SIMULATION APPROACH TO TWO-STAGE BOND PORTFOLIO OPTIMIZATION PROBLEM

BY CHUAN XU

A thesis submitted to the

Graduate School—New Brunswick

Rutgers, The State University of New Jersey

in partial fulfillment of the requirements

for the degree of

Master of Science

Graduate Program in Operations Research

Written under the direction of

DR. ANDRÁS PRÉKOPA

and approved by

_____

_____

_____

New Brunswick, New Jersey

May, 2014

ABSTRACT OF THE THESIS

SIMULATION APPROACH TO TWO-STAGE BOND PORTFOLIO OPTIMIZATION PROBLEM

by CHUAN XU

Thesis Director: DR. ANDRÁS PRÉKOPA

Studies on two sides are done in this thesis. First, we consider bond portfolio optimization problem under stochastic optimization structure; second, specific algorithm to solve the problem is explored. A stochastic model for the problem is constructed. Investor is able to minimize the cost of setting up bond portfolio to cover random obligations with our model. The idea of rebalancing is introduced into our model. Investor could adjust the portfolio after he have set up the bond portfolio. Thus, we develop a two-stage stochastic programming with recourse model for bond optimization problem. Specific algorithms to solve the problem are also discussed in the thesis. We focus on simulation approach since it is able to handle special case of the problem whose random variables in constraints have continuous distribution. The key points of the approach are introduced and discussed.

We successfully implement the approach on our model. Various numerical example tests with different scenario settings are carried out to see the impacts of different factors on the optimum value, optimum solution and the quality of results. The validity of our model and the efficiency of simulation approach are proved by the results. Several future research directions on this topic are also discussed in the thesis.

Acknowledgements

I would like to express my deep gratitude to Dr. András Prékopa, my thesis advisers, for his contribution, guidance, and invaluable critiques of this research. Without his inspiration, it's impossible for me to figure out the model and find the right direction to solve the problem.

I would also like to extend my special thanks Jin Xin and Bin Zhao for their help in offering resources and support.

Finally, I wish to thank my family and my girlfriend Jie Shen, for the unwavering support, encouragement and belief in me throughout the course of my study.

Table of Contents

List of Tables

List of Figures

Chapter 1

Introduction

Based on the definition of Wikipedia, a bond is an instrument of indebtedness of the bond issuer to the holders. Bond has many good features we can explore, for instance, it can provide cash flows periodically to holders based on its coupon rate before maturity date and return the principal to holders eventually. Normally, we think bond is a kind of very safe investment instrument. Thus it's a very popular investment instruments in insurance industry. Based on Capital Special Market [1], by the end of 2010, bonds have the biggest weight in the investment portfolio of U.S. insurance companies. That's about 70%. The second one is common stock which has about 10% weight. Based on these data, we can see how important bonds are in insurance industry. Countless studies about quantitative management approaches on bonds have been carried out by numerous scholars, such as Lev Dynkin and Frank J. Fabozzi. Those researches focused on portfolio active management, index replicating, liquidation management and so on. Part of those results can be found in [5], [8] and [7].

Beside those interesting topics, one special strategy, Exact Matching or Dedication, was developed due to the special properties of bonds. More introductions about the strategy can be find in page 533 to page 537 of [6]. The idea behind the strategy is very simple: find out a bond portfolio which can provide certain pattern of cash flows.

[1] http://www.naic.org/capital_markets_archive/110819.htm

Since it's nature for us to try to minimize the cost for such pattern, we successfully transform the problem into a optimization problem. If all the information are known and deterministic, this is only a simple linear or non-linear programming problem which we can handle easily. But when random factors are involved, the problem become quite complex. Moreover, since the random nature of our world, investors would always keep adjusting their portfolio based on new information to correct the errors which caused by differences between predictions and facts. Thus, a two-stage stochastic programming model for the problem is worth to discuss. Dr. Prékopa proposed a stochastic programming model for bond portfolio management problem in his Stochastic Programming lecture notes. The way to solve two-stage stochastic programming model with discrete distribution can be found in [19]. But it's nature to think what will happen when those random variables have continuous distribution. Thus, we have two purposes in this thesis: First, we try to develop a two-stage stochastic programming model for bond portfolio optimization problem; second, we try to explore the approaches to solve the model when the random variables has continuous distribution.

This thesis make two primary contribution: first, we develop a two-stage stochastic programming model for bond portfolio optimization problem; second, we successfully implement the simulation approach on the model and give out numerical examples. The simulation algorithm is proved to be efficient on the model. Moreover, the numerical results also prove the validity of our model.

This thesis is organized as following. Chapter 2 is literature review. Chapter 3 introduces mathematical formulation of bond optimization problem with two-stage stochastic programming. Chapter 3 gives details about the simulation approach. Chapter 5 shows the settings of different scenarios and their results. Some future research directions on the topic are discussed in Chapter 6. The full MATLAB codes of simulation approach to our model can be found in Appendices.

Chapter 2

Literature Review

As I mentioned before, quantitative management of bond portfolio has been proposed and discussed for a long time. For instance, in [5], bond selection, asset allocation, index replicating, risk management and liquidation management are discussed. Multiple types of bonds are involved in their discussion. Frank J. Fabozzi is involved several books on bond portfolio management, such as [8] and [7]. Similar topics as in [5] are covered in these books.

Operations research methodologies were applied into the bond portfolio management long time ago. In [1], a dynamic model of bond portfolio management was proposed. The bond portfolio management problem was taken as a multistage decision problem in which actions were taken in the successive points. In [13], a linear programming model was proved to be able to improve the bond portfolio performance efficiently. Based on these results, Ronn further developed the linear programming model of bond portfolio management. He considered trading cost and tax-clientle effects with the model in [22]. He pointed out that the model was able to show investors' yield curve.

Optimization of the system under uncertainty can be traced back to fifties, such as the [4] wrote by George B. Dantzig. But not until eighties, with the development of computer power, this topic got more discussions. Dr. Prékopa introduced various

topics on stochastic programming in his book [19]. Specifically, he showed how to use decomposition algorithm to solve the two-stage stochastic programming problems in the book. Frauendorfer published a special book [9] for stochastic two-stage programming problems in 1992. This book included from the basic analysis of stochastic two-stage programming problems to the special techniques to solve the problems.

Talking about the idea to use simulation approach to solve stochastic programming problems, it's hard to trace back the origin. Rubinstein and Shapiro discussed the stochastic counterparts algorithm in [23]. The score function method, or likelihood ratio method, was able to handle the discrete event dynamic systems with a single simulation experiment. In [24], Shapiro provided the convergence analysis for gradient decent algorithms. In [17], a method related to retrospective simulation optimization was developed by Plambeck, Fu, Suri and Robinson. The method overcame some limitations of stochastic approximation but was still under the structure of discrete systems. More recent study on this topic can see [18]. Polson and Sorensen developed a simulation-based approach to estimate the Q-values to solve the Bellman equation. This method works in both continuous and discrete state.

A lot of literatures focused on investigations of stochastic programs with recourse. [27], the corresponding deterministic problem of stochastic programs with fixed recourse was investigated. The solvability, stability and dualizability of the equivalent deterministic program was reviewed in this paper. Higle and Sen developed methods to test the optimality of solutions for two-stage stochastic programming with recourse in [12]. The work was carried out under the framework of Stochastic Decomposition algorithm. The stopping criterion of such algorithm which satisfied KKT conditions was proposed in the paper. In [21], quantitative continuity of optimal solution sets to convex stochastic programs with complete recourse was investigated. A general sufficient conditions for the crucial strong-convexity assumption was given and verified.

Sample-path optimization method, or sample average approximation method, is based

on Monte Carlo ideas. A brief introduction of this method was given in [11]. More works on how to apply this method on two-stage stochastic programs with recourse problems can be found in Shapiro's and his students Tito Hemem-de-Mello's work. In [14] and [25], they analysed the basic principal of this method, gave out details of algorithm and showed several numerical examples. Part of my work in the thesis is just built on these results.

The discussions about distribution of insurance aggregate claim size during some time intervals can be found in many literatures. For instance, in [16], Pai, Shand and Wang tested the aggregate claim size of Canadian pet insurance with Compound Poison distribution with covariates. They tested the model with eight-year period data. Based on the results, the model works well. In [2], the authors investigated the distribution of property loss insurance claim indices and found that lognormal distribution is better than Paretian.

Chapter 3

Mathematical Formulation

3.1 Generic Two-Stage Stochastic Programming Model

Let x and y designate the first and second stage decision vectors, and let ξ be the random vector to be observed. Assume x and ξ are fixed, then we can formulate the second stage problem or recourse problem as follows:

$$\begin{aligned} &\text{Minimize} \qquad q^T y \\ &\text{subject to:} \\ &Tx + Wy = \xi \\ &y \geq 0 \end{aligned} \tag{3.1}$$

Assume that the first stage decision vector x has some deterministic constraints:

$$Ax = b, \quad x \geq 0 \tag{3.2}$$

let $q(x, \xi)$ designate the optimum value of (3.1), then $Q(x) = E[q(x, \xi)]$ is called ***recourse function***. Thus we can formulate the first stage problem as follows:

$$
\begin{aligned}
&\text{Minimize} \qquad c^T x + Q(x) \\
&\text{subject to:} \\
&Ax = b \\
&x \geq 0 \\
&x \in S
\end{aligned}
\tag{3.3}
$$

S is the set of x when problem (3.1) has feasible solutions.

3.2 Bond Portfolio Optimization Problem

Let me repeat the bond portfolio optimization problem briefly: How to minimize the cost to set up a bond portfolio which is used to cover some obligations.

First, let me introduce some notations we will use in the formulation:

p_{ij}	the price of bond i at time j
B_{ij}	the purchase amount of bond i at time j
S_{ij}	the selling amount of bond i at time j
a_{ij}	cash flow generated by bond i at time j
Z_j	the cash surplus we transferred from time j to $j+1$
r_j	the risk-free reinvestment rate from time j to $j+1$
ξ_j	the random aggregate obligation size from time $j+1$ to $j+2$
b_j	the deterministic aggregate obligation size from time $j+1$ to $j+2$
N	the number of candidate bond
T	the number of time interval
t^*	the time to do the portfolio adjustment

Table 3.1: Notations Introduction

3.2.1 Deterministic Model

We have several assumptions for the deterministic model: the obligations have no random properties; the portfolio will not be rebalanced since it has been set up; all transaction costs can be neglected; the reinvestment rate during the whole time horizon is known; all cash flows generated by bonds can be collected immediately. Then we have the deterministic bond portfolio model as follows:

$$
\begin{aligned}
&\text{Minimize} \qquad \sum_{i=1}^{N} P_{i1} B_{i1} + Z_1 \\
&\text{subject to:} \\
&\sum_{i=1}^{N} a_{ij} B_{i1} + Z_j(1 + r_j) - Z_{j+1} = b_j, \quad j = 1, \ldots, T-1 \\
&\sum_{i=1}^{N} a_{ij} B_{i1} + Z_j(1 + r_j) = b_j, \quad j = T \\
&B_{i1} \geq 0, \quad i = 1, \ldots, N \\
&Z_j \geq 0, \quad j = 1, \ldots, T
\end{aligned}
\tag{3.4}
$$

In the formulation, we minimize the cost to set up the bond portfolio through the objective function. In a real-world bond portfolio, it is usually constructed by two parts: bond and cash. $\sum_{i=1}^{N} P_{i1} B_{i1}$ represents the bond part, Z_1 is the cash part. The constraints guarantee that the obligations are satisfied by the cash flows generated by those bonds we purchased when we set up the portfolio and the cash we invested on the risk-free reinvestment rate. At the last period, there is no need to move any cash forward to the next period, so the constraint is a little different. All variables are positive due to the nature of our finance world.

The model was first proposed in [13] by S. D. Hodges and S. M. Schaefer. They proved the model could reduce the cost to set up bond portfolio with same cash flow pattern.

Ehud I. Ronn developed this model in [22] by introducing bid price and ask price of bonds into the model. He also discussed tax-clientele effects on the pricing of U.S. government bonds and derived tax-specific bond portfolio with the model. This model is a deterministic model. The problem can be solved easily with linear programming algorithm if we have all the necessary information.

3.2.2 Two-Stage Stochastic Programming Model

Let me change some of the assumptions in 3.2.1: the obligations are random events; we will rebalance the portfolio after setting up it.

It's worth to discuss briefly why the two assumptions are involved. Most events in our real world are random, for instance, the aggregate insurance claim size in one year follows Compound Poisson distribution(see [16]). And bond is very populous in insurance industry. Besides, people usually have better short-term forecasting ability than long term. Thus using new information between two stages to update the model may give the portfolio better performance.

In the model, the reinvestment rate is critical. We can also use stochastic model to describe the behaviours of reinvestment rate. It has been discussed a lot(see e.g., [3], [10]). This has been beyond our topic. So in our discussion, we will make some assumptions on it for simplicity.

Thus, following the generic model in 3.1, the second stage of bond portfolio problem can be formulated as follows:

$$\text{Minimize} \qquad \sum_{i=1}^{N} P_{it^*} B_{it^*} - \sum_{i=1}^{N} P_{it^*} S_{it^*} - Z_{t^*}$$

subject to:

$$\begin{aligned}
&\sum_{i=1}^{N} a_{ij}(B_{i1} + B_{it^*} - S_{it^*}) + Z_j(1 + r_j) - Z_{j+1} = \xi_j, \quad j = t^*, \ldots, T-1 \\
&\sum_{i=1}^{N} a_{ij}(B_{i1} + B_{it^*} - S_{it^*}) + Z_j(1 + r_j) = \xi_j, \quad j = T \qquad (3.5) \\
&B_{i1} + B_{it^*} \geq S_{it^*}, \quad i = 1, \ldots, N \\
&B_{it^*} \geq 0, \quad i = 1, \ldots, N \\
&S_{it^*} \geq 0, \quad i = 1, \ldots, N \\
&Z_j \geq 0, \quad j = t^* + 1, \ldots, T
\end{aligned}$$

The first thing we should notice is that the random variables on the Right Hand Side (RHS) of the constraints. That means we have random obligations to cover with our bond portfolio in the second stage. The properties of these random variables are very important in the problem since it decides the ways we can use to deal with this problem. We will see more discussion on it in the next parts.

At the beginning of the second stage, since we have some cash surplus transferred from the end of first stage, we need to minus those surplus cash Z_{t^*} from the objective function. The thing we need to be careful about is that Z_{t^*} is not a part of the decision variables of the second stage problem, in fact, it's a decision variable of the first stage problem.

The third line of the constraints in (3.5) is not trivial. It keeps us from exploring the opportunities of short-selling some over-priced bonds which happened in real finance world. With this constraint, for a certain bond, the amount we can sell at the second stage is the total amount we bought at the first and second stage. The reason we impose the constraint here is for the simplicity, but it can be cancelled. Also, in our model, for

simplicity, we assume that the bid price and ask price for bonds are just the same and we neglect the other trade costs. Thus, it's possible to see that in some solutions of the second stage, we buy and sell same amount for certain bonds simultaneously. In fact, that makes no sense for such behaviours. But this flaw will be overcome if we introduce the real bid price and ask price and trade costs. In fact, this constraint can be written as $B_{i1} \geq S_{it^*}, \quad i = 1, \ldots, N$ if we put more reality into this model, but again, for simplicity, we keep our assumption for the same bid price and ask price.

Similarly, as the end of the second stage problem, the constraint is a little different since we have no need to transfer more money into the next period. And all decision variables in the second stage problem keep positive due to the nature of finance world and the no short-selling assumption in our model.

Just as what we have showed in the (3.3), let $Q(x)$ be the recourse function of the second stage, the we can formulate the first stage as follows:

$$
\begin{aligned}
&\text{Minimize} \qquad \sum_{i=1}^{N} P_{i1} B_{i1} + Z_1 + Q(B_{i1}, Z_j) \\
&\text{subject to:} \\
&\sum_{i=1}^{N} a_{ij} B_{i1} + Z_j(1 + r_j) - Z_{j+1} = b_j, \quad j = 1, \ldots, t^* - 1 \\
&B_{i1} \geq 0, \quad i = 1, \ldots, N \\
&Z_j \geq 0, \quad j = 1, \ldots, t^* \\
&B_{i1} \in S, \quad i = 1, \ldots, N \\
&Z_j \in S, \quad j = 1, \ldots, t^*
\end{aligned}
\tag{3.6}
$$

The same as in 3.1, instead of random variables, the RHS of the constrains in the first stage problem are some numbers we already know the values. Thus, except the recourse function $Q(B_{i1}, Z_j)$, the first stage problem is a deterministic problem.

Even $Z_j, \quad j = 1, \ldots, t^* - 1$ have zero coefficients in the second stage problem, they still have impacts in second stage, since Z_{t^*} will affect the objective value of the second stage problem. Moreover, since at the end of the first stage, we will still move some money to the next period (to the beginning of the second stage problem), so all the constraints have uniform form in the first stage. Just we stated in 3.1, S is the set of B_{i1} when (3.5) has feasible solutions. Just as we have repeated for many times, all decision variables must be positive in the first stage.

Chapter 4

Approach to Two-Stage Stochastic Programming with Recourse

4.1 ξ Has Discrete Distribution

For (3.3), if several conditions are met: ξ has discrete distribution; there are finite possible values of ξ; for each possible value $\xi_1,\ \xi_2, \ldots,\ \xi_N$, there are corresponding probability $p_1,\ p_2, \ldots,\ p_N$. Then we can write problem (3.3) as follows:

$$
\begin{aligned}
&\text{Minimize} \qquad c^T x + p_1 q^T y_1 + p_2 q^T y_2 + \cdots + p_N q^T y_N \\
&\text{subject to:} \\
&Ax \qquad\qquad\qquad\quad = b \\
&Tx + Wy_1 \qquad\qquad = \xi_1 \\
&\vdots \qquad \ddots \qquad\quad \vdots \\
&Tx \qquad\qquad + Wy_N = \xi_N \\
&x \geq 0,\ y_1 \geq 0,\ y_2 \geq 0, \ldots,\ y_N \geq 0
\end{aligned}
\tag{4.1}
$$

Then it can be solved by decomposition algorithm. More details of this approach can be found in [19] and [20]. One numerical example was given by M. Mine Subasi and Ersoy Subasi in [26].

4.2 ξ Has Continuous Distribution

First, if ξ has continuous distribution, then let us assume that the P.D.F. of ξ, $p(\cdot)$ exists. In such a case, (4.1) is not valid. Here, we want to show the simulation based approach to solve (3.3).

First, let us make the following assumption:

For the vector $z = (Tx - \xi)^T$, we define the following two sets:

$$\{y : Wy = z, y \geq 0\} \quad \text{and} \quad \{z : W^T z \leq q\} \tag{4.2}$$

We assume that the two sets y and z are not empty.

Based on the assumption, we can claim that (3.1) has finite optimum values.

And if the expectation of ξ exists, then we claim $Q(x)$ exists (see [19] for the proofs of both claims).

For simplicity, we can rewrite (3.1) as the following form:

$$G(z) = q(x, \xi) = inf\{q^T y : Wy = z, y \geq 0\} \tag{4.3}$$

Suppose now that a random sample of $\xi_1,\ \xi_2, \ldots,\ \xi_N$ of i.i.d. (independent identical distribution) are generated, then $Q(x)$ can be estimated by the sample average function:

$$\hat{Q}(x) = N^{-1} \sum_{i=1}^{N} G(\xi_i - Tx) \tag{4.4}$$

Then, (3.3) can be approximated by

$$\begin{aligned} &\text{Minimize} \qquad c^T x + \hat{Q}(x) \\ &\text{subject to:} \\ &Ax = b \\ &x \geq 0 \\ &x \in S \end{aligned} \tag{4.5}$$

(4.5) is a deterministic linear programming or non-linear programming problem.

With the previous arguments, we can construct the conceptual idea fo simulation based algorithm to solve two-stage stochastic programming problem:

Algorithm 1 Simulation Based Algorithm for Two-Stage Stochastic Programming Problem (Conceptual Idea)

Let $x^0 \leftarrow$ any initial guess

Generate a random sample $\xi_1,\ \xi_2, \ldots,\ \xi_{N_0}$ based on the corresponding P.D.F. $p(\cdot)$ of ξ

repeat

Estimate $\hat{Q}_i(x^i)$ with (4.4)

Combining (3.3) with $\hat{Q}_i(x^i)$ to get a new iteration point x^{i+1} with a chosen algorithm

until stopping criterion is satisfied

This approach is known as *sample-path optimization* or *sample average approximation method*. More details of this approach can be found in [11], [23], [17].

Now we have the conceptual idea to solve the two-stage stochastic programming with recourse when the random variable has continuous distribution, but more details need to be explored. A general discussion to develop the verification methods of optimality and to test the solution optimality of two stage stochastic programming with recourse can be found in [12]. The central point of [12] is: for stochastic decomposition algorithm, it may be difficult to obtain valid estimators of a lower bound on the optimal value. It would be easier to use the derivatives of objective function to do the test. A. Shappiro and Y. Wardi gave the framework of providing the convergence of Monte Carlo based approximation algorithm with probability one in [24]. Tito Homem-de-Mello further developed the idea proposed in [12] and gave detail analysis on the algorithm. In [25] and [14], Tito Homem-de-Mello and Alexander Shapiro gave more details about the stopping criterion designation for algorithm to two-stage stochastic programming with recourse. Here, let me follow the framework he used in [14] to give a brief introduction to those basic principals of the algorithm.

4.2.1 Validation Analysis

Let we consider the following non-linear optimization problem:

$$\begin{aligned} &\text{Minimize} && f(x) \\ &\text{subject to:} && \\ &g_i(x) \geq 0, && i = 1, \ldots, m \\ &g_j(x) = 0, && j = m+1, \ldots, l \\ &x \in \mathbb{R}^m && \end{aligned} \tag{4.6}$$

By generalized KKT conditions, we have that:

$$\nabla f(x^*) - \sum_{i \in I(x^*)} \lambda_i \nabla g_i(x^*) = 0 \tag{4.7}$$

Where:

- x^* is and optimal solution of (4.6)
- λ_i is the Lagrange multipliers such that $\lambda_i \geq 0,\ i \in J(x)$
- $J(x) = \{i : g_i(x) = 0,\ i = 1, \ldots, m\}$ is the constraints active at x
- $I(x) = \{m+1, \ldots, l\} \cup J(x)$

Consider:

$$G(x) = \{z \in \mathbb{R}^m : z = \sum_{i \in I(x)} \alpha_i \nabla g_i(x),\ \alpha_i \geq 0, i \in J(x)\} \tag{4.8}$$

Then (4.7) can be written in the form $\nabla f(x^*) \in G(x^*)$.

Suppose that $f(x) = E[f(x, \xi)]$ is differentiable at point x^0, $\nabla f(x_0)$ can be estimated by a random vector $\gamma(x_0)$ such that $\gamma(x_0) \to \nabla f(x_0)$ with probability one as $N \to \infty$, and $\gamma(x_0)$ has asymptotically a multivariate normal distribution with the mean vector $\nabla f(x_0)$ and a covariance matrix Ω. Combining with previous discussion, we can test the following hypothesis with the estimator $\gamma_(x_0)$:

$$H_0 : \nabla f(x_0) \in G(x_0); \qquad H_1 : \nabla f(x_0) \notin G(x_0) \tag{4.9}$$

Suppose that covariance matrix Ω is nonsingular, and a consistent estimater $\hat{\Omega}$ is available. We define the following statistic:

$$T_1 = \min_{z \in G(x_0)} (\gamma_N(x_0) - z)^T \hat{\Omega}^{-1} (\gamma_N(x_0) - z) \tag{4.10}$$

Then by [15], this is an asymptotic analogue of Hotelling's test. In particular, under H_0, the null distribution of T_0 is central chi-square with $m-s$ degrees of freedom where:

$$s = card(I(x_0)) = m + card(J(x_0)) \tag{4.11}$$

Therefore we can calculate the p-value with the value of T_0, that is

$$p = P\{\chi^2_{m-s} \geq T_0\} \tag{4.12}$$

This p-value indicate the quality of the approximation result. A large p-value means high precision of algorithm, a small p-value means we are far from the optimal value.

The thing we need to be careful here is that for Ω, we assume that it is nonsigular. But in practice, it may be nearly singular, thus it's not invertible. Then, the following alternative test is used:

$$T_2 = \min_{z \in G(x_0)} (\gamma_N(x_0) - z)^T (\gamma_N(x_0) - z) \tag{4.13}$$

What we need to be careful about is T_2 is not central or non-central chi-square. In fact, the distribution of T_2 is a weighted sum of chi-square, variables. We may use *Pearson's approach* to approximate the distribution.

4.2.2 Choice of Sample Size

The sample size choice is a very important issue in the simulation based algorithm, since it's a trade-off between precision and efficiency. At the beginning stage, since we are far from the real optimal solution, there is no need to use a big sample size. In the evolution of algorithm, we have to choose a appropriate sample size to ensure that the algorithm proceeds in significant improvement of current solution. Basically, we employed similar idea in 4.2.1 to choose the the sample size.

With similar assumption, we have

$$Q = N(\gamma_N(x_0) - \nabla f(x))^T \hat{\Omega}_N^{-1}(\gamma_N(x_0) - \nabla f(x)) \tag{4.14}$$

Where $\hat{\Omega}_N$ is sample covariance matrix and $\hat{\Omega} = \hat{\Omega}_N/N$. Q has approximately (asymptotically) a chi-square distribution with m degrees of freedom, where m is dimension of x. Then, an (approximately) $100(1-\alpha)\%$ confidence region for $\bigtriangledown f(x)$ is given by the following ellipsoid:

$$E_N(x) = \{z \in \mathbb{R}^m : (z - \gamma_N(x))^T \hat{\Omega}_N^{-1}(z - \gamma_N(x)) \leq r_N\} \tag{4.15}$$

Where

$$r_N = \chi_m^2(\alpha)/N \tag{4.16}$$

Consider the null space $\mathcal{L}$ generated by constraints defining the set S in (3.3), and let P be the orthogonal projection on to $\mathcal{L}$, we want to find a N which satisfies that the

vector $P(c+z)$ forms an acute angle with $P(c+\gamma_N(x))$ under $100(1-\alpha)\%$ confidence level where $z \in E_N(x)$.

Thus, we need to compute $r_N^* = \max\{r : \psi(r) \geq 0\}$ where $\psi(r) = \min_{z \in E_N(x)} \alpha^T P(c+z)$ and $\alpha = P(c+\gamma_N(x))$. By KKT conditions, we have that:

$$r_N^* = (\alpha^T \alpha)^2 (\alpha^T \hat{\Omega}_N \alpha)^{-1} \tag{4.17}$$

Combining (4.16) and (4.17), we have the new sample size:

$$N^{'} = \max\{\frac{\chi_m^2(\alpha)}{r_N^*}, N\} \tag{4.18}$$

In practice, we add a factor to restrict that the jump from N to $N^{'}$ in case $N^{'}$ is too big. In our algorithm, the factor is 10.

4.2.3 Statistical Inference

To test whether x^{k+1} is a significantly better solution than x^k, we can use standard t-test. We have the following hypothesis:

$$H_0 : f(x^k) = f(x^{k+1}); \qquad H_1 : f(x^k) > f(x^{k+1}) \tag{4.19}$$

Suppose N_1 and N_2 are two independent samples of sizes, we can reject H_0 if:

$$c^T x^k + \hat{Q}_{N_1}(x^k) > c^T x^{k+1} + \hat{Q}_{N_2}(x^{k+1}) + z_\alpha(\frac{\hat{\sigma}^2_{N^1}(\hat{x}^k)}{N_1} + \frac{\hat{\sigma}^2_{N_2}(\hat{x}^k + 1)}{N_2})^{1/2} \qquad (4.20)$$

Where $z_{\alpha/2}$ is a constant related to the selected confidence level, for instance, $z_{\alpha/2} = 2.58$ corresponding to 99% confidence level.

If N_1 and N_2 are not independent, we can test the previous hypothesis with paired t-test. Suppose $\hat{Q}_N(x^k)$ and $\hat{Q}_N(x^{k+1})$ has the same sample size N and $H_1, H_2, \ldots, H_N$ are the generated random sample to estimate (4.4). Let $\overline{H} = N^{-1}\sum_{i=1}^N H_i$ represent $\hat{Q}_N(x^k) - \hat{Q}_N(x^{k+1})$ and sample variance be $s^2 = (N-1)\sum_{i=1}^N (H_i - \overline{H})^2$, then we reject H_0 if:

$$c^T(x^k - x^{k+1}) + \overline{H} > z_\alpha N^{-1/2} s \qquad (4.21)$$

Let $\hat{v}_N$ be the optimal value and $\hat{x}_N$ be the optimal solution to (4.5), let v_* be the optimal value and x_* be the optimal solution to (3.3), the upper bound and lower bound of $\hat{v}_N$ can be calculated by the following way:

By the Central Limit Theorem we have that

$$N^{1/2}[\hat{Q}(x) - Q(x)] \Rightarrow N(0, \sigma^2(x)) \qquad (4.22)$$

Where $N(0, \sigma^2(x))$ denotes a normal distribution with mean zero and variance $\sigma^2(x)$. And $\sigma^2(x)$ can be estimated by the sample variance $\hat{\sigma}^2_N(x)$.

Based on the discussion in [14], therefore, under mild regularity conditions, we have:

$$N^{1/2}[\hat{v}_N - v_*] \Rightarrow N(0, \sigma^2(x_*)) \tag{4.23}$$

Then a confidence interval of v_* can be written in the form:

$$[\hat{v}_N - z_{\alpha/2} N^{1/2} \hat{\sigma}_N(\hat{x}_N), \quad \hat{v}_N + z_{\alpha/2} N^{1/2} \hat{\sigma}_N(\hat{x}_N)] \tag{4.24}$$

With the previous information, now we have the following algorithm:

Algorithm 2 Simulation Based Algorithm for Two-Stage Stochastic Programming Problem

Let $x^0 \leftarrow$ any initial guess

Generate a random sample $\xi_1,\ \xi_2, \ldots,\ \xi_{N_0}$ based on the corresponding P.D.F. $p(\cdot)$ of ξ

repeat

Estimate $\hat{Q}_i(x^i)$ with (4.4)

Combining (3.3) with $\hat{Q}_i(x^i)$ to get a new iteration point x^{i+1} with a chosen algorithm

if $f(x^i)$ and $f(x^{i+1})$ fail to reject H_0 with the paired t-test in 4.21 **then**

Calculate N'_{i+1} as in 4.2.2 and expand the sample to $\xi_{N'_{i+1}}$

end if

Test the optimality of x^{i+1} and $f(x^{i+1})$ as in 4.2.1

until Stopping criterion is satisfied

More detail about the algorithm can be found in [14].

Chapter 5

Result and Analysis

5.1 Generic Model

First, let us see two examples of generic two-stage stochastic programming with recourse model as in 3.1. The same parameters of the numerical example in [14] were used here. We gave different random number generator seeds to each example to show how the algorithm would respond to different sample. All results keep four digits behind the decimal point. Sequential quadratic algorithm (SQP) was used in the non-linear programming step. All codes are completed in MATLAB.

Iter.	x_1	x_2	x_3	x_4	x_5	x_6	x_7	x_8
1	0.7661	0.0167	0.6704	0.0000	0.9652	0.2354	0.9483	1.1307
9	0.7790	0.0000	0.6290	0.0000	0.8241	0.2972	0.8744	0.9777
18	0.7790	0.0000	0.6290	0.0000	0.8241	0.2972	0.8744	0.9777
Iter.	x_9	x_{10}	$f_N(x^i)$	Δ	T_2 Value	p-Value	N	
1	0.6061	0.2744	24.4841	1.0000	0.0490	0.8392	50	
9	0.6981	0.3020	23.9744	0.0000	0.0000	0.8648	500	
18	0.6981	0.3020	23.9218	0.0000	0.0000	1.0000	500	

Table 5.1: Result of Generic Model (1)

Iter.	x_1	x_2	x_3	x_4	x_5	x_6	x_7	x_8
1	0.7665	0.0000	0.6693	0.0000	0.9635	0.2195	0.9475	1.1271
3	0.8423	0.6665	0.6944	0.0000	1.0110	0.8827	0.9474	1.2665
23	0.8693	0.6092	0.6940	0.0000	0.9787	0.8203	0.9501	1.2230
28	0.8693	0.6091	0.6939	0.0000	0.9789	0.8202	0.9500	1.2230
Iter.	x_9	x_{10}	$f_N(x^i)$	Δ	T_2 Value	p-Value	N	
1	0.6075	0.2817	24.3728	1.0000	0.3300	0.6124	50	
3	0.5904	0.0000	25.8009	0.6040	0.1547	0.7413	500	
23	0.5856	0.0209	25.4843	0.0000	2.8319	0.1224	1331	
28	0.5856	0.0210	25.4610	0.0000	0.0814	0.8555	5000	

Table 5.2: Result of Generic Model (2)

The time cost of 5.1 is 39 minutes. The upper bound for the optimum value is 23.9574, the lower bound is 23.8845.

The time cost of 5.2 is 16 minutes. The upper bound for the optimum value is 25.4934, the lower bound is 25.4287.

Based on the two examples, the performance of the algorithm is quite good. Both examples return us the optimum solution with high p-Value before the algorithm reaches its limit of sample size (5000). The optimum values and optimum solutions are quite close for the two examples. This quite makes sense since we used the same mean, covariance matrix and other parameters for the two examples. Besides, the time costs of two example are reasonable.

5.2 Bond Portfolio Model

Let me introduce the assumptions we used in the numerical examples first.

- No trading cost.
- The bid price and ask price of bonds are the same.
- The risk-free reinvestment rate is known during all periods.
- No default happens.
- All bonds will pay coupons twice a year.
- All cash flows can be collected immediately.
- ξ has multivariate normal distribution.

5.2.1 Bond Data Introduction

Because of no default assumption, only U.S.A. government bonds are considered as the candidates. All the data of bonds are collected from Bloomberg.

There are 98 U.S.A. government bonds whose prices are available on March 31, 2014 and maturities are greater than 5-year. The following is the scatter plot of the prices of the 98 bonds:

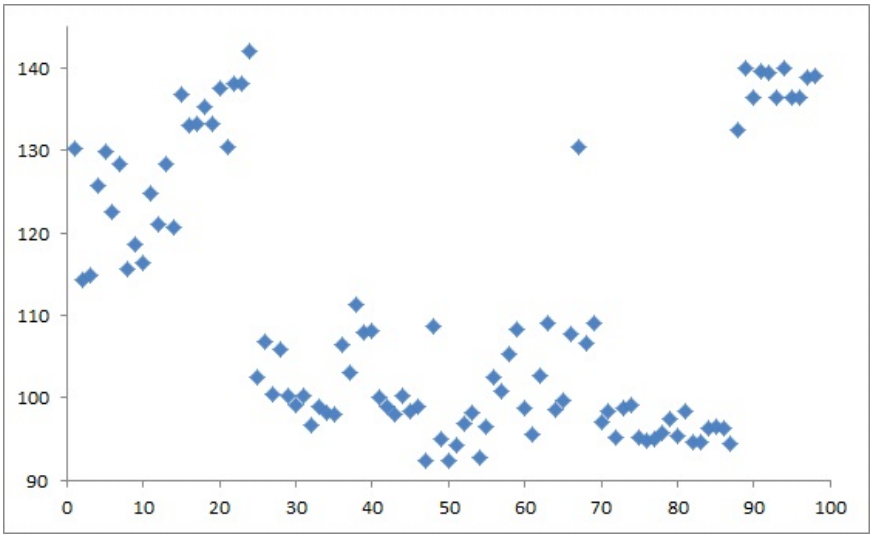

Figure 5.1: The Scatter Plot of Bond Price

The following is the scatter plot of the corresponding coupon rates of the 98 bonds.

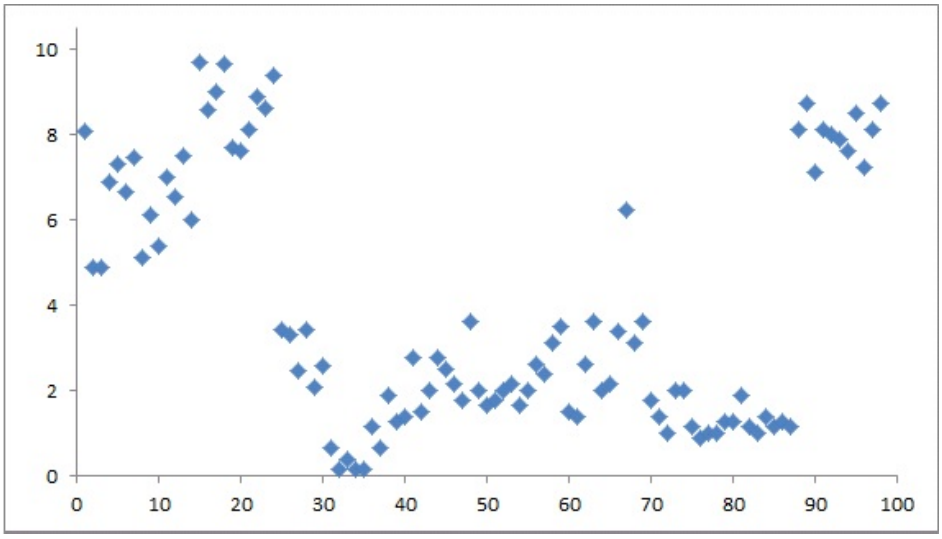

Figure 5.2: The Scatter Plot of Coupon Rate

The prices of bonds at t^* are randomly generated based on their prices on March 31, 2014, the following figure is the scatter plot of the bonds prices at t^*:

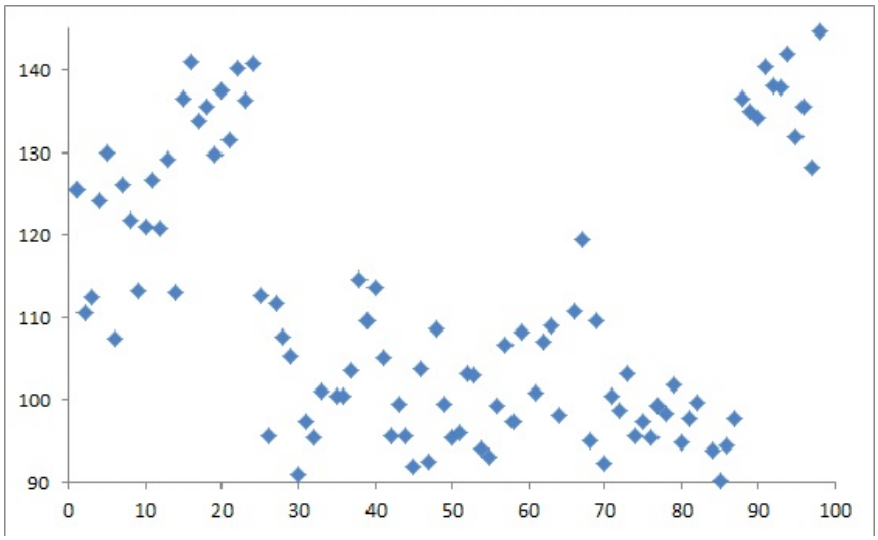

Figure 5.3: The Scatter Plot of Bond Price at t^*

5.2.2 Numerical Results

The detail of the example are as following: The time span of the problem is 5-year, each year is divided into two periods. This means $T = 10$. We want set up a bond portfolio on March 31, 2014 and will rebalance the portfolio at the beginning of the third year, thus $t^* = 5$. Since we assume that those bonds would pay coupon twice a year, the cash flows generated during each period are used to cover the obligation of the next period. The obligation of the first four periods are known, and they are random in the next 6 periods. The random variables ξ have a multivariate normal distribution and all ξ_i are independent. Besides, we impose a upper bound 10 to cash position at $t = 1$.

All the necessary parameters are randomly generated as following:

$b = [94.7800, 108.8700, 107.8400, 74.9900]^T$;

$\mu = [101.6400, 107.4800, 97.2700, 115.7600, 95.1900, 103.2800]$;

$\Omega = [10.16400, 10.74800, 9.72700, 11.57600, 9.51900, 10.32800]$;

Due to the time cost, we only involve 30 bonds in the numerical examples. They are randomly selected from the 98 bonds. And the upper bound of sample size is 50,000. Five different scenarios are considered, they are $r = 0.02, 0.045, 0.07$, upward reinvestment rate and downward reinvestment rate. For each scenario, we used three different seeds for random number generator to generate samples. Only those variables with values ≥ 0 will be showed. The (*) in the table represent that new samples are generated in the current iteration. The lower bound and upper bound of optimum value is for the last iteration in table.

When $r = 0.02$:

The time cost of 5.3 is about 5.8 minutes. The upper bound for the optimum value is 3192.30, the lower bound is 3189.20.

Iter.	x_1	Z_1	Z_2	Z_3	Z_4	Z_5
1	24.7595	10.0000	15.6959	7.4158	0.0000	0.0000
3	25.0226	10.0000	16.7614	9.5680	3.2607	29.6773
5	24.9289	8.0210	14.3636	6.7430	0.0000	25.9721
7	24.7759	10.0000	15.7625	7.5503	0.2038	25.5604
Iter.	$f_N(x^i)$	Δ	T_2 Value	p-Value	N	
1	3192.60	1.0000	1.8125	0.2705	50 (*)	
3	3165.20	6.2672	1.0001	0.3173	500 (*)	
5	3185.00	1.3862	30.6842	0.0000	5000 (*)	
7	3190.80	0.3465	0.0241	0.8787	50000 (*)	

Table 5.3: Result of $r = 0.02$ (Seed 1)

The time cost of 5.4 is about 69 minutes. The upper bound for the optimum value is 3206.80, the lower bound is 3205.00.

Iter.	x_1	x_2	Z_1	Z_2	Z_3	Z_4	Z_5
1	24.7595	0.0000	10.0000	15.6959	7.4158	0.0000	0.0000
2	24.7595	0.0000	10.0000	15.6959	7.4158	0.0000	24.5774
4	25.0969	0.0000	10.0000	17.0624	10.1761	4.1821	30.9182
8	0.0000	42.4069	1.0863	9.6948	4.3855	0.0000	28.3768
Iter.	$f_N(x^i)$	Δ	T_2 Value	p-Value	N		
1	3188.80	1.0000	0.9961	0.3183	50 (*)		
2	3167.60	25.0690	0.9933	0.3189	500 (*)		
4	3161.40	1.5668	14.3469	0.0000	5000 (*)		
8	3206.2	0.0979	0.0245	0.5103	50000		

Table 5.4: Result of $r = 0.02$ (Seed 2)

The time cost of 5.5 is about 63 minutes. The upper bound for the optimum value is 3206.80, the lower bound is 3205.00.

Iter.	x_1	x_3	x_8	Z_1	Z_2	Z_3	Z_4	Z_5
1	24.7595	0.0000	0.0000	10.0000	15.6959	7.4158	0.0000	0.0000
2	24.7595	0.0000	0.0000	10.0000	15.6959	7.4158	0.0000	24.5775
4	25.6207	0.0000	0.0000	8.9840	4.0577	10.1761	0.0629	28.8382
7	0.0000	36.3371	77.5572	1.2244	10.0371	4.9360	0.7629	29.3563
Iter.	$f_N(x^i)$	Δ	T_2 Value	p-Value	N			
1	3287.00	1.0000	4.1489	0.0486	50 (*)			
2	3185.00	25.0690	0.9262	0.3359	500 (*)			
4	3167.10	1.5668	0.9877	0.3203	5000 (*)			
7	3516.80	0.0979	0.7968	0.3720	50000 (*)			

Table 5.5: Result of $r = 0.02$ (Seed 3)

When $r = 0.045$:

The time cost of 5.6 is about 68 minutes. The upper bound for the optimum value is 3100.00, the lower bound is 3099.20.

Iter.	x_1	x_4	x_5	Z_1	Z_2	Z_3	Z_4	Z_5
1	24.6911	0.0000	0.0000	10.0000	15.6959	7.5033	0.0000	0.0000
3	25.3087	0.0000	0.0000	10.0000	15.6691	7.5033	0.0000	23.9233
5	24.9289	0.0000	0.0000	8.0210	14.3636	6.7430	0.0000	25.9721
10	2.9274	22.8130	2.7407	10.0000	16.2614	8.7145	1.8581	27.5431
Iter.	$f_N(x^i)$	Δ	T_2 Value	p-Value	N			
1	3186.60	1.0000	0.9474	0.3319	50 (*)			
3	3159.50	6.2499	1.0000	0.3173	500 (*)			
5	3165.90	0.3906	2.3721	0.1260	5000 (*)			
10	3099.60	0.0000	1.0526	0.3049	50000			

Table 5.6: Result of $r = 0.045$ (Seed 1)

The time cost of 5.7 is about 73 minutes. The upper bound for the optimum value is 3181.70, the lower bound is 3180.80.

Iter.	x_1	Z_1	Z_2	Z_3	Z_4	Z_5
1	24.6911	10.0000	15.6691	7.5033	0.0000	0.0000
3	24.6911	10.0000	15.6691	7.5033	0.0000	25.0091
5	24.6911	10.0000	15.6691	7.5033	0.0000	25.0091
7	24.6911	10.0000	15.6691	7.5033	0.0000	25.0091
Iter.	$f_N(x^i)$	Δ	T_2 Value	p-Value	N	
1	3181.70	1.0000	1.5292	0.2168	50 (*)	
3	3178.80	6.2499	11.7211	0.0000	500 (*)	
5	3180.60	0.0000	1.0092	0.3151	5000 (*)	
7	3181.30	0.0000	1.0003	0.3172	50000 (*)	

Table 5.7: Result of $r = 0.045$ (Seed 2)

The time cost of 5.8 is about 63 minutes. The upper bound for the optimum value is 3542.60, the lower bound is 3541.70.

Iter.	x_1	x_3	Z_1	Z_2	Z_3	Z_4	Z_5
1	24.6911	0.0000	10.0000	15.6691	7.5033	0.0000	0.0000
3	25.3087	0.0000	3.1239	10.9849	5.1096	0.0000	27.5104
6	25.0200	0.6695	2.1791	10.4867	5.0413	0.3997	28.3992
7	25.4190	0.0000	2.3059	10.5768	5.1298	0.4678	28.4460
Iter.	$f_N(x^i)$	Δ	T_2 Value	p-Value	N		
1	3282.90	1.0000	2.4357	0.1253	50 (*)		
3	3151.60	6.2499	1.0000	0.3173	500 (*)		
6	3156.60	0.0977	1.2848	0.2756	5000		
7	3542.20	0.0244	1.0430	0.3071	50000 (*)		

Table 5.8: Result of $r = 0.045$ (Seed 3)

When $r = 0.07$:

The time cost of 5.9 is about 52 minutes. The upper bound for the optimum value is 3004.20, the lower bound is 3003.40.

Iter.	x_1	Z_1	Z_2	Z_3	Z_4	Z_5
1	24.6255	10.0000	15.6413	7.5875	0.0000	0.0000
6	26.0955	10.0000	21.6067	19.9358	19.1780	51.2171
13	26.1566	9.5747	21.3992	19.9615	19.4532	51.7593
17	26.1596	9.5855	21.4230	19.9991	19.5056	51.8275
Iter.	$f_N(x^i)$	Δ	T_2 Value	p-Value	N	
1	3178.50	1.0000	0.7657	0.3885	50 (*)	
6	3009.30	1.5581	2107.90	0.0000	861 (*)	
13	3004.50	0.0487	167.4845	0.0000	8497 (*)	
17	3003.60	0.0122	1.0004	0.3172	50000 (*)	

Table 5.9: Result of $r = 0.07$ (Seed 1)

The time cost of 5.10 is about 69 minutes. The upper bound for the optimum value is 3170.20, the lower bound is 3169.30.

Iter.	x_1	x_3	Z_1	Z_2	Z_3	Z_4	Z_5
1	24.6226	0.0000	10.0000	15.6413	7.5875	0.0000	0.0000
2	0.0000	40.7024	10.0000	15.6413	7.5876	0.0000	23.2994
4	24.6225	0.0000	10.0000	15.6413	7.5875	0.0000	24.7313
6	24.6225	0.0000	10.0000	15.6413	7.5875	0.0000	24.7313
Iter.	$f_N(x^i)$	Δ	T_2 Value	p-Value	N		
1	3159.30	1.0000	1.0000	0.3173	50 (*)		
2	3168.20	24.9303	9.0440	0.0026	500 (*)		
4	3169.70	0.3830	0.9640	0.3262	5000 (*)		
6	3170.30	0.0000	1.0194	0.3127	50000 (*)		

Table 5.10: Result of $r = 0.07$ (Seed 2)

The time cost of 5.11 is about 57 minutes. The upper bound for the optimum value is 3144.60, the lower bound is 3143.80.

Iter.	x_1	x_3	x_4	Z_1	Z_2	Z_3	Z_4	Z_5
1	24.6226	0.0000	0.0000	10.0000	15.6413	7.5875	0.0000	0.0000
2	0.0000	40.7024	0.0000	10.0000	15.6413	7.5876	0.0000	23.2994
4	25.3691	0.0000	0.0000	2.0655	10.1749	4.7619	0.0000	27.7548
8	24.9692	0.0000	0.4392	2.7072	10.7570	5.2803	0.4502	28.1320
Iter.	$f_N(x^i)$	Δ	T_2 Value	p-Value	N			
1	3278.80	1.0000	0.9995	0.3174	50 (*)			
2	3155.60	24.9303	3.2995	0.0695	500 (*)			
4	3147.90	1.5581	0.9909	0.3195	5000 (*)			
8	3144.20	0.0243	1.1197	0.2900	50000			

Table 5.11: Result of $r = 0.07$ (Seed 3)

For the downward reinvestment rate, we have $r = 0.07$ at $t = 1, 2, 3$, $r = 0.02$ at $t = 4, 5, 6, 7$, $r = 0.0025$ at $t = 8, 9, 10$:

The time cost of 5.12 is about 68 minutes. The upper bound for the optimum value is 3167.00, the lower bound is 3166.10.

Iter.	x_1	x_3	Z_1	Z_2	Z_3	Z_4	Z_5
1	24.6225	0.0000	10.0000	15.6413	7.5875	0.0000	0.0000
3	24.8958	0.0000	10.0000	16.4575	9.8783	3.5578	29.4669
5	24.8062	0.0000	10.0000	16.3849	9.1268	2.3906	27.9134
8	0.0000	41.0158	10.0000	16.4087	9.1761	2.4672	28.0153
Iter.	$f_N(x^i)$	Δ	T_2 Value	p-Value	N		
1	3198.60	1.0000	0.1394	0.7093	50 (*)		
3	3168.50	6.2326	1.0254	0.0000	500 (*)		
5	3177.70	0.3895	1.5972	0.3318	5000 (*)		
8	3166.60	0.0061	0.2600	0.6401	50000		

Table 5.12: Result of Downward r (Seed 1)

The time cost of 5.13 is about 68 minutes. The upper bound for the optimum value is 3146.30, the lower bound is 3145.50.

Iter.	x_1	x_3	Z_1	Z_2	Z_3	Z_4	Z_5
1	24.6226	0.0000	10.0000	15.6413	7.5875	0.0000	0.0000
3	24.6225	0.0000	10.0000	15.6413	7.5875	0.0000	24.7313
7	25.6527	0.0000	2.2189	11.4877	7.3152	3.8807	32.8618
10	11.7044	23.0779	1.4724	10.7392	6.5647	3.1279	32.1442
Iter.	$f_N(x^i)$	Δ	T_2 Value	p-Value	N		
1	3191.30	1.0000	0.7270	0.4001	50 (*)		
3	3189.80	6.2326	1.4932	0.2220	500 (*)		
7	3150.00	1.5581	0.0725	0.7878	5000		
10	3145.90	0.0974	0.2558	0.6236	50000		

Table 5.13: Result of Downward r (Seed 2)

The time cost of 5.14 is about 65 minutes. The upper bound for the optimum value is 3165.60, the lower bound is 3164.70.

Iter.	x_1	x_3	x_4	Z_1	Z_2	Z_3	Z_4	Z_5
1	24.6225	0.0000	0.0000	10.0000	15.6413	7.5875	0.0000	0.0000
2	0.0000	40.7026	0.0000	10.0000	15.6413	7.5875	0.0000	24.4415
4	0.0000	33.5056	5.8833	4.6693	12.5965	6.9944	2.0302	29.4669
6	15.4616	16.1465	0.0000	5.1565	12.9158	7.1283	1.9656	29.1932
Iter.	$f_N(x^i)$	Δ	T_2 Value	p-Value	N			
1	3287.90	1.0000	0.9995	0.3174	50 (*)			
2	3173.20	24.9303	9.1647	0.0026	500 (*)			
4	3143.80	1.5581	3.4092	0.0651	5000 (*)			
6	3165.10	0.0974	0.9996	0.3174	50000 (*)			

Table 5.14: Result of Downward r (Seed 3)

For the upward reinvestment rate, we have $r = 0.0025$ at $t = 1, 2, 3$, $r = 0.02$ at $t = 4, 5, 6, 7$, $r = 0.07$ at $t = 8, 9, 10$:

The time cost of 5.15 is about 33 minutes. The upper bound for the optimum value is 3184.80, the lower bound is 3183.90.

Iter.	x_1	Z_1	Z_2	Z_3	Z_4	Z_5
1	24.8072	10.0000	15.7141	7.3525	0.0000	0.0000
2	24.8072	10.0000	15.7141	7.3525	0.0000	24.6248
4	24.8072	10.0000	15.7141	7.3525	0.0000	25.4791
7	24.8072	10.0000	15.7141	7.3525	0.0000	25.4791
Iter.	$f_N(x^i)$	Δ	T_2 Value	p-Value	N	
1	NaN	1.0000	0.1665	0.6847	50 (*)	
2	3189.70	25.1173	1.0001	0.3173	500 (*)	
4	3182.30	0.2179	1.8810	0.1703	5000 (*)	
7	3184.30	0.0000	0.0517	0.8207	50000	

Table 5.15: Result of Upward r (Seed 1)

The time cost of 5.16 is about 72 minutes. The upper bound for the optimum value is 3185.00, the lower bound is 3184.00.

Iter.	x_1	Z_1	Z_2	Z_3	Z_4	Z_5
1	24.8072	10.0000	15.7141	7.3525	0.0000	0.0000
4	24.8072	10.0000	15.7141	7.3525	0.0000	25.4791
6	24.8072	10.0000	15.7141	7.3525	0.0000	25.4791
11	24.8072	10.0000	15.7141	7.3525	0.0000	25.4791
Iter.	$f_N(x^i)$	Δ	T_2 Value	p-Value	N	
1	3181.80	1.0000	0.0412	0.8440	50 (*)	
4	3178.90	0.2178	0.9999	0.3173	500 (*)	
6	3182.10	0.0000	1.1672	0.2800	5000 (*)	
11	3184.50	0.0000	0.6140	0.4333	50000	

Table 5.16: Result of Upward r (Seed 2)

The time cost of 5.17 is about 60 minutes. The upper bound for the optimum value is 3153.00, the lower bound is 3152.20.

Iter.	x_1	Z_1	Z_2	Z_3	Z_4	Z_5
1	24.8072	10.0000	15.7141	7.3525	0.0000	0.0000
2	24.8072	10.0000	15.7141	7.3525	0.0000	24.6248
4	25.0806	10.0000	16.8215	9.5700	3.3304	29.9835
6	25.2528	8.4466	15.9615	9.4052	3.8626	31.2236
Iter.	$f_N(x^i)$	Δ	T_2 Value	p-Value	N	
1	3282.80	1.0000	0.3125	0.5762	50 (*)	
2	3181.90	25.1173	5.5118	0.0191	500 (*)	
4	3161.30	1.5698	0.9999	0.3173	5000 (*)	
6	3152.60	0.3925	0.9995	0.3174	50000	

Table 5.17: Result of Upward r (Seed 3)

5.2.3 Several Special Examples

Let us see a example with larger sample size limit. In this example, instead of 50,000, we set sample size limit as 500,000. We used Seed 1 to generate the random samples, set $r = 0.07$ and kept the other parameters.

The time cost of 5.18 is about 590 minutes. The upper bound for the optimum value is 3004.00, the lower bound is 3003.80.

Iter.	x_1	Z_1	Z_2	Z_3	Z_4	Z_5
1	24.6225	10.0000	15.6413	7.5875	0.0000	0.0000
6	26.0955	10.0000	21.6067	19.9358	19.1780	51.2171
13	26.1566	9.5747	21.3992	19.9615	19.4532	51.7593
17	26.1596	9.5855	21.4230	19.9991	19.5056	51.8275
24	26.1616	9.5438	21.3862	19.9675	19.4795	51.8073
Iter.	$f_N(x^i)$	Δ	T_2 Value	p-Value	N	
1	NaN	1.0000	0.7657	0.3885	50 (*)	
6	3009.30	1.5581	2107.90	0.0000	861 (*)	
13	3004.50	0.0487	167.4845	0.0000	8497 (*)	
17	3003.60	0.0122	1.0004	0.3172	84970 (*)	
24	3003.90	0.0000	0.0201	0.8872	500000	

Table 5.18: Result of $r = 0.07$, Sample Size Limit = 500,000 (Seed 1)

In the previous examples, we assume that all random variables ξ are independent and the covariance matrix Ω is equal to 10% of μ. We keep first part of the assumption but change the covariance matrix Ω to 50% of μ. We used Seed 1, set $r = 0.07$ and kept the other parameters.

The time cost of 5.19 is about 70 minutes. The upper bound for the optimum value is 3122.00, the lower bound is 3120.00.

Iter.	x_1	x_3	Z_1	Z_2	Z_3	Z_4	Z_5
1	24.626	0.0000	10.0000	15.6413	7.5875	0.0000	0.0000
4	24.8913	0.0000	10.0000	16.7296	9.8403	3.4988	29.5634
6	25.1798	0.0000	6.0417	13.6628	7.0232	2.0426	29.5634
11	0.0000	41.7662	4.6323	12.5036	6.8360	1.8017	29.2649
Iter.	$f_N(x^i)$	Δ	T_2 Value	p-Value	N		
1	3330.30	1.0000	0.8292	0.3698	50 (*)		
4	3130.60	1.5581	1.2086	0.2716	500 (*)		
6	3127.00	0.3895	1.0037	0.3164	5000 (*)		
11	3121.00	0.1298	1.9406	0.1636	50000		

Table 5.19: Result of $r = 0.07$ and Higher Covariance Matrix (Seed 1)

We gave a upper bound (10) on cash position in previous examples. In the following example, we kept all the other settings except cancelled the upper bound on cash.

The time cost of 5.20 is about 95 minutes. The upper bound for the optimum value is 3542.40, the lower bound is 3477.40.

Iter.	x_1	Z_1	Z_2	Z_3	Z_4	Z_5
1	0.2469	325.5226	254.5295	164.4762	69.1495	0.0000
2	0.1852	326.7259	255.5667	165.3364	69.8199	0.4673
4	0.1389	327.7175	256.4402	166.0835	70.4318	0.9346
6	0.1341	329.7977	256.5065	166.1349	70.4673	0.9530
Iter.	$f_N(x^i)$	Δ	T_2 Value	p-Value	N	
1	NaN	1.0000	1.0074	0.3155	50 (*)	
2	NaN	0.2500	0.6229	0.4336	500 (*)	
4	3534.10	0.0625	1.2098	0.2714	5000 (*)	
6	3510.40	0.0039	1.0546	0.3044	50000 (*)	

Table 5.20: Result of $r = 0.07$ and No Upper Bound on Cash (Seed 1)

Due to the consideration on time cost, we selected 30 from 98 bonds and reduced the decision variables from 103 to 35. Now let us see a example with 98 bonds. We set $r = 0.07$, used a different seed and kept the other parameters.

The time cost of 5.21 is about 383 minutes. The upper bound for the optimum value is 2704.10, the lower bound is 2703.80.

Iter.	x_7	x_{15}	x_{17}	x_{18}	x_{28}	Z_1	Z_2	Z_3
1	0.0000	8.9888	2.0136	9.7543	0.0000	10.0000	15.6413	7.5875
4	0.0000	0.0000	0.0000	20.8932	0.0000	10.0000	16.7296	9.8403
6	0.0000	0.0000	0.0000	21.2969	0.0000	3.4028	11.6183	6.3189
8	0.0000	8.1295	0.0000	13.0243	0.0000	5.2986	13.1599	7.4815
11	11.2260	0.0000	0.0000	0.0000	35.8596	0.5668	9.1424	4.2282
Iter.	Z_4	Z_5	$f_N(x^i)$	Δ	T_2 Value	p-Value	N	
1	0.0000	0.0000	2912.20	1.0000	3.2816	0.0775	50 (*)	
4	3.4988	29.5634	2740.60	1.5581	55.1468	0.0000	500 (*)	
6	1.6785	29.5634	2729.40	0.3895	1.0005	0.3172	5000 (*)	
8	2.4356	29.8866	2735.40	0.0974	0.1272	0.7226	50000 (*)	
11	0.0000	28.3259	2704.00	0.0568	0.0118	0.9135	500000	

Table 5.21: Result of $r = 0.07$, Bonds Number = 98

5.2.4 Result Analysis

Based on the previous examples, we can see that the algorithm works quite well on the model. The optimum values converge in the evolution of algorithm. The p-Value is greater than 0.29 in all examples. We get high p-Value (≥ 0.80) in several examples, such as in 5.3, 5.15

The results are reasonable. The optimum values should be close since similar parameters are used. The idea is proved by the results. In most case, the optimum values are around 3100. And the changes on reinvestment rate r don't bring too many effects on optimum values.

In most cases we choose similar bonds, such as x_1, x_3 and x_4. Since the bond data we used are collected from real market. The arbitrage opportunities in the real financial market must be rare. Our model should be able to find out such opportunities and explore them as much as possible. Thus, it quite makes sense to see a few bonds in our optimum solutions.

In 5.2.2, we set 50,000 as the sample size limit. The p-Value is around 30% in most cases. Since generally speaking, larger sample would bring higher precision, thus we want to see what the impact would be on p-Value if we use a larger sample. In 5.18, we raise the sample size limit to 500,000. With the same seed and parameter setting, we improve the p-Value from 0.3172 to 0.8872 with the cost of operation time.

In 5.19, we still have similar optimum solution and optimum value as in 5.2.2. It seems that bigger covariance doesn't affect the result a lot.

In 5.20, we can see that the upper bound on cash has great impact on result. The result proves the validity of our model again. Since as I mentioned before, our model should be able to recognize those arbitrage opportunities and try to explore them as much as possible. In the examples, we impose no cost on cash (the coefficient of Z_1 is 1). This makes cash much cheaper than bonds. Once we cancelled the limit on cash position when we set up portfolio, the model successfully explored the opportunity.

The last example shows that our model works well when we have larger inputs. The optimum value is lower than the examples in 5.2.2, p-Value is high (≥ 0.90). The thing

we should notice is in 5.21, those bonds appear in optimum solutions are not among those 30 bonds we used in 5.2.2.

Chapter 6

Conclusion

This paper developed two-stage stochastic programming with recourse model for bond portfolio optimization problem. We introduced simulation based algorithm to two-stage stochastic programming and applied it on this model to handle when random variables ξ has continuous distribution instead of discrete distribution. We carried out the tests of numerical examples under various scenarios and got good results. The efficiency of the simulation based algorithm on our model has been proved by these results.

There are several possible directions for future research on the topic:

First, we can add more realistic factors into the model, for instance, the possibility of bond default. Since in generic two-stage stochastic programming model, beside ξ, T could be random too. Thus we can include bond default into the model by randomizing T. By including bond default, more categories of bonds can be candidates of our model, such as corporate bonds and even those junk bonds. We can see in 5.21, more bond candidates may reduce the cost of setting up portfolio.

Second, finding ways to increase p-Value.Most final optimum solutions in our examples have p-Value around 0.30. But in some cases, we have p-Value which is greater than 0.80. So it's important to find our how to get higher p-Value for the cases whose p-Value

are relatively low. One guess is that the violation of the assumption (4.2) may bring low p-Value. Instead, in examples 5.1 and 5.2, the assumption is always guaranteed. So it's worthy to do further study on whether eliminating the infeasibility or the violation of the previous assumption would bring good p-Value.

Third, based on our examples, we can see that the time cost is huge in some cases, especially when we have large sample size limit, such as 500,000. But when we have many decision variables, large sample size is necessary to guarantee the precision of result. There are many ways to increase the efficiency of algorithm, for instance, coding with C++ instead of MATLAB and improving the code structure.

References

[1] Sthephen P. Bradley and Dwight B. Crane. A dynamic model for bond portfolio mangament. *Managment Science*, 19(2), October 1972.

[2] Krzysztof Burnecki, Grzegorz Kukla, and Rafal Weron. Property insurance loss distribution. *Physica A*, 287:269–278, 2000.

[3] Andrew J.G. Cairns. *Interest Rate Models.* John Wiley and Sons, 2004.

[4] George B. Dantzig. Linear programming under uncertainty. *Management Science*, 1(3/4):197–206, 1955.

[5] Lev Dynkin, Anthony Gould, Jay Hyman, Vadim Konstantinovsky, and Bruce Phelps. *Quantitative Management of Bond Portfolio.* Princeton Unversity Press, 2006.

[6] Edwin J. Elton, Martin J. Gruber, Stephen J. Brown, and William N. Goetzmann. *Mordern Portfolio Theory and Investment Analysis.* Wiley, sixth edition, 2003.

[7] Frank J. Fabozzi. *Bond Portfolio Management.* Frank J. Fabozzi Associates, second edition, 2001.

[8] Frank J. Fabozzi, Lionel Martellini, and Philippe Priaulet, editors. *Advanced Bond Portfolio Management.* Wiley, 2006.

[9] Karl Frauendorfer. *Stochastic Two-Stage Programming.* Springer-Verlag, 1992.

[10] Rajna Gibson, Franois-Serge Lhabitant, and Denis Talay. Modeling the term structure of interest rates: An overview. *The Journal of Risk*, 1(3), 1999.

[11] Gül Gürkan, A. Yonca Özge, and Stephen M. Robinson. Sample-path optimization in simulation. In *Proceedings of the 1994 Winter Simulation Conference*, July 1994.

[12] Julia L. Higle and Suvrajeet Sen. Statistical verification of optimality conditions for stochastic programs with recourse. *Annals of Operations Research*, 30:215–240, 1991.

[13] S. D. Hodges and S. M. Schaefer. A model for bond portfolio improvement. *Journal of Financial and Quantitative Analysis*, 12:243–260, June 1977.

[14] Tito Homem-de-Mello and Alexander Shapiro. A simulation-based approach to two-stage stochastic programming with recourse. *Mathematical Programming*, 81:301–325, 1998.

[15] Robb J. Muirhead. *Aspects of multivariate statistical theory*. Wiley, 1982.

[16] Jeffrey S. Pai, Kevin J. Shand, and Xikui Wang. Compound poisson model with covariates: a case study on pet insurance. *North American Actuarial Journal*, 10(4):219–234, April 2007.

[17] Erica L. Plambeck, Bor-Ruey Fu, Stephen M. Robinson, and Rajan Suri. Sample-path optimization of convex stochastic performance functions. *Mathematical Programming*, 75:137–176, 1996.

[18] Nicholas G. Polson and Morten Sorensen. A simulation-based approach to stochastic dynamic programming. *Applied Stochastic Models In Business and Industry*, 27:151–163, 2011.

[19] András Prékopa. *Stochastic Programming*. Akadémiai Kiadó and Kluwer Academic Publishers, 1995.

[20] András Prékopa. *Stochastic Programming*. Rutgers Center of Operations Research, Jan. 2013.

[21] Werner Römisch and Rüdiger Schultz. Stability of solutions for stochastic programs with complete recourse. *Mathematics of Operations Research*, 18(3):590–609, August 1993.

[22] Ehud I. Ronn. A new linear programming approach to bond portfolio management. *The Journal of Financial and Quantitative Analysis*, 22(4):439–466, Dec. 1987.

[23] Reuven Y. Rubinstein and Alexander Shapiro. Optimization of static simulation models by the score function method. *Mathematics and Computers in Simulation*, 32:373–392, 1990.

[24] A. Shapiro and Y. Wardi. Convergence analysis of stochastic algorithms. *Mathematics of Operations Research*, 21(3):615–628, Aug. 1996.

[25] Alexander Shapiro. Stochastic programming by monte carlo simulation methods. *Stochastic Programming E-Print Series*, 2000.

[26] M.Mine Subasi and Ersoy Subasi. Two-stage bond portfolio optimization problem. Research Project, Spring 2002.

[27] Roger J.-B. Wets. Stochastic programs with fixed recourse: The equivalent deterministic program. *SIAM Revier*, 16(3), July 1974.

Appendices

Algorithm Codes

```
t1 = tic;
pR = 0.70;  % the required P level
delta =[];
delta = [delta 1]; % default delta value
zAlpha = 1.64; % alpha = 0.05, one tail
zAlpha2 = 1.96; % alpha = 0.05, two tails
chiSquareAlpha = 0.05;
NUpLimit = 50000;
k = 0;
options = optimset('Algorithm','sqp');

optX = [];
optX = [optX, x0];
N =[];
N =[N, 50];
optValue = [];
T2 = [];
T22 = [];
alphaOpt = [];
pIndex = [-1];
p = [];
p = [p, -3];
opTime = [];

numTest;
rng(seed);
Z = sampleGen(mu,sigma,N(k + 1));
Z1 = Z;

sizeX = size(optX);
```

```
lb = zeros(sizeX);
ub = ones(sizeX) * Inf;
ub(31) = 10;

endCount = 3;

while p(k + 1) < pR || pIndex(k + 1) == 0
    time = tic;

    if k == 0

        [gK,gdK,gddK,fDK] = ggdRCal(optX(:, k + 1),optX(:, k + 1),...
            c,Z,N(k + 1),T,W,WP,q,mu,sigma);
        [xK, fValK] = fmincon(@(x)lKCal(x,optX(:, k + 1),T,c,gK,gdK),...
            optX(:, k + 1),[],[],A,b,lb,ub,...
            @(x)nLCon(x,T,optX(:, k + 1),delta(k + 1)),options);
        optX = [optX, xK];
        k = k + 1;
        N = [N, N(k)];
        [gK,gdK,gddK,fDK,GK,LK,sRK,fGK,rIK] = ggdRCal(optX(:, k + 1),...
            optX(:, k + 1),c,Z,N(k),T,W,WP,q,mu,sigma);
        optValueK = fKCal(optX(:, k + 1),c,fGK);
        optValue = [optValue, optValueK];

        [T2K, alphaOptK] = T2Cal(A,c,T,gdK);
        T2 = [T2, T2K];
        alphaOpt = [alphaOpt, alphaOptK];
        [T22K,pK] = pCal(A,N(k),fDK,T2K);
        p = [p, pK];
        T22 = [T22, T22K];
```

```
61          if alphaOptK > 0
62              pIndex = [pIndex, 1];
63          else
64              pIndex = [pIndex, 0];
65          end
66
67      else
68
69          [gK1,gdK1,gddK1,fDK1,GK1,LK1,sRK1,fGK1,rIK1] = ggdRCal(...
70              optX(:, k + 1),optX(:, k + 1),c,Z1,N(k + 1),T,W,WP,q,mu,sigma);
71          rK = rKCal(optX(:, k),optX(:, k + 1),c,T,gK,gK1,gdK,gddK);
72          Z = Z1;
73          if rK < 0.25
74              deltaK = norm(T * optX(:, k + 1) - T * optX(:, k), Inf) / 4;
75              delta = [delta, deltaK];
76          else
77              if rK > 0.75 && norm(T * optX(:, k + 1) - T * ...
78                      optX(:, k), Inf) == delta(k)
79                  deltaK = 2 * delta(k);
80                  delta = [delta deltaK];
81              else
82                  deltaK = delta(k);
83                  delta = [delta deltaK];
84              end
85          end
86
87          gK = gK1;
88          gdK = gdK1;
89          [xK, fValK] = fmincon(@(x)lKCal(x,optX(:, k + 1),T,c,gK,gdK),...
90              optX(:, k + 1),[],[],A,b,lb,ub,...
```

```
91              @(x)nLCon(x,T,optX(:, k + 1),delta(k + 1)),options);
92          optX = [optX, xK];
93          k = k + 1;
94          N = [N, N(k)];
95          [gK,gdK,gddK,fDK,GK,LK,sRK,fGK,rIK] = ggdRCal(optX(:, k + 1),...
96              optX(:, k + 1),c,Z,N(k),T,W,WP,q,mu,sigma);
97          optValueK = fKCal(optX(:, k + 1),c,fGK);
98          optValue = [optValue, optValueK];
99
100         if N(k) == N(k - 1)
101
102             t = pairedTTest(optX(:, k),optX(:, k + 1),...
103                 gK1,gK,c,N(k),GK1,LK1,GK,LK,zAlpha);
104
105             if t == 0
106
107                 NK = nCal1(optX(:, k + 1),c,N(k),...
108                     NUpLimit,T,A,fDK,gdK,chiSquareAlpha);
109                 N(k + 1) = NK;
110                 rng(seed);
111                 Z1 = sampleGen(mu,sigma,N(k + 1));
112
113             end
114
115         end
116
117         [T2K, alphaOptK] = T2Cal(A,c,T,gdK);
118         T2 = [T2, T2K];
119         alphaOpt = [alphaOpt, alphaOptK];
120         [T22K,pK] = pCal(A,N(k),fDK,T2K);
```

```
121           p = [p, pK];
122           T22 = [T22, T22K];
123           if alphaOptK > 0
124              pIndex = [pIndex, 1];
125           else
126              pIndex = [pIndex, 0];
127           end
128
129       end
130
131       tK = toc(time);
132       opTime = [opTime, tK];
133
134       if N(k) == NUpLimit
135           endCount = endCount - 1;
136       end
137
138       if endCount == 0
139           break;
140       end
141
142   end
143
144   [optVLowerBound, optVUpperBound] = boundCal(optX(:, k + 1),...
145       optX(:, k + 1),c,Z,N(k),T,W,WP,q,mu,sigma,zAlpha2,optValue(k));
146
147   tW = toc(t1) / 60;
```

```
function [g,gd,gdd,fD,G,L,secResult,feaG,resultIndex] ...
    = ggdRCal(x,xK,c,Z,N,T,W,WP,q,mu,sigma)

tK = tKCal(T,xK);
t = T * x;
YK = yKCal(Z,tK,N);
G = [];
sizeQ = size(q);
lb = zeros(sizeQ);
resultIndex = zeros(N,1);
secResult = [];


for i = 1 : N
    [f,fVal,exitflag] = linprog(q,WP,xK,W,YK(:,i),lb);
    secResult = [secResult, f];
    G = [G,fVal];
    if exitflag == 1
       resultIndex(i) = 1;
    end
end

L = [];
YKT =YK';
tT = t';

for i = 1 : N
    temp = mvnpdf(YKT(i,:),mu-tT,sigma) / mvnpdf(YKT(i,:),mu-tK',sigma);
    L = [L,temp];
end
```

```
31
32  cSigma = diag(sigma);
33  cISigma = cSigma^(-1);
34  D = [];
35
36  for i  = 1 : N
37      temp = YK(:,i) - mu' + t;
38      D = [D,temp];
39  end
40
41  g = (G * L') / N;
42
43  feaG = 0;
44
45  for i = 1 : N
46      if resultIndex(i) == 1
47          temp = G(i) * L(i);
48          feaG = feaG + temp;
49      end
50  end
51
52  feaG = feaG / sum(resultIndex);
53
54  sizeYK = size(YK);
55  gd = zeros(sizeYK(1),1);
56
57  for i = 1 : N
58      temp = G(i) * L(i) * cISigma * D(:,i);
59      gd = gd + temp;
60  end
```

```
61
62  gd = gd ./ N;
63
64  fD = [];
65
66  for i = 1 : N
67      temp = c + T' * (G(i) * L(i) * cISigma * D(:,i));
68      fD = [fD, temp];
69  end
70
71  gdd = cISigma * 0;
72
73  for i = 1 : N
74      temp = G(i) * L(i) * ((cISigma * D(:,i)) ...
75          * (cISigma * D(:,i))' - cISigma);
76      gdd = gdd + temp;
77  end
78
79  gdd = gdd ./ N;
```

```
function t = pairedTTest(xK0,xK1,g0,g1,c,N,G0,L0,G1,L1,zAlpha)

wA = fKCal(xK0,c,g0) - fKCal(xK1,c,g1);
wI = [];

for i = 1 : N
    temp = c' * xK0 + G0(i) * L0(i) - c' * xK1 - G1(i) * L1(i);
    wI = [wI, temp];
end

rN2 = sum((wI - wA).^2) / (N - 1);
sN = sqrt(rN2 / N);

if wA > zAlpha * sN
    t = 1;
else
    t = 0;
end

```

```
function nP = nCal1(xK,c,N,NUpperLimit,T,A,fD,gd,chiSquareAlpha)

sampleCM = cov(fD');
W = A';
P = W * (W' * W) ^ (-1) * W';
v = P * (c + T' * gd);
rNOpt = (v' * v) ^ 2 / (v' * sampleCM * v);
sizeX = size(xK);
chiSquareV = chi2inv(1 - chiSquareAlpha, sizeX(1));
nT = ceil(chiSquareV / rNOpt);
nP = max(nT, N);

if nP >= 10 * N
    nP = 10 * N;
end

nP = min(nP, NUpperLimit);
```

```
1 function [T2,alphaOpt] = T2Cal(A,c,T,gd)
2
3 AT = A';
4 gammaN = c + T' * gd;
5 alphaOpt = (AT' * AT)^(-1) * AT' * gammaN;
6 T2 = ( gammaN - AT * alphaOpt)' * (gammaN - AT * alphaOpt);
```

```
function [T22,p] = pCal(A,N,fD,T2)

AT = A';
Q = AT * (AT' * AT) ^ (-1) * AT';

sampleCM = cov(fD');
simCMN = sampleCM ./ N;
sizeQ = size(Q);
I = eye(sizeQ);
beta = eig((I - Q) * simCMN);

theta1 = sum(beta);
theta2 = sum(beta.^2);
theta3 = sum(beta.^3);

v = theta2 ^ 3 / theta3 ^ 2;
c = theta3 / theta2;
b = theta1 - (theta2 ^ 2 / theta3);
T22 = (T2 - b) / c;

if isinteger(v)
   p = 1 - chi2cdf((T2 - b) / c, v);
else
   p = 1 - gamcdf((T2 - b) / c, v / 2, 2);
end
```

```
function [lbb, ubb] = boundCal(x,xK,c,Z,N,T,W,WP,q,mu,sigma,zAlpha2,optR)

tK = tKCal(T,xK);
t = T * x;
YK = yKCal(Z,tK,N);
G = [];
sizeQ = size(q);
lb = zeros(sizeQ);
resultIndex = zeros(N,1);

for i = 1 : N
    [f,fVal,exitflag] = linprog(q,WP,xK,W,YK(:,i),lb);
    G = [G,fVal];
    if exitflag == 1
        resultIndex(i) = 1;
    end
end

L = [];
YKT =YK';
tT = t';

for i = 1 : N
    temp = mvnpdf(YKT(i,:),mu-tT,sigma) / mvnpdf(YKT(i,:),mu-tK',sigma);
    L = [L,temp];
end

optV = [];

for i = 1 : N
```

```
31      if resultIndex(i) == 1
32          temp = c' * x + G(i) * L(i);
33          optV = [optV, temp];
34      end
35  end
36
37  standDevOptV = sqrt(var(optV));
38
39  lbb = optR - zAlpha2 * standDevOptV / sqrt(sum(resultIndex));
40  ubb = optR + zAlpha2 * standDevOptV / sqrt(sum(resultIndex));
```

```
function Z = sampleGen(mu,sigma,N)

Z = [];

for i = 1 : N
  Z = [Z;mvnrnd(mu,sigma)];
end

Z = Z';
```

```
function [c,ceq] = nLCon(x,T,xK,delta)

tK = tKCal(T,xK);
t = T * x;
c = norm((t - tK),Inf) - delta;
ceq = [];
```

```
1  function f = fKCal(xK,c,g)
2
3  f = c' * xK + g;
```

```
1  function lK = lKCal(x,xK,T,c,g,gd)
2
3  tK = tKCal(T,xK);
4  t = T * x;
5  cT = c';
6  lK = cT * x + g - gd' * (t - tK);
```

```
1  function qK = qKCal(x,xK,g,gd,gdd,c,T)
2
3  tK = tKCal(T,xK);
4  t = T * x;
5  cT = c';
6  qK = cT * x + g - gd'  * (t - tK) + 0.5 * (t - tK)' * gdd * (t - tK);
```

```
1  function rK = rKCal(xK0,xK1,c,T,g0,g1,gd0,gdd0)
2
3  rK = (fKCal(xK0,c,g0) - fKCal(xK1,c,g1)) ...
4       / (fKCal(xK0,c,g0) - qKCal(xK1,xK0,g0,gd0,gdd0,c,T));
```

```
1  function tK = tKCal(T,x)
2
3  tK = T * x;
```

```
function YK = yKCal(Z,t,N)

YK = [];

for i = 1: N
    temp = Z(:,i) - t;
    YK = [YK,temp];
end
```