Large Scale Portfolio Optimization with Piecewise Linear Transaction Costs

Marina Potaptchik* Levent Tunçel† Henry Wolkowicz‡

revised: May 24, 2008

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
AMS subject classifications: 91B28,62P05,90C31

Abstract

We consider the fundamental problem of computing an optimal portfolio based on a quadratic mean-variance model for 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 first solved *approximately* using a primal-dual interior-point method applied to the smoothed problem. Then, we *crossover* to an active set method applied to the original nonsmooth problem to attain a high accuracy solution. 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

1 Introduction

We consider the problem of selecting an optimal portfolio that includes transaction costs. (See e.g. [7].) We let $x = (x_1, x_2, \ldots, x_n)^T$ denote the vector of proportions (weights) for each of n given assets of the portfolio. Under the classical mean-variance model, an investor minimizes the quadratic function $F(x) = \frac{1}{2}x^T Qx - td^T x$, under linear inequality constraints, i.e. he solves 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. Thus, the *utility function* $F(x)$ combines the two objectives of "maximizing expected return" (i.e. $\min -d^T x$) and "minimizing the expected risk" (i.e. $\min x^T Qx$). The function $F(x)$ combines these two ingredients with the weighting parameter $t > 0$; $\frac{1}{t}$ represents the degree of risk aversion of the investor. The additional linear inequality constraints in $\boldsymbol{QP}$ arise from e.g. budget, diversity, and short-selling constraints.

In this paper, we consider the *portfolio optimization problem with transaction costs*, i.e. the minimization of $f(x) = F(x) + \sum_{i=1}^{n} f_i(x_i)$ subject to linear constraints, where the additional functions $f_i(x)$ are piecewise linear, convex functions that represent the transaction costs associated with changing the holdings in asset i from the current holding $\hat{x}_i$ to x_i. Rather than applying a classical active set approach to handle the piecewise linear constraints, we *smooth* these nondifferentiable functions with splines and *approximately* solve the resulting smooth problem using a specialized primal-dual interior-point method, denoted $\boldsymbol{IPM}$. Then, once we are *close enough* to the set of optimal solutions, we apply a *crossover technique* to an active set method on the original nonsmooth problem. The specialized interior-point method ensures a major savings in total computation time. The active set method guarantees high accuracy of our final solution. We include numerical tests and comparisons.

1.1 Background and Motivation

The portfolio optimization problem with transaction costs involves the minimization of the nonsmooth function $f(x)$. The classical approach to these problems is to apply an active set method and replace gradients by subgradients, see e.g. the work on ℓ_1 penalty methods [33, 26], and more recently, the discussion in [19]. A second approach involves lifting the problem into a higher dimensional space so that the resulting problem is differentiable. Again, the advantage of this approach is that standard active set and $\boldsymbol{IPMs}$ for $\boldsymbol{QPs}$ can be applied.

For the ℓ_1 penalty problems, one can expect relatively few components of the optimum x to be at points of nondifferentiability, see e.g. [19]. However (as seen in our Section 4), we can expect a relatively large number of components at points of nondifferentiability (at breakpoints). Therefore, active set methods that deal with the nondifferentiability can be inefficient. Our aim is to test a third approach based on smoothing the objective function using splines. Then, we take advantage of the efficiency of $\boldsymbol{IPMs}$ on large scale smooth problems, and also exploit the special structure of the portfolio optimization problem.

Transaction costs arise from e.g. brokerage fees and market impact. (See e.g. [4, 22, 31, 3, 3].) Though brokerage fees are often modeled by concave functions, they are dominated by market impact costs for large transactions, and the latter can be modeled using piecewise linear, separable, convex functions.

For more details on the transaction cost model, see Appendix A. Related work on portfolio optimization with transaction costs appears in e.g. [27, 21, 23, 11, 15, 31]. See e.g. [1, 5, 6, 9, 16, 17, 24, 32, 34] for approaches to partially separable optimization problems; and see [35, 20] for smoothing techniques. To the best of our knowledge, none of the pre-existing work in portfolio optimization problems with transaction costs seems to have the versatility of our approach in terms of speed and accuracy. For example, the approaches utilizing second-order cone programming formulations (or semidefinite optimization formulations) with interior-point algorithms cannot obtain the high accuracy solutions that our approach generates with proper choices for the smoothing parameter and stopping criteria to switch to the active set algorithm. Since we handle the portfolio optimization problem by utilizing a combination of an ***IPM*** and an active set approach, and since we control the smoothness parameter of the smoothed problem, depending on the application and needs of the final user, our algorithm is more adaptable to reach a satisfactory compromise (involving speed and accuracy) for the final user.

1.2 Outline; Main Results

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 3.1. We include a sensitivity analysis. In particular, this proves that more accurate spline approximations yield more accurate solutions of the original problem, i.e. with exact arithmetic, we get continuity of the spline approximations. An alternative approach that replaces the nondifferentiability with additional variables and constraints, a lifting, is presented in Section 3.2.

In Section 4, 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 5. These results show the advantage of the smoothing technique for large, sparse, problems, compared to the lifting approach. Concluding remarks are given in Section 6.

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

$$f(x) := F(x) + \sum_{i=1}^{n} f_i(x_i), \qquad F(x) := \frac{1}{2} x^T G x + c^T x, \tag{2.2}$$

G is symmetric positive definite, and $f_i(\alpha)$ is a convex, piecewise linear function on $\mathbb{R}$, with break-points at $d_{ik}, i = 1, \ldots, M_i$, i.e.

$$f_i(\alpha) := \left\{ \; f_{il}(\alpha) := p_{il}\alpha + h_{il}, \qquad \text{for} \quad d_{il} \le \alpha \le d_{il+1}, \; l = 0, \ldots, M_i, \right. \tag{2.3}$$

with $d_{i0} << 0$ arbitrarily small, and $d_{iM_i+1} >> 0$ arbitrarily large. Our approach does not really need **strict** convexity (mere convexity is sufficient); however, we assume strict convexity for the sake of a cleaner presentation and notational convenience.

Let the feasible set 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

$$E(x) := \{i : x_i = d_{il} \text{ for some } l \in \{1, .., M_i\}\}, \quad N(x) := \{1, \ldots, n\} \backslash E(x). \tag{2.4}$$

2.2 Duality and Optimality

The Lagrangian dual of (P) is $\max_{u \ge 0} \min_x L(x, u) := f(x) + u^T(Ax - b)$. The inner-minimization is an unconstrained convex minimization problem. Therefore, we can write down the Wolfe dual program

$$(D) \qquad \begin{array}{ll} \max & L(x, u) \\ \text{s.t.} & 0 \in \partial_x L(x, u), \quad u \ge 0, \end{array} \tag{2.5}$$

where $\partial_x L(x, u)$ denotes the subgradient of L [30], i.e.

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

Theorem 2.1 ([2]) *A point $x \in \mathbb{R}^n$ minimizes f over S if and only if the following system is consistent*

$$\begin{array}{cc} u \ge 0, \quad 0 \in \partial_x L(x, u) & \textit{dual feas.} \\ Ax \le b & \textit{primal feas.} \\ u^T(Ax - b) = 0 & \textit{compl. slack.} \end{array}$$

■

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*

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

■

From Proposition 2.1, we get

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

Moreover, the definition of $f_i(\alpha)$ (2.3) yields $\nabla f_{il} = p_{il}$ and

$$\partial f_i(\alpha) = \begin{cases} p_{il} & \text{if} \quad d_{il} < \alpha < d_{il+1}, \;\; l = 0, ..., M_i, \\ [p_{il-1}, p_{il}] & \text{if} \quad \alpha = d_{il}, \;\; l = 1, ..., M_i, \end{cases}$$

By abuse of notation, we think of f_i as a function both on $\mathbb{R}$ and on $\mathbb{R}^n$, i.e. $f_i(x) \cong f_i(x_i)$. Then we can identify $\partial f_i(x) = \partial f_i(x_i) e_i$ (where e_i is the ith unit vector). Now, by Proposition 2.1,

$$0 \in \partial L(x,u) \quad \Leftrightarrow \quad 0 \in (\nabla F(x))_i + \partial f_i(x_i) + (A^T u)_i, \quad i = 1, \ldots, n.$$

Corollary 2.1 *A point $x \in \mathbb{R}^n$ is optimal for (P) if and only if there exists $u \in \mathbb{R}^m$ such that*

$$\begin{array}{ll} u \geq 0 & \\ (Gx)_i + c_i + (A^T u)_i \in \begin{cases} -\{p_{il}\} & \text{if} \quad i \in N(x), x_i \in (d_{il}, d_{il+1}) \\ -(p_{il-1}, p_{il}) & \text{if} \quad i \in E(x), x_i = d_{il} \end{cases} & \textit{(dual feas.)} \\ s \geq 0, \;\; Ax + s = b & \textit{(primal feas.)} \\ u_i s_i = 0, \;\; i = 1, \ldots, m & \textit{(compl. slack.)} \end{array} \tag{2.6}$$

■

Applying an ***IPM*** directly to the nondifferentiable system (2.6) would force us to follow a nondifferentiable "central path". Note that (x, s, u) is on this *central path* corresponding to the barrier parameter $\mu > 0$, if and only if it is a solution of the system (2.6) with the complementary slackness conditions perturbed to

$$u_i s_i = \mu, \;\; i = 1, \ldots, m.$$

Remark 2.1 *An alternative derivation of the central path starts with the family of convex optimization problems*

$$\begin{array}{ll} \min & f(x) - \mu \sum_{i=1}^{m} \ln(s_i) \\ s.t. & Ax + s = b, \end{array}$$

parameterized by $\mu > 0$. Under the assumptions that there exists $\bar{x}$ such that $A\bar{x} < b$ and $\{x : Ax \leq b\}$ is bounded, we conclude that for every $\mu > 0$, the above convex optimization problem has a unique optimal solution. This set of optimal solutions also defines the central path and establishes the existence and uniqueness of central points for each $\mu > 0$.

3 Smoothing

We consider two techniques for smoothing (P). The first one approximates the piecewise linear functions using smooth splines. The second one extends the linear pieces and maintains the correct values by lifting the problem into a higher dimensional space and adding constraints.

3.1 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. In many cases the nondifferentiable functions are themselves an approximation of some smooth function. Then, using a convex cubic spline would give a better approximation of the original function, e.g. [29]; for a general reference on splines, see [10].

However, for our portfolio optimization problem, transaction cost functions are generally nonsmooth. Therefore, in this paper, we focus on using smooth convex splines that approximate the given piecewise linear convex functions $f_i(x_i)$.

3.1.1 Interior-Point Method for Smooth Approximations

We suppose that the function $f_i(\alpha)$ is approximated by a smooth family of functions $\bar{f}_i(\alpha, \epsilon)$, parameterized by ϵ. We denote the first and second derivatives, with respect to α, by $\bar{f}_i'(\alpha, \epsilon)$ and $\bar{f}_i''(\alpha, \epsilon)$, respectively. For a fixed $\epsilon > 0$, and for each $\mu > 0$, the following system of perturbed optimality conditions is considered at each iteration of the ***IPM*** , e.g. [13],

$$\begin{array}{ll} u > 0 & \\ (Gx)_i + c_i + \bar{f}_i'(x_i, \epsilon) + (A^T u)_i = 0, \quad \forall i = 1, \ldots, n & \text{dual feas.} \\ s > 0,\ Ax + s = b & \text{primal feas.} \\ u_i s_i = \mu,\ i = 1, \ldots, m & \text{pert. compl. slack.} \end{array} \tag{3.1}$$

Let (x, u, s) be a current iterate for solving (3.1), with $(u, s) > 0$. We set the barrier parameter $\mu := \frac{u^T s}{m}$, and define the vector of first derivatives $g := (\bar{f}_1'(x_1, \epsilon), \ldots, \bar{f}_n'(x_n, \epsilon))^T$,

and the diagonal matrices

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

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)

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

We can add upper and lower bounds $b_l \leq x \leq b_u$ to the problem and exploit the special structure of these constraints, rather than treating them as part of A. (The details are well-known and can be found in [28] and the references therein.)

So, far, we focused on the general structure of our choice for the search direction. Another very important ingredient of interior-point algorithms is the step size selection (in our case, step size choices for the dual are particularly critical). We used an adaptive approach to find the search direction and the step size choices. We based the choice of the centering parameter σ on the size of the step from the previous iteration, i.e. we decreased (respectively increased) σ if the last step size was large (respectively small). (E.g. if the step size was less than 0.001, we chose $\sigma = 2.1$, which results in an increase in the current value of μ.) We also monitored the changes in the dual constraints. If a step size of α resulted in the infinity norm of the dual residual vector to more than double, we replaced α by 0.9α repeatedly, until the dual residual was under control.

3.1.2 Quadratic and Cubic Splines

We actually approximate the piecewise linear convex function $f_i(\alpha)$ with a *parameterized* (by ϵ) family of quadratic and cubic convex splines

$$\bar{f}_i(\alpha, \epsilon) := f_i(\alpha) + s_i(\alpha, \epsilon). \tag{3.5}$$

Let $\Delta p_{il} := p_{il} - p_{il-1}$. For the quadratic spline we get

$$s_i(x_i, \epsilon) = \begin{cases} \frac{\Delta p_{il}}{4\epsilon}(x_i - d_{il} + \epsilon)^2 & \text{if } \epsilon > 0, x_i \in [d_{il} - \epsilon, d_{il} + \epsilon], \text{for some } l \in \{1, \ldots, M_i\} \\ 0 & \text{otherwise.} \end{cases} \tag{3.6}$$

For the cubic spline, we have

$$s_i(x_i, \epsilon) = \begin{cases} \frac{\Delta p_{il}}{6\epsilon^2}(x_i - d_{il} + \epsilon)^3 & \text{if } \epsilon > 0, x_i \in [d_{il} - \epsilon, d_{il}], \text{ for some } l \in \{1, \ldots, M_i\} \\ -\frac{\Delta p_{il}}{6\epsilon^2}(x_i - d_{il} - \epsilon)^3 + (\Delta p_{il})x_i & \text{if } \epsilon > 0, x_i \in [d_{il}, d_{il} + \epsilon], \text{ for some } l \in \{1, \ldots, M_i\} \\ 0 & \text{otherwise.} \end{cases} \tag{3.7}$$

For $0 < \epsilon < \frac{1}{2}\min_{i,l}(d_{il} - d_{il-1})$, the functions $\bar{f}_i(x_i, \epsilon)$ defined above are continuously differentiable; while in the case of the cubic spline, they are twice continuously differentiable. We define a conservative upper bound for ϵ with

$$\bar{\epsilon} := \frac{1}{3}\min_{i,l}(d_{il} - d_{il-1}). \tag{3.8}$$

Note that the smooth spline functions above provide uniform approximations to the non-smooth objective function. Therefore, it follows easily that the objective value of the smooth problem is continuous in ϵ on $[0, \bar{\epsilon}]$. For related classical results, see e.g. Fiacco [12].

3.2 Lifting

Because of the special piecewise linear structure of the nondifferentiable part of the objective function, it is common practice to convert problem (2.1) to a smooth one by introducing new variables and constraints. For example, for each $i = 1, \ldots, n$, we can introduce new sets of variables x_{il}^+, $l = 0, \ldots, M_i^+$, and x_{il}^-, $l = 0, \ldots, M_i^-$. We can then rewrite problem (2.1) in the form

$$\begin{array}{ll} \min & F(x) + \sum_{i=1}^n \sum_{l=0}^{M_i^+} f_{il}^+(x_{il}^+) + \sum_{i=1}^n \sum_{l=0}^{M_i^-} f_{il}^-(x_{il}^-) \\ \text{s.t.} & Ax \le b, \\ & x_i - \sum_{l=0}^{M_i^+} x_{il}^+ + \sum_{l=0}^{M_i^-} x_{il}^- = \hat{x}_i, \ \text{ for } i = 0, ..., n, \\ & 0 \le x_{il}^+ \le d_{il+1}^+, \ \text{ for } i = 1, ..., n, \ l = 0, ..., M_i^+, \\ & 0 \le x_{il}^- \le d_{il+1}^-, \ \text{ for } i = 1, ..., n, \ l = 0, ..., M_i^-. \end{array} \tag{3.9}$$

(The functions f_{il}^+, f_{il}^-,and indices M_i^+, M_i^- are defined in Appendix A.) The problem (3.9) 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 Section 3.1.1. To compare the underlying interior-point algorithms, we can eliminate the "new variables", i.e. those not present in the formulations of Section 3.1.1. 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:

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

Compared to the search directions from Section 3.1.1, the search direction derived here has only a diagonal perturbation to the left-hand-side matrix and this perturbation $\text{Diag}\left[v'(x)\right]$ 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 $v'(x)$.

3.2.1 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

$$\begin{array}{ll} \min & F(x) + \sum_{i\in N(x)} f_i(x_i) + \sum_{i\in E(x)} (f_{il}^+(x_i^+) + f_{il}^-(x_i^-)) \\ \text{s.t.} & Ax \le b, \\ & x_i = d_{il} + x_i^+ - x_i^-,\ i \in E(x), \\ & x_i^+,\ x_i^- \ge 0,\ i \in E(x). \end{array} \tag{3.10}$$

A point $x \in \mathbb{R}^n$ is a point on a central path corresponding to (3.10), if and only if

$$\left.\begin{array}{ll} Ax + s = b,\ s > 0 & \\ (Gx)_i + c_i + p_{il} + (A^T u)_i = 0, & \text{for all } i \in N(x) \\ & \text{with } x_i \in (d_{il}, d_{il+1}) \\ (Gx)_i + c_i + p_{il-1} + (A^T u)_i + v_i = 0, & \text{for all } i \in E(x) \\ & \text{with } x_i = d_{il} \\ u > 0, & \\ u_i s_i = \mu, & i = 1, \ldots, m \\ x_i^+, > 0,\ x^- > 0, & i \in E(x) \\ w_i > 0,\ v_i > 0, & i \in E(x) \\ v_i + w_i = \Delta p_{il}, & i \in E(x) \\ x_i = d_{il} + x_i^+ - x_i^-, & i \in E(x) \\ v_i x_i^- = \mu, & i \in E(x) \\ w_i x_i^+ = \mu, & i \in E(x). \end{array}\right\} \tag{3.11}$$

The last four groups of equations form the system:

$$\left.\begin{array}{l} v_i + w_i = \Delta p_{il} \\ x_i = d_{il} + x_i^+ - x_i^- \\ v_i x_i^- = \mu \\ w_i x_i^+ = \mu \end{array}\right\} \tag{3.12}$$

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

$$v_i(x_i) = \frac{2\mu\Delta p_{il}}{2\mu - \Delta p_{il}(x_i - d_{il}) + (4\mu^2 + \Delta p_{il}^2 (x_i - d_{il})^2)^{\frac{1}{2}}}. \tag{3.13}$$

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 the system (3.11) is equivalent to solving

$$\left.\begin{array}{ll} Ax + s = b,\ s > 0 & \\ (Gx)_i + c_i + p_{il} + (A^T u)_i = 0, & \text{for all } i \in N(x) \\ & \text{with } x_i \in (d_{il}, d_{il+1}) \\ (Gx)_i + c_i + p_{il-1} + (A^T u)_i + v_i(x_i) = 0, & \text{for all } i \in E(x) \\ & \text{with } x_i = d_{il} \\ u > 0, & \\ u_i s_i = \mu, & i = 1, \ldots, m. \end{array}\right\} \tag{3.14}$$

Therefore, this approach is similar to the *spline approximation* approach, i.e. we model 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 3.1 *Suppose an interior-point method is applied to the problem (3.10) and a search direction $(\Delta x, \Delta u, \Delta s)$ is obtained at a point (x, u, s). Then the same direction is obtained by applying the interior-point method to the problem (3.1), with $\bar{f}_i(x_i) := f_i(x_i) + s_i(x_i)$, and $s_i'(x_i) = v_i(x_i)$ in (3.13).*

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

$$[G + D + A^T(S^{-1}U)A]\Delta x \quad = -(Gx + c + d) - A^T S^{-1}[Ur_b + \sigma\mu e], \tag{3.15}$$

where D and d are, respectively, a diagonal matrix and a vector determined by a particular approach (e.g., smoothing via quadratic spline, smoothing via cubic spline, global lifting, local lifting). This follows from the fact that we derived our algorithms from the necessary and sufficient conditions for optimality, and these conditions vary slightly between formulations. We did not experiment with this local lifting approach (the above derivation aims at establishing the fact that all of the approaches can be studied in the same framework from the point-of-view of search directions generated from the underlying normal equation system). Indeed, derivation of the underlying theory and robust algorithms is expected to require significantly more effort than the other approaches; moreover, because of the above derivation of the search direction for the local lifting approach, it is clear that we can simulate the generation of underlying search direction by utilizing a proper spline whose s_i' leads to the desired v_i above for each i.

4 Probabilistic Estimates for the Number of Breakpoints

Recall that the objective function for (P), $f : \mathbb{R}^n \to \mathbb{R}^n$, is convex and differentiable everywhere, except at the breakpoints of the piecewise linear components. For every $v \in \mathbb{R}$, define the level set of f:

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

Suppose that our optimization problem has an optimal value v^* and it is attained. Then we ask

Question 4.1 *How likely is it that there exist*

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

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

We first note the following elementary result.

Proposition 4.1 *Let f, $C(v)$, S, and v^* be as defined above. Then, $C(v)$ is a compact, convex set for every $v \in \mathbb{R}$. Moreover, if $S \neq \emptyset$, then v^* exists and is attained by a unique $x^* \in S$.*

Proof. Since f is the sum of a strictly convex quadratic function and n, convex, piecewise linear functions, f is strictly convex and coercive. Thus, $C(z)$ is convex and compact for every $z \in \mathbb{R}^n$. We deduce that if $S \neq \emptyset$, then v^* is finite and attained. Strict convexity of f now yields the uniqueness. ■

Recall that $\hat{x}$ denotes the current investment holdings. Without loss of generality (after a translation), we can assume that $\hat{x} = 0 \in S$. Then, 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 f_i has a breakpoint at $x_i = 0$. 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 Question 4.1. 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 is difficult.

In order to find an estimate for the number of coordinates of the optimal solution that are at breakpoints, we use a simplified problem: we consider an *unconstrained minimization of* $f(x)$; and we assume that the matrix G is diagonal, the functions $f_i(x_i)$ are the same for each coordinate i, and the breakpoints d_{il} and the gradients p_{il} are equally spaced. We denote the common differences/spaces by $\Delta d = d_{il+1} - d_{il}$ and $\Delta p = p_{il+1} - p_{il}$.

From the optimality conditions (2.6), x minimizes $f(x)$ if and only if $0 \in \partial f(x)$ or

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

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,

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

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 $G_i \Delta d$. Thus, when we focus on the space of subgradients and assume that the locations of these two intervals are uniformly chosen over a large interval containing zero, we deduce that the two probabilities are proportional to the lengths of the two intervals.

We now have an estimate to the answer of Question 4.1 in the special case that G is diagonal and the functions f_i are equal $\forall i$, i.e. the ratio of: the probability of the i_0-th coordinate of the optimal x being at a breakpoint *to* the probability of the complementary event is approximately

$$\frac{\Delta p}{G_{i_0 i_0} \Delta d}. \tag{4.1}$$

Our tests in Table 5.3 in Section 5.3, below, show the strength of the estimate (4.1) for the original problem (P).

5 Computational Experiments

Our algorithm, using smoothing with splines and the crossover, was implemented in the MATLAB 7.3 environment; and it was tested on randomly generated data. In Section 5.2 we show how the parameters of the problem affect the performance of the algorithm. In Section 5.3 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. The crossover to an active set algorithm is tested in Section 5.4. Implementation details are discussed in Section 5.5. Results with large scale sparse data sets are reported in Section 5.6. Comparisons are done using the commercial package MOSEK 3 as the main competitor, i.e. we use the global lifting procedure in Section 3.2 to obtain a QP problem and then solve this problem with the general purpose interior-point software in Mosek 3.

The tests were run on a SUNW, UltraSparc-IIIi, (1002 MHz, 2048 Megabytes of RAM). Execution times are 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 5.6, we request this same accuracy from MOSEK; though in Section 5.4 where the crossover method is tested, the relative gap termination tolerance for MOSEK was increased to 10^{-14}, to match our termination criteria.

5.1 Data Generation

The vector c corresponding to the vector of the expected returns was randomly generated with elements in the interval (1, 1.3). The current holdings $\hat{x}$ was set to zero. The number of points of nondifferentiability was the same for each transaction cost function, i.e. $M_i = M, \forall i$.

1. **Dense Data.** We set the matrix $G = \alpha C^T C$ where C is $n \times n$ with random entries in the interval (–0.5, 0.5). Thus, the constant α corresponds to the inverse of the risk parameter t. (The effect of changing α is discussed in in Section 5.2.) The data A, b are randomly generated with entries in the interval (–0.5, 0.5); we call this *Type 1 data*. In the second series of experiments, we generate A with random integer entries from the set $\{0, 1, 2, 3\}$.

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

We refer to this as *Type 2 data*. For both data types, the transaction costs were chosen randomly in the same interval (–0.5, 0.5). In addition, each dense problem includes one (budget) constraint $x_1+x_2+\ldots+x_n = 1$. Note that for portfolio optimization applications, an equality budget constraint is common and without loss of much generality, since each investor has access to a risk free asset (e.g., a bank account). Moreover, replacing the above equality by an inequality does not have any notable impact on the number of iterations or on the computation time per iteration.

2. **Sparse Data.** We used the *sprandsym* command in MATLAB to generate both sparse matrices G as well as sparse matrices G with nonzero pattern made up of overlapping diagonal blocks. The sparse constraint and transaction data were randomly generated as for the dense case above. We did ensure that the constraints did not have a zero row or column.

5.2 Effect of Parameters Related to Smoothness

5.2.1 Number of Breakpoints M; Size of Spline Neighborhood ϵ

Table 5.1 presents the CPU time and the number of iterations for our IPM MATLAB code. We varied the number of breakpoints M from 3 to 101 and the size of the spline intervals ϵ from 10^{-3} to 10^{-5}. The dimension and number of constraints are $n = 1000, m = 500$. Figure 5.1 illustrates the CPU time for just the cubic spline case, with the inverse of the risk parameter $\alpha = 1$.

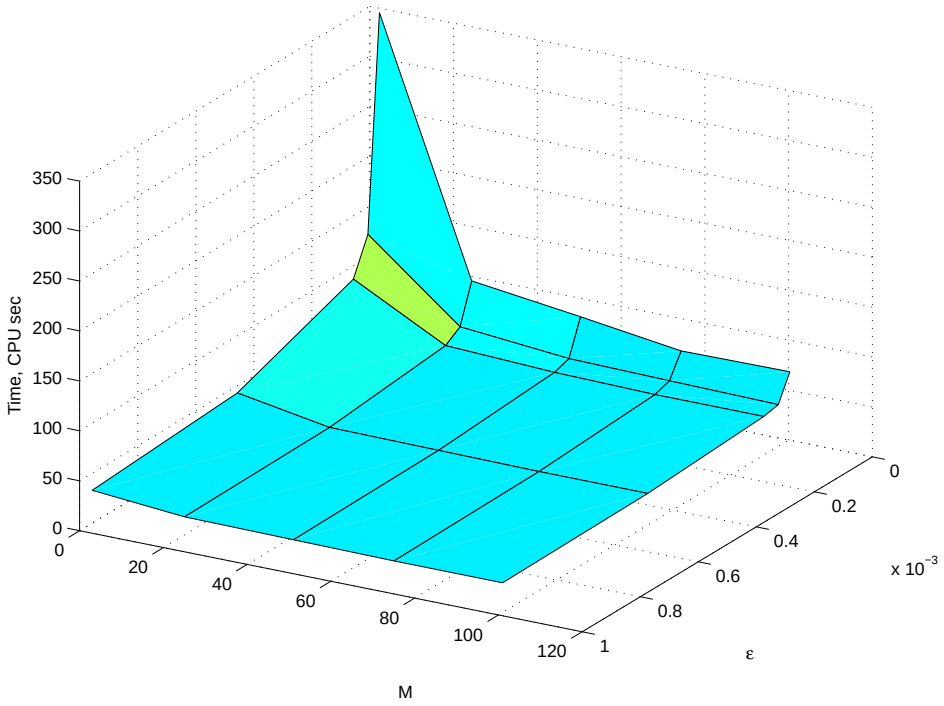

Figure 5.1: CPU for MATLAB ***IPM*** ; $n = 1000, m = 500$, cubic spline.

We can see that increasing ϵ consistently decreases the number of iterations and the CPU time. Though the theory shows that the accuracy relative to the true optimum decreases. Also, increasing the number of intervals (breakpoints) decreases the number of iterations and the CPU time. As expected, in both cases the problem is farther from nondifferentiability, which results in better behaviour for the ***IPM*** .

5.2.2 Scaling the Quadratic Term, Risk Aversion

We then ran our ***IPM*** code on the same set of problems, as in Section 5.2.1 above, but we changed the value of the parameter α in the definition of the matrix G. We used only the cubic spline and all the remaining parameters were fixed: $M = 101$, $\epsilon = 0.0001$, $n = 1000, m = 500$. We see that decreasing α increases the CPU time, see Table 5.2. 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 parameter t. 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.

	α=1	α=0.5	α=0.1	α=0.05
Problem from Figure 5.1 with $M = 101, \epsilon = 0.0001$	43 (20)	51 (22)	129 (46)	216 (74)

Table 5.2: CPU (iter) for MATLAB ***IPM*** ; $n = 1000, m = 500$.

5.3 Expected Number of Active Breakpoints

We now test the estimate given in (4.1) in Section 4. The data is generated as described above in Section 5.1. We first restrict $G = \alpha I$. These results are presented in Table 5.3. Next, we

	$\alpha = 1$	$\alpha = 2$	$\alpha = 4$	$\alpha = 8$
$\Delta p = \Delta d$				
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 5.3: # (%) of coordinates of optimum at breakpoint; $n = 400, m = 800$.

restrict G to be diagonal with integer diagonal entries $1, 2, 3, 4$ in equal proportion of occurrence. Table 5.4 shows the number of coordinates of the optimal solution at a breakpoint in each of these subgroups.

Both Tables 5.3 and 5.4 show that the predicted values and empirical values are close. In addition, the number (percentage) of active breakpoints decreases as the parameter α increases, i.e., as the risk factor decreases. Therefore, problems with high risk factor should be more difficult/expensive to solve.

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

5.4 Crossover for Obtaining Higher Accuracy

Rather than forcing the interior-point algorithm to obtain highly accurate solutions (which may be futile because it would require smaller intervals for the spline approximation and can result in numerical instability as well as a significant increase in computation time), we use relatively large intervals for the spline approximation, run the interior-point algorithm on this smooth problem to a rough tolerance, and then apply a crossover technique to an active set algorithm on the original problem.

At each iteration of the active set method, the indices $i = 1, \ldots, n$ are subdivided into basic and nonbasic sets. (A coordinate i is nonbasic if x_i is equal to one of the breakpoints. The indices for the set of constraints also provides elements for the basic/nonbasic sets.) We apply a standard active set approach, e.g. [14]. [1]

We tested two variations for the crossover. The first variation used the last iterate of the interior-point method as an initial point for the active set method. However, the result is that 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 and few coordinates were at breakpoints for the optimum.

The second variation first applied a purification step, i.e. we also use the last iterate from the ***IPM*** and then perform several iterations of the gradient projection method, e.g. [25] to increase the size of the active set as much as we can (without sacrificing the objective function value). 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 the active set algorithm. This variation 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.

The numerical tests are summarized in Tables 5.5 and 5.6. From Table 5.5, we see that doing the purification step before the crossover is always faster. We only present the faster purify option in the remaining Table 5.6.

For problems where the number of breakpoints is large, our program performs faster than MOSEK (overall run times). We found that terminating the ***IPM*** 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. (For benchmarking purposes, we ran MOSEK and our code on the same differentiable problems, and MOSEK was approximately 2.5 times faster than our MATLAB code.)

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

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

Table 5.5: CPU (iter) for: Crossover, Data Type 1; $n = 1000, m = 500, M = 101, \epsilon = .0001$.

5.5 Comparing Linear System Solvers

We considered four different ways of solving the linear system for the search direction, i.e. the augmented system in (3.3) or the normal equation system (3.4).

1. **Cholesky:** Form $A^T(S^{-1}U)A$ using sparse arithmetic; then use a (dense) Cholesky factorization of the matrix
$$\left[G + H + A^T(S^{-1}U)A\right] \tag{5.1}$$
to solve the system (3.4).

2. **Augmented:** Directly solve the *augmented system* (3.3) using the MATLAB 'backslash,\' command. (The \ command is based on an LU factorization but also takes advantage of sparsity.)

3. **Backsolve:** Use the MATLAB 'backslash' command as in the Augmented approach above in Item 2, but apply it to the normal equations system (3.4).

4. **Block LU:** We solve the *augmented system* (3.3) using a block LU approach: first compute the LU factorization of the upper left block $G + H = L_{11}U_{11}$; then solve the triangular systems $L_{11}U_{12} = A^T$ and $L_{21}U_{11} = A$ for U_{12} and L_{21}, respectively; finally, 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*.

		MOSEK 217 (31)		
MATLAB Termin. Tol.	MATLAB ***IPM***	Purification Step (MATLAB)	Active Set step	Crossover total
10^{-3}	25 (11)	18 (247)	76 (56)	119
10^{-4}	34 (15)	18 (248)	52 (33)	104
10^{-5}	36 (16)	18 (248)	57 (37)	111

Table 5.6: CPU (iter) for: Crossover with purif. step, Data Type 2; $n = 1000, m = 500, M = 101, \epsilon = 0.0001$,

G \ A	100% dense			60% dense			40% dense			5% dense		
	Chol	Aug	Bcksl	Chol	Aug	Bcksl	Chol	Aug	Bcksl	Chol	Aug	Bcksl
100%	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 5.7: CPU for: different densities and different linear solvers; $n = 1000, m = 500, M = 101, \epsilon = 0.0001$.

Remark 5.1 *In the Block LU approach, Item 4, 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 $n >> m$. 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. At each iteration, only the $m \times m$ matrix Z has to be factored. MATLAB is 2-3 times faster than MOSEK on such QP examples.

In Table 5.7, we summarize CPU times for the Cholesky/Augmented/Backsolve techniques for solving the linear system for various densities of the data matrices A, G. 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 nonzeros, **Chol.** beat **Aug.** by a factor of five.

G \ A	5% dense			1% dense			
	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 5.8: CPU for: different densities and different linear solvers; $n = 3000, M = 101, \epsilon = 0.0001$.

Table 5.8 is similar to Table 5.7, but Only sparse data is considered and CPU times for the **Dense Cholesky** 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% to 5% dense, Cholesky outperforms 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. Consider a multi-stage decision making environment. In such applications, large n, sparse G, and banded (or near-banded) matrix structures make sense, see e.g. [21]. As another application, consider a particular sector for investment opportunities, say the energy sector, where there is no limit on the nationality of the companies. In this case there may be a lot of correlation among all energy stocks. These correlations are strong within a geographical region and across regions, but strongest within the similar products (e.g., oil). Again, we find that the block diagonal structures (with moderate overlap) provides a good mathematical model.

For these 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 $x_1+x_2+\ldots+x_n = 1$. We noticed that addition of the lower/upper bounds does not change the timings significantly. But the budget constraint makes the matrix of a normal equation 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 5.9.

A density	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 5.9: CPU for: different linear solvers with up/low bnds; G 200×200 blocks, 10% den.; $m = 200, M = 101, \epsilon = 0.0001$.

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 5.10: CPU (iter) for: $n = 5000$, G 0.5% dens.; $m = 300, A$ 1% dens.

5.6 Experiments with Sparse Data

In this section, we compare the CPU times for MOSEK (using the global lifting formulation) and our algorithm (in MATLAB using the smoothed formulation) on sparse large scale data. We set the spline approximation parameter $\epsilon = 10^{-5}$. Both MOSEK and MATLAB were terminated when the relative gap was less than 10^{-7}. For all experiments, the objective function $f(x)$ at the solutions given by MOSEK and MATLAB agreed to seven digits accuracy.

For the Tables 5.10, 5.11 and 5.12 matrix G was sparse but had no special structure. (See also Figures 5.2 and 5.3.) In Table 5.10 we solve the same problems changing only the number of the breakpoints. The CPU times for our method stays virtually unchanged; while the CPU times for the lifted problem solved by MOSEK increase. (See also Figure 5.2.)

In the next series of tests we increase the dimension of the problem so that G has 20 nonzeros per row, $M = 25$, and all the remaining parameters are fixed. In this case our code beats MOSEK by a constant factor, see Table 5.11. For Table 5.12, we also increase the dimension of

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 5.11: CPU (iter) for: G has 20 nonzeros per row; $m = 300$, A 1% dens.; $M = 25$.

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 5.12: CPU (iter) for: G 0.5% dens.; $m = 300$, A 1% dens.; $M = 25$.

the problem, but keep the sparsity of G constant. In this case, MOSEK performs better with the increase in dimension.

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 an increase in the number of breakpoints; while the CPU times for the lifted problem solved in MOSEK increases, see Table 5.13 and Figure 5.4.

For Table 5.14, 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 transaction costs than our MATLAB code. (See also Figure 5.5.)

Some additional experiments on very large data are reported in Table 5.15. 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 5.13: CPU (iter) for: G has 45 200×200 blocks, 10% dens.; $m = 200$, A 10% dens.; with 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 5.14: CPU (iter) G has 200×200 blocks; 10% den.; $m = 200$, A 10% den.; up/low bnds, $M = 25$.

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 5.15: CPU (iter) for: large-scale problems.

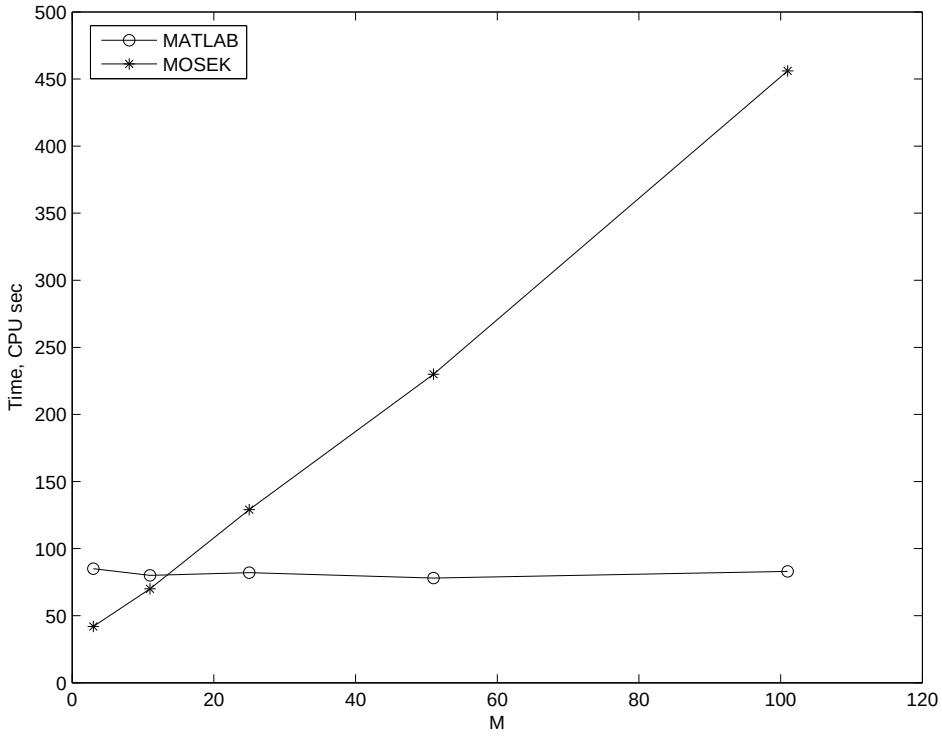

Figure 5.2: CPU for: $n = 5000$, G 0.5% den.; $m = 300$, A 1% dense.

6 Conclusion

In this paper, we considered the optimal portfolio problem with convex, piecewise linear transaction costs and linear inequality constraints. We solved these problems efficiently with a three stage approach. First, we smoothed the problem by approximating the transaction costs with spline functions; and we solved the smoothed problem with an ***IPM*** . Second, using a purification algorithm (including a gradient projection method) on the final solution from ***IPM*** , we generated good feasible solutions to start the active set algorithm on the original nonsmooth portfolio optimization problem. Thirdly, we obtained very high accuracy solutions by employing an active set method from the good feasible solution generated by the purification stage.

Our numerical tests showed that we can solve large scale problems efficiently and accurately. These numerical tests suggest that this approach is useful for general nonsmooth problems with piecewise linear portions in the objective function and for which one can expect many components of the optimum at breakpoints.

Hastie et al. [18] work with a nondifferentiable unconstrained problem, where nondifferentiability arises from penalization of the residual $[Ax - b]_+$. In their case, A and b come from

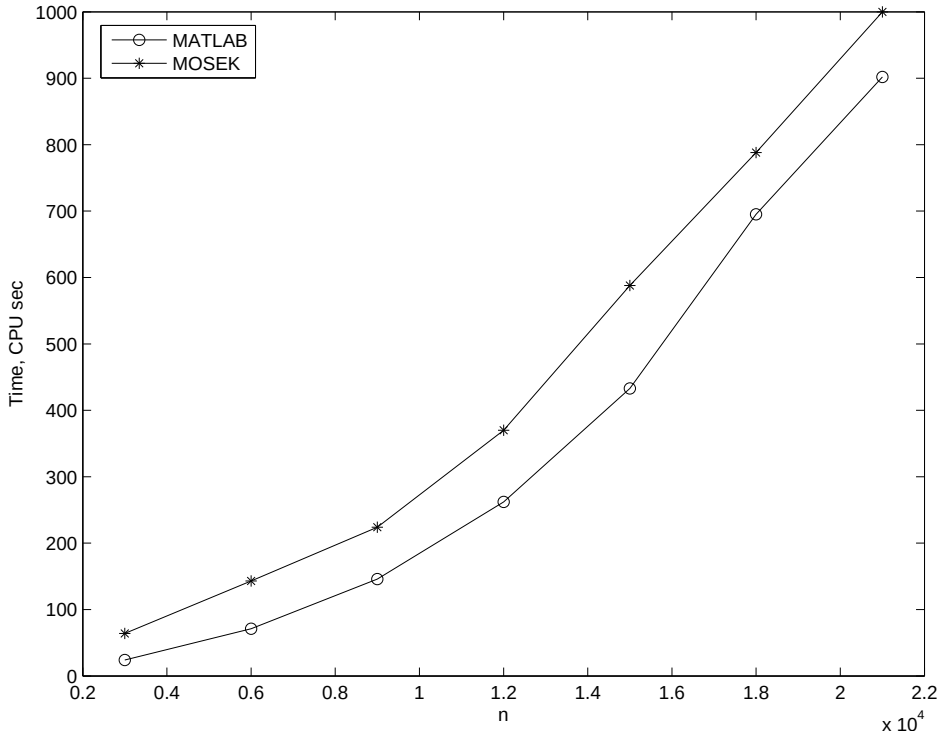

Figure 5.3: CPU for: G has 20 nonzeros per row; $m = 300$, A 1% den.; $M = 25$.

statistical observations and the *constraints* $Ax \leq b$ are of a different nature (e.g., small violation in a few constraints may be attributed to measurement errors). However, in our problems, $Ax \leq b$ represents many hard constraints such as budget requirements, no short selling etc. Moreover, in many cases no violation is tolerated. Therefore, an approach analogous to that of [18] may be useful as an initialization heuristic for our ***IPM*** .

Acknowledgments: We thank three anonymous referees for many, constructive and helpful comments, in particular, for pointing out the references [18, 19, 35].

A Transaction Costs

Recall that $\hat{x} \in \mathbb{R}^n$ represents the current holdings in assets (given). $f_i(x_i)$ denotes the transaction cost of changing the holdings in asset i from $\hat{x}_i$ to x_i. d_{il}^- and d_{il}^+ denote the absolute values of breakpoints on the transaction costs (d_{il}^- on the negative side, corresponding to selling). M_i^- denotes the number of breakpoints on the negative side (for selling asset i) and M_i^+ denotes the number of breakpoints on the positive side (for buying asset i). f_{il}^- and f_{il}^+ are the corresponding

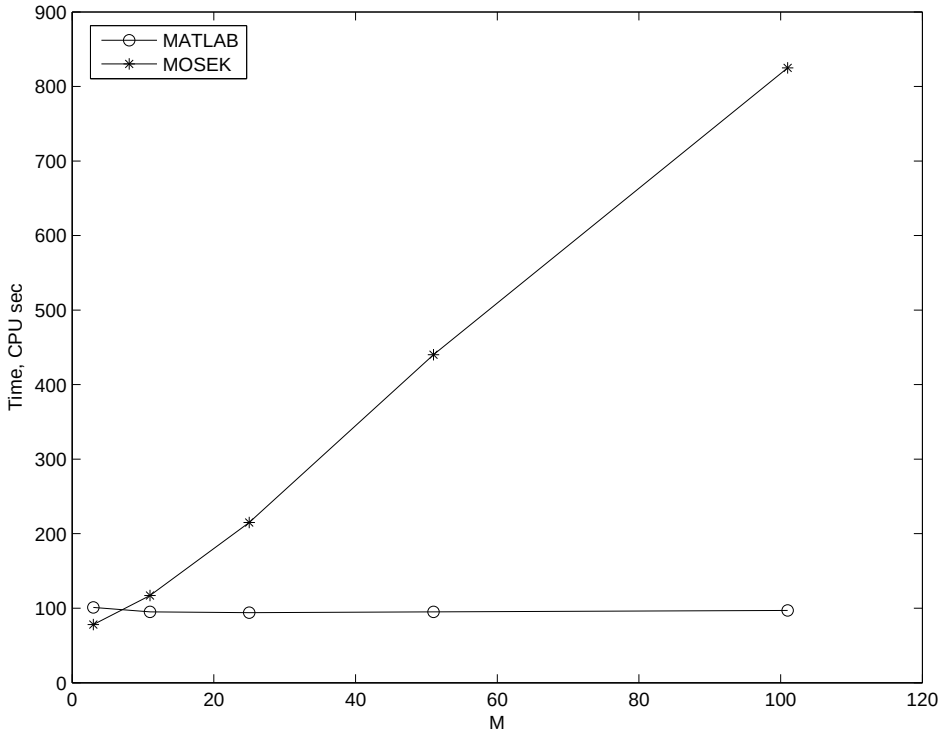

Figure 5.4: CPU (iter) for: G has 45 blocks 200×200, 10% den.; $m = 200$, A 10% den.; up/low bnds.

cost functions. So, $\hat{x}$, M_i-, M_i^+, d_{il}^-, d_{il}^+, f_{il}^- and f_{il}^+ are all given as part of the data. 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

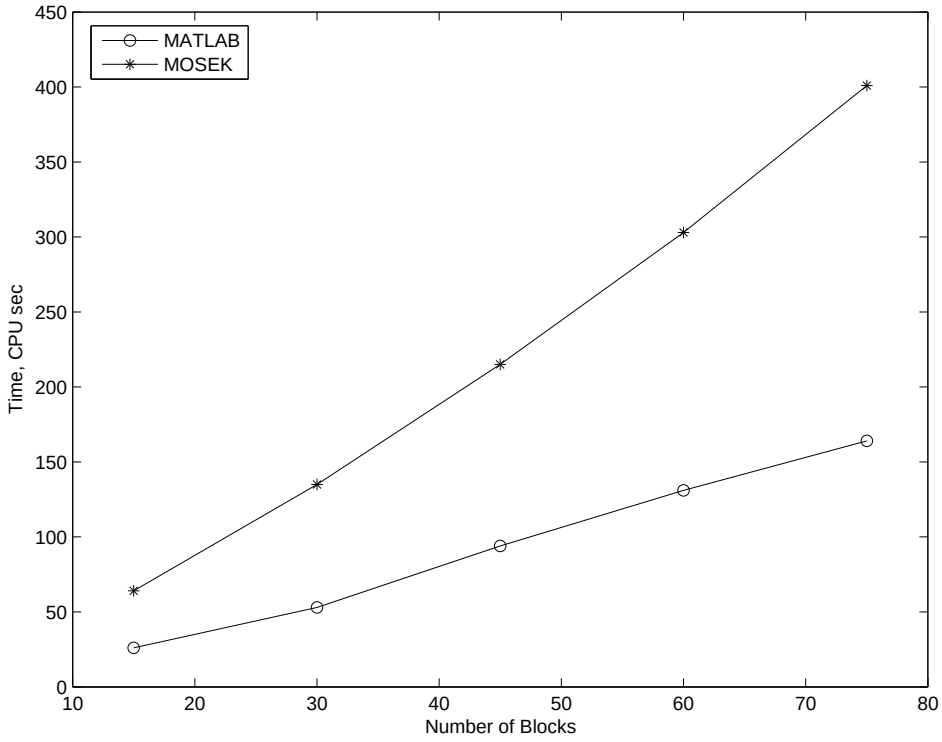

Figure 5.5: CPU (iter); G 200×200 blocks; 10% den.; $m = 200$, A 10% den.; up/low bnds, $M = 25$.

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:

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

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] M.J. BEST and J. HLOUSKOVA. Portfolio selection and transactions costs. *Comput. Optim. Appl.*, 24(1):95–116, 2003.

[4] M.J. BEST and J. HLOUSKOVA. An algorithm for portfolio optimization with transaction costs. *Management Science*, 51:1676–1688, 2005.

[5] 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.

[6] 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.

[7] G. CORNUÉJOLS and R. TÜTÜNCÜ. *Optimization methods in finance*. Mathematics, Finance and Risk. Cambridge University Press, Cambridge, 2007.

[8] 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.

[9] 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.

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

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

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

[13] A. FORSGREN, P.E. GILL, and M.H. WRIGHT. Interior methods for nonlinear optimization. *SIAM Rev.*, 44(4):525–597 (electronic) (2003), 2002.

[14] P.E. GILL, W. MURRAY, and M.H. WRIGHT. *Practical Optimization*. Academic Press, New York, London, Toronto, Sydney and San Francisco, 1981.

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

[16] 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.

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

[18] T. HASTIE, S. ROSSET, R. TIBSHIRANI, and J. ZHU. The entire regularization path for the support vector machine. *J. Mach. Learn. Res.*, 5:1391–1415 (electronic), 2003/04.

[19] S.-J. KIM, K. KOH, M. LUSTIG, S. BOYD, and D. GORINEVSKY. A method for large-scale ℓ_1-regularized least squares problems with applications in signal processing and statistics. *IEEE Journal on Selected Topics in Signal Processing*, to appear, 2008.

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

[21] A. LI and L. TUNÇEL. Some applications of symmetric cone programming in financial mathematics. *Transactions on Operational Research*, 17:1–19, 2006.

[22] M.S. LOBO. *Convex and Robust Optimization with Applications in Finance*. PhD thesis, Stanford University, USA, 2000.

[23] M.S. LOBO, M. FAZEL, and S. BOYD. Portfolio optimization with linear and fixed transaction costs. *Ann. Oper. Res.*, 157:341–365, 2007.

[24] J.E. MITCHELL and S. BRAUN. Rebalancing an investment portfolio in the presence of convex transaction costs. Technical report, Rensselaer Polytechnic, Troy, NY, 2004.

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

[26] M.L. OVERTON. Algorithms for nonlinear l_1 and l_∞ fitting. In M. J. D. Powell., editor, *Nonlinear Optimization 1981*. 1982. NATO conference series.

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

[28] M. POTAPTCHIK. *Portfolio Selection Under Nonsmooth Convex Transaction Costs*. PhD thesis, University of Waterloo, 2006.

[29] H. QI and X. YANG. Regularity and well-posedness of a dual program for convex best C^1-spline interpolation. *Comput. Optim. Appl.*, 37(3):409–425, 2007.

[30] R.T. ROCKAFELLAR. *Convex Analysis*. Princeton University Press, Princeton, NJ, 1970.

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

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

[33] N. Z. SHOR. *Minimization methods for nondifferentiable functions*. Springer-Verlag, Berlin, 1985. Translated from the Russian by K. C. Kiwiel and A. Ruszczyński.

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

[35] I. ZANG. A smoothing-out technique for min-max optimization. *Math. Programming*, 19(1):61–77, 1980.