Large Scale Portfolio Optimization with Piecewise Linear Transaction Costs

Marina Potaptchik* Levent Tunçel† Henry Wolkowicz‡

September 26, 2006

University of Waterloo
Department of Combinatorics & Optimization
Waterloo, Ontario N2L 3G1, Canada
Research Report CORR 2006-19

Keywords: Portfolio Optimization, Quadratic Programming, Piecewise Differentiable Functions, Separable Problems

Abstract

We consider the fundamental problem of computing an optimal portfolio based on a quadratic mean-variance model of the objective function and a given polyhedral representation of the constraints. The main departure from the classical quadratic programming formulation is the inclusion in the objective function of piecewise linear, separable functions representing the transaction costs. We handle the nonsmoothness in the objective function by using spline approximations. The problem is then solved using a primal-dual interior-point method with a crossover to an active set method. Our numerical tests show that we can solve large scale problems efficiently and accurately.

Contents

*Research supported by The Natural Sciences and Engineering Research Council of Canada. E-mail: mpotaptc@uwaterloo.ca

†Research supported in part by a Discovery Grant from NSERC. Email: ltuncel@math.uwaterloo.ca

‡Research supported by The Natural Sciences and Engineering Research Council of Canada. Email: hwolkowicz@uwaterloo.ca

List of Tables

List of Figures

1 Introduction

We consider the problem of selecting a portfolio for an investor in an optimal way. Assume that n assets are available. We denote by $x = (x_1, x_2, \ldots, x_n)^T$ the vector of proportions of the money invested in each asset. Under the mean-variance model, the investor acts to minimize the quadratic function $F(x) = \frac{1}{2}x^T Q x - td^T x$, under linear inequality constraints, i.e. we solve a quadratic program, $\boldsymbol{QP}$. Here $-d$ is the vector of the expected returns of the assets, Q is a covariance matrix and t is a fixed positive scalar parameter. In this paper we consider minimizing $f(x) = F(x) + \sum_{i=1}^{n} f_i(x_i)$, where the functions f_i are piecewise linear convex functions that represent the transaction costs. We introduce an algorithm that approximates the nondifferentiable functions with splines and then solves the resulting smooth problem using a primal-dual interior-point method. We apply a crossover technique to an active set method once we are *close enough* to the set of optimal solutions. We are able to solve large scale problems efficiently and accurately.

Transaction costs arise when an investor buys or sells some of his/her holdings. Two major sources of transaction costs are brokerage fees and market impact costs. The broker's commission rates are often decreasing in the amount of trade, and therefore the transaction costs resulting from these fees are modeled by concave functions. However, this is the case only when the amounts of transactions are not very large and should be taken into account only by smaller investors. If the trade volume is large enough, the commissions can be modeled by a linear function.

The market impact costs are the changes in the price of the assets that result from large amounts of these assets being bought or sold. The price is going up if someone is buying large quantities of an asset, and the price is going down if a lot of shares of this asset are for sale. The market impact costs are normally modeled by convex functions. The piecewise linear convex function is the most common example.

Therefore, from the point of view of a large institutional investor, transaction costs can be adequately modeled by a piecewise linear convex function.

We assume that the vector $\hat{x} = (\hat{x_1}, \hat{x_2}, \ldots, \hat{x_n})^T$ represents the current holdings of assets. The cost associated with changing the holdings in asset i from $\hat{x}_i$ to x_i will be denoted by $f_i(x_i)$. For most practical purposes, it is safe to assume that transaction costs on each asset depend only on the amount of the holdings in this asset purchased or sold and do not depend on the amount of transactions in other assets. Therefore, we model the transaction costs by a separable function of the amount sold or bought, i.e. the cost associated with changing the portfolio from $\hat{x}$ to x is $\sum_{i=1}^{n} f_i(x_i)$. We discuss the transaction cost model in more detail in Appendix A, page 37.

See [17, 14, 15, 8, 10, 19] for work related to portfolio optimization under nonsmooth transaction costs. And also [1, 3, 4, 6, 11, 12, 20, 21] for approaches to partially separable optimization

problems. For an approach that replaces a nondifferentiable problem by a smooth problem, see [13].

1.1 Outline

In Section 2 we present the details of the problem as well as the associated duality and optimality conditions. The smoothing by splines is done in Section 3. We include sensitivity analysis in Section 3.3. In particular, this section proves that more accurate spline approximations yield more accurate solutions of the original problem, i.e. continuity of the spline approximations. An alternative approach that replaces the nondifferentiability with additional variables and constraints is given in Section 4. For this approach, we use the software package MOSEK to solve the resulting $\boldsymbol{QP}$.

In Section 5 we study the expected number of variables x_i that have values at points of nondifferentiability. These theoretical observations agree with our empirical results.

Computational results are reported in Section 6 and concluding remarks are given in Section 7.

2 Problem Formulation and Optimality Conditions

2.1 Formulation

We consider the problem of minimization of the function $f(x)$ subject to linear inequality constraints.

$$(P) \qquad \begin{array}{ll} \min & f(x) \\ \text{s.t.} & Ax \leq b, \end{array} \tag{2.1}$$

where A is an $m \times n$-matrix and $b \in \mathbb{R}^m$. The objective function $f(x)$ is defined as follows:

$$f(x) := F(x) + \sum_{i=1}^{n} f_i(x_i), \tag{2.2}$$

where

$$F(x) := \frac{1}{2} x^T G x + c^T x \tag{2.3}$$

is a strictly convex quadratic function on $\mathbb{R}^n$, G is symmetric positive definite, and $f_i(x_i)$ is a piecewise linear function on $\mathbb{R}$, with break-points at d_{ik}, i.e.

$$f_i(x_i) := \begin{cases} f_{i0} := p_{i0} x_i + h_{i0}, & \text{if} \quad x_i \leq d_{i1}, \\ f_{il} := p_{il} x_i + h_{il}, & \text{if} \quad d_{il} \leq x_i \leq d_{il+1},\ l = 1, \ldots, M_i, \\ f_{iM_i} := p_{iM_i} x_i + h_{iM_i}, & \text{if} \quad x_i \geq d_{iM_i}. \end{cases} \tag{2.4}$$

Let the feasible region of the problem (2.1) be denoted by S; and, for each $x \in \mathbb{R}^n$, let the set of *active breakpoints*, and its complement, be denoted by

$$E(x) := \{i : x_i = d_{il} \text{ for some } l \in \{1, .., M_i\}\}, \quad N(x) = \{1, \ldots, n\} \backslash E(x). \tag{2.5}$$

2.2 Duality and Optimality

The Lagrangian dual of (P) is

$$\max_{u \geq 0} \min_{x} L(x, u) := f(x) + u^T(Ax - b).$$

The inner-minimization is an unconstrained convex minimization problem. Therefore, we can write down the Wolfe dual program

$$\begin{array}{lll} & \max & L(x,u) \\ (D) & \text{s.t.} & 0 \in \partial_x L(x,u), \\ & & u \geq 0, \end{array} \tag{2.6}$$

where $\partial_x L(x, u)$ denotes the subgradient of L, i.e.

$$\partial_x L(x, u) = \{\phi \in \mathbb{R}^n : \phi^T(y - x) \leq L(y, u) - L(x, u), \quad \forall y \in \mathbb{R}^n\}.$$

We can now state the well-known optimality conditions.

Theorem 2.1 *A point $x \in \mathbb{R}^n$ minimizes f over S if and only if the following system holds*

$$\begin{array}{cc} u \geq 0, \quad 0 \in \partial_x L(x,u) & \textit{dual feasibility} \\ Ax \leq b & \textit{primal feasibility} \\ u^T(Ax - b) = 0 & \textit{complementary slackness.} \end{array}$$

■

To further simplify the optimality conditions, we use the following property of subgradients:

Proposition 2.1 ([2]) *Let $\theta = \sum_{i=1}^m \theta_i$, where $\theta_i : \mathbb{R}^n \to \mathbb{R}$ are convex functions, $i = 1, \ldots, m$. Then $\partial\theta(x)$ is equal to the Minkowski sum*

$$\partial\theta(x) = \sum_{i=1}^{m} \partial\theta_i(x).$$

Recall that

$$L(x,u) := F(x) + \sum_{i=1}^{n} f_i(x_i) + u^T(Ax - b).$$

Since the first and the last terms of this sum are differentiable, we get

$$\partial L(x,u) = \nabla F(x) + \partial(\sum_{i=1}^{n} f_i(x_i)) + A^T u.$$

It follows from the definition of $f_i(x_i)$ that

$$\partial f_i(x_i) = \begin{cases} \frac{df_{i0}}{dx_i}(x_i), & \text{if} \quad x_i < d_{i1}, \\ \frac{df_{il}}{dx_i}(x_i), & \text{if} \quad d_{il} < x_i < d_{il+1}, \ \ l = 1, ..., M_i, \\ [\frac{df_{il-1}}{dx_i(x_i)}, \frac{df_{il}}{dx_i}(x_i)] & \text{if} \quad x_i = d_{il}, \ \ l = 1, ..., M_i, \\ \frac{df_{iM_i}}{dx_i}(x_i), & \text{if} \quad x_i > d_{iM_I}. \end{cases}$$

We can think of f_i as a function from $\mathbb{R}^n$ to $\mathbb{R}$, defined as $f_i(x) = f_i(x_i)$. Then $\partial f_i(x) = \partial f_i(x_i)e_i$ and $\sum_{i=1}^n f_i(x_i) = \sum_{i=1}^n f_i(x)$. By Proposition 2.1, $\partial(\sum_{i=1}^n f_i(x_i))$ is equal to the Minkowski sum $\sum_{i=1}^n \partial f_i(x_i)e_i$. From the definition of the Minkowski sum, the latter sum is equal to a direct product of $\partial f_i(x_i)$. Therefore,

$$0 \in \partial L(x,u)$$

if and only if, for every $i = 1, \ldots, n$,

$$0 \in (\nabla F(x))_i + \partial f_i(x_i) + (A^T u)_i.$$

This allows us to reformulate the optimality conditions for (2.1).

Corollary 2.1 *Let the function f be given by (2.2). A point $x \in \mathbb{R}^n$ minimizes f over S, if and only if there exists $u \in \mathbb{R}^m$, $v \in \mathbb{R}^{E(x)}$ such that*

$$\left.\begin{array}{ll} Ax \le b, & \\ (\nabla F(x))_i + \frac{df_{il}}{dx_i}(x_i) + (A^T u)_i = 0, & \text{for all } i \in N(x), \\ & \text{with } x_i \in (d_{il}, d_{il+1}), \\ (\nabla F(x))_i + \frac{df_{il-1}}{dx_i}(d_{il}) + (A^T u)_i + v_i = 0, & \text{for all } i \in E(x), \\ & \text{with } x_i = d_{il}, \\ 0 \le v_i \le \frac{df_{il}}{dx_i}(d_{il}) - \frac{df_{il-1}}{dx_i}(d_{il}), & \text{for all } i \in E(x), \\ & \text{with } x_i = d_{il}, \\ u \ge 0, & \\ u^T(Ax - b) = 0. & \end{array}\right\} \qquad (2.7)$$

We would like to see if an interior-point method (***IPM***) can be applied to solve this problem efficiently. Applying the ***IPM*** directly to the nondifferentiable problem would force us to follow a nondifferentiable "central path".

A point $x \in \mathbb{R}^n$ is a point on the *central path*, if and only if, for $\mu > 0$,

$$\left.\begin{array}{ll} Ax + s = b,\ s \geq 0 & \\ (\nabla F(x))_i + \frac{df_{il}}{dx_i}(x_i) + (A^T u)_i = 0, & \text{for all } i \in N(x), \\ & \text{with } x_i \in (d_{il}, d_{il+1}), \\ (\nabla F(x))_i + \frac{df_{il-1}}{dx_i}(d_{il}) + (A^T u)_i + v_i = 0, & \text{for all } i \in E(x), \\ & \text{with } x_i = d_{il}, \\ 0 \leq v_i \leq \frac{df_{il}}{dx_i}(d_{il}) - \frac{df_{il-1}}{dx_i}(d_{il}), & \text{for all } i \in E(x), \\ & \text{with } x_i = d_{il}, \\ u \geq 0, & \\ u_i s_i = \mu,\ i = 1, \ldots, m, & \end{array}\right\} \tag{2.8}$$

or

$$\left.\begin{array}{ll} Ax + s = b,\ s \geq 0 & \\ (Gx)_i + c_i + p_{il} + (A^T u)_i = 0, & \text{for all } i \in N(x), \\ & \text{with } x_i \in (d_{il}, d_{il+1}), \\ (Gx)_i + c_i + p_{il-1} + (A^T u)_i + v_i = 0, & \text{for all } i \in E(x), \\ & \text{with } x_i = d_{il}, \\ 0 \leq v_i \leq p_{il} - p_{il-1}, & \text{for all } i \in E(x), \\ & \text{with } x_i = d_{il}, \\ u \geq 0, & \\ u_i s_i = \mu,\ i = 1, \ldots, m. & \end{array}\right\} \tag{2.9}$$

3 Smoothing via Splines

Approximating the nondifferentiable functions $f_i(x_i)$ by smooth functions allows us to fully use the theory of differentiable optimization, and in particular, interior-point methods.

There are two approaches that could be taken here. In many cases the nondifferentiable function $f_i(x_i)$ is just an approximation of some smooth function based on a given data set. Then, using a convex cubic spline on this data set would give a better approximation of the original function, and the objective of the optimization problem (2.1) would become smooth. For more details on this approach, see for example [18]; and for a general reference on splines, see [7].

However, in real life, the original data set is often not available and it is best to find a solution to the given data. Transaction cost functions, for example, are always non-smooth. Therefore, in this paper, we focus on the second approach. We use smooth convex splines that approximate the given piecewise linear convex functions $f_i(x_i)$ that represent the transaction costs.

3.1 Interior Point Method for Smooth Approximations

Suppose the functions $f_i(x_i)$ are approximated by smooth functions $\bar{f}_i(x_i)$. We denote the first and second derivatives by $\bar{f}_i'(x_i)$ and $\bar{f}_i''(x_i)$, respectively.

Let (x, u, s) be a current iterate, with $(u, s) > 0$. The following system of perturbed optimality conditions will be considered at each iteration of the ***IPM***

$$\left.\begin{array}{l} Ax + s = b,\ s > 0 \\ (Gx)_i + c_i + \bar{f}_i'(x_i) + (A^T u)_i = 0, \quad \text{for all } i = 1, \ldots, n, \\ u > 0, \\ u_i s_i = \mu,\ i = 1, \ldots, m. \end{array}\right\} \tag{3.1}$$

Given x, s, and u, we define the barrier parameter $\mu := \frac{u^T s}{m}$, the vector of first derivatives $g = (\bar{f}_1'(x_1), \ldots, \bar{f}_n'(x_n))^T$, and the diagonal matrices

$$U := \mathrm{Diag}(u_1, \ldots, u_m), \quad S := \mathrm{Diag}(s_1, \ldots, s_m), \quad H := \mathrm{Diag}(\bar{f}_1''(x_1), \ldots, \bar{f}_n''(x_n)).$$

Then the search direction for (3.1) is found from the linearized system (Newton's equation)

$$\begin{bmatrix} G + H & A^T & 0 \\ A & 0 & I \\ 0 & S & U \end{bmatrix} \begin{pmatrix} \Delta x \\ \Delta u \\ \Delta s \end{pmatrix} = \begin{bmatrix} -r_c \\ -r_b \\ -Us + \sigma\mu e \end{bmatrix}, \tag{3.2}$$

where the residuals

$$r_c := Gx + c + g + A^T u, \quad r_b := Ax + s - b,$$

e is the vector of ones, and $\sigma \in [0, 1]$ is the centering parameter.

We can use block eliminations to simplify the linearized system. We first solve

$$\Delta s = -U^{-1} S \Delta u - s + \sigma\mu U^{-1} e.$$

Then we can eliminate Δs and rewrite (3.2) as the symmetric, indefinite, linear system ($n + m$ sized, augmented or quasidefinite system)

$$\begin{bmatrix} G + H & A^T \\ A & -U^{-1} S \end{bmatrix} \begin{pmatrix} \Delta x \\ \Delta u \end{pmatrix} = \begin{bmatrix} -r_c \\ -r_b + s - \sigma\mu U^{-1} e \end{bmatrix}. \tag{3.3}$$

Further, since

$$\Delta u = S^{-1}U[A\Delta x + r_b - s + \sigma\mu U^{-1}e] = -u + S^{-1}[U(A\Delta x + r_b) + \sigma\mu e],$$

we can eliminate Δu and obtain the (n sized, normal equation system)

$$\begin{aligned}[G + H + A^T(S^{-1}U)A]\Delta x &= -r_c + A^T(S^{-1}U)[-r_b + s - \sigma\mu U^{-1}e] \\ &= -r_c + A^T[u - S^{-1}(Ur_b + \sigma\mu e)] \\ &= -(Gx + c + g) - A^T S^{-1}[Ur_b + \sigma\mu e].\end{aligned} \tag{3.4}$$

We can add upper and lower bounds $b_l \le x \le b_u$ to the problem. Let $0 < (x, u, u_l, u_u, s, s_l, s_u)$ be a current iterate of the ***IPM*** , where u_l, u_u and s_l, s_u are the dual variables and slack variables corresponding to the upper and lower bounds, respectively. In addition, we redefine the barrier parameter $\mu := \frac{u^T s + u_l^T s_l + u_u^T s_u}{m+2n}$, and define the diagonal matrices

$$U_l = \mathrm{Diag}(u_l),\ U_u = \mathrm{Diag}(u_u),\ S_l = \mathrm{Diag}(s_l),\ S_u = \mathrm{Diag}(s_u).$$

Then the search direction is now found from the Newton equation

$$\begin{bmatrix} G+H & A^T & I & -I \\ A & -U^{-1}S & 0 & \\ I & 0 & -U_u^{-1}S_u & 0 \\ -I & 0 & 0 & -U_l^{-1}S_l \end{bmatrix} \begin{pmatrix} \Delta x \\ \Delta u \\ \Delta u_u \\ \Delta u_l \end{pmatrix} = \begin{bmatrix} -r_c \\ -r_b + s - \sigma\mu U^{-1}e \\ -r_{bu} + s_u - \sigma\mu U_u^{-1}e \\ -r_{bl} + s_l - \sigma\mu U_l^{-1}e \end{bmatrix}, \tag{3.5}$$

with residuals $r_c := Gx + c + g + A^T u$, $r_b := Ax + s - b$, $r_{bu} := x - b_u$, $r_{bl} = -x + b_l$. We can repeat the block eliminations and find

$$\begin{aligned}
\Delta s &= -U^{-1}S\Delta u - s + \sigma\mu U^{-1}e, \\
\Delta s_u &= -U_u^{-1}S_u\Delta u_u - s_u + \sigma\mu U_u^{-1}e, \\
\Delta s_l &= -U_l^{-1}S_l\Delta u_l - s_l + \sigma\mu U_l^{-1}e, \\
\Delta u_u &= S_u^{-1}U_u[\Delta x + r_{bu} - s_u + \sigma\mu U_u^{-1}e] = -u_u + S_u^{-1}[U_u(\Delta x + r_{bu}) + \sigma\mu e], \\
\Delta u_l &= S_l^{-1}U_l[-\Delta x + r_{bl} - s_l + \sigma\mu U_l^{-1}e] = -u_l + S_l^{-1}[U_l(-\Delta x + r_{bl}) + \sigma\mu e], \\
\Delta u &= S^{-1}U[A\Delta x + r_b - s + \sigma\mu U^{-1}e] = -u + S^{-1}[U(A\Delta x + r_b) + \sigma\mu e].
\end{aligned}$$

The linearized systems become:

$$\begin{bmatrix} G + H + U_u^{-1}S_u + U_l^{-1}S_l & A^T \\ A & -U^{-1}S \end{bmatrix} \begin{pmatrix} \Delta x \\ \Delta u \end{pmatrix} = \begin{bmatrix} r_0 \\ -r_b + s - \sigma\mu U^{-1}e \end{bmatrix}, \tag{3.6}$$

where

$$r_0 = -r_c + u_u - S_u^{-1}[U_u r_{bu} + \sigma\mu e] + u_l - S_l^{-1}[U_l r_{bl} + \sigma\mu e],$$

and

$$\begin{aligned}[G + H + U_u^{-1}S_u + U_l^{-1}S_l + A^T(S^{-1}U)A]\Delta x &= -r_0 + A^T(S^{-1}U)[-r_b + s - \sigma\mu U^{-1}e] \\ &= -r_0 + A^T[u - S^{-1}(Ur_b + \sigma\mu e)].\end{aligned} \tag{3.7}$$

3.2 Quadratic and Cubic Splines

Recall that $f_i(x_i)$ is a piecewise linear convex function. We can approximate it by a spline $\bar{f}_i(x_i, \epsilon)$ in the following way. We let

$$\bar{f}_i(x_i, \epsilon) := f_i(x_i) + s_i(x_i, \epsilon). \tag{3.8}$$

Let us denote $\Delta p_{il} := p_{il} - p_{il-1}$.

For the quadratic spline,

$$s_i(x_i, \epsilon) := \begin{cases} \frac{\Delta p_{il}}{4\epsilon}(x_i - d_{il} + \epsilon)^2, & \text{if } x_i \in [d_{il} - \epsilon, d_{il} + \epsilon] \text{ for some } l \in \{1, \ldots, M_i\}, \\ 0 & \text{otherwise,} \end{cases} \tag{3.9}$$

the first partial derivative of $s_i(x_i, \epsilon)$ with respect to x_i is

$$\frac{\partial s_i(x_i, \epsilon)}{\partial x_i} = \begin{cases} \frac{\Delta p_{il}}{2\epsilon}(x_i - d_{il} + \epsilon), & \text{if } x_i \in [d_{il} - \epsilon, d_{il} + \epsilon] \text{ for some } l \in \{1, \ldots, M_i\}, \\ 0 & \text{otherwise,} \end{cases} \tag{3.10}$$

and the second partial derivative of $s_i(x_i, \epsilon)$ with respect to x_i is

$$\frac{\partial^2 s_i(x_i, \epsilon)}{\partial x_i^2} = \begin{cases} \frac{\Delta p_{il}}{2\epsilon}, & \text{if } x_i \in [d_{il} - \epsilon, d_{il} + \epsilon] \text{ for some } l \in \{1, \ldots, M_i\}, \\ 0 & \text{otherwise.} \end{cases} \tag{3.11}$$

For the cubic spline,

$$s_i(x_i, \epsilon) := \begin{cases} \frac{\Delta p_{il}}{6\epsilon^2}(x_i - d_{il} + \epsilon)^3, & \text{if } x_i \in [d_{il} - \epsilon, d_{il}] \text{ for some } l \in \{1, \ldots, M_i\}, \\ -\frac{\Delta p_{il}}{6\epsilon^2}(x_i - d_{il} - \epsilon)^3 + (\Delta p_{il})x_i, & \text{if } x_i \in [d_{il}, d_{il} + \epsilon] \text{ for some } l \in \{1, \ldots, M_i\} \\ 0 & \text{otherwise,} \end{cases} \tag{3.12}$$

the first partial derivative of $s_i(x_i, \epsilon)$ with respect to x_i is

$$\frac{\partial s_i(x_i, \epsilon)}{\partial x_i} = \begin{cases} \frac{\Delta p_{il}}{2\epsilon^2}(x_i - d_{il} + \epsilon)^2, & \text{if } x_i \in [d_{il} - \epsilon, d_{il}] \text{ for some } l \in \{1, \ldots, M_i\}, \\ -\frac{\Delta p_{il}}{2\epsilon^2}(x_i - d_{il} - \epsilon)^2 + \Delta p_{il}, & \text{if } x_i \in [d_{il}, d_{il} + \epsilon] \text{ for some } l \in \{1, \ldots, M_i\}, \\ 0 & \text{otherwise,} \end{cases} \tag{3.13}$$

and the second partial derivative of $s_i(x_i, \epsilon)$ with respect to x_i is

$$\frac{\partial^2 s_i(x_i, \epsilon)}{\partial x_i^2} = \begin{cases} \frac{\Delta p_{il}}{\epsilon^2}(x_i - d_{il} + \epsilon), & \text{if } x_i \in [d_{il} - \epsilon, d_{il}] \text{ for some } l \in \{1, \ldots, M_i\}, \\ -\frac{\Delta p_{il}}{\epsilon^2}(x_i - d_{il} - \epsilon), & \text{if } x_i \in [d_{il}, d_{il} + \epsilon] \text{ for some } l \in \{1, \ldots, M_i\}, \\ 0 & \text{otherwise.} \end{cases} \tag{3.14}$$

It is trivial to check that the functions $\bar{f}_i(x_i, \epsilon)$ defined above are continuously differentiable. In case of the cubic spline, they are twice continuously differentiable. Note that the largest value of ϵ that we want to use should satisfy

$$\epsilon < \frac{1}{2} \min_{i,l}(d_{il} - d_{il-1}).$$

Let us define

$$\bar{\epsilon} := \frac{1}{3} \min_{i,l}(d_{il} - d_{il-1}). \tag{3.15}$$

3.3 Sensitivity Analysis

First, we recall some basic sensitivity results, see [9].

Definition 3.1 *Let Γ be a point-to-set mapping from $T \subset \mathbb{R}^n$ to subsets of $\mathbb{R}^m$, and let $t_n \in T$ be such that $t_n \to t_0$. Then Γ is closed at $t_0 \in T$, if $x_n \in \Gamma(t_n)$, for each n, and $x_n \to x_0$ together imply that $x_0 \in \Gamma(t_0)$.*

Theorem 3.1 *(2.2.2 from [9]) If f is a continuous real-valued function defined on the space $\mathbb{R}^m \times T$ and R is a continuous mapping of T into $\mathbb{R}^m$ such that $R(\epsilon) \neq \emptyset$ for each $\epsilon \in T$, then the (real-valued) function f^* defined by*

$$f^*(\epsilon) = \inf\{f(x, \epsilon) \mid x \in R(\epsilon)\} \tag{3.16}$$

is continuous on T.

If g is an affine function from $\mathbb{R}^n$ to $\mathbb{R}^m$, i.e., if $g(x) = Ax + b$, where A is an $m \times n$ constant matrix and $b \in \mathbb{R}^m$ is a constant vector, and if

$$R_M(g) = \{x \in M \mid g(x) \geq 0\},$$

where $M \subset \mathbb{R}^n$, the function g is said to be *non-degenerate* with respect to the set M if $R_M(g)$ has a non-empty interior and no component of g is identically zero on M. The continuity of the map defined by

$$S_M(g) = \{x \in R_M(g) \mid f(x) = \inf_z\{f(z) \mid z \in R_M(g)\}\} \tag{3.17}$$

is given by the following theorem. This result has been proven in [5].

Theorem 3.2 *(2.2.4 from [9]) If f is continuous, g is affine, and M is closed and convex, then S_M is closed at every non-degenerate g.*

We next apply these sensitivity results to our problem. Let us introduce a function $f_i(x_i, \epsilon)$ defined on $\mathbb{R} \times [0, \bar{\epsilon}]$, for every $i = 1, \ldots, n$, as follows

$$f_i(x_i, \epsilon) := \begin{cases} \bar{f}_i(x_i, \epsilon) & \text{if } \epsilon > 0, \\ f_i(x_i) & \text{if } \epsilon = 0, \end{cases} \tag{3.18}$$

where the functions $\bar{f}_i(x_i, \epsilon)$ are defined by (3.8) and the functions $s_i(x, \epsilon)$ are defined by (3.9) or (3.12). Finally, let

$$f(x, \epsilon) = F(x) + \sum_{i=1}^{n} f_i(x_i, \epsilon), \tag{3.19}$$

and consider the problem

$$\begin{array}{ll} \min & f(x) \\ \text{s.t.} & Ax \leq b. \end{array} \tag{3.20}$$

Proposition 3.1 *Let $f(x, \epsilon)$ be a function defined on $\mathbb{R}^n \times [0, \bar{\epsilon}]$ by (3.19), and let $\bar{\epsilon} > 0$ be given by (3.15). Then the optimal value function $f^*(\epsilon)$, defined by*

$$f^*(\epsilon) = \min\{f(x, \epsilon) \mid Ax \leq b\}, \tag{3.21}$$

is continuous on $[0, \bar{\epsilon}]$. Furthermore, the mapping S, defined by

$$S(\epsilon) = \{x \mid Ax \leq b,\ f(x, \epsilon) = f^*(\epsilon)\}, \tag{3.22}$$

is a closed mapping of $[0, \bar{\epsilon}]$ into $\mathbb{R}^n$.

Proof. We first show that $f_i(x_i, \epsilon)$ is continuous on $\mathbb{R} \times [0, \bar{\epsilon}]$ for every $i = 1, \ldots, n$. This is true for $\epsilon \neq 0$ because $\bar{f}_i(x_i, \epsilon)$ is a continuous function.

Let us show the continuity of $f_i(x_i, \epsilon)$ at $\epsilon = 0$. Consider a sequence $(x^k, \epsilon^k) \to (\bar{x}, 0)$. Suppose, first that $\bar{x}_i \neq d_{il}$ for any $l = 1, \ldots, M_i$. Then, for a sufficiently small value of ϵ^k, $x_i^k \notin [d_{il} - \epsilon_k, d_{il} + \epsilon_k]$ for any $l = 1, \ldots, M_i$. By definition of $f_i(x_i, \epsilon)$, $f_i(x_i^k, \epsilon^k) = f_i(x_i^k)$ and $f_i(\bar{x}_i) = f_i(\bar{x}_i, 0)$. Since $f_i(x_i)$ is continuous, $f_i(x_i^k) \to f_i(\bar{x}_i)$. Combining these, we obtain that $f_i(x_i^k, \epsilon^k) = f_i(x_i^k) \to f_i(\bar{x}_i) = f_i(\bar{x}_i, 0)$, or $f_i(x_i^k, \epsilon^k) \to f_i(\bar{x}_i, 0)$.

Suppose, next that $\bar{x}_i = d_{il}$ for some $l = 1, \ldots, M_i$. Then, for a sufficiently small value of ϵ^k, $x_i^k \in (d_{il-1} + \epsilon^k, d_{il+1} - \epsilon^k)$. We now subdivide (x^k, ϵ^k) into two subsequences. If $x_i^k \notin$

$[d_{il} - \epsilon^k, d_{il} + \epsilon^k]$, then $f_i(x_i^k, \epsilon^k) = f_i(x_i^k) \to f_i(\bar{x}_i) = f_i(\bar{x}_i, 0)$ since $f_i(x_i)$ is continuous. If $x_i^k \in [d_{il} - \epsilon^k, d_{il} + \epsilon^k]$, then, in the case of a quadratic spline,

$$s_i(x_i^k) = \frac{\Delta p_{il}}{4\epsilon^k}(x_i^k - d_{il} + \epsilon^k)^2 \leq \frac{\Delta p_{il}}{4\epsilon^k}(2\epsilon^k)^2 \to 0, \tag{3.23}$$

when $\epsilon^k \to 0$. In the case of a cubic spline,

$$s_i(x_i^k) = \frac{\Delta p_{il}}{6(\epsilon^k)^2}(x_i^k - d_{il} + \epsilon^k)^3 \leq \frac{\Delta p_{il}}{6(\epsilon^k)^2}(2\epsilon^k)^3 \to 0, \tag{3.24}$$

when $\epsilon^k \to 0$. Therefore, $f_i(x_i^k, \epsilon^k) = f_i(x_i^k) + s_i(x_i^k) \to f_i(\bar{x}_i) = f_i(\bar{x}_i, 0)$.

This completes the proof of the fact that $f_i(x_i, \epsilon)$ is continuous for every $i = 1, \ldots, n$. Hence $f_i(x, \epsilon)$ is continuous as a sum of continuous functions.

Applying Theorem 3.1, we obtain (3.21).

We next reformulate the problem

$$\min\{f(x, \epsilon) \mid Ax \leq b\}$$

as

$$\min\{f(x, x_{n+1}) \mid Ax \leq b,\ x_{n+1} = \epsilon\}.$$

This allows us to apply Theorem 3.2, and proves (3.22). ■

By Proposition 3.1, approximating the piecewise linear functions $f_i(x_i)$ by smooth splines in the ϵ-neighborhood of the break-points for small values of ϵ implies small changes in the optimal solution and the optimal value of the problem.

Suppose (x^*, s^*, u^*, v^*) satisfy optimality conditions (2.7), i.e. $x^* \in S^*(0)$. From the optimality conditions (2.7),

$$\left.\begin{array}{ll}
(\nabla F(x^*))_i + p_{il} + (A^T u^*)_i = 0, & \text{for all } i \in N(x^*), \\
 & \text{with } x_i^* \in (d_{il}, d_{il+1}), \\
(\nabla F(x^*))_i + p_{il-1} + (A^T u^*)_i + v_i^* = 0, & \text{for all } i \in E(x^*), \\
 & \text{with } x_i^* = d_{il}, \\
Ax^* + s^* = b,\ s^* \geq 0 & \\
0 \leq v_i^* \leq p_{il} - p_{il-1}, & \text{for all } i \in E(x^*), \\
 & \text{with } {}^*x_i = d_{il}, \\
u^* \geq 0, & \\
u_i^* s_i^* = 0,\ i = 1, \ldots, m. &
\end{array}\right\} \tag{3.25}$$

Suppose next that $(x^* + \Delta x, s^* + \Delta s, u^* + \Delta u)$ are optimal for the problem (3.20) for a given $\epsilon > 0$, i.e. $x^* \in S^*(\epsilon)$. Let us denote $f_\epsilon(x) = f(x, \epsilon)$. The optimality conditions for this problem imply

$$\left.\begin{array}{l} \nabla f_\epsilon(x^* + \Delta x) + A^T(u^* + \Delta u) = 0, \\ A(x^* + \Delta x) + (s^* + \Delta s) = b,\ s^* + \Delta s \geq 0, \\ (u^* + \Delta u) \geq 0, \\ (u_i^* + \Delta u_i)(s_i^* + \Delta s_i) = 0,\ i = 1, \ldots, m. \end{array}\right\} \tag{3.26}$$

Approximating the gradient of $f_\epsilon(x)$ by it's Taylor series, we obtain

$$\left.\begin{array}{l} \nabla f_\epsilon(x^*) + \nabla f_\epsilon^2(x^*)\Delta x + A^T(u^* + \Delta u) = 0, \\ A(x^* + \Delta x) + (s^* + \Delta s) = b,\ s^* + \Delta s \geq 0, \\ (u^* + \Delta u) \geq 0, \\ (u_i^* + \Delta u_i)(s_i^* + \Delta s_i) = 0,\ i = 1, \ldots, m, \end{array}\right\} \tag{3.27}$$

or

$$\left.\begin{array}{ll} (\nabla F(x^*))_i + p_{il-1} + (\nabla s_\epsilon(x^*))_i + (\nabla F^2(x^*)\Delta x)_i & \text{for all } i \in E(x^*), \\ \quad +(\nabla s_\epsilon^2(x^*)\Delta x)_i + (A^T(u^* + \Delta u))_i = 0, & \text{with } x_i^* = d_{il}, \\ (\nabla F(x^*))_i + p_{il-1} + (\nabla F^2(x^*)\Delta x)_i + (A^T(u^* + \Delta u))_i = 0, & \text{for all } i \in N(x^*), \\ & \text{with } x_i^* \in [d_{il-1}, d_{il}], \\ A(x^* + \Delta x) + (s^* + \Delta s) = b,\ s^* + \Delta s \geq 0, & \\ (u^* + \Delta u) \geq 0, & \\ (u_i^* + \Delta u_i)(s_i^* + \Delta s_i) = 0,\ i = 1, \ldots, m. & \end{array}\right\} \tag{3.28}$$

Using systems (3.25) and (3.28), we get

$$\left.\begin{array}{ll} (\nabla s_\epsilon(x^*))_i + (\nabla F^2(x^*)\Delta x)_i + (\nabla s_\epsilon^2(x^*)\Delta x)_i - v_i & \text{for all } i \in E(x^*), \\ +(A^T(\Delta u))_i = 0, & \text{with } x_i^* = d_{il}, \\ (\nabla F^2(x^*)\Delta x)_i + (A^T(\Delta u))_i = 0, & \text{for all } i \in N(x^*), \\ & \text{with } x_i^* \in [d_{il-1}, d_{il}], \\ A\Delta x + \Delta s = 0, & \\ u_i^*\Delta s_i + s_i^*\Delta u_i + \Delta s_i \Delta u_i = 0,\ i = 1, \ldots, m. & \end{array}\right\} \tag{3.29}$$

If we disregard the last term of the last equation $\Delta s_i \Delta u_i$, we can obtain $(\Delta x, \Delta s, \Delta u)$ by solving a linear system

$$\begin{bmatrix} \nabla F^2(x^*) + H & A^T & 0 \\ A & 0 & I \\ 0 & S & U \end{bmatrix} \begin{bmatrix} \Delta x \\ \Delta u \\ \Delta s \end{bmatrix} = \begin{bmatrix} r \\ 0 \\ 0 \end{bmatrix}, \tag{3.30}$$

where

$$r_i = \begin{cases} v_i - \nabla s_\epsilon(x^*)_i & \text{if} \quad x_i^* = d_{il}, \\ 0, & \text{otherwise}, \end{cases} \tag{3.31}$$

and H is a diagonal matrix

$$H_{ii} = \begin{cases} \nabla s_\epsilon^2(x^*)_i & \text{if} \quad x_i^* = d_{il}, \\ 0, & \text{otherwise}. \end{cases} \tag{3.32}$$

This proves the following:

Theorem 3.3 *Let $\epsilon > 0$ and $x(\epsilon)$ be an optimal solution for problem(3.20) and let $s(\epsilon)$ and $u(\epsilon)$ be the slack variables and dual variables associated with this optimal solution. Then there exists an optimal solution x^* for problem(2.1) such that the first-order approximation $(x(\epsilon), s(\epsilon), u(\epsilon))$ in a neighborhood of $\epsilon = 0$ is given by*

$$(x(\epsilon), s(\epsilon), u(\epsilon)) = (x^*, s^*, u^*) + (\Delta x, \Delta s, \Delta u) + o(\epsilon),$$

where s^* and u^* are the slack variables and dual variables associated with x^*, and $(\Delta x, \Delta s, \Delta u)$ can be found from (3.30).

■

Note that the LHS of the system (3.30) is similar to (3.2).

If we assume strict complementarity at x^*, the last line of (3.30) implies that the active set will not change for small values of ϵ. Let us denote by $\bar{A}$ the matrix of the constraints active at x^*. Then the system (3.30) can be rewritten as follows:

$$\begin{bmatrix} \nabla F^2(x^*) + H & \bar{A}^T \\ \bar{A} & 0 \end{bmatrix} \begin{bmatrix} \Delta x \\ \Delta u \end{bmatrix} = \begin{bmatrix} r \\ 0 \end{bmatrix}. \tag{3.33}$$

We further note that the norm of H is equal to a constant multiple of $\max_{i,l} \frac{\Delta p_{il}}{\epsilon}$ for both quadratic and cubic splines. For small values of ϵ, it can be used as an estimate of the norm of the LHS matrix of (3.33). Also, $0 \leq r_i \leq \max_l \Delta p_{il}$. Hence, we expect that the norm of Δx will be order ϵ. This is consistent with our computational experiments.

4 Smooth Formulations via Lifting

4.1 Global Lifting

Because of the special structure of the nondifferentiable part of the objective function, problem (2.1) can be converted to a smooth one by introducing new variables and constraints into the

problem. For example, for each $i = 1, \ldots, n$, we can introduce a new set of variables x_{il}^+ where $l = 0, \ldots, M_i^+$ and x_{il}^- where $l = 0, \ldots, M_i^-$. Then problem (2.1) can be rewritten in the form

$$\begin{array}{ll} \min & F(x) + \sum_{i=1}^n \sum_{l=0}^{M_i^+} f_{il}^+(x_{il}^+) + \sum_{i=1}^n \sum_{l=0}^{M_i^-} f_{il}^-(x_{il}^-) \\ \text{s.t.} & Ax \leq b, \\ & x_i - \sum_{l=0}^{M_i^+} x_{il}^+ + \sum_{l=0}^{M_i^-} x_{il}^- = \hat{x}_i, \ \text{ for } i = 0, ..., n, \\ & 0 \leq x_{il}^+ \leq d_{il+1}^+, \ \text{ for } i = 1, ..., n, \ l = 0, ..., M_i^+, \\ & 0 \leq x_{il}^- \leq d_{il+1}^-, \ \text{ for } i = 1, ..., n, \ l = 0, ..., M_i^-, \end{array} \tag{4.1}$$

where f_{il}^+, f_{il}^- are defined in the Appendix. The problem (4.1) is a linearly constrained, convex and twice differentiable problem and standard techniques can be used to solve it. However, this higher dimensional problem is computationally expensive to solve.

We can design an interior-point algorithm for this problem in a way analogous to our derivation in Subsection 3.1. First, we express the primal-dual central path. The central path is continuously differentiable; however, it is expressed in a very high dimensional space compared to our formulations in Section 3. To compare the underlying interior-point algorithms, we can eliminate the "new variables" (those not present in the formulations of Section 3) from the nonlinear system defining the central path. Let $v \in \mathbb{R}^n$ denote the dual variables corresponding to the linear equations expressing x in terms of $\hat{x}, x^+$, and x^-.

After eliminating all of these new variables except v, the nonlinear system of equations and inequalities are equivalently written as

$$\begin{aligned} Gx + c + A^T u + v(x) &= 0, \quad u > 0; \\ Ax + s &= b, \quad s > 0; \\ Su &= \mu e. \end{aligned}$$

In the above $v(x) : \mathbb{R}^n \to \mathbb{R}^n$ is continuously differentiable at all interior points and is completely separable, that is, $[v(x)]_i$ only depends on x_i. Therefore, if we derive the search directions based on this latest system, we end up with the normal equations determining Δx whose coefficient matrix is:

$$G + \text{Diag}\,[v'(x)] + A^T(S^{-1}U)A.$$

Compared to the search directions from Section 3, the search direction derived here has only a diagonal perturbation to the left-hand-side matrix and this perturbation $\text{Diag}\,[v'(x)]$ is independent of A, b, c, G and only depends on the nondifferentiable part of the data.

Another approach to comparing different interior-point algorithms in our setting would be to derive the search directions for each formulation in their own space and then consider the Δx components of each of these search directions, i.e., compare the projections of the search

directions in the x-space. This way of comparing the search directions could lead to different conclusions than the above. As it will become clear, our way of comparing these different formulations exposes the similarities in an extremely easy and elegant way. One drawback of the compact central path system that we derived is, the compact system is "more nonlinear" than the high dimensional, original formulation. The latter is the usual approach to quadratic programming and yields a bilinear system of equations and strict linear inequalities. In our compact system above, in addition to the usual bilinear equations (such as $Su = \mu e$), we also have the nonlinear system involving $v'(x)$.

4.2 Local Lifting

We can introduce a pair of new variables $x_i^+,\ x_i^-$ for each $i \in E(x)$ with $x_i = d_{il}$. This converts the problem into a differentiable one in the neighborhood of the current iterate. The problem becomes

$$\begin{array}{ll} \min & F(x) + \sum_{i\in N(x)} f_i(x_i) + \sum_{i\in E(x)} (f_{il}^+(x_i^+) + f_{il}^-(x_i^-)) \\ \text{s.t.} & Ax \le b, \\ & x_i = d_{il} + x_i^+ - x_i^-,\ i \in E(x), \\ & x_i^+,\ x_i^- \ge 0,\ i \in E(x). \end{array} \tag{4.2}$$

A point $x \in \mathbb{R}^n$ is a point on a central path corresponding to (4.2), if and only if

$$\left.\begin{array}{ll} Ax + s = b,\ s \ge 0 & \\ (\nabla F(x))_i + \frac{df_{il}}{dx_i}(x_i) + (A^T u)_i = 0, & \text{for all } i \in N(x), \\ & \text{with } x_i \in (d_{il}, d_{il+1}), \\ (\nabla F(x))_i + \frac{df_{il-1}}{dx_i}(d_{il}) + (A^T u)_i + v_i = 0, & \text{for all } i \in E(x), \\ & \text{with } x_i = d_{il}, \\ (\nabla F(x))_i + \frac{df_{il}}{dx_i}(d_{il}) + (A^T u)_i - w_i = 0, & \text{for all } i \in E(x), \\ & \text{with } x_i = d_{il}, \\ u \ge 0, & \\ x_i = d_{il} + x_i^+ - x_i^-,\ i \in E(x), & \\ x_i^+,\ x_i^- \ge 0,\ i \in E(x), & \\ v_i,\ w_i \ge 0,\ i \in E(x), & \\ u_i s_i = \mu,\ i = 1, \ldots, m, & \\ v_i x_i^- = \mu,\ i \in E(x), & \\ w_i x_i^+ = \mu,\ i \in E(x). & \end{array}\right\} \tag{4.3}$$

Note that $v_i + w_i = \Delta p_{il}$. For the quadratic function, the above system becomes

$$\left.\begin{array}{ll} Ax + s = b,\ s \geq 0 & \\ (Gx)_i + c_i + p_{il} + (A^T u)_i = 0, & \text{for all } i \in N(x), \\ & \text{with } x_i \in (d_{il}, d_{il+1}), \\ (Gx)_i + c_i + p_{il-1} + (A^T u)_i + v_i = 0, & \text{for all } i \in E(x), \\ & \text{with } x_i = d_{il}, \\ u \geq 0, & \\ u_i s_i = \mu, & i = 1, \ldots, m, \\ x_i^+, \geq 0,\ x^- \geq 0, & i \in E(x), \\ w_i \geq 0,\ v_i \geq 0, & i \in E(x), \\ v_i + w_i = \Delta p_{il}, & i \in E(x), \\ x_i = d_{il} + x_i^+ - x_i^-, & i \in E(x), \\ v_i x_i^- = \mu, & i \in E(x), \\ w_i x_i^+ = \mu, & i \in E(x). \end{array}\right\} \tag{4.4}$$

The last four groups of equations form the system:

$$\left.\begin{array}{l} v_i + w_i = \Delta p_{il}, \\ x_i = d_{il} + x_i^+ - x_i^-, \\ v_i x_i^- = \mu, \\ w_i x_i^+ = \mu. \end{array}\right\} \tag{4.5}$$

This allows us to express the dual variable v_i as a function of x_i

$$v_i(x_i) = \frac{2\mu\Delta p_{il}}{2\mu - \Delta p_{il}(x_i - d_{il}) + (4\mu^2 + \Delta p_{il}^2 (x_i - d_{il})^2)^{\frac{1}{2}}}. \tag{4.6}$$

Note that $v_i(d_{il}) = \frac{\Delta p_{il}}{2} > 0$. In a neighborhood of d_{il} the variables w_i, x_i^+ and x_i^- are positive and solving a system (4.4) is equivalent to solving

$$\left.\begin{array}{ll} Ax + s = b,\ s \geq 0 & \\ (Gx)_i + c_i + p_{il} + (A^T u)_i = 0, & \text{for all } i \in N(x), \\ & \text{with } x_i \in (d_{il}, d_{il+1}), \\ (Gx)_i + c_i + p_{il-1} + (A^T u)_i + v_i(x_i) = 0, & \text{for all } i \in E(x), \\ & \text{with } x_i = d_{il}, \\ u \geq 0, & \\ u_i s_i = \mu, & i = 1, \ldots, m. \end{array}\right\} \tag{4.7}$$

This approach appears to be similar to the "spline approximation" approach. We are just modeling the jumps in the gradient of $f_i(x_i)$ by a function $s_i(x_i)$, such that

$$s_i'(x_i) = v_i(x_i).$$

Proposition 4.1 *Suppose an interior-point method is applied to the problem (4.2) and a search direction $(\Delta x, \Delta u, \Delta s)$ is obtained at a point (x, u, s). Then the same direction can also be obtained by applying the interior-point method to the problem (3.1), with $\bar{f}_i(x_i) := f_i(x_i) + s_i(x_i)$, where $s_i'(x_i) := v_i(x_i)$ is given by (4.6).*

Therefore, the search direction computed in this *local lifting* approach can also be treated in the class of search directions Δx obtained from solving the system

$$[G + D + A^T(S^{-1}U)A]\Delta x \;\; = -(Gx + c + d) - A^T S^{-1}[Ur_b + \sigma\mu e], \tag{4.8}$$

where D and d are the diagonal matrix and a vector determined by a particular approach (e.g., smoothing via quadratic spline, smoothing via cubic spline, global lifting, local lifting).

Note that the above unification of these various approaches goes even further. In subsection 3.3, the sensitivity analysis leads to the linear system of equations (3.30) which is also in the above form.

The main reason for this is the fact that we derived our algorithm from the necessary and sufficient conditions for optimality. And, these conditions change slightly going from one of our formulations to another.

5 Probability Analysis for Number of Breakpoints

Recall that $f : \mathbb{R}^n \to \mathbb{R}^n$ is a convex function which is the sum of a strictly convex quadratic function and n, convex, separable, piecewise linear functions. So, f is differentiable everywhere except at the breakpoints of the piecewise linear components.

The problem we have been considering is

$$\begin{array}{ll} \min & f(x) \\ \text{s.t.} & x \in P, \end{array}$$

where $P \subseteq \mathbb{R}^n$ is a polyhedron.

For every $z \in \mathbb{R}$, define the level set of f:

$$C(z) := \{x \in \mathbb{R}^n : f(x) \leq z\},$$

which is convex (since f is) and closed (since f is continuous). Suppose that our optimization problem has an optimal value z^* and it is attained. Then we ask the question: "How likely is it that there exist

$$\bar{x} \in C(z^*) \cap P \text{ and } i \in \{1, 2, \ldots, n\} \text{ such that}$$

$\bar{x}_i$ is a breakpoint of f_i?"

Proposition 5.1 *Let f, $C(z)$, P, and z^* be as defined above. Then, $C(z)$ is a compact, convex set for every $z \in \mathbb{R}^n$. Moreover, if $P \neq \emptyset$, then z^* exists and is attained by a unique $x^* \in P$.*

Proof. We already established that $C(z)$ is closed and convex for every $z \in \mathbb{R}^n$. Since f is the sum of a strictly convex quadratic function and n, convex, piecewise linear functions, f is coercive. Thus, $C(z)$ is bounded for every $z \in \mathbb{R}^n$. Therefore, $C(z)$ is compact for every $z \in \mathbb{R}^n$. We deduce that if $P \neq \emptyset$, then z^* is finite and is attained. f is strictly convex (since it is the sum of a strictly convex function and some other convex functions); hence, there must be a unique minimizer of f on the compact, convex set $C(z^*) \cap P$. ■

In our analysis, restricted to the domain of our applications, we may assume $0 \in P$. Recall that $\hat{x}$ denotes the current investment holdings and we would expect it to be feasible. (By a simple translation of the coordinate system, we can place $\hat{x}$ at the origin.) Now, for each j, $x_j > 0$ represents buying and $x_j < 0$ represents selling. Since neither of these two activities is free, we conclude that each piecewise linear function has a breakpoint at zero. Therefore, the objective function f is nondifferentiable on the hyperplanes,

$$\{x \in \mathbb{R}^n : x_j = 0\}$$

for every j.

From a practical viewpoint, we immediately have an answer to our question. Since the investor cannot be expected to trade every single stock/commodity in every planning horizon, breakpoints at optimality are unavoidable!

From the theoretical viewpoint the answers depend on the probabilistic model used and calculating the probabilities exactly would be complicated.

In order to get a rough estimate of the number of the coordinates of the optimal solution that are at breakpoints, we looked at a simpler problem of unconstrained minimization of $f(x)$. In addition, we assumed the the matrix G is diagonal, functions $f_i(x_i)$ are the same for each coordinate, the breakpoints d_{il} and the gradients p_{il} are equally spaced. We denote by $\Delta d = d_{il+1} - d_{il}$ and by $\Delta p = p_{il+1} - p_{il}$. From the optimality conditions (2.7), $\bar{x}$ minimizes $f(x)$ if and only if $0 \in \partial f(x)$ or

$$0 \in G_i x_i + c_i + \partial f_i(x_i),\ i = 1, \ldots, n.$$

We can think of a subdifferential as a mapping from $\mathbb{R}^n \to \mathbb{R}^n$. If none of the coordinates of x are at breakpoints, this point is mapped to a single point. If a coordinate $x_i = d_{il}$ is at a breakpoint, then the i-th component of the subdifferential is an interval and the probability of having $x_i = d_{il}$ is equal to the probability of zero being in this interval.

If the coordinate $x_i = d_{il}$ is at a breakpoint,

$$0 \in [G_i d_{il} + c_i + p_{il-1}, G_i d_{il} + c_i + p_{il}].$$

Note that the length of this interval is equal to Δp.

If the coordinate $x_i \in (d_{il}, d_{il+1})$ is not at a breakpoint,

$$0 = G_i x_i + c_i + p_{il}.$$

The interval (d_{il}, d_{il+1}) is mapped to an interval

$$[G_i d_{il} + c_i + p_{il}, G_i d_{il+1} + c_i + p_{il}]$$

of length $G_i \Delta d$.

This suggest a hypothesis that ratio of the probability of the i-th coordinate being at a breakpoint to the probability of the opposite event is

$$\Delta p : G_i \Delta d.$$

We ran some tests for the constrained minimization of such functions. In some examples this estimate was pretty close, in other cases the number of the breakpoints was less than predicted, see Table 6.3 in the next section.

6 Computational Experiments

The algorithm is implemented in MATLAB and tested on randomly generated data.

In Section 6.2 we show how the parameters of the problem affect the performance of the algorithm. In Section 6.3 we look at the connection between these parameters and the number of the coordinates of the optimal solution x_i that have values at points of nondifferentiability. A crossover to an active set algorithm is tested in Section 6.4.

For all the above experiments medium-scale dense data is used.

We also tested the algorithm on large-scale sparse data. These results are reported in Section 6.6 and some implementation details are discussed in Section 6.5.

In order to compare the performance of our implementation with that of a commercial package, we convert our problem into a differentiable one (4.1) by introducing nM new variables $x^{1+}, x^{2+}, \ldots, x^{M+}$, $x^{1-}, x^{2-}, \ldots, x^{M-}$. This problem is then solved using MOSEK 3, using an interior-point method.

We run all the experiments on a SUNW, UltraSparc-IIIi, (1002 MHz, 2048 Megabytes of RAM). All execution times are given in CPU seconds. We repeat each experiment 10 times for the smaller dense problems and 5 times for the large sparse ones. The average execution times are reported in each table. The requested accuracy for our MATLAB code is $\epsilon/100$, where ϵ is the parameter of the spline approximation. In Section 6.6, we request the same accuracy from MOSEK. In Section 6.4, where the crossover method is tested, the relative gap termination tolerance for MOSEK was increased to 10e-14.

6.1 Data Generation

The data was generated in the following way. Vector c corresponds to the vector of the expected returns, randomly generated in the range (1, 1.3). The target vector $\hat{x}$ is set to zero. The number of the points of nondifferentiability $M_i = M$ is the same for each coordinate. The transaction costs are chosen as follows.

$$f_i(x_i) := \begin{cases} p_{min}x_i + h_{i0}, & \text{if} \quad x_i \leq d_{min}, \\ (p_{min} + \frac{(p_{max}-p_{min})l}{M-1})x_i + h_{il}, & \text{if} \quad \begin{cases} d_{min} + \frac{(d_{max}-d_{min})(l-1)}{M-1} \leq x_i \leq \\ \qquad d_{min} + \frac{(d_{max}-d_{min})l}{M-1}, \end{cases} \\ p_{max}x_i + h_{iM}, & \text{if} \quad x_i \geq d_{max}. \end{cases} \tag{6.1}$$

We will say that the transaction costs varied from p_{min} to p_{max} in this case.

1. **Dense Data.** In order to guarantee that the matrix G is positive semidefinite, we first construct an $n \times n$ matrix C with random entries in the range (–0.5, 0.5) and then form $G = \alpha C^T C$. Note that the the constant α corresponds to the inverse of the risk aversion parameter t. We discuss the effect of changing this constant on the problem in Section 6.2.

   The matrix of the inequality constraints, A, and the vector of the right hand side, b, are also generated randomly. In the first series of experiments we generate A and b with random entries in the range (–0.5, 0.5). We refer to this kind of data as Type 1. In the second series of experiments, we generate A with random integer entries from the set $\{0, 1, 2, 3\}$. We refer to this kind of data as Type 2. Each dense problem has one equality constraint $x_1 + x_2 + \ldots + x_n = 1$. The transaction costs varied from –0.5 to 0.5.

2. **Sparse Data.** First, we use the *sprandsym* command in MATLAB to generate the matrices G with a given sparsity and multiply by a constant α. Another type of data arising in large-scale applications is block-diagonal matrices or matrices with overlapping blocks on the diagonal. We use *sprandsym* command in MATLAB to generate each of the blocks. The transaction costs varied from –0.05 to 0.05. The matrix of the inequality constraints, A, is also sparse. In all the experiments, we ensure that A has no zero column or zero row. If the randomly generated matrix A has one or more zero columns, then we add a random number to one element in each of these columns, but in a different row. Then we check the resulting matrix for zero rows and eliminate them in a similar way.

6.2 Varying Parameters Related to Smoothness

6.2.1 Varying Number of Breakpoints M and Spline Neighbourhood ϵ

Table 6.1 presents the CPU time and the number of iterations for the IPM MATLAB program. We varied the number of breakpoints M from 3 to 101 and the size of the spline intervals ϵ from 0.001 to 0.00001. The dimension and number of constraints $n = 1000, m = 500$. Figure 6.1 illustrates the CPU time for just the cubic spline case.

We can see that increasing ϵ consistently decreases the number of iterations and CPU time. (Though our theoretical sensitivity results show that the accuracy relative to the true optimum decreases, see Section 3.3). Also, increasing the number of intervals (breakpoints) decreases the number of iterations and CPU time. In both cases, the problem becomes more like a smooth problem.

6.2.2 Scaling the Quadratic Term

We then ran our ***IPM*** code on the same set of problems, as used in Section 6.2.1 above, but we changed the value of the constant α in the definition of the matrix G. We used only the cubic spline and all the remaining parameters were fixed: $M = 51$, $\epsilon = 0.0001$, $n = 1000, m = 500$. We noticed that decreasing α increases the CPU time for the problems with transaction costs, see Table 6.2. We also report the expected return for the optimal solutions of the problems with transaction costs. Note that smaller values of α correspond to larger values of the risk aversion parameter. For example $\alpha = 0.05$ gives an extremely risky portfolio with expected return of 321%. In all the remaining experiments, we used values of α that correspond to realistic risks.

6.3 Expected Number of Breakpoints

In support of the probability analysis in Section 5, we would like to test how the parameters of the problem influence the number of the coordinates of the optimal solution coinciding with a breakpoint. For this set of experiments, the Hessian matrix G is always diagonal. We first take $G = \alpha I$, i.e., G is a multiple of the identity matrix. The matrix of the linear system A has random entries in the range (-0.5, 0.5), the vector of the right hand sides b has random entries in the range (0, 1). Note that zero is always feasible for these problems. The rest of the data is generated as described above in Section 6.1. The results are presented in Table 6.3.

In Table 6.4 matrix G has $4, 3, 2$ or 1 on the diagonal, 25% of each. This subdivides the set of all coordinates into 4 groups. The rest of the data is as above. This table shows the number

M \ ϵ	0.001	0.0005	0.0001	0.00005	0.00001
	Quadratic Spline				
3	44 (20)	55 (23)	109 (38)	148 (46)	407(98)
25	36 (17)	38 (17)	52 (21)	61 (23)	107 (31)
51	36 (17)	37 (17)	43 (19)	52 (20)	87 (28)
75	36 (16)	37 (17)	41 (18)	46 (19)	77 (26)
101	35 (16)	38 (17)	43 (19)	45 (19)	71 (25)
	Cubic Spline				
3	43 (20)	53 (24)	97 (39)	133 (48)	348 (104)
25	35 (16)	37 (17)	49 (21)	59 (24)	98 (33)
51	34 (16)	36 (17)	44 (20)	49 (21)	84 (30)
75	33 (16)	35 (16)	42 (19)	47 (20)	70 (26)
101	33 (15)	35 (16)	42 (19)	45 (20)	71 (26)

Table 6.1: CPU (iter) for MATLAB ***IPM*** ; $n = 1000, m = 500$.

	α=1	α=0.5	α=0.1	α=0.05
Problem with Trans. Costs	43 (20)	51 (22)	129 (46)	216 (74)
Expected return	1.35	1.56	2.46	3.21

Table 6.2: CPU (iter) for MATLAB ***IPM*** ; $n = 1000, m = 500, M = 101, \epsilon = 0.0001$.

of coordinates of the optimal solution at a breakpoint in each of these subgroups. The Tables 6.3,6.4 both show that the predicted values and empirical values are reasonably close.

6.4 Crossover for Obtaining Higher Accuracy

If a more accurate optimal solution is needed, we can do a crossover to an active set algorithm.

The active set method was implemented in C for the problem in a standard equality form. At each iteration of the active set method, the variables are subdivided into basic and nonbasic. A coordinate can be nonbasic only if it is equal to one of the breakpoints (if upper or lower bounds are present, they are treated as breakpoints too). A Newton search direction is taken in the basic subspace. This requires solving a symmetric linear system at each iteration. The step size is taken so that all the variables stay in the corresponding intervals between the breakpoints.

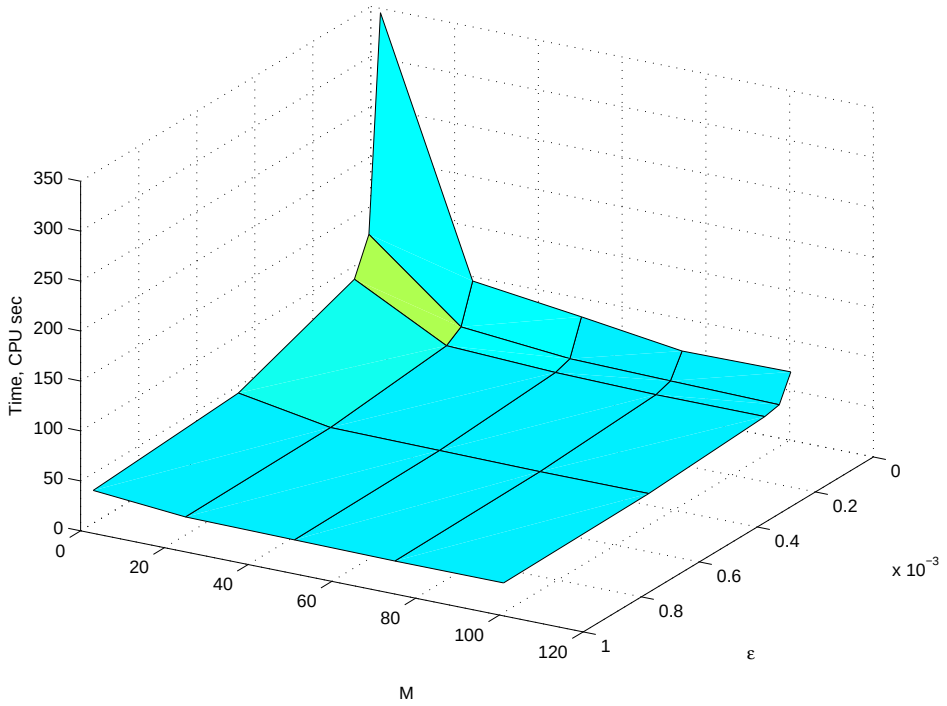

Figure 6.1: CPU for MATLAB ***IPM***; $n = 1000, m = 500$, cubic spline.

At the end of each iteration, either one variable is added to the basis or it is dropped from the basis. We store the inverse of the coefficient matrix of the system. The next linear system differs form the previous one by one row and column. We use this fact to update the inverse.[1] These routines were converted to C by an automatic translator. To further improve efficiency we used CBLAS routines to perform basic matrix and vector operations.

Two versions of a crossover method between the ***IPM*** and active set method were tested. In the first type of crossover, we used the last iterate of the interior-point method as an initial point for the active set method. However, because of the nature of the active set method, starting it with an interior point makes all the slack variables basic. The number of iterations needed for the active set method to finish the problem is at least the number of the constraints active at the optimum. Since our active set algorithm takes a Newton step at each iteration, this method is time consuming. It could perform well if only few constraints were active at the optimum and few coordinates were at breakpoints.

Another approach is to take a purification step first. We also use the last iterate of the interior

[1]The FORTRAN routines for these updates were kindly provided by Professor M.J. Best, University of Waterloo.

	$\alpha = 1$	$\alpha = 2$	$\alpha = 4$	$\alpha = 8$
$\Delta p = \Delta d$				
Experiment	167 (42%)	110 (28%)	68 (17%)	48 (12%)
Predicted	50%	33%	20%	11%
$\Delta p = 2\Delta d$				
Experiment	232 (58%)	179 (45%)	122 (31%)	78 (20%)
Predicted	66%	50%	33%	20%
$2\Delta p = \Delta d$				
Experiment	109 (27%)	72 (18%)	33 (8%)	21 (5%)
Predicted	33%	20%	11%	6%

Table 6.3: # (%) of coordinates of optimum at breakpoint; $n = 400, m = 800$.

	G_{ii}=4	G_{ii}=3	G_{ii}=2	G_{ii}=1
Experiment	18(18%)	23(23%)	30(30%)	39(39%)
Predicted	20%	25%	33%	50%

Table 6.4: # (%) of coordinates of optimum at breakpoint in each subgroup; n=400, m=800, $\Delta p = \Delta d$.

point method as an initial point and we perform several iterations of the gradient projection method, e.g. [16]. We stop if the optimal solution is found or if a constraint should be dropped. In the latter case the last iterate of the purification step is used to start an active set algorithm.

The purification step was implemented in MATLAB. At each iteration, we keep track of the set of active constraints and the projection matrix corresponding to these constraints. We find the orthogonal projection of the negative gradient of the objective function at the current iterate onto the subspace parallel to the affine space of the currently active constraints. We find the maximal feasible step size in this direction and also perform a line search to minimize the true objective function, along this direction. We either add one more constraint to the active set and update the projection matrix, or stop, depending on the outcome of the line search. This method has the guarantee that the true objective function value of the final solution from the purification is at least as good as that of the final ***IPM*** solution. This method performed best in most cases.

These results are summarized in Tables 6.5 and 6.6. From Table 6.5, we see that doing the purification step before the crossover is always faster. We only present the faster option in the remaining table; the number of iterations is given in brackets.

For problems where the number of breakpoints is large, our program performs faster than MOSEK. We found that terminating the approximated problem when the relative gap is equal to ϵ gives slightly better timings. Also note that our ***IPM*** was implemented in MATLAB and is generally slower than MOSEK on differentiable problems. (We ran MOSEK and our code on the same differentiable problems, and MOSEK was approximately 2.5 times faster than our MATLAB code.)

				MOSEK 102 (25)	
	MATLAB ***IPM***	Purification Step(MATLAB)	ASet after Pur.	ASet after ***IPM***	Crossover total
tol=10e-3					
With Pur.Step	24 (10)	18 (250)	85 (65)		127
No Pur.Step	24 (10)	-		430 (333)	
tol=10e-4					
With Pur.Step	32 (14)	18 (250)	50 (32)		100
No Pur.Step	32 (14)	-		390 (281)	
tol=10e-5					
With Pur.Step	35 (16)	18 (246)	48 (30)		101
No Pur.Step	35 (16)	-		389 (278)	

Table 6.5: CPU (iter), Crossover, $n = 1000, m = 500, M = 101, \epsilon = .0001$, Data Type 1.

6.5 Comparing Linear System Solvers

We compared MATLAB CPU times for the following three different ways of solving the linear system for the search direction in the interior-point algorithm:

1. **Chol.** Form a sparse matrix $A^T(S^{-1}U)A$, and perform a Cholesky decomposition on the matrix
$$\left[G + H + A^T(S^{-1}U)A\right] \tag{6.2}$$
converted into dense format.

2. **Aug.** Directly solve the *augmented system* whose coefficient matrix is
$$\begin{bmatrix} G+H & A^T \\ A & -U^{-1}S \end{bmatrix}$$

			MOSEK 217 (31)	
MATLAB Term. Tol.	MATLAB ***IPM***	Purification Step(MATLAB)	ASet after Pur.	Crossover total
10e-3	25 (11)	18 (247)	76 (56)	119
10e-4	34 (15)	18 (248)	52 (33)	104
10e-5	36 (16)	18 (248)	57 (37)	111

Table 6.6: CPU (iter), Crossover with purif. step, $n = 1000, m = 500, M = 101, \epsilon = 0.0001$, Data Type 2.

using the MATLAB 'backslash' command.

3. **Bcksl.** Solve the sparse system whose coefficient matrix is $\left[G + H + A^T(S^{-1}U)A\right]$ using the MATLAB 'backslash' command.

4. **Block LU** Solve the *augmented system* using a block LU approach. We first compute the LU factorization of the upper left block $G + H = L_{11}U_{11}$. We then solve triangular systems $L_{11}U_{12} = A^T$ and $L_{21}U_{11} = A$ for U_{12} and L_{21}, respectively. Finally we form a matrix $Z = -U^{-1}S - L_{21}U_{12}$ (a Schur complement of $G+H$) and find the LU factorization $L_{22}U_{22} = Z$.

   Then
   $$\begin{bmatrix} G+H & A^T \\ A & -U^{-1}S \end{bmatrix} = \begin{bmatrix} L_{11} & 0 \\ L_{21} & L_{22} \end{bmatrix} \begin{bmatrix} U_{11} & U_{12} \\ 0 & U_{22} \end{bmatrix}$$
   is the LU factorization of the *augmented system*. Since the system is symmetric and the matrix $G + H$ is positive definite, it would make more sense to perform the Cholesky decomposition of $G + H$. But the sparse LU decomposition proved to be much faster in MATLAB.

   This approach is beneficial when the matrix $G+H$ has some special structure, for example banded or block-diagonal with $n >> m$.

Remark 6.1 *Note that the above approach can be used in solving smooth convex quadratic problems with $n >> m$, since the blocks L_{11}, L_{21}, U_{11} and U_{12} have to be calculated only once.*

G \ A	dense			60%			40%			5%		
	Chol	Aug	Bcksl	Chol	Aug	Bcksl	Chol	Aug	Bcksl	Chol	Aug	Bcksl
dense	1.8	3.3	2.1	6.6	3.3	6.5	4.1	3.3	4	1.8	8.7	1.7
40%	12	6.3	10.2	6.8	5.3	6.6	4.2	10.4	4.1	1.8	3.1	1.8
5%	12	6.7	11.8	6.7	6.3	6.5	4.2	7.6	4.1	1.6	4.5	1.6

Table 6.7: MATLAB CPU, different solvers; $n = 1000, m = 500$

At each iteration, only the $m \times m$ matrix Z has to be factorized. MATLAB is 2-3 times faster than MOSEK on such QP examples.

In Table 6.7, we summarize the CPU times for different ways of solving the linear system per iteration of ***IPM*** . The problem parameters are $M = 101$, $\epsilon = 0.0001$. In the case when both matrices are dense, we store them in a dense format. For the Cholesky case, we do the matrix multiplication in (6.2) in sparse format, but then convert the matrix into dense format before performing the Cholesky decomposition. For the remaining two methods the data is kept in a sparse format.

For $\frac{n}{4} \leq m \leq \frac{n}{2}$, we found that whenever G and A were both full dense, CPU times for **Chol.** were half of those for **Aug.**. When we made A more sparse (while keeping G full dense), the CPU times became equal around 40% density for A. When A had only 5% of its entries as non-zeroes, **Chol.** beat **Aug.** by a factor of five.

We notice that keeping the data in a sparse format is only beneficial when both matrices are very sparse, 5% or less. Otherwise, keeping the data in a dense format and doing the Cholesky decomposition is the fastest choice in MATLAB. We'll refer to this option as **Dense Chol.**.

Table 6.8 is created in a similar way. Only very sparse data is considered and the CPU time for the **Dense Chol.** option is given in a separate column. We can see that the Cholesky factorization is always faster than the backslash command. When G and A are $1 - 5\%$ sparse, Cholesky dominates all other methods. We notice that increasing the sparsity and decreasing the number of constraints improves the performance of the augmented system. When the sparsity is around 0.5% and the number of constraints is 10% of the number of variables, the augmented system becomes the fastest choice.

In the next series of experiments we model real-life, large-scale portfolio optimization problems with thousands of variables. In such applications with very large n, a very sparse G, banded (or near-banded) matrix structure makes sense. For this set of experiments, we generate G with overlapping blocks on the diagonal. We also add upper and lower bounds on all the variables. The number of constraints is equal to the number of blocks. We also add a budget constraint

G \ A	5%			1%			
	Chol	Aug	Bcksl	Chol	Aug	Bcksl	Dense Chol
m=1000							
1%	22	26	33	13	12	34	33
0.5%	26	23	33	16	7	34	33
m=300							
1%	17	5	28	12	3	24	17
0.5%	16	3	28	12	1	19	17

Table 6.8: MATLAB CPU, different solvers; $n = 3000$.

$x_1 + x_2 + \ldots + x_n \leq 1$. We noticed that addition of the bounds does not change the timings significantly. But the budget constraint makes the matrix of a condensed system dense. For these problems, the augmented system gives much better results. Therefore, we only present results for the **Aug.** and **Block LU** methods in Table 6.9.

6.6 Experiments with Sparse Data

In this section, we compare the CPU time for MOSEK and our algorithm (in MATLAB) on sparse large scale data. The spline approximation parameter $\epsilon = 10e - 5$. Both MOSEK and MATLAB were terminated when the relative gap was less than 10e-7. We noticed that for all the examples solved, the objective function $f(x)$ at the solutions given by MOSEK and MATLAB differ in the seventh or eights digit.

For the Tables 6.10, 6.11 and 6.12 matrix G was sparse but had no special structure. In Table 6.10 we solve the same problems changing only the number of the breakpoints. The CPU time for our method stays virtually unchanged; while the CPU time for the lifted problem solved in MOSEK increase. In the next series of tests we increase the dimension of the problem, G has 20 non-zeros per row, all the remaining parameters are fixed, $M = 25$. In this case our code beats MOSEK by a constant, see Table 6.11. For Table 6.12, we also increase the dimension of the problem, but keep the sparsity of G constant. In this case, MOSEK performs better with the increase in dimension.

	A					
	10%		5%		1%	
	Aug	Block LU	Aug	Block LU	Aug	Block LU
n=3000 (15 blocks)	2.1	1.7	1.6	1.5	0.6	1.2
n=6000 (30 blocks)	5.4	3.5	3.2	3.2	1.4	2.6
n=9000 (45 blocks)	10.6	5.6	5.6	5.1	2.4	4.0
n=12000 (60 blocks)	7.2	7.9	9.1	7.0	3.8	5.6

Table 6.9: MATLAB CPU, different solvers; G 200×200 blocks, 10% den.; $m = 200$; up/low bnds.

Number of Breakpoints	MATLAB	MOSEK
$M = 101$	83 (15)	456 (13)
$M = 51$	78 (14)	230(12)
$M = 25$	82(15)	129 (12)
$M = 11$	80 (15)	70 (11)
$M = 3$	85 (15)	42 (10)
No Trans. Costs	74 (15)	30 (9)

Table 6.10: CPU (iter); $n = 5000$, $G0.5\%$ den.; $m = 300, A1\%$ den.

Dimension	MATLAB	MOSEK
n=21000	902 (13)	1000 (15)
n=18000	695 (14)	788 (15)
n=15000	433 (14)	588(15)
n=12000	262 (13)	370 (13)
n=9000	146 (13)	224 (11)
n=6000	71 (14)	143 (11)
n=3000	24 (14)	64 (11)

Table 6.11: CPU (iter); G has 20 nonzeros per row; $m = 300$, A 1% den.; $M = 25$.

Dimension	MATLAB	MOSEK
n=12000	1980 (13)	1026(11)
n=9000	593(14)	425 (11)
n=6000	117 (13)	162 (11)
n=3000	16 (13)	63 (11)

Table 6.12: CPU (iter); G is 0.5% den.; $m = 300$, A 1% den.; $M = 25$.

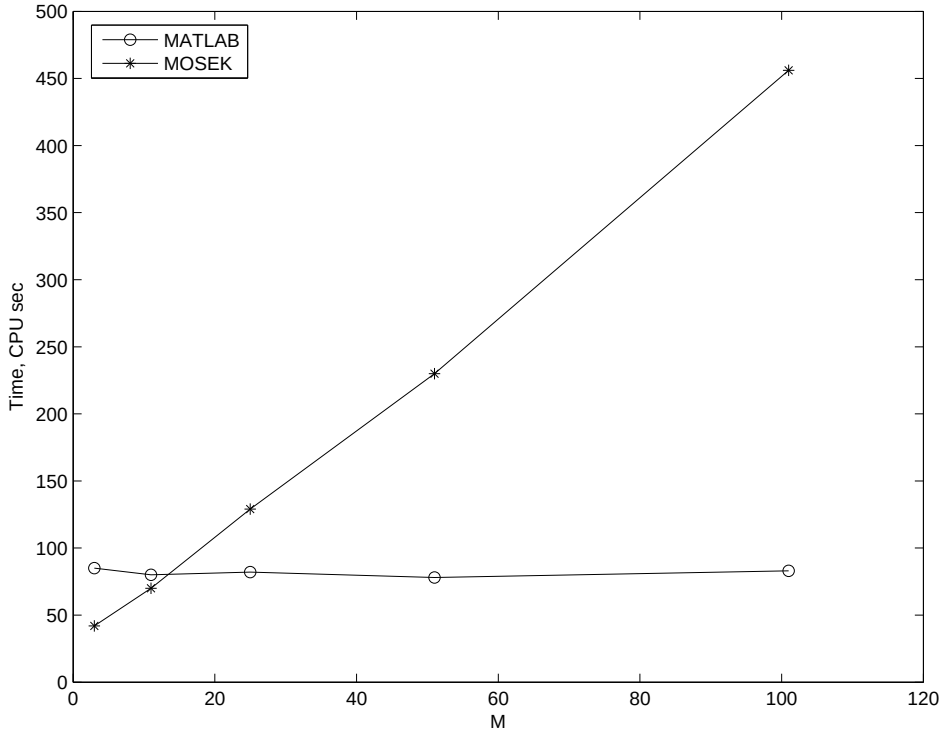

Figure 6.2: CPU (iter); $n = 5000$, G 0.5% den.; $m = 300$, A 1% dense.

In the remaining tables, the matrix G is block-diagonal with block size approximately 200×200. The blocks are overlapping by 10 diagonal elements on average. Each block is sparse.

As before, CPU times for our method stays virtually unchanged with increase in the number of breakpoints; while the CPU time for the lifted problem solved in MOSEK increases, Table 6.13. For Table 6.14, we increase the dimension of the problem, but keep the block size constant. In this case our MATLAB code beats MOSEK by a constant factor. Also note that MOSEK is approximately 2 times faster on a smooth problem without the transaction costs than our MATLAB code.

Some additional experiments on a very large data are reported in Table 6.15. Note that for these problems, MOSEK spends around 50% of the time on preprocessing.

Number of Breakpoints	MATLAB	MOSEK
$M = 101$	97 (13)	825 (13)
$M = 51$	95 (13)	440 (13)
$M = 25$	94 (13)	215 (11)
$M = 11$	95 (13)	117 (10)
$M = 3$	101 (14)	78 (10)
No Trans. Costs	93 (13)	46 (9)

Table 6.13: CPU (iter); G has 45 200×200 blocks, 10% den.; $m = 200$, A 10% den.; up/low bnds.

Number of Blocks	MATLAB	MOSEK
75 blocks n=15000	164 (13)	401 (11)
60 blocks n=12000	131 (13)	303(11)
45 blocks n=9000	94 (13)	215 (11)
30 blocks n=6000	53 (12)	135 (11)
15 blocks n=3000	26 (12)	64 (11)

Table 6.14: CPU (iter) G has 200×200 blocks; 10% den.; $m = 200$, A 10% den.; up/low bnds, $M = 25$.

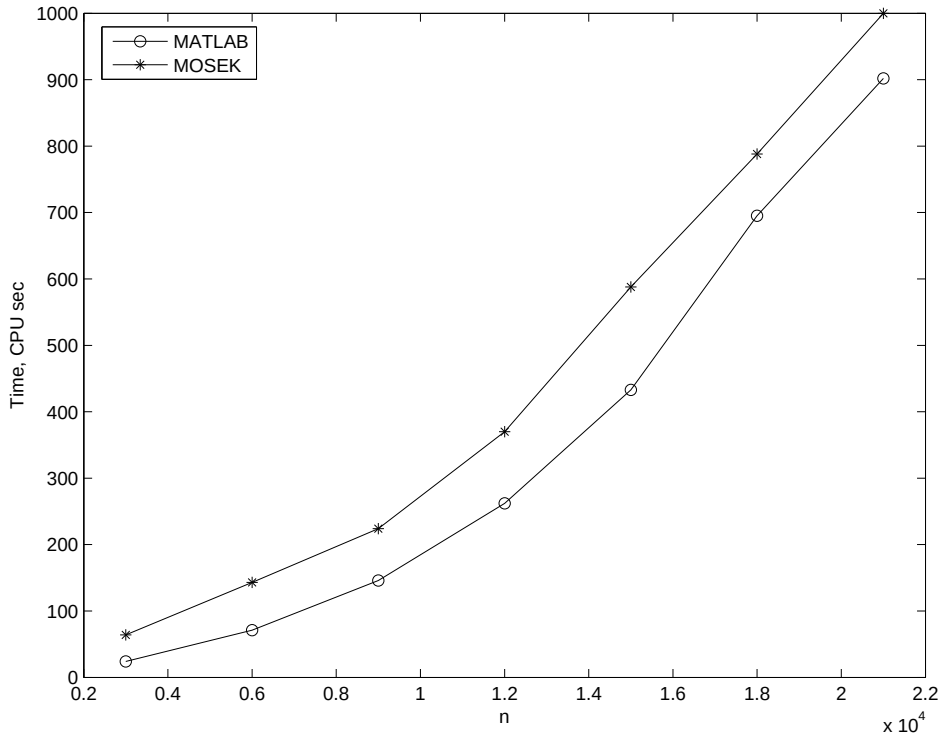

Figure 6.3: CPU (iter); G has 20 non-zeros per row; $m = 300$, A 1% den.; $M = 25$.

7 Conclusion

In this paper, we considered the expected utility maximization problem in the presence of convex, non-differentiable transaction costs and linear or piece-wise linear constraints. We used the subdifferential to derive the optimality conditions for this problem.

We showed that approximating the transaction costs with spline functions and solving the smooth problem with the interior-point methods give a very good approximation to the optimal solution. When a higher accuracy was needed, we did a crossover to an active set method.

Our numerical tests showed that we can solve large scale problems efficiently and accurately.

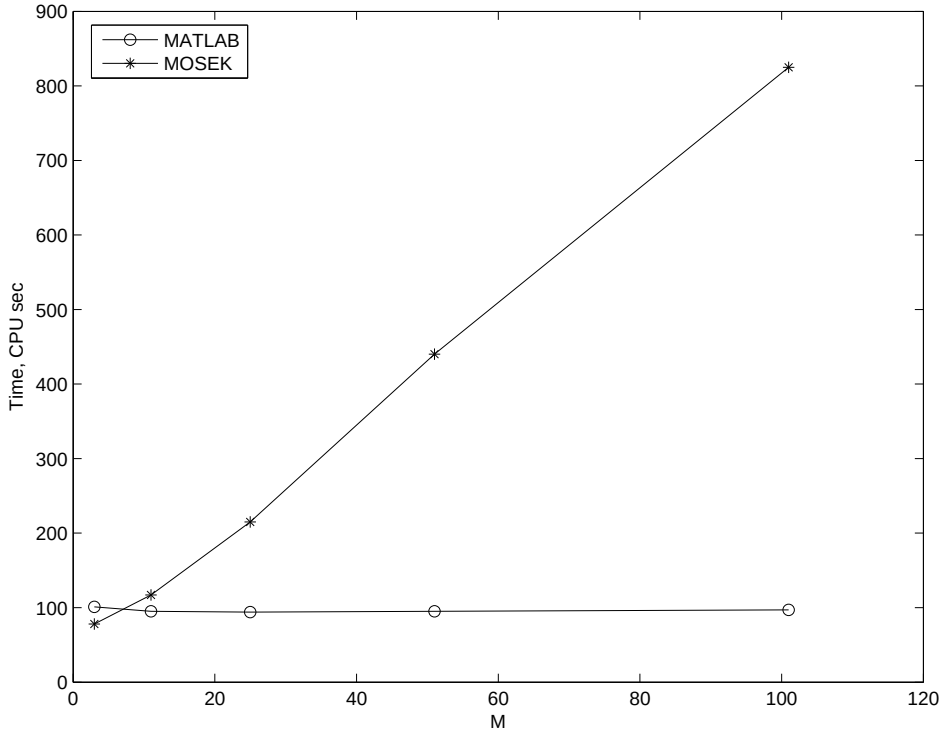

Figure 6.4: CPU (iter); G has 45 blocks 200×200, 10% den.; $m = 200$, A 10% den.; up/low bnds.

A Transaction Costs

We assume that the transaction costs are given by the following function

$$f_i(x_i) = \begin{cases} f_{il}^-(-x_i + \hat{x}_i - d_{il}^-) + \sum_{j=1}^{l} f_{ij-1}^-(d_{ij}^-), & \text{if } x_i - \hat{x}_i \in [-d_{il+1}^-, -d_{il}^-], \\ & \text{for some } l \in \{0, .., M_i^-\}, \\ f_{il}^+(x_i - \hat{x}_i - d_{il}^+) + \sum_{j=1}^{l} f_{ij-1}^+(d_{ij}^+), & \text{if } x_i - \hat{x}_i \in [d_{il}^+, d_{il+1}^+], \\ & \text{for some } l \in \{0, .., M_i^+\}. \end{cases}$$

where $d_{i0}^+ = d_{i0}^- = 0$, $d_{iM_i^+ +1}^+ = d_{iM_i^- +1}^- = +\infty$.

If the holding in the asset number i has not changed, i.e. $x_i = \hat{x}_i$, the transaction costs associated with this asset should be equal to zero. Therefore $f_i(x)$ should satisfy the conditions

$$f_i(\hat{x}_i) = 0. \tag{A.1}$$

The above notation comes naturally from the statement of the problem, but we can simplify it for the purpose of formulating the solution algorithm. Let $M_i = M_i^+ + M_i^- + 1$, so that M_i

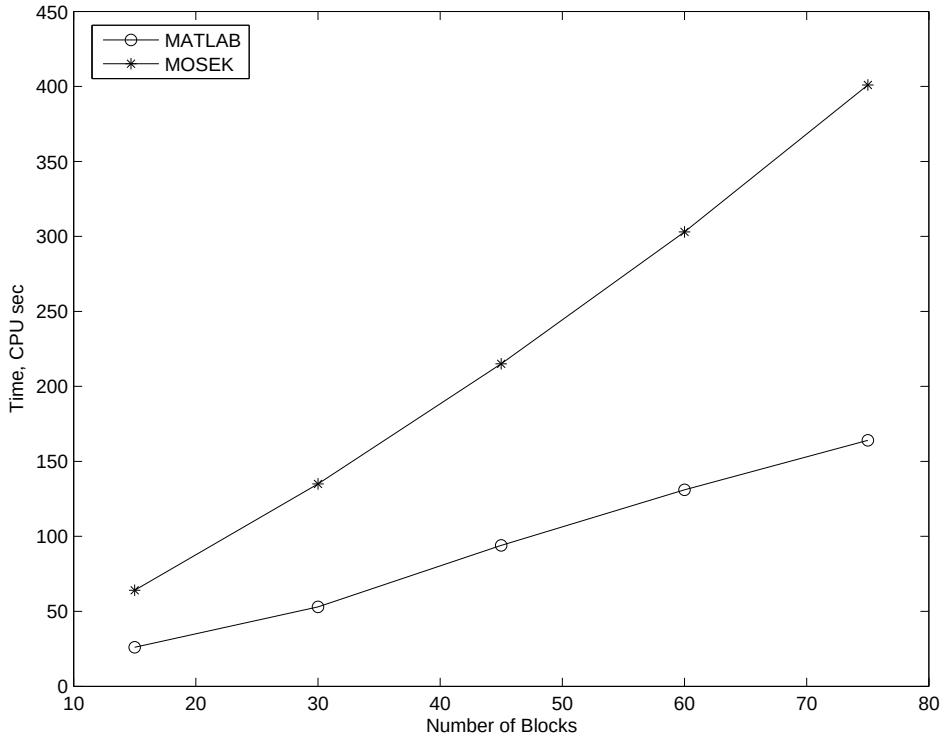

Figure 6.5: CPU (iter); G 200×200 blocks; 10% den.; $m = 200$, A 10% den.; up/low bnds, $M = 25$.

is the total number of end points of the intervals ("breakpoints"). We further denote

$$\begin{aligned} d_{il} &= \hat{x}_i - d^-_{i(M_i^- - l+1)}, \quad l = 0, ..., M_i^- + 1, \\ d_{il} &= \hat{x}_i + d^+_{i(l-M_i^+ +1)}, \quad l = M_i^- + 2, ..., M_i + 1, \end{aligned}$$

and

$$\begin{aligned} f_{il}(x_i) &= f^-_{i(M_i^- - l)}(-x_i + \hat{x}_i - d^-_{i(M_i^- - l)}) + \textstyle\sum_{j=1}^{(M_i^- - l)} f^-_{ij-1}(d^-_{ij}), \; l = 0, ..., M_i^-, \\ f_{il}(x_i) &= f^+_{i(l-M_i^+)}(x_i - \hat{x}_i - d^+_{i(l-M_i^+)}) + \textstyle\sum_{j=1}^{(l-M_i^+)} f^+_{ij-1}(d^+_{ij}), \quad l = M_i^- + 1, ..., M_i. \end{aligned}$$

Thus we can rewrite the cost functions in the following more compact way:

$$f_i(x_i) = \begin{cases} f_{i0}(x_i), & \text{if} \quad x_i \le d_{i1}, \\ f_{il}(x_i), & \text{if} \quad x_i \in [d_{il}, d_{il+1}], \quad l = 1, .., M_i, \\ f_{iM_i}(x_i), & \text{if} \quad x_i \ge d_{iM_i}. \end{cases} \tag{A.2}$$

n	Blocks				A		M	MATLAB	MOSEK
	Number	Size	Density	Overlap	m	Density			
53400	89	600	0.006	9	500	0.1	51	3114 (15)	8797 (11)
100000	1000	100	0.1	10	200	0.01	25	2966 (15)	5595 (16)
150000	5000	30	0.1	5	300	0.01	11	8890 (18)	near opt. 5674 (28)
200000	10000	20	0.1	5	300	0.01	11	18010 (17)	can't solve

Table 6.15: CPU (iter), large-scale problems.

References

[1] L. ALTANGEREL, R.I. BOŢ, and G. WANKA. Duality for convex partially separable optimization problems. *Mong. Math. J.*, 7:1–18, 2003.

[2] D.P. BERTSEKAS. *Nonlinear Optimization*. Athena Scientific, Belmont, MA, 1999.

[3] A.R. CONN, N. GOULD, M. LESCRENIER, and P.L. TOINT. Performance of a multifrontal scheme for partially separable optimization. In *Advances in optimization and numerical analysis (Oaxaca, 1992)*, volume 275 of *Math. Appl.*, pages 79–96. Kluwer Acad. Publ., Dordrecht, 1994.

[4] A.R. CONN, N. GOULD, and P.L. TOINT. Improving the decomposition of partially separable functions in the context of large-scale optimization: a first approach. In *Large scale optimization (Gainesville, FL, 1993)*, pages 82–94. Kluwer Acad. Publ., Dordrecht, 1994.

[5] G. DANTZIG, J. FOLKMAN, and N. SHAPIRO. On the continuity of the minimum set of a continuous function. *Journal Math. Anal. Appl.*, 17:519–548, 1967.

[6] M.J. DAYDÉ, J.Y. L'EXCELLENT, and N.I.M. GOULD. Element-by-element preconditioners for large partially separable optimization problems. *SIAM J. Sci. Comput.*, 18(6):1767–1787, 1997.

[7] C. de BOOR. *A Practical Guide to Splines*. Springer-Verlag, Berlin, 1978.

[8] E. ERDOGAN, D. GOLDFARB, and G. IYENGAR. Robust portfolio management. CORC Technical Report TR-2004-11, IEOR Dept., Columbia University, New York, NY, 2004.

[9] A.V. FIACCO. *Introduction to Sensitivity and Stability Analysis in Nonlinear Programming*, volume 165 of *Mathematics in Science and Engineering*. Academic Press, 1983.

[10] D. GOLDFARB and G. IYENGAR. Robust portfolio selection problems. *Math. Oper. Res.*, 28(1):1–38, 2003.

[11] A. GRIEWANK and P.L. TOINT. Numerical experiments with partially separable optimization problems. In *Numerical analysis (Dundee, 1983)*, volume 1066 of *Lecture Notes in Math.*, pages 203–220. Springer, Berlin, 1984.

[12] A.O. GRIEWANK and PH.L. TOINT. On the unconstrained optimization of partially separable functions. In M.J.D. Powell, editor, *Nonlinear Optimization*. Academic Press, London, 1982.

[13] J. KREIMER and R. Y. RUBINSTEIN. Nondifferentiable optimization via smooth approximation: General analytical approach. *Annals of Operations Research*, 39:97–119, 1992.

[14] A. LI. Some applications of symmetric cone programming in financial mathematics. Master's thesis, University of Waterloo, Canada, 2005.

[15] M.S. LOBO, M. FAZEL, and S. BOYD. Portfolio optimization with linear and fixed transaction costs. *Ann. Oper. Res.*, to appear:–, 2006.

[16] K.G. MURTY. *Linear Complementarity, Linear and Nonlinear Programming*. Heldermann Verlag, Berlin, 1988.

[17] A.F. PEROLD. Large-scale portfolio optimization. *Management Sci.*, 30(10):1143–1160, 1984.

[18] H. QI and X. YANG. Regularity and well-posedness of a dual program for convex best c^1-spline interpolation. Report, University of Southhampton, Southhampton, England, 2004.

[19] J.B. SCHATTMAN. *Portfolio selection under nonconvex transaction costs and capital gains taxes*. PhD thesis, Rutgers Center for Operations Research, Rutgers University, USA, 2000.

[20] J.W. SCHMIDT. Dual algorithms for solving convex partially separable optimization problems. *Jahresber. Deutsch. Math.-Verein.*, 94(1):40–62, 1992.

[21] P.L. TOINT. Global convergence of the partitioned BFGS algorithm for convex partially separable optimization. *Math. Programming*, 36(3):290–306, 1986.