Fall 2009

Kernel-Based Interior-Point Algorithms for the Linear Complementarity Problem

Jason N. Brandies

KERNEL-BASED INTERIOR-POINT ALGORITHMS FOR THE LINEAR COMPLEMENTARITY PROBLEM

by

JASON N. BRANDIES

(Under the Direction of Dr. Goran Lesaja)

ABSTRACT

In this thesis, we consider the Linear Complementarity Problem (LCP), which is a well-known mathematical problem with many practical applications. The objective of the LCP is to find a certain vector that will satisfy a set of linear inequalities and (non-linear) complementarity equation. A kernel-based primal-dual Interior-Point Method (IPM) for solving LCP was introduced and analyzed. The class of kernel functions used in this thesis is a class of so-called eligible kernel functions that are fairly general. We have shown for a positive semi-definite matrix M, that the algorithm is globally convergent and has very good convergence properties. For some instances of the eligible kernel functions, the complexity of the algorithm, in terms of the number of iterations, considered in this thesis matches the best complexity results obtained in the literature for these types of methods. This is the main emphasis of the thesis. The theoretical concepts were illustrated by basic implementation in MATLAB for the classical kernel function ψ_1 and for the parametric kernel function ψ_{10} (Table 3.3). A series of numerical tests were conducted that shows that even these basic implementations have a potential for good performance. Better implementation and more numerical testing would be necessary to draw more definite conclusions.

Index Words: kernel function, primal-dual, interior-point method, linear complementarity problem

2010 *Mathematics Subject Classification*: 90C33, 90C51

KERNEL-BASED INTERIOR-POINT ALGORITHMS FOR THE LINEAR COMPLEMENTARITY PROBLEM

by

JASON N. BRANDIES

B.S. in Mathematics, Georgia Southern University, 2007

A.S. in Computer Science, Coastal Georgia Community College, 2005

A Thesis Submitted to the Graduate Faculty of Georgia Southern University in Partial Fulfillment of the Requirement for the Degree

MASTER OF SCIENCE

STATESBORO, GEORGIA

2009

KERNEL-BASED INTERIOR-POINT ALGORITHMS FOR THE LINEAR COMPLEMENTARITY PROBLEM

by

JASON N. BRANDIES

Major Professor: Dr. Goran Lesaja

Committee: Dr. Scott Kersey

Dr. Billur Kaymakcalan

Electronic Version Approved:

December, 2009

DEDICATION

This thesis is dedicated to my entire family. My family has made my educational dreams a reality. My family has provided me with their undying love and support throughout my life. I would like to extend a thank you, from the bottom of my heart, to my family. This thesis is for: Brooke, Mom, Dad, Sandy, Ashley, Grandma, Gran, Pop, Nan, Nana, Papa, and Riley. Of course, a very special thank you to my fiancé Brooke for putting up with me throughout this wild adventure. We have made many great memories together and I want us to make countless more. I love you.

ACKNOWLEDGMENTS

First and foremost, I would like to extend my deepest gratitude to Dr. Goran Lesaja for all of his guidance through, not only this thesis, but my entire college career at Georgia Southern University. Dr. Lesaja has been a mentor for many years at GSU. I would like to give a special thank you to my committee members, Dr. Scott Kersey and Dr. Billur Kaymakcalan. I also acknowledge every member of the outstanding faculty and staff in the Georgia Southern University Mathematical Sciences Department. Their hard work and dedication to the study of mathematics has made, not only this thesis, but also my educated future possible.

James $\geq$ Mike $\geq \epsilon$

TABLE OF CONTENTS

LIST OF TABLES

Table Page

LIST OF FIGURES

Figure Page

CHAPTER 1
INTRODUCTION

1.1 The Problem - A Brief Overview

The Linear Complementarity Problem (LCP) is an important problem with rich theory, numerous efficient algorithms, and a plethora of practical applications in a variety of areas. Some instances of the LCP can be traced back to the early 1940's; however, larger interest in LCP was taken in the early to mid 1960's. Since then, many publications and research have been devoted to studying this problem and its many properties.

The LCP is not an optimization problem. However, it is closely related to optimization problems because Kurush-Kuhn-Tucker (KKT) optimality conditions for many optimization problems can be formulated as the LCP. For example, KKT conditions of linear and quadratic optimization problems can be formulated as LCP. In addition there are problems that can be directly formulated as LCP. This is the reason why the LCP is often considered as a problem in the mathematical programming area with applications that include, but are not limited to economics, engineering (game and equilibrium theory), transportation, and many other areas of operations research.

The objective of the LCP is to find a vector in a finite real vector space that satisfies a certain system of linear equations and a nonlinear complementarity equation. In the sequel, we will explore these in more detail.

The LCP has several different formulations, including the standard, mixed, horizontal, and geometric formulation. We may choose from the different formulations

for many different reasons, such as: efficiency, given initial conditions, or even for certain output wanted. We may also use special types of matrices in the different formulations. Since the general LCP is NP-complete, i.e, there exists no polynomial algorithms for solving it, we may consider special types of the LCP problem for which a polynomial algorithm exists.

1.2 The Methods - A Brief Overview

A Linear Programming (LP) model determines the "best" outcome in a particular mathematical model given certain requirements. These outcomes can range from raising profits, reducing costs, to managing network flow. The LP has vast numbers of practical applications, in many areas. For example, company management and economics have great use for the LP. In the modern business environment, where efficiency and maximizing profit while minimizing costs is important, the LP is an important mathematical tool that helps achieve these goals. That is why the LP is one of the most important branches of optimization.

The LP can be mathematically defined as a technique for the optimization (minimizing or maximizing) of a *linear objective function*

$$z = c_1x_1 + c_2x_2 + c_3x_3 + \ldots + c_nx_n,$$

subject to a number of *linear constraints* of the form

$$a_{i1}x_1 + a_{i2}x_2 + a_{i3}x_3 + \ldots + a_{in}x_n \left\{ \begin{array}{c} \geq \\ \leq \\ = \end{array} \right\} b_i;\ i = 1, \ldots, n$$

The linear constraints define the *feasible region.* Geometrically, the feasible region takes the form of a polyhedral, i.e, a convex hull of a finite set of vertices (points).

The resulting largest (or smallest) value of the objective function is called the *optimal value*. The vector

$$x = (x_1, x_2, x_3, \ldots, x_n),$$

for which the optimal value is achieved is called the *optimal solution*.

For this mathematical model, we needed some effective methods for solving it. In 1947, George Dantzig was the first mathematician to successfully develop a method to solving the LP. He created the *Simplex Method* (SM) to numerically solve the LP. Basically, the main idea of the SM is to travel along from vertex to vertex on the boundary of the feasible region. The method constantly increases (or decreases) the objective function until either an optimal solution is found or the SM concludes that such an optimal solution does not exist.

Theoretically, the algorithm could have a worse-case scenario of 2^n iteration, with n being the size of the problem, which is an exponential number. This was shown in 1972 by Klee and Minty [8]. However, on behalf of the SM, it is remarkably efficient in practice and an exponential number of iterations has never been observed in practice. It usually requires $O(n)$ iterations to solve a particular problem. There exists many resources and excellent software for the SM.

Another great advancement in the area of solving convex optimization problems was the *ellipsoid method*. This method was introduced by Nemirovsky and Yudin in 1976 [19] and by Shor in 1977 [16]. The algorithm works by encapsulating the minimizer of a convex function in a sequence of ellipsoids whose volume decreases at each iteration. Later Khachiyan showed in 1984 that the ellipsoid method can be used to solve the LP in polynomial time [7]. This was the first polynomial time algorithm for the LP. Unfortunately, in practice, the method was far surpassed by the simplex

method. Nevertheless, the theoretical importance of the ellipsoid method is hard to neglect.

In 1984, Karmarkar introduced an *Interior-Point Method* (IPM) for LP [6]. Karmarkar used the efficiency of the simplex method with the theoretical advantages of the ellipsoid method to create his efficient polynomial algorithm. The algorithm is based on projective transformations and the use of Karmarkar's primal potential function. This new algorithm sparked much research, creating a new direction in optimization - the field of IPMs. Unlike the SM, which travels from vertex to vertex along the edges of the feasible region, the IPM follows approximately a central path in the interior of the feasible region and reaches the optimal solution only asymptotically. As a result of finding the optimal solution in this fashion, the analysis of the IPMs become substantially more complex than that of the SM.

Since the first IPM was developed, many new and efficient IPM algorithms for solving LP have been created. Many researches have proposed different interior-point methods, which can be grouped into two different groups: potential reduction algorithms and path-following algorithms. Each of the two groups contains algorithms based on primal, dual, or primal-dual formulations of the LP. Also, computational results show that the primal-dual formulation is superior to either the primal or dual formulation of the algorithm. We will focus on the *primal-dual* path-following IPMs, which have become the standard of efficiency in practical applications. These primal-dual methods are based on using Newton's method in a careful and controlled manner.

Soon after the SM was developed, a similar method for solving LCP was introduced by Lemke [10]. It is a pivoting algorithm similar to the SM. Unfortunately,

Lemke's algorithm can sometimes fail to produce a solution even if one exists. Nevertheless, Lemke's algorithm was extremely useful. However, researchers kept searching for other methods for the LCP. Much later, in the 1990's, the ritual of immediate generalizations from LP to LCP continued even more strongly in the case of the IPMs [9]. In this thesis, we will focus on extending a class of primal-dual IPMs, the so called kernel-based IPMs, from LP to LCP.

In addition, IPMs have been generalized to solve many other important optimization problems, such as semidefinite optimization, second order cone optimization, and general convex optimization problems. The unified theory of IPMs for general convex optimization problems was first developed by Nesterov and Nemirovski in 1994 [12].

The first comprehensive monograph that considers in-depth analysis of the LCP and methods for solving it is the monograph of Cottle, Pang, and Stone [3]. More recent results on the LCP as well as nonlinear complementarity problems and variational inequalities are contained in the monograph of Facchinei and Pang [4].

CHAPTER 2

THE LINEAR COMPLEMENTARITY PROBLEM

In this chapter the linear complementarity problem is introduced, defined, and discussed. Also, several direct applications of the linear complementarity problem are presented and discussed.

2.1 The Introductory Examples

The linear complementarity problem, LCP, has many applications. Some examples of the LCP include but are by far not limited to: the bimatrix game, optimal invariant capital stock, optimal stopping, convex hulls in the plane, and the market equilibrium problems. Each one of the listed problems can be reformulated into the linear complementarity problem. In the sequel, we will describe several applications.

Example 1: The Market Equilibrium Problem

The state of an economy where the supplies of producers and the demands of consumers are balanced at the resulting price level is called *market equilibrium* . We can use a linear programming model to describe the supply side that captures technological details of production activities for a particular market equilibrium problem. Econometric models with commodity prices as the primary independent variables generates the market demand function. Basically, we need find vector x^* and subsequent vectors p^* and r^* such that the conditions below are satisfied for supply, demand, and equilibrium:

supply conditions:

$$\begin{aligned} \text{minimize} \quad & c^T x \\ \text{subject to} \quad & Ax \geq b \\ & Bx \geq r^* \\ & x \geq 0 \end{aligned} \tag{2.1}$$

where c is the cost vector for the supply activities, x is the vector production activities. Technological constraints on production are represented by the first condition in (2.1) and the demand requirement constraints are represented by the second condition in (2.1);

demand conditions:

$$r^* = Q(p^*) = Dp^* + d \tag{2.2}$$

where $Q(\cdot)$ is the market demand function with p^* and r^* representing the vectors of demand prices and quantities, respectively. $Q(\cdot)$ is assumed to be an affine function;

equilibrium condition:

$$p^* = \pi^* \tag{2.3}$$

where the (dual) vector of market supply prices corresponding to the second constraint in (2.1) is denoted by π^*.

Using Karush-Kuhn-Tucker conditions for problem (2.1), we see that a vector x^* is an optimal solution of problem (2.1) if and only if there exists vectors v^* and π^* such that:

$$\begin{aligned} y^* &= c - A^T v^* - B^T \pi^* \geq 0, & x^* \geq 0, \quad (y^*)^T x^* = 0, \\ u^* &= -b + Ax^* \geq 0, & v^* \geq 0, \quad (u^*)^T v^* = 0, \\ \delta^* &= -r^* + Bx^* \geq 0, & \pi^* \geq 0, \quad (\delta^*)^T \pi^* = 0. \end{aligned} \tag{2.4}$$

If for r^*, we substitute the demand function (2.2) and we use condition (2.3), then we can see that the conditions in (2.4) gives us the linear complementarity problem where

$$q = \begin{bmatrix} c \\ -b \\ -d \end{bmatrix} \quad \text{and} \quad M = \begin{bmatrix} 0 & -A^T & -B^T \\ A & 0 & 0 \\ B & 0 & -D \end{bmatrix}. \tag{2.5}$$

As it could have been seen, the Karush-Kuhn-Tucker optimization conditions of the market equilibrium problem, and in fact the linear problem in general, can be expressed in the LCP framework. This can also be extended to quadratic programming problems as discussed below.

Example 2: Quadratic Programming

Quadratic programming is another application of the linear complementarity problem. It is the problem of minimizing or maximizing a quadratic function of several variables subject to linear constraints on these variables. The quadratic program (QP) is defined as

$$\begin{aligned} \text{minimize} \quad & f(x) = c^T x + \tfrac{1}{2} x^T Q x \\ \text{subject to} \quad & Ax \geq b \\ & x \geq 0 \end{aligned} \tag{2.6}$$

where $Q \in \mathbb{R}^{nxn}$ is symmetric, $c \in \mathbb{R}^n$, $A \in \mathbb{R}^{mxn}$ and $b \in \mathbb{R}^m$. Note: The case where $Q = 0$ gives rise to a linear program (LP). If x is a locally optimal solution of the quadratic program (2.6), then there exists a vector $y \in \mathbb{R}^m$ such that the pair (x, y) satisfies the Karush-Kuhn-Tucker optimality conditions

$$\begin{aligned} u &= c + Qx - A^T y \geq 0, \quad x \geq 0, \quad x^T u = 0, \\ v &= -b + Ax \geq 0, \qquad\quad y \geq 0, \quad y^T v = 0. \end{aligned} \tag{2.7}$$

If Q is positive semi-definite (the objective function $f(x)$ is convex), then the conditions in (2.7) are sufficient for the vector x to be a globally optimal solution of (2.6).

The Karush-Kuhn-Tucker conditions in (2.6) define the LCP where

$$q = \begin{bmatrix} c \\ -b \end{bmatrix} \quad \text{and} \quad M = \begin{bmatrix} Q & -A^T \\ A & 0 \end{bmatrix}. \tag{2.8}$$

Note that M is not symmetric, even though Q is symmetric. However, M does have a property known as *bisymmetry*. A square matrix A is *bisymmetric* if it can be brought to the form

$$A = \begin{bmatrix} G & -A^T \\ A & H \end{bmatrix},$$

where both G and H are symmetric. Also, if Q is positive semi-definite, then so is M. In general, a square matrix M is positive semi-definite if $z^T M z \geq 0$ for every vector z.

This convex quadratic programming model, in the form of (2.6), has a magnitude of practical applications in engineering, finance, and many other areas. The size of these practical problems can become very large. Thus, the LCP plays an important role in the numerical solution of these problems.

The previous two examples showed the close connection of the linear complementarity problem, which is not an optimization problem, to a large class of optimization problems. However, the linear complementarity problem may appear independently of optimization problems, as in a direct formulation of practical problems. In the following example, we will see how finding convex hulls in the plane gives us a direct formulation of the LCP.

Example 3: Convex Hulls in the Plane

Finding the convex hull of a given set of points is a very important problem in computational geometry. Furthermore, a special case of this problem has surfaced where all of the points lie on a particular plane. This special case has attracted much attention and has had several efficient algorithms developed for solving it.

Given a set $(x_i, y_i)_{i=0}^{n+1}$ of points in the plane, we want to find the extreme points and the facets of the convex hull in the order in which they appear. We can break this problem into two parts; first we want to find the lower collection of the given points, then we want to find the upper collection. We will denote the lower collection and upper collection as LC and UC, respectively. While we are trying to find the LC, we may assume that the x_i's are distinct (for $x_i = x_j$ and $y_i \leq y_j$, then we may ignore the point (x_j,y_j) without changing the LC). Thus, we assume $x_0 < x_1 < \cdots < x_{n+1}$.

Let $f(x)$ is the point-wise maximum over all convex functions $g(x)$ in which $g(x_i) \leq y_i$ for all $i = 0, \ldots, n+1$. The function $f(x)$ is convex and piecewise linear. The collection of breakpoints between the pieces of linearity is a subset of $(x_i, y_i)_{i=0}^{n+1}$. Let $t_i = f(x_i)$ and let $z_i = y_i - t_i$ for $i = 0, \ldots, n+1$, where z_i represents the vertical distance between the point (x_i,y_i) and the LC. We can now make several observations: $z_0 = z_{n+1} = 0$ and if (x_i,y_i) is a breakpoint, then $t_i = y_i$ and $z_i = 0$. Also, the segment of the LC between (x_{i-1},t_{i-1}) and (x_i,t_i) has a different slope than the segment between (x_i,t_i) and (x_{i+1},t_{i+1}). Since $f(x)$ is convex, then the previous segment must have a smaller slope than that of the latter segment. This implies that strict inequality holds in

$$\frac{t_i - t_{i-1}}{x_i - x_{i-1}} \leq \frac{t_{i+1} - t_i}{x_{i+1} - x_i}. \tag{2.9}$$

One last observation is that if $z_i > 0$, then (x_i, y_i) cannot be a breakpoint of $f(x)$. Hence, equality holds in (2.9).

If we combine the above observations together then we can see that the vector $z = \{z_i\}_{i=1}^n$ must solve the LCP (as defined in the following section) where $M \in \mathbb{R}^{nxn}$ and $q \in \mathbb{R}^n$ are defined by

$$q_i = \beta_i - \beta_{i-1} \quad \text{and} \quad m_{ij} = \begin{cases} \alpha_{i-1} + \alpha_i & \text{if } j = i, \\ -\alpha_i & \text{if } j = i+1, \\ -\alpha_j & \text{if } j = i-1, \\ 0 & \text{otherwise,} \end{cases} \tag{2.10}$$

where

$$\alpha_i = \tfrac{1}{x_{i+1} - x_i} \quad \text{and} \quad \beta_i = \alpha_i(y_{i+1} - y_i) \quad \text{for } i = 0, \ldots, n.$$

It can be shown that the LCP, as defined above, has a unique solution. This solution gives us the quantities $z = \{z_i\}_{i=1}^n$ which is the LC of the convex hull. Similarly, the UC can also be found. So by solving two linear complementarity problems, which both have the same matrix M, we can find the convex hull of a finite set of points in a certain plane. Let us note that in this particular case the variable of the LCP is denoted as z while in the next section the variable is denoted in a usual manner as x.

We have given several examples of practical problems, all of which can be reduced to basically solving the LCP. In the next section, we will formally define and discuss the linear complementarity problem, in more detail.

2.2 The Linear Complementarity Problem

The main idea of the linear complementarity problem, LCP, is to find a particular vector in a finite real vector space that satisfies a certain system of inequalities. Mathematically, given a vector $q \in \mathbb{R}^n$ and a matrix $M \in \mathbb{R}^{nxn}$, we want to find a vector $x \in \mathbb{R}^n$ (or to show such a vector does not exist) such that

$$\begin{aligned} x &\geq 0, \\ q + Mx &\geq 0, \\ x^T(q + Mx) &= 0. \end{aligned} \tag{2.11}$$

By introducing a vector,

$$s = q + Mx, \tag{2.12}$$

the above system (2.11) can be rewritten to give us this useful equivalent formulation:

$$\begin{aligned} s &= Mx + q, \\ x^T s &= 0, \\ (x, s) &\geq 0. \end{aligned} \tag{2.13}$$

Since $(x, s) \geq 0$, the complementarity equation $x^T s = 0$ can be written equivalently as

$$xs = 0,$$

which represents component-wise product of vectors, as follows,

$$xs = (x_1 s_1, x_2 s_2, \ldots, x_n s_n)^T. \tag{2.14}$$

The feasible set of points (feasible region) of the LCP as defined in (2.13) is the following set:

$$F = \left\{(x, s) \in \mathbb{R}^{2n} : s = Mx + q, x \geq 0, s \geq 0\right\}. \tag{2.15}$$

Furthermore, the set of strictly feasible points of the LCP is the following set:

$$F_0 = \{(x, s) \in F : x > 0, s > 0\}.$$

The solution set of the LCP is given by

$$F^* = \left\{(x^*, s^*) \in F : x^{*T} s^* = 0\right\}. \tag{2.16}$$

An important subset of the above solution set is a set of strict complementarity solutions

$$F_s^* = \{(x^*, s^*) \in F^* : x^* + s^* > 0\}. \tag{2.17}$$

We can now say that the main idea of the LCP is to find a certain vector x that is both feasible and complementary. This vector is called a solution of the LCP. The LCP is always solvable with the zero vector being a trivial solution , if $q \geq 0$.

2.3 Alternate Formulations of the LCP

In the previous section, we stated the LCP as a problem of finding a solution vector $(x, s) \in \mathbb{R}^{2n}$ such that all conditions of (2.13) are satisfied. This formulation is usually known as the *standard* LCP (SLCP). There are several other important LCP formulations that are going to be discussed, such as: mixed, horizontal, and geometric (generalized) LCP.

Next, if we consider a QP in the equality form

$$\begin{aligned} \min \quad & \tfrac{1}{2} x^T Q x + c^T x \\ \text{subject to} \quad & Ax = b, \\ & x \geq 0, \end{aligned} \tag{2.18}$$

where Q is given by an nxn symmetric matrix, A is given by an mxn matrix with $m < n$, and c, b are given vectors of the corresponding dimension, then its Kurush-Kuhn-Tucker system is of the form

$$\begin{aligned} Ax = b, \quad x \geq 0, \\ A^T y + s - Qx = c, \quad s \geq 0, \\ x^T s = 0. \end{aligned} \tag{2.19}$$

The above system can be formulated as SLCP by removing the free variable $y \in \mathbb{R}^m$ in the usual manner $y = y^+ - y^-$ where $y^+, y^- \in \mathbb{R}^m_+$. This unfortunately results in a large increase in problem size which is not welcomed. Also, it is very likely that the algorithm for SLCP applied to this newly reformulated system, (2.19), will not be as efficient because the algorithm employs the special structure of that system.

Alternatively, the system (2.19) leads to the following LCP form which is called *mixed* LCP (MLCP):

$$\begin{aligned} M\begin{pmatrix} x \\ y \end{pmatrix} + q &= \begin{pmatrix} s \\ 0 \end{pmatrix}, \\ x^T s &= 0, \\ (x, s) &\geq 0, \end{aligned} \tag{2.20}$$

where $x, s \in \mathbb{R}^n_+$, $y \in \mathbb{R}^m$ and

$$M = \begin{pmatrix} Q & -A^T \\ A & 0 \end{pmatrix} \quad \text{and} \quad q = \begin{pmatrix} c \\ -b \end{pmatrix}. \tag{2.21}$$

For the system (2.19), the MLCP formulation is simpler than the SLCP formulation. We conclude that the formulation of a problem as LCP may lead to the different forms of LCP and restricting the LCP to only one form may have a high price for different formulations.

Another formulation of the LCP is the horizontal LCP (HLCP):

$$\begin{aligned} Mx + Ns &= q, \\ x^T s &= 0, \\ (x, s) &\geq 0, \end{aligned} \tag{2.22}$$

where $M, N \in \mathbb{R}^{nxn}$ matrices, and $q \in \mathbb{R}^n$.

Instead of counting on an algebraic representation, we can also focus on a geometric approach by observing that

$$\begin{aligned} M_1 &= \left\{(x, s) : Ax = b, A^T y + s - Qx = c \text{ for some } y\right\}, \\ M_2 &= \{(x, s) : Mx + q = s\}, \\ M_3 &= \{(x, s) : Mx + Ns = q\}, \end{aligned}$$

are all instances of a linear manifold M. For any vector $\overline{z}^* \in \mathbb{R}^{2n}$ and any subspace $\Phi \in \mathbb{R}^{2n}$, we define the linear manifold $M = \Phi(\overline{z}^*)$ as

$$M = \Phi(\overline{z}^*) = \overline{z}^* + \Phi = \{z \in \mathbb{R}^{2n} : z - \overline{z}^* \in \Phi\}. \tag{2.23}$$

Now (2.13), (2.19), and (2.22) can all be considered as instances of the geometric (generalized) LCP (GLCP):

$$\begin{array}{rrcl} \text{Find} & z & = & (x, s) \in M, \\ \text{such that} & x^T s & = & 0, \\ & (x, s) & \geq & 0. \end{array} \tag{2.24}$$

We refer to the GLCP as the pair $(\Phi, \overline{z}^*)$.

In general LCP is NP-complete, which means that there exists no polynomial algorithms for solving it. Thus, the problem needs to be restricted to certain classes of matrices for which the polynomial algorithms exist. We now list several such classes of matrices M for SLCP. They are as follows:

- Skew-symmetric matrices (SS):

$$(x \in \mathbb{R}^n)(x^T M x = 0). \tag{2.25}$$

- Positive semi-definite matrices (PSD):

$$(x \in \mathbb{R}^n)(x^T M x \geq 0). \tag{2.26}$$

- P-matrices: Matrices with all principal minors positive or equivalently

$$(0 \neq x \in \mathbb{R}^n)(\exists i \in I)(x_i(Mx)_i > 0). \tag{2.27}$$

- P_0-matrices: Matrices with all principal minors nonnegative or equivalently

$$(0 \neq x \in \mathbb{R}^n)(\exists i \in I)(x_i \neq 0 \text{ and } x_i(Mx)_i \geq 0). \tag{2.28}$$

- Sufficient matrices (SU): Matrices which are column and row sufficient
  - Column sufficient matrices (CSU):

$$(\forall x \in \mathbb{R}^n)(\forall i \in I)(x_i(Mx)_i \leq 0 \Rightarrow x_i(Mx)_i = 0). \tag{2.29}$$

  - Row sufficient matrices (RSU): M is row sufficient if M^T is column sufficient.

- $P_*(\kappa)$: Matrices such that

$$(1 + 4\kappa) \sum_{i \in I^+(x)} x_i(Mx)_i + \sum_{i \in I^-(x)} x_i(Mx)_i \geq 0, \forall x \in \mathbb{R}^n,$$

where

$$I^+(x) = \{i : x_i(Mx)_i > 0\}, I^-(x) = \{i : x_i(Mx)_i < 0\},$$

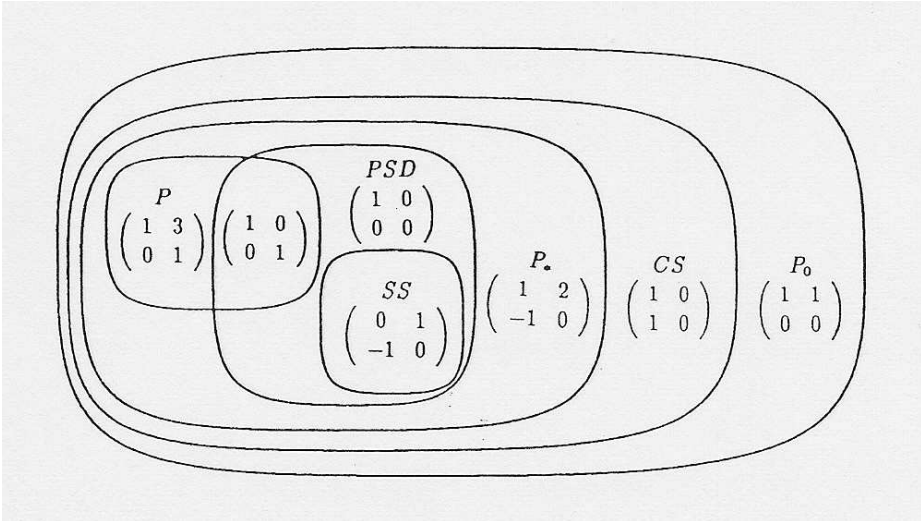

Figure 2.1: Relations and examples of the classes of matrices.

or equivalently

$$x^T Mx \geq -4\kappa \sum_{i \in I^+(x)} x_i (Mx)_i, \forall x \in \mathbb{R}^n, \tag{2.30}$$

and

$$P_* = \bigcup_{\kappa \geq 0} P_*(\kappa). \tag{2.31}$$

Especially interesting, important (and nontrivial) is that the P_* matrices are just sufficient.

The relationship between some of the above classes is as follows:

$$SS \subset PSD \subset P_* = SU \subset CS \subset P_0, \quad P \subset P_*, \quad P \cap SS = \emptyset. \tag{2.32}$$

Some of these relations are obvious, like PSD $= P_*(0) \subset P_*$ or $P \subset P_*$, while others require proof. Refer to Figure 2.1, which was first published in [9], to see a visual flow of how these classes of matrices are related. Also, all of the above classes have the

nice property that if matrix M belongs to one of these classes, then every principal sub-matrix of M also belongs to the class.

In this thesis, we will assume that matrix M is a positive semi-definite (PSD) matrix. This case is not most general, but it is certainly most commonly used both in theory and practice. Hence, this is reason why we will focus on this class of matrices in the thesis. The SLCP with a PSD matrix M is called *monotone* LCP.

CHAPTER 3

KERNEL-BASED INTERIOR-POINT ALGORITHMS

The previous chapter gave us insight on the problem, which is trying to be solved. In this chapter, we will discuss the generic IPM method for solving a monotone LCP. This method will be based on the concept of kernel functions. First, we will explain the concept of the central path, then convey the general outline of the generic interior-point primal-dual method. We will also define, discuss, and expand on kernel functions and their role in design and analysis of IPM for LCP.

3.1 The Central Path

We consider the linear complementarity problem in the standard form, SLCP, which is finding a point $(x, s) \in \mathbb{R}^{2n}$ that satisfies the following conditions

$$\begin{aligned} Mx + q - s = 0, \quad (x, s) \geq 0, \\ xs = 0, \end{aligned} \tag{3.1}$$

where $M \in \mathbb{R}^{nxn}$, $q \in \mathbb{R}^n$ and where xs in the last equation represents the component-wise (Hadamard) product of the vectors x and s.

The general idea is to solve (3.1) using Newton's method. However, Newton's method can "get stuck" at the complementarity equation $xs = 0$. Therefore, the main idea of primal-dual interior-point methods is to replace the last equation in (3.1), the so called *complementarity equation*, with the parameterized equation $xs = \mu e$, with parameter $\mu > 0$. So we consider the following system

$$\begin{aligned} Mx + q - s = 0, \quad (x, s) \geq 0, \\ xs = \mu e, \end{aligned} \tag{3.2}$$

where e is defined as a vector of ones of size n. By the last equation, any solution (x, s) of (3.2) will satisfy $x > 0$ and $s > 0$. Suppose, there exits a point $(x^0, s^0) > 0$ such that

$$Mx^0 + q - s^0 = 0, \tag{3.3}$$

which means that the interior of the feasible region of (3.1) is not empty. This assumption is called the *interior-point condition* (IPC) of the LCP. If IPC is not satisfied the modified LCP can be constructed so that it satisfies the IPC. From the solution of the modified LCP, the solution of the original LCP can easily be found. See chapter five in Kojima et al. [9]. Thus, we can, and in this thesis we will, always assume that the IPC is satisfied.

It can be shown that for certain classes of matrices, if M has a full rank, i.e. $\text{rank}(M) = n$ and IPC holds, then the parameterized system (3.2) has a unique solution, for each $\mu > 0$ (see Lemma 4.3 in [9]). This is particularly true for positive semi-definite matrices that we are considering in this thesis. This solution is denoted as $(x(\mu), s(\mu))$ and we call $(x(\mu), s(\mu))$ the μ-center of (3.1). The set of μ-centers (with μ running through all positive real numbers) gives a homotopy path, which is called *the central path* of (3.1). The importance of the central path for the LP was discovered first by Sonnevend [17] and Megiddo [13] and later generalized to LCP by Kojima et al. [9]. The main property of the central path is that if $\mu \to 0$, then the limit of the central path exists and since the limit points satisfy the complementarity condition, the limit yields the optimal solutions for (3.1).

This limiting property of the central path leads to the main idea of the iterative methods for solving (3.1): Trace the central path while reducing μ at each iteration. Theoretically, an exact trace is wanted, but practically it is too inefficient. However,

it has been shown that it is only necessary to trace the central path approximately in order to maintain favorable convergence properties of the given algorithms.

3.2 Main Idea of the Method

As discussed previously, the IPMs trace the central path approximately. The general outline of the generic interior-point primal-dual method is discussed below. Foremost, without loss of generality, we assume that a point (x, s) is "close" to the μ-center, $(x(\mu), s(\mu))$ for some parameter $\mu > 0$. Then, μ is decreased to $\mu_+ := (1-\theta)\mu$, for some $\theta \in (0, 1)$. Next, we redefine $\mu = \mu_+$, then we solve the following Newton system

$$\begin{aligned} -M\Delta x + \Delta s &= 0, \\ s\Delta x + x\Delta s &= \mu e - xs. \end{aligned} \tag{3.4}$$

Since M has full row rank, the system (3.4) has a unique solution for any $(x, s) > 0$. The solution $(\Delta x, \Delta s)$ is known as the Newton direction. By taking a step along this search direction, we construct a new ordered pair (x_+, s_+) with

$$x_+ = x + \alpha\Delta x, \quad s_+ = s + \alpha\Delta s, \tag{3.5}$$

where α denotes the step size, $\alpha \in (0, 1)$, which must be chosen carefully. If needed, we repeat the procedure until we find iterates that are in a certain neighborhood of the μ-center $(x(\mu), s(\mu))$. Then, again, μ is reduced by the factor $1-\theta$ and Newton's method is applied again targeting the new μ-center, and so on. We repeat this process until μ is small enough, i.e. $n\mu \leq \epsilon$, where ϵ is a small positive number. At this stage in the algorithm, we have found ϵ-approximate solutions of (3.1).

Before formally stating the algorithm, we introduce important scaling that allows generalization and introduction of kernel-based barrier functions. For any triple

(x, s, μ) with $x > 0$, $s > 0$ and $\mu > 0$, we introduce the so called *variance vector*:

$$v := \sqrt{\frac{xs}{\mu}}. \tag{3.6}$$

Note that the pair (x, s) coincides with the μ-center $(x(\mu), s(\mu))$ if and only if $v = e$.

The *scaled search directions* d_x and d_s are then defined as

$$d_x := \frac{v\Delta x}{x}, \quad d_s := \frac{v\Delta s}{s}, \tag{3.7}$$

where each of the operations are component-wise product and division.

Lemma 3.2.1. *If v is defined, as in (3.6) and the search directions d_x, d_s are defined as in (3.7), then the Newton system from (3.4) can be transformed into the following system:*

$$\begin{aligned} -\widetilde{M}d_x + d_s &= 0, \\ d_x + d_s &= v^{-1} - v, \end{aligned} \tag{3.8}$$

where

$$\widetilde{M} := DMD, \quad D := X^{\frac{1}{2}}S^{-\frac{1}{2}}, \quad S := diag(s), \quad and\ X := diag(x).$$

Proof. Recall the Newton system given in (3.4)

$$-M\Delta x + \Delta s = 0, \tag{3.9}$$

$$s\Delta x + x\Delta s = \mu e - xs. \tag{3.10}$$

The scaled search directions d_x,d_s as defined in (3.7), can be rewritten as

$$\Delta x = \frac{xd_x}{v}, \quad \Delta s = \frac{sd_s}{v}, \tag{3.11}$$

where v is defined in (3.6).

By applying (3.11) to (3.10), we obtain

$$\begin{aligned} s\left(\tfrac{xd_x}{v}\right) + x\left(\tfrac{sd_s}{v}\right) &= \mu e - xs \\ \left(\tfrac{sx}{v}\right) d_x + \left(\tfrac{xs}{v}\right) d_s &= \mu e - xs \\ d_x + d_s &= \tfrac{v}{sx}\left(\mu e - xs\right) \\ d_x + d_s &= v^{-1} - v. \end{aligned}$$

We have shown the transformation for (3.10) using (3.11). Next we will focus our attention on transforming (3.9). If we apply (3.11) to (3.9), we see

$$-M\left(\frac{xd_x}{v}\right) + \left(\frac{sd_s}{v}\right) = 0. \tag{3.12}$$

The above equation can be transformed in the following way. First, observe that any vector $a \in \mathbb{R}^n$ can be written as

$$a = [\mathrm{diag}(a)]\, e,$$

where

$$\mathrm{diag}(a) = \begin{bmatrix} a_1 & & & \\ & a_2 & & \\ & & \ddots & \\ & & & a_n \end{bmatrix} \tag{3.13}$$

and e is a vector of all ones.

Therefore, vector $\frac{xd_x}{v}$ can be written as

$$\begin{aligned} \frac{xd_x}{v} &= \left(XV^{-1}D_x\right) e \\ &= XV^{-1}\left(D_x e\right) \\ &= XV^{-1} d_x \end{aligned} \tag{3.14}$$

where

$$X = \mathrm{diag}(x), \quad V^{-1} = \mathrm{diag}(v^{-1}), \quad D_x = \mathrm{diag}(d_x).$$

Similarly, vector $\frac{sd_s}{v}$ can be written as

$$\begin{aligned} \frac{sd_s}{v} &= \left(SV^{-1}D_x\right)e \\ &= SV^{-1}\left(D_s e\right) \\ &= SV^{-1}d_s \end{aligned} \tag{3.15}$$

where

$$S = \text{diag}(s), \quad V^{-1} = \text{diag}(v^{-1}), \quad D_s = \text{diag}(d_s).$$

Substitution of (3.14) and (3.15) into (3.12) leads to

$$\begin{aligned} S^{-1}V\left(-MXV^{-1}d_x + SV^{-1}d_s\right) = 0 \\ -S^{-1}VMXV^{-1}d_x + d_s = 0 \end{aligned} \tag{3.16}$$

The matrix $S^{-1}VMXV^{-1}$ can be simplified by observing that

$$VS^{-1} = \text{diag}\left(\frac{v}{s}\right) = \text{diag}\left(\sqrt{\frac{x}{\mu s}}\right) = \frac{1}{\sqrt{\mu}}X^{\frac{1}{2}}S^{-\frac{1}{2}} = \frac{1}{\sqrt{\mu}}D. \tag{3.17}$$

and

$$XV^{-1} = \text{diag}\left(\frac{x}{v}\right) = \text{diag}\left(\sqrt{\frac{\mu x}{\mu s}}\right) = \sqrt{\mu}X^{\frac{1}{2}}S^{-\frac{1}{2}} = \sqrt{\mu}D. \tag{3.18}$$

where $D := X^{\frac{1}{2}}S^{-\frac{1}{2}}$, $S :=$ diag(s), and X:= diag(x).

Next, by applying (3.17) and (3.18) to (3.16), we get

$$-\left[DMD\right]d_x + d_s = 0.$$

If we denote $\widetilde{M} := DMD$, we obtain

$$-\widetilde{M}d_x + d_s = 0.$$

Hence, the lemma is proved. □

Lemma 3.2.2. *If matrix M is positive semi-definite, then $\widetilde{M}$ is also positive semi-definite.*

Proof. Let $a \in \mathbb{R}^n$ and $\widetilde{M}$ be as defined above, then

$$\begin{aligned} a^T \widetilde{M} a &= a^T (DMD) a \\ &= \left(a^T D\right) M (Da) \\ &= (Da)^T M (Da) \\ &\geq 0. \end{aligned}$$

By assumption, we know M is positive semi-definite. Hence, by definition, $\widetilde{M}$ is positive semi-definite. □

Note that:

$$d_x = d_s = 0 \quad \Leftrightarrow \quad v^{-1} - v = 0 \quad \Leftrightarrow \quad v = e.$$

Therefore, we see that $d_x = d_s = 0$ if and only if the pair (x, s) coincides with the μ-center $(x(\mu), s(\mu))$. Unfortunately, d_x and d_s are not, in general, orthogonal vectors, as in the LP case, which will complicate the analysis of the algorithm.

A very important observation is that the right hand side $v^{-1} - v$ in the last equation of (3.8) equals the negative gradient of the function

$$\Psi_c(v) := \sum_{i=1}^{n} \left(\frac{v_i^2 - 1}{2} - \log v_i \right), \tag{3.19}$$

which can be written as,

$$d_x + d_s = -\nabla \Psi_c(v). \tag{3.20}$$

This equation is known as the *scaled centering equation.* The scaled centering equation basically defines the search directions. An easy verification is that $\nabla^2 \Psi_c(v) = \text{diag}(e + v^{-2})$. Since this matrix is positive definite, $\Psi_c(v)$ is strictly convex. We can see that $\nabla \Psi_c(e) = 0$, hence $\Psi_c(v)$ attains its minimal value at $v = e$, with $\Psi_c(e) = 0$. So, it follows that $\Psi_c(v)$ is non-negative everywhere and vanishes at $v = e$, which means it

vanishes at the μ-center $(x(\mu), s(\mu))$. Therefore, we can conclude that the μ-center $(x(\mu)s(\mu))$ can be characterized as the minimizer of the function $\Psi_c(v)$. Thus, $\Psi_c(v)$ serves as a measure of how close (x, s) is to the μ-center.

Another purpose of $\Psi_c(v)$ is visible from equation (3.20) which is $\Psi_c(v)$ essentially determines the search direction (d_x, d_s). To summarize, $\Psi_c(v)$ has two crucial properties:

1. $\Psi_c(v)$ determines the search direction.
2. $\Psi_c(v)$ serves as a measure of closeness to the μ-center.

Hence, it basically controls the whole algorithm.

From the previous paragraph, the following important generalization follows: we can replace $\Psi_c(v)$ by any strictly convex function $\Psi(v), v \in \mathbb{R}^n_{++}$, such that $\Psi(v)$ is minimal at $v = e$ and $\Psi(e) = 0$. This new function $\Psi(v)$ is called a *scaled barrier function*. Hence, the new scaled centering equation is reformulated as

$$d_x + d_s = -\nabla\Psi(v). \tag{3.21}$$

We still have

$$d_x = d_s = 0 \Leftrightarrow v = e \Leftrightarrow x = x(\mu) \text{ and } s = s(\mu).$$

Note that alternate barrier functions lead to alternate Newton directions. Likewise, the measure of closeness to the μ-center will also be different than for $\Psi_c(v)$. Thus, the scaled Newton system is

$$\begin{aligned} -\widetilde{M}d_x + d_s &= 0, \\ d_x + d_s &= -\nabla\Psi(v). \end{aligned} \tag{3.22}$$

After finding d_x and d_s from (3.22), we can find the original directions Δx and Δs from (3.7). Alternatively, Δx and Δs can be found directly from the following system

$$\begin{aligned} -M\Delta x + \Delta s &= 0, \\ s\Delta x + x\Delta s &= -\mu v \nabla \Psi(v). \end{aligned} \tag{3.23}$$

3.3 Generic Interior-Point Primal-Dual Algorithm

We can now formally describe the generic primal-dual algorithm. As we mentioned, this algorithm follows the central path approximately. Suppose we start with (x, s) close to μ-center, then μ is reduced to $\mu^+ = (1-\theta)\mu$. Therefore, new v becomes $v^+ = \frac{v}{\sqrt{1-\theta}}$. As a consequence, $\Psi(v)$ changes to $\Psi(v^+)$. The inequality, $\Psi(v) \leq \tau$, means that (x, s) is in a τ-neighborhood of the μ-center $(x(\mu), s(\mu))$, where $\tau > 0$ represents a certain threshold value. Recall that, we measure the closeness of (x, s) to μ-center $(x(\mu), s(\mu))$ by the value of $\Psi(v)$. However, after the θ-update, the updated $\Psi(v^+)$ may be greater than τ, if so, we need to perform further steps to reduce $\Psi(v^+)$ to get closer to the new μ-center, i.e, to get back to the τ-neighborhood of a new μ-center.

To accomplish this, we need to first find the direction Δx and Δs by solving the Newton system (3.23). We update x and s using a chosen step size α and the recently found search directions Δx and Δs, respectively. This process is repeated until $\Psi(v) \leq \tau$, upon which the process begins again. We begin again by reducing μ and updating v, and so on until we have a μ-center that is ϵ-close to the actual solution. The generic form of the algorithm is shown in Table 3.1. In the sequel, we will refer to it as simply the Generic Algorithm.

Generic Primal-Dual Algorithm for LCP

Input:

Determine input parameters:

threshold parameter $\tau > 0$,

fixed barrier update parameter θ, $0 < \theta < 1$,

accuracy parameter $\epsilon > 0$.

begin

Set $(x_0, s_0, \mu_0) > 0$ so that the IPC is satisfied;

while $n\mu \geq \epsilon$ **do**

$\mu := (1 - \theta)\mu$;

$v := \sqrt{\frac{xs}{\mu}}$;

while $\Psi(v) > \tau$ **do**

Calculate direction $(\Delta x, \Delta s)$ by solving (3.23);

Calculate step size α;

Update $x := x + \alpha\Delta x$ and $s := s + \alpha\Delta s$;

Update $v := \sqrt{\frac{xs}{\mu}}$;

end do

end do

end

Table 3.1: Generic Primal-Dual Algorithm for LCP

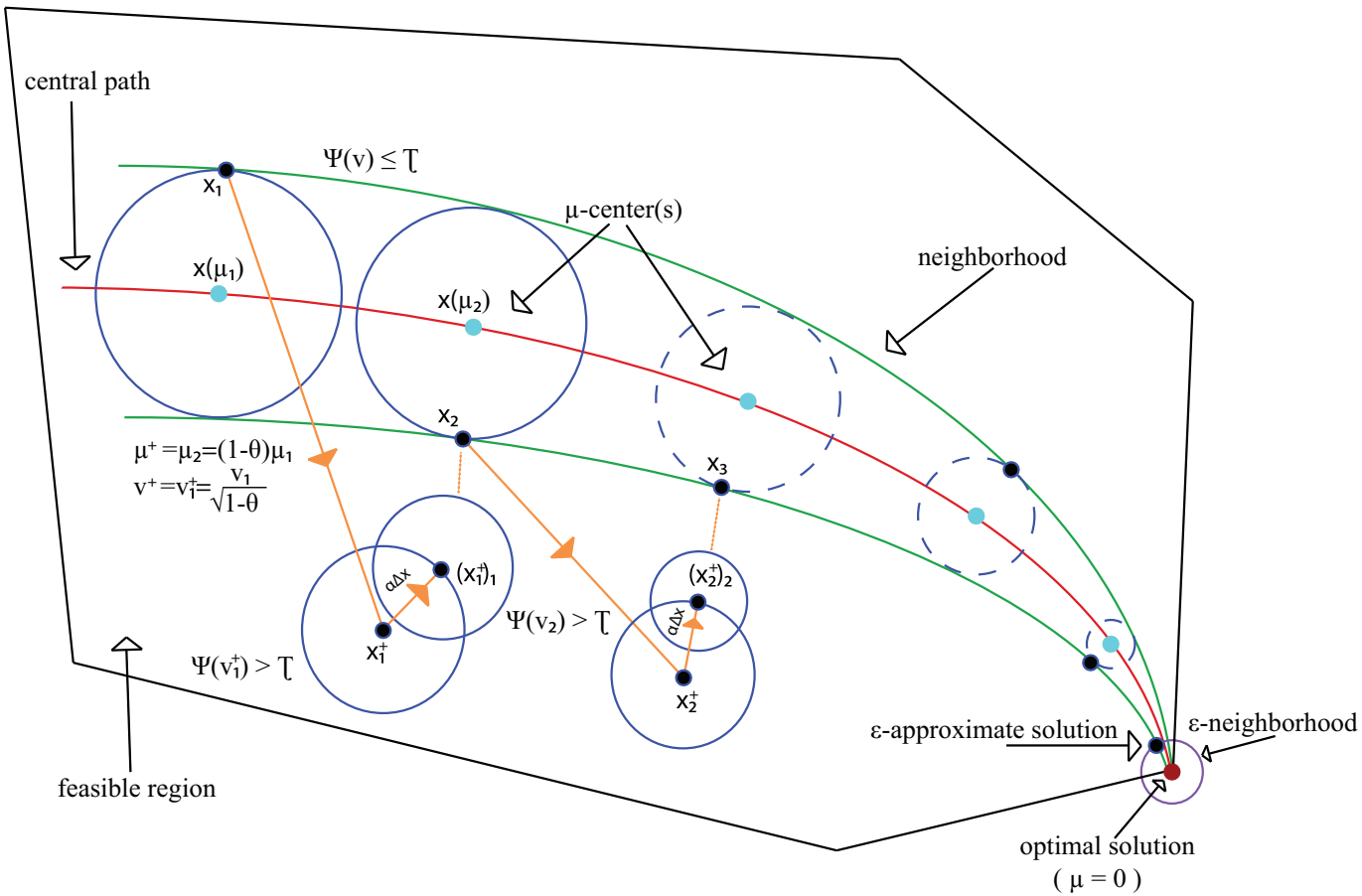

Figure 3.1: Representation of the Generic Primal-Dual Algorithm for LCP.

We want to "optimize" the algorithm by minimizing the number of iterates in the algorithm. To do this we must carefully choose the parameters τ, θ, and the step size α. Choosing the *barrier update parameter* θ is very important in application and theory. If θ is a constant number which is independent of the dimension n of the problem, i.e. $\theta = O(1)$, then the algorithm is called a *large update* method. If θ depends on the dimension n of the problem, then we call the algorithm a *small update* method. In this case, θ is usually chosen to be the following: $\theta = O\left(\frac{1}{\sqrt{n}}\right)$.

Choosing the step size, $\alpha > 0$, is another key step in obtaining good convergence properties of the algorithm. It must be set in such a way that the closeness of the iterates to the current μ-center improves by a sufficient amount. This will be discussed later in the text.

3.4 Eligible Kernel Functions

For the sake of this thesis, we will consider a *barrier function* $\Psi(v)$ that is a separable function with identical coordinate functions $\psi(v_i)$. Thus,

$$\Psi(v) = \sum_{i=1}^{n} \psi(v_i), \tag{3.24}$$

where $\psi : (0, \infty) \to [0, \infty)$, $\psi(v)$ is twice differentiable, and it attains its minimum at $t = 1$ with $\psi(1) = 0$. We call this univariate function $\psi(t)$ the *kernel function* of the barrier function $\Psi(v)$.

If we consider the following particular case

$$\psi_c(t) := \frac{t^2 - 1}{2} - \log t, \tag{3.25}$$

then $\Psi(v) = \Psi_c(v)$, as obtained in (3.19). In this case, the search direction becomes the classical Newton direction for primal-dual methods. Notice that the term $-\log t$

commands the behavior of this kernel function if $t \to 0$ and the term $\frac{t^2-1}{2}$ commands the behavior of the kernel function if $t \to \infty$. We call the first term the *barrier term* and the second term the *growth term* of the kernel function. We call the kernel function defined in (3.25) a *logarithmic kernel function*.

For our purposes, we require that the general kernel function ψ be twice differentiable and go to infinity if either $t \to 0$ or $t \to \infty$. Hence ψ satisfies the following

$$\psi'(1) = \psi(1) = 0, \tag{3.26}$$

$$\psi''(t) > 0, \tag{3.27}$$

$$\lim_{t\to 0} \psi(t) = \lim_{t\to\infty} \psi(t) = \infty. \tag{3.28}$$

We can easily see that (3.26) and (3.27) indicate that $\psi(t)$ is a non-negative strictly convex function such that $\psi(t)$ achieves its minimum at $t = 1$, i.e, $\psi(1) = 0$. This tells us that since $\psi(t)$ is twice differentiable, it is completely determined by its second derivative:

$$\psi(t) = \int_1^t \int_1^\xi \psi''(\zeta)\ \mathrm{d}\zeta\ \mathrm{d}\xi. \tag{3.29}$$

Moreover, (3.28) tells us that $\psi(t)$ is coercive and has the barrier property.

As mentioned in the previous section, $\Psi(v)$ is not only used to define a search direction, but also as a measure of closeness of the current iterates to the μ-center. In the analysis of the algorithm, we also use the norm-based proximity measure $\delta(v)$ defined by

$$\delta(v) := \frac{1}{2} \|\nabla\Psi(v)\| = \frac{1}{2} \|d_x + d_s\| . \tag{3.30}$$

Both measures are determined by the kernel function.

To prove later complexity results we impose additional conditions on the kernel

functions:

$$t\psi''(t) + \psi'(t) > 0, \quad t < 1, \tag{3.31}$$

$$\psi'''(t) < 0, \quad t > 0, \tag{3.32}$$

$$2\psi''(t)^2 - \psi'(t)\psi'''(t) > 0, \quad t < 1, \tag{3.33}$$

$$\psi''(t)\psi'(\beta t) - \beta\psi'(t)\psi''(\beta t) > 0, \quad t > 1, \quad \beta > 1. \tag{3.34}$$

Furthermore, we have an additional condition

$$t\psi''(t) - \psi'(t) > 0, \quad t > 1. \tag{3.35}$$

We list this extra condition because conditions (3.32) and (3.35) imply condition (3.34). Condition (3.35) is introduced since it is easier to verify (3.35) than (3.34), which is very technical. Moreover, many eligible kernel functions satisfy (3.35). Thus, the kernel function is also eligible if it satisfies conditions (3.31)-(3.33) and (3.35).

In the following lemma, we can see that condition (3.34) is not independent from the other conditions.

Lemma 3.4.1 (**Lemma 2.2.4 in [5]**). *If $\psi(t)$ satisfies (3.35) and (3.32), then $\psi(t)$ satisfies (3.34).*

Proof. For $t > 1$, we consider

$$f(\beta) := \psi''(t)\psi'(\beta t) - \beta\psi'(t)\psi''(\beta t), \quad \beta \geq 1,$$

Note that $f(1) = 0$. Moreover,

$$\begin{aligned} f'(\beta) &= t\psi''(t)\psi''(\beta t) - \psi'(t)\psi''(\beta t) - \beta t\psi'(t)\psi'''(\beta t) \\ &= \psi''(\beta t)(t\psi''(t) - \psi'(t)) - \beta t\psi'(t)\psi'''(\beta t) > 0. \end{aligned}$$

$\psi(t)$	(3.31)	(3.35)	(3.32)	(3.33)
$t^2 - t - 1 + e^{1-t}$	-	+	+	+
$(t+2)(t-1) - 3\log t$	+	-	+	+
$t^3 + t^{-3} - 2$	+	+	-	+
$8t^2 - 11t + 1 + \frac{2}{\sqrt{t}} - 4\log t$	+	+	+	-

Table 3.2: Independent Kernel Condtions

The last inequality follows since $\psi''(\beta t) > 0$, $t\psi''(t) - \psi'(t) > 0$, by (3.35) and $-\beta t\psi'(t)\psi'''(\beta t) > 0$, since $t > 1$, which implies $\psi'(t) > 0$, and $\psi'''(\beta t) < 0$, by (3.32). Thus it follows that $f(\beta) > 0$ for $\beta > 1$. $\square$

Lemma 3.4.1 tells us that any kernel function satisfying conditions (3.31) - (3.33) and condition (3.35), is an eligible kernel function. For the remainder of this thesis, we will only consider eligible kernel functions.

Remark 3.4.2. *Note that conditions (3.31), (3.35), (3.32), (3.33) are all logically independent, however condition (3.34) is not independent of the other conditions. In Table 3.2, that is taken from [1], we can see four kernel functions and the signs indicate whether the particular condition is satisfied or not.*

Condition (3.31) is satisfied if $t \geq 1$, since then $\psi'(t) \geq 0$ and condition (3.35) is satisfied if $t \leq 1$, since then $\psi'(t) \leq 0$. Furthermore, an important consequence of condition (3.32) is that $\psi''(t)$ is decreasing for $t > 0$. Note that conditions (3.31) and (3.32) are conditions on the barrier behavior of $\psi(t)$. Also, condition (3.35) only deals with $t \geq 1$ and thus deals with only the growth behavior of $\psi(t)$.

The following lemma lists some of the equivalent representations of condition (3.31).

Lemma 3.4.3 (**Lemma 2.2.2 in [5]**). *Let $\psi(t)$ be a twice differentiable function for $t > 0$. Then the following three properties are equivalent:*

$$\begin{aligned} &(i) \quad \psi\left(\sqrt{t_1 t_2}\right) \leq \tfrac{1}{2}\left(\psi(t_1) + \psi(t_2)\right), \ \textit{for } t_1, t_2 > 0; \\ &(ii) \quad \psi'(t) + t\psi''(t) \geq 0, \ t > 0; \\ &(iii) \quad \psi(e^{\xi}) \ \textit{is convex.} \end{aligned}$$

Proof. (iii) $\Leftrightarrow$ (i): From the definition of convexity, we know that $\psi(\exp(\xi))$ is convex if and only if for any $\xi_1, \xi_2 \in \mathbb{R}$, the following inequality holds

$$\psi\left(\exp\left(\frac{1}{2}(\xi_1 + \xi_2)\right)\right) \leq \frac{1}{2}(\psi(\exp(\xi_1)) + \psi(\exp(\xi_2))).$$

Letting $t_1 = \exp(\xi_1)$, $t_2 = \exp(\xi_2)$, obviously one has $t_1, t_2 \in (0, +\infty)$, and the above relation can be rewritten as

$$\psi(\sqrt{t_1 t_2}) \leq \frac{1}{2}(\psi(t_1) + \psi(t_2)).$$

(iii) $\Leftrightarrow$ (ii): The function $\psi(\exp(\xi))$ is convex if and only if the second derivative with respect to ξ is non-negative. This gives $\exp(2\xi)\psi''(\exp(\xi)) + \exp(\xi)\psi'(\exp(\xi)) \geq 0$. Substituting $t = \exp(\xi)$, one gets $t\psi'(t) + t^2\psi''(t) \geq 0$ which is equivalent to $\psi'(t) + t\psi''(t) \geq 0$. □

The property described in Lemma 3.4.3 is called *exponential convexity*, or it is also known as *e-convexity*. This property is essential in the analysis of primal-dual IPMs based on kernel functions.

Below is a technical result that is needed for the sequel.

Lemma 3.4.4 (**Lemma 2.2.5 in [5]**). *One has*

$$t\psi'(t) \geq \psi(t), \ if \ t \geq 1.$$

Proof. Defining $g(t) := t\psi'(t) - \psi(t)$ one has $g(1) = 0$ and $g'(t) = t\psi''(t) \geq 0$. Hence $g(t) \geq 0$ for $t \geq 1$ and the lemma follows. □

The following two lemmas deal with the results of condition (3.32).

Lemma 3.4.5 (**Lemma 2.5 in [1]**). *If the kernel function $\psi(t)$ satisfies (3.32), then*

$$\begin{aligned} &\psi(t) > \tfrac{1}{2}\,(t-1)\,\psi'(t) \quad \textit{and} \quad \psi'(t) > (t-1)\,\psi''(t), \quad \textit{if } t > 1, \\ &\psi(t) < \tfrac{1}{2}\,(t-1)\,\psi'(t) \quad \textit{and} \quad \psi'(t) < (t-1)\,\psi''(t), \quad \textit{if } t < 1. \end{aligned} \tag{3.36}$$

Proof. Consider the function $f(t) = 2\psi(t) - (t-1)\psi'(t)$. One has $f(1) = 0$ and $f'(t) = \psi'(t) - (t-1)\psi''(t)$. Hence $f'(1) = 0$ and $f''(t) = -(t-1)\psi'''(t)$. Using that $\psi'''(t) < 0$ it follows that if $t > 1$ then $f''(t) > 0$, whence, $f'(t) > 0$ and $f(t) > 0$ and if $t < 1$ then $f''(t) < 0$, whence $f'(t) > 0$ and $f(t) < 0$. From this the lemma follows. □

Lemma 3.4.6 (**Lemma 2.6 in [1]**). *If the kernel function $\psi(t)$ satisfies (3.32), then*

$$\begin{aligned} &\tfrac{1}{2}\psi''(t)\,(t-1)^2 < \psi(t) < \tfrac{1}{2}\psi''(1)\,(t-1)^2 \quad , \textit{if } t > 1, \\ &\tfrac{1}{2}\psi''(1)\,(t-1)^2 < \psi(t) < \tfrac{1}{2}\psi''(t)\,(t-1)^2 \quad , \textit{if } t < 1. \end{aligned} \tag{3.37}$$

Proof. By using Taylor's theorem and $\psi(1) = \psi'(1) = 0$, we get

$$\psi(t) = \frac{1}{2}\psi''(1)\,(t-1)^2 + \frac{1}{3!}\psi'''(\xi)\,(\xi-1)^3\,, \tag{3.38}$$

where $1 < \xi < t$ if $t > 1$ and $1 > \xi > t$ if $t < 1$. Since $\psi'''(\xi) < 0$ the second inequality for $t > 1$ and the first inequality for $t < 1$ in the lemma follow. The remaining two inequalities are an immediate consequence of Lemma 3.4.5. □

Furthermore, there are two inverse functions related to the kernel function that are essential to the analysis of the algorithm.

i	Kernel functions $\psi_i(t)$
1	$\frac{t^2-1}{2} - \log t$
2	$\frac{t^2-1}{2} + \frac{t^{1-q}-1}{q(q-1)} - \frac{q-1}{q}(t-1), \quad q > 1$
3	$\frac{t^2-1}{2} + \frac{(e-1)^2}{e}\frac{1}{e^t-1} - \frac{e-1}{e}$
4	$\frac{1}{2}(t - \frac{1}{t})^2$
5	$\frac{t^2-1}{2} + e^{\frac{1}{t}-1} - 1$
6	$\frac{t^2-1}{2} - \int_1^t e^{\frac{1}{\xi}-1} d\xi$
7	$\frac{t^2-1}{2} + \frac{t^{1-q}-1}{q-1}, \quad q > 1$
8	$t - 1 + \frac{t^{1-q}-1}{q-1}, \quad q > 1$
9	$\frac{t^{1+p}-1}{1+p} - \log t, \quad p \in [0,1]$
10	$\frac{t^{1+p}-1}{1+p} + \frac{t^{1-q}-1}{q-1}, \quad p \in [0,1], \quad q > 1$

Table 3.3: Ten Kernel Functions

Definition 3.4.7. *Given the kernel function ψ, we define the following functions:*

$$\begin{aligned} &(i) \quad \gamma : [0,\infty) \to [1,\infty) \text{ is the inverse function of } \psi(t) \text{ for } t \geq 1; \\ &(ii) \quad \rho : [0,\infty) \to (0,1] \text{ is the inverse function of } -\tfrac{1}{2}\psi'(t) \text{ for } t \leq 1. \end{aligned} \tag{3.39}$$

Later, we will use the fact that γ is an increasing function, since $\psi(t)$ is increasing for $t \geq 1$. Similarly, ρ is a decreasing function because $\psi'(t)$ is increasing, while $-\frac{1}{2}\psi'(t)$ is decreasing.

In Table 3.3, we list ten eligible kernel functions that are generally used for study.

CHAPTER 4

ANALYSIS OF THE ALGORITHM

In this chapter, we will discuss, in depth, the analysis of the Generic Algorithm for solving the LCP that is described in Table 3.1. We will see how to obtain a bound for the growth of the barrier function, during an outer iteration. We will also see how to determine the step size during the inner iterations. Finally, we look at how the barrier function is reduced as a result of the step size, in inner iterations.

4.1 Outer Iteration: Growth Behavior of Barrier Function

At the beginning of each outer iteration of the algorithm, we have $\Psi(v) = \tau$, before the update of the parameter μ by the factor of $(1-\theta)$. The μ-update causes the vector v to be divided by the factor $\sqrt{1-\theta}$, with $0 < \theta < 1$, which leads to an increase in the value of $\Psi(v)$. To counter this increase, during subsequent inner iterations $\Psi(v)$ decreases until it surpasses the threshold value τ again. From this, we can note that during the algorithm, the largest values of $\Psi(v)$ occur immediately after the μ-updates. Hence, there exists a need for an estimate for the effect of a μ-update on the value of $\Psi(v)$. In this section, we will do exactly that, derive such an estimate.

In other words, by defining β as

$$\beta := \frac{1}{\sqrt{1-\theta}},$$

we want to find an upper bound for $\Psi(\beta v)$ in terms of $\Psi(v)$. We begin with the following lemma to help us.

Lemma 4.1.1 (Lemma 2.2.8 in [5]). *Suppose that $\psi(t_1) = \psi(t_2)$, with $t_1 \leq 1 \leq t_2$*

and $\beta \geq 1$. *Then*

$$\psi(\beta t_1) \leq \psi(\beta t_2).$$

Equality holds if and only if $\beta = 1$ *or* $t_1 = t_2 = 1$.

Proof. Consider

$$f(\beta) := \psi(\beta t_2) - \psi(\beta t_1).$$

One has $f(1) = 0$ and

$$f'(\beta) = t_2\psi'(\beta t_2) - t_1\psi(\beta t_1).$$

Since $\psi''(t) \geq 0$ for all $t > 0$, $\psi'(t)$ is monotonically non-decreasing. Hence $\psi'(\beta t_2) \geq \psi'(\beta t_1)$. Substitution gives

$$f'(\beta) = t_2\psi'(\beta t_2) - t_1\psi'(\beta t_1) \geq t_2\psi'(\beta t_2) - t_1\psi'(\beta t_2) = \psi'(\beta t_2)(t_2 - t_1) \geq 0.$$

The last inequality holds since $t_2 \geq t_1$, and $\psi'(t) \geq 0$ for $t \geq 1$. This proves that $f(\beta) \geq 0$ for $\beta \geq 1$, and hence the inequality in the lemma follows. If $\beta = 1$ then we obviously have equality. Otherwise, if $\beta > 1$, and $f(\beta) = 0$, then the mean value theorem implies $f'(\xi) = 0$ for some $\xi \in (1, \beta)$. But this implies $\psi'(\xi t_2) = \psi(\xi t_1)$. Since $\psi'(t)$ is strictly monotonic, this implies $\xi t_2 = \xi t_1$, whence $t_2 = t_1$. Since also $t_1 \leq 1 \leq t_2$, we obtain $t_2 = t_1 = 1$. □

The following theorem gives us an upper bound for $\Psi(v)$ after the μ-update in terms of the inverse function of $\psi(t)$ for $t \geq 1$.

Theorem 4.1.2 (Theorem 3.2 in [1]). *Let* $\gamma : [0, \infty) \to [1, \infty)$ *be the inverse function of* $\psi(t)$ *for* $t \geq 1$. *Then we have for any positive vector* v *and any* $\beta \geq 1$*:*

$$\Psi(\beta v) \leq n\psi\left(\beta\gamma\left(\frac{\Psi(v)}{n}\right)\right).$$

Proof. First we consider the case where $\beta > 1$. We consider the following maximization problem:

$$\max_v \{\Psi(\beta v) : \Psi(v) = z\},$$

where z is any non-negative number. The first order optimality conditions for this problem are

$$\beta\psi'(\beta v_i) = \lambda\psi'(v_i), \quad i = 1, \ldots, n, \tag{4.1}$$

where λ denotes the Lagrange multiplier. Since $\psi'(1) = 0$ and $\beta\psi'(\beta) > 0$, we must have $v_i \neq 1$ for all i. We even may assume that $v_i > 1$ for all i. To see this, let z_i be such that $\psi(v_i) = z_i$. Given z_i, this equation has two solutions: $v_i = v_i^{(1)} < 1$ and $v_i = v_i^{(2)} > 1$. As a consequence of Lemma (4.1.1), we have $\psi(\beta v_i^{(1)}) \leq \psi(\beta v_i^{(2)})$. Since we are maximizing $\Psi(\beta v)$, it follows that we may assume $v_i = v_i^{(2)} > 1$. Thus we have shown that without loss of generality we may assume that $v_i > 1$ for all i. Note that then (4.1) implies $\beta\psi'(\beta v_i) > 0$ and $\psi'(v_i) > 0$, whence also $\lambda > 0$. Now defining $g(t)$ according to

$$g(t) := \tfrac{\psi'(t)}{\psi'(\beta t)}, \quad t \geq 1,$$

we deduce from (4.1) that $g(v_i) = \frac{\beta}{\lambda}$ for all i. One has

$$g'(t) = \frac{\psi''(t)\psi'(\beta t) - \beta\psi'(t)\psi''(\beta t)}{(\psi'(\beta t))^2}.$$

At this stage we use that $\psi(t)$ satisfies condition (3.34). Due to this we have $g'(t) > 0$, for $t > 1$ and $\beta > 1$. So $g(t)$ is strict monotonically increasing. Hence it follows that all v_i's are mutually equal. Putting $v_i = t > 1$, for all i, we deduce from $\Psi(v) = z$ that $n\psi(t) = z$. This implies $t = \gamma(\frac{z}{n})$. Hence the maximal value that $\Psi(v)$ can attain is given by

$$\Psi(\beta te) = n\psi(\beta t) = n\psi\left(\beta\gamma\left(\frac{z}{n}\right)\right) = n\psi\left(\beta\gamma\left(\frac{\Psi(v)}{n}\right)\right).$$

This proves the theorem if $\beta > 1$. For the case $\beta = 1$ it suffices to observe that both sides of the inequality in the theorem are continuous in β. $\square$

Corollary 4.1.3 (**Corollary 3.3 in [1]**). *Using the notation of Theorem 4.1.2, we have*

$$\Psi(\beta v) \leq \frac{n}{2}\psi''(1)\left(\beta\gamma\left(\frac{\Psi(v)}{n}\right) - 1\right)^2. \tag{4.2}$$

Proof. Since $\beta \geq 1$ and $\gamma(\frac{\Psi(v)}{n}) \geq 1$, the corollary follows from Theorem 4.1.2 by using also Lemma 3.4.6. $\square$

Hence, as a result of Theorem 4.1.2, we have that if $\Psi(v) \leq \tau$ and $\beta = \frac{1}{\sqrt{1-\theta}}$ then

$$L_\psi(n, \theta, \tau) := n\psi\left(\frac{\gamma(\frac{\tau}{n})}{\sqrt{1-\theta}}\right) \tag{4.3}$$

is an upper bound for $\Psi(\frac{v}{\sqrt{1-\theta}})$, the value of $\Psi(v)$ after the μ-update.

4.2 Inner Iteration: Determining the Step Size

In this section, we determine a step size α which gives rise to a large decrease of $\Psi(v)$. At the same time, we want the step size to keep the iterations feasible, during each inner iteration. The analysis below follows the same line of arguments that were used in [1].

In each inner iteration, we first find the search direction $(\Delta x, \Delta s)$ from the system (3.23). Suppose a step size α is given, then the new iterate, (x_+, s_+), is calculated by

$$x_+ = x + \alpha\Delta x, \quad s_+ = s + \alpha\Delta s.$$

Recall that during an inner iteration the parameter μ is fixed. Hence, after the step in the direction $(\Delta x, \Delta s)$ with the step size α the new v vector is given by,

$$v_+ = \sqrt{\frac{x_+ s_+}{\mu}}. \tag{4.4}$$

By using some previous definitions, such as

$$v = \sqrt{\tfrac{xs}{\mu}}, \quad d_x = \tfrac{v\Delta x}{x}, \quad d_s = \tfrac{v\Delta s}{s},$$

we have

$$x_+ = x\left(e + \alpha\frac{\Delta x}{x}\right) = x\left(e + \alpha\frac{d_x}{v}\right) = \frac{x}{v}\left(v + \alpha d_x\right),$$

and

$$s_+ = s\left(e + \alpha\frac{\Delta s}{s}\right) = s\left(e + \alpha\frac{d_s}{v}\right) = \frac{s}{v}\left(v + \alpha d_s\right),$$

so we obtain that

$$v_+ = \sqrt{\left(v + \alpha d_x\right)\left(v + \alpha d_s\right)}. \tag{4.5}$$

Next, we consider the decrease of Ψ as a function of α. We define a function:

$$f(\alpha) := \Psi(v_+) - \Psi(v). \tag{4.6}$$

The exponential convexity (e-convexity) property, which is the first property (i) in Lemma 3.4.3 implies that

$$\Psi(v_+) = \Psi\left(\sqrt{\left(v + \alpha d_x\right)\left(v + \alpha d_s\right)}\right) \leq \frac{1}{2}\left(\Psi\left(v + \alpha d_x\right) + \Psi\left(v + \alpha d_s\right)\right) \tag{4.7}$$

If we define

$$f_1(\alpha) := \frac{1}{2}\left(\Psi\left(v + \alpha d_x\right) + \Psi\left(v + \alpha d_s\right)\right) - \Psi(v). \tag{4.8}$$

then we have $f(\alpha) \leq f_1(\alpha)$. We can see from the above inequality that $f_1(\alpha)$ is an upper bound of $f(\alpha)$. The reason for considering $f_1(\alpha)$ instead of $f(\alpha)$ is that $f_1(\alpha)$ is a convex function in α while $f(\alpha)$ may not be convex. Hence, it is easier to deal with the function $f_1(\alpha)$.

In order to show that $f_1(\alpha)$ is convex, we need to examine its second derivative. The first derivative of f_1 with respect to α is

$$f_1'(\alpha) = \frac{1}{2}\sum_{i=1}^{n}\left(\psi'\left(v_i + \alpha d_{xi}\right) d_{xi} + \psi'\left(v_i + \alpha d_{si}\right) d_{si}\right)$$

Using (3.21) and (3.30), we get

$$f_1'(0) = \frac{1}{2}\nabla\Psi(v)^T(d_x + d_s) = -\frac{1}{2}\nabla\Psi(v)^T\nabla\Psi(v) = -2\delta(v)^2. \tag{4.9}$$

If we differentiate once more, we obtain

$$f_1''(\alpha) = \frac{1}{2}\sum_{i=1}^{n}\left(\psi''\left(v_i + \alpha d_{xi}\right) d_{xi}^2 + \psi''\left(v_i + \alpha d_{si}\right) d_{si}^2\right) > 0, \tag{4.10}$$

except in the case when $d_x = d_s = 0$. We may also note that during an inner iteration x and s are not both at the μ-center since $\Psi(v) \geq \tau > 0$. Hence, we may conclude that $f_1(\alpha)$ is strictly convex in α. We can also note that

$$f(0) = f_1(0) = 0.$$

In the following, we state several lemmas that will help obtain a suitable lower bound on the step size α. Also, we will simplify some notation:

$$v_{min} := \min(v), \quad \delta := \delta(v),$$

where $\delta(v)$ is defined in (3.30).

The key step in the analysis are based on the effort to find an upper bound on $\|d_x\|$ and $\|d_s\|$ in terms of the proximity measure δ. The following lemma gives us such a bound.

Lemma 4.2.1. *If $\widetilde{M}$ is positive semi-definite, then for search directions d_x, d_s obtained in (3.7) the following upper bounds hold:*

$$\|d_x\| \leq 2\delta \quad \textit{and} \quad \|d_s\| \leq 2\delta. \tag{4.11}$$

Proof. Consider the first equation in (3.8)

$$-\widetilde{M}d_x + d_s = 0.$$

We can rewrite the above equation as

$$\begin{aligned} d_s &= \widetilde{M}d_x \\ d_x^T d_s &= d_x^T\left(\widetilde{M}d_x\right) \\ d_x^T d_s &= d_x^T\widetilde{M}d_x \end{aligned} \tag{4.12}$$

By Lemma 3.2.2, we know $\widetilde{M}$ is postive semi-definite. Hence, $d_x^T d_s \geq 0$.

Next, we observe the following

$$\begin{aligned} \|d_x + d_s\|^2 &= (d_x + d_s)^T (d_x + d_s) \\ &= d_x^T d_x + 2d_x^T d_s + d_s^T d_s \\ &= \|d_x\|^2 + 2d_x^T d_s + \|d_s\|^2 \\ &\geq \|d_x\|^2 + \|d_s\|^2 . \end{aligned} \tag{4.13}$$

From (3.30), we see

$$\|d_x + d_s\| = \|\nabla\Psi(v)\| = 2\delta(v). \tag{4.14}$$

By combining, (4.13) and (4.14), we obtain

$$4\delta^2 = \|d_x + d_s\|^2 \geq \|d_x\|^2 + \|d_s\|^2 \geq \|d_x\|^2$$

So we clearly see,

$$\|d_x\|^2 \leq 4\delta^2 \Rightarrow \|d_x\| \leq 2\delta.$$

Similarly,

$$4\delta^2 = \|d_x + d_s\|^2 \geq \|d_x\|^2 + \|d_s\|^2 \geq \|d_s\|^2$$

So we can also see,

$$\|d_s\|^2 \leq 4\delta^2 \Rightarrow \|d_s\| \leq 2\delta.$$

The lemma has been proved. □

Lemma 4.2.2 (**Lemma 4.1 in [1]**). *One has:*

$$f_1''(\alpha) \leq 2\delta^2 \psi''(v_{min} - 2\alpha\delta). \tag{4.15}$$

Proof. We can see from Lemma 4.2.1 that $\|d_x\| \leq 2\delta$ and $\|d_s\| \leq 2\delta$. Therefore,

$$v_i + \alpha d_{xi} \geq v_{min} - 2\alpha\delta, \quad v_i + \alpha d_{si} \geq v_{min} - 2\alpha\delta, \quad 1 \leq i \leq n.$$

Due to (3.32), $\psi''(t)$ is monotonically decreasing. Therefore from (4.10) and (4.14) we obtain

$$f_1''(\alpha) \leq \frac{1}{2}\psi''(v_{min} - 2\alpha\delta) \sum_{i=1}^{n} \left(d_{xi}^2 + d_{si}^2\right) = 2\delta^2 \psi''(v_{min} - 2\alpha\delta)$$

This proves the lemma. □

Lemma 4.2.3 (**Lemma 4.2 in [1]**). *If α satisfies the inequality*

$$-\psi'(v_{min} - 2\alpha\delta) + \psi'(v_{min}) \leq 2\delta. \tag{4.16}$$

then the following holds

$$f_1'(\alpha) \leq 0.$$

Proof. We may write, using Lemma 4.2.2, and also (4.9),

$$\begin{aligned} f_1'(\alpha) &= f_1'(0) + \int_0^\alpha f_1''(\xi) d\xi \\ &\leq -2\delta^2 + 2\delta^2 \int_0^\alpha \psi''(v_{min} - 2\xi\delta) d\xi \\ &= -2\delta^2 - \delta \int_0^\alpha \psi''(v_{min} - 2\xi\delta) d(v_{min} - 2\xi\delta) \\ &= -2\delta^2 - \delta \left(\psi'(v_{min} - 2\alpha\delta) - \psi'(v_{min})\right). \end{aligned}$$

Hence, $f_1'(\alpha) \leq 0$ will certainly hold if α satisfies

$$-\psi'(v_{min} - 2\alpha\delta) + \psi'(v_{min}) \leq 2\delta,$$

which proves the lemma. □

Lemma 4.2.4 (Lemma 4.3 in [1]). *Let ρ denote the inverse function of the restriction of $-\frac{1}{2}\psi'(t)$ to the interval $(0,1]$, as defined in (3.39). Then the largest step size α that satisfies (4.16) is given by*

$$\overline{\alpha} := \frac{1}{2\delta}\left(\rho\left(\delta\right) - \rho\left(2\delta\right)\right). \tag{4.17}$$

Proof. We want α such that (4.16) holds, with α as large as possible. Since $\psi''(t)$ is decreasing, the derivative with respect to v_{min} of the expression at the left in (4.16) (i.e. $-\psi''(v_{min} - 2\alpha\delta) + \psi''(v_{min})$) is negative. Hence, fixing δ, the smaller v_{min} is, the smaller α will be. One has

$$\delta = \frac{1}{2}\left\|\nabla\Psi(v)\right\| \geq \frac{1}{2}\left|\psi'(v_{min})\right| \geq -\frac{1}{2}\psi'(v_{min}).$$

Equality hold if and only if v_{min} is the only coordinate in v that differs from 1, and $v_{min} \leq 1$ (in which case $\psi'(v_{min}) \leq 0$). Hence, the worst situation for the step size occurs when v_{min} satisfies

$$-\frac{1}{2}\psi'(v_{min}) = \delta. \tag{4.18}$$

The derivative with respect to α of the left expression in (4.16) equals $2\delta\psi''(v_{min} - 2\alpha\delta) \geq 0$, and hence the left hand side is increasing in α. So the largest possible value of α satisfying (4.16), satisfies

$$-\frac{1}{2}\psi'(v_{min} - 2\alpha\delta) = 2\delta. \tag{4.19}$$

Due to the definition of ρ, (4.18) and (4.19) can be written as

$$v_{min} = \rho(\delta), \quad v_{min} - 2\alpha\delta = \rho(2\delta),$$

respectively. This implies,

$$\alpha = \frac{1}{2\delta}(v_{min} - \rho(2\delta)) = \frac{1}{2\delta}(\rho(\delta) - \rho(2\delta)),$$

proving the lemma. □

Lemma 4.2.5 (**Lemma 2.3.7 in [5]**). *Let $\overline{\alpha}$ be as defined in Lemma 4.2.4. Then*

$$\overline{\alpha} \geq \frac{1}{\psi''(\rho(2\delta))}. \tag{4.20}$$

Proof. By the definition of ρ ,

$$-\psi'(\rho(\delta)) = 2\delta.$$

Taking the derivative with respect to δ, we find

$$-\psi''(\rho(\delta))\rho'(\delta) = 2,$$

which implies that

$$\rho'(\delta) = -\frac{2}{\psi''(\rho(\delta))} < 0. \tag{4.21}$$

Hence ρ is monotonically decreasing in δ. An immediate consequence of (4.17) and (4.21) is

$$\overline{\alpha} = \frac{1}{2\delta}\int_{2\delta}^{\delta} \rho'(\sigma)d\sigma = \frac{1}{\delta}\int_{\delta}^{2\delta} \frac{d\sigma}{\psi''(\rho(\sigma))}. \tag{4.22}$$

The obtain a lower bound for $\overline{\alpha}$, we want to replace the argument of the last integral by its minimal value. So we want to know when $\psi''(\rho(\sigma))$ is maximal, for $\sigma \in [\delta, 2\delta]$. Due to (3.32), ψ'' is monotonically decreasing. So $\psi''(\rho(\sigma))$ is maximal when $\rho(\sigma)$ is minimal for $\sigma \in [\delta, 2\delta]$. Since ρ is monotonically decreasing this occurs when $\sigma = 2\delta$. Therefore

$$\overline{\alpha} = \frac{1}{\delta}\int_{\delta}^{2\delta} \frac{d\sigma}{\psi''(\rho(\sigma))} \leq \frac{1}{\delta}\frac{\delta}{\psi''(\rho(2\delta))} = \frac{1}{\psi''(\rho(2\delta))},$$

which proves the lemma. □

In the sequel, we will use the following notation:

$$\widetilde{\alpha} := \frac{1}{\psi''(\rho(2\delta))}, \tag{4.23}$$

and we will use $\widetilde{\alpha}$ as the default step size. From Lemma 4.2.5, we can see that $\overline{\alpha} \geq \widetilde{\alpha}$.

4.3 Reduction of Barrier Function during Inner Iteration

Using the lower bound on the step size that we previously determined, we can obtain the results on the decrease of the barrier function. Foremost, we begin with a technical lemma which we will use later.

Lemma 4.3.1 (**Lemma A.1.3 in [5]**). *Let $h(t)$ be a twice differentiable convex function with $h(0) = 0$, $h'(0) < 0$, and let $h(t)$ attain its (global) minimum at $t^* > 0$. If $h''(t)$ is increasing for $t \in [0, t^*]$ then*

$$h(t) \leq \tfrac{th'(0)}{2}, \quad 0 \leq t \leq t^*.$$

Proof. Using the hypothesis of the lemma we may write

$$\begin{aligned} h(t) &= \int_0^t h'(\xi)d\xi \\ &= h'(0)t + \int_0^t \int_0^\xi h''(\zeta)d\zeta d\xi \leq h'(0)t + \int_0^t \xi h''(\xi)d\xi \\ &= h'(0)t + \int_0^t \xi dh'(\xi) = h'(0)t + (\xi h'(\xi))|_0^t - \int_0^t h'(\xi)d\xi \\ &\leq h'(0)t - \int_0^t dh'(\xi) = h'(0)t - h(t). \end{aligned}$$

This implies the lemma. □

Lemma 4.3.2 (**Lemma 2.3.8 in [5]**). *If the step size α is such that $\alpha \leq \overline{\alpha}$ then*

$$f(\alpha) \leq -\alpha\delta^2. \tag{4.24}$$

Proof. Let $h(\alpha)$ be defined by

$$h(\alpha) := -2\alpha\delta^2 + \alpha\delta\psi'(v_{min}) - \frac{1}{2}\psi(v_{min}) + \frac{1}{2}\psi(v_{min} - 2\alpha\delta).$$

Then

$$h(0) = f_1(0) = 0, \quad h'(0) = f_1'(0) = -2\delta^2, \quad h''(\alpha) = 2\delta^2\psi''(v_{min} - 2\alpha\delta).$$

Due to Lemma 4.2.2, $f_1''(\alpha) \leq h''(\alpha)$. As a consequence, $f_1'(\alpha) \leq h'(\alpha)$ and $f_1(\alpha) \leq h(\alpha)$. Taking $\alpha \leq \overline{\alpha}$, with $\overline{\alpha}$ as defined in Lemma 4.2.4, we have

$$\begin{aligned} h'(\alpha) &= -2\delta^2 + 2\delta^2 \int_0^\alpha \psi''(v_{min} - 2\xi\delta)d\xi \\ &= -2\delta^2 - \delta(\psi'(v_{min} - 2\alpha\delta) - \psi'(v_{min})) \leq 0. \end{aligned}$$

Since $h''(\alpha)$ is increasing in α, using Lemma 4.3.1, we may write

$$f_1(\alpha) \leq h(\alpha) \leq \frac{1}{2}\alpha h'(0) = -\alpha\delta^2.$$

Since $f(\alpha) \leq f_1(\alpha)$, the proof is complete. □

If we combine the results of Lemma 4.2.5 and Lemma 4.3.2, we obtain

Theorem 4.3.3 (**Theorem 4.6 in [1]**). *With $\widetilde{\alpha}$ being the default step size, as given by (4.23), one has*

$$f(\widetilde{\alpha}) \leq -\frac{\delta^2}{\psi''(\rho(2\delta))}. \tag{4.25}$$

Lemma 4.3.4 (**Lemma 4.7 in [1]**). *The right hand side expression in (4.25) is monotonically decreasing in δ.*

Proof. Putting $t = \rho(2\delta)$, which implies $t \leq 1$, and which is equivalent to $4\delta = -\psi'(t)$, t is monotonically decreasing if δ increases. Hence the right hand expression in (4.25) is monotonically decreasing in δ if and only if the function

$$g(t) := \frac{(\psi'(t))^2}{16\psi''(t)}$$

is monotonically decreasing for $t \leq 1$. Note that $g(1) = 0$ and

$$g'(t) = \frac{2\psi'(t)\psi''(t)^2 - \psi'(t)^2\psi'''(t)}{16\psi''(t)^2}.$$

Hence, since $\psi'(t) < 0$ for $t < 1$, $g(t)$ is monotonically decreasing for $t \leq 1$ if and only if

$$2\psi''(t)^2 - \psi'(t)\psi'''(t) \geq 0, \quad t \leq 1.$$

The last inequality is satisfied, due to condition (3.33). Hence the lemma is proved.

□

Next, we would like to show the decrease as a function of $\Psi(v)$ instead of δ. For this, we need a lower bound on $\delta(v)$ in terms of $\Psi(v)$. The following lemma gives needed structure to find this particular bound.

Lemma 4.3.5 (Lemma 4.8 in [1]). *Suppose that $\psi(t_1) = \psi(t_2)$, with $t_1 \leq 1 \leq t_2$. Then $\psi'(t_1) \leq 0$ and $\psi'(t_2) \geq 0$, whereas*

$$-\psi'(t_1) \geq \psi'(t_2).$$

Proof. The lemma is obvious if $t_1 = 1$ or $t_2 = 1$, because then $\psi(t_1) = \psi(t_2) = 0$ implies $t_1 = t_2 = 1$. So we may assume that $t_1 < 1 < t_2$. Since $\psi(t_1) = \psi(t_2)$, Lemma 3.4.5 implies:

$$\frac{1}{2}(t_1 - 1)^2\psi''(1) < \psi(t_1) = \psi(t_2) < \frac{1}{2}(t_2 - 1)^2\psi''(1).$$

Hence, since $\psi''(1) > 0$, it follows that $t_2 - 1 > 1 - t_1$. Using this and Lemma 3.4.5, while assuming $-\psi'(t_1) < \psi'(t_2)$, we may write

$$\psi(t_2) > \frac{1}{2}(t_2-1)\psi'(t_2) > \frac{1}{2}(1-t_1)\psi'(t_2) > -\frac{1}{2}(1-t_1)\psi'(t_1) = \frac{1}{2}(t_1-1)\psi'(t_1) > \psi(t_1).$$

This contradiction proves the lemma. □

Theorem 4.3.6 (**Lemma 2.3.11 in [5]**). *One has*

$$\delta(v) \geq \frac{1}{2}\psi'\left(\gamma\left(\Psi(v)\right)\right).$$

Proof. The proof for this theorem can be found in [5]. □

Corollary 4.3.7 (**Corollary 2.3.13 in [5]**). *One has*

$$\delta(v) \geq \frac{\Psi(v)}{2\gamma(\Psi(v))}.$$

Proof. Using Theorem 4.3.6, i.e., $\delta(v) \geq \frac{1}{2}\psi'(\gamma(\Psi(v)))$, we obtain from Lemma 3.4.4 that

$$\delta(v) \geq \frac{1}{2}\psi'(\gamma(\Psi(v))) \geq \frac{\psi(\gamma(\Psi(v)))}{2\gamma(\Psi(v))} = \frac{\Psi(v)}{2\gamma(\Psi(v))}.$$

This proves the corollary. □

By compiling the results of Theorem 4.25 and Theorem 4.3.6, we obtain

$$f(\widetilde{\alpha}) \leq -\frac{(\psi'(\gamma(\Psi(v))))^2}{4\psi''(\rho(\psi'(\gamma(\Psi(v)))))}. \tag{4.26}$$

This expression shows the decrease in $\Psi(v)$ during an inner iteration completely in terms of ψ, its first and second derivatives, and the inverse functions ρ and γ.

CHAPTER 5

COMPLEXITY OF THE ALGORITHM

In the following section, we will calculate iteration bounds for short and long step algorithms for several eligible kernel functions. Note that this bound will depend on the choice of kernel function and θ. The choice of θ will lead to either a short or long step algorithm. We will also give a scheme that streamlines the calculation process.

5.1 Iteration Bounds

We prove the following two technical lemmas that will be needed in the sequel. The first lemma provides a result used in the proof of the second lemma.

Lemma 5.1.1 (**Lemma A.1 in [1]**). *If $\alpha \in [0,1]$, then*

$$(1+t)^{\alpha} \leq 1 + \alpha t, \quad \forall\, t \geq -1. \tag{5.1}$$

Proof. Consider the function $f(t) = (1+t)^{\alpha} - 1 - \alpha t$ for $t \geq -1$. One has $f'(t) = \alpha(1+t)^{\alpha-1} - \alpha$ and $f''(t) = \alpha(\alpha-1)(t+1)^{\alpha-2}$. Since $f''(t) \leq 0$, $f(t)$ is concave. Since $f'(0) = 0$, the function f is maximal at $t = 0$. Finally, since $f(0) = 0$, the lemma follows. □

Lemma 5.1.2 (**Lemma A.2 in [1]**). *Let $t_0, t_1, \ldots, t_K$ be a sequence of positive numbers such that*

$$t_{k+1} \leq t_k - \kappa t_k^{1-\omega}, \quad \textit{for } k = 0, 1, \cdots, K-1, \tag{5.2}$$

where $\kappa > 0$ and $0 < \omega \leq 1$. Then

$$K \leq \left\lfloor \frac{t_0^{\omega}}{\kappa\omega} \right\rfloor.$$

Proof. Using (5.2), we may write

$$0 < t_{k+1}^{\omega} \leq \left(t_k - \kappa t_k^{1-\omega}\right)^{\omega} = t_k^{\omega}\left(1 - \kappa t_k^{-\omega}\right)^{\omega} \leq t_k^{\omega}\left(1 - \kappa\omega t_k^{-\omega}\right) = t_k^{\omega} - \kappa\omega,$$

where the second inequality follows from (5.1). Hence, for each k, $t_k^{\omega} \leq t_0^{\omega} - k\omega\kappa$. Taking $k = K$ we obtain $0 < t_0^{\theta} - K\omega\kappa$, which implies the lemma. □

We would like to count the number of inner iterations needed to return the situation where $\Psi(v) \leq \tau$. After the update of μ to $(1-\omega)\mu$ we have, by Theorem 4.1.2 and (4.3),

$$\Psi(v_+) \leq L_{\psi}(n, \theta, \tau) = n\psi\left(\frac{\gamma\left(\frac{\tau}{n}\right)}{\sqrt{1-\theta}}\right). \tag{5.3}$$

We denote the value of $\Psi(v)$ after the μ-update as Ψ_0, and the subsequent values are denoted as Ψ_k, $k = 1, 2, \ldots$. The decrease on each inner iteration is given by (4.26). In the following section, we assume that the expression in the right hand side expression of (4.26) satisfies

$$\frac{\left(\psi'\left(\gamma\left(\Psi(v)\right)\right)\right)^2}{4\psi''\left(\rho\left(\psi'\left(\gamma\left(\Psi(v)\right)\right)\right)\right)} \geq \kappa\Psi(v)^{1-\omega}, \tag{5.4}$$

for some positive constants κ and ω, with $\omega \in (0, 1]$.

Lemma 5.1.3 (**Lemma 5.1 in [1]**). *If K denotes the number of inner iterations, we have*

$$K \leq \frac{\Psi_0^{\omega}}{\kappa\omega}. \tag{5.5}$$

Proof. The definition of K implies $\Psi_{K-1} > \tau$. After K iterations, we should return in the τ-neighborhood of the new μ-center, that is, $\Psi_K \leq \tau$. We know that by (4.26) and (5.4) we have,

$$\begin{aligned} f(\widetilde{\alpha}) &= \Psi_{k+1} - \Psi_k \\ &= \Psi(v^+) - \Psi(v) \\ &\leq -\kappa\Psi_k^{1-\omega}, \end{aligned}$$

which leads to

$$\Psi_{k+1} \leq \Psi_k - \kappa \Psi_k^{1-\omega}, \quad for\ k = 0, 1, \cdots, K-1.$$

We apply Lemma 5.1.1, with $t_k = \Psi_k$. This yields the desired inequality. □

The previous lemma gives us an estimate for the number of inner iterations in terms of Ψ_0 and the constants κ and ω. Recall, Ψ_0 is bounded above according to (5.3).

Next, we would like to find an upper bound on the number of outer iterations, which is represented by the number of barrier parameter updates. We provide the following lemma for finding such a bound.

Lemma 5.1.4 (**Lemma 11.17 in [15]**). *If the barrier parameter update μ has the initial value μ_0 and is repeatedly multiplied by $(1-\theta)$, with $0 < \theta < 1$, then after at most*

$$\left\lceil \frac{1}{\theta} log \frac{n\mu_0}{\epsilon} \right\rceil \tag{5.6}$$

iterations we have $n\mu \leq \epsilon$.

Proof. Initially, the duality gap is $n\mu_0$, and in each iteration it is reduced by the factor $(1-\theta)$. Hence, after k iterations the duality gap is smaller than ϵ if

$$(1-\theta)^k n\mu_0 \leq \epsilon.$$

Taking logarithms, this becomes

$$klog(1-\theta) + log(n\mu_0) \leq log\epsilon.$$

Since $-log(1-\theta) \geq \theta$, this certainly holds if

$$k\theta \geq log(n\mu_0) - log(\epsilon) = log\frac{n\mu_0}{\epsilon}.$$

This implies the lemma. □

Finally, we obtain an upper bound for the total number of iterations by multiplying the number of inner and outer iterations.

Theorem 5.1.5. *The upper bound on the total number of iterations to obtain ϵ-approximate solution of problem (2.13) using Generic Algorithm (in Table 3.1) is*

$$\frac{\Psi_0^{\omega}}{\theta\kappa\omega} log\frac{n}{\epsilon}. \tag{5.7}$$

Note that since Ψ_0 is usually not known, we use the upper bound on Ψ_0 given by (5.3). Hence, this leads to the following upper bound on the total number of iterations:

$$\frac{\Psi_0^{\omega}}{\theta\kappa\omega} log\frac{n}{\epsilon} \leq \frac{1}{\theta\kappa\omega}\left(n\psi\left(\frac{\gamma\left(\frac{\tau}{n}\right)}{\sqrt{1-\theta}}\right)\right)^{\omega} log\frac{n}{\epsilon}. \tag{5.8}$$

5.2 Introduction of the Scheme

The previous results can be summarized in the following way:

1. Input a kernel function ψ; an update parameter θ, $0 < \theta < 1$; a threshold parameter τ; and an accuracy parameter ϵ.

2. Solve the equation $-\frac{1}{2}\psi'(t) = s$ to get $\rho(s)$, the inverse function of $-\frac{1}{2}\psi'(t), t \in (0, 1]$. If the equation is hard to solve, derive a lower bound for $\rho(s)$.

3. Calculate the decrease of $\Psi(v)$ in terms of δ for the default step size $\widetilde{\alpha}$ from

$$f(\widetilde{\alpha}) \leq -\frac{\delta^2}{\psi''(\rho(2\delta))}.$$

4. Solve the equation $\psi(t) = s$ to get $\gamma(s)$, the inverse function of $\psi(t), t \geq 1$. If the equation is hard to solve, derive lower and upper bounds for $\gamma(s)$.

5. Derive a lower bound for δ in terms of $\Psi(v)$ by using
$$\delta(v) \geq \frac{1}{2}\psi'(\gamma(\Psi(v))).$$

6. Using the results of step 4 and step 5, find the positive constants κ and ω, with $\omega \in (0, 1]$, such that
$$f(\widetilde{\alpha}) \leq -\kappa\Psi(v)^{1-\omega}.$$

7. Calculate the uniform upper bound Ψ_0 for $\Psi(v)$ from
$$\Psi_0 \leq L_\psi(n, \theta, \tau) = n\psi\left(\frac{\gamma\left(\frac{\tau}{n}\right)}{\sqrt{1-\theta}}\right).$$

8. Derive an upper bound for the total number of iterations from
$$\frac{\Psi_0^\omega}{\theta\kappa\omega}\ log\ \frac{n}{\epsilon}.$$

9. Set $\tau = O(n)$ and $\theta = \Theta(1)$ so as to calculate an iteration bound for large update methods, or set $\tau = O(1)$ and $\theta = \Theta(\frac{1}{\sqrt{n}})$ so as to calculate an iteration bound for small update methods.

At the start of each inner iteration, we have $\Psi(v) \geq \tau$. By Theorem 4.3.6 this implies that $\delta(v) \geq -\frac{1}{2}\psi'(\gamma(\tau))$. We always assume that $\tau \geq 1$, and that τ is large enough to ensure that $\delta(v) \geq 1$ at the start of each inner iteration.

5.3 Several Technical Lemmas

In this section, we derive some technical lemmas that will turn out to be useful in finding upper and lower bounds in the scheme. This will especially prove useful in

the case where the inverse functions γ and ρ cannot be computed explicitly. In the following lemmas, we refer to the elgible kernel functions from Table 3.3.

Lemma 5.3.1 (**Lemma 2.5.1 in [5]**). *When $\psi(t) = \psi_i(t)$ and $1 \leq i \leq 7$, then*

$$\sqrt{1+2s} \leq \gamma(s) \leq 1 + \sqrt{2s}. \tag{5.9}$$

Proof. The inverse function of $\psi(t)$ for $t \in [1, \infty)$ is obtained by solving t from the equation $\psi(t) = s$, for $t \geq 1$. In almost all cases it is hard to solve this equation explicitly. However, we can easily find a lower and upper bound for t and this suffies for our goal. First one has

$$s = \psi(t) = \frac{t^2 - 1}{2} + \psi_b(t) \leq \frac{t^2 - 1}{2},$$

where $\psi_b(t)$ denotes the barrier term. The inequality is due to the fact that $\psi_b(1) = 0$ and $\psi_b'(t)$ is monotonically decreasing. It follows that

$$t = \gamma(s) \geq \sqrt{1+2s}.$$

For the second inequality we derive from (3.4.2) and $\psi''(t) \geq 1$ that

$$s = \psi(t) = \int_1^t \int_1^\xi \psi''(\zeta) d\zeta d\xi \geq \int_1^t \int_1^\xi d\zeta d\xi = \frac{1}{2}(t-1)^2,$$

which implies

$$t = \gamma(s) \leq 1 + \sqrt{2s}.$$

This completes the proof. □

Lemma 5.3.2 (**Lemma 2.5.2 in [5]**). *When $\psi(t) = \psi_i(t)$ with $i \in \{8, 10\}$, and $q \geq 2$, then*

$$t \leq 1 + \sqrt{t\psi(t)}, t \geq 1.$$

Proof. Defining $f(t) = t\psi(t) - (t-1)^2$ one has $f(1) = 0$ and $f'(t) = \psi(t) + t\psi'(t) - 2(t-1)$. Hence $f'(1) = 0$ and $f''(t) = 2\psi'(t) + t\psi''(t) - 2$. Since $f''(t) = (q-2)t^{-q} + pt^p + 2(t^p - 1) \geq 0$ for $\psi_8(t)$, and $f''(t) = (q-2)t^{-q} \geq 0$ for $\psi_{10}(t)$, the lemma follows. □

Lemma 5.3.3 (**Lemma 2.5.3 in [5]**). *Let $1 \leq i \leq 7$. Then one has*

$$L_\psi(n, \theta, \tau) \leq \frac{\psi''(1)}{2} \frac{(\sqrt{2\tau} + \theta\sqrt{n})^2}{1-\theta}.$$

Hence, if $\tau = O(1)$ and $\theta = \Theta\left(\frac{1}{\sqrt{n}}\right)$, then $\Psi_0 = O\left(\psi''(1)\right)$.

Proof. By Lemma 5.3.1 we have $\gamma(s) \leq 1 + \sqrt{2s}$. Hence, also using (5.3) we have

$$L_\psi(n, \theta, \tau) = n\psi\left(\frac{\gamma\left(\frac{\tau}{n}\right)}{\sqrt{1-\theta}}\right) \leq n\psi\left(\frac{1 + \sqrt{\frac{2\tau}{n}}}{\sqrt{1-\theta}}\right).$$

Applying Lemma 3.4.6 we obtain

$$\begin{aligned} L_\psi(n, \theta, \tau) \leq \tfrac{n\psi''(1)}{2}\left(\tfrac{1+\sqrt{\frac{2\tau}{n}}}{\sqrt{1-\theta}} - 1\right)^2 &\leq \tfrac{n\psi''(1)}{2}\left(\tfrac{\theta+\sqrt{\frac{2\tau}{n}}}{\sqrt{1-\theta}}\right)^2 \\ &= \tfrac{\psi''(1)}{2}\tfrac{(\sqrt{2\tau}+\theta\sqrt{n})^2}{1-\theta}, \end{aligned}$$

where we also used

$$1 - \sqrt{1-\theta} = \frac{\theta}{1 + \sqrt{1-\theta}} \leq \theta. \tag{5.10}$$

This proves the lemma. □

Lemma 5.3.4 (**Lemma 2.5.4 in [5]**). *Let $\underline{\rho} : [0, \infty) \to (0, 1]$ be the inverse function of the restriction of $-\psi_b'(t)$ to the interval $(0, 1]$. When $\psi(t) = \psi_i(t)$ and $1 \leq i \leq 7$, then*

$$\rho(s) \geq \underline{\rho}(1 + 2s).$$

Proof. Let $t = \rho(s)$. Due to the definition of ρ as the inverse function of $-\frac{1}{2}\psi'(t)$ for $t \leq 1$ this means that

$$-2s = \psi'(t) = t + \psi_b'(t), t \leq 1.$$

Since $t \leq 1$ this implies

$$-\psi_b'(t) = t + 2s \leq 1 + 2s.$$

Since $-\psi_b'(t)$ is monotonically decreasing ni all seven cases, it follows from this that

$$t = \rho(s) \geq \underline{\rho}(1 + 2s),$$

proving the lemma. □

5.4 Analysis of Several Eligible Kernel Functions

In this section, we will apply the scheme, described at the beginning of this chapter, to several eligible kernel functions listed in Table 3.3.

Example 1

We consider $\psi(t) = \psi_1(t)$:

$$\psi(t) = \frac{t^2 - 1}{2} - log(t).$$

Before we begin with the scheme, we will first provide the first three derivatives of ψ.

$\psi'(t)$	$t - \frac{1}{t}$
$\psi''(t)$	$1 + \frac{1}{t^2}$
$\psi'''(t)$	$-\frac{2}{t^3}$

We will show that the imposed conditions (3.31), (3.33), (3.34) and (3.35) hold for this particular kernel function. The left hand side of the conditions are given in the following table.

(3.31)	$2t$
(3.33)	$2+\frac{6}{t^2}$
(3.34)	$\frac{2(\beta^2-1)}{\beta t}$
(3.35)	$\frac{2}{t}$

It is easy to see that the conditions are satisfied, i.e. the function is an eligible kernel function.

We are now ready to begin with the scheme:

2. First, we want to solve the equation $-\frac{1}{2}\psi'(t) = s$ for $t \in (0,1]$, to obtain $\rho(s)$. By using the first derivative from above, we get the following:

$$\begin{aligned} -\tfrac{1}{2}\left(t-\tfrac{1}{t}\right) &= s \\ \left(t-\tfrac{1}{t}\right) &= -2s \\ \tfrac{t^2-1}{t} &= -2s \\ t^2+2st-1 &= 0. \end{aligned}$$

We can easily see that

$$t = -s+\sqrt{1+s^2},$$

which can be written equivalently as

$$t = \rho(s) = \frac{1}{s+\sqrt{s^2+1}}.$$

3. Now, we would like to calculate the decrease of $\Psi(v)$ in terms of δ. It follows that

$$\begin{aligned} f(\widetilde{\alpha}) &\leq -\frac{\delta^2}{\psi''(\rho(2\delta))} \\ &= -\frac{\delta^2}{\psi''\left(\frac{1}{2\delta+\sqrt{1+4\delta^2}}\right)} \\ &= -\frac{\delta^2}{1+\left(2\delta+\sqrt{1+4\delta^2}\right)^2} \\ &= -\frac{\delta^2}{2+8\delta^2+4\delta\sqrt{1+4\delta^2}}. \end{aligned}$$

Note that we have a bound on δ, i.e. $\delta \geq 1$. Hence, the above equation becomes

$$f(\widetilde{\alpha}) \leq -\frac{1}{10+4\sqrt{5}} \leq -\frac{1}{19}.$$

4. As we will see later, step 4 is not needed.

5. Since we have $\delta \geq 1$, we do not need to find the bound for δ again.

6. Next, we want to find the positive constants κ and ω. We have

$$f(\widetilde{\alpha}) \leq -\kappa\psi(v)^{1-\omega},$$

where $\kappa = \frac{1}{19}$ and $\omega = 1$.

7. In this step, we want to find an upper bound for $\Psi(v)$ immediately after a μ-update. Using Lemma 5.3.3, with $\psi''(1) = 2$, we obtain

$$\begin{aligned} \Psi_0 &\leq \frac{\psi''(1)}{2}\frac{(\sqrt{2\tau}+\theta\sqrt{n})^2}{1-\theta} \\ &\leq \frac{(\sqrt{2\tau}+\theta\sqrt{n})^2}{1-\theta}. \end{aligned}$$

8. Hence the total number of iterations is bounded above by the following

$$\frac{19(\sqrt{2\tau}+\theta\sqrt{n})^2}{\theta(1-\theta)}\ log\ \frac{n}{\epsilon}.$$

9. For large updated methods, with $\tau = O(n)$ and $\theta = \Theta(1)$, the right hand side expression is

$$O(n\ log\ \frac{n}{\epsilon}).$$

For small updated methods, with $\tau = O(1)$ and $\theta = \Theta(\frac{1}{\sqrt{n}})$, the right hand side expression is

$$O(\sqrt{n}\ log\ \frac{n}{\epsilon}).$$

Example 2

We consider $\psi(t) = \psi_{10}(t)$:

$$\psi(t) = \frac{t^{p+1} - 1}{p+1} + \frac{t^{1-q} - 1}{q-1},\ for\ p \in [0,1],\ and\ q > 1.$$

Before we begin with the scheme, we will first provide the first three derivatives of ψ.

$\psi'(t)$	$t^p - t^{-q}$
$\psi''(t)$	$pt^{p-1} + qt^{-q-1}$
$\psi'''(t)$	$-p(1-p)t^{p-2} - q(q+1)t^{-q-2}$

We will show that the imposed conditions (3.31), (3.33), (3.34) and (3.35) hold for this particular kernel function. The left hand side of the conditions are given in the following table.

(3.31)	$(p+1)t^p + (q-1)t^{-q}$
(3.33)	$\frac{p(p+1)t^{2p}+(q^2-p+4pq+p^2+q)t^{p-q}+q(q-1)t^{-2q}}{t^2}$
(3.34)	$\frac{(p+q)(\beta^p-\beta^q)}{t^{q+1-p}}$
(3.35)	$(p-1)t^p + (q+1)t^{-q}$

It is easy to see that the conditions are satisfied, i.e. the function is an eligible kernel function. Note that we will refer to this kernel function as the new kernel function.

We are now ready to begin with the scheme:

2. First, we want to solve the equation $-\frac{1}{2}\psi'(t) = s$ for $t \in (0, 1]$, to obtain $\rho(s)$. By using the first derivative from above, we get the following:

$$-\frac{1}{2}\left(t^p - t^{-q}\right) = s,$$

or equivalently

$$t^{-q} - t^p = 2s, \ t \in (0, 1].$$

Since $t \leq 1$, we have $t^{-q} = 2s + t^p \leq 2s + 1$, which implies

$$t \geq \frac{1}{(2s+1)^{\frac{1}{q}}}.$$

Hence,

$$\rho(s) \geq \frac{1}{(2s+1)^{\frac{1}{q}}}.$$

3. Next, we would like to calculate the decrease of $\Psi(v)$ in terms of δ. It follows that

$$\begin{aligned} f(\widetilde{\alpha}) &\leq -\frac{\delta^2}{\psi''(\rho(2\delta))} \\ &= -\frac{\delta^2}{\psi''\left((1+4\delta)^{-\frac{1}{q}}\right)} \\ &= -\frac{\delta^2}{p(\frac{1}{1+4\delta})^{\frac{-(1-p)}{q}} + q(\frac{1}{1+4\delta})^{\frac{-(1+q)}{q}}} \\ &= -\frac{\delta^2}{p(1+4\delta)^{\frac{1-p}{q}} + q(1+4\delta)^{\frac{1+q}{q}}} \\ &\leq -\frac{1}{p(1+4\delta)^{\frac{1-p}{q}} + q(1+4\delta)^{\frac{1+q}{q}}}. \end{aligned}$$

Since $(1+4\delta)^{1-p} \leq (1+4\delta)^{1+q}$, for $p \in [0,1]$, and $q \geq 1$, it follows that

$$\begin{aligned} f(\widetilde{\alpha}) &\leq -\frac{1}{p(1+4\delta)^{\frac{1+q}{q}} + q(1+4\delta)^{\frac{1+q}{q}}} \\ &\leq -\frac{1}{(p+q)(1+4\delta)^{\frac{1+q}{q}}}. \end{aligned} \tag{5.11}$$

4. Now, we want to calculate the upper and lower bounds on the inverse function $t = \gamma(s)$. This means that we have to find the upper and lower bounds of t in terms of s by using the equation $s = \psi(t)$. We have

$$s = \frac{t^{p+1}-1}{p+1} + \frac{t^{1-q}-1}{q-1}, \ \textit{where } p \in [0,1] \textit{ and } q > 1.$$

First, we find the lower bound. We have, from the above equation,

$$\begin{aligned} \frac{t^{p+1}-1}{p+1} &= s - \frac{t^{1-q}-1}{q-1} \\ &= s + \frac{1-t^{1-q}}{q-1}. \end{aligned}$$

Since $0 < \frac{1-t^{1-q}}{q-1} < 1$, we see

$$\begin{aligned} \frac{t^{p+1}-1}{p+1} &\geq s \\ t^{p+1}-1 &\geq s(p+1) \\ t &\geq (s(p+1)+1)^{\frac{1}{p+1}}. \end{aligned}$$

Hence, we have our lower bound for $\gamma(s)$.

Next, we want to find the upper bound. Again, we begin with

$$\begin{aligned} \frac{t^{p+1}-1}{p+1} &= s + \frac{1-t^{1-q}}{q-1} \\ t^{p+1}-1 &= s(p+1) + \frac{1-t^{1-q}}{q-1}(p+1) \\ t^{p+1} &= 1 + s(p+1) + \frac{p+1}{q-1}(1-t^{1-q}). \end{aligned}$$

We see that since $0 < 1 - t^{1-q} < 1$, then

$$\begin{aligned} t^{p+1} &\leq 1 + s(p+1) + \frac{p+1}{q-1} \\ &= s(p+1) + \frac{(q-1)+(p+1)}{q-1} \\ &= s(p+1) + \frac{p+q}{q-1}. \end{aligned}$$

Hence, our upper bound on $\gamma(s)$ is

$$t \leq \left(s(p+1) + \frac{p+q}{q-1} \right)^{\frac{1}{p+1}}$$

We may now write both the lower and upper bounds as follows:

$$(1 + (1+p)s)^{\frac{1}{1+p}} \leq \gamma(s) \leq \left((1+p)s + \frac{p+q}{q-1} \right)^{\frac{1}{1+p}}. \tag{5.12}$$

5. Next, by using $\delta(v) \geq \frac{1}{2}\psi'(\gamma(\Psi(v)))$, we determine a lower bound for δ in terms of $\Psi(v)$. Since $\psi''(t) > 0$ for $t \geq 1$, we know that ψ' is monotonically increasing for $t \geq 1$. Since $\psi'(t)$ is monotonically increasing, we may replace $\gamma(\Psi(v))$ by a smaller value. We will use the lower bound of $\gamma(\Psi(v))$ found in (5.12). We obtain the following:

$$\begin{aligned} \delta(v) &\geq \frac{1}{2}\psi'\left(\gamma(\Psi(v))\right) \\ &\geq \frac{1}{2}\psi'\left((1+(p+1)\Psi(v))^{\frac{1}{p+1}} \right) \\ &= \frac{1}{2}\left((1+(p+1)\Psi(v))^{\frac{p}{p+1}} - \frac{1}{(1+(p+1)\Psi(v))^{\frac{q}{p+1}}} \right) \\ &\geq \frac{1}{2}\left((1+(p+1)\Psi(v))^{\frac{p}{p+1}} - \frac{1}{(1+(p+1)\Psi(v))^{\frac{1}{p+1}}} \right) \\ &= \frac{1}{2}\left(\frac{(1+(p+1)\Psi(v))^{\frac{p}{p+1}}(1+(p+1)\Psi(v))^{\frac{1}{p+1}} - 1}{(1+(p+1)\Psi(v))^{\frac{1}{p+1}}} \right) \\ &= \frac{1}{2}\left(\frac{(1+(p+1)\Psi(v))^{\frac{p+1}{p+1}} - 1}{(1+(p+1)\Psi(v))^{\frac{1}{p+1}}} \right) \\ &= \frac{(p+1)\Psi(v)}{2(1+(p+1)\Psi(v))^{\frac{1}{p+1}}}. \end{aligned}$$

Then since $\Psi(v) \geq 1$, we obtain

$$\delta(v) \geq \frac{\Psi(v)}{2(1+2\Psi(v))^{\frac{1}{p+1}}} \geq \frac{\Psi(v)}{2(3\Psi(v))^{\frac{1}{p+1}}} \geq \frac{1}{6}\frac{\Psi(v)^{\frac{p+1}{p+1}}}{\Psi(v)^{\frac{1}{p+1}}} = \frac{1}{6}\Psi(v)^{\frac{p}{p+1}}.$$

Therefore, we have our lower bound for $\delta(v)$.

6. In this step, we will find the positive constants κ and ω. We begin with the right hand side expression in (5.11). Since it is monotonically decreasing in δ and using the lower bound we found in the previous step, we obtain

$$\begin{aligned} f(\widetilde{\alpha}) &\leq -\frac{1}{(p+q)(1+4\delta)^{\frac{q+1}{q}}} \\ &\leq -\frac{\Psi(v)^{\frac{2p}{p+1}}}{36(p+q)(\frac{2}{3}\Psi(v)^{\frac{p}{p+1}}+1)^{\frac{q+1}{q}}}. \end{aligned} \tag{5.13}$$

We can see, by observing the denominator from the above equation, that

$$\begin{aligned} (\frac{2}{3}\Psi(v)^{\frac{p}{p+1}}+1)^{\frac{q+1}{q}} &\leq (\frac{2}{3}\Psi(v)^{\frac{p}{p+1}}+\Psi(v)^{\frac{p}{p+1}})^{\frac{q+1}{q}} \\ &= (\frac{5}{3}\Psi(v)^{\frac{p}{1+p}})^{\frac{q+1}{q}} \\ &= (\frac{5}{3})^{\frac{q+1}{q}}\Psi(v)^{\frac{p}{p+1}\frac{q+1}{q}}. \end{aligned}$$

Hence, from (5.13), we have

$$\begin{aligned} f(\widetilde{\alpha}) &\leq -\frac{\Psi(v)^{\frac{2p}{p+1}}}{36(p+q)(\frac{5}{3})^{\frac{q+1}{q}}\Psi(v)^{\frac{p}{p+1}\frac{q+1}{q}}} \\ &= -\frac{\Psi(v)^{\frac{2p}{p+1}-\frac{p}{p+1}\frac{q+1}{q}}}{36(p+q)(\frac{5}{3})^{1+\frac{1}{q}}} \\ &= -\frac{\Psi(v)^{\frac{p(q-1)}{q(p+1)}}}{36(p+q)(\frac{5}{3})^{1+\frac{1}{q}}} \\ &\leq -\frac{\Psi(v)^{\frac{p(q-1)}{q(p+1)}}}{36(p+q)(\frac{5}{3})^{2}} \\ &= -\frac{\Psi(v)^{\frac{p(q-1)}{q(p+1)}}}{100(p+q)}. \end{aligned}$$

Therefore, we have

$$\Psi_{k+1} \leq \Psi_k - \kappa \Psi_k^{1-\omega}, \; k = 0, 1, \ldots, K-1,$$

with $\kappa = \frac{1}{100(p+q)}$ and $\omega = \frac{q+p}{q(1+p)}$. Note that K denotes the number of inner iterations. Thus K is bounded above by

$$K \leq 100(1+p)q\Psi_0^{\frac{q+p}{q(1+p)}}.$$

7. We need to find an upper bound of Ψ_0. We use Lemma 5.1.3 and $\psi(t) \leq \frac{t^{1+p}}{1+p}$ for $t \geq 1$. Thus,

$$\begin{aligned}
\Psi_0 &\leq n\psi\left(\frac{\gamma\left(\frac{\tau}{n}\right)}{\sqrt{1-\theta}}\right) \\
&\leq n\psi\left(\frac{\left(\frac{(p+1)\tau}{n} + \frac{p+q}{q-1}\right)^{\frac{1}{p+1}}}{\sqrt{1-\theta}}\right) \\
&\leq n\left(\frac{1}{p+1}\left(\frac{\left(\frac{(p+1)\tau}{n} + \frac{p+q}{q-1}\right)^{\frac{1}{p+1}}}{\sqrt{1-\theta}}\right)^{p+1}\right) \\
&= n\left(\frac{\frac{(p+1)\tau}{n} + \frac{p+q}{q-1}}{(p+1)(1-\theta)^{\frac{p+1}{2}}}\right) \\
&= \left(\frac{(p+1)\tau + \frac{p+q}{q-1}n}{(p+1)(1-\theta)^{\frac{p+1}{2}}}\right).
\end{aligned}$$

8. Hence, we get an upper bound for the total number of iterations,

$$\frac{100(1+p)q}{\theta(1+\theta)^{\frac{p+q}{2q}}}\left(\frac{(1+p)\tau + \frac{q+p}{q-1}n}{1+p}\right)^{\frac{p+q}{q(1-p)}} \; log\; \frac{n}{\epsilon}.$$

9. For large update methods the bound becomes

$$O\left(qn^{\frac{p+q}{q(1+p)}}\; log\; \frac{n}{\epsilon}\right),$$

and for small update methods it becomes

$$O\left(q\sqrt{n}n^{\frac{p+q}{q(1+p)}}\ log\ \frac{n}{\epsilon}\right).$$

The last bound can be refined, as shown below. We can go back to step 4, and use Lemma 5.3.2 to derive the tighter bounds for the inverse function γ of $\psi(t)$.

4. We have that

$$t \leq 1 + \sqrt{t\psi(t)}.$$

Substituting $t \leq \left((1+p)\psi(t) + \frac{q+p}{q-1}\right)^{\frac{1}{p+1}}$, we obtain the following

$$\begin{aligned} t = \gamma(s) &\leq 1 + \sqrt{t + \psi(t)} \\ &= 1 + \sqrt{\psi(t)}\sqrt{t} \\ &\leq 1 + \sqrt{\psi(t)}\sqrt{\left((p+1)\psi(t) + \frac{q+p}{q-1}\right)^{\frac{1}{p+1}}} \\ &\leq 1 + \sqrt{s}\left((p+1)s + \frac{q+p}{q-1}\right)^{\frac{1}{2(p+1)}} \end{aligned}$$

7. Thus, we obtain the following upper bound for Ψ_0:

$$\begin{aligned}
\Psi_0 &\leq n\psi\left(\frac{\gamma(\frac{\tau}{n})}{\sqrt{1-\theta}}\right) \\
&= n\psi\left(\frac{1+\sqrt{\frac{\tau}{n}}\left((p+1)\frac{\tau}{n}+\frac{p+q}{q-1}\right)^{\frac{1}{2(p+1)}}}{\sqrt{1-\theta}}\right) \\
&\leq n\left(\frac{1}{p+1}\left(\frac{1+\sqrt{\frac{\tau}{n}}\left((p+1)\frac{\tau}{n}+\frac{p+q}{q-1}\right)^{\frac{1}{2(p+1)}}}{\sqrt{1-\theta}}\right)^{p+1}\right) \\
&= n\left(\frac{1+\sqrt{\frac{\tau}{n}}\left((p+1)\frac{\tau}{n}+\frac{p+q}{q-1}\right)^{\frac{p+1}{2(p+1)}}}{(p+1)(1-\theta)^{\frac{p+1}{2}}}\right) \\
&\leq \frac{n(p+q)}{2}\left(\frac{1+\sqrt{\frac{\tau}{n}}\left((p+1)\frac{\tau}{n}+\frac{p+q}{q-1}\right)^{\frac{1}{2(p+1)}}}{\sqrt{1-\theta}}-1\right)^2.
\end{aligned}$$

Next, by using $1-\sqrt{1-\theta}\leq\theta$, from (5.10), we get

$$\begin{aligned}
\Psi_0 &= \frac{n(p+q)}{2}\left(\frac{1+\sqrt{\frac{\tau}{n}}\left((p+1)\frac{\tau}{n}+\frac{p+q}{q-1}\right)^{\frac{1}{2(p+1)}}-\sqrt{1-\theta}}{\sqrt{1-\theta}}\right)^2 \\
&\leq \frac{n(p+q)}{2}\left(\frac{\sqrt{\frac{\tau}{n}}\left((p+1)\frac{\tau}{n}+\frac{p+q}{q-1}\right)^{\frac{1}{2(p+1)}}+\theta}{\sqrt{1-\theta}}\right)^2 \\
&= \frac{n(p+q)}{2(1-\theta)}\left(\sqrt{\frac{\tau}{n}}\left((p+1)\frac{\tau}{n}+\frac{p+q}{q-1}\right)^{\frac{1}{2(p+1)}}+\theta\right)^2 \\
&\leq \frac{(p+q)}{2(1-\theta)}\left(\tau\left((p+1)\frac{\tau}{n}+\frac{p+q}{q-1}\right)^{\frac{1}{2(p+1)}}+\theta\sqrt{n}\right)^2.
\end{aligned}$$

8. Hence, the total number of iterations is bounded above by

$$\frac{100(p+1)q}{\theta}\left(\left(\frac{p+q}{2(1-\theta)}\right)\left(\tau\left((p+1)\frac{\tau}{n}+\frac{p+q}{q-1}\right)^{\frac{1}{2(p+1)}}+\theta\sqrt{n}\right)^2\right)^{\frac{(p+q)}{q(p+1)}} log\,\frac{n}{\epsilon}.$$

Since $\frac{p+q}{q(p+1)} \leq 1$ for all $p \in [0,1]$ and $q \geq 2$, then the bound can be updated to the following

$$\frac{50q(p+1)(p+q)}{\theta(1-\theta)}\left(\tau\left((p+1)\frac{\tau}{n}+\frac{p+q}{q-1}\right)^{\frac{1}{2(p+1)}}+\theta\sqrt{n}\right)^2 log\ \frac{n}{\epsilon}.$$

9. For small update methods and $p \in [0,1]$, the right hand side expression is

$$O\left(q^2\sqrt{n}\ log\ \frac{n}{\epsilon}\right).$$

Example 3

We consider $\psi(t) = \psi_6(t)$:

$$\psi(t) = \frac{t^2-1}{2} - \int_1^t e^{\frac{1}{\xi}-1}d\xi.$$

Before we begin with the scheme, we will first provide the first three derivatives of ψ.

$\psi'(t)$	$t - e^{\frac{1}{t}-1}$
$\psi''(t)$	$1 + \frac{e^{\frac{1}{t}-1}}{t^2}$
$\psi'''(t)$	$-\frac{1+2t}{t^4}e^{\frac{1}{t}-1}$

We will show that conditions (3.31), (3.33) and (3.35) hold for this particular kernel function. The left hand side of the conditions are given in the following table.

(3.31)	$2t + \frac{1-t}{t}e^{\frac{1}{t}-1}$
(3.33)	$\frac{1}{t^4}\left(2\left(1+e^{\frac{1}{t}-1}\right)^2 + (1+2t)\left(t-e^{\frac{1}{t}-1}\right)e^{\frac{1}{t}-1}\right)$
(3.35)	$\frac{1+t}{t}e^{\frac{1}{t}-1}$

It is easy to see that the conditions are satisfied, i.e. the function is an eligible kernel function.

We are now ready to begin with the scheme:

2. The inverse function of $-\psi_b'(t) = e^{\frac{1}{t}-1}$ is given by $\rho(s) = \frac{1}{1+log(s)}$. Hence, by Lemma 5.3.4, we have
$$\rho(s) \geq \frac{1}{1+log(1+2s)}.$$

3. We want to calculate the decrease of $\Psi(v)$ in terms of δ. It follows that
$$\begin{aligned} f(\widetilde{\alpha}) &\leq -\frac{\delta^2}{\psi''(\rho(2\delta))} \\ &\leq -\frac{\delta^2}{\psi''\left(\frac{1}{1+log(1+4\delta)}\right)} \\ &= -\delta^2\left(1+\frac{e^{\left(\frac{1}{\left(\frac{1}{1+log(1+4\delta)}\right)}-1\right)}}{\left(\frac{1}{1+log(1+4\delta)}\right)^2}\right)^{-1} \\ &= -\frac{\delta^2}{(1+log(1+4\delta))^2\left(e^{1+log(1+4\delta)-1}\right)} \\ &= -\frac{\delta^2}{1+(1+4\delta)(1+log(1+4\delta))^2}. \end{aligned} \tag{5.14}$$

4. Next, we would like to calculate the upper and lower bounds on the inverse function $t = \gamma(s)$. By Lemma 5.3.1 the inverse function of $\psi(t)$ for $t \in [1, \infty)$ satisfies
$$\sqrt{1+2s} \leq \gamma(s) \leq 1+\sqrt{2s}.$$
Also, we have that
$$\gamma\left(\Psi(v)\right) \geq \sqrt{1+2\Psi(v)}.$$

5. Using that $\delta(v) \geq \frac{1}{2}\psi'(\gamma(\Psi(v)))$, we determine a lower bound for δ in terms of $\Psi(v)$. We obtain

$$\begin{aligned}
\delta(v) &\geq \frac{1}{2}\psi'\left(\gamma(\Psi(v))\right) \\
&\geq \frac{1}{2}\left(\sqrt{1+2\Psi(v)}\right) \\
&\geq \frac{1}{2}\left(\sqrt{1+2\Psi} - e^{\frac{1}{\sqrt{1+2\Psi}}-1}\right).
\end{aligned}$$

Since $\frac{1}{\sqrt{1+2\Psi(v)}} - 1 < 0$, we see that

$$\begin{aligned}
\delta(v) &\geq \frac{1}{2}\left(\sqrt{1+2\Psi} - 1\right) \\
&= \frac{(\sqrt{1+2\Psi(v)}-1)(\sqrt{1+2\Psi(v)}+1)}{2(\sqrt{1+2\Psi(v)}+1)} \\
&= \frac{\Psi}{1+\sqrt{1+2\Psi}}.
\end{aligned}$$

6. We want to find the positive constants κ and ω. We begin with the right hand side expression in (5.14). Using the lower bound of $\delta(v)$ found in the previous step, we obtain the following:

$$\begin{aligned}
f(\widetilde{\alpha}) &\leq -\frac{\delta^2}{1+(1+log(1+4\delta))^2(1+4\delta)} \\
&\leq -\frac{\left(\frac{\Psi(v)}{1+\sqrt{1+2\Psi(v)}}\right)^2}{1+(1+log(1+4\delta))^2(1+4\delta)}.
\end{aligned}$$

Since $1 \leq \sqrt{\Psi(v)} \leq \Psi(v)$ and $\sqrt{\Psi(v)} \leq 1+\sqrt{1+2\Psi(v)} \leq 3\sqrt{\Psi(v)}$, we obtain

$$\begin{aligned}
f(\widetilde{\alpha}) &\leq -\frac{\left(\frac{\Psi(v)}{3\sqrt{\Psi(v)}}\right)^2}{1+(1+log(1+4\delta))^2(1+4\delta)} \\
&\leq -\frac{\left(\Psi(v)^{\frac{1}{2}}\right)^2}{9\left(1+(1+log(1+4\delta))^2(1+4\delta)\right)} \\
&\leq -\frac{\Psi(v)}{9\left(2(1+4\delta)(1+log(1+4\delta))^2\right)}.
\end{aligned}$$

Since $1+4\delta \leq \sqrt{\Psi(v)}+4\sqrt{\Psi(v)} = 5\sqrt{\Psi(v)}$ and $\Psi_0(v) \geq \Psi(v) \geq \tau \geq 1$, we have

$$\begin{aligned} f(\widetilde{\alpha}) &\leq -\frac{\Psi(v)}{90\sqrt{\Psi(v)}(1+log(1+4\sqrt{\Psi(v)}))^2} \\ &\leq -\frac{\Psi(v)^{\frac{1}{2}}}{90(1+log(1+4\sqrt{\Psi(v)}))^2} \\ &\leq -\frac{\Psi(v)^{\frac{1}{2}}}{90(1+log(1+4\sqrt{\Psi_0(v)}))^2}. \end{aligned}$$

Thus, it follows that

$$\Psi_{k+1} \leq \Psi_k - \kappa\Psi_k^{1-\omega}, k = 0, 1, \ldots, K-1,$$

with $\kappa = \frac{1}{90(1+log(1+4\sqrt{\Psi_0(v)}))^2}$, and $\omega = \frac{1}{2}$. As before, K denotes the number of inner iterations. Hence the number K of inner iterations is bounded above by the following:

$$K \leq \frac{\Psi_0^{\omega}}{\kappa\omega} = 180(1+log(1+4\sqrt{\Psi_0(v)}))^2\Psi_0(v)^{\frac{1}{2}}. \tag{5.15}$$

7. Next, we need to find an upper bound of Ψ_0. Using Lemma (5.3.3), with $\psi''(1) = 2$, we see

$$\Psi_0 \leq \frac{\psi''(1)}{2}\frac{(\sqrt{2\tau}+\theta\sqrt{n})^2}{(1-\theta)} = \frac{(\sqrt{2\tau}+\theta\sqrt{n})^2}{(1-\theta)}.$$

By using substitution in (5.15), we obtain

$$\begin{aligned} K &\leq 180\left(1+log\left(1+4\sqrt{\frac{(\sqrt{2\tau}+\theta\sqrt{n})^2}{(1-\theta)}}\right)\right)^2\left(\frac{(\sqrt{2\tau}+\theta\sqrt{n})^2}{(1-\theta)}\right)^{\frac{1}{2}} \\ &\leq 180\left(1+log\left(1+\frac{4(\sqrt{2\tau}+\theta\sqrt{n})}{\sqrt{1-\theta}}\right)\right)^2\left(\frac{\sqrt{2\tau}+\theta\sqrt{n}}{\sqrt{1-\theta}}\right). \end{aligned}$$

8. Hence, the upper bound for the total number of iterations is given by

$$180\left(1+log\left(1+\frac{4(\sqrt{2\tau}+\theta\sqrt{n})}{\sqrt{1-\theta}}\right)\right)^2\left(\frac{\sqrt{2\tau}+\theta\sqrt{n}}{\sqrt{1-\theta}}\right) log\ \frac{n}{\epsilon}.$$

9. For large update methods the bound becomes

$$O\left(\sqrt{n}(log(n))^2\ log\ \frac{n}{\epsilon}\right),$$

and for small update methods it becomes

$$O\left(\sqrt{n}\ log\ \frac{n}{\epsilon}\right).$$

5.5 Complexity Remarks

In the previous section, we calculated iteration bounds for several eligible kernel functions that are summarized in the table below.

Function	Short Step	Long Step
	$\theta = O(\frac{1}{\sqrt{n}})$, $\tau = O(1)$	$\theta = O(1)$, $\tau = O(n)$
$\psi_1 = \frac{t^2-1}{2} - log(t)$	$O(\sqrt{n}log\frac{n}{\epsilon})$	$O(nlog\frac{n}{\epsilon})$
$\psi_{10} = \frac{t^{1+p}-1}{1+p} + \frac{t^{1-q}-1}{q-1}$	$O(q^2\sqrt{n}log\frac{n}{\epsilon})$	$O(qn^{\frac{p+q}{q(1+p)}}log\frac{n}{\epsilon})$
$\psi_6 = \frac{t^2-1}{2} - \int_1^t e^{\frac{1}{\xi}-1}d\xi$	$O(\sqrt{n}log\frac{n}{\epsilon})$	$O(\sqrt{n}(logn)^2log\frac{n}{\epsilon})$

Table 5.1: Short and Long Step Complexity Bounds

We observe that short step methods give basically the same complexity. Long step methods have significant differences in complexity bounds. In the case when $p = 1$ and $q = log(n)$, the complexity of the long step method for ψ_{10} becomes

$$O(\sqrt{n}log(n)log\frac{n}{\epsilon}),$$

which matches the best known complexity for the large step IPM. Short step methods have better theoretical complexity but behave much worse in practice. Long step methods have worse theoretical complexity but behave much better in practice. This

discrepancy is well known in theory of IPM. One of the main reasons for the introduction of the eligible kernel functions was to improve theoretical complexity of the long step IPM.

CHAPTER 6

NUMERICAL RESULTS

In this chapter, The Generic Algorithm, as given in Table 3.1, is implemented in MATLAB for the classical logarithmic kernel function ψ_1 and for the parametric kernel function ψ_{10}. We summarize the results of the numerical tests for our implementation of the algorithm. There is a brief explanation of how the positive semidefinite matrix M matrix is generated. We will explain the reasoning of how and why the step size α was particularly chosen as a version of the minimum ratio test, rather than the theoretical bound. We will also analyze the effects of choosing certain values of p and q for the parametric kernel function ψ_{10}. Finally, we will discuss how different values τ and θ to affect the behavior of the algorithm. The testing was averaged for one-thousand tests, where each test includes the running of the classical IPM based on the classical kernel function ψ_1 and the "new" IPM algorithm based on the parametric kernel function ψ_{10} using all variations of given parameters.

6.1 Generating a PSD M Matrix

In this thesis, we considered a positive semi-definite matrix M (PSD). For general effectiveness of evaluation, our program first generates a random vector A of size n. We then created matrix B as the diagonal matrix from A. Then matrix B is input into a Householder QR factorization algorithm (A.2), which produces the matrices Q and R. From this we create a new M matrix, which is defined as

$$M = Q^T BQ.$$

We can see that M is guaranteed to be positive semi-definite from Householder QR factorization.

This new M matrix is an nxn diagonal with each diagonal element being one of the eigenvalues of the matrix. This particular formulation of M provides computationally faster results within the inner iterations, where solving the Newton system is performed. Considering the dense semidefinite matrices would not change the conclusions but would slow down the calculations and therefore were not considered. Note that for each of the trials performed, the identical M matrix was used to find solutions for the classical IPM and the new IPM. When a new test begins, a new PSD matrix M is generated and used.

6.2 Calculating the Step Size

Calculating the step size, α, is a very important process in the IPM. Using the scheme given in Chapter 6, we calculate the theoretical bound for the step size. This theoretical step size guarantees convergence but in practice it yields unfavorable results. Thus, we opted for a version of the minimum ratio test, which theoretically does not guarantee convergence, but generally works very well in practice.

The step size is chosen such that the positivity of x and s are preserved after their updates. We denote α_{max} as the maximum step size until one of the variables reaches 0. Hence,

$$\alpha_{max} = max\left\{\alpha \geq 0 : x_k + \alpha d_x \geq 0, s_k + \alpha d_s \geq 0\right\}.$$

In the program, we calculate α_{max} as the following:

$$\alpha_{max(primal)} = min\left\{-\frac{x_i}{(d_x)_i} : (d_x)_i < 0, i = 1, \ldots, n\right\},$$
$$\alpha_{max(dual)} = min\left\{-\frac{s_i}{(d_s)_i} : (d_s)_i < 0, i = 1, \ldots, n\right\},$$
$$\alpha_{max} = min\left\{\alpha_{max(primal)}, \alpha_{max(dual)}\right\}.$$

Size	Alpha Choice	Avg. Outer	Avg. Inner	Avg. CPU
10x10	Min Ratio	7	38	1.1707493e-03
10x10	Bounded	7	318	5.1859645e-03
100x100	Min Ratio	8	67	3.8338912e-03
100x100	Bounded	8	2204	6.3664592e-02

Table 6.1: Alpha Choice with Classical Alg. for $\theta = 0.95$ and $\tau = 1.50$

Since all variables must remain positive, we set

$$\alpha = min\left\{1, \nu\alpha_{max}\right\},$$

where $\nu \in (0, 1)$.

Table 6.1 clearly shows a significant reduction of the number of iterations by using the minimum ratio test instead of the theoretical step size. Note that any data testing from this point forward uses the minimum ratio test as the default step size.

6.3 Assigning Input Parameters

Another key step in the numerical testing is the choice of input parameters, namely θ and τ. From the Generic Algorithm, in Table 3.1, we know that θ represents the fixed barrier update parameter and τ represents a threshold parameter for the neighborhood size. These two parameters can substantially change the effectiveness of the algorithms, so choosing them insightfully is keen.

We begin with the update parameter, θ. For numerical testing, we varied θ as $\theta = \{0.95, 0.70, 0.45\}$. As we can see in Tables 6.3 - 6.11, assigning an aggressive theta, i.e. $\theta = 0.95$ or $\theta = 0.90$, gives the best results for the number of iterations and the CPU times. As $\theta \to 0$, the number of iterations and CPU time increase

drastically. Eventually, when θ is close to 0, the complementarity condition (3.1) fails and we are forced to chose another θ. Also, for $0.95 < \theta < 1$ there was minimal change in the number of iterations and CPU times.

Next, we discuses choosing the threshold neighborhood parameter, τ. For numerical testing, we varied τ as $\tau = \{1.50, 1.00, 0.50\}$. Results favored $\tau \approx 1.5$. If $\tau > 1.5$, then the number of iterations and the CPU times remained basically unchanged for all variations of θ. As $\tau \to 1$, the number of iterations and the CPU time slowly increased.

We conclude that for $\theta = 0.95$ and $\tau \approx 1.5$, the number of iterations and CPU times give the best results.

6.4 Choosing Values of p and q

As we have seen in Example 2 in Chapter 6, the parametric kernel function ψ_{10} is dependent on the choice of parameters p and q. We have the restrictions that $p \in [0, 1]$ and $q > 1$. For testing, we varied p and q such that $p = \{1.00, 0.50, 0\}$ and $q = \left\{1.1, \frac{1}{2}log(n), 2\right\}$. The results are listed in Tables 6.3 - 6.11.

We note that p has a greater effect on the number of iterations and CPU time. It seems that when $p \to 1$, we obtain the best results. On the other hand, the choice of q has limited effect on either number of iterations or CPU time. As $p \to 0$, the effect of q increases. On the other hand, as $p \to 2$, the effect of q decreases. Regardless of the value of p, we have $q \approx 2$ as the best value. Note that if $q >> 2$, then the results do not vary much or at all from $q \approx 2$.

6.5 Size of the Problem

The results listen in Tables 6.3 - 6.11, consider matrices M of size 10x10. However, the observation discussed above remains the same although the total number of iterations and CPU times increases. We refer to Table 6.2. The details of the numerical testing can be found in the Appendix.

Algorithm Used	Avg Outer	Avg Inner	Avg CPU Time
Classical (Ex. 1)			
10x10	7	38	1.1781826e-03
100x100	8	66	3.3573118e-03
400x400	8	72	4.8974455e-02
New (Ex. 2)			
p = 1 q = 2			
10x10	7	37	1.1699593e-03
100x100	8	68	3.3921420e-03
400x400	8	87	7.4341221e-02

Table 6.2: Size Comparison, using $\theta = 0.95$ and $\tau = 1.50$

6.6 Summary of Numerical Results

This section includes all of the results from numerical testing for the 10x10 matrices. The data has been summarized into tables, each of which represent the testing for particular parameters θ and τ.

Algorithm Used		Avg Outer	Avg Inner	Avg CPU Time
Classical (Ex. 1)		7	38	1.1781826e-03
New (Ex. 2)				
p = 1	q = 1.1	7	38	1.2739439e-03
	q = $\frac{1}{2}$log(n)	7	38	1.2518298e-03
	q=2	7	37	1.1699593e-03
p = 0.5	q = 1.1	7	109	3.6327258e-03
	q = $\frac{1}{2}$log(n)	7	107	3.6163204e-03
	q = 2	7	95	3.1560050e-03
p = 0	q = 1.1	7	277	9.7894558e-03
	q = $\frac{1}{2}$log(n)	7	275	9.7421903e-03
	q = 2	7	229	7.9720329e-03

Table 6.3: 10x10, $\theta = 0.95$, and $\tau = 1.50$

Algorithm Used		Avg Outer	Avg Inner	Avg CPU Time
Classical (Ex. 1)		7	40	1.1866316e-03
New (Ex. 2)				
p = 1	q = 1.1	7	39	1.2804327e-03
	q = $\frac{1}{2}$log(n)	7	39	1.2749560e-03
	q = 2	7	37	1.1766186e-03
p = 0.5	q = 1.1	7	109	3.5946404e-03
	q = $\frac{1}{2}$log(n)	7	108	3.5594126e-03
	q = 2	7	95	3.0650855e-03
p = 0	q = 1.1	7	267	9.0728374e-03
	q = $\frac{1}{2}$log(n)	7	260	8.8747501e-03
	q = 2	7	210	7.0380544e-03

Table 6.4: 10x10, $\theta = 0.95$, and $\tau = 1.00$

Algorithm Used		Avg Outer	Avg Inner	Avg CPU Time
Classical (Ex. 1)		7	44	1.2566194e-03
New (Ex. 2)				
p = 1	q = 1.1	7	42	1.3482595e-03
	q = $\frac{1}{2}$log(n)	7	42	1.3385351e-03
	q = 2	7	40	1.2241807e-03
p = 0.5	q = 1.1	7	112	3.6192852e-03
	q = $\frac{1}{2}$log(n)	7	110	3.5843564e-03
	q = 2	7	95	3.0168458e-03
p = 0	q = 1.1	7	254	8.2529216e-03
	q = $\frac{1}{2}$log(n)	7	248	8.0858721e-03
	q = 2	7	198	6.3271093e-03

Table 6.5: 10x10, $\theta = 0.95$, and $\tau = 0.50$

Algorithm Used		Avg Outer	Avg Inner	Avg CPU Time
Classical (Ex. 1)		15	113	1.8925094e-03
New (Ex. 2)				
p = 1	q = 1.1	15	113	2.0663405e-03
	q = $\frac{1}{2}$log(n)	15	114	2.0596193e-03
	q = 2	15	113	1.9769997e-03
p = 0.5	q = 1.1	15	200	3.4284957e-03
	q = $\frac{1}{2}$log(n)	15	197	3.3871744e-03
	q = 2	15	163	2.7369191e-03
p = 0	q = 1.1	15	400	6.3703423e-03
	q = $\frac{1}{2}$log(n)	15	388	6.1679001e-03
	q = 2	15	297	4.9362982e-03

Table 6.6: 10x10, $\theta = 0.70$, and $\tau = 1.50$

Algorithm Used		Avg Outer	Avg Inner	Avg CPU Time
Classical (Ex. 1)		15	116	1.9165180e-03
New (Ex. 2)				
p = 1	q = 1.1	15	115	2.1010919e-03
	q = $\frac{1}{2}$log(n)	15	116	2.0959718e-03
	q = 2	15	115	2.0192143e-03
p = 0.5	q = 1.1	15	211	3.5010942e-03
	q = $\frac{1}{2}$log(n)	15	209	3.4851345e-03
	q = 2	15	166	2.7595096e-03
p = 0	q = 1.1	15	403	6.2753436e-03
	q = $\frac{1}{2}$log(n)	15	391	6.1028770e-03
	q = 2	15	284	4.4973310e-03

Table 6.7: 10x10, $\theta = 0.70$, and $\tau = 1.00$

Algorithm Used		Avg Outer	Avg Inner	Avg CPU Time
Classical (Ex. 1)		15	124	1.9858573e-03
New (Ex. 2)				
p = 1	q = 1.1	15	122	2.1569292e-03
	q = $\frac{1}{2}$log(n)	15	122	2.1503614e-03
	q = 2	15	123	2.0806743e-03
p = 0.5	q = 1.1	15	224	3.6439707e-03
	q = $\frac{1}{2}$log(n)	15	219	3.5712820e-03
	q = 2	15	186	2.9310942e-03
p = 0	q = 1.1	15	421	6.4398217e-03
	q = $\frac{1}{2}$log(n)	15	406	6.2483806e-03
	q = 2	15	284	4.2733957e-03

Table 6.8: 10x10, $\theta = 0.70$, and $\tau = 0.50$

Algorithm Used		Avg Outer	Avg Inner	Avg CPU Time
Classical (Ex. 1)		28	360	3.4357949e-03
New (Ex. 2)				
p = 1	q = 1.1	28	361	3.7553539e-03
	q = $\frac{1}{2}$log(n)	28	361	3.7695443e-03
	q = 2	28	369	3.6099402e-03
p = 0.5	q = 1.1	28	373	3.9148508e-03
	q = $\frac{1}{2}$log(n)	28	371	3.8952529e-03
	q = 2	28	371	3.7543344e-03
p = 0	q = 1.1	28	684	6.3069938e-03
	q = $\frac{1}{2}$log(n)	28	662	6.1120600e-03
	q = 2	28	470	4.4385748e-03

Table 6.9: 10x10, $\theta = 0.45$, and $\tau = 1.50$

Algorithm Used		Avg Outer	Avg Inner	Avg CPU Time
Classical (Ex. 1)		28	378	3.4566270e-03
New (Ex. 2)				
p = 1	q = 1.1	28	378	3.7766377e-03
	q = $\frac{1}{2}$log(n)	28	376	3.7693849e-03
	q = 2	28	375	3.6139315e-03
p = 0.5	q = 1.1	28	396	4.0213644e-03
	q = $\frac{1}{2}$log(n)	28	389	3.9832340e-03
	q = 2	28	378	3.7564657e-03
p = 0	q = 1.1	28	703	6.3953198e-03
	q = $\frac{1}{2}$log(n)	28	702	6.3912497e-03
	q = 2	28	479	4.3836347e-03

Table 6.10: 10x10, $\theta = 0.45$, and $\tau = 1.00$

Algorithm Used		Avg Outer	Avg Inner	Avg CPU Time
Classical (Ex. 1)		28	378	3.4193476e-03
New (Ex. 2)				
p = 1	q = 1.1	28	378	3.7338356e-03
	q = $\frac{1}{2}$log(n)	28	378	3.7340061e-03
	q = 2	28	378	3.5833424e-03
p = 0.5	q = 1.1	28	465	4.4334947e-03
	q = $\frac{1}{2}$log(n)	28	443	4.2742547e-03
	q = 2	28	378	3.7157530e-03
p = 0	q = 1.1	28	780	6.7857580e-03
	q = $\frac{1}{2}$log(n)	28	737	6.4707788e-03
	q = 2	28	517	4.5454239e-03

Table 6.11: 10x10, $\theta = 0.45$, and $\tau = 0.50$

CHAPTER 7
CONCLUSION

In this thesis, we consider the linear complementarity problem (LCP). The LCP is observed in many practical problems such as the market equilibrium problem. It is also a framework in which some theoretical problems can be formulated such as a geometrical problem of finding a convex hull of a finite number of points in the plane. However, its importance mostly stems from the fact that optimality conditions of many important optimization problems, such as linear and quadratic programing problems, can be formulated as LCP.

We consider LCP in the standard form, although there exist other formulations (see Section 2.3). Also, different classes of matrix M can be considered. However, we concentrated on the class of positive semi-definite matrices because most practical applications and theoretical results involve this class of matrices.

The method used to solve the LCP is a kernel-based interior-point method (IPM). Kernel functions and their importance in the design and analysis of the IPM are discussed in Chapter 3. In this thesis, we consider a class of eligible kernel functions that is fairly general and includes the classical kernel function ψ_1 (Table 3.3) and recently considered class of self-regular functions [14]. The kernel functions were introduced with the intention to improve theoretical and practical performance of IPMs. The main emphasis of the thesis is the convergence analysis of the Generic Algorithm described in Table 3.1.

We have shown that the algorithm is globally convergent and provided a unified scheme (see Chapter 6) to calculate the upper bound on the total number of iterations for different kernel functions. In addition, we illustrated the process by providing

detailed calculations for several specific eligible kernel functions (see Examples in Chapter 6). We managed to obtain the best know complexity bounds for certain values of parameters of the parametric kernel function ψ_{10}.

The theoretical concepts were illustrated by basic implementation in MATLAB for the classical kernel function ψ_1 and for the parametric kernel function ψ_{10}, which involves parameters p and q. Both functions are listed in Table 3.3. A series of numerical tests were conducted showing that even these basic implementations have a potential for a good performance. The results indicate that an aggressive choice of values for parameter θ and τ, in the Generic Algorithm in Table 3.1, lead to a faster convergence. The same is the case if the version of minimum ratio test is used as a choice for the step size instead of the theoretical bound. Also, the choice of parameter p has a greater effect than the choice of q. A better implementation and more numerical testing would be necessary to draw more definite conclusions. However, that was not the main focus of the thesis.

APPENDIX A

MATLAB Code

The following contains all implemented MATLAB code.

A.1 Main Test Code : *testtrials.m*

The following is the code used to test and output the results from implementing the IPM algorithm using ψ_1 and ψ_{10}, where values of τ, θ, p, and q were used.

```
%This code creates z trials of both the classical method and the new methods where the user specifies n and z.
clear
clc
format long
%We first start with a size n and the number of test trials z.
z = 1000;
n = 100;
epsilon = 10^-6;

%outputs all output from the command window to a txt file
diary output10.txt
diary on

%Initial theta and tau, during the inner loop below tau is reduced by 0.5
%and similarly, theta is reduced by .25 each outer iteration
theta = .75;

%Input parameters, which may be changed to vary result speed and accuracy
for r = 1:3
 tau = 1.5;
     for b = 1:3

for l = 1:z
%Generate z random M matrices that are all nxn, then stored in the 3-D C matrix.

%Using n, we generate a diagonal matrix of size n, with random entries.
A = rand([n,1]);
B = diag(A);

%Use Householder QR Factorization to decompose B
[Q,R] = houseqr(B);

%We create a positive definite matrix M from matrix Q we found from QR method
%It is then stored in the 3D Matrix C
C(:,:,l) = Q*B*Q';
end
```

```
%Runs the IPM algorithm, z times, to get an average cpu time.  For each
%loop, we will use another Positive definite matrix out of our C matrix.

for k = 1:z
%For each test trial, we choose the set random matrix stored in the 3 Dimensional
%C matrix
M = C(:,:,k);

%Runs and times Interior-Point Algorithm for the 1st Kernel Function

[i1,xx1,ss1,inner1,time1]=IPM1(M,epsilon,tau,theta,n);
time1p(k) = time1;
outerit1(k) = i1;
innerit1(k) = inner1;

%Runs and times Interior-Point Algorithm for the 10th Kernel Function
%Input parameters, for psi_10,

%%%%% p = 1 %%%%%%%%   q varies
p1=1;
q1=1.1;
[i10,xx10_1,ss10_1,inner10,time10]=IPM10(M,epsilon,tau,theta,n,p1,q1);
time10_aa(k) = time10;
outerit10_aa(k) = i10;
innerit10_aa(k) = inner10;

p2=1;
q2=2;
[i10,xx10_2,ss10_2,inner10,time10]=IPM10(M,epsilon,tau,theta,n,p2,q2);
time10_ab(k) = time10;
outerit10_ab(k) = i10;
innerit10_ab(k) = inner10;

p3=1;
q3=.5*log(n);
[i10,xx10_3,ss10_3,inner10,time10]=IPM10(M,epsilon,tau,theta,n,p3,q3);
time10_ac(k) = time10;
outerit10_ac(k) = i10;
innerit10_ac(k) = inner10;

%%%%% p = .5 %%%%%%%%   q varies
p4=.5;
q4=1.1;
[i10,xx10_4,ss10_4,inner10,time10]=IPM10(M,epsilon,tau,theta,n,p4,q4);
time10_ba(k) = time10;
outerit10_ba(k) = i10;
innerit10_ba(k) = inner10;

p5=.5;
q5=2;
[i10,xx10_5,ss10_5,inner10,time10]=IPM10(M,epsilon,tau,theta,n,p5,q5);
time10_bb(k) = time10;
```

```
outerit10_bb(k) = i10;
innerit10_bb(k) = inner10;

p6=.5;
q6=.5*log(n);
[i10,xx10_6,ss10_6,inner10,time10]=IPM10(M,epsilon,tau,theta,n,p6,q6);
time10_bc(k) = time10;
outerit10_bc(k) = i10;
innerit10_bc(k) = inner10;

%%%%% p = 0 %%%%%%%%   q varies
p10=0;
q10=1.1;
[i10,xx10_10,ss10_10,inner10,time10]=IPM10(M,epsilon,tau,theta,n,p10,q10);
time10_da(k) = time10;
outerit10_da(k) = i10;
innerit10_da(k) = inner10;

p11=0;
q11=2;
[i10,xx10_11,ss10_11,inner10,time10]=IPM10(M,epsilon,tau,theta,n,p11,q11);
time10_db(k) = time10;
outerit10_db(k) = i10;
innerit10_db(k) = inner10;

p12=0;
q12=.5*log(n);
[i10,xx10_12,ss10_12,inner10,time10]=IPM10(M,epsilon,tau,theta,n,p12,q12);
time10_dc(k) = time10;
outerit10_dc(k) = i10;
innerit10_dc(k) = inner10;

end
avgtime1 = sum(time1p)/(z);
avgtime10aa = sum(time10_aa)/(z);
avgtime10ab = sum(time10_ab)/(z);
avgtime10ac = sum(time10_ac)/(z);
avgtime10ba = sum(time10_ba)/(z);
avgtime10bb = sum(time10_bb)/(z);
avgtime10bc = sum(time10_bc)/(z);
avgtime10da = sum(time10_da)/(z);
avgtime10db = sum(time10_db)/(z);
avgtime10dc = sum(time10_dc)/(z);

avgi1 = sum(outerit1)/z;
avgi10aa = sum(outerit10_aa)/z;
avgi10ab = sum(outerit10_ab)/z;
avgi10ac = sum(outerit10_ac)/z;
avgi10ba = sum(outerit10_ba)/z;
avgi10bb = sum(outerit10_bb)/z;
avgi10bc = sum(outerit10_bc)/z;
avgi10da = sum(outerit10_da)/z;
avgi10db = sum(outerit10_db)/z;
```

```
avgi10dc = sum(outerit10_dc)/z;

avginner1 = sum(innerit1)/z;
avginner10aa = sum(innerit10_aa)/z;
avginner10ab = sum(innerit10_ab)/z;
avginner10ac = sum(innerit10_ac)/z;
avginner10ba = sum(innerit10_ba)/z;
avginner10bb = sum(innerit10_bb)/z;
avginner10bc = sum(innerit10_bc)/z;
avginner10da = sum(innerit10_da)/z;
avginner10db = sum(innerit10_db)/z;
avginner10dc = sum(innerit10_dc)/z;

%Prints tables for results from above
fprintf('                      %d Test Trials                \n', z)
fprintf('               theta = %1.2f and tau = %1.2f         \n', theta, tau)
fprintf('                       For psi1(v)                  \n')
fprintf('------------------------------------------------------------------------ \n')
fprintf('------------------------------------------------------------------------ \n')
fprintf('  Size    |   Avg CPU time   |   Avg Outer   |   Avg Inner \n')
fprintf('__________|__________________|_______________|_____________\n')
fprintf(' %dx%d       %10.7d        %4.0f         %4.0f    \n',n,n,avgtime1,avgi1,avginner1)
fprintf('------------------------------------------------------------------------ \n')
%fprintf('\n Test No. | Inner It. | Outer It. |    CPU Time    \n')
%fprintf('__________|___________|___________|____________________\n')
%for u = 1:z
%  fprintf(' %4d         %4d       %4d        %10.7d   \n', u, outerit1(u), innerit1(u), time1p(u))
%end

fprintf('\n')
fprintf('************************************************************************ \n')
fprintf('\n')

fprintf('                       For psi10(v)                  \n')
fprintf('------------------------------------------------------------------------ \n')
fprintf('------------------------------------------------------------------------ \n')
fprintf('  Size    |  p  |  q  |       Avg CPU time     |   Avg Outer   |   Avg Inner \n')
fprintf('__________|_____|_____|________________________|_______________|_____________\n')
fprintf(' %dx%d     %1.1f   %1.1f       %10.7d          %4.0f           %4.0f    \n',n,n,n, p1, q1, avgtime10aa, avgi10aa, avginner10aa)
%fprintf('------------------------------------------------------------------------ \n')
%fprintf('\n Test No. | Inner It. | Outer It. |    CPU Time    \n')
%fprintf('__________|___________|___________|____________________\n')
%for j = 1:z
%  fprintf(' %4d         %4d       %4d        %10.7d   \n', j, outerit10_aa(j), innerit10_aa(j), time10_aa(j))
%end

%fprintf('\n')
%fprintf('************************************************************************ \n')
%fprintf('\n')

%fprintf('------------------------------------------------------------------------ \n')
%fprintf('  Size    |  p  |  q  |       Avg CPU time     |   Avg Outer   |   Avg Inner \n')
%fprintf('__________|_____|_____|________________________|_______________|_____________\n')
```

```
fprintf(' %dx%d     %1.1f   %1.1f       %10.7d          %4.0f           %4.0f    \n',n,n, p2, q2, avgtime10ab, avgi10ab, avginner10ab)
%fprintf('------------------------------------------------------------------------------ \n')
%fprintf('\n Test No. | Inner It. | Outer It. |    CPU Time    \n')
%fprintf('__________|___________|___________|____________________\n')
%for j = 1:z
%  fprintf(' %4d         %4d       %4d        %10.7d   \n', j, outerit10_ab(j), innerit10_ab(j), time10_ab(j))
%end

%fprintf('\n')
%fprintf('************************************************************************** \n')
%fprintf('\n')

%fprintf('------------------------------------------------------------------------- \n')
%fprintf('  Size    |  p  |  q  |       Avg CPU time     |   Avg Outer   |   Avg Inner \n')
%fprintf('__________|_____|_____|________________________|_______________|_____________\n')
fprintf(' %dx%d     %1.1f   %1.1f       %10.7d          %4.0f           %4.0f    \n',n,n,n, p3, q3, avgtime10ac, avgi10ac, avginner10ac)
%fprintf('------------------------------------------------------------------------- \n')
%fprintf('\n Test No. | Inner It. | Outer It. |    CPU Time    \n')
%fprintf('__________|___________|___________|____________________\n')
%for j = 1:z
%  fprintf(' %4d         %4d       %4d        %10.7d   \n', j, outerit10_ac(j), innerit10_ac(j), time10_ac(j))
%end

%fprintf('\n')
%fprintf('************************************************************************** \n')
%fprintf('\n')

%fprintf('------------------------------------------------------------------------- \n')
%fprintf('  Size    |  p  |  q  |       Avg CPU time     |   Avg Outer   |   Avg Inner \n')
%fprintf('__________|_____|_____|________________________|_______________|_____________\n')
fprintf(' %dx%d     %1.1f   %1.1f       %10.7d          %4.0f           %4.0f    \n',n,n,n, p4, q4, avgtime10ba, avgi10ba, avginner10ba)
%fprintf('------------------------------------------------------------------------- \n')
%fprintf('\n Test No. | Inner It. | Outer It. |    CPU Time    \n')
%fprintf('__________|___________|___________|____________________\n')
%for j = 1:z
%  fprintf(' %4d         %4d       %4d        %10.7d   \n', j, outerit10_ba(j), innerit10_ba(j), time10_ba(j))
%end

%fprintf('\n')
%fprintf('************************************************************************** \n')
%fprintf('\n')

%fprintf('------------------------------------------------------------------------- \n')
%fprintf('  Size    |  p  |  q  |       Avg CPU time     |   Avg Outer   |   Avg Inner \n')
%fprintf('__________|_____|_____|________________________|_______________|_____________\n')
fprintf(' %dx%d     %1.1f   %1.1f       %10.7d          %4.0f           %4.0f    \n',n,n,n, p5, q5, avgtime10bb, avgi10bb, avginner10bb)
%fprintf('------------------------------------------------------------------------- \n')
%fprintf('\n Test No. | Inner It. | Outer It. |    CPU Time    \n')
%fprintf('__________|___________|___________|____________________\n')
%for j = 1:z
%  fprintf(' %4d         %4d       %4d        %10.7d   \n', j, outerit10_bb(j), innerit10_bb(j), time10_bb(j))
%end
```

```
%fprintf('\n')
%fprintf('********************************************************************** \n')
%fprintf('\n')

%fprintf('---------------------------------------------------------------------- \n')
%fprintf('  Size    |  p  |  q  |       Avg CPU time     |   Avg Outer   |   Avg Inner \n')
%fprintf('__________|_____|_____|________________________|_______________|_____________\n')
fprintf(' %dx%d     %1.1f   %1.1f       %10.7d          %4.0f           %4.0f    \n',n,n, p6, q6, avgtime10bc, avgi10bc, avginner10bc)
%fprintf('---------------------------------------------------------------------- \n')
%fprintf('\n Test No. | Inner It. | Outer It. |    CPU Time    \n')
%fprintf('__________|___________|___________|____________________\n')
%for j = 1:z
%  fprintf(' %4d         %4d       %4d        %10.7d   \n', j, outerit10_bc(j), innerit10_bc(j), time10_bc(j))
%end

%fprintf('\n')
%fprintf('********************************************************************** \n')
%fprintf('\n')

%fprintf('---------------------------------------------------------------------- \n')
%fprintf('  Size    |  p  |  q  |       Avg CPU time     |   Avg Outer   |   Avg Inner \n')
%fprintf('__________|_____|_____|________________________|_______________|_____________\n')
fprintf(' %dx%d     %1.1f   %1.1f       %10.7d          %4.0f           %4.0f    \n',n,n,n, p10, q10, avgtime10da, avgi10da, avginner10da)
%fprintf('---------------------------------------------------------------------- \n')
%fprintf('\n Test No. | Inner It. | Outer It. |    CPU Time    \n')
%fprintf('__________|___________|___________|____________________\n')
%for j = 1:z
%  fprintf(' %4d         %4d       %4d        %10.7d   \n', j, outerit10_da(j), innerit10_da(j), time10_da(j))
%end

%fprintf('\n')
%fprintf('********************************************************************** \n')
%fprintf('\n')

%fprintf('---------------------------------------------------------------------- \n')
%fprintf('  Size    |  p  |  q  |       Avg CPU time     |   Avg Outer   |   Avg Inner \n')
%fprintf('__________|_____|_____|________________________|_______________|_____________\n')
fprintf(' %dx%d     %1.1f   %1.1f       %10.7d          %4.0f           %4.0f    \n',n,n,n, p11, q11, avgtime10db, avgi10db, avginner10db)
%fprintf('---------------------------------------------------------------------- \n')
%fprintf('\n Test No. | Inner It. | Outer It. |    CPU Time    \n')
%fprintf('__________|___________|___________|____________________\n')
%for j = 1:z
%  fprintf(' %4d         %4d       %4d        %10.7d   \n', j, outerit10_db(j), innerit10_db(j), time10_db(j))
%end

%fprintf('\n')
%fprintf('********************************************************************** \n')
%fprintf('\n')

%fprintf('---------------------------------------------------------------------- \n')
%fprintf('  Size    |  p  |  q  |       Avg CPU time     |   Avg Outer   |   Avg Inner \n')
%fprintf('__________|_____|_____|________________________|_______________|_____________\n')
fprintf(' %dx%d     %1.1f   %1.1f       %10.7d          %4.0f           %4.0f    \n',n,n,n, p12, q12, avgtime10dc, avgi10dc, avginner10dc)
```

```
%fprintf('---------------------------------------------------------------------- \n')
%fprintf('\n Test No. | Inner It. | Outer It. |    CPU Time    \n')
%fprintf('__________|___________|___________|____________________\n')
%for j = 1:z
%  fprintf(' %4d         %4d       %4d        %10.7d   \n', j, outerit10_dc(j), innerit10_dc(j), time10_dc(j))
%end

fprintf('\n')
fprintf('************************************************************************ \n')
fprintf('************************************************************************ \n')
fprintf('\n')

%reduces tau by set value 0.50
     tau = tau - 0.5;
     end
%reduces theta by set value 0.25
theta = theta - 0.25;
end
diary off
```

A.2 Step Size Bound Tests : *alphatesttrials.m*

The following is the code used to test different bounds for selecting step size, α. It is an adapted form of the above code.

```
clear
clc
format long
%We first start with a size n and the numerber of test trials z.
z = 1000;
n = 100;

%Input parameters, which may be changed to vary result speed and accuracy
theta = .95;
tau = 1.5;
epsilon = 10^-6;

diary ALPHATEST100.txt
diary on

%Runs the IPM algorithm, z times, to get an average cpu time.  For each
%loop, we will generate another Positive definite matrix of the same size.
for k = 1:z

%For algorithm beta testing, we use the seed command to fix one particular
%random matrix, during numerical analysis, we comment this out to generate
%a new random matrix during each trial.
%rand('seed',10);
```

```
 %Using n, we generate a diagonal matrix of size n, with random entries.
A = rand([n,1]);
B = diag(A);

%Use Householder QR Factorization to decompose B
[Q,R] = houseqr(B);

%We create a positive definite matrix M from matrix Q we found from QR method
M = Q*B*Q';

%Runs and times Interior-Point Algorithm for the 1st Kernel Function
%minRatioTest for alpha
[i1,xx1,ss1,inner1,time1]=IPM1(M,epsilon,tau,theta,n);
time1p(k) = time1;
outerit1(k) = i1;
innerit1(k) = inner1;

%Runs and times Interior-Point Algorithm for the 1st Kernel Function
%bound for alpha

[i1,xx1_1,ss1_1,inner1,time1]=IPM1a(M,epsilon,tau,theta,n);
time1_a(k) = time1;
outerit1_a(k) = i1;
innerit1_a(k) = inner1;

end
avgtime1 = sum(time1p)/(z);
avgtime1a = sum(time1_a)/(z);

avgi1 = sum(outerit1)/z;
avgi1a = sum(outerit1_a)/z;

avginner1 = sum(innerit1)/z;
avginner1a = sum(innerit1_a)/z;

fprintf('                   For psi1(v) using alpha as min ratio                \n')
fprintf('---------------------------------------------------------------- \n')
fprintf('---------------------------------------------------------------- \n')
fprintf('  Size    |   Avg CPU time   |   Avg Outer   |   Avg Inner \n')
fprintf('__________|__________________|_______________|_____________\n')
fprintf(' %dx%d       %10.7d        %4.0f         %4.0f    \n',n,n,avgtime1,avgi1,avginner1)
%fprintf('------------------------------------------------------------------------------ \n')
%fprintf('\n Test No. | Inner It. | Outer It. |    CPU Time    \n')
%fprintf('__________|___________|___________|____________________\n')
%for u = 1:z
%  fprintf(' %4d         %4d       %4d        %10.7d   \n', u, outerit1(u), innerit1(u), time1p(u))
%end

%fprintf('\n')
%fprintf('************************************************************************** \n')
%fprintf('\n')

fprintf('               For psi1(v) using alpha bounded                  \n')
```

```
fprintf('------------------------------------------------------------ \n')
fprintf('------------------------------------------------------------ \n')
fprintf('  Size    |   Avg CPU time   |   Avg Outer   |   Avg Inner \n')
fprintf('__________|__________________|_______________|_____________\n')
fprintf(' %dx%d       %10.7d        %4.0f         %4.0f    \n',n,n,avgtime1a,avgi1a,avginner1a)
%fprintf('------------------------------------------------------------------------ \n')
%fprintf('\n Test No. | Inner It. | Outer It. |    CPU Time    \n')
%fprintf('__________|___________|___________|____________________\n')
%for j = 1:z
%  fprintf(' %4d         %4d       %4d        %10.7d   \n', j, outerit1_a(j), innerit1_a(j), time1_a(j))
%end

%fprintf('\n')
%fprintf('************************************************************************ \n')
%fprintf('\n')

diary off
```

A.3 M Generator : *houseqr.m*

The code below is an adaptation of Householder QR factorization; it was used in the process of generating a PSD M matrix.

```
%This function performs Householder QR factorization of a m by n matrix A
%variables:
%A -- given inital matrix
%m,n -- size of A matrix

function [Q,R]=houseqr(A)
[m,n]=size(A);
Q=eye(m);

for k=1:min(m-1,n)
    c=norm(A(k:m,k),2)*sign(A(k,k));
    v=A(k:m,k);
    v(1)=A(k,k)+c;
    a=2/(v'*v);
    A(k:m,k)=-c*eye(m-k+1,1);

    for j=k+1:n
        A(k:m,j)=A(k:m,j)-a*(v'*A(k:m,j))*v;
    end

    for j=1:m
        Q(k:m,j)=Q(k:m,j)-a*(v'*Q(k:m,j))*v;
    end
end
R=A;
Q=Q';
```

```
%This program restructures A so that in the end of the algorithm the
%initial A matrix is made to be the R matrix.  We also had to add the
%stipulation of the sign of c, so that we avoid cancelation.
```

A.4 Primal-Dual Algorithm for ψ_1 : *IPM1.m*

This program was used as the primal-dual algorithm for ψ_1.

```
%This function solves the IPM using the classical psi function
function [i1,xx1,ss1,inner1,time1]=IPM1(M,epsilon,tau,theta,n)
    i=1;
    in=0;
    clear x s v mu xx1 ss1 dx ds inds indx
    % x, s are initialized as a vector of ones.
    x=ones(n,1);
    s=ones(n,1);
    mu=1;

    v = zeros(n,1);
    xx1 = zeros(n,n);
    ss1 = zeros(n,n);
    in1 = zeros(100*n,1);

    %We initialize variables xx, ss, mu2 to keep a record of the results of
    %x, s, and mu in each outer iteration, respectively.
    xx1(i,:)=x;
    ss1(i,:)=s;

    tic;
    %Outer Loop
    while n*mu > epsilon
        i = i+1;
        mu=(1-theta)*mu;
        v=sqrt(x.*s./mu);
        %Inner Loop
        while sum(psi1(v)) > tau
            %Solves the Newton system
            [dx,ds]=SolveSystem1(M,x,s,mu,v);

            %Chooses alpha based on the minimum ratio test.
            inds=find(ds<0);
            indx=find(dx<0);
            alpha=min(abs([1;theta*s(inds)./ds(inds);theta*x(indx)./dx(indx)]));

            %Update x, s, and v
            x=x+alpha*dx;
            s=s+alpha*ds;
            v=sqrt(x.*s./mu);
            in = in + 1;
        end
           %When inner loop finishes we are left with a final value of x,s,
           %and mu for the current iterate.
           xx1(i,:)=x';
           ss1(i,:)=s';
           in1(i) = in;
```

```
            %This loop prevents an infinite loop
            if i>1000
               fprintf('Algorithm fails to terminate. Try different values
               for tau and theta.')
               break;
            end
    end
    %Checks the complementarity condition (x'*s=0) for the final resulting
    %values of x,s when the algorithm terminates.  If condition is out of
    %tolerance, then user is warned.
    comp = x'*s;
   if comp > 10^-2
       fprintf('The complementarity condition is too large, choose different input
       values for theta and/or tau!')
   end
   time1=toc;
   inner1=sum(in1);
   i1=i;
end
```

A.5 Primal-Dual Algorithm for ψ_{10} : *IPM10.m*

This program was used as the primal-dual algorithm for ψ_{10}.

```
%This function solves the IPM using the new psi function
function [i10,xx10,ss10,inner10,time10]=IPM10(M,epsilon,tau,theta,n,p,q)
    i=1;
    in=0;

    clear x s v mu xx10 ss10 dx ds inds indx

    % x, s are initialized as a vector of ones.
    x=ones(n,1);
    s=ones(n,1);
    mu=1;

    xx10 = zeros(n,n);
    ss10 = zeros(n,n);
    in10 = zeros(100*n,1);
    v = zeros(n,1);

    %We initialize variables xx, ss, mu2 to keep a record of the results of
    %x, s, and mu in each outer iteration, respectively.
    xx10(i,:)=x;
    ss10(i,:)=s;

    tic;
    %Outer Loop
    while n*mu > epsilon
```

```
        i = i+1;
        mu=(1-theta)*mu;
        v=sqrt(x.*s./mu);
        %Inner Loop
        while sum(psi10(v,p,q)) > tau
            %Solves the
            [dx,ds]=SolveSystem10(M,x,s,mu,v,p,q);

            %Chooses alpha based on the minimum ratio test.
            inds=find(ds<0);
            indx=find(dx<0);
            alpha=min(abs([1;theta*s(inds)./ds(inds);theta*x(indx)./dx(indx)]));

            %Update x, s, and v
            x=x+alpha*dx;
            s=s+alpha*ds;
            v=sqrt(x.*s./mu);
            in = in + 1;
        end
           %When inner loop finishes we are left with a final value of x,s,
           %and mu for the current iterate.
           xx10(i,:)=x';
           ss10(i,:)=s';
           in10(i) = in;

            %This loop prevents an infinite loop
            if i>1000
               fprintf('Algorithm fails to terminate.  Try different values for tau and theta.')
               break;
            end
    end
    %Checks the complementarity condition (x'*s=0) for the final resulting
    %values of x,s when the algorithm terminates.  If condition is out of
    %tolerance, then user is warned.
    comp = x'*s;
   if comp > 10^-2
       fprintf('The complementarity condition is too large, choose different input
       values for theta and/or tau!')
   end
   time10=toc;
   inner10=sum(in10);
   i10=i;
end
```

A.6 Step-Size Primal-Dual Algorithm : *IPMa.m*

This Primal-Dual Algorithm was adapted for use in the test trials for determining efficiency of different step-sizes.

```
%This function solves the IPM using the classical psi function
%BUT alpha is chosen by its bound!
function [i1,xx1,ss1,inner1,time1]=IPM1a(M,epsilon,tau,theta,n)
    i=1;
    in=0;
    clear x s v mu xx1 ss1 dx ds inds indx
    % x, s are initialized as a vector of ones.
    x=ones(n,1);
    s=ones(n,1);
    mu=1;

    v = zeros(n,1);
    xx1 = zeros(n,n);
    ss1 = zeros(n,n);
    in1 = zeros(100*n,1);

    %We initialize variables xx, ss, mu2 to keep a record of the results of
    %x, s, and mu in each outer iteration, respectively.
    xx1(i,:)=x;
    ss1(i,:)=s;

    tic;
    %Outer Loop
    while n*mu > epsilon
        i = i+1;
        mu=(1-theta)*mu;
        v=sqrt(x.*s./mu);
        %Inner Loop
        while sum(psi1(v)) > tau
            %Solves the Newton system
            [dx,ds]=SolveSystem1(M,x,s,mu,v);

            %This is a bound for alpha
            delta = .5*norm(dx+ds,2);
            alpha = 1/(psi1_2((1/(2*delta+sqrt(1+4*delta^2)))));

            %Update x, s, and v
            x=x+alpha*dx;
            s=s+alpha*ds;
            v=sqrt(x.*s./mu);
            in = in + 1;
        end
           %When inner loop finishes we are left with a final value of x,s,
           %and mu for the current iterate.
           xx1(i,:)=x';
           ss1(i,:)=s';
           in1(i) = in;
            %This loop prevents an infinite loop
            if i>1000
               fprintf('Algorithm fails to terminate.  Try different values
               for tau and theta.')
               break;
            end
```

```
    end
    %Checks the complementarity condition (x'*s=0) for the final resulting
    %values of x,s when the algorithm terminates.  If condition is out of
    %tolerance, then user is warned.
    comp = x'*s;
   if comp > 10^-2
       fprintf('The complementarity condition is too large, choose different
       input values for theta and/or tau!')
   end
   time1=toc;
   inner1=sum(in1);
   i1=i;
end
```

A.7 Newton System Solver for ψ_1 : *SolveSystem1.m*

The following code was used specifically for ψ_1 to solve the Newton system given in (3.23).

```
function [dx,ds]=SolveSystem1(M,x,s,mu,v)
% This function solves the following system
% -M*dx + ds = 0
% s*dx + x*ds = - mu*v*gradient(Psi(v))
%
%where gradient(v)=psi1_1(v)
    X=diag(x);
    S=diag(s);

%   We see since, ds = M*dx, then we have that
%   (S+X*M)dx = r
%   Mtilda*dx = r
    Mtilda = S + X.*M;
    r = -mu.*v.*psi1_1(v);

%   Hence, we solve the following to solve the system:
    dx = Mtilda\r;
    ds = M*dx;
end
```

A.8 Newton System Solver for ψ_{10} : *SolveSystem10.m*

The following code was used specifically for ψ_{10} to solve the Newton system given in (3.23).

```
function [dx,ds]=SolveSystem10(M,x,s,mu,v,p,q)
```

```
% This function solves the following system
% -M*dx + ds = 0
% s*dx + x*ds = - mu*v*gradient(Psi(v))
%
%where gradient(v)=psi10_1(v)
    X=diag(x);
    S=diag(s);

%   We see since, ds = M*dx, then we have that
%   (S+X*M)dx = r
%   Mtilda*dx = r
    Mtilda = S + X.*M;
    r = -mu.*v.*psi10_1(v,p,q);
%   Hence, we solve the following to solve the system:
    dx = Mtilda\r;
    ds = M*dx;
end
```

A.9 ψ Function Files : *psi1.m, psi11.m ,psi12.m, psi10.m, psi101.m*

```
%function file for the classical kernel function (psi1.m)
function y = psi1(v)
y = (v.^2-1)./2 -log(v);

%first derivative for the classical kernel function (psi1_1.m)
function y = psi1_1(v)
y = v - 1./v;

%this function is the second derivative for the classical kernel function
function y = psi1_2(v)
y = 1 + 1./v.^2;

%function file for the new kernel function (psi10.m)
function y = psi10(v,p,q)
y = (v.^(1+p)-1)./(1+p) + (v.^(1-q)-1)./(q-1);

%function file for the first derivative for the new kernel function (psi10_1.m)
function y = psi10_1(v,p,q)
y = v.^p - v.^(-q);
```

APPENDIX B

MATLAB Output

The output for the 10x10 tests were placed into concise tables in Chapter 7. Below we provide the entire output of our tests with problems of the size 100x100.

B.1 *100x100 Output*

```
          1000 Test Trials \ theta = 0.95 and tau = 1.50
                       For psi1(v)
-------------------------------------------------------------------------
  Size    |   Avg CPU time   |   Avg Outer   |   Avg Inner
----------|------------------|---------------|-------------
 100x100       3.3573118e-03           8           66
-------------------------------------------------------------------------
                       For psi10(v)
-------------------------------------------------------------------------
  Size    |  p  |  q  |       Avg CPU time     |   Avg Outer   |   Avg Inner
----------|-----|-----|------------------------|---------------|-------------
 100x100     1.0   1.1       4.0775980e-03             8             66
 100x100     1.0   2.0       3.3921420e-03             8             68
 100x100     1.0   2.3       4.6125026e-03             8             82
 100x100     0.5   1.1       1.1335552e-02             8            160
 100x100     0.5   2.0       8.2885870e-03             8            132
 100x100     0.5   2.3       9.3813821e-03             8            128
 100x100     0.0   1.1       2.2376523e-02             8            340
 100x100     0.0   2.0       1.6947724e-02             8            273
 100x100     0.0   2.3       1.9629687e-02             8            264
***************************************************************************
          1000 Test Trials \ theta = 0.95 and tau = 1.00
                       For psi1(v)
-------------------------------------------------------------------------
  Size    |   Avg CPU time   |   Avg Outer   |   Avg Inner
----------|------------------|---------------|-------------
 100x100       3.3835071e-03           8           67
-------------------------------------------------------------------------
                       For psi10(v)
-------------------------------------------------------------------------
  Size    |  p  |  q  |       Avg CPU time     |   Avg Outer   |   Avg Inner
----------|-----|-----|------------------------|---------------|-------------
 100x100     1.0   1.1       4.0985234e-03             8             67
 100x100     1.0   2.0       3.4764623e-03             8             70
 100x100     1.0   2.3       4.8655830e-03             8             87
 100x100     0.5   1.1       1.1522834e-02             8            163
 100x100     0.5   2.0       8.2047270e-03             8            132
 100x100     0.5   2.3       9.2830872e-03             8            128
```

100x100	0.0	1.1	2.2100840e-02	8	342
100x100	0.0	2.0	1.5401373e-02	8	260
100x100	0.0	2.3	1.8439248e-02	8	257

****

1000 Test Trials \ theta = 0.95 and tau = 0.50

For psi1(v)

Size	Avg CPU time	Avg Outer	Avg Inner
100x100	3.5095046e-03	8	70

For psi10(v)

Size	p	q	Avg CPU time	Avg Outer	Avg Inner
100x100	1.0	1.1	4.2749197e-03	8	69
100x100	1.0	2.0	3.9661317e-03	8	84
100x100	1.0	2.3	5.7096255e-03	8	106
100x100	0.5	1.1	1.2077167e-02	8	168
100x100	0.5	2.0	8.1892895e-03	8	133
100x100	0.5	2.3	9.2105366e-03	8	130
100x100	0.0	1.1	2.2410064e-02	8	348
100x100	0.0	2.0	1.4352666e-02	8	254
100x100	0.0	2.3	1.6830090e-02	8	247

****

1000 Test Trials \ theta = 0.70 and tau = 1.50

For psi1(v)

Size	Avg CPU time	Avg Outer	Avg Inner
100x100	5.2362389e-03	17	193

For psi10(v)

Size	p	q	Avg CPU time	Avg Outer	Avg Inner
100x100	1.0	1.1	6.4808110e-03	17	193
100x100	1.0	2.0	5.2432047e-03	17	186
100x100	1.0	2.3	6.3071416e-03	17	185
100x100	0.5	1.1	1.3155406e-02	17	363
100x100	0.5	2.0	8.9075779e-03	17	272
100x100	0.5	2.3	1.0404918e-02	17	272
100x100	0.0	1.1	2.0037951e-02	17	621
100x100	0.0	2.0	1.1483201e-02	17	408
100x100	0.0	2.3	1.2133882e-02	17	372

****

1000 Test Trials \ theta = 0.70 and tau = 1.00

For psi1(v)

Size	Avg CPU time	Avg Outer	Avg Inner
100x100	5.1536775e-03	17	204

For psi10(v)

Size	p	q	Avg CPU time	Avg Outer	Avg Inner
100x100	1.0	1.1	6.3529403e-03	17	204
100x100	1.0	2.0	5.2127181e-03	17	199
100x100	1.0	2.3	6.2482600e-03	17	196
100x100	0.5	1.1	1.4259951e-02	17	408
100x100	0.5	2.0	8.5343369e-03	17	272
100x100	0.5	2.3	9.9658893e-03	17	272
100x100	0.0	1.1	2.0764084e-02	17	664
100x100	0.0	2.0	1.1004421e-02	17	408
100x100	0.0	2.3	1.2354002e-02	17	394

****

1000 Test Trials \ theta = 0.70 and tau = 0.50

For psi1(v)

Size	Avg CPU time	Avg Outer	Avg Inner
100x100	5.4036479e-03	17	218

For psi10(v)

Size	p	q	Avg CPU time	Avg Outer	Avg Inner
100x100	1.0	1.1	6.6931856e-03	17	217
100x100	1.0	2.0	5.4881685e-03	17	213
100x100	1.0	2.3	7.5225191e-03	17	269
100x100	0.5	1.1	1.4308512e-02	17	408
100x100	0.5	2.0	8.5710937e-03	17	272
100x100	0.5	2.3	1.0003543e-02	17	272
100x100	0.0	1.1	2.1116726e-02	17	680
100x100	0.0	2.0	1.1037940e-02	17	408
100x100	0.0	2.3	1.3134756e-02	17	408

****

1000 Test Trials \ theta = 0.45 and tau = 1.50

For psi1(v)

Size	Avg CPU time	Avg Outer	Avg Inner
100x100	7.6501109e-03	32	496

For psi10(v)

Size	p	q	Avg CPU time	Avg Outer	Avg Inner
100x100	1.0	1.1	9.5997407e-03	32	496
100x100	1.0	2.0	7.9044264e-03	32	496
100x100	1.0	2.3	9.6376067e-03	32	496
100x100	0.5	1.1	1.5504239e-02	32	786
100x100	0.5	2.0	9.2780011e-03	32	496
100x100	0.5	2.3	1.1081671e-02	32	498
100x100	0.0	1.1	2.1819398e-02	32	1277

100x100	0.0	2.0	1.1883117e-02	32	800
100x100	0.0	2.3	1.3573008e-02	32	736

****

1000 Test Trials \ theta = 0.45 and tau = 1.00

For psi1(v)

Size	Avg CPU time	Avg Outer	Avg Inner
100x100	7.6315791e-03	32	496

For psi10(v)

Size	p	q	Avg CPU time	Avg Outer	Avg Inner
100x100	1.0	1.1	9.5355533e-03	32	496
100x100	1.0	2.0	7.8837769e-03	32	496
100x100	1.0	2.3	9.9525437e-03	32	496
100x100	0.5	1.1	1.9180187e-02	32	992
100x100	0.5	2.0	9.2532009e-03	32	496
100x100	0.5	2.3	1.1087260e-02	32	500
100x100	0.0	1.1	2.4916900e-02	32	1457
100x100	0.0	2.0	1.4627533e-02	32	992
100x100	0.0	2.3	1.3732071e-02	32	751

****

1000 Test Trials \ theta = 0.45 and tau = 0.50

For psi1(v)

Size	Avg CPU time	Avg Outer	Avg Inner
100x100	7.5091278e-03	32	496

For psi10(v)

Size	p	q	Avg CPU time	Avg Outer	Avg Inner
100x100	1.0	1.1	9.3971434e-03	32	496
100x100	1.0	2.0	7.7380353e-03	32	496
100x100	1.0	2.3	9.4625936e-03	32	497
100x100	0.5	1.1	1.8870816e-02	32	992
100x100	0.5	2.0	9.1642285e-03	32	497
100x100	0.5	2.3	1.1079987e-02	32	516
100x100	0.0	1.1	2.4794242e-02	32	1488
100x100	0.0	2.0	1.4412400e-02	32	992
100x100	0.0	2.3	1.6202923e-02	32	964

****

BIBLIOGRAPHY

[1] Bai Y., Ghami M. El, Roos C., *A Comparative Study of Kernel Functions for Primal-Dual Interior-Point Algorithms in Linear Optimization*, SIAM Journal on Optimization, Vol. 15, No. 1, (2004).

[2] Bai Y., Lesaja G., Roos C., Wang G., Ghami M. El, *A Class of Large and Small Update Primal-Dual Interior-Point Algorithms for Linear Optimization*, Journal of Optimization Theory and Applications, Vol. 138, No. 3, pp. 341-359, (2008).

[3] Cottle R., Pang J., Stone R., *The Linear Complementarity Problem*, Academic Press, Inc., Boston, (1992).

[4] Facchinei F., Pang J.S., *Finite-Dimensional Variational Inequalities and Complementarity Problems*, Springer, New York, (2003)

[5] Ghami M. El, *New Primal-dual Interior-point Methods Based on Kernel Functions*, PhD dissertation, Delft University of Technology, Netherlands, (2005).

[6] Karmarkar N., *A New Polynomial Time Algorithm for Linear Programming*, Combinatorica, Vol 4, No. 4, (1984).

[7] Khachiyan L.G., *A Polynomial Algorithm in Linear Programming*, Soviet Mathematics Doklady, 20, pp. 373-395 (1984).

[8] Klee V. and Minty G.J., *How Good is the Simplex Algorithm?* Inequalities, III, pp. 159-175, Academic Press, New York, NY, (1972).

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

[10] Lemke C.E., *Bimatrix Equilibrium Points and Mathematical Programming*, Management Science II, pp. 681-689, (1965).

[11] Lesaja G., *Introducing Interior-Point Methods for Introductory Operations Research Courses and/or Linear Programming Courses*, Open Operational Research Journal, Vol. 3, pp. 1-12, (2009).

[12] Nesterov Y., Nemirovski A., *Interior-Point Polynomial Algorithms in Convex Programming*, SIAM Studies in Applied Mathematics, Philadelphia, PA, (1994).

[13] Megiddo N., *Pathways to the Optimal Set in Linear Programming*, Progress in Mathematical Programming: Interior Point and Related Methods, pp. 131-158, Springer, New York, (1989).

[14] Peng J.S., Roos C., Terlak T., *Self-Regularity: A New Paradigm for Primal-Dual Interior-Point Algorithms*, Princeton University Press, (2002)

[15] Roos C., Terlaky T., Vial J.P., *Theory and Algorithms for Linear Optimization*, John Wiley and Sons, Chichester, UK (1997).

[16] Shor N.Z., *Cut-off Method with Space Extension in Convex Programming Problems*, Cybernetics 13, pp. 94-96, (1977).

[17] Sonnevend G., *An "analytic center" for polyhedrons and new classes of global algorithms for linear (smooth, convex) programming*, System Modeling and Optimization. Proceedings of the 12th IFIP-Conference, Budapest, Hungary, September 1985. Lecture Notes in Control and Information Sciences, vol. 84, pp. 866-876. Springer, Berlin (1986).

[18] Wright S., *Primal-Dual Interior-Point Methods*, SIAM Publishing, Philadelphia, PA (1997).

[19] Nemirovski A., Yudin D.B., *Informational Complexity and Effective Methods of Solution for Convex Extremal Problems*, Ekonomika i Matematicheshie Metody 12 (in Russian), pp. 357-369, (1976).