Interior Point Methods for Large Scale Portfolio Optimization

Diplomarbeit

vorgelegt von

Stefan Klebor

aus Lüdinghausen

angefertigt am
Institut für Numerische und Angewandte Mathematik
der Georg-August-Universität Göttingen
1994

This thesis is dedicated to my teachers Michael Rasche who sparked my interest in mathematics and Prof. Dr. P.E. Gill who sparked my interest in interior point methods.

It is also dedicated to the wonderful institution INTERNET. I feel that without it this thesis would not have been possible.

Acknowledgements

My first and special thanks goes to Prof. Dr. J. Werner for the freedom he gave me in selecting the topic, for the supervision of my work, for a lot of useful comments and fruitful discussions and last but not least for his intervention in getting the permission to write this thesis in english.

I am also most grateful for the invaluable support of The MathWorks, Inc. and Bausch-Gall GmbH and here in particular Cleve B. Moler, Sandra Heaney and Adriana Ionescu. Their free provision of the powerful tool MATLAB made the implementations and the numerical experiments possible. Thanks to Gerhard Siebrasse for installing the UNIX version of MATLAB and to Peter Beisel for granting me access to MATLAB at the University of Wuppertal.

In this context I want to extend a special thanks to Messrs. Ken-ichi Suzuki and Hiroshi Konno for the provision of their test data from the Tokyo Stock Exchange Market.

Thanks also to Tamra J. Carpenter, Masakazu Kojima, Kenneth O. Kortanek, Irvin J. Lustig, David F. Shanno, Richard A. Tapia, Robert J. Vanderbei, Yin Zhang and Eberhard Kranich for useful hints and for providing me with interesting preprints of papers.

I would like to particularly thank my parents for all the support they have given me throughout my studies and before. I know that they helped me more than I would be able to imagine.

It is impossible to thank my beloved fiancée Lara enough for everything she has given me. I hope I will have a lifetime at her side to show her my gratitude.

Contents

1. Introduction

In 1959 Harry M. Markowitz laid the foundation for modern portfolio[1] theory with the introduction of his mean-variance (MV) model [16]. From a mathematical point of view it is nothing more than a convex quadratic program (QP) with a matrix in the objective function that is usually completely dense. This circumstance often makes it difficult to solve large MV models. In fact it is difficult even to generate the model as this involves the estimation of a great number of parameters.

As a result the MV model was not used very much in practice throughout the 1960s and '70s. Instead other –simplified– models that were based on the MV model but much easier to solve, such as CAPM [20] gained increasing popularity. The drawback of course was, that because of the idealistic, simplifying assumptions these models imposed, their results could be used only as a first order approximation. The search for more reliable models produced the multiple factor and the index matching models (cf. [6]) which require considerably more computational work than CAPM but due to the increasing performance of computers that seemed to be acceptable.

The next leap forward in large scale portfolio optimization was again marked by Markowitz together with Perold through the appearance of their paper [17] in 1981, in which they used a scenario model to generate a sparse representation of an MV model, i.e. with a sparse matrix in the objective function of the QP. This together with sparse matrix techniques made it possible for the first time to solve a large scale MV model in an efficient amount of time.

In Chapter 2 of this thesis we will show how to transform an MV model, i.e. a QP with a dense matrix, into a separable QP, i.e. a QP with an ultimately sparse diagonal matrix. We will demonstrate that this sparse representation can be solved very efficiently. At the same time we will show how one can alleviate the problem of model generation by using readily available historical data. This has the nice side effect of producing portfolios with a manageable number of assets.

Chapter 3 contains a brief introduction to a fairly new approach for solving QPs, namely interior point methods (IPMs), which became popular in 1984 through an often cited paper by Karmarkar [10]. IPMs have been proven to be often superior to other

[1] The word *portfolio* has its origin in the french word *portefeuille* which means wallet or briefcase.

QP solvers especially for large problems [2] and they are particularly well suited for making use of sparsity. We will present a model algorithm for which the duality gap superlinearly converges to zero under certain assumptions.

Chapter 4 provides a more specific primal-dual algorithm for which the duality gap is globally and superlinearly convergent. These theoretical convergence results require quite some technicalities.

In Chapter 5 we will show that these technicalities are not entirely necessary in order to get a practically efficient algorithm. The implementations of two established practical IPMs are described in detail, together with some techniques that make IPMs work well in practice. The very promising results of our numerical experiments with test data from the Tokyo Stock Exchange Market indicate that there is a big potential in this approach to large scale portfolio optimization.

2. Portfolio Optimization

2.1 The Mean-Variance Model

Consider an investor with a given budget who can choose among n risky asset types S_i, $i = 1, \dots, n$ to invest her money. Let x_i be the rate of money to be invested in S_i out of the total budget (normalized to 1) and R_i a random variable representing the rate of return of S_i (per period). Further we define $\tilde{r}_i := E[R_i]$, where $E[\cdot]$ is the expected value and $\tilde{v}_{ij} := \text{cov}[R_i, R_j]$. Then the total expected return of a portfolio x is given by $\tilde{r}^T x$ and the total variance, which is being used as the measure of risk is given by $x^T \tilde{V} x$.

In its original form [16] the MV model intends to find all *efficient* portfolios $\bar{x}$, i.e. all portfolios for which

(i) $\bar{x} \in M := \{x \in \mathbb{R}^n : e^T x = 1,\ x \geq 0\}$, where $e^T := (1, \dots, 1)$

(ii) $\forall x \in M$ with $\tilde{r}^T x > \tilde{r}^T \bar{x}$ it holds that $x^T \tilde{V} x > \bar{x}^T \tilde{V} x$

(iii) $\forall x \in M$ with $x^T \tilde{V} x < \bar{x}^T \tilde{V} x$ it holds that $\tilde{r}^T x < \tilde{r}^T \bar{x}$.

The efficient portfolios can also be obtained by solving the following convex parametric optimization problem

$$(\text{EF}) \qquad \begin{array}{ll} \text{Minimize} & x^T \tilde{V} x - \tau\, \tilde{r}^T x \\[1ex] \text{subject to} & e^T x = 1 \\ & x \geq 0 \\[1ex] \forall \tau \in [\,0, \infty). & \end{array}$$

The convexity is due to the fact that $\tilde{V}$ is a covariance matrix and as such the expected value of a quadratic.

In most situations it is not necessary to compute *all* efficient portfolios but it suffices to compute the portfolio with minimum risk for a given required total rate of return α (cf. [11]) which leads to the following model

$$\text{(MV1)} \qquad \begin{array}{ll} \text{Minimize} & x^T\tilde{V}x \\ \\ \text{subject to} & \tilde{r}^T x \geq \alpha \\ & e^T x = 1 \\ & x \geq 0 \; . \end{array}$$

Sometimes [21] the first constraint appears as an equality rather than an inequality which usually does not make much difference because a higher return normally also forces a higher risk. We chose the above formulation because it is more flexible and does not require substantially more work to solve.

Practical applications frequently require additional constraints, like linear institutional constraints or transaction costs. Since their only effect from a theoretical point of view is the enlargement of the model we will not include them here. In parts they will be covered together with the implementation issues in chapter 5.

2.2 A Separable Representation

As was mentioned before, there are two major problems associated with model (MV1) especially for large n. The first is to obtain the required data $\tilde{r}$ and $\tilde{V}$ and the second is the solution itself since the covariance-matrix $\tilde{V}$ will usually be completely dense. This means that even sophisticated hard- and software will take quite a long time to solve the model if it involves several thousand assets.

Luckily a solution to the first problem almost automatically brings one for the second problem as well [11]. Consider a discrete payoff-matrix $P \in \mathbb{R}^{k\times n}$ that contains k independent samples of the random variables R_i, $i = 1, \ldots, n$, e.g. the realized returns of the most recent k periods over the n assets. This data will in most cases be readily available. Then we get unbiased estimators [12] r and V for $\tilde{r}$ and $\tilde{V}$, respectively as

$$r := \frac{1}{k}P^T e \tag{2.1}$$

and

$$\begin{aligned} v_{ij} &:= \frac{1}{k-1}\sum_{l=1}^{k}(p_{li} - r_i)(p_{lj} - r_j) \\ \Leftrightarrow \quad V &= \frac{1}{k-1}P^T(I - \frac{1}{k}ee^T)P \, , \end{aligned} \tag{2.2}$$

where I is the $k \times k$ identity matrix. Substituting $\tilde{r}$ and $\tilde{V}$ by the above estimators in (MV1) yields

$$\text{(MV2)} \qquad \begin{array}{ll} \text{Minimize} & \frac{1}{k-1} x^T P^T (I - \frac{1}{k} ee^T) P x \\ & \\ \text{subject to} & \frac{1}{k} e^T P x \geq \alpha \\ & e^T x = 1 \\ & x \geq 0 \end{array}$$

which is equivalent to the separable program

$$\text{(MV3)} \qquad \begin{array}{ll} \text{Minimize} & x_f^T x_f \\ & \\ \text{subject to} & (I - \frac{1}{k} ee^T) P x_p - x_f = 0 \\ & \frac{1}{k} e^T P x_p \geq \alpha \\ & e^T x_p = 1 \\ & x_p \geq 0, \ x_f \text{ free.} \end{array}$$

For the case when then the first constraint in model (MV2) is an equality constraint we get an even simpler separable representation [21]

$$\text{(MV3}'\text{)} \qquad \begin{array}{ll} \text{Minimize} & x_f^T x_f \\ & \\ \text{subject to} & P x_p - x_f = \alpha e \\ & e^T x_f = 0 \\ & e^T x_p = 1 \\ & x_p \geq 0, \ x_f \text{ free.} \end{array}$$

So what did we gain from all these transformations? Let us compare models (MV1) and (MV3). Obviously model (MV3) contains k more design variables $[x_f]_1, \ldots, [x_f]_k$ and k more constraints. (Note that here and later we will sometimes –where necessary to avoid double subscripts– denote the components of a vector x by $[x]_i$.) On the other hand the $(k+n) \times (k+n)$ matrix in the objective functions is much more sparse than $\tilde{V}$ with only k non-zeros that lie on the main diagonal. The first hint that this is a good bargain gives us the fact that k is always much smaller than n. Particularly in chapter 5 we will see that the sparsity of Q saves a lot more computational time than has to be sacrificed for the additional variables and constraints.

A nice side-effect of model (MV3) is that it has a solution such that at most $k+2$ components of x_p^* are non-zero. This is very relevant in practice because it is desirable to have a portfolio with a manageable number of assets [11]. It is straightforward to prove this property: Let $(x_p^*, x_f^*) \in \mathbb{R}^{n+k}$ be an optimal solution of (MV3). Now consider the linear system

$$\begin{pmatrix} (I - \frac{1}{k}ee^T)Px_p = x_f^* \\ \frac{1}{k}e^T Px_p \geq \alpha \\ e^T x_p = 1 \\ x_p \geq 0 \end{pmatrix} \tag{2.3}$$

This system has a feasible solution x_p^* and hence also a basic feasible solution $\hat{x}_p$ [24] with at most $k+2$ non-zero components because the rank of the system is less than or equal to $k+2$. Obviously $(\hat{x}_p, x_f^*)$ is also an optimal solution of (MV3).

Evidently model (MV3) can be transformed into a standard form convex QP using slack variables and variable splitting. We will not use this transformation in practice because it leads to a considerable enlargement of the model but it is important for establishing the convergence results in the following chapter.

3. Introduction to Interior Point Methods

3.1 Logarithmic Barrier Transformations

Consider the standard form QP [2]

$$
\text{(P)}\qquad \begin{array}{ll} \text{Minimize} & \frac{1}{2}x^TQx + c^Tx \\[1em] \text{subject to} & Ax = b \\ & x \geq 0 \end{array}
$$

where $x \in \mathbb{R}^n$, $Q \in \mathbb{R}^{n\times n}$ and $A \in \mathbb{R}^{m\times n}$, $m < n$. As usual for IPMs we assume that

$$\operatorname{rank}(A) = m \tag{3.1}$$

and

$$\dot{M}_P := \{x : Ax = b,\ x > 0\} \neq \emptyset\,. \tag{3.2}$$

Additionally we assume that Q is at least positive semi-definite hence making (P) a convex problem.

For an obvious reason all $x \in \dot{M}_P$ are called *interior* or *strictly feasible* points for (P). As we will see later, we can obtain far more effective algorithms by working on primal and dual problems simultaneously. Hence we will now also consider the dual of problem (P)

$$
\text{(D)}\qquad \begin{array}{ll} \text{Maximize} & -\frac{1}{2}x^TQx + b^T\lambda \\[1em] \text{subject to} & A^T\lambda + y - Qx = c \\ & y \geq 0 \end{array}
$$

where just like for the primal it is assumed that

$$\dot{M}_D := \{(x, y, \lambda) : A^T\lambda + y - Qx = c,\ y > 0\} \neq \emptyset\,.$$

Let us quickly review the first order necessary and sufficient conditions for simultaneous optimality of (P) and (D):

$$\begin{pmatrix} Ax - b \\ A^T\lambda + y - Qx - c \\ XYe \end{pmatrix} = 0, \quad (x, y) \geq 0 \tag{3.3}$$

where here and subsequently we will use the convention that $X := diag(x)$ and similarly for Y and other capitalized vector names. In order to simplify notation and *only* for that reason we will eliminate the dual variable λ from the above system. To do this, let $B \in \mathbb{R}^{(n-m)\times n}$ be any matrix such that the columns of B^T form a basis for the null space of A. Pre-multiplying the second equation of (3.3) by the nonsingular matrix $\begin{bmatrix} A^T & B^T \end{bmatrix}^T$ and remembering that $BA^T = 0$ yields

$$0 = \begin{bmatrix} A \\ B \end{bmatrix} (A^T\lambda + y - Qx - c) = \begin{pmatrix} AA^T\lambda - A(-y + Qx + c) \\ By - BQx - Bc \end{pmatrix}.$$

Since AA^T is nonsingular, λ is uniquely determined once x and y are known. Hence we can remove the first equation of the above system to arrive at the following optimality conditions

$$F(x, y) := \begin{pmatrix} Ax - b \\ By - BQx - Bc \\ XYe \end{pmatrix} = 0, \quad (x, y) \geq 0\,. \tag{3.4}$$

Accordingly we define the primal-dual strict feasibility set as

$$\dot{M} := \{(x, y) : Ax = b, By - BQx = Bc,\ (x, y) > 0\}\,.$$

Our goal now is to develop an algorithm that

(i) successively finds solutions for both (P) and (D),

(ii) operates on $\dot{M}$, i.e. we start with strictly feasible points for (P) and (D) and all iterates remain strictly feasible (this property is the reason why the algorithm is called *interior point method*),

(iii) at least partially inherits the excellent local convergence properties of Newton's method.

The vehicle that we use to achieve all three goals are so called *logarithmic barrier transformations* [2]. Barrier methods evolved as a means for solving mathematical programs subject to inequality constraints. Acting from a strictly feasible point, barrier methods seek to minimize an unconstrained function formed from the original objective plus *barrier terms* which prevent crossing a boundary. These barriers are imposed by functions which are smooth throughout the feasible region but become indefinite at boundaries. The natural logarithm is a good choice to serve as barrier.

In our case we have both inequality *and* equality constraints so we have to work a little harder but the principle remains the same so that we get the following barrier transformed problems ($\bar{\text{P}}$) and ($\bar{\text{D}}$) for (P) and (D), respectively:

$$(\bar{\text{P}}) \qquad \begin{array}{ll} \text{Minimize} & \frac{1}{2}x^TQx + c^Tx - \mu\sum_{i=1}^{n}\ln x_i \\ & \\ \text{subject to} & Ax = b\,, \quad x > 0\,. \end{array}$$

$$(\bar{\text{D}}) \qquad \begin{array}{ll} \text{Maximize} & -\frac{1}{2}x^TQx + b^T\lambda + \mu\sum_{i=1}^{n}\ln y_i \\ & \\ \text{subject to} & A^T\lambda + y - Qx = c\,, \quad y > 0\,. \end{array}$$

The Lagrangian functions associated with ($\bar{\text{P}}$) and ($\bar{\text{D}}$) are

$$\frac{1}{2}x^TQx + c^Tx - \mu\sum_{i=1}^{n}\ln x_i - \lambda^T(Ax - b) \tag{3.5}$$

and

$$-\frac{1}{2}x^TQx + b^T\lambda + \mu\sum_{i=1}^{n}\ln y_i - x^T(A^T\lambda + y - Qx - c)\,, \tag{3.6}$$

respectively. Let us look at the first order optimality conditions again. For (3.5) they are

$$\begin{pmatrix} Ax - b \\ A^T\lambda + \mu X^{-1}e - Qx - c \end{pmatrix} = 0 \tag{3.7}$$

and similarly for (3.6)

$$\begin{aligned} \begin{pmatrix} Ax - b \\ -Qx - A^T\lambda - y + 2Qx + c \\ \mu Y^{-1}e - Xe \end{pmatrix} &= 0 \\ \Leftrightarrow \quad \begin{pmatrix} Ax - b \\ A^T\lambda + y - Qx - c \\ XYe - \mu e \end{pmatrix} &= 0\,. \end{aligned} \tag{3.8}$$

Together (3.7) and (3.8) provide necessary and sufficient conditions for simultaneous optimality in (3.5) and (3.6). However we can get an even more concise description by eliminating redundant equations. Obviously the first equations in (3.7) and (3.8) are the same. Furthermore, if we pre-multiply the third equation of (3.8) by X^{-1} and plug the result into the second equation of (3.7) this equation is equal to the second equation of (3.8). Therefore, the equations characterizing simultaneous optimality in (3.5) and (3.6) for a fixed $\mu \geq 0$ are

$$\begin{pmatrix} Ax - b \\ A^T\lambda + y - Qx - c \\ XYe - \mu e \end{pmatrix} = 0 \tag{3.9}$$

and the nonnegativity requirements $x, y \geq 0$.

Clearly (3.9) is almost identical to (3.3), the only difference being the μe term on the right hand side of the third equation. Hence if we choose $\mu = 0$ we get exactly (3.3). This already gives us a first hint that asymptotically we will have to reduce μ to zero in our algorithm to get optimal solutions for (P) and (D). Also (3.9) is exactly the same as (3.8) so we see that working on primal and dual problem simultaneously does not require more work than working on just one of them.

Analogous to (3.3) we can transform (3.9) to eliminate λ which renders

$$F_\mu(x, y) := \begin{pmatrix} Ax - b \\ By - BQx - Bc \\ XYe - \mu e \end{pmatrix} = 0\,. \tag{3.10}$$

It is well known (cf. e.g. [25]) that (3.10) has a unique solution (x_μ, y_μ) for every positive μ and that the so called *central path*,

$$S_{\text{cen}} := \{(x_\mu, y_\mu) : \mu > 0\} \tag{3.11}$$

forms a continuous curve which converges to a solution of (3.4), i.e. of (P) and (D) as μ tends to zero. IPMs which iterates stay in a certain neighborhood

$$\mathcal{N}(\gamma) := \{(x, y) \in \dot{M} : x_i y_i \geq \gamma\,(x^T y/n),\ i = 1, \ldots, n\} \tag{3.12}$$

of the central path for some $\gamma \in (0, 1]$ are called *path-following* methods. We will see that this property is essential in our superlinear convergence analysis.

3.2 A Model Algorithm

The next step is to apply Newton's method to (3.10). More precisely let us formulate a coarse model algorithm [26].

Algorithm 1

Given a strictly feasible pair (x_0, y_0). For $k = 0, 1, \ldots$, do

Step 1 Optimality Test: If (x_k, y_k) satisfies the optimality criteria $\rightarrow$ STOP

Step 2 Choose $\sigma_k \in [\,0,1)$ and set $\mu_k := \sigma_k \frac{x_k^T y_k}{n}$

Step 3 Compute the *Newton directions*

$$\begin{pmatrix} \Delta x_k \\ \Delta y_k \end{pmatrix} = -\,\left[F'_{\mu_k}(x_k, y_k)\right]^{-1} F_{\mu_k}(x_k, y_k)\ ,$$

where $F'_{\mu_k} = F'$ is the Jacobian of F.

Step 4 Choose $\tau_k \in (0,1)$ and compute the steplength

$$\alpha_k := \frac{-\tau_k}{\min\left(X_k^{-1}\Delta x_k, Y_k^{-1}\Delta y_k, -\tau_k\right)}\,,$$

where $\min(\cdot)$ refers to the smallest component of all vectors in the parentheses.

Step 5 Compute the new iterates

$$\begin{pmatrix} x_{k+1} \\ y_{k+1} \end{pmatrix} := \begin{pmatrix} x_k \\ y_k \end{pmatrix} + \alpha_k \begin{pmatrix} \Delta x_k \\ \Delta y_k \end{pmatrix}\,,$$

set $k := k+1$ and go to **Step 1**

The first thing one should notice about Algorithm 1 is that there are two important parameters which control its behaviour. One is σ_k which determines the centering parameter μ_k. Since only σ_k is directly under our control we will from now on call σ_k the centering parameter. As we mentioned before we ultimately have to 'phase out' μ_k to get solutions for our original problems (P) and (D). Since we will show below that all iterates produced by Algorithm 1 are strictly feasible, the term $(x_k^T y_k)/n$ in the definition of μ_k provides a proximity measure towards optimality (cf. (3.4)) hence making the extent of centering dependent upon the closeness to the solution which sounds very reasonable. The parameter α_k controls the steplength making the algorithm a *damped* variant of Newton's method. Its definition guarantees that all components of (x_{k+1}, y_{k+1}) will remain strictly positive.

We will now establish that Algorithm 1 is well defined.

Proposition 3.1 *Let $F_\mu(x,y)$ be defined as in (3.10) then it holds that*

1. *$F'_\mu(x,y) = F'(x,y)$ is nonsingular for $(x,y) > 0$.*
2. *If $(x_0, y_0) \in \dot{M}$ then all iterates produced by Algorithm 1 are strictly feasible, i.e. $(x_k, y_k) \in \dot{M}$, $\forall\, k \geq 0$.*

Proof:

Ad 1. From (3.4) it holds that

$$F'(x,y) = \begin{bmatrix} A & 0 \\ -BQ & B \\ Y & X \end{bmatrix}.$$

So we have to show that

$$\begin{bmatrix} A & 0 \\ -BQ & B \\ Y & X \end{bmatrix} \begin{pmatrix} u \\ v \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \\ 0 \end{pmatrix} \Rightarrow \begin{pmatrix} u \\ v \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \end{pmatrix}. \tag{3.13}$$

Remembering that the columns of B^T form a basis for the null space of A, the first equation of (3.13) indicates that $u = B^T w$ for some w. Using this result and solving the third equation for v gives us $v = -X^{-1}YB^Tw$. Plugging the equations for u and v into the second equation yields

$$-BQB^Tw - BX^{-1}YB^Tw = 0\,.$$

Premultiplying this by w^T, we get

$$-w^TBQB^Tw - w^TBX^{-1}YB^Tw = 0\,.$$

Now BQB^T is positive semi-definite and $BX^{-1}YB^T$ is positive definite and therefore the left hand side is zero if and only if $w = 0$. From the above equations for u and v it now follows that $(u^T, v^T) = (0, 0)$.

Ad 2. Given (x_k, y_k) is feasible, the feasibility of (x_{k+1}, y_{k+1}) is automatically guaranteed by the first two defining equations of $(\Delta x_k^T, \Delta y_k^T)$ (cf. Step 3 of Algorithm 1)

$$\begin{pmatrix} A\Delta x_k \\ -BQ\Delta x_k + B\Delta y_k \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \end{pmatrix}.$$

As mentioned before the strict positivity of (x_{k+1}, y_{k+1}) is achieved through the choice of α_k. To demonstrate that α_k suits this purpose we define

$$\hat{\alpha}_k := -\frac{1}{\min\left(X_k^{-1}\Delta x_k, Y_k^{-1}\Delta y_k\right)}\,. \tag{3.14}$$

In Remark 3.2 below we will show that $\min(X_k^{-1}\Delta x_k, Y_k^{-1}\Delta y_k) < 0$ for all k and thus it obviously holds that $\alpha_k < \hat{\alpha}_k$. In fact $\hat{\alpha}_k$ is the steplength for which exactly one component of (x_{k+1}, y_{k+1}) becomes zero. To prove this we have to show that

$$\hat{\alpha}_k = \min\left\{ \min_i \left\{ -\frac{[x_k]_i}{[\Delta x_k]_i} \ : [\Delta x_k]_i < 0 \right\},\ \min_j \left\{ -\frac{[y_k]_j}{[\Delta y_k]_j} \ : [\Delta y_k]_j < 0 \right\} \right\}.$$

Equivalently we can write this as

$$\hat{\alpha}_k = \min\left\{ -\max_i \left\{ \frac{[x_k]_i}{[\Delta x_k]_i} \; : [\Delta x_k]_i < 0 \right\}, \; -\max_j \left\{ \frac{[y_k]_j}{[\Delta y_k]_j} \; : [\Delta y_k]_j < 0 \right\} \right\} .$$

Taking reciprocals, we can drop the limitations $[\Delta x_k]_i \, , [\Delta y_k]_j < 0$ because the positive components of the search directions are not relevant when taking the minimum:

$$\hat{\alpha}_k = \min\left\{ -\left(\min_i \left\{ \frac{[\Delta x_k]_i}{[x_k]_i} \right\} \right)^{-1}, \; -\left(\min_j \left\{ \frac{[\Delta y_k]_j}{[y_k]_j} \right\} \right)^{-1} \right\} .$$

But this is (3.14) just stated differently.

□

Remark 3.2 *It holds that $\hat{\alpha}_k > 0, \; \forall \, k \geq 0$.*

Proof: It suffices to show that $\min(\Delta x_k, \Delta y_k) < 0$. From the third defining equation of $(\Delta x_k^T, \Delta y_k^T)$ (cf. Step 3 of Algorithm 1) it follows that

$$Y_k \Delta x_k + X_k \Delta y_k = -X_k Y_k e + \sigma_k \frac{x_k^T y_k}{n} \, e \, . \tag{3.15}$$

Remembering that $\sigma_k \in [0, 1)$ we get

$$0 \leq \sigma_k \frac{x_k^T y_k}{n} < \max(X_k Y_k e)$$

so that at least one component of $Y_k \Delta x_k + X_k \Delta y_k$ has to be negative. This in turn implies that Δx_k or Δy_k must have at least one negative component. □

3.3 Superlinear Convergence

The reason for the great publicity of Karmarkar's method [10] was that it was the first method for solving a linear program with a polynomial complexity bound and reportedly 50 times faster solution times than the simplex method at least for some large problems. As a result a lot of the subsequent research on IPMs focused on complexity issues. It soon became evident though that the IPMs with good theoretical complexity bounds were not the ones that performed well in practice. This is true for linear as well as nonlinear problems. Zhang and Tapia were among the first who shed light on the aspect of fast local convergence which has always been an important issue in continuous optimization [27, 28]. It seems that algorithms with a good local convergence perform better in practice [14]. In Chapter 4 we will present an algorithm which blends the often conflicting objectives of global and fast local convergence.

For now we will analyze the local convergence properties of Algorithm 1. Most of the results will be useful for the convergence analysis in the next chapter as well. To simplify things here and particularly in the next chapter we adopt the following notation:

$$\begin{aligned}
x_k(\alpha) &:= x_k + \alpha \Delta x_k\,, && (3.16)\\
y_k(\alpha) &:= y_k + \alpha \Delta y_k\,, && (3.17)\\
f_k(\alpha) &:= X_k(\alpha) Y_k(\alpha) e\,, && (3.18)\\
f_k^{\min}(\alpha) &:= \min\left(f_k(\alpha)\right)\,, && (3.19)\\
f_k^{\max}(\alpha) &:= \max\left(f_k(\alpha)\right)\,, && (3.20)\\
f_k^{\text{ave}}(\alpha) &:= \left(x_k(\alpha)^T y_k(\alpha)\right)/n\,. && (3.21)
\end{aligned}$$

Whenever $\alpha = 0$, we will drop the argument, e.g. $x_k \equiv x_k(0)$.

Our goal in this section is to prove the following theorem [26] concerning the superlinear convergence of Algorithm 1. Comparing it to similar results for Newton's method [24] it should be noted that it does not require the nonsingularity of $F'(x_*, y_*)$.

Theorem 3.3 *Let $\{(x_k, y_k)\}$ be generated by Algorithm 1 and $(x_k, y_k) \to (x_*, y_*)$, where (x_*, y_*) is a solution of (3.4). If*

(i) *strict complementarity holds at (x_*, y_*), i.e.*

$$[x_*]_i = 0 \Rightarrow [y_*]_i > 0\,, \quad [x_*]_i > 0 \Rightarrow [y_*]_i = 0\,,$$

(ii) *the sequence $\left\{f_k^{\text{ave}}/f_k^{\min}\right\}$ is bounded,*

(iii) *$\tau_k \to 1$ and $\sigma_k \to 0$,*

then the sequence $\{F(x_k, y_k)\}$ componentwise converges to zero Q-superlinearly.

Remark 3.4 *Componentwise Q-superlinear convergence of a sequence implies its superlinear convergence.*

Proof: Let $z_k \to z_*$, then componentwise Q-superlinear convergence means that

$$\lim_{k\to\infty} \frac{|\,[z_{k+1}]_i - [z_*]_i\,|}{|\,[z_k]_i - [z_*]_i\,|} = 0 \quad \text{for } i = 1, \dots, n\,.$$

We have to show that this implies

$$\lim_{k\to\infty} \frac{\|\, z_{k+1} - z_*\|}{\|\, z_k - z_*\|} = 0$$

for some norm. If we choose the ∞-norm we get

$$0 \leq \frac{\| z_{k+1} - z_* \|_\infty}{\| z_k - z_* \|_\infty} = \frac{\max\limits_{i=1,\ldots,n} | [z_{k+1}]_i - [z_*]_i |}{\max\limits_{i=1,\ldots,n} | [z_k]_i - [z_*]_i |} .$$

Let $i(k+1)$ be the index for which the maximum is achieved in the numerator, then the right hand side of the above inequality chain is equal to

$$\frac{| [z_{k+1}]_{i(k+1)} - [z_*]_{i(k+1)} |}{\max\limits_{i=1,\ldots,n} | [z_k]_i - [z_*]_i |} \leq \frac{| [z_{k+1}]_{i(k+1)} - [z_*]_{i(k+1)} |}{| [z_k]_{i(k+1)} - [z_*]_{i(k+1)} |} \to 0$$

which implies our claim. $\square$

Theorem 3.3 is somewhat unsatisfactory because it places conditions on the quantity $f_k^{\text{ave}}/f_k^{\min}$ which is is not directly under our control. This gives reason to the concern that Theorem 3.3 might be a vicious circle. However in the next chapter we will place additional conditions on τ_k and σ_k which will make assumption (ii) obsolete. Note that assumption (ii) makes sure that the iterates lie in $\mathcal{N}(\gamma)$ (cf. (3.12)) for some suitable γ, thus making Algorithm 1 a path-following method.

Another cause for concern is assumption (i) because unlike for LP, a strictly complementary solution may not exist for convex QP. Unfortunately though the assumption that the QP at least possesses a strictly complementary solution seems to be essential. In their paper on a slightly different IPM for monotone linear complementarity problems (LCP) [1], Anstreicher and Ye give an example of an LCP which does not possess a strictly complementary solution and for which their IPM converges no faster than linear. In our numerical experiments we always observed superlinear convergence (cf. Chapter 5), so in practice this assumption does not seem overly restrictive.

In order to prove Theorem 3.3 we need the following two lemmas.

Lemma 3.5 *Let $(\Delta x_k, \Delta y_k)$ be search directions produced by Algorithm 1. Then it holds that*

$$\Delta x_k^T \Delta y_k \geq 0, \quad \forall\, k \geq 0 .$$

Proof: As we had already established in the proof of Proposition 3.1 we have

$$\begin{pmatrix} A\Delta x_k \\ -BQ\Delta x_k + B\Delta y_k \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \end{pmatrix} .$$

We will show more generally that for all (u, v) for which

(i) $Au = 0$ and

(ii) $-BQu + Bv = 0$

it holds that $u^T v \geq 0$.

Again we remember that the columns of B^T form a basis for the null-space of A. Then (i) implies that $u = B^T w$ for some w. Plugging this in (ii) yields $Bv = BQB^T w$. Premultiplying this by w^T and noting that BQB^T is positive semi-definite, we get

$$0 \leq w^T BQB^T w = w^T Bv = (B^T w)^T v = u^T v \,,$$

which completes the proof. □

Lemma 3.6 *Under the assumptions of Theorem 3.3,*

$$\lim_{k\to\infty} \alpha_k = 1 \,.$$

Proof: Define at each iteration

$$p_k := X_k^{-1}\Delta x_k \quad \text{and} \quad q_k := Y_k^{-1}\Delta y_k \,. \tag{3.22}$$

We will show that for every i either $[p_k]_i \to 0$ and $[q_k]_i \to -1$ or $[p_k]_i \to -1$ and $[q_k]_i \to 0$. Recalling the definition of α_k in Step 4 of Algorithm 1 and our assumption $\tau_k \to 1$ it is easily verified that this implies the lemma.

To do this it is necessary to first establish that α_k is bounded away from zero. It follows from the definition of α_k that it suffices to show that $\{p_k\}$ and $\{q_k\}$ are bounded. Premultiplying (3.15) by $(X_k Y_k)^{-1}$ we get

$$p_k + q_k = -e + \sigma_k T_k e \,, \tag{3.23}$$

where

$$T_k := f_k^{\text{ave}} (X_k Y_k)^{-1} \,. \tag{3.24}$$

Multiplying both sides of (3.23) by $(X_k Y_k)^{-\frac{1}{2}}$ and taking the square of the 2-norm results in

$$\left\|(X_k Y_k)^{\frac{1}{2}} p_k\right\|_2^2 + \left\|(X_k Y_k)^{\frac{1}{2}} q_k\right\|_2^2 + 2\Delta x_k^T \Delta y_k =$$

$$x_k^T y_k \left(1 - 2\sigma_k + \sigma_k^2 f_k^{\text{ave}} \frac{e^T (X_k Y_k)^{-1} e}{n}\right) \,. \tag{3.25}$$

Dividing both sides by f_k^{ave} and using Lemma 3.5 we have

$$\left\|T_k^{-\frac{1}{2}} p_k\right\|_2^2 + \left\|T_k^{-\frac{1}{2}} p_k\right\|_2^2 \leq n \left(1 - 2\sigma_k + \sigma_k^2 \frac{e^T T_k e}{n}\right) \,. \tag{3.26}$$

Since T_k is closely related to $\left\{f_k^{\text{ave}}/f_k^{\text{min}}\right\}$ which by assumption (ii) is bounded, it is easy to see that $\{\|T_k\|\}$ is bounded above and $\{\|T_k^{-\frac{1}{2}}\|\}$ is bounded away from zero.

Therefore (3.26) implies that both $\{p_k\}$ and $\{q_k\}$ are bounded. As mentioned above this suffices for α_k in order to be bounded away from zero.

Now we distinguish two cases. First assume that $[x_*]_i > 0$. An equivalent way of stating x_{k+1} and y_{k+1} using p_k and q_k is

$$x_{k+1} = X_k(e + \alpha_k p_k) \quad \text{and} \quad y_{k+1} = Y_k(e + \alpha_k q_k)\,, \tag{3.27}$$

which is why

$$1 = \lim_{k\to\infty} \frac{[x_{k+1}]_i}{[x_k]_i} = \lim_{k\to\infty}(1 + \alpha_k [p_k]_i)\,.$$

This implies $[p_k]_i \to 0$, because $\{\alpha_k\}$ is bounded away from zero. Since $f_k^{\text{ave}}/f_k^{\text{min}} = \|T_k e\|_\infty$, assumption (ii) together with $\sigma_k \to 0$ applied to (3.23) implies

$$\lim_{k\to\infty}(p_k + q_k) = -e\,. \tag{3.28}$$

This in turn implies now that $[q_k]_i \to -1$. If on the other hand $[x_*]_i = 0$, then $[y_*]_i > 0$ by assumption (i). The same argument as for the first case, interchanging the roles of p_k and q_k gives $[q_k]_i \to 0$ and $[p_k]_i \to -1$. □

Now we are ready to prove Theorem 3.3.

Proof of Theorem 3.3: Let

$$F_1(x, y) = \begin{pmatrix} Ax - b \\ By - BQx - Bc \end{pmatrix} \quad \text{and} \quad F_2(x, y) = XYe\,.$$

As we have already seen $F_1(x_k, y_k)$ is always zero hence we only have to show that $\{F_2(x_k, y_k)\}$ componentwise converges to zero Q-superlinearly. From (3.27) it follows that

$$X_k^{-1} x_{k+1} = e + \alpha_k p_k \quad \text{and} \quad Y_k^{-1} y_{k+1} = e + \alpha_k q_k\,.$$

Adding the above equations

$$X_k^{-1} x_{k+1} + Y_k^{-1} y_{k+1} = 2e + \alpha_k(p_k + q_k)\,,$$

taking the limit and using (3.28) and Lemma 3.6, we get

$$\lim_{k\to\infty}(X_k^{-1} x_{k+1} + Y_k^{-1} y_{k+1}) = e\,. \tag{3.29}$$

Again we will distinguish two cases. If $[x_*]_i = 0$, then by strict complementarity, $[y_*]_i > 0$ and $[y_{k+1}]_i/[y_k]_i \to 1$. It follows from (3.29) that $[x_{k+1}]_i/[x_k]_i \to 0$ and hence $[x_k]_i \to 0$ Q-superlinearly. Since (3.29) is symmetric in x and y we have $[y_k]_i \to 0$ Q-superlinearly if $[x_*]_i > 0$ by assumption (i). So for every i we have either

$$\lim_{k\to\infty} \frac{[x_{k+1}]_i}{[x_k]_i} = 0 \quad \text{and} \quad \lim_{k\to\infty} \frac{[y_{k+1}]_i}{[y_k]_i} = 1$$

or

$$\lim_{k\to\infty} \frac{[x_{k+1}]_i}{[x_k]_i} = 1 \quad \text{and} \quad \lim_{k\to\infty} \frac{[y_{k+1}]_i}{[y_k]_i} = 0\,.$$

In any case, it holds for every i that

$$\lim_{k\to\infty} \frac{[x_{k+1}]_i[y_{k+1}]_i}{[x_k]_i[y_k]_i} = \lim_{k\to\infty} \frac{[X_{k+1}Y_{k+1}e]_i}{[X_kY_ke]_i} = 0$$

which proves our Theorem. □

To solve the difficulties in our superlinear convergence analysis mentioned above is technically quite difficult and subject of the next chapter. Theorem 3.3 is surely sufficient to give a first idea how parameters should be chosen in practice to get an efficient algorithm. Moreover it served well to demonstrate which parameters influence the behaviour of primal-dual barrier IPMs in general.

4. A Globally and Superlinearly Convergent Primal-Dual Interior Point Method for Convex Quadratic Programming

4.1 Algorithm

In order to make Algorithm 1 globally and superlinearly convergent only by the choice of parameters which are directly under our control, we have to modify it somewhat. The following algorithm and the proofs of its global and superlinear convergence are based in big parts on [9].

Algorithm 2

Given a strictly feasible pair (x_0, y_0). For $k = 0, 1, \ldots$, do

Step 1 Optimality Test: If (x_k, y_k) satisfies (3.4) $\rightarrow$ STOP

Step 2 Compute the *descent directions*

$$\begin{pmatrix} \Delta x_k^D \\ \Delta y_k^D \end{pmatrix} = -\left[F'(x_k, y_k)\right]^{-1} F(x_k, y_k)$$

and the *centering directions*

$$\begin{pmatrix} \Delta x_k^C \\ \Delta y_k^C \end{pmatrix} = f_k^{\text{ave}} \left[F'(x_k, y_k)\right]^{-1} \begin{pmatrix} 0 \\ e \end{pmatrix} .$$

Step 3 Choose $\sigma_k \in [\,0, 1)$ by Procedure 1 and form the combined search directions

$$\begin{pmatrix} \Delta x_k \\ \Delta y_k \end{pmatrix} = \begin{pmatrix} \Delta x_k^D \\ \Delta y_k^D \end{pmatrix} + \sigma_k \begin{pmatrix} \Delta x_k^C \\ \Delta y_k^C \end{pmatrix} .$$

Step 4 Choose the steplength $\alpha_k \in (0, \hat{\alpha}_k)$ by Procedure 2, where $\hat{\alpha}_k$ was defined in (3.14).

Step 5 Compute the new iterates

$$\begin{pmatrix} x_{k+1} \\ y_{k+1} \end{pmatrix} := \begin{pmatrix} x_k \\ y_k \end{pmatrix} + \alpha_k \begin{pmatrix} \Delta x_k \\ \Delta y_k \end{pmatrix} ,$$

set $k := k + 1$ and go to **Step 1**.

Comparing Algorithm 2 with Algorithm 1, we notice that the search directions have been split into *descent* directions for the total complementarity—which for feasible iterates equals the duality-gap and serves as our measure of convergence— and *centering* directions. This was necessary because the choice of σ_k will depend on descent and centering directions. In the IPM literature the descent directions are also called *affine* or *affine scaling* directions.

The procedures for choosing σ_k and α_k will be described below. Proposition 3.1 is applicable to Algorithm 2 as well, hence it is well defined.

4.2 Choices of Parameters

4.2.1 Centering Parameter

Through the splitting of the search directions we are in need for more notation. Similarly to (3.22) let

$$\begin{aligned} p_k^D &:= X_k^{-1}\Delta x_k^D \ , \quad q_k^D := Y_k^{-1}\Delta y_k^D \ , \\ p_k^C &:= X_k^{-1}\Delta x_k^C \ , \quad q_k^C := Y_k^{-1}\Delta y_k^C \ . \end{aligned} \tag{4.1}$$

An important quantity in our further analysis is

$$\omega_k := \max_{1\le i\le n}(|[p_k^D]_i[q_k^D]_i|, |[p_k^D]_i[q_k^C]_i|, |[p_k^C]_i[q_k^D]_i|, |[p_k^C]_i[q_k^C]_i|) \ . \tag{4.2}$$

For technical reasons which will become clear in the course of this chapter we have to introduce the following set of $2n$ points (cf. also [27])

$$\Sigma_k := \left\{ -\frac{[p_k^D]_i}{[p_k^C]_i}, -\frac{[q_k^D]_i}{[q_k^C]_i} : i = 1, \ldots, n \right\} \tag{4.3}$$

and define the distance from a scalar ξ to the set Σ_k as

$$\operatorname{dist}(\xi, \Sigma_k) := \min \{ |\xi - \varsigma| : \varsigma \in \Sigma_k \} \ .$$

Now we can state our procedure for choosing the centering parameter σ_k.

Procedure 1

Given $\sigma \in (0,1),\ \gamma \in (0, \min\left(1/2, f_0^{\min}/f_0^{\text{ave}}\right)]\ ,\ \rho^l = \gamma^2\sigma/(2n),\ \rho^u \geq 24n$.

Step 1 Compute ω_k according to (4.2).

Step 2 Compute $\rho_k^u = \min(\rho^u, \sigma/\omega_k)$.

Step 3 Let $\sigma_k := \min\left\{\bar{\sigma} : \bar{\sigma} \in \left[\frac{\omega_k(\rho^l + \rho_k^u)}{2}\ ,\ \omega_k\rho_k^u\right]\ ,\ \operatorname{dist}(\bar{\sigma}, \Sigma_k) \geq \frac{\pi_k\omega_k}{8n+4}\right\}$, where $\pi_k := \rho_k^u - \rho^l$.

It should be remarked that the complicated choice of σ_k in Step 3 of the above procedure is merely for technical problems associated with the superlinear convergence analysis of Algorithm 2. For practical purposes it will suffice to define $\sigma_k := \rho_k^u\omega_k$.

The following lemma confirms that σ_k is well defined and $\sigma_k \in (0,1)$.

Lemma 4.1 *Let (x_k, y_k) and $(\Delta x_k, \Delta y_k)$ be produced by Algorithm 2, then it holds that*

1. $(\Delta x_k^D)^T\Delta y_k^D \geq 0$ *and* $(\Delta x_k^C)^T\Delta y_k^C \geq 0$,
2. *if* $f_k^{\min}/f_k^{\text{ave}} \geq \gamma$ *then* $\omega_k \leq n/\gamma^2$,
3. $\rho_k^u \geq \dfrac{\gamma^2\sigma}{n}$,
4. *the set* $\left\{\bar{\sigma} : \bar{\sigma} \in \left[\frac{\omega_k(\rho^l + \rho_k^u)}{2}\ ,\ \omega_k\rho_k^u\right]\ ,\ \operatorname{dist}(\bar{\sigma}, \Sigma_k) \geq \frac{\pi_k\omega_k}{8n+4}\right\}$ *is nonempty.*

Remark: As we will see in the next section, $f_k^{\min}/f_k^{\text{ave}} \geq \gamma$ is ensured by the choice of α_k. □

Proof:

Ad 1. Analogous to the proof of Lemma 3.5.

Ad 2. In analogy to (3.15) it holds that

$$Y_k\Delta x_k^D + X_k\Delta y_k^D = -X_kY_ke\ .$$

Multiplying both sides by $(X_kY_k)^{-\frac{1}{2}}$ and taking the 2-norm results in

$$\left\|(X_kY_k)^{\frac{1}{2}}p_k^D\right\|_2^2 + \left\|(X_kY_k)^{\frac{1}{2}}q_k^D\right\|_2^2 + 2(\Delta x_k^D)^T\Delta y_k^D = x_k^Ty_k\ .$$

Considering 1. and dividing both sides by f_k^{ave}, we get

$$\left\|T_k^{-\frac{1}{2}} p_k^D\right\|_2^2 + \left\|T_k^{-\frac{1}{2}} q_k^D\right\|_2^2 \leq n \ ,$$

where T_k is the diagonal matrix defined in (3.24). Our choice of γ and our assumption $f_k^{\min}/f_k^{\text{ave}} \geq \gamma$ now imply

$$\left|[p_k^D]_i\right| \leq \sqrt{\frac{n}{\gamma}} \leq \frac{\sqrt{n}}{\gamma} \quad \text{and} \quad \left|[q_k^D]_i\right| \leq \sqrt{\frac{n}{\gamma}} \leq \frac{\sqrt{n}}{\gamma} \ .$$

For the centering directions we have

$$Y_k \Delta x_k^C + X_k \Delta y_k^C = f_k^{\text{ave}} e \ ,$$

which using the same strategy as above leads to

$$\left\|T_k^{-\frac{1}{2}} p_k^C\right\|_2^2 + \left\|T_k^{-\frac{1}{2}} q_k^C\right\|_2^2 \leq \left\|T_k^{\frac{1}{2}} e\right\|_2^2 \leq n\gamma \leq n \ .$$

Hence we have also

$$\left|[p_k^C]_i\right| \leq \frac{\sqrt{n}}{\gamma} \quad \text{and} \quad \left|[q_k^C]_i\right| \leq \frac{\sqrt{n}}{\gamma} \ .$$

The result in 2. follows directly from the definition of ω_k.

Ad 3. By 2. we have

$$\rho_k^u = \min\left(\rho^u, \frac{\sigma}{\omega_k}\right) \geq \min\left(24n, \frac{\gamma^2\sigma}{n}\right) = \frac{\gamma^2\sigma}{n} \ . \tag{4.4}$$

Ad 4. First we note that by 3. it holds that

$$\pi_k = \rho_k^u - \rho^l \geq \frac{\gamma^2\sigma}{2n} \ . \tag{4.5}$$

Hence the interval $[\,\omega_k(\rho^l + \rho_k^u)/2, \omega_k\rho_k^u]$ of length $\pi_k\omega_k/2$ is nonempty. Partition this interval into $2n+1$ equal sub-intervals of length $\pi_k\omega_k/(4n+2)$ each. If the interior of any one of the sub-intervals does not intersect Σ_k, then the midpoint of this sub-interval will have the required distance to Σ_k. Since Σ_k only has $2n$ points, it cannot intersect the interiors of all the $2n+1$ sub-intervals.

This proof illustrates why it is necessary for σ_k to be chosen from an interval. Otherwise it would be impossible to ensure that we can always find a σ_k which has the required distance from Σ_k. $\square$

From the definition of σ_k in Step 3 of Procedure 1 it is now evident that σ_k is well defined and $0 < \sigma_k \leq \rho_k^u\omega_k \leq \sigma < 1$.

4.2.2 Steplength

As we already know from Theorem 3.3 our choice of steplength has to guarantee that $\{f_k^{\text{ave}}/f_k^{\min}\}$ is bounded. It is always bounded below by 1, though. Hence it suffices to require that for $\alpha = \alpha_k$ it holds that

$$\frac{f_k^{\min}(\alpha)}{f_k^{\text{ave}}(\alpha)} \geq \gamma\,, \quad \alpha > 0\,, \tag{4.6}$$

where γ was already chosen in Procedure 1.

Of course we still want to choose α_k as large as possible. To do so let us for notational convenience first introduce the following function

$$h_k(\alpha) = f_k^{\min}(\alpha) - \gamma\, f_k^{\text{ave}}(\alpha)\,. \tag{4.7}$$

Obviously (4.6) is equivalent to

$$h_k(\alpha) \geq 0\,, \quad \alpha > 0\,. \tag{4.8}$$

Since (4.8) is not the only condition we will place on α_k we define

$$\alpha_k^\gamma := \min\{\alpha > 0 : h_k(\alpha) = 0\}\,. \tag{4.9}$$

Let us make sure α_k^γ is well defined and satisfies (4.6).

Lemma 4.2 *The quantity α_k^γ is well defined and $\alpha_k^\gamma \in (0, \hat{\alpha}_k)$. Condition (4.6) is satisfied for all $\alpha \in (0, \alpha_k^\gamma]$.*

Proof: The proof is by induction over k. First we note that γ was chosen to satisfy $\gamma \leq f_0^{\min}/f_0^{\text{ave}}$. Now assume $\gamma \leq f_k^{\min}/f_k^{\text{ave}}$. Then it follows that

$$h_k(0) = f_k^{\min} - \gamma f_k^{\text{ave}} \geq 0\,.$$

On the other hand recalling (3.14) we have

$$h_k(\hat{\alpha}_k) = \underbrace{f_k^{\min}(\hat{\alpha}_k)}_{=0} - \gamma \underbrace{f_k^{\text{ave}}(\hat{\alpha}_k)}_{>0} < 0$$

In order to establish now that $h_k(\alpha)$ has a root in $[0, \hat{\alpha}_k)$ it is necessary to demonstrate that $h_k(\alpha)$ is a continuous function. Let us take a closer look at its components:

$$\begin{aligned} [f_k(\alpha)]_i &= [x_k(\alpha)]_i[y_k(\alpha)]_i \\ &= [x_k]_i[y_k]_i + \alpha\left([y_k]_i[\Delta x_k]_i + [x_k]_i[\Delta y_k]_i\right) + \alpha^2[\Delta x_k]_i[\Delta y_k]_i \\ &\overset{(3.15)}{=} [f_k]_i - \left([f_k]_i - \sigma_k f_k^{\text{ave}}\right)\alpha + [\Delta x_k]_i[\Delta y_k]_i\,\alpha^2\,, \end{aligned} \tag{4.10}$$

hence $f_k^{\min}(\alpha)$ is a continuous, piecewise quadratic function.

$$\begin{aligned} f_k^{\mathrm{ave}}(\alpha) &= \frac{1}{n}\sum_{i=1}^{n}[f_k(\alpha)]_i \\ &\overset{(4.10)}{=} \frac{1}{n}\sum_{i=1}^{n}[f_k]_i - \left(\frac{1}{n}\sum_{i=1}^{n}[f_k]_i - \sigma_k\frac{1}{n}nf_k^{\mathrm{ave}}\right)\alpha + \frac{1}{n}\sum_{i=1}^{n}[\Delta x_k]_i[\Delta y_k]_i\,\alpha^2 \\ &= f_k^{\mathrm{ave}}\left(1-(1-\sigma_k)\,\alpha\right) + \frac{\Delta x_k^T\Delta y_k}{n}\,\alpha^2\,, \end{aligned} \tag{4.11}$$

so $f_k^{\mathrm{ave}}(\alpha)$ is also a continuous, piecewise quadratic function which combined with (4.10) leads to the continuity of $h_k(\alpha)$ and therefore to a root in $[0,\hat{\alpha}_k)$. In case $h(0)>0$ there must be a root in $(0,\hat{\alpha}_k)$. If $h(0)=0$ then we have, considering (4.10) and (4.11)

$$\begin{aligned} h'(0^+) &= -(f_k^{\min}-\sigma_k f_k^{\mathrm{ave}}) + \gamma\,(1-\sigma_k)f_k^{\mathrm{ave}} \\ &= [-\underbrace{(f_k^{\min}/f_k^{\mathrm{ave}}+\gamma)}_{=0} + \underbrace{(1-\gamma)\sigma_k]}_{>0}\,\underbrace{f_k^{\mathrm{ave}}}_{>0} \\ &> 0\,. \end{aligned}$$

Therefore $h(\alpha)>0$ for positive but suitably small α. Thus $\alpha_k^{\gamma}\in(0,\hat{\alpha}_k)$ is well defined and condition (4.6) is automatically satisfied for all $\alpha\in(0,\alpha_k^{\gamma}]$ by definition (4.9). □

Our ultimate goal is to reduce the duality gap $x_k(\alpha)^T y_k(\alpha)$. So we will examine how the choice of steplength influences the reduction in the duality gap.

$$\begin{aligned} x_k(\alpha)^T y_k(\alpha) &= x_k^T y_k + \alpha(x_k^T\Delta y_k + y_k^T\Delta x_k) + \alpha^2\Delta x_k^T\Delta y_k \\ &\overset{(3.15)}{=} x_k^T y_k + \alpha(-x_k^T y_k + \sigma_k x_k^T y_k) + \alpha^2\Delta x_k^T\Delta y_k \\ &= x_k^T y_k(1-(1-\sigma_k))\,\alpha + \Delta x_k^T\Delta y_k\,\alpha^2 \end{aligned} \tag{4.12}$$

is a quadratic function of α with the second derivative $\Delta x_k^T\Delta y_k$. We know from Lemma 3.5 that $\Delta x_k^T\Delta y_k \geq 0$. If $\Delta x_k^T\Delta y_k>0$, the duality gap will reach its minimum at

$$\alpha_k^{\nu} := \frac{(1-\sigma_k)x_k^T y_k}{2\Delta x_k^T\Delta y_k} \tag{4.13}$$

and if $\Delta x_k^T\Delta y_k = 0$ it is a decreasing function of α and we should take the biggest steplength otherwisely possible.

Taking this into consideration, we are now ready to state our procedure for choosing the steplength α_k (cf. [9]).

Procedure 2

Given γ from Procedure 1.

Step 1 Compute α_k^{γ} according to (4.9) and if $\Delta x_k^T\Delta y_k>0$, α_k^{ν} according to (4.13).

Step 2 Let

$$\alpha_k := \begin{cases} \min(1, \alpha_k^{\gamma}, \alpha_k^{\nu}), & \text{if } \Delta x_k^T \Delta y_k > 0 \\ \min(1, \alpha_k^{\gamma}), & \text{otherwise} \end{cases} . \tag{4.14}$$

As we have seen above this procedure guarantees that $\alpha_k \in (0, \hat{\alpha}_k)$, the duality gap is reduced as much as possible and the prerequisite (4.6) for superlinear convergence is satisfied.

4.3 Global Convergence

First though we will analyze the global convergence of our objective function, i.e. the duality gap. As was to be expected this once again requires some new notation,

$$\eta_k := \frac{\gamma}{n} \Delta x_k^T \Delta y_k - \min([\Delta x_k]_i [\Delta y_k]_i) . \tag{4.15}$$

If we plug α_k into equation (4.12) it comes already close to a global convergence result

$$x_{k+1}^T y_{k+1} = x_k^T y_k \left(1 - (1 - \sigma_k) \alpha_k + \frac{\Delta x_k^T \Delta y_k}{x_k^T y_k} \alpha_k^2 \right) ,$$

but we still need to show that for

$$\delta_k := \left(1 - \sigma_k - \frac{\Delta x_k^T \Delta y_k}{x_k^T y_k} \alpha_k \right) \alpha_k \tag{4.16}$$

it holds that $\{\delta_k\}$ is bounded away from zero and $\delta_k < 1$. The latter is trivial considering that $\sigma_k \in (0, 1)$, $\Delta x_k^T \Delta y_k \geq 0$, $x_k^T y_k > 0$ and $\alpha_k \in (0, 1]$. Thus our goal in this section is to prove the following result

Theorem 4.3 *Let $\{(x_k, y_k)\}$ be generated by Algorithm 2. Then it holds that*

$$x_{k+1}^T y_{k+1} \leq x_k^T y_k (1 - \delta_k) ,$$

and $\delta_k \in (\delta, 1)$, where $\delta > 0$ is a constant independent of k. Hence the duality gap globally converges to zero at least at a linear rate.

Before we prove Theorem 4.3 we will first establish some lemmas that give estimates of some quantities.

Lemma 4.4

1. Let η_k be given by (4.15). It holds that $\eta_k \leq 6 \omega_k f_k^{\max}$.

2. Let α_k^γ be given by (4.9). If $\eta_k \le 0$ then $\alpha_k^\gamma \ge 1$; otherwise

$$\alpha_k^\gamma \ge \frac{(1-\gamma)\sigma_k f_k^{\text{ave}}}{\eta_k}\,.$$

Proof:

Ad 1. Note that

$$\begin{aligned}
|[\Delta x_k]_i[\Delta y_k]_i| &\overset{(3.23)}{=} |[x_k]_i[p_k]_i[y_k]_i[q_k]_i| \\
&\le \max(X_kY_ke)\max_{1\le i\le n}(|[p_k]_i[q_k]_i|) \\
&= f_k^{\max}\max_{1\le i\le n}(|[p_k^D]_i[q_k^D]_i + \sigma_k[p_k^D]_i[q_k^C]_i + \\
&\qquad\qquad +\sigma_k[p_k^C]_i[q_k^D]_i + \sigma_k^2[p_k^C]_i[q_k^C]_i|) \\
&\overset{(4.2)}{\le} 4f_k^{\max}\omega_k\,.
\end{aligned}$$

It follows that

$$\Delta x_k^T\Delta y_k \le 4nf_k^{\max}\omega_k \tag{4.17}$$

and therefore

$$\begin{aligned}
\eta_k &= \frac{\gamma}{n}\Delta x_k^T\Delta y_k - \min([\Delta x_k]_i[\Delta y_k]_i) \\
&\le 4\gamma f_k^{\max}\omega_k + 4f_k^{\max}\omega_k \\
&\le 6f_k^{\max}\omega_k\,,
\end{aligned}$$

because we chose $\gamma \le 1/2$.

Ad 2. Let $\alpha \in [\,0,1]$. Then we have

$$\begin{aligned}
[f_k(\alpha)]_i &\overset{(4.10)}{=} [f_x]_i(1-\alpha) + \sigma_k f_k^{\text{ave}}\alpha + [\Delta x_k]_i[\Delta y_k]_i\,\alpha^2 \\
&\ge f_k^{\min}(1-\alpha) + \sigma_k f_k^{\text{ave}}\alpha + \min([\Delta x_k]_i[\Delta y_k]_i)\,\alpha^2
\end{aligned}$$

and therefore, recalling (4.11)

$$\begin{aligned}
[f_k(\alpha)]_i - \gamma f_k^{\text{ave}}(\alpha) &\ge \underbrace{(f_k^{\min} - \gamma f_k^{\text{ave}})}_{\ge 0}\underbrace{(1-\alpha)}_{\ge 0} + (1-\gamma)\sigma_k f_k^{\text{ave}}\alpha - \eta_k\alpha^2 \\
&\ge (1-\gamma)\sigma_k f_k^{\text{ave}}\alpha - \eta_k\alpha^2\,.
\end{aligned} \tag{4.18}$$

Since we assumed that $\alpha \in [\,0,1]$ it follows now considering (4.7) that if $\eta_k \le 0$, $h_k(\alpha)$ has no root in $(0,1)$. Hence $\alpha_k^\gamma \ge 1$.

On the other hand if $\eta_k > 0$ the right hand side of (4.18) has a unique positive root in

$$\bar\alpha = \frac{(1-\gamma)\sigma_k f_k^{\text{ave}}}{\eta_k}$$

and it is greater than zero for all $\alpha \in (0,\bar\alpha)$. It follows from (4.18) that $h_k(\alpha) > 0$ for $\alpha \in (0,\bar\alpha)$. So if $\alpha_k^\gamma < 1$ by (4.9) α_k^γ must be greater than or equal to $\bar\alpha$. □

Lemma 4.5 *Let (x_k, y_k), $(\Delta x_k, \Delta y_k)$ and α_k^γ be generated by Algorithm 2.*

1. There exists a constant $\beta > 0$, such that

$$\min(1, \alpha_k^\gamma) \geq \frac{\beta}{n^2} \,. \tag{4.19}$$

2. It holds that

$$\frac{\Delta x_k^T \Delta y_k}{x_k^T y_k} \leq \frac{1}{4\gamma} \,. \tag{4.20}$$

Proof:

Ad 1. By Lemma 4.4 (2.), it obviously suffices to consider the case where $\eta_k > 0$ for which follows

$$\begin{aligned}
\min(1, \alpha_k^\gamma) &\geq \min\left(1, \frac{(1-\gamma)\sigma_k f_k^{\mathrm{ave}}}{6\omega_k f_k^{\max}}\right) , && \text{by Lemma 4.4,} \\
&\geq \min\left(1, \frac{\rho^l}{12n}\right) , && \text{since } \gamma \leq \frac{1}{2} \,,\ \sigma_k \geq \rho^l \omega_k \text{ and } \frac{f_k^{\max}}{f_k^{\mathrm{ave}}} \leq n \,, \\
&= \min\left(1, \frac{\gamma^2\sigma}{24n^2}\right) , && \text{since } \rho^l = \frac{\gamma^2\sigma}{2n} \,.
\end{aligned}$$

Thus,

$$\beta := \frac{\gamma^2 \sigma}{24} \tag{4.21}$$

is a suitable choice.

Ad 2. Using previous results we get

$$\begin{aligned}
4\Delta x_k^T \Delta y_k &\leq \left\| X_k^{-\frac{1}{2}} Y_k^{\frac{1}{2}} \Delta x_k - X_k^{\frac{1}{2}} Y_k^{-\frac{1}{2}} \Delta y_k \right\|_2^2 + 4\Delta x_k^T \Delta y_k \\
&= \left\| (X_k Y_k)^{\frac{1}{2}} p_k \right\|_2^2 + \left\| (X_k Y_k)^{\frac{1}{2}} q_k \right\|_2^2 + 2\Delta x_k^T \Delta y_k \\
&\overset{(3.25)}{=} x_k^T y_k \left(1 - 2\sigma_k + \sigma_k^2 f_k^{\mathrm{ave}} \, \frac{e^T (X_k Y_k)^{-1} e}{n}\right) \\
&\overset{(4.6)}{\leq} x_k^T y_k \left(1 - 2\sigma_k + \frac{\sigma_k^2}{\gamma}\right) .
\end{aligned}$$

Recalling $\sigma_k < 1$ it now follows that

$$\begin{aligned}
\frac{\Delta x_k^T \Delta y_k}{x_k^T y_k} &\leq \frac{1}{4}\left(1 - 2\sigma_k + \frac{\sigma_k^2}{\gamma}\right) \\
&= \frac{1}{4}\left((1-\sigma_k)^2 + \sigma_k^2\left(\gamma^{-1} - 1\right)\right) \\
&\leq \frac{1}{4\gamma} \,,
\end{aligned}$$

which completes the proof. □

The preceding lemmas are helpful in getting a better understanding of our choice of σ_k. From the proof of Lemma 4.4 (1.) we see how the definition of ω_k came about and the proof of Lemma 4.5 (1.) illustrates that ω_k has to be part of σ_k in order to get a uniform lower bound on α_k^γ. We again emphasize that this far it would have sufficed to choose $\sigma_k = \rho_k^u \omega_k$.

Proof of Theorem 4.3: In our analysis we will use the following two functions of α

$$\delta_k(\alpha) := \left(1 - \sigma_k - \frac{\Delta x_k^T \Delta y_k}{x_k^T y_k}\,\alpha\right)\alpha \tag{4.22}$$

and

$$\bar\delta(\alpha) := \left(1 - \sigma - \frac{1}{4\gamma}\,\alpha\right)\alpha\,. \tag{4.23}$$

Using (4.20) and the fact that $\sigma_k < \sigma$, we see that for $\alpha \geq 0$ it holds that $\delta_k(\alpha) \geq \bar\delta(\alpha)$.

We will first demonstrate that

$$\alpha_k = \arg\max\left\{\delta_k(\alpha) : \alpha \in [\,0, \min(1, \alpha_k^\gamma)]\right\}\,. \tag{4.24}$$

The derivatives of $\delta_k(\alpha)$ are

$$\delta_k'(\alpha) = 1 - \sigma_k - 2\,\frac{\Delta x_k^T \Delta y_k}{x_k^T y_k}\,\alpha$$

and

$$\delta_k''(\alpha) = -2\,\frac{\Delta x_k^T \Delta y_k}{x_k^T y_k}\,.$$

Hence only if $\Delta x_k^T \Delta y_k > 0$, $\delta_k(\alpha)$ has a unique maximum in α_k^ν and for $\alpha < \alpha_k^\nu$, $\delta_k(\alpha)$ is an increasing function of α. For this case (4.24) now follows from definition (4.14) of α_k. If $\Delta x_k^T \Delta y_k = 0$, $\delta_k(\alpha)$ is an increasing function of α altogether and (4.24) again follows from (4.14).

Therefore we have for all $\alpha \in [\,0, \min(1, \alpha_k^\gamma)]$,

$$\delta_k = \delta_k(\alpha_k) \geq \delta_k(\alpha) \geq \bar\delta(\alpha)\,. \tag{4.25}$$

If we define β as in (4.21) then (4.19) implies that (4.25) holds for $\alpha = \beta/n^2$. Hence we get

$$\delta_k \geq \delta := \bar\delta(\beta/n^2) = \left(1 - \sigma - \frac{\beta}{4\gamma n^2}\right)\frac{\beta}{n^2}$$

which completes the proof and this section. □

4.4 Superlinear Convergence

For the superlinear convergence analysis we will make use of the results of the preceding chapter. One of the assumptions of Theorem 3.3 was that the sequence $\{f_k^{\text{ave}}/f_k^{\min}\}$ be bounded. This is ensured by our choice of α_k. Looking at the proof of Theorem 3.3 it is evident that the assumption $\tau_k \to 1$ can be replaced by $\alpha_k \to 1$ which is more meaningful in our case.

However, to make our superlinear convergence result even more general, we will first show that it is not necessary to require that (x_k, y_k) converges to a strictly complementary solution of (P). For $x \in \mathbb{R}^n$ we define

$$I^+(x) := \{i : x_i > 0\}$$

and note that for any solution $(\hat{x}, \hat{y})$ of the optimality conditions (3.4) of our problem (P) it holds that

$$I^+(\hat{x}) \cap I^+(\hat{y}) = \emptyset\,. \tag{4.26}$$

If (3.4) has a strictly complementary solution (x_*, y_*) we have on top of this

$$I^+(x_*) \cup I^+(y_*) = \{1, \ldots, n\}\,. \tag{4.27}$$

The following result is based in big parts on Güler [8].

Lemma 4.6 *Let (x_k, y_k) be generated by Algorithm 2. Assume (3.4) has a strictly complementary solution (x_*, y_*). Then*

1. *for every solution $(\hat{x}, \hat{y})$ of (3.4) it holds that*

$$I^+(\hat{x}) \subseteq I^+(x_*) \quad \text{and} \quad I^+(\hat{y}) \subseteq I^+(y_*)\,,$$

2. *for every limit point (x_∞, y_∞) of the sequence $\{(x_k, y_k)\}$ it holds that (x_∞, y_∞) solves (3.4) and*

$$I^+(x_\infty) = I^+(x_*) \quad \text{and} \quad I^+(y_\infty) = I^+(y_*)\,,$$

   i.e. every limit point is a strictly complementary solution of (3.4).

Proof:

Ad 1. First note that by the positive semi-definiteness of Q for any $x_1, x_2 \in \mathbb{R}^n$

$$\begin{aligned} 0 &\leq (x_1 + x_2)^T Q (x_1 + x_2) \\ &= x_1^T Q x_1 + x_2^T Q x_2 + 2 x_1^T Q x_2 \end{aligned}$$

and thus

$$2 x_1^T Q x_2 \leq x_1^T Q x_1 + x_2^T Q x_2\,.$$

Recalling the equivalent optimality conditions (3.3) it now follows

$$\begin{aligned} 0 &= x_*^T y_* + \hat{x}^T \hat{y} \\ &= x_*^T(-A^T\lambda_* + Qx_* + c) + \hat{x}^T(-A^T\hat{\lambda} + Q\hat{x} + c) \\ &= -b^T\lambda_* + x_*^T Q x_* + c^T x_* - b^T\hat{\lambda} + \hat{x}^T Q\hat{x} + c^T\hat{x} \\ &\geq -b^T\hat{\lambda} + x_*^T Q\hat{x} + c^T\hat{x} - b^T\lambda_* + \hat{x}^T Q x_* + c^T x_* \\ &= x_*^T\hat{y} + \hat{x}^T y_*\,, \end{aligned}$$

for some $\lambda_*, \hat{\lambda}$. Since $(x_*, y_*), (\hat{x}, \hat{y}) \geq 0$ the result follows immediately.

Ad 2. From Theorem 4.3 it follows that for all k

$$x_k^T y_k \leq x_0^T y_0\,.$$

Using this estimate and proceeding similarly to 1. we get

$$x_k^T y_0 + x_0^T y_k \leq x_k^T y_k + x_0^T y_0 \leq 2x_0^T y_0$$

and hence

$$[x_k]_i \leq \frac{2x_0^T y_0}{[y_0]_i} \quad \text{and} \quad [y_k]_i \leq \frac{2x_0^T y_0}{[x_0]_i}\,.$$

Consequently the sequence $\{(x_k, y_k)\}$ is bounded and therefore it has at least one limit point. Considering that $x_k^T y_k \to 0$, every limit point has to satisfy $x_\infty^T y_\infty = 0$ and thus every limit point solves (3.4).

By 1. it remains to be shown that

$$I^+(x_*) \subseteq I^+(x_\infty) \quad \text{and} \quad I^+(y_*) \subseteq I^+(y_\infty)\,.$$

Recalling $x_*^T y_* = 0$, for every k we have

$$\begin{aligned} & x_*^T y_k + x_k^T y_* \leq x_k^T y_k + x_*^T y_* = x_k^T y_k \\ \Leftrightarrow\quad & \sum_{i\in I^+(x_*)} [x_*]_i [y_k]_i \; + \sum_{i\in I^+(y_*)} [x_k]_i [y_*]_i \leq x_k^T y_k \end{aligned}$$

We see from (4.26) and (4.27) that for every i, either $i \in I^+(x_*)$ or $i \in I^+(y_*)$. If $i \in I^+(x_*)$,

$$\begin{aligned} & x_k^T y_k \geq [x_*]_i [y_k]_i = [f_k]_i \frac{[x_*]_i}{[x_k]_i} \geq f_k^{\min} \frac{[x_*]_i}{[x_k]_i} \\ \Rightarrow\quad & [x_k]_i \geq n\frac{f_k^{\min}}{f_k^{\text{ave}}}\,[x_*]_i \overset{(4.6)}{\geq} n\gamma\,[x_*]_i \\ \Rightarrow\quad & [x_\infty]_i \geq n\gamma\,[x_*]_i > 0 \\ \Rightarrow\quad & I^+(x_*) \subseteq I^+(x_\infty)\,. \end{aligned}$$

The case $i \in I^+(y_*)$ is treated analogously. □

In view of Lemma 4.6 it would be very nice now of course if we could get a superlinear convergence result without having to assume the convergence of the iteration sequence $\{(x_k, y_k)\}$ at all. However, unfortunately we were unable to prove the following Theorem without this assumption. An improvement in comparison to Theorem 3.3 and Theorem 5.2 in [9] though is that we no longer need to assume the convergence of $\{(x_k, y_k)\}$ to a strictly complementary solution, but we merely need convergence. The proof of Theorem 4.7 will also give a final explanation for our choice of σ_k.

Theorem 4.7 *Let $\{(x_k, y_k)\}$ be generated by Algorithm 2. Assume that (3.4) has a strictly complementary solution and that $(x_k, y_k) \to (x_*, y_*)$. Then the sequence $\{X_k Y_k e\}$ componentwise converges to zero Q-superlinearly.*

Proof: As we have already explained we can use the results of Theorem 3.3. By Lemma 4.6, (x_*, y_*) is a strictly complementary solution of (3.4). Hence all that is left for us to show is that $\sigma_k \to 0$ and $\alpha_k \to 1$.

We first prove $\sigma_k \to 0$. Since ρ_k^u is bounded, by the definition of σ_k in Step 3 of Procedure 1 it suffices to show $\omega_k \to 0$.

Let $[x_*]_i > 0$. Then clearly

$$1 = \lim_{k\to\infty} \frac{[x_{k+1}]_i}{[x_k]_i} = \lim_{k\to\infty} (1 + \alpha_k [p_k]_i) .$$

In order for this to imply that $[p_k]_i \to 0$ we still need that α_k is bounded away from zero. The answer gives us Lemma 4.5, because

$$\min(1, \alpha_k^\gamma) \geq \frac{\beta}{n}$$

and furthermore

$$\frac{\Delta x_k^T \Delta y_k}{x_k^T y_k} \leq \frac{1}{4\gamma}$$

indicates that

$$\alpha_k^\nu = \frac{(1-\sigma_k) x_k^T y_k}{2 \Delta x_k^T \Delta y_k} \geq 2(1-\sigma)\gamma .$$

Hence by the definition of α_k we see that it is bounded away from zero.

If on the other hand $[x_*]_i = 0$, then $[y_*]_i > 0$ by strict complementarity. Proceeding analogue to above we get $[q_k]_i \to 0$. As a result for each i,

$$\text{either} \quad [p_k]_i = [p_k^D]_i + \sigma_k [p_k^C]_i \to 0 \quad \text{or} \quad [q_k]_i = [q_k^D]_i + \sigma_k [q_k^C]_i \to 0 . \tag{4.28}$$

This is where the reason for our choice of σ_k lies. Even though we have (4.28), we cannot guarantee that if, say $[p_k]_i \to 0$ this also holds for $[p_k^D]_i$ and $[p_k^C]_i$. Accordingly we have to make sure that σ_k stays far enough from $-[p_k^D]_i/[p_k^C]_i$. Hopefully our choice of parameters is fully transparent to the reader now.

The proof of $\omega_k \to 0$ is by contradiction. Suppose the opposite. Then, there must exist a subsequence $\{\omega_{k(j)}\} \subseteq \{\omega_k\}$ which is bounded away from zero since $\omega_k \geq 0$. The definition of σ_k and (4.5) then imply that $\{\text{dist}(\sigma_{k(j)}, \Sigma_{k(j)})\}$ is bounded away from zero.

Assume $[p_k]_i \to 0$. Then, we claim that $\{[p^C_{k(j)}]_i\} \to 0$. If this was not true then there would be a subsequence $\{[p^C_{k(l)}]_i\} \subseteq \{[p^C_{k(j)}]_i\}$ for which $\{|[p^C_{k(l)}]_i|\}$ is bounded away from zero. Since $[p_k]_i \to 0$ this must also hold for every subsequence and thus also for $[p_{k(l)}]_i$ so that we have

$$[p_{k(l)}]_i = [p^D_{k(l)}]_i + \sigma_{k(l)}[p^C_{k(l)}]_i = [p^C_{k(l)}]_i \left(\frac{[p^D_{k(l)}]_i}{[p^C_{k(l)}]_i} + \sigma_{k(l)} \right) \to 0\,.$$

Considering that $\{|[p^C_{k(l)}]_i|\}$ is bounded away from zero, this implies

$$\frac{[p^D_{k(l)}]_i}{[p^C_{k(l)}]_i} + \sigma_{k(l)} \to 0\,,$$

which contradicts the fact that $\{\text{dist}(\sigma_{k(l)}, \Sigma_{k(l)})\} \subseteq \{\text{dist}(\sigma_{k(j)}, \Sigma_{k(j)})\}$ is bounded away from zero. Therefore $\{[p^C_{k(j)}]_i\} \to 0$. In view of (4.28) this indicates that $\{[p^D_{k(j)}]_i\} \to 0$, as well.

The case where $[q_k]_i \to 0$ can be treated analogously. Consequently for each i, either $[p_k^D]_i$ and $[p_k^C]_i$ or $[q_k^D]_i$ and $[q_k^C]_i$ converge to zero. In the proof of Lemma 4.1 we showed that all these sequences are bounded and therefore by definition (4.2) of ω_k it follows that $\omega_{k(j)} \to 0$. This contradicts our hypothesis that $\{\omega_{k(j)}\}$ is bounded away from zero. Hence $\omega_k \to 0$ and ergo $\sigma_k \to 0$.

In order to prove $\alpha_k \to 1$ we have to distinguish two cases. If $\Delta x_k^T \Delta y_k = 0$, then $\alpha_k = \min(1, \alpha_k^\gamma)$. From Lemma 4.4, $\gamma < 1/2$ and $f_k^{\max}/f_k^{\text{ave}} \leq n$ it follows that

$$\alpha_k^\gamma \geq \frac{\rho^l + \rho_k^u}{24n}\,.$$

Since $\rho_k^u = \min(\rho^u, \sigma/\omega_k)$, $\omega_k \to 0$ and $\rho^u \geq 24n$, we have for k sufficiently large

$$\rho^l + \rho_k^u \geq \rho_k^u = \rho^u \geq 24n\,.$$

Consequently $\alpha_k = 1$ for k sufficiently large in this case.

Note that we did not use $\Delta x_k^T \Delta y_k = 0$ to get this result. If $\Delta x_k^T \Delta y_k > 0$ we have $\alpha_k = \min(1, \alpha_k^\gamma, \alpha_k^\nu)$ and so all we need to show is that $\alpha_k^\nu \geq 1$ for k sufficiently large.

This can be seen from

$$\begin{aligned}
\alpha_k^\nu &= \frac{(1-\sigma_k)x_k^T y_k}{2\Delta x_k^T \Delta y_k} \\
&\geq \frac{(1-\sigma)x_k^T y_k}{2\Delta x_k^T \Delta y_k}, \quad \text{since } \sigma_k < \sigma \\
&\geq \frac{(1-\sigma)f_k^{\text{ave}}}{8\omega_k f_k^{\max}}, \quad \text{by (4.17)} \\
&\geq \frac{1-\sigma}{8n\omega_k}, \qquad \text{since } \frac{f_k^{\text{ave}}}{f_k^{\max}} \geq \frac{1}{n}.
\end{aligned}$$

Now in view of $\omega_k \to 0$ it follows that for k sufficiently large $\alpha_k^\nu \geq 1$ and consequently $\alpha_k = 1$. □

5. Implementation

It is a well known fact in numerical mathematics that in practice many things are different, e.g. the algorithms with good theoretical convergence properties and the algorithms that perform well in practice frequently have very little in common. On the other hand to get a practical algorithm it is often necessary to find solutions for problems that do not exist in theory due to idealizing assumptions. These are the main reasons why we dedicate a lot of space to the topic of implementing IPMs for portfolio optimization problems. We will present two algorithms that performed best in our numerical experiments. As we will see their choice of parameters is consistent with Theorem 3.3.

All algorithms were implemented in the excellent, high-performance numeric computation package MATLAB 4.0 by The MathWorks, Inc., Natick, Mass. All subroutines listed in the Appendix are written in the macro-language of MATLAB.

5.1 The Models in Practice

The Models that we will consider are models (MV3) and (MV3′) from Chapter 2 with box constraints and slack variable x_s for model (MV3).

$$\text{(PMV)} \qquad \begin{array}{ll} \text{Minimize} & x_f^T x_f \\[1em] \text{subject to} & (I - \frac{1}{k} ee^T) P x_p - x_f = 0 \\ & \frac{1}{k} e^T P x_p - x_s = \alpha \\ & e^T x_p = 1 \\ & 0 \le x_p \le u_p \\ & 0 \le x_s \le \infty \\ & -\infty \le x_f \le \infty \end{array}$$

and

$$
(\mathrm{PMV}') \qquad
\begin{array}{ll}
\text{Minimize} & x_f^T x_f \\[1em]
\text{subject to} & P x_p - x_f = \alpha e \\
 & e^T x_f = 0 \\
 & e^T x_p = 1 \\
 & 0 \le x_p \le u_p \\
 & -\infty \le x_f \le \infty \, ,
\end{array}
$$

for some upper bounds vector $u_p \in \mathbb{R}^n$. Remember that $x_p \in \mathbb{R}^n$, $x_f \in \mathbb{R}^k$ and $P \in \mathbb{R}^{k \times n}$. We have to include these upper bounds on x_p, because we want to avoid the situation where the weight of certain assets in the solution-portfolio becomes too big hence making the portfolio too susceptible to fluctuations in the prices of those assets. We will typically choose $u_p = Ce$, where C is a scalar which of course depends on n. It is also practically convenient to express the free variable x_f as one with infinite box constraints. We chose not to include any other additional linear constraints because those constraints are mostly situation specific. Plus it is very easy to extend our models to include them without having to change any part of our algorithms.

In order to make our models more concise we define for (PMV)
the $(n+1+k) \times (n+1+k)$-matrix

$$
Q := \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & I \end{bmatrix} , \tag{5.1}
$$

the $(k+2) \times (n+1+k)$-matrix

$$
A := \begin{bmatrix} (I - \frac{1}{k} e e^T) P & 0 & -I \\ \frac{1}{k} e^T P & -1 & 0 \\ e^T & 0 & 0 \end{bmatrix} , \tag{5.2}
$$

the $(k+2)$-vector

$$
b := \begin{pmatrix} 0 \\ \alpha e \\ 1 \end{pmatrix} \tag{5.3}
$$

and the $(n+1+k)$-vectors

$$
x := \begin{pmatrix} x_p \\ x_s \\ x_f \end{pmatrix} , \quad l := \begin{pmatrix} 0 \\ 0 \\ -\infty \end{pmatrix} , \quad u := \begin{pmatrix} u_p \\ \infty \\ \infty \end{pmatrix} . \tag{5.4}
$$

Now (PMV) is equivalent to

$$
\text{(PQP)} \qquad \begin{array}{ll} \text{Minimize} & \frac{1}{2}x^TQx \\ \text{subject to} & Ax = b \\ & l \le x \le u\,. \end{array}
$$

Similarly, for (PMV′), we define
the $(n+k) \times (n+k)$-matrix

$$
Q' := \begin{bmatrix} 0 & 0 \\ 0 & 0 \\ 0 & I \end{bmatrix}, \tag{5.5}
$$

the $(k+2) \times (n+k)$-matrix

$$
A' := \begin{bmatrix} P & -I \\ 0 & e^T \\ e^T & 0 \end{bmatrix}, \tag{5.6}
$$

the $(k+2)$-vector

$$
b' := \begin{pmatrix} \alpha e \\ 0 \\ 1 \end{pmatrix} \tag{5.7}
$$

and the $(n+k)$-vectors

$$
x' := \begin{pmatrix} x_p \\ x_f \end{pmatrix}, \quad l' := \begin{pmatrix} 0 \\ -\infty \end{pmatrix}, \quad u' := \begin{pmatrix} u_p \\ \infty \end{pmatrix}. \tag{5.8}
$$

Accordingly (PMV′) is equivalent to

$$
\text{(PQP}'\text{)} \qquad \begin{array}{ll} \text{Minimize} & \frac{1}{2}x'^TQ'x' \\ \text{subject to} & A'x' = b' \\ & l' \le x' \le u'\,. \end{array}
$$

These notations enable us to treat (PQP) and (PQP′) as identical for the most part. Hence from now on we will work only with (PQP).

In the implementation the models are being generated by the subroutine `mvtosta` (**m**ean **v**ariance **to** **sta**ndard model) which is listed in the Appendix. It only requires P, α and u_p as input data and due to the structure of Q we can make good use of MATLAB's sparse data type [22].

If u_p is empty it is uniformly set to infinity. If it is shorter than x_p, say of length i, then only the first i components of x_p are treated as bounded, while the remaining components have infinite bounds.

We define m and $\bar{n}$ as the number of rows and columns of A, respectively. The first order optimality conditions of (PQP) are (cf. (3.3))

$$G_0(x,y,z,\lambda) := \begin{pmatrix} Ax - b \\ A^T\lambda + y - z - Qx \\ S_l Y e - 0e \\ S_u Z e - 0e \end{pmatrix} = 0, \qquad \begin{array}{l} l \le x \le u \\ 0 \le y \\ 0 \le z \end{array} , \tag{5.9}$$

where $S_l := X - L$ and $S_u := U - X$. Theoretically it is of course impossible to satisfy the last two equations, because of the infinite bounds. In practice however we set $[y_k]_i$ and $[z_k]_i$ to zero for all k if $l_i = -\infty$ or $u_i = \infty$, respectively. This method has proven more effective in our numerical experiments than the variable splitting technique known from the simplex method (cf. also [23]).

5.2 Pure Primal-Dual Algorithm

The algorithm described in this section is implemented in the subroutine `pd_ipm` which is also listed in the Appendix. In essence the algorithm follows the scheme of Algorithm 1 except that we do not require (x_0, y_0) to be strictly feasible and the centering parameter μ_k and the steplength α_k are computed by special procedures which will be described below. Of course the termination criteria have to be also different to suit practical considerations.

5.2.1 Initialization

There are two checks that are performed at the very beginning. First the algorithm makes sure that the dimensions of the input data match. Then it checks if the constraint matrix A has full (row-)rank. If any of the checks fail, the algorithm stops with an error message. Currently there is no preprocessor included that removes redundant equations.

Unlike the theoretical algorithms in Chapters 3 and 4, the algorithm does *not* require an initial strictly feasible solution but will use one if one is given by the user. Instead it has proven far more efficient [13] to compute initial values for x, y, z and λ such that the box constraints are satisfied and the other constraints are approximately satisfied and then start from this infeasible point. There have been first efforts to establish theoretical convergence results for these so called *infeasible* IPMs, e.g. [26, 29], which indicate that very similar choices of parameters as for Algorithm 2 are adequate. However our choice of parameters is influenced by the more practically oriented works of Lustig et al. [4, 2, 13].

The default procedure for computing the initial points was partly motivated by the one in [14].

Procedure 3

Given $\bar{\theta} > 0$.

Step 1 Compute $\bar{x}_0 := A^T(AA^T)^{-1}b$ and $\theta_P := \bar{\theta}\,\|\bar{x}_0\|_1/n$.

Step 2 For $i = 1, \ldots, n$, set $[x_0]_i := (u_i - l_i)/2$, if $u_i - l_i <= 2\theta_P$ or else

$$[x_0]_i := \begin{cases} u_i - \theta_P\,, & \text{if } u_i - [\bar{x}_0]_i < \theta_P \\ l_i + \theta_P\,, & \text{if } [\bar{x}_0]_i - l_i < \theta_P \\ [x_0]_i\,, & \text{otherwise.} \end{cases}$$

Step 3 Set $\lambda_0 := 0$. Compute $v_0 := Qx_0 + c$ and $\theta_D := \bar{\theta}\,\|v_0\|_1/n$.

Step 4 For $i = 1, \ldots, n$, set $[y_0]_i := 0$, if $l_i = -\infty$ or else

$$[y_0]_i := \begin{cases} [v_0]_i\,, & \text{if } [v_0]_i > \theta_D \\ \theta_D\,, & \text{otherwise,} \end{cases}$$

and $[z_0]_i := 0$, if $u_i = \infty$ or else

$$[z_0]_i := \begin{cases} -[v_0]_i\,, & \text{if } [v_0]_i < -\theta_D \\ \theta_D\,, & \text{otherwise.} \end{cases}$$

The computation of the pseudoinverse of A in Step 1 is possible, because we have made sure that A has full row-rank. Overall, Procedure 3 computes initial values x_0, y_0 and z_0 which are at least a certain threshold θ_P for x_0 and θ_D for y_0 and z_0 away from the relevant boundaries. The user can influence this threshold value by changing the scaling parameter $\bar{\theta}$. After extensive testing it seems that $\bar{\theta} = .3$ is a good universal choice.

On the other it is attempted to keep the norm of the initial infeasibility or residual, i.e.

$$\|r_0\|_2 := \left\| \begin{pmatrix} Ax_0 - b \\ A^T\lambda_0 + y_0 - z_0 - Qx_0 \end{pmatrix} \right\|_2 ,$$

as small as possible.

5.2.2 Computation of Centering Parameter

Our procedure for computing the centering parameter μ_k was motivated by the ones in the primal-dual algorithms for linear-programming described in [13, 14]. In the case of infeasible starting points, μ_k must play the important role of a feasibility parameter. This means that μ_k should be large as long as primal and dual feasibility have not been attained yet, because the centering directions point from the current iterates away from the boundary into the interior of the feasible region. Hence more centering will allow for larger steps before the nonnegativity constraints restrict the steplength. Of course another result of this technique is that the emphasis on attaining feasibility is greater than on attaining optimality. However, from our numerical experiments we feel that this has no significant negative effect on the efficiency of the algorithm.

Our procedure for computing μ_k is as follows.

Procedure 4

Given $\epsilon > 0$, $\xi > 0$,

$$\phi := \begin{cases} \bar{n}^2\,, & \text{if } n \leq 5000, \\ \bar{n}^{3/2}\,, & \text{if } n > 5000, \end{cases} \tag{5.10}$$

and

$$M := \xi\, \phi \, \max(Q, b)\,, \tag{5.11}$$

where $\max(Q, b)$ refers to the largest component of both items in the parenthesis and ϵ is the same that is being used in the termination criteria.

Step 1 If primal and dual feasibility has been attained, i.e.

$$\frac{\|Ax_k - b\|_1}{1 + \|x_k\|_1} < \epsilon \quad \text{and} \quad \frac{\|A^T\lambda_k + y_k - z_k - Qx_k - c\|_1}{1 + \|x_k\|_1 + \|y_k\|_1 + \|z_k\|_1 + \|\lambda_k\|_1} < \epsilon\,,$$

then

$$\mu_k := \frac{(x_k - l)^T y_k + (u - x_k)^T z_k}{\phi}\,,$$

otherwise

$$\mu_k := \frac{x_k^T Q x_k - b^T\lambda_k - l^T y_k + u^T z_k + M\|r_k\|_1/\|r_0\|_1}{\phi}$$

where like above r_k is the residual of primal and dual constraints at the k-th step.

This procedure merits an explanation. First it is apparent that for feasible iterates μ_k is similar to the one in Algorithm 1 with $\sigma_k = n/\phi$. Thus σ_k is a constant rather than converging to zero. This works much better in practice than for instance choosing σ_k as the minimum of duality gap and say .99, in which case the centering parameter converges to zero much slower than for our choice.

When feasibility has not been achieved yet, we obviously have to compute the duality gap as the difference between primal and dual objective function since the total complementarity no longer equals the duality gap. In this case M controls the influence the degree of infeasibility has on μ_k. The bigger M, the larger μ_k will be chosen when the degree of infeasibility is still high. M takes into account the scaling of the problem and can be controlled by the user by adjusting the parameter ξ. We choose $\xi = .45$.

Since we are dealing with a very homogeneous class of problems there is no need to adjust any of the parameters involved in the computation of μ_k. For an interesting discussion about this topic for the case of LP see [13].

5.2.3 Computation of Search Directions

Due to the infeasible start, the system defining the search directions in our case is of course also different from the feasible case when the upper part of the right hand side is zero. We still proceed similarly as in Algorithm 1 though and define the search directions as solutions to the system

$$G'_{\mu_k}(x_k, y_k, z_k, \lambda_k) \begin{pmatrix} \Delta x_k \\ \Delta y_k \\ \Delta z_k \\ \Delta \lambda_k \end{pmatrix} = -G_{\mu_k}(x_k, y_k, z_k, \lambda_k) \,,$$

where $G.$ was defined in (5.9). Dropping the index k for notational convenience, this is equivalent to

$$\begin{pmatrix} A & 0 & 0 & 0 \\ -Q & I & -I & A^T \\ Y & S_l & 0 & 0 \\ -Z & 0 & S_u & 0 \end{pmatrix} \begin{pmatrix} \Delta x \\ \Delta y \\ \Delta z \\ \Delta \lambda \end{pmatrix} = \begin{pmatrix} b - Ax \\ -A^T\lambda - y + z + Qx \\ \mu e - S_l Y e \\ \mu e - S_u Z e \end{pmatrix} . \qquad (5.12)$$

If we set

$$\begin{pmatrix} p_1 \\ p_2 \\ p_3 \\ p_4 \end{pmatrix} := \begin{pmatrix} b - Ax \\ -A^T\lambda - y + z + Qx \\ \mu e - S_l Y e \\ \mu e - S_u Z e \end{pmatrix} , \qquad (5.13)$$

we can compute the solutions to (5.12) directly by

$$\begin{pmatrix} \Delta \lambda \\ \Delta x \\ \Delta y \\ \Delta z \end{pmatrix} = \begin{pmatrix} (AH^{-1}A^T)^{-1}\left[AH^{-1}(p_2 - S_l^{-1}p_3 + S_u^{-1}p_4) + p_1\right] \\ H^{-1}(-p_2 + S_l^{-1}p_3 - S_u^{-1}p_4 + A^T\Delta\lambda) \\ S_l^{-1}(p_3 - Y\Delta x) \\ S_u^{-1}(p_4 + Z\Delta x) \end{pmatrix} \qquad (5.14)$$

with $H := Q + S_l^{-1}Y + S_u^{-1}Z$. When computing S_l^{-1} and S_u^{-1}, the diagonal elements associated with infinite bounds are set to zero.

The computation of the search directions, in particular of $(AH^{-1}A^T)^{-1}$ uses up most of the computation time in each iteration. Fortunately H is a diagonal-matrix and as such easy to invert. Moreover the increasingly ill condition of H which is an inherent property of all IPMs does not pose a problem in practice. It is worth noting though that H can only be inverted when x_k, y_k and z_k are not too close to their respective bound which is why the starting points are made to meet this requirement.

Recalling (5.2) and (5.6) we see that A unfortunately is mostly dense. Our actual strategy therefore is to compute the cholesky factors of $AH^{-1}A^T$ and then solve the defining equation for $\Delta\lambda$ by forward and backward substitution. The quantities that are needed several times are just computed once and then stored in temporary variables. All diagonal matrices needed in the computation are stored as sparse matrices again [22].

It should be remarked that Monteiro and Adler [19] devise an updating scheme for the cholesky factors of $AH^{-1}A^T$ which exploits the fact that only diagonal elements of H change at each iteration. This reduces the bound on the number of operations per iteration [7] from $O(n^3)$ to $O(n^{2.5})$. We did not implement this scheme because we feel that it would not have a significant effect on the performance of the algorithm.

5.2.4 Computation of Steplength

Our choice of steplength in practice is not limited by technical theoretical considerations. The vast majority of practical IPM implementations (e.g. [15, 23]) use a slightly smaller steplength than the maximal possible, i.e. in order for x_k, y_k and z_k to stay strictly within their respective limits.

We compute the maximal possible steplengths separately for primal ($\hat{\alpha}^P$) and dual ($\hat{\alpha}^D$) variables. For simplicity we will drop the index k again.

$$\hat{\alpha}^P = \min\left\{\min_i\left\{-\frac{x_i - l_i}{\Delta x_i} \ : \Delta x_i < 0\right\},\ \min_i\left\{\frac{u_i - x_i}{\Delta x_i} \ : \Delta x_i > 0\right\}\right\} \tag{5.15}$$

and

$$\hat{\alpha}^D = \min\left\{\min_i\left\{-\frac{y_i}{\Delta y_i} \ : \Delta y_i < 0\right\},\ \min_i\left\{-\frac{z_i}{\Delta z_i} \ : \Delta z_i < 0\right\}\right\}. \tag{5.16}$$

For LP it has proven very efficient to use different step sizes for primal and dual variables [13, 15] and there is also no reason for concern from a theoretical point of view against this technique, because in LP there are no primal variables in the dual constraints and vice versa. In QP this is different however since we have the term Qx in the dual constraints, hence when we take different steplengths the next iterate may violate these constraints. Therefore a theoretically correct choice of steplength would be

$$\alpha := \begin{cases} c_\alpha \min(\hat{\alpha}^P, \hat{\alpha}^D)\,, & \text{if } \min(\alpha^P, \alpha^D) \le 1\,, \\ 1\,, & \text{otherwise} \end{cases}$$

with typically $c_\alpha := .99995$.

After extensive testing with various choices of steplengths, including the extreme case of different steplengths for all variables —which did not work for one single problem by the way— it became evident that when using different steplengths for primal and dual variables we never achieved a slower but in most cases a considerably faster convergence than for the case when the same steplength was used for all variables. Moreover in most cases the iterates became feasible after less steps. This will certainly not always be the case and especially not for all convex QP but since it worked so well we stuck with the technique.

Consequently our procedure for computing the steplengths and for updating the iterates is the following.

Procedure 5

Given $c_\alpha := .99995$,

Step 1 Compute $\hat{\alpha}_k^P$ and $\hat{\alpha}_k^D$ by (5.15) and (5.16), respectively.

Step 2 Compute

$$\alpha_k^P := \begin{cases} c_\alpha \hat{\alpha}_k^P \,, & \text{if } c_\alpha \hat{\alpha}_k^P \leq 1 \,, \\ 1 \,, & \text{otherwise} \end{cases} \quad \text{and} \quad \alpha_k^D := \begin{cases} c_\alpha \hat{\alpha}_k^D \,, & \text{if } c_\alpha \hat{\alpha}_k^D \leq 1 \,, \\ 1 \,, & \text{otherwise.} \end{cases}$$

Step 3 Update the iterates by

$$\begin{aligned} x_k &\leftarrow x_k + \alpha_k^P \Delta x_k \\ y_k &\leftarrow y_k + \alpha_k^D \Delta y_k \\ z_k &\leftarrow z_k + \alpha_k^D \Delta z_k \\ \lambda_k &\leftarrow \lambda_k + \alpha_k^D \Delta \lambda_k \end{aligned}$$

and set $k \leftarrow k + 1$.

5.2.5 Termination Criteria

Due to the above choice of steplengths we cannot solely rely on the duality-gap as termination criterion, but we also have to make sure that the iterates are still "sufficiently" feasible. Therefore the algorithm terminates if for the total complementarity it holds that

$$(x_k - l)^T y_k + (u - x_k)^T z_k < \epsilon$$

and for the relative infeasibility

$$\frac{\|r_k\|_1}{1 + \|x_k\|_1 + \|y_k\|_1 + \|z_k\|_1 + \|\lambda_k\|_1} < \epsilon \,,$$

with currently $\epsilon = 10^{-7}$. This choice of ϵ is mainly to allow performance comparisons with other implementations. In most situations the algorithm converges also for smaller ϵ.

Since we did not include a mechanism for detecting infeasible or ill-posed problems the alternative termination criterion is that a given maximal number of iterations has been reached. Currently this number is 100 which is reasonable because typically the algorithm terminates after less than 20 iterations. Unsolvable problems usually give themselves away by causing numerical difficulties after about 20 iterations because one of the objective values tends to infinity.

5.3 Predictor-Corrector Algorithm

Recently it has become evident through several publications e.g. [15, 3, 14, 29, 1] to name only a few that Mehrotra's predictor-corrector algorithm [18] and variants of it clearly dominate the field of IPMs as far as practical efficiency and local convergence is concerned. This was reason enough to include an implementation of this method in our numerical experiments. To tell it right away, our results fully support the good reputation, but before we look at the results we will give a brief description of the method and our implementation of it.

5.3.1 Motivation

In motivating the predictor-corrector algorithm, we will follow the very neat description in [14].

The basic idea behind this IPM is to use the "expensive" cholesky factorization of $AH^{-1}A^T$ that is needed in the computation of the search directions, twice in every step and thereby gain extra information about the central path through the current iterates to the optimal solution.

This is achieved by first solving (5.12) for the affine directions, i.e.

$$\begin{pmatrix} A & 0 & 0 & 0 \\ -Q & I & -I & A^T \\ Y & S_l & 0 & 0 \\ -Z & 0 & S_u & 0 \end{pmatrix} \begin{pmatrix} \Delta\hat{x} \\ \Delta\hat{y} \\ \Delta\hat{z} \\ \Delta\hat{\lambda} \end{pmatrix} = \begin{pmatrix} b - Ax \\ -A^T\lambda - y + z + Qx \\ -S_l Y e \\ -S_u Z e \end{pmatrix} . \tag{5.17}$$

This is called the *predictor* step.

Similarly to Algorithm 2 these directions are then used to determine the centering parameter μ. Analogous to (5.15) and (5.16), let $\hat{\alpha}^{\mathcal{P}}$ be the minimum of the primal and dual maximal possible steplengths if the predictor directions were used and

$$\alpha^{\mathcal{P}} := \min(1, .99995\, \hat{\alpha}^{\mathcal{P}}) .$$

Then the new total complementarity after a step in the predictor directions is

$$\hat{g} := (x + \alpha^{\mathcal{P}}\Delta\hat{x} - l)^T(y + \alpha^{\mathcal{P}}\Delta\hat{y}) + (u - x - \alpha^{\mathcal{P}}\Delta\hat{x})^T(z + \alpha^{\mathcal{P}}\Delta\hat{z})$$

and μ is computed by

$$\mu := \left(\frac{\hat{g}}{(x-l)^T y + (u-x)^T z}\right)^2 \frac{\hat{g}}{n}\,. \tag{5.18}$$

Since the predictor directions are descent directions for the total complementarity the first fraction will always be smaller than 1. Moreover it will be small when good progress can be made in the predictor directions and large when these directions promise little improvement due to a small possible stepsize which usually indicates the need for more centering. The second fraction is exactly μ_k from Algorithm 1.

Next, in the corrector step, the actual search directions are computed as solutions to

$$\begin{pmatrix} A & 0 & 0 & 0 \\ -Q & I & -I & A^T \\ Y & S_l & 0 & 0 \\ -Z & 0 & S_u & 0 \end{pmatrix} \begin{pmatrix} \Delta x \\ \Delta y \\ \Delta z \\ \Delta\lambda \end{pmatrix} = \begin{pmatrix} b - Ax \\ -A^T\lambda - y + z + Qx \\ \mu e - S_l Y e - \Delta\hat{X}\Delta\hat{Y} \\ \mu e - S_u Z e + \Delta\hat{X}\Delta\hat{Z} \end{pmatrix}. \tag{5.19}$$

Clearly all that has changed in comparison to the pure primal-dual algorithm (cf. (5.12)) are the *corrector* terms $-\Delta\hat{X}\Delta\hat{Y}$ and $\Delta\hat{X}\Delta\hat{Z}$ on the right hand side. The computation of step size and the procedures for updating the iterates are the same as in the pure primal-dual algorithm.

So the extra work that has to be done for the predictor-corrector algorithm is the backsolve to compute the predictor directions and the ratio test to determine $\alpha^{\mathcal{P}}$. But what is gained? We will show that one step of the predictor-corrector algorithm approximately combines one step in the predictor or affine direction and from there one step of the pure primal-dual algorithm.

Note that we can write the search directions as

$$\begin{aligned} \Delta x &= \Delta\hat{x} + c_x \\ \Delta y &= \Delta\hat{y} + c_y \\ \Delta z &= \Delta\hat{z} + c_z \\ \Delta\lambda &= \Delta\hat{\lambda} + c_\lambda\,, \end{aligned}$$

where the correction terms satisfy

$$\begin{pmatrix} A & 0 & 0 & 0 \\ -Q & I & -I & A^T \\ Y & S_l & 0 & 0 \\ -Z & 0 & S_u & 0 \end{pmatrix} \begin{pmatrix} c_x \\ c_y \\ c_z \\ c_\lambda \end{pmatrix} = \begin{pmatrix} b - Ax \\ -A^T\lambda - y + z + Qx \\ \mu e - \Delta\hat{X}\Delta\hat{Y} \\ \mu e + \Delta\hat{X}\Delta\hat{Z} \end{pmatrix}. \tag{5.20}$$

Note further that

$$\begin{aligned} & (X + \Delta\hat{X} - L)(Y + \Delta\hat{Y})e \\ = \; & (S_l + \Delta\hat{X})(Y + \Delta\hat{Y})e \\ = \; & \underbrace{S_l Y e + Y\Delta\hat{x} + S_l\Delta\hat{y}}_{=0 \text{ by } (5.17)} + \Delta\hat{X}\Delta\hat{Y} \end{aligned}$$

and similarly

$$(U - X - \Delta\hat{X})(Z + \Delta\hat{Z})e = -\Delta\hat{X}\Delta\hat{Z}\,.$$

Thus (5.20) defines pure primal-dual search directions from the point $x + \Delta\hat{x}$, $y + \Delta\hat{y}$, $z + \Delta\hat{z}$, $\lambda + \Delta\hat{\lambda}$ that would result from a full step in the predictor directions, except that the terms $\Delta\hat{x}$, $\Delta\hat{y}$, $\Delta\hat{z}$ have not been added to the diagonal matrices on the left hand side. That means that instead of using the Jacobian at the point resulting from a full step in the affine directions, the Jacobian at the current point x, y, z, λ is being used.

Despite the approximation this technique produces excellent search directions frequently leading to a considerably smaller number of total iterations. A logical extension of this approach is to attempt to use one matrix factorization even more often. There has been some research in this direction [3, 2] which indicates that the number of iterations often can be reduced even more with more "correcting".

5.3.2 Implementation

The predictor-corrector algorithm is implemented in the subroutine `pc_ipm` which can be found in the Appendix.

In accordance with the brief description of the algorithm above, the implementation is very similar to the one of the pure primal-dual algorithm. The initialization procedure is the same as in 5.2.1, except that we set $\bar{\theta} = 1$, because for reasons given below it is desirable to keep farther away from the boundaries.

The computation of the search directions and the centering parameter is according to the description above, with a little difference. Our actual procedure for computing μ is as follows.

Procedure 6

Given $\bar{\mu} := .7$.

Step 1 If primal or dual feasibility has not been attained and

$$\frac{\|r_k\|_1}{(x_k - l)^T y_k + (u - x_k)^T z_k} > 10^3\,,$$

then set $\mu_k := \bar{\mu}$ and skip Step 2.

Step 2 If $(x_k - l)^T y_k + (u - x_k)^T z_k < 1$ and primal and dual feasibility has been attained, then

$$\mu_k := \frac{(x_k - l)^T y_k + (u - x_k)^T z_k}{\phi},$$

otherwise compute μ_k by (5.18) but with separate steplengths for primal and dual variables in computing $\hat{g}$.

The second step was partly motivated by [14], i.e. when the iterates are close to an optimal solution we use the same centering parameter as in the pure primal-dual algorithm to avoid potentially numerically unstable systems.

The first step has proven to be essential, because before it was included the algorithm mostly did not converge. This happens when the starting points are too close to their bounds which sometimes cannot be avoided due to small upper bounds. Then it often happened in our experiments that the first fraction in (5.18) was close to 1 but the second fraction was too small for this to have a significant effect on μ_k. Consequently the algorithm got caught in the erroneous assumption that it was already close to a solution and perpetually chose μ_k much too small. As a remedy we had to include a mechanism that also takes into account the infeasibility. The heuristic in Step 1 with the user-controllable parameter $\bar{\mu}$ has proven to serve this purpose very well. Typically it only affects the first step. Lustig et al. [14] work around this problem by not requiring x_k to initially satisfy the upper bound but allowing it to iterate to bound feasibility and hence choosing large initial points.

For the same practical reasons given in 5.2.4 we compute $\hat{g}$ with separate steplengths for primal and dual variables. The computation of the actual steplengths, the updating scheme for the iterates and the termination criteria are the same as for the pure primal-dual algorithm.

5.4 Numerical Results

All numerical experiments were carried out on a DEC Alpha workstation running OSF/1 with the numeric computation software package MATLAB 4.0. Although the MATLAB optimization toolbox contains a QP solver we did not use it for performance comparisons because its performance was too poor on our problems.

Three sets of test data, i.e. three matrices P, from the Tokyo Stock Exchange Market were most generously provided by Messrs. Ken-ichi Suzuki and Hiroshi Konno from the Tokyo Institute of Technology. They have been used in the numerical experiments in [11, 21] before. The test set called r8912 contains 60 monthly rates of return (January 1985 – December 1989) for the stocks included in the Nikkei225 index. The test set called r9012 contains the same kind of data for the period January 1986 – December 1990. The test set called tsem contains the same data but for all 1064 stocks traded in the Tokyo Stock Exchange Market for the 152 months between January 1980 and August 1982. The sets called tsem1, tsem2 and tsem3 are the subsets $P(1:60,1:225)$, $P(61:200,226:450)$ and $P(93:152,451:675)$ of tsem, respectively.

Besides the real world data we also used 3 randomly generated data sets. The MATLAB routine that was used to generate the data sets is called randprob and can be found in the Appendix. The size of the sets is 60×225 for rand1 and rand2 and 60×2500 for rand3. The sets were generated by rand1: `randprob(60,225,2,11.5,0,0)`, rand2: `randprob(60,225,-2,1,8,0)` and rand3: `randprob(60,2500,-2,1,8,0)`. The sets rand2 and rand3 were deliberately chosen to be almost infeasible for our choice of $\alpha = 2$ and $u_p = .03e$, to see how the algorithms behave.

In all tests the required rate of return was $\alpha = 2.0$. The columns in Tables 5.1 and 5.2 have the following meanings: 'Model' refers to the two models we are considering, where '$\geq$' means model (PQP) and '=' means model (PQP$'$). An 'M' in this column indicates that the regular MV-model (MV2) with the according equality or inequality constraint was used. 'Bound' refers to the upper bound u_p and the actual value in column 'Bound' is the uniform upper bound on all components of x_p. Obviously the 'Solver' column gives the algorithm that was used, where PC stands for the predictor-corrector algorithm and PD for the pure primal-dual algorithm. 'Total # Steps' is the number of steps the algorithm needed to reach optimality and 'Infeasible # Steps' is the number of steps the algorithm needed until primal and dual feasibility was reached. As already mentioned, the 'CPU Times' were achieved on a DEC Alpha workstation and computed by the built-in MATLAB function cputime. The column '1-Norm Residual' lists the 1-norm of the last residual vector r_k. To decide whether a stock was in or out the solution portfolio, i.e. if the corresponding component of x_p was zero or not, we used the convergence tolerance ϵ. All components of x_p smaller than ϵ were assumed to be zero. Of course there are more reliable and yet inexpensive ways to decide this but they are not within the scope of this thesis. For more information about identifying zero components we refer the interested reader to [5].

It should be remarked that the built-in QP solver of MATLAB was used to detect infeasible problems.

Even though our implementations are only research codes, they still exhibit quite a good performance, as can be seen in Tables 5.1 and 5.2. The results are definitely better than the ones reported in [11] and [21] where the same kind of data was used. The last three entries of Table 5.2 clearly demonstrate that our algorithms even find solutions for nearly infeasible problems in still an acceptable amount of time. On the other hand there is no doubt that there is still room for improvement in our implementations.

An important remark concerning the results of data set rand1 is that the fact that there are 225 stocks in the solution does not contradict the result at the end of Chapter 2. According to this result there exists a solution of model (MV3) such that there are no more than 62 stocks in the solution portfolio. Unfortunately the algorithm did not find one of these solutions in this case. Hence we will have to do some extra computations, i.e. solve an *easy* LP to identify a solution with the desired properties.

The results of our experiments support the clear dominance of the predictor-corrector algorithm. It only got beat once by the pure primal-dual algorithm, as far as the total number of iterations and the CPU-time is concerned, and this might even be due to a weakness in our implementation.

As far as the models are concerned, the six examples we included are sufficient to demonstrate that the compact models (PQP) and (PQP$'$) are clearly much faster to solve than the regular MV-model (MV2).

A final remark has to be made regarding MATLAB's `spdiags` routine. When called with an empty first argument, e.g. `spdiags([],0,0,0)` it produces an error message while one would expect it to simply return an empty matrix. This somewhat unpredictable behavior causes trouble when calling `pd_ipm` or `pc_ipm` with empty lower or upper bounds on the design variable. Thus instead of programming around this "bug" we decided to rather "fix" the `spdiags` routine in such a way that it now returns an empty result when called with an empty first argument.

Table 5.1: Numerical Results I

Data	Model	Bound	Solver	Total # Steps	Infeasible # Steps	CPU Time	1-Norm Residual	# Stocks in solution
r8912	‘=’	1	PC	10	8	4.67	1.11E-12	28
r8912	‘=’	1	PD	15	11	6.05	4.31E-12	28
r8912	M ‘=’	1	PC	14	2	49.03	2.11E-15	28
r8912	M ‘=’	1	PD	23	5	79.15	3.78E-15	28
r8912	‘≥’	1	PC	10	9	4.68	4.93E-12	32
r8912	‘≥’	1	PD	16	7	6.38	4.39E-13	32
r8912	‘=’	.03	PC	11	10	5.07	1.06E-11	42
r8912	‘=’	.03	PD	16	13	6.40	9.39E-14	42
r8912	‘≥’	.03	PC	12	7	5.35	8.01E-13	48
r8912	‘≥’	.03	PD	19	13	7.45	9.17E-13	48
r9012	‘=’	1	PC	11	10	5.03	1.14E-10	21
r9012	‘=’	1	PD	14	7	5.77	1.77E-12	21
r9012	‘≥’	1	PC	11	10	5.08	5.08E-11	21
r9012	‘≥’	1	PD	14	7	5.72	1.35E-12	21
r9012	‘=’	.03	PC	14	10	6.13	2.14E-13	45
r9012	‘=’	.03	PD	18	12	7.05	1.97E-12	45
r9012	‘≥’	.03	PC	14	11	6.18	2.09E-12	45
r9012	‘≥’	.03	PD	18	12	7.08	2.38E-12	45
r9012	M ‘≥’	.03	PC	12	1	43.87	1.55E-15	45
r9012	M ‘≥’	.03	PD	24	3	86.12	5.55E-16	45
tsem1	‘=’	1	PC	10	8	4.68	1.80E-12	29
tsem1	‘=’	1	PD	15	5	6.02	1.79E-11	29
tsem1	‘≥’	1	PC	10	8	4.70	2.17E-12	29
tsem1	‘≥’	1	PD	15	5	6.07	2.54E-11	29
tsem1	‘=’	.03	PC	9	8	4.30	1.93E-11	40
tsem1	‘=’	.03	PD	16	11	6.47	4.53E-14	40
tsem1	‘≥’	.03	PC	9	8	4.37	7.17E-11	40
tsem1	‘≥’	.03	PD	16	11	6.38	1.49E-11	40
tsem2	‘=’	1	PC	10	8	4.67	1.83E-11	32
tsem2	‘=’	1	PD	15	9	6.02	1.50E-11	32
tsem2	‘≥’	1	PC	11	8	5.08	4.31E-10	36
tsem2	‘≥’	1	PD	17	10	6.77	7.09E-10	36
tsem2	‘=’	.03	PC	11	9	5.05	1.65E-11	48
tsem2	‘=’	.03	PD	16	11	6.45	1.58E-12	48
tsem2	‘≥’	.03	PC	10	8	4.72	6.25E-11	47
tsem2	‘≥’	.03	PD	16	10	6.38	8.88E-12	47

Table 5.2: Numerical Results II

Data	Model	Bound	Solver	Total # Steps	Infeasible # Steps	CPU Time	1-Norm Residual	# Stocks in solution
tsem3	'='	1	PC	17	13	7.27	2.56E-14	10
tsem3	'='	1	PD	20	15	7.78	2.02E-14	10
tsem3	'≥'	1	PC	17	13	7.23	4.13E-14	10
tsem3	'≥'	1	PD	20	15	7.85	4.60E-14	10
tsem3	M '≥'	1	PC	16	5	58.33	1.55E-15	10
tsem3	M '≥'	1	PD	20	5	71.73	1.33E-15	10
tsem3	'='	.03	Infeasible Problem					
tsem3	'≥'	.03	Infeasible Problem					
rand1	'='	1	PC	7	3	3.58	1.11E-15	225
rand1	'='	1	PD	14	5	5.83	1.55E-14	225
rand1	'≥'	1	PC	7	3	3.50	1.55E-14	225
rand1	'≥'	1	PD	14	5	5.77	2.00E-15	225
rand1	'='	.03	PC	7	3	3.55	9.99E-16	225
rand1	'='	.03	PD	13	4	5.45	6.66E-16	225
rand1	'≥'	.03	PC	7	3	3.53	5.00E-15	225
rand1	'≥'	.03	PD	13	4	5.48	1.78E-15	225
rand2	'='	1	PC	13	10	5.77	8.81E-12	6
rand2	'='	1	PD	12	7	5.05	1.91E-12	6
rand2	'≥'	1	PC	12	9	5.50	1.21E-11	6
rand2	'≥'	1	PD	18	8	7.12	1.03E-11	6
rand2	'='	.03	Infeasible Problem					
rand2	'≥'	.03	Infeasible Problem					
tsem	'='	1	PC	12	10	128.1	1.42E-11	36
tsem	'='	1	PD	16	11	156.6	3.77E-10	36
tsem	'≥'	1	PC	12	10	126.6	4.31E-12	36
tsem	'≥'	1	PD	16	11	159.3	6.09E-10	36
tsem	'≥'	.03	PC	17	12	169.2	1.60E-09	49
tsem	'≥'	.03	PD	21	13	196.3	1.88E-09	49
rand3	'='	1	PC	26	6	102.2	7.56E-12	33
rand3	'='	1	PD	33	9	118.1	7.30E-10	34
rand3	'≥'	1	PC	23	7	89.9	1.17E-11	33
rand3	'≥'	1	PD	32	6	113.2	1.39E-09	34
rand3	'≥	.03	Infeasible Problem					
rand3	'≥'	.04	PC	33	13	126.4	1.44E-11	37
rand3	'≥'	.04	PD	44	16	154.6	1.49E-11	37

6. Conclusions and Summary

In this thesis we demonstrated that by transforming a MV model into a separable QP and by applying IPMs to this separable representation, Markowitz' model and hence also most models based on it now can be solved in a very practical amount of time even for large problems.

The transformation itself makes the model generation very easy by using historical data and permits to solve the MV model in usually 10% of the time that would be needed to solve the regular MV model. At the same time the algorithms that we implemented generally find solution portfolios with a very manageable number of assets. Even if they don't it only requires a little extra work to find such a solution as its existence is guaranteed by the separable representation.

We have seen that the behavior of the presented primal-dual IPM is mainly controlled by the centering and the steplength parameters and that it required a substantial amount of work to find choices of these parameters which guarantee the global and superlinear convergence of the duality gap. On the other hand the numerical experiments have shown that in practice the choices of parameters do not have to be restricted so narrowly to get practically efficient algorithms. Very interesting, especially from a practical point of view is the possibility of starting with infeasible iterates and yet to achieve reliable fast convergence in all solvable cases that we tested. A fairly new approach might be the way we deal with free variables in practice, i.e. to treat them as variables with infinite lower and upper bounds which works very well.

From our and the cited results it is save to say that at this time the different variants of Mehrotra's predictor-corrector IPM offer the best practical efficiency. The cited sources seem to support its dominance over all other IPMs also in theory.

Considering that portfolio models are used in a variety of areas, more professional versions of the presented portfolio optimization methods have the potential of being included in a wide range of software packages in the future.

Appendix

A. Listings of Routines

A.1 Model Generator: mvtosta

```
function [H,c,A,b,lboundx,uboundx] = mvtosta(P,alpha,uboundx,which)
% MVTOSTA
%       [H,c,A,b,lboundx,uboundx] = mvtosta(P,alpha,uboundx,which)
%       transforms the values of a problem given in the form of a Mean-
%       Variance model with compact factorization of the covariance-
%       matrix into a standard QP-model with lower and upper bounds on
%       (possibly just parts of) the design variable x:
%
%       Min x'Hx+c'x
%
%       s.t Ax=b , lboundx<=x<=uboundx
%
%       Due to the size and structure of the problems considered the function
%       takes advantage of the sparse matrix type: H and c are returned
%       as sparse matrices.


[k,n] = size(P);

% First handle missing arguments
if nargin < 3 uboundx=[];end

if which == '1'                         % Regular case
  lboundx = zeros(n,1);
  A = [[P ; zeros(1,n) ; ones(1,n)] [-eye(k); ones(1,k);zeros(1,k)]];
  b = [alpha*ones(k,1);0;1];
  H = [sparse(n,k+n);sparse(k,n) speye(k)];
  c = sparse(n+k,1);
elseif which == '2'                     % >= constraint for alpha
  lboundx = zeros(n+1,1);
  A = [[(eye(k)-(1/k)*ones(k))*P ; 1/k*ones(1,k)*P ; ones(1,n)] ...
      [zeros(k,1);-1;0] [-eye(k); zeros(2,k)]];
  b = [sparse(k,1);alpha;1];
  H = [sparse(n+1,k+n+1);sparse(k,n+1) speye(k)];
  c = sparse(n+k+1,1);
elseif which == '3'                     % Point on the efficient frontier
  lboundx = zeros(n,1);
  A = [[(eye(k)-(1/k)*ones(k))*P ; ones(1,n)]  [-eye(k); zeros(1,k)]];
  b = [sparse(k,1);1];
  H = [sparse(n,k+n);sparse(k,n) speye(k)];
  c = [-alpha/k*P'*ones(k,1);sparse(k,1)];
end
```

A.2 Pure Primal-Dual Algorithm: pd_ipm

```
function [xstar,ystar,zstar,lambda]=pd_ipm(A,H,b,c,lboundx,uboundx,x0,maxit,tol)
% [xstar,ystar,zstar,lambda]=pd_ipm(A,H,b,c,lboundx,uboundx,x0,y0,maxit,tol)
%
% Primal-Dual Interior Point Method Quadratic Optimization
%
% pd_ipm solves convex Quadratic Programs
%
% Minimize 1/2 x'*H*x + c'*x subject to A*x == b, lboundx <= x <= uboundx
%
% using a primal-dual interior point method.
%
% H must be positive semi-definite and A must have full rank.
%
% The user has the option of incorporating lower and upper bounds on x, using
% lboundx and uboundx, respectively. If lboundx or uboundx are shorter than x,
% the remainig components of x are assumed to be unbounded in the respective
% direction.
%
% A starting value x0 can be given but it has to satisfy the box-constraints.
%
% The default maximal number of iterations is 100 and can be adjusted by setting
% maxit.
%
% The default termination tolerance is 1E-7 and can be adjusted by setting tol.
%
% The routine returns the solution xstar and the lagrangian multipliers, where
% ystar is associated with lboundx, zstar is associated with uboundx and lambda
% is associated with the equality constraints.


% First set a couple of default values
%
  defmaxit = 100;             % Default value for the maximum number of iterations
  deftol   = 1e-7;            % Default tolerance for objective value and residuals

  threshold_scale = .3;       % This is used for x0 and y0
  xi = .45;                   % Only an initial value used for sigmak,
                              % the centering parameter
  alpha_scale=.99995;         % Used for computation of alphak


% Determine dimensions of the problem
%
  [m n] = size(A);
  if (size(H)~=[n,n])|(size(b)~=[m,1])|(size(c)~=[n,1])| ...
     (size(uboundx,1)>n)|(size(lboundx,1)>n)
    error('Dimensions mismatch !!!');
  end

  nucstr=size(uboundx,1);    % Number of upper-constrained variables
  nufree = n-nucstr;
  nlcstr=size(lboundx,1);    % Number of lower-constrained variables
  nlfree = n-nlcstr;


  if n <= 5000                   %
    phi = n^2;                   %   phi and max_bcH are used in the calculation of
  else                           %   sigmak, the centering parameter
    phi = n^(1.5);               %   cf. Lustig, Marsten, Shanno
  end                            %



% Determine if A has full rank
%
  if rank(A) < m , error('A does not have full row-rank'); end

% If no tolerance is given use default
%
  if nargin < 9, tol = deftol;

% If no limit on iterations is given use default value
%
  if nargin < 8, maxit = defmaxit;

% If no starting-point is given compute one by default procedure
%
  if nargin < 7
    xk = A'/(A*A')*b;   % This is legitimate since we know that A has full row-rank
                        % and it's much faster than pinv(A)*b !!!

    threshold=norm(xk,1)/n*threshold_scale;
    for i = 1:min([nlcstr,nucstr])
    if uboundx(i)-lboundx(i) < 2*threshold
      xk(i) = (uboundx(i)-lboundx(i))/2;
    else
      if xk(i)-lboundx(i) < threshold
        xk(i) = lboundx(i)+threshold;
      else
        if uboundx(i)-xk(i) < threshold
          xk(i) = uboundx(i) - threshold;
        end;
      end;
    end;end;
    if nlcstr < nucstr
      for i = nlcstr+1:nucstr
        if uboundx(i) - xk(i) < threshold
          xk(i) = uboundx(i)-threshold;
        end;
      end;
    else
      for i = nucstr+1:nlcstr
        if xk(i)-lboundx(i) < threshold
          xk(i) = lboundx(i) + threshold;
        end;
      end;
    end;
  else
    xk=x0;
  end,end,end


  lambdak=zeros(m,1);
  yk=c+H*xk;
  threshold=norm(yk,1)/n*threshold_scale;
  if threshold == 0 , threshold = threshold_scale; end
  zk = yk;
  temp = (zk<=0);
  zk=abs(zk).*temp;
  yk=zk.*(~temp);
  yk(nlcstr+1:n)=zeros(nlfree,1);
  zk(nucstr+1:n)=zeros(nufree,1);
  temp = yk(1:nlcstr) < threshold;
  yk([temp;zeros(nlfree,1)]) = threshold * ones(nnz(temp),1);
  temp = zk(1:nucstr) < threshold;
  zk([temp;zeros(nufree,1)]) = threshold * ones(nnz(temp),1);


% Pick global parameters based on x0 and y0
%
  fk1 = ([xk(1:nlcstr)-lboundx;zeros(nlfree,1)]).*yk;
  fk2 = ([uboundx-xk(1:nucstr);zeros(nufree,1)]).*zk;
  objective1=sum(fk1);
  objective2=sum(fk2);
  objective=objective1+objective2;
  residual = [A*xk-b;A'*lambdak-H*xk+yk-zk-c];



  k=0;
  converged=0;

  x_feasible=0;
  yz_feasible=0;
  dxyk=zeros(3*n+m,1);               % Allocate space !!!
  M = xi*phi*max([max(max(H));b;c]); % For sigmak
  residual0 = norm(residual,1);

% Start iteration
%
while (~converged) & (k <= maxit)

  % Here the centering parameter is computed
  %
   if x_feasible & yz_feasible
     sigmak = objective / phi;
   else
      sigmak = (c'*xk+xk'*H*xk-b'*lambdak+uboundx'*zk(1:nucstr)- ...
               lboundx'*yk(1:nlcstr)+ M*(norm(residual,1)/residual0))/ phi;
   end;
  % Compute the search directions
  % dxyk = [ dxkN dxkC
  %          dykN dykC
  %          dzkN dzkC
  %           lambda  ]
  %
    Slinv=spdiags([(-lboundx+xk(1:nlcstr)).^(-1);zeros(nlfree,1)],0,n,n);
    Suinv=spdiags([( uboundx-xk(1:nucstr)).^(-1);zeros(nufree,1)],0,n,n);
    SlinvYk=spdiags(Slinv*yk,0,n,n);
    SuinvZk=spdiags(Suinv*zk,0,n,n);
    Ginv=H+SlinvYk+SuinvZk;
    if k == 0
      if condest(Ginv) > 1e12
        H_ill = 1;
        disp('Warning: H is ill-conditioned !!!');
      else
        H_ill = 0;
      end;
    end;
    if H_ill
      Ginv=Ginv+5*max(max(Ginv))*eps*speye(n);
    end
    Ginv=inv(chol(Ginv)); % This seems to be numerically more stable than
    Ginv=Ginv*Ginv';                   % inv(H+SlinvYk+SuinvZk) !
    Rchol=chol(A*Ginv*A');
    temp1=Slinv*(-fk1+sigmak*ones(n,1));
    temp2=Suinv*(-fk2+sigmak*ones(n,1));
    temp3=temp1-temp2-(-residual(m+1:m+n));
    dxyk(3*n+1:3*n+m)=Rchol\(Rchol'\(-A*Ginv*temp3+(-residual(1:m))));
    dxyk(1:n)=Ginv*(temp3+A'*dxyk(3*n+1:3*n+m,:));
    dxyk(n+1:2*n)=temp1-SlinvYk*dxyk(1:n,:);
    dxyk(2*n+1:3*n)=temp2+SuinvZk*dxyk(1:n,:);


  % Compute alphak, the step-length
  %
    minimum = min([Slinv*dxyk(1:n,1);-Suinv*dxyk(1:n,1)]);
    if minimum < 0
      alphak_p=min([(-alpha_scale/minimum);1]);
    else
      alphak_p=1;
    end;

    minimum = min([spdiags(yk(1:nlcstr).^(-1),0,nlcstr,nlcstr)* ...
                   dxyk(n+1:n+nlcstr,1); ...
                   spdiags(zk(1:nucstr).^(-1),0,nucstr,nucstr)* ...
                   dxyk(2*n+1:2*n+nucstr,1)]);
    if minimum < 0
      alphak_d=min([(-alpha_scale/minimum);1]);
    else
      alphak_d=1;
    end;

%    if alphak_p < alphak_d
%      alphak_d = alphak_d
%    else
%      alphak_p = alphak_d
%    end;


  % Compute the new iterates
  %
    xk = xk + alphak_p * dxyk(1:n,1);
    yk = yk + alphak_d * dxyk(n+1:2*n,1);
    zk = zk + alphak_d * dxyk(2*n+1:3*n,1);
    lambdak = lambdak + alphak_d * dxyk(3*n+1:3*n+m,1);

  % Increase k the iteration parameter and display it
  %
    k = k+1;
    disp([num2str(k) ' steps completed']);


  % Test sequence for convergence rate
  %
    co = (objective + max(abs(residual)))^1;


  % Update quantities
  %
    fk1 = ([xk(1:nlcstr)-lboundx;zeros(nlfree,1)]).*yk;
    fk2 = ([uboundx-xk(1:nucstr);zeros(nufree,1)]).*zk;
    objective1=sum(fk1);
    objective2=sum(fk2);
    objective=objective1+objective2;
    residual = [A*xk-b;A'*lambdak-H*xk+yk-zk-c];

  % Test sequence for convergence rate
  %
    co = objective + max(abs(residual)) / co;
    disp(['co = ' num2str(co)]);

  % Feasibility test
  %
    if  ~x_feasible & norm(residual(1:m),1)/(1+norm(xk,1))<=tol
      disp(['xk feasible after ' num2str(k) ' steps.']);
      x_feasible=1;
    end
    if ~yz_feasible & norm(residual(m+1:m+n),1)/ ...
                          (1+norm([xk;lambdak;yk;zk],1))<=tol
      disp(['yk & zk feasible after ' num2str(k) ' steps.']);
      yz_feasible=1;
    end

  % Optimality test
  %
    if (abs(objective)<tol) & norm(residual,1)/ ...
                                  (1+norm([xk;lambdak;yk;zk],1))<tol
        converged = 1;
    end
end
```

```
if k > maxit
  disp(['No solution after ' num2str(maxit) ' steps !!!']);
  disp('Last iterates were:');
else
  disp(['Solution found after ' num2str(k) ' steps.']);
end
xstar = xk; ystar = yk;  zstar=zk; lambda=lambdak;
```

A.3 Predictor-Corrector Algorithm: pc_ipm

```
function [xstar,ystar,zstar,lambda]=pc_ipm(A,H,b,c,lboundx,uboundx,x0,maxit,tol)
% [xstar,ystar,zstar,lambda]=pc_ipm(A,H,b,c,lboundx,uboundx,x0,y0,maxit,tol)
%
% Predictor-Corrector Interior Point Method Quadratic Optimization
%
% pc_ipm solves convex Quadratic Programs
%
% Minimize 1/2 x'*H*x + c'*x subject to A*x == b, lboundx <= x <= uboundx
%
% using a predictor-corrector interior point method.
%
% H must be positive semi-definite and A must have full rank.
%
% The user has the option of incorporating lower and upper bounds on x, using
% lboundx and uboundx, respectively. If lboundx or uboundx are shorter than x,
% the remainig components of x are assumed to be unbounded in the respective
% direction.
%
% A starting value x0 can be given but it has to satisfy the box-constraints.
%
% The default maximal number of iterations is 100 and can be adjusted by setting
% maxit.
%
% The default termination tolerance is 1E-7 and can be adjusted by setting tol.
%
% The routine returns the solution xstar and the lagrangian multipliers, where
% ystar is associated with lboundx, zstar is associated with uboundx and lambda
% is associated with the equality constraints.


% First set a couple of default values
%
  defmaxit = 100;            % Default value for the maximum number of iterations
  deftol   = 1e-7;           % Default tolerance for objective value and residuals

  threshold_scale = 1;       % This is used for x0 and y0
  sigma_scale=.7;            % This used as centering parameter
                             % if residual >> objective
  alpha_scale=.99995;        % Used for computation of alphak


% Determine dimensions of the problem
%
  [m n] = size(A);
  if (size(H)~=[n,n])|(size(b)~=[m,1])|(size(c)~=[n,1])| ...
     (size(uboundx,1)>n)|(size(lboundx,1)>n)
    error('Dimensions mismatch !!!');
  end

  nucstr=size(uboundx,1);    % Number of upper-constrained variables
  nufree = n-nucstr;
  nlcstr=size(lboundx,1);    % Number of lower-constrained variables
  nlfree = n-nlcstr;


  if n <= 5000                %
    phi = n^2;                %   phi is used in the calculation of
  else                        %   sigmak, the centering parameter
    phi = n^(1.5);            %   cf. Lustig, Marsten, Shanno
  end                         %


% Determine if A has full rank
%
  if rank(A) < m , error('A does not have full row-rank'); end

% If no tolerance is given use default
%
  if nargin < 9, tol = deftol;

% If no limit on iterations is given use default value
%
  if nargin < 8, maxit = defmaxit;

% If no starting-point is given compute one by default procedure
%
  if nargin < 7
    xk = A'/(A*A')*b;   % This is legitimate since we know that A has full row-rank
                        % and it's much faster than pinv(A)*b !!!

    threshold=norm(xk,1)/n*threshold_scale;
    for i = 1:min([nlcstr,nucstr])
    if uboundx(i)-lboundx(i) < 2*threshold
      xk(i) = (uboundx(i)-lboundx(i))/2;
    else
      if xk(i)-lboundx(i) < threshold
        xk(i) = lboundx(i)+threshold;
      else
        if uboundx(i)-xk(i) < threshold
          xk(i) = uboundx(i) - threshold;
        end;
      end;
    end;end;
    if nlcstr < nucstr
      for i = nlcstr+1:nucstr
        if uboundx(i) - xk(i) < threshold
          xk(i) = uboundx(i)-threshold;
        end;
      end;
    else
      for i = nucstr+1:nlcstr
```

```
        if xk(i)-lboundx(i) < threshold
          xk(i) = lboundx(i) + threshold;
        end;
      end;
    end;
  else
    xk=x0;
  end,end,end


% Compute initial values for y, z and lambda based on x
%
  lambdak=zeros(m,1);
  yk=c+H*xk;
  threshold=norm(yk,1)/n*threshold_scale;
  if threshold == 0 , threshold = threshold_scale; end
  zk = yk;
  temp = (zk<=0);
  zk=abs(zk).*temp;
  yk=zk.*(~temp);
  yk(nlcstr+1:n)=zeros(nlfree,1);
  zk(nucstr+1:n)=zeros(nufree,1);
  temp = yk(1:nlcstr) < threshold;
  yk([temp;zeros(nlfree,1)]) = threshold * ones(nnz(temp),1);
  temp = zk(1:nucstr) < threshold;
  zk([temp;zeros(nufree,1)]) = threshold * ones(nnz(temp),1);


% Pick global parameters based on x0 and y0
%
  fk1 = ([xk(1:nlcstr)-lboundx;zeros(nlfree,1)]).*yk;
  fk2 = ([uboundx-xk(1:nucstr);zeros(nufree,1)]).*zk;
  objective1=sum(fk1);
  objective2=sum(fk2);
  objective=objective1+objective2;
  residual = [A*xk-b;A'*lambdak-H*xk+yk-zk-c];


  k=0;
  converged=0;

  x_feasible=0;
  yz_feasible=0;
  dxyk=zeros(3*n+m,1);        % Allocate space !!!


% Start iteration
%
while (~converged) & (k <= maxit)

  % Compute the search directions
  % dxyk = [ dxkN dxkC
  %          dykN dykC
  %          dzkN dzkC
  %           lambda  ]
  %
    Slinv=spdiags([(-lboundx+xk(1:nlcstr)).^(-1);zeros(nlfree,1)],0,n,n);
  Suinv=spdiags([( uboundx-xk(1:nucstr)).^(-1);zeros(nufree,1)],0,n,n);
  SlinvYk=spdiags(Slinv*yk,0,n,n);
  SuinvZk=spdiags(Suinv*zk,0,n,n);
  Ginv=H+SlinvYk+SuinvZk;
  if k == 0
    if condest(Ginv) > 1e12
      H_ill = 1;
      disp('Warning: H is ill-conditioned !!!');
    else
      H_ill = 0;
    end;
  end;
  if H_ill
    Ginv=Ginv+5*max(max(Ginv))*eps*speye(n);
  end
  Ginv=inv(chol(Ginv));          % This seems to be numerically more stable than
  Ginv=Ginv*Ginv';               % inv(H+SlinvYk+SuinvZk) !
  Rchol=chol(A*Ginv*A');


% First compute the predictor step
%
  temp1=Slinv*(-fk1);
  temp2=Suinv*(-fk2);
  temp3=temp1-temp2-(-residual(m+1:m+n));
  dxyk(3*n+1:3*n+m)=Rchol\(Rchol'\(-A*(Ginv*temp3)-residual(1:m)));
  dxyk(1:n)=Ginv*(temp3+A'*dxyk(3*n+1:3*n+m));
  dxyk(n+1:2*n)=temp1-SlinvYk*dxyk(1:n);
  dxyk(2*n+1:3*n)=temp2+SuinvZk*dxyk(1:n);


% Here the centering parameter is computed
%
  if ~(x_feasible & yz_feasible) & norm(residual,1) / objective > 1e3
    sigmak = sigma_scale;
  else
    if objective < 1 & x_feasible & yz_feasible
      sigmak = objective / phi;
    else
      minimum = min([Slinv*dxyk(1:n,1);-Suinv*dxyk(1:n,1)]);
      if minimum < 0
       alphak_p=min([(-alpha_scale/minimum);1]);
      else
        alphak_p=1;
      end;

      minimum = min([spdiags(yk(1:nlcstr).^(-1),0,nlcstr,nlcstr)* ...
                    dxyk(n+1:n+nlcstr); ...
                    spdiags(zk(1:nucstr).^(-1),0,nucstr,nucstr)* ...
                    dxyk(2*n+1:2*n+nucstr)]);
      if minimum < 0
        alphak_d=min([(-alpha_scale/minimum);1]);
      else
        alphak_d=1;
      end;
      g_hat1=([xk(1:nlcstr)-lboundx+alphak_p*dxyk(1:nlcstr);...
               zeros(nlfree,1)])'* (yk+alphak_d*dxyk(n+1:2*n));
      g_hat2=([uboundx-xk(1:nucstr)-alphak_p*dxyk(1:nucstr);...
              zeros(nufree,1)])'* (zk+alphak_d*dxyk(2*n+1:3*n));
      sigmak=((g_hat1+g_hat2)/objective)^2 * ((g_hat1+g_hat2)/n);
    end;
  end;

  disp(['sigmak = ' num2str(sigmak)]);

% Now the centered corrector step
%
  temp1=Slinv*(sigmak*ones(n,1)-fk1-dxyk(1:n).*dxyk(n+1:2*n));
  temp2=Suinv*(sigmak*ones(n,1)-fk2+dxyk(1:n).*dxyk(2*n+1:3*n));
  temp3=temp1-temp2-(-residual(m+1:m+n));
  dxyk(3*n+1:3*n+m,:)=Rchol\(Rchol'\(-A*(Ginv*temp3)-residual(1:m)));
  dxyk(1:n)=Ginv*(temp3+A'*dxyk(3*n+1:3*n+m));
  dxyk(n+1:2*n)=temp1-SlinvYk*dxyk(1:n);
  dxyk(2*n+1:3*n)=temp2+SuinvZk*dxyk(1:n);


% Compute alphak, the step-length
%
  minimum = min([Slinv*dxyk(1:n,1);-Suinv*dxyk(1:n,1)]);
  if minimum < 0
    alphak_p=min([(-alpha_scale/minimum);1]);
  else
    alphak_p=1;
  end;

  minimum = min([spdiags(yk(1:nlcstr).^(-1),0,nlcstr,nlcstr)* ...
                  dxyk(n+1:n+nlcstr,1); ...
                 spdiags(zk(1:nucstr).^(-1),0,nucstr,nucstr)* ...
                  dxyk(2*n+1:2*n+nucstr,1)]);
  if minimum < 0
    alphak_d=min([(-alpha_scale/minimum);1]);
  else
    alphak_d=1;
  end;



% Compute the new iterates
%
  xk = xk + alphak_p * dxyk(1:n,1);
  yk = yk + alphak_d * dxyk(n+1:2*n,1);
  zk = zk + alphak_d * dxyk(2*n+1:3*n,1);
  lambdak = lambdak + alphak_d * dxyk(3*n+1:3*n+m,1);

% Increase k the iteration parameter and display it
%
  k = k+1;
  disp([num2str(k) ' steps completed']);


% Test sequence for convergence rate
%
  co = (objective + max(abs(residual)))^1;
  % Update quantities
  %
    fk1 = ([xk(1:nlcstr)-lboundx;zeros(nlfree,1)]).*yk;
    fk2 = ([uboundx-xk(1:nucstr);zeros(nufree,1)]).*zk;
    objective1=sum(fk1);
    objective2=sum(fk2);
    objective=objective1+objective2;
    residual = [A*xk-b;A'*lambdak-H*xk+yk-zk-c];


  % Test sequence for convergence rate
  %
    co = objective + max(abs(residual)) / co;
    disp(['co = ' num2str(co)]);

  % Feasibility test
  %
    if  ~x_feasible & norm(residual(1:m),1)/(1+norm(xk,1))<=tol
      disp(['xk feasible after ' num2str(k) ' steps.']);
      x_feasible=1;
    end
    if ~yz_feasible & norm(residual(m+1:m+n),1)/(1+norm([xk;lambdak;yk;zk],1))<=tol
      disp(['yk & zk feasible after ' num2str(k) ' steps.']);
      yz_feasible=1;
    end


  % Optimality test
  %
    if (abs(objective)< tol) & norm(residual,1)/(1+norm([xk;lambdak;yk;zk],1))<tol
        converged = 1;
    end
end

if k > maxit
  disp(['No solution after ' num2str(maxit) ' steps !!!']);
  disp('Last iterates were:');
else
  disp(['Solution found after ' num2str(k) ' steps.']);
end
xstar = xk; ystar = yk;  zstar=zk; lambda=lambdak;
```

A.4 Data Generator: `randprob`

```
function P = randprob(samples,variables,mu,sigma,r_mu,r_sigma)
% RANDPROB
%
%      P = randprob(samples,variables,mu,sigma,r_mu,r_sigma)
%
%      creates a test-matrix for portfolio optimization of the size samples by
%      variables. It imitates a matrix that contains #samples samples of
%      #variables variables, for which the expected values of the variables are
%      contained in an interval of length r_mu with its center at mu and the
%      standard-deviations of the variables are contained in an interval of
%      length r_sigma with its center at sigma.
%      It creates a different matrix each time it is invoked.


randn('seed',sum(100*clock));
P = randn(samples,variables);
Exp = rand(variables,1)*r_mu + (mu - r_mu/2) * ones(variables,1);
Std = rand(variables,1)*r_sigma + (sigma - r_sigma/2) * ones(variables,1);
P = P*diag(Std) + ones(samples,variables)*diag(Exp);
```

Bibliography

[1] Anstreicher, K. M., Y. Ye (1993) "On quadratic and $O(\sqrt{n}L)$ convergence of a predictor-corrector algorithm for LCP." Mathematical Programming 62, 537–551.

[2] Carpenter, T. (1992) *Practical Interior Point Methods for Quadratic Programming.* Dissertation, Department of Civil Engineering and Operations Research, Princeton University, Princeton.

[3] Carpenter, T., I. Lustig, J. Mulvey and D. Shanno (1993) "Higher–order predictor–corrector interior point methods with application to quadratic objectives." SIAM Journal on Opimization 3, 696–725.

[4] Carpenter, T., I. Lustig, J. Mulvey and D. Shanno (1993) "Separable quadratic programming via a primal–dual interior point method and its use in a sequential procedure." ORSA Journal on Computing 5, 182–191.

[5] El-Bakry, A.S., R.A. Tapia and Y. Zhang (1994) "A study of indicators for identifying zero variables for interior point methods." SIAM Review 36, 45–72.

[6] Elton, E. J. and M. J. Gruber (1987) *Modern Portfolio Theory and Investment Analysis (3rd Edition).* John Wiley & Sons, Inc., New York.

[7] Gill, P., W. Murray and M. Wright (1991) *Numerical linear algebra and optimization.* Addison–Wesley Publishing Company, Redwood City.

[8] Güler, O. and Y. Ye (1993) "Convergence behavior of interior–point algorithms." Mathematical Programming 60, 215–228.

[9] Ji, J., F. Potra, R. Tapia and Y. Zhang (1991) "An interior–point method for linear complementarity problems with polynomial complexity and superlinear convergence." Technical Report TR91-23, Dept. Mathematical Sciences, Rice University, Houston.

[10] Karmarkar, N. (1984) "A new polynomial-time algorithm for linear programming." Combinatorica 4, 373–395.

[11] KONNO, H. AND K. SUZUKI (1992) "A fast algorithm for solving large scale mean-variance models by compact factorization of covariance matrices." Journal of the Operations Research Society of Japan 35, 93–104.

[12] KRENGEL, U. (1988) *Einführung in die Wahrscheinlichkeitstheorie und Statistik.* Friedr. Vieweg & Sohn, Braunschweig–Wiesbaden.

[13] LUSTIG, I., R. MARSTEN AND D. SHANNO (1991) "Computational experience with a primal–dual interior point method for linear programming." Linear Algebra and its Appications 152, 191–222.

[14] LUSTIG, I., R. MARSTEN AND D. SHANNO (1992) "On implementing Mehrotra's predictor–corrector interior–point method for linear programming." SIAM Journal on Optimization 2, 435–449.

[15] LUSTIG, I., R. MARSTEN AND D. SHANNO (1994) "Interior point methods for linear programming: Computational state of the art." ORSA Journal on Computing 6, 1–14.

[16] MARKOWITZ, H. M. (1959) *Portfolio Selection: Efficient Diversification of Investments.* John Wiley & Sons, New York.

[17] MARKOWITZ, H. M. AND A. PEROLD (1981) "Sparsity and piecewise linearity in large scale portfolio optimization problems." In *Sparse Matrices and Their Uses*, 89–108, I. S. Duff (ed.), Academic Press, New York.

[18] MEHROTRA, S. (1992) "On the implementation of a primal–dual interior point method." SIAM Journal on Optimization 2, 575–601.

[19] MONTEIRO, D. C., I. ADLER (1989) "Interior path following primal–dual algorithms. Part II: Convex quadratic programming." Mathematical Programming 44, 43–66.

[20] SHARPE, W. F. (1964) "Capital Asset Prices: A Theory of Market Equilibrium under Conditions of Risk." The Journal of Finance 19, 425–442.

[21] TAKEHARA, H. (1992) "An interior point algorithm for large-scale portfolio optimization." Annals of Operations Research, to appear.

[22] THE MATHWORKS, INC. (2/1993) MATLAB *User's Guide.* The MathWorks, Inc., Natick.

[23] VANDERBEI, R. (1992) "LOQO User's Manual." Program in Statistics & Operations Research, Princeton University, Princeton.

[24] WERNER, J. (1992) *Numerische Mathematik 1/2.* Friedr. Vieweg & Sohn, Braunschweig–Wiesbaden.

[25] WERNER, J. (1993) *Nichtlineare Optimierung.* Vorlesungsskript, Institut für Numerische und Angewandte Mathematik, Georg–August–Universität, Göttingen.

[26] ZHANG, Y., R. TAPIA AND F. POTRA (1993) “On the superlinear convergence of interior point algorithms for a general class of problems.” SIAM Journal on Optimization 3, 413–422.

[27] ZHANG, Y. AND R. TAPIA (1993) “A superlinearly convergent polynomial primal–dual interior point algorithm for linear programming.” SIAM Journal on Optimization 3, 118–133.

[28] ZHANG, Y., R. TAPIA AND J. DENNIS (1992) “On the superlinear and quadratic convergence of primal–dual interior point linear programming algorithms.” SIAM Journal on Optimization 2, 304–324.

[29] ZHANG, Y. (1991) “On the Convergence of a Class of Infeasible Interior-Point Methods for the Horizontal Linear Complementarity Problem.” Research Report 92-07, Dept. of Mathematics and Statistics, University of Maryland Baltimore County, Baltimore.