**Operator Splitting Methods and Its Applications in Quantitative Finance**

A Dissertation presented

by

**Jiazhou Wang**

to

The Graduate School

in Partial Fulfillment of the

Requirements

for the Degree of

**Doctor of Philosophy**

in

**Applied Mathematics and Statistics**

Stony Brook University

**December 2018**

**Stony Brook University**

The Graduate School

**Jiazhou Wang**

We, the dissertation committe for the above candidate for the

Doctor of Philosophy degree, hereby recommend

acceptance of this dissertation

**Andrew Mullhaupt - Dissertation Advisor**
**Research Professor, Applied Mathematics and Statistics**

**Jiaqiao Hu - Chairperson of Defense**
**Associate Professor, Applied Mathematics and Statistics**

**Yue Zhao - Inside Committee Member**
**Assistant Professor, Electrical and Computer Engineering**

**Raphael Douady - Outside Committee Member**
**Research Professor, University Paris 1-Sorbonne**

This dissertation is accepted by the Graduate School

Dean of the Graduate School

Abstract of the Dissertation

**Operator Splitting Methods and Its Applications in Quantitative Finance**

by

**Jiazhou Wang**

**Doctor of Philosophy**

in

**Applied Mathematics and Statistics**

Stony Brook University

**2018**

Numerical method plays main role in quantitative finance, such as stochastic differential equations and portfolio optimization. In this thesis, we consider two operator splitting methods as the numerical schemes. First, we consider Forward-Backward splitting in the portfolio optimization. We propose two methods: proximal gradient Newton (PGN) method and proximal gradient diagonal Hessian (PGDH) method. These two methods have significant better performance in both execution time and accuracy than other known methods. Second, we consider Peaceman-Rachford splitting in the SDEs. We propose using Peaceman-Rachford splitting as a novel scheme solving SDEs. Numerical result shows that the proposed scheme outperforms other classical numerical schemes.

## Dedication Page

*To my family.*

# Contents

# List of Figures

# List of Tables

# List of Abbreviations

ADMM Alternating Direction Method of Multipliers, page 22

IVP Initial Value Problem, page 1

LCQP Linear Constrained Quadratic Program, page 11

ODEs Ordinary Differential Equations, page 2

PDEs Partial Differential Equations, page 2

PGDH Proximal Gradient Diagonal Hessian method, page 4

PGDP Proximal Gradient Dynamic Programming method, page 78

PGN Proximal Gradient Newton method, page 3

SDEs Stochastic Differential Equations, page 1

**Acknowledgements**

I was lucky enough to pursue my doctorate at Stonybrook University. Many exceptional people have accompanied me in this wonderful journey. Without their help and support, the completion of this thesis would not have been possible.

My deepest gratitude goes to Professor Andrew Mullhaupt giving me this opportunity and for being the best advisor ever. Andrew helped me transform from a young graduate student to an experienced researcher. Having an advisor with such a vast scientific knowledge, a great intuition and a deep insight between theory and practice was amazing! I thank him for showing me such an interesting portfolio selection problem and giving me the freedom to pursue new research directions while keeping me in touch with the reality, and for his close guidance throughout my Ph.D.

Thanks to Alphacrest for the generous financial support. The SPIR program supported by this company allowed me to focus on the research. I thank to Mika Toikka, Ed Coakley, Xiao Yu, Han Li and Ke Zhang for the help of daily works and suggestions on my research.

I am grateful to the many friends: Chi Kong, Kai Zhuang, Juehui Zhang, Ruibo Yang, Yuhang Guan, Zhi Li, Yao Kuang, Ge Song, Xingxing Ye, Matthew Johns, Eric Werneburg etc. They have made my life happy and my stay in Stonybrook memorable.

I would also like to express my gratitude and love to the better half of me, Qi Zhang. From the bottom of my heart I thank her for her constant love, patience and kindness in all these years.

Last but not least, I would like to thank to my parents Liancheng and Lihua for their unconditional love and support, and for always standing by me. I am very happy for having them in my life.

Jiazhou Wang

Stonybrook, December 2018

# 1 INTRODUCTION

## 1.1 Introduction

Modelling many problems in quantitative finance gives rise to systems of *Initial Value Problem (IVP)*

$$\frac{d\mathbf{y}}{dt} = \mathbf{f}(t, y), \quad \text{with} \quad \mathbf{y}(0) = \mathbf{y}_0 \tag{1.1}$$

One example is solving *Stochastic Differential Equations (SDEs)*

$$dX = a(t, X)dt + b(t, X)dW_t. \tag{1.2}$$

Another example is convex optimization, which can be interpreted as a discretized method for solving a differential equation

$$\frac{d}{dt}x(t) = -\nabla F(x(t)) \tag{1.3}$$

called *gradient flow* for the differentiable convex function $F$.

This dissertation is concerned with using *operator splitting methods* solve SDEs and portfolio optimization problems.

Most IVPs do not have a closed form solution. Therefore one must approximate the solutions using numerical integral (or an updating formula). One way of doing this is expected to generate a sequence of approximations $y_1, y_2, ..., y_N$ to $y_t$ at sample-times $t_0 = 0 < t_1 < t_2 < ... < t_N = T$[53]. The simplest numerical methods for IVPs are the Forward Euler methods

$$y_{n+1} = y_n + \theta_n \mathbf{f}(t_n, y_n) \tag{1.4}$$

and the Backward Euler methods

$$y_{n+1} = y_n + \theta_n \mathbf{f}(t_{n+1}, y_{n+1}) \tag{1.5}$$

In the optimization context, the Forward Euler step coincides with gradient descent methods

$$x_{k+1} = x_k - \alpha_k \nabla F(x_k) \tag{1.6}$$

and the Backward Euler step is also called the proximal point methods

$$x_{k+1} = \underset{u}{\operatorname{argmin}} \ \nabla F(u) + \frac{1}{2\alpha_k} \|u - x_k\|^2 \tag{1.7}$$

Usually the backward Euler step has both better convergence property and better numerical stability. However, in order to get the Backward Euler step, one often needs to solve a system, which is much harder than just updating the Forward Euler step. There is a way to overcome this issue when the operator $\mathbf{f}$ has a special property: it can be decomposed as the sum of two operators

$$f = g + h$$

where at least one of them has an easy updating formula (or even a closed form) of Backward Euler step. This kind of technique, which called *operator splitting methods*, has been widely used in Ordinary Differential Equations (ODEs) and Partial Differential Equations (PDEs) in the part several decades, for example, *Alternating direction implicit method*.

In this dissertation, we consider two splitting schemes. One is Forward-Backward splitting (also called proximal gradient methods in optimization context). In particular, we introduce a Hessian-free proximal gradient method which combines the modern approach of proximal algorithms with classical techniques for smooth unconstrained optimization that allow for fast asymptotic convergence. The other is Peaceman-Rachford splitting. We develop a new numerical integral scheme for SDEs and the Brownian motion in unitary group U(N) using Peaceman-Rachford splitting scheme. The new scheme has better strong convergence property than other classical schemes.

## 1.2 Contributions and organization

We start from some basic mathematics background in chapter 2. Then in Chapter 3 we describe the simple mean variance portfolio selection model for single-period trading and multi-period trading. We formulate the problem with the cost model and portfolio constraints. Then we show a standard way to solve this problem.

In Chapter 4 we show a fast algorithm for a single-instrument multi-period portfolio selection problem which is a special case of the problem defined in Chapter 3. We show that this reduced problem satisfies the *Pontryagin's maximum principle* so that it can be solved using dynamic programming idea with memoization technology. We describe the implementation in detail in order to achieve the best performance. A back test using AAPL data in Jan. 2017 shows evidence that the performance of this algorithm is significantly better than traditional quadratic programming solver.

We try to extend the dynamic programming algorithm shown in Chapter 4 to cases with a larger number of instruments. We first show an extension to two instrument, with a description of difficulty in higher dimension. Then we turn to an iterative solving technology called *proximal mapping*. We propose a proximal gradient newton method (Algorithm 5) that combine our dynamic programming solver with the proximal gradient iteration. In order to increase the performance, we introduce a step size choice framework with a merit function in each iteration. Combined these techniques together, we propose Algorithm 6 as the final solver. The numerical experiments show that our algorithm has better performance against both traditional solvers and the proximal gradient solver.

With this proximal gradient technique, we are able to solve more optimization problems in the quantitative finance area. Two examples considered in Chapter 6 are a single-period long only portfolio and a single-period long short portfolio with position constraints. The main issue in these two problems is the dimensionality (i.e. number of assets). To overcome this issue, we propose two algorithms. One is proximal gradient Newton method (PGN) in Chapter 5 with simple proximal

mapping. The other one uses the quasi-Newton idea, especially we use a diagonal hessian approximation so that each iteration has only few matrix-vector multiplications. We refer this method as proximal gradient diagonal Hessian method (PGDH) in Algorithm 7. Numerical experiments show that our algorithm has significantly better performance than other known methods.

In Chapter 8, we develop a new numerical scheme for SDEs using Peaceman-Rachford splitting. We test the new splitting scheme in three tests which are often considered in mathematical finance: geometric Brownian motion, Cox–Ingersoll–Ross model and the Brownian motion on the unitary group U(N). In these tests, we compare our proposed schemes with other classical methods such as Euler-Maruyama method, Milstein method and Euler-Heun methods. Numerical results show that the proposed scheme outperforms other classical numerical schemes.

# 2 MATHEMATICS BACKGROUND

In this chapter, we collect the mathematical materials which will be used in the following chapters. Throughout the thesis, $\mathcal{H}$ is a real Hilbert space equipped with inner product $\langle \cdot, \cdot \rangle$ with associated norm $\|\cdot\|$.

## 2.1 Semismooth Analysis

In this section we introduce some extensions of classical results in real analysis for smooth functions to nonsmooth and semismooth functions. These tools are developed when considering the numerical methods for solving the systems of semismooth equations of the form

$$G(x) = 0 \tag{2.1}$$

where $G : \Omega \subset \mathbb{R}^n \to \mathbb{R}^m$ is locally Lipshchitz on the open set $\Omega$. The results in this section are fundamental and can be found in [32]

**Definition 2.1** (B-subdifferential)**.** Let $F : \mathbb{R}^n \to \mathbb{R}^m$ be a given function that is locally Lipschitz continuous in a neighborhood $\mathcal{N} \in \mathbb{R}^n$ of a vector $x \in \mathbb{R}^n$. Define the *B-subdifferential* (or *limiting Jacobian*) of $F$ of $x$ is

$$\partial_B F(x) := \{H \in \mathbb{R}^{n \times m} \mid \exists\, x^k \subset C_F \text{ with} x^k \to x, J_F \to H\} \tag{2.2}$$

where $C_F$ is the subset of points in $\mathbb{R}^n$ such that $F$ is differentiable, and $J_F(x)$ is the classical Jacobian of $F$ at point $x$.

An intuitive interpolation of B-subdifferential is that pick a sequence $\{x^k\}$ where the sequence converges to $x$, and $F$ is differentiable at each point of $\{x^k\}$, then the limit value of the Jacobian of $F$ is in the set of B-differential of F.

**Example 2.2** (B-differential of absolute value)**.** The B-subdifferential

of absolution function on the real line

$$f(x) = |x|\,, \quad x \in \mathbb{R}^1 \tag{2.3}$$

is as follows

$$\partial_B f(x) = \begin{cases} 1, & x > 0 \\ \{-1, 1\}, & x = 0 \\ -1, & x < 0 \end{cases}$$

It is easy to see that the B-differential at origin has two elements since one can approach to the origin from either positive part or the negative part of the real line.

With the definition of B-differential, we are ready to introduce the Clarke *generalized Jacobian*.

**Definition 2.3** (Clarke generalized Jacobian)**.** Let $F : \mathbb{R}^n \to \mathbb{R}^n$ be locally Lipschitz at $x \in \mathbb{R}^n$. The Clarke *generalized Jacobian* of $F$ at $x$ is:

$$\partial_C F(x) = \text{conv } \partial_B F(x) \tag{2.4}$$

where conv$C$ is the convex hull of set $C$. When $m = 1$, Clarke *generalized Jacobian* reduces to *generalized gradient* (also called *subgradient*).

**Example 2.4** (Generalized gradient of absolute function)**.** The generalized gradient of absolution function (2.3) is

$$\partial_C f(x) = \begin{cases} 1, & x > 0 \\ [-1, 1], & x = 0 \\ -1, & x < 0 \end{cases}$$

we refer this function as sign function sign$(x)$.

The Clarke generalized Jacobian plays an important rule in Newton-type algorithm for solving semismooth system. Algorithm 1 gives a simple semismooth Newton method for solving (2.1). It requires the

**Algorithm 1** Semismooth Newton Method

**Require:** $G,\ x^0, \epsilon > 0$;
**Ensure:** $x$;
tolerance $= \epsilon * 10$;
$k = 0$;
**while** tolerance $> \epsilon$ **do**
  Select an element $H^k \in \partial_c G(x^k)$. Find a direction such that

$$G(x^k) + H^k d^k = 0. \tag{2.5}$$

  $x^{k+1} = x^k + d^k$;
  tolerance $= \left\| d^k \right\|$
  $k = k + 1$;
**end while**
**return** $u^{k+1}$;

computation of Newtion direction for each iteration. The system (2.5) has unique solution when $H^k$ is nonsingular. We present the following Lemma which ensures the existence of such a nonsingular $H^k$.

**Lemma 2.5** ([32] 7.5.2)**.** Let $J : \Omega \subset \mathbb{R}^n \to \mathbb{R}^{n\times n}$, with $\Omega$ open, be a compact-valued, upper semicontinuous set-valued mapping. Suppose that a point $\bar{x} \in \Omega$ all the matrices in $J(\bar{x})$ are nonsingular. There exist positive constants $\kappa$ and $\delta$ such that

$$\sup_{x\in\mathbb{B}(\bar{x},\delta), H\in J(x)} \max\{\|H\|, \left\|H^{-1}\right\|\} \leq \kappa.$$

In particular, all the matrices $H \in J(x)$ for $x \in \mathbb{B}(\bar{x}, \delta)$ are nonsingular.

*Proof.* Since $J$ is upper semicontinuous and compact, for every positive $\epsilon$, there exists a positive $\delta$ such that $J(x)$ is contained in $J(\bar{x}) + \mathbb{B}(0, \epsilon)$ for every $x \in \mathbb{B}(\bar{x}, \delta)$. The lemma follows readily from this observation and from the continuity of the determinant. □

The following theorem ensures that algorithm 1 has linear (even quadratic) convergence rate with a suitable condition on $G$.

**Theorem 2.6** ([32] 7.5.3)**.** Let $G : \Omega \subset \mathbb{R}^n \to \mathbb{R}^n$, with $\Omega$ open, be semismooth at $x^* \in \Omega$ satisfying $G(x^*) = 0$. If $\partial G(x^*$ is nonsin-

gular, then there exist a $\delta > 0$ such that, if $x^0 \in \mathbb{B}(x^*, \delta)$, the sequence $\{x^k\}$ generated by Algorithm 1 is well defined and convergences Q-superlinearly to $x^*$. Furthermore, if $G$ is strongly semismooth at $x^*$, then the convergence rate is Q-quadratic.

We omit the proof and refer the chapter 7 of [32] for the detail.

## 2.2 Convex analysis

In this section we introduce the fundamental notions, results and examples from convex analysis, convex optimization which will be used in the following chapters. These results are standard, and can be found for instance in [4]

### 2.2.1 Convex sets

**Definition 2.7** (Convex set)**.** A set $\mathcal{C}$ of $\mathcal{H}$ is *convex* if, for every pair of points $x, y \in C$, the points on the line segment joining $x$ and $y$ are in the set $C$, i.e.

$$tx + (1-t)y \in C, \quad \forall t \in (0,1)$$

**Example 2.8** (Ball)**.** A ball in Hilbert space

$$C := \{x \mid \|x - x_0\| \leq r\}$$

is a convex set.

**Example 2.9** (Box)**.** A box in Hilbert space

$$C := \{x \mid l \leq x \leq u\}$$

is a convex set.

A fundamental but very important result the intersection of convex sets is still convex:

**Proposition 2.10** (Set intersections[71])**.** The intersection of an arbitrary collection of convex sets is convex.

### 2.2.2 Convex functions

**Definition 2.11** (Convex function)**.** A function $f : \mathcal{H} \to \bar{\mathbb{R}}$ if $dom(f)$ is convex and, for every pair of points $x, y \in \mathcal{H}$,

$$f(tx + (1-t)y) \leq tf(x) + (1-t)f(y), \quad \forall t \in (0,1)$$

A convex function is *proper* if

$$f(x) > -\infty$$

for every $x$ and

$$f(x) < +\infty$$

for at least one $x$. The the set of points for which $f$ takes on finite values is called the *effective domain* of $f$, i.e.

$$\mathrm{dom}(f) := \{x \in \mathcal{H} \mid f(x) < +\infty\}$$

The *epigraph* of $f$ is defined as

$$\mathrm{epi}(f) := \{(x, y) \in C \times \mathbb{R} \mid x \in \mathrm{dom}(f),\ f(x) \leq y\}$$

Note that the epigraph of a closed proper convex function is a nonempty closed convex set.

**Example 2.12** (Indicator function)**.** Let $C \subseteq \mathcal{H}$ be a non-empty closed convex set, define the indicator function $\mathcal{I}_C$ corresponding to $C$ as:

$$\mathcal{I}_C(x) := \begin{cases} 0, & \text{if } x \in C, \\ +\infty, & \text{otherwise} \end{cases}$$

For the convenience of the computation in the following chapter, we

extend the generalized Jacobian of indicator function as follows:

$$\partial_C \mathcal{I}_C(x) := \begin{cases} J_{\mathcal{I}_C(x)}, & x \in \text{interior}(C) \\ \text{conv } (\partial_B \mathcal{I}_C(x) \cup \{+\infty\}), & x \in \partial C \\ +\infty, & \text{otherwise} \end{cases} \tag{2.6}$$

Note that we add all the directions where $\mathcal{I}_C$ does not have finite representation to the B-differential, and we assign the generalized Jacobian to $+\infty$ where the value of indicator function is $+\infty$. These operations are safe because they do not exclude any elements of $\partial_C \mathcal{I}_C(x)$ when $x \in C$.

**Definition 2.13** (Proper convex function)**.** A convex function $f : \mathbb{R}^n \to \mathbb{R}^1 \cup \infty$ is proper if there is at least one $x \in \mathbb{R}^n$ such that $f(x)$ is finite and for any $x \in \mathbb{R}^n$, $f(x) > -\infty$.

### 2.2.3 Strong convexity

**Definition 2.14** (Strong convexity)**.** A differentiable function $f : C \to \bar{\mathbb{R}}$ is *strong convex* with parameter $\mu$ ($\mu > 0$) if, for every pair of points $x \in \text{dom}(f)$ and $y \in C$,

$$f(y) \geq f(x) + \nabla f(x)^T (y - x) + \frac{\mu}{2} \|y - x\|^2$$

## 2.3 Convex Programming

In this section we give a brief review of convex optimization problems. Recall that a *convex optimization problem* is a minimization problem of the form

$$\begin{aligned} \underset{x}{\text{minimize}} \quad & f(x) \\ \text{subject to} \quad & g_i(x) \leq 0, \quad i = 1, 2, ..., n \\ & h_k(x) = 0, \quad k = 1, 2, ..., m \end{aligned} \tag{2.7}$$

where each function $g_i : C \to \bar{\mathbb{R}}$ is convex, and each function $h_k : C \to \bar{\mathbb{R}}$ is affine. The function $f$ is called *objective function*, while the equations

$g_i(x) \leq 0$ and $h_k(x) = 0$ are referred as *constraints*. The set of points that satisfy all the constraints is called the *feasible set*. If there is not point satisfy all the constraints, we say that the underlying problem is *infeasible*.

**Example 2.15** (Linear constrained quadratic programming)**.** The general linear constrained quadratic program (LCQP) can be stated as

$$\begin{aligned} \underset{x}{\text{minimize}} \quad & \frac{1}{2}x^T Hx + f^T x \\ \text{subject to} \quad & Gx \leq h \\ & Ax = b \end{aligned} \tag{2.8}$$

where $H \in \mathbb{R}^{n\times n}$ is symmetric positive semi-definite matrix, $G \in \mathbb{R}^{m\times n}$ is a $m$ by $n$ matrix, $A \in \mathbb{R}^{p\times n}$ is a $p$ by $n$ matrix, and $x \in \mathbb{R}^n$, $f \in \mathbb{R}^n$, $h \in \mathbb{R}^m$, $b \in \mathbb{R}^p$ are vectors.

### 2.3.1 Karush–Kuhn–Tucker conditions

Karush-Kuhn-Tucker conditions (KKT conditions) are first-order necessary conditions for a solution in nonlinear programming to be optimal. We first introduce the Lagrangian function, then shows:

**Definition 2.16** (Lagrangian function)**.** Given a convex optimization problem

$$\begin{aligned} \underset{x}{\text{minimize}} \quad & f(x) \\ \text{subject to} \quad & g_i(x) \leq 0, \quad i = 1, 2, ..., n \\ & h_k(x) = 0, \quad k = 1, 2, ..., m \end{aligned}$$

the corresponding Lagrangian function is defined as

$$\mathcal{L}(x, \lambda, \mu) = f(x) + \sum_{i=1}^{n} \lambda_i g_i(x) + \sum_{k=1}^{n} \mu_k h_k(x)$$

Lagrangian function introduce new variables $\lambda$ and $\mu$(also called Lagrange multipliers or dual variables) into the problem. This leads to

a Lagrangian dual function[61]

$$q(\lambda,\mu) := \inf_x \mathcal{L}(x,\lambda,\mu) = \inf_x \{f(x) + \sum_{i=1}^{n} \lambda_i g_i(x) + \sum_{k=1}^{n} \mu_k h_k(x)\}$$

and a *Lagrangian dual problem*:

$$\begin{aligned} \underset{\lambda,\mu}{\text{maximize}} \quad & q(\lambda,\mu) \\ \text{subject to} \quad & \lambda \geq 0 \\ & \mu \geq 0 \end{aligned} \tag{2.9}$$

We refer (2.7) as the primal problem, which gives us the exact solution of the target problem, and (2.9) as the corresponding dual problem. Note that the solution of the dual problem (2.9) provides a lower bound to the solution of the primal problem[11]. The difference between the objective function values of the primal problem and the dual problem is called *duality gap*.

Now we are ready to introduce the KKT conditions:

**Definition 2.17** (KKT conditions[11])**.** Let $x^*$ be the primal optimal point of 2.7, $(\lambda^*, \mu^*)$ the dual optimal point of 2.9 with zero duality gap. The *Karush–Kuhn–Tucker conditions* is defined as

$$\begin{aligned} \nabla f(x) + \sum_{i=1}^{n} \lambda_i g_i(x) + \sum_{k=1}^{m} \mu_k h_k(x) &= 0 \\ g_i(x) &\leq 0, \quad i = 1,2,3,...,n \\ g_k(x) &\leq 0, \quad m = 1,2,3,...,m \\ \lambda_i &\geq 0, \quad i = 1,2,3,...,n \\ \lambda_i g_i(x) &= 0, \quad i = 1,2,3,...,n \end{aligned} \tag{2.10}$$

### 2.3.2 Classical methods for solving QP

Here we introduce two classes of methods solving general QP which are used in the commercial software MATLAB. One is called *trust region reflective methods*, the other one is called *interior point methods*. These

two methods will be compared as the benchmarks in the numerical experiments chapter.

Given objective function $f(x)$ associated with constraints, the trust region method keep tracking a simpler model $q(x)$, as a resonable approximation of $f(x)$ with a neighborhood $\mathcal{N}$, called *trust region*. A trial step $s$ is computed by solving a trust-region subproblem

$$\begin{aligned} \underset{s}{\text{minimize}} \quad & q(s) \\ \text{subject to} \quad & s \in \mathcal{N} \end{aligned}$$

A simple but powerful approximation model is using Hessian $H$ and the gradient $g$:

$$q(s) = \frac{1}{2} x^T H s + g^T s$$

The trial step is accepted when the objective function value is decreased, i.e.

$$f(x + s) < f(x)$$

The neighborhood $\mathcal{N}$ is updated after the trial according to the standard rules. The algorithm runs all the procedures above iteratively until convergence. For more details, we refer [61].

The interior point methods extend the barrier methods which were developed in the 1960s. One practical interpretation is that the methods use slack variables transfer the original problem to an equivalent problem and then solve the equivalent problem iteratively with Newton's method. As an example, for a generalized convex optimization problem 2.7, we can re-write the problem using slack variables as follows:

$$\begin{aligned} \underset{x,s}{\text{minimize}} \quad & f(x) \\ \text{subject to} \quad & g_i(x) + s_i = 0, \quad i = 1, 2, ..., n \\ & h_k(x) = 0, \quad k = 1, 2, ..., m \\ & s_i \geq 0, \quad i = 1, 2, ..., n \end{aligned} \tag{2.11}$$

According to (2.17), the KKT conditions can be written as

$$\begin{aligned} \nabla f(x) + J_g^T(x)\mu + J_h^T(x)\lambda &= 0 \\ D_s\lambda - e &= 0 \\ h(x) &= 0 \\ g(x) + s &= 0 \\ s &\geq 0 \\ \lambda &\geq 0 \end{aligned} \tag{2.12}$$

where $J_g(x)$ and $J_h(x)$ are Jacobian matrices of the functions $g(x)$ and $h(x)$, and $D_s$ is a diagonal matrix whose diagonal entries are given by the vector $s$. Applying Newton's method, we can drive a linear system from (2.12):

$$\begin{bmatrix} \nabla^2_{xx}\mathcal{L} & 0 & J_g^T(x) & J_g^T(x) \\ 0 & D_\lambda & 0 & D_s \\ -J_g^T(x) & 0 & 0 & 0 \\ -J_g^T(x) & I & 0 & 0 \end{bmatrix} \begin{bmatrix} p_x \\ p_s \\ p_\mu \\ p_\lambda \end{bmatrix} = - \begin{bmatrix} \nabla f(x) + J_g^T(x) + J_h^T(x) \\ D_s\lambda - e \\ h(x) \\ g(x) + s \end{bmatrix} \tag{2.13}$$

where $\mathcal{L}$ denotes the Lagrangian for (2.12), and $D_\lambda$ is a diagonal matrix whose diagonal entries are given by $\lambda$. The procedure for one iteration includes solving (2.12) to get a Newton's search direction and using line search method update the current trail. We repeat the iteration until a stopping criterion is satisfied.

## 2.4 Convex composite minimization form

The general quadratic programming problem can be written as a composite minimization form:

$$\underset{x}{\text{minimize}} \quad h(x) + g(x) \tag{2.14}$$

where $h(x)$ is a convex smooth function, and $g(x)$ is a convex (possibly nonsmooth) function.

**Example 2.18.** The box constrained quadratic programming problem

$$\begin{aligned} \underset{x}{\text{minimize}} \quad & F(x) = \frac{1}{2}x^T Hx + f^T x \\ \text{subject to} \quad & l \leq x \leq u \end{aligned} \tag{2.15}$$

has the following composite minimization form:

$$\begin{aligned} \underset{x}{\text{minimize}} \quad & \tilde{F}(x) = h(x) + g(x) \\ \text{subject to} \quad & h(x) = \frac{1}{2}x^T Hx + f^T x \\ & g(x) = \mathcal{I}_{l \leq x \leq u} \end{aligned} \tag{2.16}$$

Convex composite minimization form are also used in other applications. Regularization methods in signal processing and machine learning often lead to solving a convex optimization problem of the form (2.14). In that context, $f$ usually represents the convex loss function, for example, $\mathcal{L}^2$ loss function, and $g$ is the regularization function (also called penalty function) in order to achieve some desired properties. Examples are provided as the following.

**Example 2.19** (Lasso regression)**.** The *lasso* model for sparse linear regression solves the following quadratic programming problem

$$\underset{x}{\text{minimize}} \quad \|Ax - b\|_2^2 + \lambda \|x\|_1$$

Here the smooth function $f(x)$ is the $\mathcal{L}^2$ norm as the loss function for the given data points and the non-smooth function $g(x)$ is the $l^1$ norm.

**Example 2.20** (Ridge regression)**.** The *ridge regression* (also known as Tikhonov regularization) for statistics solves the following quadratic programming problem

$$\underset{x}{\text{minimize}} \quad \|Ax - b\|_2^2 + \|\Gamma x\|_2^2$$

Ridge regression is the most commonly used method to avoid over-fitting.

Notice that the composite minimization form is not unique. For example, in (2.16) $h(x)$ and $g(x)$ can some parts between each other, like the affine part $f^T x$. This will give us the identical solution. The convex composite minimization form has several advantages. One is that this form avoids auxiliary variables or Lagrange multipliers when we write down the KKT conditions. Another advantages, which is the main advantages, is that convex composite minimization form leads to a broad family of so called *operator splitting methods*.

## 2.5 Operator splitting methods in ODE

Operator Splitting is a class of methods which are commonly used in solving differential equations and linear or nonlinear systems. The idea is natural for solving a physical processes: when a system, a reaction-diffusion equation, for example, contains different parts of terms representing different physics, one simple idea is solving the different physical processes individually to simplify the overall process. In this section we start from an example of operator splitting methods in ordinary differential equations(ODE), and then extend this idea to a general convex optimization problem. These are standard results, and can be found in [34]

Given a initial value problem:

$$\begin{cases} \frac{dX}{dt} + (A+B)X = 0, \ in \ (0,T), \\ X(0) = X_0 \end{cases} \tag{2.17}$$

where $A, B$ are $d \times d$ matrices, $X_0 \in \mathbb{C}^d$ is a d-dimensional complex valued vector, $T \in (0, +\infty) \cup +\infty$ is ending time. One example is the reaction-diffusion system:

$$\frac{\partial u}{\partial t} = D\nabla^2 u + R(u) \tag{2.18}$$

where the system consists of two parts: the diffusion operator $D\nabla^2$ and the reaction operator $R$.

The symbolic solution of (2.17) is

$$X(t) = e^{-(A+B)t}X_0 \tag{2.19}$$

From the Lie product formula [40], we know that for arbitrary $n \times n$ complex matrices $A$ and $B$,

$$e^{A+B} = \lim_{n\to+\infty}(e^{\frac{A}{n}}e^{\frac{B}{n}})^n \tag{2.20}$$

This leads to an approximation solution of the problem (2.17):

$$\begin{cases} X^0 = X_0 \\ X^{n+1} = e^{-A\Delta t}e^{-B\Delta t}X^n,\ n = 0, 1, 2, ... \end{cases} \tag{2.21}$$

where $\Delta t = T/n$ is the discretized time step. (2.21) is called Lie's scheme. One can achieve first order accuracy using the linear approximation for the exponential map in the Lie scheme. And Lie's scheme is exact when the operator $A$ and $B$ are commute. As a simple extension, the Lie's scheme for multiple (more than 2) operators is defined as follows:

$$\begin{cases} X^0 = X_0 \\ X^{n+1} = e^{-A_m\Delta t}e^{-A_{m-1}\Delta t}...e^{-A_1\Delta t}X^n,\ n = 0, 1, 2, ... \end{cases} \tag{2.22}$$

One splitting scheme which is widely used in hyperbolic problems is Strang splitting (also called leapfrog method)[77]:

$$\begin{cases} X^0 = X_0, \\ X^{n+1} = e^{-A\frac{\Delta t}{2}}e^{-B\Delta t}e^{-A\frac{\Delta t}{2}}X^n,\ n = 0, 1, 2, ... \end{cases} \tag{2.23}$$

The Strang splitting is second order accurate by showing that

$$e^{-(A+B)\Delta t} - e^{-A\frac{\Delta t}{2}}e^{-B\Delta t}e^{-A\frac{\Delta t}{2}} = O(\Delta t^3)$$

Another important type of splitting scheme is called Alternating direction implicit (ADI) method. ADI method was introduced by J. Douglas, D. Peaceman and H. Rachford 60 years ago for solving the elliptic and parabolic equations such as the heat conduction problem. There are two kinds of splitting scheme for ADI methods: one the the *Peaceman-Rachford scheme*[68], the other is *Douglas-Rachford scheme*[27]. These two schemes are important since they lighten two kinds of optimization methods. So we describe the basic ideas of them below: Recall that the initial value problem:

$$\begin{cases} \frac{dX}{dt} + A(X,t) + B(X,t) = 0, \ in \ (0,T), \\ X(0) = X_0 \end{cases} \tag{2.24}$$

with an updating formula $Q$ on the time interval $[t^n, t^{n+1}]$

$$X^{n+1} = Q(X^n, t^{n+1} - t^n) \tag{2.25}$$

The *Peaceman-Rachford* scheme performs a half step update on one operator $A$ using *backward Euler* updating formula and a half step update on the other operator $B$ using *forward Euler* updating formula on $[t^n, t^{n+\frac{1}{2}}]$

$$X^{n+\frac{1}{2}} = (\mathbb{1} + \frac{\Delta t}{2}A)^{-1}(\mathbb{1} - \frac{\Delta t}{2}B)X^n \tag{2.26}$$

which is essentially solving the equation

$$X^{n+\frac{1}{2}} - X^n + \frac{\Delta t}{2}A(X^{n+\frac{1}{2}}, t^{n+\frac{1}{2}}) + \frac{\Delta t}{2}B(X^n, t^n) = 0 \tag{2.27}$$

for $X^{n+\frac{1}{2}}$. Then it switches the updating role of $A$ and $B$ on the second half interval $[t^{n+\frac{1}{2}}, t^{n+1}]$

$$X^{n+1} = (\mathbb{1} + \frac{\Delta t}{2}B)^{-1}(\mathbb{1} - \frac{\Delta t}{2}A)X^{n+\frac{1}{2}} \tag{2.28}$$

which is corresponding to the solution of the following equation

$$X^{n+1} - X^{n+\frac{1}{2}} + \frac{\Delta t}{2}A(X^{n+\frac{1}{2}}, t^{n+\frac{1}{2}}) + \frac{\Delta t}{2}B(X^{n+1}, t^{n+1}) = 0 \tag{2.29}$$

Combining (2.26) and (2.28), we have the full update formula

$$\begin{cases} X^{n+1} = Q(X^n, \Delta t) \\ Q = (\mathbb{1} + \frac{\Delta t}{2}B)^{-1}(\mathbb{1} - \frac{\Delta t}{2}A)(\mathbb{1} + \frac{\Delta t}{2}A)^{-1}(\mathbb{1} - \frac{\Delta t}{2}B)X^n \end{cases} \tag{2.30}$$

The *Douglas-Rachford* scheme has similar idea with *Peaceman-Rachford* scheme. On the time interval $[t^n, t^{n+1}]$, *Douglas-Rachford* scheme first performs a full step update on operator A using *backward Euler* updating formula and a full step update on B using *forward Euler* updating formula

$$\hat{X}^{n+1} = (\mathbb{1} + \Delta t A)^{-1}(\mathbb{1} - \Delta t B)X^n, \tag{2.31}$$

which is equivalent with solving the system

$$\hat{X}^{n+1} - X^n + \Delta t A(\hat{X}^{n+1}, t^{n+1}) + \Delta t B(X^n, t^n) = 0, \tag{2.32}$$

then it performs another *backward Euler* update on $B$ based on $\hat{X}^{n+1}$

$$X^{n+1} = (\mathbb{1} + \Delta t B)^{-1}(X^n - \Delta t A(\hat{X}^{n+1})) \tag{2.33}$$

by solving the system

$$X^{n+1} - X^n + \Delta t A(\hat{X}^{n+1}, t^{n+1}) + \Delta t B(X^{n+1}, t^{n+1}) = 0. \tag{2.34}$$

The full updating formula of *Douglas-Rachford* scheme is

$$X^{n+1} = (\mathbb{1} + \Delta t B)^{-1}[\mathbb{1} - \Delta t A(\mathbb{1} + \Delta t A)^{-1}(\mathbb{1} - \Delta t B)]X^n \tag{2.35}$$

The *Peaceman-Rachford* scheme is second order accurate when the linear operators $A$ and $B$ are commute, and is at best first order accurate in general. The *Douglas-Rachford* scheme is generically first order accurate at best, too. We omit the details of the convergence proof and refer [55] to readers who are interested in it.

## 2.6 Operator splitting methods in Convex optimization

In this section we introduce several algorithms in convex optimization which share the same formulation with the operator splitting scheme we mentioned in previous section.
Given a convex optimization problem

$$\underset{x\in\Omega}{\text{minimize}} \quad F(x) \tag{2.36}$$

One simple algorithm is the *subgradient method*[8]

$$x^{k+1} = x^k - \gamma_k p_k,\ p_k \in \partial F(x^k),\ \gamma_k > 0 \tag{2.37}$$

When the subdifferential mapping $\partial F(x^k)$ has unique value, the *subgradient method* reduces to the gradient descent method. The *subgradient method* has a wide range of applications since it is easy to understand and implement, but it has relatively poor performance even with precise choices on the step-size parameter $\gamma_k$ (see [8] for details). One can rewrite (2.37) as

$$x^{k+1} \in (\mathbb{1} - \gamma_k \partial F)(x^k),\ \gamma_k > 0 \tag{2.38}$$

The updating formula (2.38) shows that the *subgradient method* actually performs a forward Euler step.

Another method which can obtain much stronger convergence is called the proximal point algorithm[70]

$$\begin{aligned} x^{k+1} &= \text{prox}_{\gamma_k F}(x^k) \\ &= \underset{u}{\text{argmin}}\{F(u) + \frac{1}{2\gamma_k}\left\|u - x^{k+1}\right\|^2\} \end{aligned} \tag{2.39}$$

where the mapping

$$\text{prox}_{\gamma f}(x) = \underset{u}{argmin} \quad \{f(u) + \frac{1}{2\gamma}\|u - x\|^2\} \tag{2.40}$$

is called *proximal mapping*[57]. In the optimization context, the parameter $\gamma$ in 2.39 is the step-size parameter: when $\gamma$ is small, the next trail point $x^{k+1}$ is close to the previous choice $x^k$; and $x^{k+1}$ gets closer to the minimizer of $f(u)$ as $\gamma$ becomes larger. The proximal point algorithm essentially performs a backward euler step (see Appendix for detail), and it has stronger convergence properties than subgradient descent. This is consistent with the well known result in ODE. However, in order to get the next trail point, proximal point algorithm requires computing the proximal mapping, which is the solution a subproblem, in each iteration. The subproblem is easy to solve for some functions[64], but as hard as the original problem in general.

To overcome the difficulty of computing proximal mapping, splitting algorithms have been proposed. Instead of computing the proximal mapping of the whole function $F(x)$, splitting algorithms decompose $F$ into two parts $f$ and $g$,

$$F(x) = f(x) + g(x),$$

and only evaluate the proximal mapping of $f$ or $g$ or both of them individually. Here we show two main splitting schemes using in the following chapters. Readers are referred [4] for the theoretical analysis on the proximal algorithms and operator splitting methods.

**Forward-backward splitting(FBS).** When the gradient of $f$ is Lipschitz continuous, and the proximal mapping of $g$ can be evaluated efficiently, one can solve the problem (2.14) by alternating the gradient steps (or the forward step) on $f$ and the proximal mapping steps (or the backward step) on $g$, i.e.

$$x^{k+1} = \text{prox}_{\gamma g}(x^k - \nabla f(x^k),\ \gamma > 0. \tag{2.41}$$

This type of iterative formula has been invented and discussed in different fields and contexts, for example, ISTA[21], FISTA[6] and the reference therein.

FBS is known to converge when the step-size parameter $\gamma$ is in a

suitable range. [4] gives the convergence proof when $\gamma \in (0, 2/L_f)$, where $L_f$ is the Lipschitz constant of $\nabla f$. The global sublinear convergence rate and its fast variants are provided in [60, 59, 6]. A complete guide for forward backward splitting can be found in [64], and [35] for the efficient implementation details.

**Douglas-Rachford splitting.** The Douglas-Rachford splitting can also be applied under optimization context[55]. This scheme runs the proximal mapping steps on both $f$ and $g$ individually,

$$\begin{aligned} y^k &= \text{prox}_{\gamma f}(x^k) \\ z^k &= \text{prox}_{\gamma g}(2y^k - x^k) \\ x^{k+1} &= x^k + \lambda_k(z^k - y^k) \end{aligned} \tag{2.42}$$

where $\gamma > 0$ and the stepsize $\lambda_k \in (0, 2]$ satisfies[75]

$$\sum_{i=1}^{+\infty} \lambda_k(2 - \lambda_k) = +\infty.$$

A simple choice is $\lambda_k = 1$ $(k = 1, 2, ...)$. The DRS scheme has been shown to be eqivualent to a well-known algorithm called *alternating direction method of multipliers* (ADMM) on the equality constrained convex problems

$$\begin{aligned} \underset{x}{\text{minimize}} \quad & f(x) + g(z) \\ \text{subject to} \quad & Ax + Bz = b \end{aligned} \tag{2.43}$$

The convergence proof is provided in [30, 22, 29].

**Remarks**.Many the examples of proximal mapping are well-known in the literature; see, e.g., Vandenberghe [81], Bauschke [3], Bauschke and Combettes [4], Combettes and Pesquet [20], Zarantonello [85], and Boyd and Vandenberghe [11]. Many of the results on norms, especially the $l1$ norm and variants, come from the statistical and machine learning literature. See, for example, references on the lasso [80], soft thresholding [26], group lasso [84], sum-of-norms regularization [62], the CAP family of penalties [86], $l1$ trend filtering [49], covariance selection [24], sparse recovery [43], basis pursuit [18], Huber loss [45], singular value thresh-

olding [15], and sparse graph selection [56]. There is a vast literature on projection operators and proximal operators for more exotic sets and functions. For a few representative examples, see, e.g., [5, 74, 69, 58, 39]. We also highlight the paper by Chiercia et al. [19], which explores the connection between the proximal operator of a function and the projection onto its epigraph.

# 3 MEAN-VARIANCE PORTFOLIO THEORY

In this chapter we review the so-called Mean-variance portfolio theory and give some detail of our portfolio selection problem in both single-period and multi-period settings. We introduce our transaction cost model from practical observation and theoretical derivation.

## 3.1 Single-period mean-variance portfolio optimization

We consider a capital market with $n$ risky assets with random rate of returns. We denote $r_i$ as the rate of return of $i$th assets and $r = [r_1, r_2, ..., r_n]^T$ as the vector of these returns. A portfolio is represented by the $n$-dimensional vector $u = [u_1, u_2, ..., u_n]^T$ where $u_i$ is the inventory (in exact dollar value) of $i$th asset invected by the investor. The total return (in dollar) of the portfolio, denoted by $\omega_p$, is a linear combination of the inventories

$$\omega_p(u) = r_1u_1 + r_2u_2 + ... + r_nu_n \tag{3.1}$$

As the measure of risk, we denote $\sigma_{i,j}$ as the covariance between $r_i$ and $r_j$, and $\Sigma$ the symmetric $n \times n$ covariance matrix of the return vector $r$

$$\Sigma = \begin{bmatrix} \sigma_{1,1} & \sigma_{1,2} & ... & \sigma_{1,n} \\ \sigma_{2,1} & \sigma_{2,2} & ... & \sigma_{2,n} \\ ... & ... & ... & ... \\ \sigma_{n,1} & \sigma_{n,2} & ... & \sigma_{n,n} \end{bmatrix}$$

Notice that $\sigma_{i,i} = \sigma_i^2$ is the variance of $i$th instrument, and $\sigma_{i,j} = \sigma_{j,i}$. All valid covariance matrices are positive semidefinite, i.e. $u^T\Sigma u \geq 0$ for all $u$, which means all the eigenvalues of $\Sigma$ are non-negative. We denote

the variance of the portfolio return as

$$V(u) = u^T \Sigma u$$

and the volatility of the portfolio return as the standard deviation of portfolio return, which is the square root of the variance

$$v(u) = \sqrt{V(u)} = \sqrt{u^T \Sigma u}$$

We let $\mu = [\mu_1, \mu_2, ..., \mu_n]^T$ represent the expected return of assets, where $\mu_i = E(r_i)$ for $i = 1, 2, ..., n$. Under this setting, the mean-variance portfolio optimization solves the following quadratic programming problem:

$$\min_{u \in \Omega} \quad \beta u^T \Sigma u - \mu^T u \tag{3.2}$$

where $\beta$ is a Lagrange multiplier controlling the trade-off between the expected portfolio return and portfolio risk, and $\Omega \subset \mathbb{R}^n$ is the accessible portfolio sets such that all $u \in \Omega$ satisfy the constraints imposed by investor. We mark (3.2) as single-period, multiple instruments mean-variance portfolio optimization problem since it only use one period forecasting.

## 3.2 Trading Cost and transaction cost model

In real trading, a systematic trading system based on the forecast of asset return works according to the following chart:

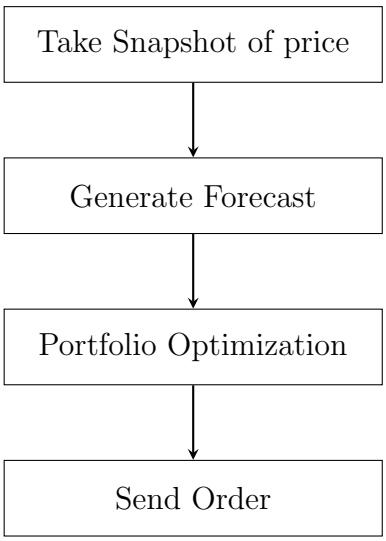


Figure 1: Trading cycle flow chart

The price of an asset at which an order is filled may not be the same as the price which was used to determine the order, which people usually call it "looking price". There is a difference between $r_T$, the asset return on day T calculated by using the looking price $lprice_T$ and the looking price of the next day $lprice_{T+1}$:

$$r_T = lprice_{T+1} - lprice_T$$

as opposed to the asset return $\widetilde{r_T}$ on day T which calculated using the average fill price today $fprice_T$ and the looking price of the next day $lprice_{T+1}$:

$$\widetilde{r_T} = lprice_{T+1} - fprice_T$$

The difference

$$\Delta r = fprice_T - lprice_T$$

which is often called the "transaction cost", consists of three parts. Recall that when a trading system is running, it spends time on several procedures: sending the price into the system and generating the forecast, optimizing the desired position, placing orders and then waiting for execution. Therefore the difference $\Delta r$ can be represented as the

following:

$$\Delta r = \Delta r_{system} + \Delta r_{opt} + \Delta r_{slippage}$$

where $\Delta r_{system}$ is the price difference coming from the trading system construction (sending price and placing order), $\Delta r_{opt}$ is price change due to running the portfolio optimization, and $\Delta r_{slippage}$ is the slippage when the order is being executed. $\Delta r_{system}$ is usually controled by mechanical condition like network speed, cables etc. $\Delta r_{slippage}$ is modeled as what we called transaction cost model. $\Delta r_{opt}$ can be optimized by a faster machine, or a faster algorithm, which is the main topic in this chapter. In this work, we omit the system delay part and focus on constructing high performance algorithm which can efficiently minimize the time cost (which is almost equivalent with minimizing the $\Delta r_{opt}$) with a proper transaction cost model.

We denote $T(u, u_0) : \mathbb{R}^2 \to \mathbb{R}^1$ as a scalar valued function of transaction cost of a trading with the inventory quantity in dollar amount $u_0$ and the desired position $u$ in dollar amount. One model for the scalar transaction cost functions is [38, 9]

$$T(u, u_0) = \lambda_1 a|u - u_0| + \lambda_2 b\sigma \frac{|u - u_0|^{3/2}}{V^{1/2}} + \lambda_3 c(u - u_0) \tag{3.3}$$

where $a, b, c, V$ and $\sigma$ are real numbers described below, and $\lambda_1, \lambda_2, \lambda_3$ are the Lagrange multipliers corresponding to the three terms. The number $a$ is one half the bid-ask spread for the asset at the beginning of the time period, the term $a|x|$ is naturally taken into account when the investor use mid price (the sum of bid and ask divided by two) as the marker for the accounting of their portfolios. The number is a positive number with unit inverse dollars. The number c is used to create asymmetry of transaction costs. When $c = 0$, the transaction cost is the same for buying and selling. When $c > 0$, this model gives lower transaction cost on selling than buying with same dollar amount. Negative transaction costs can happen when $|c| > |a|$. The number $V$ is the trading volume of the asset in the market during this period in dollar value. The number $\sigma$ is the price volatility over recent time periods.

The model (3.3) and the similar variants are suggested by some practitioners in the references [54, 36, 7]. However, in this thesis, we use the following transaction cost model:

$$T(u, u_0) = \lambda_1 a|u - u_0| + \lambda_2 b(u - u_0)^2 + \lambda_3 c(u - u_0) \tag{3.4}$$

where the first term $\lambda_1 a|u - u_0|$ and the third term $\lambda_3 c(u - u_0)$ are the same as (3.3). The only difference is the quadratic transaction cost term $\lambda_2 b(u-u_0)^2$. Here we provide several reasons why we prefer the quadratic transaction cost. One is for the convenience of computation. Notice that the objective function (3.2) keeps the quadratic form by adding transaction cost model (3.4). Then the KKT system (2.17) of this optimization problem is a Linear Complementarity Problem (LCP) and can be solved in a standard solver, while (3.3) makes the KKT system to a Non-linear Complementarity Problem (NLCP) which is much harder to solve. Also the whole problem can be fit into standard quadratic program with transaction cost model (3.4). Another reason is interpretation of the quadratic transaction cost term. Transaction costs are known and not fully predictable before trade are made. These properties give the randomness to the transaction costs, i.e. transaction cost is a random variable at each trading period. Therefore, the quadratic transaction cost term not only fits the exact value of the transaction cost function, but also models the variance of the linear transaction cost as a random variable. Last but not least, the quadratic transaction cost term is derived by the observation of the variance of slippage of asset as time goes on. Figure 2 plots the variance of the slippage of AAPL from 11:30 a.m. to 4 p.m. in 2013. The plots shows clearly that the variance of slippage increases linearly from 11:30 a.m. to 3 p.m., which supports the idea that transaction cost is a random variable in a not obvious way. We mention that the similar structures are used in [37, 42, 12, 33]

To summarize the single-period portfolio optimization with transaction cost model, we get the desired position of each assets by solving

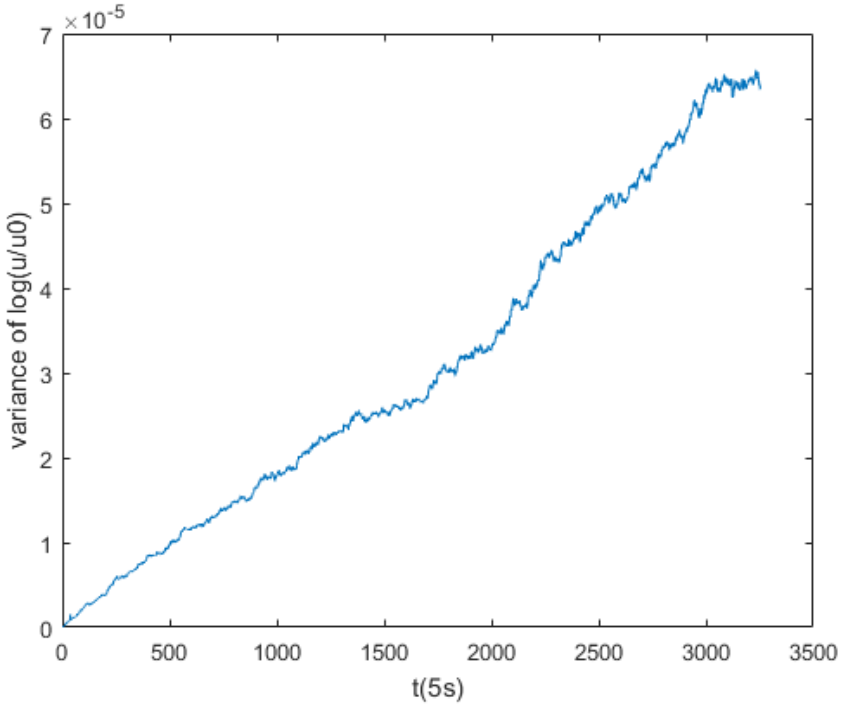


Figure 2: The variance of log return of AAPL since 11:30 a.m. in 2013 v.s. time period. x axis is every five seconds from 11:30 a.m. to 4:00 p.m.. y axis is the variance of the logarithm of the ratio of the prices between every 5 seconds and the price at 11:30 a.m.

the following quadratic programming problem:

$$\underset{u\in\Omega}{\text{minimize}} \quad \beta u^T \Sigma u - \mu^T u + \lambda_1 a|u - u_0| + \lambda_3(u - u_0) + \lambda_2 b(u - u_0)^2 \tag{3.5}$$

## 3.3 Multi-period portfolio optimization problem

In this section we format the portfolio optimization problem with the consideration of multiple periods information. There are several reasons drive us into multi-period setting. One reason is that transaction cost dominates the behaviour in multi-period portfolio optimization problem. One period portfolio optimization, as a kind of greedy strategy, will give us a desired position which is optimal in the next period. In this case, trades happen as long as the expected return is profitable enough to cover the one period transaction cost. But in the multi-period case, the position which is optimal at the first period may bring the portfolio into a sub-optimal condition which can never beat the trajectory considering the overall path and costs. Another reason is that the profit of a portfolio is not only determined by the trading part, but also affected by the holding part. That is to say, even though the prediction of expected return in some periods are not profitable, the portfolio optimizer may still do some trading so that it will have much more profits as the holding inventory in the future. These two facts are illustrated in details in the following example.

**Example 3.1** (single-period optimization v.s. multi-period optimization)**.** Consider a two periods portfolio optimization problem with the following setting: the expected return $r = [2, 1]^T$; the variance estimation $\sigma = [1, 1]^T$; the linear transaction cost multiplier $\tau = [1, 1]^T$ and the quadratic transaction cost multiplier $\kappa = [1, 1]^T$. Figure 3 shows the two trajectories optimized by single-period portfolio optimization (greedy trajectory) and multi-period portfolio optimization. The blue line keeps increasing the inventory but the red line increases the inventory to 0.5 and then hold the position at the second trading period. The

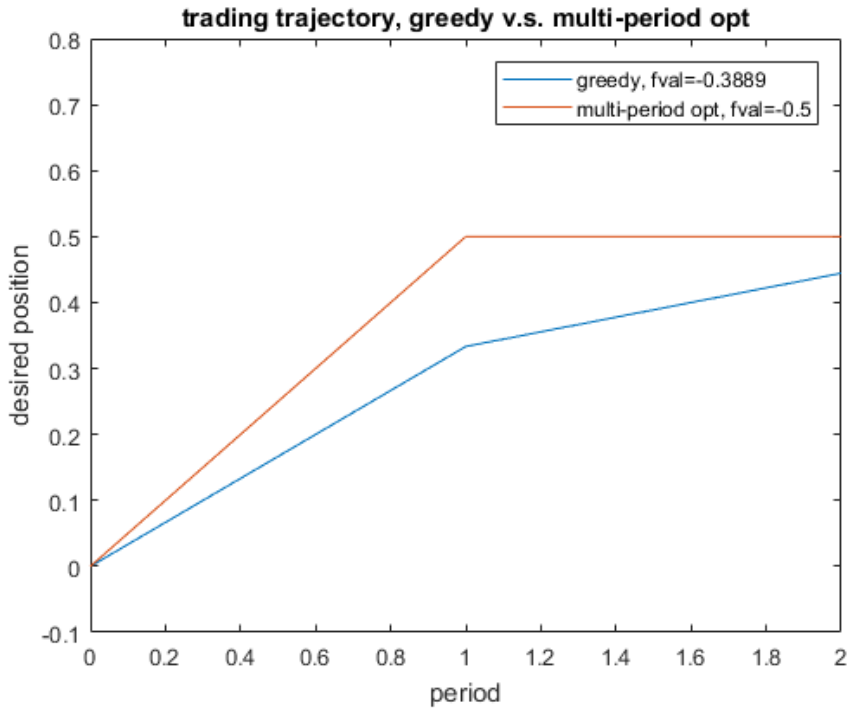


Figure 3: Two trading trajectories, the blue line is optimized by two steps of single-period portfolio optimization, whereas the red line is optimized by one step multi-period portfolio optimization

second trajectory has the lower objective function value comparing to the blue line which is sub-optimal.

One more practical reason is that the multi-period portfolio optimization can incorporate with some hard constraints on the portfolio. We list some of them here:

- *Getting in or getting out the position.* As a new portfolio being set up or an existing portfolio being liquidated, the transactions can be achieved without too much impact by imposing the different constraints on different periods in multi-period portfolio optimization.

- *Day trading with zero over-night position.* Day trading is buying and selling financial instruments within the same trading day. If the day traders do not want to have over-night position which leads to more unexpected risk, they will close all the positions before the

market closes for the trading day. In this case the optimal trading trajectory can be handled naturally in multi-period portfolio optimization by adding the zero position constraint at the final period.

A multi-period portfolio policy $\pi$ is a sequence of investment

$$\begin{aligned}\pi &= \{u_1, u_2, ..., u_T\} \\ &= \left\{ \begin{bmatrix} u_1^1 \\ u_1^2 \\ ... \\ u_1^n \end{bmatrix}, \begin{bmatrix} u_2^1 \\ u_2^2 \\ ... \\ u_2^n \end{bmatrix}, ..., \begin{bmatrix} u_T^1 \\ u_T^2 \\ ... \\ u_T^n \end{bmatrix} \right\}\end{aligned} \tag{3.6}$$

where $T$ is the number of trading periods. As the single-period case, we have to estimate the unknown terms in the objective function values. We denote $\hat{X}_{\tau|t}$ as the estimation of $X_\tau$ given all the information up to period $t$. For example, the expected return estimator $\hat{r}_{t|t}$ is the estimation made at time period t of expected return of underlying assets at period t, and $\hat{r}_{t+1|t}$ is the estimation made at time period t of expected return of underlying assets at period t, etc.

With the same setting as single-period portfolio optimization, we derive the objective function of $i$th period

$$f_i(u_i, u_{i-1}) = \beta_i u_i^T \Sigma u_i - \mu_i^T u_i + T(u_i, u_{i-1}). \tag{3.7}$$

Here $u_i$ is a vector that represents the desired position at $i$th period ($i = 1, 2, ..., T$). It also represents the inventory of each asset at the start of ($i$+1)th period ($i = 1, 2, ..., T-1$). The vector $\mu_i$ is the expected return estimator at $i$th period, and $T(u_i, u_{i-1})$ is a vector valued function gives the transaction cost estimation given the inventory $u_i$ and the desired position $u^{i+1}$.

A natural objective function is the summation of all the individual

period objective functions over the horizon

$$\begin{aligned} F(u) &= \sum_{i=1}^{T} f_i(u_i, u_{i-1}) \\ &= \sum_{i=1}^{T} (\beta_i u_i^T \Sigma^i u_i - \mu_i^T u_i + T(u_i, u_{i-1})) \end{aligned} \tag{3.8}$$

With this objective function, we can get an optimal portfolio trajectory by solving the quadratic program

$$\begin{aligned} \underset{u}{\text{minimize}} \quad F(u) \quad &= \sum_{i=1}^{T} (\beta_i u_i^T \Sigma^i u_i - \tilde{\mu_i}^T u_i + \tau_i^T |u_i - u_{i-1}| \\ &\qquad + (u_i - u_{i-1})^T D_{\kappa_i} (u_i - u_{i-1})) \end{aligned} \tag{3.9}$$

where $\tilde{\mu_i} = \mu_i - \lambda_3^i c^i$ is transaction cost adjusted return at $i$th period, $\tau_i = \lambda_1^i a^i$ is the vector that represents the Lagrange multiplier with half bid-ask spread for each asset at $i$th period, $D_{\kappa_i}$ is a diagonal matrix whose diagonal elements are the Lagrange multipliers of the quadratic transaction cost term for each asset.

## 3.4 Portfolio constraints

There are many constraints on a long-time running portfolio. Some of them aim to decrease the potential portfolio risk, some of them act in order to reduce the transaction cost, and some of them are hard liability constraints which cannot be violate due to the compliance requirement. Here we list some common portfolio constraints, and integrate them in to a generic set of constraints including *position bound* and *trade bound*.

One common constraint is long only constraint. This constraint require that the position of each asset can only be zero or positive, i.e.

$$u_i^j \geq 0 \quad \text{for all } i, j$$

Notice that the long only constraint does not mean "buying only". One can still sell the assets as long as the quantity of asset after trading

is positive. Long only constraints can be interpreted as position lower bounds with zero quantities. Let $poslb_i^j$ be the position lower bound for $j$th instrument at $i$th time period, we have

$$u_i^j \geq poslb_i^j$$
$$poslb_i^j = 0$$

Market cap constraint is a holding constraint that allow the investor limit any asset in their portfolios beyond a portion of the total market capitalization of this asset. To convert this constraint to our position bound notation, let $C_j^i$ be the market cap of $j$th asset at $i$th period, then we have

$$poslb_i^j \leq u_i^j \leq posub_i^j$$
$$poslb_i^j = -\delta_i^j C_i^j$$
$$posub_i^j = \delta_i^j C_i^j$$

where $\delta_j^i > 0$ is a fixed number determined by the investor as the desired portion of the total market cap of $j$th asset at $i$th period.

Another practical holding constraint is concentration constraint. The concentration constraint is a dollar amount limit, which is usually calculated by a fixed fraction of the total dollar value of the portfolio, that investor prevents potential risk of the whole portfolio from holding one asset to much. Let $B_i$ be the total dollar value of the portfolio (also called book size) at $i$th period, we have

$$poslb_i^j \leq u_i^j \leq posub_i^j$$
$$poslb_i^j = -B_i$$
$$posub_i^j = B_i$$

Trading constraints restrict the choice of trades $u_i - u_{i-1}$. Two kinds of trading constraints are considered here, one is no-buy constraint

$$u_i^j - u_{i-1}^j \leq trdub_i^j$$
$$trdub_i^j = 0$$

where $trdub_i^j$ is the trading upper bound of $j$th asset at $i$th period. Similarly, we have the no-sell constraint

$$u_i^j - u_{i-1}^j \geq trdlb_i^j$$
$$trdlb_i^j = 0$$

and no-trade constraint

$$trdlb_i^j \leq u_i^j - u_{i-1}^j \leq trdub_i^j$$
$$trdlb_i^j = 0$$
$$trdub_i^j = 0$$

where $trdlb_i^j$ is the trading lower bound of $j$th asset at $i$th period. We mention that the no-trade constraint is equivalent with the following equality constraint

$$u_i^j = u_{i-1}^j$$

Another trading constraint is participation rate constraint. This constraint is a natural constraint since an investor cannot buy or sell more than the volume provided on the market. The investor may have much lower participation rate in order to limit the transaction cost of the whole portfolio in an acceptable range. Let $V_j^i$ be the dollar volume of $j$th asset traded at $i$th period, then the participation rate constraint can be written as

$$trdlb_i^j \leq u_i^j - u_{i-1}^j \leq trdub_i^j$$
$$trdlb_i^j = -\delta_i^j V_i^j$$
$$trdub_i^j = \delta_i^j V_i^j$$

Notice that the trading volume $V_j^i$ are unknown when the trading decision is made. Practitioners usually estimate it from the historical trading data.

Each holding constraint (or trading constraint) gives a lower bound and an upper bound of the desired holding quantity $u_i^j$ (or the desired trading quantity $u_i^j - u_{i-1}^j$). We can reduce the number of inequality constraint by taking the highest value of lower bounds and the lowest value of upper bounds since other bounds cannot act in any sense. After the reducing procedure, we have a set of final constraint

$$\begin{cases} poslb_i^j \leq u_i^j \leq posub_i^j \\ trdlb_i^j \leq u_i^j - u_{i-1}^j \leq trdub_i^j \end{cases} \tag{3.10}$$

Finally we can align the objective function and the constraints together as the target portfolio optimization problem

$$\begin{aligned} \underset{u_1,u_2,...,u_T}{\text{minimize}} \quad & \sum_{i=1}^{T} (\beta_i u_i^T \Sigma_i u_i - r_i^T u_i + \tau_i^T |u_i - u_{i-1}| \\ & \qquad + (u_i - u_{i-1})^T D_{\kappa_i} (u_i - u_{i-1})) \\ \text{subject to} \quad & poslb_i \leq u_i \leq posub_i, \ i = 1, 2, ..., T \\ & trdlb_i \leq u_i \leq trdub_i, \ i = 1, 2, ..., T \end{aligned} \tag{3.11}$$

where $poslb_i, posub_i, trdlb_i, trdub_i$ are vector valued function whose entries are $poslb_i^j, posub_i^j, trdlb_i^j, trdub_i^j$, $j = 1, 2, ....$ (3.11) is the fundamental problem in this thesis and we will discuss the several variants from this problem.

## 3.5 The standard quadratic programming (QP) solution

In this section we discuss the standard quadratic programming method for the problem (3.11). It is easy to see the objective function in (3.11) is convex: The objective function in each period consists of four parts: two quadratic parts, one affine parts, and one absolute form part. Since

all the three kinds of parts are convex, the objective function in each period (which is the sum of these part) is convex. So as the summation of the objective functions for all the periods. In order to fit the problem (3.11) into a standard quadratic programming form, we need a splitting of the variable $u_i$ called *buy-sell splitting*:

$$\begin{aligned} u_i - u_{i-1} &= b_i - s_i \\ b_i &\geq 0 \\ s_i &\leq 0 \end{aligned} \tag{3.12}$$

Here we split each trade $u_i - u_{i-1}$ into two non-negative part: buying part and selling part. An equivalent way to understand this splitting is that we take the dual of the original problem. The buy-sell splitting has a complementarity property when the objective function has non-negative transaction cost term

$$b_i^T s_i = 0 \tag{3.13}$$

With the buy-sell splitting, we can rewrite the problem (3.11) as (see Appendix A for details)

$$\begin{aligned} \underset{b,s}{\text{minimize}} \quad & \frac{1}{2}\begin{bmatrix} b & s \end{bmatrix}\begin{bmatrix} I & I \\ -I & I \end{bmatrix}\begin{bmatrix} L^T D_\sigma L & 0 \\ 0 & 2D_\kappa \end{bmatrix}\begin{bmatrix} I & -I \\ I & I \end{bmatrix}\begin{bmatrix} b \\ s \end{bmatrix} + g^T \begin{bmatrix} b \\ s \end{bmatrix} \\ \text{subject to} \quad & b \geq 0 \\ & s \geq 0 \\ & b \leq trdub \\ & s \leq trdlb \\ & L(b-s) \leq posub - e \otimes u_0 \\ & L(b-s) \geq poslb - e \otimes u_0 \end{aligned}$$

# 4 AN ALGORITHM FOR PORTFOLIO SELECTION

In this chapter, we develop a dynamic programming algorithm solving one instrument multi-period portfolio optimization problem with proposed transaction cost model and bound constraints. We show the problem satisfy the Pontryagin's maximum principle, so that it is possible to solve this problem by dynamic programming method. We show the optimal trajectory is a piece-wise linear function for each stage. An extension to two instruments case is proposed with numerical experiments. Recall that the multi-period mean-variance portfolio selection problem is

$$\underset{u_i}{\text{minimize}} \quad \sum_{i=1}^{T}(u_i^T C_i^2 u_i - r^T u_i)$$

For the one instrument case, the covariance matrix reduces to a scalar representing the variance of the underlying instrument. Adding the transaction cost model and bound constraints, we get the one instrument multi-period portfolio optimization problem:

$$\begin{aligned} \underset{u}{\text{minimize}} \quad & \sum_{i=1}^{T}(\frac{1}{2}\sigma_i u_i^2 - r_i u_i + \tau_i|u_i - u_{i-1}| + \kappa_i(u_i - u_{i-1})^2) \\ \text{subject to} \quad & posl b_i \leq u_i \leq posub_i, \quad i = 1, 2, ..., T \\ & trdlb_i \leq u_i - u_{i-1} \leq trdub_i, \quad i = 1, 2, ..., T \end{aligned} \tag{4.1}$$

Here $T$ is the terminal time, $u = [u_1, u_2, ..., u_T]^T$ is a $T$ dimensional vector where each entry $u_i$ represents the inventory of the underlying asset at time period $i$, $\sigma_i$ is the variance estimator of the asset, $r_i$ is the return estimator, $\tau_i$ and $\kappa_i$ are the multipliers for porportional transaction cost and quadratic transaction cost term, $poslb_i$ and $posub_i$ are the position bound constrains where $poslb \leq posub$, $trdlb_i$ and $trdub_i$ are the trade bound constraints where $trdlb \leq trdub$.

## 4.1 The Pontryagin's maximum principle

In this section we introduce the Pontryagin's maximum principle and show that the problem (4.1) satisfy the principle.

We follow the notation on [31] .Consider a dynamic system

$$\begin{cases} \dot{x}(t) = f(x(t), \alpha(t)) \quad (0 \leq t \leq T) \\ x(0) = x_0 \end{cases} \tag{4.2}$$

where $x(t) \in R^n$ is a time dependent variable at time $t$, $\dot{x}(t)$ is the change of $x(t)$ (or the derivative for the continuous time case) at time $t$, and $\alpha(t)$ is the control on $x$ at time t. Let $A \subset \mathbb{R}^m$ and $f : \mathbb{R}^n \times A \to \mathbb{R}^n$,$x_0 \in R^n$. Therefore the set of admissible controls is defined as

$$\mathcal{A} = \{\alpha(\cdot) :\to A | \alpha(\cdot) \text{ is measurable}\}$$

and the payoff functional:

$$P[\alpha(\cdot)] = \int_0^T r(x(t), \alpha(t))dt + g(x(T)) \tag{4.3}$$

where the terminal time $T > 0$, running payoff funtional $r : \mathbb{R}^n \times A \to \mathbb{R}$ and terminal payoff $g : \mathbb{R}^n \to \mathbb{R}$ are given. Then a basic problem is to find a control $\alpha^*(\cdot)$ maximizing the payoff functional, i.e.

$$P[\alpha(\cdot)] = \max_{\alpha(\cdot) \in \mathcal{A}} P[\alpha(\cdot)] \tag{4.4}$$

By defining an appropriate Hamiltonian of the problem (4.4)

$$H(x, p, a) = f(x, a) \cdot p + r(x, a) \quad (x, p \in R^n, a \in A), \tag{4.5}$$

the Pontryagin's Maximum Principle asserts that there exists a function $p^*(\cdot)$, which together with the optimal trajectory $x^*(\cdot)$, satisfies an analog of the dynamic system (4.2). We state it formally below:

**Theorem 4.1** (Pontryagin's Maximum Principle.)**.** Assume that $\alpha^*$ is optimal for the dynamic system (4.2) and (4.4), and $x^*(\cdot)$ is the corre-

sponding trajectory. Then there exists a function $p^* : [0, T] \to R^n$ such that

$$\dot{x}^*(t) = \nabla_p H(x^*(t), p^*(t)\alpha^*(t)),$$
$$\dot{p}^*(t) = -\nabla_x H(x^*(t), p^*(t)\alpha^*(t)),$$

and

$$H(x^*(t), p^*(t), \alpha^*(t)) = \max_{a \in A} H(x^*(t), p^*(t), a) \quad (0 \leq t \leq T).$$

In addition, the mapping

$$t \mapsto H(x^*(t), p^*(t), \alpha^*(t))$$

is constant. The terminal condition is

$$p^*(T) = \nabla g(x^*(T)).$$

In order to apply the maximum principle on the portfolio optimization problem, we first convert (4.1) into a dynamic system.

let $T$ be the terminal time, $w(t)$ the cash holding at time period $t$, $u(t)$ the amount of asset holding at time period $t$, $\alpha(t)$ amount of buying (or selling) asset at time period $t$, $r(t)$ the estimation of log return (or the rate of return) of asset over time period $t$ and $T_{cost}$ the total transaction cost (including proportional transaction cost and quadratic transaction cost) estimation over time period $t$. On each time period, we also have the constraints on the control (trading) variable $\alpha(t)$:

$$posl b_t - u(t-1) \leq \alpha(t) \leq posub_t - u(t-1)$$
$$trdlb_t \leq \alpha(t) \leq trdub_t$$

Here a positive $\alpha(t)$ means buying the asset, a negative $\alpha(t)$ means selling the asset and $\alpha(t) = 0$ means holding the position over time period $t$.

In order to minimize the objective function in (4.1), we can define the payoff functional as the negative value of (4.1) with the new variable

setting:

$$P[\alpha(\cdot)] = \sum_{t=1}^{T}[w(t) + r(t)u(t) - \frac{1}{2}\sigma(t)^2 u(t)^2] \tag{4.6}$$

where $\sigma(t)^2$ is the variance estimator of asset at time period $t$. And the evolution is

$$\begin{cases} \dot{u}(t) = u(t) - u(t-1) = \alpha(t) \\ \dot{w}(t) = \alpha(t) - T_{cost}(\alpha(t)) \end{cases} \tag{4.7}$$

Now we are ready to show how Pontryagin's Maximum Principle can be applied on the portfolio optimization problem.

The Hamiltonian of this evolution is

$$H(x, p, t, a) = p_1 a + p_2(a - T_{cost}(a)) + [w(t) + r(t)u(t) - \frac{1}{2}\sigma(t)^2 u(t)^2] \tag{4.8}$$

Then the adjoint dynamics is

$$\begin{cases} \dot{p}^1(t) = \sigma(t)^2 u(t) - r(t) \\ \dot{p}^2(t) = -1 \end{cases} \tag{4.9}$$

Since the terminal time condition here is $g(x(T)) = 0$, we have

$$\begin{cases} p^1(T) = 0 \\ p^2(T) = 0 \end{cases} \tag{4.10}$$

Therefore the costate $[p^1, p^2]^T$ is

$$\begin{cases} p^1(t) = -\sum_{s=t+1}^{T}[\sigma(s)^2 u(s) - r(s)] \\ p^2(t) = T - t \end{cases} \tag{4.11}$$

And the maximum principle tells us

$$\begin{aligned} H(x(t), p(t), t, \alpha(t)) = \max_{a \in \mathcal{A}} \; & p_1 a + p_2(a - T_{cost}(a)) + [w(t) \\ & + r(t)u(t) - \frac{1}{2}\sigma(t)^2 u(t)^2] \end{aligned} \tag{4.12}$$

with the costate (4.11).

## 4.2 The dynamic programming algorithm

In this section we describe the practical details of the algorithm. Recall that the idea of dynamic programming is breaking the whole optimization problem into several sub-problems and then recursively finding the optimal solutions to the sub-problems. We first show a way to break the one instrument portfolio optimization problem into several sub-problem. Then using the memoization technique, we solve the problem by construct a optimal solution mapping.

### 4.2.1 Sub-problem formulation

Using the indicator function, we can write the original problem (4.1) as

$$\begin{aligned} \underset{u}{\text{minimize}} \quad F(u) = \sum_{i=1}^{n} (&\frac{1}{2}\sigma_i u_i^2 - r_i u_i + \tau_i |u_i - u_{i-1}| \\ &+\kappa_i (u_i - u_{i-1})^2 + \mathcal{I}_i(u_i)) \end{aligned} \tag{4.13}$$

where the indicator function $\mathcal{I}_i$ indicates the constraints at $i$th time period:

$$\mathcal{I}_i(u_i) = \begin{cases} 0, & \text{if } poslb_i \le u_i \le posub_i \\ & \text{and } tradlb_i \le u_i - u_{i-1} \le trdub_i, \\ +\infty, & \text{otherwise} \end{cases}$$

Let $G_{\tilde{u}_1,\tilde{u}_2,...,\tilde{u}_i}(u)$ be the minimum objective function value $F(u)$ when the inventories of first $i$ periods have been assigned to $\tilde{u}_1, \tilde{u}_2, ..., \tilde{u}_i$, i.e.

$$G_{\tilde{u}_1,\tilde{u}_2,...,\tilde{u}_i}(u) = \inf_{u_{i+1},u_{i+2},...,u_n} \{F(u) | u_1 = \tilde{u}_1, u_2 = \tilde{u}_2, ..., u_i = \tilde{u}_i\} \tag{4.14}$$

and the corresponding optimal solution vector $[\tilde{u}^*_{i+1}, \tilde{u}^*_{i+2}, ..., \tilde{u}^*_n]^T$ is

$$[\tilde{u}^*_{i+1}, \tilde{u}^*_{i+2}, ..., \tilde{u}^*_n]^T = \underset{u_{i+1}, u_{i+2}, ..., u_n}{\operatorname{argmin}} \{F(u) | u_1 = \tilde{u}_1, u_2 = \tilde{u}_2, ..., u_i = \tilde{u}_i\} \tag{4.15}$$

Note that changing the value of $\tilde{u}_1, \tilde{u}_2, ..., \tilde{u}_{i-1}$ does not change the optimal point (4.15) since changing the constant does not change the solution of convex optimization problem. But changing the value of $\tilde{u}_i$ leads to a different optimal point because $\tilde{u}_i$ is embedded in the proportional cost $|u_i - \tilde{u}_{i-1}|$ and the quadratic transaction cost $(u_i - \tilde{u}_{i-1})^2$. Since $\sigma_i > 0$ for $i = 1, 2, ..., n$ and the convex optimization problem is feasible, it has one and only one optimal point for different value of $\tilde{u}_i$, which means $\tilde{u}_i$ characterize the optimal trajectory of time period $i+1, i+2, ..., n$. Thus there is a series of functions $H_i(u_i)$ which gives the optimal inventory in the future given the inventory of time period i, i.e.

$$\begin{aligned} H_i(u_i) = \underset{u_{i+1}, u_{i+2}, ..., u_n}{\operatorname{argmin}} \sum_{k=i+1}^{n} (\frac{1}{2}\sigma_k u_k^2 - r_k u_k + \tau_k |u_k - u_{k-1}| \\ + \kappa_k (u_k - u_{k-1})^2 + \mathcal{I}_k(u_k)) \end{aligned} \tag{4.16}$$

We can write down (4.16) as a recursive form

$$\begin{aligned} H_i(u_i) &= \underset{u_{i+1}, u_{i+2}, ..., u_n}{\operatorname{argmin}} \sum_{k=i+1}^{n} (\frac{1}{2}\sigma_k u_k^2 - r_k u_k + \tau_k |u_k - u_{k-1}| \\ &\quad + \kappa_k (u_k - u_{k-1})^2 + \mathcal{I}_k(u_k)) \\ &= \underset{u_{i+1}}{\operatorname{argmin}} \underset{u_{i+2}, ..., u_n}{\operatorname{argmin}} \sum_{k=i+1}^{n} (\frac{1}{2}\sigma_k u_k^2 - r_k u_k + \tau_k |u_k - u_{k-1}| \\ &\quad + \kappa_k (u_k - u_{k-1})^2 + \mathcal{I}_k(u_k)) \\ &= \underset{u_{i+1}}{\operatorname{argmin}}\ H_{i+1}(u_{i+1}) \end{aligned} \tag{4.17}$$

Now it is clear that $H_{i+1}(u_{i+1})$ is a sub-problem of $H_i(u_i)$ and by solving all the sub-problems we can get the optimal trajectory at current time period $i$.

### 4.2.2 Memoization for optimal condition

Dynamic programming is a powerful technique to break a complex problem into small pieces; however, it does not save the computation. In fact, the computational complexity of dynamic programming is exponential without memoization technique. Here we discuss in detail that how to apply the memoization technique in this optimization problem.

Follow the notation of (4.13), we can write down the subderivative of $f$ against $u_i$ as follows

$$\frac{\partial F}{\partial u_i} = \begin{cases} \sigma_i u_i - r_i + \tau_i \text{sign}(u_i - u_{i-1}) + 2\kappa_i(u_i - u_{i-1}) + \mathcal{I}_i'(u_i) \\ \quad -\tau_{i+1}\text{sign}(u_{i+1} - u_i) - 2\kappa_{i+1}(u_{i+1} - u_i) - \mathcal{I}_{i+1}'(u_{i+1}), \\ \qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad 1 \leq i < n, \\ \sigma_n u_n - r_n + \tau_n \text{sign}(u_n - u_{n-1}) + 2\kappa_n(u_n - u_{n-1}) + \mathcal{I}_n'(u_n), \\ \qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad i = n, \end{cases} \tag{4.18}$$

where $\text{sign}(x)$ is the sign function. The subderivative shows a chain relationship among the inventory of $(i-1)$th, $i$th and $(i+1)$th time period. With this subderivative, we get an alternative representation of the optimal condition for problem (4.13)

$$0 \in \frac{\partial F}{\partial u_i} \tag{4.19}$$

This suggests the following system

$$\begin{cases} \qquad\qquad\qquad\qquad\qquad\qquad \tau_i \text{sign}(u_i - u_{i-1}) + 2\kappa_i(u_i - u_{i-1}) + \mathcal{I}_i'(u_i) = \\ r_i - \sigma_i u_i + \tau_{i+1}\text{sign}(u_{i+1} - u_i) + 2\kappa_{i+1}(u_{i+1} - u_i) + \mathcal{I}_{i+1}'(u_{i+1}), \\ \qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad 1 \leq i < n, \\ \tau_n \text{sign}(u_n - u_{n-1}) + 2\kappa_n(u_n - u_{n-1}) = r_n - \sigma_n u_n + \mathcal{I}_n'(u_n), \\ \qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad i = n, \end{cases} \tag{4.20}$$

For the convenience of the following analysis, we denote $\nabla F_{\text{left}}(u_{i-1}, u_i)$ be the left side of (4.20)

$$\nabla F_{\text{left}}(u_{i-1}, u_i) = \tau_i \text{sign}(u_i - u_{i-1}) + 2\kappa_i(u_i - u_{i-1}) + \mathcal{I}_i'(u_i), \quad (4.21)$$

and $\nabla F_{\text{right}}(u_i, u_{i+1})$ the right side

$$\nabla F_{\text{right}}(u_i, u_{i+1}) = \begin{cases} r_i - \sigma_i u_i + \tau_{i+1}\text{sign}(u_{i+1} - u_i) \\ +2\kappa_{i+1}(u_{i+1} - u_i) + \mathcal{I}_{i+1}'(u_{i+1}) \\ \qquad\qquad 1 \le i < n, \\ r_n - \sigma_n u_n, \qquad\qquad i = n \end{cases} \quad (4.22)$$

Instead of solving the symbolic system (4.20), one can solve this system numerically by tracking the function value of $\nabla F_{\text{left}}(u_{i-1}, u_i)$ and $\nabla F_{\text{right}}(u_i, u_{i+1})$: Given the value of $u_{i-1}$, $\nabla F_{\text{left}}(u_{i-1}, u_i)$ is a piece-wise linear increasing function of $u_i$ (notice that the "slope" $\sigma_i$, $\tau_i$ and $\kappa_i$ are non-negative ).Similarly we have $\nabla F_{\text{right}}(u_i, u_{i+1})$ is a piece-wise linear decreasing function of $u_i$ given the value of $u_{i+1}$. Thus we have two function value graphs ($\nabla F_{\text{left}}(u_{i-1}, u_i)$,$u_i$) and ($\nabla F_{\text{right}}(u_i, u_{i+1})$,$u_i$) for the same variable $u_i$. Then the intersection point between these two graphs indicates a solution tuple $(u_{i-1}, u_i^*, u_{i+1})$, where $u_i^*$ is the "x" value of the intersection point on the graph which is also the optimal value of $u_i$ given $u_{i-1}$ and $u_{i+1}$. Figure shows an example of the graph ($\nabla F_{\text{left}}(u_{i-1}, u_i)$,$u_i$) and ($\nabla F_{\text{right}}(u_i, u_{i+1})$,$u_i$), and how they intersect each other.

One important property of this graph with intersection is that the two graph ($\nabla F_{\text{left}}(u_{i-1}, u_i)$,$u_i$) and ($\nabla F_{\text{right}}(u_i, u_{i+1})$ have exact one intersection point if $\sigma_i > 0$ and $\kappa_i > 0$ for $i = 1, 2, ..., n$. To see this, suppose we have two lines $f(x) = ax + b$ and $g(x) = cx + d$, the intersection point is $(\frac{d-b}{a-c}, \frac{ad-bc}{a-c})$, the intersection is unique as long as $a \neq c$. Notice that the slope of $\nabla F_{\text{left}}(u_{i-1}, u_i)$ is always positive (can be $+\infty$) when $\kappa_i > 0$, and the slope of $\nabla F_{\text{right}}(u_i, u_{i+1})$ is always negative when $\sigma_i + \kappa_{i+1} > 0$, therefore we have exact one solution of $u_i$ since the slope

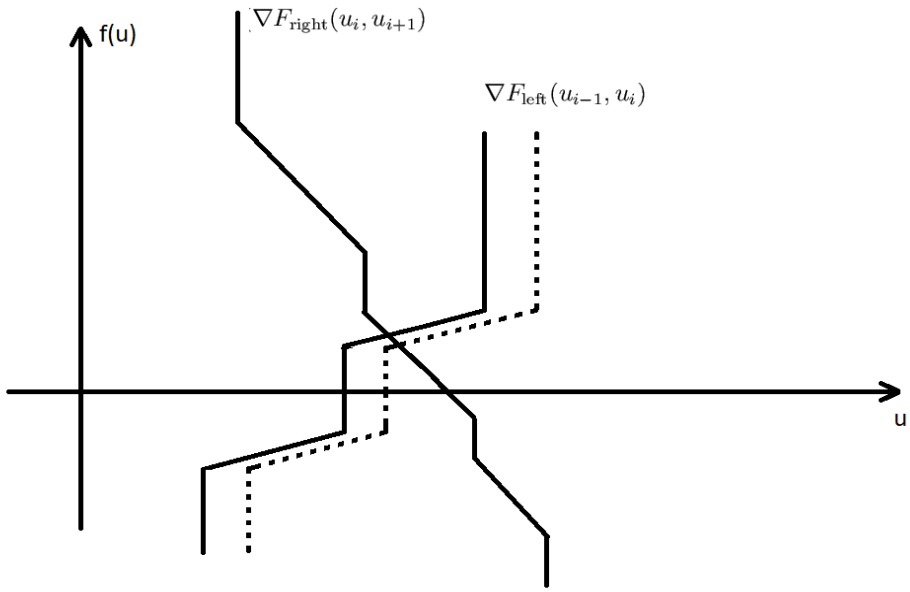


Figure 4: Finding the intersection between two piece-wise linear functions $\nabla F_{\text{right}}(u_i, u_{i+1})$ and $\nabla F_{\text{left}}(u_{i-1}, u_i)$

of two functions cannot be the same in any cases. Also, the "x" value of intersection point, which can be treated as a function of $u_{i-1}$ and $u_{i+1}$, is a linear against $u_{i-1}$ and $u - i + 1$ since the "x" value of the intersection point of two lines is a linear function of the intercepts of these two lines. Figure 4 shows this property visually.

Saving all the solutions for different scheme is not possible in general. But the piece-wise linearity of the two graph immediately leads to a way to memorize all the intersections: one can save the breakpoints of each segments of the whole piece-wise linear function, then all the solutions located in each line segment can be re-constructed by the corresponding breakpoints (see figure for example). Since the graph ($\nabla F_{\text{left}}(u_{i-1}, u_i)$,$u_i$) has 4 line segments in general, the number of breakpoints will increase at most 4 after the process and will be used in the next process. So we call this process the "merge" process as we "merge" the breakpoints of the graph ($\nabla F_{\text{left}}(u_{i-1}, u_i)$,$u_i$) to the graph ($\nabla F_{\text{right}}(u_i, u_{i+1})$,$u_i$).

## 4.3 Practical implementation

In this section we describe the implementation for the algorithm, the overall complexity is showed at the end of this section. Throughout this section, we denote $x(i)$ be the $i$th entry of vector $x$.

We start from the last period. From the system (4.20) we know

$$\nabla F_{\text{right}}(u_n) = r_n - \sigma_n u_n + \mathcal{I}'_n(u_n) \tag{4.23}$$

Let $w_n = [poslb_n, posub_n]^T$, then the function value $\nabla F_{\text{right}}$ is fully characterized by $[r_n - \sigma_n w_n(1), r_n - \sigma_n w_n(2)]^T$. Similarly we define a vector $v = [trdlb_n, 0, 0, trdub_n]^T$, then we know the corresponding function value of $\nabla F_{\text{left}}$ is $[-\tau_n + 2\kappa_n trdlb_n, -\tau_n, \tau_n, \tau_n + 2\kappa_n trdub_n]^T$. The range of $\nabla F_{\text{left}}$ and $\nabla F_{\text{right}}$ usually are not the same. Therefore for the convenience of the computation, we add the entries to one whose range is smaller than the other one at both the top and the bottom of the graph. For example, in figure the range of $\nabla F_{\text{right}}$ is smaller than $\nabla F_{\text{left}}$ so we add two points $(poslb_n, \tau_n + 2\kappa_n trdub_n)$ and $(posub_n, -\tau_n + 2\kappa_n trdlb_n)$ to the graph ($\nabla F_{\text{right}}(u_n)$,$u_i$).

The next step is running the merge process. We start from the highest value of $\nabla F_{\text{left}}(u_n)$ and $\nabla F_{\text{right}}(u_n)$, align each breakpoint of two graphs in the descending order. The optimal solution tuple $(u_{n-1}, u_n^*)$ are computed at the breakpoint. Figure shows the entire procedure in detail.

After the merge procedure, we have a mapping $A_n : u_{n-1} \to u_n^*$ tells us given the inventory $u_{n-1}$ which $u_n$ is optimal. Then we shift the stage one period back to $n-1$. At the new stage we use $u_n^*$ from the last stage construct the graph $\nabla F_{\text{right}}(u_{n-1})$ and run the merge process again until we have the first mapping $A_1 : u_0 \to u_1$. Since the initial inventory $u_0$ is given, we recursively apply the mappings $A_1, A_2, ..., A_n$ to $u_0$ in order to recover the optimal trajectory $u_1, u_2, ..., u_n$.

## 4.4 Complexity analysis

Algorithm 1 shows the whole procedure of the dynamic programming method described in the previous sections. The algorithm contains the optimal mapping generating part and the optimal trajectory recovery part. For the optimal mapping generating part, there is a sub-loop of $6 \times n + 2$ FLOPs (notice that each loop the number breakpoints will increase 6) contained in the main loop of length n. So the complexity of finding the optimal mapping is $O(n \times (6 \times n + 2)) \sim O(n^2)$. For the optimal trajectory recovery part, we perform an $O(n)$ or $O(\log n)$ on searching on a sorted array in each time period. Thus the total time complexity is $O(n^2)$.

## 4.5 Numerical Experiment

We use the New York Stock Exchange (NYSE) Historical Trades & Quotes (TAQ) database as the source of test set. We use AAPL as the test instrument. The test problems various from 1-min trading to daily trading. We setup the other parameters and estimators as the following: The risk estimator is estimated by the variance of the log returns in previous 100 trading periods, i.e. the variance of 1 minute log return in the past 100 minutes for 1 minute trading, or the variance of 5 minutes log return in the past 500 minutes for 5 minutes trading.

**Environment system** The experiment system is 8x Intel(R) Core(TM) i7-6700K CPU @ 4.00GHz, 16GB DDR4 memory. All the tests are run in MATLAB 2018a.

**Execution time test.** We compare the execution time of the portfolio optimization problem (4.1) using Algorithm 1 against 2 benchmarks. The first one is *quadprog* function in MATLAB. It is a well implemented, high performance solver for quadratic objective functions with linear constraints. The *quadprog* solver uses *interior point method* solving the inequality constrained quadratic programming problem. Therefore we also refer it as an implementation of *interior point method* solver. The

**Algorithm 2** find optimal portfolio path u

**Require:** $\sigma,\ r,\ \tau,\ \kappa,\ poslb,\ posub,\ trdlb,\ trdub,\ u_0,\ n$;
**Ensure:** $u$;
$\nabla F_{\text{right}} \leftarrow [0; 0]$;
$w_n = [poslb_n; posub_n]$;
$k \leftarrow n$;
$A \leftarrow [\ ]$;
% construct optimal mapping A
**while** $k > 1$ **do**
  $\nabla F^{new}_{\text{right}} = [\ ]$; $w_0 = [\ ]$; $w_1 = [\ ]$;
  $\nabla F_{\text{right}} \leftarrow r_k - \sigma_k * w_n + \nabla F_{\text{right}}$;
  $\nabla F_{\text{left}} \leftarrow [\tau + 2 * \kappa * trdub_i; \tau; -\tau; -\tau + 2 * \kappa * trdlb_i]$;
  **while** $\nabla F_{\text{right}}$ and $\nabla F_{\text{left}}$ are not empty **do**
    **if** $\nabla F_{\text{right}}(end) \leq \nabla F_{\text{left}}(end)$ **then**
      append $\nabla F_{\text{right}}(end)$ into $\nabla F^{new}_{\text{left}}$;
      compute the corresponding solution pair $(u_{i-1}, u_i)$ in (4.20);
      append the solution pair into $w_0$ and $w_1$;
      $\nabla F_{\text{right}} \leftarrow \nabla F_{\text{right}}(1 : end - 1)$;
    **else** $\{\nabla F_{\text{left}}(end) > \nabla F_{\text{right}}(end),\}$
      append $\nabla F_{\text{left}}(end)$ into $\nabla F^{new}_{\text{right}}$;
      compute the corresponding pair $(u_{i-1}, u_i)$ in (4.20);
      append the solution pair into $w_0$ and $w_1$;
      $\nabla F_{\text{left}} \leftarrow \nabla F_{\text{left}}(1 : end - 1)$;
    **end if**
  **end while**
  $A_k \leftarrow [w_0, w_1]$;
  append $A_k$ into $A$;
  $\nabla F_{\text{right}} \leftarrow \nabla F^{new}_{\text{right}}$;
  $k \leftarrow k - 1$;
**end while**
% generate portfolio path
**for** $k = 1 : n$ **do**
  given inventory $u_{k-1}$ and piece-wise linear mapping $A_k$, find optimal position $u^*_k$;
  $u_k \leftarrow u^*_k$;
**end for**

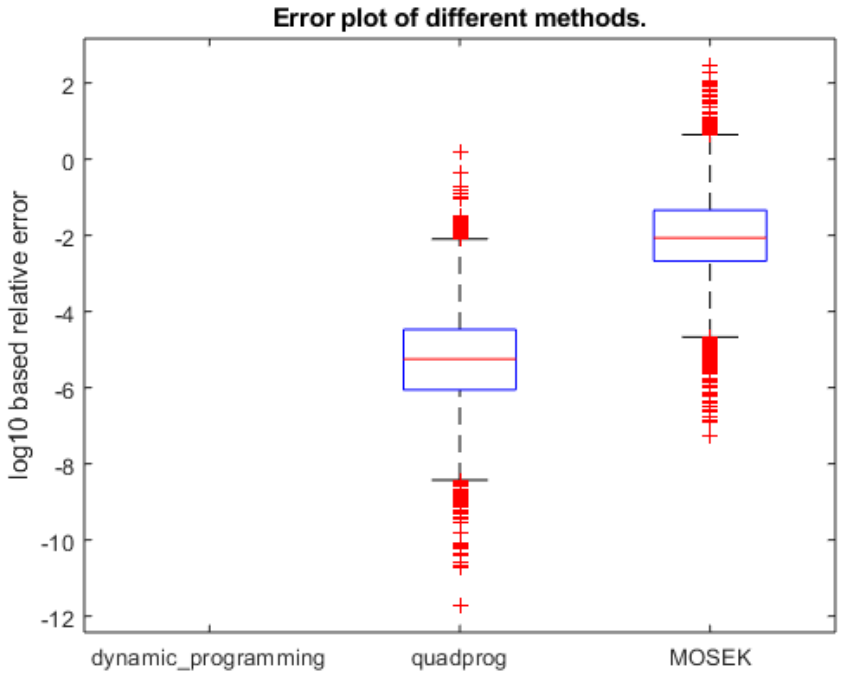


Figure 5: The relative error plot of the objective function value of 1-minute based portfolio optimization problem with for AAPL quote data in Januay 2017.

other one is a high performance commercial softwares: MOSEK[1]. It has promised performance in finance. The quadratic programming problem is treated as conic programming problem in MOSEK. Thus we also refer it as an implementation of *conic programming method* solver.

We choose trading systems in 2 different resolutions: 1-minute system and 5-minute system. In order to cover the whole trading day, the corresponding forecasting periods are 390 periods and 78 periods. We use the quotes data of AAPL in January 2017. This one month data contains 8190 1-minute trading cycles and 1638 5-minute trading cycles. Figure 6 shows the boxplot of execution time for computing the desired position of 1-minute trading system. The median execution time of dynamic programming algorithm, quadprog and mosek are 0.0047 seconds, 0.0643 seconds and 0.0167 seconds, respectively. Notice that the dynamic programming algorithm is almost 15 times faster than quadprog and 4 times faster than mosek. We mention that the our algo-

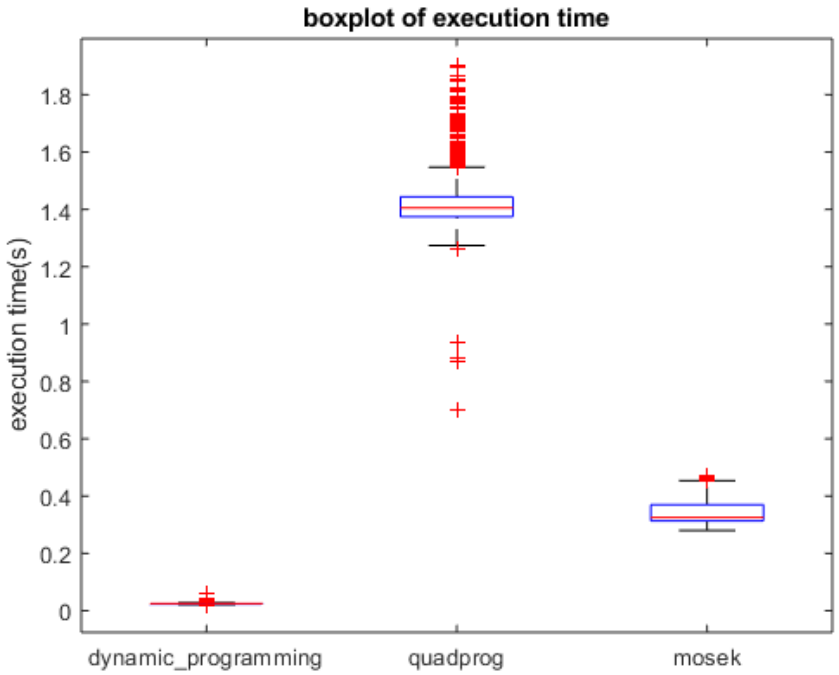


Figure 6: The boxplot of execution time of 1-minute based portfolio optimization problem with for AAPL quote data in Januay 2017.

rithm has potentially higher performance advantage since the routines of quadprog solver and mosek solver are implemented in high performance language (like C or mex-file), but our algorithm only uses MATLAB language. Figure 8 shows the execution time for 5-minute system. The median execution time of dynamic programming algorithm, quadprog and mosek are 0.0011 seconds, 0.0041 seconds and 0.0031 seconds, respectively.

Beside the median execution time advantage, of our dynamic programming algorithm is also more stable than quadprog solver and mosek solver. The range of the boxplot of our algorithm is much thinner than the quadprog solver, which has the widest execution time range, and the mosek solver. In fact, in 1-minute system test, the variances of three solvers are 2.2e-7, 8.4e-5 and 1.9e-6, and in the 5-minute system test the variances are 2.1e-8, 2.1e-7 and 1.6e-7.

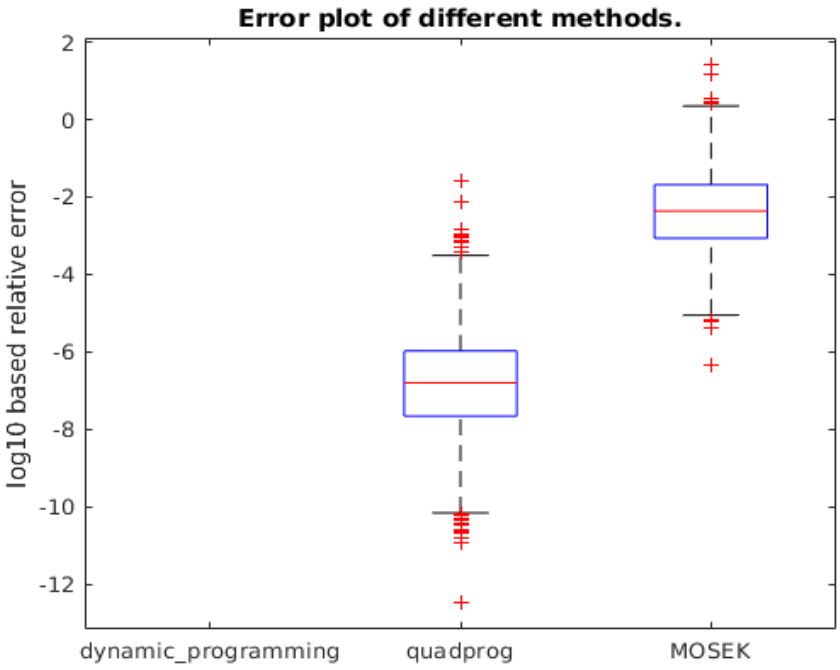


Figure 7: The relative error plot of the objective function value of 5-minute based portfolio optimization problem with for AAPL quote data in Januay 2017.

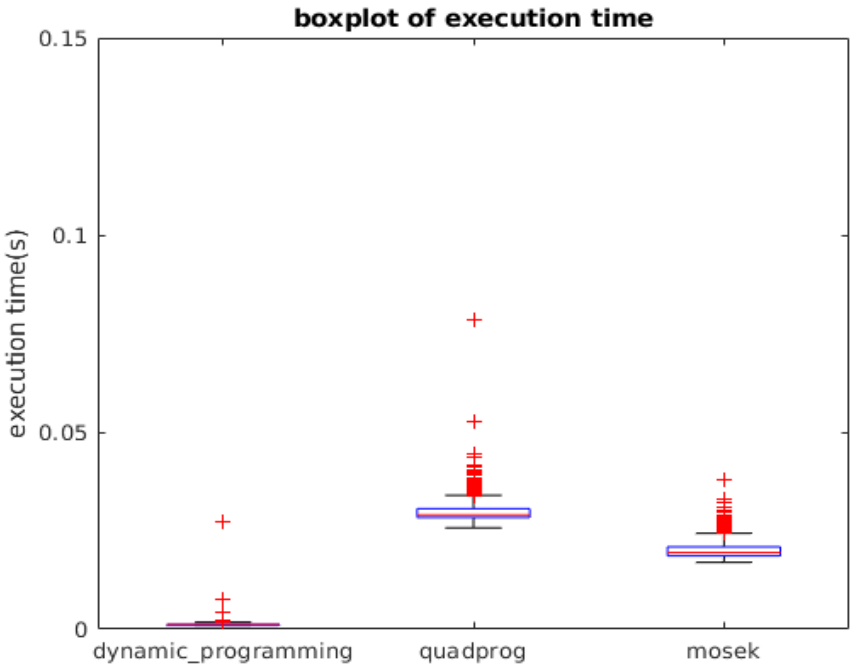


Figure 8: The boxplot of execution time of 5-minute based portfolio optimization problem with for AAPL quote data in Januay 2017.

# 5 ALGORITHM FOR MULTI-INSTRUMENT PORTFOLIO SELECTION

In this chapter we discuss the algorithms that extends the dynamic programming idea of portfolio optimization problem to higher dimension cases. We start from a direct extension in two dimension case with an explanation of the difficulty of extending the idea into even higher dimension (higher than 2). Then we show an operator splitting scheme called trading holding splitting that allow us successfully extending the dynamic programming algorithm into higher dimension as an iterative solver.

## 5.1 An extension to two instruments case

In this section we discuss the possible extensions of the dynamic programming algorithm above, and show why it is hard to extend this algorithm to higher dimension.

For the simplicity of notation, we assume in the two instrument case, the risk multiplier $\beta_i$ is 1 for all the periods. We also assume the covaraince estimation $\Sigma_i$ keeps the same over the period

$$\Sigma_i = \begin{bmatrix} \sigma_{11} & \sigma_{12} \\ \sigma_{21} & \sigma_{22} \end{bmatrix}.$$

Then the two instrument multi-period portfolio optimization problem is

$$\begin{aligned}
\underset{u_1,u_2,...,u_n}{\text{minimize}} \quad & \sum_{i=1}^{n}\Big(\frac{1}{2}\begin{bmatrix}u_i^1\\u_i^2\end{bmatrix}^T\begin{bmatrix}\sigma_{11} & \sigma_{12}\\ \sigma_{21} & \sigma_{22}\end{bmatrix}\begin{bmatrix}u_i^1\\u_i^2\end{bmatrix} - \begin{bmatrix}r_i^1\\r_i^2\end{bmatrix}^T\begin{bmatrix}u_i^1\\u_i^2\end{bmatrix} \\
& + \begin{bmatrix}\tau_i^1\\\tau_i^2\end{bmatrix}^T\begin{bmatrix}\left|u_i^1-u_{i-1}^1\right|\\ \left|u_i^2-u_{i-1}^2\right|\end{bmatrix} + \begin{bmatrix}u_i^1\\u_i^2\end{bmatrix}^T\begin{bmatrix}\kappa_i^1 & 0\\ 0 & \kappa_i^2\end{bmatrix}\begin{bmatrix}u_i^1\\u_i^2\end{bmatrix}\Big) \\
\text{subject to} \quad & posl b_i^1 \le u_i^1 \le posub_i^1, \quad i=1,2,...,n \\
& posl b_i^2 \le u_i^2 \le posub_i^2, \quad i=1,2,...,n \\
& trdlb_i^1 \le u_i^1 - u_{i-1}^1 \le trdub_i^1, \quad i=1,2,...,n \\
& trdlb_i^2 \le u_i^2 - u_{i-1}^2 \le trdub_i^1, \quad i=1,2,...,n
\end{aligned} \tag{5.1}$$

We can write down the sub-derivative of the objective function with respect to each instrument:

$$\frac{\partial F}{\partial u_i^1} = \begin{cases}
\sigma_{11}u_i^1 + \sigma_{12}u_i^2 - r_i^1 + \tau_i^1\text{sign}(u_i^1-u_{i-1}^1) + 2\kappa_i^1(u_i^1-u_{i-1}^1) \\
\qquad +\mathcal{I}_i(u_i^1) - \tau_{i+1}^1\text{sign}(u_{i+1}^1-u_i^1) - 2\kappa_{i+1}^1(u_{i+1}^1-u_i^1) \\
\qquad\qquad 1 \le i < n, \\
\sigma_{11}u_n^1 + \sigma_{12}u_n^2 - r_n^1 + \tau_n^1\text{sign}(u_n^1-u_{n-1}^1) + 2\kappa_n^1(u_n^1-u_{n-1}^1) \\
\qquad +\mathcal{I}_n(u_n^1) \\
\qquad i = n
\end{cases}$$

$$\frac{\partial F}{\partial u_i^2} = \begin{cases}
\sigma_{21}u_i^1 + \sigma_{22}u_i^2 - r_i^2 + \tau_i^2\text{sign}(u_i^2-u_{i-1}^2) + 2\kappa_i^2(u_i^2-u_{i-1}^2) \\
\qquad +\mathcal{I}_i(u_i^2) - \tau_{i+1}^2\text{sign}(u_{i+1}^2-u_i^2) - 2\kappa_{i+1}^2(u_{i+1}^2-u_i^2) \\
\qquad\qquad 1 \le i < n, \\
\sigma_{21}u_n^1 + \sigma_{22}u_n^2 - r_n^2 + \tau_n^2\text{sign}(u_n^2-u_{n-1}^2) + 2\kappa_n^2(u_n^2-u_{n-1}^2) \\
\qquad +\mathcal{I}_n(u_n^2) \\
\qquad i = n
\end{cases} \tag{5.2}$$

Comparing the sub-derivative (5.2) with single instrument case (4.18), one can see that the only difference is the risk adjusted return terms $\sigma_{12}u_i^2$ and $\sigma_{21}u_i^1$. In other words, given a position of $u_i^2$, the subderivative $\partial F/\partial u_i^1$ is as same as the one instrument case with an adjusted return

$(r_i^1 - \sigma_{12}u_i^2)$. Also given a position of $u_i^1$, the subderivative $\partial F/\partial u_i^2$ is as same as the one instrument case with an adjusted return $(r_i^2 - \sigma_{21}u_i^1)$. Therefore the sub-derivative for each instrument has a symmetry, which allow us to discuss only one of them without loss of generality.

To illustrate this process clearly, we focus on the first instrument $u^1$ in the following. The optimal condition for $u^1$ in this problem is

$$\begin{cases} 0 \in \sigma_{11}u_i^1 + \sigma_{12}u_i^2 - r_i^1 + \tau_i^1 \text{sign}(u_i^1 - u_{i-1}^1) + 2\kappa_i^1(u_i^1 - u_{i-1}^1) \\ \qquad\qquad +\mathcal{I}_i'(u_i^1) - \tau_{i+1}^1 \text{sign}(u_{i+1}^1 - u_i^1) - 2\kappa_{i+1}^1(u_{i+1}^1 - u_i^1) \\ \qquad\qquad\qquad\qquad 1 \le i < n, \\ 0 \in \sigma_{11}u_n^1 + \sigma_{12}u_n^2 - r_n^1 + \tau_n^1 \text{sign}(u_n^1 - u_{n-1}^1) + 2\kappa_n^1(u_n^1 - u_{n-1}^1) \\ \qquad\qquad +\mathcal{I}_n'(u_n^1) \\ \qquad\qquad\qquad\qquad i = n \end{cases} \tag{5.3}$$

The equation (5.3) typically means there is a series of values $u_i^1 = [u_1^1, u_2^1, ..., u_n^1]^T$ satisfy the equation

$$\begin{cases} \tau_i^1 \text{sign}(u_i^1 - u_{i-1}^1) + 2\kappa_i^1(u_i^1 - u_{i-1}^1) = r_i^1 - \sigma_{12}u_i^2 - \sigma_{11}u_i^1 \\ \qquad\qquad -\tau_{i+1}^1 \text{sign}(u_{i+1}^1 - u_i^1) - 2\kappa_{i+1}^1(u_{i+1}^1 - u_i^1) + \mathcal{I}_i'(u_i^1) \\ \qquad\qquad\qquad\qquad 1 \le i < n, \\ \tau_n^1 \text{sign}(u_n^1 - u_{n-1}^1) + 2\kappa_n^1(u_n^1 - u_{n-1}^1) = \\ \qquad\qquad\qquad\qquad r_n^1 - \sigma_{11}u_n^1 - \sigma_{12}u_n^2 + \mathcal{I}_n'(u_n^1) \\ \qquad\qquad\qquad\qquad i = n \end{cases} \tag{5.4}$$

Similarly with what we have shown in one instrument case, we define a function $F_{\text{left}}(u_{i-1}^1, u_i^1, u_i^2)$ as

$$\nabla F_{\text{left}}(u_{i-1}^1, u_i^1, u_i^2) = \tau_i^1 \text{sign}(u_i^1 - u_{i-1}^1) + 2\kappa_i^1(u_i^1 - u_{i-1}^1) + \mathcal{I}_i'(u_i^1), \tag{5.5}$$

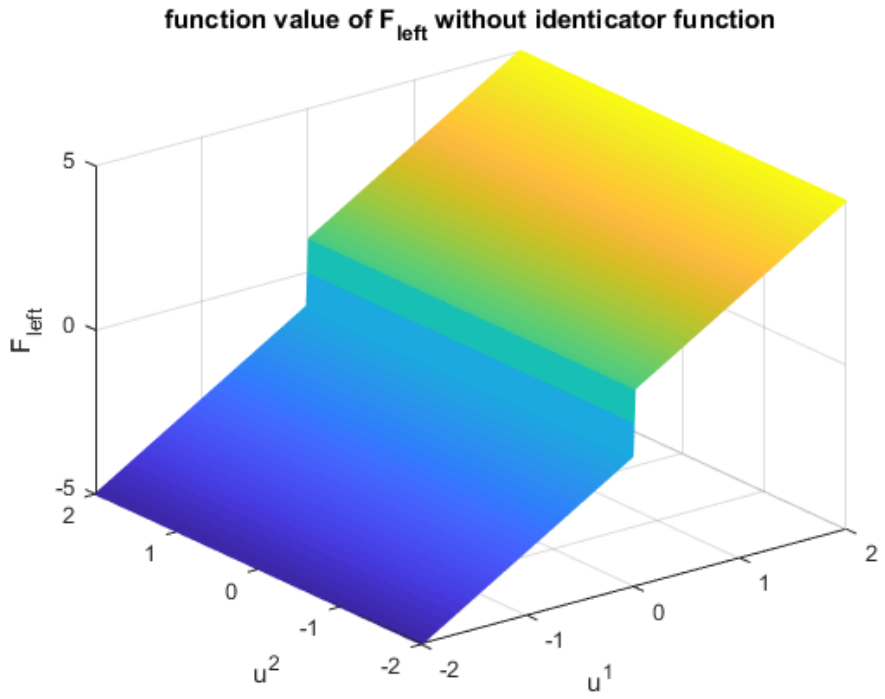


Figure 9: function value plot of $F_{\text{left}}(u_{i-1}^1, u_i^1, u_i^2)$ against $u_i^1$ and $u_i^2$ without the indicator function $\mathcal{I}_i'(u_i^1)$, parameters are set as $\tau_i^1 = 1, u_{i-1}^1 = 0, \kappa_i^1 = 1$

and $\nabla F_{\text{right}}(u_i^1, u_{i+1}^1, u_i^2)$

$$\nabla F_{\text{right}}(u_i^1, u_{i+1}^1, u_i^2) = \begin{cases} r_i^1 - \sigma_{11}u_i^1 - \sigma_{12}u_i^2 + \tau_{i+1}^1 \text{sign}(u_{i+1}^1 - u_i^1) \\ \qquad +2\kappa_{i+1}^1(u_{i+1}^1 - u_i^1) + \mathcal{I}_{i+1}'(u_{i+1}^1) \\ \qquad\qquad\qquad\qquad\qquad 1 \le i < n, \\ r_n^1 - \sigma_{11}u_n^1 - \sigma_{12}u_n^2, \qquad\qquad i = n \end{cases} \tag{5.6}$$

As same as the one instrument case, the aim is matching the value of $F_{\text{left}}(u_{i-1}^1, u_i^1, u_i^2)$ and $\nabla F_{\text{right}}(u_i^1, u_{i+1}^1, u_i^2)$. Notice that $u_i^2$ does not affect the value of $F_{\text{left}}(u_{i-1}^1, u_i^1, u_i^2)$ since it does not have any terms related to $u_i^2$. This means the projection of $F_{\text{left}}(u_{i-1}^1, u_i^1, u_i^2)$ on the $(F_{\text{left}}, u_i^1)$ plane has exact the same shape as (5.5). Figure 9 gives an example of the shape of $F_{\text{left}}(u_{i-1}^1, u_i^1, u_i^2)$ when $\tau_i^1 = 1, u_{i-1}^1 = 0, \kappa_i^1 = 1$ and. The right hand side $\nabla F_{\text{right}}(u_i^1, u_{i+1}^1, u_i^2)$ also has similar shape with one in-

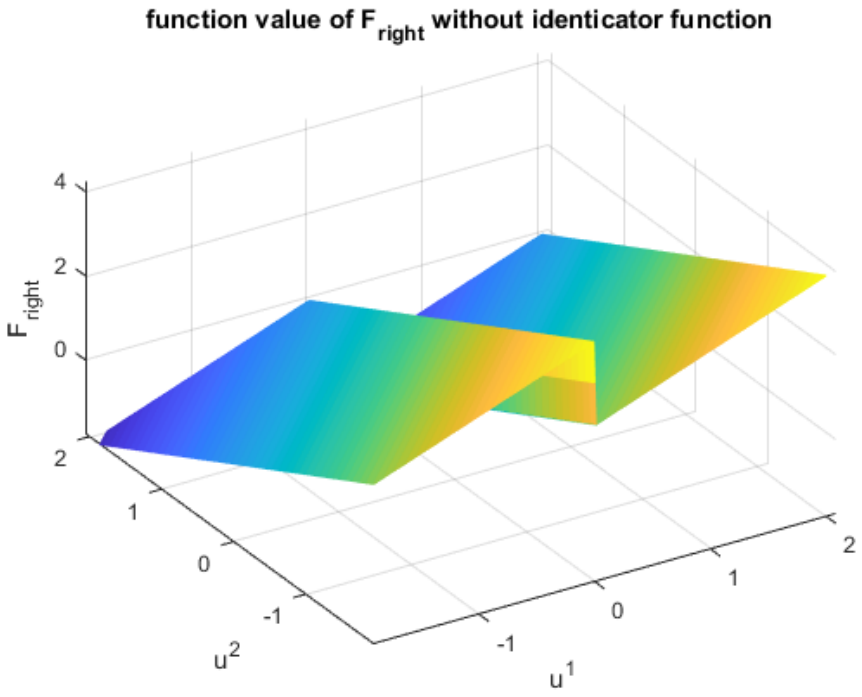


Figure 10: The function value plot of $\nabla F_{\text{right}}(u_i^1, u_{i+1}^1, u_i^2)$ $(i < n)$ without the indicator function $\mathcal{I}'_{i+1}(u_{i+1}^1)$, given that $u_{i+1}^1 = 0, r_i^1, \sigma_{11} = 1, \sigma_{12} = 1, \tau_{i+1}^1 = 1, \kappa_i^1 = 1$

strument case except the linear adjustment part $\sigma_{12}u_i^2$. Figure 10 gives a simple example plot of $\nabla F_{\text{right}}(u_i^1, u_{i+1}^1, u_i^2)$ with respect to $u_i^1$ and $u_i^2$ when $i < n$.

Finding the intersection of two 3-D shapes are not trivial in general. However, with the help of piece-wise linearity, we can find the intersection and perform the merge process by running two merge processes at two different values of $u_i^2$. Then all the other merge processes can be characterized by these two processes. In practical, we compute two merge process with $u_i^2 = poslb_i^2$ and $u_i^2 = posub_i^2$ since all the possible values taken by $u_i^2$ are in the interval $[poslb_i^2, posub_i^2]$. With these two merge processes (breakpoint mapping in practical), we can interpolate all the merge processes with any permissible value of $u_i^2$. Thus we can save all the possible optimal solution of $u_i^1(u_i^2)$ given any value of $u_i^2(u_i^1)$ by running two merge process, so the total number of merge process running by the algorithm are four.

Running the merge process in two dimension case can give us an optimal solution mapping

$$A_i^1(u_{i-1}^1, u_i^2) = (u_i^1)^*$$
$$A_i^2(u_{i-1}^2, u_i^1) = (u_i^2)^*$$

One can get the optimal solution pair $((u_i^1)^*, (u_i^2)^*)$ by combining two optimal mapping $A_i^1$ and $A_i^2$. We discuss the detail of this process in the following. We still focus on the first instrument $u_i^1$ since process works as same as $u_i^2$. The first is the recovery of the shape of $A_i^1$. (5.4) shows that with the perturbation of $u_i^2$, i.e. from $u_i^2$ to $u_i^2 + \delta$, the optimal value $(u_i^1)^*$ moves with the value $\frac{\sigma_{12}\delta}{2\kappa_i^1}$ when $\kappa_i^1 \neq 0$ and keeps the same when $\kappa_i^1 = 0$ or in the no trade region. With this knowledge, we can recover the shape of $A_i^1$ as the following: we start from one bound of $u_i^2$, for example, the position upper bound $posub_i^2$, find the optimal value with the current inventory $u_{i-1}^1$, then generate the value of $u_i^1$ linearly against $u_i^2$ with the slope $\frac{\sigma_{12}\delta}{2\kappa_i^1}$, until hitting a no trade region. In no trade region the optimal value of $u_i^1$ keeps the same for any value of $u_i^2$, until there is a trade for the first instrument. We alternate the different "regimes" between the trade regions and no trade regions until $u_i^2$ hits the position lower bound $poslb_i^2$.

Figure 11 shows an example of the shape of $A_i^1$ described above. The horizontal line on the top and bottom gives the position upper bound and the position lower bound of $u_i^2$. The vertical line on the left and right gives the position upper bound and the position lower bound of $u_i^1$. The red line segments are the desired position $(u_i^1)^*$ under different value of $u_i^2$. These line segments are connected. Region A and E are the trading bound regions. In these two region the optimal solution $(u_i^1)^*$ hits the trade upper(lower) bound thus the shape of $(u_i^1)^*$ hits the boundary of A and E. Region B and D are trade regions. In these two region $(u_i^1)^*$ is a linear function of $u_i^2$ with $\frac{\sigma_{12}}{2\kappa_i^1}$ as the slope. Region C is no trade region so that the desired position $(u_i^1)^*$ in this region is a vertial line segment.

The generation of the optimal solution pair $((u_i^1)^*, (u_i^2)^*)$ is simple

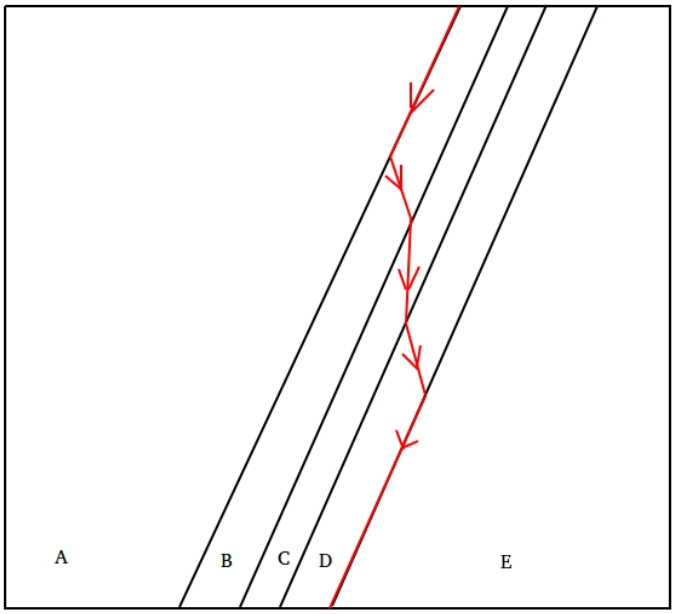


Figure 11: The shape of $A_i^1$.

once the shapes of optimal solution mapping $A_i$ are computed. One idea is enumerating all the line segments and to examine if two line segments in $A_i^1$ and $A_i^2$ have intersection.

We test the dynamic programming algorithm in 2 instrument case with 5 period prediction. Figure 12 gives a simple performance test between the dynamic programming algorithm and the quadprog algorithm in MATLAB. The median execution time of dynamic programming algorithm is 0.015 seconds whereas the median execution time of quadprog is 0.069 seconds.

**The difficulty.** We provide an explanation of the difficulty of extending this algorithm into even higher dimension: We have seen the merge process, which is the key process in this algorithm, is a one dimension process (merging the breakpoints) in one instrument case. And it is a two dimension process (merging the breaklines) in two instrument case. Therefore in even higher dimension we have to create a merge process on an n-dimensional item, which is very hard and time consuming without

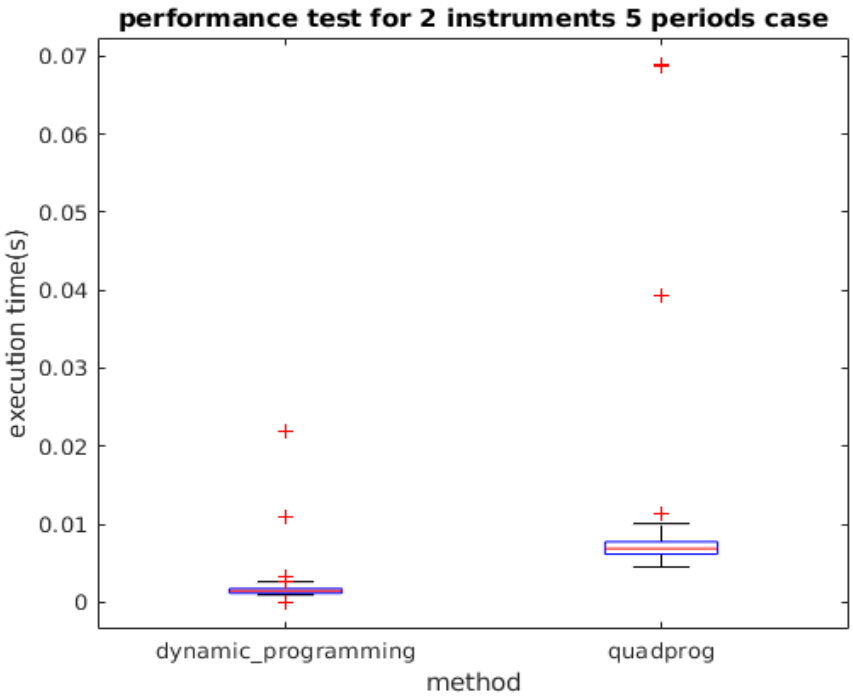


Figure 12: Performance test for two instrument five period test.

any dimension reduction technique.

## 5.2 Holding-Trading splitting

The key issue of extending the idea in chapter 4 into larger number of instrument case is the covariance part in objective function. To see this, recall that in the two instruments case the optimal mapping $A_i$ is a function of $u_i^1$ and $u_i^2$, where the only term $u_i^2$ involved in is the off diagonal part $\sigma_{12}u_i^2$. In fact, the multi-period multi-instrument portfolio optimization problem (3.11) is reduced to m(number of instruments) individual one instrument multi-period portfolio optimization problem if the covariance is a diagonal matrix. This tells us that if we can split off diagonal part of the covariance part out of the whole problem, we can get the desired position of $i$th period $u_i^*$ by running m times dynamic programming. With this aim, we are ready to introduce the *Holding-Trading splitting*.

We split the objective function in (3.11) with the following two pieces:

$$\begin{aligned} F(u) &= f(u) + g(u) \\ f(u) &= \sum_{i=1}^{T} (\beta_i u_i^T \Sigma_i u_i - r_i^T u_i) \\ g(u) &= \sum_{i=1}^{T} (\tau_i^T |u_i - u_{i-1}| + (u_i - u_{i-1})^T D_{\kappa_i} (u_i - u_{i-1})) \end{aligned} \tag{5.7}$$

We call this splitting the *Holding-Trading splitting scheme* because of the corresponding interpretation. The first part $f(u)$ has the same form as the objective function of multi-period Markowitz portfolio selection problem. Here we provide another interpretation for $f(u)$. The objective function for $i$th period

$$f_i(u) = \beta_i u_i^T \Sigma_i u_i - r_i^T u_i$$

can be seen as the utility from the start of $i$th period to the end of $i$th period (here we assume the trading get done without any cost of time). Therefore this utility is fully determined by the holding position in $i$th period. Hence the sum of all the periods is the utility generated by holding the position $u_i$ at $i$th period. The objective function $g(u)$ consists of the transaction costs of each period, which can be seen as the sum of the trading utility over each period. We put the constraints of the problem (3.11) into the function $g(u)$ since the constraints essentially regulates the trades of each period. With this scheme, we can rewrite the portfolio selection problem as a minimization problem of the sum of

two convex functions.

$$\begin{aligned}
\underset{u_1,u_2,\ldots,u_T}{\text{minimize}} \quad & F(u) = f(u) + g(u) \\
\text{subject to} \quad & f(u) = \sum_{i=1}^{T}(\beta_i u_i^T \Sigma_i u_i - r_i^T u_i) \\
& g(u) = \sum_{i=1}^{T}(\tau_i^T |u_i - u_{i-1}| + (u_i - u_{i-1})^T D_{\kappa_i}(u_i - u_{i-1}) \\
& \qquad + \mathcal{I}_{\Omega_i}(u_i))
\end{aligned} \tag{5.8}$$

where the indicator function

$$\mathcal{I}_{\Omega_i}(u_i) = \begin{cases} 0, \ u_i \in \Omega_i \\ +\infty, \ \text{otherwise} \end{cases} \tag{5.9}$$

gives the permissible portfolio region. By the definition of (2.13) one can see the indicator function (5.9) is a proper convex function.
The problem (5.8) can be solved by proximal gradient algorithm (see chapter 2 for detail)

$$u^{k+1} = \text{prox}_{\gamma_k g}(u^k - \nabla f(u^k)) \tag{5.10}$$

In the context of (5.8), we use a half step represent the proximal gradient algorithm with *Holding-Trading splitting*

$$\begin{aligned}
u^{k+\frac{1}{2}} &= u^k - \nabla f(u^k) \\
&= u^k - \Sigma u^k \\
u^{k+1} &= \text{prox}_{\gamma g}(u^{k+1}) \\
&= \underset{x}{\text{argmin}} \quad g(x) + \frac{1}{2\gamma_k}\left\| u^{k+\frac{1}{2}} - x \right\|_2^2 \\
&= \underset{x}{\text{argmin}} \ \sum_{i=1}^{T}(\frac{1}{2\gamma} x_i^T x_i - \frac{1}{\gamma_k} u_i^{k+\frac{1}{2}} x_i + \tau_i^T |x_i - x_{i-1}| + \\
& (x_i - x_{i-1})^T D_{\kappa_i}(x_i - x_{i-1}) + \mathcal{I}_{\Omega_i}(x_i))
\end{aligned} \tag{5.11}$$

The splitting scheme makes sense when each component is easy to evaluate. In the holding-trading splitting scheme, we have to evaluate two half steps several times during the whole procedure: a gradient descent step on holding utility

$$u^{k+\frac{1}{2}} = u^k - \nabla f(u^k) \tag{5.12}$$

and then a proximal point evaluation on trading utility

$$\begin{aligned} \underset{x_1,x_2,...,x_n}{\text{minimize}} \quad & \sum_{i=1}^{T}(\frac{1}{2\gamma_k}x_i^T x_i - \frac{1}{\gamma_k}u_i^{k+\frac{1}{2}}x_i + \tau_i^T|x_i - x_{i-1}| + \\ & (x_i - x_{i-1})^T D_{\kappa_i}(x_i - x_{i-1}) + \mathcal{I}_{\Omega_i}(x_i)) \\ \text{subject to} \quad & poslb_i \le u_i \le posub_i,\ i = 1, 2, ..., T \\ & trdlb_i \le u_i \le trdub_i,\ i = 1, 2, ..., T \end{aligned} \tag{5.13}$$

(5.12) is common matrix-vector multiplication. For (5.13), let

$$\begin{aligned} g(x) = \sum_{i=1}^{T}(&\frac{1}{2\gamma_k}x_i^T x_i - \frac{1}{\gamma_k}u_i^{k+\frac{1}{2}}x_i + \tau_i^T|x_i - x_{i-1}| + \\ & (x_i - x_{i-1})^T D_{\kappa_i}(x_i - x_{i-1}) + \mathcal{I}_{\Omega_i}(x_i)) \end{aligned}$$

and $x_{i,j}$ be the position of $i$th instrument over $j$th period. Then we have

$$\begin{aligned} g(x) &= \sum_{i=1}^{T}(\frac{1}{2\gamma_k}x_i^T x_i - \frac{1}{\gamma_k}u_i^{k+\frac{1}{2}}x_i + \tau_i^T|x_i - x_{i-1}| + \\ &\qquad (x_i - x_{i-1})^T D_{\kappa_i}(x_i - x_{i-1}) + \mathcal{I}_{\Omega_i}(x_i)) \\ &= \sum_{i=1}^{T}\sum_{j=1}^{n}(\frac{1}{2\gamma_k}x_{i,j}^2 - \frac{1}{\gamma_k}u_{i,j}^{k+\frac{1}{2}}x_{i,j} + \tau_{i,j}\big|x_{i,j} - x_{i-1,j}\big| + \\ &\qquad \kappa_{i,j}(x_{i,j} - x_{i-1,j})^2 + \mathcal{I}_{\Omega_{i,j}}(x_{i,j})) \\ &= \sum_{j=1}^{n}\sum_{i=1}^{T}(\frac{1}{2\gamma_k}x_{i,j}^2 - \frac{1}{\gamma_k}u_{i,j}^{k+\frac{1}{2}}x_{i,j} + \tau_{i,j}\big|x_{i,j} - x_{i-1,j}\big| + \\ &\qquad \kappa_{i,j}(x_{i,j} - x_{i-1,j})^2 + \mathcal{I}_{\Omega_{i,j}}(x_{i,j})) \\ &= \sum_{j=1}^{n} g_j(x_{\cdot,j}). \end{aligned} \tag{5.14}$$

(5.14) shows that the objective function $g(x)$ is separable, i.e., it is the sum of the objective function of one instrument multi-instrument portfolio optimization problem over all instruments, which can be solved using the fast dynamic programming algorithm introduced in chapter 2.

Algorithm 3 gives the whole algorithm described in this section. It is well known that the proximal gradient method can achieve linear convergence, i.e., it attains $\epsilon$ accuracy within $O(\log \frac{1}{\epsilon})$ iterations.

**Algorithm 3** find optimal portfolio u

**Require:** $\Sigma,\ r,\ \tau,\ \kappa,\ poslb,\ posub,\ trdlb,\ trdub,\ u_0,\ n, \epsilon$;
**Ensure:** $u$;
tolerance $= \epsilon * 10$;
$k = 1$;
**while** tolerance $> \epsilon$ **do**
  $\gamma_k = 1$;
  **for** $i = 1 : n$ **do**
    $u_i^{k+\frac{1}{2}} = u_i^k - \Sigma_i u_i^k$;
  **end for**
  run algorithm 2 for each instrument $u_{\cdot,j}^{k+1}$ with variance $\frac{1}{2\gamma_k}$ and the return $\frac{1}{\gamma_k} u_{\cdot,j}^{k+\frac{1}{2}}$;
  tolerance $= \frac{\left\|u^{k+1} - u^k\right\|}{\left\|u^k\right\|}$
  **if** tolerance $\leq \epsilon$ **then**
    **return** $u^{k+1}$;
  **else** {tolerance $> \epsilon$,}
    $k = k + 1$;
  **end if**
**end while**

## 5.3 Semismooth Newton Methods

In the previous section we have shown an algorithm solving the multi-instrument multi-period portfolio optimization problem (3.11). In this section we introduce semismooth newton methods in order to improve the convergence rate to second order. The method largely depends on the nonsmooth analysis, we refer [32] for backgrounds.

Recall that the algorithm 3 is essentially a fixed point iteration of

the following equation

$$u = \text{prox}_{\gamma g}(u - \nabla f(u)) \tag{5.15}$$

It is equivalent to solve the system

$$G(u) = u - \text{prox}_{\gamma g}(u - \nabla f(u)) = 0 \tag{5.16}$$

The system (5.16) can be solved by any iterative methods (like Newton's method) when the left side has continuous second order derivative. But this requires the proximal mapping $\text{prox}_{\gamma g}(x)$ in $\mathcal{C}^2$. However, since $g$ does not have continuous second order derivative (one can see this from the permissible portfolio identification function $\mathcal{I}(u)$ and the proportional transaction cost term $\tau^T|u - u_0|$), the proximal mapping can only be a function in $\mathcal{C}^1$ but not in $\mathcal{C}^2$. One way to overcome this issue is using the nonsmooth analysis tools, typically clarke's generalized Jacobian (see chapter 2 for background) to extend the Newton's method to nonsmooth functions. The main advantage for semismooth Newton methods is that they are very similar to the classical Newton methods for smooth equations with same characteristics.

Algorithm 4 gives a simple semismooth Newton method for solving the fixed point iteration system (5.16). It requires the computation of the Clarke generalized Jacobian $\partial G(u)$ in each iteration (see chapter 2 for the definition). Here we discuss a simple way to evaluate one element of Clarke generalized Jacobian for our portfolio optimization problem.

We choose the generalized Jacobian by simply taking the partial derivative of $G(u)$ in (5.16) against each variable $u_{i,j}$. First, from the chain rule we know

$$\begin{aligned} \partial_C G(u) &= I - (I - \nabla^2 f)^T \partial_C(\text{prox}_{\gamma g})(u) \\ &= I - (I - \nabla^2 f)\partial_C(\text{prox}_{\gamma g})(u) \end{aligned} \tag{5.17}$$

**Algorithm 4** Semismooth Newton Method for (5.16)

**Require:** $f,\ g,\ u_0, \epsilon$;
**Ensure:** $u$;
$G(u^k) = u^k - \text{prox}(u^k - \nabla f(u^k))$;
tolerance $= \epsilon * 10$;
$k = 1$;
**while** tolerance $> \epsilon$ **do**
$\gamma_k = 1$;
Select an element $H^k \in \partial G(u^k)$. Find a direction such that

$$G(u^k) + H^k d^k = 0.$$

$u^{k+1} = u^k + d^k$;
tolerance $= \left\| d^k \right\|$
**if** tolerance $\leq \epsilon$ **then**
**return** $u^{k+1}$;
**else** {tolerance $> \epsilon$,}
$k = k + 1$;
**end if**
**end while**

Here $f$ is a smooth holding utility function so that $(I - \nabla^2 f)$ is a symmetric positive semi-definite matrix and has closed form

$$(I - \nabla^2 f)_i = I - \Sigma_i \tag{5.18}$$

The evaluation of $\partial_C(\text{prox}_{\gamma g})(u)$ is much harder. From (5.14) we know that the proximal mapping is separable, so as the Clarke generalized Jacobian of it. Therefore we can evaluate the Clarke generalized Jacobian for each instrument and combined them together. For each instrument, we can get an element of Clarke generalized Jacobian using the perturbation idea.

Recall that for the one instrument multi-period portfolio optimization problem, the proximal mapping is

$$\text{prox}_{\gamma g}(u) = \underset{x}{\text{argmin}}\ \{g(x) + \frac{1}{2\gamma}\|u - x\|^2\}.$$

where $x$ and $u$ are a $T$-dimension vectors. By perturbing $u_i$, we have a representation of $\partial_C(\text{prox}_{\gamma g})(u)$

$$\partial_C(\text{prox}_{\gamma g})(u)_{j,k} = [\text{prox}_{\gamma g}(u + \delta u_j) - \text{prox}_{\gamma g}(u)]_k \tag{5.19}$$

Under the context of one instrument multi-period portfolio optimization problem, it is equivalent to ask how will the desired position (optimal solution) change if we perturb one period of return by a small amount. We can ask this question from the merge process. We analyze the effect of return perturbation in two parts:

(i) The effect on $x_j$ from the perturbation of $u_j$:
First, if the desired position hits the bound (either position bound or trade bound), the perturbation on the return does not change the desired position unless the desired position just hits the bound (this means the desired position keeps the same without this bound constraint). However, it is safe to say the desire position does not change under this condition since it is in the convex hull of limiting Jacobian. Then in this case, we know

$$\partial_C(\text{prox}_{\gamma g})(u)_{j,j} = 0. \tag{5.20}$$

The second condition is the desired position locates in no-trade region. With the similar analysis we know that the desired position does not change under the small return perturbation

$$\partial_C(\text{prox}_{\gamma g})(u)_{j,j} = 0 \quad \text{if no trade.} \tag{5.21}$$

The remaining case is the desired position locates in trade region but does not hit the bound constraint. In this case, from the analysis in one instrument case (4.20) we know that the desired position is proportional to the return perturbation:

$$u_i^* = \begin{cases} \frac{r_i}{2(\kappa_i+\kappa_{i+1})} + C, & 1 \le i < n, \\ \frac{r_i}{2\kappa_n} + C, & i = n, \end{cases} \tag{5.22}$$

Therefore the Clarke generalized Jacobian is

$$\partial_C(\text{prox}_{\gamma g})(u)_{j,j} = \begin{cases} \frac{1}{2(\kappa_i+\kappa_{i+1})}, & 1 \leq i < n, \\ \frac{1}{2\kappa_n}, & i = n, \end{cases} \tag{5.23}$$

(ii) The effect on $x_k$ from the perturbation on $u_j (k \neq j)$.
In this case, we need consider all the changes of desired position between $k$th period and $j$th period. Here we assume $k < j$ for the following analysis as an example. We mention that the analysis keeps the same when $k > j$. Similarly with (i), when any desired position between $k$th period and $j$th period hits bound constraints or hits the no-trade region, the desired position at $k$th period does not change with the return perturbation at $j$th period

$$\partial_C(\text{prox}_{\gamma g})(u)_{j,k} = 0. \tag{5.24}$$

If all the desired positions from $k$th period to $j$th period have trades and do not hit any constraints, the Clarke generalized Jacobian is the production of each individual period by the chain rule

$$\partial_C(\text{prox}_{\gamma g})(u)_{j,k} = \begin{cases} \prod_{i=j}^{k} \frac{1}{2(\kappa_i+\kappa_i)}, & 1 < k < n, \\ \frac{1}{2\kappa_n} \prod_{i=j}^{n-1} \frac{1}{2(\kappa_i+\kappa_i)}, & k = n, \end{cases} \tag{5.25}$$

## 5.4 Choice of Step Size

Step size plays an important role in optimization. The main advantage is that with a careful choice of step size, the global convergence is guaranteed. The semismooth Newton method metioned above is a local method, which means the convergence can be guaranteed with unit step lengths unless it starts close to the solution. Otherwise the Jacobian may blow up, or the iterates move between distinct regions of the parameter space without approaching a root. In order to make semismooth Newton method more robust, several techniques are proposed. Here we

introduce two methods used in our problem.

**Backtracking line search**. One simple way is backtracking line search. Before describing the technique, we need introduce the *merit function* for $G(u)$.

We use the sum of squares, which is the most widely used merit function, as the merit function in our problem

$$m(u) = \left\|G(u)\right\|_2^2 = \left\|u - \text{prox}_{\gamma g}(u - \nabla f(u))\right\|_2^2 \tag{5.26}$$

One interpretation of the merit function (5.26) is that it measure the norm of residues. When the merit function is zero, the residual is zero, too. Then the values of $u^k$ at current iteration is exact the root of (5.16). Therefore solving the system (5.16) is equivalent with minimizing the merit function (5.26).

Semismooth Newton method only provides a vector $d^k$ indicating the direction to the root. This direction is also the descent direction of the merit function (5.26). In order to achieve sufficient progress in one iteration, one can choose the step size $\alpha_k$ that satisfies a sufficient decreasing condition like Armijo condition

$$\left\|G(u^k + \alpha_k d^k)\right\|_2^2 \leq \left\|G(u^k)\right\|_2^2 + c\alpha_k G(u^k)^T d^k \tag{5.27}$$

by using any line search technique, for example, the bisection method. Algorithm 5 shows the whole procedure of semismooth Newton method with backtracking line search methods.

**Barzilai-Borwein(BB) step**. Backtracking line search is simple but powerful when the evaluation of the merit function is easy. However, the merit function in our case contains a proximal mapping that requires an additional optimization procedure. Therefore we propose using *Barzilai-Borwein step*[2] instead of traditional backtracking line search.

The main idea of Barzilai-Borwein step is to use the information in the previous interation to determine the step-size of the current iteration.

Recall that the iteration formula is

$$u^{k+1} = u^k - (\partial_C G(u^k))^{-1} G(u^k) \tag{5.28}$$

The goal is choose a scalar $\alpha_k$ so that $\alpha_k G(u^k)$ approximates the Newton step $(\partial_C G(u^k))^{-1} G(u^k)$. Let

$$\begin{aligned} s^k &= u^k - u^{k-1} \\ y^k &= G^k - G^{k-1} \end{aligned} \tag{5.29}$$

We can get the approximation of Newton step by solving the least square problem

$$\begin{aligned} \alpha^k &= \underset{\alpha}{\operatorname{argmin}} \frac{1}{2} \left\| s^{k-1} - y^{k-1}\alpha \right\|^2 \\ &= \frac{(s^{k-1})^T y^{k-1}}{(y^{k-1})^T y^{k-1}} \end{aligned} \tag{5.30}$$

Algorithm 5 shows the whole procedure of semismooth Newton method with Barzilai-Borwein step size.

## 5.5 Numerical Experiment

In this section we compare the Algorithm 5 with several benchmarks described as the following:

**ADMM**[10]. Alternating Direction Method of Multipliers (ADMM) is an algorithm that solves convex optimization problems by breaking them into smaller pieces, each of which are then easier to handle. It has recently found wide application in a number of areas. We describe the basic ideas of ADMM and its implement for multi-instrument multi-period portfolio optimization problem (3.11).

ADMM solves problems in the form

$$\begin{aligned} \underset{x,z}{\text{minimize}} \quad & f(x) + g(z) \\ \text{subject to} \quad & Ax + Bz = c \end{aligned} \tag{5.31}$$

**Algorithm 5** Semismooth Newton Method for (5.16) with backtracking line search

**Require:** $f,\ g,\ u_0, \epsilon,\ c$;
**Ensure:** $u$;
$G(u^k) = u^k - \text{prox}(u^k - \nabla f(u^k))$;
tolerance $= \epsilon * 10$;
$k = 1$;
**while** tolerance $> \epsilon$ **do**
$\gamma_k = 1$;
Select an element $H^k \in \partial G(u^k)$. Find a direction such that

$$G(u^k) + H^k d^k = 0.$$

Find the max value of $\alpha_k \in \{1,\ 2^{-1},\ 2^{-2}, ...\}$ such that $\alpha_k$ satisfies the Armijo backtracking condition for the merit function (5.27);
Set $u^{k+1} = u^k + \alpha_k d^k$;
tolerance $= |\alpha_k| \left\| d^k \right\|$
**if** tolerance $\leq \epsilon$ **then**
**return** $u^{k+1}$;
**else** {tolerance $> \epsilon$,}
$k = k + 1$;
**end if**
**end while**

**Algorithm 6** Semismooth Newton Method for (5.16) with Barzilai-Borwein step

**Require:** $f,\ g,\ u_0, \epsilon,\ c$;
**Ensure:** $u$;
$G(u^k) = u^k - \text{prox}(u^k - \nabla f(u^k))$;
tolerance $= \epsilon * 10$;
$k = 1$;
**while** tolerance $> \epsilon$ **do**
$\gamma_k = 1$;
Select an element $H^k \in \partial G(u^k)$. Find a direction such that

$$G(u^k) + H^k d^k = 0.$$

Set $s^k = u^k - u^{k-1}$;
Set $y^k = G(u^k) - G(u^{k-1})$;
Set $\alpha_k = \frac{(s^{k-1})^T y^{k-1}}{(y^{k-1})^T y^{k-1}}$;
Set $u^{k+1} = u^k + \alpha_k d^k$;
tolerance $= |\alpha_k| \left\| d^k \right\|$
**if** tolerance $\leq \epsilon$ **then**
**return** $u^{k+1}$;
**else** {tolerance $> \epsilon$,}
$k = k + 1$;
**end if**
**end while**

with variables $x \in \mathbb{R}^n$ and $z \in \mathbb{R}^m$, where $A \in \mathbb{R}^{p\times n}$,$B \in \mathbb{R}^{p\times m}$, and $c \in \mathbb{R}^p$. In out problem, the variable $x$ and $z$ are the same, so ADMM reduces to

$$\begin{aligned} \underset{x,z}{\text{minimize}} \quad & f(x) + g(z) \\ \text{subject to} \quad & x - z = 0 \end{aligned} \tag{5.32}$$

By forming the augmented Lagrangian

$$L_\rho(x, z, y) = f(x) + g(z) + y^T(x - z) + \frac{\rho}{2}\|x - z\|^2 \tag{5.33}$$

ADMM consists of the iterations

$$\begin{aligned} x^{k+1} &= \underset{x}{\text{argmin}}\ L_\rho(x, z^k, y^k) \\ z^{k+1} &= \underset{z}{\text{argmin}}\ L_\rho(x^{k+1}, z, y^k) \\ y^{k+1} &= y^k + \rho(x^{k+1} - z^{k+1}) \end{aligned} \tag{5.34}$$

With the expression of $f$ and $g$ in (5.8), we have the ADMM iterations for out portfolio optimization problem

$$\begin{aligned} x_i^{k+1} &= (\frac{\rho}{2}I + \beta_i\Sigma_i)^{-1}(y_i^k + z_i^k) \\ z_{\cdot,j}^{k+1} &= \underset{z}{\text{argmin}} \sum_{i=1}^{T}(\frac{\rho}{2}z_i^2 - \rho y_i^k x_i^{k+1} z_i + \tau_i|z_i - z_{i-1}| \\ &\qquad\qquad +\kappa_i(z_i - z_{i-1})^2 + \mathcal{I}_{\Omega_i}) \\ y_i^{k+1} &= y_i^k + \rho(x_i^{k+1} - y_i^{k+1}) \end{aligned} \tag{5.35}$$

The parameter $\rho$ is chosen by the following logic[41, 10]

$$\rho^{k+1} = \begin{cases} 2\rho^k, & \text{if } \left\|r^k\right\|_2 > 10\left\|s^k\right\|_2 \\ \rho^k/2, & \text{if } \left\|s^k\right\|_2 > 10\left\|r^k\right\|_2 \\ \rho^k & \text{otherwise,} \end{cases} \tag{5.36}$$

where $s^{k+1} = \rho^k(z^k - z^{k+1})$ is the *dual residual* and $r^{k+1} = x^{k+1} - z^{k+1}$ is the *primal residual*.

**FBN**[67]. Forward-Backward truncated Newton method (FBN) is an

efficient Newton-like algorithm for convex optimization problems in composite form

$$\text{minimize } F(x) = f(x) + g(x) \tag{5.37}$$

Instead of minimizing $F(x)$, it considers a *forward-backward envelope* $F_\gamma$ of $F$

$$F_\gamma(x) = f(x) - \frac{\gamma}{2}\left\|\nabla f(x)\right\|_2^2 + g^\gamma(x - \gamma\nabla f(x)) \tag{5.38}$$

where $g^\gamma$ is the *Moreau envelope* of $g$

$$g^\gamma(x) = \inf_u \{g(u) + \frac{1}{2\gamma}\|u - x\|^2\} \tag{5.39}$$

FBN uses *approximate generalized Hessian*

$$\hat{\partial}^2 F_\gamma(x) = \{\gamma^{-1}(I - \gamma\nabla^2 f(x))(I - P(I - \gamma\nabla^2 f(x)))\} \tag{5.40}$$

where

$$P \in \partial_C(\text{prox}_{\gamma g})(x - \gamma\nabla f(x)) \tag{5.41}$$

With the approximate generalized Hessian, FBN uses conjugate gradient method to compute a descent direction $d^k$ and run a backtracking line search untill convergence. It has Q-quadratic convergence when the forward-backward envelope $F_\gamma$ is nice enough. We emphasize that FBN is the only splitting method which has a proved quadratic convergence so far.

We compare our algorithm with ADMM and FBM in two tests, the random test and real data back test.

**Random test**. In this test we run a 500 instrument 3 period portfolio optimization problem with trade bound and position bound constraints. We use random generated asset returns, random transaction cost multipliers $\tau > 0$ and $\kappa > 0$, random trade bound and position bound constraints with guaranteed feasibility (this means the problem has at least one solution), and a randomized factor model with 20 factors as the risk estimator.

Figure 13 shows the convergence result. As the benchmark, the

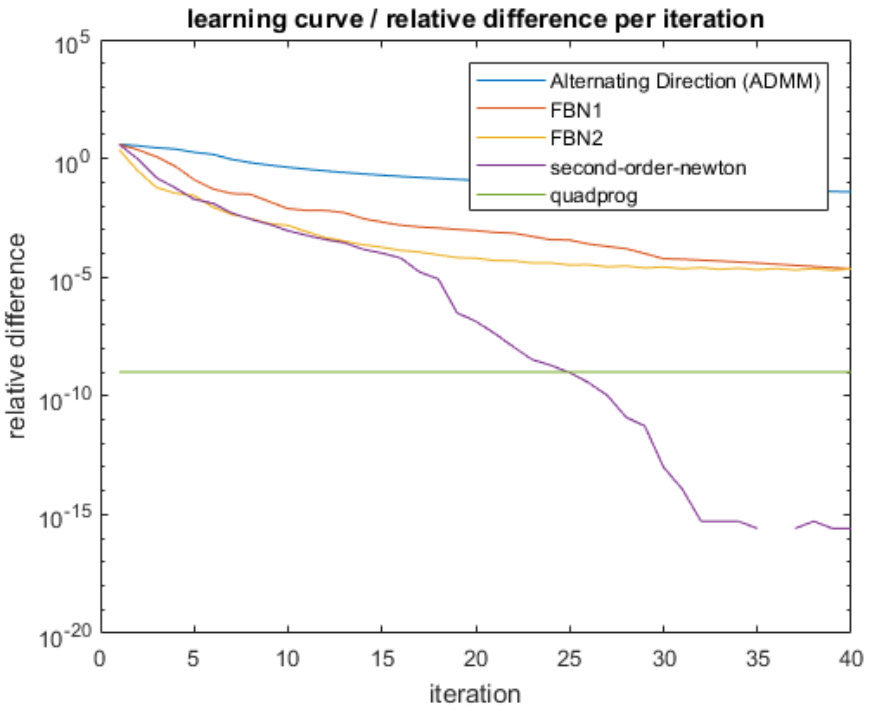


Figure 13: The convergence result of random test. The blue line is the convergence of ADMM, the red line and yellow line are two forward-backward Newton method and its variant in [67], the purple line is Algorithm 5. As a benchmark, the green line is the result coming from quadprog in MATLAB.

green line shows the relative difference of objective function value of quadprog function in MATLAB. ADMM (the blue line) and FBN (the red and yellow line) have linear convergence rate with a relatively slow parameter. And they stick in high relative error after 30 iterations. On the contrast, Algorithm 5 has similar convergence rate with FBN methods in the first 15 iterations. Then it beats FBN and converges in a quadratic rate, and beats quadprog after 25 iterations.

**Real quote data test**. Another test is using the historical quote data to examine the performance of our multi-instrument multi-period portfolio optimizer. In this test we simulate a back test of 5 minutes trading system with 78 periods of predictions in January 2017. Therefore the total trading cycle is 78 * 20 = 1560. We choose 10 Exchange-traded funds(ETF), which have the highest volume traded in the New York Stock Exchange (NYSE), as our target instruments in the simulation. Table (1) shows the names and descriptions of these ETFs. We setup the estimators as the following: For the covariance matrix, we use a shrinkage estimator [52] of the sample covaraince estimated using 100 periods historical log return data. We use the multiple input multiple output triangular input balanced (MIMO-TIB) filter [51, 83] as the predictions of ETF returns. The linear transaction cost multipliers $\tau$ are estimated as the half value of the median bid-ask spread for the same trading period over 3 months, i.e. the linear transaction cost multiplier for 10:30 a.m. is the half of the median bid-ask spread at 10:30 a.m. over past 3 month. We choose the quadratic transaction multipliers simply as 1. The trade bounds are estimated as 5% of the median trading volume over past 3 month. The position bounds are estimated as 5% of the asset's market capitalization in previous day.

We compare the Algorithm 6 with three benchmarks. One is the *quadprog* function in MATLAB which is the standard interior point solver with high performance implementation. Another one is a commercial solver "MOSEK"[1], which is widely used in mathematical finance area. The third one is a splitting method solver "OSQP"[76], which is the first standard splitting method solver implemented by ADMM

| NYSE ticker name | description |
|---|---|
| SPY | SPDR S&P 500 ETF |
| QQQ | Invesco QQQ Trust |
| DIA | SPDR Dow Jones Industrial Average ETF |
| GLD | SPDR Gold Shares |
| USO | United States Oil |
| TLT | iShares 20+ Year Treasury Bond ETF |
| XLK | Technology Select Sector SPDR ETF |
| XLF | Financial Select Sector SPDR Fund |
| VXX | iPath S&P 500 VIX ST Futures ETN |
| EEM | iShares MSCI Emerging Markets ETF |

Table 1: Selected Exchange-traded fund (ETF) names for quote data test and their descriptions

method. Figure 15 shows the boxplot of the execution time for the portfolio optimization in each trading cycle. We refer the Algorithm 6 as PGDP (proximal gradient dynamic programming) as the first bar in the figure. We report that the median execution times for each methods are 0.6140s for PGDP, 0.8717s for quadprog, 1.6081s for MOSEK and 1.9766s for OSQP. PGDP has about 30% performance improvement than the best of benchmarks (quadprog) and it is more stable (notice that there are two outliers in quadprog).

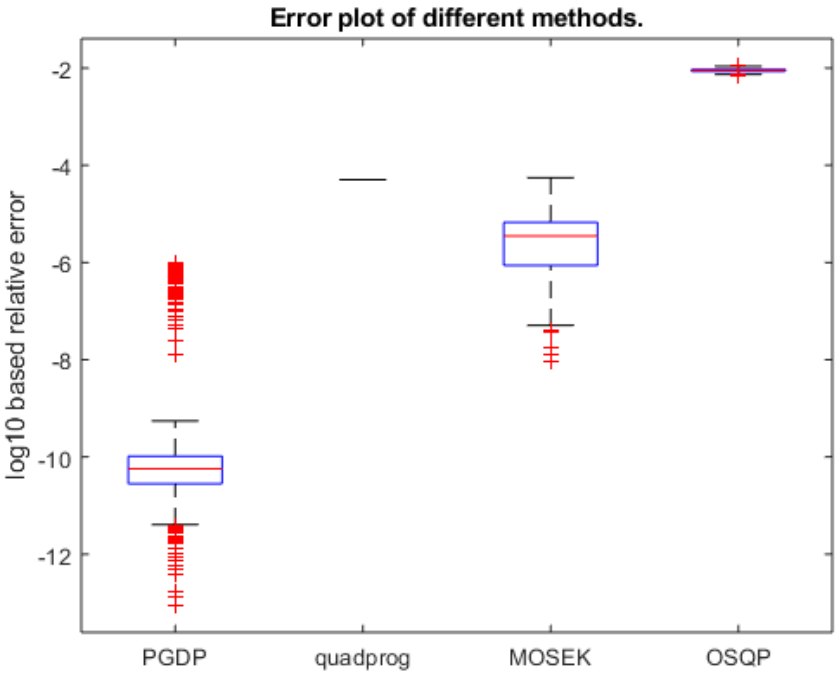


Figure 14: The boxplot of the relative error of portfolio optimization for each trading cycle.

# 6 A HESSIAN FREE SPLITTING SOLVER FOR REGULARIZED QP PROBLEM

In this chapter we develop a Hessian free solver for regularized quadratic programming (QP) problem

$$\begin{aligned} \underset{x}{\text{minimize}} \quad & h(x) + \mathcal{I}(x) \\ \text{subject to} \quad & h(x) = \frac{1}{2} x^T H x + f^T x \end{aligned} \tag{6.1}$$

using forward-backward splitting scheme in chapter 5. We start from the general discussion of the iteration formula of the solver. Then we discuss the key idea of this solver: the Barzilai-Borwein (BB) step size. As the examples, the nonnegative constrained QP, box constrained QP, general linear constrained QP and box constrained QP with L1 penalty are provided in the numerical experiments.

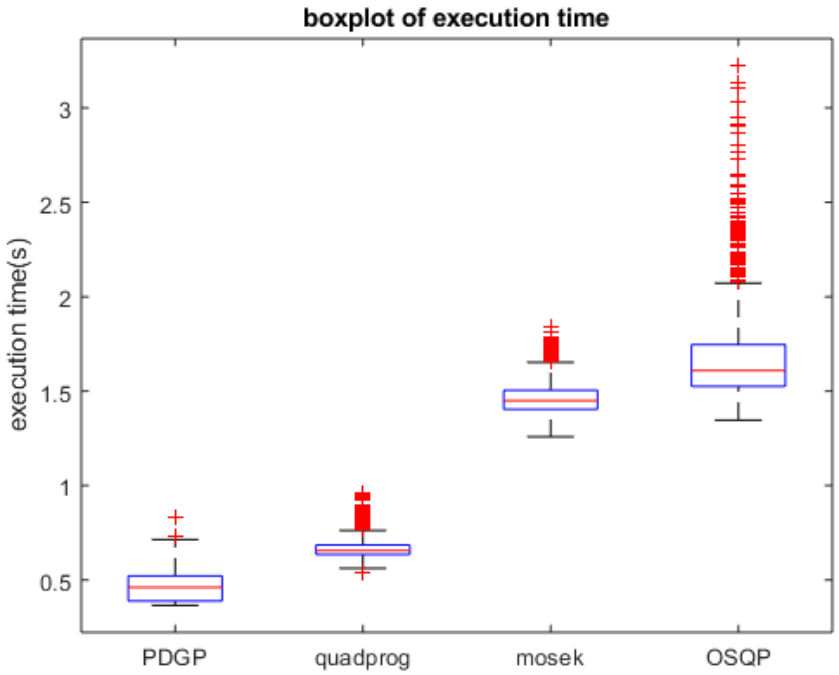


Figure 15: The boxplot of the execution time of portfolio optimization for each trading cycle.

## 6.1 The iteration formula

There is a natural splitting scheme for (6.1): the quadratic objective function part $h(x)$ and the regularization part $\mathcal{I}(x)$. Following the analysis of chapter 2, a fixed point iteration of (6.1) can be written as follows

$$x^{k+1} = x^k - \text{prox}_{\gamma\mathcal{I}}(x^k - \gamma\nabla h(x^k)) \tag{6.2}$$

where $\text{prox}_{\gamma\mathcal{I}}(\cdot)$ is the proximal mapping operator of $\mathcal{I}$. As a special case, when $\mathcal{I}$ is an indicator function, its proximal mapping operator is the projection operator

$$\text{prox}_{\gamma\mathcal{I}_{x\in\mathcal{C}}}(x) = \underset{u\in\mathcal{C}}{\text{argmin}}\|u - x\|_2^2 \tag{6.3}$$

In order to improve the convergence performance, one can use the semismooth newton method

$$\begin{aligned} G(x^k) &= x^k - \text{prox}_{\gamma\mathcal{I}}(x^k - \gamma\nabla h(x^k)) \\ x^{k+1} &= x^k - \partial_C G(x^k) \end{aligned} \tag{6.4}$$

where the Clarke generalized Jacobian (see chapter 2 for detail) $\partial_C G(x^k)$ is

$$\partial_C G(x^k) = I - (I - \gamma\nabla^2 h)\partial_C \text{prox}_{\gamma g}(I - \gamma\nabla^2 h) \tag{6.5}$$

One can keep iterating (6.4) with suitable step size until converge.

## 6.2 Quasi-Newton methods

There are two challenges for the iteration (6.4). One is that we have to find an approximation of Hessian matrix $H_k \in \partial_C G(x^k)$. However, in practice the proximal mapping is not trivial, therefore the generalized Jacobian (which is also the approximation of the Hessian) is hard to evaluate. The other one is numerical performance. Newton method requires solving a linear system

$$H_k d^k = -\nabla f(x_k) = -G(x_k) \tag{6.6}$$

to get a descent direction. In general, solving a linear system like (6.6) needs $O(n^3)$ floating point operations. One also compute Cholesky decomposition of the Hessian if it keeps the same in each iteration. Then solving (6.6) reduces to solve two triangular system which can be done by $O(n^2)$. But in the proximal gradient case the Hessian matrices in each iteration are not the same. Therefore we have to solve a new system in each iteration.

With these two major issues, we propose using quasi-Newton methods to get the descent direction in each period. Typically, we develop a quasi-Newton methods[25] using the multiple of identity matrix as the approximation matrix so that we can determine the step size parameter

without evaluating the proximal mapping.

Instead of solving (6.6) directly, quasi-Newton methods seek to find an approximation of the Hessian under some rules, and solve a new system

$$B_k d_k = -G(x_k) \tag{6.7}$$

with the approximated Hessian $B_k$. Perhaps the most famous quasi-Newton methods is BFGS and its limited memory variant L-BFGS. However, BFGS and L-BFGS are not favourable in the context with proximal mapping becauce they need to evaluate the proximal mapping several times in order to get a proper step size to converge. This needs a lot extra work since in general the evaluation of proximal mapping is equivalent with solving a quadratic programming problem except in some special cases. Invoke from the Barzilai-Borwein method[2] in the unconstrained quadratic programming problem, we propose using the multiple of identity as the approximation of true Hessian $H_k$.

**The method**. Recall that the Newton direction is

$$d_k = -H_k^{-1} G(x_k). \tag{6.8}$$

Let

$$\begin{aligned} s_k &= x_k - x_{k-1} \\ y_k &= G(x_k) - (x_{k-1}) \end{aligned} \tag{6.9}$$

Then we have

$$H_k s_{k-1} = y_{k-1} \tag{6.10}$$

Here we choose an multiple of identity matrix $\alpha_k^{-1} I$ so that

$$\alpha_k^{-1} I s_{k-1} \approx y_{k-1}. \tag{6.11}$$

One plausible choice is choosing $\alpha_k$ by minimizing the euclidean norm

$$\alpha_k = \underset{\alpha}{\operatorname{argmin}} \left\| \alpha^{-1} s_{k-1} \approx y_{k-1} \right\|^2 \tag{6.12}$$

which has the closed form solution

$$\alpha_k = \frac{s_{k-1}^T y_{k-1}}{y_{k-1}^T y_{k-1}} \tag{6.13}$$

Algorithm 7 gives the whole procedure of our quasi-Newton algorithms

**Algorithm 7** Quasi-Newton method for (6.1) with diagonal hessian (PGDH)

**Require:** $H,\ f,\ \mathcal{I},\ x_0,\ \epsilon$;
**Ensure:** $x$;
  **for** $k = 1, 2, ...$ **do**
    Set $G_1 = x_k - \text{prox}_{\gamma\mathcal{I}}(x_k - (\gamma H x_k + f))$;
    **if** $k > 1$ **then**
      Set $s_k = \alpha_{k-1} * d_{k-1}$;
      Set $y_k = G_1 - G_0$;
      Set $\alpha_k = \frac{s_k^T y_k}{y_k^T y_k}$;
      Set $x_k = x_{k-1} - \alpha G_1$;
    **else** $\{k = 1,\}$
      Set $\alpha_k = 1$;
      Set $x_k = x_0 - G_1$;
    **end if**
    Set $G_0 = G_1$;
    Set $k = k + 1$;
    **if** $\|\alpha_k G_1\| < \epsilon$ **then**
      **return** $\text{prox}_{\gamma\mathcal{I}}(x_k - (\gamma H x_k + f)$
    **end if**
  **end for**

which solves the regularized quadratic programming problem (6.1).

## 6.3 Numerical Experiments

In this section we compare the Algorithm 7 with other solvers. The first one is Matlab's built-in "quadprog" solver, it solves the standard quadratic programming problem using interior point methods. Another one is a commercial solver MOSEK, which formulates the quadratic programming problem as a cone programming problem. We also compare our algorithm with two known splitting method solver. One is FBS, the other one is OSQP.

The proposed test problems are largely based on the practical quadratic programming in portfolio selection area. The dimension of the problems ranges from 1000 to 3000 based on the number of trading universe. Structure Hessian matrix is another feature in this area since the sample covariance in the stock market is always ill-conditioned. Therefore we test the problem with both randomized covariance and a randomized factor model.

### 6.3.1 Long only portfolio

A long only portfolio seeks to find an optimal asset allocation without any short positions. With the mean-variance portfolio optimization structure, one can get an optimal long only portfolio by solving a standard quadratic programming problem with nonneagtive orthant constraints

$$\begin{array}{ll} \underset{u}{\text{minimize}} & \frac{1}{2}u^T\Sigma x - r^T u \\ \text{subject to} & u \geq 0 \end{array} \tag{6.14}$$

where $r$ is the return estimator, $\Sigma$ is the variance estimator. To fit (6.14) to our formulation, we rewrite the problem as

$$\begin{array}{ll} \underset{u}{\text{minimize}} & h(u) + g(u) \\ \text{subject to} & h(u) = \frac{1}{2}u^T\Sigma u - r^T u \\ & g(u) = \mathcal{I}_{u\geq 0}(u) \end{array} \tag{6.15}$$

From the analysis of proximal mapping operator in Chapter 2, we know

$$\text{prox}_{\gamma\mathcal{I}_{u\geq 0}}(u) = (u)_+, \tag{6.16}$$

where the nonnegative part operator $(\cdot)_+$ is taken elementwise. Thus, to get the evaluation of proximal mapping, we simply replace each negative component of $x$ with zero.

**Randomized covariance test**. We test the proposed algorithms and the benchmarks mentioned above in with 100 random long only portfolio selection problem with 1500 instruments. The covariance is generated

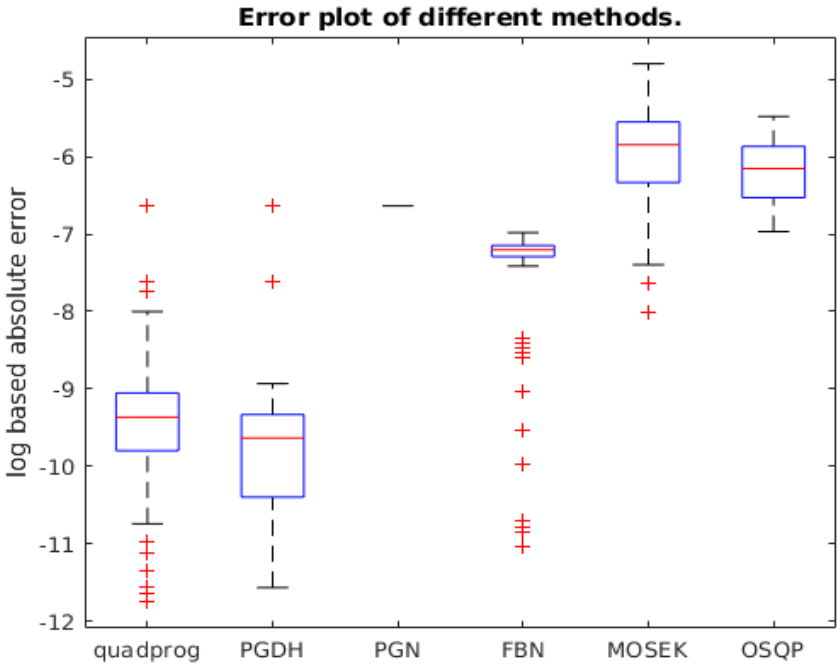


Figure 16: The box plot of the log based absolute error of different algorithms against the minimum objective function value. The test runs 100 random samples of portfolio selection problems with randomized covariance matrix of 1500 instruments.

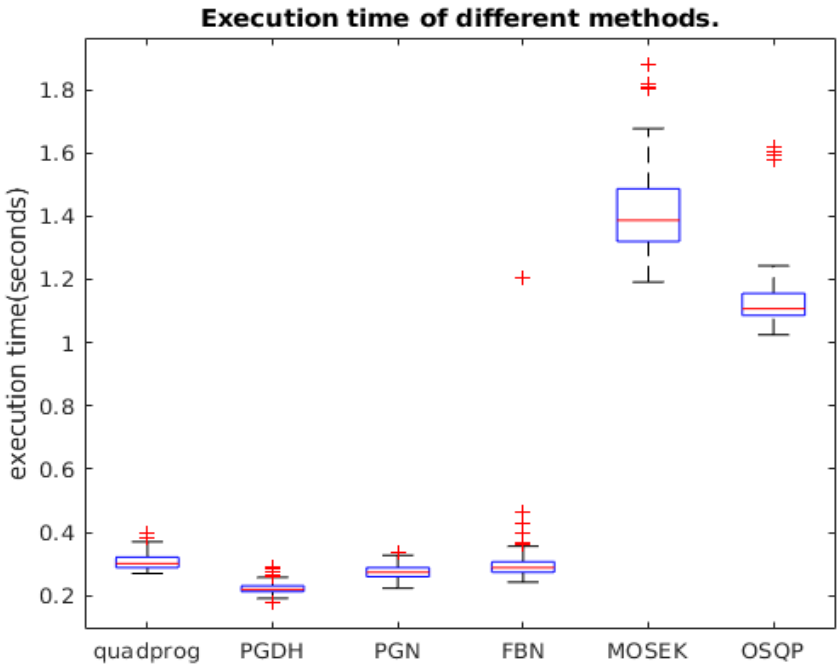


Figure 17: The box plot of the execution time of different algorithms. The test runs 100 random samples of portfolio selection problem with randomized covariance matrix of 1500 instruments.

with the following rule: we first generate a random n by n matrix $A$ with standard normal distribution for each entry, then we compute the Hessian by $H = AA^T$, which will enforce the Hessian be positive definite. We use both the objective function value and execution time as the performance measure. Figure 16 shows the boxplot of the relative differences between different algorithms and the one with lowest objective function value. From the plot we see the proposed algorithm has generally the lowest objective function value over 100 test. Figure 17 shows the boxplot of execution time for different algorithms over 100 tests. Still the proposed algorithm is generally faster than others.

**Randomized factor model test**. Instead of using a full sample covariance matrix, practitioners usually use a factor model specify the systematic risk (market risk) and the unsystematic risk (idiosyncratic risk). Here we use a randomized factor model with 20 factors, i.e.

$$\Sigma = D + VV^T \tag{6.17}$$

where $\Sigma \in \mathbb{R}^{n\times n}$ is the (shrunken) sample covariance matrix, and $D$ is a diagonal matrix representing the idiosyncratic risk, $V \in \mathbb{R}^{n\times 20}$ is a factor loading matrix representing the market risk.

We use the same environment as random covariance test except replacing the randomized covariance matrix by a randomized factor model. Figure 18 and 19 show the error plot of objective function value and the execution time of different methods. The proposed algorithms still have faster execution time and lower objective function values.

### 6.3.2 Long short portfolio with bound constraint

Long short strategies are widely used in active portfolio management, for example, the Hedge Funds. It has the potential to enhance returns from active security selection[46]. It can also reduce a lot risk than a long only portfolio since a long short portfolio has larger permissible portfolio space to search in general. Despite the huge advantages, the cost in an active portfolio management fund is considerable. Therefore

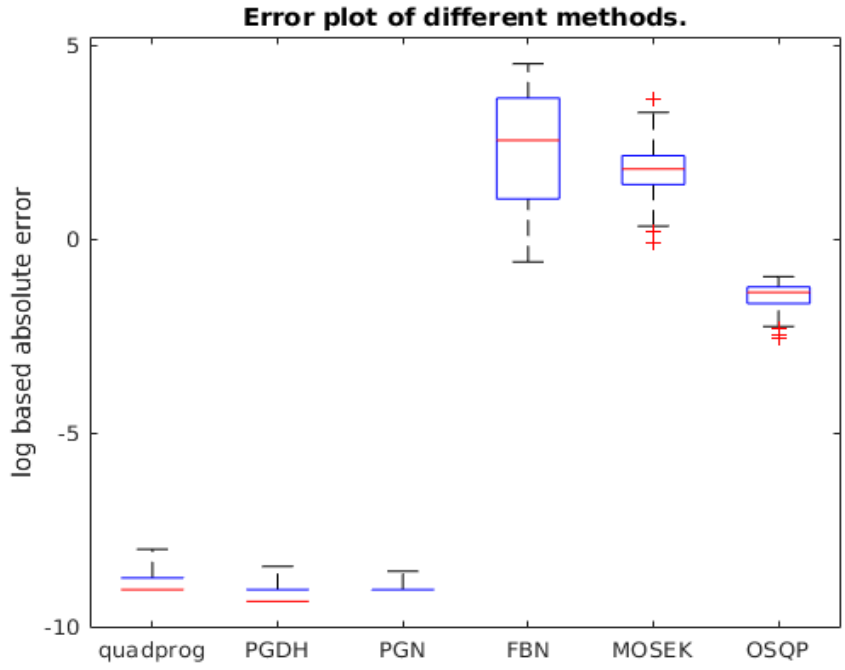


Figure 18: The box plot of the log based absolute error of different algorithms against the minimum objective function value. The test runs 100 random samples of portfolio selection problems with randomized factor model of 1500 instruments.

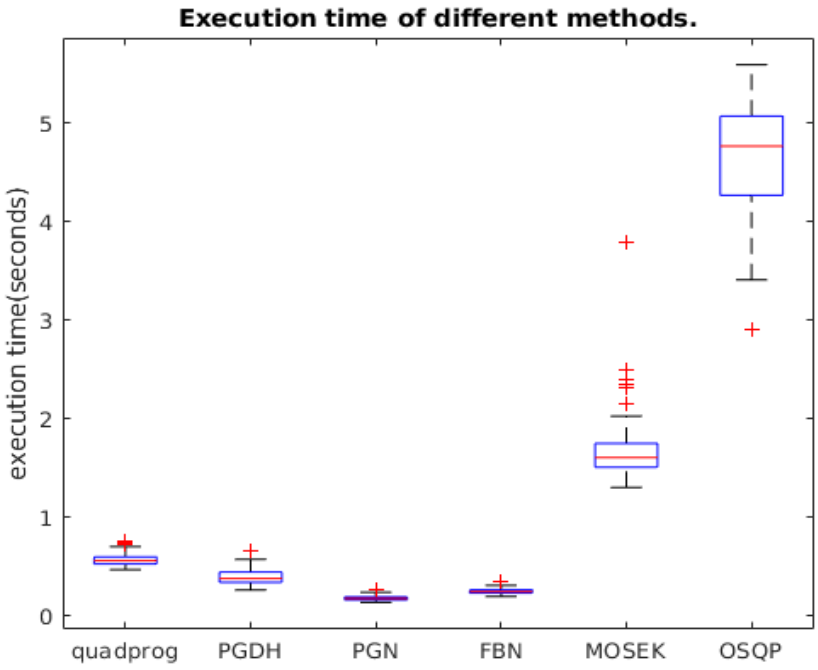


Figure 19: The box plot of the execution time of different algorithms. The test runs 100 random samples of portfolio selection problems with randomized factor model of 1500 instruments.

the objective function for an active management portfolio often adds the transaction cost penalty.

In this test, we form a mean variance portfolio selection problem with proportional transaction cost and portfolio bound constraints

$$\begin{array}{ll} \underset{u}{\text{minimize}} & \frac{1}{2}u^T\Sigma u - r^T u + \tau^T \|u\|_1 \\ \text{subject to} & lb \leq u \leq ub. \end{array} \tag{6.18}$$

where $r$ is the return estimator, $\Sigma$ is the variance estimator, $\tau$ is the proportional transaction cost estimator, $\|\cdot\|_1$ is the 1-norm which is the sum of absolute values, $lb$ is the lower position bound of each asset and $ub$ is the upper position bound for each asset. For this problem we use the following splitting scheme

$$\begin{array}{ll} \underset{u}{\text{minimize}} & h(u) + g(u) \\ \text{subject to} & h(u) = \frac{1}{2}u^T\Sigma u - r^T u \\ & g(u) = \tau^T \|u\|_1 + \mathcal{I}_{lb\leq u\leq ub}(u) \end{array} \tag{6.19}$$

Then the proximal mapping for $g(u)$ is

$$\text{prox}_{\gamma\mathcal{I}_{lb\leq u\leq ub}}(u) = \min(\max((u-\gamma\tau)_+ - (-u-\gamma\tau)_+, lb), ub), \tag{6.20}$$

**Random test with randomized covariance**. Similar with the long only portfolio test, we run 100 randomized long only portfolio selection problem with 1500 instruments in order to compare the performance of our proposed algorithm and other benchmarks. In this part we generate the Hessian by $H = AA^T$, where $A$ is an n by n random matrix. We still use both the objective function value and execution time as the performance measure.

Figure 20 shows the boxplot of the relative differences between different algorithms and the one with lowest objective function value. From the plot we see the proposed algorithm has generally the lowest objective function value over 100 test. Figure 21 shows the boxplot of

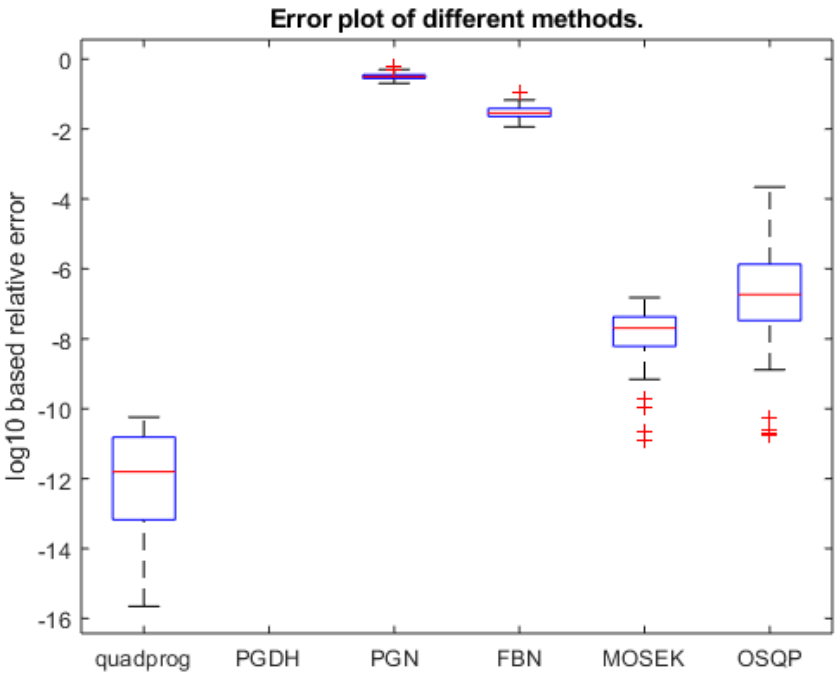


Figure 20: The box plot of the log based relative error of different algorithms against the minimum objective function value. The test runs 100 random samples of long short portfolio selection problems with 1500 instruments and random covariance matrix. In the test PGDH always has the lowest value so that its log10 based error is negative infinity.

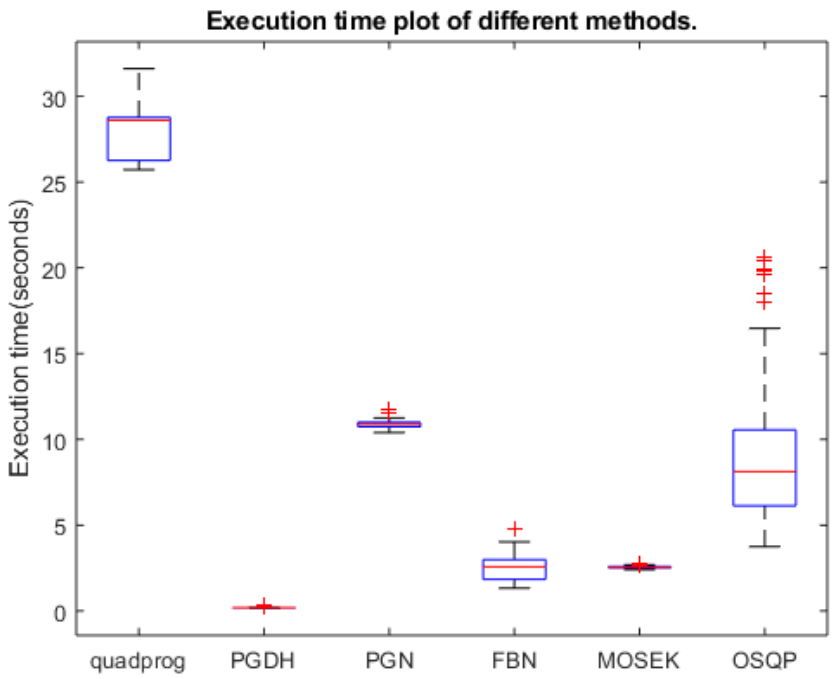


Figure 21: The box plot of the execution time of different algorithms. The test runs 100 random samples of portfolio selection problems with 1500 instruments and random covariance matrix.

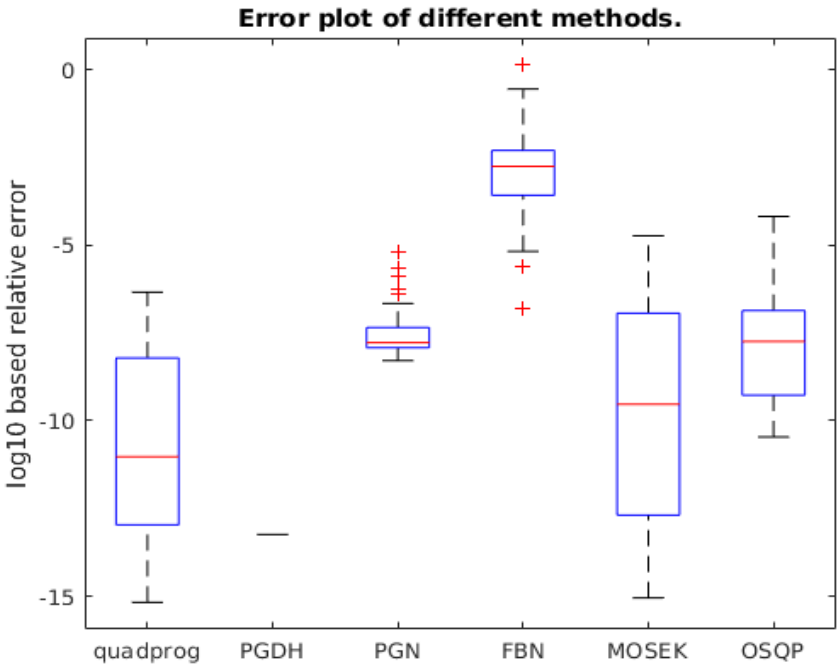


Figure 22: The box plot of the log based relative error of different algorithms against the minimum objective function value. The test runs 100 random samples of portfolio selection problems with 1500 instruments and factor model.In the test PGDH has the lowest value in the most of time so that its boxplot only has a 95% upper bar.

execution time for different algorithms over 100 tests. Still the proposed algorithm is generally faster than others.

**Random test with randomized factor model**. In this test we run 100 random long short portfolio selection problem with the same setting as above except that in this test we use factor model as the covariance estimator. We use 20 factors therefore the factor loading matrix $V$ is a matrix with size 1500 by 20.

Figure 22 and 23 show the relative error and the execution time result. The proposed method PGDH has both the lowest relative error and the fastest execution time. We also mention that the execution time of FBN has some outliers therefore it is not very robust from the result of our test.


Figure 23: The box plot of the execution time of different algorithms. The test runs 100 random samples of portfolio selection problems with 1500 instruments and factor model.

# 7 SPLITTING METHODS FOR SDES

The theory of stochastic differential equations (SDEs) is one of the most important area in mathematical finance. It has become the standard model for asset prices, interest rates, and their derivatives. Unlike the ODEs, SDEs have solutions that are continuous time stochastic processes. Further, most SDEs do not have closed form solutions. Therefore the numerical method plays an important role of solving SDEs.

There is a vast literature of strongly convergence schemes in the classical SDEs. Many of them are based on truncating the stochastic Taylor expansion [79, 44, 82, 43]. Others include stochastic Runge–Kutta schemes [72, 13], and Castell–Gaines schemes [16]. In this chapter, we develop a splitting scheme for the numerical approximation of the Stratonovich integral.

We start from the background of stochastic integral, then we introduce the Peaceman-Rachford scheme for the Stratonovich integral. We test the proposed scheme against the classical scheme on geometric brownian motion and Cox–Ingersoll–Ross model. Finally we show the application of Peaceman-Racheford scheme on unitary brownian motion.

Throughout this chapter, we denote $\Omega$ as the sample space, $\mathcal{F}$ as the $\sigma$-algebra on $\Omega$, and $\mathbb{P}$ as the probability measure. Therefore the triple $(\Omega, \mathcal{F}, \mathbb{P})$ is a probability space.

## 7.1 Stochastic differential equations and numerical schemes

In this section we review the most fundamental concepts from stochastic calculus that are needed to proceed with our description of numerical methods. We refer [50, 63, 73].

**Definition 7.1** (Brownian motion[73])**.** Let $(\Omega, \mathcal{F}, \mathbb{P})$ be a probability space, For each $\omega \in \Omega$, suppose there is a continuous function $W(t)$ of $t \geq 0$ that satisfies $W(0) = 0$ and that depends on $\omega$. Then $W(t)$, $t \geq 0$,

is a Borwnian motion if for all $0 = t_0 < t_1 < ... < t_m$ the increments

$$W(t_1) = W(t_1) - W(t_0), W(t_2) - W(t_1), ..., W(t_m) - W(t_{m-1}) \quad (7.1)$$

are independent and each of these increments is normally distributed with

$$\mathbb{E}[W(t_{i+1}) - W(t_i)] = 0, \mathrm{Var}[W(t_{i+1}) - W(t_i)] = t_{i+1} - t_i \quad (7.2)$$

A typical diffusion process in finance is modeled as a differential equation involving drift terms and diffusion terms, the latter represented by a brownian motion, as in the equation

$$dX = a(t, X)dt + b(t, X)dW_t \quad (7.3)$$

The solution of (7.3) is a stochastic process

$$X(t) = X(0) + \int_0^t a(t, X)dt + \int_0^t b(t, X)dW_t \quad (7.4)$$

We mention that the integral of diffusion part in (7.4) is so called Itô integral, which is a limit in probability of the following Riemann sums

$$\int_0^t HdW = \lim_{n\to\infty} \sum_{[t_{i-1},t_i]\in\pi_n} H_{t_{i-1}}(W_{t_i} - W_{t_{i-1}}). \quad (7.5)$$

Here $\pi_n$ is a sequence of partitions of $[0, t]$ with mesh going to zero. Instead of using (7.5), one can also use the mid-points as the approximation of Riemann sums

$$\int_0^t H \circ dW = \lim_{n\to\infty} \sum_{[t_{i-1},t_i]\in\pi_n} \frac{H_{t_{i-1}} + H_{t_i}}{2}(W_{t_i} - W_{t_{i-1}}). \quad (7.6)$$

(7.6) is called Stratonovich integral. The solution of Itô integral (7.4)

coincides with the following Stratonovich integral [78]

$$X(t) = X(0) + \int_0^t (a(t,X) - \frac{1}{2}b(t,X)\frac{\partial b}{\partial X}(t,X))dt + \int_0^t b(t,X) \circ dW_t \tag{7.7}$$

Therefore one can use different numerical schemes integrate (7.5) or (7.7) and get the same process.

Although there is a relationship between Itô integral and Stratonovich integral, the numerical schemes for integrating these two integrals are not the same. In the setting of Itô integral, one can use Euler–Maruyama (EM) method which is corresponding to the Euler step in ODEs

$$X_j = X_{j-1} + a(X_{j-1})\Delta t + b(X_{j-1})(W_j - W_{j-1}),\ j = 1, 2, ..., N \tag{7.8}$$

where $\Delta t$ is the time interval discretized by some positive integer $N$, and $(W_j - W_{j-1})$ is the increments of Brownian motion during this interval. Another numerical scheme for Itô integral is Milstein's method

$$\begin{aligned} X_j = X_{j-1} + a(X_{j-1})\Delta t + b(X_{j-1})(W_j - W_{j-1}) \\ + \frac{1}{2}b(X_{j-1})\frac{\partial b}{\partial X}(X_{j-1})((W_j - W_{j-1})^2 - \Delta t) \end{aligned} \tag{7.9}$$

It is well known that Euler–Maruyama converges with strong order 1/2 and weak order 1, while Milstein's method converges with strong order 1 and weak order 1.

For stratonovich integral, a classical scheme is Euler-Heun method

$$\begin{aligned} \tilde{X} &= X_{j-1} + a(X_{j-1})\Delta t + b(X_{j-1})(W_j - W_{j-1}) \\ X_j &= X_{j-1} + a(\frac{X_{j-1} + X_j}{2})\Delta t + b(\frac{X_{j-1} + X_j}{2})(W_j - W_{j-1}) \end{aligned} \tag{7.10}$$

Basically it does one step extrapolation on Euler-Maruyama step to get the approximation of next step.

## 7.2 Peaceman-Rachford splitting scheme for SDEs

There are obviously two processes in SDEs: the deterministic part and the stochastic part. Therefore, for (7.3) a natural spitting in the present case is

$$\begin{aligned} dX^* &= a(t, X)dt \\ dX^{**} &= b(t, X)dW_t \end{aligned} \tag{7.11}$$

Then one can apply the Peaceman-Rachford splitting scheme on (7.3), i.e. perform a forward Euler step on the deterministic part and a backward Euler step on the stochastic part, then switch the roles of two parts and do another Euler step update on them. The updating formula for (7.3) is shown as the following

$$\begin{aligned} X^1 &= X_{j-1} + a(X_{j-1})\frac{\Delta t}{2} \\ X^2 &= X^1 + b(X^2)\frac{W_j - W_{j-1}}{2} \\ X^3 &= X^2 + b(X^2)\frac{W_j - W_{j-1}}{2} \\ X_j &= X^3 + a(X_j)\frac{\Delta t}{2} \end{aligned} \tag{7.12}$$

## 7.3 Numerical experiment

In this section we compare the Peaceman-Rachford splitting scheme with other classical numerical schemes introduced in previous sections.

**Geometric Brownian motion**. Usually the dynamics of the underlying asset price is modeled by the geometric Brownian motion. Let $S(t)$ be the asset price at time t, then the model is of the form

$$dS(t) = \mu S(t)dt + \sigma S(t)dW(t) \tag{7.13}$$

where $W(t)$ is the Wiener process or standard Brownian motion, and $\mu$, $\sigma$ are the real parameters called "drift" and "volatility", respectively.

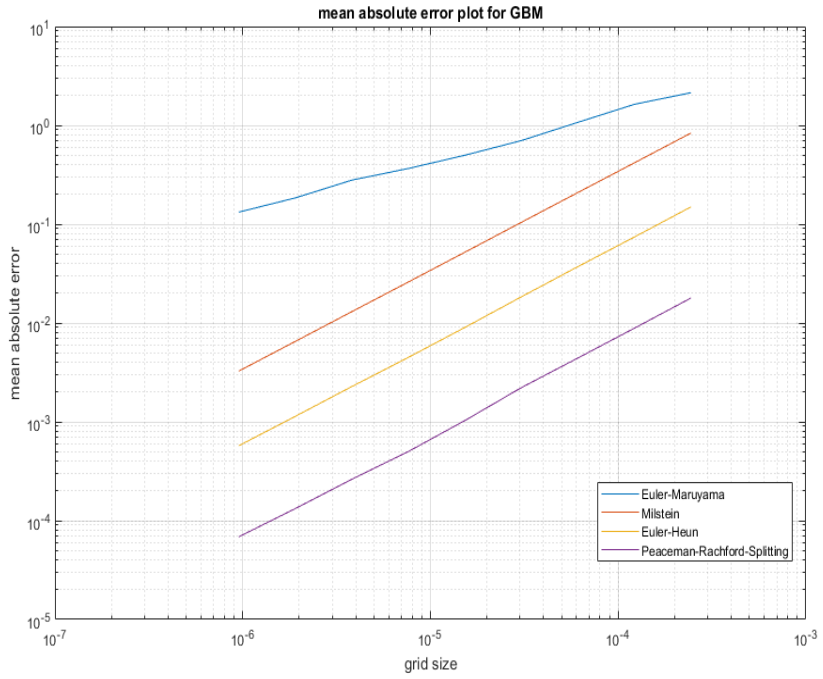


Figure 24: Strong convergence of different methods applied to the geometric brownian motion with parameters $\mu = 1$, $\sigma = 2$. The value of each data point is computed by averaging 500 samples.

The Peaceman-Rachford splitting scheme for (7.13) is

$$S_j = (1 - \mu\frac{\Delta t}{2})^{-1}(1 + \sigma\frac{W_j - W_{j-1}}{2})(1 - \sigma\frac{W_j - W_{j-1}}{2})^{-1}(1 + \mu\frac{\Delta t}{2})S_{j-1} \tag{7.14}$$

Figure 24 shows the strong convergence result for our proposed method and other classical methods under the average of 500 simulation paths. Clearly our proposed method (purple line) has significantly better performance than other methods. In fact, using the same size of discretization on time, our proposed method has one digit better accuracy than the best (Euler-Heun method, yellow line) of other methods. In order to achieve the same accurary, our proposed method only need 1/10 number of grid on time scale, which rouly means we only need 1/10 number of computations.

**Cox–Ingersoll–Ross (CIR) model**. CIR model is a stochastic model

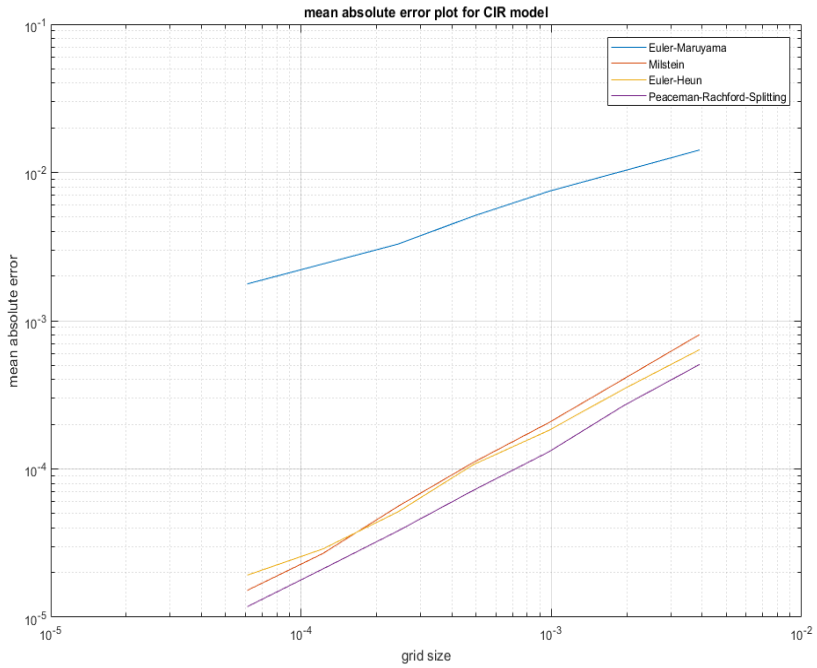


Figure 25: Strong convergence of different methods applied to the CIR model with parameters $a = 2$, $b = 1$ and $\sigma = 1$. The value of each data point is computed by averaging 500 samples.

widely used in interest rate modelling. It is of the form

$$dr_t = a(b - r_t)dt + \sigma\sqrt{r_t}dW_t \tag{7.15}$$

where $r_t$ is the interest rate at time $t$, a, b, and $\sigma$ are parameters, $W_t$ is the standard Brownian motion.

The Peaceman-Rachford splitting scheme for CIR model is

$$\begin{aligned} r^1 &= (1 + \frac{1}{2}a(b - X_{j-1})dt)r_{j-1} \\ r^2 &= r^1 + \frac{1}{2}\sigma\sqrt{r^2}dW_t \\ r^3 &= r^2 + \frac{1}{2}\sigma\sqrt{r^3}dW_t \\ r_j &= \frac{r^3 + \frac{ab}{2}dt}{1 + \frac{a}{2}dt} \end{aligned} \tag{7.16}$$

Figure 25 shows the the strong convergence result for our proposed

method and other classical methods under the average of 500 simulation paths. Still our proposed method (purple line), though in a small scale, outperforms other methods.

**Brownian motion on unitary group**. Recently stochastic differential equations on the group of unitary matrices U(N) have begun to find a variety of applications. For example, the interest rate model on Lie groups [65], robotics applications [14, 66] and quantum control and physics [23]. Here we test our proposed scheme against the classical numerical schemes on the brownian motion on unitary group $U(10)$, i.e. the brownian motion on the 10 by 10 unitary matrices. We focus on the numerical integral itself, and refer [28, 48, 47, 17] for the background.

For a Brownian motion on unitary group $U(N)$, the Itô equation is defined as

$$dU(t) = iU(t)dX(t) - \frac{1}{2}U(t)dt \tag{7.17}$$

where

$$X(t) = \frac{1}{2}[Z^N(t) + Z^N(t)^*] \tag{7.18}$$

and

$$[Z^N(t)]_{jk} = \frac{1}{\sqrt{N}}[W_{jk}(t) + iW'_{jk}(t)]. \tag{7.19}$$

Here $\{W_{jk}(t), W'_{jk}(t)\}_{1\leq j,k\leq N}$ are independent Brownian motions. The stratonovich equation for (7.17) is

$$dU(t) = U(t) \circ dX(t) \tag{7.20}$$

Since (7.20) does not have drift term, the Peaceman-Rachford splitting for (7.20) only contains the diffusion part

$$U_j = U_{j-1}(I - \frac{1}{2}(X_j - X_{j-1}))^{-1}(I + \frac{1}{2}(X_j - X_{j-1})) \tag{7.21}$$

Figure 26 illustrates the mean uniform squared convergence of Peaceman-Rachford splitting scheme and other classical schemes. It is clear that Peaceman-Rachford scheme drastically outperforms Euler-Maruyama for all step sizes considered. It also outperforms Euler-Heun

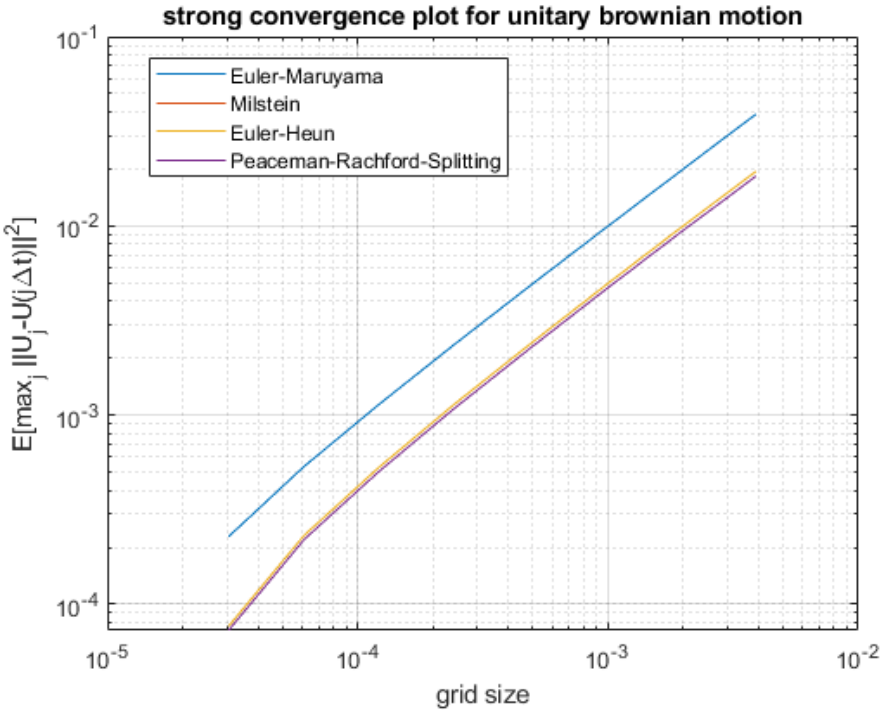


Figure 26: Convergence of the proposed method and other methods to the unitary brownian motion (7.17) and (7.20) with N = 20 in the metric $Y = \mathbb{E}[\max_j ||U_j - U(j\Delta t)||^2]$. The exact solution used in the error computation is based on Euler-Heun method with discretized time interval $\Delta t = 2^{-16}$, while the schemes tested have $\Delta t = 2^{-8}, 2^{-9}, ..., 2^{-15}$. In this test, the Milstein method (red line) coincides Euler-Heun method (yellow line)

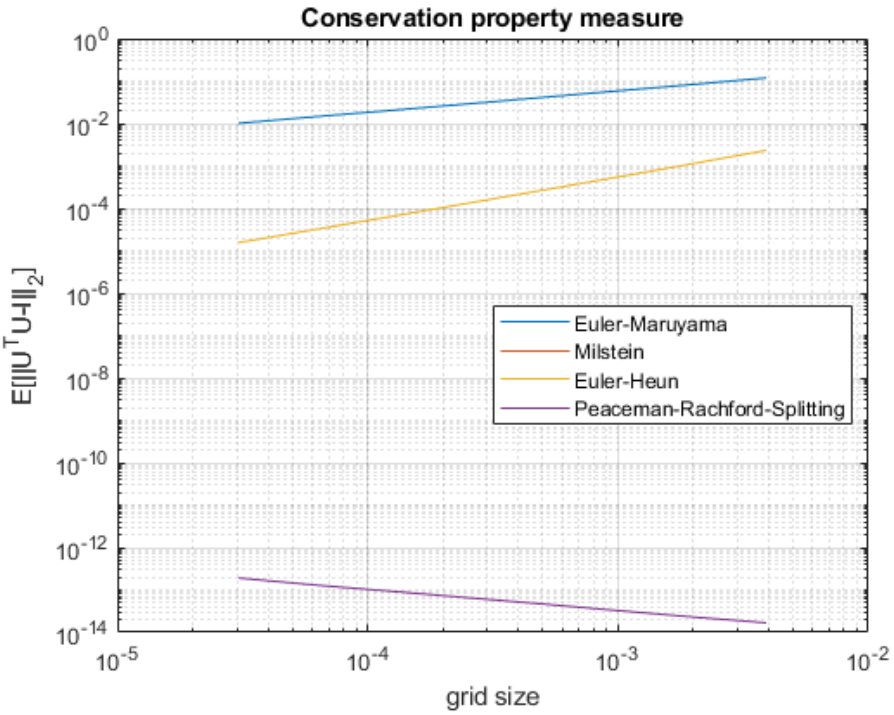


Figure 27: The conservation property measure of the proposed method and other methods to the unitary brownian motion (7.17) and (7.20) with N = 20. The Y axis is computed by $\mathbb{E}\left[\left\|U_j^T U_j - I\right\|_2\right]$. This measure has 0 value when $U_j$ is a unitary matrix. The schemes tested have $\Delta t = 2^{-8}, 2^{-9}, ..., 2^{-15}$. In this test, the Milstein method (red line) coincides Euler-Heun method (yellow line)

method and Milstein method though in a relatively small scale. Figure 27 gives the conservation property measure for all schemes. That is, the distance between the result coming from the numerical schemes and the unitary group $U(20)$. It turns out Peaceman-Rachford splitting scheme always gives a unitary matrix as the result, while the result of other schemes have noticeable distance from the unitary group $U(20)$ as a manifold.

# 8 Appendix

## Appendix A: Buy/sell split

The original optimization problem is

$$\begin{aligned}
\underset{w_1,w_2,\ldots,w_n}{minimize} \quad & f = \sum_{i=1}^{n}(\frac{1}{2}w_i^T C_i^2 w_i - r_i^T w_i + \tau_i^T |w_i - w_{i-1}| + \\
& (w_i - w_{i-1})^T \kappa_i (w_i - w_{i-1})) \\
\text{subject to} \quad & w_i - w_{i-1} \geq trdlb \\
& w_i - w_{i-1} \leq trdub \\
& w_i \geq poslb \\
& w_i \leq posub
\end{aligned}$$

One technique we can use in this problem is splitting every trade into a buy and a sell:

$$w_i - w_{i-1} = b_i - s_i$$

$$b_i \geq 0$$

$$s_i \geq 0$$

here we can assume a complementary condition:

$$b_i > 0 \Rightarrow s_i = 0$$

$$s_i > 0 \Rightarrow b_i = 0$$

$$\Rightarrow b^T s = 0$$

then each trade can be written as the following:

$$|w_i - w_{i-1}| = b_i - s_i$$

$$w_i = w_0 + \sum_{k=1}^{i}(b_i - s_i)$$

The objective function becomes:

$$
\begin{aligned}
f &= \sum_{i=1}^{n}(\frac{1}{2}w_i^T C_i^2 w_i - r_i^T w_i + \tau_i^T |w_i - w_{i-1}| + \\
&(w_i - w_{i-1})^T \kappa_i (w_i - w_{i-1})) \\
&= \sum_{i=1}^{n}(\frac{1}{2}(w_0 + \sum_{k<i}(b_k - sk))^T C_i^2 (w_0 + \sum_{k<i}(b_k - s_k)) - \\
&r_i^T (w_0 + \sum_{k<i}(b_k - sk)) + \tau_i^T (b_i + s_i) + (b_i + s_i)^T \kappa_i (b_i + s_i)) \\
&= \sum_{i=1}^{n}(\frac{1}{2}(\sum_{k<i}(b_k - s_k))^T C_i^2 (\sum_{k<i}(b_k - s_k)) + w_0 C_i^2 (\sum_{k<i}(b_k - s_k)) - \\
&r_i^T \sum_{k<i}(b_k - sk) + \tau_i^T (b_i + si) + (b_i + s_i)^T \kappa_i (b_i + s_i) + const.) \\
&= \frac{1}{2}(b-s)^T L^T C^2 L (b-s) + (e \otimes w_0)^T C^2 L (b-s) - r^T L (b-s) + \\
&\tau^T (b+s) + (b+s)^T \kappa (b+s) + const. \\
&= \frac{1}{2}\begin{bmatrix} b-s & b+s \end{bmatrix} \begin{bmatrix} L^T C^2 L & 0 \\ 0 & 2\kappa \end{bmatrix} \begin{bmatrix} b-s \\ b+s \end{bmatrix} + \\
&\begin{bmatrix} (e \otimes w_0) C^2 L - r^T L & \tau^T \end{bmatrix} \begin{bmatrix} b-s \\ b+s \end{bmatrix} + const.
\end{aligned}
$$

where $I$ is identity matrix, $\otimes$ is the Kronecker product, and

$$
\begin{aligned}
L &= L_0 \otimes I \\
&= \begin{bmatrix} 1 & 0 & 0 & ... & 0 & 0 \\ 1 & 1 & 0 & ... & 0 & 0 \\ 1 & 1 & 1 & ... & 0 & 0 \\ ... & ... & ... & ... & ... & ... \\ 1 & 1 & ... & ... & 1 & 0 \\ 1 & 1 & ... & ... & 1 & 1 \end{bmatrix} \otimes \begin{bmatrix} 1 & 0 & 0 & ... & 0 & 0 \\ 0 & 1 & 0 & ... & 0 & 0 \\ 0 & 0 & 1 & ... & 0 & 0 \\ ... & ... & ... & ... & ... & ... \\ 0 & 0 & ... & ... & 1 & 0 \\ 0 & 0 & ... & ... & 0 & 1 \end{bmatrix}
\end{aligned}
$$

Therefore, in the multiple-instrument multiple-period case, we rewrite:

$$
\begin{aligned}
D &= I \otimes C^2 \\
g &= \begin{bmatrix} (e \otimes w_0)^T C^2 L - r^T L & \tau^T \end{bmatrix}^T \\
&= \begin{bmatrix} \tau & L^T (r - C^2 (e \otimes w_0)) \end{bmatrix}
\end{aligned}
$$

The optimization problem becomes:

$$
\begin{aligned}
\underset{b,s}{minimize} \quad & f = \frac{1}{2} \begin{bmatrix} b & s \end{bmatrix} \begin{bmatrix} I & I \\ -I & I \end{bmatrix} \begin{bmatrix} L^T D L & 0 \\ 0 & 2\kappa \end{bmatrix} \begin{bmatrix} I & -I \\ I & I \end{bmatrix} \begin{bmatrix} b \\ s \end{bmatrix} \\
& + g^T \begin{bmatrix} b \\ s \end{bmatrix} \\
\text{subject to} \quad & b \geq 0 \\
& s \geq 0 \\
& b \leq trdub \\
& s \leq trdlb \\
& L(b - s) \leq posub - e \otimes w_0 \\
& L(b - s) \geq poslb - e \otimes w_0
\end{aligned}
$$

The constraints are equivalent to:

$$
\begin{bmatrix} -I & 0 \\ 0 & -I \\ I & 0 \\ 0 & I \\ L & -L \\ -L & L \end{bmatrix} \begin{bmatrix} b \\ s \end{bmatrix} \leq \begin{bmatrix} 0 \\ 0 \\ trdub \\ trdlb \\ posub - e \otimes w_0 \\ -(poslb - e \otimes w_0) \end{bmatrix}
$$

## Appendix B: quadratic programming with equality constraint

Consider the general linear equality constrained quadratic programming problem

$$\begin{aligned} \underset{x}{\text{minimize}} \quad & \frac{1}{2}x^T Hx + f^T x \\ \text{subject to} \quad & Ax = b \end{aligned} \tag{8.1}$$

where $x \in \mathbb{R}^n$, $H \in \mathbb{R}^{n\times n}$,$f \in \mathbb{R}^n$,$A \in \mathbb{R}^{m\times n}$,$b \in \mathbb{n}$. Notice that the problem (8.1) is not feasible when $m > n$, and it has an unique solution when $m = n$ (in this case one just need to solve the euqation $Ax = b$). Therefore in this problem we only consider $m < n$. In this part,we try to answer the following two questions: (i) Does the problem (8.1) has a closed form solution? (ii) Does the problem (8.1) has a gradient descent updating formula? These two questions are essentially the same since the objective function in the problem (8.1) is a smooth function.

The key idea is to get rid of the equality constraint $Ax = b$, which is equivalent to find the general solution of the linear system $Ax = b$. There are many ways to do this. Here we describe a practical way to do this. We first align the matrix $A^T$ with a random matrix $N \in \mathbb{R}^{n\times(n-m)}$, and denote the new matrix by $M$

$$M = \begin{bmatrix} A^T & N \end{bmatrix} \tag{8.2}$$

Then we take the *QR decomposition* of $M$, and split the unitary matrix $Q$ into two piece, the orthonormal basis $Q_A^T$ for the range of $A^T$ and the orthonormal basis $Q_N^T$ for the kernel of $A^T$. Then we have

$$AQ_N y = 0, \quad \forall y \in \mathbb{R}^{n-m} \tag{8.3}$$

Then we know the matrix $Q_N$ maps any vector $y \in \mathbb{R}^{n-m}$ to a vector $x \in \mathbb{R}^{\mathrm{n}}$ in the null space of $A$. Once we have the null space mapping $Q_N$, the problem (8.1) can be written as an unconstrained quadratic

programming problem

$$\underset{y}{\text{minimize}} \quad \frac{1}{2}(Q_N y + A^\dagger b)^T H(Q_N y + A^\dagger b)y + f^T(Q_N y + A^\dagger b) \tag{8.4}$$

where $A^\dagger$ is the Moore–Penrose inverse of $A$. The optimal solution $y^*$ of (8.4) is

$$y^* = -(Q_N^T H Q_N)^{-1} Q_N^T (H A^\dagger b + f) \tag{8.5}$$

and the solution for (8.1) is

$$x^* = Q_N y^* + A^\dagger b \tag{8.6}$$

# References


[1] MOSEK ApS. *The MOSEK optimization toolbox for MATLAB manual. Version 8.1.*, 2017.

[2] Jonathan Barzilai and Jonathan M Borwein. Two-point step size gradient methods. *IMA journal of numerical analysis*, 8(1):141–148, 1988.

[3] Heinz H Bauschke. *Projection algorithms and monotone operators*. PhD thesis, Theses (Dept. of Mathematics and Statistics)/Simon Fraser University, 1996.

[4] Heinz H Bauschke, Patrick L Combettes, et al. *Convex analysis and monotone operator theory in Hilbert spaces*, volume 408. Springer, 2011.

[5] Heinz H Bauschke and Valentin R Koch. Projection methods: Swiss army knives for solving feasibility and best approximation problems with halfspaces. *Contemp. Math*, 636:1–40, 2015.

[6] Amir Beck and Marc Teboulle. A fast iterative shrinkage-thresholding algorithm for linear inverse problems. *SIAM journal on imaging sciences*, 2(1):183–202, 2009.

[7] Nataliya Bershova and Dmitry Rakhlin. The non-linear market impact of large trades: Evidence from buy-side order flow. *Quantitative finance*, 13(11):1759–1778, 2013.

[8] Dimitri P Bertsekas and Athena Scientific. *Convex optimization algorithms*. Athena Scientific Belmont, 2015.

[9] Stephen Boyd, Enzo Busseti, Steve Diamond, Ronald N Kahn, Kwangmoo Koh, Peter Nystrup, Jan Speth, et al. Multi-period trading via convex optimization. *Foundations and Trends® in Optimization*, 3(1):1–76, 2017.

[10] Stephen Boyd, Neal Parikh, Eric Chu, Borja Peleato, Jonathan Eckstein, et al. Distributed optimization and statistical learning via the alternating direction method of multipliers. *Foundations and Trends® in Machine learning*, 3(1):1–122, 2011.

[11] Stephen Boyd and Lieven Vandenberghe. *Convex optimization*. Cambridge university press, 2004.

[12] William J Breen, Laurie Simon Hodrick, and Robert A Korajczyk. Predicting equity liquidity. *Management Science*, 48(4):470–483, 2002.

[13] Kevin Burrage and Pamela M Burrage. Order conditions of stochastic runge–kutta methods by b-series. *SIAM Journal on Numerical Analysis*, 38(5):1626–1646, 2000.

[14] Sachit Butail and Derek A Paley. Vision-based estimation of three-dimensional position and pose of multiple underwater vehicles. In *IROS*, pages 2477–2482, 2009.

[15] Jian-Feng Cai, Emmanuel J Candès, and Zuowei Shen. A singular value thresholding algorithm for matrix completion. *SIAM Journal on Optimization*, 20(4):1956–1982, 2010.

[16] Fabienne Castell and Jessica Gaines. The ordinary differential equation approach to asymptotically efficient schemes for solution of stochastic differential equations. In *ANNALES-INSTITUT HENRI POINCARE PROBABILITIES ET STATISTIQUES*, volume 32, pages 231–250. Gauthier-Villars, 1996.

[17] Guillaume Cébron and Todd Kemp. Fluctuations of brownian motions on gl_n. *arXiv preprint arXiv:1409.5624*, 2014.

[18] Scott Shaobing Chen, David L Donoho, and Michael A Saunders. Atomic decomposition by basis pursuit. *SIAM review*, 43(1):129–159, 2001.

[19] Giovanni Chierchia, Nelly Pustelnik, Jean-Christophe Pesquet, and Béatrice Pesquet-Popescu. Epigraphical projection and proximal tools for solving constrained convex optimization problems: Part i. *Signal, Image and Video Processing*, 2012.

[20] Patrick L Combettes and Jean-Christophe Pesquet. Proximal splitting methods in signal processing. In *Fixed-point algorithms for inverse problems in science and engineering*, pages 185–212. Springer, 2011.

[21] Ingrid Daubechies, Michel Defrise, and Christine De Mol. An iterative thresholding algorithm for linear inverse problems with a sparsity constraint. *Communications on Pure and Applied Mathematics: A Journal Issued by the Courant Institute of Mathematical Sciences*, 57(11):1413–1457, 2004.

[22] Damek Davis and Wotao Yin. Convergence rate analysis of several splitting schemes. In *Splitting Methods in Communication, Imaging, Science, and Engineering*, pages 115–163. Springer, 2016.

[23] François Delarue, Stephane Menozzi, Eulalia Nualart, et al. The landau equation for maxwellian molecules and the brownian motion on $so_N(\mathbb{R})$. *Electronic Journal of Probability*, 20, 2015.

[24] Arthur P Dempster. Covariance selection. *Biometrics*, pages 157–175, 1972.

[25] John E Dennis, Jr and Jorge J Moré. Quasi-newton methods, motivation and theory. *SIAM review*, 19(1):46–89, 1977.

[26] David L Donoho. De-noising by soft-thresholding. *IEEE transactions on information theory*, 41(3):613–627, 1995.

[27] Jim Douglas and Henry H Rachford. On the numerical solution of heat conduction problems in two and three space variables. *Transactions of the American mathematical Society*, 82(2):421–439, 1956.

[28] Bruce K Driver, Brian C Hall, and Todd Kemp. The large-n limit of the segal–bargmann transform on un. *Journal of Functional Analysis*, 265(11):2585–2644, 2013.

[29] Jonathan Eckstein. *Splitting methods for monotone operators with applications to parallel optimization*. PhD thesis, Massachusetts Institute of Technology, 1989.

[30] Jonathan Eckstein and Dimitri P Bertsekas. On the douglas—rachford splitting method and the proximal point algorithm for maximal monotone operators. *Mathematical Programming*, 55(1-3):293–318, 1992.

[31] Lawrence C Evans. An introduction to mathematical optimal control theory version 0.2. *Lecture notes available at http://math. berkeley. edu/˜ evans/control. course. pdf*, 1983.

[32] Francisco Facchinei and Jong-Shi Pang. *Finite-dimensional variational inequalities and complementarity problems*. Springer Science & Business Media, 2007.

[33] Nicolae Gârleanu and Lasse Heje Pedersen. Dynamic trading with predictable returns and transaction costs. *The Journal of Finance*, 68(6):2309–2340, 2013.

[34] Roland Glowinski, Stanley J Osher, and Wotao Yin. *Splitting Methods in Communication, Imaging, Science, and Engineering*. Springer, 2017.

[35] Tom Goldstein, Christoph Studer, and Richard Baraniuk. A field guide to forward-backward splitting with a fasta implementation. *arXiv preprint arXiv:1411.3406*, 2014.

[36] Carla Gomes and Henri Waelbroeck. Is market impact a measure of the information value of trades? market response to liquidity vs. informed metaorders. *Quantitative Finance*, 15(5):773–793, 2015.

[37] Richard Grinold. A dynamic model of portfolio management. *Journal of Investment Management*, 4(2):5, 2006.

[38] Richard C Grinold and Ronald N Kahn. *Active portfolio management*. McGraw Hill New York, 2000.

[39] SJ Grotzinger and C Witzgall. Projections onto order simplexes. *Applied mathematics and Optimization*, 12(1):247–270, 1984.

[40] Brian Hall. *Lie groups, Lie algebras, and representations: an elementary introduction*, volume 222. Springer, 2015.

[41] BS He, Hai Yang, and SL Wang. Alternating direction method with self-adaptive penalty parameters for monotone variational inequalities. *Journal of Optimization Theory and applications*, 106(2):337–356, 2000.

[42] John Heaton and Deborah J Lucas. Evaluating the effects of incomplete markets on risk sharing and asset pricing. *Journal of political Economy*, 104(3):443–487, 1996.

[43] Desmond J Higham. An algorithmic introduction to numerical simulation of stochastic differential equations. *SIAM review*, 43(3):525–546, 2001.

[44] Desmond J Higham, Xuerong Mao, and Andrew M Stuart. Strong convergence of euler-type methods for nonlinear stochastic differential equations. *SIAM Journal on Numerical Analysis*, 40(3):1041–1063, 2002.

[45] Peter J Huber et al. Robust estimation of a location parameter. *The annals of mathematical statistics*, 35(1):73–101, 1964.

[46] Bruce I Jacobs, Kenneth N Levy, and David Starer. Long-short portfolio management: An integrated approach. *Journal of Portfolio Management*, 25:23–32, 1999.

[47] Todd Kemp. The large-n limits of brownian motions on $\mathbb{GL}_N$. *International Mathematics Research Notices*, 2016(13):4012–4057, 2015.

[48] Todd Kemp. Heat kernel empirical laws on $\mathbb{U}_N$ and $\mathbb{GL}_N$. *Journal of Theoretical Probability*, 30(2):397–451, 2017.

[49] Seung-Jean Kim, Kwangmoo Koh, Stephen Boyd, and Dimitry Gorinevsky. $l\_1$ trend filtering. *SIAM review*, 51(2):339–360, 2009.

[50] Fima C Klebaner. *Introduction to stochastic calculus with applications*. World Scientific Publishing Company, 2012.

[51] Chi Kong. *Information Geometry and Dimensionality Reduction*. PhD thesis, Stonybrook University, 2018. Copyright - Database copyright ProQuest LLC; ProQuest does not claim copyright in the individual underlying works; Last updated - 2018-08-09.

[52] Olivier Ledoit and Michael Wolf. Honey, i shrunk the sample covariance matrix. *UPF Economics and Business Working Paper No. 691*, 2003.

[53] Ren-Cang Li. *Raising the orders of unconventional schemes for ordinary differential equations*. PhD thesis, University of California, Berkeley, 1995.

[54] Fabrizio Lillo, J Doyne Farmer, and Rosario N Mantegna. Econophysics: Master curve for price-impact function. *Nature*, 421(6919):129, 2003.

[55] Pierre-Louis Lions and Bertrand Mercier. Splitting algorithms for the sum of two nonlinear operators. *SIAM Journal on Numerical Analysis*, 16(6):964–979, 1979.

[56] Nicolai Meinshausen, Peter Bühlmann, et al. High-dimensional graphs and variable selection with the lasso. *The annals of statistics*, 34(3):1436–1462, 2006.

[57] Jean-Jacques Moreau. Proximité et dualité dans un espace hilbertien. *Bull. Soc. Math. France*, 93(2):273–299, 1965.

[58] AB Németh and SZ Németh. How to project onto an isotone projection cone. *Linear Algebra and its Applications*, 433(1):41–51, 2010.

[59] Yu Nesterov. Gradient methods for minimizing composite functions. *Mathematical Programming*, 140(1):125–161, 2013.

[60] Yurii E Nesterov. A method for solving the convex programming problem with convergence rate o (1/kˆ 2). In *Dokl. Akad. Nauk SSSR*, volume 269, pages 543–547, 1983.

[61] Jorge Nocedal and Stephen J Wright. Numerical optimization 2nd, 2006.

[62] Henrik Ohlsson, Lennart Ljung, and Stephen Boyd. Segmentation of arx-models using sum-of-norms regularization. *Automatica*, 46(6):1107–1111, 2010.

[63] Bernt Øksendal. Stochastic differential equations. In *Stochastic differential equations*, pages 65–84. Springer, 2003.

[64] Neal Parikh, Stephen Boyd, et al. Proximal algorithms. *Foundations and Trends® in Optimization*, 1(3):127–239, 2014.

[65] Frank Chongwoo Park, CM Chun, CW Han, and Nick Webber. Interest rate models on lie groups. *Quantitative Finance*, 11(4):559–572, 2011.

[66] Wooram Park, Jin Seob Kim, Yu Zhou, Noah J Cowan, Allison M Okamura, and Gregory S Chirikjian. Diffusion-based motion planning for a nonholonomic flexible needle model. In *Robotics and Automation, 2005. ICRA 2005. Proceedings of the 2005 IEEE International Conference on*, pages 4600–4605. IEEE, 2005.

[67] Panagiotis Patrinos, Lorenzo Stella, and Alberto Bemporad. Forward-backward truncated newton methods for convex composite optimization. *arXiv preprint arXiv:1402.6655*, 2014.

[68] Donald W Peaceman and Henry H Rachford, Jr. The numerical solution of parabolic and elliptic differential equations. *Journal of the Society for industrial and Applied Mathematics*, 3(1):28–41, 1955.

[69] Ariadna Quattoni, Xavier Carreras, Michael Collins, and Trevor Darrell. An efficient projection for $l_{1,\infty}$ regularization. In *Proceedings of the 26th Annual International Conference on Machine Learning*, pages 857–864. ACM, 2009.

[70] R Tyrrell Rockafellar. Monotone operators and the proximal point algorithm. *SIAM journal on control and optimization*, 14(5):877–898, 1976.

[71] Ralph Tyrell Rockafellar. *Convex analysis*. Princeton university press, 2015.

[72] Andreas Rößler. Runge–kutta methods for the strong approximation of solutions of stochastic differential equations. *SIAM Journal on Numerical Analysis*, 48(3):922–952, 2010.

[73] Steven Shreve. *Stochastic calculus for finance I: the binomial asset pricing model*. Springer Science & Business Media, 2012.

[74] Suvrit Sra. Fast projections onto $l_{1,q}$-norm balls for grouped feature selection. In *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*, pages 305–317. Springer, 2011.

[75] Lorenzo Stella. *Proximal envelopes: Smooth optimization algorithms for nonsmooth problems*. PhD thesis, IMT School for Advanced Studies Lucca, 2017.

[76] B. Stellato, G. Banjac, P. Goulart, A. Bemporad, and S. Boyd. OSQP: An operator splitting solver for quadratic programs. *ArXiv e-prints*, November 2017.

[77] Gilbert Strang. On the construction and comparison of difference schemes. *SIAM Journal on Numerical Analysis*, 5(3):506–517, 1968.

[78] RL Stratonovich. A new representation for stochastic integrals and equations. *SIAM Journal on Control*, 4(2):362–371, 1966.

[79] Denis Talay. Numerical solution of stochastic differential equations. *Stochastics and Stochastic Reports*, 1994.

[80] Robert Tibshirani. Regression shrinkage and selection via the lasso. *Journal of the Royal Statistical Society. Series B (Methodological)*, pages 267–288, 1996.

[81] L Vandenberghe. Lecture on proximal gradient method, 2010.

[82] W Wagner. On a taylor formula for a class of ito processes. *Proba. and Math. Statist.*, 3:37–51, 1982.

[83] Xiao Yu. *Identification and Model Reduction of MIMO systems in Triangular Input Balanced Form*. PhD thesis, The Graduate School, Stony Brook University: Stony Brook, NY., 2014.

[84] Ming Yuan and Yi Lin. Model selection and estimation in regression with grouped variables. *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, 68(1):49–67, 2006.

[85] Eduardo H Zarantonello. Projections on convex sets in hilbert space and spectral theory: Part i. projections on convex sets: Part ii. spectral theory. In *Contributions to nonlinear functional analysis*, pages 237–424. Elsevier, 1971.

[86] Peng Zhao, Guilherme Rocha, Bin Yu, et al. The composite absolute penalties family for grouped and hierarchical variable selection. *The Annals of Statistics*, 37(6A):3468–3497, 2009.