On Implementation of a Self-Dual Embedding Method for Convex Programming

By
Cheng Tak Wai, Johnny

August, 2003

Supervised By
Prof. Zhang Shuzhong

A Thesis
submitted in partial fulfillment of the requirements
for the degree of Master of Philosophy
Division of Systems Engineering and Engineering Management
The Chinese University of Hong Kong

摘要

在此篇论文中，我们研究张树中提出的一个求解非线性最优化问题的方法。其方法将一般的凸最优化问题转换为锥最优化问题。之后，使用自对偶齐次化嵌入技术(self-dual embedding technique)来解答此转换后的问题。其方法最大的好处乃为其使用者不需要知道问题的可行状态，而可直接求解。我们采用Anderson和叶荫宇的方法，建立优点函数(merit function)用来选定步距。我们设计一些测试问题来研究我们提出的算法。考虑的测试问题包括了使用对数，指数与二次函数的约束条件。然后，我们讨论问题的规模与算法求解时间的关系。我们特别研究算法对几何规划问题的求解效果及其数值测试结果。最后结论是我们提出的算法在一般情况下数值效果良好，表现稳定。乃为求解一般最优化问题的可行及有效的计算方法。

[**关键字**]: 凸规划　凸锥　自对偶齐次化嵌入技术　Path-Following　几何规划

Abstract

In this thesis, we implement Zhang's method [43], which transforms a general convex optimization problem with smooth convex constraints into a convex conic optimization problem and then apply the technique of self-dual embedding for solving the resulting conic problem optimization. A crucial advantage of the approach is that no initial solution is required, and the method is particularly suitable when the feasibility status of the problem is unknown. In our implementation, we use a merit function approach proposed by Andersen and Ye [1] to determine the step size along the search direction. We investigate the efficiency of the proposed algorithm based on its performance on some test problems, which include logarithmic functions, exponential functions and quadratic functions in the constraints. We discuss the relationship between the size of the problem and the solution time. Furthermore, we consider in particular the geometric programming problems. Numerical results of our algorithm on this class of optimization problems are reported. We conclude that the method is stable, efficient and easy-to-use in general.

Keywords: Convex Programs, Convex Cones, Self-Dual Embedding, Path-Following, Geometric programs.

Acknowledgments

I am grateful to my supervisor: Prof. Zhang Shuzhong, for directing me to the field of conic optimization during the last two years. His inspiring guidance, valuable suggestions and continuous encouragement made it possible for me to finish this thesis. Also, I would like to thanks to Prof. Li Duan and Prof. Zhou Xunyu for their comments on my research.

Thanks to all technical and administration staffs in department of System Engineering and Engineering Management, CUHK. Their helpfulness provide me a congenial working environment.

Moreover, I would like to thanks to all of my school fellows, especially Jin Hanqing, Xie Jiang, Chiu Chung-Hung, Ah Fan, Alvin, Rachel and Paul. They always support me and make my school life enjoyable.

Apart from the above people, I would also like to give my great appreciation to my parents and sisters. They give me endless support so that I could complete this thesis.

Again, thanks all of the above people. Without them, I never could complete my thesis successfully.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

In 1979, Khacijan [21] showed that the ellipsoid algorithm applied to solve the linear programming problem would run in polynomial time. This result was not only important for the complexity status of linear programming, but also many other combinatorial optimization problems, as shown by Grotschel et al. [16].

Unfortunately, an algorithm with polynomial computational complexity does not necessarily lead to satisfactory computational efficiency in practice. Such is the case with the ellipsoid method; the method remains mainly a theoretical tool. The introduction of polynomial-time interior-point methods is one of the most remarkable events in the development of mathematical programming in the 1980s.

The first interior-point algorithm was introduced for linear programming in a landmark paper of Karmarker (see [19]). The computational complexity result contained in that paper, as well as the claim that performance of the new method on real-world problems is significantly better than that of the simplex method, made Karmarkar's work a sensation and subsequently

inspired very intensive and fruitful studies. Karmarkar's paper triggered a tremendous amount of research on what is now commonly called the interior point methods. Hundreds of researchers all over the world went into the subject; over 2000 papers were written (see Kranich [24] for a bibliography).

Related to the Karmarkar algorithm, an important techniques is the so-called logarithmic barrier method, introduced by Frisch [11] in 1955. Later, Barnes [5] and Vanderbei et. al. [38] proposed the so-called affine scaling algorithm as a simplified version of Karmarkar's method, which turned out to be just a rediscovery of a method developed by Dikin [9] in 1967.

Reasons can be given why such type of methods were out of fashion since early 1970s, but regained so much interest in the mathematical programming society after Karmarkar's work. First, it is of theoretical significance. Interior point techniques were originally developed to solve nonlinear programming problems (NLP) with inequality constraints. For LP the simplex method performed reasonably well, and there was no incentive to investigate the theoretical properties of the interior methods when applied to LP, as theoretical complexity of the algorithm was not regarded to be an issue in the 1960s.

In fact it was only around 1970 that complexity theory was developed, mainly in the field of combinatorial optimization (see Karp [20] and Garey and Johnson [12]), and for convex optimization by Judin and Nemirovskii [18]. It was shown by Klee and Minty [22] that certain variants of the simplex method require, in the worst case, an exponential number of arithmetic

operations. Since then, the search for a polynomial method being efficient in practice was a challenge, without considering the possibility that existing methods, when sufficiently adjusted, could satisfy these requirements.

Shortly after the publication of Karmarkar's paper, Gill et. al. [13] showed that Karmarkar's projective algorithm was closely related to the logarithmic barrier method. Following this connection, theoretical work on interior point methods soon led to the introduction of the analytic center by Sonnevend [36] and analysis of the central path in a primal-dual setting by Megiddo [28] which are the central themes in both theoretical work as well as in practical implementations of interior point techniques.

In 1987, Roos and Vial [35] derived a very elegant and simple complexity proof of the basic logarithmic barrier method, showing a new property of an essentially old method. Renegar [33] derived the complexity of a method using analytic centers which can be traced back to Huard [17]. Anstreicher [4] analyzed SUMT [31], an old implementation of an interior point method and showed it to be polynomial.

Immediately after Karmarkar's paper, the activity in the field of interior-point methods focused mainly on linear programs. Later, it was discovered that the nature of the method is in fact independent of the specific properties of the LP problems; these methods can be extended to solve more general convex programs. This led to the revival of Newton's method for convex programming with a beautiful analysis of certain interior point methods for convex programming by Nesterov and Nemirovsky [32]. These significant

results include the study of the so-called semi-definite programming (SDP) and its applications, e.g., in control theory and combinatorial optimization (see Boyd el. al. [7, 39], Alizadeh [3]), and the development of efficient practical algorithm for NLP problems (see Yamashita [42], Vial [40]).

Convex optimization relates to a class of nonlinear optimization problems where the objective to be minimized, and the constraints are convex. Convex optimization problems are attractive because a large class of these problems can now be efficiently solved. However, the difficulty is often to recognize the convexity; convexity is harder to recognize than say, linearity.

One important features of convexity is that it is possible to address hard, non convex problems (such as "combinatorial optimization" problems) using convex approximations that are more efficient than classical linear ones. Convex optimization is especially relevant when the data of the problem at hand is uncertain, and "robust" solutions are sought.

In addition, convex optimization problems are known since 1960s. This kind problem has a nice property which is that the local optimal are the global optimal. Moreover, the convex analysis is well developed by 1970s (Rockafellar) such as separating, supporting hyperplanes, and sub-gradient calculus. In 1990s, powerful primal-dual interior-point methods extremely efficient, handle nonlinear large scale problems.

Up till this day, it is generally believed that the primal-dual interior point methods, such as the one introduced by Kojima et al. [23], are among the most efficient methods for sloving linear programming problems. The general

principle of primal-dual interior point method is based on sequentially solving a certain perturbed Karush-Kuhn-Tucker (KKT) system. In its original form, the primal-dual interior point method requires the availability of some initial primal-dual strongly feasible solutions.

In 1994, Ye, Todd and Mizuno [30] introduced the so-called homogeneous self-dual embedding technique to solve linear programs. The main idea of this technique is to construct an artificial problem by embedding a primal-dual problem pair. When this artificial problem is solved, the original primal and dual problems are automatically solved. There are several nice features of this method. The most important advantage is that no initial solution is required to start the algorithm: trivial initial solution is available for the artificially constructed self-dual embedded problem. It also solves the problem in polynomial time without using any 'Big-M' type constants.

The idea of the interior-point method was further extended to solve general convex optimization problems. The most important work is probably the so-call self-concordant barrier theory developed by Nesterov and Nemirovskii [32]. Based on this theory, the interior-point methods can be applied to efficiently solve several important classes of convex optimization problems, where the self-concordant barrier functions are known.

The idea of self-dual embedding was also extended to solve more general constrained convex optimization problems. Andersen and Ye [1, 2], developed a special type of self-dual embedding model based on the simplified model of Xu, Hung and Ye [41] for linear programming; Luo, Sturm and Zhang [27]

proposed a self-dual embedding model for conic optimization.

The purpose of this thesis is to specialize an interior-point implementation for a new self-dual embedding method, proposed by Zhang [43], for convex programming problems. Moreover, we study the efficiency of the proposed algorithm for solving numerous test problems.

The outline of this thesis is as follows. In Chapter 2, we propose a particular conic formulation for the inequality-constrained convex program, and construct a self-dual embedding model for solving the resulting conic optimization problem. Then we develop an interior-point algorithm to solve this model in Chapter 3. We present some numerical results in Chapter 4 for solving randomly generated test problems which involved logarithmic, exponential and quadratic constraints. According to the numerical results, we study the relationship between the running time and the size of the problem in the same chapter. Next, we consider geometric optimization problems in Chapter 5. We first state the general primal and dual form of the problem and then transform the problem into the conic form. After that we solve the conic model by our algorithm. Computational results of some existing test problems are presented in the same chapter. Finally, in the last chapter, Chapter 6, we conclude the whole thesis.

Chapter 2

Background

In this chapter, we shall review the fundamentals of conic optimization and the self-dual embedding technique. The focus will be placed on a recent paper of Zhang, [43], entitled "A New Self-Dual Embedding Method for Convex Programming".

2.1 Self-dual embedding

The primal and dual liner programming (LP) problems are as foliows. Let A be a $m \times n$ matrix, b be a $m \times 1$ vector and c be a $n \times 1$ vector. Then the primal LP problem is

$$\begin{array}{lll} (LP) & \min & c^T x \\ & \text{s.t.} & Ax = b \\ & & x \geq 0 \end{array}$$

and its dual problem is

$$\begin{array}{lll} (LD) & \max & b^T y \\ & \text{s.t.} & A^T y + s = c \\ & & s \geq 0. \end{array}$$

Take any $x^0 > 0$, $s^0 > 0$, and $y^0 \in \Re^m$. Moreover, define

$r_p = b - Ax^0$, $r_d = s^0 - c + A^T y^0$, $r_g = 1 + c^T x^0 - b^T y^0$, and $\beta = 1 + x^{0T} s^0 > 1$.

Following Mizuno, Todd and Ye [30], we can combine the primal and dual linear problems together to form a self-dual embedded LP model:

$$\begin{array}{lccccccc} \min & & & \beta\theta & & & & \\ \text{s.t.} & Ax & -b\tau & +r_p\theta & & & = & 0 \\ -A^Ty & & +c\tau & +r_d\theta & -s & & = & 0 \\ b^Ty & -c^Tx & & +r_g\theta & & -\kappa & = & 0 \\ -r_p^Ty & -r_d^Tx & -r_g\tau & & & & = & -\beta \\ & x \geq 0, & \tau \geq 0, & & s \geq 0, & \kappa \geq 0. & & \end{array} \tag{2.1}$$

It is easy to find a feasible starting point for this problem. Indeed, one can check that $(x, y, s, \tau, \kappa, \theta) = (x_0, y_0, s_0, 1, 1, 1)$ is a suitable choice. This program is self-dual. meaning that its dual is identical to itself. Moreover, the optimal value is 0.

2.2 Conic optimization

In this section, we shall introduce conic optimization. We first state the definition of a cone.

Definition 2.1 *A set $\mathcal{K} \neq \emptyset$ is a cone if and only if the following holds*

$$\forall x \in \mathcal{K} \Rightarrow \lambda x \in \mathcal{K} \quad \forall \lambda \in \Re_+.$$

We are now in a position to define a conic optimization problem. Let $\mathcal{K} \subseteq \Re^n$ be a closed convex cone. The primal conic optimization problem is

$$\begin{array}{lll} (CP) & \min & c^Tx \\ & \text{s.t.} & Ax = b \\ & & x \in \mathcal{K} \end{array}$$

where $x \in \Re^n$ is the decision variable and the problem data are given by the cone $\mathcal{K}$, an $m \times n$ matrix A, and two vectors b and c. The vectors b and c belong to $\Re^m$ and $\Re^n$ respectively.

Definition 2.2 *The dual of a cone $\mathcal{K} \subseteq \Re^n$ is defined by*

$$\mathcal{K}^* = \{s \mid x^T s \geq 0 \ \forall x \in \mathcal{K}\}.$$

The dual of the primal conic problem (CP) is given as

$$\begin{array}{lll} (CD) & \max & b^T y \\ & \text{s.t.} & A^T y + s = c \\ & & s \in \mathcal{K}^* \end{array}$$

where $y \in \Re^m$ and $s \in \Re^n$ are the decision variables.

2.3 Self-dual embedded conic optimization

We can now use the idea of self-dual embedding in linear programming to construct the self-dual embedded conic optimization model. Take any $x^0 \in$ int $\mathcal{K}$, $s^0 \in$ int $\mathcal{K}^*$, and $y^0 \in \Re^m$. Moreover, define

$$r_p = b - Ax^0,\ r_d = s^0 - c + A^T y^0,\ r_g = 1 + c^T x^0 - b^T y^0,\ \text{and}\ \beta = 1 + x^{0T} s^0 > 1,$$

and consider the following embedded optimization model

$$\begin{array}{lccccccc} \min & & & & \beta\theta & & & \\ \text{s.t.} & & Ax & -b\tau & +r_p\theta & & = & 0 \\ & -A^T y & & +c\tau & +r_d\theta & -s & = & 0 \\ & b^T y & -c^T x & & +r_g\theta & -\kappa & = & 0 \\ & -r_p^T y & -r_d^T x & -r_g\tau & & & = & -\beta \\ & & x \in \mathcal{K}, & \tau \geq 0, & & s \in \mathcal{K}^*, \ \kappa \geq 0 & & \end{array} \tag{2.2}$$

where the decision variables are $(y, x, \tau, \theta, s, \kappa)$.

The following result is well known; see its analog in [30] for the linear programming case.

Proposition 2.1 *Problem (2.1) has a maximally complementary optimal solution, denoted by* $(y^*, x^*, \tau^*, \theta^*, s^*, \kappa^*)$, *such that* $\theta^* = 0$ *and* $(x^*)^T s^* + \tau^* \kappa^* = 0$. *Moreover, if* $\tau^* > 0$, *then* x^*/τ^* *is an optimal solution for* (CP), *and* $(y^*/\tau^*, s^*/\tau^*)$ *is an optimal solution for* (CD). *If* $\kappa^* > 0$ *then either* $c^T x^* < 0$ *or* $b^T y^* > 0$; *in the former case* (CD) *is infeasible, and in the latter case* (CP) *is infeasible.*

If τ^* and κ^* do not exhibit strict complementarity, namely $\tau^* = \kappa^* = 0$, then in that case we can only conclude that (CP) and (CD) do not have complementary optimal solutions. In fact, more information can still be deduced, using the notion of, e.g., *weak infeasibility*; for more details see [27], [26] and [37].

We now introduce a ν-logarithmically homogeneous barrier function $F(x)$ for the cone $\mathcal{K}$, i.e.

$$F(tx) = F(x) - \nu \log t$$

for all $x \in \text{int}\,\mathcal{K}$ and $t > 0$. Suppose that $F(x)$ is a ν-logarithmically homogeneous barrier function for $\mathcal{K}$, then $F^*(s)$ is a ν-logarithmically homogeneous barrier function for $\mathcal{K}^*$; see Nesterov and Nemirovski [32] for details. In addition, $F^*(s)$ is the conjugate of the convex function $F(x)$. Hence, we can solve (2.2) by the barrier approach with $\mu > 0$ as the barrier parameter, namely,

$$\begin{array}{lccccccc}
\min & & \mu F(x) & -\mu \log \tau & +\beta\theta & +\mu F^*(s) & -\mu \log \kappa & \\
\text{s.t.} & & Ax & -b\tau & +r_p\theta & & & = 0 \\
 & -A^T y & & +c\tau & +r_d\theta & -s & & = 0 \\
 & b^T y & -c^T x & & +r_g\theta & & -\kappa & = 0 \\
 & -r_p^T y & -r_d^T x & -r_g\tau & & & & = -\beta.
\end{array}$$

2.4 Connection with convex programming

We would like to use the above idea to formulate a convex program into a conic optimization problem. First of all, let us state a general convex program as follows:

$$\begin{array}{lll} (P) & \min & c^T x \\ & \text{s.t.} & Ax = b \\ & & f_i(x) \leq 0,\ i = 1, ..., m \end{array} \tag{2.3}$$

where $f_i(x)$ is smooth and convex, $i = 1, ..., m$. We consider the case $m = 1$ to explain the transformation. Let the decision variable be

$$\bar{x} := \begin{bmatrix} p \\ q \\ x \end{bmatrix} \in \Re^1 \times \Re^1 \times \Re^n,$$

and the problem data are

$$\bar{c} := \begin{bmatrix} 0 \\ 0 \\ c \end{bmatrix} \in \Re^1 \times \Re^1 \times \Re^n,$$

$$\bar{b} := \begin{bmatrix} 1 \\ 0 \\ b \end{bmatrix} \in \Re^1 \times \Re^1 \times \Re^m$$

and

$$\bar{A} := \begin{bmatrix} 1 & 0 & 0^T \\ 0 & 1 & 0^T \\ 0 & 0 & A \end{bmatrix} \in \Re^{(m+2)\times(n+2)}.$$

Let

$$\mathcal{K} = \text{cl}\,\{\bar{x} \mid p > 0,\ q - pf(x/p) \geq 0\}$$

which is a closed cone. The following Lemma shows that $\mathcal{K}$ is a closed cone.

Lemma 1 *The function $-q + pf(x/p)$ is convex in $\Re^1_{++} \times \Re^1 \times \Re^n$.*

Proof. We only need to show that $pf(x/p)$ is convex in $\Re^1_{++} \times \Re^n$. By simple calculation, it follows that

$$\nabla^2(pf(x/p)) = \frac{1}{p}\begin{bmatrix} (x/p)^T\nabla^2 f(x/p)(x/p) & -(x/p)^T\nabla^2 f(x/p) \\ -\nabla^2 f(x/p)(x/p) & \nabla^2 f(x/p) \end{bmatrix}.$$

Let $H = \nabla^2 f(x/p)$ and $h = x/p$. Then for any $\bar{\xi}^T = (\xi_0,\ \xi^T) \in \Re^{n+1}$, we have

$$\begin{aligned} \bar{\xi}^T\nabla^2(pf(x/p))\bar{\xi} &= \tfrac{1}{p}\left[\xi_0^2 h^T H h - 2\xi_0\xi^T H h + \xi^T H \xi\right] \\ &\geq \tfrac{1}{p}(\xi_0\|H^{1/2}h\| - \|H^{1/2}\xi\|)^2 \\ &\geq 0. \end{aligned}$$

Therefore, $-q + pf(x/p)$ is convex in $\Re^1_{++} \times \Re^1 \times \Re^n$. **Q.E.D.**

We have an equivalent conic formulation for (P) as given by

$$\begin{array}{lll} (KP) & \min & \bar{c}^T\bar{x} \\ & \text{s.t.} & \bar{A}\bar{x} = \bar{b} \\ & & \bar{x} \in \mathcal{K}. \end{array}$$

The convex barrier function for $\mathcal{K}$ is

$$F(\bar{x}) = -\log p - \log\left(q - pf\left(x/p\right)\right),$$

which is a 2-logarithmically homogeneous barrier function. We can easily check: That for any $\bar{x} = \begin{bmatrix} p \\ q \\ x \end{bmatrix}$, it follows that

$$\begin{aligned} F(t\bar{x}) &= -\log(tp) - \log\left[t(q - pf(x/p))\right] \\ &= F(\bar{x}) - 2\log t. \end{aligned}$$

Related to the duality of convex cones, the conjuate of the convex function $f(x)$ is defined as

$$f^*(s) = \sup\{(-s)^T x - f(x) \mid x \in \text{dom } f\}.$$

By the definition, we can see that for $x \in \text{int}\{\text{dom } f\}$ and $s \in \text{int}\{\text{dom } f^*\}$ the following three statements are equivalent

$$s = -\nabla f(x) \tag{2.4}$$

$$x = -\nabla f^*(s) \tag{2.5}$$

$$-x^T s = f(x) + f^*(s). \tag{2.6}$$

Related to the definition of the conjugate function, we have the following theorem.

Theorem 1 *It holds that*

$$\mathcal{K}^* = \text{cl}\left\{ \bar{s} = \begin{bmatrix} u \\ v \\ s \end{bmatrix} : v > 0,\ u - v f^*(s/v) \geq 0 \right\} \tag{2.7}$$

and

$$F^*(\bar{s}) = \log v - \log(u - v f^*(s/v)) \tag{2.8}$$

which is a barrier function for $\mathcal{K}^*$.

We include the proof for the equation (2.7) and omit the proof for (2.8). For details, one is referred to Zhang [43].

Proof. For any $\begin{bmatrix} u \\ v \\ s \end{bmatrix}$ with $v > 0$ and $u - v f^*(s/v) \geq 0$, and $\bar{x} = \begin{bmatrix} p \\ q \\ x \end{bmatrix} \in \mathcal{K}$ we have

$$\begin{aligned} pu + qv + x^T s &= pv[u/v + q/p + (x/p)^T (s/v)] \\ &\geq pv[u/v + q/p - f(x/p) - f^*(s/v)] \\ &= v(q - pf(x/p)) + p(u - vf^*(s/v)) \\ &\geq 0 \end{aligned}$$

Hence,

$$\left\{ \bar{s} = \begin{bmatrix} u \\ v \\ s \end{bmatrix} \ : \ v > 0, \ u - vf^* \left(\frac{s}{v} \right) \geq 0 \right\} \subseteq \mathcal{K}^*.$$

On the other hand, take any $\bar{s} = \begin{bmatrix} u \\ v \\ s \end{bmatrix} \in \text{int } \mathcal{K}^*$. Obviously $v \geq 0$. Since int $\mathcal{K}^*$ is open, we conclude that $v > 0$. Let $\hat{x} = -\nabla f^*(s/v)$. Consider

$$\begin{bmatrix} 1 \\ f(\hat{x}) \\ \hat{x} \end{bmatrix} \in \mathcal{K}.$$

As $f(\hat{x}) + f^*(s/v) = -\hat{x}^T(s/v)$ and so

$$0 \leq \bar{s}^T \begin{bmatrix} 1 \\ f(\hat{x}) \\ \hat{x} \end{bmatrix} = u - vf^*(s/v).$$

This shows that

$$\mathcal{K}^* = \text{cl} \left\{ \bar{s} = \begin{bmatrix} u \\ v \\ s \end{bmatrix} \ : \ v > 0, \ u - vf^*(s/v) \geq 0 \right\}.$$

□

Therefore, we get the dual of conic problem:

$$\begin{array}{lll} (KD) & \max & \bar{b}^T \bar{y} \\ & \text{s.t.} & \bar{A}^T \bar{y} + \bar{s} = c \\ & & \bar{s} \in \mathcal{K}^* \end{array}$$

where $\mathcal{K}^* = \text{cl} \left\{ \bar{s} = \begin{bmatrix} u \\ v \\ s \end{bmatrix} \ : \ v > 0, \ u - vf^*(s/v) \geq 0 \right\}$.

We consider the following barrier approach for solving (KP) with $\mu > 0$ as the barrier parameter.

$$\begin{array}{lccccccc}
\min & \mu F(\bar{x}) & -\mu\log\tau & +\beta\theta & +\mu F^*(\bar{s}) & -\mu\log\kappa & & \\
\text{s.t.} & \bar{A}\bar{x} & -\bar{b}\tau & +\bar{r}_p\theta & & & = & 0 \\
-\bar{A}^T\bar{y} & & +\bar{c}\tau & +\bar{r}_d\theta & -\bar{s} & & = & 0 \\
\bar{b}^T\bar{y} & -\bar{c}^T\bar{x} & & +\bar{r}_g\theta & & -\kappa & = & 0 \\
-\bar{r}_p^T\bar{y} & -\bar{r}_d^T\bar{x} & -\bar{r}_g\tau & & & & = & -\beta.
\end{array}$$

Due to the self-duality we derive the following KKT optimality condition for (KP), where the solution is denoted by $(\bar{y}, \bar{x}, \tau, \theta, \bar{s}, \kappa)$,

$$\left\{\begin{array}{lccccccc}
 & \bar{A}\bar{x}(\mu) & -\bar{b}\tau(\mu) & +r_p\theta(\mu) & & & = & 0 \\
-\bar{A}^T\bar{y}(\mu) & & +\bar{c}\tau(\mu) & +\bar{r}_d\theta(\mu) & -\bar{s}(\mu) & & = & 0 \\
\bar{b}^T\bar{y}(\mu) & -\bar{c}^T\bar{x}(\mu) & & +\bar{r}_g\theta(\mu) & & -\kappa(\mu) & = & 0 \\
-\bar{r}_p^T\bar{y}(\mu) & -\bar{r}_d^T\bar{x}(\mu) & -\bar{r}_g\tau(\mu) & & & & = & -\beta \\
 & & \tau(\mu)\kappa(\mu) & & & & = & \mu \\
 & & & & \bar{s}(\mu) & & = & -\mu\nabla F(\bar{x}).
\end{array}\right. \tag{2.9}$$

We use the chain rule to calculate $\nabla F(\bar{x})$ and then substitute the term. After that we get the following KKT conditions

$$(*)\left\{\begin{array}{lcl}
\left.\begin{array}{lccccccc}
 & \bar{A}\bar{x}(\mu) & -\bar{b}\tau(\mu) & +\bar{r}_p\theta(\mu) & & & = & 0 \\
-\bar{A}^T\bar{y}(\mu) & & +\bar{c}\tau(\mu) & +\bar{r}_d\theta(\mu) & -\bar{s}(\mu) & & = & 0 \\
\bar{b}^T\bar{y}(\mu) & -\bar{c}^T\bar{x}(\mu) & & +\bar{r}_g\theta(\mu) & & -\kappa(\mu) & = & 0 \\
-\bar{r}_p^T\bar{y}(\mu) & -\bar{r}_d^T x(\mu) & -\bar{r}_g\tau(\mu) & & & & = & -\beta
\end{array}\right\} & (1) \\
\left.\begin{array}{rcl}
\tau(\mu)\kappa(\mu) & = & \mu \\
u(\mu)[q(\mu) - p(\mu)f(x(\mu)/p(\mu))] & = & \mu[q(\mu)/p(\mu) - 2f(x(\mu)/p(\mu)) \\
 & & +\nabla^T f(x(\mu)/p(\mu))(x(\mu)/p(\mu)) \\
v(\mu)[q(\mu) - p(\mu)f(x(\mu)/p(\mu))] & = & \mu \\
s(\mu)[q(\mu) - p(\mu)f(x(\mu)/p(\mu))] & = & -\mu\nabla f(x(\mu)/p(\mu)).
\end{array}\right\} & (2)
\end{array}\right.$$

2.5 Chapter summary

In this chapter, the main ideas in [43] were reviewed. According to the results in this paper, we can turn a general convex optimization problem into the

conic form by adding 2 extra variables. Then we can follow the self-dual embedding approach to solve it. In the next chapter, we will construct an algorithm to implement this method.

Chapter 3

Implementation of the algorithm

In this chapter, we shall construct an algorithm to implement the method introduced in the previous chapter. We first consider the case where there is only one inequality constraint. Then we shall focus on finding the Newton type direction. The conditions for selecting the step length along the Newton direction will be discussed as well. Finally, we extend our analysis to the multi-constraint case.

3.1 The new search direction

In this section, we first discuss how to find the new search direction for the single constraint case. As discussed in Chapter 2, we have the following KKT

system for the central solution:

$$(*)\begin{cases} \left.\begin{array}{rrrrrrcl} & Ax(\mu) & -b\tau(\mu) & +r_p\theta(\mu) & & & = & 0 \\ -A^Ty(\mu) & & +c\tau(\mu) & +r_d\theta(\mu) & -s(\mu) & & = & 0 \\ b^Ty(\mu) & -cx(\mu) & & +r_g\theta(\mu) & & -\kappa(\mu) & = & 0 \\ -r_p^Ty(\mu) & -r_d^Tx(\mu) & -r_g\tau(\mu) & & & & = & -\beta \end{array}\right\} (1) \\ \left.\begin{array}{rcl} \tau(\mu)\kappa(\mu) & = & \mu \\ u(\mu)[q(\mu)-p(\mu)f(x(\mu)/p(\mu))] & = & \mu[q(\mu)/p(\mu)-2f(x(\mu)/p(\mu)) \\ & & \quad +\nabla^T f(x(\mu)/p(\mu))(x(\mu)/p(\mu)) \\ v(\mu)[q(\mu)-p(\mu)f(x(\mu)/p(\mu))] & = & \mu \\ s(\mu)[q(\mu)-p(\mu)f(x(\mu)/p(\mu))] & = & -\mu\nabla f(x(\mu)/p(\mu)). \end{array}\right\} (2) \end{cases}$$

Suppose that the current iterates is $(p,\ x,\ u,\ v,\ s,\ y,\ \theta,\ \tau,\ \kappa)$. Then the displacements $(\Delta p,\ \Delta x,\ \Delta u,\ \Delta v,\ \Delta s,\ \Delta y,\ \Delta\theta,\ \Delta\tau,\ \Delta\kappa)$ should satisfy

$$\begin{cases} \begin{array}{rrrrrrcl} & \bar{A}\Delta\bar{x} & -\bar{b}\Delta\tau & +\bar{r}_p\Delta\theta & & & = & 0 \\ -\bar{A}^T\Delta\bar{y} & & +\bar{c}\Delta\tau & +\bar{r}_d\Delta\theta & -\Delta\bar{s} & & = & 0 \\ \bar{b}^T\Delta\bar{y} & -\bar{c}^T\Delta\bar{x} & & +\bar{r}_g\Delta\theta & & -\Delta\kappa & = & 0 \\ -\bar{r}_p^T\Delta\bar{y} & -\bar{r}_d^T\Delta\bar{x} & -\bar{r}_g\Delta\tau & & & & = & 0 \\ & & \kappa\Delta\tau & & & +\tau\Delta\kappa & = & \mu-\kappa\tau \end{array} \\ (u+\Delta u)(p+\Delta p)\left[(q+\Delta q)-(p+\Delta p)f\left(\frac{x+\Delta x}{p+\Delta p}\right)\right] = \mu q+\Delta q-2\mu(p+\Delta p)f\left(\frac{x+\Delta x}{p+\Delta p}\right) \\ \qquad +\mu\nabla f\left(\frac{x+\Delta x}{p+\Delta p}\right)^T(x+\Delta x) \\ (v+\Delta v)\left[(q+\Delta q)-(p+\Delta p)f\left(\frac{x+\Delta x}{p+\Delta p}\right)\right] = \mu \\ (s+\Delta s)\left[(q+\Delta q)-(p+\Delta p)f\left(\frac{x+\Delta x}{p+\Delta p}\right)\right] = -\mu\nabla f\left(\frac{x+\Delta x}{p+\Delta p}\right). \end{cases}$$

This is a system of nonlinear equations with $2n+m+7$ variables and $2n+m+7$ equations. We shall linearize the system as suggested by the Newton method for solving nonlinear equations. To this end, we rewrite the form of the last three equation in the above system as follows

$$(u+\Delta u)(p+\Delta p)\left[(q+\Delta q)-(p+\Delta p)f\left(\frac{x+\Delta x}{p+\Delta p}\right)\right]$$

$$= \mu q + \Delta q - 2\mu\,(p+\Delta p)\, f\left(\frac{x+\Delta x}{p+\Delta p}\right) + \mu \nabla f\left(\frac{x+\Delta x}{p+\Delta p}\right)^T (x+\Delta x) \qquad (3.1)$$

$$(v+\Delta v)\left[(q+\Delta q) - (p+\Delta p)\, f\left(\frac{x+\Delta x}{p+\Delta p}\right)\right] = \mu \qquad (3.2)$$

$$(s+\Delta s)\left[(q+\Delta q) - (p+\Delta p)\, f\left(\frac{x+\Delta x}{p+\Delta p}\right)\right] = -\mu \nabla f\left(\frac{x+\Delta x}{p+\Delta p}\right). \qquad (3.3)$$

We are now concerned with the left hand side of (3.1). Using the Taylor expansion to the first order, and dropping all the higher order terms,

$$\begin{aligned} & (u+\Delta u)\,(p+\Delta p)\,[(q+\Delta q) - (p+\Delta p)\, f\,((x+\Delta x)/(p+\Delta p))] \\ \cong\ & (up + u\Delta p + p\Delta u)\left[(q+\Delta q) - (p+\Delta p)\, f\left(x/p + \Delta x/p - x\Delta p/p^2\right)\right]. \end{aligned}$$

We further drop the terms which are higher than or equal to second order, the above quantity becomes approximately

$$\begin{aligned} & upq + uq\Delta p + qp\Delta u + up\Delta q - \left(up^2 + up\Delta p + p^2\Delta u + up\Delta p\right) f\,(x/p) \\ & -up^2 \nabla f\,(x/p)^T \left(\Delta x/p - x\Delta p/p^2\right) \\ \cong\ & upq + \left[uq - 2upf\,(x/p) + u\nabla f\,(x/p)^T x\right]\Delta p + \left[qp - p^2\,(x/p)\right]\Delta u \\ & + up\Delta q - up\nabla f\,(x/p)^T \Delta x. \end{aligned}$$

As for the right hand side of (3.1), similarly we have

$$\begin{aligned} & \mu\,[q + \Delta q - 2(p+\Delta p) f((x+\Delta x)/(p+\Delta p)) \\ & \qquad + \nabla f((x+\Delta x)/(p+\Delta p))^T (x+\Delta x)\big] \\ \cong\ & \mu q + \mu\Delta q - 2\mu(p+\Delta p) f\,(x/p) - 2\mu p \nabla f\,(x/p)^T \left(\Delta x/p - x\Delta p/p^2\right) \\ & + \mu \nabla f\,(x/p)^T (x+\Delta x) + \mu\left(\Delta x/p - x\Delta p/p^2\right)^T \nabla^2 f\,(x/p)\,(x+\Delta x) \\ \cong\ & \mu q - 2\mu p f\,(x/p) + \mu \nabla f\,(x/p)^T x + \left[-\mu \nabla f\,(x/p)^T + (\mu/p) x^T \nabla^2 f\,(x/p)\right]\Delta x \\ & - \left[2\mu f\,(x/p) + 2\mu/p \nabla f\,(x/p)^T x + \mu/p^2 x^T \nabla^2 f\,(x/p)\,x\right]\Delta p. \end{aligned}$$

Finally we linearize the equation (3.1) and get

$$\begin{aligned} & \left[uq + 2(\mu - up) f(x/p) + (u + 2\mu/p) \nabla f(x/p)^T x + (\mu/p^2) x^T \nabla^2 f(x/p)\, x\right] \Delta p \\ & + \left[qp - p^2 f(x/p)\right] \Delta u + (up - \mu) \Delta q + \left[(\mu - up) \nabla f(x/p)^T - (\mu/p) x^T \nabla^2 f(x/p)\right] \Delta x \\ = \;& \mu q - upq - 2\mu p f(x/p) + \mu \nabla f(x/p)^T x + up^2 f(x/p). \end{aligned}$$

Consider now the equation (3.2)

$$(v + \Delta v)\left[(q + \Delta q) - (p + \Delta p) f\left((x + \Delta x)/(p + \Delta p)\right)\right] \;=\; \mu.$$

By Taylor expansion and dropping high order terms, this yields

$$\begin{aligned} \mu \;\cong\; & vq + v\Delta q + q\Delta v - (vp + v\Delta p + p\Delta v)\left[f(x/p) + \nabla f(x/p)^T \left(\Delta x/p - (x\Delta p)/p^2\right)\right] \\ \cong\; & vq - vpf(x/p) + v\Delta q + \left[q - pf(x/p)\right] \Delta v + \left[v/p \nabla f(x/p)^T x - vf(x/p)\right] \Delta p \\ & - v \nabla f(x/p)^T \Delta x. \end{aligned}$$

Therefore, we get the Newton equation

$$\begin{aligned} & \left[q - pf(x/p)\right] \Delta v + \left[v/p \nabla f(x/p)^T x - vf(x/p)\right] \Delta p + v\Delta q - v \nabla f(x/p)^T \Delta x \\ = \;& \mu - vq + vpf(x/p). \end{aligned}$$

Similarly, we can linearize the third equation (3.3) as follows

$$(s + \Delta s)\left[(q + \Delta q) - (p + \Delta p) f\left((x + \Delta x)/(p + \Delta p)\right)\right] = -\mu \nabla f\left((x + \Delta x)/p + \Delta p\right),$$

leading to

$$\begin{aligned} & sq + s\Delta q + q\Delta s - (ps + s\Delta p + p\Delta s) f(x/p) - ps \nabla f(x/p)^T (\Delta x/p - (x\Delta p)/p^2) \\ = \;& -\mu \nabla f(x/p) - \mu \nabla^2 f(x/p)^T (\Delta x/p - x\Delta p/p^2) \end{aligned}$$

and this further leads to

$$\begin{aligned} & \left[q - pf(x/p)\right] \Delta s + \left[-sf(x/p) + s/p \nabla f(x/p)^T x - \mu/p^2 \nabla^2 f(x/p)^T x\right] \Delta p + s\Delta q \\ & + \left[-s \nabla f(x/p)^T + \mu/p \nabla^2 f(x/p)^T\right] \Delta x = -\mu \nabla f(x/p) - sq + spf(x/p). \end{aligned}$$

Now we wish to reformulate the equations in the matrix format. Let

$$\begin{aligned}
L_1 &= pq - p^2 f(x/p) \\
L_2 &= uq + 2(\mu - up) f(x/p) + (u + 2\mu/p) \nabla f(x/p)^T x + \mu/p^2 x^T \nabla^2 f(x/p) x \\
L_3 &= (\mu - up) \nabla f(x/p)^T - (\mu/p) x^T \nabla^2 f(x/p) \\
L_4 &= v/p \, \nabla f(x/p)^T x - v f(x/p) \\
L_5 &= -s \nabla f(x/p)^T + \mu/p \nabla^2 f(x/p) \\
L_6 &= -s f(x/p) + s/p \nabla f(x/p)^T x - \mu/p^2 \nabla^2 f(x/p) x,
\end{aligned}$$

and we write the last three equations in terms of L_i, $i = 1, ..., 6$:

$$\begin{aligned}
p^2 L_1 \Delta u + (up - \mu) \Delta q + L_2 \Delta p + L_3 \Delta x &= M_1 \\
L_1 \Delta v + pv \Delta q + vp \nabla f(x/p)^T \Delta x + L_4 \Delta p &= M_2 \\
pL_1 \Delta s + p^2 s \Delta q + L_5 \Delta x + L_6 \Delta p &= M_3,
\end{aligned}$$

where

$$\begin{aligned}
M_1 &= \mu q - upq - 2\mu p f(x/p) + \mu \nabla f(x/p)^T x + up^2 f(x/p) \\
M_2 &= \mu - vq + vp f(x/p) \\
M_3 &= -\mu \nabla f(x/p) - sq + sp f(x/p).
\end{aligned}$$

Observe that $M_1, M_2 \in \Re$, which are dependent on (u, v, s) and (p, q, x). Hence we can rewrite the above equations in the matrix form as follows

$$D_1 L_1 \Delta \bar{s} + D_2 \Delta \bar{x} = \bar{M},$$

with

$$D_1 = \begin{pmatrix} p^2 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & p \end{pmatrix}$$

$$D_2 = \begin{pmatrix} L_2 & up-\mu & L_3 \\ L_4 & vp & vp\nabla f\,(x/p)^T \\ L_6 & p^2 s & L_5 \end{pmatrix}$$

$$\bar{M} = \begin{pmatrix} M_1 \\ M_2 \\ M_3 \end{pmatrix}$$

where D_1 and D_2 are $(n+2)\times(n+2)$ matrix. Clearly D_1^{-1} exists since it is a positive diagonal matrix. However, D_2 is not guaranteed to be invertible. For convenience we shall replace it by $D_2+\varepsilon I$ with $\varepsilon > 0$ a small perturbation parameter in case D_2 is not invertible.

Define

$$\begin{aligned} Q_1 &= -\left(\bar{A}D_2^{-1}D_1\bar{A}^T\right)^{-1}\bar{M} \\ Q_2 &= -\left(AD_2^{-1}D_1\bar{A}^T\right)^{-1}\left(\bar{b}-\bar{A}D_2^{-1}D_1\bar{c}\right) \\ Q_3 &= -\left(\bar{A}D_2^{-1}D_1\bar{A}^T\right)^{-1}\left(\bar{r_p}+\bar{A}D_2^{-1}D_1\bar{r}_d\right). \end{aligned}$$

Then we have

$$\Delta\bar{y} = Q_1 + Q_2\Delta\tau + Q_3\Delta\theta.$$

Moreover, by putting

$$\begin{aligned} R_1 &= D_2^{-1}\left(\bar{M}+D_1\bar{A}^TQ_1\right) \\ R_2 &= D_2^{-1}\left(D_1\bar{A}^TQ_2-D_1\bar{c}\right) \\ R_3 &= D_2^{-1}\left(D_1\bar{A}^TQ_3-D_1\bar{r}_d\right) \end{aligned}$$

we obtain

$$\Delta\bar{x} = R_1 + R_2\Delta\tau + R_3\Delta\theta.$$

Similarly, letting

$$N_1 = \frac{-\bar{r}_p^TQ_1-\bar{r}_d^TR_1}{\bar{r}_g+\bar{r}_p^TQ_2+\bar{r}_d^TR_2}$$

$$N_2 = \frac{-\bar{r}_p^T Q_2 - \bar{r}_d^T R_3}{\bar{r}_g + \bar{r}_p^T Q_2 + \bar{r}_d^T R_2}$$

we have

$$\Delta\tau = N_1 + N_2 \Delta\theta$$

and

$$\Delta\theta = \frac{\mu - \kappa\tau - \tau\bar{b}^T Q_1 - \tau\bar{b}^T Q_2 N_1 + \tau\bar{c} R_1 + \tau\bar{c}^T R_2 N_1 - \kappa N_1}{\tau\bar{b}^T Q_2 N_2 + \tau\bar{b}^T Q_3 - \tau\bar{c}^T R_2 N_2 - \tau\bar{c}^T R_3 + \kappa N_3 + \tau\bar{r}_g}. \tag{3.4}$$

Observe that $\Delta\theta$ can be directly computed since all terms on the right side are known, and we can consequently computer other variables as follows:

$$\Delta\tau = N_1 + N_2 \Delta\theta,$$

$$\Delta\bar{x} = R_1 + R_2 \Delta\tau + R_3 \Delta\theta,$$

$$\Delta\bar{y} = Q_1 + Q_2 \Delta\tau + Q_3 \Delta\theta,$$

$$\Delta\kappa = \bar{b}^T \Delta\bar{y} - \bar{c}^T \Delta\bar{x} + \bar{r}_g \Delta\theta$$

and

$$\Delta\bar{s} = -\bar{A}^T \Delta\bar{y} + \bar{c}\Delta\tau + \bar{r}_d \Delta\theta.$$

3.2 Select the step-length

After computing the search direction, the iterates may in principle be updated. Next, we would like to discuss how to choose a proper step-size α. First, we follow Andersen and Ye [1], and define the merit function.

$$\phi(y, p, q, x, \tau, \theta, u, v, s, \kappa) := \lambda\psi_1^T e + \|\psi_2\|,$$

in which $\lambda \in (0,1)$ is a parameter,

$$\psi_1 = \begin{pmatrix} \tau\kappa \\ \frac{u(q-pf(x/p))}{q/p-2f(x/p)+\nabla^T f(x/p)(x/p)} \\ v\left(q-pf\left(x/p\right)\right) \\ -\left(q-pf\left(x/p\right)\right) s \circ \left[\nabla f\left(x/p\right)\right]^{-1} \end{pmatrix},$$

and

$$\psi_2 = \begin{pmatrix} \bar{A}\bar{x} - \bar{b}\tau + \bar{r}_p\theta \\ -\bar{A}\bar{y} + \bar{c}\tau + \bar{r}_d\theta - \bar{s} \\ \bar{b}^T\bar{y} - \bar{c}^T\bar{x} + \bar{r}_g\theta - \kappa \\ -\bar{r}_p^T\bar{y} - \bar{r}_d^T\bar{x} - \bar{r}_g\tau + \bar{\beta} \end{pmatrix}$$

where "$\circ$" is the Hadamard product of the two vectors, and "$\left[\nabla f\left(x/p\right)\right]^{-1}$" is the component-wise inverse of $\nabla f\left(x/p\right)$, i.e.

$$\nabla f\left(x/p\right)^{-1} = \begin{cases} \frac{1}{\nabla f_i(x/p)} & if \quad \nabla f_i(x/p) \neq 0 \\ 0 & if \quad \nabla f_i(x/p) = 0. \end{cases}$$

We see that ψ_1 is used to measure the duality gap and ψ_2 is used to measure the feasibility. This can be seen from Equation (*).

In each iteration, the step-size is selected such that all iterations satisfy the following three conditions.

The first condition:

$$\frac{\left(\psi_1^+\right)^T e}{\left(\psi_1^0\right)^T e} \geq \beta_1 \frac{\|\psi_2^+\|}{\|\psi_2^0\|},$$

where $\psi_i^+ = \psi_i\left(y^+, p^+, q^+, x^+, \tau^+, \theta^+, u^+, v^+, s^+, \kappa^+\right)$ for $i = 1, 2$. This condition prevents the iterates from converging towards a complementarity solution faster than the feasibility improvement.

The second condition:

$$\min \psi_1^+ \geq \beta_2 \frac{\left(\psi_1^+\right)^T e}{n+3}.$$

This condition prevents the iterates from converging towards the boundary of the positive orthant prematurely.

The third condition:

$$\phi^+ \leq \phi + \beta_3 \alpha \nabla \phi^T \cdot (\Delta y; \Delta p; \Delta q; \Delta x; \Delta \tau; \Delta \theta; \Delta u; \Delta v; \Delta s; \Delta \kappa).$$

The last Armijo-like condition requires that the merit function to be reduced in all iterations.

3.3 The multi-constraint case

Next we consider the general formulation of (P) where $m \geq 1$.

$$\begin{array}{lll} (P) & \min & c^T x \\ & \text{s.t.} & Ax = b \\ & & f_i(x) \leq 0, \ i = 1, ..., m \end{array} \tag{3.5}$$

where $f_i(x)$ is smooth and convex, $i = 1, ..., m$. Similarly we have its conic representation, (KP), with

$$\mathcal{K} = \bigcap_{i=1}^{m} \mathcal{K}_i,$$

where

$$\mathcal{K}_i = \text{cl}\left\{\bar{x} \mid \ p > 0, \ q - p f_i(x/p) \geq 0\right\} \subseteq \mathcal{R}^{n+2}, \ \ i = 1, ..., m.$$

The natural $2m$-logarithmically homogeneous barrier function for $\mathcal{K}$ is

$$F(\bar{x}) = -m\log p - \sum_{i=1}^{m}\log(q - pf_i(x/p)).$$

The dual cone of $\mathcal{K}$ is

$$\mathcal{K}^* = \mathrm{cl}(\mathcal{K}_1^*\oplus\cdots\oplus\mathcal{K}_m^*) = \mathrm{cl}\left\{\sum_{i=1}^{m}\bar{s}_i = \sum_{i=1}^{m}\begin{bmatrix} u_i \\ v_i \\ s_i \end{bmatrix} : v_i > 0, u_i - v_i f_i^*(s_i/v_i) \geq 0, i = 1,...,m\right\}.$$

The central path for the embedded problem is characterized by :

$$\left\{\begin{array}{rrrrrrrl}
 & \bar{A}\bar{x}\,(\mu) & -\bar{b}\tau\,(\mu) & +r_p\theta\,(\mu) & & & = & 0 \\
-\bar{A}^T\bar{y}\,(\mu) & & +\bar{c}\tau\,(\mu) & +r_d\theta\,(\mu) & -\bar{s}\,(\mu) & & = & 0 \\
\bar{b}^T y\,(\mu) & -\bar{c}x\,(\mu) & & +r_g\theta\,(\mu) & & -\kappa\,(\mu) & = & 0 \\
-r_p^T\bar{y}\,(\mu) & -r_d^T\bar{x}\,(\mu) & -r_g\tau\,(\mu) & & & & = & -\beta \\
 & \bar{s}(\mu) - \sum_{i=1}^{m}\bar{s}_i(\mu) & = & 0 & & & & \\
 & \tau(\mu)\kappa(\mu) & = & \mu & & & & \\
u_i(\mu)[q(\mu) - p(\mu)f_i(x(\mu)/p(\mu))] & = & \mu[q(\mu)/p(\mu) - 2f_i(x(\mu)/p(\mu)) & & & & & \\
 & & & +\nabla^T f_i(x(\mu)/p(\mu))(x(\mu)/p(\mu))] & & & & \\
v_i(\mu)[q(\mu) - p(\mu)f_i(x(\mu)/p(\mu))] & = & \mu & & & & & \\
s_i(\mu)[q(\mu) - p(\mu)f_i(x(\mu)/p(\mu))] & = & -\mu\nabla f_i(x(\mu)/p(\mu)) \text{ for } i = 1,...,m. & & & & &
\end{array}\right.$$

We linearize the last three equations and then we find out the new direction of the system. Remember that in the one constraint case we have

$$D_1\Delta\bar{s} + D_2\Delta\bar{x} = \bar{M},$$

with

$$D_1 = \begin{pmatrix} p^2L_1 & 0 & 0 \\ 0 & L_1 & 0 \\ 0 & 0 & pL_1I \end{pmatrix}$$

$$D_2 = \begin{pmatrix} L_2 & up - \mu & L_3 \\ L_4 & vp & vp\nabla f\,(x/p)^T \\ L_6 & p^2s & L_5 \end{pmatrix}$$

$$\bar{M} = \begin{pmatrix} M_1 \\ M_2 \\ M_3 \end{pmatrix}.$$

Similarly, in the m constraints case, we have

$$D_1^i \Delta\bar{s} + D_2^i \Delta\bar{x} = \bar{M}^i, \text{ for } i = 1, ..., m.$$

where

$$D_1^i = \begin{pmatrix} p^2 L_1^i & 0 & 0 \\ 0 & L_1^i & 0 \\ 0 & 0 & pL_1^i \end{pmatrix}$$

$$D_2^i = \begin{pmatrix} L_2^i & u_i p - \mu & L_3^i \\ L_4^i & v_i p & v_i p \nabla f_i (x/p)^T \\ L_6^i & p^2 s_i & L_5^i \end{pmatrix}$$

$$\bar{M}^i = \begin{pmatrix} M_1^i \\ M_2^i \\ M_3^i \end{pmatrix}$$

and

$$L_1^i = pq - p^2 f_i (x/p)$$

$$L_2^i = u_i q + 2(\mu - u_i p) f_i (x/p) + (u_i + 2\mu/p) \nabla f_i (x/p)^T x + \mu/p^2 x^T \nabla^2 f_i (x/p) x$$

$$L_3^i = (\mu - u_i p) \nabla f_i (x/p)^T - (\mu/p) x^T \nabla^2 f_i (x/p)$$

$$L_4^i = (v_i/p) \nabla f_i (x/p)^T x - v_i f_i (x/p)$$

$$L_5^i = -s_i \nabla f_i (x/p)^T + (\mu/p) \nabla^2 f_i (x/p)$$

$$L_6^i = -s_i f_i (x/p) + s_i \nabla f_i (x/p)^T x/p - (\mu/p^2) \nabla^2 f_i (x/p) x$$

where

$$M_1^i = \mu q - u_i p q - 2\mu p f_i (x/p) + \mu \nabla f_i (x/p)^T x + u_i p^2 f_i (x/p)$$

$$M_2^i = \mu - v_i q + v_i p f_i(x/p)$$

$$M_3^i = -\mu \nabla f_i(x/p) - s_i q + s_i p f_i(x/p),$$

and hence

$$\Delta \bar{s}_i = (D_1^i)^{-1}(\bar{M}^i - D_2^i \Delta \bar{x}) \text{ for } i = 1, ..., m.$$

Therefore,

$$\sum_{i=1}^{m} \Delta \bar{s}_i = \sum_{i=1}^{m} (D_1^i)^{-1}(\bar{M}^i - D_2^i \Delta \bar{x})$$

Because $\sum_{i=1}^{m} \Delta \bar{s}_i = \Delta \bar{s}$, so

$$\Delta \bar{s} = \sum_{i=1}^{m} (D_1^i)^{-1} \bar{M}^i - \sum_{i=1}^{m} (D_1^i)^{-1} D_2^i \Delta \bar{x}$$

Let

$$\bar{M} = \sum_{i=1}^{m} (D_1^i)^{-1} \bar{M}^i$$

$$D_2 = \sum_{i=1}^{m} (D_1^i)^{-1} D_2^i.$$

We have a new matrix form equation

$$\Delta \bar{s} + D_2 \Delta \bar{x} = \bar{M}.$$

Following similar calculation as for the case $m = 1$, we obtain

$$\Delta \tau = N_1 + N_2 \Delta \theta, \tag{3.6}$$

$$\Delta \bar{x} = R_1 + R_2 \Delta \tau + R_3 \Delta \theta, \tag{3.7}$$

$$\Delta \bar{y} = Q_1 + Q_2 \Delta \tau + Q_3 \Delta \theta, \tag{3.8}$$

$$\Delta \kappa = \bar{b}^T \Delta \bar{y} - \bar{c}^T \Delta \bar{x} + \bar{r}_g \Delta \theta \tag{3.9}$$

and

$$\Delta\bar{s} = -\bar{A}^T\Delta\bar{y} + \bar{c}\Delta\tau + \bar{r}_d\Delta\theta, \tag{3.10}$$

where $\Delta\theta$ is given by (3.4). Also

$$\Delta\bar{s}_i = (D_1^i)^{-1}(\bar{M}^i - D_2^i\Delta\bar{x}), \tag{3.11}$$

for $i = 1, ..., m$. After searching the direction, we need to choose a step length. Similar as in Section 3.2, we first should define the merit function

$$\Phi(y, p, q, x, \tau, \theta, u_1, v_1, s_1, ..., u_m, v_m, s_m, \kappa) := \lambda\Psi_1^T e + \|\Psi_2\|,$$

in which $\lambda \in (0, 1)$ is a parameter, and

$$\Psi_1 = \begin{pmatrix} \tau\kappa \\ \frac{u_1(q-pf_1(x/p))}{q/p-2f_1(x/p)+\nabla^T f_1(x/p)(x/p)} \\ v_1\left(q - pf_1\left(x/p\right)\right) \\ -\left(q - pf_1\left(x/p\right)\right) s_1 \circ \left[\nabla f_1\left(x/p\right)\right]^{-1} \\ \vdots \\ \vdots \\ \frac{u_m(q-pf_m(x/p))}{q/p-2f_m(x/p)+\nabla^T f_m(x/p)(x/p)} \\ v_m\left(q - pf_m\left(x/p\right)\right) \\ -\left(q - pf_m\left(x/p\right)\right) s_m \circ \left[\nabla f_m\left(x/p\right)\right]^{-1} \end{pmatrix},$$

and

$$\Psi_2 = \begin{pmatrix} \bar{A}\bar{x} - \bar{b}\tau + \bar{r}_p\theta \\ -\bar{A}\bar{y} + \bar{c}\tau + \bar{r}_d\theta - \sum_{i=1}^{m} \bar{s}_i \\ \bar{b}^T\bar{y} - \bar{c}^T\bar{x} + \bar{r}_g\theta - \kappa \\ -\bar{r}_p^T\bar{y} - \bar{r}_d^T\bar{x} - \bar{r}_g\tau + \bar{\beta} \end{pmatrix}$$

where "$\circ$" is the Hadamard product of the two vectors. In addition, the step-size is selected such that all iterates satisfy the following three conditions.

The first condition:

$$\frac{\left(\Psi_1^+\right)^T e}{\left(\Psi_1^0\right)^T e} \geq \beta_1 \frac{\|\Psi_2^+\|}{\|\Psi_2^0\|}, \tag{3.12}$$

where $\Psi_i^+ = \Psi_i\left(y^+, p^+, q^+, x^+, \tau^+, \theta^+, u_1^+, v_1^+, s_1^+, ..., u_m^+, v_m^+, s_m^+, \kappa^+\right)$ for $i = 1, 2$. This condition prevents the iterates from converging towards a complementarity solution faster than the feasibility measure.

The second condition:

$$\min \Psi_1^+ \geq \beta_2 \frac{\left(\Psi_1^+\right)^T e}{n+3}. \tag{3.13}$$

This condition prevents the iterates from converging towards the boundary of the positive orthant prematurely.

The third condition:

$$\begin{aligned} \Phi^+ &\leq \Phi \\ &+ \beta_3 \alpha \nabla \Phi^T \cdot (\Delta y; \Delta p; \Delta q; \Delta x; \Delta \tau; \Delta \theta; \Delta u_1; \Delta v_1; \Delta s_1; ...; \Delta u_m; \Delta v_m; \Delta s_m; \Delta \kappa). \end{aligned} \tag{3.14}$$

The last Armijo-like condition requires that the merit function to be reduced in all iterations.

We now arrive at a general algorithmic scheme as follows.

Algorithm

Step 0. Let

$$\begin{aligned} & (y, p, q, x, \tau, \theta, u_1, v_1, s_1, ..., u_m, v_m, s_m, \kappa) \\ = \; & (y^0, p^0, q^0, x^0, \tau^0, \theta^0, u_1^0, v_1^0, s_1^0, ..., u_m^0, v_m^0, s_m^0, \kappa^0) \end{aligned}$$

be the initial solution.

Step 1. If $\Phi(y, p, q, x, \tau, \theta, u_1, v_1, s_1, ..., u_m, v_m, s_m, \kappa) > \epsilon$, then go to step 2. Otherwise stop.

(*Remark*: If the value of the merit function is close to zero, then it follows that the iterate is close to the optimal solution.)

Step 2. Let $\mu = 0.8 * \Psi_1(y, p, q, x, \tau, \theta, u_1, v_1, s_1, ..., u_m, v_m, s_m, \kappa)/(n+3)$.

Step 3. Solving the direction

$$(\Delta y, \Delta p, \Delta q, \Delta x, \Delta \tau, \Delta \theta, \Delta u_1, \Delta v_1, \Delta s_1, ..., \Delta u_m, \Delta v_m, \Delta s_m, \Delta \kappa)$$

as given by (3.6)- (3.11), for $i = 1, ..., m$.

Step 4. Find the maximum step length

$$\begin{aligned} \alpha = \text{argmax} \; \{ \; & \alpha \in [0,1] : \\ & (p; q; x) + \alpha \, (\Delta p; \Delta q; \Delta x) \in \mathcal{K} \\ & (u_i; v_i; s_i) + \alpha \, (\Delta u_i; \Delta v_i; \Delta s_i) \in \mathcal{K}_i^*, \text{ for } i = 1, ..., m\} \end{aligned}$$

Step 5.

$$\begin{aligned}&\left(y^+, p^+, q^+, x^+, \tau^+, \theta^+, u_1^+, v_1^+, s_1^+, ..., u_m^+, v_m^+, s_m^+, \kappa^+\right)\\ =\ &(y, p, q, x, \tau, \theta, u_1, v_1, s_1, ..., u_m, v_m, s_m, \kappa)\\ &+ \alpha\,(\Delta y, \Delta p, \Delta q, \Delta x, \Delta\tau, \Delta\theta, \Delta u_1, \Delta v_1, \Delta s_1, ..., \Delta u_m, \Delta v_m, \Delta s_m, \Delta\kappa)\end{aligned}$$

Step 6. If

$$\frac{\left(\Psi_1^+\right)^T e}{\left(\Psi_1^0\right)^T e} < \beta_1 \frac{\|\Psi_2^+\|}{\|\Psi_2^0\|}$$

or

$$\min \Psi_1^+ < \beta_2 \frac{\left(\Psi_1^+\right)^T e}{n+3}$$

or

$$\Phi^+ > \Phi + \beta_3 \alpha \nabla \Phi^T \cdot (\Delta y, \Delta p, \Delta q, \Delta x, \Delta\tau, \Delta\theta, \Delta u_1, \Delta v_1, \Delta s_1, ..., \Delta u_m, \Delta v_m, \Delta s_m, \Delta\kappa)$$

$$\alpha = 0.8 * \alpha,$$

then go to step 5, otherwise go to step 7.

Step 7.

$$\begin{aligned}&(y, p, q, x, \tau, \theta, u_1, v_1, s_1, ..., u_m, v_m, s_m, \kappa)\\ =\ &\left(y^+, p^+, q^+, x^+, \tau^+, \theta^+, u_1^+, v_1^+, s_1^+, ..., u_m^+, v_m^+, s_m^+, \kappa^+\right).\end{aligned}$$

Step 8. Update the value of $\Phi\,(y, p, q, x, \tau, \theta, u_1, v_1, s_1, ..., u_m, v_m, s_m, \kappa)$. Then go to step 1.

3.4 Chapter summary

In conclusion, we use Newton's method to solve the approximative KKT system. Besides, we follow the approach of Andersen and Ye [1] to form a merit function. Using this function, we select the step length and measure

the performance for each iteration. Finally, we propose an algorithm to solve this problem.

Chapter 4

Numerical results on randomly generated problem

In this chapter, we report the computational results for our algorithm. We use the algorithm described in Chapter 3 to solve some test problems, through which we wish to show the efficiency of our method. For each randomly generated test problem, we use our algorithm to solve 10 problems for each n and get the statistic results. In addition, we would like to find out how the computational time grows with the size of the problem. The algorithm is coded in Matlab and the tests are conducted on Ultra 5-333 computer with 333MHz CPU and 128MB RAM running Solaris 2.8. We have not used the parallel capability of the computer in our test, that is, the program is run on one CPU only. Moreover, all reported timing results are measured in CPU minutes. Let us introduce the notation as follows:

n: number of variables

m_1: number of inequality constraints

m_2: number of equality constraints

it : mean number of iterations

Gap : value of merit function

G_1 : value of duality gap

G_2 : value of feasibility

error: the absolute error of the optimal objective value.

4.1 Single-constraint problems

First we consider convex programming problem with one constraint.

Problem 4.1

$$\begin{aligned} \min \quad & \mathbf{1}^T x \\ s.t. \quad & Ax = b \\ & x^T x - n \leq 0 \end{aligned}$$

n	m_2	it	μ	Gap	CPU time	error
10	2	40.6	7.87E-07	G1 =1.20E-05 G2 =1.48E-12	0.0499	1.23×10^{-5}
100	25	26.8	3.98E-07	G1 =9.16E-06 G2 =7.70E-10	0.2849	2.37×10^{-5}
150	38	25.9	2.62E-07	G1 =4.34E-06 G2 =2.34E-09	0.5690	1.98×10^{-5}
200	50	42.1	5.20E-08	G1 =1.24E-5 G2=4.24E-9	2.1519	3.67×10^{-5}

Table 4.1: Numerical results for Problem 4.1

We generate 10 random problem instances for each n and our algorithm solve these instances correctly. The average results are given in Table 4.1. Random problem instances are generated as follows. We generated matrix A and vector b so that $x = -\mathbf{1}_{n\times 1}$ satisfies $Ax = b$. i.e. $x = -\mathbf{1}_{n\times 1}$ is a feasible

solution.

Problem 4.2

$$\begin{aligned} \min \quad & \mathbf{1}^T x \\ s.t. \quad & Ax = b \\ & -\sum_{i=1}^{m} \log x_i + n \leq 0 \end{aligned}$$

n	m_2	it	μ	Gap	CPU time	error
10	2	17.4	1.06E-06	G1 =4.834e-006 G2 =2.555e-013	0.0147	4.23×10^{-4}
100	25	29.2	9.16E-08	G1 =8.950e-006 G2 =4.637e-011	0.2500	5.21×10^{-5}
150	38	24.1	6.90E-08	G1 =9.658e-006 G2 =1.410e-010	0.3422	6.71×10^{-5}
200	50	25.8	4.72E-08	G1 =8.758e-006 G2 =2.278e-010	0.7150	7.21×10^{-5}

Table 4.2: Numerical results for Problem 4.2

The constraint of this problem involves the logarithmic function. The numerical results are shown in Table 4.2.

4.2 Multi-constraint problems

Now we consider the multi-constraint problem.

Problem 4.3

$$\begin{aligned} \min \quad & \mathbf{1}^T x \\ s.t. \quad & Ax = b \\ & x^T x - n \leq 0 \\ & \sum_{i=1}^{m} e^{x_i} - ne \leq 0 \end{aligned}$$

n	m_2	it	μ	Gap	CPU time	error
10	2	40.3	8.98E-07	G1 =1.8706e-005 G2 =1.5545e-014	0.0798	3.13×10^{-5}
100	25	30.7	6.61E-08	G1 =7.4737e-006 G2 =4.9340e-013	0.3700	2.22×10^{-5}
150	38	37.5	3.21E-08	G1 =5.1814e-006 G2 =1.8140e-012	1.2152	6.14×10^{-5}
200	50	41.4	3.91E-08	G1 =9.1017e-006 G2 =1.9786e-012	2.6612	1.78×10^{-5}
600	150	42.9	9.50E-09	G1 =9.5900e-06 G2 =1.5802e-11	24.2459	5.19×10^{-5}

Table 4.3: Numerical results for Problem 4.3

For the two-constraint case, the constraints contain quadratic and exponential function. We observe that the number of iterations is similar as the single constraint case but the running time is longer. This is reasonable since the direction finding problem is larger.

Problem 4.4

$$\begin{aligned} \min \quad & \mathbf{1}^T x \\ s.t. \quad & Ax = b \\ & x^T x - n \leq 0 \\ & -\sum_{i=1}^{n} \log x_i \leq 0 \\ & \sum_{i=1}^{n} e^{x_i} - ne \leq 0 \end{aligned}$$

For the three-constraint case, we add logarithmic, exponential and quadratic constraints into the problems. The numerical results are shown in Table 4.4. The results show that the method is stable and quite fast under the combi-

n	m_2	it	μ	Gap	CPU time	error
10	2	26.1	3.19E-07	G1 =7.7397e-006 G2 =3.2204e-014	0.0599	2.85×10^{-5}
100	25	34.6	2.71E-08	G1 =9.7740e-006 G2 =3.2915e-013	1.0675	3.34×10^{-5}
200	50	26.3	4.72E-08	G1 =8.7584e-006 G2 =2.2782e-010	4.2184	9.29×10^{-5}
600	150	32.4	9.07E-09	G1 =2.1513e-05 G2 =6.6534e-12	30.7784	1.28×10^{-5}

Table 4.4: Numerical results for Problem 4.4

nation of these three quite different kinds of constraints.

The above problems only have a few constraints. We now want to solve problems with more inequality constraints. Thus, we generate a convex quadratic programming in the following format.

Problem 4.5

$$\begin{aligned} \min \quad & \mathbf{1}^T x \\ s.t. \quad & Ax = b \\ & x^T Q_i x + \bar{p}_i^T x + r_i \leq 0, \ for\ i = 1, ..., m_1, \end{aligned}$$

where $Q_i \succeq 0$, $\bar{p}_i \in \Re^n$ and $r_i \in \Re\ \forall i$.

We fix $m_1 = 10$ and $m_1 = 20$ respectively for the experiments. The numerical results are showed in Tables 4.5 & 4.6.

According to the experimental results, we see that the convex quadratically constrained optimization problems can be solved by our algorithm. Also, the number of iterations is insensitive to the number of inequality to the number of inequality constraints.

n	m_2	it	μ	CPU time	error
50	13	28.1	9.3570E-08	0.2544	1.34×10^{-5}
100	25	22.6	1.2858E-07	0.5388	2.37×10^{-5}
150	38	26.7	3.4517E-08	1.4334	9.27×10^{-5}
200	50	28.4	8.9996E-09	2.7720	2.44×10^{-5}
250	63	28.3	7.0660E-09	4.5445	3.37×10^{-5}
300	75	28.7	5.3687E-09	7.3843	7.69×10^{-5}
350	88	30.9	1.3110E-08	11.6029	5.67×10^{-5}
400	100	26.5	1.9246E-08	35.5398	2.39×10^{-5}

Table 4.5: Numerical results of Problem 4.5 for $m_1 = 10$

n	m_2	it	μ	CPU time	error
100	25	33.4	3.4761E-08	1.9741	3.72×10^{-5}
150	38	29.8	1.8103E-08	3.1436	4.32×10^{-5}
200	50	34.3	9.2642E-09	7.0024	8.11×10^{-5}
250	63	27.5	5.9290E-08	10.2357	1.34×10^{-5}
300	75	30.1	8.2334E-09	59.3983	2.27×10^{-5}
350	88	27.9	1.1174E-08	87.9260	7.35×10^{-5}
400	100	28.2	5.1696E-09	155.3597	5.26×10^{-5}

Table 4.6: Numerical results of Problem 4.5 for $m_1 = 20$

4.3 Running time and the size of the problem

In the following, we study in a more structured way the relationship between the running time and the size of the problem. In particular, we wish to find the relationship between the running time and the number of the constraints in the problem. We fix the number of variables to be 100 and the problem has the following format.

Problem 4.6

$$\begin{aligned} \min \quad & \mathbf{1}^T x \\ s.t. \quad & Ax = b \\ & f_{1k} = x^T Q_k x + \bar{p}_k^T x + r_k \leq 0, \\ & f_{2k} = -\sum_{i=1}^n d_{ik} \log x_i \leq 0 \\ & f_{3k} = \sum_{i=1}^n l_{ik} e^{x_i} - ne \leq 0 \quad for\ k = 1, 2, ..., m. \end{aligned}$$

where $\sum_{i=1}^n d_{ik} = \sum_{i=1}^n l_{ik} = n$ and $Q_k \succeq 0\ \forall k$.

Here, the number of the constraints is given by $m_1 = 3m$. We change the value of m from 2 to 15 and solve the problem for each m, we get the following results.

m_1	CPU time	it	m_1	CPU time	it
6	0.9999	30.2	27	4.2565	36.8
9	1.1524	26.7	30	7.1943	44.6
12	1.6147	28.9	33	7.4618	40.9
15	2.5015	29.3	36	9.3294	30.2
18	3.1902	30.1	39	11.4079	41.3
21	3.5103	34.7	42	11.9733	38.2
24	4.0299	42.5	45	12.0513	40.4

Table 4.7: Numerical results of Problem 4.6 for $n = 100$

We plot the running time against the number of the constraints. The solid line is based on the linear regression. It is showed in Figure 4.1.

One observes a linear growth pattern for the CPU time in terms of the number of constraints. Moreover the number of iteration is similar for different m. Next, we increase the number of variable to 200. Similar results are observed. The results are shown in Table 4.8 and Figure 4.2.

Then, we set out to check how the number of iterations is related to the number of the decision variables. Therefore we fix the number of constraints

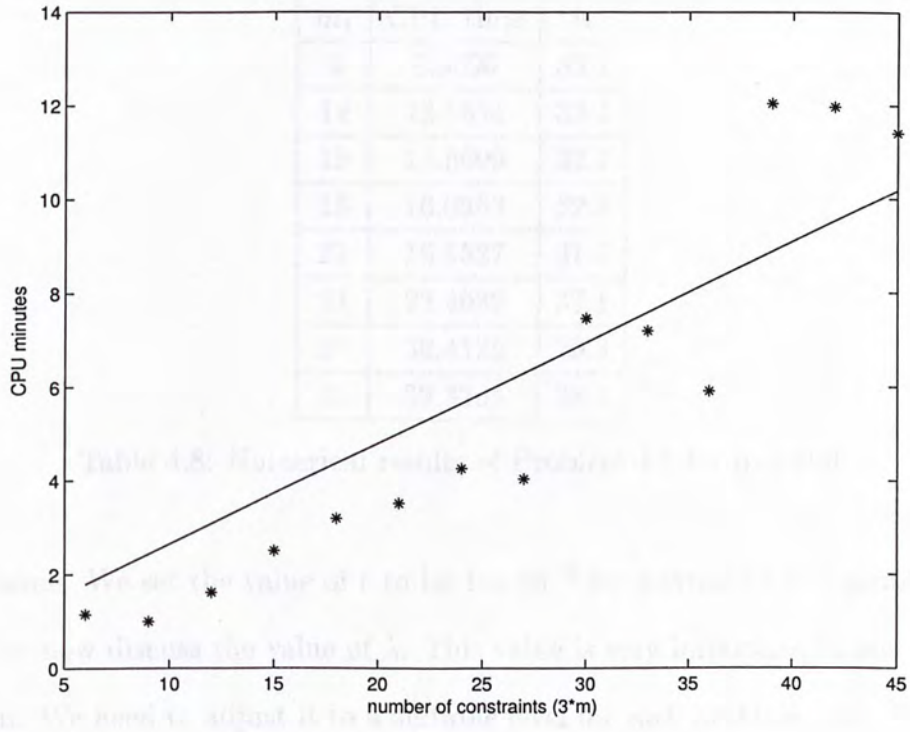

Figure 4.1: Running time against the no. of the constraints ($n = 100$)

to be 21, i.e. $m = 7$ and do some experiments for varying number of variables.

Table 4.9 shows the experimental results. We can see that the number of iterations is insensitive to the number of variables.

Now we are in the position to talk about the value of algorithmic parameter. All problems are solved by using the value of parameters as shown in Table 4.10

First of all, we are concerned with three parameters β_1, β_2 and β_3 in conditions (3.12) - (3.14). Based on our experiments, we find that the second condition (3.13) is most difficult to satisfy, so β_2 is always less than or equal to the other two parameters. The values for β_1 and β_3 are always set to be

m_1	CPU time	it
9	8.3696	33.2
12	13.1694	33.4
15	13.5609	32.1
18	16.0863	32.3
21	16.5527	31.7
24	22.4689	37.1
27	30.4125	29.3
30	39.3208	38.5

Table 4.8: Numerical results of Problem 4.6 for $n = 200$

the same. We set the value of ϵ to be 1×10^{-5} for solving all test problems.

We now discuss the value of λ. This value is very important in our algorithm. We need to adjust it to a suitable level for each problem type. Then, for each problem type, we use the same algorithmic parameters to do the experiments for different n. We conclude that the best values of algorithmic parameters are independent on the size of problem but are dependent on the structure of the problem. Thus, we may need to adjust the values of the algorithmic parameters for each new problem type.

4.4 Chapter summary

In this chapter, we applied our algorithm to solve on a number of randomly generated problem instances. The numerical results show the stability and accuracy of the method. In addition, the relationship between the number of constraints and the computational time is plotted. According to the experimental results, we observe a linear growth pattern. Also, the number of

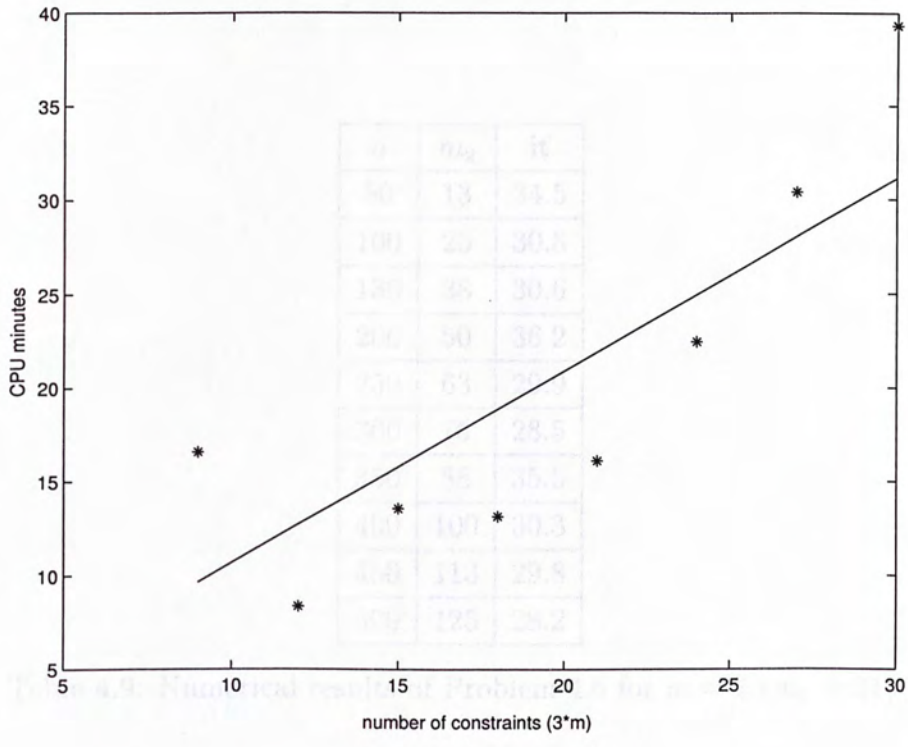

Figure 4.2: Running time against the no. of the constraints ($n = 200$)

iterations is rather insensitive to the size of the problem. Besides, the best values for algorithmic parameters are not very much dependent on the size of problem, but are indeed dependent on the structure of problem.

n	m_2	it
50	13	34.5
100	25	30.8
150	38	30.6
200	50	36.2
250	63	29.9
300	75	28.5
350	88	35.5
400	100	30.3
450	113	29.8
500	125	28.2

Table 4.9: Numerical results of Problem 4.6 for $m = 7$ ($m_1 = 21$)

	ϵ	β_1	β_2	β_3	λ
Problem 4.1	1×10^{-5}	1×10^{-4}	1×10^{-8}	1×10^{-4}	$0.8 \sim 1$
Problem 4.2	1×10^{-5}	1×10^{-6}	1×10^{-8}	1×10^{-6}	$0.8 \sim 1$
Problem 4.3	1×10^{-5}	1×10^{-6}	1×10^{-8}	1×10^{-6}	$0.8 \sim 1$
Problem 4.4	1×10^{-5}	1×10^{-8}	1×10^{-8}	1×10^{-8}	$0.8 \sim 1$
Problem 4.5	1×10^{-5}	1×10^{-10}	1×10^{-10}	1×10^{-10}	0.7
Problem 4.6	1×10^{-5}	1×10^{-10}	1×10^{-10}	1×10^{-10}	0.8

Table 4.10: Values of algorithmic parameters.

Chapter 5

Geometric optimization

In this chapter, we consider geometric optimization problem and apply our method to solve this type of problems.

5.1 Geometric programming

5.1.1 Monomials and posynomials

A function $g : \Re^n \to \Re$ with $dom\ g = \Re^n_{++}$, defined as

$$g(t) = ct_1^{a_1} t_2^{a_2} \cdots t_n^{a_n},$$

where $c \geq 0$ and $a_i \in \Re$, is called a monomial function, or simply, a monomial. The exponents a_i of a monomial can be any real numbers. A sum of monomials, i.e. a function of the form

$$g(t) = \sum_{i=1}^{N} c_i t_1^{a_{1i}} t_2^{a_{2i}} \cdots t_n^{a_{ni}},$$

where $c_j \geq 0$, is called a posynomial function (with N terms), or simply, a posynomial. If a posynomial is multiplied by a monomial, the result is a posynomial. A posynomial can be divided by a nonzeros monomial, resulting also in a posynomial.

5.1.2 Geometric programming

The primal geometric programming problem is as follows.

Primal problem:

$$\begin{array}{ll} \min & g_0(t) \\ s.t. & g_1(t) \leq 1, \cdots, g_p(t) \leq 1, \\ & t_1 > 0, \cdots, t_m > 0 \end{array}$$

where

$$g_k(t) = \sum_{i \in J[k]} c_i t_1^{a_{i1}} t_2^{a_{i2}} \cdots t_m^{a_{im}}$$

for $k = 0, 1, ..., p$, and

$$J[k] = \{m_k, m_k + 1, m_k + 2, ..., n_k\}$$

for $k = 0, 1, ..., p$ and

$$m_0 = 1,\ m_1 = n_0 + 1,\ m_2 = n_1 + 1, ...,\ m_p = n_{p-1} + 1,\ n_p = n.$$

The exponents a_{ij} can be any real number, but the coefficients c_i are assumed to be positive, i.e., the function $g_k(t)$ are posynomials. The posynomial to be minimized, namely $g_0(t)$, is termed the primal function, and the variables $t_1,\ t_2, ...,\ t_m$ are called primal variables.

The dual program corresponding to primal program is the following:

Dual problem:

$$\max\ v(\delta) = \left[\prod_{i=1}^{n} \left(\frac{c_i}{\delta_i}\right)^{\delta_i}\right] \prod_{k=1}^{p} \lambda_k(\delta)^{\lambda_k(\delta)},$$

where

$$\lambda_k(\delta) = \sum_{i \in J[k]} \delta_i,$$

for $k = 1, 2, ..., p$. The factors c_i are assumed to be positive and the vector $\delta = (\delta_1, ..., \delta_n)$ is subject to the following linear constraints: for $i = 1, ...n$, and $j = 1, 2, ..., m$

$$\delta_i \geq 0,$$
$$\textstyle\sum_{i \in J[0]} \delta_i = 1,$$

and

$$\sum_{i=1}^{n} a_{ij}\delta_i = 0.$$

5.1.3 Geometric program in convex form

In general, geometric program is not a convex optimization problem in its original form, but it can be transformed into a convex problem by the change of variables: $x_i = \log t_i$ so $t_i = e^{x_i}$, $i = 1, ..., n$. If g is the monomial function of t, i.e.,

$$g(t) = c t_1^{a_1} t_2^{a_2} \cdots t_n^{a_n},$$

then

$$\begin{aligned} g(t) &= g(e^{x_1}, \ldots, e^{x_n}) \\ &= c(e^{x_1})^{a_1} \cdots (e^{x_n})^{a_n} \\ &= e^{a^T x - b}, \end{aligned}$$

where $b = -\log c$. The change of variables $x_i = \log t_i$, $i = 1, ..., n$, turns a monomial function into the exponential of an affine function. Similarly, g is a posynomial, i.e,

$$g(t) = \sum_{k=1}^{N} c_k t_1^{a_{1k}} t_2^{a_{2k}} \cdots t_n^{a_{nk}},$$

then after the variable change, we have

$$g(x) = \sum_{k=1}^{N} e^{a_k^T x - b_k},$$

where $a_k = (a_{1k}, \ldots, a_{nk})$ and $b_k = \log c_k$, namely, after the change of variables, a posynomial becomes a sum of exponentials of affine functions. The geometric program can be expressed in terms of the new variable x as

$$\begin{array}{ll} \min & \sum_{k=1}^{N_0} e^{a_{0k}^T x - b_{0k}} \\ \text{s.t.} & \sum_{k=1}^{N_i} e^{a_{ik}^T x - b_{ik}} \leq 1, \ i = 1, ..., m. \end{array} \tag{5.1}$$

5.2 Conic transformation

The system (5.1) is readily seen to be equivalent to

$$\begin{array}{ll} \min & e^{x_0} \\ \text{s.t.} & \sum_{k=1}^{N_0} e^{a_{0k}^T x - b_{0k}} \leq e^{x_0} \\ & \sum_{k=1}^{N_i} e^{a_{ik}^T x - b_{ik}} \leq 1, \ i = 1, ..., m, \end{array}$$

where we introduce a new variable x_0 to express the posynomial objective. Noticing that minimizing e^{x_0} amount to minimizing x_0, we can rewrite this last problem as

$$\begin{array}{lll} (GP) & \min & c^T x \\ & \text{s.t.} & \sum_{i=1}^{N_0} e^{a_0^T x - x_0 - b_{0k}} - 1 \leq 0 \\ & & \sum_{i=1}^{N_j} e^{a_i^T x - b_i} - 1 \leq 0, \ j = 0, 1, ..., m. \end{array}$$

By adding N variables into the problem, with $N = \sum_{j=0}^{m} N_j$, namely, by introducing

$$x_{n+N_{j-1}+i} := -a_i^T x + b_i$$

for $i = 1, ..., N_j$, $j = 0, 1, ..., m$ and $N_{-1} = 0$. Problem (GP) is equivalently transformed into the following problem

$$\begin{array}{lll} (NGP) & \min & c^T x \\ & \text{s.t.} & Ax = b \\ & & \sum_{i=1}^{N_j} e^{-x_{n+N_{j-1}+i}} - 1 \leq 0, \ j = 0, 1, ...m. \end{array}$$

We can now represent the geometric programming in the conic form (PCGP) since the constraint are clearly smooth convex. For completeness, we would like to formulate the dual of (PCGP). For our approach, we need to find the conjugate of $f_j(x)$ with $f_j(x) = \sum_{i=1}^{N_j} e^{-x_{n+N_{j-1}+i}} - 1$, which is defined by

$$f_j^*(s) = \sup_x \{-x^T s - \sum_{i=1}^{N_j} e^{-x_i} + 1\}.$$

By putting $g(x) = -x^T s - \sum_{i=1}^{n} e^{-x_i} + 1$, let us compute the conjugate of $g(x)$. For any $s > 0$,

$$\nabla g(x) = -s + \begin{bmatrix} e^{-x_1} \\ \vdots \\ e^{-x_n} \end{bmatrix}$$

and so the first order optimality condition $\nabla g(x) = 0$ leads to

$$\begin{aligned} \sup_x g(x) &= \sum_{i=1}^{n} s_i \log s_i - \sum_{i=1}^{n} s_i + 1 \\ &= \sum_{i=1}^{n} s_i \left[\log s_i - 1\right] + 1 \end{aligned}$$

for $s_i > 0$ and $i = 1, ...n$. Thus the conjugate of function $g(x)$ is

$$g^*(s) = \sum_{i=1}^{n} s_i \left[\log s_i - 1\right] + 1$$

defined on $\Re^n_{++}$. Therefore, the primal and dual geometric problems in the conic form are

$$\begin{aligned} (PCGP) \quad \min \quad & c^T \bar{x} \\ \text{s.t.} \quad & A\bar{x} = b \\ & \bar{x} \in \mathcal{K} \end{aligned}$$

and

$$\begin{aligned} (DCGP) \quad \max \quad & b^T y \\ \text{s.t.} \quad & A^T y + \bar{s} = c \\ & \bar{s} \in \mathcal{K}^* \end{aligned}$$

where

$$\mathcal{K} = \text{cl} \bigcap_{j=0}^{m} \{\bar{x} = \begin{bmatrix} p \\ q \\ x \end{bmatrix} \mid p > 0,\ q - pf_j(x/p) \geq 0\}$$

and its dual is

$$\mathcal{K}^* = \text{cl} \sum_{j=0}^{m} \{\bar{s}_j = \begin{bmatrix} u_j \\ v_j \\ s_j \end{bmatrix} \mid v_j > 0,\ u_j - v_j f_j^*(s_j/v_j) \geq 0\}.$$

5.3 Computational results of geometric optimization problem

We shall now apply our approach to solve the geometric programming problems as reported in Dow's paper [10] and then present our numerical results. In addition, we generated some additional geometric programming problems and then solve them by our method.

Problem 5.1

$$\begin{aligned} \min \quad & 0.5t_1t_2^{-1} - t_1 - 5t_2^{-1} \\ s.t. \quad & 0.01(t_2t_3^{-1} + t_1) + 0.0005t_1t_3 \ \leq \ 1. \end{aligned}$$

We can transform the above problem to the following problem (see Murray [10] for the details)

$$\begin{aligned} \min \quad & t_4 \\ \text{s.t.} \quad & 0.01(t_2t_3^{-1} + t_1) + 0.0005t_1t_3 \leq 1 \\ & (0.2423t_1^{0.5172}t_2^{-0.9957} + 44.8261t_1^{-0.4828}t_2^{0.0043})t_4^{-0.5129} \leq 1. \end{aligned}$$

and then we use our method to solve it. By putting $t_i = e^{x_i}$ for $i = 1, ..., 4$, we have

$$\begin{aligned} \min \quad & e^{x_4} \\ \text{s.t.} \quad & e^{x_2 - x_3 + \ln 0.01} + e^{x_1 + \ln 0.01} + e^{x_1 + x_3 + \ln 0.0005} \leq 1 \\ & e^{0.5172x_1 - 0.9957x_2 - 0.5129x_4 + \ln 0.2423} + e^{-0.4828x_1 + 0.0043x_2 - 0.5129x_4 + \ln 44.8261} \leq 1. \end{aligned}$$

This problem is equivalent to the following problem.

$$\begin{array}{lll} \min & x_4 & \\ \text{s.t.} & x_2 - x_3 + x_5 & = -\ln 0.01 \\ & x_1 + x_6 & = -\ln 0.01 \\ & x_1 + x_3 + x_7 & = -\ln 0.0005 \\ & 0.5172x_1 - 0.9957x_2 - 0.5129x_4 & = -\ln 0.2423 \\ & -0.4828x_1 + 0.0043x_2 - 0.5129x_4 & = -\ln 44.8261 \\ & e^{-x_5} + e^{-x_6} + e^{-x_7} \leq 1 & \\ & e^{-x_8} + e^{-x_9} \leq 1 & \end{array}$$

and hence we can apply our algorithm to solve it. Similarly, our algorithm can be used to solve the following problems as introduced in Rijckaert and Martens [34].

Problem 5.2

$$\begin{array}{ll} \min & t_1 t_2 - t_1^{-1} t_2^{-1} \\ \text{s.t.} & 0.25 t_1^{0.5} + t_2 \leq 1. \end{array}$$

We transform this to

$$\begin{array}{ll} \min & t_3 \\ \text{s.t.} & 0.25 t_1^{0.5} + t_2 \leq 1 \\ & t_1 t_2 t_3^{-1} + t_1^{-1} t_2^{-1} t_3^{-1} \leq 1 \end{array}$$

Put $t_i = e^{x_i}$ for $i = 1, ..., 3$, then we have

$$\begin{array}{ll} \min & e^{x_3} \\ \text{s.t.} & e^{0.5x_1 + \ln 0.25} + e^{x_2} \leq 1 \\ & e^{x_1 + x_2 - x_3} + e^{-x_1 - x_2 - x_3} \leq 1. \end{array}$$

It is equivalent to

$$\begin{array}{lll} \min & x_3 & \\ \text{s.t.} & 0.5x_1 + x_4 & = -\ln 0.25 \\ & x_2 + x_5 & = 0 \\ & x_1 + x_2 - x_3 + x_6 & = 0 \\ & -x_1 - x_2 - x_3 + x_7 & = 0 \\ & e^{-x_4} + e^{-x_5} \leq 1 & \\ & e^{-x_6} + e^{-x_7} \leq 1. & \end{array}$$

Problem 5.3

$$\begin{aligned} \min \quad & t_1^2 + t_2^2 - 4t_1 - 2t_2 + 6.475 \\ \text{s.t.} \quad & 0.25t_1^2 + t_2^2 \leq 1 \\ & -t_1 + 2t_2 \geq 1. \end{aligned}$$

We transform this to

$$\begin{aligned} \min \quad & t_3 \\ \text{s.t.} \quad & 0.25t_1^2 + t_2^2 \leq 1 \\ & 0.5t_1t_2^{-1} + 0.5t_2^{-1} \leq 1 \\ & (0.2162t_1^{1.7293}t_2^{-0.2030} + 0.2162t_1^{-0.2707}t_2^{1.7970} + 1.4002t_1^{-0.2707}t_2^{-0.2030})t_3^{-0.5262} \leq 1. \end{aligned}$$

Similarly, let $t_i = e^{x_i}$ for $i = 1, ..., 3$, and we get

$$\begin{aligned} \min \quad & x_3 \\ \text{s.t.} \quad & 2x_1 + x_4 = -\ln 0.25 \\ & 2x_2 + x_5 = 0 \\ & x_1 - x_2 + x_6 = -\ln 0.5 \\ & -x_2 + x_7 = -\ln 0.5 \\ & 1.7293x_1 - 0.203x_2 - 0.5262x_3 + x_8 = -\ln 0.2162 \\ & -0.2707x_1 - 1.797x_2 - 0.5262x_3 + x_9 = -\ln 0.2162 \\ & -0.2707x_1 - 0.2030x_2 - 0.5262x_3 + x_{10} = -\ln 1.4002 \\ & e^{-x_4} + e^{-x_5} \leq 1 \\ & e^{-x_6} + e^{-x_7} \leq 1 \\ & e^{-x_8} + e^{-x_9} + e^{-x_{10}} \leq 1. \end{aligned}$$

The numerical results are shown in the Table 5.1. We can see that our method can be used to stably solve the geometric programming problems.

	iteration	error
Problem 5.1	22	1.66×10^{-3}
Problem 5.2	11	4.66×10^{-5}
Problem 5.3	36	7.44×10^{-4}

Table 5.1: Numerical results of Problem 5.1-5.3

According to the Section 5.2, we know that solving (GP) is equivalent to solving the following problem,

$$\begin{aligned} (NGP) \quad \min \quad & c^T x \\ \text{s.t.} \quad & Ax = b \\ & \textstyle\sum_{i=1}^{N_j} e^{-x_{n+N_{j-1}+i}} - 1 \leq 0, \ j = 1, ..., m. \end{aligned}$$

Hence we generate some additional problems in this format and to test our algorithm. Consider the following problem:

Problem 5.4

$$\begin{aligned} \min \quad & \mathbf{1}^T x \\ s.t. \quad & Ax = b \\ & \textstyle\sum_{i=j}^{n+j-m_1} e^{-x_i} - (n - m_1)e \leq 0, \ j = 1, ..., m_1. \end{aligned}$$

The numerical result for this problem are shown in Table 5.2-5.4.

Moreover, we consider the following problem:

Problem 5.5

$$\begin{aligned} \min \quad & \mathbf{1}^T x \\ s.t. \quad & Ax = b \\ & e^{-x_j} + e^{-x_{j+1}} - 2e \leq 0, \ j = 1, ..., n - 1. \end{aligned}$$

This geometric program has $n-1$ inequality-constraints. The computational results for this problem are shown in Table 5.5.

Now we discuss the value of the algorithmic parameters for solving geometric programming. In the above experiments, we use the same value of ϵ for solving all problems. The detail can be found in the Table 5.6. We can see that β_i's are set to be constant for all problems. But the value of λ is different for Problem 5.5. The difference is due to the structure of this problem. Clearly, Problem 5.5 has $n - 1$ inequality-constraints, where n is

number of variables	CPU time	iteration
50	0.5985	43
100	0.7901	27
150	2.0649	31
200	3.6052	31
250	6.2759	32
300	9.5093	30

Table 5.2: Computational results of the Problem 5.4 for $m_1 = 10$

number of variables	CPU time	iteration
50	1.3598	47
100	2.6656	36
150	7.7078	41
200	12.0421	35
250	24.4377	42
300	45.7886	38
350	108.5256	37
400	278.3176	37

Table 5.3: Computational results of Problem 5.4 for $m_1 = 20$

the number of the decision variables. Therefore, the difficulty for solving this type of problem is higher than that of the other problems discussed in this chapter. We found that if the number of constraints is dependent on the number of decision variables, then we need to decrease the value of λ. However, the value of λ cannot be set too low. In particular, if $\lambda < 0.3$, then our algorithm does not seem to converge to the optimal solution. Thus, adjusting the proper λ value is quite important for our algorithm.

number of variables	CPU time	m_1	iteration
20	0.1484	2	21
40	0.2910	4	25
60	0.5913	6	27
80	1.2311	8	29
100	5.5088	10	43
120	5.9460	12	33
140	18.1382	14	32

Table 5.4: Computational results of Problem 5.4

number of variables	CPU time	iteration
20	0.2641	30
40	1.2845	32
60	6.8604	38
80	32.6032	45
100	49.818	39
120	123.9953	37
140	274.8943	33

Table 5.5: Computational results of Problem 5.5

5.4 Chapter summary

In this chapter, we state the general primal and dual geometric programs and then transform them into the convex form. After that we use our approach to further transfer the problem into the conic form. Finally, we solve numerous test problems in this format. We conclude that our algorithm is efficient in solving geometric programming probelms.

	ϵ	β_1	β_2	β_3	λ
Problem 5.1	1×10^{-5}	1×10^{-10}	1×10^{-10}	1×10^{-10}	$0.8 \sim 0.9$
Problem 5.2	1×10^{-5}	1×10^{-10}	1×10^{-10}	1×10^{-10}	$0.8 \sim 0.9$
Problem 5.3	1×10^{-5}	1×10^{-10}	1×10^{-10}	1×10^{-10}	$0.8 \sim 0.9$
Problem 5.4	1×10^{-5}	1×10^{-10}	1×10^{-10}	1×10^{-10}	$0.8 \sim 0.9$
Problem 5.5	1×10^{-5}	1×10^{-10}	1×10^{-10}	1×10^{-10}	0.7

Table 5.6: Values of algorithmic parameters.

Chapter 6

Conclusion

In this thesis we study the new self-dual embedding method for convex programming which is proposed by Zhang [43]. Based on this paper, we can turn the general convex optimization problem into the conic form by adding 2 extra variables. Then we apply the self-dual embedding technique to solve the resulting problem. Certainly, an obvious advantage of the new approach is that it does not require an initial feasible solution of the convex program to start with, which is a generic virtue of the self-dual embedding method.

We specialize an interior-point algorithm and discuss in detail how this algorithm can be constructed and implemented. First, we use Newton's method to solve the approximative KKT system. Second, we follow the approach of Andersen and Ye [1] to construct a merit function. This merit function is used to select the step length and measure the performance in all iterations. We apply the so-constructed algorithm to solve some randomly generated test problems which involve logarithmic, exponential and quadratic constraints.

In addition, we consider geometric programming problems. We state the

general primal and dual geometric programs, and then transform them into the convex form. Finally, we apply our method and further transform the problem into the conic form. Numerical tests are carried out for this type of problems.

All numerical results show the stability and accuracy of the method. Also, we observe a linear growth pattern between the computational effort and the number of constraints. The number of iterations is insensitive to the number of decision variables. Besides, the best values for the algorithmic parameters are not very much dependent on the size of problem, but can indeed be dependent on the structure of problem.

Finally we conclude that this new self-dual embedding method is numerically stable and efficient for solving general convex programming problems.

Bibliography

[1] E.D. Andersen and Y. Ye, *A Computional Study of the Homogeneous Algorithm for Large-scale Convex Optmization*, Computational Optimization and Applications, 10, pp. 243-269, 1998.

[2] E.D. Andersen and Y. Ye, *On a Homogeneous Algorithm for Monotone Complementary Problem*, Mathematical Programming, 84, pp. 375-399, 1999.

[3] F. Alizadeh, *Combinatorial Optimization with Interior Point Methods and Semi-definite Matrices*, PhD thesis, University of Minnesota, Minneapolis, USA, 1991.

[4] K.M. Anstreicher, *On Long Step Path Following and SUMT for Linear and Quadratic Programming*, SIAM Journal on Optimization, 6, pp. 33-46, 1996.

[5] E.R. Barnes, *A Variation on Karmårker's Algorithm for Solving Linear Programming Problems*, Mathematical Programming, 36, pp. 174-182, 1986.

[6] C. Beightler and D. Phillips, *Applied Geometric Programming*, John Wiley and Sons, New York, 1976.

[7] S.E. Boyd, L.El Ghaoui, E. Feron, and V. Balakrishnan, *Linear Matrix Inequalities in System and Control Theory*, SIAM Studies in Applied Mathematics, 15, SIAM, Philadelphia, USA, 1994.

[8] R.S. Dembo, *Dual to Primal Conversion of Geometric Programming*, Journal of Optimization Theory and Applications, 26, 1978.

[9] I.I. Dikin, *Iterative Solution of Problems of Linear and Quadratica Programming*, Doklady Akademii Nauk SSSR, 174, pp. 747-748, 1967. (Translated in: Soviet Mathematics Doklady, 8, pp. 674-675, 1967).

[10] M. Dow, *A Fortran Code for Geometric Programming*, Supercomputer Facility Australian National University Canberra Australia, http://anusf.anu.edu.au/ mld900/math/, August 25, 1999.

[11] K.R. Frisch, *The Logarithmic Potential Method for Convex Programming*, Unpublished Manuscript, Institute of Economics, University of Oslo, Oslo, Noeway, 1955.

[12] M.R. Garey, and D.S. Johnson, *Computers and Intractability: A Guide to the Theory of NP-Completeness*, W.H. Freeman and Company, Publishers, San Fransisco, USA, 1979.

[13] P.E. Gill, W. Murray, M.A. Saunders, J.A. Tomlin, and M.H. Wright, *On Projected Newton Barrier Methods for Linear Programming and an Equivalence to Karmarlar's Projective Method*, Mathematical Programming, 36, pp. 183-209, 1986.

[14] F. Glineur, *Topics in Convex Optimization: Interior-point Methods, Conic Duality and Approximations*, Ph.D. thesis, Faculté Polytechnique de Mons, Belgium, December, 2000.

[15] F. Glineur, *Proving Strong Duality for Geometric Optimization using a Conic Formulation*, Annals of Operations Research, 105, pp. 155-184, July, 2001.

[16] M. Grötschel, L.A. Lovász, and A. Schrijver. *Geometric Algorithm and combinational optimization*, Springer Verlag, Berlin, 1988.

[17] P. Hung, *Resolution of Mathematical Programming with Nonlinear Constraints by the Method of Centers*, In J. Abadie, editor, Nonlinear programming, pp. 207-219, North Holland, Amsterdam, 1967.

[18] D.B. Judin, and A.S. Nemiroskii, *Problem Complexity and Method Efficiency in Optimization*, Wiley-Interscience, Chichester, USA, 1983.

[19] N.K. Karmarker, *A New Polynomial-time Algorithm for Linear Programming*, Combinatorica, 4, pp. 373-395, 1984.

[20] R.M. Karp, *Reducibility among Combinatorial Probblem*, In R.E. Miller and J.W. Thatcher, editors, Complextity of computions, pp. 85-103, Plenum Press, New York, 1972.

[21] L.G. Khacijan. *A Polynomial Time Algorithm in Linear Programming*, Soviet Mathematics Doklady, 20, pp. 191-194, 1979.

[22] V. Klee, and G.J. Minty,*How Good is the Simplex Algorithm?*, In O. Shisha, editor, Inequalities III. Academic Press, New York, 1972.

[23] 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 Algorithm and Related Methods*, pp. 29-47. Springer Verlag, Berlin, 1989.

[24] E. Kranich, *Interior Point Methods for Mathematical Programming: A Bibliogrphy*, Discussionsbeitrag 171, FernUniversitätHagen, Hagen, Germany, 1991.

[25] M.L. Lenard and M. Minkoff, *Randomly Generated Test Problems for Positive Definite Quadratic Programmin*, ACM Transactions on Mathematical Software, 10, pp. 86-96, 1984.

[26] Z.-Q. Luo, J.F. Sturm, and S. Zhang, *Duality Results for Conic Covex Programming*, Technical Report 9719/A, Econometric Institute, Eramus University Rotterdam, The Netherlands, 1997.

[27] Z.-Q. Luo, J.F. Sturm, and S. Zhang, *Conic Convex Programming and Self-dual Embedding*, Optimization Methods and Software, 14, pp. 169-218, 2000.

[28] N. Megiddo, *Pathways to the Optimal Set in Linear Programming*, In N. Megiddo, editor, Progress in Mathematical Programming: Interior Point and Related Methods, pp. 131-158, Spring Verlag, New York, 1989.

[29] S. Mizuno, M.J. Todd and Y. Ye, *On Adaptive-Step Primal-Dual Interior-Point Algorithm for Linear Programming*, Mathematics of Operations Research, 18, No 4, pp. 964-975, November, 1993.

[30] S Mizuno, M.J. Todd and Y. Ye, *An $O(\sqrt{n}L)$-Iteration Homogeneous and Self-Dual Linear Programming Algorithm*, Mathematics of Operations Research, 19, No 1, pp. 53-67, February, 1994.

[31] W.C. Mylander, R.L. Holmes, and G.P. MeCormick, *A guide to SUMT-Version 4: the Computer Program Implementing the Sequential Unconstrainted Minimization Technique for Nonlinear Programming*, Research Paper RAC-P-63, Research Analysis Corporation, McLean, USA, 1971.

[32] Yu. Nesterov and A. Nemirovski, *Interior Piont Ploynomial Methods in Convex Programming*, Studies in Applied Mathematics, 13, SIAM, Philadelphia, 1994.

[33] J. Renegar, *A Polynomial-Time Algorithm, Based on Newton's Method, for Linear Programming*, Mathematical Programming, 40, pp. 59-93, 1993.

[34] M.J. Rijckaert and X.M. Martens, *Comparison of Generalized Geometric Programming*, Journal of Optimization Theory and Apllications, 26, pp. 243-245, 1978.

[35] C. Roos, and J.-Ph. Vial, *A Polynomial method of Approximate Centers for Linear Programming*, Mathematical Programming, 54, pp. 295-305, 1992.

[36] G. Sonnevend, *An "Analytic Center" for Polyhedrons and New Classes of Global Algorithm for Linear (smooth convex) Programming*, In A. Prekopa, J. Szelezsan and B. Strazicky, editors, System Modelling and optimization: Proceedings of the 12th IFIP-Conference held in Budapest Hungary, September 1985, 84 of Lecture Notes in Control and Information Sciences, pp. 866-876, Spring Verlag, Berlin, Germany, 1986.

[37] J.F. Sturm, *Duality (Chapter 2)*, in High Performance Otimization, eds. H. Frenk, T. Terlaky, K. Roos and S. Zhang, Kluwer Academic Publishers, Dordrecht, The Netherlands, 2000.

[38] R.J. Vanderbei, M.S. Meketon, and B.A. Freedman, *A Modification of Karmarkar's Linear Programming Algorithm*, Algorithmica, 1, pp. 395-407, 1986.

[39] L. Vanderbei, and S.E. Boyd, *Semidefinite Programming*, SIAM Review, 38, pp. 49-95, 1996.

[40] J.-Ph Vial, *Computational Experience with a Primal-Dual Interior-Point Method for Smooth Convex Programming*, Optimization Methods and Software, 3, 285-316, 1994.

[41] X. Xu, P.F. Hung, and Y. Ye, *A Simplified Homogeneous Self-dual Llinear Programming algorithm and its implementation*, Annals of Operations Research, 62, pp. 151-171, 1996.

[42] H. Yamashita, *A Globally Convergent Primal-Dual Interior Point Method for Constrainted Optimization*, Mathematical System Institute, Inc., Tokyo, Japan, 1992.

[43] S. Zhang, *A New Self-Dual Embedding Method for Convex Programming*, SEEM Report 2001- 09, the Chinese University of Hong Kong, 2001. To appear in Joural of Global Optimization.