Portfolio Optimization with Transaction Costs and Preconceived Portfolio Weights

by

Jeremy D. Myers

Submitted to the Department of Electrical Engineering and Computer Science
in partial fulfillment of the requirements for the degree of

Masters of Engineering in Electrical Engineering and Computer Science

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

September 2009

Author . . . .
Department of Electrical Engineering and Computer Science
August 21, 2009

Certified by . . . .
Leonid Kogan
Nippon Telephone and Telegraph Professor of Management
Thesis Supervisor

Accepted by . . . .
Dr. Christopher J. Terman
Chairman, Department Committee on Graduate Theses

Portfolio Optimization with Transaction Costs and Preconceived Portfolio Weights

by

Jeremy D. Myers

Submitted to the Department of Electrical Engineering and Computer Science
on August 21, 2009, in partial fulfillment of the
requirements for the degree of
Masters of Engineering in Electrical Engineering and Computer Science

Abstract

In the financial world, many quantitative investment managers have developed sophisticated statistical techniques to generate signals about expected returns from previous market data. However, the manner in which they apply this information to rebalancing their portfolios is often ad-hoc, trading off between rebalancing their assets into an allocation that generates the greatest expected return based on the generated signals and the incurred transaction costs that the reallocation will require. In this thesis, we develop an approximation to our investor's true value function which incorporates both return predictability and transaction costs. By optimizing our approximate value function at each time step, we will generate a portfolio strategy that closely emulates the optimal portfolio strategy, which is based on the true value function. In order to determine the optimal set of parameters for our approximate function which will generate the best overall portfolio performance, we develop a simulation-based method. Our computational implementation is verified against well-known base cases. We determine the optimal parameters for our approximate function in the single stock and bond case. In addition, we determine a confidence level on our simulation results. Our approximate function gives us useful insight into the optimal portfolio allocation in complex higher dimensional cases. Our function derivation and simulation methodology extend easily to portfolio allocation in higher dimensional cases, and we implement the modifications required to run these simulations. Simple cases are tested and more complex tests are specified for testing when appropriate dedicated computing resources are available.

Thesis Supervisor: Leonid Kogan
Title: Nippon Telephone and Telegraph Professor of Management

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Portfolio optimization has been the Holy Grail of Investment Finance over the past half-century. At the personal finance level, individual investors looking to secure enough money for retirement are always debating the correct distribution of wealth distribute stocks and bonds. Fixed financial instruments such as bonds guarantee the investor a steady but low interest rate, whereas stocks provide the investor an opportunity for high level gains, but with significant risk. Many individuals who are not blessed with the opportunity to take a graduate level finance course use rules of thumb, such as the common 60/40 stock-to-bond ratio as a safe course of action. Before the stock market crisis of these past few years, many individuals who were far from retirement would invest all of their retirement savings in the stock market, convinced that by the time they retired, the stock market would assuredly be at new record heights.

At the corporate finance level, multi-billion dollar financial corporations and hedge funds continue to conduct research on the optimal asset allocation method which incorporates as much information and flexibility as possible. Although the goals of these different types of investors are slightly different, both share a common need for an efficient asset allocation method.

1.1 History of Portfolio Optimization

Even though the half-century of history of portfolio optimization theory is relatively short compared to the longevity of the stock market, which saw its birth in the 1700s after the American Revolution, the progress researchers have made during that time is astonishing. In 1952, Harry Markowitz introduced what we know today as Modern Portfolio Theory, which describes how investors will diversify their portfolios to minimize the variance in the portfolio for a given level of expected return [9]. Robert Merton helped revolutionize the field with his papers on the asset allocation problem [10]. Merton made it possible to obtain closed-form solutions to the asset allocation problem, given a few assumptions and constraints (this model will be discussed further in Section 2.4). However, today we find that the assumptions made by Merton in developing his model to be very unrealistic in our current financial markets.

One key assumption of the Merton model is that there are no transactions costs. We not only know that transaction costs exist in the financial markets, but also that their cumulative effect is significant. According to the Merton model, at every possible trading opportunity, the investor should trade so that the Merton ratio of stocks to bonds is maintained. With the introduction of transaction costs, it is no longer optimal for the investor to maintain this ratio at every trading opportunity. Today's investor must weigh the benefit of transacting to an optimal portfolio position and the transaction costs that would be incurred in doing so. If an investor were to trade continuously, the transaction costs would reduce the portfolio's value drastically. A few decades ago, this situation in the financial world was unheard of, as people would buy and sell relatively infrequently, compared to today. However, over the last decade came the advent of the algorithmic traders, who use intense and constant computational power to identify arbitrage opportunities in the market. These types of investors understand transactions costs to be of the utmost importance. Opportunities for arbitrage, for which trade volumes can easily exceed hundreds of thousands for a single opportunity, strongly depend upon the seemingly small transaction costs that the transactions will incur.

1.2 Introduction of Transactions Costs to Portfolio Optimization

George Constantinides explicitly proved how the existence of transaction costs leads to less frequent trading [3]. In the same paper, Constantinides proved that there exist transaction boundaries for the single stock and bond investor. If the value fluctuations of the portfolio's holdings led the portfolio allocation to be outside the transaction boundary lines, the optimal asset allocation decision would be to transact back to the nearest boundary line. Constantinides proved the shape of these transaction boundaries were convex cones, given transaction costs that are proportional to the transaction amount. In addition, he was able to numerically calculate these boundaries for a power utility investor in the infinite horizon case [4]. The infinite horizon case of portfolio optimization is when the investor never withdraws from the market and invests forever. Considering that this specification simplifies the portfolio optimization problem, the infinite horizon case was further developed by Davis and Norman [6] who were able to model the portfolio allocation problem as a stochastic singular control problem in the event that transaction costs applied only to the risky asset. Oksendal and Sulem [11] also considered this infinite horizon case, but coupled the optimal consumption problem along with portfolio optimization. However, for practical applications, one must consider the solution to the finite horizon case, as no investor invests forever. Intuitively, one will come to the conclusion that as an investor approaches the end of his investment time horizon (e.g., prepares to retire and withdraw from market), the transaction boundary lines for his investment portfolio should grow monotonically. In other words, as the investor prepares to exit the market, he will be more tolerant of his portfolio's composition. The (not so) long run benefits of adjusting the portfolio are outweighed by the immediate transaction costs the rebalancing would require.

Significant work has also been done in the finite horizon case. Gennotte and Jung [7] were able to develop a numerical method which allows one to obtain a numerical approximation for the no-transaction boundaries. Boyle and Lin [1] go on to provide an explicit closed-form solution to the finite horizon case using proportional transactions costs and a power utility function to describe the risk aversion of the investor. Current work, such as the one by Zakamouline [12], considers both fixed and proportional transactions costs while numerically solving the asset allocation problem.

In addition to the introduction of transactions costs to asset allocation models, return predictability has recently become an important factor in portfolio choice. With the current computational capabilities of today's hardware and software, financial institutions are now able to research historical market data and generate relatively reliable expected future stock returns. Many papers, like the working paper by Campbell and Viceira [2], show that returns are predictable to some extent.

1.3 Return Predictability vs. Transaction Costs

There is a hole in the literature on the trade-offs between rebalancing one's portfolio based on return predictability and the incurred transaction costs. The paper by Lynch and Balduzzi [8] attempts to reconcile the effects of return predicatability and transaction costs. In order to maximize their function (which is a Bellman equation), they backward solve from $t = T-1$ to get their optimal solution. This is problematic on two counts. First, investors historically do a poor job of estimating their true time horizon; factors of personal preference play too large a role and are generally not well understood until later stages. Second, simulating backwards requires one to take into account all possible paths the stock may take from $t = 0$ to T, to compute the total probability, just like in the computation of quantum probabilities in Feynman diagrams. In more applied settings, many quantitative portfolio managers use statistical techniques and computing power to generate signals concerning expected stock returns. Instead of having an asset allocation method that includes both the signals on stock returns and transaction costs, managers use ad-hoc methods in weighing these two factors when rebalancing their portfolios. For the individual investor, rules of thumb or personal preference often replace numerical values for predicted stock returns in recommending that they maintain a certain portfolio composition.

1.4 Scope of the Paper

In this paper, we develop a new function that approximates an investor's true value function. The function takes into account transaction costs and return predictability. We develop the portfolio optimization problem in a discrete time domain, employing the Cox, Ross, and Rubinstein discrete time approximation for returns on stocks and bonds after we rebalance the portfolio at each time iteration (see Section 2.6). the approximate function includes several parameters to allow tuning to particular return and preferential situations. In order to determine the optimal parameters for the approximate value function, we perform numerical simulations on the evolution of various case portfolios using MATLAB. The computational simulations utilize dynamic portfolio choice to find the optimal portfolio composition given the parameters at each time step. The function provides a high degree of flexibility, allowing the user to alter parameters to reflect the risk aversion of the investor, the degree of user certainty in their estimate of the optimal portfolio composition, and their estimate of expected returns. Shorting stocks or bonds is allowable as well. With minor additions, the function is extended to incorporate more than one stock or bond in the portfolio choice parameters. In addition, we test the accuracy of the chosen parameters for the approximate function resulting from the solution through a simulation-based evaluation method.

The paper is organized as follows. Chapter 2 presents assumptions and important results from past papers that we will use in this paper. The discrete-time simulated returns and investor utility function are discussed. Chapter 3 presents the theory and derivation of the new approximate value function. Using the results and assumptions from chapter 2, we derive a approximate function which also takes return predictability and transaction costs into account. Chapter 4 presents the computational implementation of the portfolio optimization simulations. All code needed to simulate the function is sectioned into parts, and are described in detail. Chapter

5 presents the results and analysis of the simulations to recover the optimal parameters for the function. We compare the function's performance against benchmark values and simulate in situations where no closed form solution is known. Chapter 6 concludes the paper and discusses possible research extensions stemming from the paper.

Chapter 2

Assumptions for Paper and Important Results from Theory of Portfolio Optimization

In the previous section, we discussed some of the relevant background results in portfolio optimization. There are a few theoretical results that will be foundational to what we will do in this paper. In this section, we will discuss in depth those necessary results and assumptions that we will need as the foundation for the new approximate value method; these will be key in optimizing the portfolio in the single stock and bond case, as well as in higher dimensional cases.

2.1 Returns Processes of Stocks and Bonds

One of the most important assumptions is how we will model the returns process of a stock and of a bond. Modeling the stock price process stochastically as a Geometric Brownian Motion is an industry standard, since prices move continuously and randomly. If we model the bond price process movement continuously as well, the price processes for the stock and bond are:

$$\begin{aligned} dY_t = & \quad Y_t \mu dt + Y_t \sigma dZ_t \\ dX_t = & \qquad X_t r dt \end{aligned} \tag{2.1}$$

where μ represents the mean return of the stock (drift of the Brownian Motion), σ represents the variance of the stock (variance of the Brownian Motion), r represents the return of the bond, Y_t represents the price of the stock at time t, and X_t represents the price of the stock at time t. Solutions to these equations are analyzed in most graduate finance textbooks.

2.2 Investor's Utility Function

A second major assumption that is needed for the new function is the form of the investor's utility function, which determines how the investor values wealth. The Constant Relative Risk Aversion (CRRA) utility function is a widely accepted utility function to describe risk averse investors:

$$U(W_t) = \frac{1}{1-\gamma}(W_t)^{1-\gamma}(\text{for } \gamma \geq 0) \tag{2.2}$$

where U is the utility function, W_t is the wealth at time t, and γ is our risk aversion coefficient. A risk averse investor is one who values an increment of wealth more when their total wealth is low, and values an increment of wealth less when their total wealth is higher. The bound on γ is necessary because when $\gamma = 0$, the investor is considered risk-neutral, meaning he values an increment of wealth the same given any current total wealth. When $\gamma < 0$, he is considered to be a risk loving investor. An investor is said to be risk loving in the sense that given a guaranteed return and a risky return whose expected value is below the guaranteed return, that investor will choose the risky return; this is not considered rational behavior.

The utility function in Eq. (2.2) is the base of the value function in that it determines how we value different levels of wealth. For any distribution of wealth W_t, an investor is able to assign a theoretical value to that level of wealth. This is important in describing the investor's true value function in section (2.3).

2.3 Investor's True Value Function

Now that we have the form of the investor's utility function, we can define the form of the investor's true value function. Using the continuous-time models of the stock and bond returns in Eq. (2.1), the CRRA utility function defined in Eq. (2.2), and knowledge of dynamic programming from Ph.D Finance courses, we can reduce our true value function to the form:

$$V(W_t, \pi_t) = \frac{1}{1-\gamma} * F(\pi_t) * (W_t)^{1-\gamma} \tag{2.3}$$

where π_t is the portfolio composition (ratio of stocks to total wealth) and F is a function of π_t that reflects the dependence of portfolio return on portfolio composition. It is possible to determine an exact expression for the form of the function F. However, trying to dynamically maximize this function of π as well as the other components to the true value function is very inefficient. Our new approximate value function that we will derive in Chapter 3 will approximate the dynamic function F in the value equation. This approximate function will emulate the true value function well and therefore will produce an approximately equivalent trading strategy as the true value function. The advantage of this approximation is that the new function will be much easier and quicker to maximize at each time step than the true value function, while they both will produce similar results. This true value function is the basis off which we will derive our new approximate value function.

2.4 Continuous-Time Solution of Portfolio Optimization in Single Stock and Bond Case

In the Literature Review section earlier, we referred to Robert Merton's asset allocation paper [10]. In it, Merton was able to derive an analytic closed-form solution to the portfolio optimization problem with a single stock and bond, and with no transaction costs. The returns processes he used for his stock and bond and the utility function he used to describe the risk-averion of his investor are as in Eq. (2.1) and Eq. (2.2) above. The solution to his portfolio choice problem was a static position in the market, meaning the optimal ratio of stock-to-bond to hold remained the same in every time period. Given the CRRA utility function and returns processes, the solution for the static portfolio to hold in all time periods, famously known as Merton's ratio, is:

$$\pi = \frac{\mu - r}{\gamma \sigma^2} \tag{2.4}$$

We will be trying to solve the portfolio optimization problem under the condition of non-zero transaction costs, which Merton does not consider. However, Merton's ratio will be a base case benchmark for our computational simulations.

2.5 Discrete vs. Continuous Time Representation

Although we could use the continuous time representation for the price processes of the stocks and bonds above, we will take a discretized approach to the price processes. The advantage of representing the price processes using a discrete-time representation is that it is a much more convenient representation for simulation purposes. If the investor checks their portfolio during consistent time intervals (every month), than the discrete-time representation will not affect the portfolio strategy. Because the investor will only trade at these interval time periods, the investor only needs to know the prices of the stocks and bonds at these periods.

One shortcoming of the discrete-time representation is that sometimes investors react when prices reach certain predetermined levels. Using the discrete time approach, our reaction to prices reaching these predetermined levels will be delayed, due to only trading (and checking prices) at our designated periods. This delay will be important when we discuss the boundary lines for trading in the next section. It is just useful at this point to acknowledge that this effect leads to a slightly sub-optimal portfolio optimization strategy.

Understanding its benefits and its deficiencies, we will use a discrete-time approximation for the behavior of the stock and bond. This representation is derived in a well known paper (and included in the book) by Cox, Ross, and Rubinstein. [5]

2.6 Cox, Ross, and Rubinstein's Representation

In the section of their book on discrete-time approximation, Cox, Ross, and Rubinstein [5] describe the stock price process as a standard recombining binomial tree, with a probability p of the stock price increasing by a factor of u and a probability $(1-p)$ of the stock price decreasing by a factor of d. The bond price process increases by a factor of r_f at each period. In order to define the values for this recombining binomial tree, we need a few input parameters. We need the mean return and standard deviation μ, σ from the stochastic price process for the stock Y_t, the annual interest rate r of the bond of the form: annual interest $= e^r - 1$ (from Eq. (2.1) above), and the constant time step Δt between investor transaction times (as well as stock and bond evaluation times).

To solve for the parameter values necessary for the recombining binomial tree representation (p, u, d, r_f), Cox, Ross, and Rubinstein generate a system of equations in their paper, given the parameters $\mu, \sigma, r, \Delta t$. The solution to this system yields:

$$p = \frac{1}{2} + \frac{1}{2}\frac{\bar{\mu}}{\sigma}\Delta t \tag{2.5}$$

$$u = e^{\sigma \Delta t} \tag{2.6}$$

$$d = \frac{1}{u} \tag{2.7}$$

$$r_f = e^{r\Delta t} \tag{2.8}$$

Here μ is the return parameter from the stochastic stock price process representation, and $\bar{\mu}$ is the risk neutral return parameter, which is equivalent to $\bar{\mu} = \mu - \frac{\sigma^2}{2}$. Our r_f variable is the risk-free rate that multiplies the amount in bonds at every time step. This summarizes the calculated parameters used in the model simulations, given the choice parameters $\mu, \sigma, r, \Delta t$ (defined in Eq. (2.1)).

Cox, Ross, and Rubinstein [5] also proved that in the limit as $\Delta t \rightarrow 0$, the optimal portfolio to hold in the single stock and bond case with no transaction costs approaches Merton's ratio. This is an important result as we can still use Merton's ratio as a base case benchmark, even though we will use a discrete time approach to the problem.

With the two major assumptions above, we are able to begin the derivation of the new approximate value function.

Chapter 3

Derivation of the Approximate Value Function and the Optimal Portfolio Policy

We now employ the tools of portfolio optimization to develop the model we will employ to optimize the portfolio allocation in the single stock and bond case, as well as in the higher dimensional cases.

3.1 Definition of Objective and Variables in the Single Stock and Bond Case

The basic goal of portfolio optimization is to determine whether it is beneficial to change one's portfolio composition at any point in time, and to determine the optimal allocation to which to transact. The base case example of portfolio optimization is determining the optimal allocation of one stock and one bond, given an initial allocation.

3.1.1 Definitions using Initial Allocations

Let us denote the terms that describe the initial allocation of wealth in a time period (before rebalancing) as follows:

x_t'	: Initial allocation of wealth in bonds at time t
y_t'	: Initial allocation of wealth in stock at time t
$W_t' = x_t' + y_t'$	: Initial total wealth at time t
$\pi_t' = \dfrac{y_t'}{(x_t' + y_t')}$	: Initial ratio of stocks to total wealth

We adopt the convention that all variables with a prime symbol on them represent a quantity before rebalancing occurs at that time period.

With these variables for describing the initial allocation of wealth in a time period, we consider a way to value the amount of wealth that we possess. Our tool for doing this is the utility function of section 2.2. In Chapter 2, we presented the true value equation of the investor based upon the Constant Relative Risk Aversion utility equation. We can rewrite the true value function in terms of the distribution of initial wealth (x_t', y_t') at time period t:

$$V(x_t', y_t', \pi_t') = \frac{1}{1-\gamma} * F(\pi_t') * (x_t' + y_t')^{1-\gamma} \tag{3.1}$$

Recall from section 2.3 that $F(\pi_t')$ reflects the dependence of portfolio return on portfolio composition.

3.1.2 Our Approximate Value Function

In section 2.2, we briefly discussed the idea of approximating the function $F(\pi_t')$ in the true value function Eq. (3.1). By approximating this complicated factor in the true value function, we are able to quickly and easily optimize the value function thereby determining the optimal portfolio policy. However, with the introduction of this approximation for the factor $F(\pi_t^{'})$, the approximate function is not a value function anymore. Even though the approximate value function may be different from the true value function by a constant factor, once we multiply the true value function by a constant, we can no longer call it a value function because it does not correspond to the terminal utility of wealth at time $t = T$. However, we note that multiplying the value function by a constant does not change the rankings between different levels and distributions of wealth $U(x_t^{'}, y_t^{'}, \pi_t^{'})$ for different $(x_t^{'}, y_t^{'}, \pi_t^{'})$. In other words, if one level and distribution of wealth is valued greater than another in the true value equation, than that relationship still holds in the approximated value equation. Because we only care about the portfolio optimization decision (where the maximum occurs in the function) and not the actual value outputted by the value function, we take advantage of this fact to approximate $F(\pi_t^{'})$ without worry. In the language of metric spaces, the approximate value function and the true value function are different but equivalent metrics.

The main contribution of this paper is the development and analysis of this approximated factor in substitution of $F(\pi_t^{'})$. We will approximate the factor $F(\pi_t^{'})$ by using a second-order Taylor expansion of $\ln(F)$. Writing out this approximation, we have:

$$\begin{aligned} F(\pi) &\approx e^{(f(\pi))} \text{ for function } f(\pi) = \ln(F(\pi)) \\ F(\pi) &\approx e^{(f_0 + f_1 * (\pi - \bar{\pi}) + f_2 * (\pi - \bar{\pi})^2)} \end{aligned}$$

The last equation above includes the second-order Taylor expansion of $f(X) = \ln(F(X))$. This Taylor expansion is a quadratic equation. We could take this expansion and write it in the form:

$$f_0 + f_1 * (\pi - \bar{\pi}) + f_2 * (\pi - \bar{\pi})^2 = b_0 + b_1 * \pi + b_2 * \pi^2 \qquad (3.2)$$

Notice how we only need three variables in order to define the quadratic equation from the previous representation in Eq. (3.2). Therefore, without loss of generality, we can set f_1 in Eq. (3.2) and still be able to recover the quadratic form from the three variables f_0, f_2, and $\bar{\pi}$. Substituting $f_1 = 0$ in the derivation of the approximation for $F(X)$ above, and renaming variables f_0 and f_1 we get:

$$F(\pi) \approx e^{(a_0 + a_1 * (\pi - \bar{\pi})^2)} \tag{3.3}$$

The approximation for the factor $F(X)$ in Eq. (3.3) is the main contribution of the paper. With this substitution, all that is required to find the optimal values for a_0, a_1, and $\bar{\pi}$ in order to approximate F(X) well. The true value function Eq. (3.1) transforms into:

$$V(x_t^{'}, y_t^{'}) = \frac{1}{1-\gamma}\left((x_t^{'} + y_t^{'})e^{a\left(\pi_t^{'} - \bar{\pi}\right)^2}\right)^{(1-\gamma)} \tag{3.4}$$

The new exponential factor derived in Eq. (3.1) and substituted in Eq. (3.4) is the approximation of $F(\pi_t^{'})$. The parameters of this exponential approximation are the Obstinance Factor a and the Composition Variable $\bar{\pi}$. The Composition Variable $\bar{\pi}$ is a vector whose size depends on the number of stocks, and represents the predetermined estimate of the optimal portfolio composition. $\bar{\pi}$ is motivated by Merton's ratio, since we know that without transaction costs, the solution to the optimal π is static (Merton's ratio). With the functional form of our exponential factor, we recover this behavior when transaction costs are 0. We will discuss this example more in section 5.1. The Obstinance Factor a is a matrix of coefficients whose dimensions depend upon the number of stocks and bonds composing the portfolio. This factor represents the reluctance of the portfolio composition to stray from the predetermined guess ($\bar{\pi}$). The bounds on these factors are:

$$a_i \leq 0 \text{ for all } i \tag{3.5}$$

$$0 \leq \bar{\pi}_j \leq 1 \text{ for all } j \tag{3.6}$$

Substituting the exponential factor into the approximate value function greatly simplifies finding the optimal portfolio policy. Instead of having to dynamically maximize the true value function which includes the unwieldy function $F(\pi_t^{'})$, the problem is reduced to optimizing over three variables: a_0, a_1, and $\bar{\pi}$. We must optimize over these three variables which parameterize the entire space of allowed portfolio strategies. This scenario is much more manageable that trying to solve for the true value function dynamically. In addition to simplifying the optimization of the portfolio policy, the substituted factor has other helpful characteristics. The factor is exponential, meaning that it is differentiable. This is a very important characteristic since optimizing the approximate value function may (will) require us to compute the gradient of the multi-variable function. In addition, the Obstinance Factor a allows the user to determine the weight of the Composition Variable in the utility function. A portfolio strategy that does not drift far from the predetermined estimate ($\bar{\pi}$) of the optimal portfolio composition will have a set to be highly negative to increase the importance of the Composition Variable. In higher dimensions, when there are multiple $\bar{\pi}$s, a portfolio's policy may insist on keeping some stocks at certain levels and not mind letting the levels of other stocks drift. The parameter a will reflect this by some of its elements having high negative values (for those that maintain certain levels), and others being approximately 0 (for those that drift). In essence, the parameter a determines the no transaction boundaries of the substitute value function. This is discussed later in Chapter 5.

3.1.3 Determining the Optimal Portfolio Policy

Defining the approximate value function above, we need to discuss why we need an approximate value function. What is the purpose of the function? How do we find an optimal portfolio policy? The purpose of the approximate value function that we proposed above is to allow the investor to determine the optimal portfolio allocation at any point in time. Given our current magnitude and distribution of wealth along with other parameters, by optimizing this approximate value function which incorporates transaction costs and return predictability, we will be able to determine the optimal portfolio policy at that point in time. A basic walkthrough of the portfolio optimization process is below. It enumerates the steps taken to find an investor's optimal portfolio policy.

1. Pick $a_0, a_1, \bar{\pi}$ as parameters of the approximate value function (choice of parameters determines portfolio policy)

2. At the beginning of each time step for each simulation of the portfolio policy, maximize the approximate value function with respect to variables (x_B, x_S, y_B, y_S) to determine how much stock/bond to buy/sell at the current step (determine optimal trading decision)

3. Average all terminal utility function values (utility function from section 2.2) at time $t = T_i$ for all i simulations.

4. Use Monte Carlo simulation to choose the best portfolio policy allowed by our parameterization (choose portfolio policy determined by a_0, a_1, and $\bar{\pi}$ which optimizes the average of all approximate value function values at time $t = T_i$ for all i simulations)

Choosing different values for parameters $a_0, a_1, \bar{\pi}$ is equivalent to selecting different portfolio policies. Choosing a_0 and a_1 to be small in magnitude translates to a portfolio policy that is rather indifferent if the portfolio composition drifts away from the specified $\bar{\pi}$, meaning the investor has a large no transaction area. $\bar{\pi}$ should be close to Merton's ratio as that solution is optimal in the no transactions cost case. It may not be optimal for $\bar{\pi}$ to be exactly Merton's ratio.

At each time step of each simulation we optimize the approximate value function which determines the optimal trading strategy for time t. We store the terminal values of the approximate value function at time $t = T$ for each simulation and then average the values together. Whichever portfolio policy produces the largest average terminal value is the best portfolio policy. Our parameterization of the space of possible trading strategies (possible values for a_0, a_1, and $\bar{\pi}$) is very compact and easily searchable. In Chapter 4, we discuss the computational implementation to perform the steps in the section to find the optimal portfolio policy.

3.1.4 Maximizing Final Total Wealth

We continue the analysis of the approximate value function by observing that this function represents the behavior of a risk-averse investor looking to maximize the final total wealth W_T. Because the realized returns for each time period are independent of one another, performing an optimization at each time period only requires the wealth at the beginning of the current time period. Therefore, at each time-step t, we want to maximize the approximate value equation, but not with the current wealth as functions of the parameters for x_t and y_t. We want to maximize the approximate value function with respect to the expected wealth of the *next* time period. At each period in time, we know what the expected wealth for the next time period will be because we defined the returns process for the stock and bond with the Cox, Ross, and Rubenstein representation (Section 2.6).

We seek to devise a method to redistribute our wealth between stocks and bonds in order to maximize the expected approximate value function of wealth, based on the estimated stock returns. While rebalancing the allocation of wealth, we assume that we incur transaction costs κ proportional to the amount transacted.

3.1.5 Definitions using Allocations After Rebalancing

While attempting to maximize the approximate value function of wealth, the total wealth invested in bonds and stocks may or may not change. If the initial allocation of wealth between stocks and bonds is close enough (defined by the Obstinance factor a in a way which will be made clear later) to the Composition factor $\bar{\pi}$, it may happen that no rebalancing occurs. This is because at the current portfolio composition, the transaction costs incurred from rebalancing, which are a first order factor, would be greater than the gain from transacting closer to the prespecified optimal portfolio allocation, which is a second order factor. To facilitate the process of rebalancing the portfolio, we introduce a new set of variables:

$$x_t \;:\; \text{Amount of wealth in bonds after rebalancing at time t} \tag{3.7}$$

$$y_t \;:\; \text{Amount of wealth in stock after rebalancing at time t} \tag{3.8}$$

$$W_t = x_t + y_t \;:\; \text{Total wealth after rebalancing at time t} \tag{3.9}$$

$$\pi_t = \frac{y_t}{(x_t + y_t)} \;:\; \text{Ratio of stocks in portfolio after rebalancing at time t} \tag{3.10}$$

$$\kappa \;:\; \text{Proportional transaction cost rate} \tag{3.11}$$

This completes the set of computational variables. We must discuss the relationship between the before-rebalancing variables and the after-rebalancing variables.

If there were no transaction costs associated with rebalancing the portfolio, then we would have $(x_t^{'} + y_t^{'}) = (x_t + y_t)$, since no wealth is lost in transacting from one portfolio allocation to another. However, because the transaction cost rate is κ and not necessarily 0, the equation to relate wealth before and after rebalancing becomes approximately:

$$(x_t^{'} + y_t^{'}) = (x_t + y_t - \kappa(|y_t - y_t^{'}| + |x_t - x_t^{'}|) = (x_t + y_t - 2\kappa(|y_t - y_t^{'}|) \tag{3.12}$$

Using this relation in Eq. (3.12) with the utility function in equation Eq. (3.4), we can define the utility equation post-rebalancing as:

$$\frac{1}{1-\gamma}\left((x_t + y_t - 2\kappa|y_t - y_t^{'}|)e^{a\left(\pi_t^{'} - \bar{\pi}\right)^2}\right) \tag{3.13}$$

We eliminate the π_t variable and replace it with its definiton in x_t and y_t, as well as rewrite a in terms of its matrix components (reducing the equation to only terms in x_t and y_t). We then have:

$$\frac{1}{1-\gamma}\left((x_t + y_t - 2\kappa|y_t - y_t^{'}|) * e^{\left(a_0+a_1\left(\frac{y_t}{(x_t+y_t-2\kappa|y_t-y_t^{'}|}-\bar{\pi}\right)^2\right)}\right)^{1-\gamma} \tag{3.14}$$

It is important to note that the no time step has occurred. Thus the structure of the progression is as follows:

$x_{t-1}, y_{t-1} \rightarrow$ simulate returns $\rightarrow x_t^{'}, y_t^{'} \rightarrow$ rebalance $\rightarrow x_t, y_t \rightarrow$ simulate returns $\rightarrow$ etc...

3.1.6 New and Improved Representation for Portfolio Rebalancing

Notice that after the introduction of transaction costs, the objective function contains an absolute value and is therefore no longer differentiable for all values of x_t, y_t. In addition, the technique of introducing transactions cost in Eq. (3.12) above was only approximate. However, we can use a linear optimization technique to remedy the situation. Instead of denoting (x_t, y_t) as our portfolio after transacting and denoting $2\kappa|y_t - y_t'|$ as the transactions costs, we will remove the variables (x_t, y_t) and replace them with four new variables:

$x_B \geq 0$: The amount of bonds bought during rebalancing at time t (3.15)

$x_S \geq 0$: The amount of bonds sold during rebalancing at time t (3.16)

$y_B \geq 0$: The amount of stocks bought during rebalancing at time t (3.17)

$y_S \geq 0$: The amount of stocks sold during rebalancing at time t (3.18)

Using the the utility equation Eq. (3.14), we can substitute the following relations:

$$
\begin{aligned}
x_t &= x_t^{'} + x_B - x_S && (3.19)\\
y_t &= y_t^{'} + y_B - y_S && (3.20)\\
2\kappa|y_t - y_t^{'}| &= \kappa(x_B + x_S + y_B + y_S) && (3.21)\\
x_B - x_S + y_B - y_S &= -\kappa(x_B + x_S + y_B + y_S) && (3.22)
\end{aligned}
$$

The last equation comes from the fact that after all transactions are complete, the only difference in total wealth are the costs incurred by the transactions. If the transaction costs were 0 (i.e. $\kappa = 0$), then $x_S = y_B$ and $y_S = x_B$ which implies that you can buy as much as you sell. In addition, the previous representation of transaction costs in Eq. (3.13) was only approximate. In Eq. (3.13), we simplified the transaction cost expression to $2\kappa|y_t - y_t^{'}|$. This is expression is not entirely correct in that it implies that the amount of transaction costs with respect to selling the stock or bond is equivalent to the costs associated with buying the stock or bond. The problem with the approximation in in Eq. (3.13) is that an investor must sell an asset first and then purchase a different asset with the acquired funds. Without loss of generality, assuming the investor sold bonds, he is left with $x_t^{'} - x_t - \kappa(x_t - x_t^{'}) = (1-\kappa)(x_t - x_t^{'})$ in cash, which he will then invest in stocks. The amount which he will invest is $\frac{(1-\kappa)(x_t - x_t^{'})}{(1+\kappa)}$ since the difference between this amount and $(1-\kappa)(x_t - x_t^{'})$ are the transaction costs required to buy the stock. The total amount of transaction costs incurred are:

$$
\left(\kappa(x_t - x_t^{'}) + \kappa\frac{(1-\kappa)(x_t - x_t^{'})}{(1+\kappa)}\right) < 2\kappa|x_t - x_t^{'}| \qquad (3.23)
$$

The left hand side of Eq. (3.23) produces the correct amount of transaction costs, whereas the previous approximate representation on the right hand side did not, which makes this new representation for transaction costs seem that much more efficient.

Rewriting the utility equation using the equations above, we have the differentiable function below:

$$
\begin{aligned}
U(x_B, x_S, y_B, y_S) &= \frac{1}{1-\gamma}\Bigg((x_t' + y_t' - \kappa(x_B + x_S + y_B + y_S)) \\
&\quad * \; e^{\left(a_0 + a_1\left(\frac{y_t'+y_B-y_S}{(x_t'+y_t'-\kappa(x_B+x_S+y_B+y_S))} - \bar{\pi}\right)^2\right)}\Bigg)^{1-\gamma} \qquad (3.24)
\end{aligned}
$$

All that is left is to optimize the expected value of the Eq. (3.24) with respect to the predicted returns, which we defined according to Cox, Ross, and Rubinstein in Section 2.6. We will use the discrete time recombining binomial tree stock price process discussed in the Previous Results / Assumptions chapter above.

3.1.7 Our Final Approximate Value Function

Writing this expected utility function out explicitly using the parameters p, u, d, r_f defined in the equations Eqs. (2.5) earlier, the approximate value function we want to optimize becomes:

$$
\begin{aligned}
V(x_B, x_S, y_B, y_S) &= \frac{p}{1-\gamma}\Bigg((r_f(x_t' + x_B - x_S - \kappa(x_B + x_S + y_B + y_S)) + u(y_t' + y_B - y_S)) \\
&\quad * \; e^{\left(a_0 + a_1\left(\frac{u(y_t'+y_B-y_S)}{(r_f(x_t'+x_B-x_S-\kappa(x_B+x_S+y_B+y_S))+u(y_t'+y_B-y_S))} - \bar{\pi}\right)^2\right)}\Bigg)^{1-\gamma} \\
&+ \frac{1-p}{1-\gamma}\Bigg((r_f(x_t' + x_B - x_S - \kappa(x_B + x_S + y_B + y_S)) + d(y_t' + y_B - y_S)) \\
&\quad * \; e^{\left(a_0 + a_1\left(\frac{d(y_t'+y_B-y_S)}{(r_f(x_t'+x_B-x_S-\kappa(x_B+x_S+y_B+y_S))+d(y_t'+y_B-y_S))} - \bar{\pi}\right)^2\right)}\Bigg)^{1-\gamma} \qquad (3.25)
\end{aligned}
$$

Remember that in Eq. (3.25) above, it is assumed that transaction costs required for rebalancing are paid with the amount of wealth stored in bonds.

Most of the values in Eq. (3.25) above are known. While performing the optimization at time step t, the values $x_t', y_t', \kappa, \gamma, \bar{\pi}, a_0, a_1$ are all known ($\kappa, \gamma, \bar{\pi}, a_0, a_1$ are predefined before the optimization is performed and x_t', y_t' are known from the previous time step). Therefore, the only unknown variables that are to be determined at each time step are x_B, x_S, y_B, and y_S.

After completing the approximate value function to optimize, we complete the model by weaving the time steps together. This is done by simulating the returns of the stock and bond after the optimization of the approcimate value function is completed. This translates into defining the relationship between $x_t^{'} + x_B - x_S, y_t^{'} + y_B - y_S$ and $x_{t+1}^{'}, y_{t+1}^{'}$. We defined the recombining binomial tree that defines the returns on stocks earlier (Section 2.6), but to explicitly show what values are used, we will repeat the model again quickly. The returns are as follows:

$$\begin{aligned} x_{t+1}^{'} &= (x_t^{'} + x_B - x_S)r_f, \text{ where } r_f \text{ is the return rate defined earlier} && (3.26) \\ y_{t+1}^{'} &= \Big((y_t^{'} + y_B - y_S)u\Big) \text{ with prob } p \text{ and} && (3.27) \\ y_{t+1}^{'} &= \Big((y_t^{'} + y_B - y_S)d\Big) \text{ with prob } (1-p) && (3.28) \end{aligned}$$

Nearly all aspects of the model have been covered except for one: choosing the best parameters for the approximate value function. Above, we discussed the newly created function, as well as the pre-specified values it requires. However, we must assign values to these prespecified parameters. These values are very important in that those values determine the portfolio policy at each time step. For the transaction costs rate κ and for the CRRA utility function's risk aversion factor γ, we will set the parameters to standard values. Namely, we choose:

$$\begin{aligned} \kappa &= 0.01 \\ \gamma &= 2 && (3.29) \end{aligned}$$

The only variables left to be determined are $\bar{\pi}, a_0$ and a_1. Because these last three variables were introduced in the substitute exponential factor, we do not yet have known good choices for these parameters. In fact, choosing the right values for these parameters will decide how well this approximate value function will perform in deciding the proper trades to make at each time step with the goal of maximizing our wealth at a final time T. When we have chosen proper values for these three remaining variables, we will then be able to test the portfolio trading strategy against simulated data in order to determine its efficiency and performance.

3.2 Approximate Value Function in the Two Stock and One Bond Case

The functional form of the approximate value function in higher dimensional cases is very similar to the form for the single stock and bond case. Extending the function to accomodate more stocks in the portfolio requires modfication in a few places. However, the expansion required by the approximate value function is easy to understand and simple to implement.

Initially, in the event of an increase in the stock value by a factor of u, the total wealth factor is:

$$(r_f(x_t^{'} + x_B - x_S - \kappa(x_B + x_S + y_B - y_S)) + u(y_t^{'} + y_B + y_S) \tag{3.30}$$

For a two stock and one bond case, it becomes necessary to add in a second stock initial wealth, as well as variables to buy and sell the second stock. In addition, there are four outcomes with non-zero probabilities at each time step which need to be integrated into the value function. Let us define p_1 as the probability that the first stock will increase by a factor of u_1. On the same note, the second stock will increase by a factor of u_2 with probability p_2. The four possible outcomes are that both stocks increase with value with probability $p_1 * p_2$, the first increases and the second decreases with probability $p_1 * (1 - p_2)$, the second increases and the first decreases with probability $(1-p_1)*p_2$, or both stocks decrease with probability $(1-p_1)*(1-p_2)$.

For the case that both stocks increase, the total wealth factor is:

$$(r_f(x_t^{'}+x_B-x_S-\kappa(x_B+x_S+y_{1_B}+y_{1_S}+y_{2_B}+y_{2_S}))+u_1(y_{1_t}^{'}+y_{1_B}+y_{1_S})+u_2(y_{2_t}^{'}+y_{2_B}+y_{2_S}) \tag{3.31}$$

Not only does the total wealth factor change, but so does the Preference factor. The required changes are predictable though as the Preference factor, in the case where the stock increases in price, changes from:

$$e^{\left(a_0+a_1\left(\frac{u(y_t^{'}+y_B-y_S)}{(r_f(x_t^{'}+x_B-x_S-\kappa(x_B+x_S+y_B+y_S))+u(y_t^{'}+y_B-y_S))}-\bar{\pi}\right)^2\right)} \tag{3.32}$$

to the following form below which incorporates both stocks and their respective $\bar{\pi}$'s:

$$e^{\left(a_0+a_1\left(\frac{u_1(y_{1t}^{'}+y_{1B}-y_{1S})}{E(W_{t+1})}-\bar{\pi}_1\right)^2+a_2\left(\frac{u_2(y_{2t}^{'}+y_{2B}-y_{2S})}{E(W_{t+1})}-\bar{\pi}_2\right)^2+a_3\left(\frac{u_1(y_{1t}^{'}+y_{1B}-y_{1S})}{E(W_{t+1})}-\bar{\pi}_1\right)\left(\frac{u_1(y_{2t}^{'}+y_{2B}-y_{2S})}{E(W_{t+1})}-\bar{\pi}_2\right)\right)} \tag{3.33}$$

where $E(W_{t+1})$ is the wealth factor when both stocks increase in price, as defined in Eq. (3.31) above.

As the number of stocks in the market (reflecting the dimensionality of the problem) increases, the changes made to the value function follow as above. In the Preference factor, there is an *a* matrix coefficient factor for each diagonal term (squared term) and for every cross term (2*... term). As the dimensionality of the problem grows, the number of unknown parameters for the value function grows rapidly. Therefore, a method, more efficient than the one described in the next chapter, of searching for the optimal parameters of the value function must be found for the value function to scale well to higher dimensional parameters. This would be a valuable extension to this paper and is discussed later.

In the next chapter, we will describe the computational implementation of the portfolio optimization routine which centers around the optimization of the value function at every time step. In addition, we will describe the method by which we will discover numerically, through simulations, the ideal values for $\bar{\pi}, a_0$ and a_1, which give us the best portfolio policy. By finding the ideal values for $\bar{\pi}, a_0$ and a_1 through simulation, we will have achieved closure for the approximate value function, defining the optimal portfolio policy at any point in time.

Chapter 4

Computational Implementation of Portfolio Optimization Problem

In the previous section, we discussed the theory and mathematics behind the approximate value function which we will utilize in order to determine an optimal portfolio policy. However, one problem that was left unsolved was determining optimal values for the parameters $\bar{\pi}, a_0$ and a_1 in the function. In this section, we will discuss the computational implementation of the portfolio optimization problem, the method by which we will discover the optimal values for $\bar{\pi}, a_0$ and a_1, and the methodology for testing the performance of the approximate value function.

4.1 MATLAB Platform

The MATLAB platform is utilized for the computational implementation of the portfolio optimization problem. There are several reasons behind this choice. A high level language is required in order to organize the code into several classes, each of which has its own specific and important function. In addition, in order to quickly perform the simulations required of the portfolio optimization problem, a numerically-based optimizer is preferred over a slower but exact symbolic algebraically-based optimizer like Mathematica. Taking into account the requirements of the problem, we choose to implement the code for the portfolio optimization problem in MATLAB.

4.2 CreateData.m : Creating the Test Data via User Preferences

As the first step in the implementation of the optimization problem, we must discuss how we will generate the test data, as well as the values for the parameters pre-specified by the user, which will be used by the portfolio optimization code. All of this data is generated/specified by the MATLAB file createData.m attached in Appendix A. Here we assign the pre-specified values that we developed earlier in the paper (values for $\gamma, \kappa, \mu, \bar{\mu}, \sigma, \Delta t, r_f, u, d, p$ in Eq. (3.29) and section 2.6). We specify the initial distribution of wealth for all simulations, which we set to $\pi_0 = .5$ $(x_0 = 100, y_0 = 100)$ as a typical initial distribution.

For a single portfolio optimization iteration, we need to specify the length of time an investor will be in the market. As an investor may pull out at any time, this generated number should be somewhat random. An accurate representation for this period is the stopping time of a Poisson distribution. This representation is equivalent to the values generated by an exponential distribution, which are the lengths of the inter-arrival times, given a parameter λ. We denote $\Delta t = \frac{1}{12}$, so that the time-steps are monthly, and so we choose the parameter $\lambda = 48$ (fthe years) for the exponential distribution (which we call simLength in the code). This is historically a reasonable amount of time for the investor to be in the market. However, the investor may choose to leave the market earlier or later and the approximate value function must perform well for both shorter term and longer term investors. Therefore, we stipulate that during the performance test of the function (with assigned values for the parameters $\bar{\pi}, a_0$ and a_1), it will perform a very large number of simulations using different simulation lengths chosen from the exponential distribution (num_runs $= 10,000$). In chapter 5, we will prove that this number is sufficient to provide us with the true mean of the final approximate value function outputs given the exponential distribution of simulation lengths. This array of different simulation lengths is represented by the variable sample_sizes.

All that is left to generate is the returns table for the stock. For each of the simulation runs, we generate an array of stock returns based on the given probability p of the stock increasing (called return_matrix). We generate a different array for each simulation run, so we set a length for the table (tableLength = 500) and make the table (10,000 x 500). If a simulation length generated is greater than 500, the table wraps around to the first cell of the row and repeats from there.

In order to accurately compare the performance of the function with different values for its parameters ($\bar{\pi}, a_0$ and a_1), we would like to use the same generated data for each of the simulations in order to avoid discrepancies in the test data. This is made possible by saving all of the generated values to a temp file (called "newtemp") and loading this file when data is needed to test the function with parameter values for $\bar{\pi}, a_0$ and a_1. Therefore, once createData.m is run once, there is no need to run it again, unless the user would like to change one of the user-specified parameters.

4.3 Objgradnew.m : The Approximate Value Function

the defined approximate value function is stored in this file. MATLAB's fmincon optimizer routine calls this function when we are optimizing the portfolio allocation at each time step for each simulation iteration in MEngCode.m. Even though the function above is differentiable, computing the gradient is not an easy task and is very complex (See Appendix B for Mathematica-computed symbolic derivatives). Therefore, we do not input a user-defined gradient for MATLAB to utilize. Instead we allow MATLAB's fmincon function take care of the optimization of the function.

There are two adjustments to the approximate value function in this code to point out. First, the function is multiplied by a factor of (-1) and second, it is multiplied by a factor of $10,000$. There are two simple reasons behind these adjustments. The first adjustment is necessary because we are utilizing MATLAB's fmincon function, which minimizes the value of the function inputted. Since we are trying to maximize the output value of the function, this is equivalent to minimizing: (-1)*output value of the value function, which is what we do in this case. With respect to the second adjustment, with the pre-specified value of $\gamma = 2$, the output value of the function becomes a very small fraction. MATLAB's fmincon function optimizes the input function (the function) through adjusting the variable parameters (x_B, x_S, y_B, y_S) by a value proportional to the value of the input function with its current variable parameter values. Because the output value of the approximate value function is a very small fraction in this case, MATLAB's fmincon function progresses towards the optimal solution in very small steps. In order to increase this step-size and therefore decrease the time necessary to maximize the function, we multiply the function by a large factor (on the order of 10,000).

4.4 MEngCode.m : The Portfolio Optimization Simulator

The backbone of the MATLAB code is the file MEngCode.m. This code performs the optimization simulations based upon the parameters generated from CreateData.m. The code optimizes the portfolio allocation at each time step for each simulation iteration by using MATLAB's fmincon function and calling the function Objgradnew.m described above. In the process of performing the portfolio optimization at each time step, we impose two constraints in the code to ensure no degenerate solutions are chosen for the portfolio. First, we limit the amount that we are allowed to short and go long for both stocks and bonds (i.e. we model a risk averse investor). In addition, if the optimization protocol calls for transactions that are minuscule in size (ie. 10^{-6}), we decide not to transact at all to avoid MATLAB rounding errors. The final π and final approximate value function value is stored after each simulation. After all 10,000 simulations are completed, the code averages these values over all simulations and outputs these results to the user or the function that called MEngCode.m (namely pi_a_solve.m).

4.5 Pi_a_solve.m and Pi_a_fmincon: Approximate Value Function Parameter Optimizer

As we have discussed, the goal of the MATLAB code is to test the performance of the function using simulations and to determine the best values for parameters $\bar{\pi}, a_0$ and a_1 of the optimization. The MATLAB codes pi_a_solve.m and pi_a_fmincon are designed to do both of these things. In pi_a_fmincon, the optimization of the function's parameters $\bar{\pi}, a_0$ and a_1 are left to the fmincon optimizer of MATLAB. However, MATLAB's fmincon was not designed to handle this particular situation. Instead of optimizing a function which may have a gradient, or can at least have its gradient approximated by MATLAB's finite difference method, we are optimizing a number which is the average of the final values of 10,000 different simulations. As there are three variables to simultaneously optimize, MATLAB's fmincon method is unable to find values that are even close to the optimal values for the function's parameters.

The only choice left is to search for the solutions manually. The method for searching for the optimal values of the approximate value function's parameters is described and justified in Chapter 5. In the code pi_a_solve.m, we are able to manually select values for $\bar{\pi}, a_0, a_1$ and run the optimization simulations on the parameter-specified function. This file collects the averages of the final function values, the final π of the portfolio, and the average total final wealth and stores them in tables. This allows the user to not only determine which values allowed the function to perform the best, but what the values of the average final portfolio composition were as well as the average final wealth of the portfolio.

4.6 Plot_objfungrad: Plot Approximate Value Function Under Different Trade Choices

For the convenience of the user and in order to create understandable plots of the function under different conditions, the MATLAB file plot_objfungrad.m was created. In this standalone file, all user modifiable data is listed, which include parameters from createData.m and the three unknown global parameters of the approximate value function $(\bar{\pi}, a_0, a_1)$. After the user designates values for all listed parameters, the file plots the function under all possible trading choices (all possible values of x_B, x_S, y_B, y_S). This allows the user to observe graphically how different trading choices will affect the output value of the function. In this single instance, the user can see graphically which trading choice is optimal at this single time-step with these specifically designated parameters. In the MEngCode.m file, instead of having to try all possible trading choices in order to find the optimal trading choice, MATLAB's fmincon function finds this solution nearly instantaneously. This optimization needs to be performed quickly as well, since MATLAB on average will need to find this optimal trading choice a large number of times:

$$\begin{aligned} \text{Total Number of Simulations} &= (\#\ \text{ of different parameter choices in pi\_a\_ solve}) \\ &* (10{,}000\ \text{ simulations per choice}) \\ &* (\text{avg }\ 48\ \text{ timesteps / simulation}) \end{aligned}$$

With the necessary MATLAB simulation code explained and implemented, we are now able to discuss the resulting data from testing as well as discuss the meaning of these results. In Chapter 5, we look at the results from the single stock and bond case, as well as get a glimpse of the results from running the simulation on higher-order situations, namely the two stock, single bond scenario.

Chapter 5

Simulation Results and Analysis

After deriving the new approximate value function and creating the necessary MATLAB simulation code, we are ready to test the performance of the function in simulations. Before using the derived approximate value function to determine the optimal allocation at every time step, we will use a base case function instead, whose optimal asset allocation has a closed form solution, in order to test the MATLAB code to make sure it is operating correctly. To put the portfolio optimization's performance into perspective, we will compare the results of this optimization against a benchmark. In addition, we will be able to discover the optimal parameters for the approximate value function $(a_0, a_1, \bar{\pi})$ through numerical simulation.

5.1 Single Stock and Bond No Transaction Costs Case

Before testing the performance of the derived approximate value function using the MATLAB simulation code, it is important to discern whether or not the MATLAB portfolio optimization code is operating properly. The most efficient way to test this is to run simulations on base cases for which we know the correct solutions. The no-transaction costs case for the single stock and bond scenario is a great base case to use. We already know that the optimal portfolio composition at every time step is static and is equal to Merton's ratio, which we discussed earlier in section 2.4.

In order to adjust the function to emulate the value function Merton employed, we simply need to eliminate transaction costs and the exponential factor for the user-recommended portfolio composition. These conditions are satisfied by setting the parameters $\kappa, a_0, a_1 = 0$. With these parameters, the transaction costs are nullified and the exponential Preference Factor $= e^0 = 1$, and no longer affects the approximate value function either. This is easily done by setting $\kappa = 0$ in the CreateData.m file and setting $a_0, a_1 = 0$ in the pi_a_solve.m file. Another way we can emulate Merton's strategy with the approximate value function is to set a_1 to be a highly negative number and set $\bar{\pi} =$ Merton's ratio. Because transaction costs are 0 and the Obstinance factor a is strongly negative, the optimal portfolio would natuarally be $\pi = \bar{\pi}$ which is Merton's ratio. Using either of the two methods mentioned produces the same optimal solution: Merton's ratio.

All that remains is to pick parameters μ, σ, r_f, γ, run the simulations using the computational implementation, and compute Merton's ratio using the formula in Eq. (2.4):

$$\pi = \frac{\mu - r_f}{\gamma\sigma^2}$$

In fact, running the simulations in MATLAB is unnecessary. All one needs to do is plot one optimization simulation of the current value function to discover the simulation's choice for π. The reason for that is with no transaction costs, the initial portfolio composition does not matter in determining the optimal portfolio to hold for the next time step. After the simulated returns at each time step, the portfolio composition and the magnitude of wealth changes, but the optimal portfolio to hold for next period does not depend on the current portfolio or the magnitude of wealth. Therefore, whatever distribution of wealth is optimal to hold in one period, that same distribution will be optimal to hold for all periods.[1]

[1]One can verify this by allowing the MATLAB code to output the values (tempx, tempy) in MEngCode.m at every time step and compute $\pi = \frac{tempy}{tempx+tempy}$ to see that π stays the same after each time step.

Fig. (5-1) is a plot in the case where $\mu = 10\%, \sigma = 20\%, r_f = 5\%, \gamma = 2$, which are all fairly standard values for the parameters.

Note: In all two-dimensional graphs, the x-axis representation is as follows: for $x = (-100, 0)$, the portfolio's change is to sell x dollars worth of stocks, and buy $\frac{x}{1+\kappa}$ worth of bonds. For $x = (0, 100)$, the portfolio's change is to sell x dollars worth of bonds, and buy $\frac{x}{1+\kappa}$ worth of stocks. This is a great way to represent the single stock and bond case graphically since the plots are continuous and are easy to interpret visually. In addition, this covers all possible inputs (combinations of x_B, x_S, y_B, y_S) for the approximate value function. Unfortunately, this representation will not translate well to a higher dimensional case.

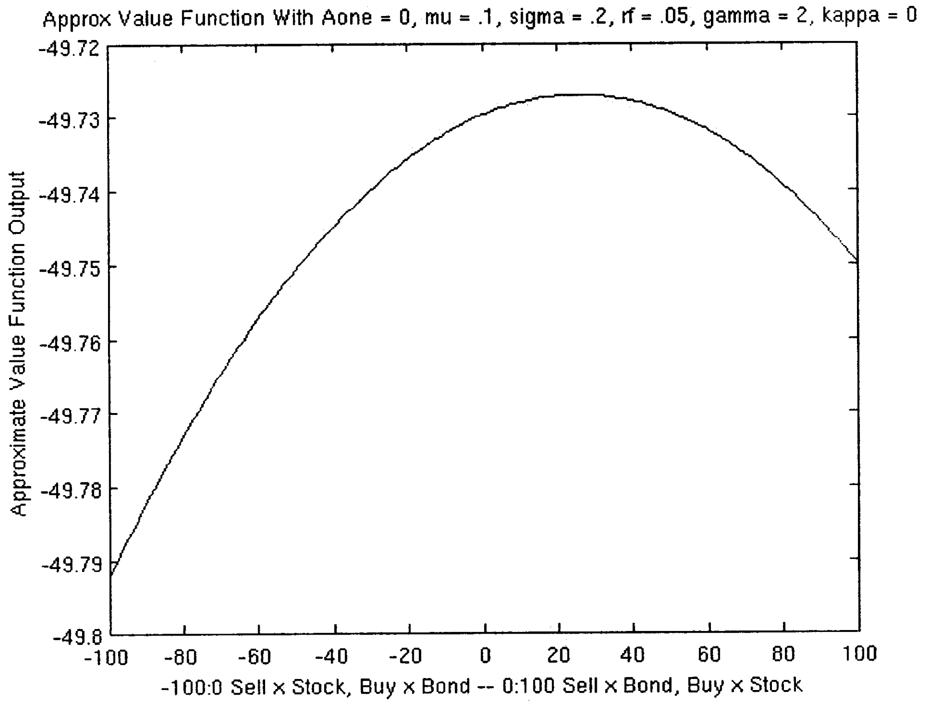

Figure 5-1: Merton Ratio Base Case: One Stock, One Bond Plot with no Transaction Costs with $\mu = 10\%, \sigma = 20\%, r_f = 5\%, \gamma = 2$

According to both the plot and the MATLAB output, the maximum on the graph occurs at $x = 27$. This is equivalent to buying \$27 worth of stocks and selling \$27 worth of bonds. Our calculated $\pi = \frac{100+27}{200} = .635$

Calculating Merton's ratio, we have: $\frac{.1-.05}{2(.2)^2} = \frac{5}{8} = .625$, which translates to holding 62.5% of our wealth in stocks and the rest in bonds. Our π and Merton's ratio are

Δ, π	.64	.635	.63	.625	.62
$\frac{1}{12}$	-49.7270737	-49.7270683	-49.7270711	-49.7270822	-49.7271016
$\frac{1}{100000}$	-49.999967189	-49.999967188	-49.9999671875	-49.999967188	-49.999967190

Table 5.1: Above is the table of values for the output of the function, varying the Δt parameter. We can see that the optimal value of the function is approaching Merton's ratio as Δt is decreased.

not equivalent. The reason for this discrepency is that Merton's ratio is based upon a continuous case whereas the approximate value function and returns process is based upon the discrete case. We can see that the solution approaches Merton's solution in the limit as $\Delta t \to 0$. Currently, it is set to $\Delta t = \frac{1}{12}$, meaning we re-evaluate the portfolio after returns occur monthly. If we adjust this parameter so that it approaches 0 and re-evaluate the function (which is shown in Table (5.1) with $\Delta t = \frac{1}{100000}$), we can see that the optimal portfolio does indeed approach the Merton Ratio. Below is a table of the function's outputs based on the value for Δt and the input x from the x-axis values (ie. $\pi = .64$ means $x = (.64 * 200) - 100 = 28$ so we evaluate the approximate value function at $x = 28$).

As further verification, Fig. (5-2) is another plot of the approximate value function using different parameters. In this case, we will use $\mu = 15\%, \sigma = 30\%, r_f = 5\%, \gamma = 3$. These parameters correspond to an investor who is more risk averse than the previous case, and a stock that has a higher return, but is more volatile than the previous case. Therefore, we should expect to see a lower optimal ratio: $\frac{.15-.05}{3(.3)^2} = \frac{10}{27} = .37$. As you can see in the caption, the discrete case approximates the continuous solution well.

We have convinced ourselves through rigourous testing that the optimization code works properly using a base case. We can now test the derived approximate value function with confidence. In addition, we will introduce transaction costs to the testing.

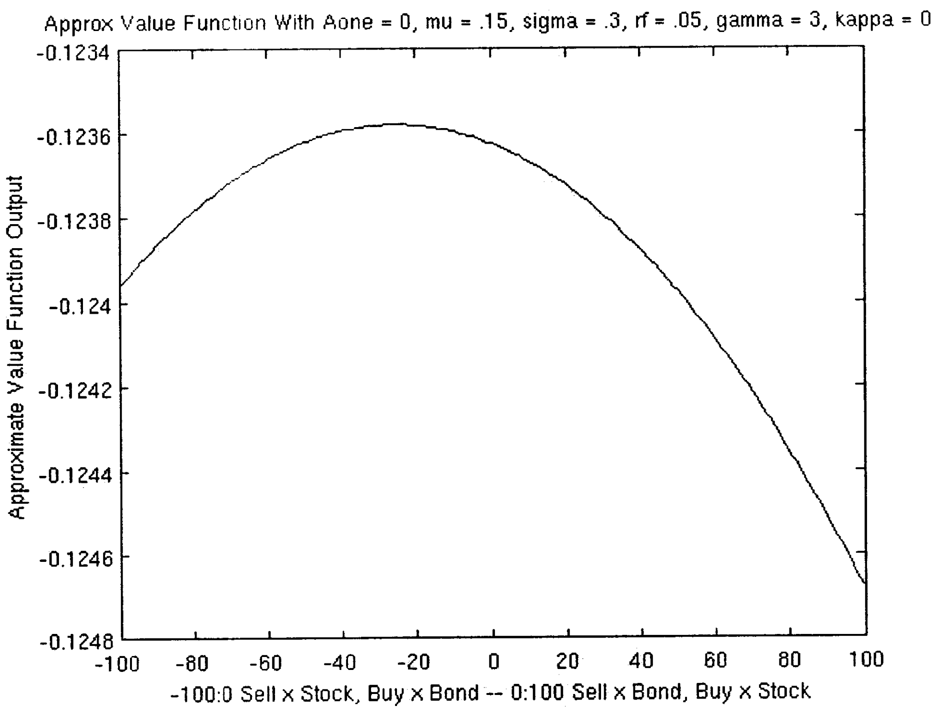

Figure 5-2: The output from MATLAB determines that the maximum in the graph above is at $x = -25$. This means the optimal π is $\frac{100-25}{200} = .375$. This result is very close to Merton's ratio which we calculated above the graph to be .37. As before, reducing the size of Δt makes these two values converge.

5.2 Single Stock and Bond Portfolio With Transaction Costs

Having established that the base cases of the approximate value function and optimization routines in MATLAB are behaving correctly, we may test the function on a more realistic case: including transaction costs. In this case, we have two extra factors and their impacts to the approximate value function to consider: the cost of rebalancing (transaction costs), and the cost of not exactly rebalancing to the suggested portfolio (exponential factor). It seems as though the balance between these factors is unstable. If one cost is set too high, that factor will dominate the other factor and the dynamic component of the portfolio will be eliminated. For example, setting the parameters $\kappa = 0.01$ and $a_1 = -5$, the portfolio will rebalance at every time step to equate its composition to that of the user-specified $\bar{\pi}$. On the other hand, if κ is set too high, the portfolio will never rebalance at any of the time steps and its composition will be decided by the returns of the stock. The first behavior is equivalent to maintaining a static portfolio whereas the second behavior is equivalent to the investor being static.

Fig. (5-3) and Fig. (5-4) each exhibits one of the two possible behaviors above, based upon the user-specified parameters. Fig. (5-3) is the example of having $\kappa = 0.01$ and $a_1 = -5$ with the rest of the parameters having default values (we will arbitrarily pick $\bar{\pi} = .8$). Fig. (5-4) is the example with $\kappa = .2$ and $a_1 = -.05$. Both plots help express the aforementioned behavior perfectly.

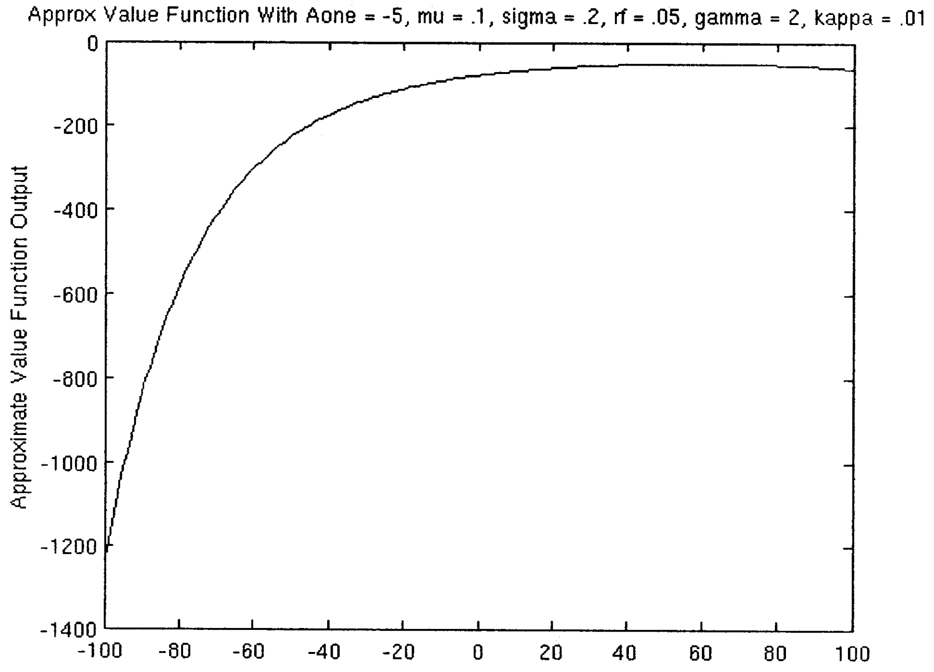

Figure 5-3: The output from MATLAB is exactly what we expect it to be: the maximum occurs at $x = 60$ This translates to $\pi = \frac{100+60}{200} = .8$.

In this one stock and bond scenario with transaction costs, we realize that a static portfolio is not necessarily the optimal investment strategy. At some trading periods, it may be optimal to trade and at others it may be optimal not to trade, depending on which of the two factors mentioned above has more weight. Ideally, we could just use MATLAB's optimization routine to optimize the choice of the global variables

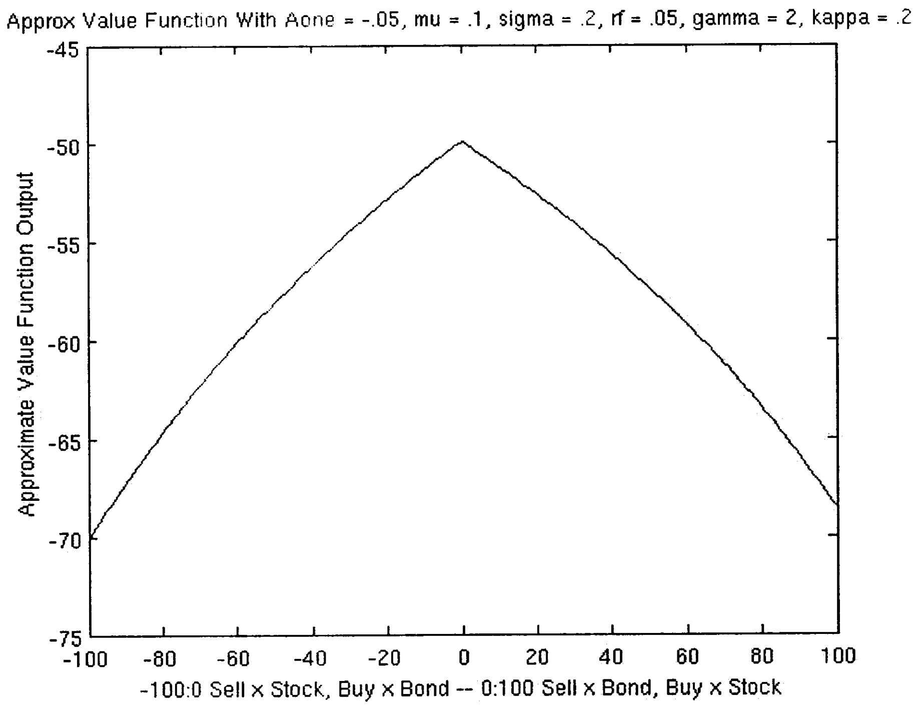

Figure 5-4: Again, the output from Matlab is exactly what we expect it to be: the maximum of the function occurs at $x = 0$ This translates to $\pi = \frac{100+0}{200} = .5$

a_0, a_1, and $\bar{\pi}$. We mentioned before in section 4.5 that this was not possible. This is because MATLAB's fmincon function does not converge to the correct solution. The reason for MATLAB's inability to converge to the correct solution is because of the finite-difference method MATLAB uses to search for an optimal solution when an approximate gradient cannot be computed (it is impossible to compute a gradient for an objective function that is based upon the average of 10,000 simulations). Therefore, a method for finding the optimal choice parameters must be devised.

In finding the optimal parameters for the approximate value function, it is impossible to optimize over more than one variable at a time. If one adjusts two variables at the same time while performing a search, when the objective function value changes for the better, it is impossible to discern which change to which variable was responsible. Therefore, maximizing one variable at a time is the only logical way to proceed with optimizing the function's parameters a_0, a_1, and $\bar{\pi}$.

In order to optimize the choice variables, we will optimize the $\bar{\pi}$ choice variable first, and then optimize the a_1 choice variable. There is no need to use simulations to find the optimal choice for a_0 as it is possible to discern graphically and through reasoning. Letting a_0 be a non-zero value is the same as multiplying the approximate value function by a factor of e^{a_0}. Having $a_0 \neq 0$ will reduce the effect of the $e^{a_1(\dots)}$ factor and decrease the entire exponential factor in magnitude. This will decrease the output value of the function. Since we are trying to maximize the function and want the $e^{a_1(\dots)}$ factor to be effective but not too small of a fraction, it is optimal to have $a_0 = 0$, since the constraint on $a_0 \leq 0$ is in effect from the approximate value function's design section. Fig. (5-5) and Fig. (5-6) illustrate varying the values of a_0 keeping all other values the same as before (setting $a_1 = -.2$ and $\bar{\pi} = .8$). Fig. (5-5) has $a_0 = -2$, a large negative value. Fig. (5-6) has $a_0 = -.02$. The effect of a_0 is easy to see between the two graphs and it is easy to tell graphically that the optimal value for a_0 is 0.

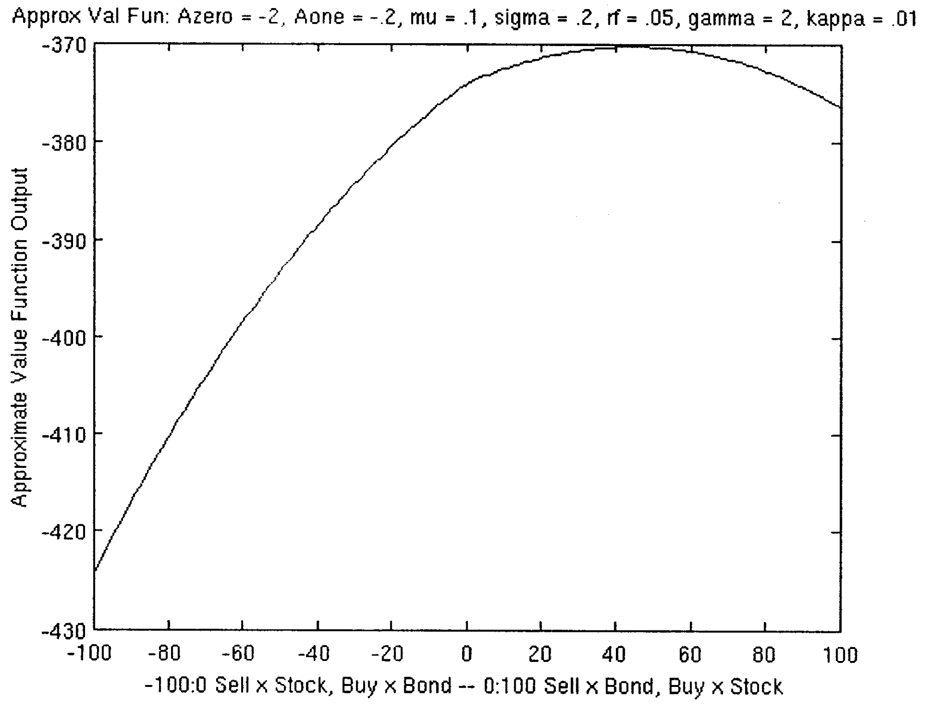

Figure 5-5: Plot of Approximate Value Function under condition that $a_0 = -2$

The optimal method for optimizing the approximate value function parameters $(a_1, \bar{\pi})$ depends upon the ability to narrow down the search space as much as possible.

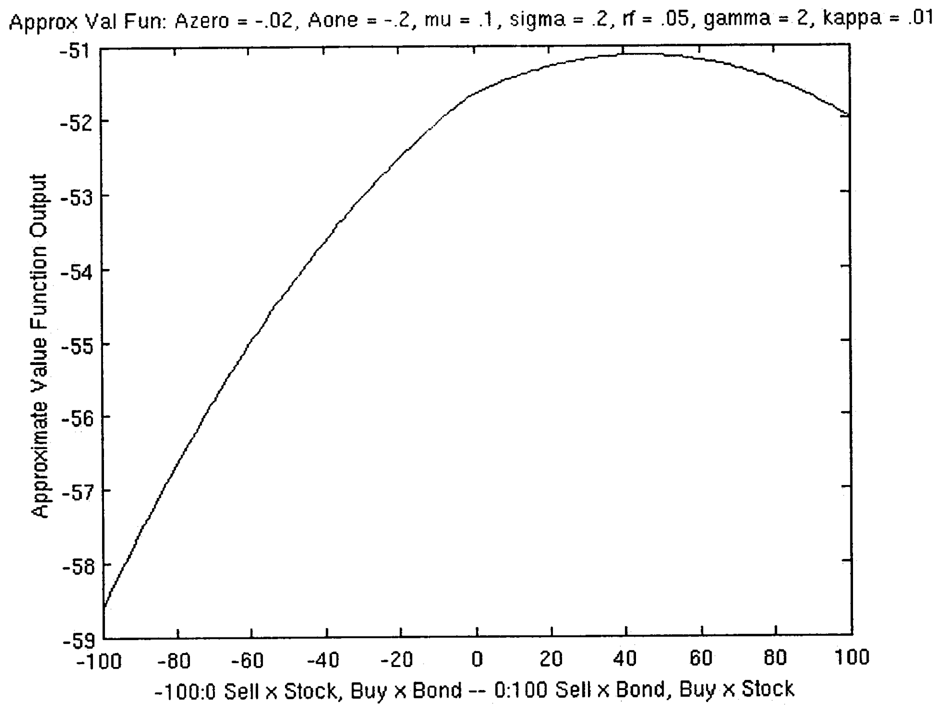

Figure 5-6: As we can see from Fig. (5-5) and this figure, the maximum value of both graphs occurs at the exact same value of x (MATLAB's output states optimal value in both graphs occurs at $x = 145, \pi = .72$). However, the difference in the scaling of the approximate value function's output is enormous. Since we are looking to maximize the function's output, we can let $a_0 = 0$ knowing that the approximate value function will return the same answer for the optimal rebalancing (since the function's output is just scaled).

When we optimize the choice for $\bar{\pi}$, we will choose a large negative value for a_1 so that the problem temporarily becomes a static portfolio choice problem. As stated before, the optimal static portfolio allocation in the no transaction costs case (and without the exponential factor) is the Merton ratio. Therefore, the optimal choice for $\bar{\pi}$ in this static portfolio case, will be close to the Merton ratio.

Fig. (5-7) is a plot, shown with its associated data in Table (5.2), of the average final function outputs over 1,000 simulations versus the choice for $\bar{\pi}$, having a_1 be a large negative value. The parameters chosen in this case are $a_0 = 0, a_1 = -1, \gamma = 2, x_0 = 100, y_0 = 100, \Delta t = \frac{1}{12}, \kappa = 0.01, \mu = .1, \sigma = .2, r_f = .05, \lambda = 48$. This simulation provides us with a view over a wide range of portfolio compositions so we know where to concentrate the search.

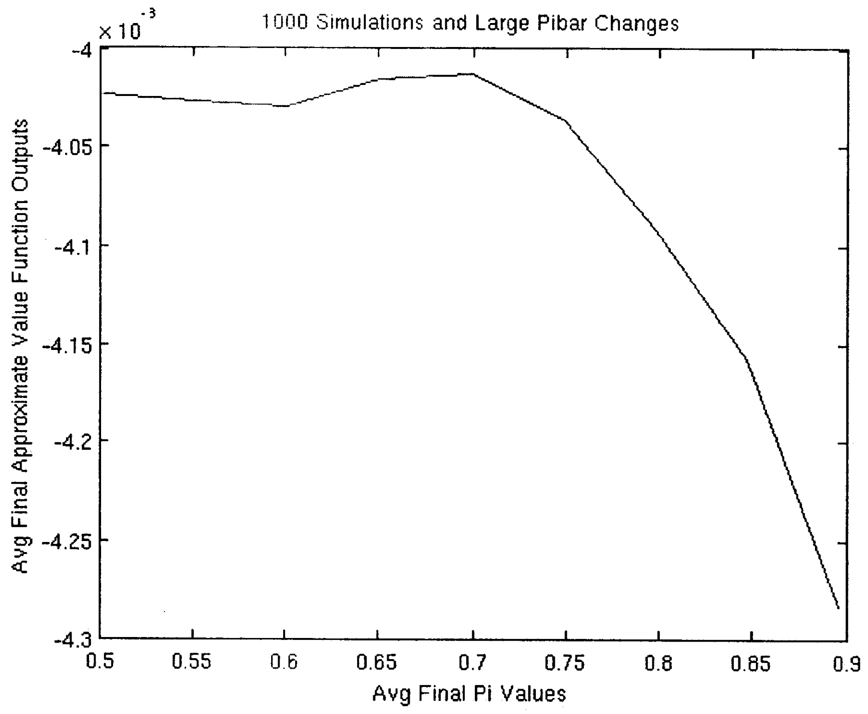

Figure 5-7: Plot of the average final approximate value function outputs with 1000 simulations and a_1 is a large negative value for value function ($a_1 = -1$)

$\bar{\pi}$	.85	.80	.75	.70	.65	.60
Average π	.8471	.7982	.7492	.6999	.6504	.6010
Avg Value	-	-	-	-	-	-
Fun Result	41.5758	40.9179	40.3620	40.1250	40.1529	40.2702

Table 5.2: Table of Values of the Average Approximate Final Value Function Outputs with 1000 Simulations and a_1 is a Large Negative Value for Function ($a_1 = -1$)

Note: Even though we have a large enough negative value for a_1, the average π values are not equivalent to the corresponding $\bar{\pi}$. This is because after rebalancing the portfolio at time $T-1$, we simulate the returns to get the portfolio value at time T, and then exit the market. We do not rebalance the portfolio at time T. Therefore, after the last return simulation, the values for π will be slightly different than $\bar{\pi}$. For the rest of the paper, $\bar{\pi}$ will no longer be included in the data tables if rounding the average π values gives us the value of $\bar{\pi}$.

From the simulation above, we can see that the maximal average value for the approximate value function occurs when the value for $\bar{\pi}$ is somewhere in the range of .70 to .55. We should not narrow the area down too much since 1,000 simulations is small enough for the data to still have some significant variance in it.

With the slightly narrowed range, we run the simulation again only changing the number of simulations from 1,000 to 10,000 in order to get a more accurate picture of what the true mean of the final approximate value function outputs is over all simulations. This simulation generates Fig. (5-8) and its corresponding data in Table (5.3):

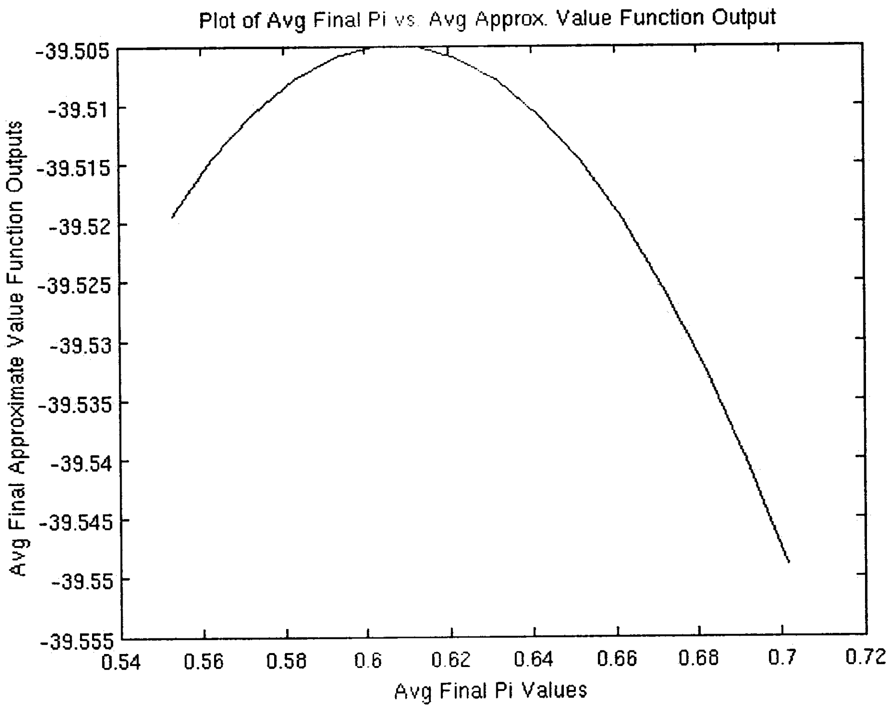

Figure 5-8: Plot of the average final value approximate function outputs with 10000 simulations and a_1 is a large negative value for function $(a_1 = -1)$

Graphically in Fig. (5-8) (and in its data in Table (5.3)), we can see that the optimal choice for $\bar{\pi}$ is $.60 \leq \bar{\pi} \leq .61$. This result makes a lot of sense. We know that the optimal static portfolio should be close to the Merton ratio because κ is relatively small. We also know that the optimal $\bar{\pi}$ will be less than the Merton ratio because

Average π	.6914	.6815	.6717	.6618	.6519	.6420	.6321
cont'd	.6223	.6124	.6022	.5926	.5827	.5728	.5629
Avg Value Fun Result	-39.54015	-39.53233	-39.52549	-39.51963	-39.51479	-39.51091	-39.50800
cont'd	-39.50604	-39.50504	-39.50502	-39.50599	-39.50791	-39.51078	-39.51460

Table 5.3: Table of values of the average final approximate value function outputs with 10000 simulations and a_1 is a large negative value for function ($a_1 = -1$)

calculating the Merton ratio using the above parameters, we get .625. This number is above .5 which is the initial $\bar{\pi}$ from the beginning of the simulation, due to the choice parameters ($x_0 = 100, y_0 = 100$) Therefore, the $\bar{\pi}$ should be close to the Merton ratio, but, because of the transaction costs, be slightly below the Merton ratio.

It is unclear, at this point, what the actual reason is as to why the optimal $\bar{\pi}$ is equal to this particular value. In other words, why does the optimal $\bar{\pi} = .02$ less than the Merton ratio? Why is this magnitude of difference not more or less? The reason is that at this particular value of $\bar{\pi}$, the marginal utility of transacting closer to the optimal portfolio (Merton's ratio) is less than the marginal utility of having κ extra wealth at the current $\bar{\pi}$ (stay and do not incur transaction costs).

We have solved the static portfolio problem with transaction costs. Using this solution, we must look to solve for the optimal value of a_1, knowing that the optimal $\bar{\pi}$ may change a little in the process. First, we should ask ourselves "how does changing the magnitude of a_1 affect the MATLAB code's optimization at each timestep?" a_1 determines the importance of the current portfolio composition versus the pre-specified ideal composition. If the magnitude of a_1 decreases, the importance of how far we are from the ideal composition also decreases, which means that the importance of incurring transcations cost increases. Therefore, by decreasing a_1, we are increasing the size of the no-transaction area.

We know that a no-transaction area exists in the portfolio optimization model. It was discussed in the section 1.2, but the idea also makes logical sense. If the current portfolio composition is close enough to the pre-specified composition, then the cost of transacting to get even closer to the pre-specified ideal portfolio is greater than the utility gain of being closer to the ideal portfolio. In this situation, the

$-\kappa(x_B + x_S + y_B + y_S)$ term has a larger marginal effect than the exponential term. As the magnitude of a_1 decreases, the exponential term behaves like the following (consider $a_{1_{orig}} = -1, a_{1_{new}} = -.08$):

$$\begin{aligned} a_{1_{orig}} &< a_{1_{new}} \\ (a_0 + a_{1_{orig}} \cdots) &< (a_0 + a_{1_{new}} \cdots) \\ e^{(a_0 + a_{1_{orig}} \cdots)} &< e^{(a_0 + a_{1_{new}} \cdots)} \end{aligned} \tag{5.1}$$

As we can see, the new exponential factor is greater than the old (with the same $\bar{\pi}$ used). In addition, if one were to compute the derivatives of the exponential factor using arbitrary values for $a_{1_{orig}}$ and $a_{1_{new}}$ (both terms in the last inequality), one would see that the derivative value of the new factor is less than the derivative value of the original factor. Therefore, we can say that the new factor's importance (exponential factor with $a_{1_{new}}$) in the approximate value function decreases, which means that there is a larger area where the marginal cost of the transaction costs factor outweighs the marginal cost of the exponential factor (the no-transaction region increases).

With a larger no-transaction region, the range of portfolios which we are allowed to hold increases. This means that the optimal portfolio is dynamic and no longer static. In the Merton case, the portfolio was allowed to be dynamic as well. However, the optimal solution to Merton's case happened to be a static portfolio. Although we know the optimal portfolio may no longer be static, the average of the optimal portfolio choice across all time periods and all simulations in the dynamic case should still be close to the static case. We can think of the choice for $\bar{\pi}$ to be the midpoint between two boundary lines where the portfolio composition lies after every rebalancing. Whenever the portfolio composition leaves the boundary lines, we transact back to the closest boundary. On the other hand, whenever the composition is within the boundary lines, we do not transact. Therefore, the composition moves between the boundary lines depending on the return of the stock over simulated time periods.

Average $\bar{\pi}$	.64178	.63194	.62209	.61223	.60237	.59250	.58263
Avg Value Fun Result	-39.47268	-39.47010	-39.46848	-39.46782	-39.46813	-39.46941	-39.47164

Table 5.4: Table of values of the average final approximate value function outputs with 10000 simulations and a_1 is a smaller negative value for function ($a_1 = -.5$)

If the stock return is u, then the composition moves towards the upper boundary whereas if the stock return is d, the composition moves towards the lower boundary. Over a large number of simulations and time periods for each simulation, the mean will be close to this $\bar{\pi}$ value. Therefore, the optimal choice for $\bar{\pi}$, during the optimization of the choice for a_1, may change a little. but should still be close to the above static problem solution.

If we needed, we could plot the boundary lines formed by the choices of $\bar{\pi}$ and a_1. To do so theoretically would mean to take the derivative of the approximate value function given the choices for $\bar{\pi}$ and a_1 and set the derivative equal to 0 under two circumstances: when we are given a high initial π distribution of wealth (all money in stocks and none in bonds before rebalancing), and when we are given a low initial π distribution of wealth (all money in bonds and none in stocks). This would give upper and lower boundary points under a certain magnitude of wealth. In order to draw lines, we must plot these values under different magnitudes of wealth, but using the same bipolar initial distributions of wealth.

In order to test how much effect changing the magnitude of a_1 will have, Fig. (5-9) and Table (5.4) displays the results for the case when $a_1 = -.5$ with all other parameters and data remaining the same.

Comparing Table (5.4) and Fig. (5-9) to the previous graph and table above (Fig. (5-8) and Table (5.3) whose only difference with this data set and graph is the fact that $a_1 = -1$ previously), we see that the optimal $\bar{\pi}$ moves slightly away from .60 to .61. In addition, the function values are smaller, which is an expected effect from changing a_1 described above. A less obvious difference to notice is that the average π

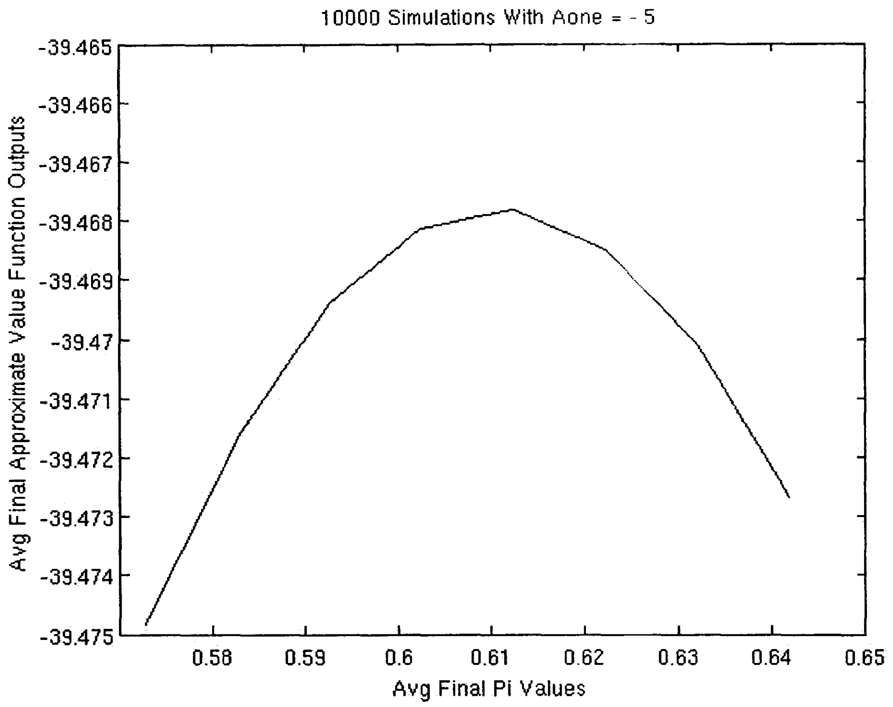

Figure 5-9: Plot of the average final value approximate function outputs with 10000 simulations and a_1 is a smaller negative value for function ($a_1 = -.5$)

values, in the case when $a_1 = -.5$, are smaller than the average π values in the case when $a_1 = -1$. We will discuss the reason for this discrepancy using Table (5.5).

Table (5.5) is a table of values averaging the final approximate value function outputs, given the choice variables over 10,000 simulations and varying the values of a_1 so that the portfolio optimization is dynamic. We vary the values of $\bar{\pi}$ as well but within a small region, close to the static portfolio solution. The cells for the table are meant to be read as follows: (avg π, avg approximate value function result).

Continuing the point made early in the discussion of Table (5.2), we notice that as the magnitude for a_1 decreases, the value for $\bar{\pi}$ also decreases. The effect in Table (5.5) is much more obvious than in Table (5.2). The reason is that as the value for a_1 decreases, the approximate value function is less restricted by the preconceived notion that π should be equal to $\bar{\pi}$. Therefore, the weight of the exponential factor decreases, which means that the comparative weight of the transaction costs increases.

Specified $\bar{\pi}$	.62	.61	.60
a_1			
-.4	(.62126, -39.46145)	(.61144, -39.46098)	(.60160, -39.46149)
-.3	(.61855, -39.45473)	(.60877, -39.45473)	(.59898, -39.45568)
-.2	(.60850, -39.45386)	(.59883, -39.45416)	(.58913, -39.45640)
-.1	(.56063, -39.49238)	(.55513, -39.49342)	(.55045, -39.49378)

Table 5.5: Table of values of the average final approximate value function outputs with 10000 simulations and a_1 varies between different smaller negative values for function

Specified $\bar{\pi}$	.65	.64	.63	.62
a_1				
-.19	(.63518, -39.45458)	(.62563, -39.45335)	(.61604, -39.45305)	(.60642, -39.45368)
-.18	(.63267, -39.45501)	(.62314, -39.45406)	(.61357, -39.45403)	(.60397, -39.45493)
-.17	(.62969, -39.45573)	(.62019, -39.45514)	(.61064, -39.45546)	(.60106, -39.45669)
-.16	(.62618, -39.45698)	(.61669, -39.45675)	(.60716, -39.45743)	(.59760, -39.45904)

Table 5.6: Table of values of the average final approximate value function outputs with 10000 simulations and a_1 varies in a very concentrated range

This means that as a_1 decreases, the function puts more weight on saving money by reducing transaction costs than rebalancing the portfolio so that $\pi = \bar{\pi}$. Because the orignal wealth distribution for this data was $\pi = .5$ $(y_0 = 100, x_0 = 100)$, minimizing transactions cost means less movement away from $\pi = .5$. This is why when a_1 decreases in magnitude, π values become smaller and approach .5.

We notice that the optimal value in Table (5.5) occurs when $a_1 = -.2$ and $\bar{\pi} = .62$. In order to numerically narrow down the region where the optimal parameters lie, we simulate the approximate value function again on a narrower set for a_1 in Table (5.6):

As we can see in Table (5.6), the optimal values for $\bar{\pi}$ and a_1 are .63 and $-.19$. Looking back, the optimal value of $\bar{\pi} = .63$ in the dynamic case (when a_1 is not a highly negative constant and is allowed to vary) is close to the optimal value of $\bar{\pi} = .61$ in the static case (when a_1 is a highly negative constant).

How do we know that the optimal values for $\bar{\pi}$ and a_1 that the simulations produce is correct? In order to determine the accuracy of these results, we calculate the standard error of the mean of the approximate value function outputs for all simulations. Calculating the standard deviation of the function's 10,000 final simulation values from the mean (39.45305 from Table (5.6)), we get $\sigma_{stddev} = 0.0246235$. From statistics, we know that:

$$\begin{aligned} \text{Standard Error} &= \frac{\sigma_{stddev}}{\sqrt{\text{num\_runs}}} \\ \text{Standard Error} &= \frac{0.02462}{\sqrt{10,000}} \\ \text{Standard Error} &= 2.46235 * 10^{-4} \end{aligned} \tag{5.2}$$

Using the 95% confidence interval for the standard error calculated in Eq. (5.2) (95% confidence interval $\rightarrow$ 1.96 * Standard Error = range for mean):

$$\text{True mean of } (t = T) \text{ Approximate Value Function Outputs} \in (39.45305 \pm 4.82620 * 10^{-4}) \tag{5.3}$$

Eq. (5.3) defines the range where the true mean lies in given a 95% confidence interval. This interval overlaps with the mean utility value in Table 5.6 given $\bar{\pi} = .64$ and $a_1 = -.19$, but with no others. This means that we have successfully found the optimal value of a_1 with 95% certainty and a small range for the optimal value of $\bar{\pi} \in (.63, .64)$ with 95% certainty. Due to the large number of simulations chosen (10,000), we can say that the simulations produced accurate results.

We have found the optimal values for the parameters of the approximate value function. By finding the optimal values for $\bar{\pi}$ and a_1 for the new function, we have provided a new portfolio policy for risk averse investors to employ, in addition to other suggested strategies by others for the single stock and bond case with transaction costs. We cannot directly compare the output values of the approximate value function with other strategies because the output value magnitudes from the approximate value function are arbitrary.

One possible method of comparing strategies is to run the approximate value function against a different published value function/trading strategy. Instead of comparing the average of the final value function values of each simulation, we can compare the average of the final total wealth values of each simulation. Even this method of comparison may not be possible because the results may just mean that the risk aversion of an investor using one value function is greater than the risk aversion of another investor using a different value function. Unfortunately, I was unable to find other trading strategies that translated to using a value function that was compatible to the MATLAB simulation setup. Additional research related to this paper could include finding compatible strategies developed by others and testing the performance of the approximate value function against theirs.

Our approximate value function capabilities are not limited to only the single stock and bond scenario. Not only does the MATLAB code allow us to find optimal values for the approximate value function parameters in the one stock case, but the code is versatile enough to test the function in order to discover its optimal parameters in higher dimensional cases. Only minor extensions to portions of the code are necessary in order to handle higher dimensional cases. A possible extension to this paper would be to rewrite the code to allow the usage of more than one stock and bond without the need to adjust significant parts of the code whenever the user decides to change the dimensionality of the problem.

5.3 Two Stocks, One Bond With Transaction Costs Case

We have used the approximate value function model with the changes promoted in section 3.2 along with changes to the associated code to begin analysis of the case of Two Stocks-One Bond with Transaction Costs Case. Small-sized runs are convincing that the computational implementation is working as intended and returns useful

Specified $\bar{\pi}_2$	.3	.4	.5	.6	.7
$\bar{\pi}_1$					
.3	-250.08364	-247.27020	-245.31997	-244.01101	-243.36878
.4	-247.26962	-245.31918	-244.01147	-243.36877	0
.5	-245.31922	-244.01176	-243.36908	0	0
.6	-244.01127	-243.36985	0	0	0

Table 5.7: Table of values of the average final approximate value function outputs with two stocks, 1000 simulations and a_1, a_2, a_3, are largely negative

results in this case. As discussed in section 5.2, we need on the order of $n = 10000$ iterations in order to achieve a minimal standard error of the estimate. Unfortunately, we do not currently have access to computational assets that we can control for long enough in order to complete analyses of this size. We will pursue this when computational assets become available.

Setting $n = 1000$, the computational implementation produces interesting results. Selecting two stocks whose returns are completely independent of each other (the probability of each stock increasing and decreasing is independent of the other), selecting $a_1 = a_2 = a_3 = -4$ (largely negative to make the problem static), and setting $\mu_1 = .1$, $\mu_2 = .15$, $\sigma_1 = .2$, $\sigma_2 = .3$, the approximate value function determines optimal values for the parameters $\bar{\pi}_1$ and $\bar{\pi}_2$ in the static case.

Table (5.7) allows us to determine that the optimal choices for parameters $\bar{\pi}_1$ and $\bar{\pi}_2$ in the static case are $\bar{\pi}_1 = .4$ and $\bar{\pi}_2 = .6$. The number of simulations performed in the two stock case is a factor of 10 less than the number of simulations performed in the single stock and bond case. Looking at Table (5.7), we see that the average final approximate value function outputs when $\bar{\pi}_1 = .4$ and $\bar{\pi}_2 = .6$ and when $\bar{\pi}_1 = .3$ and $\bar{\pi}_2 = .7$ are very close to each other. Because the number of simulations performed is low, it is likely that the true optimal values for $\bar{\pi}_1$ and $\bar{\pi}_2$ in the static case lie somewhere within this range of values.

Given the values $\bar{\pi}_1$ and $\bar{\pi}_2$ above, the approximate value function states that a risk averse investor, given two independent stocks and a risk-free bond with returns equivalent to those for the simulation above, will choose to invest in only the two stocks, and not in the risk-free bond. This may seem unreasonable, but given our assumptions, this solution makes sense. In the simulation. we stipulated that the behavior of the two stocks were independent of each other. Therefore, diversifying between the two stocks reduces much of the risk in our portfolio; it reduces the risk so much that given the expected returns, the approximate value function tells us that even a risk averse investor is willing to fully invest in stocks.

However, this assumption of the independence of stock returns is not entirely realistic. In the stock market, the performance of many stocks are correlated to each other. The recent crash of the market is obvious evidence of this interdependence. It is very difficult to find two stocks that are completely independent of one another, although one can come close by selecting stocks that involve separate industries of the economy. With access to increased computational capabilities, we should simulate the approximate value function again over a set of stocks which have a certain degree of correlation. As the correlation between our stocks' returns grows, the portfolio becomes less diversified. It would be expected that as the correlation increases, the percentage of wealth held in bonds would also increase to diversify our risk.

A set of important parameters that we would like to vary to increase our insight into this higher dimensional problem, given increased computational capabilities, is a_1, a_2, and a_3. In the Table (5.7), we chose a_1, a_2, a_3 to all be highly negative and equivalent values. In order for the portfolio to become dynamic and perform better, we would need to decrease and vary these values so that all three parameters are not necessarily equivalent.

With the data analysis completed, we are able to concisely tie our results together in order to draw conclusions as well as suggest topics of further research which would be valuable in extending this paper.

Chapter 6

Possible Research Extensions and Conclusion

6.1 Possible Research Extensions

In several sections of the paper thus far, we have mentioned possible extensions to the research which would help the efficiency/applicability of this paper to the field of portfolio optimization. These opportunities are described below.

6.1.1 Better Multi-Dimensional Optimization Simulator

During the course of performing the data analysis, we mentioned that MATLAB's fmincon optimizer did not work effectively in the case of pi_a_fmincon.m. Because the "function" we were optimizing was a value which was the average of a number of simulations, MATLAB was unable to pursue the optimal solution properly. A very useful piece of code would be a multi-dimensional optimizer routine that is based on simulations, and not based on a gradient or a finite difference approximation. With this extension available for the paper, we could pursue the optimal parameters for the approximate value function in higher order dimensions more quickly and efficiently than we are able to currently.

6.1.2 Generalize Code to Construct Approximate Value Function

In transitioning from a single stock and bond scenario to a two stock, one bond scenario, we previously discussed that it was necessary to adjust the approximate value function as well as the simulation code to handle this higher order dimensional data. Ideally, we could construct code in a higher level language, like Java, to create the approximate value function and simulation code based upon the number of user-specified stocks and their μ, σ parameters. For example, pseudocode that would perform the task of create the function would be of the form:

```
1. For each stock $i$ in input $stock\_array$ with cell data $y_{i0}$, $\mu_i$, $\sigma_i$
2. ValFunct(wealthfactorup) = ValFunct(wealthfactorup) $+u_i * (y_{i0} + y_{iB} - y_{iS}) - r_f * \kappa * (y_{iB} + y_{iS})$;
3. temp = ValFunct(expfactor)
4. For each other stock $j$ already processed in $stock\_array$
5. temp = temp * $e^{2*a_{ij}*(\pi_i - \bar{\pi}_i)*(\pi_j - \bar{\pi}_j)}$;
6. end
7. ValFunct(expfactor) = temp $*e^{a_{ii}*(\pi_i - \bar{\pi}_i)^2}$;
```

Other code would need to be written the generate the rest of the code in MEngCode.m. For each stock, generate appropriate bounds on values, etc...

Having this higher-level code would mean that we are no longer required to adjust the approximate value function and simulation code everytime we would like to make a change to the dimensional parameters of the problem. This code would take care of this change in dimension for the user.

6.1.3 Allow Approximate Value Function Parameters to Vary Across Time Periods

In the current approximate value function, once the user picks the value of $\bar{\pi}$, this value is fixed for all time periods throughout the simulation. However, there may be times when the investor wants to change his $\bar{\pi}$ in the middle of his investment period. A good extension to the function would be to allow this change mid-simulation, however, in order to make this change, we would impose a penalty to the final average function output (new output to optimize = (avg final function output across simulations) $-c*\sum(\Delta\bar{\pi})^2$ where c is a parameter to be optimized). This additional degree of freedom would provide another factor that we must optimize in adherence to. Although the optimization would become slightly more complicated, the approximate value function would be able to handle a wider range of situations.

6.1.4 Find Compatible Trading Strategies to Test Against

Finally, one other extension that would be useful for the purposes of this paper would be to find other researchers' portfolio optimization strategies whose trading strategies translate to using a function that needs to be optimized and is compatible with the simulation setup for this paper (discrete, binomial stock movement, etc.). With other trading strategies at hand, we could compare the performance of the approximate value function against those provided by others under numerous test conditions to see which provides the investor with optimal performance.

6.2 Conclusion

In this paper, we developed a new approximate value function that takes into account transaction costs and return predictability. Key to this new function is the approximation of a factor in the true value function with a second order exponential factor. This factor is parameterized by a pre-specified portfolio composition $\bar{\pi}$, the Composition Factor, and the portfolio's insistence on staying close to this composition, denoted by a, the Obstinance Factor. This approximate value function emulates the true value function well and is much quicker to optimize at each time step than using dynamic programming to solve the true value equation. Using the Cox, Ross, and Rubinstein's discrete time approximation for returns on stocks and bonds [5], we formulated the portfolio optimization problem using discrete time steps and optimized the portfolio in accordance with the approximate value function at each time step.

Using the single stock and bond with no transaction costs base case scenario, we verified that the approximate value function model returns Merton's ratio as the optimal portfolio choice, providing evidence that the model and computational implementation are operating correctly. We are able to determine the optimal parameters for the function in the single stock and bond scenario with transaction costs to within a 95% confidence interval. Determining these optimal parameters equivalently determines the optimal dynamic portfolio strategy. With only minor modifications we were able to have extend the approximate value function and computational implementation to higher dimensional portfolios. We have performed simulations of the two stock, one bond scenario for low numbers of iterations and have verified that the results are reasonable and consistent with expectations, and obtain some insight as to what the optimal parameters would be for that case. We are ready to perform these computations at a high confidence level when sufficient uninterrupted time on a dedicated computing platform can be obtained.

Appendix A

Computational Implementation

The actual code of the Computational Implementation (Section 4) section of the paper is shown here. As discussed earlier in Section 4, the environment chosen for the computational implementation is MATLAB. Below are each of the different MATLAB files which were described in detail in Section 4. The comments in the files describe the MATLAB specifics of our implementation.

```
% This file generates the data for the computational simulations of our approximate value
% function.  The parameters are sectioned into parameters that are subject to change by the
% user and parameters that are generated from other parameters.

% Parameters to be adjusted by the user
gam = 2;                    % Risk aversion factor
kappa = 0.01;               % Transaction costs rate
num_runs = 10000;           % Number of simulations
% Lambda parameter for exponential distribution to determine time invested in the market
% for each simulation (t = T)
simLength = 48;
tableLength = 500;          % Maximum length of returns stored in table
mu = .1;                    % Return of stock
sigma = .2;                 % Variance of stock
deltaT = 1/12;              % Length of time-steps (1 month)
r = .05;                    % Annual RF-rate parameter
x_0 = 100;                  % Initial distributions of wealth
y_0 = 100;

% Parameters determined by parameter values above
mu_bar = mu - (sigma)^2/2;          % Adjusted return of stock
RFrate = exp(r*deltaT);             % Monthly RF-rate
u = exp(sigma*sqrt(deltaT));        % u factor gain if stock price increases
d = 1/u;                            % d factor loss if stock price decreases
p = (1/2) + (1/2) * (mu_bar/sigma) * sqrt(deltaT); % probability of stock price increase


% Set return matrix to randomly return u or d at each time step given p
return_matrix = rand(num_runs,tableLength);
for i=1:num_runs,
  for j=1:tableLength,
    if (return_matrix(i,j) < p)
      return_matrix(i,j) = u;
    else
      return_matrix(i,j) = d;
    end
  end
end

% Determine array of different simulation lengths for each simulation
sample_sizes = ceil(exprnd(simLength,1,num_runs));

% Save generated data for use by pi_a_solve.m and MEngcode.m
save('newtemp', 'gam', 'kappa', 'num_runs', 'simLength', 'tableLength', 'mu', 'sigma', 'deltaT',
     'r', 'RFrate', 'u', 'd', 'p', 'x_0', 'y_0', 'return_matrix', 'sample_sizes');
```

```
% This file contains our Approximate Value Function which is to be maximized at every time step
% to determine the optimal trade for the risk averse investor to make

function f = objgradnew(x,yamount,xamount,kappa,gam,pibar_yt,Azero,Aone,RFrate,u,d,p)

% Unknown Variables to be Optimized
% x(1) = xB
% x(2) = xS
% x(3) = yB
% x(4) = yS

% Break down approximate value function into factors to piece to together to avoid any mistakes

% With prob p that the stock price increases
% totalWealthup = RFrate*(xamount+x(1)-x(2)-(kappa*(x(1)+x(2)+x(3)+x(4)))
%                        +u*(yamount+x(3)-x(4)))
% yRatioup = (((yamount+x(3)-x(4))/totalWealthup)-pibar_yt);
% correlationFactorup = exp(Azero+Aone*(yRatioup)^2);

% With prob 1-p that the stock price decreases
% totalWealthdown = RFrate*(xamount+x(1)-x(2)-(kappa*(x(1)+x(2)+x(3)+x(4)))
%                          +d*(yamount+x(3)-x(4)))
% yRatiodown = (((yamount+x(3)-x(4))/totalWealthdown)-pibar_yt);
% correlationFactordown = exp(Azero+Aone*(yRatiodown)^2);

% Approximate Value Function broken down into pieces
% f = -(p*1/(1-gam)*(totalWealthup*correlationFactorup)^(1-gam) +
%   (1-p)*1/(1-gam)*(totalWealthdown*correlationFactordown)^(1-gam))

% Approximate Value Function in terms of input parameters and variables to be optimized
f = 10000*(-(p*1/(1-gam)*((RFrate*(xamount+x(1)-x(2)-(kappa*(x(1)+x(2)+x(3)+x(4))))
         +u*(yamount+x(3)-x(4)))
*exp(Azero+Aone*(((u*(yamount+x(3)-x(4))/
(RFrate*(xamount+x(1)-x(2)-(kappa*(x(1)+x(2)+x(3)+x(4))))+u*(yamount+x(3)-x(4))))-pibar_yt))^2))^(1-gam)
+ (1-p)*1/(1-gam)*((RFrate*(xamount+x(1)-x(2)-(kappa*(x(1)+x(2)+x(3)+x(4))))
                  +d*(yamount+x(3)-x(4)))
*exp(Azero+Aone*(((d*(yamount+x(3)-x(4))/ (RFrate*(xamount+x(1)-x(2)-(kappa*(x(1)+x(2)+x(3)+x(4))))
                                               + d*(yamount+x(3)-x(4))))-pibar_yt))^2))^(1-gam)));
end
```

```
% In Section 4 of the paper, we mentioned that pi_a_fmincon did not correctly. We also discussed
% the reason why the function was not operating correctly. If MATLAB's fmincon worked properly
% in this situation, this code would find the optimal parameters pibar_yt, Azero, and Aone for our
% approximate value function. We would not need to manually search for the solution ourselves.

% Extend Decimal Output
format long;

% Designate initial guess values for parameters to be optimized
pibar_yt = .6;
Azero = 0;
Aone = -.5;

% Create starting guess input for fmincon optimizer
input = [pibar_yt; Azero; Aone];

%Create holder function for fmincon
MENGholder = @(input)MEngCode(input);

%set appropriate options
options = optimset('Algorithm','active-set');
options = optimset(options, 'Display','off');

% Set bounds on our parameters to be optimized
lb = [0 -1 -1];
ub = [1 0 0];

% Find the optimal values for our parameters
[x, fval] = fmincon(MENGholder,input,[],[],[],[],lb,ub,[],options)
```

```
% This file is the master file. This file takes in the user's input for parameters for the value
% function -> parameters pibar, a0, a1. For each set of choices of parameters pibar, a0, a1,
% this file sends those values to MEngCode.m which performs the computational simulations
% on that specified approximate value function and returns values which determines its
% performance. The values are stored in tables and are accessible to the user after the file has
% completed running. The user is then able to compare the performance of different approximate
% value functions (approximate value functions of the same form with different parameter values)

% Increase decimal output length
format long;
% Load data for simulations
load newtemp;

% Optimal Azero = 0 -> See Section 5 for more details
Azero = 0;

% Initialize data tables to store results
AVG_MENGUtility_Table = zeros(1, 6);
AVG_MENGpiyt_Table = zeros(1, 6);
final_utility_Table = zeros(3,num_runs);

% Initialize indices for data tables
AVG_Table_Aone_index = 1;
AVG_Table_pibar_index = 1;

for pibar_yt = [.64 .63 .62 .61 .6 .59],
  AVG_Table_Aone_index = 1;
  for Aone = -.19,
    % Create input for MEngCode.m
    input = [pibar_yt; Azero; Aone];

    % Simulate approximate value function with chosen parameters in MEngCode.m
    [fval pi_yt finalutility] = MEngCode(input);

    % Save resulting avg value function output, avg pi, and final approximate value function
    % outputs for all simulations (for standard error calculations)
    AVG_MENGUtility_Table(AVG_Table_Aone_index, AVG_Table_pibar_index) = fval;
    AVG_MENGpiyt_Table(AVG_Table_Aone_index, AVG_Table_pibar_index) = pi_yt;
    final_utility_Table(AVG_Table_pibar_index,:) = finalutility;
    AVG_Table_Aone_index = AVG_Table_Aone_index + 1;
  end
  AVG_Table_pibar_index = AVG_Table_pibar_index + 1;
end
```

```
% This file contains the code responsible for performing the computational simulations. This file
% calls upon the file created by createData.m for the parameters and data required for simulation.
% At each optimization iteration. it calls upon objgradnew.m for our approximate value function
% to optimize. After all simulations are performed on the current approximate value function
% (given the variables passed in by pi_a_solve.m -> pibar, a0, a1) it returns the avg final approximate
% value function output, the avg final pi of the portfolio, the avg final wealth, and the final value
% function outputs of all the simulations.

function [avgutil avgpi_yt avgwealth finalutility] = MEngCode(input)

% Load data created by createData.m
load newtemp;
% Increase decimal output
format long;
% Turn off all warning messages (hinders speed of simulations)
warning off all;

% Acoeff in form Acoeff = a_0 + a_1 * (pi_yt - pibar_yt)^2
% Receive parameters for approximate value function from pi_a_solve.m
pibar_yt = input(1);
Azero = input(2);
Aone = input(3);

%Create Matrices to store info gathered from completed simulations
finalutility = zeros(1,num_runs);
finalpi_yt = zeros(1,num_runs);
finalwealth = zeros(1,num_runs);

% For each single simulation instance
for i=1:num_runs,

  % Create temporary arrays to store important values over the simulation
  tempSize = sample_sizes(i);  % Length of the simulation i
  xtemp = zeros(1,tempSize+1);  % Vector of xt values
  ytemp = zeros(1,tempSize+1);  % Vector of yt values
  xtemp(1,1) = x_0;             % Initial wealth distribution
  ytemp(1,1) = y_0;

  % Vector of Approximate Value function outputs for simulation i
  utilityValues = zeros(1,tempSize);

  % For each time step j in simulation i
  for j=1:tempSize,

    % Initial wealth distribution
    xamount = xtemp(1,j);
    yamount = ytemp(1,j);
```

```
% Create holder function to optimize
holderFunction =
      @(x)objgradnew(x,yamount,xamount,kappa,gam,pibar_yt,Azero,Aone,RFrate,u,d,p);

% Make initial guess for objgradnew.m to use during optimization
% Guess = trade to make pi = pibar
correcty = pibar_yt * (xamount + yamount);
if (yamount <= correcty)
   guesssellx = correcty - yamount;
   guessbuyy = guesssellx / (1+kappa);
   guessselly = 0;
   guessbuyx = 0;
else
   guesssellx = 0;
   guessbuyy = 0;
   guessselly = yamount - correcty;
   guessbuyx = guessselly / (1+kappa);
end
inputx = [guessbuyx; guesssellx; guessbuyy; guessselly];

% Set option below on and turn option below that off to see MATLAB's optimization steps for
% the approximate value function
% options =
% optimset('TolX',1e-15,'TolFun',1e-20,'TolCon', 1e-6, 'MaxFunEvals', 1e3, 'MaxIter', 1e3, 'Display', 'iter');
options = optimset('TolX',1e-15,'TolFun',1e-20,'TolCon',1e-6,'MaxFunEvals',1e3,'MaxIter',1e3);

% Select options for maximization method
options = optimset(options, 'Algorithm','active-set');

% Turn off optimization solution display for each time step
% Very Large constraint on throughput
options = optimset(options, 'Display','off'); % Turn off Display

% Turn on option below if gradient calculated by user and inputted into objgradnew.m
% options = optimset(options, 'GradObj','on');

% Set upper bounds on variables so that user is not allowed to short sell more than total
% portfolio worth
maxYSell = (yamount + xamount);
maxXSell = (xamount + yamount);
maxYBuy = maxXSell;
maxXBuy = maxYSell;

% Set lower bound on variables (given that xB, xS, yB, yS >= 0)
lb = [0 0 0 0];
ub = [maxXBuy maxXSell maxYBuy maxYSell];

% Set constraint that calculates transactions cost during optimization
BuyCoeff = 1+kappa;
```

```
        SellCoeff = -1;
        A = [BuyCoeff SellCoeff BuyCoeff SellCoeff];
        b = 0;

        % Perform optimization of approximate value function using MATLAB's fmincon function
        % on objgradnew.m which holds our approximate value function
        [x, fval] = fmincon(holderFunction,inputx,A,b,[],[],lb,ub,[],options);

        % If the optimal trade size is <= 10^-6, do not trade at all
        % (Realistically cannot trade that little)
        if (((x(1) < 1e-6) && (x(2) < 1e-6) && (x(3) < 1e-6) && (x(4) < 1e-6))
           || (fval < 0))
           x(1) = 0;
           x(2) = 0;
           x(3) = 0;
           x(4) = 0;
           fval = objgradnew(x,yamount,xamount,kappa,gam,pibar_yt,Azero,
                       Aone,RFrate,u,d,p);
        end

        % Assign new values for our wealth in stocks and bonds and add the approximate value
        % function output to our storage array

        % NOTE: Since we found the minimum of our Approximate Value Function * -1, to get the
        % maximum value back, we have to multiply fval by -1
        utilityValues(1,j) = -1 * fval;
        tempx = xamount + x(1) - x(2);
        tempy = yamount + x(3) - x(4);

        % Simulate returns on the portfolio after transactions have been made during the current time
        % step period
        xtemp(1,j+1) = tempx*RFrate;
        if (mod(j,tableLength) > 0)
           ytemp(1,j+1) = tempy*return_matrix(i,mod(j,tableLength));
        else
           ytemp(1,j+1) = tempy*return_matrix(i,tableLength);
        end
     end

     % We must collect the final approximate value function outputs, the final pi, and the final total
     % wealth in order to eventually average them together in order to determine the optimal values
     % for parameters pibar, a0, a1 of our approximate value function
     finalpi_yt(1,i) = ytemp((1,tempSize+1)/(ytemp(1,tempSize+1)+xtemp(1,tempSize+1)));
     finalutility(1,i) = utilityValues(1,tempSize);
     finalwealth(1,i) = ytemp(1,tempSize+1) + xtemp(1,tempSize+1);
  end

  % Calculate the desired averages from the arrays of values collected throughout the simulations.
  % These values are returned to the function pi_a_solve.m
```

```
% These values do not have their outputs terminated in order for the user to determine the progress
% of pi_a_solve.m so far
avgutil = mean(finalutility)
avgpi_yt = mean(finalpi_yt)
avgwealth = mean(finalwealth)

end
```

```
% This file allows the user to graph the approximate value function for a single time step.  All
% parameter specifications from createData.m are in this file for the user to manipulate.  This file is
% independent of all other files.  This allows the user to change parameters without worry.  The
% function plots our approximate value function over all possible portfolio adjustment choices.

%Extend decimal output
format long;

%Initialize arrays for plotting our approximate value function
x = zeros(201,4);
yvalue = zeros(201,1);

%Designate parameters of our exponential factor
Aone = -.2;
Azero = -.02;
pibar_yt = .8;

%Designate parameters of our optimization problem
gam = 2;                    % Risk aversion factor
kappa = .01;                % Transaction cost rate
num_runs = 5000;            % Number of Simulations
simLength = 48;             % Parameter for distribution for length of portfolio (T)
tableLength = 500;          % Maximum length stored in table
mu = .1;                    % Return of stock
sigma = .2;                 % Volatility of stock
mu_bar = mu - (sigma)^2/2; % Adjusted return of stock
deltaT = 1/12;              % Time step length
r = .05;                    % Annual RF-rate.
RFrate = exp(r*deltaT); % Monthly RF-rate
u = exp(sigma*sqrt(deltaT));  % u factor for stock price increase
d = 1/u;                              % d factor for stock price decrease
p = (1/2) + (1/2) * (mu_bar/sigma) * sqrt(deltaT); % probability of stock price increase

x_0 = 100;                  % Initial allocations of wealth
y_0 = 100;


% Determine approximate value function output for each possible portfolio change
for i = -100:100,
   k = i+101;
   buy = i/(1+kappa);
   sell = i;
   if (i < 0)
      x(k,:) = [-buy; 0; 0; -sell];
   else
      x(k,:) = [0; sell; buy; 0];
   end
   yvalue(k) = objgradnew(x(k,:),y_0,x_0,kappa,gam,pibar_yt,Azero,Aone,RFrate,u,d,p);
end
```

```
% Graph approximate value function
l = -100:100;
plot(l,yvalue)

% Determine optimal change in portfolio
% See index system in for loop above to determine the action equalivalent of the index
% outputted as the choice to optimize our value function
[c j] = min(yvalue)
```

Appendix B

Gradient of Value Function

The gradient of the approximate value function is listed in this Appendix. The four partial derivatives are with respect to x_B, x_S, y_B, and y_S. These four variables are the variables to be maximized in the approximate value function at every time step in our code. These derivatives were computed using Mathematica. As we can see, the derivatives are rather unwieldy. As the dimensions of the portfolio allocation problem grows, so does the size and complexity of the approximate value function, making the derivative that much harder to compute. In the computational implementation, we approximate the gradient by using MATLAB's finite difference method. If the available computing power is capable of computing the gradient in the higher dimensional problem, it would be more efficient to invest the time to compute the user-supplied gradient in Mathematica once rather than rely on MATLAB to approximate the gradient at every time step.

```
totalWealthup[x1_, x2_, x3_, x4_] :=
  RFrate * (xamount + x1 - x2 - (kappa * (x1 + x2 + x3 + x4))) + u * (yamount + x3 - x4);
totalWealthdown[x1_, x2_, x3_, x4_] :=
  RFrate * (xamount + x1 - x2 - (kappa * (x1 + x2 + x3 + x4))) + d * (yamount + x3 - x4);
yRatioup[x1_, x2_, x3_, x4_] := (yamount + x3 - x4) / totalWealthup[x1, x2, x3, x4] - pibaryt;
yRatiodown[x1_, x2_, x3_, x4_] := (yamount + x3 - x4) / totalWealthdown[x1, x2, x3, x4] - pibaryt;
correlationFactorup[x1_, x2_, x3_, x4_] := Exp[a1 + a2 * (yRatioup[x1, x2, x3, x4]) ^2];
correlationFactordown[x1_, x2_, x3_, x4_] := Exp[a1 + a2 * (yRatiodown[x1, x2, x3, x4]) ^2];

f[x1_, x2_, x3_, x4_] :=
 -(p * (1 / (1 - gamma) * (totalWealthup[x1, x2, x3, x4] * correlationFactorup[x1, x2, x3, x4]) ^
        (1 - gamma)) + (1 - p) * (1 / (1 - gamma) *
       (totalWealthdown[x1, x2, x3, x4] * correlationFactordown[x1, x2, x3, x4]) ^ (1 - gamma)))

TraditionalForm[f[x1, x2, x3, x4]]
```

$$-\frac{1}{1-\text{gamma}}(1-p)\left(e^{\text{a2}\left(\frac{\text{x3}-\text{x4}+\text{yamount}}{\text{RFrate}\,(\text{x1}-\text{x2}-\text{kappa}\,(\text{x1}+\text{x2}+\text{x3}+\text{x4})+\text{xamount})+d\,(\text{x3}-\text{x4}+\text{yamount})}-\text{pibaryt}\right)^2+\text{a1}}\right.$$

$$\left.(\text{RFrate}\,(\text{x1}-\text{x2}-\text{kappa}\,(\text{x1}+\text{x2}+\text{x3}+\text{x4})+\text{xamount})+d\,(\text{x3}-\text{x4}+\text{yamount}))\right)^{1-\text{gamma}}-$$

$$\frac{1}{1-\text{gamma}}\,p\left(e^{\text{a2}\left(\frac{\text{x3}-\text{x4}+\text{yamount}}{\text{RFrate}\,(\text{x1}-\text{x2}-\text{kappa}\,(\text{x1}+\text{x2}+\text{x3}+\text{x4})+\text{xamount})+u\,(\text{x3}-\text{x4}+\text{yamount})}-\text{pibaryt}\right)^2+\text{a1}}\right.$$

$$\left.(\text{RFrate}\,(\text{x1}-\text{x2}-\text{kappa}\,(\text{x1}+\text{x2}+\text{x3}+\text{x4})+\text{xamount})+u\,(\text{x3}-\text{x4}+\text{yamount}))\right)^{1-\text{gamma}}$$

```
fx1 = D[f[x1, x2, x3, x4], x1]
```

$$
\begin{aligned}
&-(1-p)\Bigg(\Bigg(e^{a1+a2\left(-pibaryt+\frac{x3-x4+yamount}{RFrate\,(x1-x2-kappa\,(x1+x2+x3+x4)+xamount)+d\,(x3-x4+yamount)}\right)^2}\\
&\qquad (RFrate\,(x1-x2-kappa\,(x1+x2+x3+x4)+xamount)+d\,(x3-x4+yamount))\Bigg)^{-gamma}\\
&\quad\Bigg(e^{a1+a2\left(-pibaryt+\frac{x3-x4+yamount}{RFrate\,(x1-x2-kappa\,(x1+x2+x3+x4)+xamount)+d\,(x3-x4+yamount)}\right)^2}\,(1-kappa)\,RFrate-\\
&\qquad\Bigg(2\,a2\,e^{a1+a2\left(-pibaryt+\frac{x3-x4+yamount}{RFrate\,(x1-x2-kappa\,(x1+x2+x3+x4)+xamount)+d\,(x3-x4+yamount)}\right)^2}\,(1-kappa)\,RFrate\,(x3-x4+yamount)\\
&\qquad\quad\left(-pibaryt+\frac{x3-x4+yamount}{RFrate\,(x1-x2-kappa\,(x1+x2+x3+x4)+xamount)+d\,(x3-x4+yamount)}\right)\Bigg)\Bigg/\\
&\qquad (RFrate\,(x1-x2-kappa\,(x1+x2+x3+x4)+xamount)+d\,(x3-x4+yamount))\Bigg)-\\
&p\Bigg(\Bigg(e^{a1+a2\left(-pibaryt+\frac{x3-x4+yamount}{RFrate\,(x1-x2-kappa\,(x1+x2+x3+x4)+xamount)+u\,(x3-x4+yamount)}\right)^2}\\
&\qquad (RFrate\,(x1-x2-kappa\,(x1+x2+x3+x4)+xamount)+u\,(x3-x4+yamount))\Bigg)^{-gamma}\\
&\quad\Bigg(e^{a1+a2\left(-pibaryt+\frac{x3-x4+yamount}{RFrate\,(x1-x2-kappa\,(x1+x2+x3+x4)+xamount)+u\,(x3-x4+yamount)}\right)^2}\,(1-kappa)\,RFrate-\\
&\qquad\Bigg(2\,a2\,e^{a1+a2\left(-pibaryt+\frac{x3-x4+yamount}{RFrate\,(x1-x2-kappa\,(x1+x2+x3+x4)+xamount)+u\,(x3-x4+yamount)}\right)^2}\,(1-kappa)\,RFrate\,(x3-x4+yamount)\\
&\qquad\quad\left(-pibaryt+\frac{x3-x4+yamount}{RFrate\,(x1-x2-kappa\,(x1+x2+x3+x4)+xamount)+u\,(x3-x4+yamount)}\right)\Bigg)\Bigg/\\
&\qquad (RFrate\,(x1-x2-kappa\,(x1+x2+x3+x4)+xamount)+u\,(x3-x4+yamount))\Bigg)
\end{aligned}
$$

```
fx2 = D[f[x1, x2, x3, x4], x2]
```

$$
(1-\text{p})\left(\mathbb{e}^{\text{a1}+\text{a2}\left(-\text{pibaryt}+\frac{\text{x3}-\text{x4}+\text{yamount}}{\text{RFrate}\,(\text{x1}-\text{x2}-\text{kappa}\,(\text{x1}+\text{x2}+\text{x3}+\text{x4})+\text{xamount})+\text{d}\,(\text{x3}-\text{x4}+\text{yamount})}\right)^2}\right.
$$

$$
\left.(\text{RFrate}\,(\text{x1}-\text{x2}-\text{kappa}\,(\text{x1}+\text{x2}+\text{x3}+\text{x4})+\text{xamount})+\text{d}\,(\text{x3}-\text{x4}+\text{yamount}))\right)^{-\text{gamma}}
$$

$$
\left(\mathbb{e}^{\text{a1}+\text{a2}\left(-\text{pibaryt}+\frac{\text{x3}-\text{x4}+\text{yamount}}{\text{RFrate}\,(\text{x1}-\text{x2}-\text{kappa}\,(\text{x1}+\text{x2}+\text{x3}+\text{x4})+\text{xamount})+\text{d}\,(\text{x3}-\text{x4}+\text{yamount})}\right)^2}(-1-\text{kappa})\,\text{RFrate}-\right.
$$

$$
\left(2\,\text{a2}\,\mathbb{e}^{\text{a1}+\text{a2}\left(-\text{pibaryt}+\frac{\text{x3}-\text{x4}+\text{yamount}}{\text{RFrate}\,(\text{x1}-\text{x2}-\text{kappa}\,(\text{x1}+\text{x2}+\text{x3}+\text{x4})+\text{xamount})+\text{d}\,(\text{x3}-\text{x4}+\text{yamount})}\right)^2}(-1-\text{kappa})\,\text{RFrate}\,(\text{x3}-\text{x4}+\text{yamount})\right.
$$

$$
\left.\left(-\text{pibaryt}+\frac{\text{x3}-\text{x4}+\text{yamount}}{\text{RFrate}\,(\text{x1}-\text{x2}-\text{kappa}\,(\text{x1}+\text{x2}+\text{x3}+\text{x4})+\text{xamount})+\text{d}\,(\text{x3}-\text{x4}+\text{yamount})}\right)\right)\Big/
$$

$$
\left.(\text{RFrate}\,(\text{x1}-\text{x2}-\text{kappa}\,(\text{x1}+\text{x2}+\text{x3}+\text{x4})+\text{xamount})+\text{d}\,(\text{x3}-\text{x4}+\text{yamount}))\right)-
$$

$$
\text{p}\left(\mathbb{e}^{\text{a1}+\text{a2}\left(-\text{pibaryt}+\frac{\text{x3}-\text{x4}+\text{yamount}}{\text{RFrate}\,(\text{x1}-\text{x2}-\text{kappa}\,(\text{x1}+\text{x2}+\text{x3}+\text{x4})+\text{xamount})+\text{u}\,(\text{x3}-\text{x4}+\text{yamount})}\right)^2}\right.
$$

$$
\left.(\text{RFrate}\,(\text{x1}-\text{x2}-\text{kappa}\,(\text{x1}+\text{x2}+\text{x3}+\text{x4})+\text{xamount})+\text{u}\,(\text{x3}-\text{x4}+\text{yamount}))\right)^{-\text{gamma}}
$$

$$
\left(\mathbb{e}^{\text{a1}+\text{a2}\left(-\text{pibaryt}+\frac{\text{x3}-\text{x4}+\text{yamount}}{\text{RFrate}\,(\text{x1}-\text{x2}-\text{kappa}\,(\text{x1}+\text{x2}+\text{x3}+\text{x4})+\text{xamount})+\text{u}\,(\text{x3}-\text{x4}+\text{yamount})}\right)^2}(-1-\text{kappa})\,\text{RFrate}-\right.
$$

$$
\left(2\,\text{a2}\,\mathbb{e}^{\text{a1}+\text{a2}\left(-\text{pibaryt}+\frac{\text{x3}-\text{x4}+\text{yamount}}{\text{RFrate}\,(\text{x1}-\text{x2}-\text{kappa}\,(\text{x1}+\text{x2}+\text{x3}+\text{x4})+\text{xamount})+\text{u}\,(\text{x3}-\text{x4}+\text{yamount})}\right)^2}(-1-\text{kappa})\,\text{RFrate}\,(\text{x3}-\text{x4}+\text{yamount})\right.
$$

$$
\left.\left(-\text{pibaryt}+\frac{\text{x3}-\text{x4}+\text{yamount}}{\text{RFrate}\,(\text{x1}-\text{x2}-\text{kappa}\,(\text{x1}+\text{x2}+\text{x3}+\text{x4})+\text{xamount})+\text{u}\,(\text{x3}-\text{x4}+\text{yamount})}\right)\right)\Big/
$$

$$
\left.(\text{RFrate}\,(\text{x1}-\text{x2}-\text{kappa}\,(\text{x1}+\text{x2}+\text{x3}+\text{x4})+\text{xamount})+\text{u}\,(\text{x3}-\text{x4}+\text{yamount}))\right)
$$

```
fx3 = D[f[x1, x2, x3, x4], x3]
```

$$-(1-p)\left(e^{a1+a2\left(-\text{pibaryt}+\frac{x3-x4+\text{yamount}}{\text{RFrate}\,(x1-x2-\text{kappa}\,(x1+x2+x3+x4)+\text{xamount})+d\,(x3-x4+\text{yamount})}\right)^2}\right.$$
$$\left.(\text{RFrate}\,(x1-x2-\text{kappa}\,(x1+x2+x3+x4)+\text{xamount})+d\,(x3-x4+\text{yamount}))\right)^{-\text{gamma}}$$
$$\left(e^{a1+a2\left(-\text{pibaryt}+\frac{x3-x4+\text{yamount}}{\text{RFrate}\,(x1-x2-\text{kappa}\,(x1+x2+x3+x4)+\text{xamount})+d\,(x3-x4+\text{yamount})}\right)^2}(d-\text{kappa}\,\text{RFrate})+\right.$$
$$2\,a2\,e^{a1+a2\left(-\text{pibaryt}+\frac{x3-x4+\text{yamount}}{\text{RFrate}\,(x1-x2-\text{kappa}\,(x1+x2+x3+x4)+\text{xamount})+d\,(x3-x4+\text{yamount})}\right)^2}$$
$$(\text{RFrate}\,(x1-x2-\text{kappa}\,(x1+x2+x3+x4)+\text{xamount})+d\,(x3-x4+\text{yamount}))$$
$$\left(-\frac{(d-\text{kappa}\,\text{RFrate})\,(x3-x4+\text{yamount})}{(\text{RFrate}\,(x1-x2-\text{kappa}\,(x1+x2+x3+x4)+\text{xamount})+d\,(x3-x4+\text{yamount}))^2}+\right.$$
$$\left.\frac{1}{\text{RFrate}\,(x1-x2-\text{kappa}\,(x1+x2+x3+x4)+\text{xamount})+d\,(x3-x4+\text{yamount})}\right)$$
$$\left.\left(-\text{pibaryt}+\frac{x3-x4+\text{yamount}}{\text{RFrate}\,(x1-x2-\text{kappa}\,(x1+x2+x3+x4)+\text{xamount})+d\,(x3-x4+\text{yamount})}\right)\right)-$$
$$p\left(e^{a1+a2\left(-\text{pibaryt}+\frac{x3-x4+\text{yamount}}{\text{RFrate}\,(x1-x2-\text{kappa}\,(x1+x2+x3+x4)+\text{xamount})+u\,(x3-x4+\text{yamount})}\right)^2}\right.$$
$$\left.(\text{RFrate}\,(x1-x2-\text{kappa}\,(x1+x2+x3+x4)+\text{xamount})+u\,(x3-x4+\text{yamount}))\right)^{-\text{gamma}}$$
$$\left(e^{a1+a2\left(-\text{pibaryt}+\frac{x3-x4+\text{yamount}}{\text{RFrate}\,(x1-x2-\text{kappa}\,(x1+x2+x3+x4)+\text{xamount})+u\,(x3-x4+\text{yamount})}\right)^2}(-\text{kappa}\,\text{RFrate}+u)+\right.$$
$$2\,a2\,e^{a1+a2\left(-\text{pibaryt}+\frac{x3-x4+\text{yamount}}{\text{RFrate}\,(x1-x2-\text{kappa}\,(x1+x2+x3+x4)+\text{xamount})+u\,(x3-x4+\text{yamount})}\right)^2}$$
$$(\text{RFrate}\,(x1-x2-\text{kappa}\,(x1+x2+x3+x4)+\text{xamount})+u\,(x3-x4+\text{yamount}))$$
$$\left(-\frac{(-\text{kappa}\,\text{RFrate}+u)\,(x3-x4+\text{yamount})}{(\text{RFrate}\,(x1-x2-\text{kappa}\,(x1+x2+x3+x4)+\text{xamount})+u\,(x3-x4+\text{yamount}))^2}+\right.$$
$$\left.\frac{1}{\text{RFrate}\,(x1-x2-\text{kappa}\,(x1+x2+x3+x4)+\text{xamount})+u\,(x3-x4+\text{yamount})}\right)$$
$$\left.\left(-\text{pibaryt}+\frac{x3-x4+\text{yamount}}{\text{RFrate}\,(x1-x2-\text{kappa}\,(x1+x2+x3+x4)+\text{xamount})+u\,(x3-x4+\text{yamount})}\right)\right)$$

fx4 = D[f[x1, x2, x3, x4], x4]

$$-(1-p)\left(e^{\text{a1}+\text{a2}\left(-\text{pibaryt}+\frac{\text{x3}-\text{x4}+\text{yamount}}{\text{RFrate}\,(\text{x1}-\text{x2}-\text{kappa}\,(\text{x1}+\text{x2}+\text{x3}+\text{x4})+\text{xamount})+\text{d}\,(\text{x3}-\text{x4}+\text{yamount})}\right)^2}\right.$$
$$\left.(\text{RFrate}\,(\text{x1}-\text{x2}-\text{kappa}\,(\text{x1}+\text{x2}+\text{x3}+\text{x4})+\text{xamount})+\text{d}\,(\text{x3}-\text{x4}+\text{yamount}))\right)^{\text{gamma}}$$
$$\left(e^{\text{a1}+\text{a2}\left(-\text{pibaryt}+\frac{\text{x3}-\text{x4}+\text{yamount}}{\text{RFrate}\,(\text{x1}-\text{x2}-\text{kappa}\,(\text{x1}+\text{x2}+\text{x3}+\text{x4})+\text{xamount})+\text{d}\,(\text{x3}-\text{x4}+\text{yamount})}\right)^2}(-\text{d}-\text{kappa}\,\text{RFrate})+\right.$$
$$2\,\text{a2}\,e^{\text{a1}+\text{a2}\left(-\text{pibaryt}+\frac{\text{x3}-\text{x4}+\text{yamount}}{\text{RFrate}\,(\text{x1}-\text{x2}-\text{kappa}\,(\text{x1}+\text{x2}+\text{x3}+\text{x4})+\text{xamount})+\text{d}\,(\text{x3}-\text{x4}+\text{yamount})}\right)^2}$$
$$(\text{RFrate}\,(\text{x1}-\text{x2}-\text{kappa}\,(\text{x1}+\text{x2}+\text{x3}+\text{x4})+\text{xamount})+\text{d}\,(\text{x3}-\text{x4}+\text{yamount}))$$
$$\left(-\frac{(-\text{d}-\text{kappa}\,\text{RFrate})\,(\text{x3}-\text{x4}+\text{yamount})}{(\text{RFrate}\,(\text{x1}-\text{x2}-\text{kappa}\,(\text{x1}+\text{x2}+\text{x3}+\text{x4})+\text{xamount})+\text{d}\,(\text{x3}-\text{x4}+\text{yamount}))^2}-\right.$$
$$\left.\frac{1}{\text{RFrate}\,(\text{x1}-\text{x2}-\text{kappa}\,(\text{x1}+\text{x2}+\text{x3}+\text{x4})+\text{xamount})+\text{d}\,(\text{x3}-\text{x4}+\text{yamount})}\right)$$
$$\left.\left(-\text{pibaryt}+\frac{\text{x3}-\text{x4}+\text{yamount}}{\text{RFrate}\,(\text{x1}-\text{x2}-\text{kappa}\,(\text{x1}+\text{x2}+\text{x3}+\text{x4})+\text{xamount})+\text{d}\,(\text{x3}-\text{x4}+\text{yamount})}\right)\right)-$$
$$\text{p}\left(e^{\text{a1}+\text{a2}\left(-\text{pibaryt}+\frac{\text{x3}-\text{x4}+\text{yamount}}{\text{RFrate}\,(\text{x1}-\text{x2}-\text{kappa}\,(\text{x1}+\text{x2}+\text{x3}+\text{x4})+\text{xamount})+\text{u}\,(\text{x3}-\text{x4}+\text{yamount})}\right)^2}\right.$$
$$\left.(\text{RFrate}\,(\text{x1}-\text{x2}-\text{kappa}\,(\text{x1}+\text{x2}+\text{x3}+\text{x4})+\text{xamount})+\text{u}\,(\text{x3}-\text{x4}+\text{yamount}))\right)^{-\text{gamma}}$$
$$\left(e^{\text{a1}+\text{a2}\left(-\text{pibaryt}+\frac{\text{x3}-\text{x4}+\text{yamount}}{\text{RFrate}\,(\text{x1}-\text{x2}-\text{kappa}\,(\text{x1}+\text{x2}+\text{x3}+\text{x4})+\text{xamount})+\text{u}\,(\text{x3}-\text{x4}+\text{yamount})}\right)^2}(-\text{kappa}\,\text{RFrate}-\text{u})+\right.$$
$$2\,\text{a2}\,e^{\text{a1}+\text{a2}\left(-\text{pibaryt}+\frac{\text{x3}-\text{x4}+\text{yamount}}{\text{RFrate}\,(\text{x1}-\text{x2}-\text{kappa}\,(\text{x1}+\text{x2}+\text{x3}+\text{x4})+\text{xamount})+\text{u}\,(\text{x3}-\text{x4}+\text{yamount})}\right)^2}$$
$$(\text{RFrate}\,(\text{x1}-\text{x2}-\text{kappa}\,(\text{x1}+\text{x2}+\text{x3}+\text{x4})+\text{xamount})+\text{u}\,(\text{x3}-\text{x4}+\text{yamount}))$$
$$\left(-\frac{(-\text{kappa}\,\text{RFrate}-\text{u})\,(\text{x3}-\text{x4}+\text{yamount})}{(\text{RFrate}\,(\text{x1}-\text{x2}-\text{kappa}\,(\text{x1}+\text{x2}+\text{x3}+\text{x4})+\text{xamount})+\text{u}\,(\text{x3}-\text{x4}+\text{yamount}))^2}-\right.$$
$$\left.\frac{1}{\text{RFrate}\,(\text{x1}-\text{x2}-\text{kappa}\,(\text{x1}+\text{x2}+\text{x3}+\text{x4})+\text{xamount})+\text{u}\,(\text{x3}-\text{x4}+\text{yamount})}\right)$$
$$\left.\left(-\text{pibaryt}+\frac{\text{x3}-\text{x4}+\text{yamount}}{\text{RFrate}\,(\text{x1}-\text{x2}-\text{kappa}\,(\text{x1}+\text{x2}+\text{x3}+\text{x4})+\text{xamount})+\text{u}\,(\text{x3}-\text{x4}+\text{yamount})}\right)\right)$$

Bibliography

[1] P.P. Boyle and X. Lin. Optimal portfolio selection with transaction costs. *North American Actuarial Journal*, 1(2):27–39, 1997.

[2] John Y. Campbell and Luis M. Viceira. Consumption and portfolio decisions when expected returns are time varying. Harvard institute of economic research working papers, Harvard - Institute of Economic Research, 1998.

[3] G. M. Constantinides. Multiperiod consumption and investment behavior with convex transaction costs. *Management Science*, 25:1127–1137, 1979.

[4] G. M. Constantinides. Capital market equilibrium with transaction costs. *Journal of Political Economy*, 94:842–862, 1986.

[5] J. M. Cox, S. A. Ross, and M. Rubinstein. Option pricing: A simplified approach. *Journal of Financial Economics*, 7:229–263, 1979.

[6] M. H. Davis and A. R. Norman. Portfolio selection wiht transaction costs. *Mathematics of Operations Research*, 15(4):676–713, 1990.

[7] G. Gennotte and A. Jung. Investment strategies under transactions costs: The finite horizon case. *Management Science*, 38(11):385–404, 1994.

[8] Anthony W. Lynch and Pierluigi Balduzzi. Predictability and transaction costs: The impact on rebalancing rules and behavior. *The Journal of Finance*, 55(5):2285–2309, Oct 2000.

[9] Harry M. Markowitz. Portfolio selection. *The Journal of Finance*, 7(1):77–91, 1952.

[10] Robert C. Merton. *Continuous-Time Finance.* Wiley-Blackwell, Hoboken, NJ, second edition, Nov 1992.

[11] B. Oksendal and A. Sulem. Optimal consumption and portfolio with both fixed and proportional transaction costs: A combined stochastic control and impulse control model. *SIAM Journal on Control and Optimization*, 40(6):1765–1790, 2001.

[12] Valeri I. Zakamouline. Optimal portfolio selection with both fixed and proportional transactions costs for a crra investor with finite horizon. In *Discussion Papers*. Norwegian School of Economics and Business Administration, Department of Finance and Management Science, 2002.