed in the preceding section, a few dense columns of A can make system (2.2.24) dense. In this case, the dense columns of A must be separated in order to solve Newton systems efficiently. Next, we describe the procedure to separate such dense columns of A (see, e.g., [13, 32]).

Without loss of generality, we assume that all sparse columns of A are located before its dense columns. In other words, we assume that A has the following structure:

$$A = (A_s, A_d), \tag{2.3.8}$$

where A_s consists of the sparse columns of A, whereas, A_d consists of the dense columns of A. Corresponding to the partition of A in (2.3.8), we also partition the matrix D^2 as follows:

$$D^2 = \begin{pmatrix} D_s^2 & 0 \\ \\ 0 & D_d^2 \end{pmatrix}. \tag{2.3.9}$$

Then, it follows from (2.3.8) and (2.3.9) that

$$AD^2A^T = (A_s, A_d) \begin{pmatrix} D_s^2 & 0 \\ \\ 0 & D_d^2 \end{pmatrix} \begin{pmatrix} A_s^T \\ \\ A_d^T \end{pmatrix} = A_s D_s^2 A_s^T + A_d D_d^2 A_d^T. \tag{2.3.10}$$

Substituting (2.3.10) into (2.2.24), we find Δy satisfies

$$\left(A_s D_s^2 A_s^T + A_d D_d^2 A_d^T + \bar{a}\hat{a}^T\right) \Delta y = \xi. \tag{2.3.11}$$

Let $P = A_s D_s^2 A_s^T$, $\quad R = [A_d D_d \quad \bar{a}]$ and $S = [A_d D_d \quad \hat{a}]$. Then (2.3.11) can be rewritten as

$$(P + RS^T)\Delta y = \xi. \tag{2.3.12}$$

If A_s has full row rank, then P is nonsingular. In this case, it follows from (2.3.1) that

$$\Delta y = P^{-1}\xi - P^{-1}R\left(I + S^T P^{-1} R\right)^{-1} S^T P^{-1}\xi. \tag{2.3.13}$$

Similar to the procedure to compute Δy from (2.3.2), calculations of the right-hand side of (2.3.13) consists of three steps: first computing $P^{-1}\xi$, then $P^{-1}R$, finally $(I + S^T P^{-1} R)^{-1} S^T P^{-1}\xi$. Let's denote by

$$v_0 = P^{-1}\xi, \qquad v_1 = P^{-1}R, \qquad v_2 = (I + S^T P^{-1} R)^{-1} S^T P^{-1}\xi. \tag{2.3.14}$$

Then, by (2.3.14), we see that v_0, v_1 and v_2 are determined by the solutions of the following linear systems, respectively:

$$Pv_0 = \xi, \tag{2.3.15}$$

$$Pv_1 = R, \tag{2.3.16}$$

and

$$(I + S^T P^{-1} R)v_2 = S^T v_0. \tag{2.3.17}$$

Since matrix P is sparse, therefore the sparse matrix solver WSMP can be used to solve systems (2.3.15) and (2.3.16) efficiently. We notice again that systems (2.3.15) and (2.3.16) have the same coefficient matrix P, and thus only one Cholesky factorization is needed to solve both v_0 and v_1. In other words, one Cholesky factorization and multiple forward-backward substitutions are sufficient for solving both (2.3.15) and (2.3.16). This means that the computational cost for solving (2.3.15) and (2.3.16) is essentially one Cholesky factorization because forward-backward substitutions are just cheap operations. Once v_0 is found from (2.3.15), then v_2 can be solved from (2.3.17). If R and S have low rank, then (2.3.17) is a small linear system. This means that solving (2.3.17) does not cost much although it is a dense system. After we get v_0, v_1 and v_2, then, by (2.3.13), Δy is given by

$$\Delta y = v_0 - v_1 v_2. \tag{2.3.18}$$

We notice that the above method to calculate Δy from (2.3.13) only works when the matrix P is nonsingular. If A_s does not have full row rank, then $P = A_s D_s^2 A_s^T$ is singular. In this case, formula (2.3.13) does not apply. Next, we discuss how to modify the matrix P such that Δy can still be solved efficiently even A_s does not have full row rank.

Assume that the rank of matrix A_s is m_1 with $m_1 < m$. Since any matrix can be transformed to a upper triangular matrix by the Gussian elimination method, therefore, without loss of generality, we assume that A_s is a upper triangular matrix with the last $m - m_1$ rows being zeros. we now construct a matrix H as follows:

$$H = (e_{m_1+1}, e_{m_1+2}, \cdots, e_m), \tag{2.3.19}$$

where e_i $(i = m_1 + 1, \cdots, m)$ is the unit vector with the ith element being 1 and all other elements being 0. With such a matrix H given in (2.3.19), we claim that the matrix $A_s D^2 A_s^T + HH^T$ is nonsingular (see, e.g., [2, 32]). Let

$$\tilde{P} = A_s D^2 A_s^T + HH^T, \qquad \tilde{R} = [-H \quad R], \qquad \tilde{S} = [H \quad S]. \tag{2.3.20}$$

Then system (2.3.12) can be rewritten as

$$(\tilde{P} + \tilde{R}\tilde{S}^T)\Delta y = \xi. \tag{2.3.21}$$

Since matrix $\tilde{P}$ is symmetric and positive definite, the procedure discussed above still can be applied to solve system (2.3.21) efficiently. However, in this case, we have to solve a larger dense system than (2.3.17) because $\tilde{R}$ and $\tilde{S}$ are bigger than R and S, respectively.

In this section, we have discussed how to construct a sparse linear system by splitting the dense columns of A. It is clear that, not only the dense columns of A, but also dense columns of D can make system (2.2.24) dense. In the next section, we describe the method for separating such dense columns of D.

2.3.3 Separating Dense Columns of D

As we know, the sparse structure of matrix D depends on the structure of K, which contains the linear and quadratic cones involved in a SOCO problem. If K does not contain quadratic cones or the quadratic cones are all small in size, then D is a sparse diagonal or block-diagonal matrix. But, if K contains at least one large quadratic cone, then D has at least one dense column that can make (2.2.24) dense. In this case we need to split the dense column of D to build a sparse linear system. This is what we plan to do next.

Let $K = K_1 \times K_2 \times \cdots \times K_k$ and $K_i \subseteq I\!R^{n_i}$ $(i = 1, \cdots, k)$. Assume that K has l linear cones and a small number of large quadratic cones. If the variables in the quadratic cones have no upper bounds, then the matrix AD^2A^T can be written as the following:

$$\begin{aligned} AD^2A^T & \\ &= (A_1, \cdots, A_l, A_{l+1}, \cdots, A_k) \begin{pmatrix} D_1^2 & & & & & & \\ & \ddots & & & & & \\ & & D_l^2 & & & & \\ & & & D_{l+1}^2 & & & \\ & & & & & \ddots & \\ & & & & & & D_k^2 \end{pmatrix} \begin{pmatrix} A_1^T \\ \vdots \\ A_l^T \\ A_{l+1}^T \\ \vdots \\ A_k^T \end{pmatrix} \\ &= \sum_{i=1}^{l} A_i D_i^2 A_i^T + \sum_{i=l+1}^{k} A_i D_i^2 A_i^T \\ &= \sum_{i=1}^{l} A_i D_i^2 A_i^T + \sum_{i=l+1}^{k} A_i \theta_i^{-2} G_i^{-2} A_i^T \end{aligned}$$

$$
\begin{aligned}
&= \sum_{i=1}^{l} A_i D_i^2 A_i^T + \sum_{i=l+1}^{k} A_i \theta_i^{-2} \left(-Q_i + 2(Q_i g_i)(Q_i g_i)^T\right) A_i^T \\
&= \sum_{i=1}^{l} A_i D_i^2 A_i^T - \sum_{i=l+1}^{k} \theta_i^{-2} A_i Q_i A_i^T + 2 \sum_{i=l+1}^{k} \theta_i^{-2} (A_i Q_i g_i)(A_i Q_i g_i)^T
\end{aligned}
$$

Let $Q_i^{(q)}$ be the matrix associated with the quadratic cone $\mathbb{R}^{n_i}$. Then from the above, we have

$$
\begin{aligned}
&AD^2A^T \\
&\quad = \sum_{i=1}^{l} (A_i T_i) D_i^2 (A_i T_i)^T + \sum_{i=l+1}^{k} \left((A_i T_i)(-\theta_i^{-2} Q_i^{(q)})(A_i T_i)^T\right) \\
&\qquad +2 \sum_{i=l+1}^{k} \theta_i^{-2} (A_i Q_i g_i)(A_i Q_i g_i)^T \\
&\quad = (AT) \begin{pmatrix} D_1^2 & & & & & & \\ & \ddots & & & & & \\ & & D_l^2 & & & & \\ & & & \theta_{l+1}^{-2} I & & & \\ & & & & \ddots & & \\ & & & & & \theta_k^{-2} I \end{pmatrix} (AT)^T \\
&\qquad -2\theta_{l+1}^{-2} (AT)_{l+1} \left((AT)_{l+1}\right)^T - \cdots - 2\theta_k^{-2} (AT)_{n_1+\cdots+n_{k-1}+1} \left((AT)_{n_1+\cdots+n_{k-1}+1}\right)^T \\
&\qquad +2 \sum_{i=l+1}^{k} \theta_i^{-2} (A_i Q_i g_i)(A_i Q_i g_i)^T \\
&= (AT) \begin{pmatrix} D_1^2 & & & & & & \\ & \ddots & & & & & \\ & & D_l^2 & & & & \\ & & & \theta_{l+1}^{-2} I & & & \\ & & & & \ddots & & \\ & & & & & \theta_k^{-2} I \end{pmatrix} (AT)^T \\
&\quad + \left(\sqrt{2}\theta_{l+1}^{-1} (AT)_{l+1}, \ \cdots, \ \sqrt{2}\theta_k^{-1} (AT)_{n_1+\cdots+n_{k-1}+1}\right) \begin{pmatrix} -\sqrt{2}\theta_{l+1}^{-1} (AT)_{l+1}^T \\ \vdots \\ -\sqrt{2}\theta_k^{-1} (AT)_{n_1+\cdots+n_{k-1}+1}^T \end{pmatrix}
\end{aligned}
$$

$$+\left(A_{l+1}Q_{l+1}(\sqrt{2}\theta_{l+1}^{-1})g_{l+1},\ \cdots,\ A_kQ_k(\sqrt{2}\theta_k^{-1})g_k\right)\begin{pmatrix}(A_{l+1}Q_{l+1}(\sqrt{2}\theta_{l+1}^{-1})g_{l+1})^T\\ \vdots\\ (A_kQ_k(\sqrt{2}\theta_k^{-1})g_k)^T\end{pmatrix}.$$

In order to simplify the above expression, we introduce the following notations:

$$\tilde{D} = T\begin{pmatrix} D_1^2 & & & & & \\ & \ddots & & & & \\ & & D_l^2 & & & \\ & & & \theta_{l+1}^{-2}I & & \\ & & & & \ddots & \\ & & & & & \theta_k^{-2}I \end{pmatrix}T^T, \tag{2.3.22}$$

and $P = A\tilde{D}A^T$. Denote by

$$R_1 = \left(\sqrt{2}\theta_{l+1}^{-1}(AT)_{l+1},\ \cdots,\ \sqrt{2}\theta_k^{-1}(AT)_{n_1+\cdots+n_{k-1}+1}\right), \tag{2.3.23}$$

and

$$R_2 = \left(A_{l+1}Q_{l+1}(\sqrt{2}\theta_{l+1}^{-1})g_{l+1},\ \cdots,\ A_kQ_k(\sqrt{2}\theta_k^{-1})g_k\right). \tag{2.3.24}$$

Then it follows that

$$AD^2A^T = P + (R_1, R_2)(-R_1, R_2)^T. \tag{2.3.25}$$

substituting (2.3.25) into (2.2.24), we get the following system:

$$\left(P + (R_1, R_2)(-R_1, R_2)^T + \bar{a}\hat{a}^T\right)\Delta y = \xi. \tag{2.3.26}$$

Let $R = (R_1, R_2, \bar{a})$ and $S = (-R_1, R_2, \hat{a})$. Then system (2.3.26) can be rewritten as:

$$(P + RS^T)\Delta y = \xi.$$

We notice that this system has already been discussed in the previous sections and can be solved by Sherman-Morrison formula efficiently.

So far, we have discussed how to solve the Newton search directions from (2.2.10). Once the Newton search direction $(\Delta x, \Delta\tau, \Delta y, \Delta s, \Delta\kappa)$ is found, then, by the classical Newton methods, the next iteration point is given by $(x + \Delta x, \tau + \Delta\tau, y + \Delta y, s + \Delta s, \kappa + \Delta\kappa)$. But, in our case for solving the SOCO problems, this new iteration point must belong to the interior of the cone K because, otherwise, the next

Newton search directions are not well-defined and the interior point methods have to terminate before the optimal solutions are found. This leads to the following question: how to find the desired iteration point if $(x + \Delta x, \tau + \Delta\tau, y + \Delta y, s + \Delta s, \kappa + \Delta\kappa)$ does not belong to the interior of K. This subject is addressed in the next section.

2.4 Newton Steps

In this section, we discuss how to determine the next iteration point such that it belongs to the interior of K. We also require that the next iteration point be close to the central path in order to reduce the iteration number. To this end, we define the next iteration point as $(x, \tau, y, s, \kappa) + \alpha(\Delta x, \Delta\tau, \Delta y, \Delta s, \Delta\kappa)$ rather than (x, τ, y, s, κ) $+ (\Delta x, \Delta\tau, \Delta y, \Delta s, \Delta\kappa)$ to ensure that the next iteration point is an interior point of K and not far away from the central path. Here, $\alpha > 0$ is called the Newton step we need to determine. In other words, in order to determine the next iteration point in $\text{int}(K)$, we first need to calculate the Newton steps. The calculation of Newton steps for linear optimization is standard (see, e.g., [13, 32]). Next, we describe the procedure to compute the Newton step for an SOCO problem.

Step 1 Find maximal $\alpha_x > 0$ such that $x + \tilde{\alpha}_x \Delta x \in \text{int}(K)$ for all $0 < \tilde{\alpha}_x < \alpha_x$.

Step 2 Find maximal $\alpha_s > 0$ such that $s + \tilde{\alpha}_s \Delta s \in \text{int}(K)$ for all $0 < \tilde{\alpha}_s < \alpha_s$.

Step 3 Define $\alpha_\tau > 0$ as

$$\alpha_\tau = \begin{cases} +\infty, & \text{if } \Delta\tau \geq 0; \\ -\frac{\tau}{\Delta\tau}, & \text{if } \Delta\tau < 0. \end{cases}$$

Step 4 Define $\alpha_\kappa > 0$ as

$$\alpha_\kappa = \begin{cases} +\infty, & \text{if } \Delta\kappa \geq 0; \\ -\frac{\kappa}{\Delta\kappa}, & \text{if } \Delta\kappa < 0. \end{cases}$$

Step 5 Determine the maximum Newton step $\tilde{\alpha}$ as follows:

$$\tilde{\alpha} = \min\left\{\gamma\alpha_x, \gamma\alpha_s, \gamma\alpha_\tau, \gamma\alpha_\kappa, 1\right\}, \tag{2.4.1}$$

where γ is a fixed positive constant that is used to ensure the next iteration point is indeed in the interior of K, not on the boundary of K. Usually, we take γ as a constant close to 1, for instance, 0.995. After the maximum Newton step is found, we shrink $\tilde{\alpha}$, if necessary, to make sure the next iteration point stay in a specified neighborhood of the central path. The neighborhood of the central path is given in the next chapter. Next, we discuss how to compute α_x and α_s in order to find the maximum Newton step.

2.4.1 Calculation of Maximum Newton Steps

In this section, we discuss the details to compute the maximum Newton steps. The reader may skip this section if not interested in such details. For simplicity, we assume that K consists of a single cone in the sequel. But , the results can be generalized to the case of multiple cones in an obvious way. Let $x = (x_1, \cdots, x_n)^T \in \text{int}(K)$ and $\Delta x = (\Delta x_1, \cdots, \Delta x_n)^T \in I\!R^n$. Then the calculations of α_x are divided into three cases that depend on the types of K.

Case 1. K is a linear cone $I\!R_+$. In this case, α_x is given by

$$\alpha_x = \begin{cases} +\infty, & \text{if } \Delta x_i \geq 0 \text{ for all } i = 1, \cdots, n; \\ \min\left\{-\frac{x_i}{\Delta x_i} \ : \ \Delta x_i < 0, \ i = 1, \cdots, n\right\}, & \text{otherwise.} \end{cases}$$

Case 2. K is a quadratic cone K_q. By definition, in this case, we want to find the maximal α_x such that $x + \lambda \Delta x \in \text{int}(K_q)$ for all $0 < \lambda < \alpha_x$. To this end, let's define a quadratic function f as follows:

$$f(\lambda) = (x_1 + \lambda \Delta x_1)^2 - \|(x + \lambda \Delta x)_{2:n}\|^2,$$

that is,

$$f(\lambda) = (x_1 + \lambda \Delta x_1)^2 - (x_2 + \lambda \Delta x_2)^2 - \cdots - (x_n + \lambda \Delta x_n)^2 . \tag{2.4.2}$$

By (2.4.2) we see that the point $x + \lambda \Delta x$ belongs to $\text{int}(K_q)$ if and only if $f(\lambda) > 0$ and $x_1 + \lambda \Delta x_1 > 0$. Simplifying (2.4.2), we find

$$f(\lambda) = \gamma_1 \lambda^2 + \gamma_2 \lambda + \gamma_3, \tag{2.4.3}$$

where

$$\gamma_1 = (\Delta x_1)^2 - (\Delta x_2)^2 - \cdots - (\Delta x_n)^2, \tag{2.4.4}$$

$$\gamma_2 = 2\left(x_1 \Delta x_1 - x_2 \Delta x_2 - \cdots - x_n \Delta x_n\right), \tag{2.4.5}$$

and

$$\gamma_3 = x_1^2 - x_2^2 - \cdots - x_n^2. \tag{2.4.6}$$

Next, we solve the inequality $f(\lambda) > 0$. More precisely, we decide the maximal number $\tilde{\lambda} > 0$ such that

$$f(\lambda) > 0 \quad \text{for all} \ \ 0 < \lambda < \tilde{\lambda}. \tag{2.4.7}$$

The constant $\tilde{\lambda}$ actually depends on the roots of the quadratic equation:

$$f(\lambda) = 0. \tag{2.4.8}$$

We first discuss the case where equation (2.4.8) has no roots, and then consider the case where equation (2.4.8) has at least one root.

Case 2.1. $\gamma_2^2 - 4\gamma_1\gamma_3 < 0$. In this case, the equation (2.4.8) has no roots. Since $\gamma_2^2 - 4\gamma_1\gamma_3 < 0$, it follows that

$$\gamma_1\gamma_3 > 0. \tag{2.4.9}$$

Since $x \in \text{int}(K)$, we have

$$\gamma_3 > 0. \tag{2.4.10}$$

By (2.4.9) and (2.4.10), we find

$$\gamma_1 > 0. \tag{2.4.11}$$

Note that f given in (2.4.3) can be rewritten as

$$f(\lambda) = \gamma_1 \left(\lambda + \frac{\gamma_2}{2\gamma_1}\right)^2 - \frac{\gamma_2^2 - 4\gamma_1\gamma_3}{4\gamma_1}. \tag{2.4.12}$$

Then, by $\gamma_2^2 - 4\gamma_1\gamma_3 < 0$, it follows from (2.4.11) and (2.4.12) that $f(\lambda) > 0$ for any number λ. Therefore, in this case, we define $\tilde{\lambda} = +\infty$, which satisfies (2.4.7).

Case 2.2. $\gamma_2^2 - 4\gamma_1\gamma_3 \geq 0$. In this case, equation (2.4.8) has at least one root. Assume that λ_1 and λ_2 are two roots of (2.4.8). If equation (2.4.8) has only one root, then we set $\lambda_1 = \lambda_2$. We notice that, if $\gamma_1 \neq 0$, then solving (2.4.7) is equivalent to finding $\tilde{\lambda} > 0$ such that

$$f(\lambda) = \gamma_1(\lambda - \lambda_1)(\lambda - \lambda_2) > 0 \quad \text{for all} \;\; 0 < \lambda < \tilde{\lambda}. \tag{2.4.13}$$

Next, we define the constant $\tilde{\lambda}$ according the coefficient γ_1 as follows.

Case 2.2.1. $\gamma_1 > 0$. Since $x \in \text{int}(K)$, we have $\gamma_3 > 0$. If $\gamma_1 > 0$, then we find

$$\lambda_1\lambda_2 = \frac{\gamma_1}{\gamma_3} > 0. \tag{2.4.14}$$

Inequality (2.4.14) implies that either

$$\lambda_1 > 0 \qquad \text{and} \qquad \lambda_2 > 0; \tag{2.4.15}$$

or

$$\lambda_1 < 0 \qquad \text{and} \qquad \lambda_2 < 0. \tag{2.4.16}$$

If (2.4.15) holds, then we define $\tilde{\lambda}$ as:

$$\tilde{\lambda} = \min\{\lambda_1, \; \lambda_2\}, \tag{2.4.17}$$

which satisfies (2.4.13). If (2.4.16) holds, then we see that $f(\lambda) > 0$ for any $\lambda > 0$. Therefore, in this case, $\tilde{\lambda}$ is defined by $\tilde{\lambda} = +\infty$.

Case 2.2.2. $\gamma_1 = 0$. In this case, $\tilde{\lambda}$ is defined as follows:

$$\tilde{\lambda} = \begin{cases} +\infty & \text{if } \gamma_2 \geq 0; \\ -\frac{\gamma_3}{\gamma_2} & \text{if } \gamma_2 < 0. \end{cases} \tag{2.4.18}$$

If $\gamma_1 = 0$, then $f(\lambda) = \gamma_2\lambda + \gamma_3$. Therefore, by (2.4.18), we have $f(\lambda) > 0$ for all $0 < \lambda < \tilde{\lambda}$, as desired.

Case 2.2.3. $\gamma_1 < 0$. In this case, we have $\lambda_1\lambda_2 = \frac{\gamma_1}{\gamma_3} < 0$ since $\gamma_3 > 0$ due to $x \in \text{int}(K)$. Define $\tilde{\lambda}$ as:

$$\tilde{\lambda} = \begin{cases} \lambda_1 & \text{if } \lambda_1 > 0; \\ \lambda_2 & \text{if } \lambda_2 > 0, \end{cases} \tag{2.4.19}$$

which satisfies (2.4.13).

So far, we have discussed how to find the constant $\tilde{\lambda} > 0$ such that (2.4.7) is satisfied. Next, we find a constant $\bar{\lambda} > 0$ such that

$$x_1 + \lambda\Delta x_1 > 0, \qquad \text{for all } 0 < \lambda < \bar{\lambda}. \tag{2.4.20}$$

Actually, the constant $\bar{\lambda}$ is given by

$$\bar{\lambda} = \begin{cases} +\infty & \text{if } \Delta x_1 \geq 0; \\ -\frac{x_1}{\Delta x_1} & \text{if } \Delta x_1 < 0. \end{cases} \tag{2.4.21}$$

Since $x \in \text{int}(K)$, we have $x_1 > 0$. Hence, the constant $\bar{\lambda}$ given in (2.4.21) satisfies (2.4.20). Now, let

$$\alpha_x = \min\{\tilde{\lambda},\ \bar{\lambda}\}. \tag{2.4.22}$$

Then, for any $0 < \lambda < \alpha_x$, we have both $f(\lambda) > 0$ and $x_1 + \lambda\Delta x_1 > 0$. This implies $x + \lambda\Delta x \in \text{int}(K_q)$ for all $0 < \lambda < \alpha_x$, as desired. Next, we discuss the case where K is a rotated quadratic cone.

Case 3. K is a rotated quadratic cone K_r. Again, in this case, we want to find the maximal α_x such that $x + \lambda\Delta x \in \text{int}(K_r)$ for all $0 < \lambda < \alpha_x$. We define a quadratic function f as follows:

$$f(\lambda) = 2(x_1 + \lambda\Delta x_1)(x_2 + \lambda\Delta x_2) - \|(x + \lambda\Delta x)_{3:n}\|^2,$$

that is,

$$f(\lambda) = 2(x_1 + \lambda\Delta x_1)(x_2 + \lambda\Delta x_2) - (x_3 + \lambda\Delta x_3)^2 - \cdots - (x_n + \lambda\Delta x_n)^2. \tag{2.4.23}$$

By (2.4.23) we see that the point $x + \lambda\Delta x$ belongs to $\text{int}(K_r)$ if and only if the following conditions are satisfied:

$$f(\lambda) > 0, \qquad x_1 + \lambda\Delta x_1 > 0, \qquad \text{and} \qquad x_2 + \lambda\Delta x_2 > 0. \tag{2.4.24}$$

Simplifying (2.4.23), we get

$$f(\lambda) = \gamma_1\lambda^2 + \gamma_2\lambda + \gamma_3, \tag{2.4.25}$$

where

$$\gamma_1 = 2\Delta x_1\Delta x_2 - (\Delta x_3)^2 - \cdots - (\Delta x_n)^2, \tag{2.4.26}$$

$$\gamma_2 = 2\left(x_1\Delta x_2 + x_2\Delta x_1 - x_3\Delta x_3 - \cdots - x_n\Delta x_n\right), \tag{2.4.27}$$

and

$$\gamma_3 = 2x_1x_2 - x_3^2 - \cdots - x_n^2. \tag{2.4.28}$$

Now, we want to find the maximal number $\tilde{\lambda} > 0$ such that

$$f(\lambda) > 0 \quad \text{for all} \quad 0 < \lambda < \tilde{\lambda}. \tag{2.4.29}$$

The procedure to determine the constant $\tilde{\lambda}$ for the rotated quadratic cone K_r is almost identical to that for the quadratic cone K_q in the above, and therefore the details are omitted here. Next, we calculate the constants $\bar{\lambda}_1 > 0$ and $\bar{\lambda}_2$ such that

$$x_1 + \lambda\Delta x_1 > 0, \qquad \text{for all } 0 < \lambda < \bar{\lambda}_1, \tag{2.4.30}$$

and

$$x_2 + \lambda\Delta x_2 > 0, \qquad \text{for all } 0 < \lambda < \bar{\lambda}_2. \tag{2.4.31}$$

In this case, the constant $\bar{\lambda}_1$ and $\bar{\lambda}_2$ are actually given by

$$\bar{\lambda}_1 = \begin{cases} +\infty & \text{if } \Delta x_1 \geq 0; \\ -\frac{x_1}{\Delta x_1} & \text{if } \Delta x_1 < 0, \end{cases} \tag{2.4.32}$$

and

$$\bar{\lambda}_2 = \begin{cases} +\infty & \text{if } \Delta x_2 \geq 0; \\ -\frac{x_2}{\Delta x_2} & \text{if } \Delta x_2 < 0. \end{cases} \tag{2.4.33}$$

We note that $x \in \text{int}(K_r)$. This implies that $x_1 > 0$ and $x_2 > 0$. Therefore, the constants $\bar{\lambda}_1$ and $\bar{\lambda}_2$ given in (2.4.32) and (2.4.33) satisfy (2.4.30) and (2.4.31), respectively. Now, let

$$\alpha_x = \min\{\tilde{\lambda},\ \bar{\lambda}_1,\ \bar{\lambda}_2\}, \tag{2.4.34}$$

where $\tilde{\lambda}$, $\bar{\lambda}_1$ and $\bar{\lambda}_2$ are given by (2.4.29),(2.4.32) and (2.4.33), respectively. Then, for any $0 < \lambda < \alpha_x$, we have $f(\lambda) > 0$, $x_1 + \lambda\Delta x_1 > 0$ and $x_2 + \lambda\Delta x_2 > 0$, as desired in (2.4.24). Therefore, $x + \lambda\Delta x \in \text{int}(K_r)$ for all $0 < \lambda < \alpha_x$.

Now, we have described the procedure to find the Newton step α_x such that $x + \tilde{\alpha}_x\Delta x \in \text{int}(K)$ for all $0 < \tilde{\alpha}_x < \alpha_x$. The same procedure also applies to the variable s, that is, by repeating the above procedure, we can also find the Newton step α_s such that $s + \tilde{\alpha}_x\Delta s \in \text{int}(K)$ for all $0 < \tilde{\alpha}_s < \alpha_s$, as stated in Step 2 in the beginning of this section. Once α_x and α_s are found, then, the Newton step α follows from (4.3.1) immediately.

In order to perform the next iteration to slove the SOCO problems, we also need to decide how to update the central path parameter $\nu > 0$, which is the question to be addressed in the next section.

2.5 Predictor-Corrector Strategy

In this section, we discuss how to choose the parameter ν in the Newton system, which determines the central path for the optimization problems. Theoretically, for each fixed $\nu > 0$, we need to solve the Newton system once. Since we must find the optimal solutions for a optimization problem within a finite number of iterations, we can only solve the Newton system for a finite sequence of ν's. Now, the question is: how to choose this finite sequence of ν's such that the optimal solutions can be found as fast as possible. In this thesis, we adopt the widely used Methrotra's predictor-corrector technique [12] to select the sequence of ν's.

Actually, the predictor-corrector method consists of two steps: first solving the predictor direction, and then solving the corrector direction. The predictor direction is given by the the unique solution of system (2.2.10) with $\nu = 0$, $E_{\kappa\tau} = 0$, and $E_{xs} = 0$. Once the predictor direction is found, then the Newton step along this direction can be calculated by the formulas described in the last section. Assume that the predictor direction is $(\Delta x_p, \Delta\tau_p, \Delta y_p, \Delta s_p, \Delta\kappa_p)$, and the Newton step for this direction is α_p. Then the predicted complementarity gap g_p is defined by

$$g_p = (x + \alpha_p\Delta x_p)^T(s + \alpha_p\Delta s_p) + (\kappa + \Delta\kappa_p)(\tau + \Delta\tau_p).$$

The gap g_p is used to solve the corrector direction.

Let g be the current complementarity gap, i,e.,

$$g = x^T s + \kappa\tau. \tag{2.5.1}$$

In order to set up the linear system for the corrector direction, we first need to define the centering parameter ν, the second order approximation terms E_{xs} and $E_{\kappa\tau}$ that

are given by the following:

$$\nu = \left(\frac{g_p}{g}\right)^2 \frac{g_p}{k+1}, \tag{2.5.2}$$

$$E_{xs} = (\Delta\tilde{X}_p)(\Delta\tilde{S}_p)e, \qquad E_{\kappa\tau} = \Delta\kappa_p\Delta\tau_p, \tag{2.5.3}$$

where $\Delta\tilde{X}_p$ and $\Delta\tilde{S}_p$ are given by

$$\Delta\tilde{X}_p = \mathrm{mat}(T\Theta G\Delta x_p), \qquad \Delta\tilde{S}_p = \mathrm{mat}(T(\Theta G)^{-1}\Delta s_p).$$

The the corrector direction is defined as the unique solution of (2.2.10) with ν, E_{xs} and $E_{\kappa\tau}$ given by (2.5.2) and (2.5.3), respectively. Assume that $(\Delta x, \Delta\tau, \Delta y, \Delta s, \Delta\kappa)$ is the corrector direction and α is the Newton step along this direction that is computed by the formulas in the last section. Then the next iteration point is given by $(x, \tau, y, s, \kappa) + \alpha(\Delta x, \Delta\tau, \Delta y, \Delta s, \Delta\kappa)$.

We notice that, by using the predictor-corrector method, we need to solve two linear systems rather than one system at each iteration. Since these two systems have the same coefficient matrix, we only need to facterize the matrix once, and perform two forward-backward substitutions. Since forward-backward substitutions are cheap operations, therefore, the computational cost by predictor-corrector method does not increase very much, compared to solving only one linear system.

In the next section, we summarize what we have done in this chapter, and describe our algorithms to solve the SOCO problems by using the predictor-corrector techniques.

2.6 Algorithms

In this section, we discuss our algorithms to solve the SOCO problems by the Newton search directions (see, e.g., [21]). We first solve the Newton system (2.2.10) for the Newton search directions by using the formulas given in Section 2. At this stage, we take the centering parameter ν as zero, and ignore the second order approximation terms E_{xs} and $E_{\kappa\tau}$. Then, we calculate the Newton steps for the Newton search directions by the formulas given in Section 3. After that, the centering parameter ν and the second order approximation terms E_{xs} and $E_{\kappa\tau}$ are updated by formulas (2.5.2) and (2.5.3), respectively. Next, we solve the Newton system (2.2.10) again for the updated Newton search directions. This time, we use the updated centering parameter and the second order approximation terms. Finally, the Newton steps for the updated Newton search directions are calculated, and the next iteration point follows immediately. Our algorithms can be described as follows.

Algorithms with Newton Search Directions

input:
an accuracy parameter $\epsilon > 0$;
an initial point $(x^{(0)}, \tau^{(0)}, y^{(0)}, s^{(0)}, \kappa^{(0)}) \in \text{int}(K) \times I\!R_+ \times I\!R^m \times \text{int}(K) \times I\!R_+$;
begin
$x = x^{(0)},\ \tau = \tau^{(0)},\ y = y^{(0)},\ s = s^{(0)},\ \kappa = \kappa^{(0)}$;
while $(x^T s + \kappa\tau) > \epsilon$ **do**
begin
solve predictor direction by (2.2.10) with $\nu = 0,\ E_{xs} = 0$ and $E_{\kappa\tau} = 0$;
calculate Newton step α_p by formulas in Section 2.4;
update centering parameter ν by (2.5.2);
solve corrector direction by (2.2.10) with E_{xs} and $E_{\kappa\tau}$ given by (2.5.3);
calculate Newton step α by formulas in Section 2.4;
calculate next iteration point:

$$x = x + \alpha\Delta x, \qquad \tau = \tau + \alpha\Delta\tau, \qquad y = y + \alpha\Delta y,$$
$$s = s + \alpha\Delta s, \qquad \kappa = \kappa + \alpha\Delta\kappa;$$

end
if $\tau > 0$, $(\frac{x}{\tau}, \frac{y}{\tau}, \frac{s}{\tau})$ is a primal-dual optimal solution;
else if $\kappa = 0$, problem is ill-posed;
else if $c^T x < 0$, dual problem is infeasible;
else if $b^T y > 0$, primal problem is infeasible.
end
end

Chapter 3

Interior Point Methods for SOCO Problems with Self-Regular Search Directions

In this chapter, we discuss interior point methods to solve SOCO problems with self-regular search directions. We first review self-regular functions and introduce some typical examples that are used in our implementation. Then, in Section 2, we define the self-regular search directions by using self-regular functions. As we will see later, self-regular search directions are actually generalizations of Newton search directions. Finally, we describe a procedure to solve the self-regular search directions in Section 3. Algorithms based on self-regular search directions and the Mehrotra's predictor-corrector technique are also presented there.

3.1 Self-Regular Functions

In this section, we review self-regular functions. We first introduce the definition of self-regular functions, and then discuss some typical examples that are particularly important both in theory and implementation. Self-regular functions are used to measure the distance from a point to the central path, and were introduced by Peng, Roos and Terlaky (see, e.g., [18, 19, 20, 21]). The definition of self-regular functions is sated as follows.

Definition 3.1.1 *Assume that* $\psi : (0, +\infty) \longrightarrow I\!R_+$ *is a twice differentiable function. We say* ψ *is self-regular if the following two conditions are satisfied:*

(i) $\psi(t)$ is strictly convex with zero global minimum reached at $t = 1$. In addition, there exist constants $\nu_2 \geq \nu_1 > 0$ and $p \geq 1,\ q \geq 1$ such that

$$\nu_1 \left(t^{p-1} + t^{-1-q}\right) \leq \psi''(t) \leq \nu_2 \left(t^{p-1} + t^{-1-q}\right), \quad \forall t \in (0, +\infty); \tag{3.1.1}$$

(ii) for any $t_1,\ t_2 \in (0, +\infty)$ and $r \in [0, 1]$, the following is true:

$$\psi(t_1^r t_2^{1-r}) \leq r\psi(t_1) + (1 - r)\psi(t_2). \tag{3.1.2}$$

We notice that the first condition actually means that the second order derivative of a self-regular function is essentially determined by the function $t^{p-1} + t^{-1-q}$, whereas, the second condition implies that if ψ is self-regular, then the function $\psi(e^{\xi})$ is convex (see, e.g., [18]). Usually, we refer to the parameter q in condition (i) as barrier degree, and p as growth degree. As a special case, if $\nu_1 = \nu_2 = 1$, then it follows from the first condition that

$$\psi''(t) = t^{p-1} + t^{-1-q}, \quad \forall t \in (0, +\infty). \tag{3.1.3}$$

In what follows, we use $\Upsilon_{p,q}$ to denote the function determined by (3.1.3) with $\psi(1) = \psi'(1) = 0$, that is,

$$\Upsilon_{p,q}(t) := \begin{cases} \frac{t^{p+1}-1}{p(p+1)} + \frac{p-1}{p}(t-1) - \log(t), & q = 1; \\ \frac{t^{p+1}-1}{p(p+1)} + \frac{t^{1-q}-1}{q(q-1)} + \frac{p-q}{pq}(t-1), & q > 1. \end{cases} \tag{3.1.4}$$

We can prove that $\Upsilon_{p,q}$ given above is a self-regular function for each $p,\ q \geq 1$ (see, e.g., [18]), and the derivatives of $\Upsilon_{p,q}$ are given by the following:

$$(\Upsilon_{p,q})'(t) = \frac{t^p}{p} - \frac{t^{-q}}{q} + \frac{p-q}{pq}. \tag{3.1.5}$$

If $p = q = 1$, by (3.1.4) we get

$$\Upsilon_{1,1}(t) = \frac{1}{2}t^2 - \log(t) - \frac{1}{2}. \tag{3.1.6}$$

This function is called a logarithmic barrier function and has been widely studied in the literature (see, e.g., [18, 23]).

By (3.1.4), we see that when $t \to +\infty$, the behavior of the self-regular function $\Upsilon_{p,q}(t)$ is dominated by the term $\frac{1}{p(p+1)}t^{p+1}$ for all $q \geq 1$. This means the behavior of $\Upsilon_{p,q}(t)$ with different q's is almost the same as long as t is big enough. However, when $t \to 0$, the behavior of $\Upsilon_{p,1}(t)$ is dominated by the function $-\log(t)$, whereas,

the behavior of $\Upsilon_{p,q}(t)$ with $q > 1$ is governed by the term $\frac{1}{q(q-1)}t^{1-q}$. Since t^{1-q} goes to $+\infty$ much faster than $-\log(t)$ when $t \to 0^+$, we conclude that $\Upsilon_{p,q}(t)$ with $q > 1$ has much stronger barrier property than $\Upsilon_{p,1}(t)$.

Another family of self-regular functions is given by:

$$\Gamma_{p,q}(t) := \frac{t^{p+1}-1}{p+1} + \frac{t^{1-q}-1}{q-1}, \quad p \geq 1, \quad q > 1. \tag{3.1.7}$$

As a special case, if $p = q > 1$, then we see $\Gamma_{p,p} = p\Upsilon_{p,p}(t)$.

We note that, by Definition 3.1.1, the self-regular functions are only defined for linear cones $\mathbb{R}_+$. In order to employ self-regular functions to solve SOCO problems, we need to generalize the concept of self-regularity to the case of quadratic cones. In other words, the following definition (see, e.g., [6, 18]) is necessary.

Definition 3.1.2 *Assume that ψ is a function in $(0, +\infty)$ and K is a linear or quadratic cone. Then the function $\Psi : K \longrightarrow \mathbb{R}^n$ associated with ψ and K is defined by*

$$\Psi(x) = \begin{cases} \left(\frac{\psi(\lambda_1(x))+\psi(\lambda_2(x))}{2}, \quad \frac{\psi(\lambda_1(x))-\psi(\lambda_2(x))}{2\|x_{2:n}\|}x_{2:n}^T\right)^T, & \text{if } x_{2:n} \neq 0; \\ (\psi(\lambda_1(x)), 0, \cdots, 0)^T, & \text{if } x_{2:n} = 0, \end{cases} \tag{3.1.8}$$

where $x = (x_1, \cdots, x_n)^T \in K$, $\lambda_1(x) = x_1 + \|x_{2:n}\|$ and $\lambda_2(x) = x_1 - \|x_{2:n}\|$. If ψ is a self-regular function, then we also say Ψ is self-regular.

By simple computations, we see that, if Ψ is the function associated with ψ and K, then Ψ maps $\text{int}(K)$ into itself. For example, if K is a quadratic cone K_q and $\psi(t) = t^{-1}$, then the function $\Psi(x)$ can be expressed as

$$\Psi(x) = \frac{1}{x_1^2 - \|x_{2:n}\|^2}(x_1, -x_2, \cdots, -x_n)^T, \qquad \forall x \in \text{int}(K_q). \tag{3.1.9}$$

This example is useful when we show that Newton search directions are a special case of self-regular search directions later. For convenience, hereafter, we denote by $\nabla\Psi$ the function associated with ψ' and K that is given by Definition 3.1.2. If K is a product of multiple cones, i.e., $K = K_1 \times K_2 \times \cdots \times K_k$, then we use $\Psi(x)$ to denote the following vector:

$$\Psi(x) = (\Psi(x^{(1)}), \Psi(x^{(2)}), \cdots, \Psi^{(k)}))^T, \quad \forall x = (x^{(1)}, x^{(2)}, \cdots, x^{(k)}) \in K.$$

Self-regular functions can be used to generalize the Newton search directions, and therefore can be applied to solve optimization problems. In the next section, we discuss how to define search directions based on self-regular functions.

3.2 Self-Regular Search Directions

In this section, we define the self-regular search directions for the SOCO problems. Actually, self-regular search directions are generalizations of Newton search directions. Next, we first show that Newton system (2.2.10) is equivalent to a linear system that is determined by a particular self-regular function. Then, we define the general self-regular search directions by using general self-regular functions instead of the particular self-regular function corresponding to Newton search directions. For self-regular search directions, we refer the reader to [21]. Next, we look at the self-regular functions that produce the Newton search directions.

Ignoring the second order approximation terms E_{xs} and $E_{\kappa\tau}$ in (2.2.10), then we find the Newton search directions are determined by:

$$\begin{aligned} A\Delta x - b\Delta\tau &= r_p\nu - (Ax - b\tau), \\ -A^T\Delta y + c\Delta\tau - \Delta s &= r_d\nu - (-A^Ty + c\tau - s), \\ b^T\Delta y - c^T\Delta x - \Delta\kappa &= r_g\nu - (b^Ty - c^Tx - \kappa), \\ \tilde{X}T(\Theta G)^{-1}\Delta s + \tilde{S}T(\Theta G)\Delta x &= \nu\mu^{(0)}e - \tilde{X}\tilde{S}e\ , \\ \kappa\Delta\tau + \tau\Delta\kappa &= \nu\mu^{(0)} - \kappa\tau. \end{aligned} \tag{3.2.1}$$

Next, we deal with the right-hand sides of the last two equations in (3.2.1). To this end, we denote by

$$v_{xs} = \frac{1}{\sqrt{\nu\mu^{(0)}}}\tilde{x}, \tag{3.2.2}$$

and

$$v_{\kappa\tau} = \sqrt{\frac{\kappa\tau}{\nu\mu^{(0)}}}. \tag{3.2.3}$$

Since $\tilde{x} = \tilde{s}$, it follows from (3.2.2) that

$$v_{xs} = \frac{1}{\sqrt{\nu\mu^{(0)}}}\tilde{s}. \tag{3.2.4}$$

Using $\tilde{S} = \mathrm{mat}(T\tilde{s})$ and (3.2.4), we get

$$\tilde{S}e = T\tilde{s} = \sqrt{\nu\mu^{(0)}}(Tv_{xs}). \tag{3.2.5}$$

Since $\tilde{X} = \mathrm{mat}(T\tilde{x})$, it follows from (3.2.2) and Lemma 1.1.1 that

$$\tilde{X}^{-1}e = \frac{1}{(\tilde{x})^TQ(\tilde{x})}\begin{pmatrix} (T\tilde{x})_1 \\ -(T\tilde{x})_{2:n} \end{pmatrix} = \frac{1}{\sqrt{\nu\mu^{(0)}}v_{xs}^TQv_{xs}}\begin{pmatrix} (Tv_{xs})_1 \\ -(Tv_{xs})_{2:n} \end{pmatrix} \tag{3.2.6}$$

Let $\Psi_1^{\Upsilon}(x)$ be the function associated with $\Upsilon_{1,1}(t)$ given in (3.1.6) and K_q. Then, by (3.1.9) we get

$$\nabla\Psi_1^{\Upsilon}(x) = x - \frac{1}{x_1^2 - \|x_{2:n}\|^2}\begin{pmatrix} x_1 \\ -x_{2:n} \end{pmatrix}, \quad \forall x \in K_q. \tag{3.2.7}$$

Since components of Tv_{xs} belong to either $I\!R_+$ or $I\!R_q$, it follows from (3.2.7) that

$$\nabla\Psi_1^{\Upsilon}(Tv_{xs}) = Tv_{xs} - \frac{1}{v_{xs}^T Q v_{xs}}\begin{pmatrix} (Tv_{xs})_1 \\ -(Tv_{xs})_{2:n} \end{pmatrix}. \tag{3.2.8}$$

By (3.2.5), (3.2.6) and (3.2.8), the right-hand side of fourth equation in (3.2.1) can rewritten as:

$$\begin{aligned} \nu\mu^{(0)}e - \tilde{X}\tilde{S}e &= \tilde{X}\left(\nu\mu^{(0)}\tilde{X}^{-1} - \tilde{S}e\right) \\ &= \sqrt{\nu\mu^{(0)}}\tilde{X}\left(\frac{1}{v_{xs}^T Q v_{xs}}\begin{pmatrix} (Tv_{xs})_1 \\ -(Tv_{xs})_{2:n} \end{pmatrix} - Tv_{xs}\right) \\ &= -\sqrt{\nu\mu^{(0)}}\tilde{X}\nabla\Psi_1^{\Upsilon}(Tv_{xs}) \\ &= -\sqrt{\nu\mu^{(0)}}\mathrm{mat}(T\tilde{x})\nabla\Psi_1^{\Upsilon}(Tv_{xs}). \end{aligned} \tag{3.2.9}$$

By (3.2.2) and (3.2.9), we get

$$\nu\mu^{(0)}e - \tilde{X}\tilde{S}e = -\nu\mu^{(0)}\mathrm{mat}(Tv_{xs})\nabla\Psi_1^{\Upsilon}(Tv_{xs}). \tag{3.2.10}$$

Now, we deal with the right-hand side of the last equation in (3.2.1). Let $\Psi_2^{\Upsilon}(x)$ be the function associated with $\Upsilon_{1,1}(t)$ and $I\!R_+$. Actually, in this case, $\Psi_2^{\Upsilon}(x) = \Upsilon_{1,1}(x)$ for all $x \in I\!R_+$. However, in order to keep the self-regular search directions with uniform notations, we still use $\Psi_2^{\Upsilon}(x)$ rather than $\Upsilon_{1,1}(x)$, even in this case. By definition, we see

$$\nabla\Psi_2^{\Upsilon}(x) = x - \frac{1}{x}, \qquad \forall x \in I\!R_+. \tag{3.2.11}$$

Therefore, the right-hand side of (3.2.1) can be rewritten as:

$$\nu\mu^{(0)} - \kappa\tau = \nu\mu^{(0)}v_{\kappa\tau}\left(\frac{1}{v_{\kappa\tau}} - v_{\kappa\tau}\right) = -\nu\mu^{(0)}v_{\kappa\tau}\nabla\Psi_2^{\Upsilon}(v_{\kappa\tau}). \tag{3.2.12}$$

By (3.2.10) and (3.2.12), we see that Newton system (3.2.1) is equivalent to the

following system with self-regular functions:

$$\begin{aligned} A\Delta x - b\Delta\tau &= r_p\nu - (Ax - b\tau), \\ -A^T\Delta y + c\Delta\tau - \Delta s &= r_d\nu - (-A^Ty + c\tau - s), \\ b^T\Delta y - c^T\Delta x - \Delta\kappa &= r_g\nu - (b^Ty - c^Tx - \kappa), \\ \tilde{X}T(\Theta G)^{-1}\Delta s + \tilde{S}T(\Theta G)\Delta x &= -\nu\mu^{(0)}\mathrm{mat}(Tv_{xs})\nabla\Psi_1^{\Upsilon}(Tv_{xs}) \ , \\ \kappa\Delta\tau + \tau\Delta\kappa &= -\nu\mu^{(0)}v_{\kappa\tau}\nabla\Psi_2^{\Upsilon}(v_{\kappa\tau}). \end{aligned} \tag{3.2.13}$$

Here, Ψ_1^{Υ} and Ψ_2^{Υ} are functions associated with the self-regular function $\Upsilon_{1,1}$ for quadratic cones and linear cones, respectively. Assume that ψ is an arbitrary self-regular function, Ψ_1 and Ψ_2 are functions associated with ψ for quadratics cones and linear cones, respectively. Then, by replacing $\Upsilon_{1,1}$ with ψ in (3.2.13) we can define a general self-regular system as follows:

$$\begin{aligned} A\Delta x - b\Delta\tau &= r_p\nu - (Ax - b\tau), \\ -A^T\Delta y + c\Delta\tau - \Delta s &= r_d\nu - (-A^Ty + c\tau - s), \\ b^T\Delta y - c^T\Delta x - \Delta\kappa &= r_g\nu - (b^Ty - c^Tx - \kappa), \\ \tilde{X}T(\Theta G)^{-1}\Delta s + \tilde{S}T(\Theta G)\Delta x &= -\nu\mu^{(0)}\mathrm{mat}(Tv_{xs})\nabla\Psi_1(Tv_{xs}) \ , \\ \kappa\Delta\tau + \tau\Delta\kappa &= -\nu\mu^{(0)}v_{\kappa\tau}\nabla\Psi_2(v_{\kappa\tau}), \end{aligned} \tag{3.2.14}$$

where v_{xs} and $v_{\kappa\tau}$ are given by (3.2.2) and (3.2.3), respectively.

We refer to the solutions of (3.2.14) as self-regular search directions. Therefore, by system (3.2.13) we see that Newton search directions are special self-regular search directions and the corresponding self-regular function is $\Upsilon_{1,1}$.

As stated in the previous chapter, when we solve optimization problems, we actually use the predictor-corrector techniques to solve the search directions in order to reduce the total number of iterations. By doing so, we approximate the second order terms $(\Delta\tilde{X})(\Delta\tilde{S})e$ and $\Delta\kappa\Delta\tau$ by known quantities instead of ignoring them. As before, Let E_{xs} and $E_{\kappa\tau}$ be the approximation terms to $(\Delta\tilde{X})(\Delta\tilde{S})e$ and $\Delta\kappa\Delta\tau$, respectively, and add E_{xs} and $E_{\kappa\tau}$ to system (3.2.14). Then we get the following

self-regular system with second order approximation terms:

$$\begin{aligned} A\Delta x - b\Delta\tau &= r_p\nu - (Ax - b\tau), \\ -A^T\Delta y + c\Delta\tau - \Delta s &= r_d\nu - (-A^T y + c\tau - s), \\ b^T\Delta y - c^T\Delta x - \Delta\kappa &= r_g\nu - (b^T y - c^T x - \kappa), \\ \tilde{X}T(\Theta G)^{-1}\Delta s + \tilde{S}T(\Theta G)\Delta x &= -\nu\mu^{(0)}\mathrm{mat}(Tv_{xs})\nabla\Psi_1(Tv_{xs}) - E_{xs}\ , \\ \kappa\Delta\tau + \tau\Delta\kappa &= -\nu\mu^{(0)}v_{\kappa\tau}\nabla\Psi_2(v_{\kappa\tau}) - E_{\kappa\tau}, \end{aligned} \tag{3.2.15}$$

where E_{xs} and $E_{\kappa\tau}$ are determined by the predictor-corrector methods later.

We mention that self-regular search directions were proposed by Peng, Roos and Terlaky (see, e.g., [18, 19, 20, 21]). Self-regular search directions have two remarkable features. First, self-regular search directions broaden search directions to solve optimization problems, including Newton search directions a special case. This allows the users to choose the appropriate search directions for their own purposes when necessary. In our implementation, we use the search directions produced by the self-regular functions $\Upsilon_{p,q}$ with different p and q. Second, the interior point methods with self-regular search directions have so far the best theoretical complexity bounds for large update methods. Here, the large update methods mean the centering parameter ν is reduced by a constant factor in every iteration. Actually, for any self-regular function with barrier degree $q = \log(k)$, the total number of iterations to find optimal solutions is bounded by $\sqrt{k}\log(k)\log(\frac{k}{\epsilon})$, where ϵ is the accuracy parameter and k is the number of cones in the SOCO problems. We refer the readers to [18, 20, 21] for the details about the complexity results for the SOCO problems.

In the next section, we describe the algorithms to solve SOCO problems by using self-regular search directions.

3.3 Algorithms with Self-Regular Search Directions

In this section, we discuss how to solve SOCO problems by using self-regular search directions. We first describe the procedure to solve the self-regular directions determined by system (3.2.15). Then, we present the algorithms that are based on the self-regular search directions and predictor-corrector techniques. Similar algorithms can also be found in book [21].

In order to solve system (3.2.15), let's denote by r_1, r_2, r_3, r_4, r_5 the correspond-

ing right-hand sides of (3.2.15), respectively, that is,

$$\begin{aligned} r_1 &= r_p\nu - (Ax - b\tau), \\ r_2 &= r_d\nu + (A^T y - c\tau + s), \\ r_3 &= r_g\nu - (b^T y - c^T x - \kappa), \\ r_4 &= -\nu\mu^{(0)}\mathrm{mat}(Tv_{xs})\nabla\Psi_1(Tv_{xs}) - E_{xs}, \\ r_5 &= -\nu\mu^{(0)}v_{\kappa\tau}\nabla\Psi_2(v_{\kappa\tau}) - E_{\kappa\tau}. \end{aligned} \tag{3.3.1}$$

Substituting (3.3.1) into (3.2.15), we get

$$\begin{aligned} A\Delta x - b\Delta\tau &= r_1, \\ -A^T\Delta y + c\Delta\tau - \Delta s &= r_2, \\ b^T\Delta y - c^T\Delta x - \Delta\kappa &= r_3, \\ \tilde{X}T(\Theta G)^{-1}\Delta s + \tilde{S}T(\Theta G)\Delta x &= r_4, \\ \kappa\Delta\tau + \tau\Delta\kappa &= r_5. \end{aligned} \tag{3.3.2}$$

We notice that system (3.3.2) has exactly the same form as system (2.2.12). Therefore, self-regular search directions can be solved by exactly the same procedure used to solve Newton search directions described in Chapter 2, just with different right-hand sides. The following is the sketch of this procedure to solve self-regular directions with predictor-corrector methods.

We first solve the predictor directions by solving the following linear system:

$$\begin{aligned} A\Delta x - b\Delta\tau &= -Ax + b\tau, \\ -A^T\Delta y + c\Delta\tau - \Delta s &= A^T y - c\tau + s, \\ b^T\Delta y - c^T\Delta x - \Delta\kappa &= -b^T y + c^T x + \kappa, \\ \tilde{X}T(\Theta G)^{-1}\Delta s + \tilde{S}T(\Theta G)\Delta x &= -\tilde{X}\tilde{S}e\ , \\ \kappa\Delta\tau + \tau\Delta\kappa &= -\kappa\tau, \end{aligned} \tag{3.3.3}$$

which is the same system used to solve the predictor directions for Newton search directions. Then we calculate the Newton steps for the predictor directions by formulas given in Section 3 of Chapter 2. After that, the centering parameter ν and

the second order approximation terms E_{xs} and $E_{\kappa\tau}$ are updated by formulas (2.5.2) and (2.5.3), respectively. Next, we use the updated centering parameter and second order approximation terms to solve system (3.2.15) for the corrector directions, which are also the self-regular search directions for the current iteration. Finally, the Newton steps for self-regular search directions are calculated, and thus the next iteration point follows immediately. In order to reduce the iteration number, the Newton step is chosen in such a way that the next iteration point is located in a neighborhood of the central path. In our implementation, we use self-regular functions to define such neighborhood. To this purpose, we denote by $v = \left(v^{(1)}, v^{(2)}, \cdots, v^{(k)}\right)$ with $v^{(i)} \in K_i$ $(i = 1, \cdots, k)$ for any vector $v \in K = K_1 \times K_2 \cdots \times K_k$. Let

$$\lambda_1(v^{(i)}) = v_1^{(i)} + \|v_{2:n_i}^{(i)}\| \quad \text{and} \quad \lambda_2(v^{(i)}) = v_1^{(i)} - \|v_{2:n_i}^{(i)}\|,$$

where $v^{(i)} = \left(v_1^{(i)}, v_2^{(i)}, \cdots, v_{n_i}^{(i)}\right)$. Then given positive constants ν and γ, the neighborhood of the central path with respect to ν and γ is defined as

$$\mathcal{N}(\nu, \gamma) = \left\{ (x, s, \tau, \kappa) : \ 2\psi(v_{\kappa\tau}) + \sum_{i=1}^{k} \left(\psi(\lambda_1(v_{xs}^{(i)})) + \psi(\lambda_2(v_{xs}^{(i)}))\right) < \gamma \right\}, \quad (3.3.4)$$

where $x, s \in \text{int}(K)$, $\tau, \kappa > 0$, v_{xs} and $v_{\kappa\tau}$ are given in (3.2.2) and (3.2.3), respectively. At each iteration, we need to shrink the Newton steps, if necessary, to make sure the next iteration point stay within the specified neighborhood. We notice that if (x, s, τ, κ) is on the central path, then we must have $(x, s, \tau, \kappa) \in \mathcal{N}(\nu, \gamma)$ since in that case, $\psi(v_{\kappa\tau}) = \psi(\lambda_1(v_{xs}^{(i)})) = \psi(\lambda_2(v_{xs}^{(i)})) = 0$ for all $i = 1, \cdots, k$.

Finally, we conclude this chapter with the algorithms that are based on the self-regular search directions.

Algorithms with Self-Regular Search Directions

---

input:

a self-regular function ψ;

an accuracy parameter $\epsilon > 0$;

an initial point $(x^{(0)}, \tau^{(0)}, y^{(0)}, s^{(0)}, \kappa^{(0)}) \in \text{int}(K) \times I\!R_+ \times I\!R^m \times \text{int}(K) \times I\!R_+$;

begin

$x = x^{(0)},\ \tau = \tau^{(0)},\ y = y^{(0)},\ s = s^{(0)},\ \kappa = \kappa^{(0)}$;

while $(x^T s + \kappa\tau) > \epsilon$ **do**

begin

solve predictor direction by (3.3.3) ;
calculate Newton step α_p by formulas in Section 2.4;
update centering parameter ν by (2.5.2);
solve corrector direction by (3.2.15) with E_{xs} and $E_{\kappa\tau}$ given by (2.5.3);
calculate Newton step α by formulas in Section 2.4;
calculate next iteration point:

$$x = x + \alpha\Delta x, \qquad \tau = \tau + \alpha\Delta\tau, \qquad y = y + \alpha\Delta y,$$
$$s = s + \alpha\Delta s, \qquad \kappa = \kappa + \alpha\Delta\kappa;$$

end

if $\tau > 0$, $(\frac{x}{\tau}, \frac{y}{\tau}, \frac{s}{\tau})$ is a primal-dual optimal solution;
else if $\kappa = 0$, problem is ill-posed;
else if $c^T x < 0$, dual problem is infeasible;
else if $b^T y > 0$, primal problem is infeasible.
end

end

Chapter 4

Interior Point Methods for SOCO Problems with Upper-Bound Constraints

In this chapter, we discuss interior point methods for the SOCO problems with upper-bound constraints. We first derive the linear systems that determine the self-regular search directions for this type of optimization problems. Then, we solve the linear systems and present the formulas that are used to calculate the search directions. We also describe the procedure to determine the Newton steps in this case. Finally, the algorithms to solve SOCO problems with upper-bound constraints are presented. For linear optimization with upper-bound constraints, we refer the reader to [13, 32].

4.1 SOCO Problems with Upper-Bounds

In this section, we deal with SOCO problems with upper-bound constraints. We first transform this type of problems into the standard SOCO problems, and then derive the linear systems that are used to determine the self-regular search directions for the present SOCO problems.

A SOCO problem with upper-bound constraints takes the following form:

$$\begin{aligned} \text{minimize} \quad & c^T x \\ \text{subject to} \quad & Ax = b, \\ & x_i \leq u_i, \quad i \in \mathcal{I}, \\ & x \in K, \end{aligned} \tag{4.1.1}$$

where $x, c \in I\!R^n$, $b \in I\!R^m$, A is an $m \times n$ matrix with $\text{rank}(A) = m$ and $m \leq n$, $\mathcal{I}$ is a subset of $\{1, \cdots, n\}$, $K = K_1 \times K_2 \times \cdots \times K_k$ with K_i $(i = 1, \cdots, k)$ being a linear, quadratic or rotated quadratic cone.

Next, we transform the upper-bound constraints to equality constraints by introducing a slack variable for each inequality constraint. Let $z_i = u_i - x_i$ for each $i \in \mathcal{I}$. Then the upper-bound constraints can be rewritten as:

$$Fx + z = u, \tag{4.1.2}$$

where $z = (z_1, \cdots, z_{m_u})^T \in I\!R^{m_u}_+$ with $m_u = |\mathcal{I}|$, $u = (u_1, \cdots, u_{m_u}) \in I\!R^{m_u}$, and F is an $m_u \times n$ matrix with each row being a unit vector. By (4.1.2), we see that problem (4.1.1) is equivalent to the following:

$$\begin{array}{lrcl} \text{minimize} & c^T x & & \\ \text{subject to} & Ax & = & b, \\ & Fx + z & = & u, \end{array} \tag{4.1.3}$$

where, $x \in K$, $z \in I\!R^{m_u}_+$. In order to solve problem (4.1.3), we first need to transfer it into the standard form. To this end, we denote by $\hat{K} = I\!R^{m_u}_+ \times K$, and

$$\hat{x} = \begin{pmatrix} z \\ x \end{pmatrix}, \quad \hat{c} = \begin{pmatrix} 0 \\ c \end{pmatrix}, \quad \hat{b} = \begin{pmatrix} u \\ b \end{pmatrix}, \quad \hat{A} = \begin{pmatrix} I & F \\ 0 & A \end{pmatrix}. \tag{4.1.4}$$

Then problem (4.1.3) can be rewritten as:

$$\begin{array}{ll} \text{minimize} & \hat{c}^T \hat{x} \\ \text{subject to} & \hat{A}\hat{x} = \hat{b}, \\ & \hat{x} \in \tilde{K}. \end{array} \tag{4.1.5}$$

We note that problem (4.1.5) is a standard SOCO problem that takes exactly the same form as problem (P). Hence, it follows from system (3.2.15) that the self-regular search directions for (4.1.5) are given by the solutions of the following linear

system:

$$\begin{aligned}
\hat{A}\Delta\hat{x} - \hat{b}\Delta\tau &= \hat{r}_p\nu - (\hat{A}\hat{x} - \hat{b}\tau), \\
-\hat{A}^T\Delta\hat{y} + \hat{c}\Delta\tau - \Delta\hat{s} &= \hat{r}_d\nu - (-\hat{A}^T\hat{y} + \hat{c}\tau - \hat{s}), \\
\hat{b}^T\Delta\hat{y} - \hat{c}^T\Delta\hat{x} - \Delta\kappa &= \hat{r}_g\nu - (\hat{b}^T\hat{y} - \hat{c}^T\hat{x} - \kappa), \\
\bar{X}\hat{T}(\hat{\Theta}\hat{G})^{-1}\Delta\hat{s} + \bar{S}\hat{T}(\hat{\Theta}\hat{G})\Delta\hat{x} &= -\nu\mu^{(0)}\mathrm{mat}(\hat{T}v_{\hat{x}\hat{s}})\nabla\Psi_1(\hat{T}v_{\hat{x}\hat{s}}) - E_{\hat{x}\hat{s}}\ , \\
\kappa\Delta\tau + \tau\Delta\kappa &= -\nu\mu^{(0)}v_{\kappa\tau}\nabla\Psi_2(v_{\kappa\tau}) - E_{\kappa\tau},
\end{aligned} \tag{4.1.6}$$

where $\hat{y} \in I\!R^{m+m_u}$, $\hat{s} \in \hat{K}$, $\kappa > 0$, $\tau > 0$, $\Delta\hat{y} \in I\!R^{m+m_u}$, $\Delta\hat{s} \in I\!R^{n+m_u}$, Ψ_1 and Ψ_2 are given by Definition 3.1.2 that are functions associated with a self-regular function ψ, $\hat{K}$ and $I\!R_+$, respectively. Also, $\hat{T}$, $\hat{\Theta}$ and $\hat{G}$ are matrices given by (1.2.3), (1.2.6) and (1.2.5) with respect to cone $\hat{K}$. As before, we here use the following notations:

$$\bar{X} = \mathrm{mat}(\hat{T}\hat{\Theta}\hat{G}\hat{x}), \tag{4.1.7}$$

$$\bar{S} = \mathrm{mat}(\hat{T}(\hat{\Theta}\hat{G})^{-1}\hat{s}), \tag{4.1.8}$$

$$v_{\hat{x}\hat{s}} = \hat{\Theta}\hat{G}\hat{x}/\sqrt{\nu\mu^{(0)}}, \tag{4.1.9}$$

$$v_{\kappa\tau} = \sqrt{\kappa\tau/\nu\mu^{(0)}}, \tag{4.1.10}$$

$$\hat{r}_p = (\hat{A}\hat{x}^{(0)} - \hat{b}\tau^{(0)})/\nu^{(0)}, \tag{4.1.11}$$

$$\hat{r}_d = (-\hat{A}^T\hat{y}^{(0)} + \hat{c}\tau^{(0)} - \hat{s}^{(0)})/\nu^{(0)}, \tag{4.1.12}$$

$$\hat{r}_g = (\hat{b}^T\hat{y}^{(0)} - \hat{c}^T\hat{x}^{(0)} - \kappa^{(0)})/\nu^{(0)}, \tag{4.1.13}$$

$$\mu^{(0)} = \left((\hat{x}^{(0)})^T\hat{s}^{(0)} + \kappa^{(0)}\tau^{(0)}\right)/(k + m_u + 1), \tag{4.1.14}$$

$E_{\hat{x}\hat{s}}$ and $E_{\kappa\tau}$ are approximation terms to $\mathrm{mat}(\hat{T}\hat{\Theta}\hat{G}\Delta\hat{x})$ and $\Delta\kappa\Delta\tau$, respectively.

Next, we simplify system (4.1.6). For convenience, we denote by:

$$\hat{y} = \begin{pmatrix} y_1 \\ y \end{pmatrix}, \qquad \Delta\hat{y} = \begin{pmatrix} \Delta y_1 \\ \Delta y \end{pmatrix}, \tag{4.1.15}$$

where $y_1,\ \Delta y_1 \in I\!R^{m_u}$ and $y,\ \Delta y \in I\!R^m$. We also partition $\hat{s}$ as follows:

$$\hat{s} = \begin{pmatrix} w \\ s \end{pmatrix}, \qquad \Delta\hat{s} = \begin{pmatrix} \Delta w \\ \Delta s \end{pmatrix}, \tag{4.1.16}$$

where $w \in I\!R^{m_u}_+,\ s \in K,\ \Delta w \in I\!R^{m_u}$ and $\Delta s \in I\!R^n$. Substituting (4.1.4) and (4.1.11) into the first equation of (4.1.6), we get

$$\begin{aligned} A\Delta x - b\Delta\tau &= r_{p_1}\nu - (Ax - b\tau), \\ -F\Delta x - \Delta z + u\Delta\tau &= r_{p_2}\nu - (-Fx - z + u\tau), \end{aligned} \tag{4.1.17}$$

where r_{p_1} and r_{p_2} are given by

$$r_{p_1} = \frac{Ax^{(0)} - b\tau^{(0)}}{\nu^{(0)}}, \tag{4.1.18}$$

and

$$r_{p_2} = \frac{-Fx^{(0)} - z^{(0)} + u\tau^{(0)}}{\nu^{(0)}}. \tag{4.1.19}$$

Substituting (4.1.4), (4.1.12), (4.1.15) and (4.1.16) into the second equation of (4.1.6), we have

$$-\Delta y_1 - \Delta w = -\tfrac{\nu}{\nu^{(0)}}\left(F^T y_1^{(0)} + A^T y^{(0)} - c\tau^{(0)} + s^{(0)}\right) + \left(F^T y_1 + A^T y - c\tau + s\right). \tag{4.1.20}$$

It follows from the first equation of (4.1.20) that

$$\Delta y_1 = \frac{\nu}{\nu^{(0)}}\left(y_1^{(0)} + w^{(0)}\right) - (y_1 + w) - \Delta w. \tag{4.1.21}$$

Substituting (4.1.21) into the second equation of (4.1.20), we find

$$-A^T\Delta y + F^T\Delta w + c\Delta\tau - \Delta s = r_d\nu - \left(-A^T y + F^T w + c\tau - s\right), \tag{4.1.22}$$

where r_d is given by

$$r_d = \frac{-A^T y^{(0)} + F^T w^{(0)} + c\tau^{(0)} - s^{(0)}}{\nu^{(0)}}. \tag{4.1.23}$$

By (4.1.4), (4.1.13) and (4.1.15), we infer that

$$\begin{aligned} u^T\Delta y_1 + b^T\Delta y - c^T\Delta x - \Delta\kappa &= \tfrac{\nu}{\nu^{(0)}}\left(u^T y_1^{(0)} + b^T y^{(0)} - c^T x^{(0)} - \kappa^{(0)}\right) \\ &\quad - \left(u^T y_1 + b^T y - c^T x - \kappa\right). \end{aligned} \tag{4.1.24}$$

Substituting (4.1.21) into (4.1.24), we get

$$b^T\Delta y - u^T\Delta w - c^T\Delta x - \Delta\kappa = r_g\nu - \left(b^T y - u^T w - c^T x - \kappa\right), \tag{4.1.25}$$

where r_g is given by

$$r_g = \frac{b^T y^{(0)} - u^T w^{(0)} - c^T x^{(0)} - \kappa^{(0)}}{\nu^{(0)}}. \tag{4.1.26}$$

Next, we simplify the fourth equation of (4.1.6). For this purpose, we need the following notations:

$$\hat{T} = \begin{pmatrix} T_1 & 0 \\ 0 & T \end{pmatrix}, \qquad \hat{\Theta} = \begin{pmatrix} \Theta_1 & 0 \\ 0 & \Theta \end{pmatrix}, \qquad \hat{G} = \begin{pmatrix} G_1 & 0 \\ 0 & G \end{pmatrix}, \tag{4.1.27}$$

where $T_1,\ \Theta_1$ and G_1 are $m_u \times m_u$ matrices, whereas, $T,\ \Theta$ and G are $n \times n$ matrices. We also partition vectors $v_{\hat{x}\hat{s}}$ and $E_{\hat{x}\hat{s}}$ as follows:

$$v_{\hat{x}\hat{s}} = \begin{pmatrix} v_{zw} \\ v_{xs} \end{pmatrix}, \qquad E_{\hat{x}\hat{s}} = \begin{pmatrix} E_{zw} \\ E_{xs} \end{pmatrix}, \tag{4.1.28}$$

where v_{zw} and E_{zw} are $m_u \times 1$ vectors, v_{xs} and E_{xs} are $n \times 1$ vectors.

Substituting (4.1.27) and (4.1.28) into the fourth equation of ((4.1.6)), we get

$$\begin{aligned} &\mathrm{mat}(T\Theta G x)T(\Theta G)^{-1}\Delta s + \mathrm{mat}(T(\Theta G)^{-1}s)T\Theta G\Delta x \\ &= -\nu\mu^{(0)}\mathrm{mat}(Tv_{xs})\nabla\Psi_1(Tv_{xs}) - E_{xs}, \end{aligned} \tag{4.1.29}$$

and

$$\begin{aligned} &\mathrm{mat}(T_1\Theta_1 G_1 z)T_1(\Theta_1 G_1)^{-1}\Delta w + \mathrm{mat}(T_1(\Theta_1 G_1)^{-1}w)T_1\Theta_1 G_1\Delta z \\ &= -\nu\mu^{(0)}\mathrm{mat}(T_1 v_{zw})\nabla\tilde{\Psi}_2(T_1 v_{zw}) - E_{zw}, \end{aligned} \tag{4.1.30}$$

where Ψ_1 and $\tilde{\Psi}_2$ are functions associated with K and $\mathbb{R}^{m_u}_+$, respectively. Since $z,\ w \in \mathbb{R}^{m_u}_+$, by Definition 1.1.6 and Definition 1.1.9 we know that

$$T_1 = I, \qquad G_1 = I, \qquad \Theta_1 = \sqrt{\frac{w}{z}}. \tag{4.1.31}$$

Plug (4.1.31) into (4.1.30). Then we find that

$$\mathrm{mat}(w)\Delta z + \mathrm{mat}(z)\Delta w = -\nu\mu^{(0)}\mathrm{mat}(v_{zw})\nabla\tilde{\Psi}_2(v_{zw}) - E_{zw}, \tag{4.1.32}$$

where v_{zw} is given by

$$v_{zw} = \sqrt{\frac{zw}{\nu\mu^{(0)}}}. \tag{4.1.33}$$

By (4.1.17), (4.1.22), (4.1.25), (4.1.29) and (4.1.32), it follows from (4.1.6) that the self-regular search directions for SOCO problems with upper-bound constraints are determined by the following linear systems:

$$\begin{aligned}
A\Delta x - b\Delta\tau &= r_{p_1}\nu - (Ax - b\tau),\\
-F\Delta x - \Delta z + u\Delta\tau &= r_{p_2}\nu - (-Fx - z + u\tau),\\
-A^T\Delta y + F^T\Delta w + c\Delta\tau - \Delta s &= r_d\nu - \left(-A^Ty + F^Tw + c\tau - s\right),\\
b^T\Delta y - u^T\Delta w - c^T\Delta x - \Delta\kappa &= r_g\nu - \left(b^Ty - u^Tw - c^Tx - \kappa\right),\\
\tilde{X}T(\Theta G)^{-1}\Delta s + \tilde{S}T\Theta G\Delta x &= -\nu\mu^{(0)}\mathrm{mat}(Tv_{xs})\nabla\Psi_1(Tv_{xs}) - E_{xs},\\
W\Delta z + Z\Delta w &= -\nu\mu^{(0)}\mathrm{mat}(v_{zw})\nabla\Psi_2(v_{zw}) - E_{zw},\\
\kappa\Delta\tau + \tau\Delta\kappa &= -\nu\mu^{(0)}v_{\kappa\tau}\nabla\Psi_3(v_{\kappa\tau}) - E_{\kappa\tau},
\end{aligned} \tag{4.1.34}$$

where $x,\ s \in K;\ \ z,\ w \in I\!R^{m_u}_+;\ \ y \in I\!R^m;\ \ \kappa,\ \tau \in I\!R_+;\ \ \Psi_1,\ \Psi_2$ and Ψ_3 are given by Definition 3.1.2 that are functions associated with $K,\ I\!R^{m_u}_+$ and $I\!R_+$, respectively. Also, the following notations are used in (4.1.34):

$$\tilde{X} = \mathrm{mat}(T\Theta Gx), \quad \tilde{S} = \mathrm{mat}(T(\Theta G)^{-1}s), \quad W = \mathrm{mat}(w), \quad Z = \mathrm{mat}(z),$$

$$v_{xs} = \frac{\Theta Gx}{\sqrt{\nu\mu^{(0)}}}, \quad v_{zw} = \sqrt{\frac{zw}{\nu\mu^{(0)}}}, \quad v_{\kappa\tau} = \sqrt{\frac{\kappa\tau}{\nu\mu^{(0)}}}, \tag{4.1.35}$$

and

$$\begin{aligned}
r_{p_1} &= (Ax^{(0)} - b\tau^{(0)})/\nu^{(0)},\\
r_{p_2} &= (-Fx^{(0)} - z^{(0)} + u\tau^{(0)})/\nu^{(0)},\\
r_d &= (-A^Ty^{(0)} + F^Tw^{(0)} + c\tau^{(0)} - s^{(0)})/\nu^{(0)},\\
r_g &= (b^Ty^{(0)} - u^Tw^{(0)} - c^Tx^{(0)} - \kappa^{(0)})/\nu^{(0)},
\end{aligned} \tag{4.1.36}$$

$$\mu^{(0)} = \left((x^{(0)})^Ts^{(0)} + (z^{(0)})^Tw^{(0)} + \kappa^{(0)}\tau^{(0)}\right)/(k + m_u + 1). \tag{4.1.37}$$

As a special case, if we take $\psi = \Upsilon_{1,1}$ that is given by (3.1.6), then the self-regular search directions determined by (4.1.34) reduce to the Newton search directions that

are given by the solutions of the following systems:

$$\begin{aligned}
A\Delta x - b\Delta\tau &= r_{p_1}\nu - (Ax - b\tau),\\
-F\Delta x - \Delta z + u\Delta\tau &= r_{p_2}\nu - (-Fx - z + u\tau),\\
-A^T\Delta y + F^T\Delta w + c\Delta\tau - \Delta s &= r_d\nu - \left(-A^Ty + F^Tw + c\tau - s\right),\\
b^T\Delta y - u^T\Delta w - c^T\Delta x - \Delta\kappa &= r_g\nu - \left(b^Ty - u^Tw - c^Tx - \kappa\right),\\
\tilde{X}T(\Theta G)^{-1}\Delta s + \tilde{S}T\Theta G\Delta x &= \nu\mu^{(0)}e - \tilde{X}\tilde{S}e - E_{xs},\\
W\Delta z + Z\Delta w &= \nu\mu^{(0)}e - ZWe - E_{zw},\\
\kappa\Delta\tau + \tau\Delta\kappa &= \nu\mu^{(0)} - \kappa\tau - E_{\kappa\tau}.
\end{aligned} \tag{4.1.38}$$

Once again, we see that Newton search directions are special cases of self-regular search directions.

In the next section, we discuss how to solve system (4.1.34) to find the self-regular search directions for SOCO problems with upper-bound constraints.

4.2 Search Directions

In this section, we derive explicit formulas for the solutions of system (4.1.34) that determines the self-regular search directions for the SOCO problems with upper-bound constraints. We here solve system (4.1.34) by following a similar procedure to solve system (3.2.15) for SOCO problems without upper-bounds constraints as discussed in Chapter 3.

For convenience, we denote by r_i $(i = 1, \cdots, 7)$ the corresponding right-hand

side of the ith equation in (4.1.34), that is,

$$\begin{aligned} r_1 &= r_{p_1}\nu - (Ax - b\tau), \\ r_2 &= r_{p_2}\nu - (-Fx - z + u\tau), \\ r_3 &= r_d\nu + (A^T y - F^T w - c\tau + s), \\ r_4 &= r_g\nu - (b^T y - u^T w - c^T x - \kappa), \\ r_5 &= -\nu\mu^{(0)}\mathrm{mat}(Tv_{xs})\nabla\Psi_1(Tv_{xs}) - E_{xs}, \\ r_6 &= -\nu\mu^{(0)}\mathrm{mat}(v_{zw})\nabla\Psi_2(v_{zw}) - E_{zw}, \\ r_7 &= -\nu\mu^{(0)}v_{\kappa\tau}\nabla\Psi_3(v_{\kappa\tau}) - E_{\kappa\tau}. \end{aligned} \tag{4.2.1}$$

Substituting (4.2.1) into (4.1.34) yields

$$\begin{aligned} A\Delta x - b\Delta\tau &= r_1, \\ -F\Delta x - \Delta z + u\Delta\tau &= r_2, \\ -A^T\Delta y + F^T\Delta w + c\Delta\tau - \Delta s &= r_3, \\ b^T\Delta y - u^T\Delta w - c^T\Delta x - \Delta\kappa &= r_4, \\ \tilde{X}T(\Theta G)^{-1}\Delta s + \tilde{S}T\Theta G\Delta x &= r_5, \\ W\Delta z + Z\Delta w &= r_6, \\ \kappa\Delta\tau + \tau\Delta\kappa &= r_7. \end{aligned} \tag{4.2.2}$$

From the last three equations of (4.2.2), we find that Δs, Δz and $\Delta\kappa$ can be expressed as:

$$\Delta s = \Theta G T\tilde{X}^{-1}r_5 - (\Theta G)^2\Delta x, \tag{4.2.3}$$

$$\Delta z = W^{-1}(r_6 - Z\Delta w), \tag{4.2.4}$$

$$\Delta\kappa = (r_7 - \kappa\Delta\tau)/\tau. \tag{4.2.5}$$

Substituting (4.2.3), (4.2.4) and (4.2.5) into (4.2.2), we get the following system:

$$\begin{aligned} A\Delta x - b\Delta\tau &= r_1, \\ -F\Delta x + W^{-1}Z\Delta w + u\Delta\tau &= r_2', \\ -A^T\Delta y + F^T\Delta w + c\Delta\tau + (\Theta G)^2\Delta x &= r_3', \\ b^T\Delta y - u^T\Delta w - c^T\Delta x + \tfrac{\kappa}{\tau}\Delta\tau &= r_4', \end{aligned} \tag{4.2.6}$$

where r_1', r_2' and r_3' are given by:

$$\begin{aligned} r_2' &= r_2 + W^{-1}r_6, \\ r_3' &= r_3 + \Theta G T\tilde{X}^{-1}r_5, \\ r_4' &= r_4 + \tfrac{r_7}{\tau}. \end{aligned} \tag{4.2.7}$$

It follows from the second equation of (4.2.6) that

$$\Delta w = Z^{-1}Wr_2' - Z^{-1}Wu\Delta\tau + Z^{-1}WF\Delta x. \tag{4.2.8}$$

Substituting (4.2.8) into (4.2.6), we have

$$\begin{aligned} A\Delta x - b\Delta\tau &= r_1, \\ -A^T\Delta y + \left(F^TZ^{-1}WF + (\Theta G)^2\right)\Delta x + \left(c - F^TZ^{-1}Wu\right)\Delta\tau &= r_3'', \\ b^T\Delta y - \left(u^TZ^{-1}WF + c^T\right)\Delta x + \left(u^TZ^{-1}Wu + \tfrac{\kappa}{\tau}\right)\Delta\tau &= r_4'', \end{aligned} \tag{4.2.9}$$

where r_3'' and r_4'' are given by

$$\begin{aligned} r_3'' &= r_3' - F^TZ^{-1}Wr_2', \\ r_4'' &= r_4' + u^TZ^{-1}Wr_2'. \end{aligned} \tag{4.2.10}$$

Note that system (4.2.9) can be put into the following matrix form:

$$\begin{pmatrix} A & 0 & -b \\ F^TZ^{-1}WF + (\Theta G)^2 & -A^T & c - F^TZ^{-1}Wu \\ -u^TZ^{-1}WF - c^T & b^T & u^TZ^{-1}Wu + \frac{\kappa}{\tau} \end{pmatrix} \begin{pmatrix} \Delta x \\ \Delta y \\ \Delta\tau \end{pmatrix} = \begin{pmatrix} r_1 \\ r_3'' \\ r_4'' \end{pmatrix}. \tag{4.2.11}$$

Introduce a matrix D as follows:

$$D = \left(F^T Z^{-1} W F + (\Theta G)^2\right)^{-\frac{1}{2}}. \tag{4.2.12}$$

Substituting (4.2.12) into (4.2.11), we have

$$\begin{pmatrix} A & 0 & -b \\ D^{-2} & -A^T & c - F^T Z^{-1} W u \\ -u^T Z^{-1} W F - c^T & b^T & u^T Z^{-1} W u + \frac{\kappa}{\tau} \end{pmatrix} \begin{pmatrix} \Delta x \\ \Delta y \\ \Delta \tau \end{pmatrix} = \begin{pmatrix} r_1 \\ r_3'' \\ r_4'' \end{pmatrix}. \tag{4.2.13}$$

It follows from the second equation of (4.2.13) that

$$\Delta x = D^2 r_3'' + D^2 A^T \Delta y - D^2 \left(c - F^T Z^{-1} W u\right) \Delta \tau. \tag{4.2.14}$$

Substituting (4.2.14) into (4.2.13), we get the following system for Δy and $\Delta \tau$:

$$\begin{pmatrix} AD^2 A^T & a_1 \\ -a_2 & a_3 \end{pmatrix} \begin{pmatrix} \Delta y \\ \Delta \tau \end{pmatrix} = \begin{pmatrix} r_1' \\ r_4''' \end{pmatrix}, \tag{4.2.15}$$

where a_1, a_2, a_3, r_1' and r_4''' are given by:

$$\begin{aligned} a_1 &= -b - AD^2 \left(c - F^T Z^{-1} W u\right), \\ a_2 &= -b^T + \left(u^T Z^{-1} W F + c^T\right) D^2 A^T, \\ a_3 &= \tfrac{\kappa}{\tau} + u^T Z^{-1} W u + \left(c^T + u^T Z^{-1} W F\right) D^2 \left(c - F^T Z^{-1} W u\right), \\ r_1' &= r_1 - AD^2 r_3'', \\ r_4''' &= r_4'' + \left(u^T Z^{-1} W F + c^T\right) D^2 r_3''. \end{aligned} \tag{4.2.16}$$

It follows form the last equation of (4.2.15) that

$$\Delta \tau = \frac{r_4''' + a_2 \Delta y}{a_3}. \tag{4.2.17}$$

By (4.2.15) and (4.2.17), we find that Δy is determined by

$$\left(AD^2 A^T + \bar{a}\hat{a}^T\right) \Delta y = \xi, \tag{4.2.18}$$

where $\bar{a}$, $\hat{a}$ and ξ are given by

$$\bar{a} = \frac{a_1}{a_3}, \qquad \hat{a} = a_2^T, \qquad \xi = r_1' - \frac{r_4'''}{a_3} a_1. \tag{4.2.19}$$

Now, let's summarize what we have done so far. In order to find the self-regular search directions from system (4.1.34), we first solve Δy from (4.2.18), then solve $\Delta\tau$ from (4.2.17), Δx from (4.2.14), Δw from (4.2.8), $\Delta\kappa$ from (4.2.5), Δz from (4.2.4), finally, Δs from (4.2.3). In other words, solving self-regular search directions consists of the following seven steps.

Step 1. Solve (4.2.18) for Δy;
Step 2. Calculate (4.2.17) for $\Delta\tau$;
Step 3. Calculate (4.2.14) for Δx;
Step 4. Calculate (4.2.8) for Δw;
Step 5. Calculate (4.2.5) for $\Delta\kappa$.
Step 6. Calculate (4.2.4) for Δz;
Step 7. Calculate (4.2.3) for Δs.

We notice that system (4.2.18) has exactly the same form as system (2.2.24) for SOCO problems without upper-bound constraints. Therefore system (4.2.18) can be solved by the methods and techniques discussed in Section 3 of Chapter 2. Since the sizes of systems (4.2.18) and system (2.2.24) are equal, the Cholesky factorization to solve (4.2.18) costs almost the same as (2.2.24). This means that the computational cost for solving SOCO problems with upper-bound constraints does not increase very much, although system (4.1.34) may be much larger than system (2.2.10).

In the next section, we calculate the Newton steps for the self-regular search directions.

4.3 Newton Steps

In this section, we determine the Newton steps for the self-regular search directions of the SOCO problems with upper-bound constraints. We only present the main steps in this case, since the details for calculations of Newton steps were already discussed in Section 4 of Chapter 2.

The procedure to compute the Newton steps consists of the following seven steps:

Step 1 Find maximal $\alpha_x > 0$ such that $x + \tilde{\alpha}_x \Delta x \in \text{int}(K)$ for all $0 < \tilde{\alpha}_x < \alpha_x$.
Step 2 Find maximal $\alpha_s > 0$ such that $s + \tilde{\alpha}_s \Delta s \in \text{int}(K)$ for all $0 < \tilde{\alpha}_s < \alpha_s$.
Step 3 Define $\alpha_z > 0$ as

$$\alpha_z = \begin{cases} +\infty, & \text{if } \Delta z_i \geq 0,\ \forall\ i = 1, \cdots, m_u; \\ \min\left\{-\frac{z_i}{\Delta z_i} :\ \Delta z_i < 0,\ i = 1, \cdots, m_u\right\}. \end{cases}$$

Step 4 Define $\alpha_w > 0$ as

$$\alpha_w = \begin{cases} +\infty, & \text{if } \Delta w_i \geq 0,\ \forall\ i = 1, \cdots, m_u; \\ \min\left\{-\frac{w_i}{\Delta w_i} :\ \Delta w_i < 0,\ i = 1, \cdots, m_u\right\}. \end{cases}$$

Step 5 Define $\alpha_\tau > 0$ as

$$\alpha_\tau = \begin{cases} +\infty, & \text{if } \Delta\tau \geq 0; \\ -\frac{\tau}{\Delta\tau}, & \text{if } \Delta\tau < 0. \end{cases}$$

Step 6 Define $\alpha_\kappa > 0$ as

$$\alpha_\kappa = \begin{cases} +\infty, & \text{if } \Delta\kappa \geq 0; \\ -\frac{\kappa}{\Delta\kappa}, & \text{if } \Delta\kappa < 0. \end{cases}$$

Step 7 Determine the maximum Newton step $\tilde{\alpha}$ as follows:

$$\alpha = \min\left\{\gamma\alpha_x, \gamma\alpha_s, \gamma\alpha_z, \gamma\alpha_w, \gamma\alpha_\tau, \gamma\alpha_\kappa, 1\right\}, \tag{4.3.1}$$

where γ is a fixed positive constant close to 1. If necessary, we shrink the maximum Newton step such that the next iteration point stays in a neighborhood of the central path.

We mention that the formulas to calculate α_x and α_s in Steps 1 and 2 were already presented in Section 4 of Chapter 2, and therefore the details are omitted here. Now, we are ready to describe the algorithms to solve the SOCO problems with upper-bound constraints, which are the topics of the next section.

4.4 Algorithms

In this section, we discuss our algorithms to solve the SOCO problems with upper-bound constraints by using the self-regular search directions. The predictor-corrector techniques are employed to approximate the second order terms.

The following is the idea of our algorithms. We first solve the predictor directions from system (4.1.38) with $\nu = 0$, $E_{xs} = 0$ and $E_{\kappa\tau} = 0$. The solutions of this system can be calculated by the formulas given in Section 2. Then we compute the Newton steps for the predictor directions by the formulas stated in Section 3. Next, the centering parameter ν and the second order approximation terms E_{xs} and $E_{\kappa\tau}$ are updated by formulas (2.5.2) and (2.5.3), respectively. After that, we solve the

corrector directions from system (4.1.34) with the updated centering parameter and second order approximation terms. Finally, we calculate the Newton steps for the corrector directions and define the next iteration point. The algorithms for SOCO problems with upper-bound constraints are described as follows.

Algorithms for SOCO Problems

---

input:
a self-regular function ψ;
an accuracy parameter $\epsilon > 0$;
an initial point $(x^{(0)}, z^{(0)}, \tau^{(0)}, y^{(0)}, s^{(0)}, w^{(0)}, \kappa^{(0)}) \in$
$\text{int}(K) \times I\!R^{m_u}_{++} \times I\!R_{++} \times I\!R^m \times \text{int}(K) \times I\!R^{m_u}_{++} \times I\!R_{++}$;
begin
$x = x^{(0)},\ z = z^{(0)},\ \tau = \tau^{(0)},\ y = y^{(0)},\ s = s^{(0)},\ w = w^{(0)},\ \kappa = \kappa^{(0)}$;
while $(x^T s + z^T w + \kappa\tau) > \epsilon$ **do**
begin
solve predictor directions by (4.1.38) ;
calculate Newton step α_p by formulas in Section 2.4;
update centering parameter ν by (2.5.2);
solve corrector directions by (4.1.34) with E_{xs} and $E_{\kappa\tau}$ given by (2.5.3);
calculate Newton step α by formulas in Section 2.4;
calculate next iteration point:

$$x = x + \alpha\Delta x, \quad z = z + \alpha\Delta z, \quad \tau = \tau + \alpha\Delta\tau, \quad y = y + \alpha\Delta y,$$
$$s = s + \alpha\Delta s, \quad w = w + \alpha\Delta w, \quad \kappa = \kappa + \alpha\Delta\kappa;$$

end
if $\tau > 0$, $(\frac{x}{\tau}, \frac{z}{\tau}, \frac{y}{\tau}, \frac{s}{\tau}, \frac{w}{\tau})$ is a primal-dual optimal solution;
else if $\kappa = 0$, problem is ill-posed;
else if $c^T x < 0$, dual problem is infeasible;
else if $b^T y - u^T w > 0$, primal problem is infeasible.
end
end

Chapter 5

Implementation of Interior Point Methods for SOCO Problems

In this chapter, we present the implementation issues of our SOCO solver. We first discuss the computational environment of the software in Section 5.1. Then, we describe the general structure of the program in Section 5.2. Section 5.3 is devoted to solving the search directions. The Watson Sparse Matrix Package (WSMP) [8] is introduced in this section and the procedure of data transfer between WSMP and MATLAB is explained in detail. Finally, in Section 5.4, we present our strategy for choosing starting points and discuss the stopping criteria that are used in our implementation. Some issues about line search are also addressed there.

5.1 Computational Environment

Our software package is built on an IBM RS-6000-44p-270 workstation with four processors. The operating system is IBM AIX 4.3. MATLAB 6.1 and IBM C programming Language are used to implement the software. Our software is based on MATLAB environment, but we also use the C language to implement the computationally demanding subroutines that may consume too much CPU time if they are written in MATLAB. The subroutines written in C are called from MATLAB via MEX[1] that are gateway functions from MATLAB to C environment.

In order to solve the Newton system efficiently, we use the Watson Sparse Matrix Package (WSMP) [8] as our large linear sparse system solver. As stated in our algorithms, to complete one iteration, we have to use Cholesky factorization of the Newton system again and again. But, the Cholesky factorization is an intermediate

[1]MEX files are MATLAB executable files

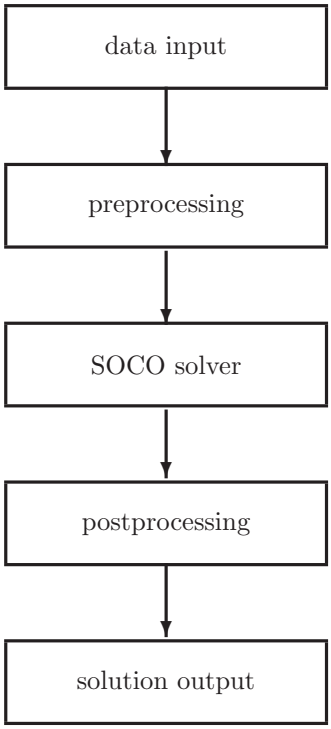


Figure 5.1: Structure of Program

result produced by WSMP, and it is not returned to the users. This causes tremendous complications to call WSMP from MATLAB. In order to apply the intermediate results stored internally in WSMP, we have to keep WSMP running until optimal solutions are found. Otherwise, all intermediate results produced by WSMP would be lost. Another problem to call WSMP from MATLAB is how to send data to WSMP and then retrieve data from WSMP. This problem will be discussed in detail in Section 5.3.

5.2 Program Structure

The flowchart of our program is shown in Figure 5.1 that consists of five parts: data input, preprocessing, SOCO solver, postprocessing, solution output. Next, we describe these five parts one by one in detail.

5.2.1 Data Input

Our software accepts data for SOCO problems stored in either MATLAB format or MPS format. The data stored in MATLAB format should contain seven quantities:

$$A,\ b,\ c,\ \text{lb},\ \text{ub},\ \text{BIG},\ K$$

representing the following SOCO problem:

$$\begin{aligned} \text{minimize} \quad & c^T x \\ \text{subject to} \quad & Ax = b, \\ & \text{lb} \leq x \leq \text{ub}, \\ & x^k \in K. \end{aligned}$$

Here, we denote by $x = (\hat{x}, x^k)$, $\hat{x}$ consists of all variable not in a quadratic or rotated quadratic cone, and $x^k = (x^k_{(1)}, x^k_{(2)}, \cdots, x^k_{(l)})$ with $x^k_{(i)} \in K_i$, $(i = 1, \cdots, l)$, where K_i is a quadratic or rotated quadratic cone. $K = K_1 \times K_2 \times \cdots, K_l$. BIG is a large positive number used for the variables without lower or upper bounds, i.e., $\text{ub}_i =$ BIG means that x_i has no upper bound; whereas, $\text{lb}_i = -$ BIG means that x_i has no lower bound. We assume that each component of $\hat{x}$ has a lower bound.

Our software also accepts data in MPS format that is the standard industrial input format for linear optimization problems. There is a software package for solving conic optimization problems called SeDuMi [26], which has a subroutine to transform the data for linear problems in the MPS format into the MATLAB format. Actually, SeDuMi performs the transformation by just calling the MPS reader subroutine contained in LIPSOL [31] that is a MATLAB based package for solving linear problems. In our implementation, we call the SeDuMi's reader subroutine to convert data from MPS format into MATLAB format, and it only works for linear optimization problems.

5.2.2 Preprocessing and Postprocessing

Preprocessing is a process to remove redundant constraints from the original optimization problems. This includes elimination of fixed variables and cancellation of zero rows and columns.

Postprocessing is a reverse process of preprocessing that recovers the original optimization problems. The main task of postprocessing is to produce solutions for the original problems, based on the solutions for the preprocessed problems.

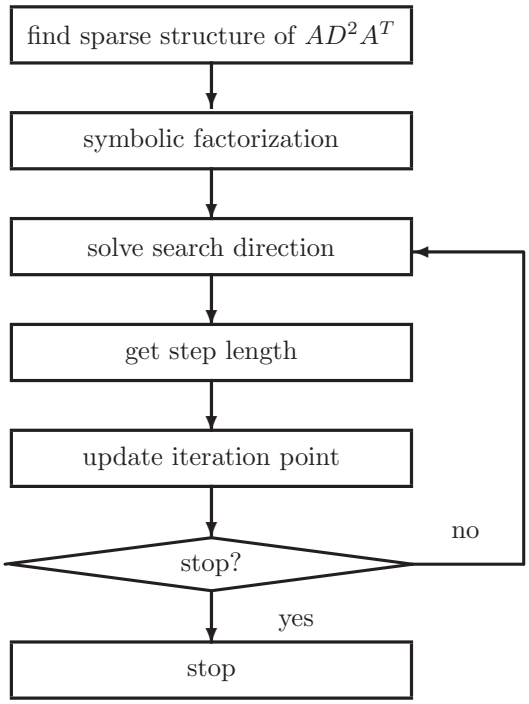


Figure 5.2: Structure of SOCO solver: the first three cases

We apply SeDuMi's preprocessing and postprocessing for the SOCO problems. SeDuMi actually invokes the preprocessing and postprocessing subroutines of LIPSOL in the linear case.

5.2.3 SOCO Solver

The task of our software is to solve the preprocessed optimization problems. The structure of our SOCO solver depends on the number of quadratic cones involved in the problems. We divide SOCO problems into four cases: (i) SOCO with no second order cones, (ii) SOCO with a small number of second order cones, (iii) SOCO with a large number of small second order cones, and (iv) general SOCO. Let nsoc be the number of the second order cones involved in a problem, and maxsoc be the maximal size of the cones. Then, we say a SOCO problem has a small number of second order cones if nsoc < 500. We say the problem has a large number of small second order

cones if nsoc ≥ 500 and maxsoc < 5. Accordingly, the criteria for classifying a SOCO problem are given by the following table:

	nsoc	maxsoc
case i	0	0
case ii	< 500	∞
case iii	≥ 500	< 5
case iv	≥ 500	≥ 5

Notice that, in case 1, the problem has no second order cones and thus the problem is actually a linear optimization. For the first three cases, the structures of our solver are sketched in Figure 5.2, whereas, for the last case, the structure of the solver is outlined in Figure 5.3. Next, we explain Figure 5.2 step by step in the linear case.

Find the sparse structure of AD^2A^T. The sparse structure of AD^2A^T is needed for WSMP to solve Newton system (4.2.18) efficiently. In order to get the sparse structure, we first need to separate the dense columns of A. Let m be the number of rows of A, and nnz be the number of nonzero elements in a column. Then, we say a column of A is dense if $\frac{\text{nnz}}{m} >$ nzratio, where nzratio is a specified number. Otherwise, we say the column is sparse. In our implementation, the value nzratio is defined according to the following table that depends on m, the row number of A:

$500 < m \leq 1000$	nzratio $= 0.2$
$1000 < m \leq 5000$	nzratio $= 0.1$
$m > 5000$	nzratio $= 0.05$

After separating the dense columns of A, then we determine the sparse structure of AD^2A^T. Assume $A = (A_s, A_d)$, where A_s consists of all sparse columns of A, and A_d of dense columns. Let $D = \text{diag}(D_s, D_d)$ corresponding to the partition of A. Then, the sparse structure of AD^2A^T is given by the sparse structure of $A_sD_sA_s^T$, which is sent to WSMP for solving a sparse linear system. Since the matrix D_s is diagonal, and therefore the sparse structure of $A_sD_sA_s^T$ can be determined by the sparse structure of $\tilde{A}\tilde{A}^T$, where $\tilde{A}$ is the matrix obtained by replacing every element of A with its absolute value. Note that the sparse structure of $\tilde{A}\tilde{A}^T$ is fixed for all iterations.

Symbolic factorization. We call WSMP to perform the symbolic factorization. For details about the WSMP, we refer the reader to [8].

Find search direction. We use WSMP to solve the Newton systems to get the search directions at each iteration. As stated before, since WSMP does not return useful intermediate results to the users, many difficulties arise to call WSMP from

MATLAB. Particularly, how to communicate between WSMP and MATLAB is a major issue in our implementation that is addressed in the next section.
Find step length. We first use the formulas given in Chapter 4 to calculate the maximal step length for each iteration. Then we shrink the maximal step length, if necessary, to make sure that the next iteration point stays within the desired neighborhood of the central path. More details are given in Section 5.4.
Update iteration point. Based on the current iteration point and the step length, calculate the next iteration point directly.
Check stopping criteria. Check if the current iteration point satisfies the stopping criteria. If the stopping criteria are satisfied, then terminate the program, otherwise, continue with the next iteration.

In the above, we explained the structure in Figure 5.2 for case i, the linear optimization. As we noticed early, this structure also works for cases ii and iii. However, in these cases, the computations of sparse structure of matrix AD^2A^T are different. This is because the sparse structure of AD^2A^T depends not only on the sparse structure of A, but also on the sparse structure of D. In the linear case, the matrix D given by (4.2.12) is a positive definite diagonal matrix, and it does not affect the sparse structure of AD^2A^T. That is why, in the linear case, we only need to separate the dense columns of A in order to get the sparse structure of AD^2A^T. However, this strategy to find the sparse structure of AD^2A^T does not work for nonlinear case, since, for nonlinear problems, the matrix D is no longer a diagonal matrix. Next, we discuss how to get the sparse structure of AD^2A^T in case ii, where the SOCO problem has a small number of second order cones.

Since, in case ii, a SOCO problem has only a few second order cones, D is a block-diagonal matrix with only a few blocks. By (4.2.12), we see that each block of D, corresponding to a second order cone, is a dense matrix. In this case, we can separate the dense blocks by using the formulas presented in Section 2.3.3. After performing this separation, the matrix AD^2A^T can be rewritten as

$$AD^2A^T = A\tilde{D}A^T + (R_1, R_2)(-R_1, R_2)^T, \tag{5.2.1}$$

where $\tilde{D}$, R_1 and R_2 are given by (2.3.22), (2.3.23) and (2.3.24), respectively. Note that the numbers of columns of R_1 and R_2 are both equal to the number of second order cones involved in the SOCO problem. Since, in this case, the SOCO problem has only a few second order cones, R_1 and R_2 do not have too many columns. This means that both R_1 and R_2 are low rank matrices, and therefore, the Sherman-Morrison formula can be used to solve the corresponding linear system efficiently, which is already discussed in Section 2.3.1. By (2.3.22), we see that $\tilde{D}$ is a positive definite diagonal matrix, and therefore, the sparse structure of $A\tilde{D}A^T$ can be found

exactly by the same procedure as for the linear case. After separating the dense columns of A, then we finally get

$$AD^2A^T = A_sD_sA_s^T + RS^T,$$

where $A = (A_s, A_d)$, A_s consists of all sparse columns of A, and A_d consists of all dense columns of A. $\tilde{D} = \text{diag}(D_s, D_d)$ that corresponds to the partition of A. $R = (A_dD_d, R_1, R_2)$ and $S = (A_dD_d, -R_1, R_2)$. Then the sparse structure of AD^2A^T is given by the sparse structure of $A_sD_sA_s^T$, which is sent to WSMP for solving a sparse linear system. Notice that the matrix D_s is diagonal, and therefore the sparse structure of $A_sD_sA_s^T$ can be determined by the sparse structure of $\tilde{A}\tilde{A}^T$, where $\tilde{A}$ is the matrix obtained by replacing every element of A_s by its absolute value. Clearly, the sparse structure of $\tilde{A}\tilde{A}^T$ is fixed for all iterations.

Next, we discuss the sparse structure of AD^2A^T in case iii. In this case, a SOCO problem has a large number of small second order cones, and therefore, D is block-diagonal matrix with a large number of small blocks. In this case, D can be considered as a sparse matrix. Let $\tilde{D}$ be the matrix obtained by replacing each element of the diagonal blocks by 1. Then, the positions of nonzero elements in D must correspond to the positions of nonzero elements in $\tilde{D}$. Therefore, in this case, the sparse structure of $A\tilde{D}A^T$ is preserved for all iterations.

We notice that, in the first three cases, the sparse structure of AD^2A^T we just found is fixed for all iterations. Therefore only one symbolic factorization is needed. However, for general problems, the sparse structure of matrix AD^2A^T may change, and thus symbolic factorization is needed at each iteration. In this case, the loop actually starts in the first step as shown in Figure 5.3

5.3 Search Directions

The search directions for each iteration are determined by the corresponding Newton systems. In order to solve the search directions efficiently, we need a high performance sparse linear system solver. In our implementation, we use the Watson Sparse Matrix Package (WSMP) to solve the large sparse Newton systems. In what follows, we first introduce WSMP, and then discuss how to use WSMP to get the search directions.

5.3.1 Watson Sparse Matrix Package

WSMP is a software package for solving sparse linear systems of the form $MX = B$, where M is a symmetric positive-definite or symmetric indefinite matrix. B is a

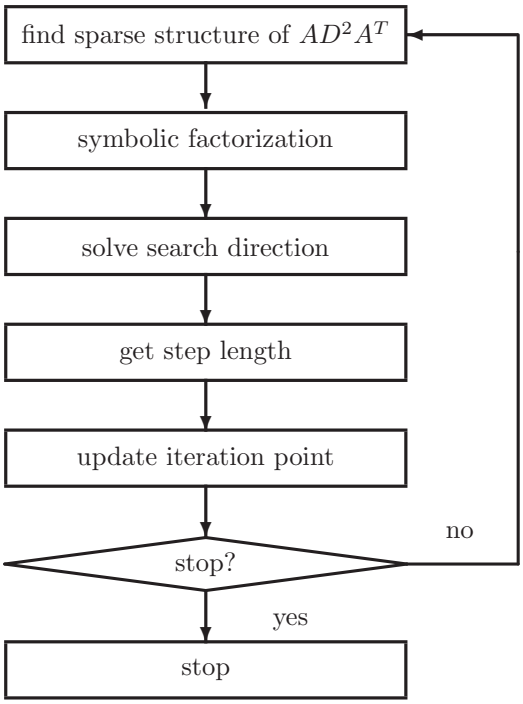


Figure 5.3: Structure of SOCO solver: general case

matrix with one or multiple columns. Since the matrix M is symmetric, WSMP only needs the lower triangular (including diagonal) portion of M in order to solve the systems. One way for WSMP to denote the sparse matrix M is to use three arrays: one double array AVALS, and two integer arrays IA and JA. This representation is the so-called compressed sparse row/column method. The array AVALS consists of all nonzero elements in the lower triangular portion of M, JA contains the column indices of each element in AVALS, and IA denotes the position in AVALS of the first nonzero element in each column.

WSMP has the following functionality on the matrix M: (1) perform permutation on M to minimize the possible nonzero elements produced during the factorization process; (2) perform symbolic factorization; (3) perform numerical factorization. After the factorization process, WSMP is ready to solve the system of equations $Mx = B$ by forward substitution and backward substitution. In order to get high precision solutions, WSMP also has a subroutine that performs iterative refinement and improve the precision of the solutions produced by forward and backward substitution. For more details about WSMP, we refer the reader to [8].

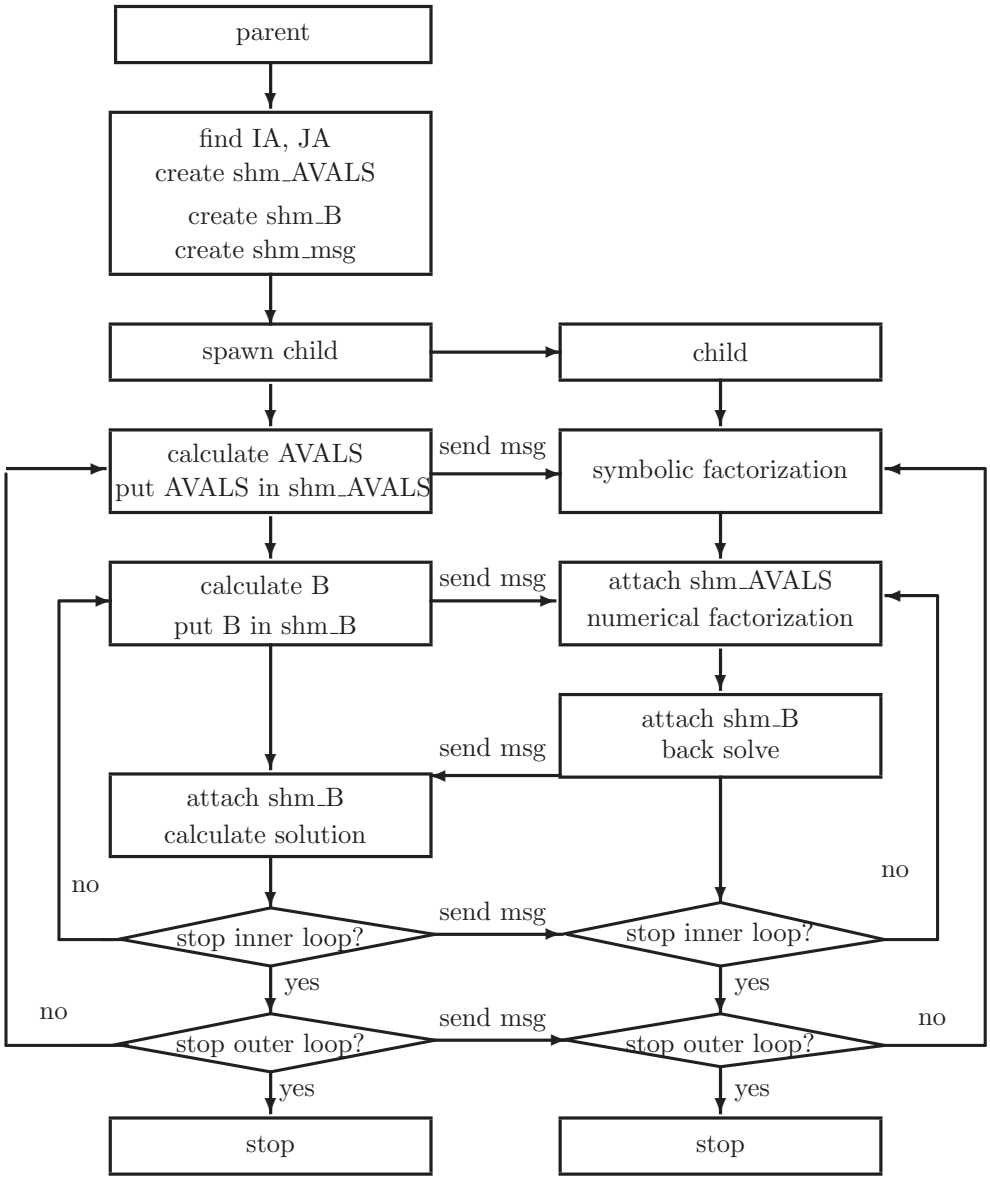


Figure 5.4: Data communication between MATLAB and WSMP: the first three cases

5.3.2 Data Communication

From the algorithms presented in Chapter 4, we see that the Cholesky factorization of the same coefficient matrix is repeatedly used within one iteration. This means that we have to keep the same Cholesky factorization until one iteration is completed. Unfortunately, the Cholesky factorization produced by WSMP becomes inaccessible when WSMP terminates. This is because the Cholesky factorization is stored internally in WSMP and not returned to the users. Therefore, in order to keep the Cholesky factorization accessible, we have to spawn a child process to run WSMP until optimal solutions are found. In other words, we have to run two processes simultaneously, one is the parent process, the other is the child process. The MATLAB files are executed by the parent process, whereas WSMP is executed by the child process. To run these two processes at the same time, one important issue arises: how to transfer data between the parent and the child processes. In our implementation, we use shared memory to solve this problem, that is, we store the data to be transferred in the shared memory such that the data are accessible for both the parent and the child processes. The data communication procedure for the first three cases is sketched in Figure 5.4. For general SOCO problems, the data communication procedure is outlined in Figure 5.5. Next, we elaborate Figure 5.4 step by step.

Step 1. Program starts running. This is the parent process.

Step 2. In this step, we first separate the dense columns and sparse columns of A and form AD^2A^T according to the procedure discussed in Chapter 2. This allows us to rewrite AD^2A^T as $AD^2A^T = P + RS^T$, where P is sparse, R and S are dense matrices with low ranks. The sparse matrix P is the matrix to be factorized by WSMP. Therefore, we need to represent P in the WSMP format, that is, we have to compute the integer arrays IA, JA, and double precision array AVALS for matrix P. IA and JA remain the same for all iterations since they depend only on the sparse structure of P, but AVALS keeps changing at each iteration since it depends on the values of the elements of P. In this step, we also create shared memory, shm_AVALS, to store the array AVALS that is used by WSMP. The shared memory shm_B is created for the parent process to send B to the child process. It is also used by the child process to send the solutions back to the parent process. In order to make sure the parent process and child process access the shared memory at the right time, they must get permission before they enter the shared memory. Another shared memory segment, shm_msg, is created to store the permission that is used to control who should access shm_AVALS and shm_B at a given time. The shared memory shm_msg actually serves as a message queue.

Step 3. In this step, the parent process spawn a child process, and pass IA and JA

to the child. The child process starts running WSMP.
Step 4. In this step, the outer loop begins. The parent process first calculates AVALS, and then puts AVALS into the shared memory shm_AVALS. In the meantime, the child process calls WSMP to perform the symbolic factorization if it is the first iteration. The symbolic factorization is performed only once. After the parent process completes the transfer of AVALS into shm_AVALS, it sends a message to the child process signaling that AVALS is ready for numerical factorization.
Step 5. In this step, the inner loop begins. The parent process first calculates the right-hand side B of the Newton systems and then put B into the share memory shm_B. In the meantime, the child process first attaches to the shared memory shm_AVALS and then performs numerical factorization. At the end of this step, the parent process sends a message to the child process signaling that B is ready for forward and backward substitution.
Step 6. The child process first attaches the shared memory shm_B and then calls WSMP to perform the forward and backward substitution. When the substitution is completed, the child process sends a message to the parent process signaling that the solutions produced by WSMP are available. In this step, the parent process does nothing, it is just waiting for the message.
Step 7. The parent process first attaches the shared memory shm_B, then retrieves the solutions produced by WSMP. Finally, the parent process calculates the search directions in terms of the formulas in Chapter 4.
Step 8. The parent process checks if the stopping criteria for the inner loop are satisfied, and sends the information to the child process. If the stopping criteria were satisfied, then both processes quit the inner loop, otherwise, the algorithm returns to Step 5.
Step 9. The parent process checks if the stopping criteria for the outer loop are satisfied, and sends the information to the child process. If the stopping criteria were satisfied, then both processes quit the outer loop, otherwise, the algorithm returns to Step 4.

We mention that the structure in Figure 5.4 only works for SOCO problems in the first three cases that are defined in Section 5.2.3. In these cases, IA and JA are invariant for all iterations. For general SOCO problems, since IA and JA may change at each iteration, we need to create the shared memory segments to store IA, JA and AVALS every time when the program enters the outer loop. These shared memory segments are removed by the child process after numerical factorization. The data communication procedure for a general SOCO problem is shown in Figure 5.5.

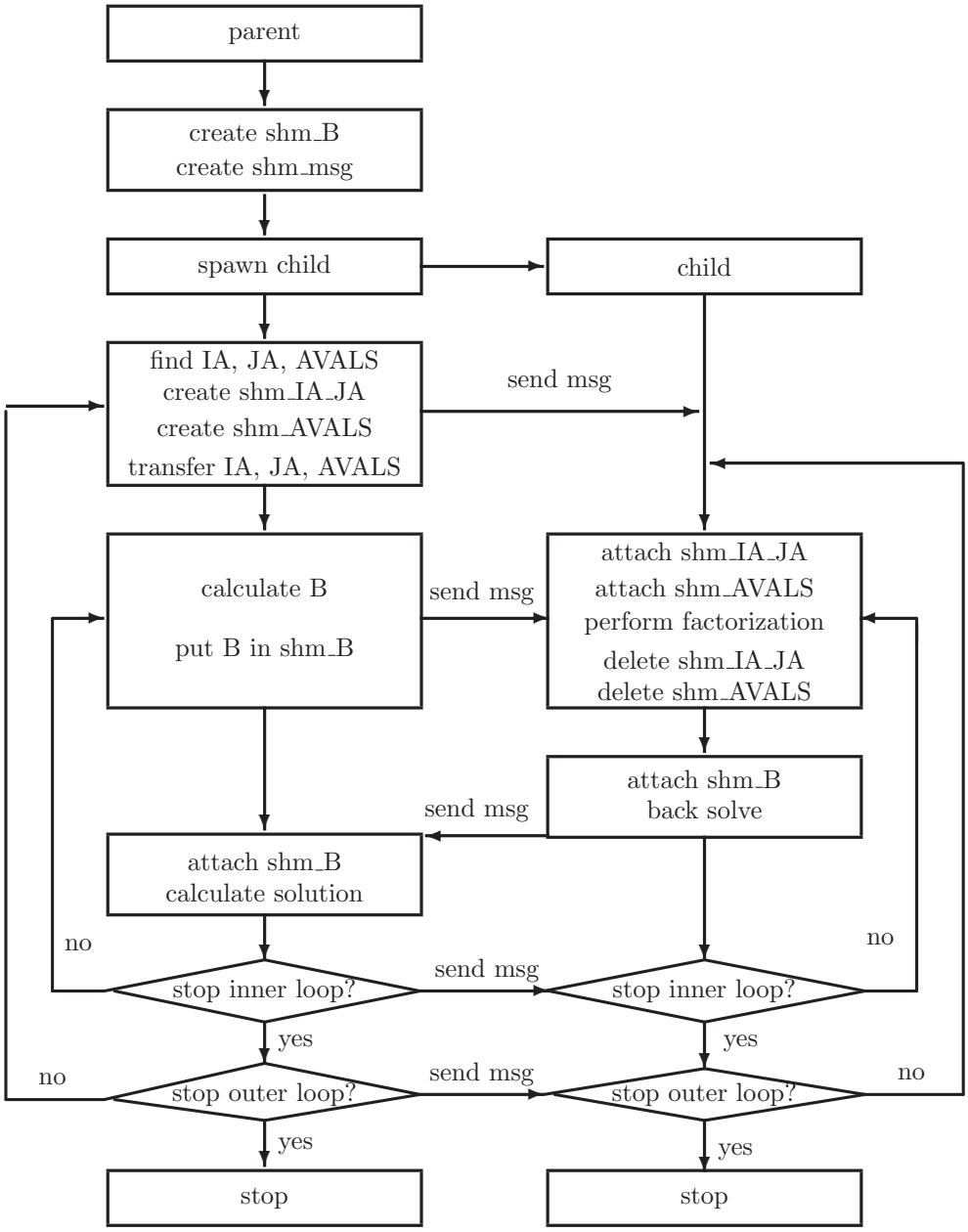


Figure 5.5: Data communication between MATLAB and WSMP: general case

5.4 Starting Points and Stopping Criteria

In this section, we present the starting points and stopping criteria used in our implementation, which are actually standard and also used in other implementations (see, e.g., [3]) as well.

5.4.1 Starting Points

The following are our starting points:

$$x^{(0)} = Te, \quad s^{(0)} = Te, \quad y^{(0)} = 0, \quad \tau^{(0)} = 1, \quad \kappa^{(0)} = 1.$$

5.4.2 Stopping Criteria

In order to determine when to terminate the program, we introduce the following quantities:

$$\rho_p^{(k)} = \frac{\|Ax^{(k)} - b\tau^{(k)}\|_\infty + \|Fx^{(k)} + z^{(k)} - u\tau^{(k)}\|_\infty}{\max(1, \|[A, b]\|_\infty, \|[F, u]\|_\infty)},$$

$$\rho_d^{(k)} = \frac{\|A^T y^{(k)} - F^T w^{(k)} + s^{(k)} - c\tau^{(k)}\|_\infty}{\max(1, \|[A^T, c]\|_\infty, \|F\|_\infty)},$$

$$\rho_g^{(k)} = \frac{|-c^T x^{(k)} + b^T y^{(k)} - u^T w^{(k)} - \kappa^{(k)}|}{\max(1, \|[c^T, b^T, u^T]\|_\infty)},$$

where $\rho_p^{(k)}$ is the scaled primal infeasibility measure, $\rho_d^{(k)}$ is the scaled dual infeasibility measure, whereas $\rho_g^{(k)}$ is the scaled gap infeasibility measure at iteration k. Denote by

$$\rho_a^{(k)} = \frac{|c^T x^{(k)}/\tau^{(k)} - b^T y^{(k)}/\tau^{(k)} + u^T w^{(k)}/\tau^{(k)}|}{10^{-10} + |c^T x^{(k)}/\tau^{(k)}|}.$$

Then, $-\log(\rho_a^{(k)})$ indicates the significant digits in the objective value at iteration k. We notice that if the kth iteration point is the exact optimal solution, then all the quantities defined above are zeros. Based on this fact, we introduce the following stopping criteria that are also used in [3]. The program is terminated if one of the following three conditions is satisfied:

(i) $\rho_p^{(k)} \leq tol\_\rho_p, \quad \rho_d^{(k)} \leq tol\_\rho_d, \quad \rho_a^{(k)} \leq tol\_\rho_a, \quad$ and $\tau^{(k)} > tol\_\rho_I \max(1, \kappa^{(k)})$;

(ii) $\rho_p^{(k)} \leq tol\_\rho_p, \quad \rho_d^{(k)} \leq tol\_\rho_d, \quad \rho_g^{(k)} \leq tol\_\rho_g, \quad \text{and } \tau^{(k)} \leq tol\_\rho_I \max(1, \kappa^{(k)});$

(iii) $g^{(k)} \leq \epsilon g^{(0)} \qquad \text{and} \quad \tau^{(k)} \leq tol\_\rho_I \min(1, \kappa^{(k)}),$

where tol_ρ_p, tol_ρ_d, tol_ρ_a, tol_ρ_g, tol_ρ_I and ϵ are all constants in $(0, 1]$ that can be specified by users. Next, we explain the above conditions one by one.

If the first condition is satisfied, then we consider the kth iteration point as almost feasible and optimal with large $\tau^{(k)}$. In this case, the solution

$$(x, y, s) = \left(\frac{x^{(k)}}{\tau^{(k)}}, \frac{y^{(k)}}{\tau^{(k)}}, \frac{s^{(k)}}{\tau^{(k)}} \right),$$

is reported to be a pair of optimal solutions of problems (P) and (D) (see, e.g., [3]).

If the second condition is satisfied, then we consider the kth iteration point as almost feasible with small $\tau^{(k)}$ (see, e.g., [3]). In this case, it follows from Theorem 1.3.1 that the primal or the dual problem is infeasible. If $b^T y^{(k)} - u^T w^{(k)} > 0$, then the primal problem is infeasible. If $c^T x^{(k)} < 0$, then the dual problem is infeasible.

If the last condition is satisfied, then the problem is reported to be ill-posed (see, e.g., [3]).

5.4.3 Dynamic Algorithm and Linear Search

In our implementation, we use an algorithm that dynamically chooses the barrier parameter q for self-regular search directions. This algorithm has been implemented for linear optimization in [33]. Next, we explain the algorithm in detail.

When the program starts, we set $q = 1$ for self-regular directions. In this case, self-regular search directions actually coincide with the Newton search direction. Then, we calculate the centering parameter ν by (2.5.2), and the maximal step length $\alpha\_\max$ by the formulas given in Section 4.3. If $\alpha\_\max$ for the current Newton search direction is not too small, say $\alpha > 0.001$, then we accept the current direction and keep $q = 1$ for the next iteration. Otherwise, we stay in the same point, increase q and re-solve the Newton system by re-using the same factorization for a new self-regular direction. When $q > 1$, the centering parameter ν is calculated by

$$\nu = \left(\frac{x^T s + z^T w + \tau\kappa}{(x^{\frac{1-q}{2}})^T s^{\frac{1-q}{2}}} \right)^{\frac{2}{q+1}}, \tag{5.4.1}$$

which is given in [24] for all $q > 1$, and in [22] for $q = 3$. The maximal step length $\alpha\_\max$ for $q > 1$ is calculated by the same formulas as $q = 1$. After the $\alpha\_\max$ is

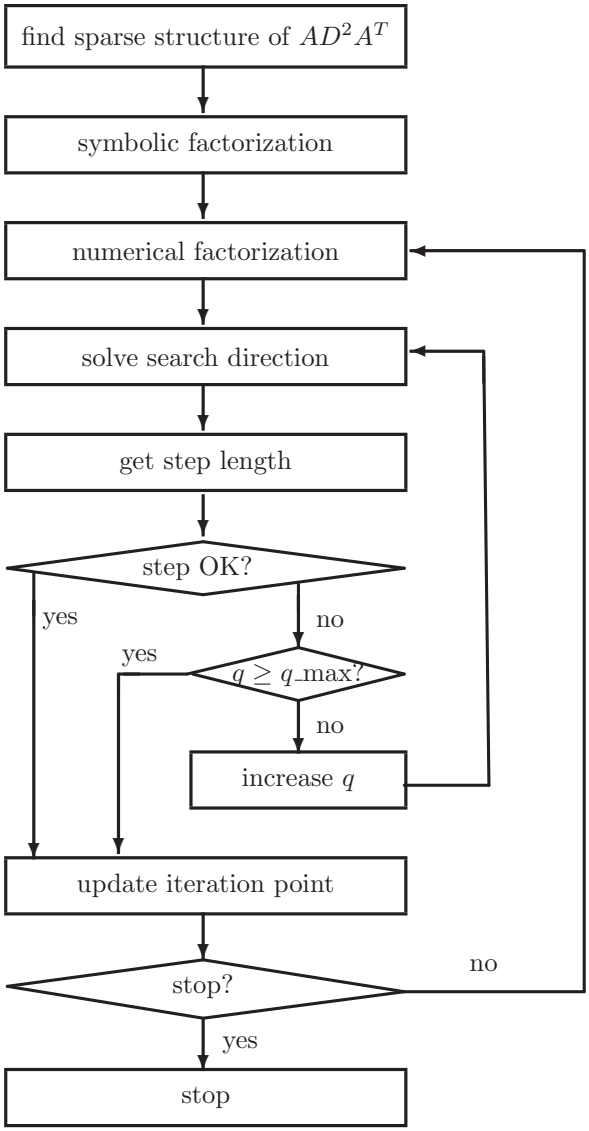


Figure 5.6: Structure of the dynamic self-regular IPM algorithm: the first three cases

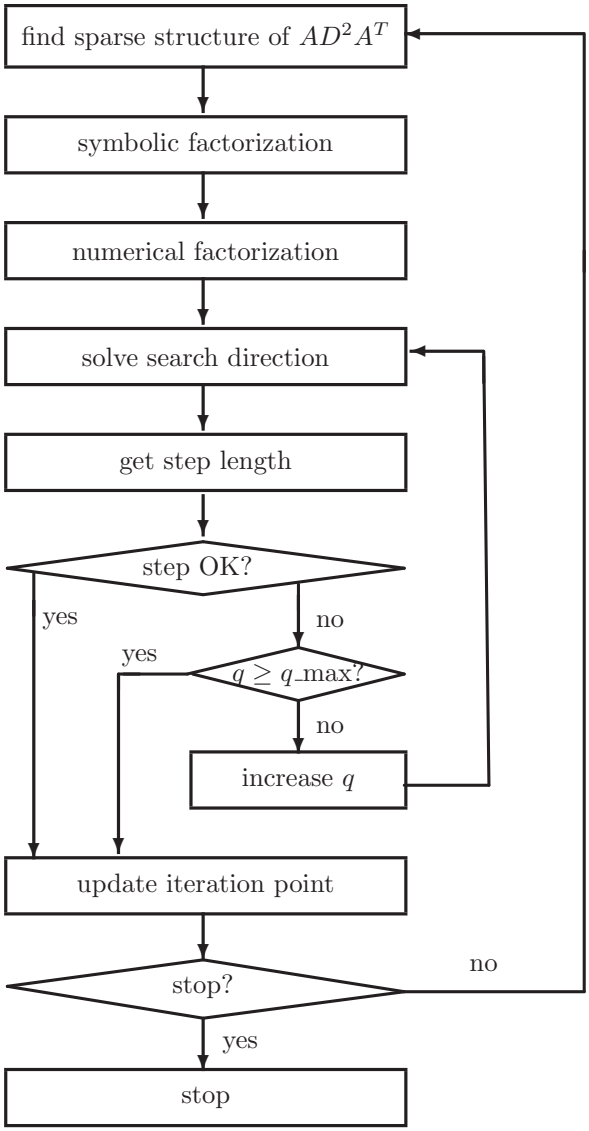


Figure 5.7: Structure of the dynamic self-regular IPM algorithm: general case

found, we shrink α_max by the back tracking strategy for $q = 1$, and the forward tracking strategy for $q > 1$, to ensure that the next iteration point stays in a specified neighborhood of the central path. The neighborhood of the central path is given by (3.3.4) in Section 3.3. By back tracking, we mean that α_max is cut back by a decreasing sequence of fractions like [0.9995, 0.95, 0.85, 0.70, 0.50]; whereas, by forward tracking, we mean that α_max is reduced by a increasing sequence of fractions like [0.5, 0.7, 0.85, 0.95].

In the first three cases, according to the classifications in Section 5.2.3, the structure of our implementation with the dynamic self-regular search directions is sketched in Figure 5.6. The structure of the algorithm with dynamic search directions for a general SOCO problem is outlined in Figure 5.7. In this case, the symbolic factorization is needed for all iterations.

Finally, we conclude this section by summarizing dynamic algorithms as follows.

Dynamic Algorithms for SOCO Problems

---

input:
a parameter q_max for maximum value of q;
an accuracy parameter $\epsilon > 0$;
an initial point $(x^{(0)}, z^{(0)}, \tau^{(0)}, y^{(0)}, s^{(0)}, w^{(0)}, \kappa^{(0)}) \in$
$\text{int}(K) \times I\!R^{m_u}_{++} \times I\!R_{++} \times I\!R^m \times \text{int}(K) \times I\!R^{m_u}_{++} \times I\!R_{++}$;
begin
$x = x^{(0)},\ z = z^{(0)},\ \tau = \tau^{(0)},\ y = y^{(0)},\ s = s^{(0)},\ w = w^{(0)},\ \kappa = \kappa^{(0)}$;
$q = 1$;
while $(x^T s + z^T w + \kappa\tau) > \epsilon$ **do**
begin
solve predictor directions by (4.1.38);
calculate Newton step α_p by formulas in Section 2.4;
update centering parameter ν by (2.5.2);
solve corrector directions by (4.1.34) with E_{xs} and $E_{\kappa\tau}$ given by (2.5.3);
calculate Newton step α by formulas in Section 2.4;
while $\alpha \leq 0.05$ and $q \leq q$_max **do**
begin
increase q, calculate ν by (5.4.1);
solve corrector directions by (4.1.34) with E_{xs} and $E_{\kappa\tau}$ given by (2.5.3);
calculate Newton step α by formulas in Section 2.4;

forward tracking;
end
back tracking;
set $q = 1$;
calculate next iteration point:
$x = x + \alpha\Delta x, \quad z = z + \alpha\Delta z, \quad \tau = \tau + \alpha\Delta\tau, \quad y = y + \alpha\Delta y,$
$s = s + \alpha\Delta s, \quad w = w + \alpha\Delta w, \quad \kappa = \kappa + \alpha\Delta\kappa;$
end
if $\tau > 0$, $(\frac{x}{\tau}, \frac{z}{\tau}, \frac{y}{\tau}, \frac{s}{\tau}, \frac{w}{\tau})$ is a primal-dual optimal solution;
else if $\kappa = 0$, problem is ill-posed;
else if $c^T x < 0$, dual problem is infeasible;
else if $b^T y - u^T w > 0$, primal problem is infeasible.
end
end

Chapter 6

Computational Results

In this chapter, we test our software, McIPM-SOC, by a set of SOCO optimization problems. We first present the computational results produced by McIPM-SOC for these problems. Then, we compare the numerical results obtained by McIPM-SOC with SeDuMi [26], which is another package to solve conic optimization problems.

All computational tests are performed on an IBM RS/6000 workstation with four processors and 8GB RAM. The operating system is IBM AIX 4.3. We refer the reader to Section 5.1 for more details about the computational environment.

6.1 Testing Results

Our testing problems are from the web site of the DIMACS[1]. The complete description of this set of problems is given in Table 6.1. The name of each problem is shown in the first column of Table 6.1. The numbers of rows, columns and nonzero elements of matrix A are displayed in the second, third and fourth columns, respectively. The fifth column contains the number of linear cones involved in the problem, and the last column contains the number of quadratic cones. In our implementation, we classify a SOCO problem in terms of the quadratic cones involved. According to the definition in Section 5.2.3, the classifications for problems of DIMACS set are listed in Table 6.2. From Table 6.1, we see that the last eight problems have one or two large second order cones, therefore the dense columns of the corresponding scaling matrices must be separated by the method presented in Section 2.3.3

We use two sets of parameters to test our software, which are presented in Table 6.3. The first column of Table 6.3 is for the name of each parameter, the second column is for the value in set I of the parameter, and the last column is for the value

[1] http://dimacs.rutgers.edu/Challenges/Seventh/Instance

Table 6.1: Problem Statistics

problem	rows	columns	nonzeros	linear cones	quadratic cones
nb	123	2383	192439	4	793
nb_L1	915	3176	193104	797	793
nb_L2	123	4195	404397	4	839
nb_L2_bessel	123	2641	209924	4	839
nql180	130080	226802	970919	129602	32400
nql30	3680	6302	26819	3602	900
nql60	14560	25202	107639	14402	3600
qssp180	130141	261366	1355051	2	65341
qssp30	3691	7566	36851	2	1891
qssp60	14581	29526	149291	2	7381
sched_100_100_orig	8338	18240	104902	10002	2
sched_100_100_scaled	8337	18238	114899	10002	1
sched_100_50_orig	4844	9746	55291	5002	2
sched_100_50_scaled	4843	9744	60288	5002	1
sched_200_100_orig	18087	37889	260503	20002	2
sched_200_100_scaled	18086	37887	280500	20002	1
sched_50_50_orig	2527	4979	25488	2502	2
sched_50_50_scaled	2526	4977	27985	2502	1

in set II. If the solutions converge within less than eighteen iterations, then the first, more demanding, parameter setting is used. Otherwise, the second one is used. Our computational results are shown in Table 6.4. There are seven columns in Table 6.4. The first column contains the name of the problem, the second column contains the primal objective value, The middle three columns are for the relative primal residual, dual residual and gap residual, respectively. The last two columns are the number of iterations and CPU time that are needed to solve the problems. From Table 6.4 we see that our software can solve most SOCO problems successfully, and produce high precision solutions. However, the biggest two problems, nql180 and qssp180, cannot be solved by our software at this moment since MATLAB complains about memory limitations and warns "out of memory". For problem sched_200_100_orig, the primal and dual residuals are between 10^{-4} and 10^{-5} that could be improved further.

6.2 Effect of the Barrier Degree q

In this section, we examine the performance of different self-regular search directions. As noticed earlier, a self-regular search direction is determined by a corresponding

Table 6.2: Classifications of Problems of DIMACS Set

problem	case i	case ii	case iii	case iv
nb			X	
nb_L1			X	
nb_L2				X
nb_L2_bessel				X
nql180			X	
nql30			X	
nql60			X	
qssp180			X	
qssp30			X	
qssp60			X	
sched_100_100_orig		X		
sched_100_100_scaled		X		
sched_100_50_orig		X		
sched_100_50_scaled		X		
sched_200_100_orig		X		
sched_200_100_scaled		X		
sched_50_50_orig		X		
sched_50_50_scaled		X		
total	0	8	8	2

Table 6.3: Parameter Setting

parameter	set I	set II
tol_ ρ_p	1×10^{-9}	2×10^{-6}
tol_ ρ_d	6×10^{-7}	1×10^{-5}
tol_ ρ_g	1×10^{-8}	2×10^{-6}
tol_ ρ_a	3×10^{-8}	2×10^{-6}
tol_ ρ_I	1×10^{-10}	1×10^{-8}
ϵ	1×10^{-10}	1×10^{-8}

Table 6.4: Computational Results with McIPM-SOC

problem	primal objective	R_p	R_d	R_g	iter	time(s)
nb	$-5.0701955507e-02$	$9.2e-11$	$2.2e-06$	$2.8e-08$	16	38.9
nb_L1	$-1.3011534400e+01$	$6.8e-11$	$3.6e-06$	$6.7e-08$	18	54.6
nb_L2	$-1.6279868740e+00$	$4.7e-09$	$4.8e-05$	$7.2e-07$	27	486.4
nb_L2_bessel	$-1.0254479106e-01$	$7.5e-10$	$7.6e-06$	$2.4e-07$	16	42.0
nql30	$-9.4602550159e-01$	$1.7e-09$	$7.3e-07$	$2.7e-08$	14	24.2
nql60	$-9.3504626551e-01$	$1.4e-09$	$5.1e-08$	$2.2e-08$	15	193.8
qssp30	$-6.4961604237e+00$	$5.5e-09$	$2.2e-06$	$7.9e-08$	22	45.3
qssp60	$-6.5621634624e+00$	$1.5e-09$	$1.0e-05$	$2.1e-08$	18	211.4
sched_100_100_orig	$7.1781951462e+05$	$4.0e-06$	$2.0e-05$	$7.9e-08$	38	136.2
sched_100_100_scaled	$3.0186852539e+01$	$6.1e-10$	$6.1e-10$	$3.0e-08$	32	111.3
sched_100_50_orig	$1.8220634711e+05$	$3.8e-06$	$1.8e-06$	$5.4e-07$	34	65.6
sched_100_50_scaled	$6.7256200278e+01$	$2.2e-09$	$2.0e-10$	$3.3e-07$	38	68.3
sched_200_100_orig	$1.4138947263e+05$	$3.0e-04$	$1.1e-05$	$1.2e-08$	41	368.2
sched_200_100_scaled	$5.2390827043e+01$	$9.9e-11$	$8.9e-10$	$4.6e-07$	47	408.0
sched_50_50_orig	$2.6682229467e+04$	$5.1e-06$	$2.9e-08$	$2.3e-07$	34	32.7
sched_50_50_scaled	$7.8585786983e+00$	$2.2e-10$	$1.2e-08$	$4.0e-07$	24	22.2

self-regular function with two parameters: growth parameter p and barrier parameter q. We here test the effect of self-regular search directions when q varies, but $p = 1$ is fixed. As a special case, the self-regular search direction is the same as the Newton search direction when both p and q are equal to 1. The algorithm with dynamic choice of q is described in Section 5.4.3, and the line search strategy in this case is also presented there. Note that both the back tracking and the forward tracking techniques are used in the dynamic algorithm for line search. However, when q is fixed, only the back tracking technique is needed. In other words, in this case, we use a decreasing sequence of fractions to cut back the maximal step length to ensure the next iteration point stays in the neighborhood (3.3.4) of the central path.

The numerical results for different search directions are presented in Table 6.5. Note that the last two columns of Table 6.5 are used to display the number of times of $q = 2$ and $q = 3$ in the dynamic algorithm, respectively. Next, we compare the performance of the self-regular directions with the Newton direction.

Table 6.5 shows that for problems nb_L2 and sched_200_100_scaled, the search directions with dynamic choice of q need fewer iterations than the Newton direction ($q = 1$). In other cases, the numbers of iterations for both cases are the same. We see that, for problem nb_L2, the dynamic algorithm changes the barrier parameter from 1 to 2, and then from 2 to 3. For problem sched_200_100_scaled, the barrier parameter is increased from 1 to 2. It is clear that, in these two cases, the total numbers of

Table 6.5: Performance of Search Directions with Different Barrier Parameters

	$q = 1$		$q = 2$		$q = 3$		dynamic q			
problem	iter	digit	iter	digit	iter	digit	iter	digit	$q = 2$	$q = 3$
nb	16	8	17	7	32	7	16	8	0	0
nb_L1	18	7	18	7	18	7	18	7	0	0
nb_L2	91	6	100	7	100	6	27	6	1	1
nb_L2_bessel	16	7	100	6	100	5	16	7	0	0
nql30	14	8	18	6	22	6	14	8	0	0
nql60	15	8	18	7	31	6	15	8	0	0
qssp30	22	7	72	9	100	4	22	7	0	0
qssp60	18	8	37	7	100	3	18	8	0	0
sched_100_100_orig	38	7	42	9	44	8	38	7	0	0
sched_100_100_scaled	32	8	41	7	34	8	32	8	0	0
sched_100_50_orig	34	6	41	8	42	8	34	6	0	0
sched_100_50_scaled	38	6	39	6	39	6	38	6	0	0
sched_200_100_orig	41	8	38	7	43	8	41	8	0	0
sched_200_100_scaled	50	6	50	6	51	6	47	6	1	0
sched_50_50_orig	34	7	35	6	36	6	34	7	0	0
sched_50_50_scaled	24	6	30	9	29	8	24	6	0	0

iterations are reduced by using self-regular search directions.

When $q = 2$, for thirteen problems, the self-regular directions take more iterations than the Newton direction. However, there is one problem, sched_200_100_orig, that need fewer iterations for self-regular directions. For problem nb_L1 and problem sched_200_100_scaled, the iteration numbers for $q = 2$ are the same as the Newton direction.

When $q = 3$, for fifteen problems, the self-regular directions take more iterations than the Newton direction. But for problem nb_L1, the iteration number for $q = 3$ is the same as the Newton direction.

Based on the above observation, we conclude that the self-regular directions with higher barrier parameter q still work well, although, these directions usually takes more iterations than the Newton direction. However, for some of the more difficult problems, a larger q results in less iterations. The self-regular directions with dynamic barrier parameter perform better than the Newton directions, and the dynamic directions usually need fewer iterations.

Table 6.6: Comparison with SeDuMi: DIMACS test problems

	McIPM-SOC			SeDuMi's Results		
problem	iter	residual	digit	iter	residual	digit
nb	16	$9.2e-11$	8	18	$7.7e-12$	11
nb_L1	18	$6.8e-11$	7	18	$2.1e-12$	10
nb_L2	27	$4.7e-09$	6	20	$5.5e-13$	10
nb_L2_bessel	16	$7.5e-10$	7	19	$4.6e-13$	9
nql30	14	$1.7e-09$	8	17	$6.2e-12$	8
nql60	15	$1.4e-09$	8	16	$6.8e-12$	8
qssp30	22	$5.5e-09$	7	20	$6.6e-12$	10
qssp60	18	$1.5e-09$	8	22	$2.5e-12$	10
sched_100_100_orig	38	$4.0e-06$	7	44	$1.2e-10$	8
sched_100_100_scaled	32	$6.1e-10$	8	54	$1.6e-11$	7
sched_100_50_orig	34	$3.8e-06$	6	43	$1.6e-11$	8
sched_100_50_scaled	38	$2.2e-09$	6	32	$4.2e-11$	7
sched_200_100_orig	41	$3.0e-04$	8	96	$2.3e-10$	11
sched_200_100_scaled	47	$9.9e-11$	6	54	$6.5e-11$	7
sched_50_50_orig	34	$5.1e-06$	7	46	$1.9e-11$	9
sched_50_50_scaled	24	$2.2e-10$	6	27	$2.5e-13$	10

6.3 Comparison Between Software Packages

In this section, we compare our software with SeDuMi [26]. The computational results produced by our package and SeDuMi are presented in Table 6.6 and Table 6.7. The problems solved in Table 6.6 are DIMACS test problems, and the problems solved in Table 6.7 are Netlib [7] test problems. In both tables, the first column of Table 6.6 contains the name of the problems. The middle three columns are used to display our results, and the last three columns to display SeDuMi's results. For each problem, the total number of iterations, the relative primal residual and the number of significant digits are listed in the tables. When a problem has no upper-bound constraints, then the number of the significant digit of a solution is calculated by the following formula that was used by SeDuMi:

$$\text{digit} = \begin{cases} -\log_{10} \frac{c^T x - b^T y}{|b^T y| + 10^{-10}}, & \text{if } c^T x - b^T y > 0; \\ \infty, & \text{otherwise.} \end{cases}$$

If a problem has upper bounds, then the significant digit is given by:

$$\text{digit} = \begin{cases} -\log_{10} \frac{c^T x - b^T y + u^T w}{|b^T y - u^T w| + 10^{-10}}, & \text{if } c^T x - b^T y + u^T w > 0; \\ \infty, & \text{otherwise.} \end{cases}$$

Table 6.7: Comparison with SeDuMi: Netlib test problems

	McIPM-SOC			SeDuMi's Results		
problem	iter	residual	digit	iter	residual	digit
80bau3b	43	$5.6e-08$	6	38	$2.0e-12$	8
bandm	19	$5.6e-07$	7	17	$5.2e-14$	14
beaconfd	17	$4.2e-07$	8	12	$4.3e-14$	16
bgprtr	16	infeasible		8	infeasible	
blend	17	$8.7e-10$	10	11	$2.6e-16$	15
bnl1	35	$1.8e-06$	6	42	$1.4e-16$	15
bnl2	41	$7.9e-07$	6	48	$6.9e-12$	10
bore3d	24	$1.1e-07$	7	24	$8.3e-14$	16
brandy	21	$2.2e-06$	7	17	$7.1e-14$	16
cplex2	33	infeasible		41	infeasible	
d2q06c	51	$2.8e-06$	6	40	$3.8e-11$	8
d6cube	23	$8.4e-08$	6	15	$7.5e-13$	12
degen2	18	$2.5e-08$	7	12	$9.3e-17$	16
degen3	19	$6.5e-08$	7	16	$2.7e-15$	15
e226	23	$2.7e-08$	7	22	$1.1e-10$	9
etamacro	28	$4.0e-08$	6	26	$6.1e-11$	8
finnis	30	$4.2e-07$	6	26	$4.3e-11$	9
fit1d	30	$8.8e-09$	8	29	$4.9e-13$	8
fit1p	19	$5.2e-08$	6	21	$9.3e-17$	16
fit2d	25	$2.5e-09$	8	24	$2.2e-10$	11
forplan	43	$1.0e-11$	6	27	$8.3e-11$	6
grow7	22	$1.5e-07$	7	16	$3.5e-16$	16
kb2	20	$5.3e-06$	7	18	$2.4e-13$	16
klein1	19	infeasible		20	infeasible	
klein2	20	infeasible		13	infeasible	
lotfi	28	$4.2e-09$	6	22	$2.3e-15$	15
maros-r7	18	$3.5e-06$	8	15	$1.0e-15$	15
modszk1	36	$1.4e-09$	6	27	$9.7e-11$	3
perold	50	$1.7e-09$	7	54	$3.0e-12$	9
pilot	55	$6.9e-10$	7	78	$2.8e-10$	10
pilotja	48	$1.3e-09$	6	49	$1.7e-10$	10
pilotwe	44	$4.7e-08$	6	33	$1.3e-10$	5
pilot4	41	$1.8e-09$	6	49	$4.5e-11$	10
pilotnov	34	$8.4e-10$	7	24	$1.6e-10$	9
reactor	23	infeasible		19	infeasible	
refinery	19	infeasible		19	infeasible	
scorpion	18	$8.3e-07$	6	11	$3.8e-16$	16
scrs8	30	$6.7e-08$	6	30	$1.2e-15$	15
sctap1	22	$2.6e-07$	6	18	$3.6e-16$	16

Table 6.6 shows that there are twelve problems that are solved with fewer iterations by our software than SeDuMi. There are three problems that need more iterations by our software. For problem nb_L1, both packages have the same iteration numbers. From Table 6.6, it is clear that the solutions produced by both packages are of high precision, and the precision of SeDuMi's solutions is even higher. In the future, we still need to improve the primal residuals produced by our package. We notice that our software cannot solve the biggest two problems since MATLAB terminates with the warning "out of memory", though they are solved by SeDuMi successfully. As we mentioned early, WSMP is used to solve Newton systems in our package. Since WSMP and MATLAB are not a integrated system, we have to transfer data between WSMP and MATLAB back and forth. This may slow down the program. Currently, all our CPU times to solve SOCO problems are longer than the SeDuMi's CPU times.

Our software is designed to solve SOCO problems rather than linear optimization. But as a special case of SOCO, our package is able to solve linear optimization problems as demonstrated by Table 6.7. From Table 6.7, we see that both packages can solve linear optimization problems successfully for feasible problems, and detect the infeasibility for infeasible problems. The solutions of both packages listed in Table 6.7 have high precision, and the precision of SeDuMi's solutions is even higher than ours.

Chapter 7

Conclusions

In this thesis, we have summarized the theory of SOCO problems and elements of self-regular IPMs. We have also discussed implementation issues of self-regular interior point algorithms that are designed to solve SOCO problems efficiently. In theoretic aspects, we have proposed a strategy to build the sparse structures of Newton systems for an SOCO problem. We have also proposed a method for computing the Newton step length for this problem.

In practical aspects, we have developed a software package to solve SOCO problems efficiently. We have implemented the proposed strategy for building the sparse structures of Newton systems. We have also implemented the back tracking and forward tracking techniques for determining the Newton step length. The Mehrotra's predictor-corrector technique has been incorporated in our package for reducing the total number of iterations. We have tested our software by the DIMACS set of test problems. We have also tested the performance of our program for different barrier parameters that determine the self-regular search directions.

The computational results are encouraging and demonstrate that our package is comparable with the state of the art software SeDuMi. However, as a useful software package, our implementation still has room to improve. Next we list a few of suggestions for future research.

(i) Handle numerical singularity of the matrix AD^2A^T. Currently, we use WSMP to deal with this problem. In order to develop a package independent of a third-party software, we need to implement our own subroutine to tackle the singularity of AD^2A^T.

(ii) Develop our own preprocessing and postprocessing subroutines. Currently, we use SeDuMi's preprocessing and postprocessing subroutines in our implementation. In order to develop a complete software, we need to implement these two

subroutines by ourselves in the future.

(iii) Investigate performance of different self-regular proximity functions, and propose an efficient strategy for choosing the proximity functions. Notice that, in our implementation, we classify SOCO problems into four cases depending on the cones involved in the problems. We need to test more and propose a best set of parameters for classifying an SOCO.

We notice that our software is designed for solving SOCO problems only. In the future, we plan to extend our package to cover more general nonlinear optimization problems, such as semi-definite optimization and nonlinear complementarity problem.

Bibliography

[1] F. Alizadeh, J.A. Haeberly, and M.L. Overton. Primal-dual interior-point methods for semidefinite programming: convergence rates, stability and numerical results. *SIAM J. Optim.*, 8(3):746-768, 1998.

[2] E.D. Andersen, A modified Schur complement method for handling dense columns in interior-point methods for linear programming. *ACM Trans. Math. Software*, 22(3):348-356, 1996.

[3] E.D. Andersen, C. Roos, and T. Terlaky. On implementing a primal-dual interior-point method for conic quadratic optimization. *Math. Programming*, 95(6):249-277, 2003.

[4] E.D. Andersen, C. Roos, T. Terlaky, T. Trafalis, and J.P. Warners. The use of low-rank updates in interior-point methods. AdvOl-Report No. 2000/9, Advanced Optimization Laboratory, Department of Computing and Software, McMaster University, Hamilton, Canada, 2000.

[5] A. Ben-Tal and A. Nemirovski. *Lectures on Mondern Convex Optimization: Analysis, Algorithms, and Engineering Applications*. MPS/SIAM Series on Optimization. SIAM 2001.

[6] M. Fukushima, Z. Luo, and P. Tseng. Smoothing functions for second-order cone complementarity problems. Preprint,Department of Mathematics, University of Washington, 2000.

[7] D.M. Gay. Electronic mail distribution of linear programming testing problems. *Mathematical Programming Society COAL Newsletter*, 13:10-12, 1985.

[8] A. Gupta. *WSMP: Watson Sparse Matrix Package*. Technical Report RC 21886. IBM T.J. Watson Research Center, 2002.

[9] W.W. Hager. Updating the inverse of a matrix. *SIAM Rev.*, 31(2):221-239, 1989.

[10] E. Klerk, C. Roos, and T. Terlaky. Initialization in semidefinite programming via a self-dual, skew-symmetric embedding. *Operations Research Letters*, 20:213-221, 1997.

[11] M. Kojima, S. Shindoh, and S. Hara. Interior-point methods for the monotone linear complementarity problem in symmetric matrices. *SIAM J. Optim.*, 7:86-125, 1997.

[12] S. Mehrotra. On the implementation of a primal-dual interior point method. *SIAM J. Optimization*, 2(4):575-601, 1992.

[13] Q. Miao. *Implementation of the New Interior-Point Methods.* M.Si. Thesis, Department of Computing and Software, McMaster University, 2002.

[14] R.D.C. Monteiro and T. Tsuchiya. Polynomial convergence of primal-dual algorithms for the second-order cone program based on the MZ-family of search directions. *Math. Programming*, 88(1):61-83, 2000.

[15] Y. Nesterov and A. Nemirovski. *Interior Point Polynomial Algorithms in Convex Programming.* SIAM, Philadelphia, 1994.

[16] Y. Nesterov and M.J. Todd. Self-scaled barriers and interior-point methods for convex programming. *Math. Oper. Res.,* 22(1):1-42, 1997.

[17] Y. Nesterov, M.J. Todd, and Y. Ye. Infeasible-start primal-dual methods and infeasibility detectors for nonlinear programming problems. *Math. Programming*, 84(2):227-267, 1999.

[18] J. Peng. *New Design and Analysis of Interior Point Methods.* Ph.D. Thesis, Faculty of Information Technology, Delft University of Technology, 2001.

[19] J. Peng, C. Roos, and T. Terlaky. Self-regular functions and new search directions for linear and semidefinite optimization. *Math. Programming*, 93(1): 129-171, 2002.

[20] J. Peng, C. Roos, and T. Terlaky. Primal-dual interior-point methods for second-order conic optimization based on self-regular proximities. *SIAM J. Optim*, 13(1): 179-203, 2002.

[21] J. Peng, C. Roos, and T. Terlaky, *Self-Regularity: A New Paradigm for Primal-Dual Interior-Point Methods.* Princeton University Press, 2002.

[22] J. Peng and T. Terlaky, A dynamic large-update primal-dual interior-point method for linear optimization. *Optim. Methods Software*, 17:1077-1104, 2003.

[23] C. Roos, T. Terlaky, and J.Ph Vial. *Theory and Algorithms for Linear Optimization. An Interior Point Approach.* John Wiley, Chichester, UK, 1997.

[24] M. Salahi and T. Terlaky, A self-regular proximity based dynamic large-update primal-dual interior-point method for linear optimization. Preprint, 2003.

[25] J.F. Sturm. *Primal-dual interior point approach to semidefinite programming.* Ph.D. Thesis, Tinbergen Institute, Erasmus University Rotterdam, 1997.

[26] J.F. Sturm. SeDuMi 1.02, a MATLAB toolbox for optimizing over symmetric cones. *Optim. Methods Software*, 11-12:625-653, 1999.

[27] T. Terlaky. An easy way to teach interior-point methods. *European J. Oper. Res.,* 130:1-19, 2001.

[28] T. Tsuchiya. A convergence analysis of the scaling-invariant primal-dual path-following algorithms for second-order cone programming. *Optim. Methods Software*, 11-12:141-182, 1999.

[29] Y. Ye. *Interior Point Algorithms, Theory and Analysis.* John Wiley, Chichester, UK, 1997.

[30] Y. Ye, M.J. Todd, and S. Mizuno. An $O(\sqrt{n}L)$ iteration homogeneous and self-dual linear programming algorithm. *Math. Oper. Res,* 19:53-67, 1994.

[31] Y. Zhang. User's guide to LIPSOL linear-programming interior point solver V0.4. *Optim. Methods Software*, 11-12:385-396, 1999.

[32] X. Zhu. *Implementing the New Self-Regular Proximity Based IPMs.* M. Sci. thesis, Advanced Optimization Laboratory, Department of Computing and Software, McMaster University, Hamilton, Canada, 2002.

[33] X. Zhu, J. Peng, T. Terlaky, and G. Zhang. On implementing self-regular proximity based feasible IPMs. AdvOl-Report No. 2003/2, Advanced Optimization Laboratory, Department of Computing and Software, McMaster University, Hamilton, Canada, 2003.