Implementation of Interior Point Methods for Mixed Semidefinite and Second Order Cone Optimization Problems

Jos F. Sturm*
Department of Econometrics, Tilburg University, The Netherlands.
j.f.sturm@kub.nl
http://fewcal.kub.nl/sturm

August 2, 2002

Abstract

There is a large number of implementational choices to be made for the primal-dual interior point method in the context of mixed semidefinite and second order cone optimization. This paper presents such implementational issues in a unified framework, and compares the choices made by different research groups. This is also the first paper to provide an elaborate discussion of the implementation in SeDuMi.

Keywords: Semidefinite Programming, Second Order Cone Programming, Linear Programming, Interior Point Method, Optimization.

AMS subject classification: 90C22, 90C20.

JEL code: C61.

1 Introduction and Overview

The study of modern interior point methods was initiated by Karmarkar [32] for linear programming. A good overview of the developments in the first five years following the publication of [32] is provided by Gonzaga [28]. The introduction of the primal-dual interior point method by Kojima et al. [35, 34] had a huge impact on the research in this area after 1989. Their work removed the need for barrier, center and potential functions. This freedom was used by Lustig, Marsten and Shanno [43, 44, 46] to build highly effective interior point based solvers for linear programming. A further landmark was the introduction of the self-dual embedding technique by Ye et al. [85, 84], which provides a more elegant method for dealing with (in)feasibility issues than the infeasible interior point framework of Lustig [42]. A more complete (and probably less biased) overview is given by Freund and Mizuno [19]. A nice overview focusing on the implementational aspects is provided by Andersen et al. [6].

*Research supported by the Netherlands Organisation for Scientific Research, file 016.025.026.

During the nineties, it was discovered by Alizadeh [2] and Nesterov and Nemirovsky [56] that the interior point method was especially suited for solving semidefinite programming problems. The possibility of efficiently solving semidefinite programming problems had a huge impact on research in various application areas, see Boyd et al. [11, 79].

Initially, research papers gave the impression that extension of the interior point methodology to semidefinite programming was rather straightforward. The current insight however is that a substantial research effort on the interior point method for semidefinite programming is still necessary. One of the first surprises came when several research groups each introduced quite different generalizations of the primal-dual interior point method to semidefinite programming. In particular, Alizadeh et al. [5] introduced the AHO direction, Helmberg et al. [30] introduced the HRVW/KSH/M direction, or more concisely the HKM direction, Nesterov and Todd [57, 58] introduced the NT direction, and Kojima et al. [37] introduced a whole family of search directions, including the HKM and NT directions. Sturm and Zhang [72, 73] extended the primal-dual framework of Kojima et al. [34] to semidefinite programming, thus allowing for other search directions than those dictated by barrier, potential and center functions; see e.g. [33, 66]. Monteiro and Zhang [55, 86] introduced the so-called similarity symmetrization operator, which allows the study of AHO, HKM and NT search directions in a unified fashion; see e.g. [75]. More recent approaches include [52, 38].

In an attempt to extend the primal-dual approach beyond semidefinite programming, Nesterov and Todd [57, 58] introduced the concept of self-scaled cones. The gain in generality appeared to be rather limited when it later turned out that self-scaled cones are not more general than symmetric cones, which can always been described as conic sections of the cone of positive semidefinite (p.s.d.) matrices in a polynomially bounded dimension [14]. Nevertheless, [57, 58] can be considered as the first papers dealing with mixed semidefinite and second order cone optimization problems. However, the area was really brought to life by Alizadeh et al. [3] with the introduction of SDPPack, a software package for solving optimization problems from this class. The practical importance of second order cone programming was demonstrated by Lobo et al. [39] and many later papers. From then on, it was no longer sufficient to treat second order cone programming as a special case of semidefinite programming. Faybusovich [15, 17, 16, 18] demonstrated that the well developed theory of Euclidean Jordan algebra provides an elegant approach to study semidefinite and second order cone programming in a unified way.

There are other approaches to solve semidefinite programs than the primal-dual interior point method. Such approaches include dual-only interior point methods, bundle methods, augmented Lagrangian methods, non-smooth Newton methods, among others. Such approaches are not discussed in this paper. We refer to [81] for an elaborate discussion of results in semidefinite programming.

This paper is organized as follows. Section 2 presents the standard form of the optimization problems that we study in this paper. Rather intimidating notation is introduced to deal with linear, semidefinite and second order cone constraints explicitly. Fortunately, it is often possible to treat these constraints in a unified fashion. For this purpose, we introduce Jordan algebra notation in Section 3. In Section 4, we give a brief outline of the generic primal-dual interior point method. An important design parameter in the primal-dual interior point method is the scaling operator. We discuss several scaling operators in Section 5.

A computationally demanding part of the interior point method is the so-called *Building Phase*, in which the system that defines the search direction is formed. This is in fact a system of normal equations, and Section 6 presents sparsity exploiting techniques for this phase. The next step is to factor this system, the so-called *Factorization Phase*. In this step, it is also crucial to exploit sparsity, but numerical issues have to be addressed as well. The *Factorization Phase* is discussed in Section 7. In order to improve sparsity in the normal equations system, it can be fruitful to handle dense columns individually, as discussed in Section 8. In Section 9, we show how the structure of simple upper bound and fixing constraints can be exploited to reduce the size of the matrix that has to be factored.

In Section 10 we describe a method to update the scaling operator and the iterates in a product form. This method avoids numerical problems in this *Update Phase* near the optimal solution set. In Section 11, we describe the Mehrotra-type predictor-corrector approach to construct the search direction (we call this the *Step Phase* of the algorithm). We show that there are different ways to extend Mehrotra's scheme from linear programming, leading to different second order search directions, even if one sticks to a particular type of scaling. The issues of initialization and detecting infeasibility are discussed in Section 12. We discuss both the infeasible interior point approach and the self-dual embedding approach. In Section 13, we evaluate the computational profile of SeDuMi 1.05 on a set of problems collected by [60]. It turns out that three of the four distinguished computational phases in the algorithm can be viewed as critical. Hence, there is not a single bottleneck. We conclude the paper in Section 14.

2 Primal and Dual Problems

We study so-called *cone linear programming* problems in the standard canonical form:

$$\inf\{c^{\mathrm{T}}x \mid Ax = b,\, x \in \mathcal{K}\}, \tag{1}$$

where $x \in \Re^n$ is the vector of decision variables, $\mathcal{K} \subseteq \Re^n$ is a pre-specified convex cone, and $b \in \Re^m$, $c \in \Re^n$, $A \in \Re^{m\times n}$ are given data. Basic properties of cone linear programs were derived in Duffin [13], and more recently in [40, 56]. Despite its name, cone linear programs are non-linear, since $\mathcal{K}$ need not be polyhedral.

Important subclasses of cone linear programming are linear programming, semidefinite programming, second order cone programming, and a mixture of these. These subclasses arise by letting $\mathcal{K}$ in (1) be the nonnegative orthant ($\mathcal{K} = \Re^n_+$), the cone of positive semidefinite matrices, a Cartesian product of Lorentz cones, or a symmetric cone [14], respectively.

In order to comply with the canonical form (1), the cone of positive semi-definite matrices should be considered as a set of vectors in $\Re^n$, even if the natural definition is of course in terms of $\nu \times \nu$ matrices. This issue is easily resolved by vectorization ('column stacking'): we may let

$$\mathcal{K}^s = \{\, \mathrm{vec}\,(X) \mid X \in \Re^{\nu\times\nu} \text{ is symmetric positive semi-definite}\}.$$

In this case, $\mathcal{K}^s \subset \Re^n$ with $n = \nu^2$. (The superscript 's' stands for semidefinite.) Since the vectorized matrices are symmetric, $\mathcal{K}^s$ is actually a cone in an $(n+\nu)/2$ dimensional subspace of

$\Re^n$. (Alternatively, one may represent the symmetric positive semidefinite matrices as a cone in $\Re^n$ with $n = \nu(\nu+1)/2$ using the more economical symmetric vectorization [5].) We remark that if X is a Hermitian matrix with complex, quaternion or octonion entries, then X can be vectorized to $\Re^n$ with $n = \nu^2$, $n = \nu(2\nu-1)$ and $n = \nu(4\nu-3)$, respectively. The solvers SDPT3 (up to Version 2.3) [76] and SeDuMi [67] do support complex values, but quaternion or octonion values are not supported. Throughout this paper, we restrict ourselves to the real symmetric case.

The Lorentz cone in $\Re^n$ is defined as

$$\mathcal{K}^q = \left\{ x \in \Re^n \,\middle|\, x_1 \geq \sqrt{\sum_{i=2}^{n} x_i^2} \right\}. \tag{2}$$

(The superscript 'q' stands for *quadratic*.) An example of a second order cone problem is

$$\inf \left\{ x_1 - x_2 \,\middle|\, x_1 \geq \sqrt{1 + x_2^2} \right\},$$

which fits (1) with $\mathcal{K} = \mathcal{K}^q$ and $c = \begin{bmatrix} 1, & -1, & 0 \end{bmatrix}^{\mathrm{T}}$, $A = \begin{bmatrix} 0, & 0, & 1 \end{bmatrix}$, and $b = 1$. In this example, the objective value $x_1 - x_2$ can be arbitrarily close to zero, but the infimum (which is zero) cannot be attained. Therefore, 'inf' cannot me replaced by 'min'. Nevertheless, we call this a minimization problem.

The above example has only one Lorentz cone constraint, but non-trivial second order cone programs involve multiple Lorentz cone (second order cone) constraints. Similarly, linear programs involve multiple non-negativity constraints. In these cases, the cone $\mathcal{K}$ is a Cartesian product of so-called primitive symmetric cones, viz. Lorentz cones, or nonnegative real half-lines ($\Re_+$), respectively. The cone of positive semidefinite matrices is also a primitive symmetric cone.

A *mixed semidefinite and second order cone optimization problem* can be formulated as a standard cone linear program (1) with the following structure:

$$(P) \quad \begin{array}{ll} \text{minimize} & (c^l)^{\mathrm{T}} x^l + (c^q)^{\mathrm{T}} x^q + (c^s)^{\mathrm{T}} x^s \\ \text{such that} & A^l x^l + A^q x^q + A^s x^s = b \\ & x^l \in \Re_+^{\kappa(l)},\ x^q \in \mathcal{K}^q,\ x^s \in \mathcal{K}^s. \end{array}$$

In this formulation, $\kappa(l)$ denotes the number of nonnegative variables, $\mathcal{K}^q = \mathcal{K}^q_1 \times \cdots \times \mathcal{K}^q_{\kappa(q)}$ is a Cartesian product of $\kappa(q)$ Lorentz cones, and $\mathcal{K}^s = \mathcal{K}^s_1 \times \cdots \times \mathcal{K}^s_{\kappa(s)}$ is a Cartesian product of $\kappa(s)$ cones of positive semidefinite matrices. We let $\kappa = \kappa(l) + \kappa(q) + \kappa(s)$ denote the number of primitive symmetric cone constraints. We partition the x-variables into sub-vectors accordingly, as

$$x = \begin{bmatrix} x^l \\ x^q \\ x^s \end{bmatrix} = \begin{bmatrix} x[1] \\ x[2] \\ \vdots \\ x[\kappa] \end{bmatrix},\ x^q = \begin{bmatrix} x^q[1] \\ x^q[2] \\ \vdots \\ x^q[\kappa(q)] \end{bmatrix},\ x^s = \begin{bmatrix} x^s[1] \\ x^s[2] \\ \vdots \\ x^s[\kappa(s)] \end{bmatrix}.$$

A similar convention will be used for other vectors in $\Re^n$, such as c or the rows of A. We remark that a non-negativity constraint can also be seen as a 1×1 positive semi-definiteness constraint.

In principle, we can therefore remove A^l, c^l and x^l from the formulation of (P), and treat the non-negativity constraints as $\kappa(l)$ scalar blocks in x^s; indeed, non-negativity has been modeled like this in SP [78]. However, for computational reasons it is advantageous to treat non-negativity constraints separately. Observe also from (2), that the first component of each Lorentz cone is always nonnegative, i.e. $x_1^q[k] \geq 0,\ k = 1, \ldots, \kappa(q)$. We let $x_1^q \in \Re_+^{\kappa(q)}$ denote the vector of these first components. The vector of the remaining components is denoted $x_{2:}^q$, so that $(x_1^q, x_{2:}^q)$ partitions the components of x^q. We partition the columns of A^q into $(A_1^q, A_{2:}^q)$ accordingly. With this notation, we have

$$A^q x^q = A_1^q x_1^q + A_{2:}^q x_{2:}^q. \tag{3}$$

To illustrate part of the above notation, consider the problem

$$\inf \left\{ \text{tr } X \;\middle|\; x_{11} + x_{22} \geq 1 + x_{13},\ x_{13} \geq 0,\ X = \begin{bmatrix} x_{11} & x_{12} & x_{13} \\ x_{12} & x_{22} & x_{23} \\ x_{13} & x_{23} & x_{33} \end{bmatrix} \text{ is p.s.d.} \right\}.$$

We can model this in the form of (P) by introducing slack variables $x^l \in \Re_+^2$ and vectorizing X, as

$$x^l = \begin{bmatrix} x[1] \\ x[2] \end{bmatrix} = \begin{bmatrix} x_{11} + x_{22} - x_{13} - 1 \\ x_{13} \end{bmatrix},\ x^s = x^s[1] = x[3] = \text{vec}\left(\begin{bmatrix} x_{11} & x_{12} & x_{13} \\ x_{12} & x_{22} & x_{23} \\ x_{13} & x_{23} & x_{33} \end{bmatrix} \right),$$

and by defining the data

$$c = \begin{bmatrix} c[1] \\ c[2] \\ c[3] \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ \text{vec}(I) \end{bmatrix},\ A^l = \begin{bmatrix} -1 & 0 \\ 0 & -1 \end{bmatrix},$$

and

$$A^s = \begin{bmatrix} \text{vec}\left(\begin{bmatrix} 1 & 0 & -1/2 \\ 0 & 1 & 0 \\ -1/2 & 0 & 0 \end{bmatrix} \right), & \text{vec}\left(\begin{bmatrix} 0 & 0 & 1/2 \\ 0 & 0 & 0 \\ 1/2 & 0 & 0 \end{bmatrix} \right) \end{bmatrix}^{\text{T}}$$

with $\kappa(l) = 2$, $\kappa(q) = 0$ and $\kappa(s) = 1$. This example has three decision variables, viz. $x[1]$, $x[2]$ and $x[3]$. The first two decision variables are scalar, but $x[3]$ is multi-dimensional (a matrix variable). In a more elaborate example, there can be many of such multi-dimensional variables, each restricted to a Lorentz cone or a cone of positive semidefinite matrices. Without such variables, we would simply have a linear programming problem.

It is clear that notations in this setting can easily become quite cumbersome. Fortunately, we can discuss mixed semidefinite and second order cone programs in a unified way, using Jordan algebra operations. The use of Jordan algebra in the context of cone linear programming was advocated by Faybusovich [15, 17, 16, 18]; we give a short introduction to the the basic Jordan algebra operations in Section 3.

Associated with (1) is a dual problem, viz.

$$\sup\{b^{\text{T}} y \mid A^{\text{T}} y + z = c,\ z \in \mathcal{K}^*\}, \tag{4}$$

where $y \in \Re^m$ and $z \in \Re^n$ are the decision variables, and

$$\mathcal{K}^* := \{z \in \Re^n \mid z^{\mathrm{T}} x \geq 0 \text{ for all } x \in \mathcal{K}\} \tag{5}$$

is the dual cone to $\mathcal{K}$. For mixed semidefinite and second order cone optimization, the dual problem has the following structure:

$$(D) \quad \begin{array}{ll} \text{maximize} & b^{\mathrm{T}} y \\ \text{such that} & (A^l)^{\mathrm{T}} y + z^l = c^l \\ & (A^q)^{\mathrm{T}} y + z^q = c^q \\ & (A^s)^{\mathrm{T}} y + z^s = c^s \\ & z^l \in \Re_+^{\kappa(l)},\ z^q \in \mathcal{K}^q,\ z^s \in (\mathcal{K}^s)^*. \end{array}$$

Here, we have $(\mathcal{K}^s)^* = (\mathcal{K}_1^s)^* \times \cdots \times (\mathcal{K}_{\kappa(s)}^s)^*$. Furthermore, $(\mathcal{K}^s)^* = \mathcal{K}^s$ if we restrict z in definition (5) to a proper lower dimensional Euclidean space to take care of symmetry, viz.

$$\mathcal{K}_i^s = \{z = \text{ vec}\,(Z) \mid Z = Z^{\mathrm{T}},\ z^{\mathrm{T}} x \geq 0 \text{ for all } x \in \mathcal{K}_i^s\}.$$

This requires that c and the rows of A are also in this lower dimensional Euclidean space. Stated differently, the $c^s[k]$ and $a_i^s[k]$ blocks must be vectorizations of symmetric matrices. For convenience, we assume without loss of generality that this is indeed the case, so that $\mathcal{K}$ is self-dual.

3 The Jordan Algebra Operations

We summarize in this section some properties of symmetric cones that are relevant for this paper. Concise overviews on this subject in the context of optimization were also given by Tunçel [77], Güler and Tunçel [29], Faybusovich [15, 17, 16, 18], and Sturm [69]. A detailed treatment of symmetric cones can be found in [14].

An important quantity is the *order of a symmetric cone*, denoted by $\nu(\mathcal{K})$. Any Lorentz cone has order $\nu(\mathcal{K}_i^q) = 2$. For a Cartesian product of symmetric cones $\mathcal{K}_1$ and $\mathcal{K}_2$, it holds that

$$\nu(\mathcal{K}_1 \times \mathcal{K}_2) = \nu(\mathcal{K}_1) + \nu(\mathcal{K}_2), \tag{6}$$

so that in particular $\nu(\mathcal{K}^q) = 2\kappa(q)$. The order of the cone of $\nu \times \nu$ positive semi-definite matrices is ν. In particular, we have $\nu(\Re_+) = 1$ and $\nu(\mathcal{K}^s) = \sum_{i=1}^{\kappa(s)} \nu_i(s)$. Since a symmetric cone $\mathcal{K}$ can be decomposed as the Cartesian product of primitives (Proposition III.4.5 in [14]), the above rules suffice to compute the order $\nu(\mathcal{K})$ of an arbitrary symmetric cone $\mathcal{K}$. The worst case iteration bound of interior point algorithms depends on $\nu(\mathcal{K})$, which is the barrier parameter in Nesterov and Todd [57]. The worst case iteration bound for SeDuMi is a multiple of $\sqrt{\nu(\mathcal{K})}$, see [66].

Associated with a primitive symmetric cone $\mathcal{K} \subset \Re^n$ of order $\nu = \nu(\mathcal{K})$ is a set of *Jordan frames*, which are also known as complete systems of idempotents. A Jordan frame is a matrix $F \in \Re^{n \times \nu}$, such that

$$F^{\mathrm{T}} F = I, \quad F\Re_+^n = \text{ Img}\,(F) \cap \mathcal{K}, \tag{7}$$

i.e. the columns $f_1, f_2, \ldots, f_\nu$ of F form an orthonormal basis of an ν–dimensional linear subspace of $\Re^n$, viz. $\text{Img}\,(F) = \{Fy | y \in \Re^\nu\}$, and the nonnegative orthant on this subspace, i.e. $F\Re_+^\nu =$

$\{Fy \mid y \geq 0\}$ is precisely the intersection of this subspace with the cone $\mathcal{K}$. For instance, for the cone of positive semi-definite matrices, the usual basis of the subspace of diagonal matrices, i.e.

$$\text{vec}\,(e_1 e_1^{\mathrm{T}}),\ \text{vec}\,(e_2 e_2^{\mathrm{T}}), \ldots,\ \text{vec}\,(e_\nu e_\nu^{\mathrm{T}}),$$

is a Jordan frame.

Any vector $x \in \Re^n$ has a *spectral decomposition* $x = F(x)\lambda(x)$ with respect to $\mathcal{K}$, where $F(x)$ is a Jordan frame. The vector $\lambda(x) \in \Re^\nu$ is unique up to permutations. Thus,

$$x \in \mathcal{K} \iff \lambda(x) \geq 0. \tag{8}$$

The components $\lambda_1(x), \ldots, \lambda_\nu(x)$ of the vector $\lambda(x)$ are called the spectral values of x. If $\lambda(x)$ is the all–one vector e, then x is the identity solution, denoted by ι. The identity solution is the same for all Jordan frames associated with $\mathcal{K}$, i.e.

$$Fe = \iota \text{ for any Jordan frame } F. \tag{9}$$

For the Lorentz cone we have $\iota = \sqrt{2}e_1$, where e_1 denotes the first column of the identity matrix I. For the positive semidefinite cone we have $\iota = \text{vec}\,(I)$. The spectral decomposition associated with the Lorentz cone decomposes a vector $x = (x_1, x_{2:}) \in \Re \times \Re^{n-1}$ as $x = F\lambda$ with

$$F = \frac{\sqrt{2}}{2\|x_{2:}\|} \begin{bmatrix} \|x_{2:}\| & \|x_{2:}\| \\ -x_{2:} & x_{2:} \end{bmatrix}, \quad \lambda = \frac{\sqrt{2}}{2} \begin{bmatrix} x_1 - \|x_{2:}\| \\ x_1 + \|x_{2:}\| \end{bmatrix}.$$

The spectral decomposition associated with the cone of positive semidefinite matrices corresponds to the usual symmetric eigenvalue decomposition.

The concepts of spectral decomposition and identity solution extend to non-primitive symmetric cones by means of direct summation. Namely, we let

$$F(x) = F(x[1]) \oplus F(x[2]) \oplus \cdots \oplus F(x[\kappa]), \quad \lambda(x) = \begin{bmatrix} \lambda(x[1]) \\ \vdots \\ \lambda(x[\kappa]) \end{bmatrix}.$$

As usual, $M_1 \oplus M_2$ denotes the direct sum of two matrices M_1 and M_2:

$$M_1 \oplus M_2 = \begin{bmatrix} M_1 & 0 \\ 0 & M_2 \end{bmatrix}.$$

Given the spectral decomposition $x = F(x)\lambda(x)$, the trace and determinant are defined as $\text{tr}\ x = \sum_{i=1}^{\nu} \lambda_i(x)$, $\det(x) = \prod_{i=1}^{\nu} \lambda_i(x)$.

Associated with $x \in \Re^n$ are $n \times n$ matrices $L(x)$ and $P(x)$, which are known as the *Jordan product representation* and the *quadratic representation*, respectively. For the Lorentz cone,

$$J := \iota\iota^{\mathrm{T}} - I, \quad L(x) = (\iota x^{\mathrm{T}} + x\iota^{\mathrm{T}} - (\text{ tr } x)J)/2, \quad P(x) = xx^{\mathrm{T}} - \det(x)J;$$

due to its nonzero pattern, $L(x)$ is also known as the *arrow matrix* in second order cone programming [26]. For the cone of positive semidefinite matrices,

$$L(\text{ vec}\,(X)) = (I \otimes X + X \otimes I)/2, \quad P(\text{ vec}\,(X)) = X \otimes X,$$

where '$\otimes$' denotes the standard Kronecker product [31]. The definitions of $L(x)$ and $P(x)$ extend to non-primitive symmetric cones by means of a direct sum. Namely, we let

$$L(x) = L(x[1]) \oplus L(x[2]) \oplus \cdots \oplus L(x[\kappa]), \quad P(x) = P(x[1]) \oplus P(x[2]) \oplus \cdots \oplus P(x[\kappa]).$$

Since $\Re_+ = \mathrm{PSD}(1)$, we have for linear programming that $L(x)$ and $P(x)$ are the diagonal matrices with entries $x_1, x_2, \ldots, x_n$ and $x_1^2, x_2^2, \ldots, x_n^2$ on their respective diagonals. (In the linear programming literature, one usually denotes $L(x)$ and $P(x)$ by X and X^2.)

4 Primal-Dual Interior Point Method

Consider the primal-dual pair (P), (D) of mixed semidefinite and second order cone programming. We discuss the *feasible* interior point method for solving (P) and (D); the feasible method applies to possibly infeasible problems using the technique of self-dual embedding, as discussed in Section 12. A main iteration of the interior point method consists of computing a search direction that is added to the current feasible iterative solution (x, y, z) with a certain step length $t > 0$, yielding the next feasible solution (x^+, y^+, z^+), i.e.

$$(x^+, y^+, z^+) = (x, y, z) + t(\Delta x, \Delta y, \Delta z). \tag{10}$$

A special class of interior point methods are path-following methods. A first-order path-following direction solves the linearized central path equations. The central path can be defined as

$$\text{central path } = \{(x(\mu), y(\mu), z(\mu)) \in \mathcal{F} | \lambda(P(x(\mu)^{1/2}) z(\mu)) = \mu e,\ \mu > 0\}, \tag{11}$$

where

$$\mathcal{F} = \{(x, y, z) \in \mathcal{K} \times \Re^m \times \mathcal{K} | Ax = b,\ A^{\mathrm{T}} y + z = c\}$$

is the feasible solution set and e is the all-one vector in $\Re^{\nu(\mathcal{K})}$. The central path exists only if both the primal and the dual programs have interior feasible solutions. In Section 12 we discuss techniques to overcome this problem. There are other ways to characterize the central path, such as

$$\text{central path } = \{(x(\mu), y(\mu), z(\mu)) \in \mathcal{F} | L(x(\mu)) z(\mu) = \mu \iota,\ \mu > 0\}, \tag{12}$$

or

$$\text{central path } = \{(x(\mu), y(\mu), z(\mu)) \in \mathcal{F} | z(\mu) = \mu x(\mu)^{-1},\ \mu > 0\}, \tag{13}$$

among others. Different characterizations have different linearizations and hence lead to different search directions. Furthermore, linearization of (11) leads to an under-determined system.

Generally speaking, the search direction $(\Delta x, \Delta y, \Delta z)$ is implicitly defined by a system of equations, as follows:

$$\begin{aligned} \Delta x + \Pi \Delta z &= r & (14)\\ A \Delta x &= 0 & (15)\\ A^{\mathrm{T}} \Delta y + \Delta z &= 0. & (16) \end{aligned}$$

The system depends on an invertible $n \times n$ block diagonal matrix 'Π' and a vector $r \in \Re^n$, which depend not only on the iterate, but also on the specific algorithmic choices of the interior point method. E.g. setting $r = -x$ corresponds to the so-called *predictor* (or: primal-dual *affine scaling*) direction. In all usual approaches, including AHO [4], HKM [30] and NT [57], the invertible matrix Π is such that

$$\Pi^{\mathrm{T}} z = x. \tag{17}$$

The diagonal blocks $\Pi^s[k]$, $k = 1, 2, \ldots, \kappa(s)$, should also map the $\nu(\mathcal{K}_i^s) \times (\nu(\mathcal{K}_i^s)+1)/2$ Euclidean space of symmetric matrices onto itself, in order to maintain symmetry of the primal matrix variables $x^s[k]$. (In the literature on semidefinite programming where $\mathcal{K} = \mathcal{K}^s$, $\kappa = 1$, $x = \mathrm{vec}\,(X)$, $z = \mathrm{vec}\,(Z)$, several researchers have considered $\Pi = Z^{-1} \otimes X$ which does not satisfy this symmetry preservation rule [30, 37, 38].) The choices of Π that we consider in Section 5 also satisfy the relations

$$\Pi z = \Pi^{\mathrm{T}} z, \quad \Pi x^{-1} = z^{-1}. \tag{18}$$

Interestingly, the rate of change in the duality gap along the direction $(\Delta x, \Delta y, \Delta z)$ does not depend on the specific choice of Π, as long as this choice obeys (17). Namely, pre-multiplying (14) with z^{T} yields

$$z^{\mathrm{T}} r = z^{\mathrm{T}} \Delta x + z^{\mathrm{T}} \Pi \Delta z \stackrel{(17)}{=} z^{\mathrm{T}} \Delta x + x^{\mathrm{T}} \Delta z. \tag{19}$$

Since $(\Delta z)^{\mathrm{T}} \Delta x = -(\Delta y)^{\mathrm{T}} A \Delta x = 0$, see (15)–(16), we have from (10) that

$$(x^+)^{\mathrm{T}} z^+ = x^{\mathrm{T}} z + t(z^{\mathrm{T}} \Delta x + x^{\mathrm{T}} \Delta z) \stackrel{(19)}{=} x^{\mathrm{T}} z + t(z^{\mathrm{T}} r). \tag{20}$$

In particular, the predictor choice '$r = -x$' yields that $(x^+)^{\mathrm{T}} z^+ = (1-t) x^{\mathrm{T}} z$. A well studied choice for r is the first order path-following direction $r = \sigma z^{-1} - x$, for which $(x^+)^{\mathrm{T}} z^+ = x^{\mathrm{T}} z + t(\nu\sigma - x^{\mathrm{T}} z)$, yielding a descent direction for $\sigma < x^{\mathrm{T}} z / \nu$. We remark that if Π satisfies (18) then

$$\Delta x + \Pi \Delta z = \sigma z^{-1} - x \iff \Pi^{-1} \Delta x + \Delta z = \sigma x^{-1} - z,$$

revealing the primal-dual symmetry of the search direction.

Instead of dealing with (14)-(16) directly, one often concentrates on the so-called reduced system (21). The reduced system is derived by pre-multiplying (16) with $A\Pi$ yielding

$$0 = A\Pi(A^{\mathrm{T}} \Delta y + \Delta z) \stackrel{(14)}{=} A\Pi A^{\mathrm{T}} \Delta y + A(r - \Delta x) \stackrel{(15)}{=} A\Pi A^{\mathrm{T}} \Delta y + Ar.$$

Re-arranging terms, we have

$$A\Pi A^{\mathrm{T}}\, \Delta y = -Ar. \tag{21}$$

Notice that for the predictor direction with $r = -x$ that $-Ar = b$, and (21) can be further simplified to

$$A\Pi A^{\mathrm{T}}\, \Delta y = b. \tag{22}$$

Once the Δy direction is known, the Δz and Δx directions then follow easily as

$$\Delta z = -A^{\mathrm{T}} \Delta y, \quad \Delta x = r - \Pi \Delta z. \tag{23}$$

The main computational effort in the interior point method lies therefore in solving a system of the form (21) or (22).

The algorithm below outlines the now well-known basic scheme of a primal-dual interior point method. Due to the notation from Jordan algebra, the analogy to the linear programming case is clear. The algorithm determines its step length based on a neighborhood $\mathcal{N} \subset \Re^{\nu}_{+}$.

Algorithm 1 (Interior Point Method)

Step 0 Initial solution $(x, y, z) \in \mathcal{K} \times \Re^m \times \mathcal{K}$ *with* $Ax = b$ *and* $A^T y + z = c$ *such that* $\lambda(P(x)^{1/2} z) \in \mathcal{N}$.

Step 1 If $x^T z \leq \epsilon$ *then STOP.*

Step 2 Choose Π *and* r *according to the algorithmic settings. Compute the search direction* $(\Delta x, \Delta y, \Delta z)$ *from (14)–(16). Then determine a 'large enough' step length* $t > 0$ *such that* $\lambda(P(x + t\Delta x)^{1/2}(z + t\Delta z)) \in \mathcal{N}$.

Step 3 Update

$$(x, y, z) \leftarrow (x + t\Delta x, y + t\Delta y, z + t\Delta z),$$

and return to Step 1.

We remark that $\lambda(P(x)^{1/2} z) = \lambda(P(z)^{1/2} x)$ by similarity, see Sturm [69]. The above algorithm is thus symmetric in duality.

The choice of $\mathcal{N}$ in Algorithm 1 is important, since it determines the step length strategy. A well studied neighborhood is the $\mathcal{N}_2$-neighborhood, defined as

$$\mathcal{N}_2(\beta) := \left\{ w \in \Re^{\nu} \,\middle|\, \|w - \mu e\|_2 \leq \beta\mu,\ \mu = \sum_{i=1}^{\nu} w_i/\nu \right\},$$

which is based on the ℓ_2-norm. Notice that for large ν, this neighborhood is a rather narrow cone in $\Re^n_+$ around the central path, resulting in a conservative step length. The $\mathcal{N}_\infty^-$-neighborhood, defined as

$$\mathcal{N}_\infty^-(\theta) := \left\{ w \in \Re^{\nu} \,\middle|\, w_j \geq \theta\mu \text{ for all } j = 1, \ldots, \nu,\ \mu = \sum_{i=1}^{\nu} w_i/\nu,\ \mu \geq 0 \right\},$$

is therefore considered more suitable for practical purposes. The individual benefits of these two neighborhoods are combined in the wide region neighborhood $\mathcal{N}^{\mathrm{wr}}$,

$$\mathcal{N}^{\mathrm{wr}}(\theta, \beta) := \left\{ w \in \Re^{\nu}_{+} \,\middle|\, \mathrm{dist}(w, \mathcal{N}_\infty^-(\theta)) \leq \beta\theta\mu,\ \mu = \sum_{i=1}^{\nu} w_i/\nu \right\}.$$

The distance function has to be in accordance with the centering component of the search direction. Sturm and Zhang [71] used

$$\mathrm{dist}(w, \mathcal{N}_\infty^-(\theta)) = \min \left\{ \sum_{i=1}^{\nu} (\sqrt{w_i} - \sqrt{\hat{w}_i})^2 \,\middle|\, \hat{w} \in \mathcal{N}_\infty^-(\theta) \right\}.$$

The wide region neighborhood is used in SeDuMi, and discussed in [71, 66].

Another important algorithmic choice is the definition of Π and r in (14)–(15), since this determines the actual search direction. We discuss this in Section 5 and Section 11, respectively.

Finally, we need to quantify what we mean by 'large enough' in Step 2 of Algorithm 1. In SeDuMi, a simple bisection search is used to approximate the step length t^* to the boundary of $\mathcal{N}^{\mathrm{wr}}$. Assuming that $z^{\mathrm{T}}r = -x^{\mathrm{T}}z$, the bisection procedure terminates with a t satisfying

$$\frac{t^*}{2} \leq t \leq t^*, \quad 1 - t \leq 2(1 - t^*).$$

The former condition is used to guarantee global polynomial convergence, whereas the latter condition allows for superlinear convergence. Superlinear convergence cannot always be achieved.

5 The Scaling Operation

One of the most fascinating issues in semidefinite and second order cone programming concerns the method of scaling. We review the first introduced and still the most popular primal-dual scaling techniques, namely AHO, NT and HKM scaling. We discuss the main benefits of these different scaling techniques. The issue of computing and updating the scaling operation is the subject of Section 10.

In order to be more specific, we start with a brief definition of the three scaling operators that we discuss.

The *AHO-scaling* technique was introduced for semidefinite programming by Alizadeh, Haeberly and Overton [4, 5]. Its generalization to the symmetric cone setting is straightforward, namely

$$\Pi = L(z)^{-1}L(x). \tag{24}$$

The *HKM-scaling* was introduced for semidefinite programming independently by Helmberg et al. [30] and Kojima et al. [37], and derived differently later by Monteiro [50]. The scaling in its primal form can be generalized to the symmetric cone setting as

$$\Pi = P(z)^{-1/2}L(P(z)^{1/2}x)P(z)^{-1/2}, \tag{25}$$

which reduces to $\Pi = (X \otimes Z^{-1} + Z^{-1} \otimes X)/2$ for the semidefinite programming (where $\mathcal{K} = \mathcal{K}^s$, $\kappa = 1$ and $x = \mathrm{vec}\,(X)$, $z = \mathrm{vec}\,(Z)$). In an analogous way, one defines the dual variant of HKM scaling as

$$\Pi = P(x)^{1/2}L(P(x)^{1/2}z)^{-1}P(x)^{1/2}, \tag{26}$$

which reduces to $\Pi^{-1} = (Z \otimes X^{-1} + X^{-1} \otimes Z)/2$ for semidefinite programming.

The *NT-scaling* was introduced for semidefinite programming by Nesterov and Todd [57, 58], and derived differently later by Sturm and Zhang [72]. Its generalization to the symmetric cone setting is straightforward, namely

$$\Pi = P(z)^{-1/2}P(P(z)^{1/2}x)^{1/2}P(z)^{-1/2}, \tag{27}$$

or equivalently

$$\Pi = P(d),$$

where $d \in \mathcal{K}$ is implicitly defined by the equation '$P(d)z = x$', see (17). An explicit formula for d is

$$d = P(x^{1/2})(P(x^{1/2})z)^{-1/2}. \tag{28}$$

It is easily verified that $\Pi^{\mathrm{T}} z = x$ for all these choices of Π. One can also show that (18) is satisfied. There are many other possible choices of Π satisfying these relationships, see e.g. Todd. However, such choices are not known to have substantial benefits over the 'classical' choices (AHO, HKM and NT).

In the case of AHO-scaling, Π is invertible but non-symmetric, so that the rank of $A\Pi A^{\mathrm{T}}$ can be smaller than the rank of A. This means that the AHO-direction is not well defined for general $x, z \in$ int $\mathcal{K}$. However, with a suitable choice of the neighborhood $\mathcal{N}$, the AHO-direction is always well defined in the interior point algorithm [64, 53]. The predictor direction with AHO scaling is the tangent direction of an analytic curve to the optimal solution set [62], allowing for quadratic convergence under strict complementarity conditions [36]. In Section 11 we give two equivalent characterizations of this curve. The AHO scaling has been implemented as default in SDPPack [3] and as an option in SDPA [20] and SDPT3 [76]. An important practical drawback of the AHO scaling is that it does not allow for sparsity exploiting techniques for building the normal equations, such as discussed in Section 6.1 for NT-scaling.

The main advantage of HKM scaling is its simplicity for semidefinite programming. Unlike AHO and NT scaling, this scaling operator can be computed by merely basic linear algebra operations. In particular, no symmetric eigenvalue decomposition is needed. However, it is not symmetric in duality, leading to two versions of this scaling: a primal version and a dual version. The HKM scaling has been implemented in SDPA [20], CSDP [10] and SDPT3 [76].

An appealing property of NT scaling is its scale invariance. In particular, it holds that $\Pi\mathcal{K} = \Pi^k\mathcal{K} = \mathcal{K}$ for any power $k \in \Re$. This means among others that we may locally scale the data (A, b, c) to $(A\Pi^{1/2}, b, \Pi^{1/2}c)$ so that the current primal solution x and dual solution z are mapped onto the same vector $v = \Pi^{1/2}z = \Pi^{-1/2}x$. The cone $\mathcal{K}$ remains invariant under this transformation. This makes it possible to extend the primal-dual framework of Kojima et al. [34] from linear programming to semidefinite or symmetric cone programming, see [72] and also Sections 10 and 11 later in this paper. The NT scaling has been implemented as default scaling in SeDuMi [67] and MOSEK [7, 8], and as an option in SDPA [20] and SDPT3 [76].

6 Building the Normal Equations

There are several approaches to solving a system of the form '$A\Pi A^{\mathrm{T}}\Delta y = b$' as in (22) when Π is symmetric positive definite. Given a factor Φ such that $\Pi = \Phi\Phi^{\mathrm{T}}$, one can solve the system based on a QR-factorization of the matrix $\Phi^{\mathrm{T}}A^{\mathrm{T}}$. For numerical reasons, this is the preferred method for small to medium sized normal equations [12], and it was considered in the context of semidefinite programming by [75]. However, for most practical cone linear optimization problems, A is a large sparse matrix with considerably more columns than rows. (A matrix is said to be *sparse*

if a considerable number of its entries are zero; the zero entries are not stored in memory, and are skipped during additions and multiplications.) In this situation, it is impractical even to build the matrix $\Phi^{\mathrm{T}}A^{\mathrm{T}}$, especially in terms of memory requirements. However, the matrix Φ is diagonal for linear programs, resulting in a matrix $\Phi^{\mathrm{T}}A^{\mathrm{T}}$ that has the same sparsity structure as A^{T}. In this case, one may indeed efficiently apply sparse QR-factorization techniques to $\Phi^{\mathrm{T}}A^{\mathrm{T}}$, producing only the R matrix. According to Saunders [63], this approach is nevertheless outperformed in terms of computational time by the Cholesky approach, which we discuss below.

The most efficient approach in terms of memory is the conjugate gradient method, which was already used in Karmarkar's implementation of the interior point method [32]. However, experimental results indicate that this approach requires specific tailoring of a pre-conditioner, which limits its use for a general purpose solver. Nevertheless, effective preconditioners for special classes exist [61].

The usual approach in general purpose interior point codes [43, 6] is very straightforward: build the matrix $A\Pi A^{\mathrm{T}}$ and compute its Cholesky factorization. In this section, we address the issue of computing the matrix $A\Pi A^{\mathrm{T}}$ in the case of NT-scaling, where $\Pi = P(d)$. A similar approach is possible for HKM-scaling; see Fujisawa et al. [21]. The case of AHO-scaling is quite different, see [75] for some details.

6.1 Exploiting Sparsity

On the ith column and jth row in the matrix $A\Pi A^{\mathrm{T}}$ is the entry $a_i^{\mathrm{T}}\Pi a_j$. Due to the block diagonal structure of Π, we see that this entry is zero whenever the constraints '$a_i^{\mathrm{T}}x = b_i$' and '$a_j^{\mathrm{T}}x = b_j$' do not have any decision variable in common, i.e.

$$\sum_{k=1}^{\kappa} \|a_i[k]\|\|a_j[k]\| = 0 \Longrightarrow a_i^{\mathrm{T}}\Pi a_j = 0. \tag{29}$$

If there is only one decision variable, i.e. if $\kappa = 1$, then $A\Pi A^{\mathrm{T}}$ is dense; this case is treated in Fujisawa et al. [21]. For models where $\kappa > 1$, each individual constraint typically involves only a small number of decision variables, so that $A\Pi A^{\mathrm{T}}$ is likely to be sparse.

Consider now the case of NT scaling. Thus, $\Pi = P(d)$, where d is uniquely defined by the relation $x = P(d)z$. We have

$$AP(d)A^{\mathrm{T}} = A^l P(d^l)(A^l)^{\mathrm{T}} + A^q P(d^q)(A^q)^{\mathrm{T}} + A^s P(d^s)(A^s)^{\mathrm{T}}.$$

The $\kappa(l) \times \kappa(l)$ matrix $P(d^l)$ is a diagonal matrix, viz.

$$P(d^l) = P(d^l[1]) \oplus \cdots \oplus P(d^l[\kappa(l)]) = d_1^l \oplus \cdots \oplus d_{\kappa(l)}^l.$$

Furthermore, it follows from (17) that

$$P(d_i^l) = (d_i^l)^2 = \frac{x_i^l}{z_i^l} \text{ for } i = 1, 2, \ldots, \kappa(l).$$

(All scalings obeying (17) are identical for the cone $\Re_+$.) In the linear programming literature, one often writes $\Pi = XZ^{-1}$, where X and Z are diagonal matrices with x and z on their diagonals, respectively. The efficient computation of $A^l P(d^l)(A^l)^{\mathrm{T}}$ with sparse A^l is rather straightforward. We refer to [43, 6] for implementational issues in the setting of linear programming.

The matrix $P(d^q)$ consists of $\kappa(q)$ diagonal blocks $P(d^q[k])$, $k = 1, \ldots, \kappa(q)$. Each diagonal block is the sum of a diagonal matrix and a rank-1 matrix, viz.

$$P(d^q[k]) = d^q[k](d^q[k])^{\mathrm{T}} - \det(d^q[k])J.$$

For each Lorentz cone $\mathcal{K}_k^q \subset \Re^{n[k]}$, $k = 1, 2, \ldots, \kappa(q)$, we define

$$H[k] = \det(d^q[k])I \in \Re^{(n[k]-1)\times(n[k]-1)} \tag{30}$$

and

$$\begin{cases} D^q = d^q[1] \oplus d^q[2] \oplus \cdots \oplus d^q[\kappa(q)] \\ \mathrm{Det}(d^q) = \det(d^q[1]) \oplus \det(d^q[2]) \oplus \cdots \oplus \det(d^q[\kappa(q)]) \\ H = H[1] \oplus H[2] \oplus \cdots \oplus H[\kappa(q)]. \end{cases} \tag{31}$$

Partitioning the columns of A^q as in (3), we have

$$A^q P(d^q)(A^q)^{\mathrm{T}} = (A^q D^q)(A^q D^q)^{\mathrm{T}} + A^q_{2.} H (A^q_{2.})^{\mathrm{T}} - A^q_1 \mathrm{Det}(d^q)(A^q_1)^{\mathrm{T}}. \tag{32}$$

Since H and $\mathrm{Det}(d^q)$ are diagonal matrices, the construction is completely analogous to the linear programming case, except for the low rank update $(A^q D^q)(A^q D^q)^{\mathrm{T}}$.We have

$$\mathtt{nnz}(A^q D^q) \leq \mathtt{nnz}(A^q),$$

where '`nnz`' stands for *number of nonzeros*. However, the density of nonzeros in $A^q D^q$ can be much higher than in A^q, since $A^q D^q$ has merely $\kappa(q)$ columns. We will see in Section 8 that it can be wise to omit some columns of $A^q D^q$ at this stage, in order to promote sparsity in the remaining part of $A\Pi A^{\mathrm{T}}$.

The problem of efficiently computing $A^s P(d^s)(A^s)^{\mathrm{T}}$ was first addressed by Fujisawa et al. [21]. A different approach was implemented in SeDuMi [67], which we will describe here.

Since $A^s P(d^s)(A^s)^{\mathrm{T}}$ is symmetric, it suffices to compute for each $j = 1, \ldots, m$ the quantities

$$(a_i^s)^{\mathrm{T}} P(d^s) a_j^s \text{ for } i = 1, 2, \ldots, j.$$

For given j, we will compute only those entries in the vector $P(d^s)a_j^s$ that are needed to evaluate the above j quantities. In particular, if l is such that $a_{il}^s = 0$ for all $i = 1, 2, \ldots, j$, then the lth entry in $P(d^s)a_j^s$ is not evaluated. Thus, for each semidefinite block $k = 1, \ldots, \kappa(s)$, we only need to evaluate certain entries in

$$P(d^s[k])a_j^s[k] = \mathrm{vec}\,(D^s[k]A_j^s[k]D^s[k]).$$

(The matrices $D_s[k]$ and $A_j^s[k]$ are implicitly defined by the relations $d^s[k] = \mathrm{vec}\,(D_s[k])$ and $a_j^s[k] = \mathrm{vec}\,(A_j^s[k])$.) Since the matrix $D^s[k]A_j^s[k]D^s[k]$ is symmetric, we can reduce the list of entries to be computed in $P(d^s)a_j^s$ even further. Moreover, we order (permute) the constraints $a_1, \cdots, a_m$ according to the following principle:

1. a_1^s is as sparse as possible, i.e. $\texttt{nnz}(a_1^s) = \min\{\texttt{nnz}(a_i^s) \mid i = 1, 2, \ldots, m\}$.

2. For each $i > 1$, there cannot exist $j > i$ such that the number of all-zero rows in the matrix $\left[\begin{array}{cc} A_{1:i-1}^s, & a_j^s \end{array}\right]$ exceeds the number of all-zero rows in the matrix $A_{1:i}^s$, where

$$A_{1:i}^s := \left[\begin{array}{cccc} a_1^s, & a_2^s, & \ldots, & a_i^s \end{array}\right].$$

   (Otherwise, we exchange the constraints a_i and a_j.)

The advantage of this ordering is that the number of entries to be computed in $P(d^s)(A^s)^{\mathrm{T}}$ is as small as possible (given the restriction that the ordering is the same for all semidefinite blocks). We remark that Fujisawa et al. [21] order the restrictions such that $\texttt{nnz}(a_1) \geq \texttt{nnz}(a_2) \geq \cdots \geq \texttt{nnz}(a_m)$.

Consider the kth semidefinite block in the jth constraint, $A_j^s[k]$. To simplify notations, let $\hat{A}$ and $\hat{D}$ denote $A_j^s[k]$ and $D^s[k]$, respectively. We consider the problem of evaluating a limited number of entries in the matrix $\hat{D}\hat{A}\hat{D}$, where $\hat{A}$ is sparse, and $\hat{D}$ is dense. We let T be an upper triangular matrix such that $T + T^{\mathrm{T}} = \hat{A}$. We obviously have $\texttt{nnz}(T) \leq \texttt{nnz}(A)$. Let $\tilde{m}$ denote the number of columns in T that contain nonzeros, and let $\tilde{T}$ denote the $\nu_k^s \times \tilde{m}$ sparse matrix consisting of those columns in T that have nonzeros. We select the same set of columns (i.e. where T has nonzeros) from $\hat{D}$, yielding the $\nu_k^s \times \tilde{m}$ dense matrix $\tilde{D}$. Observe that

$$\hat{D}\hat{A}\hat{D} = (\hat{D}\tilde{T})\tilde{D}^{\mathrm{T}} + \tilde{D}(\hat{D}\tilde{T})^{\mathrm{T}}. \tag{33}$$

Thus, we first use sparse computations to obtain the $\nu_k^s \times \tilde{m}$ dense matrix $\hat{D}\tilde{T}$. Given this matrix, each desired off-diagonal (diagonal) entry of $\hat{D}\hat{A}\hat{D}$ can be computed using merely two (one) dot-product(s) of length $\tilde{m}$, as can be seen from (33). In the models that we encountered, $\tilde{m}$ is typically very small.

7 Solving the Normal Equations

After building the matrix $A\Pi A^{\mathrm{T}}$ as discussed above, it remains to solve Δy from the normal equations $A\Pi A^{\mathrm{T}}\Delta y = -Ar$ or more specifically $A\Pi A^{\mathrm{T}}\Delta y = b$; see (21) and (22). We assume that Π is positive definite, as is the case with HKM and NT scaling. The basic approach is to compute the $L\Theta L^{\mathrm{T}}$ Cholesky factorization of $A\Pi A^{\mathrm{T}}$. In other words, we compute a lower triangular matrix L with all-1 diagonal entries $l_{ii} = 1$, $i = 1, 2, \ldots, m$, and a positive (semi-)definite diagonal matrix Θ such that $A\Pi A^{\mathrm{T}} = L\Theta L^{\mathrm{T}}$. In the initialization phase of the interior point method, the rows of the A-matrix are ordered using the minimum degree heuristic, as implemented in SPARSPAK-A [22]; the minimum degree ordering promotes sparsity in the lower triangular factor L. The ordering is determined only once, but a new numerical factorization is computed in each main iteration of the interior point method. The numerical block sparse Cholesky factorization combines sparse and dense techniques based on the super-nodal structure. This is the usual approach in the interior point method for linear programming [6]. Some primal-dual interior point based solvers for semidefinite programming, including CSDP [10] and SDPA [20], do not exploit sparsity in factorizing the normal equations. The reason is that semidefinite programming models arising

in combinatorial optimization typically reference the same matrix variable in all rows of A, thus leading to a completely dense matrix $A\Pi A^{\mathrm{T}}$.

After the Cholesky factors L and Θ have been computed, one obtains $L^{-1}b$, $\Theta^{-1}(L^{-1}b)$ and $\Delta y = L^{-\mathrm{T}}(\Theta^{-1}L^{-1}b)$ by a forward solve, element-wise division, and backward solve, respectively. (This does assume that A has full row rank, so that Θ^{-1} exists.)

The forward and backward solve procedures may lead to serious loss of numerical accuracy if $\max_{i,j}|l_{ij}|$ is large, say more than 1000. Causes of large entries in L are poor scaling of the problem, and, more importantly, ill-conditioning in $A\Pi A^{\mathrm{T}}$ when the optimum is approached.

In principle, it is always possible to order the rows in A such that $\max_{i,j}|l_{ij}| \leq 1$, but such an ordering is not practical since it ignores the sparsity structure of $A\Pi A^{\mathrm{T}}$. A compromise is to achieve $\max_{i,j}|l_{ij}| \leq \texttt{maxl}$ where $\texttt{maxl} \geq 1$ is a user defined parameter; such an L-factor can be obtained by delaying pivots from the minimum degree ordering when necessary. However, this approach leads to excessive memory requirements when the number of delayed pivots increases. A pivot delaying technique was implemented in SeDuMi 1.04, but it has been removed in later versions.

Instead of *delaying* pivots, one may also obtain an L matrix with $\max_{i,j}|l_{ij}| \leq \texttt{maxl}$ by *skipping* pivots as in S.J. Wright [83], or by *adding* a low rank diagonal matrix to $A\Pi A^{\mathrm{T}}$ as in Gill, Murray and M.H. Wright [23] and Algorithm A5.5.2 in Dennis and Schnabel [12]. Both approaches avoid fill-in of nonzeros into the L-matrix.

Let $L\Theta L^{\mathrm{T}}$ denote the modified Cholesky factorization that is obtained by either skipping pivots or by adding quantities on a few diagonal entries. In the former case, $L\Theta L^{\mathrm{T}}$ differs from $A\Pi A^{\mathrm{T}}$ only in the rows and columns that correspond to the skipped pivots, and hence $\mathrm{rank}(A\Pi A^{\mathrm{T}} - L\Theta L^{\mathrm{T}}) \leq 2 \times \texttt{nskip}$, where `nskip` denotes the number of skipped pivots. In the latter case, where we add quantities to `nadd` diagonal entries, we have that $A\Pi A^{\mathrm{T}} - L\Theta L^{\mathrm{T}}$ is a diagonal matrix of rank `nadd`. Observe in any case that the matrix

$$(L\Theta^{1/2})^{-1}(A\Pi A^{\mathrm{T}})(L\Theta^{1/2})^{-\mathrm{T}} = I + (L\Theta^{1/2})^{-1}(A\Pi A^{\mathrm{T}} - L\Theta L^{\mathrm{T}})(L\Theta^{1/2})^{-\mathrm{T}}$$

has at most $1 + \mathrm{rank}(A\Pi A^{\mathrm{T}} - L\Theta L^{\mathrm{T}})$ distinct eigenvalues. In theory, we can therefore solve (22) using the pre-conditioned conjugate gradient (P-CG) method in at most $1 + 2 \times \texttt{nskip}$ iterations in the case of skipping pivots, or $1 + \texttt{nadd}$ iterations in the case of adding quantities on the diagonal; see Theorem 5.4 in [59]. In practice however, it is not recommended to proceed the conjugate gradient process for more than say 25 iterations. If many many pivots had to be modified, it is therefore crucial that the modified Cholesky factor results in a good conditioning and clustering of the eigenvalues of the pre-conditioned system.

Pre-conditioning in the P-CG method with the modified Cholesky factor $L\Theta^{1/2}$ is of course analogous to the popular incomplete Cholesky pre-conditioning. We note though that in our setting, L has the same sparsity structure as the exact Cholesky factor, but it has better numerical properties as $A\Pi A^{\mathrm{T}}$ gets ill-conditioned; the aim is to improve numerical accuracy. On the other hand, the traditional incomplete Cholesky approach aims at a sparser Cholesky factor, possibly sacrificing numerical accuracy.

We remark that other Cholesky based interior point codes also modify the Cholesky factor, in most cases by skipping pivots but some codes also add small quantities on the diagonal. However,

these interior point codes combine the modified Cholesky merely with one step of residual correction (iterative refinement). We have replaced the residual correction procedure by the P-CG method.

Unlike the nature of the residual correction step, the residual in (22) may actually increase during a P-CG iteration. However, the convergence of the P-CG method is (conceptually) monotone in a forward error sense. Namely, let Δy be the exact solution to (22). If we denote the iterative solutions in the P-CG method to (22) by $\Delta\hat{y}_1, \Delta\hat{y}_2, \ldots$, then we have

$$\|\Phi^{\mathrm{T}} A^{\mathrm{T}} \Delta\hat{y}_k - \Phi^{\mathrm{T}} A^{\mathrm{T}} \Delta y\| \geq \|\Phi^{\mathrm{T}} A^{\mathrm{T}} \Delta\hat{y}_{k+1} - \Phi^{\mathrm{T}} A^{\mathrm{T}} \Delta y\|,$$

with equality if and only if $\Delta\hat{y}_k = \Delta y$. Indeed, we observed that the P-CG method is capable of approximating $\Phi^{\mathrm{T}} A^{\mathrm{T}} \Delta y = \Phi^{\mathrm{T}} A^{\mathrm{T}} (A\Pi A^{\mathrm{T}})^{-1} b$ very accurately, even if $A\Pi A^{\mathrm{T}}$ is extremely ill-conditioned. The P-CG method may be the 'spare engine' that Saunders was hoping for in [63].

8 Dense Columns

Recall from (29) that if two primal constraints i and j do not have any variable in common then $(A\Pi A^{\mathrm{T}})_{ij} = 0$. Put another way, if there is a variable that occurs in all primal constraints, then the nonzero pattern of $A\Pi A^{\mathrm{T}}$ will generally be fully dense. This situation arises for instance in certain standard SDP relaxations of combinatorial problems, where there is merely one variable: a large $\nu \times \nu$ matrix variable. However, even in the more favorable case where there is a large number of variables which are all of low dimensionality, a few variables occurring in a large number of constraints can account for most of the nonzeros in $A\Pi A^{\mathrm{T}}$. The latter situation is our concern in this section.

In order to simplify notations, let $\bar{A} = A\Phi$ with $\Phi\Phi^{\mathrm{T}} = \Pi$, so that (22) becomes

$$\text{find } \Delta y \text{ such that } \bar{A}\bar{A}^{\mathrm{T}} \Delta y = b. \tag{34}$$

Partitioning $\bar{A}$ in columns as

$$\bar{A} = \left[\begin{array}{cccc} \bar{a}_1, & \bar{a}_2, & \cdots, & \bar{a}_n \end{array} \right], \tag{35}$$

we may write

$$\bar{A}\bar{A}^{\mathrm{T}} = \sum_{i=1}^{n} \bar{a}_i \bar{a}_i^{\mathrm{T}}. \tag{36}$$

We see that a column $\bar{a}_i$ with $\mathrm{nnz}(\bar{a}_i)$ nonzeros implies $\mathrm{nnz}(\bar{a}_i \bar{a}_i^{\mathrm{T}}) = \mathrm{nnz}(\bar{a}_i)^2$ nonzeros in $\bar{A}\bar{A}^{\mathrm{T}}$. A dense column is a column $\bar{a}_i$ where $\mathrm{nnz}(\bar{a}_i)$ is relatively large. Relatively large could for instance mean more than ten times the number of nonzeros in $\bar{a}_j$ for 75% of the columns $\bar{a}_j$ of $\bar{A}$. As observed in [25], the number of nonzeros in a dense column can be much smaller than the dimension m, and hence sparsity should be exploited in dense columns as well.

The basic idea of dense column handling is to first remove the dense columns from (36) in order to promote sparsity in the remaining part of $\bar{A}\bar{A}^{\mathrm{T}}$. We will then factor the resulting simplified and sparse system before incorporating the dense columns by means of iterative methods [1] or low rank updates [43].

Reorder the columns of $\bar{A}$ such that the first k columns of $\bar{A}$ are sparse, whereas the remaining $n-k$ columns are dense. We can then start by factoring the contribution of the sparse part of $\bar{A}$,

$$\bar{A}_{1:k}\bar{A}_{1:k} = \sum_{i=i}^{k} \bar{a}_i \bar{a}_i^{\mathrm{T}} = L_k \Theta_k L_k^{\mathrm{T}} \tag{37}$$

by means of a (block) sparse Cholesky factorization, yielding a sparse unit diagonal lower triangular matrix L_k and a diagonal matrix Θ_k. Obviously, the entries in Θ_k are nonnegative. Andersen and Andersen [7] provide a recent survey of block sparse Cholesky techniques in the context of interior point methods.

Unfortunately, the removal of dense columns typically leaves $L_k\Theta_k L_k^{\mathrm{T}}$ rank deficient, so that some diagonal entries of Θ_k can be zero. This means that we cannot introduce the contribution of the dense columns to $(\bar{A}\bar{A}^{\mathrm{T}})^{-1}$ by simply applying the low rank update formula of Sherman, Morrison and Woodbury. As a remedy to this problem, Andersen [9] has proposed to add artificial quantities on the diagonal of $L_k\Theta_k L_k^{\mathrm{T}}$ in order to make this matrix full rank. The subsequent Sherman-Morrison-Woodbury update will then both add the dense columns and remove the artificial columns. Another option is to use the modified full rank Cholesky factor as a preconditioner in the conjugate gradient method, as in Adler et al. [1]. Both approaches suffer from numerical problems. Higher numerical stability is achieved with the product form Cholesky technique which was recently advocated by Goldfarb and Scheinberg [25] and subsequently implemented in SeDuMi [67].

After factoring $\bar{A}_{1:k}\bar{A}_{1:k}$ using a block sparse Cholesky factorization, as in (37), the product form Cholesky technique proceeds with incorporating the contribution of the rank-1 matrix $\bar{a}_(k+1)\bar{a}_(k+1)^{\mathrm{T}}$ into the factorization. We have

$$\begin{aligned} L_k\Theta_k L_k^{\mathrm{T}} + \bar{a}_{k+1}\bar{a}_{k+1}^{\mathrm{T}} &= L_k(\Theta_k + \tilde{a}_{k+1}\tilde{a}_{k+1}^{\mathrm{T}})L_k^{\mathrm{T}} \\ &= L_k L(\tilde{a}_{k+1}, \beta_{k+1})\Theta_{k+1}L(\tilde{a}_{k+1}, \beta_{k+1})^{\mathrm{T}} L_k^{\mathrm{T}}, \end{aligned} \tag{38}$$

where

$$\tilde{a}_{k+1} := L_k^{-1}\bar{a}_{k+1} \tag{39}$$

and $L(p,q) - I$ is the below-diagonal part of the rank-1 matrix pq^{T}. More precisely,

$$L(p,q)i,j = \begin{cases} 0 \text{ for } i = 1,2,\ldots,j-1 \\ 1 \text{ for } i = j \\ p_i q_j \text{ for } i = j+1, j+2, \ldots, m, \end{cases} \tag{40}$$

for all $j = 1,2,\ldots,m$. Thus, the rank-1 update of the Cholesky factorization corresponds to solving $\beta_{k+1} \in \Re^m$ and the order m diagonal matrix Θ_{k+1} from

$$\Theta_k + \tilde{a}_{k+1}\tilde{a}_{k+1}^{\mathrm{T}} = L(\tilde{a}_{k+1}, \beta_{k+1})\Theta_{k+1}L(\tilde{a}_{k+1}, \beta_{k+1})^{\mathrm{T}},$$

which can be done in linear time [12, 25]. After computing β_{k+1} we proceed by setting

$$L_{k+1} = L_k L(\tilde{a}_{k+1}, \beta_{k+1}).$$

Then we satisfy (37) with $k = k+1$. We continue the process of incorporating rank-1 contributions into the factorization until no dense columns are left, i.e. $k = n$.

Recall from Section 5 that Π is a block diagonal matrix with block $\Pi[i]$ matching the size of $x[i]$. In linear programming, all variables are scalar so that Π is diagonal and $\bar{A}$ has exactly the same sparsity structure as A. In this case, the dense columns of $\bar{A}$ correspond directly with the dense columns of A.

Since $P(d^q)$ is not diagonal but merely block diagonal, this correspondence is lost in the setting of second order cone programming. However, Goldfarb et al. [26] observed that a slight change in the definition of $\bar{A}$ re-establishes the correspondence to a large extent. Namely, we have from (32) that

$$A^q P(d^q)(A^q)^{\mathrm{T}} = (A^q D^q)(A^q D^q)^{\mathrm{T}} + A^q_{2:} H (A^q_{2:})^{\mathrm{T}} - A^q_1 \mathrm{Det}(d^q)(A^q_1)^{\mathrm{T}},$$

where H and $\mathrm{Det}(d^q)$ are diagonal matrices (as in the linear programming case). We let

$$\bar{A} = \left[\begin{array}{cccc} A^l P(d^l)^{1/2}, & A^q D^q, & A^q_{2:} H^{1/2}, & A^s P(d^s)^{1/2} \end{array} \right], \tag{41}$$

so that

$$AP(d)A^{\mathrm{T}} = \bar{A}\bar{A}^{\mathrm{T}} - A^q_1 \mathrm{Det}(d^q)(A^q_1)^{\mathrm{T}}.$$

Let $\gamma > 0$ be a threshold such that $\bar{a}_i$ is treated as a dense column if and only if $\mathrm{nnz}(\bar{a}_i) > \gamma$. We partition $\bar{A}$ as $(\bar{A}_{\mathcal{S}}, \bar{A}_{\neg\mathcal{S}})$, where

$$\mathcal{S} = \{i \mid \mathrm{nnz}(\bar{a}_i) \leq \gamma\}$$

is the set of sparse columns and $\neg\mathcal{S} = \{1, 2, \ldots, n\} \setminus \mathcal{S}$ is the set of dense columns. Similarly, we let

$$\mathcal{S}(q) = \{k \mid \mathrm{nnz}(A^q[k]d^q[k]) \leq \gamma\},$$

where $\mathrm{nnz}(A^q[k]d^q[k])$ is evaluated symbolically (independent of the numerical value of d^q). It is clear that if $A^q[k]d^q[k] \in \Re^m$ is sparse then all columns of $A^q[k]$ are sparse. The matrix

$$M = \bar{A}_{\mathcal{S}}\bar{A}^{\mathrm{T}}_{\mathcal{S}} - \sum_{k \in \mathcal{S}(q)} \det(d^q[k]) a^q_1[k](a^q_1[k])^{\mathrm{T}}$$

is positive semidefinite. We will now factor the sparse matrix M using a block sparse Cholesky factorization, and then update the factorization as in (38), until the matrix $AP(d)A^{\mathrm{T}}$ has been factored completely. See [26] for a numerical analysis of this approach. In this section, we have focused on NT scaling, but the product form Cholesky technique can be extended to HKM and AHO scaling, albeit with some complications. Most notably, the Cholesky factorization has to be replaced by an LU-factorization in the case of AHO scaling.

9 Upper-Bounds and Fixing Constraints

A bound (or box) constraint is a constraint of the form $0 \leq x^l[i] \leq \mathtt{ub}[i]$ where $\mathtt{ub}[i]$ is a given scalar upper bound on the nonnegative decision variable $x^l[i]$. A fixing constraint is a constraint of the form $x_i[k] = \mathtt{fx}_i[k]$, which fixes the ith component of decision variable $x[k]$ to a given scalar quantity $\mathtt{fx}_i[k]$. Such constraints impose hardly any computational cost per iteration if their structure is exploited. Upper bound constraints have been treated in such a way since the very beginning of primal-dual interior point methods, probably because it was customary to do so in the simplex

method as well [43]. The special handling of fixing constraints in second order cone programming has been developed by Andersen et al. [8].

In the standard form (1), an upper-bound constraint on $x^l[i]$ is modeled with a slack variable, say $x^l[j]$, which does not occur in any other constraint. The upper-bound constraint therefore takes the following form in (1):

$$x^l[i] + x^l[j] = \mathtt{ub}[i]. \tag{42}$$

We order the nonnegative variables such that the first $m[\mathtt{ub}]$ of them are upper-bounded variables, and the last $m[\mathtt{ub}]$ are the corresponding slack variables. The collection of all upper-bound constraints in a given model can now be written as

$$\left[\begin{array}{ccc} I, & 0, & I \end{array}\right] x^l = \mathtt{ub}, \tag{43}$$

where I is the $m[\mathtt{ub}] \times m[\mathtt{ub}]$ identity matrix.

For nonnegative scalar variables, a fixing constraint fixes the entire variable $x[k]$ which can therefore be eliminated from the problem. However, such an elimination in impossible for Lorentz cone variables and semidefinite cone variables which are only fixed in some components. (In fact, all constraints in the famous semidefinite relaxation of the MAX-CUT problem [24] are fixing constraints!)

In this section, we consider only those fixing constraints which affect Lorentz cone variables. We remark that if two components of a Lorentz cone variable are fixed, we can eliminate one of them. For instance, a Lorentz cone constraint in $\Re^{n[k]}$ with two fixed entries, say

$$x_1[k] \geq \sqrt{\sum_{i=2}^{n[k]} x_i[k]^2},\ x_1[k] = 5,\ x_{n[k]}[k] = 4$$

is equivalent to a Lorentz cone constraint in $\Re^{n[k]-1}$ with one fixed entry, viz.

$$x_1[k] \geq \sqrt{\sum_{i=2}^{n[k]-1} x_i[k]^2},\ x_1[k] = 3 = \sqrt{5^2 - 4^2}.$$

Thus, we may assume that an individual fixing constraint has the following form:

$$x^q_{\sigma(k)}[k] = \mathtt{fx}[k], \tag{44}$$

where $\sigma(k) \in \{1, 2, \ldots, n^q[k]\}$ denotes the given index of the fixed component in $x^q[k]$. We order the Lorentz cone variables in such a way that fixing constraints exist for the first $m[\mathtt{fx}]$ variables. The collection of all such fixing constraints in a given model can now be written as

$$\left[\begin{array}{cc} \tilde{F}^q & 0 \end{array}\right] x^q = \mathtt{fx}, \quad \tilde{F}^q = \left(e_{\sigma(1)} \oplus e_{\sigma(2)} \oplus \cdots \oplus e_{\sigma(m[\mathtt{fx}])}\right)^{\mathrm{T}}, \tag{45}$$

where e_i denotes the ith column of the identity matrix.

In order to treat (43) and (45) explicitly, we partition the data as follows:

$$A^l = \begin{bmatrix} I & 0 & I \\ 0 & 0 & 0 \\ \tilde{A}^l_1 & \tilde{A}^l_2 & 0 \end{bmatrix}, \; A^q = \begin{bmatrix} I & 0 \\ \tilde{F}^q & 0 \\ \tilde{A}^q_1 & \tilde{A}^q_2 \end{bmatrix}, \; A^s = \begin{bmatrix} 0 \\ 0 \\ \tilde{A}^s \end{bmatrix}, \; b = \begin{bmatrix} \mathtt{ub} \\ \mathtt{fx} \\ \tilde{b} \end{bmatrix}. \tag{46}$$

We may assume without loss of generality that

$$\tilde{A}^q_1 (\tilde{F}^q)^{\mathrm{T}} = 0. \tag{47}$$

Namely, one may otherwise replace $\tilde{b}$ by $\tilde{b} - \tilde{A}^q_1 (\tilde{F}^q)^{\mathrm{T}} \mathtt{fx}$, and remove all nonzeros from the columns in $\tilde{A}^q$ corresponding to fixed components of x^q.

Since we like to treat upper-bounds and fixing constraints in a unified fashion, we also partition A as

$$A = \begin{bmatrix} F \\ \tilde{A} \end{bmatrix}, \quad F \in \Re^{(m[\mathtt{ub}]+m[\mathtt{fx}])\times n}. \tag{48}$$

Using the Schur-complement (or 2×2 block Cholesky) technique, we have

$$\begin{aligned} A\Pi A^{\mathrm{T}} &= \begin{bmatrix} F\Pi F^{\mathrm{T}} & F\Pi \tilde{A}^{\mathrm{T}} \\ \tilde{A}\Pi F^{\mathrm{T}} & \tilde{A}\Pi \tilde{A}^{\mathrm{T}} \end{bmatrix} \\ &= \begin{bmatrix} I & 0 \\ L_{21} & I \end{bmatrix} \begin{bmatrix} F\Pi F^{\mathrm{T}} & 0 \\ 0 & \tilde{A}\tilde{\Pi}\tilde{A}^{\mathrm{T}} \end{bmatrix} \begin{bmatrix} I & L_{21}^{\mathrm{T}} \\ 0 & I \end{bmatrix}, \end{aligned}$$

where

$$L_{21} := \tilde{A}\Pi F^{\mathrm{T}} (F\Pi F^{\mathrm{T}})^{-1},$$

and $\tilde{\Pi}$ is a matrix satisfying

$$\tilde{A}\tilde{\Pi}\tilde{A}^{\mathrm{T}} = \tilde{A}\Pi\tilde{A}^{\mathrm{T}} - \tilde{A}\Pi F^{\mathrm{T}} (F\Pi F^{\mathrm{T}})^{-1} F\Pi\tilde{A}^{\mathrm{T}} = \tilde{A}\Pi\tilde{A}^{\mathrm{T}} - L_{21}(F\Pi F^{\mathrm{T}})L_{21}^{\mathrm{T}}. \tag{49}$$

It is obvious that such $\tilde{\Pi}$ exists. Below, we show the less obvious fact that we can choose $\tilde{\Pi}$ with the same structure as $\Pi = P(d)$ in the case of NT scaling. This means that we have essentially reduced the original normal equations system to a smaller normal equations system, without causing any fill-in of nonzeros.

It is clear from (29) that $F\Pi F^{\mathrm{T}}$ is a diagonal matrix. In fact, if we let

$$s(i) := i + \kappa(l) - m[\mathtt{ub}] \text{ for } i = 1, 2, \ldots, m[\mathtt{ub}]$$

denote the index of the slack variable in the ith upper-bound constraint, we have

$$\begin{aligned} F\Pi F^{\mathrm{T}} &= \left((\frac{x_1}{z_1} + \frac{x_{s(1)}}{z_{s(1)}}) \oplus \cdots \oplus (\frac{x_{m[\mathtt{ub}]}}{z_{m[\mathtt{ub}]}} + \frac{x_{s(m[\mathtt{ub}])}}{z_{s(m[\mathtt{ub}])}}) \right) \oplus \\ &\quad \left(\pi^q_{\sigma(1),\sigma(1)}[1] \oplus \cdots \oplus \pi^q_{\sigma(m[\mathtt{fx}]),\sigma(m[\mathtt{fx}])}[m[\mathtt{fx}]] \right), \end{aligned} \tag{50}$$

where $\pi^q_{i,i}[k]$ denotes the ith diagonal entry of $\Pi^q[k]$.

We have

$$L_{21}e_i = \frac{x_i z_{s(i)}}{x_i z_{s(i)} + x_{s(i)} z_i} \tilde{a}_i^l \text{ for } i = 1, 2, \ldots, m[\texttt{ub}], \tag{51}$$

and

$$L_{21}e_{m[\texttt{ub}]+i} = \frac{1}{\pi^q_{\sigma(i),\sigma(i)}[i]} \tilde{A}^q[i]\Pi^q[i]e_{\sigma(i)} \text{ for } i = 1, 2, \ldots, m[\texttt{fx}]. \tag{52}$$

For NT-scaling, one has

$$\Pi^q[i] = P(d^q[i]) = d^q[i](d^q[i])^{\mathrm{T}} - \det(d^q[i])J.$$

Using (47), one can simplify (52) to

$$L_{21}e_{m[\texttt{ub}]+i} = \frac{d_i^q[i]}{\pi^q_{\sigma(i),\sigma(i)}[i]} \tilde{A}^q[i]d^q[i] \text{ for } i = 1, 2, \ldots, m[\texttt{fx}]. \tag{53}$$

From (49)–(53) we have

$$\begin{aligned} \tilde{A}\tilde{\Pi}\tilde{A}^{\mathrm{T}} &= \tilde{A}\Pi\tilde{A}^{\mathrm{T}} - \sum_{i=1}^{m[\texttt{ub}]} \frac{x_i}{z_i} \cdot \frac{x_i z_{s(i)}}{x_i z_{s(i)} + x_{s(i)} z_i} \tilde{a}_i^l (\tilde{a}_i^l)^{\mathrm{T}} \\ &\quad - \sum_{i=1}^{m[\texttt{fx}]} \frac{d_i^q[i]^2}{\pi^q_{\sigma(i),\sigma(i)}[i]} \tilde{A}^q[i]d^q[i](\tilde{A}^q[i]d^q[i])^{\mathrm{T}}. \end{aligned}$$

This shows that we may set $\tilde{\Pi}[k] = \Pi[k]$ for all blocks, except the first $m[\texttt{ub}]$ diagonal blocks in Π^l and the first $m[\texttt{fx}]$ diagonal blocks in Π^q. The modified blocks are given by the formulas

$$\tilde{\pi}^l[i] = \frac{x_i}{z_i}\left(1 - \frac{x_i z_{s[i]}}{x_i z_{s(i)} + x_{s(i)} z_i}\right) = 1/\left(\frac{z_i}{x_i} + \frac{z_{s(i)}}{x_{s(i)}}\right) \text{ for } i = 1, 2, \ldots, m[\texttt{fx}] \tag{54}$$

and

$$\tilde{\Pi}^q[i] = \alpha_i d^q[i](d^q[i])^{\mathrm{T}} - \det(d^q[i])J \text{ for } i = 1, 2, \ldots, m[\texttt{fx}], \tag{55}$$

where

$$\alpha_i = 1 - \frac{d_i^q[i]^2}{\pi^q_{\sigma(i),\sigma(i)}[i]} = \begin{cases} -2\det(d^q[i])/\|d\|^2 \text{ if } \sigma(i) = 1 \\ \det(d^q[i])/(\det(d^q[i]) + (d^q_{\sigma(i)}[i])^2) \text{ if } \sigma(i) > 1. \end{cases} \tag{56}$$

One may have concerns about the lack of positive definiteness of $\tilde{\Pi}^q$. However, one could recover positive definiteness by either adding something to the $\sigma(i)$th diagonal entry of $\tilde{\Pi}^q[i]$ or by removing the $\sigma(i)$th row and column from $\tilde{\Pi}^q[i]$. Such modifications do not affect the actual computation, since $\tilde{A}^q$ is all-zero in the columns corresponding to fixed components, see (47).

The computational savings of handling bounds and fixing constraints in the above explained manner can be substantial. First, applying a general sparse ordering heuristic to $A\Pi A^{\mathrm{T}}$ directly may cause more fill-in in the resulting Cholesky factor than an ordering with the upper-bound and fixing constraints up front. Second, and more importantly, we do not need to calculate $L_{21}F\Pi F^{\mathrm{T}}L_{21}^{\mathrm{T}}$ in (49) explicitly, as a general Cholesky factorization of $A\Pi A^{\mathrm{T}}$ does. Instead, we form $\tilde{A}\tilde{\Pi}\tilde{A}^{\mathrm{T}}$ directly using the technique described in Section 6.

10 Iterates in Product Form

Referring to Section 5, we observe that the matrix Π involves $L(z)^{-1}$ in the case of AHO scaling, and $P(z)^{-1/2}$ in the case of HKM or NT scaling. The matrices $L(z)^{-1}$ and $P(z)^{-1/2}$ have the same spectral radius, viz. $1/\lambda_{\min}(z)$; see [14, 69]. When the optimum is approached, the z-iterates approach the boundary of the cone $\mathcal{K}$, i.e. $\lambda_{\min}(z)$ and $\det(z)$ approach zero. For linear programming, we simply have that the 'non-basic' entries in z approach zero, so that computing Π does not pose any numerical problems. However, for the semi-definite and Lorentz cone variables, the boundary of the cone can be approached without any of the entries in z getting small. Computing $\lambda_{\min}(z)$ or $\det(z)$ from the floating point representation of z will therefore result in severe numerical cancellation when the optimum is approached. Hence, it is impossible to compute $L(z)^{-1}$, $P(z)^{-1/2}$, or Π accurately, if the computations are based on the floating point representation of a near optimal z. To resolve this problem, we may store and update the z-solutions in a product form, where small spectral values of z correspond to small entries in the factors of this product form. This approach has been proposed in [70].

In the setting of NT-scaling, it is natural to use the v-space product form [72, 70]. Recall that the NT-scaling operator is

$$\Pi = P(d) = P(d^l) \oplus P(d^q) \oplus P(d^s).$$

We let

$$\Phi = \Phi^l \oplus \Phi^q \oplus \Phi^s, \quad \Phi^l = P(d^l)^{1/2},\ \Phi^q = P(d^q)^{1/2},$$

and Φ^s is the lower triangular Cholesky factor of $P(d^s)$. We have

$$\Pi = P(d) = \Phi\Phi^{\mathrm{T}}, \quad \Phi^{\mathrm{T}}\mathcal{K} = \Phi\mathcal{K} = \Phi^{-1}\mathcal{K} = \mathcal{K}.$$

Letting $v = \Phi^{\mathrm{T}} z$, we have

$$x = \Pi z = \Phi\Phi^{\mathrm{T}} z = \Phi v, \tag{57}$$

so that also $v = \Phi^{-1}x$. The factors Φ and v thus determine x and z in the product form $x = \Phi v$ and $z = \Phi^{-\mathrm{T}} v$. We see that $\|v\|^2 = (\Phi^{\mathrm{T}} z)^{\mathrm{T}}\Phi^{-1}x = z^{\mathrm{T}}x$ is the duality gap, and hence all entries in v approach zero as the optimum is approached. In fact, it holds that $\lambda_i(v) = \sqrt{\lambda_i(P(x)^{1/2}z)}$ for all $i = 1, 2, \ldots, \nu$, see [69]. Due to the neighborhood ($\mathcal{N}_2$, $\mathcal{N}_\infty^-$ or $\mathcal{N}^{\mathrm{wr}}$) in the interior point algorithm, the matrices $L(v)$ and $P(v)$ remain well conditioned throughout the process, whereas the condition numbers of $L(z)$ and $P(z)$ tend to infinity as the optimum is approached.

Letting $\overline{\Delta x} := \Phi^{-1}\Delta x$ and $\overline{\Delta z} := \Phi^{\mathrm{T}}\Delta z$, we have $x^+ = x + t\Delta x = \Phi(v + t\overline{\Delta x})$, and $\Phi^{\mathrm{T}} z^+ = \Phi^{\mathrm{T}}(z + t\Delta z) = v + t\overline{\Delta z}$. Since v is well conditioned, we can in principle control the step length t in such a way that $\bar{z}^+ := v + t\overline{\Delta z}$ is also well conditioned. However, in practice we will give priority to fast convergence when the rate of linear convergence is close to zero; see also [70]. Below, we shall work out the approach for the Lorentz cone and semidefinite cone variables, respectively.

For the Lorentz cone, we have $\Pi^q[k] = P(d^q[k]) = d^q[k](d^q[k])^{\mathrm{T}} - \det(d^q[k])J$. In SeDuMi, we maintain both $d^q[k]$ and $\det(d^q[k])$, $k = 1, \ldots, \kappa(q)$. In principle, one could also compute $\det(d^q[k])$ from $d^q[k]$ directly, but this can lead to severe numerical cancellation when the optimum is approached. This situation arises when $x^q[k]^{\mathrm{T}} z^q[k] \to 0$ but $\|x^q[k]\|\|z^q[k]\| \not\to 0$, so that d will have one spectral value approaching zero, whereas the other approaches infinity [70].

Apart from maintaining d^q and $\det(d^q)$, we also maintain the scaled iterate v^q. In this way, we can always recover the unscaled iterates x^q and z^q by applying relation (57). After computing the scaled step

$$(\bar{x}^+, y^+, \bar{z}^+) = (v, y, v) + t(\overline{\Delta x}, \Delta y, \overline{\Delta z}), \tag{58}$$

we update the factors $d^q, \det(d^q)$ and v^q into $(d^q)^+, \det((d^q)^+)$ and $(v^q)^+$. Obviously, the update formulas are basically the same for each Lorentz block $k = 1, 2, \ldots, \kappa(q)$. To avoid unnecessarily complicated formulas, we therefore temporarily act as if the cone $\mathcal{K}$ is just a single Lorentz cone so that we can write d, v, etc. instead of the cumbersome $d^q[k]$, $v^q[k]$, etc. in the update formula below. Furthermore, the formula uses some auxiliary quantities $\alpha, \gamma, \chi, \psi$, which have only a local meaning.

1. Let
$$\begin{cases} \det(v^+) = \sqrt{\det(\bar{x}^+)\det(\bar{z}^+)} \\ \text{tr } v^+ = \sqrt{(\bar{x}^+)^{\mathrm{T}}\bar{z}^+ + 2\det(v^+)}. \end{cases}$$

2. Let
$$\chi = \frac{1}{\text{tr } v^+}(\bar{x}^+ + \det(v^+)(\bar{z}^+)^{-1}), \quad \det(\chi) = \frac{\det(v^+)}{\det(\bar{z}^+)},$$
and compute
$$d^+ = P(d^{1/2})\chi, \quad \det(d^+) = \det(d)\det(\chi).$$

3. Let
$$\psi = \bar{x}^+ - \det(v^+)(\bar{z}^+)^{-1}, \quad \alpha = \frac{d^{\mathrm{T}}\psi}{(\text{ tr } (d^+)^{1/2})^2},$$
$$\phi = \frac{1}{2\sqrt{\det(\chi)}}(\psi - \alpha\chi), \quad \gamma = \frac{\alpha + \text{ tr } \phi}{(\text{ tr } d^{1/2})^2},$$
and compute
$$\begin{cases} v_1^+ = \text{ tr } v^+/\sqrt{2} \\ v_i^+ = \phi_i + \gamma d_i \text{ for } i = 2, 3, \ldots, n. \end{cases}$$

Since the above formulas are based on the scaled iterates $\bar{x}$ and $\bar{z}$, they do not suffer from the increasing ill-conditioning of the iterates in the interior point method. We refer to [70] for details.

For the semidefinite cone, we have $\Pi^s[k] = P(d^s[k]) = D^s[k] \otimes D^s[k]$, where the positive definite matrix $D^s[k]$ is implicitly defined by the relation $d^s[k] = \text{ vec}\,(D^s[k])$. Let $U_d^s[k]$ denote the upper triangular Cholesky factor of $D^s[k]$. We have

$$(\Phi^s[k])^{\mathrm{T}} = U_d^s[k] \otimes U_d^s[k].$$

Analogous to the Lorentz cone case, we also maintain the scaled iterate $v^s[k] = \text{ vec}\,(V^s[k])$. In this way, we can always recover the unscaled iterates x^q and z^q by applying relation (57). More specifically, we have

$$X^s[k] = (U_d^s[k])^{\mathrm{T}} V^s[k] U_d^s[k], \quad Z^s[k] = (U_d^s[k])^{-1} V^s[k] (U_d^s[k])^{-\mathrm{T}} \text{ for } k = 1, 2, \ldots, \kappa(s).$$

In order to avoid problems with the inherent ill-conditioning of $X^s[k]$ and $Z^s[k]$ near the optimum, we update $U_d^s[k]$ and $V^s[k]$ based on the scaled iterates

$$(\bar{x}^s[k])^+ = \operatorname{vec}(\bar{X}^s[k])^+, \quad (\bar{z}^s[k])^+ = \operatorname{vec}(\bar{Z}^s[k])^+,$$

where $\bar{x}^+$ and $\bar{z}^+$ are defined in (58). To avoid unnecessarily complicated formulas, we temporarily act as if the cone $\mathcal{K}$ is just a single positive semidefinite cone so that we can dispense with the block indicators s and k in the update formula below. The formula uses some auxiliary matrices $\bar{U}_x^+, W, Q_w, T, Q_v^+$ and R, which have only a local meaning.

1. Compute an upper triangular matrix $\bar{U}_x^+$ as the Cholesky factorization of $\bar{X}^+$, i.e.

$$\bar{X}^+ = (\bar{U}_x^+)^{\mathrm{T}} \bar{U}_x^+.$$

2. Let $W := \bar{U}_x^+ \bar{Z}^+ (\bar{U}_x^+)^{\mathrm{T}}$; compute an orthogonal matrix Q_w and a diagonal matrix Λ_v^+ as the symmetric eigenvalue decomposition of $W^{1/2}$, i.e.

$$W = Q_w (\Lambda_v^+)^2 Q_w^{\mathrm{T}}.$$

3. Let $T := (\Lambda_v^+)^{-1/2} Q_w^{\mathrm{T}}$; compute an orthogonal matrix Q_v^+ and an upper triangular matrix R as the QR-factorization of T, i.e.

$$T = (Q_v^+)^{\mathrm{T}} R.$$

4. Compute

$$U_d^+ = R \bar{U}_x^+ U_d,$$
$$V^+ = Q_v^+ \Lambda_v^+ (Q_v^+)^{\mathrm{T}}.$$

Again, we refer to [70] for details.

11 Correcting the Search Direction

The technique of adaptively correcting the search direction has proved to be quite successful in practice. The underlying idea of this technique is that once the factorization of the reduced normal equations system (21) has been computed, it is computationally inexpensive to solve (21) for different choices of r.

The most famous correction technique is due to Mehrotra [46]. Mehrotra's scheme combines ideas of Monteiro et al. [51] on power series extension and Sonnevend et al. [65] and Mizuno et al. [49] on adaptive step predictor-corrector algorithms.

11.1 Continuous Affine Scaling Trajectory

As in Monteiro et al. [51], we consider the continuous affine scaling trajectories through the current solution (x, y, z). The continuous affine scaling trajectory is obtained by taking infinitesimal steps along the predictor (or primal-dual affine scaling) direction, while re-evaluating this direction continuously. In other words, this is the smooth curve through (x, y, z) where the tangent direction always equals the predictor direction. More formally, the trajectory is defined as follows:

$$\begin{cases} x(0) = x,\ y(0) = y,\ z(0) = z \\ (1-t)(\dot{x}(t) + \Pi(t)\dot{z}(t)) = -x(t) \text{ for } t < 1 \\ A\dot{x}(t) = 0,\ A^{\mathrm{T}}\dot{y}(t) + \dot{z}(t) = 0, \text{ for } t < 1, \end{cases} \tag{59}$$

with $\Pi(t)^{\mathrm{T}} z(t) = x(t)$. Here, we use the standard notation

$$\dot{x}(t) = \frac{dx(t)}{dt}, \quad \ddot{x}(t) = \frac{d^2x(t)}{dt^2}.$$

Observe from (14)–(16) that $(1-t)(\dot{x}(t), \dot{y}(t), \dot{z}(t))$ is exactly the predictor direction evaluated at $(x(t), y(t), z(t))$. The factor $1-t$ ensures that optimality is approached when $t \to 1$. In particular, it is easily verified that $x(t)^{\mathrm{T}} z(t) = (1-t)x^{\mathrm{T}} z$. For $t = 0$, we have

$$\dot{x}(0) + \Pi\dot{z}(0) = -x. \tag{60}$$

Differentiating (59), we obtain that

$$\begin{cases} \ddot{x}(0) + \Pi\ddot{z}(0) = (\Pi - \dot{\Pi}(0))\dot{z}(0) \\ A\ddot{x}(0) = 0,\ A^{\mathrm{T}}\ddot{y}(0) + \ddot{z}(0) = 0. \end{cases} \tag{61}$$

As we observed before, $(\dot{x}(0), \dot{y}(0), \dot{z}(0))$ is simply the predictor direction at the current solution (x, y, z). How $\dot{\Pi}(0)$ can be computed depends on the choice of Π. Since the reduced normal equations have already been factored to compute the predictor direction from (21) with $r = -x$, it is computationally inexpensive to solve (21) with $r = (\Pi - \dot{\Pi}(0))\dot{z}(0)$ in order to compute the second order derivative.

Recall that NT-scaling is implicitly defined by the relations

$$\Pi(t) = P(d(t)), \quad x(t) = P(d(t))z(t).$$

Choose Φ and define v as in Section 10, i.e.

$$P(d) = \Phi\Phi^{\mathrm{T}},\ \Phi\mathcal{K} = \mathcal{K},\ v = \Phi^{\mathrm{T}} z = \Phi^{-1} x.$$

It can be shown [66, 68] that

$$\Pi(t) = P(d(t)) \Longrightarrow \dot{\Pi}(0) = \Phi L(u)\Phi^{\mathrm{T}} \text{ with } u := L(v)^{-1}(\Phi^{-1}\dot{x}(0) - \Phi^{\mathrm{T}}\dot{z}(0)). \tag{62}$$

Thus, one has

$$\Pi - \dot{\Pi}(0) = \Phi(I - L(u))\Phi^{\mathrm{T}} = \Phi L(L(v)^{-1}v - u)\Phi^{\mathrm{T}} = \Phi L(L(v)^{-1}\Phi^{-1}\dot{x}(0))\Phi^{\mathrm{T}},$$

where we used that

$$v = \Phi^{-1}x = -\Phi^{-1}(\dot{x}(0) + \Pi\dot{z}(0)) = -(\Phi^{-1}\dot{x}(0) + \Phi^{\mathrm{T}}\dot{z}(0)).$$

To summarize, the second order derivative of the continuous affine scaling trajectory with NT-scaling is characterized by

$$\begin{cases} \ddot{x}(0) + P(d)\ddot{z}(0) = \Phi L(L(v)^{-1}\Phi^{-1}\dot{x}(0))\Phi^{\mathrm{T}}\dot{z}(0) \\ A\ddot{x}(0) = 0,\ A^{\mathrm{T}}\ddot{y}(0) + \ddot{z}(0) = 0. \end{cases} \tag{63}$$

For linear programming, one simply has

$$\ddot{x}_i(0) + \frac{x_i}{z_i}\ddot{z}(0) = \frac{\dot{x}_i(0)\dot{z}_i(0)}{z_i} \text{ for } i = 1, 2, \ldots, \kappa(l),$$

irrespective of the method of scaling.

We mention that the AHO continuous affine scaling trajectory $(x(t), y(t), z(t))$ satisfies (and is in fact characterized by)

$$L(x(t))z(t) = L(z(t))x(t) = (1-t)L(x)z \text{ for } t < 1.$$

However, the three different continuous affine scaling trajectories that are associated with NT scaling and both versions of HKM scaling satisfy the relation

$$\lambda(P(x(t))^{1/2}z(t)) = \lambda(P(z(t))^{1/2}x(t)) = (1-t)\lambda(P(x)^{1/2}z). \tag{64}$$

The continuous affine scaling trajectory technique with NT scaling leads to the second order direction of Sturm [66, 68]. A different second order direction with NT scaling was introduced by Todd et al. [75]; their derivation is based on MZ-trajectories which we discuss below. With AHO-scaling, the two approaches are equivalent.

11.2 Continuous MZ-trajectory

MZ-trajectories are defined using the similarity symmetrization operator of Zhang [86] and Monteiro and Zhang [55]. The *similarity symmetrization* operator, parameterized by an $n \times n$ invertible matrix Ψ with $\Psi\mathcal{K} = \mathcal{K}$, is defined as

$$L_\Psi(x) := L(\Psi^{-1}x)\Psi^{\mathrm{T}}. \tag{65}$$

Since $L(x)z = L(z)x$ for any x and z, one also has that

$$L_\Psi(x)z = L_{\Psi^{-\mathrm{T}}}(z)x \text{ for all } x, z, \tag{66}$$

which is a key relation in the MZ approach. The MZ-trajectory through (x, y, z) associated with Ψ is the analytic curve satisfying

$$L_\Psi(x(t))z(t) = (1-t)L_\Psi(x)z \text{ for } t < 1. \tag{67}$$

Differentiating the above relation yields

$$L_\Psi(\dot{x}(t))z(t) + L_\Psi(x(t))\dot{z}(t) = -L_\Psi(x)z. \tag{68}$$

Differentiating once more, we obtain that

$$L_\Psi(\ddot{x}(t))z(t) + L_\Psi(x(t))\ddot{z}(t) = -2L_\Psi(\dot{x}(t))\dot{z}(t). \tag{69}$$

Let

$$\Pi = L_{\Psi^{-\mathrm{T}}}(z)^{-1}L_\Psi(x). \tag{70}$$

Using (65) and (66), we see that

$$x = \Pi^{\mathrm{T}}z = \Pi z.$$

so that Π satisfies indeed (17). (Using the fact that $\Psi\mathcal{K} = \mathcal{K}$, one can also show that $\Pi x^{-1} = z^{-1}$.)

With the above choice of Π and using (66), we can reformulate (68) for $t = 0$ as

$$\dot{x}(0) + \Pi\dot{z}(0) = -x. \tag{71}$$

Similarly, we reformulate (69) for $t = 0$ as

$$\begin{aligned} \ddot{x}(0) + \Pi\ddot{z}(0) &= -2L_{\Psi^{-\mathrm{T}}}(z)^{-1}L_\Psi(\dot{x}(0))\dot{z}(0) &(72)\\ &= -2\Psi L(\Psi^{\mathrm{T}}z)^{-1}L(\Psi^{-1}\dot{x}(0))\Psi^{\mathrm{T}}\dot{z}(0). &(73)\end{aligned}$$

Clearly, $\Psi = I$ yields the AHO-scaling Π in (24). Similarly, $\Psi = P(z)^{-1/2}$ and $\Psi = P(x)^{1/2}$ yield the HKM-variants in (25) and (26) respectively. The NT-scaling is obtained by setting $\Psi = \Phi$ with $\Phi\Phi^{\mathrm{T}} = P(d)$ and $P(d)z = x$, see (27).

Obviously, (71) coincides with (60), but (61) and (73) are in general not equivalent. In particular, for the NT-scaling, (61) reduces to (63) whereas (73) reduces to

$$\ddot{x}(0) + P(d)\ddot{z}(0) = \Phi L(v)^{-1}L(\Phi^{-1}\dot{x}(0))\Phi^{\mathrm{T}}\dot{z}(0). \tag{74}$$

Such MZ-trajectory based second order NT-directions have been implemented in SDPT3 [76, 75] and MOSEK [7, 8].

We remark that Monteiro and Zanjácomo [54] have defined several different trajectories through (x, y, z) leading to an optimal solution. One of these trajectories has an NT-flavor; it is defined as

$$L_{P(d(t))^{1/2}}(x(t))z(t) = (1-t)L_{P(d)^{1/2}}(x)z \text{ for } t < 1,$$

which differs from (67) in the sense that $\Psi(t) = P(d(t))^{1/2}$ changes continuously in t. Even though this trajectory does satisfy (64), it does not lead to the NT-based continuous affine scaling trajectory. In fact, the tangent along such a trajectory is the so-called V-direction in Todd [74].

11.3 Mehrotra Direction and its Variations

By truncating the power series of $x(t)$ and $z(t)$ to first order approximations $x(t) \approx x(0) + t\dot{x}(0)$, $z(t) \approx z(0) + t\dot{z}(0)$, one arrives at the principle underlying the Newton direction, viz. linearizing the $x(t)$ and $z(t)$ curves. However, one can also work with the second order approximations

$$x(t) \approx x(0) + t\dot{x}(0) + \frac{t^2}{2}\ddot{x}(0), \quad z(t) \approx z(0) + t\dot{z}(0) + \frac{t^2}{2}\ddot{z}(0).$$

Since $(x(t), z(t))$ approaches optimality when $t \to 1$, it makes sense to use $\Delta x = \dot{x}(0) + \ddot{x}(0)/2$ and $\Delta z = \dot{z}(0) + \ddot{z}(0)/2$ as a search direction. With this approach, super-quadratically convergent algorithms can be designed for linear programming [82].

The Mehrotra direction [46, 44] is built in two phases, viz. the predictor phase and the corrector phase. In the predictor phase, $(\dot{x}(0), \dot{z}(0))$ is computed. The maximal feasible step length along this search direction is denoted t_p^*, i.e.

$$t_p^* := \max\{t | x + t\dot{x}(0) \geq 0,\, z + t\dot{z}(0) \geq 0\}.$$

Then, a prediction is made on the new duality gap that can be achieved with a second order correction. This prediction is:

$$\mu = (1 - t_p^*)^3 x^{\mathrm{T}} z. \tag{75}$$

The Mehrotra direction is now solved from

$$\begin{cases} \Delta x + \Pi \Delta z = \frac{\mu}{\nu(\mathcal{K})} z^{-1} + \dot{x}(0) + \frac{1}{2}\ddot{x}(0) + \Pi(\dot{z}(0) + \frac{1}{2}\ddot{z}(0)) \\ A\Delta x = 0,\, A^{\mathrm{T}}\Delta y + \Delta z = 0. \end{cases} \tag{76}$$

Thus, the Mehrotra direction adds a μ-centering component to the second order direction. The value of μ is based on the predictor step length t_P^*. This scheme is similar to the predictor-corrector scheme in [65, 49].

Sturm [66, 68] has proposed a variation of Mehrotra's search direction with an $O(\sqrt{\nu(\mathcal{K})}|\log\epsilon|)$ worst case iteration bound. This polynomial time algorithm has been implemented in SeDuMi [67].

Another extension of Mehrotra's search direction is the centrality correction scheme of Gondzio [6, 27]. For linear programming, the Gondzio correctors have proved their value in practice.

12 Initialization, Infeasibility and Embedding

The interior point method is normally initialized from a *cold start*, which means that no initial starting point is known. In this setting, the interior point method is initialized from a somewhat arbitrarily chosen vector in the interior of the cone $\mathcal{K}$, which may or may not satisfy the linear feasibility constraints. The interior point method should then generate either an approximate primal-dual optimal solution pair, or an approximate Farkas-type dual solution to certify that no (reasonably sized) feasible solution pair exists. In this section, we discuss two alternative approaches to deal with the cold start situation: the infeasible interior point method of Lustig [42, 43] and the self-dual embedding technique of Ye, Todd and Mizuno [85].

We remark that there are situations in which an interior feasible solution is known a priori. Such a solution could be used as an initial solution for a feasible interior point method. However, one typically knows such a solution only for the primal (or only for the dual), and the solution may not be well centered. The known feasible solution may therefore turn out to be of little or no use.

It is more common that a complementary but infeasible solution pair is known a priori. Such a solution pair can provide useful information to a *warm started* interior point method. At the time

of writing, no satisfactory warm started interior point method is known yet. Hence, this subject is not treated in this paper.

Before describing the cold start approaches, we recall that the optimality conditions for (1) and (4) are

$$\begin{aligned} b - Ax &= 0 && (77)\\ A^{\mathrm{T}}y + z - c &= 0 && (78)\\ c^{\mathrm{T}}x - b^{\mathrm{T}}y &\leq 0, && (79) \end{aligned}$$

and

$$x \in \mathcal{K},\ y \in \Re^m,\ z \in \mathcal{K}^*. \tag{80}$$

The Farkas-type conditions to certify that there cannot exist (x, y, z) satisfying (77), (78) and (80) jointly are

$$\begin{aligned} Ax &= 0 && (81)\\ A^{\mathrm{T}}y + z &= 0 && (82)\\ c^{\mathrm{T}}x - b^{\mathrm{T}}y + 1 &= 0, && (83) \end{aligned}$$

together with (80). See e.g. [56, 40, 66] for recent surveys on conic duality.

In fact, we are now in a position to define what we mean by numerically *solving* (1). It means that we should produce an approximate solution to either (77)–(80), or (80)–(83).

The cold started interior point method is initialized from a triple $(x^{(0)}, y^{(0)}, z^{(0)})$ satisfying $\lambda(P(x^{(0)})^{1/2}z^{(0)}) \in \mathcal{N}$. For instance, one may set $x^{(0)} = z^{(0)} = \iota$ and $y^{(0)} = 0$. One also defines

$$y_0^{(0)} = \frac{\nu(\mathcal{K}) + 1}{\nu(\mathcal{K})}; \tag{84}$$

the role of $y_0^{(0)}$ will be clarified later in this section. Given $(x^{(0)}, y_0^{(0)}, y^{(0)}, z^{(0)})$, the initial primal and dual residuals are defined as

$$r_p := \frac{1}{y_0^{(0)}}(b - Ax^{(0)}), \quad r_d := \frac{1}{y_0^{(0)}}(A^{\mathrm{T}}y^{(0)} + z^{(0)} - c). \tag{85}$$

After this initialization, we can proceed either with the infeasible interior point approach or the self-dual embedding approach.

12.1 Infeasible Interior Point Method

In the infeasible interior point method of Lustig [42, 43], one sets $\Delta y_0 = -y_0$ and replaces (15)–(16) by

$$\begin{aligned} A\Delta x &= -\Delta y_0 r_p && (86)\\ A^{\mathrm{T}}\Delta y + \Delta z &= \Delta y_0 r_d. && (87) \end{aligned}$$

Let $y_0(t) = y_0 + t\Delta y_0 = (1-t)y_0$. Solving (14),(86)–(87) yields a search direction with

$$Ax(t) + y_0(t)r_p = b, \quad A^{\mathrm{T}}y(t) + z(t) = c + y_0(t)r_d$$

and

$$x(t)^{\mathrm{T}}z(t) = x^{\mathrm{T}}z + tz^{\mathrm{T}}r + t^2\Delta x^{\mathrm{T}}\Delta z.$$

For the predictor choice '$r = -x$' one has that $x(t)^{\mathrm{T}}z(t) = (1-t)x^{\mathrm{T}}z + O(t^2)$, so that the duality gap reduces locally at the same rate as the infeasibility measure $y_0(t)$. However, the coefficient of the second order term, i.e. $\Delta x^{\mathrm{T}}\Delta z$, is in general nonzero in the infeasible interior point method; in particular, this quantity could be negative. If the duality gap decreases too quickly then the method could conceivably converge to (or get stalled near) an infeasible complementary solution pair. However, by incorporating a centering term in r and by using an appropriate neighborhood or step lengths rule, one can maintain $y_0 = O(x^{\mathrm{T}}z)$. In this way, an approximate solution to (77)–(79) is obtained when $x^{\mathrm{T}}z$ gets close to zero. However, $y_0 = O(x^{\mathrm{T}}z)$ implies that $x^{\mathrm{T}}z$ is bounded from below by a positive quantity when (77)–(79) is strictly infeasible. Therefore, one also defines approximate solutions to (81)–(83) as follows:

$$(\tilde{x}, \tilde{y}, \tilde{z}) := \frac{(x, y, z)}{1 + |b^{\mathrm{T}}y - c^{\mathrm{T}}x|}. \tag{88}$$

If $b^{\mathrm{T}}y - c^{\mathrm{T}}x \to \infty$, an approximate Farkas-type certificate of infeasibility is obtained. We refer to Mizuno [48] for a survey on infeasible interior point methods.

The infeasible interior point technique is widely used in practice, among others in SDPPack [3], SDPA [20] and SDPT3 [76].

12.2 Self-Dual Embedding Technique

In the self-dual embedding technique of Ye et al. [85], a slack variable z_0 is added to (79), and initialized at

$$z_0^{(0)} = \frac{(x^{(0)})^{\mathrm{T}}z^{(0)}}{\nu(\mathcal{K})}. \tag{89}$$

Furthermore, one computes

$$r_g = \frac{c^{\mathrm{T}}x^{(0)} - b^{\mathrm{T}}y^{(0)} + z_0^{(0)}}{y_0^{(0)}}, \quad x_0^{(0)} = 1.$$

The primal and dual problems (1) and (4) are embedded into a self-dual optimization problem

$$\min\{y_0 | (x_0, x, y_0, y, z_0, z) \text{ satisfies } (91), (92), (93)\}, \tag{90}$$

with decision variables x_0, x, y_0, y, z_0 and z, and constraints (91)–(93):

$$\begin{bmatrix} 0 & -A & b \\ A^{\mathrm{T}} & 0 & -c \\ -b^{\mathrm{T}} & c^{\mathrm{T}} & 0 \end{bmatrix} \begin{bmatrix} y \\ x \\ x_0 \end{bmatrix} + \begin{bmatrix} 0 \\ z \\ z_0 \end{bmatrix} = y_0 \begin{bmatrix} r_p \\ r_d \\ r_g \end{bmatrix}, \tag{91}$$

$$r_p^{\mathrm{T}} y + r_d^{\mathrm{T}} x + r_g x_0 = 1, \tag{92}$$

$$(x_0, x) \in \Re_+ \times \mathcal{K},\ (y_0, y) \in \Re^{1+m},\ (z_0, z) \in \Re_+ \times \mathcal{K}^*. \tag{93}$$

Pre-multiplying both sides of (91) with $\left[\begin{array}{ccc} y^{\mathrm{T}}, & x^{\mathrm{T}}, & x_0 \end{array}\right]$ yields the identity

$$x^{\mathrm{T}} z + x_0 z_0 = y_0 (r_p^{\mathrm{T}} y + r_d^{\mathrm{T}} x + r_g x_0) = y_0. \tag{94}$$

It is easily verified that $(x_0^{(0)}, x^{(0)}, y_0^{(0)}, y^{(0)}, z_0^{(0)}, z^{(0)})$ satisfies (91)–(93). Therefore, we can use this as an initial starting point to solve (90) using a feasible interior point method.

Given an interior feasible solution to (91)–(93), we define the normalized solution

$$(\hat{x}, \hat{y}, \hat{z}) := \frac{(x, y, z)}{x_0}, \tag{95}$$

as an approximate solution to (77)–(80). Namely, we have

$$\left\{\begin{array}{l} b - A\hat{x} = (y_0/x_0) r_b \\ A^{\mathrm{T}}\hat{y} + \hat{z} - c = (y_0/x_0) r_c \\ c^{\mathrm{T}}\hat{x} - b^{\mathrm{T}}\hat{y} < (y_0/x_0) r_g. \end{array}\right.$$

When y_0/x_0 approaches zero, the residual to (77)–(79) approaches zero as well.

However, this is not always possible, since the original problem pair (1) and (4) can be infeasible. Therefore, we also define a normalized solution

$$(\tilde{x}, \tilde{y}, \tilde{z}) := \frac{(x, y, z)}{z_0}, \tag{96}$$

as an approximate solution to (80)–(83). Namely, we have

$$\left\{\begin{array}{l} -A\tilde{x} = (y_0 r_b - x_0 b)/z_0 \\ A^{\mathrm{T}}\tilde{y} + \tilde{z} = (y_0 r_c + x_0 c)/z_0 \\ c^{\mathrm{T}}\tilde{x} - b^{\mathrm{T}}\tilde{y} + 1 = y_0 r_g / z_0. \end{array}\right.$$

If, after the final iterations of the interior point method, the residual of $(\tilde{x}, \tilde{y}, \tilde{z})$ with respect to (81)–(83) is smaller than the residual of $(\hat{x}, \hat{y}, \hat{z})$ with respect to (77)–(79), we report the original problem as infeasible, providing $\tilde{x}$ and $\tilde{y}$ as a certificate.

During the interior point process, we can predict whether the original problem pair is infeasible based on the (x_0, z_0) component of the first order predictor direction. Namely, we let

$$\texttt{feas} := \frac{\dot{x}_0(0)}{x_0} - \frac{\dot{z}_0(0)}{z_0}.$$

One can show that if a complementary solution exists then $\dot{x}_0(0)/x_0 \to 0$ and $\dot{z}_0(0)/z_0 \to -1$, so that $\texttt{feas} \to 1$. Conversely, if the problem is strictly infeasible one can show that $\texttt{feas} \to -1$. For problems without a complementary solution which are *not* strictly infeasible, this indicator is less valuable. We refer to Luo, Sturm and Zhang [41] for details.

The self-dual embedding technique is used in MOSEK [7] and SeDuMi [67], among others.

problem	T(build)	T(factor)	T(step)	T(update)	Type
filter48_socp	4%	44%	50%	2%	DENSE, MIX
minphase	7%	22%	64%	6%	PSD
truss5	23%	24%	45%	7%	PSD
truss8	17%	63%	17%	3%	PSD
copo14	4%	91%	4%	0%	PSD
copo23	0%	99%	1%	0%	PSD
nql30new	9%	32%	55%	3%	SOC
nql60new	12%	36%	48%	4%	SOC
nql180new	5%	46%	48%	1%	SOC
qssp30new	14%	34%	47%	6%	SOC
qssp60new	11%	37%	47%	5%	SOC
qssp180new	9%	57%	33%	1%	SOC
nb	59%	13%	27%	1%	SOC
nb_L1	52%	19%	28%	1%	SOC
nb_L2	46%	18%	36%	0%	SOC
nb_L2_bessel	51%	16%	33%	1%	SOC
sched_50_50_scaled	12%	31%	55%	2%	DENSE, SOC
sched_100_50_scaled	16%	33%	49%	2%	DENSE, SOC
sched_100_100_scaled	17%	31%	50%	2%	DENSE, SOC
sched_200_100_scaled	26%	30%	43%	2%	DENSE, SOC
min–max	0%–59 %	13%–99%	1%–64%	0%–7%	

Figure 1: Profile of SeDuMi 1.05 on DIMACS problems. DENSE=dense columns handled, PSD = semidefinite program, SOC = second order cone program, MIX = mixed program.

13 Computational Profile

In linear programming, the primal-dual interior point method has a single bottleneck operation, namely the (numerical) factorization of the normal equations, which has to be repeated in each main iteration. In the implementation phase, one therefore concentrates mainly on ways to speed up the *Factorization Phase*, including the techniques discussed in Sections 7–9; see also [7, 6, 43, 44].

In the literature on semidefinite programming, it has been stated that the real bottleneck may not be the *Factorization Phase*, but the *Building Phase*, i.e. the computations to form $A\Pi A^{\mathrm{T}}$ as discussed in Section 6; see also [21]. Some researchers have also claimed that the NT-scaling cannot be competitive with HKM scaling, since the former requires the computation of a symmetric eigenvalue decomposition for each of the $\kappa(s)$ positive semidefinite blocks, as in discussed in Section 10. This reasoning implicitly assumes that the bottleneck operation is the *Update Phase*, in which the scaling operator Π is updated.

Another phase in each main iteration of the interior point method is the *Step Phase*, in which the search direction $(\Delta x, \Delta y, \Delta z)$ and the step length t are determined. In this phase, we also compute the residual of the computed search direction with respect to the defining equations (14)–(16), and improve the search direction with conjugate gradient steps if needed; such need arises only near the optimal solution set. Most of the operations have to be performed a couple of times, since we use a predictor-corrector scheme to build the search direction; see Section 11. However, since the normal equations have already been factored before we enter the *Step Phase*, the computations in

this phase are often believed to have a minor impact on the total solution time.

Figure 1 shows the computational profile of SeDuMi 1.05 on some problems which were collected for for the Seventh Dimacs Challenge [60]. For each of the four computational phases described above, the percentage of the total computational time spent in this phase is listed. The last row shows the minimum and maximum proportions on this problem set. Most of the problems are pure semidefinite or second order cone problems, denoted by 'PSD' resp. 'SOC' in the last column in Figure 1, but the set also has one mixed semidefinite and second order cone programming problem, namely `filter48_socp`. In some problems, dense columns were detected and handled as in Section 8. These problems are marked by the keyword 'DENSE' in Figure 1. The techniques of Section 9 are not implemented in SeDuMi 1.05. The computations were performed under MATLAB 5.3 on a Pentium-III based system. Details on the actual computational time and the achieved accuracy can be found in [70].

We see that for the problems `copo14` and `copo23`, the *Factorization Phase* is the real bottleneck. However, for most of the problems, the *Building Phase* and the *Step Phase* also account for a substantial proportion of the computational time. Unfortunately, this means that one has to achieve computational savings in all phases of the implemented interior point algorithm in order to improve the computational time considerably. The computational time spent in the factorization, step and update phases can be reduced by using fast linear algebra routines that can be tuned for the specific platform on which the software is installed; this technique is used in CSDP [10]. However, the building step consists entirely of sparse operations. For a numerical comparison of several solvers for semidefinite and second order cone programming, we refer to Mittelmann [47].

14 Concluding Remarks

In the past years, considerable progress has been made in the implementation of the primal-dual interior point method for solving mixed semidefinite and second order cone optimization problems. A detailed and up-to-date account of implementational issues has been given in this paper. Nevertheless, some issues have been omitted from our discussion so-far. We will briefly mention such issues below, and we also provide recommendations for future research.

In this paper, we have concentrated on models that are formulated in the standard form (P) as discussed in Section 2. Unfortunately, this is rarely the most natural form to model a given optimization problem in practice. In particular, practical models may include free variables and rotated second order cone variables. In SeDuMi 1.05, such models are internally transformed in order to comply with the standard form (P). However, it is usually more efficient to avoid such transformations. A nice discussion of handling free variables directly is given in Chapter 19 of Vanderbei [80]. Techniques for handling rotated second order cone variables directly are discussed in Andersen et al. [8].

From a computational point of view, it may also be advantageous to dualize a problem, because the resulting A-matrix has a better structure. Dualization is often considered in the pre-solve phase of solvers for linear programming. Current solvers for mixed semidefinite and second order cone optimization still do not make use of pre-solve techniques. It seems wise to investigate the

possibilities for presolve techniques in this area in the future.

We have restricted to equal step lengths on the primal and dual side of the optimization problem. However, most implementations are these step lengths to be unequal.

We have also not discussed heuristics for generating the initial (x, y, z) in the cold-started framework of Section 12. In fact, SeDuMi 1.05 simply starts from the identity solution $(\iota, 0, iota)$, but other heuristics have been proposed in the literature [7, 6, 43, 45, 76]. Methods for initializing the interior point process from a warm start have not been discussed in this paper either, since the existing approaches are not so satisfactory.

We have made an attempt to compare the implications of different types of primal-dual scaling in the implementation. We have concentrated on AHO, NT and the two forms of HKM scaling. The main disadvantage of AHO scaling is that it can not benefit from sparsity in the *Building Phase* of the algorithm. However, the continuous affine scaling trajectories have a nice analyticity property that allows for fast local convergence near the optimal solution set [62]. SeDuMi also seems to achieve fast local convergence on many problems with the NT scaling. It can be interesting to study the asymptotic behavior of continuous affine scaling trajectories with different types of scaling more elaborately.

Acknowledgment I like to thank Bob Vanderbei for inviting me to write this paper in early 2000. Unfortunately, my preparation time outlived his editorship of *INFORMS Journal on Computing*. I am therefore also grateful to Tamás Terlaky for suggesting in 2002 to contribute the then still unfinished paper to this special issue of *Optimization Methods and Software*.

References

[1] I. Adler, N.K. Karmarkar, M.G.C. Resende, and G. Veiga. An implementation of Karmarkar's algorithm for linear programming. *Mathematical Programming*, 44:297–335, 1989.

[2] F. Alizadeh. Interior point methods in semidefinite programming with applications to combinatorial optimization problems. *SIAM Journal on Optimization*, 5:13–51, 1995.

[3] F. Alizadeh, J.A. Haeberly, M.V. Nayakkankuppann, M. Overton, and S. Schmieta. *SDPPack user's guide*. New York University, New York, USA, 1997.

[4] F. Alizadeh, J.A. Haeberly, and M. Overton. A new primal–dual interior point method for semidefinite programming. In J.G. Lewis, editor, *Proceedings of the Fifth SIAM Conference on Applied Linear Algebra*, pages 113–117. SIAM, Philadelpha, 1994.

[5] F. Alizadeh, J.A. Haeberly, and M. Overton. Primal–dual interior–point methods for semidefinite programming: convergence rates, stability and numerical results. *SIAM Journal on Optimization*, 8(3):746–768, 1998.

[6] E. Andersen, J. Gondzio, C. Mész aros, and X. Xu. Implementation of interior point methods for large scale linear programming. In T. Terlaky, editor, *Interior point methods for mathematical programming*, pages 189–252. Kluwer Adademic Publishers, 1996.

[7] E.D. Andersen and K.D. Andersen. The MOSEK interior point optimizer for linear programming: an implementation of the homogeneous algorithm. In J.B.G. Frenk, C. Roos, T. Terlaky, and S. Zhang, editors, *High Performance Optimization*, pages 441–466. Kluwer Academic Publishers, 2000.

[8] E.D. Andersen, C. Roos, and T. Terlaky. On implementing a primal-dual interior point method for conic quadratic optimization. *Mathematical Programming*, 2002. To appear.

[9] K.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.

[10] B. Borchers. CSDP, a C library for semidefinite programming. Technical report, New Mexico Tech Mathematics Faculty, USA, 1997.

[11] S. Boyd, L. El Ghaoui, E. Feron, and V. Balakrishnan. *Linear matrix inequalities in system and control theory*, volume 15 of *Studies in Applied Mathematics*. SIAM, Philadelphia, PA, 1994.

[12] J.E. Dennis and R.B. Schnabel. *Numerical Methods for Unconstrained Optimization and Nonlinear Equations*. Prentice-Hall, 1983.

[13] R.J. Duffin. Infinite programs. In H.W. Kuhn and A.W. Tucker, editors, *Linear Inequalities and Related Systems*, pages 157–170. Princeton University Press, Princeton, NJ, 1956.

[14] J. Faraut and A. Korányi. *Analysis on Symmetric Cones*. Oxford Mathematical Monographs. Oxford University Press, New York, 1994.

[15] L. Faybusovich. Jordan algebras, symmetric cones and interior point methods. Technical report, Department of Mathematics, University of Notre Dame, Notre Dame, IN, USA, 1995.

[16] L. Faybusovich. Euclidean Jordan algebras and interior–point algorithms. In *Positivity 1*, pages 331–357. Kluwer Academic Publishers, 1997.

[17] L. Faybusovich. Linear systems in Jordan algebras and primal–dual interior–point algorithms. *Journal of Computational and Applied Mathematics*, 86:149–175, 1997.

[18] L. Faybusovich. A Jordan–algebraic approach to potential–reduction algorithms. Technical report, Department of Mathematics, University of Notre Dame, Notre Dame, IN, USA, 1998.

[19] R.M. Freund and S. Mizuno. Interior point methods: Current status and future directions. In J.B.G. Frenk, C. Roos, T. Terlaky, and S. Zhang, editors, *High Performance Optimization*, pages 441–466. Kluwer Academic Publishers, 2000.

[20] K. Fujisawa, M. Fukuda, M. Kojima, and K. Nakata. Numerical evaluation of SDPA (semidefinite programming algorithm). In J.B.G. Frenk, C. Roos, T. Terlaky, and S. Zhang, editors, *High Performance Optimization*, pages 267–301. Kluwer Academic Publishers, 2000.

[21] K. Fujisawa, M. Kojima, and K. Nakata. Exploiting sparsity in primal-dual interior-point methods for semidefinite programming. *Mathematical Programming*, 79:235–253, 1997.

[22] A. George and J. Liu. The evolution of the minimum degree ordering algorithm. *SIAM Review*, 31:1–19, 1989.

[23] P.E. Gill, W. Murray, and M.H. Wright. *Practical Optimization*. Academic Press, London, 1981.

[24] M.X. Goemans and D.P. Williamson. Improved approximation algorithms for maximum cut and satisfyability problems using semidefinite programming. *Journal ACM*, 42:1115–1145, 1995.

[25] D. Goldfarb and K. Scheinberg. A product form Cholesky factorization method for handling dense columns in interior point methods for linear programming. Technical report, Department of IEOR, Columbia University, New York, NY, USA, 2001.

[26] D. Goldfarb, K. Scheinberg, and S. Schmieta. A product form Cholesky factorization implementation of an interior point method for second order cone programming. Technical report, Department of IEOR, Columbia University, New York, NY, USA, 2001.

[27] J. Gondzio. Multiple centrality corrections in a primal-dual method for linear programming. *Computational Optimization and Applications*, 6:137–156, 1996.

[28] C.C. Gonzaga. Path-following methods for linear programming. *SIAM Review*, 34(2):167–224, 1992.

[29] O. Güler and L. Tunçel. Characterization of the barrier parameter of homogeneous convex cones. *Mathematical Programming*, 81:55–76, 1998.

[30] C. Helmberg, F. Rendl, R.J. Vanderbei, and H. Wolkowicz. An interior–point method for semidefinite programming. *SIAM Journal on Optimization*, 6:342–361, 1996.

[31] R.A. Horn and C.R. Johnson. *Matrix Analysis*. Cambridge University Press, Cambridge, New York, 1985. Corrected reprint 1990.

[32] N.K. Karmarkar. A new polynomial time algorithm for linear programming. *Combinatorica*, 4:373–395, 1984.

[33] E. de Klerk. *Interior Point Methods for Semidefinite Programming*. PhD thesis, Delft University of Technology, Faculty of Technical Mathematics and Informatics, Delft, The Netherlands, 1997.

[34] M. Kojima, N. Megiddo, T. Noma, and A. Yoshise. *A Unified Approach to Interior Point Algorithms for Linear Complementarity Problems*. Springer-Verlag, Berlin, 1991.

[35] M. Kojima, S. Mizuno, and A. Yoshise. A primal–dual interior point algorithm for linear programming. In N. Megiddo, editor, *Progress in mathematical programming: interior point and related methods*, pages 29–37. Springer-Verlag, New York, 1989.

[36] M. Kojima, M. Shida, and S. Shindoh. A predictor–corrector interior–point algorithm for the semidefinite linear complementarity problem using the alizadeh–haeberly–overton search direction. Technical Report B–311, Dept. of Information Sciences, Tokyo Institute of Technology, 2-12-1 Oh-Okayama, Meguro-ku, Tokyo 152, Japan, 1996.

[37] M. Kojima, S. Shindoh, and S. Hara. Interior–point methods for the monotone semidefinite linear complementarity problem in symmetric matrices. *SIAM Journal on Optimization*, 7(1):86–125, 1997.

[38] S. Kruk. *High Accuracy Algorithms for the Solutions of Semidefinite Linear Programs*. PhD thesis, University of Waterloo, Waterloo, Ontario, Canada, February 2001.

[39] M.S. Lobo, L. Vandenberghe, S. Boyd, and H. Lebret. Applications of second–order cone programming. *Linear Algebra and its Applications*, 284:193–228, 1998. Special Issue on Linear Algebra in Control, Signals and Image Processing.

[40] Z.-Q. Luo, J.F. Sturm, and S. Zhang. Duality and self-duality for conic convex programming. Technical Report 9620/A, Econometric Institute, Erasmus University Rotterdam, Rotterdam, The Netherlands, 1996.

[41] Z.-Q. Luo, J.F. Sturm, and S. Zhang. Conic convex programming and self-dual embedding. *Optimization Methods and Software*, 14:196–218, 2000.

[42] I.J. Lustig. Feasibility issues in primal-dual interior point methods for linear programming. *Mathematical Programming*, 49:145–162, 1990.

[43] I.J. Lustig, R.E. Marsten, and D.F. Shanno. Computational experience with a primal–dual interior point method for linear programming. *Linear Algebra and its Applications*, 152:191–222, 1991.

[44] I.J. Lustig, R.E. Marsten, and D.F. Shanno. On implementing Mehrotra's predictor–corrector interior point method for linear programming. *SIAM Journal on Optimization*, 2:435–449, 1992.

[45] I.J. Lustig, R.E. Marsten, and D.F. Shanno. Interior point methods for linear programming: computational state of the art. *ORSA Journal on Computing*, 6(1):1–14, 1994.

[46] S. Mehrotra. On the implementation of a primal–dual interior point method. *SIAM Journal on Optimization*, 2:575–601, 1992.

[47] H.D. Mittelmann. Independent benchmark results. `http://plato.la.asu.edu/dimacs.html`, 2001.

[48] S. Mizuno. Infeasible interior point algorithms. In T. Terlaky, editor, *Interior point methods of mathematical programming*, volume 5 of *Applied Optimization*. Kluwer Academic Publishers, Dordrecht, The Netherlands, 1996. Series editors P.M. Pardalos and D. Hearn.

[49] S. Mizuno, M.J. Todd, and Y. Ye. On adaptive-step primal-dual interior-point algorithms for linear programming. *Mathematics of Operations Research*, 18:964–981, 1993.

[50] R.D.C. Monteiro. Primal–dual path following algorithms for semidefinite programming. *SIAM Journal on Optimization*, 7(3):663–678, 1997.

[51] R.D.C Monteiro, I. Adler, and M.G.C. Resende. A polynomial–time primal–dual affine scaling algorithm for linear and convex quadratic programming and its power series extension. *Mathematics of Operations Research*, 15:191–214, 1990.

[52] R.D.C. Monteiro and T. Tsuchiya. Polynomial convergence of a new family of primal–dual algorithms for semidefinite programming. Technical report, School of Industrial and Systems Engineering, Georgia Tech, Atlanta, Georgia, U.S.A., 1996.

[53] R.D.C. Monteiro and P.R. Zanjácomo. A note on the existence of the Alizadeh–Haeberly–Overton direction for semidefinite programming. Technical report, School of Industrial and Systems Engineering, Georgia Tech, Atlanta, Georgia, U.S.A., 1996. Revised January, 1997.

[54] R.D.C. Monteiro and P.R. Zanjácomo. General interior-point maps and existence of weighted paths for nonlinear semidefinite complementarity problems. *Mathematics of Operations Research*, 25:381–399, 2000.

[55] R.D.C. Monteiro and Y. Zhang. A unified analysis for a class of path–following primal–dual interior point algorithms for semidefinite programming. *Mathematical Programming*, 81:281–299, 1998.

[56] Y. Nesterov and A. Nemirovsky. *Interior point polynomial methods in convex programming*, volume 13 of *Studies in Applied Mathematics*. SIAM, Philadelphia, 1994.

[57] Y. Nesterov and M.J. Todd. Self–scaled barriers and interior–point methods for convex programming. *Mathematics of Operations Research*, 22(1):1–42, 1997.

[58] Y. Nesterov and M.J. Todd. Primal–dual interior–point methods for self–scaled cones. *SIAM Journal on Optimization*, 8:324–364, 1998.

[59] J. Nocedal and S.J. Wright. *Numerical Optimization*. Springer-Verlag, New York, 1999.

[60] G. Pataki and S. Schmieta. The DIMACS library of semidefinite-quadratic-linear programs. Technical report, Computational Optimization Research Center, Columbia University, New York, NY, USA, 1999. `http://dimacs.rutgers.edu/Challenges/Seventh/Instances/`.

[61] L. Portugal, F. Bastos, J. Judice, J. Paxiao, and T. Terlaky. An investigation of interior point algorithms for the linear transportation problem. *SIAM Journal on Scientific Computing*, 17(5):1202–1223, 1996.

[62] M. Preiß and J. Stoer. Analysis of infesible interior point paths arising with semidefinite linear complementarity problems. Technical report, Institut für Angewandte Mathematik und Statistik, Universität Würzburg, Würzburg, Germany, 2002.

[63] M.A. Saunders. Major cholesky would feel proud. *ORSA Journal on Computing*, 6(1):23–27, 1994.

[64] M. Shida, S. Shindoh, and M. Kojima. Existence and uniqueness of search directions in interior–point algorithms for the SDP and the monotone SDLCP. *SIAM Journal on Optimization*, 8(2):387–396, 1998.

[65] G. Sonnevend, J. Stoer, and G. Zhao. On the complexity of following the central path for linear programs by linear extrapolation. *Methods of Operations Research*, 63:19–31, 1989.

[66] J.F. Sturm. *Primal–Dual Interior Point Approach to Semidefinite Programming*, volume 156 of *Tinbergen Institute Research Series*. Thesis Publishers, Amsterdam, The Netherlands, 1997.

[67] J.F. Sturm. Using SeDuMi 1.02, a MATLAB toolbox for optimization over symmetric cones. *Optimization Methods and Software*, 11–12:625–653, 1999. Special issue on Interior Point Methods (CD supplement with software).

[68] J.F. Sturm. Central region method. In J.B.G. Frenk, C. Roos, T. Terlaky, and S. Zhang, editors, *High Performance Optimization*, pages 157–194. Kluwer Academic Publishers, 2000.

[69] J.F. Sturm. Similarity and other spectral relations for symmetric cones. *Linear Algebra and its Applications*, 312:135–154, 2000.

[70] J.F. Sturm. Avoiding numerical cancellation in the interior point method for solving semidefinite programs. Technical Report 2001-27, CentER, P.O.Box 90153, 5000LE Tilburg, The Netherlands, 2001. To appear in *Mathematical Programming*.

[71] J.F. Sturm and S. Zhang. On a wide region of centers and primal–dual interior point algorithms for linear programming. *Mathematics of Operations Research*, 22(2):408–431, 1997.

[72] J.F. Sturm and S. Zhang. Symmetric primal–dual path following algorithms for semidefinite programming. *Applied Numerical Mathematics*, 29:301–315, 1999.

[73] J.F. Sturm and S. Zhang. On weighted centers for semidefinite programming. *European Journal of Operational Research*, 126:391–407, 2000.

[74] M.J. Todd. A study of search directions in interior–point methods for semidefinite programming. *Optimization Methods and Software*, 11–12:1–46, 1999.

[75] M.J. Todd, K.C. Toh, and R.H. Tütüncü. On the Nesterov–Todd direction in semidefinite programming. *SIAM Journal on Optimization*, 8(3):769–796, 1998.

[76] K.C. Toh, M.J. Todd, and R.H. Tütüncü. SDPT3 - a MATLAB package for semidefinite programming. version 1.3. *Optimization Methods and Software*, 11–12:545–581, 1999. Special issue on Interior Point Methods (CD supplement with software).

[77] L. Tunçel. Primal–dual symmetry and scale invariance of interior point algorithms for convex optimization. *Mathematics of Operations Research*, 23, 1998.

[78] L. Vandenberghe and S. Boyd. *SP: Software for semidefinite programming.* Information Systems Laboratory, Electrical Engineering Department, Stanford University, Stanford, USA, 1994.

[79] L. Vandenberghe and S. Boyd. A primal–dual potential reduction method for problems involving matrix inequalities. *Mathematical Programming*, 69:205–236, 1995.

[80] R.J. Vanderbei. *Linear Programming. Foundations and Extensions.* Kluwer Adademic Publishers, Boston, USA, 1997.

[81] Henry Wolkowicz, Romesh Saigal, and Lieven Vandenberghe, editors. *Handbook on Semidefinite Programming.* Kluwer Academic Publishers, 2000.

[82] S. Wright and Y. Zhang. A superquadratic infeasible-interior-point method for linear complementarity problems. *Mathematical Programming*, 73:269–289, 1996.

[83] S.J. Wright. Modified Cholesky factorizations in interior point algorithms for linear programming. Technical Report ANL/MCS-P600-0596, Argonne National Laboratory, Argonne, IL, 1996.

[84] X. Xu, P. Hung, and Y. Ye. A simplified homogeneous and self–dual linear programming algorithm and its implementation. *Annals of Operations Research*, 62:151–172, 1996.

[85] Y. Ye, M.J. Todd, and S. Mizuno. An $O(\sqrt{n}L)$-iteration homogeneous and self–dual linear programming algorithm. *Mathematics of Operations Research*, 19:53–67, 1994.

[86] Y. Zhang. On extending some primal–dual interior–point algorithms from linear programming to semidefinite programming. *SIAM Journal on Optimization*, 8(2):365–386, 1998.